

В. Липский

Комбинаторика для программистов

Перевод с польского

В. А. Евстигнеева и

О. А. Логиновой

под редакцией

А. П. Ершова



Москва «Мир» 1988

0.1 Предисловие редактора перевода

Широкое применение ЭВМ создало новый жанр математической литературы. В книгах этого жанра изложение начинается с теоретического обзора и заканчивается описанием алгоритмов, практически готовых к автоматическому исполнению на ЭВМ. Конечно, новизна такого подхода относительно и традиция завершения математического трактата правилами вычислений восходит, по меньшей мере, к ал-Хорезми. Но все же оформление операционного багажа математики в виде машинных программ существенно отличает новые книги от классических инженерных руководств по прикладной математике, отличным примером которых служит известный справочник И.Н. Бронштейна и Н.Н. Семендяева.

Первые публикации подобного рода относились к традиционным разделам вычислительной математики. Дальнейшее расширение круга задач, решаемых на ЭВМ, потребовало выхода на модели дискретной математики, что привело к подлинному возрождению теории графов и комбинаторики, которые за несколько десятков лет трансформировались из разделов «досуговой» математики в первостепенный инструмент решения огромного числа задач.

После выхода первых трех томов еще не завершенной монументальной серии «Искусство программирования для ЭВМ» Д. Кнута [т.1 «Основные алгоритмы» (М.: Мир, 1976), т.2 «Получисленные алгоритмы» (М.: Мир, 1977), т.3 «Сортировка и поиск» (М.: Мир, 1978)] аналогичных руководств не появлялось в течение длительного времени. В отечественной литературе следовало бы в связи с этим отметить руководство В.А. Евстигнеева «Применение теории графов в программировании» (М.: Наука, 1985). При всей обстоятельности этой книги она далеко не исчерпывает операционный запас дискретной математики. Уже в ходе ее написания автор и редактор познакомились с предлагаемой вниманию читателей монографией польского специалиста Витольда Липского и увидели, насколько она хороша и полезна.

Не стоит предвосхищать авторское вступление к монографии В. Липского, да и ее оглавление говорит само за себя. Хотелось бы только обратить внимание читателей на своеобразное сочетание традиционного и новаторского подходов к изложению материала, который особенно проявляется при изложении комбинаторных задач, имеющих давнюю историю. Именно это обстоятельство побуждает высказать не столько критическое замечание, сколько пожелание авторам будущих книг.

Еще нерешенной проблемой изложения математической теории, завершающейся алгоритмами, является отсутствие доказательной логической связи между теоретическим материалом и алгоритмами. Доказательное программирование в его логическом и трансформационном подходах предлагает те или иные процедуры систематического извлечения алгоритма из спецификации задачи,

использующие отношения и факты соответствующей теории, однако мы еще далеки от того, чтобы излагать эти процедуры в стандартном и непринужденном стиле, вошедшем в обыденную научно-литературную практику. Актуальность этой проблемы постоянно нарастает, особенно потому, что часто практическая реализация алгоритмов требует их определенной модификации, которая не должна снижать степень математической достоверности окончательного варианта программы.

Это предисловие приходится закончить печальной вестью о безвременной кончине автора этой монографии, последовавшей после тяжелой болезни 30 мая 1985 г. на 36-м году жизни. За 14 лет интенсивной научной деятельности В. Липский опубликовал свыше пятидесяти работ по комбинаторике, теории информационного поиска и вычислительной геометрии, а также написал три монографии: «Комбинаторные аспекты теории информационного поиска» (1975 г.), данная монография (1982 г.) и «Комбинаторный анализ» (посмертное издание 1987 г.).

Переводчики и автор этих строк посвящают свой скромный труд доброй памяти одного из ведущих представителей польской информатики, д-ра наук, доцента Витольда Липского.

Академгородок, январь 1987 г.

А.П. Ершов

0.2 От автора

Сейчас трудно было бы, пожалуй, назвать раздел теоретической информатики, в котором в течение последнего десятилетия были бы достигнуты большие успехи, нежели в конструировании и анализе комбинаторных алгоритмов. С одной стороны, было обнаружено много новых, более эффективных методов решения комбинаторных задач с помощью ЭВМ, с другой — получены теоретические результаты, свидетельствующие все более явно о том, что для широкого класса проблем не существует «достаточно эффективных» алгоритмов. Эффективные комбинаторные алгоритмы находят применение во многих областях нечисленной обработки информации, особенно в дискретной оптимизации и в исследовании операций.

В настоящей книге представлены некоторые разделы комбинаторики, причем особое внимание уделено конструктивному алгоритмическому подходу — рядом с обсуждаемыми комбинаторными проблемами, как правило, приводятся алгоритмы их решения вместе с анализом их вычислительной сложности. Эти алгоритмы представляют собой сжатые варианты программ, написанных на языке Паскаль. На выбор обсуждаемых проблем, в большой мере случай-

ный, принимая во внимание ограниченный объем книги, а также обширность рассматриваемой области, оказали влияние как интересы автора, так и желание скорей дополнить, нежели продублировать две другие книги с родственной тематикой, подготовленные к изданию в серии «Библиотека технологии программирования» [2] и в серии «Информатика» [76].

Первая, самая большая глава данной книги содержит изложение наиболее классических разделов комбинаторики (перестановки, разбиения множеств и чисел, биномиальные коэффициенты, производящие функции, и т.д.), а также многие — необязательно классические — алгоритмы генерирования упомянутых комбинаторных объектов. Во второй главе представлены основные методы, используемые при конструировании алгоритмов на графах, в особенности методы систематичного обхода графов. Тематика, связанная с графами, затрагивается и в двух следующих главах: в одной из них обсуждаются методы нахождения кратчайших путей в графах, ребрам которых приписаны произвольные «длины», в другой — основное внимание сконцентрировано на задаче отыскания максимального потока в сети (т.е. в графе с определенными «пропускными способностями» ребер). В последней главе рассматривается применение комбинаторного понятия матроида для решения некоторого класса оптимизационных задач.

Книга предназначена для программистов, желающих расширить свои знания в области комбинаторных алгоритмов, а также пополнить свои практические знания теоретическими. От читателя требуются элементарные сведения из математики, а также знакомство с языком программирования Паскаль [36, 75] и некоторый опыт программирования на языке высокого уровня.

В заключение хочу поблагодарить д-ра наук Виктора Марека за ряд ценных замечаний, которые позволили устранить погрешности первоначального варианта этой книги.

Варшава, декабрь 1980

Витольд Липский

Глава 1

Введение в комбинаторику

1.1 Основные понятия

В этом разделе приводятся основные определения и обозначения, относящиеся к используемым логическим и теоретико-множественным понятиям, а также представленные в приводимых ниже алгоритмах.

Начнем с логических и теоретико-множественных понятий (читателя, заинтересованного в более глубоком знакомстве с этими понятиями, мы отсылаем к работам [49] и [57]). Мы будем употреблять логические связки \vee (или), \wedge (и), \neg (не), \Rightarrow (если ..., то), \Leftrightarrow (тогда и только тогда, когда). Тот факт, что x есть элемент множества X , будем записывать в виде $x \in X$, его отрицание — в виде $x \notin X$. Множество тех элементов множества X , которые удовлетворяют условию Φ , будем обозначать через $\{x \in X : \Phi\}$ (или $\{x : \Phi\}$, если известно, о каком множестве X идет речь), запись же $\{a_1, \dots, a_n\}$ будет обозначать множество, элементы которого суть a_1, \dots, a_n (в частности, единственным элементом множества $\{a\}$ является a). Теоретико-множественные операции объединения, пересечения и разности обозначаются соответственно \cap , \cup и \setminus , пустое множество обозначается \emptyset . Тот факт, что множество A содержится в множестве B (т.е. A есть подмножество множества B), будет записываться в виде $A \subseteq B$ или $B \supseteq A$ (всегда имеют место включения $\emptyset \subseteq A$, $A \subseteq A$); символ « \subsetneq » зарезервирован для случая, когда исключается равенство $A = B$ (при этом будем говорить, что A есть *собственное подмножество* множества B). Множество всех подмножеств множества X будем обозначать через $\wp(X)$, мощность множества X (т.е. число его элементов) — через $|X|$.

Последовательность длины n , члены которой суть a_1, \dots, a_n , будем обозначать через $\langle a_1, \dots, a_n \rangle$, либо просто через a_1, \dots, a_n или $a_1 \dots a_n$. Последователь-

ность $\langle a, b \rangle$ длины два будем называть *упорядоченной парой*. Декартово произведение $A \times B$ множеств A и B определяется как множество всевозможных пар $\langle a, b \rangle$, где $a \in A$, $b \in B$. Под *бинарным отношением* (с *левой областью* A и *правой областью* B) подразумевается произвольное подмножество $R \subseteq A \times B$. Если $A = B$, то будем говорить о *бинарном отношении* на множестве A . Вместо $\langle a, b \rangle \in R$ часто пишут aRb .

По поводу отношения R на множестве X говорят, что оно:

- (а) рефлексивно, если xRx для каждого $x \in X$,
- (б) транзитивно, если $(xRy \wedge yRz) \Rightarrow xRz$ для произвольных $x, y, z \in X$,
- (в) симметрично, если $xRy \Rightarrow yRx$ для произвольных $x, y \in X$,
- (г) антисимметрично, если $(xRy \wedge yRx) \Rightarrow x = y$ для произвольных $x, y \in X$.

Произвольное бинарное отношение, обладающее свойствами рефлексивности, транзитивности и симметричности, называется *отношением эквивалентности*, а обладающее свойствами рефлексивности, транзитивности и антисимметричности, — *отношением частичной упорядоченности*. Отношение частичной упорядоченности обычно обозначается « \leq », а пара $\langle X, \leq \rangle$ называется *частично упорядоченным множеством*. Будем применять также очевидные обозначения, такие как $x \geq y$ для $y \leq x$, $x < y$ для $x \leq y \wedge x \neq y$ и т.д. Примером частично упорядоченного множества может служить множество целых чисел с отношением делимости, множество целых (или вещественных) чисел с обычным отношением меньше или равно « \leq », а также множество $\wp(X)$ с отношением включения \subseteq .

Если функция (отображение) f сопоставляет каждому элементу $x \in X$ элемент $f(x) \in Y$, то будем писать $f : X \rightarrow Y$ (такая функция может трактоваться как отношение $R \subseteq X \times Y$ с тем свойством, что для каждого $x \in X$ существует в R точно одна пара вида $\langle x, y \rangle$, $y \in Y$, для наших же целей достаточно, однако, интуитивного понятия функции). Для произвольных $A \in X$, $B \in Y$ определим

$$f(A) = \{y \in Y : \text{существует такое } x \in A, \text{ что } y = f(x)\}$$

$$f^{-1}(B) = \{x \in X : f(x) \in B\}$$

(вместо $f^{-1}(\{b\})$ будем просто писать $f^{-1}(b)$).

Если $f(X) = Y$, то будем говорить о функции из X на Y . Функция $f : X \rightarrow Y$ называется *обратимой* (*взаимно однозначной*), если для произвольных $a, b \in X$

$$a \neq b \Rightarrow f(a) \neq f(b).$$

Мы часто будем использовать понятие графа (см. [9],[31]). Под *неориентированным графом* (или короче *графом*) будем понимать такую произвольную пару $G = \langle V, E \rangle$, что

$$E \subseteq \{\{u, v\} : u, v \in V \wedge u \neq v\}.$$

*Ориентированным графом*¹ будем называть такую произвольную пару $G = \langle V, E \rangle$, что $E \subseteq V \times V$ и в обоих случаях множества V и E будем называть соответственно множеством *вершин* и множеством *ребер*² графа G .

Граф обычно изображается на плоскости в виде множества точек, соответствующих вершинам, и соединяющих их линий, соответствующих ребрам.³ Линия, изображающая ребро $\{u, v\}$, или $\langle u, v \rangle$ ⁴, соединяет точки, изображающие вершины u, v причем во втором случае стрелка обозначает направление от u к v (рис. 1.1).

В контексте определенного графа $G = \langle V, E \rangle$ будем часто использовать обозначения $u - v$, $u \rightarrow v$ вместо $\{u, v\} \in E$ и $\langle u, v \rangle \in E$ соответственно. Если ребро e имеет вид $\{u, v\}$ или $\langle u, v \rangle$, то будем говорить, что ребро e *инцидентно* вершинам u и v , в то время как вершины u и v *смежны* между собой. *Степень вершины* определим как число ребер, инцидентных ей.⁵ Вершину нулевой степени будем называть *изолированной* (например, вершина v_5 на рис.1.1, а). *Путем* в графе $G = \langle V, E \rangle$ назовем последовательность вершин v_0, v_1, \dots, v_n , такую, что $k \geq 0$ и $v_i - v_{i+1}$ (или $v_i \rightarrow v_{i+1}$, если граф G — ориентированный), $i = 0, \dots, k - 1$.⁶ Вершины v_0 и v_k будем называть соответственно *началом* и *концом* пути, а число k — *длиной* пути. Путь, начало и конец которого совпадают, будем называть *циклом*⁷. Если все вершины пути v_1, \dots, v_k различны, то будем говорить об *элементарном* пути. Соответственно цикл v_1, \dots, v_k ($v_1 = v_k$) будем называть *элементарным*, если вершины v_1, \dots, v_k различны. *Подграфом* графа $G = \langle V, E \rangle$ будем называть такой произвольный граф $G' = \langle V', E' \rangle$, что

¹или короче *орграфом* — Прим. перев.

²Элементы множества E для орграфа называются *дугами* — Прим. перев.

³Дуга в орграфе изображается линией со стрелкой, указывающей ориентацию дуги, т.е. направление от ее начала к концу. — Прим. перев.

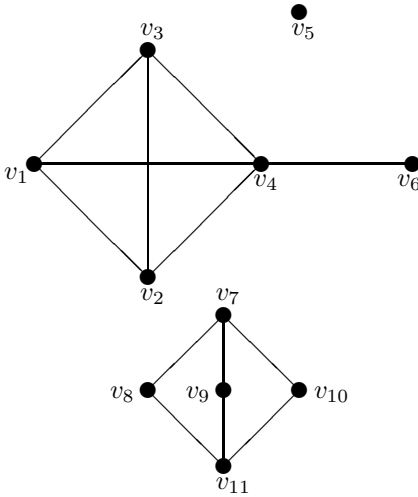
⁴Угловые скобки используются для обозначения дуг орграфа. — Прим. перев.

⁵Для вершин орграфа определяются *полустепени захода* (число заходящих в вершину дуг) и *исхода* (число выходящих дуг). Степень вершины определяется как сумма полустепеней захода и исхода. — Прим. перев.

⁶Термин «путь» в теории графов используется только в отношении орграфов, для графов используются термины «цепь» или «маршрут». — Прим. перев.

⁷Введенный так термин «цикл» в теории графов используется только в отношении графов, для орграфов используется термин «контур». — Прим. перев.

(a)



(б)

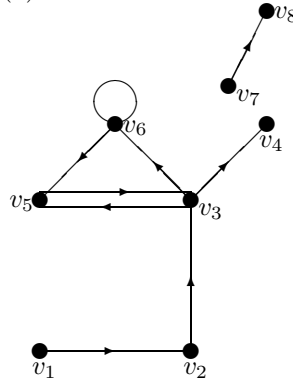


Рис. 1.1: а) неориентированный граф б) неориентированный граф

$V' \subseteq V$ и $E' \subseteq E$.⁸

Пусть $G = \langle V, E \rangle$ — произвольный неориентированный граф, и пусть $v \in V$. Пусть A — множество тех вершин $u \in V$, к которым существует путь из v . Множество A вместе с ребрами графа G , инцидентными вершинам из A , определяет некоторый подграф, называемый компонентой связности графа G . Очевидно, что множества вершин компонент связности произвольного графа попарно не пересекаются. Например, для графа на рис.1.1, а это суть множества $V_1 = \{v_1, v_2, v_3, v_4, v_6\}$, $V_2 = \{v_5\}$ и $V_3 = \{v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$.

Будем говорить, что графы $G = \langle V, E \rangle$, $G' = \langle V', E' \rangle$ *изоморфны*, если существует такое взаимно однозначное отображение f из V на V' , что для произвольных $u, v \in V$ имеем $\{u, v\} \in E \Leftrightarrow \{f(u), f(v)\} \in E'$ ($\langle u, v \rangle \in E \Leftrightarrow \langle f(u), f(v) \rangle \in E'$ в случае ориентированных графов). Обычно изоморфные графы не различаются между собой.

Для произвольного вещественного числа x мы будем употреблять обозна-

⁸В отечественной литературе по теории графов граф G' называется чаще *частью* графа, или *частичным графом*, под подграфом же понимается частичный граф, удовлетворяющий дополнительному условию $\forall x, y (x, y \in V' \wedge \{x, y\} \in E \Rightarrow \{x, y\} \in E')$. — *Прим. перев.*

чения $\lfloor x \rfloor$ и $\lceil x \rceil$ соответственно для наибольшего целого числа, не превосходящего x , и для наименьшего целого числа, не меньшего x , например $\lfloor 3.5 \rfloor = 3$, $\lceil 3.5 \rceil = 4$, $\lfloor -3.5 \rfloor = -4$, $\lceil -3.5 \rceil = -3$.

Перейдем теперь к понятиям, связанным с алгоритмами. Алгоритмы будем обычно записывать на языке программирования, являющимся неформальной версией языка Паскаль [36,75]. Если реализация какого-либо фрагмента программы очевидна, но трудоемка и затемняет идею алгоритма, то такой фрагмент будем иногда заменять описанием на естественном языке. Мы будем также применять неформальные конструкции, такие как, например, циклы (**for** $x \in X$ **do** P (выполнять команду P для всех элементов x множества X в произвольной последовательности)), СТЕК $\Leftarrow x$ (поместить значение переменной x в стек), $x \Rightarrow$ СТЕК (считать элемент x из вершины стека и принять его за значение переменной x), ОЧЕРЕДЬ $\Leftarrow x$ (включить x в очередь в качестве последнего элемента), $x \Rightarrow$ ОЧЕРЕДЬ (взять первый элемент из очереди и принять его в качестве значения переменной x) и т.д. Мы будем обычно опускать описания типов и переменных (иногда для избежания недоразумений будем помещать соответствующие пояснения в комментарий). Переменная, появляющаяся в процедуре, рассматривается как локальная для данной процедуры, исключая тот случай, когда в комментарии сказано что-либо иное. Строки программы нумеруются так, чтобы можно было указать на «цикл 17», «блок 9» и т.д.

Основным параметром алгоритма, который будет нас интересовать, является его *вычислительная сложность* (или просто *сложность*), т.е. число шагов, выполняемых алгоритмом в худшем случае как функция размерности задачи, представленной входными данными. Например, если алгоритм принимает как данные произвольный граф $G = \langle V, E \rangle$, то под размерностью задачи можно понимать $|V|$. Сложность алгоритма определяется тогда как функция f , такая что $f(n)$ равно наибольшему числу шагов алгоритма для произвольного графа с n вершинами. Можно также считать размерностью задачи пару $\langle |V|, |E| \rangle$ — тогда сложностью является функция двух переменных и $f(n, m)$ равно наибольшему числу шагов, выполняемых алгоритмом для произвольного графа с n вершинами и m ребрами. Остается еще объяснить точнее, что мы понимаем под «шагом» алгоритма. Допустим, что наши программы транслируются на машинный язык типичной ЭВМ, имеющей в наборе своих команд команды переноса слова из памяти в буфер и наоборот, арифметические операции сложения, вычитания, умножения и деления, условные переходы, операции ввода-вывода, а также косвенной адресации, выполненной аппаратно (т.е. определение аргумента операции через адрес ячейки памяти, содержащей адрес этого аргумента). Выполнение любой из указанных выше команд мы и будем считать шагом алгоритма. Очевидно, что при таком определении шага сложность алгоритма зависит от конкретного вида машинных команд. Однако нас никогда не будет интересовать

точная сложность алгоритма, а только асимптотическая сложность, т.е. асимптотическая скорость увеличения числа шагов алгоритма, когда размерность задачи неограниченно растет (чтобы можно было говорить о такой скорости роста, предполагаем, что объем памяти нашего компьютера неограниченный, а также, что каждая ячейка памяти может содержать произвольно большое целое число). Ясно, что при двух произвольных «разумных» способах трансляции соответствующие сложности различаются не более чем на мультипликативную постоянную, а их скорость роста одинакова. Читателя, желающего уточнить приведенные выше очень неформальные рассуждения, отсылаем к работам [1] и [2].

При сравнении скорости роста двух функций $f(n)$ и $g(n)$ (с неотрицательными значениями) очень удобны следующие обозначения:

- $f(n) = O(g(n)) \Leftrightarrow$ существуют константы $C, N > 0$, такие что $f(n) \leq C \cdot g(n)$ для всех $n \geq N$
- $f(n) = \Omega(g(n)) \Leftrightarrow$ существуют константы $C, N > 0$, такие что $f(n) \geq C \cdot g(n)$ для любого $n \geq N$.

Конечно, $f(n) = \Omega(g(n))$ тогда и только тогда, когда $g(n) = O(f(n))$. Символы $O(g(n))$ и $\Omega(g(n))$ читаются соответственно: «порядка не более чем $g(n)$ » и «порядка не менее чем $g(n)$ ». Если сложность какого-либо алгоритма есть $O(g(n))$, то мы говорим, что этот алгоритм «затрачивает порядка $O(g(n))$ времени»⁹. Подобным же образом определяются символы $O(g(n_1, \dots, n_k))$ и $\Omega(g(n_1, \dots, n_k))$ для функции многих переменных, например:

$f(n_1, \dots, n_k) = O(g(n_1, \dots, n_k)) \Leftrightarrow$ существуют константы $C, N \geq 0$, такие что $f(n_1, \dots, n_k) \leq g(n_1, \dots, n_k)$ для всех $n_1, \dots, n_k \geq N$

Определенную таким образом сложность иногда называют *временной сложностью* в отличие от *сложности по памяти*, определяющей величину объема памяти, использованного алгоритмом, как функцию размерности задачи.

1.2 Функции и размещения

Классической задачей комбинаторики является задача определения числа способов размещения некоторых объектов в каком-то количестве «ящиков» так, чтобы были выполнены заданные ограничения. Эту задачу можно сформулировать несколько более формально следующим образом. Даны множества X , Y , причем $|X| = n$, $|Y| = m$. Сколько существует функций $f : X \rightarrow Y$, удовлетворяющих заданным ограничениям? Элементы множества X соответствуют

⁹Символьную запись $f(n) = O(g(n))$ не следует трактовать как равенство; например, из $f(n) = O(g(n))$ и $h(n) = O(g(n))$, конечно, не вытекает $f(n) = h(n)$.

объектам, элементы множества Y — ящикам, а каждая функция $f : X \rightarrow Y$ определяет некоторое размещение, указывая для каждого объекта $x \in X$ ящик $f(x) \in Y$, в котором данный объект находится. Другую традиционную интерпретацию получим, трактуя Y как множество «цветов», а $f(x)$ как «цвет объекта x ». Наша задача, таким образом, эквивалентна вопросу, сколькими способами можно покрасить объекты так, чтобы были соблюдены некоторые ограничения.

Заметим, что без потери общности можем всегда считать, что $X = 1, \dots, n$ и $Y = 1, \dots, m$. Каждую функцию f можно тогда отождествить с последовательностью $\langle f(1), \dots, f(n) \rangle$.

Наша задача имеет самый простой вид, если не накладывается никаких ограничений на размещения. Имеет место следующая теорема.

Теорема 1.1. *Если $|X| = n$, $|Y| = m$, то число всех функций $f : X \rightarrow Y$ равно m^n .*

Доказательство. Считая, что $X = 1, \dots, n$, сводим нашу задачу к вопросу о числе всех последовательностей $\langle y_1, \dots, y_n \rangle$ с членами из m -элементного множества Y . Каждый член последовательности y_i мы можем выбрать m способами, что дает m^n возможностей выбора последовательности $\langle y_1, \dots, y_n \rangle$. ■

Легко также найти число размещений, для которых каждый ящик содержит не более одного объекта — такие размещения соответствуют взаимно однозначным функциям. Обозначим через $[m]_n$ число всех взаимно однозначных функций из n -элементного множества в m -элементное множество.

Теорема 1.2. *Если $|X| = n$, $|Y| = m$, то число всех взаимно однозначных функций $f : X \rightarrow Y$ равно*

$$[m]_n = m(m-1) \dots (m-n+1) \quad (1.1)$$

(полагаем $[m]_0 = 1$).

Доказательство. Будем определять на этот раз число инъективных (т.е. имеющих все различные члены) последовательностей $\langle y_1, \dots, y_n \rangle$ с членами из множества Y . Элемент y_1 такого множества мы можем выбрать m способами, элемент y_2 — $m-1$ способами, в общем случае если уже выбраны элементы y_1, \dots, y_{i-1} , то в качестве y_i можем выбрать любой из $m-i+1$ элементов множества $Y \setminus \{y_1, \dots, y_{i-1}\}$ (принимая $n \leq m$, если $n > m$, то очевидно, что и $[m]_n$ и искомое число функций равны нулю). (Это дает $m(m-1) \dots (m-n+1)$ возможностей выбора инъективных последовательностей $\langle y_1, \dots, y_n \rangle$) ■

Приведем в качестве примера $[4]_3 = 24$ последовательности длины 3 с эле-

ментами из множества $X = \{1, 2, 3, 4\}$:

$\langle 1, 2, 3 \rangle$	$\langle 2, 1, 3 \rangle$	$\langle 3, 1, 2 \rangle$	$\langle 4, 1, 2 \rangle$
$\langle 1, 2, 4 \rangle$	$\langle 2, 1, 4 \rangle$	$\langle 3, 1, 4 \rangle$	$\langle 4, 1, 3 \rangle$
$\langle 1, 3, 2 \rangle$	$\langle 2, 3, 1 \rangle$	$\langle 3, 2, 1 \rangle$	$\langle 4, 2, 1 \rangle$
$\langle 1, 3, 4 \rangle$	$\langle 2, 3, 4 \rangle$	$\langle 3, 2, 4 \rangle$	$\langle 4, 2, 3 \rangle$
$\langle 1, 4, 2 \rangle$	$\langle 2, 4, 1 \rangle$	$\langle 3, 4, 1 \rangle$	$\langle 4, 3, 1 \rangle$
$\langle 1, 4, 3 \rangle$	$\langle 2, 4, 3 \rangle$	$\langle 3, 4, 2 \rangle$	$\langle 4, 3, 2 \rangle$

Если $m = n$, то каждая взаимно однозначная функция $f : X \rightarrow Y$ является взаимно однозначным отображением множества X на множество Y . В таком случае $[n]_n = n(n-1) \cdot \dots \cdot 1$ обозначаем $n!$ (n факториал). Каждое взаимно однозначное отображение $f : X \rightarrow X$ называется *перестановкой* множества X . Как частный случай теоремы 1.2 получаем следующую теорему.

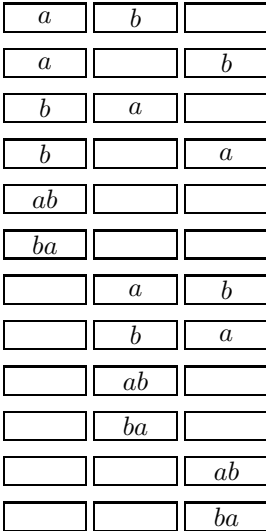
Теорема 1.3. *Число перестановок n -элементного множества равно $n!$*

Перестановки мы будем обсуждать в последующих разделах, сейчас же остановимся еще на одном типе размещения объектов по ящикам. Предположим, что мы размещаем n объектов по t ящикам так, чтобы каждый ящик содержал бы последовательность, а не множество, как прежде, помещенных в нем объектов. Два размещения назовем равными, если в каждом ящике содержится одна и та же последовательность объектов. Размещения такого типа будем называть упорядоченными размещениями n объектов по t ящикам. Обозначим число таких упорядочений через $[m]^n$.

Теорема 1.4. *Число упорядоченных размещений n объектов по t ящикам равно*

$$[m]^n = t(t+1) \dots (t+n-1) \quad (1.2)$$

Рис. 1.2: Размещение (упорядоченное) элементов a, b в трех ящиках



(полагаем $[m]^0 = 1$).

Доказательство. Будем строить упорядоченное размещение, добавляя по очереди новые объекты. Первый объект мы можем разместить t способами, второй — $t+1$ способами, ибо его можно разместить в одном из $t-1$ пустых ящиков или в ящике, содержащем первый объект, перед ним или после него. В общем случае предположим, что уже размещено $i-1$ объектов, причем для $k = 1, 2, \dots, t$ в k -м ящике находятся r_k объектов. Тогда

i -й объект можем добавить в k -й ящик $r_k + 1$ способами, что дает в сумме

$$(r_1 + 1) + \dots + (r_m + 1) = (r_1 + \dots + r_m) + m = m + i - 1$$

возможностей. Таким образом, всех упорядоченных размещений будет $m(m + 1) \dots (m + n - 1)$. ■

На рис. 1.2 представлены $[3]^2 = 12$ упорядоченных размещений элементов a , b в трех ящиках.

Приведем в заключение следующие простые зависимости:

$$[m]_n = (m - n + 1)[m]_{n-1}, \quad (1.3)$$

$$[m]_n = m!/n!, \quad (1.4)$$

$$[m]^n = [m + n - 1]_n. \quad (1.5)$$

1.3 Перестановки: разложение на циклы, знак перестановки

Напомним, что перестановкой¹⁰ n -элементного множества X называется произвольная взаимно однозначная функция $f : X \rightarrow X$. Обычно перестановка определяется с помощью таблицы с двумя строками, из которых каждая содержит все элементы множества X , причем элемент $f(x)$ помещается под элементом x . Для примера рассмотрим такую перестановку f множества $\{a, b, c, d\}$, что

$$f(a) = d; f(b) = a; f(c) = b; f(d) = c;$$

она записывается в виде

$$f = \begin{pmatrix} a & b & c & d \\ d & a & b & c \end{pmatrix}$$

Если порядок элементов в верхней строке фиксирован, то каждой перестановке однозначно соответствует последовательность, содержащаяся в нижней строке, например для перестановки f это есть $\langle d, a, b, c \rangle$. Поэтому будем называть иногда произвольную инъективную последовательность длины n с элементами из множества X перестановкой n -элементного множества X .

В наших исследованиях природа элементов множества X несущественна — примем для простоты $X = \{1, \dots, n\}$. Обозначим множество всех перестановок этого множества через S_n . Произвольная перестановка $f \in S_n$ будет обычно

¹⁰Обычно функция $f : X \rightarrow X$ называется подстановкой, а перестановкой называется вторая строка таблицы, определяющей подстановку. В этом разделе и далее термин «перестановка» используется для обозначения обоих понятий, что, однако, не приводит к какой-либо двусмысленности. — *Прим. перев.*

отождествляться с последовательностью $\langle a_1, \dots, a_n \rangle$, где $a_i = f(i)$. Под суперпозицией перестановок f и g мы будем понимать перестановку fg , определяемую следующим образом:

$$fg(i) = f(g(i)).$$

Отметим, что для суперпозиции двух перестановок, скажем

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix} \quad g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix},$$

достаточно изменить порядок столбцов в перестановке f таким образом, чтобы в первой строке получить последовательность, имеющуюся во второй строке перестановки g , тогда вторая строка перестановки f дает суперпозицию fg . В нашем случае

$$\begin{aligned} f &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix} \\ g &= \begin{pmatrix} 5 & 3 & 2 & 1 & 4 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix}, \\ fg &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix} \end{aligned}$$

Перестановку

$$e = \begin{pmatrix} 1 & 2 & \dots & n \\ 1 & 2 & \dots & n \end{pmatrix}$$

будем называть *тождественной перестановкой*. Очевидно, что $ef = fe = f$ для произвольной перестановки $f \in S_n$. Легко также заметить, что каждая перестановка $f \in S_n$ однозначно определяет перестановку f^{-1} , такую что $ff^{-1} = f^{-1}f = e$. Будем называть ее *перестановкой, обратной к f* . Чтобы ее определить, достаточно поменять местами строки в записи перестановки f . Например, для

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}$$

получаем

$$f^{-1} = \begin{pmatrix} 3 & 4 & 2 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 2 & 5 \end{pmatrix}$$

Из наших рассуждений следует, что для произвольных перестановок $f, g, h \in S_n$ выполняются условия

$$(fg)h = f(gh) \tag{1.6}$$

$$fe = ef = f \tag{1.7}$$

$$ff^{-1} = f^{-1}f = e \quad (1.8)$$

Чтобы отразить этот факт, будем говорить, что S_n образует *группу* относительно операции суперпозиции. Эту группу будем называть *симметрической группой степени n* . Произвольное подмножество $G \subseteq S_n$, для которого выполнены условия

$$f, g \in G \Rightarrow fg \in G$$

$$f \in G \Rightarrow f^{-1} \in G$$

будем называть *группой перестановок степени n* .

Каждую перестановку $f \in S_n$ можно представить графически с помощью ориентированного графа с множеством вершин $X = \{1, \dots, n\}$, в котором $x \rightarrow y$ тогда и только тогда, когда $f(x) = y$. Из каждой вершины x выходит в точности одно ребро, а именно $\langle x, f(x) \rangle$. Подобным же образом убеждаемся, что единственным ребром, входящим в вершину x , является $\langle f^{-1}(x), x \rangle$. Легко заметить, что, выходя из произвольной вершины x_0 и рассматривая по очереди вершины $x_1 = f(x_0)$, $x_2 = f(x_1)$, \dots , мы дойдем после конечного числа шагов до вершины x_0 , т.е. $x_i = f(x_{i-1}) = x_0$ для некоторого $i \geq 1$. Отсюда следует, что наш граф состоит из некоторого числа элементарных циклов с различными множествами вершин, в сумме дающих все множество X . Предположим, что в этом разложении появляются k циклов

$$a_0^{(i)} \rightarrow a_1^{(i)} \rightarrow \dots \rightarrow a_{n_i-1}^{(i)} \rightarrow a_0^{(i)}, \quad i = 1, \dots, k.$$

Каждому циклу соответствует перестановка

$$f_i = \left[a_0^{(i)} a_1^{(i)} \dots a_{n_i-1}^{(i)} \right],$$

называемая также *циклом (длины n_i)*, которая определяется следующим образом:

$$\begin{aligned} f_i \left(a_0^{(i)} \right) &= a_1^{(i)}, f_i \left(a_1^{(i)} \right) = a_2^{(i)}, \dots, f_i \left(a_{n_i-1}^{(i)} \right) = a_0^{(i)}, \\ f_i(x) &= x \quad \text{для} \quad x \in X \setminus \left\{ a_0^{(i)}, a_1^{(i)}, \dots, a_{n_i-1}^{(i)} \right\} \end{aligned}$$

Нашу перестановку можно представить в виде суперпозиции циклов

$$f = \left[a_0^{(1)} a_1^{(1)} \dots a_{n_1-1}^{(1)} \right] \left[a_0^{(2)} a_1^{(2)} \dots a_{n_2-1}^{(2)} \right] \dots \left[a_0^{(k)} a_1^{(k)} \dots a_{n_k-1}^{(k)} \right]$$

Такое представление перестановки будем называть *разложением на циклы*. Будем говорить, что перестановка f есть перестановка типа $\langle \lambda_1, \dots, \lambda_i \rangle$, если она содержит в разложении на циклы в точности λ_i циклов длины i , $i = 1, 2, \dots, n$.

Тип $\langle \lambda_1, \dots, \lambda_n \rangle$ обычно записывается символически $1^{\lambda_1} \dots n^{\lambda_n}$ (если $\lambda_i = 0$, то опускается). Например, перестановка

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 1 & 4 & 2 & 3 & 6 \end{pmatrix}$$

имеет следующее разложение на циклы:

$$f = [1763][25][4],$$

следовательно, она имеет тип $1^1 2^1 4^1$. Это проиллюстрировано на рис. 1.3. Пару $\langle a_i, a_j \rangle$, $i < j$, будем называть *инверсией* перестановки $\langle a_1, \dots, a_n \rangle$, если $a_i > a_j$. Для произвольной перестановки $f \in S_n$ обозначим через $I(f)$ число ее инверсий, а также определим знак этой перестановки следующим образом:

$$\text{sgn}(f) = (-1)^{I(f)}.$$

Перестановку f назовем *четной*, если $\text{sgn}(f) = 1$, и *нечетной*, если $\text{sgn}(f) = -1$. Проиллюстрируем эти понятия на ряде примеров. Тожественная перестановка $\langle 1, \dots, n \rangle$ не содержит никаких инверсий, и, следовательно, $I(e) = 0$, и для любого n эта перестановка будет четной. Перестановка $\langle n, n - 1, \dots, 1 \rangle$ содержит $n(n - 1)/2$ инверсий (число всех пар $\langle i, j \rangle$, $i \neq j$, равно $[n]_2 = n(n - 1)$, причем каждой паре $\langle i, j \rangle$, $i < j$, соответствует пара $\langle j, i \rangle$, $j > i$). Таким образом, наша перестановка будет четной для n вида $4k$ или $4k + 1$ и нечетной в остальных случаях. Перестановка $\langle 2, 4, 3, 5, 1 \rangle$ имеет следующие инверсии: $\langle 2, 1 \rangle$, $\langle 4, 3 \rangle$, $\langle 4, 1 \rangle$, $\langle 3, 1 \rangle$, $\langle 5, 1 \rangle$, следовательно, является нечетной.

Знак перестановки можно определить с помощью непосредственного подсчета всех инверсий, однако такой алгоритм в общем случае требует количества шагов такого же порядка, что и число инверсий, т.е. по меньшей мере $\Omega(n^2)$. Опишем теперь алгоритм сложности $O(n)$. Для этой цели нам понадобятся несколько лемм.

Произвольную перестановку, являющуюся циклом длины 2, будем называть *транспозицией*. Важную роль в дальнейших рассуждениях будут играть транспозиции соседних элементов, т.е. транспозиции вида $[i \ i + 1]$.

Лемма 1.5. *Произвольную перестановку $f \in S_n$ можно представить в виде суперпозиции $I(f)$ транспозиций соседних элементов.*

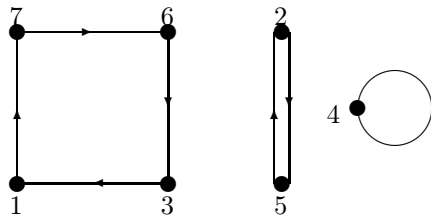


Рис. 1.3: Разложение перестановки $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 1 & 4 & 2 & 3 & 6 \end{pmatrix}$ на циклы

Доказательство. Заметим прежде всего, что если f имеет вид $\langle a_1, \dots, a_n \rangle$ и $t = [i \ i+1]$, то суперпозиция ft имеет вид $\langle a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n \rangle$. Обозначим через r_i число инверсий, расположенных перед элементом i :

$$r_i = |\{j : j < i \wedge a_j > a_i\}|.$$

Легко заметить, что в $\langle a_1, \dots, a_n \rangle$ мы можем переставить элемент на первую позицию, произведя r_1 транспозиций соседних элементов, затем элемент 2 переставить на вторую позицию, произведя r_2 транспозиций соседних элементов и т.д. В конце концов после $r_1 + \dots + r_n = I(f)$ шагов получим последовательность $\langle 1, \dots, n \rangle$. Это означает, что $ft_1 \dots t_{I(f)} = e$, где $t_1, \dots, t_{I(f)}$ — транспозиции соседних элементов. Итак,

$$f = (t_1 \dots t_{I(f)})^{-1} = t_{I(f)}^{-1} \dots t_1^{-1},$$

что завершает доказательство леммы, где $t^{-1} = t$ для произвольной транспозиции. ■

Лемма 1.6. *Для произвольных перестановок $f, g \in S_n$*

$$\operatorname{sgn}(fg) = \operatorname{sgn}(f) \operatorname{sgn}(g).$$

Доказательство. Положим сначала, что g есть транспозиция вида $t = [i \ i+1]$. Если $f = \langle a_1, \dots, a_n \rangle$, то

$$ft = \langle a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n \rangle$$

и, очевидно,

$$I(ft) = \begin{cases} I(f) + 1, & \text{если } a_i < a_{i+1} \\ I(f) - 1, & \text{если } a_i > a_{i+1} \end{cases}$$

В обоих случаях $\operatorname{sgn}(ft) = -(-1)^{I(f)} = -\operatorname{sgn}(f)$. Произвольную перестановку g мы можем на основе предыдущей леммы представить в виде t_1, \dots, t_k , где t_1, \dots, t_k суть транспозиции соседних элементов и $k = I(g)$. Имеем

$$\begin{aligned} \operatorname{sgn}(fg) &= \operatorname{sgn}(ft_1, \dots, t_k) = -\operatorname{sgn}(ft_1, \dots, t_{k-1}) = \\ &= (-1)^k \operatorname{sgn}(f) = \operatorname{sgn}(g) \operatorname{sgn}(f). \end{aligned}$$

Лемма 1.7. *Каждая транспозиция есть нечетная перестановка. Вообще знак произвольного цикла длины k равен $(-1)^k$.*

Доказательство. Первая часть леммы вытекает из того факта, что последовательность $\langle 1, \dots, i-1, j, i+1, \dots, j-1, j+1, \dots, n \rangle$ можно преобразовать в последовательность $\langle 1, \dots, n \rangle$, произведя сперва $j-i$ транспозиций $[j-i$

$[1 \ j], [j-2 \ j-1], \dots, [i \ i+1]$, а затем $(j-i)-1$ транспозиций $[i+1 \ i+2], [i+2 \ i+3], \dots, [j-1 \ j]$. Это означает, что транспозиция $[i \ j]$ может быть представлена в виде суперпозиции $(j-i) + (j-i) - 1 = 2(j-i) - 1$ транспозиций соседних элементов. Согласно предыдущим леммам, знак нашей транспозиции $[i \ j]$ равен $(-1)^{2(j-i)-1} = -1$.

Вторая часть леммы есть следствие первой и того факта, что произвольный цикл $[a_1 \dots a_k]$ есть суперпозиция $k-1$ транспозиций:

$$[a_1 \dots a_k] = [a_1 \ a_2][a_2 \ a_3] \dots [a_{k-1} \ a_k]$$

Лемма 1.8. *Знак произвольной перестановки f типа $1^{\lambda_1} \dots n^{\lambda_n}$ определяется формулой*

$$\text{sgn } f = (-1)^{\sum_{j=1}^{\lfloor n/2 \rfloor} \lambda_{2j}}$$

Доказательство. Если f имеет в разложении на циклы точно λ_i циклов длины i , то по предыдущим леммам

$$\text{sgn}(f) = \prod_{i=1}^n [(-1)^{i-1}]^{\lambda_i} = \prod_{i=1}^{\lfloor n/2 \rfloor} (-1)^{\lambda_{2i}} = (-1)^{\lambda_2 + \lambda_4 + \dots}$$

Заметим, что о числе инверсий перестановки множества X мы можем говорить только в случае $X = \{1, \dots, n\}$, более общо: когда на множестве X определен линейный порядок. Однако знак перестановки зависит только от ее типа и, следовательно, не зависит от того, как упорядочено множество X . Обещанный эффективный метод определения знака перестановки основан на знании типа этой перестановки и на использовании леммы 1.8.

Алгоритм 1.9. (Определение знака перестановки)

Данные: Произвольная перестановка $f \in S_n$, заданная в виде последовательности $P[1], \dots, P[n]$ ($P[i] = f(i)$).

Результат: По завершении работы алгоритма $s = \text{sgn}(f)$.

```

1 begin
2   s := 1;
3   for i := 1 to n do НОВЫЙ[i] := истина;
4   for i := 1 to n do
5     if НОВЫЙ[i] then (*найден цикл, содержащий i*)
6       begin j := P[i];
7         while j ≠ i do
```

```

8           begin НОВЫЙ[i]:=ложь; s := -s; j := P[j];
9           end;
10          end;
11 end

```

Этот алгоритм просматривает последовательно позиции $1, \dots, n$ перестановки (цикл 4) и каждый раз, когда элемент $P[i]$ не был еще проанализирован (НОВЫЙ[i] = истина), выявляется цикл, к которому этот элемент принадлежит (блок 6). Заметим, что если этот цикл имеет длину k , то цикл 7 исполняется $k - 1$ раз. При последовательных исполнениях этого цикла знак переменной s изменяется на противоположный тогда и только тогда, когда k четно. Первоначальное значение переменной s равно единице, и, следовательно, после обнаружения всех циклов $s = (-1)^p$, где p есть число циклов четной длины. По лемме 1.8 имеем $s = \text{sgn}(f)$.

Легко также убедиться, что число шагов алгоритма в точности равно n для произвольной перестановки $f \in S_n$. Чтобы это доказать, заметим, что суммарное число шагов, исполняемых в цикле 4, не считая шагов во внутреннем блоке 6, есть $O(n)$. Суммарное число шагов, исполняемых в блоке 6 в процессе работы алгоритма, равно сумме длин всех циклов, что в свою очередь равно $O(n)$. Это дает общую сложность $O(n)$.

1.4 Генерирование перестановок

Займемся алгоритмом генерирования всех $n!$ перестановок n -элементного множества. Этой проблеме посвящено много публикаций (см., например, статью [60]). Она имеет давнюю историю, ее возникновение можно отнести к началу XVII века, когда в Англии зародилось особое искусство колокольного боя, основанного, если говорить упрощенно, на выбивании на n разных колоколах всех $n!$ перестановок [14], [74]. Перестановки эти следовало выбивать «по памяти», что способствовало разработке сторонниками этого искусства первых простых методов систематического перечисления всех перестановок (без повторов). Некоторые из этих незаслуженно забытых методов были переоткрыты в настоящее время в связи с появлением цифровых машин. Это искусство — хотя и мало известное в Польше — просуществовало до наших дней, поскольку знаменитая «Книга рекордов Гиннеса» [29] содержит упоминание о выбивании всех $8! = 40320$ перестановок на 8 колоколах в 1963 году; установление этого рекорда потребовало 17 часов 58 1/2 минут! Конечно, использование цифровых машин позволяет генерировать перестановки значительно быстрее, однако разница не так уж велика, как можно было бы подумать — за 18 часов даже самая быстрая

машина не сможет получить все перестановки n -элементного множества, если $n > 13$ (с другой стороны, перестановки 8-элементного множества можно получить в течение доли секунды). Это простое следствие того факта, что $n!$ растет весьма быстро с ростом n .

В этом разделе мы опишем три разных метода генерирования последовательности всех $n!$ перестановок n -элементного множества. Будем предполагать, что элементы нашего множества запоминаются в виде элементов массива $P[1], \dots, P[n]$. Во всех трех методах элементарной операцией, которая применяется к массиву P , является поэлементная транспозиция, т.е. обмен значениями переменных $P[i]$ и $P[j]$, где $1 \leq i, j \leq n$. Эту операцию будем обозначать через $P[i] := P[j]$. Очевидно, что она эквивалентна последовательности команд

$$rot := P[i]; P[i] := P[j]; P[j] := rot,$$

где rot есть некоторая вспомогательная переменная.

Первый из методов, который мы опишем, легче всего понять, если в качестве переставляемых элементов взять числа $1, 2, \dots, n$. На множестве всех перестановок — более общо: на множестве всех последовательностей длины n с элементами из множества $X = \{1, \dots, n\}$ — определяется лексикографический порядок:

$$\langle x_1, x_2, \dots, x_n \rangle < \langle y_1, y_2, \dots, y_n \rangle \Leftrightarrow \exists k \geq 1 (x_k \leq y_k \wedge (x_i = y_i) \forall i < k)$$

Заметим, что если вместо чисел $1, 2, \dots, n$ взять буквы a, b, \dots, z с естественным порядком $a < b < c < \dots < z$, то лексикографический порядок определяет стандартную последовательность, в которой слова длины n появляются в словаре. Подобным же образом определяется антилексикографический порядок, обозначаемый через $<'$, с той разницей, что как очередность позиций в последовательности, так и упорядочение элементов множества X обратны по отношению к исходным:

$$\langle x_1, x_2, \dots, x_n \rangle <' \langle y_1, y_2, \dots, y_n \rangle \Leftrightarrow \exists k \geq 1 (x_k > y_k \wedge (x_i = y_i) \forall i < k)$$

Для примера приведем перестановки множества $X = \{1, 2, 3\}$ в лексикографическом (а) и антилексикографическом (б) порядке:

(а)	(б)
1 2 3	1 2 3
1 3 2	2 1 3
2 1 3	1 3 2
2 3 1	3 1 2
3 1 2	2 3 1
3 2 1	3 2 1

Алгоритм генерирования перестановок в антилексикографическом порядке сформулировать немного удобнее. Заметим с этой целью, что последовательность перестановок множества $\{1, \dots, n\}$ в этом случае имеет следующие свойства, вытекающие непосредственно из определения:

- (A1) В первой перестановке элементы идут в растущей последовательности, в последней — в убывающей; другими словами, последняя перестановка — обращение первой.
- (A2) Нашу последовательность можно разделить на n блоков длины $(n - 1)!$, соответствующих убывающим значениям элемента в последней позиции. Первые $n - 1$ позиций блока, содержащего элемент p в последней позиции, определяют последовательность перестановок множества $\{1, \dots, n\} \setminus \{p\}$ в антилексикографическом порядке.

Эти свойства определяют следующий простой рекурсивный алгоритм:

Алгоритм 1.10. (Генерирование всех последовательностей в антилексикографическом порядке.)

Данные: n .

Результат: Последовательность перестановок множества $\{1, \dots, n\}$ в антилексикографическом порядке.

```

1. procedure REVERSE( $m$ );
   (*обращение последовательности  $P[1], \dots, P[m]$ , массив  $P$  — глобальный*)
2. begin  $i := 1; j := m;$ 
3.   while  $i < j$  do
4.     begin  $P[i] := P[j]; i := i + 1; j := j - 1$ 
5.     end
6. end; (*REVERSE*)
7. procedure ANTYLEX( $m$ ); (*массив  $P$  — глобальный*)
8. begin
9.   if  $m = 1$  then (* $P[1], \dots, P[n]$  содержит новую перестановку *)
10.     $write(P[1], \dots, P[n])$ 
11.   else
12.     for  $i := 1$  to  $m$  do
13.       begin ANTYLEX( $m - 1$ );
14.       if  $i < m$  then
15.         begin  $P[i] := P[m]; REVERSE(m - 1)$ 
16.         end
17.       end
18. end; (*ANTYLEX*)

```

19. **begin** (* главная программа*)
20. **for** $i := 1$ **to** n **do** $P[i] := i$;
21. $ANTYLEX(n)$
22. **end**

Чтобы понять, как работает этот алгоритм, отметим прежде всего, что выполнение процедуры $REVERSE(m)$ приводит к обращению очередности в последовательности элементов $P[1], \dots, P[m]$. Чтобы доказать правильность алгоритма, достаточно показать индукцией по m , что если $P[1] < \dots < P[m]$, то вызов $ANTYLEX(m)$ приводит к генерированию всех перестановок множества $\{P[1], \dots, P[m]\}$ в антилексикографическом порядке (при неизменных значениях $P[m+1], \dots, P[n]$). Предположим, что $P[i] = a_i$, $1 \leq i \leq m$, $a_1 < \dots < a_m$, и рассмотрим цикл 12. Эффект выполнения первой итерации этого цикла будет следующий:

$P[1]$	$P[2]$	\dots	$P[m-2]$	$P[m-1]$	$P[m]$	
a_1	a_2		a_{m-2}	a_{m-1}	a_m	
a_{m-1}	a_{m-2}		a_2	a_1	a_m	(после выполнения $ANTYLEX(m-1)$)
a_m	a_{m-2}		a_2	a_1	a_{m-1}	(после транспозиции $P[1] := P[m]$)
a_1	a_2		a_{m-2}	a_m	a_{m-1}	(после выполнения $REVERSE(m-1)$)

(Мы воспользовались здесь индуктивным предположением о том, что $ANTYLEX(m-1)$ корректно генерирует все перестановки элементов a_1, \dots, a_{m-1} в антилексикографическом порядке) Аналогичным способом, индукцией по i , доказываем, что i -я итерация цикла 12 приводит к генерированию всех перестановок элементов $a_1, \dots, a_{m-i}, a_{m-i+2}, \dots, a_m$ при $P[m] = a_{m-i+1}$. Согласно свойству A2 это означает, что $ANTYLEX(m)$ генерирует все перестановки элементов a_i, \dots, a_m в антилексикографическом порядке.

Следует отметить, что условие $P[i] = i$, $1 \leq i \leq n$, в начале работы программы (см. строка 20) было существенно только для облегчения понимания работы алгоритма. Сам алгоритм сформулирован в терминах позиций, значения которых подвергаются изменениям, и в нем ни в коей мере не используется содержание переменных $P[1], \dots, P[n]$. Рассмотрим теперь, сколько транспозиций выполняет алгоритм 1.10 при генерировании каждой следующей перестановки. Легко отметить, что это число есть величина переменная: каждая вторая перестановка получилась за счет одной транспозиции $P[1] := P[2]$, но наряду с ними имеются и такие, которые требуют $\lfloor (n-1)/2 \rfloor + 1 = \lfloor (n+1)/2 \rfloor$ транспозиций. Хотя среднее число транспозиций, приходящихся на каждую перестановку, невелико (см. задачу 1.13), однако в некоторых приложениях лучшим был бы алгоритм, в котором каждая следующая перестановка образуется из предыду-

щей с помощью выполнения одной транспозиции. Это может оказаться существенным в ситуации, когда с каждой перестановкой связаны некоторые вычисления и когда существует возможность использования частичных результатов, полученных для предыдущей перестановки, если последовательные перестановки мало отличаются друг от друга.

Покажем сейчас, что такой алгоритм действительно возможен. Его основную схему можно описать с помощью следующей рекурсивной процедуры:

```

1. procedure PERM( $m$ ); (*массив  $P$  — глобальный*)
2. begin
3.   if  $m = 1$  then (* $P[1], \dots, P[n]$  содержит новую перестановку*)
4.     write( $P[1], \dots, P[n]$ )
5.   else
6.     for  $i := 1$  to  $m$  do
7.       begin PERM( $m - 1$ );
8.         if  $i < m$  then  $P[B[m, i]] := P[m]$ 
9.       end
10. end

```

Задачей этой процедуры является генерирование всех перестановок элементов $P[1], \dots, P[n]$ через последовательное генерирование всех перестановок элементов $P[1], \dots, P[n-1]$ и замену элемента $P[n]$ на один из элементов $P[1], \dots, P[n-1]$, определяемый из массива $B[m, i]$, $1 \leq i < m \leq n$. Очевидно, что для того чтобы эта схема работала правильно, мы должны определить массив B так, чтобы гарантировать, что каждая транспозиция $P[B[m, i]] := P[m]$ в строке 8 вводила бы новый элемент в $P[n]$. Представим теперь алгоритм, в котором значение $B[m, i]$ вычисляется динамически как функция от m и i [44]

Алгоритм 1.11 (Генерирование всех перестановок за минимальное число транспозиций.)

Данные: n .

Результат: Последовательность перестановок множества $\{1, \dots, n\}$, в котором каждая последующая перестановка образуется из предыдущей путем выполнения одной транспозиции.

```

1. procedure  $B(m, i)$ ;
2. begin
3.   if ( $m \bmod 2 = 0$ ) and ( $m > 2$ ) then
4.     if  $i < m - 1$  then  $B := i$ 
5.     else  $B := m - 2$ 
6.     else  $B := m - 1$ 
7. end; (* $B$ *)
8. procedure PERM( $m$ ); (* массив  $P$  — глобальный*)

```

9. begin

```

10.   if  $m = 1$  then (* $P[1], \dots, P[m]$  — новая перестановка*)
11.     write ( $P[1], \dots, P[n]$ )
12.   else
13.     for  $i := 1$  to  $m$  do
14.       begin  $PERM(m - 1)$ ;
15.         if  $i < m$  then  $P[B(m, i)] := P[m]$ 
16.       end
17.   end; (* $PERM^*$ *)
18. begin (* главная программа*)
19.   for  $i := 1$  to  $n$  do  $P[i] := i$ ;
20.    $PERM(n)$ 
21. end

```

Отметим, что для нечетного m транспозиция в строке 15 сводится к $P[m - 1] := P[m]$, для каждого $i < m$ для четного m значение $P[m]$ меняется последовательно на значения $P[1], P[2], \dots, P[m - 3], P[m - 2], P[m - 1]$ ($P[1]$ для $m - 2$).

Для каждого $m \geq 1$ определим перестановку φ_m следующим образом:

$\varphi_m(i)$ = индекс j , такой что $P[j]$ содержит начальное значение переменной $P[i]$ после выполнения $PERM(m)$.

Например, если сначала переменные $P[1], \dots, P[4]$ содержат последовательность 1 2 3 4, то легко убедиться, что после выполнения $PERM(4)$ эта последовательность изменится на 4 1 2 3. Это означает, что φ_4 является циклом 1 2 3 4.

Покажем теперь, что алгоритм 1.11 корректен; точнее, докажем для каждого $m \geq 1$ выполнимость следующего условия:

Условие W_m . Выполнение $PERM(m)$ вызывает генерирование всех перестановок элементов $P[1], \dots, P[m]$, причем φ_m есть транспозиция $[m \ m - 1]$, если m нечетное ($m > 1$), или цикл $[1 \ 2 \dots m]$, если m четно.

Доказательство осуществляется индукцией по m так, как это обычно делается при доказательстве корректности рекурсивных алгоритмов. Легко убедиться непосредственно, что условия W_1 и W_2 выполняются. Предположим, что $m \geq 3$. Докажем выполнимость W_m , предполагая истинность W_{m-1} . Доказательство разобьем на две части в зависимости от того, четное m или нечетное.

Случай 1, m нечетное. В силу индуктивного предположения выполнение $PERM(m-1)$ в строке 14 приводит каждый раз к сдвигу значений $P[1], \dots, P[m-1]$ вдоль цикла $[1 \ 2 \dots m - 1]$. Таким образом, транспозиция $P[m] := P[m - 1]$ в строке 15 выбирает каждый раз различные элементы в $P[m]$. Если вначале $P[i] = a_i$, $1 \leq i \leq m$, то в $P[m]$ помещаются поочередно элементы a_m, a_{m-2} ,

Число выполненных итераций цикла 13	$P[1]$	$P[2]$	$P[m-3]$	$P[m-2]$	$P[m-1]$	$P[m]$
0	a_1	a_2	a_{m-3}	a_{m-2}	a_{m-1}	a_m
1	a_m	a_2	a_{m-3}	a_{m-1}	a_{m-2}	a_1
2	a_m	a_1	a_{m-3}	a_{m-2}	a_{m-1}	a_2
...						
$m-3$	a_m	a_1	a_{m-4}	a_{m-1}	a_{m-2}	a_{m-3}
$m-2$	a_m	a_1	a_{m-4}	a_{m-3}	a_{m-1}	a_{m-2}
$m-1$	a_m	a_1	a_{m-4}	a_{m-2}	a_{m-3}	a_{m-1}
m	a_m	a_1	a_{m-4}	a_{m-3}	a_{m-2}	a_{m-1}

Таблица 1.1: Изменение значений переменных $P[1], \dots, P[m]$ во время выполнения процедуры $PERM(m)$ при m четном

$a_{m-3}, \dots, a_1, a_{m-1}$. Следовательно, $PERM(m)$ генерирует все перестановки элементов $P[1], \dots, P[m]$.

Заметим, что сдвиг $P[1], \dots, P[m-1]$ вдоль цикла $[1 \ 2 \dots m-1]$, а затем выполнение транспозиции $P[m] := P[m-1]$ эквивалентно сдвигу $P[1], \dots, P[m]$ вдоль цикла $[1 \ 2 \dots m-3 \ m-2 \ m \ m-1]$ длины m . Если бы транспозиция в строке 15 выполнялась для каждого $i \leq m$, то выполнение цикла 13 вызвало бы возвращение всех элементов на свои исходные места. В действительности же последняя транспозиция для $i = n$ не выполняется. Следовательно, $\varphi_m = [m, m-1]$.

Случай 2, m четное. Проследим за содержанием массива P во время исполнения $PERM(m)$. Изменения, которым подвергается он, представлены в табл. 1.1. Из этой таблицы следует, что каждый элемент появляется в $P[m]$, следовательно, $PERM(m)$ генерирует все перестановки, и что φ_m является циклом $[1 \ 2 \dots m]$. Доказательство правильности алгоритма тем самым закончено.

Алгоритм 1.11 обладает одним свойством, которое может оказаться полезным в некоторых приложениях. Представим себе, что мы ищем перестановку, удовлетворяющую определенным условиям. В такой ситуации, если для некоторого m значения $P[m+1], \dots, P[n]$ не удовлетворяют требуемым ограничениям, то нет необходимости в вызове $PERM(m)$ и в прохождении всех $m!$ перестановок элементов $P[1], \dots, P[m]$. Эти перестановки мы можем опустить, выполняя перестановку элементов $P[1], \dots, P[m]$, обозначаемую через φ_m . Заметим, что перестановки φ_m имеют в нашем случае очень простую форму.

Последний алгоритм генерирования перестановок, который мы здесь представляем, строит последовательность, в которой разница между двумя последовательными перестановками еще меньше: каждая следующая образуется из предыдущей с помощью однократной транспозиции соседних элементов. Этот

(а)	(б)	(в)
1 2 3 4	1 2 3 4	1 2 3 4
2 1 3 4	2 1 3 4	2 1 3 4
1 3 2 4	2 3 1 4	2 3 1 4
3 1 2 4	3 2 1 4	2 3 4 1
2 3 1 4	3 1 2 4	3 2 4 1
3 2 1 4	1 3 2 4	3 2 1 4
1 2 4 3	4 3 2 1	3 1 2 4
2 1 4 3	3 4 2 1	1 3 2 4
1 4 2 3	3 2 4 1	1 3 4 2
4 1 2 3	2 3 4 1	3 1 4 2
2 4 1 3	2 4 3 1	3 4 1 2
4 2 1 3	4 2 3 1	3 4 2 1
1 3 4 2	4 1 3 2	4 3 2 1
3 1 4 2	1 4 3 2	4 3 1 2
1 4 3 2	1 3 4 2	4 1 3 2
4 1 3 2	3 1 4 2	1 4 3 2
3 4 1 2	3 4 1 2	1 4 2 3
4 3 1 2	4 3 1 2	4 1 2 3
2 3 4 1	4 2 1 3	4 2 1 3
3 2 4 1	2 4 1 3	4 2 3 1
2 4 3 1	2 1 4 3	2 4 3 1
4 2 3 1	1 2 4 3	2 4 1 3
3 4 2 1	1 4 2 3	2 1 4 3
4 3 2 1	4 1 2 3	1 2 4 3

Рис. 1.4: Последовательности перестановок, полученные с помощью: а) алгоритма 1.10; б) алгоритма 1.11; в) алгоритма 1.12

алгоритм обычно приписывается Джонсону [38] и Троттеру [68]. Его идею легче всего проиллюстрировать на примере. Предположим, что уже построена последовательность перестановок элементов $2, 3, \dots, n$, обладающая этим свойством, например: $2\ 3, 3\ 2$ для $n = 3$. Тогда требуемую последовательность перестановок элементов $1, 2, \dots, n$ получим, вставляя элемент 1 всеми возможными способами в каждую перестановку элементов $2, 3, \dots, n$. В нашем случае получаем

$$\begin{array}{c}
 1\ 2\ 3 \\
 2\ 1\ 3 \\
 2\ 3\ 1 \\
 3\ 2\ 1 \\
 3\ 1\ 2 \\
 1\ 3\ 2
 \end{array}$$

В общем виде элемент 1 помещается между первой и последней позициями попеременно вперед и назад $(n - 1)!$ раз. На основе этой конструкции можно

легко получить рекурсивный алгоритм, генерирующий требуемую последовательность перестановок для произвольного n . Однако этот метод, примененный непосредственно, имеет недостаток: последовательность перестановок строится «целиком» и только после окончания всего построения ее можно считать. Очевидно, что решение такого типа потребовало бы огромного объема памяти. Поэтому мы сейчас покажем не рекурсивный вариант этого алгоритма. В этом варианте для каждого i , $1 \leq i < n$, булева переменная $PR[i]$ содержит информацию о том, переносится ли элемент i вперед ($PR[i]=\text{истина}$) или же назад ($PR[i]=\text{ложь}$), переменная же $C[i]$ показывает, какую из возможных $n - i - 1$ позиций элемент 1 занимает относительно элементов $i + 1, \dots, n$ на своем пути вперед или назад. Позицию элемента 1 в таблице P определяем на основании его позиции в блоке, содержащем $i, i + 1, \dots, n$, а также на основании числа элементов из $1, 2, \dots, i - 1$, которые находятся слева от этого блока. Это число, будучи значением переменной x , вычисляется как число элементов $j < i$, которые, двигаясь назад, достигли бы своего крайнего левого положения ($C[j] = n - j + 1$, $PR[j]=\text{ложь}$). Каждая новая перестановка образуется транспозицией самого меньшего из элементов j , который не находится в граничном положении (т.е., $C[j] < n - j + 1$) с его левым или правым соседом. Это реализует приведенный ниже алгоритм. Доказательство его правильности предоставляется читателю.

Алгоритм 1.12. (Генерирование всех перестановок с минимальным числом транспозиций соседних элементов)

Данные: n .

Результат: Последовательность перестановок множества $\{1, \dots, n\}$, в которой каждая последующая образуется в результате выполнения однократной транспозиции соседних элементов.

```

1. begin
2.   for  $i := 1$  to  $n$  do
3.     begin  $P[i] := i$ ;  $C[i] := 1$ ;  $PR[i] := \text{истина}$ ;
4.     end;
5.      $C[n] := 0$ ; (*так, чтобы  $C[i] \neq n - i + 1$  в строке 10 для  $i = n$ *)
6.     write( $P[1], \dots, P[n]$ );
7.      $i := 1$ ;
```

```

8.   while  $i < n$  do
9.       begin  $i := 1; x := 0;$ 
10.          while  $C[i] = n - i + 1$  do
11.              begin  $PR[i] := \text{not } PR[i]; C[i] := 1;$ 
12.                  if  $PR[i]$  then  $x := x + 1;$ 
13.                       $i := i + 1$ 
14.                  end;
15.              if  $i < n$  then
16.                  begin (* выполнение транспозиции *)
17.                      if  $PR[i]$  then  $k := C[i] + x;$ 
18.                          else  $k := n - i + 1 - C[i] + x;$ 
19.                           $P[k] := P[k + 1];$ 
20.                          write( $P[1], \dots, P[n]$ );
21.                           $C[i] := C[i] + 1$ 
22.                  end
23.              end
24. end

```

Отметим, что в этом алгоритме, как и в двух предыдущих, не используется ни в какой мере знание значений переменных $P[1], \dots, P[n]$. Это очевидно, так как данные переменные появляются только в строке 19 (если не считать инициализации в строке 3 и вывода результатов).

На рис. 1.4 представлены последовательности перестановок, полученные для $n = 4$ при помощи алгоритмов 1.10, 1.11 и 1.12.

Укажем в заключение одну любопытную интерпретацию последовательности, полученной с помощью алгоритма 1.12. Рассмотрим с этой целью граф G_n , вершины которого соответствуют всем перестановкам множества $\{1, \dots, n\}$ и в котором две вершины, соответствующие перестановкам f и g , соединены ребром тогда и только тогда, когда g образуется из f однократной транспозицией соседних элементов (таким образом, каждая вершина соединена в точности с $n - 1$ другими вершинами). Нетрудно заметить, что последовательность перестановок, полученная при помощи алгоритма 1.12, соответствует гамильтонову пути в G_n , т.е. пути, содержащему каждую вершину графа в точности один раз. Это проиллюстрировано на рис. 1.5 для $n = 3$ и $n = 4$.

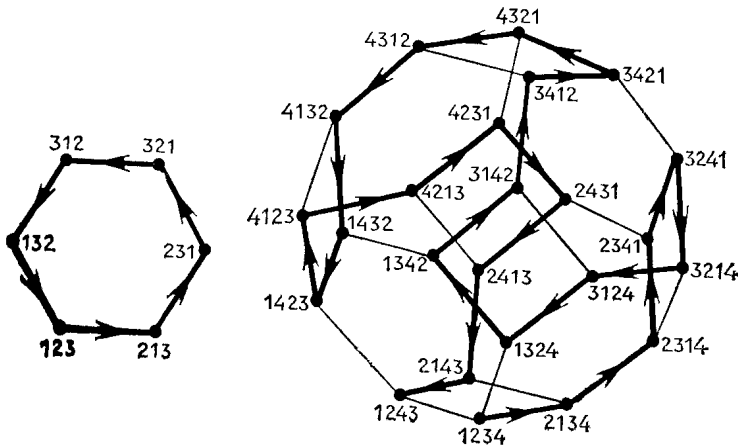


Рис. 1.5: Интерпретация последовательности перестановок, полученной с помощью алгоритма 1.12

1.5 Подмножества множества, множества с повторениями, генерирование подмножеств множества

Каждое n -элементное множество $X = \{x_1, \dots, x_n\}$ имеет в точности 2^n подмножеств. Чтобы убедиться в этом, достаточно каждому подмножеству $Y \subseteq X$ сопоставить бинарную последовательность (т.е. с членами 0 или 1) b_1, b_2, \dots, b_n , определяемую следующим образом:

$$b_i = \begin{cases} 0, & \text{если } x_i \notin Y, \\ 1, & \text{если } x_i \in Y. \end{cases}$$

Тем самым мы устанавливаем взаимно однозначное соответствие между элементами множества $\wp(X)$ всех подмножеств множества X и всеми бинарными последовательностями длины n . Число таких последовательностей в точности равно 2^n , а значит в силу установленного соответствия имеет место равенство $|\wp(X)| = 2^n$.

Определенная нами последовательность $b_1 b_2 \dots b_n$ становится удобным машинным представлением подмножества Y , особенно в ситуации, когда мощность множества X невелика и последовательность $b_1 b_2 \dots b_n$ может быть закодирована в виде одного машинного слова. Такое представление подсказывает простой

метод генерирования всех подмножеств. Достаточно заметить, что каждая бинарная последовательность $b_{n-1}b_{n-2} \dots b_0$ соответствует взаимно однозначному целому числу из интервала $0 \leq r \leq 2^n - 1$, а именно числу $r = \sum_{i=0}^{n-1} b_i 2^i$, для которого $b_{n-1}b_{n-2} \dots b_0$ есть двоичное представление. Это число будем обозначать через $[b_{n-1}b_{n-2} \dots b_0]$. Таким образом, мы можем последовательно получать все числа из интервала $0 \leq r \leq 2^n - 1$ (начиная, например, с 0 и добавляя 1 на каждом шаге), а их двоичные представления дадут все подмножества n -элементного множества. Этот метод особенно выгоден для реализации на внутреннем языке машины.

В некоторых ситуациях нам важно, чтобы каждое последующее полученное подмножество наименьшим образом отличалось от предыдущего. Объяснение полезности алгоритма такого типа совершенно аналогично случаю алгоритмов 1.11 и 1.12 генерирования перестановок: при проведении реальных вычислений, связанных с каждым полученным подмножеством, имеется возможность использовать частичные результаты, полученные для предыдущего подмножества. Заметим, что в случае описанного выше алгоритма, основанного на двоичном представлении чисел, последовательно получаемые подмножества могут сильно отличаться друг от друга: например, после $(n - 1)$ -элементного множества, отвечающего числу 011...1, идет одноэлементное множество, отвечающее числу 100...0.

Опишем теперь другой метод, при котором каждое последующее подмножество получается из предыдущего добавлением или удалением одного элемента. Опирается он на следующее простое наблюдение. Если последовательность C_1, C_2, \dots, C_m содержит все $m = 2^k$ бинарных последовательностей длины k , причем C_i отличается от C_{i+1} в точности в одной координате ($i = 1, \dots, m - 1$), то последовательность

$$C_10, C_20, \dots, C_m0, C_m1, C_{m-1}1, \dots, C_11$$

содержит все бинарные последовательности длины $k + 1$, причем каждые две соседние последовательности будут отличаться в точности в одной координате. Этим непосредственно определяется некоторый алгоритм рекуррентного построения последовательности всех подмножеств. Получаемая этим способом последовательность C_1, \dots, C_{2^n} называется *бинарным кодом Грэя порядка n* . Опишем теперь нерекуррентную версию этого алгоритма.

Алгоритм 1.13. (Генерирование всех подмножеств n -элементного множества.)

Данные: n .

Результат: Последовательность всех подмножеств n -элементного множества, в которой каждое последующее подмножество получается из предыдущего добавлением или удалением единственного элемента. Каждое подмножество пред-

ставляется бинарной последовательностью $B[1], \dots, B[n]$.

```

1. begin
2.   for  $i := 1$  to  $n$  do  $B[i] := 0$ ; (*пустое множество*)
3.    $i := 0$ ; (*  $i$ =числу сгенерированных до этого момента подмножеств*)
4.   repeat
5.     write( $B[1], \dots, B[n]$ );
6.      $i := i + 1$ ;  $p := 1$ ;  $j := i$ ;
7.     while  $j \bmod 2 = 0$  do
8.       begin (* $j2^{p-1} = i$ *)
9.          $j = j/2$ ;  $p := p + 1$ 
10.      end; (* $p = Q(i) + 1$ *)
11.     if  $p \leq n$  then  $B[p] := 1 - B[p]$  (*смена позиции  $p$ *)
12.   until  $p > n$ 
13. end

```

0	0	0	0
1	0	0	0
1	1	0	0
0	1	0	0
0	1	1	0
1	1	1	0
1	0	1	0
0	0	1	0
0	0	1	1
1	0	1	1
1	1	1	1
0	1	1	1
0	1	0	1
1	1	0	1
1	0	0	1
0	0	0	1

Рис. 1.6: Последовательность множеств, порожденных алгоритмом 1.13 для $n = 4$

Докажем теперь, что описанный алгоритм действительно генерирует все бинарные последовательности длины n . Пусть $Q(i)$ обозначает наибольшую степень двойки, которая делит i . Если мы представим i в двоичном виде как $i = [b_n b_{n-1} \dots b_1]$, то, очевидно, имеем $Q(i) + 1 = \min\{j : b_j = 1\}$. Покажем вначале, что после выхода из внутреннего цикла 7 будем иметь $p = Q(i) + 1$. С этой целью заметим, что до входа в цикл $j2^{p-1} = i$ (где $p = 1, j = i$) и каждая итерация цикла не нарушает этого равенства (так как $(j/2)2^{(p+1)-1} = j2^{p-1}$). После выхода из цикла j нечетно, следовательно, $Q(i) = p - 1$, т.е. действительно $p = Q(i) + 1$. Предположим теперь, что $k < n$ и что в первых 2^k итерациях цикла 3 были получены все 2^k бинарные последовательности $b_1 b_2 \dots b_n$, такие что $b_{k+1} = \dots = b_n = 0$ (это очевидно справедливо для $k = 0$). В последней из этих 2^k итераций переменная i принимает значение 2^k (строка 7). Переменная p получает значение $Q(2^k) + 1 = k + 1$, и, следовательно, наступает изменение значения переменной $B[k + 1]$ с 0 на 1. Рассмотрим теперь последовательность

$$Q(2^k + 1) + 1, Q(2^k + 2) + 1, \dots, Q(2^{k+1}) + 1 \quad (1.9)$$

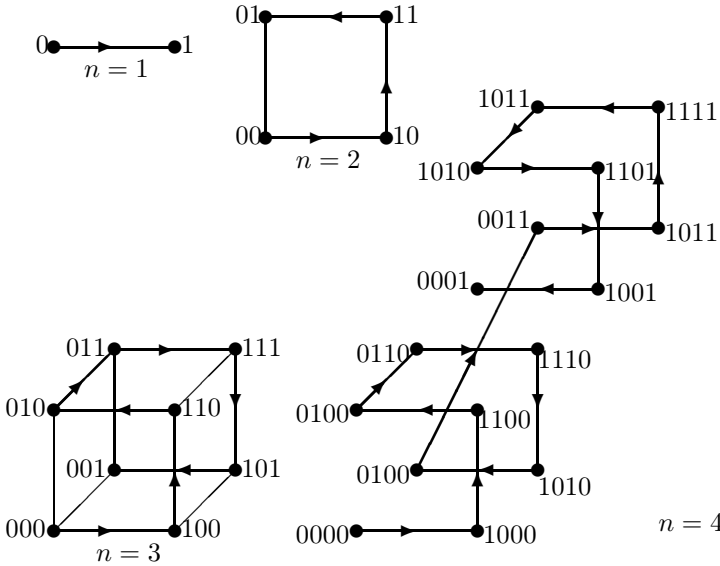


Рис. 1.7: Гамильтоновы пути в n -мерных кубах

значений переменной p , сгенерированных в последующих 2^k итерациях цикла 3. Отметим, что $Q(2^k + m) = Q(2^k - m)$ для $0 \leq m \leq 2^k - 1$ (этот факт становится особенно очевидным, если рассматривать сложение чисел $2^k + m$ и $2^k - m$, записанных в двоичной форме). Последовательность (1.9) является зеркальным отображением последовательности значений переменной p , полученных в первых 2^k итерациях цикла 3. Отсюда следует, что и последовательности $B[1], B[2], \dots, B[k]$, полученные в первых 2^k итерациях, появляются в обратном порядке в последующих 2^k итерациях. Это дает все бинарные последовательности $b_1 b_2 \dots b_n$, такие, у которых $b_{k+1} = \dots = b_n = 0$, полученные в первых 2^{k+1} итерациях. Отсюда становится ясно, что наш алгоритм генерирует все бинарные последовательности длины n в таком же порядке, что и предыдущий рекурсивный алгоритм. Последовательность подмножеств, полученных при помощи алгоритма для $n = 4$, представлена на рис. 1.6.

Можно показать, что среднее число шагов, необходимых для генерирования каждого следующего подмножества (не считая процедуры написания этого подмножества), ограничено константой, не зависящей от n (см. задачу 1.21). Последовательность подмножеств, полученных с помощью алгоритма 1.13, можно так же, как мы это делали для последовательности перестановок, полученных с помощью алгоритма 1.12, проиллюстрировать на графе, вершины которого соответствуют бинарным последовательностям длины n и две вершины которо-

го соединены ребром, если соответствующие последовательности отличаются в точности в одной позиции. Такой граф будем называть (*двоичным*) n -мерным кубом. Очевидно, что последовательность, полученная с помощью нашего алгоритма, соответствует гамильтонову пути в этом графе. На рис. 1.7 это продемонстрировано для $n = 1, 2, 3, 4$.

В некоторых приложениях более естественным, нежели понятие множества, является *множество с повторениями*. Каждый элемент множества с повторениями может появляться в этом множестве несколько раз, число этих вхождений является существенным и носит название кратности элемента в множестве. Множество с повторениями, содержащее, например, элемент a кратности 2, элемент b кратности 3 и элемент c кратности 1, будем обозначать (a, a, b, b, b, c) или $(2 * a, 3 * b, 1 * c)$. Порядок элементов не существует:

$$(a, a, b, b, b, c) = (b, a, b, a, c, b) = \dots$$

Существенна только кратность каждого элемента — имеем

$$(a, a, b, b, b, c) \neq (a, b, c)$$

в отличие от равенства множеств

$$\{a, a, b, b, b, c\} = \{a, b, c\}.$$

Если в некотором множестве с повторениями кратность каждого элемента равна единице, то, очевидно, мы можем отождествлять его с обычным множеством. Пусть A, B — два множества с повторениями. Будем говорить, что A есть *подмножество* B (обозначение $A \subseteq B$), если кратность каждого элемента в A не больше кратности этого элемента в B . Пусть X — множество с повторениями, содержащее r разных элементов x_1, \dots, x_r с кратностями k_1, \dots, k_r соответственно. Число $|X| = k_1 + \dots + k_r$ будем называть *мощностью* X . Каждому подмножеству $A \subseteq X$ однозначно соответствует последовательность

$$\langle m_1, \dots, m_r \rangle, 0 \leq m_1 \leq k_1, \dots, 0 \leq m_r \leq k_r, \quad (1.10)$$

где m_i обозначает кратность элемента X_i в A . Отсюда сразу же следует, что число всех подмножеств $A \subseteq X$ равно

$$(k_1 + 1)(k_2 + 1) \dots (k_r + 1). \quad (1.11)$$

Эти подмножества, а точнее соответствующие им последовательности (1.10), можно генерировать способом, подобным тому, который был использован в

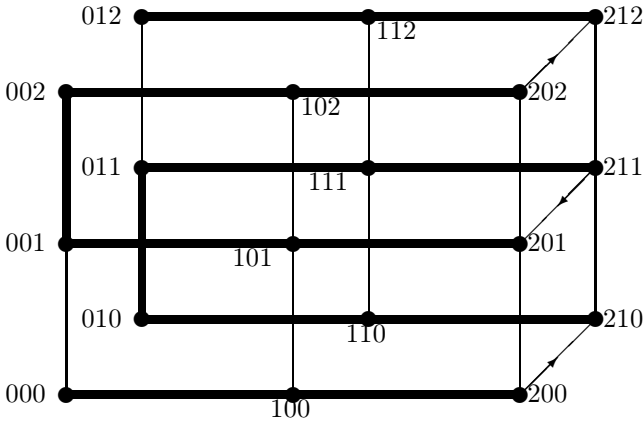


Рис. 1.8: Гамильтонов путь в графе, соответствующем подмножествам множества с повторениями $(x_1, x_1, x_2, x_3, x_3)$

алгоритме 1.13. Для этого достаточно заметить, что если последовательность C_1, C_2, \dots, C_m содержит все $p = (k_1 + 1)(k_2 + 1) \dots (k_s + 1)$ последовательностей m_1, \dots, m_s , $0 \leq m_1 \leq k_1, \dots, 0 \leq m_s \leq k_s$, то последовательность

$$C_1 0, C_2 0, \dots, C_p 0, C_p 1, C_{p-1} 1, \dots, C_1 1, C_1 2, C_2 2, C_p 2, C_p 3, \dots \quad (1.12)$$

длины $p(k_{s+1} + 1)$ содержит все последовательности $\langle m_1, \dots, m_{s+1} \rangle$, $0 \leq m_1 \leq k_1, \dots, 0 \leq m_{s+1} \leq k_{s+1}$. Очевидно, что последовательность подмножеств, полученная с помощью такого построения, будет обладать таким свойством, что каждое последующее подмножество образуется из предыдущего добавлением или удалением одного элемента. Предоставляя читателю исключить рекурсию из этого алгоритма, проиллюстрируем его с помощью гамильтонова пути в некотором графе (рис. 1.8). Этот рисунок интерпретируется так же, как и рис. 1.7.

1.6 k -элементные подмножества, биномиальные коэффициенты

Число всех k -элементных подмножеств n -элементного множества будем обозначать $\binom{n}{k}$. Символ $\binom{n}{k}$ называется *биномиальным коэффициентом*, исходя из сле-

дующей формулы для n -й степени бинома $x + y$:

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k} \quad (1.13)$$

Чтобы убедиться в истинности этой формулы, достаточно заметить, что коэффициент при $x^k y^{n-k}$ равен числу способов, которыми из n сомножителей $(x + y) \dots (x + y)$ можно выбрать k сомножителей.

Очевидно, $\binom{n}{k} = 0$ для $k > n$. Напомним, что k -элементные подмножества n -элементного множества назывались в старой литературе по комбинаторике k -членными комбинациями для n -членного множества без повторов.

С биномиальными коэффициентами связано много интересных тождеств. Вот некоторые из них.

$$\sum_{i=0}^n \binom{n}{i} = 2^n \quad (1.14)$$

Эта формула следует из того, что сумма слева выражает число всех подмножеств n -элементного множества.

$$\sum_{i=0}^n \binom{n}{i} i = n2^{n-1} \quad (1.15)$$

Для доказательства этой формулы составим список, содержащий по порядку все подмножества n -элементного множества X . Этот список содержит $\binom{n}{i}$ i -элементных подмножеств для $i = 1, \dots, n$, таким образом, его длина (так называемое число появлений элементов множества X) выражается суммой в левой части формулы (1.15). С другой стороны, каждый элемент $x \in X$ появляется в списке 2^{n-1} раз, так как таково число подмножеств, содержащих x (их столько, сколько имеется всех подмножеств множества $X \setminus \{x\}$). Поэтому длина нашего списка равна $n2^{n-1}$. Доказательство тем самым закончено.

$$\binom{n}{k} = \binom{n}{n-k} \quad (1.16)$$

Это прямое следствие того факта, что каждому k -элементному подмножеству $X \subseteq Y$ однозначно соответствует $n - k$ -элементное подмножество $X \setminus Y$ множества X .

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad (1.17)$$

Эта формула получается из следующих рассуждений. Зафиксируем некоторый элемент x n -элементного множества X . Множество всех k -элементных

Мы хотим доказать ещё одно тождество:

$$\binom{n}{k} \binom{k}{m} = \binom{n}{m} \binom{n-m}{n-k} \quad (1.20)$$

Для доказательства подсчитаем число таких пар $\langle K, M \rangle$ подмножества n -элементного множества X , что $K \supseteq M$, $|K| = k$, $|M| = m$. С одной стороны, подмножество K можно выбрать $\binom{n}{k}$ способами, а подмножество $M \subseteq K$ можно выбрать $\binom{k}{m}$ способами, что даёт $\binom{n}{k} \binom{k}{m}$ возможных пар (левая часть (1.20)). С другой стороны, для каждого из $\binom{n}{m}$ подмножеств $M \subseteq X$ можно выбрать $\binom{n-m}{n-k}$ способами надмножество $K \supseteq M$, так как такое надмножество K однозначно определяется через $(k-m)$ -элементное подмножество $X \setminus K$ $(n-m)$ -элементного множества $X \setminus M$. Искомое число пар $\langle K, M \rangle$ равно, таким образом, $\binom{n}{m} \binom{n-m}{n-k}$ (правая часть (1.20)).

Дадим теперь формулу, позволяющую вычислять $\binom{n}{k}$ непосредственно.

Теорема 1.14

$$\binom{n}{k} = \frac{[n]_k}{k!} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\dots(n-k+1)}{1 \cdot 2 \cdot \dots \cdot k} \quad (1.21)$$

Доказательство. Каждая из $[n]_k$ инъективных последовательностей длины k определяет k -элементное подмножество, состоящее из элементов, появляющихся в этой последовательности, причём одно и то же подмножество S получается в точности из $k!$ последовательностей, соответствующим всем перестановкам множества S . Отсюда $\binom{n}{k} = \frac{[n]_k}{k!}$. Теперь достаточно воспользоваться теоремой 1.2.

Отметим, что эта теорема позволяет доказать все тождества, рассмотренные в этом разделе. Например, для (1.17) имеем

$$\begin{aligned} \binom{n}{k} &= \frac{n!}{k!(n-k)!} = \frac{n!}{k!(n-k)!} \cdot \left(\frac{n-k}{n} + \frac{k}{n} \right) = \\ &= \frac{(n-1)!}{k!(n-1-k)!} + \frac{(n-1)!}{(k-1)!(n-k)!} = \binom{n-1}{k} + \binom{n-1}{k-1} \end{aligned}$$

Однако такое «алгебраическое» доказательство, основанное на механическом преобразовании выражений, гораздо менее понятно, чем «комбинаторное» доказательство, данное выше.

С другой стороны, формула (1.21) позволяет легко доказать следующее свойство биномиальных коэффициентов:

$$\begin{aligned} \binom{n}{0} &< \binom{n}{1} < \dots < \binom{n}{\lfloor n/2 \rfloor} = \\ &= \binom{n}{\lceil n/2 \rceil} > \binom{n}{\lceil n/2 \rceil + 1} > \dots > \binom{n}{n} \end{aligned} \quad (1.22)$$

($n > 1$). Достаточно заметить, что

$$\binom{n}{i+1} = \frac{n(n-1)\dots(n-i)}{1\cdot 2\cdot \dots\cdot i\cdot (i+1)} = \frac{n-i}{i+1} \cdot \frac{n(n-1)\dots(n-i+1)}{1\cdot 2\cdot \dots\cdot i} = \frac{n-i}{i+1} \cdot \binom{n}{i}$$

Имеем $(n-i)/(i+1) > 1$ для $i < (n-1)/2$, т.е. для $i < \lfloor n/2 \rfloor$, аналогично $(n-i)/(i+1) < 1$ для $i > (n-1)/2$, т.е. для $i \geq \lceil n/2 \rceil$ и $(n - \lfloor n/2 \rfloor) / (\lceil n/2 \rceil + 1) = \lceil n/2 \rceil / \lfloor n/2 \rfloor = 1$, если n нечётно (для чётного n имеем $\lfloor n/2 \rfloor = \lceil n/2 \rceil$ и равенство в (1.22), очевидно, выполняется).

Займёмся теперь отысканием числа k -элементных подмножеств множества с повторениями $(k * x_1, \dots, k * x_n)$. Другими словами, зададимся вопросом, сколькими способами можно породить k -элементное множество с повторениями, имея в распоряжении n разных элементов, скажем $1, \dots, n$, из которых каждый может использоваться произвольное количество раз. Следует отметить, что это число равно числу неубывающих последовательностей длины k с элементами из множества $\{1, \dots, n\}$. Это вытекает из того факта, что элементы каждого подмножества можно однозначно расставить в неубывающую последовательность. Очевидно, что подмножествам без повторений соответствуют возрастающие последовательности.

Теорема 1.15. Число k -элементных подмножеств множества $(k * x_1, \dots, k * x_n)$ равно

$$\binom{n+k-1}{k} \quad (1.23)$$

Доказательство. Мы должны определить число убывающих последовательностей $\langle a_1, \dots, a_n \rangle$ с элементами из множества $\{1, \dots, n\}$. Каждую такую последовательность можно разделить на n частей, где i -я часть содержит некоторое, быть может нулевое, число следующих друг за другом элементов i . Если мы отделим эти части одну от другой $n-1$ нулями, использованными в качестве разделителей, то получим последовательность длины $k-n+1$. Например, для $n=5$, $k=4$ последовательности $\langle 2, 2, 4, 5 \rangle$ соответствует последовательность $\langle 0, 2, 2, 0, 0, 4, 0, 5 \rangle$. Последняя последовательность однозначно определяется размещением появляющихся в ней нулей. Действительно, мы можем восстановить её, заполняя часть до первого нуля экземплярами элемента 1, часть от первого до второго нуля — экземплярами элемента 2 и т.д. А число размещений $n-1$ нулей на $k+n-1$ позициях равно

$$\binom{k+n-1}{n-1} = \binom{k+n-1}{(k+n-1)-(n-1)} = \binom{k+n-1}{k}$$

Полученное таким образом число подмножеств с повторениями можно записать несколько иначе:

$$\binom{k+n-1}{n-1} = \frac{n(n+1)\dots(n+k-1)}{k!} = \frac{[n]^k}{k!} \quad (1.24)$$

Дадим комбинаторную интерпретацию этого тождества. Напомним с этой целью, что $[n]^k$ есть число упорядоченных размещений k элементов x_1, \dots, x_k в n ячейках. Отметим, что каждому такому размещению, содержащему r_i элементов в i -й ячейке, соответствует k -элементное подмножество $(r_1 * y_1, \dots, r_n * y_n)$ множества с повторениями $(k * y_1, \dots, k * y_n)$. В этом упорядочении существенно только число элементов в каждой из ячеек. Таким образом, размещениям

$$\begin{array}{c|c|c} \text{ячейка 1} & \text{ячейка 2} & \text{ячейка 3} \\ \hline x_3 & x_2 & x_5 \quad x_1 \quad x_4 \\ x_4 & x_1 & x_2 \quad x_3 \quad x_5 \end{array}$$

соответствует одно и то же подмножество $2 * y_1, 3 * y_3$. Ясно, что каждые два размещения, соответствующие одному и тому же подмножеству, отличаются перестановкой объектов x_1, \dots, x_k . В результате число всех k -элементных подмножеств множества с повторениями $(k * y_1, \dots, k * y_n)$ равно $[n]^k/k!$. Очевидно, что это рассуждение вместе с формулой (1.24) представляет собой другое доказательство теоремы 1.15.

1.7 Генерирование k -элементных подмножеств

Опишем сейчас два алгоритма генерирования всех k -элементных подмножеств n -элементного множества X . Без ограничения общности можно принять $X = \{1, \dots, n\}$. Тогда каждому k -элементному подмножеству взаимно однозначно соответствует возрастающая последовательность длины k с элементами из X : например, подмножеству $\{3, 5, 1\}$ соответствует последовательность $\langle 1, 3, 5 \rangle$. Можно легко указать алгоритм, с помощью которого генерируются все такие последовательности в лексикографическом порядке. Достаточно с этой целью заметить, что при таком порядке последовательностью, непосредственно следующей за последовательностью $\langle a_1, \dots, a_k \rangle$, является

$$\langle b_1, \dots, b_k \rangle = \langle a_1, \dots, a_{p-1}, a_p + 1, a_p + 2, \dots, a_p + k - p + 1 \rangle,$$

где

$$p = \max\{i : a_i < n - k + 1\}.$$

Более того, последовательностью, непосредственно следующей за $\langle b_1, \dots, b_k \rangle$, является

$$\langle c_1, \dots, c_k \rangle = \langle b_1, \dots, b_{p'-1}, b_{p'} + 1, b_{p'} + 2, \dots, b_{p'} + k - p' + 1 \rangle,$$

где

$$p' = \begin{cases} p - 1, & \text{если } b_k = n, \\ k, & \text{если } b_k < n \end{cases}$$

(будем предполагать, что последовательности $\langle a_1, \dots, a_k \rangle$ и $\langle b_1, \dots, b_k \rangle$ отличаются от $\langle n - k + 1, \dots, n \rangle$ — последней последовательности в нашем порядке). Это приводит к следующему простому алгоритму.

Алгоритм 1.16. (Генерирование всех k -элементных подмножеств множества $\{1, \dots, n\}$ в лексикографическом порядке.)

Данные: n, k .

Результат: последовательность всех k -элементных подмножеств множества $\{1, \dots, n\}$ в лексикографическом порядке.

1. **begin**

2. **for** $i := 1$ **to** k **do** $A[i] := i$; (*первое подмножество*)

3. $p := k$;

4. **while** $p \geq 1$ **do**

5. **begin** *write* $(A[1], \dots, A[k])$;

6. **if** $A[k] = n$ **then** $p := p - 1$ **else** $p := k$;

7. **if** $p \geq 1$ **then**

8. **for** $i := k$ **downto** p **do** $A[i] := A[p] + i - p + 1$

9. **end**

10. **end**

Последовательность всех 4-элементных подмножеств множества $\{1, \dots, 6\}$, полученная с помощью этого алгоритма, представлена на рис. 1.9.

Другой алгоритм, который мы опишем ниже, генерирует все k -элементные подмножества таким образом, что каждое последующее подмножество образуется из предыдущего удалением одного элемента и добавлением другого. Этот алгоритм представим в рекурсивной форме. Обозначим с этой целью через $G(n, k)$ список, содержащий все k -элементные подмножества множества $\{1, \dots, n\}$, в котором первым подмножеством является $\{1, \dots, k\}$, последним — $\{1, 2, \dots, k - 1, n\}$ и каждое следующее подмножество образуется из предыдущего удалением некоторого элемента и добавлением другого. Отметим, что если $G(n - 1, k)$ и $G(n - 1, k - 1)$ уже построены, то $G(n, k)$ можно определить следующим образом:

$$G(n, k) = G(n - 1, k), G^*(n - 1, k - 1) \cup \{n\}, \quad (1.25)$$

1	2	3	4
1	2	3	5
1	2	3	6
1	2	4	5
1	2	4	6
1	2	5	6
1	2	4	5
1	3	4	6
1	3	5	6
1	4	5	6
2	3	4	5
2	5	4	6
2	3	5	6
2	4	5	6
3	4	5	6

Рис. 1.9: Последовательность 4-элементных подмножеств множества $\{1, \dots, 6\}$, построенная при помощи алгоритма 1.16.

где $G^*(n-1, k-1) \cup \{n\}$ обозначает список, образованный из $G(n-1, k-1)$ изменением порядка элементов списка на обратный и последующим добавлением элемента n к каждому множеству. Действительно, $G(n-1, k)$ содержит все k -элементные подмножества множества $\{1, \dots, n\}$, не содержащие n , $G^*(n-1, k-1) \cup \{n\}$ — все k -элементные подмножества, содержащие n , причем последним подмножеством в списке $G(n-1, k)$ является $\{1, 2, \dots, k-1, n-1\}$, а первым подмножеством в списке $G^*(n-1, k-1) \cup \{n\}$ является $\{1, 2, \dots, k-1, n\}$. На рис. 1.10 показан процесс построения списка $G(4, 2)$.

Списку $G(n, k)$ мы можем так же, как и в случае генерирования всех подмножеств, поставить в соответствие некоторый гамильтонов путь в графе, вершины которого соответствуют двухэлементным подмножествам множества $\{1, 2, 3, 4\}$, причем вершины, соответствующие подмножествам A и B , соединены ребром тогда и только тогда, когда $|A \cap B| = k-1 = 1$. Это проиллюстрировано на рис. 1.11.

Предоставляем читателю самому избавиться от рекурсии в описанном алгоритме (см. задачу 1.29)

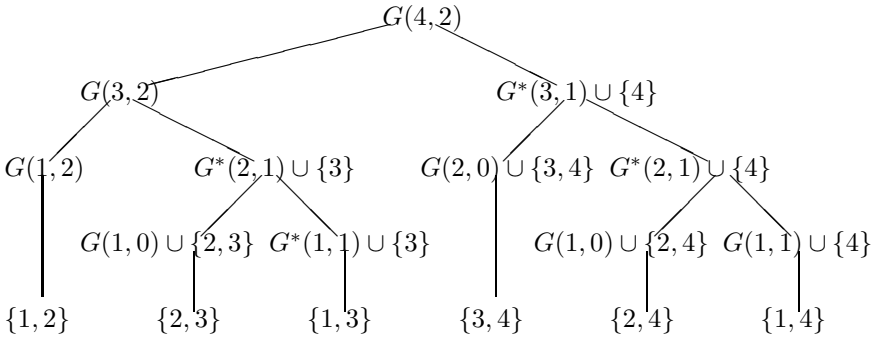


Рис. 1.10: Построение списка $G(4,2)$.

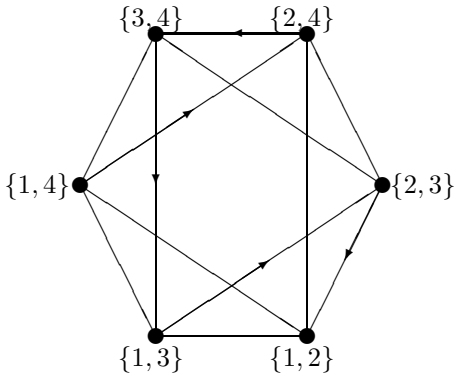


Рис. 1.11: Гамильтонов путь в графе, соответствующем всем двухэлементным подмножествам множества $\{1, 2, 3, 4\}$.

1.8 Разбиения множеств

Под *разбиением* n -элементного множества X на k блоков будем понимать произвольное семейство $\pi = \{B_1, \dots, B_k\}$, такое что $B_1 \cup B_2 \cup \dots \cup B_k = X$, $B_i \cap B_j = \emptyset$ для $1 \leq i < j \leq k$ и $B_i \neq \emptyset$ для $1 \leq i \leq k$. Подмножества B_1, \dots, B_k будем называть *блоками* семейства π . Множество всех разбиений множества X на k блоков будем обозначать $\Pi_k(X)$, а множество всех разбиений через $\Pi(X)$. Очевидно, что $\Pi(X) = \Pi_1(X) \cup \dots \cup \Pi_n(X)$ (более того, $\{\Pi_1(X), \dots, \Pi_n(X)\}$ является разбиением множества $\Pi(X)$).

С разбиением связано понятие отношения эквивалентности. Каждому разбиению π мы можем поставить в соответствие отношение эквивалентности

$$E(\pi) = \bigcup_{B \in \pi} (B \times B) \quad (1.26)$$

(элементы $x, y \in X$ находятся в отношении $E(\pi)$ тогда и только тогда, когда они принадлежат одному и тому же блоку разбиения π). И наоборот, каждому отношению эквивалентности E на множестве X мы можем поставить в соответствие разбиение

$$X/E = \{x/E : x \in X\}, \quad (1.27)$$

где x/E обозначает *класс эквивалентности* элемента x , т.е. множество всех элементов, находящихся в отношении E к элементу x :

$$x/E = \{y \in X : xEy\}. \quad (1.28)$$

Нетрудно заметить, что формулы (1.26) и (1.27) определяют взаимно однозначное соответствие между разбиениями и отношениями эквивалентности на множестве X . Если $\pi, \sigma \in \Pi(X)$ и каждый блок $B \in \sigma$ является суммой некоторого числа блоков разбиения π , то будем говорить, что π есть *измельчение* разбиения σ , и будем писать $\pi \leq \sigma$. Например:

$$\{\{1\}, \{2, 5\}, \{4, 6\}, \{3\}\} \leq \{\{1, 2, 3, 5\}, \{4, 6\}\}.$$

Легко проверить, что $\pi \leq \sigma$ тогда и только тогда, когда для отношений эквивалентности, соответствующих данным разбиениям, выполняется соотношение $E(\pi) \subseteq E(\sigma)$. Очевидно, что определенное таким образом отношение \leq является частичным упорядочением на множестве $\Pi(X)$. Некоторые свойства множества $\Pi(X)$, упорядоченного с помощью отношения измельчения, напоминают аналогичные свойства множества $\Pi(X)$, упорядоченного на основе включения. Остановимся сейчас на одном из таких свойств. Частично упорядоченное множество $\langle A, \leq \rangle$ будем называть *решеткой*, если для произвольных элементов $x, y \in A$ существуют такие элементы $a, b \in A$, что

- (а) $a \leq x$, $a \leq y$ и для произвольного элемента c , такого что $c \leq x$, $c \leq y$, имеем $c \leq a$,
- (б) $b \geq x$, $b \geq y$ и для произвольного элемента c , такого что $c \geq x$, $c \geq y$, имеем $c \geq b$.

Такие элементы, если они существуют, однозначно определяются через x и y . Действительно, для произвольных элементов a_1, a_2 , отвечающих условию (а), имеем $a_2 \leq a_1$ и $a_1 \leq a_2$, т.е. $a_1 = a_2$; аналогично доказывается (б). Элемент a будем называть *нижней границей* элементов x, y и обозначать $x \wedge y$, а элемент b будем называть *верхней границей* и обозначать $x \vee y$.

Примером решетки является $\langle \wp(X), \subseteq \rangle$. Очевидно, что в этом случае имеем $A \wedge B = A \cap B$ и $A \vee B = A \cup B$.

Теорема 1.17. *Множество $\Pi(X)$, упорядоченное на основе отношения измелъчения \leq , образует решетку, причем*

$$\pi \wedge \sigma = \{A \cap B : (A \in \pi) \wedge (B \in \sigma) \wedge (A \cap B \neq \emptyset)\}, \quad (1.29)$$

т. е.

$$E(\pi \wedge \sigma) = E(\pi) \cap E(\sigma), \quad (1.30)$$

а разбиение $\pi \vee \sigma$ определяется следующим образом:

$$\langle x, y \rangle \in E(\pi \vee \sigma) \Leftrightarrow x_1, \dots, x_k, \text{ что } x = x_1, y = x_k \text{ и } \langle x_i, x_{i+1} \rangle \in E(\pi) \text{ или } \langle x_i, x_{i+1} \rangle \in E(\sigma) \text{ для } i = 1, 2, \dots, k-1. \quad (1.31)$$

Доказательство. Отметим прежде всего, что простым следствием определения отношения эквивалентности является тот факт, что пересечение двух отношений эквивалентности есть отношение эквивалентности. Таким образом, $E(\pi) \cap E(\sigma)$ есть отношение эквивалентности, которое в свою очередь однозначно определяет некоторое разбиение α , такое что $E(\pi) \cap E(\sigma) = E(\alpha)$ (ср. (1.27)). Очевидно, что $E(\alpha) \subseteq E(\pi)$, $E(\alpha) \subseteq E(\sigma)$, а, следовательно, $\alpha \leq \pi$, $\alpha \leq \sigma$. Более того, для каждого разбиения α' , такого что $\alpha' \leq \pi$, $\alpha' \leq \sigma$, имеем $E(\alpha') \subseteq E(\pi)$, $E(\alpha') \subseteq E(\sigma)$ и в результате $E(\alpha') \subseteq E(\pi) \cap E(\sigma) = E(\alpha)$ т.е. $\alpha' \leq \alpha$. Следовательно, $\alpha = \pi \wedge \sigma$, что и доказывает формулу (1.30). Отметим, что $x, y \in E(\pi \wedge \sigma) = E(\pi) \cap E(\sigma)$ тогда и только тогда, когда x и y принадлежат к одному и тому же блоку разбиения π и к одному блоку разбиения σ , т.е. $x, y \in A \cap B$, где $A \in \pi$ и $B \in \sigma$. Последнее и доказывает формулу (1.29).

Предположим теперь, что R — бинарное отношение, определенное правой частью формулы (1.31). Нетрудно проверить, что R — отношение эквивалентности. Действительно, xRx (достаточно принять $k = 1$), xRy влечет за собой yRx (достаточно рассмотреть последовательность x_k, x_{k-1}, \dots, x_1) и xRz

влекут за собой xRz (достаточно рассмотреть конкатенацию соответствующих последовательностей). Отношение R однозначно определяет разбиение β , такое что $E(\beta) = R$. Очевидно, что $E(\pi) \subseteq R = E(\beta)$ и $E(\sigma) \subseteq R = E(\beta)$, т.е. $\pi \leq \beta$ и $\sigma \leq \beta$. Предположим теперь, что β' — произвольное разбиение, такое что $\pi \leq \beta'$, $\sigma \leq \beta'$, и пусть $\langle x, y \rangle \in E(\beta)$. Тогда существует последовательность $x = x_1, x_2, \dots, x_k = y$, удовлетворяющая правой части формулы (1.31), и, следовательно, $\langle x_i, x_{k+1} \rangle \in E(\pi) \cup E(\sigma) \subseteq E(\beta')$, $1 \leq i \leq k$. Вследствие транзитивности отношения $E(\beta')$ имеем $\langle x, y \rangle \in E(\beta')$. Таким образом, мы доказали, что $\beta \leq \beta'$, а следовательно, и $\beta = \pi \vee \sigma$.

1.9 Числа Стирлинга второго и первого рода

Число Стирлинга второго рода $S(n, k)$ определяется как число разбиений n -элементного множества на k блоков:

$$S(n, k) = |\Pi_k(X)|, \text{ где } |X| = n. \quad (1.32)$$

Например, $S(4, 2) = 7$, так как существуют в точности 7 разбиений множества $\{1, \dots, 4\}$ на два блока:

$$\begin{aligned} & \{\{1, 2, 3\}, \{4\}\} \\ & \{\{1, 2, 4\}, \{3\}\} \\ & \{\{1, 3, 4\}, \{2\}\} \\ & \{\{1, 2\}, \{3, 4\}\} \\ & \{\{1, 3\}, \{2, 4\}\} \\ & \{\{1, 4\}, \{2, 3\}\} \\ & \{\{1\}, \{2, 3, 4\}\} \end{aligned}$$

Очевидно, что $S(n, k) = 0$ для $k > n$. Примем также $S(0, 0) = 1$, так как пустое семейство блоков является в соответствии с определением разбиением пустого множества. С числами Стирлинга второго порядка связано, как и с биномиальными коэффициентами, много любопытных тождеств. Докажем сначала тождество, напоминающее тождество (1.17), связанное с треугольником Паскаля:

$$S(n, k) = S(n-1, k-1) + kS(n-1, k) \quad \text{для } 0 < k < n, \quad (1.33)$$

$$S(n, n) = 1 \quad \text{для } n \geq 0, \quad (1.34)$$

$$S(n, 0) = 0 \quad \text{для } n > 0. \quad (1.35)$$

Формулы (1.34) и (1.35) очевидны. Для доказательства формулы (1.33) рассмотрим множество всех разбиений множества $\{1, \dots, n\}$ на k блоков. Это множество распадается на два различных класса: тех разбиений, которые содержат

$n \setminus k$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0	0
7	0	1	63	301	350	140	21	1	0	0	0
8	0	1	127	966	1,701	1,050	266	28	1	0	0
9	0	1	255	3,025	7,770	6,951	2,646	462	36	1	0
10	0	1	511	9,330	34,105	42,525	22,827	5,880	750	45	1

Таблица 1.2: Числа Стирлинга второго рода

одноэлементный блок $\{n\}$, и тех разбиений, для которых n является элементом большего (по крайней мере двухэлементного) блока. Мощность первого класса равна $S(n-1, k-1)$, т.е. такова, каково число разбиений множества $\{1, \dots, n-1\}$ на $k-1$ блоков. Мощность другого класса составляет $kS(n-1, k)$, так как каждому разбиению множества $\{1, \dots, n-1\}$ на k блоков соответствует в этом классе в точности k разбиений, образованных добавлением элемента n поочередно к каждому блоку. Формулы (1.33), (1.34) и (1.35) позволяют легко вычислять значения $S(n, k)$ даже для больших значений n и k . В табл. 1.2 представлены числа $S(n, k)$ для $0 \leq n, k \leq 10$. Отметим, что эту таблицу можно трактовать как «треугольник Стирлинга», в котором каждое значение, кроме крайних, равных единице, можно получить как сумму чисел, находящихся над ним, а именно числа, расположенного в точности над ним и умноженного на k , и числа над ним с левой стороны. Вот пример другой рекуррентной зависимости, связанной с числами Стирлинга второго рода:

$$S(n, k) = \sum_{i=k-1}^{n-1} \binom{n-1}{i} S(i, k-1) \text{ для } k \geq 2 \quad (1.36)$$

Для доказательства тождества рассмотрим множество всех разбиений $S(n, k)$ множества $X = \{1, \dots, n\}$. Это множество распадается на различные классы, соответствующие разным подмножествам множества X , которые являются блоками, содержащими элемент n . Отметим, что для каждого b -элементного подмножества $B \subseteq X$, содержащего элемент n , существует в точности $S(n-b, k-1)$ разбиений множества X на k блоков, содержащих B в качестве блока. Действительно, каждое такое разбиение однозначно соответствует разбиению множества $X \setminus B$ на $k-1$ блоков. b -элементное множество $B \subseteq X$, содержащее элемент n ,

можно выбрать $\binom{n-1}{b-1}$ способами; таким образом,

$$\begin{aligned} S(n, k) &= \sum_{b=1}^{n-(k-1)} \binom{n-1}{b-1} S(n-b, k-1) = \\ &= \sum_{b=1}^{n-(k-1)} \binom{n-1}{n-b} S(n-b, k-1) = \\ &= \sum_{i=k-1}^{n-1} \binom{n-1}{i} S(i, k-1). \end{aligned}$$

Число Белла B_n определяется как число всех разбиений n -элементного множества

$$B_n = |\Pi(X)|, \text{ где } |X| = n \quad (1.37)$$

Другими словами,

$$B_n = \sum_{k=0}^n S(n, k) \quad (1.38)$$

Докажем теперь следующую простую рекуррентную зависимость, связанную с числами Белла:

$$B_{n+1} = \sum_{i=0}^n \binom{n}{i} B_i \quad (1.39)$$

(принимаем $B_0 = 1$). Доказательство проводится аналогично доказательству тождества (1.36). Множество всех разбиений множества $X = \{1, \dots, n+1\}$ можно разбить на различные классы в зависимости от блока B , содержащего элемент $n+1$, или — что равнозначно — в зависимости от множества $X \setminus B$. Для каждого множества $X \setminus B \subseteq \{1, \dots, n\}$ существует в точности $|\Pi(X \setminus B)| = B_{|X \setminus B|}$ разбиений множества X , содержащих B в качестве блока. Группируя наши классы в зависимости от мощности множества $X \setminus B$, получаем формулу (1.39).

Числа B_n для $0 \leq n \leq 20$ представлены в табл. 1.3.

Существует строгая зависимость между числами $S(n, k)$ и числом всех функций из n -элементного множества на k -элементное множество, т.е. функций $f : X \rightarrow Y$, $f(X) = Y$ для $|X| = n$, $|Y| = k$. Каждой такой функции f можно поставить в соответствие следующее разбиение множества X на k блоков:

$$N(f) = \{f^{-1}(y) : y \in Y\}, \quad (1.40)$$

называемое *ядром* функции f (условие $f(X) = Y$ дает гарантию того, что подмножества $f^{-1}(y)$ непустые). С другой стороны, нетрудно заметить, что каждому разбиению $\pi \in \Pi_k(X)$ соответствует в точности $k!$ функций из X на Y , таких что $N(f) = \pi$. Каждая такая функция взаимно однозначно соответствует соотношению блоков разбиения π элементам множества Y . Обозначая через $s_{n,k}$ число функций из X на Y , получаем, следовательно,

$$s_{n,k} = k! S(n, k) \quad (1.41)$$

Пользуясь этой формулой, мы можем доказать еще одно свойство чисел Стирлинга второго рода, которое касается связи между многочленами x^k и многочленами

$$[x]_k = x(x - 1) \dots (x - k + 1). \tag{1.42}$$

n	B_n
0	1
1	1
2	2
3	5
4	15
5	52
6	203
7	877
8	4,140
9	21,147
10	115,975
11	678,570
12	4,213,597
13	27,644,437
14	190,899,322
15	1,382,958,545
16	10,480,142,147
17	82,864,869,804
18	682,076,806,159
19	5,832,742,205,057
20	51,724,158,235,372

Произвольный многочлен $P(x)$ от неизвестного x степени n мы можем однозначно представить как $P(x) = \sum_{k=0}^n a_k [x]_k$. Это частный случай того очевидного факта, что существует однозначное разложение $P(x) = \sum_{k=0}^n a_k p_k(x)$ для произвольной последовательности многочленов $p_0(x), p_1(x), \dots$, такой что $p_k(x)$ есть многочлен степени k для каждого $k \geq 0$. Другими словами, каждая такая последовательность образует базис в линейном пространстве многочленов. Оказывается, что числа Стирлинга второго рода в точности равны коэффициентам перехода от базиса $1, x, x^2, \dots$ к базису $1, [x]_1, [x]_2, \dots$

Теорема 1.18. Для каждого $n \geq 0$

$$x^n = \sum_{i=0}^n S(n, k) [x]_k. \tag{1.43}$$

Доказательство. Предположим сначала, что x — неотрицательное целое число. Подсчитаем двумя способами число всех функций $f : A \rightarrow B$, где $|A| = n, |B| = x$. С одной стороны, оно равно x^n (см. теорему 1.1). С другой стороны, множество таких функций f мы можем классифицировать относительно множества $f(A)$. Очевидно, что каждая функция f является отображением множества A на

Таблица 1.3: Числа Белла B_n

множество $f(A)$, таким образом, для произвольного подмножества $Y \subseteq B$, где $|Y| = k$, число всех функций $f : A \rightarrow B$, таких что $f(A) = Y$, равно $s_{n,k}$, т.е. в соответствии с формулой (1.41) $k!S(n, k)$. Учитывая тот факт, что подмножество Y мощности k можно выбрать $\binom{n}{k}$ способами, получаем в конечном итоге

$$x^n = \sum_{k=0}^x \binom{x}{k} k!S(n, k) = \sum_{k=0}^n [x]_k S(n, k) \tag{1.44}$$

(верхний индекс суммирования можно заменить с x на n , так как $S(n, k) = 0$ для $k > n$ и $[x]_k = 0$ для $k < x$). Поскольку равенство многочленов выполняется

$n \setminus k$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	-1	1	0	0	0	0	0	0	0	0
3	0	2	-3	1	0	0	0	0	0	0	0
4	0	-6	11	-6	1	0	0	0	0	0	0
5	0	24	-50	35	-10	1	0	0	0	0	0
6	0	-120	274	-225	85	-15	1	0	0	0	0
7	0	720	-1.764	1.642	-735	175	-21	1	0	0	0
8	0	-5.040	13.068	-13.132	6.769	-1.960	322	-28	1	0	0
9	0	40.320	-109.584	118.124	-67.284	22.449	-4.536	546	-36	1	0
10	0	-362.380	1.026.576	-1.172.700	723.680	-269.325	63.273	-9.450	870	-45	1

Таблица 1.4: Числа Стирлинга первого рода

для произвольного целого числа $x \geq 0$, эти многочлены тождественно равны (так как их разность либо является тождественно равной нулю, либо имеет бесконечное число нулей).

Числа Стирлинга первого рода $s(n, k)$ определяются как коэффициенты при последовательных степенях переменной x в многочлене $[x]_k$:

$$[x]_n = \sum_{k=0}^n s(n, k)x^k. \quad (1.45)$$

Другими словами, числа $s(n, k)$ играют обратную роль в отношении к числам $S(n, k)$ — позволяют перейти от базиса $1, [x]_1, [x]_2, \dots$ к базису $1, x, x^2, \dots$. Очевидно, что $s(n, k) = 0$ для $k > n$. Числа $s(n, k)$ удобно вычислять, пользуясь следующими рекуррентными зависимостями:

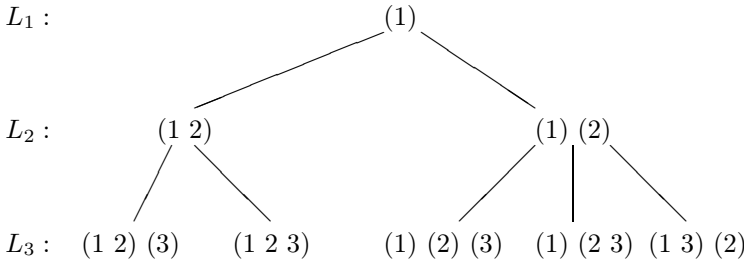
$$s(n, k) = s(n-1, k-1) - (n-1)s(n-1, k) \quad \text{для } 0 < k < n, \quad (1.46)$$

$$s(n, n) = 1 \quad \text{для } n \geq 0, \quad (1.47)$$

$$s(n, n) = 0 \quad \text{для } n > 0. \quad (1.48)$$

Формулы (1.47) и (1.48) очевидны, формулу (1.46) получаем, сравнивая коэффициенты при x^k с обеих частей равенства

$$[x]_n = [x]_{n-1}(x-n+1). \quad (1.49)$$

Рис. 1.12: Построение списков L_1 , L_2 и L_3 .

блок либо создания из него одноэлементного блока. Действительно, последовательные разбиения последовательности (1.50) отвечают этому условию. Если обратить порядок последовательности (1.50) для каждого второго разбиения списка L_{n-1} , то элемент n будет двигаться попеременно вперед и назад, и разбиения «на стыке» последовательностей, образованных из соседних разбиений списка L_{n-1} , будут мало отличаться друг от друга (при условии, что соседние разбиения списка L_{n-1} мало отличаются один от другого). На рис. 1.12 показано построение списка L_n для $n = 1, 2, 3$ (разбиения представлены в несколько упрощенной записи, например $(1\ 2)\ (3)$ означает $\{\{1, 2\}, \{3\}\}$). Дадим сейчас нерекуррентную реализацию этого алгоритма (это измененная версия алгоритма, описанного в [39]). Разбиение множества $\{1, \dots, n\}$ мы будем представлять с помощью последовательности блоков, упорядоченной по возрастанию самого маленького элемента в блоке. Этот наименьший элемент блока мы будем называть номером блока. Отметим, что номера соседних блоков, вообще говоря, не являются соседними натуральными числами. В этом алгоритме мы будем использовать переменные ПЕРЕД[i], СЛЕД[i], $1 \leq i \leq n$, содержащие соответственно номер предыдущего и номер следующего блока для блока с номером i (СЛЕД[i] = 0, если блок с номером i является последним блоком разбиения). Для каждого элемента i , $1 \leq i \leq n$, номер блока, содержащего элемент i , будет храниться в переменной БЛОК[i], направление, в котором «движется» элемент i , будет закодировано в булевой переменной ВПЕР[i] (ВПЕР[i]=true, если i движется вперед; здесь можно заметить некоторое подобие алгоритму 1.12).

Алгоритм 1.19. (Генерирование всех разбиений множества $\{1, 2, \dots, n\}$).

Данные: n .

Результат: Последовательность всех разбиений множества $\{1, \dots, n\}$, в которой каждое следующее разбиение образуется из предыдущего путем перенесения единственного элемента в другой блок.

```

1. begin
2.   for  $i := 1$  to  $n$  do (*поместить  $i$  в первый блок*)
3.     begin БЛОК[ $i$ ] := 1; ВПЕР[ $i$ ] := истина;
4.     end;
5.   СЛЕД[1] := 0;
6.   выписать разбиение;
7.    $j := n$ ; (* $j$ =активный элемент*)
8.   while  $j > 1$  do
9.     begin  $k :=$  БЛОК[ $j$ ];
10.      if ВПЕР[ $j$ ] then (* $j$  движется вперед*)
11.        begin
12.          if СЛЕД[ $k$ ]=0 then (* $k$  есть последний блок*)
13.            begin СЛЕД[ $k$ ] :=  $j$ ; ПРЕД[ $j$ ] :=  $k$ ; СЛЕД[ $j$ ] := 0
14.            end;
15.          if СЛЕД[ $k$ ] >  $j$  then (* $j$  образует новый блок*)
16.            begin ПРЕД[ $j$ ] :=  $k$ ; СЛЕД[ $j$ ] := СЛЕД[ $k$ ];
17.              ПРЕД[СЛЕД[ $j$ ]] :=  $j$ ; СЛЕД[ $k$ ] :=  $j$ 
18.            end;
19.            БЛОК[ $j$ ] := СЛЕД[ $k$ ]
20.          end
21.        else (* $j$  движется назад*)
22.          begin БЛОК[ $j$ ] := ПРЕД[ $k$ ];
23.            if  $k = j$  then (* $j$  образует одноэлементный блок*)
24.              if СЛЕД[ $k$ ] = 0 then СЛЕД[ПРЕД[ $k$ ]] := 0
25.              else begin СЛЕД[ПРЕД[ $k$ ]] := СЛЕД[ $k$ ];
26.                ПРЕД[СЛЕД[ $k$ ]] := ПРЕД[ $k$ ]
27.              end
28.            end;
29.            выписать разбиение;
30.             $j := n$ ;
31.            while ( $j > 1$ ) and
32.              ((ВПЕР[ $j$ ] and (БЛОК[ $j$ ] =  $j$ )) or (not ВПЕР[ $j$ ] and (БЛОК[ $j$ ] = 1))) do
33.                begin ВПЕР[ $j$ ] := not ВПЕР[ $j$ ];  $j := j - 1$ ;
34.              end
35.            end
36.          end

```

Этот алгоритм строит сначала разбиение $\{\{1, \dots, n\}\}$ (цикл 2) — отметим, что это первое разбиение в списке L_n , созданном при помощи описанного нами рекуррентного метода. Задача основного цикла 8 — перемещение «активного» элемента j в соседний блок — предыдущий или последующий (в последнем

(1 2 3 4)
 (1 2 3) (4)
 (1 2) (3) (4)
 (1 2) (3 4)
 (1 2 4) (3)
 (1 4) (2) (3)
 (1) (2 4) (3)
 (1) (2) (3 4)
 (1) (2) (3) (4)
 (1) (2 3) (4)
 (1) (2 3 4)
 (1 4) (2 3)
 (1 3 4) (2)
 (1 3) (2 4)
 (1 3) (2) (4)

Рис. 1.13: Последовательность разбиений множества $\{1, 2, 3, 4\}$, порожденная алгоритмом 1.19

случае может возникнуть необходимость создания нового блока вида $\{j\}$, а затем определение активного элемента во вновь образованном разбиении. Из описанного рекуррентного построения следует, что данный элемент перемещается только тогда, когда все элементы, большие его, достигают своего крайнего левого или правого положения; точнее, активный элемент j^* является таким наименьшим элементом, что для каждого большего элемента j выполняется одно из двух следующих условий:

1. **ВПЕР** $[j]$ **and** (**БЛОК** $[j] = j$), т.е. элемент движется вперед и достигает своего крайнего правого положения (очевидно, j не может быть элементом блока с наименьшим элементом, большим j).
2. **not ВПЕР** $[j]$ **and** (**БЛОК** $[j] = 1$), т.е. элемент j движется назад и достигает своего крайнего левого положения (в первом блоке).

Этот принцип реализуется в строках 30–34. Заодно меняется направление движения элементов $j > j^*$. Дополнительным условием цикла 31 является $j > 1$, так как из самого представления разбиения следует, что $j = 1$ не может быть активным элементом (очевидно, что элемент 1 всегда является элементом блока с номером 1). Если каждый из элементов $j > 1$ отвечает условию (1) или (2), то легко убедиться, что уже порождены все разбиения. В таком случае на выходе

цикла 31 имеем $j = i$ и следует выход из основного цикла 8, т.е. имеем окончание работы алгоритма. Из рекуррентности алгоритма вытекает также, что активным элементом для первого разбиения списка L_n , т.е. для $\{1, \dots, n\}$, является элемент n . Такое же значение приписывается переменной j перед входом в цикл 8 (строка 7).

Проанализируем теперь процесс переноса активного элемента (строки 9–28). Сначала отыскивается номер блока, содержащего активный элемент; пусть это будет k . Если этот элемент движется вперед, достаточно перенести его в блок с номером СЛЕД[k] (см. строку 19), а в двух остальных случаях переменную СЛЕД[k] нужно сначала модифицировать. Первый случай имеет место, когда СЛЕД[k] = 0, т.е. когда k есть номер последнего блока разбиения. Тогда j образует одноэлементный блок; при этом достаточно принять СЛЕД[k] := j и соответственно изменить значения переменных СЛЕД[j] и ПРЕД[j] (см. строку 13). Второй случай имеет место, когда СЛЕД[k] > j , он рассматривается аналогично. Условие СЛЕД[k] > j означает, что все блоки справа от блока с номером k содержат элементы, большие j (все эти элементы занимают свои крайние правые позиции, в противном случае j не был бы активным элементом). Из рекуррентности алгоритма легко вытекает, что в этом случае нужно создать одноэлементный блок, содержащий j . Это выполняется в строках 16–17 (единственная разница с первым случаем состоит в том, что в данном случае вновь созданный блок не является последним блоком разбиения). В ситуации, когда элемент j движется назад (см. строку 21), достаточно поместить его в предыдущий блок (см. строку 22) и выполнить соответствующее изменение значений переменных СЛЕД и ПРЕД, если j создавал одноэлементный блок — это имеет место в точности тогда, когда БЛОК[j] = $k = j$, так как каждый элемент $t > j$ блока с номером j был бы выбран активным элементом в цикле 31.

На рис. 1.13 представлены все разбиения множества $\{1, \dots, 4\}$, порожденные алгоритмом. Можно показать, что среднее число шагов, необходимых для построения каждого следующего разбиения, ограничено постоянной, не зависящей от n (ср. с задачей 1.38; конечно, здесь не учитывается число шагов, необходимых для написания разбиения).

1.11 Разбиения чисел

Займемся теперь следующей задачей: сколькими способами можно записать данное натуральное число n в виде суммы

$$n = b_1 + \dots + b_k, \quad (1.51)$$

где $k, b_1, \dots, b_k > 0$. При этом будем считать суммы эквивалентными, если они отличаются только порядком слагаемых. Класс эквивалентных сумм ви-

да (1.51) мы можем однозначно представить последовательностями a_1, \dots, a_k где $a_1 \geq \dots \geq a_k$ и числа a_1, \dots, a_k являются числами b_1, \dots, b_k , упорядоченными по невозрастанию (аналогичным образом мы представляли подмножества множества возрастающими последовательностями и подмножества множества с повторениями неубывающими последовательностями). Каждую такую последовательность a_1, \dots, a_k будем называть *разбиением числа n на k слагаемых*. Число разбиений числа n на k слагаемых будем обозначать $P(n, k)$, а число всех разбиений числа n (на произвольное число слагаемых) — через $P(n)$. Очевидно, что

$$P(n) = \sum_{i=0}^n P(n, k), \quad n > 0 \tag{1.52}$$

(принимая $P(0, 0) = P(0) = 1$). Например, $P(7) = 15$, а все разбиения числа 7 представлены на рис. 1.14.

Используя очень простой способ представления разбиения числа, называемый *диаграммой Феррерса*, можно доказать много любопытных свойств чисел $P(n)$. Диаграмма Феррерса для разбиения $n = a_1 + \dots + a_k$ состоит из k строк, соответствующих слагаемым разбиения, причем i -я строка содержит последовательность из a_i точек (рис. 1.15).

Каждому разбиению числа n однозначно соответствует *сопряженное разбиение* этого числа, которое получается транспозицией диаграммы Феррерса (перемена ролями строк и столбцов) (см. задачу 1.15). Легко заметить, что транспозиция диаграммы Феррерса определяет взаимно однозначное соответствие между разбиениями числа n на k слагаемых и разбиениями числа n с наибольшим слагаемым, равным k . Отметим этот факт:

Теорема 1.20. *Число разбиений числа n на k слагаемых равно числу разбиений числа n с наибольшим слагаемым, равным k .*

Докажем еще одну теорему этого типа.

Теорема 1.21. *Число разбиений числа n на попарно различные слагаемые равно числу разбиений числа n на нечетные слагаемые.*

Доказательство. Установим взаимно однозначное соответствие между разбиениями, о которых идет речь в теореме. Рассмотрим разбиение числа n на

7							
6	1						
5	2						
5	1	1					
4	3						
4	2	1					
4	1	1	1				
3	3	1					
3	2	2					
3	2	1	1				
3	1	1	1	1			
2	2	2	1				
2	2	1	1	1			
2	1	1	1	1	1		
1	1	1	1	1	1	1	1

Рис. 1.14: Последовательность разбиений числа 7, полученная с помощью алгоритма 1.22

нечетные слагаемые b_1, \dots, b_p , где слагаемое b_i появляется в разбиении r_i раз, $1 \leq i \leq p$. Пусть

$$r_i = 2^{q_1} + 2^{q_2} + \dots \quad (q_1 > q_2 > \dots)$$

есть двоичное представление числа r_i . Произведем теперь замену r_i слагаемых b_i на попарно различные слагаемые

$$b_1 2^{q_1}, \quad b_2 2^{q_2}, \quad \dots$$

(эта замена сохраняет сумму слагаемых разбиения). Повторяя эту операцию для каждого i , $1 \leq i \leq p$, и упорядочивая слагаемые по невозрастанию, получаем в результате разбиения числа n на попарно различные слагаемые. Это следует из того факта, что каждое натуральное число можно однозначно представить в виде произведения нечетного числа на степень двойки. В качестве примера проведем описанное преобразование для разбиения $26 = 7+5+5+3+3+1+1+1$:

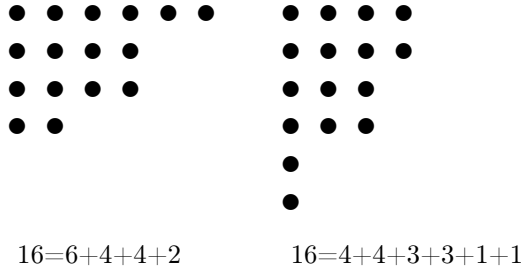


Рис. 1.15: Диаграмма Феррерса для разбиения числа и сопряженного ему разбиения.

$$\begin{aligned} 7 + 5 + 5 + 3 + 3 + 1 + 1 + 1 &= 7 \cdot 2^0 + 5 \cdot 2^1 + 3 \cdot 2^1 + 1 \cdot (2^1 + 2^0) = \\ &= 7 + 10 + 6 + 2 + 1 = 10 + 7 + 6 + 2 + 1. \end{aligned}$$

Легко заметить, что можно выполнить обратное преобразование для произвольного разбиения на попарно различные слагаемые, представляя каждое слагаемое как $p2^q$, где p нечетное, группируя затем слагаемые в зависимости от «нечетного множителя» p и заменяя каждую такую группу $p2^{q_1}, p2^{q_2} \dots$ на $r = 2^{q_1} + 2^{q_2} + \dots$ слагаемых, равных p .

Таким образом, описанное преобразование определяет взаимно однозначное соответствие между разбиениями на нечетные слагаемые и разбиениями на попарно различные слагаемые.

Другие теоремы этого типа сформулированы в задачах 1.39–1.41.

Покажем теперь простой алгоритм генерирования всех разбиений числа. Он будет генерировать разбиения в порядке, обратном лексикографическому, т.е. разбиение

$$n = c_1 + \dots + c_i \tag{1.53}$$

будет порождено — необязательно непосредственно — после разбиения

$$n = a_1 + \dots + a_k \tag{1.54}$$

тогда и только тогда, когда существует индекс $p \leq \min(k, l)$, такой что $c_p < a_p$ и $c_m = a_m$ для $1 \leq m < p$ (ср. с рис. 1.14). Очевидно, что первым разбиением в этом порядке является разбиение, содержащее одно слагаемое, равное n , а последним — разбиение на n слагаемых, равных единице. Зададимся вопросом, как выглядит разбиение, непосредственно следующее за разбиением (1.54). Будем искать разбиение, которое имеет самое большое число начальных слагаемых, равных начальным слагаемым разбиения (1.54) — обозначим эти слагаемые a_1, \dots, a_{t-1} , — и оставшиеся слагаемые которого определяются разбиением, непосредственно идущим за разбиением $s = a_t + a_{t+1} + \dots + a_k$. Легко заметить, что эти условия однозначно определяют

$$t = \max\{i : a_i > 1\}.$$

Таким образом, задача сводится к нахождению разбиения, непосредственно идущего за разбиением

$$s = a_t + \underbrace{1 + \dots + 1}_{k-t \text{ раз}}, \quad a_t > 1.$$

Нетрудно заметить, что такое разбиение имеет вид

$$s = \underbrace{l + \dots + l}_{\lfloor s/l \rfloor \text{ раз}} + (s \bmod l),$$

где $l = a_t - 1$.

В алгоритме, который мы сейчас опишем, разбиение будет представлено переменными $S[1], \dots, S[d]$, содержащими попарно различные слагаемые разбиения ($S[1] > \dots > S[d]$), а также переменными $R[1], \dots, R[d]$, где $R[i]$ содержит информацию о том, сколько раз слагаемое $S[i]$ появляется в разбиении ($R[i] > 0$). Это позволяет находить каждое следующее разбиение за число шагов, ограниченное некоторой постоянной, не зависящей от n .

Алгоритм 1.22. (Нахождение всех разбиений числа)

Данные: n .

Результат: последовательность разбиений числа n в порядке, обратном лексикографическому.

1. **begin**

2. $S[1] := n; R[1] := 1; n := 1; (*\text{первое разбиение}*)$

3. $\text{emph}\text{вывести разбиение};$

4. **while** $S[1] > 1$ **do** $(*\text{нахождение следующего разбиения}*)$

5. **begin** $\text{sum} := 0; (*\text{sum} = \text{сумма устраненных слагаемых}*)$

6. **if** $S[d] = 1$ **then** $(*\text{удаление слагаемых, равных единице}*)$

```

7.      begin  $sum := sum + R[d]; d := d - 1$ 
8.      end;
9.       $sum := sum + S[d]; R[d] := R[d] - 1; l := S[d] - 1;$ 
10.     if  $R[d] > 0$  then  $d := d + 1;$ 
11.      $S[d] := l; R[d] := sum \text{ div } l;$ 
12.      $l := sum \text{ mod } l;$ 
13.     if  $l \neq 0$  then (*добавить последнее слагаемое, равное  $l^*$ )
14.       begin  $d := d + l; S[d] := l; R[d] := 1$ 
15.       end;
16.     выписать разбиение
17.   end
18. end

```

1.12 Производящие функции

В комбинаторных задачах на подсчет числа объектов при наличии некоторых ограничений искомым решением часто является последовательность a_0, a_1, a_2, \dots , где a_k — число искомых объектов «размерности» k . Например, если мы ищем число разбиений числа, то можем принять $a_k = P(k)$, если ищем число подмножеств n -элементного множества, то $a_k = \binom{n}{k}$ и т.д. В этом случае удобно последовательности a_0, a_1, a_2, \dots поставить в соответствие формальный ряд

$$A(x) = \sum_{k=0}^{\infty} a_k x^k, \quad (1.55)$$

называемый *производящей функцией* для данной последовательности. Название «формальный ряд» означает, что (1.55) мы трактуем только как удобную запись нашей последовательности — в данном случае несущественно, для каких (действительных или комплексных) значений переменной x он сходится. Поэтому мы никогда не будем вычислять значение такого ряда для конкретного значения переменной x , мы будем только выполнять некоторые операции на таких рядах, а затем определять коэффициенты при отдельных степенях переменной x . Для произвольных рядов

$$A(x) = \sum_{k=0}^{\infty} a_k x^k, \quad B(x) = \sum_{k=0}^{\infty} b_k x^k,$$

мы определим операцию *сложения*:

$$A(x) + B(x) = \sum_{k=0}^{\infty} (a_k + b_k) x^k, \quad (1.56)$$

операцию *умножения на число* (действительное или комплексное):

$$pA(x) = \sum_{k=0}^{\infty} pa_k x^k, \quad (1.57)$$

и *произведение Коши* (коротко: *произведение*):

$$A(x) \cdot B(x) = \sum_{k=0}^{\infty} c_k x^k, \quad (1.58)$$

где

$$c_k = a_0 b_k + a_1 b_{k-1} + \dots + a_k b_0 = \sum_{i=0}^k a_i b_{k-i}. \quad (1.59)$$

Если $a_k = 0$ для $k > n$, то ряд (1.55) будем отождествлять с многочленом $a_n x^n + \dots + a_0$.

Из математического анализа известно (см., например, [41]), что если ряд (1.55) сходится в некоторой окрестности нуля, то его сумма $A(x)$ является аналитической функцией в этой окрестности и

$$a_k = A^{(k)}(0)/k! \quad k = 0, 1, 2, \dots \quad (1.60)$$

$(A^{(k)}(0))$ обозначает значение k -й производной функции $A(x)$ для $x = 0$; ряд (1.55) — это не что иное, как ряд Маклорена функции $A(x)$. Более того, если $A(x)$, $B(x)$ являются аналитическими функциями в окрестности нуля, то формулы (1.56) — (1.59) будут справедливы, если $A(x)$ и $B(x)$ трактовать как значения функций A и B в точке x , а ряды понимать в обычном смысле, т.е. так, как в математическом анализе. Это сохраняющее операции взаимно однозначное соответствие между рядами, сходящимися в окрестности нуля, и функциями, аналитическими в окрестности нуля, позволяет отождествить формальный ряд (1.55) с определенной через него аналитической функцией в случае рядов, сходящихся в окрестности нуля (несмотря на то, что ряды мы будем трактовать всегда как формальные ряды, т.е. только как формальную запись их коэффициентов). Таким образом, будем писать, например,

$$\sum_{k=0}^{\infty} x^k = (1-x)^{-1},$$

$$\sum_{k=0}^{\infty} \frac{1}{k!} x^k = e^x$$

и т. д.

Найдем теперь производящие функции для некоторых простых последовательностей. В соответствии с (1.13) имеем

$$\sum_{k=0}^{\infty} \binom{n}{k} x^k = \sum_{k=0}^n \binom{n}{k} x^k = (1+x)^n, \quad (1.61)$$

таким образом, для заданного n производящая функция последовательности $\binom{n}{0}, \binom{n}{1}, \binom{n}{2}, \dots$ есть $(1+x)^n$.

Имеем

$$\sum_{k=0}^{\infty} 2^k x^k = \sum_{k=0}^{\infty} (2x)^k = (1-2x)^{-1},$$

и, следовательно, производящая функция последовательности $1, 2, 4, 8, \dots, 2^n, \dots$ есть $(1-2x)^{-1}$. Пользуясь тем, что аналитическую функцию можно дифференцировать почленно, можно написать

$$\sum_{k=0}^{\infty} kx^k = x \sum_{k=0}^{\infty} kx^{k-1} = x \frac{d}{dx} \sum_{k=0}^{\infty} x^k = x \frac{d}{dx} (1-x)^{-1} = x(1-x)^{-2},$$

и, следовательно, производящая функция для последовательности $0, 1, 2, 3, \dots$ есть $x(1-x)^{-2}$.

Познакомимся поближе с формулой (1.61). Каждый множитель $(1+x)$ мы можем трактовать как соответствующий некоторому элементу a_i множества $X = \{a_1, \dots, a_n\}$ и как представляющий два возможных числа появлений этого элемента в подмножестве: нуль раз (слагаемое $x^0 = 1$) и один раз (слагаемое $x^1 = x$). Очевидно, что каждое подмножество множества X однозначно определяется указанием числа появлений в нем каждого из элементов, т.е. выбором одного из слагаемых из каждого множителя произведений $(1+x) \dots (1+x)$. Определенное таким образом слагаемое разложения этого произведения дает вклад в коэффициент при x^k , где k — мощность нашего подмножества, равный единице. Это рассуждение естественным образом переносится на ситуацию, в которой число появлений элемента может быть большим единицы, т.е. на множества с повторениями. Предположим, например, что $X = (2*a_1, 3*a_2, 1*a_3, 4*a_4)$, и обозначим через c_k число k -элементных подмножеств этого множества с повторениями. В силу рассуждения, аналогичного приведенному выше, производящая функция для последовательности c_0, c_1, c_2, \dots равна

$$\begin{aligned} \sum_{k=0}^{\infty} c_k x^k &= (1+x+x^2)(1+x+x^2+x^3)(1+x)(1+x+x^2+x^3+x^4) = \\ &= 1 + 4x + 9x^2 + 15x^3 + 20x^4 + 22x^5 + 20x^6 + 15x^7 + 9x^8 + 4x^9 + x^{10}. \end{aligned}$$

Соответствующим подбором i -го множителя можно накладывать произвольные ограничения на число вхождений элемента a_i . Например, этот множитель имеет вид $(1 + x^3 + x^7)$, если i -й элемент может появляться 0, 3 или 7 раз, $1 + x^2 + x^4 + \dots = (1 - x^2)^{-1}$, если этот элемент может появляться произвольное четное число раз и т.д. В частности, если мы не налагаем никаких ограничений на число вхождений элемента a_i , $1 \leq i \leq n$, то производящая функция имеет вид

$$(1 + x + x^2 + \dots) \dots (1 + x + x^2 + \dots) = (1 - x)^{-1} \dots (1 - x)^{-1} = (1 - x)^{-n}.$$

Если вспомнить, что k -я производная функции $(1 - x)^{-n}$ равна

$$\frac{d^k}{dx^k} (1 - x)^{-n} = (-n)(-n-1) \dots (-n-k+1)(1-x)^{-n-k} (-1)^k = [n]^k (1-x)^{-n-k},$$

и разложить $(1 - x)^{-n}$ в ряд Маклорена, то получим

$$(1 - x)^{-n} = \sum_{k=0}^{\infty} \frac{[n]^k}{k!} x^k = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k,$$

что дает еще одно независимое доказательство теоремы 1.15.

Производящие функции являются также удобным инструментом для доказательства тождеств, связанных с биномиальными коэффициентами. Например, (1.18) можно получить, сравнивая коэффициенты в обеих частях равенства

$$\begin{aligned} \sum_{k=0}^{m+n} \binom{m+n}{k} x^k &= (1+x)^{m+n} = (1+x)^m (1+x)^n = \\ \sum_{i=0}^m \binom{m}{i} x^i \cdot \sum_{j=0}^n \binom{n}{j} x^j &= \sum_{k=0}^{m+n} \sum_{s=0}^k \binom{m}{s} \binom{n}{k-s} x^k \end{aligned}$$

(мы воспользовались здесь формулой произведения Коши двух рядов, в нашем случае это формула произведения многочленов).

Покажем теперь, как производящие функции можно применять в некоторых задачах, связанных с разбиением чисел. Отметим сначала, что каждое разбиение $n = a_1 + \dots + a_k$ можно однозначно представить последовательностью $\langle \lambda_1, \dots, \lambda_n \rangle$, где λ_i обозначает число слагаемых, равных i , т.е. $\lambda_i = |\{j : a_j = i \wedge 1 \leq j \leq k\}|$. Имеем, очевидно,

$$\sum_{i=1}^n i \lambda_i = n$$

и каждая последовательность $\langle \lambda_1, \dots, \lambda_n \rangle$ ($\lambda_1, \dots, \lambda_n \geq 0$) отвечающая этому условию, однозначно определяет разбиение числа n , содержащее λ_i слагаемых, равных i , $1 \leq i \leq n$.

Обозначим через $P_h(n)$ число разбиений числа n на слагаемые, не превышающие h .

Теорема 1.23. Производящая функция для последовательности $P_h(0), P_h(1), P_h(2), \dots$ равна

$$(1 + x + x^2 + x^3 + \dots)(1 + x^2 + x^4 + x^6 + \dots) \dots (1 + x^h + x^{2h} + x^{3h} + \dots) = (1 - x)^{-1}(1 - x^2)^{-1} \dots (1 - x^h)^{-1}$$

Доказательство. Согласно определению произведение h рядов с правой стороны является суммой слагаемых вида $x^{\lambda_1} x^{2\lambda_2} \dots x^{n\lambda_n}$ (λ_i — номер члена, выбранного из i -го ряда). Таким образом, коэффициент при x^n равен числу последовательностей $\langle \lambda_1, \dots, \lambda_n \rangle$, таких что $\sum_{i=1}^h i\lambda_i = n$, а, следовательно, в соответствии с нашими предыдущими замечаниями он равен $P_h(n)$.

Аналогичным образом доказывается следующая теорема:

Теорема 1.24. Производящая функция для последовательности $P(0), P(1), P(2), \dots$ равна

$$(1 + x + x^2 + x^3 + \dots)(1 + x^2 + x^4 + x^6 + \dots) \dots (1 + x^h + x^{2h} + x^{3h} + \dots) \dots = \prod_{h=1}^{\infty} (1 - x^h)^{-1}$$

Замечание. В этом месте следовало бы строго определить ряд, являющийся бесконечным произведением рядов $F_1(x) \cdot F_2(x) \cdot \dots$. Обозначим через $I_n(x)$ «частичное произведение» $F_1(x) \cdot F_2(x) \cdot \dots \cdot F_n(x)$ и через p_n — наименьшую ненулевую степень с ненулевым коэффициентом в $F_n(x)$ (для произведения (1.62) имеем $p_n = n$). Положим, что коэффициент при нулевой степени в каждом из рядов $F_n(x)$ равен единице и что $\lim_{n \rightarrow \infty} p_n = \infty$ (оба этих предположения выполнены для произведения (1.62)). Тогда коэффициент при x^n в $I_m(x)$ один и тот же для всех m , больших некоторого числа m_n (такого, что $p_m > n$ для $m > m_n$). Именно этот коэффициент мы принимаем как коэффициент при x^n в ряде, определенном бесконечным произведением $F_1(x) \cdot F_2(x) \cdot \dots$.

Техника производящих функций позволяет провести простые доказательства некоторых свойств разбиений числа. Приведем в качестве примера другое доказательство теоремы 1.21. Отметим в этой связи, что производящая функция для разбиений на попарно различные слагаемые (т.е. для последовательности r_1, r_2, \dots , где r_n — число разбиений числа n на попарно различные слагаемые) равна

$$R(x) = (1 + x)(1 + x^2)(1 + x^3) \dots (1 + x^k) \dots,$$

а производящая функция для разбиений на нечетные слагаемые равна

$$N(x) = (1 - x)^{-1}(1 - x^3)^{-1} \dots (1 - x^{2k-1})^{-1} \dots$$

Пользуясь зависимостью $1 + x^k = (1 - x^{2k})/(1 - x^k)$, получаем

$$R(x) = \frac{1-x^2}{1-x} \cdot \frac{1-x^4}{1-x^2} \cdot \frac{1-x^6}{1-x^3} \cdot \dots = \frac{1}{1-x} \cdot \frac{1}{1-x^3} \cdot \frac{1}{1-x^5} \cdot \dots = N(x). \quad (1.62)$$

Приведём еще два примера применения производящих функции. Первый относится к так называемым числам Фибоначчи $F(n)$. Эти числа являются решением рекуррентного уравнения

$$F_{n+1} = F_n + F_{n-1}, \quad n \geq 1, \quad (1.63)$$

с начальными условиями

$$F_0 = F_1 = 1 \quad (1.64)$$

Производящая функция $F(x)$ для последовательности F_0, F_1, F_2, \dots удовлетворяет уравнению

$$\begin{aligned} F(x) &= \sum_{k=0}^{\infty} F_k x^k = 1 + x + \sum_{k=2}^{\infty} (F_{k-2} + F_{k-1}) x^k = \\ &= 1 + x + x^2 \sum_{k=2}^{\infty} F_{k-2} x^{k-2} + x \sum_{k=2}^{\infty} F_{k-1} x^{k-1} = \\ &= 1 + x + x^2 F(x) + x(F(x) - 1) = 1 + (x + x^2)F(x). \end{aligned}$$

Отсюда получаем производящую функцию для чисел Фибоначчи

$$F(x) = (1 - x - x^2)^{-1}$$

Найдя корни уравнения $1 - x - x^2 = 0$, получаем разложение $1 - x - x^2 = (1 - ax)(1 - bx)$, где

$$a = \frac{1 + \sqrt{5}}{2}, \quad b = \frac{1 - \sqrt{5}}{2}$$

Теперь, используя метод неопределенных коэффициентов, найдём

$$\frac{1}{(1 - ax)(1 - bx)} = \frac{A}{1 - ax} + \frac{B}{1 - bx} = \frac{A + B - (Ab + Ba)x}{(1 - ax)(1 - bx)}.$$


Сравнивая коэффициенты в числителе, получаем $A + B = 1$, т.е. $B = 1 - A$, а далее $Ab + (1 - A)a = 0$, что дает $A = a/(a - b)$, $B = b/(a - b)$; таким образом,

$$F(x) = A(1 - ax)^{-1} + B(1 - bx)^{-1} = A \sum_{k=0}^{\infty} a^k x^k + B \sum_{k=0}^{\infty} b^k x^k = \sum_{k=0}^{\infty} \frac{a^{k+1} - b^{k+1}}{a - b} x^k$$

и окончательно

$$F_k = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^{k+1} - \left(\frac{1 - \sqrt{5}}{2} \right)^{k+1} \right]. \quad (1.65)$$

$$c_0 = 1$$

$$c_1 = 1$$


$$c_2 = 2$$


$$c_3 = 5$$


Рис. 1.16: Бинарные деревья с k вершинами, $k=0,1,2,3$.

Этот метод можно перенести на произвольные линейные рекуррентные уравнения с постоянными коэффициентами (см. задачу 1.46).

В качестве последнего примера применения производящих функций вычислим число бинарных деревьев с n вершинами. Под бинарным деревом с n вершинами мы понимаем пустое дерево $T = \emptyset$, если $n = 0$, или тройку $T = \langle L, r, P \rangle$, где r — вершина, называемая корнем дерева, L (левое поддерево) — бинарное дерево с l вершинами, P (правое поддерево) — бинарное дерево с p вершинами и $l + p + 1 = n$. Будем говорить, что бинарные деревья T_1 и T_2 изоморфны, и писать $T_1 \approx T_2$, если $T_1 = T_2 = \emptyset$ или $T_1 = \langle L_1, r_1, P_1 \rangle$, $T_2 = \langle L_2, r_2, P_2 \rangle$, где $L_1 \approx L_2$ и $P_1 \approx P_2$. Обозначим через c_k число неизоморфных бинарных деревьев с k вершинами (рис. 1.16).

Из данного рекурсивного определения следует, что $c_0 = 1$ и если $0 \leq s \leq k$, то существует в точности $c_s c_{k-1-s}$ неизоморфных деревьев вида $\langle L, r, P \rangle$, где L — бинарное дерево с s вершинами. Число s может принимать любое из значений между 0 и $k - 1$, следовательно,

$$c_k = c_0 c_{k-1} + c_1 c_{k-2} + \dots + c_{k-1} c_0, \quad k > 0. \quad (1.66)$$

Неизоморфные бинарные деревья для $k = 0, 1, 2, 3$ представлены на рис. 1.16.

Рассмотрим производящую функцию $C(x) = \sum_{k=0}^{\infty} c_k x^k$. Равенство (1.66) весьма напоминает формулу для коэффициентов произведения Коши $C(x)C(x) =$

$C^2(x)$, точнее говоря, имеет место уравнение

$$C(x) = xC^2(x) + 1$$

или

$$xC^2(x) - C(x) + 1 = 0. \quad (1.67)$$

Покажем теперь, что существует функция $C(x)$, аналитическая в окрестности нуля, удовлетворяющая этому уравнению. В силу упомянутого выше взаимно однозначного соответствия между рядами и аналитическими функциями коэффициенты этого решения определяют формальный ряд, удовлетворяющий уравнению (1.67). Рассматривая (1.67) как квадратное уравнение с неизвестной $C(x)$ (значение искомой аналитической функции в точке x), получаем для $x \neq 0$

$$C(x) = \frac{1 \pm \sqrt{1 - 4x}}{2x} \quad (1.68)$$

Разложим $\sqrt{1 - 4x} = (1 - 4x)^{1/2}$ в ряд Маклорена. Для $k > 0$ имеем

$$\begin{aligned} \frac{d^k}{dx^k} (1 - 4x)^{\frac{1}{2}} &= \frac{1}{2} \cdot \left(\frac{1}{2} - 1\right) \left(\frac{1}{2} - 1\right) \dots \left(\frac{1}{2} - k + 1\right) (1 - 4x)^{\frac{1}{2} - k} (-4)^k = \\ &= 2^k (-1) \cdot 1 \cdot 3 \cdot 5 \cdot \dots \cdot (2k - 3) (1 - 4x)^{\frac{1}{2} - k} \end{aligned}$$

и, следовательно,

$$\begin{aligned} \sqrt{1 - 4x} &= 1 - \sum_{k=1}^{\infty} \frac{2^k \cdot 1 \cdot 3 \cdot 5 \cdot \dots \cdot (2k - 3)}{k!} x^k = \\ &= 1 - \sum_{k=1}^{\infty} \frac{2^k \cdot (2k - 2)!}{k! \cdot 2 \cdot 4 \cdot \dots \cdot (2k - 2)} x^k = 1 - 2 \sum_{k=1}^{\infty} \frac{(2k - 2)!}{k! (k - 1)!} x^k = \\ &= 1 - 2 \sum_{k=1}^{\infty} \frac{1}{k} \binom{2k - 2}{k - 1} x^k \end{aligned}$$

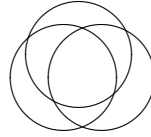
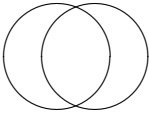
Отсюда видно, что для получения решения с положительными коэффициентами следует выбрать знак минус в (1.68). Таким образом, имеем

$$C(x) = \frac{1 - \sqrt{1 - 4x}}{2x} = \sum_{k=1}^{\infty} \frac{1}{k} \binom{2k - 2}{k - 1} x^{k-1} = \sum_{k=1}^{\infty} \frac{1}{k + 1} \binom{2k}{k} x^k$$

Отсюда окончательно получаем

$$c_k = \frac{1}{k + 1} \binom{2k}{k} \quad (1.69)$$

Числа c_k называются *числами Каталана*; они часто появляются в контексте целого ряда других комбинаторных задач (см. задачи 1.47, 1.48). *число ! Каталана*



$$|A \cup B| = |A| + |B| - |A \cap B| \quad |A \cup B \cup C| = |A| + |B| + |C| -$$

$$- |A \cap B| - |A \cap C| - |A \cap B \cap C|$$

Рис. 1.17: Простые частные случаи принципа включения и исключения

1.13 Принцип включения и исключения

Основная теорема, которую мы докажем в этом разделе, есть обобщение очевидной формулы

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

которая справедлива для произвольных множеств A, B (рис. 1.17).

Предположим, что даны подмножества A_1, \dots, A_n (необязательно различные) некоторого конечного множества X , и мы ищем мощность их объединения $A_1 \cup \dots \cup A_n$. В качестве первого «приближения» этой мощности мы можем принять

$$|A_1| + \dots + |A_n|. \quad (1.70)$$

однако это число в общем случае слишком большое, так как если $A_i \cap A_j \neq \emptyset$, то элементы пересечения $A_i \cap A_j$ считаются дважды. Попробуем исправить ситуацию, вычитая из (1.70) сумму

$$\sum_{1 \leq i < j \leq n} |A_i \cap A_j|. \quad (1.71)$$

Но в таком случае мы получим слишком маленькое число, так как если $A_i \cap A_j \cap A_k \neq \emptyset$, то элементы пересечения $A_i \cap A_j \cap A_k$ считаются в (1.71) трижды. Следующим шагом может быть добавление суммы $\sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k|$, но по причине, подобной предыдущему случаю, получим слишком большое число. Все же оказывается, что через n шагов мы всегда получаем правильный результат. То есть имеет место следующая теорема.

Теорема 1.25 (*Принцип включения и исключения*).

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n-1} |A_1 \cap \dots \cap A_n|$$

Доказательство. Применим индукцию по n . Для $n = 1$ теорема очевидно справедлива. Предположим, что для произвольных подмножеств A_1, \dots, A_{n-1}

$$\left| \bigcup_{i=1}^{n-1} A_i \right| = \sum_{i=1}^{n-1} |A_i| - \sum_{1 \leq i < j \leq n-1} |A_i \cap A_j| + \dots + (-1)^{n-2} |A_1 \cap \dots \cap A_{n-1}|$$

Применяя эту формулу к сумме

$$(A_1 \cup \dots \cup A_{n-1}) \cap A_n = \bigcup_{i=1}^{n-1} (A_i \cap A_n),$$

получаем

$$\left| \bigcup_{i=1}^{n-1} A_i \cap A_n \right| = \sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1 \leq i < j \leq n-1} |A_i \cap A_j \cap A_n| + \dots + (-1)^{n-2} |A_1 \cap \dots \cap A_n|,$$

а отсюда

$$\begin{aligned} \left| \bigcup_{i=1}^n A_i \right| &= \left| \left(\bigcup_{i=1}^{n-1} A_i \right) \cup A_n \right| = \left| \bigcup_{i=1}^{n-1} A_i \right| + |A_n| - \left| \bigcup_{i=1}^{n-1} A_i \cap A_n \right| = \\ &= \sum_{i=1}^n |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \dots + (-1)^{n-1} |A_1 \cap \dots \cap A_n|. \end{aligned}$$

Покажем теперь несколько примеров применения принципа включения и исключения.

Теорема 1.26. *Если $|X| = n$, $|Y| = m$, то число всех функций из X на Y (т.е. функций $f: X \rightarrow Y$, таких что $f(X) = Y$), равно*

$$s_{n,m} = \sum_{i=0}^{m-1} (-1)^i \binom{m}{i} (m-i)^n. \quad (1.72)$$

Доказательство. Пусть $Y = \{y_1, \dots, y_m\}$. Обозначим через A_i множество таких функций $f: X \rightarrow Y$, для которых $y_i \notin f(X)$. Тогда очевидно, что

$$f(X) \neq Y \Leftrightarrow f \in \bigcup_{i=1}^m A_i$$

Мы знаем, что число всех функций $f : X \rightarrow Y$ равно m^n (теорема 1.1), достаточно, таким образом, определить мощность объединения $A_1 \cup \dots \cup A_m$. Отметим сначала, что для произвольной последовательности $1 \leq p_1 < \dots < p_i \leq m$ пересечение $A_{p_1} \cap \dots \cap A_{p_i}$ есть множество всех функций $f : X \rightarrow Y$, таких что $y_{p_1}, \dots, y_{p_n} \notin f(X)$, а, следовательно, мощность этого пересечения составляет $(m - p)^n$ столько, сколько имеется функций $f : X \rightarrow Y \setminus \{y_{p_1}, \dots, y_{p_n}\}$. i -элементное множество $\{y_{p_1}, \dots, y_{p_n}\}$ мы можем выбрать $\binom{m}{i}$ способами, и, следовательно, согласно принципу включения и исключения имеем

$$\begin{aligned} s_{n,m} &= n^m - \left| \bigcup_{i=0}^{n-1} A_i \right| = m^n - \sum_{i=0}^{n-1} (-1)^{i-1} \binom{m}{i} (m-i)^n = \\ &= \sum_{i=0}^{m-1} (-1)^i \binom{m}{i} (m-i)^n. \end{aligned}$$

Стоит отметить, что в силу формулы (1.41) теорема дает простую формулу для вычисления чисел Стирлинга второго рода:

$$S_{n,k} = \frac{1}{k!} s_{n,k} = \frac{1}{k!} \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^n. \quad (1.73)$$

Другим применением принципа включения и исключения может служить определение числа подмножеств из k элементов множества с повторениями $X = (k_1 * a_1, \dots, k_n * c_n)$ (пока напомним только, что таких подмножеств насчитывается $\binom{n}{k}$, если $k_1 = \dots = k_n = k$, и $\binom{n+k-1}{k}$, если $k_1, \dots, k_n \geq k$). Заметим, что эта проблема равносильна проблеме определения числа целочисленных решений системы

$$\begin{aligned} x_1 + \dots + x_n &= k, \\ 0 \leq x_i &\leq k_i \text{ для } i = 1, \dots, n. \end{aligned}$$

Метод решения этой задачи проиллюстрируем на примере. Пусть $X = (4 * a_1, 3 * a_2, 7 * a_3)$, $k = 11$. Создадим множества

- A = множество всех k -элементных подмножеств множества с повторениями $(k * a_1, k * a_2, k * a_3)$,
- A_1 = множество таких $Y \in A$, которые содержат более чем 4 вхождения элемента a_1 ,
- A_2 = множество таких $Y \in A$, которые содержат более чем 3 вхождения элемента a_2 ,
- A_3 = множество таких $Y \in A$, которые содержат более чем 7 вхождений элемента a_3 .

Имеем $|A| = \binom{n+k-1}{k} = \binom{3+11-1}{11} = \binom{13}{11} = 78$, очевидно, что $|A_1|$ составляет столько, сколько имеется 6-элементных ($k-5=6$) подмножеств множества с повторениями ($11 * a_1, 11 * a_2, 11 * a_3$), т.е. $\binom{3+6-1}{6} = \binom{8}{6} = 28$, аналогично $|A_2| = \binom{3+7-1}{7} = \binom{9}{7} = 36$, $|A_3| = \binom{3+3-1}{3} = \binom{5}{3} = 10$. Также легко заметить, что $|A_1 \cap A_2| = \binom{3+2-1}{2} = \binom{4}{2} = 6$ столько, сколько имеется двухэлементных ($k-5-4=2$) подмножеств множества с повторениями ($11 * a_1, 11 * a_2, 11 * a_3$), и $A_2 \cap A_3 = A_1 \cap A_3 = \emptyset$.

Подмножество $Y \in A$ является подмножеством множества с повторениями X тогда и только тогда, когда оно не принадлежит A_1, A_2 и A_3 , а следовательно, по принципу включения и исключения искомое число k -элементных подмножеств множества с повторениями X равно $|A| - |A_1| - |A_2| - |A_3| + |A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3| - |A_1 \cap A_2 \cap A_3| = 78 - 28 - 36 - 10 + 6 + 0 + 0 - 0 = 10$.

Наконец мы остановимся еще на одном применении принципа включения и исключения: на определении числа инверсий на n -элементном множестве. Под *инверсией* на множестве $\{1, \dots, n\}$ мы будем понимать произвольную перестановку f этого множества, такую что $f(i) \neq i$ для $1 \leq i \leq n$. Обозначим через D_n множество всех инверсий на множестве $\{1, \dots, n\}$ и

$$A_i = \{f \in S_n : f(i) = i\}, \quad i = 1, \dots, n$$

(напомним, что S_n означает множество всех перестановок множества $\{1, \dots, n\}$). Перестановка f является инверсией тогда и только тогда, когда она не принадлежит ни к одному из множеств A_i , а, следовательно, согласно принципу включения и исключения

$$|D_n| = |S_n| - \sum_{i=1}^n |A_i| + \sum_{1 \leq i < j \leq n} |A_i \cap A_j| - \dots + (-1)^n |A_1 \cap \dots \cap A_n|.$$

Для произвольной последовательности $1 \leq p_1 < \dots < p_i \leq n$ пересечение $A_{p_1} \cap \dots \cap A_{p_i}$ является множеством таких перестановок f , для которых $f(p_j) = p_j$ для $1 \leq j \leq i$, и значит, $|A_{p_1} \cap \dots \cap A_{p_i}| = (n-i)!$. Заметив, что последовательность $1 \leq p_1 < \dots < p_i \leq n$ можно выбрать $\binom{n}{i}$ способами, получаем в итоге

$$\begin{aligned} |D_n| &= \sum_{i=0}^n (-1)^i \binom{n}{i} (n-i)! = \sum_{i=0}^n (-1)^i \frac{n!}{i!} = \\ &= n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!} \right). \end{aligned}$$

Стоит отметить, что сумма в скобках является начальным членом ряда $e^{-1} = \sum_{i=0}^{\infty} (-1)^i \frac{1}{i!}$. Это означает, что инверсии составляют $e^{-1} = 0.36788\dots$ всех перестановок.

1.14 Задачи

1. В соревнованиях принимают участие 8 спортсменов. Сколькими способами могут быть разделены медали (золотые, серебряные и бронзовые)?
2. Доказать, что число способов, которыми можно рассадить n человек среди m человек за круглым столом, равно $[m]_n/n$. (Размещения, отличающиеся только циклическим перемещением вокруг стола, считаем равными).
3. Сколько чисел между 1000 и 10000 состоят из нечётных цифр, сколько из различных цифр?
4. Сколько существует возможных результатов, которыми могут окончиться соревнования, в которых стартует 10 человек в трёх видах спорта, если каждый человек стартует в одном, произвольно выбранном виде спорта? (Под результатом соревнования мы будем понимать распределение мест для всех спортсменов, стартующих в каждом виде).
5. Сколько палиндромов длины n можно образовать, используя 26 букв алфавита? (*Палиндромом* называется произвольное выражение, которое читается одинаково как слева направо, так и в обратном направлении, например, топот).
6. Доказать, что порядок произвольной группы $G \subseteq S_n$ является делителем порядка S_n . (*Указание:* показать, что множества вида $fG = \{fg : g \in G, f \in S_n\}$ составляют разбиение группы S_n на непересекающиеся $|G|$ -элементные блоки).
7. Доказать, что число перестановок $f \in S_n$ типа $1^{\lambda_1} 2^{\lambda_2} \dots n^{\lambda_n}$ равно
$$\frac{n!}{(1^{\lambda_1} 2^{\lambda_2} \dots n^{\lambda_n} \lambda_1! \lambda_2! \dots \lambda_n!)}$$
8. Доказать, что перестановки $f, g \in S_n$ являются перестановками одного и того же типа тогда и только тогда, когда существует перестановка $h \in S_n$, такая, что $g = hfh^{-1}$.
9. Перестановка $f \in S_n$ называется *инволюцией*, если $f \cdot f = e$. Доказать, что $f \in S_n$ является инволюцией тогда и только тогда, когда она имеет тип $1^{\lambda_1} 2^{\lambda_2}$ ($\lambda_1 + 2\lambda_2 = n$), и что произвольная перестановка является суперпозицией двух инволюций.
10. Для перестановки $\langle a_1, \dots, a_n \rangle$ определим *инверсивный вектор* как $\langle d_1, \dots, d_n \rangle$, где $d_i = |\{j < i : a_j > a_i\}|$. Доказать, что инверсивный вектор однозначно определяет перестановку.

11. Доказать, что множество всех чётных перестановок образует группу.
12. Доказать, что алгоритм 1.10 будет генерировать все перестановки (в несколько иной очерёдности), если убрать команду $P[i] := P[m]$ в строке 15. Пусть $f_1, f_2, \dots, f_n!$ будет последовательность перестановок, полученных с помощью такого модифицированного алгоритма. Доказать, что последовательность $n!$ элементов $f_1^{-1}(i), f_2^{-1}(i), \dots, f_n^{-1}(i)$ является для каждого i ($1 \leq i \leq n$) одинаковой с точностью до циклического сдвига (ср. [45]).
13. Доказать, что асимптотическое значение (для $n \rightarrow \infty$) среднего числа транспозиций, приходящихся на каждую перестановку, полученную при помощи алгоритма 1.10, равно $\cosh 1 = 1,543\dots$ $\left(\cosh x = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}\right)$, если же алгоритм модифицирован так, как в предыдущей задаче, то это значение составляет $\sinh 1 = 1,175\dots$ $\left(\sinh x = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}\right)$.
14. Доказать, что каждое из чисел $0, \dots, n! - 1$ можно однозначно представить в виде $j = \sum_{k=1}^{n-1} d_k k!$, где $0 \leq d_i \leq i$, причём последовательности d_{k-1}, \dots, d_1 , соответствующие очередным числам, появляются в лексикографическом порядке. Преложите алгоритм построения последовательности $\langle d_{k-1}, \dots, d_1 \rangle$, соответствующей числу j .
15. Удалить рекурсию из алгоритма 1.11. Описать такую реализацию, в которой число шагов, необходимых для построения каждой следующей перестановки (не только среднее значение числа шагов!) ограничено константой, не зависящей от n . (*Указание:* рассмотрим лексикографически упорядоченный список L_n всех последовательностей $\langle c_n, \dots, c_2 \rangle$, где $1 \leq c_i \leq i$. Доказать, что i -я транспозиция, проведённая алгоритмом 1.11, имеет вид $P[B[p, c_p]] := P[p]$, где $p = \min\{i : c_i < i\}$, а $\langle c_n, \dots, c_2 \rangle$ есть j -я последовательность в списке L_n . Дальнейшие последовательности в списке L_n можно эффективно генерировать, представляя последовательность $\langle c_n, \dots, c_2 \rangle$ последовательностью $\langle \bar{c}_n, \dots, \bar{c}_2 \rangle$, где $\bar{c}_i = 1$, если $c_i = i$ и $\bar{c}_i = c_i$ в противном случае, или же, используя буфер, фактическим содержанием которого всегда является последовательность $a_1, b_1, a_2, b_2, \dots, a_s, b_s$, такая, что

$$\bigcup_{q=1}^s \{i : a_q \leq i \leq b_q\} = \{i : c_i < i\}$$

Наибольший элемент буфера определяет ранее упомянутый индекс.)

16. Написать программу, реализующую «обычную» нерекурсивную версию алгоритма 1.11, а также версию, описанную в предыдущей задаче. Проверить эмпирически, выполнив программу, сильнее ли последняя версия в смысле времени, необходимого для построения каждой следующей перестановки.
17. Предположим, что для каждого нечётного m имеет место соотношение $B[m, 1] = \dots = B[m, m - 1] = b^{(m)}$, и что для произвольного нечётного $m \geq 4$ последовательность $B[m, 1], \dots, B[m, m - 1]$ удовлетворяет одному из следующих условий:

(i) Каждый из элементов $1, \dots, m - 1$ появляется в ней в точности один раз, причём расстояние между позициями, содержащими $m - 1$ и $b^{(m-1)}$ чётное (расстояния между позициями i, j понимаем как $|i - j|$).

(ii) Каждый из элементов множества $\{1, \dots, m - 1\} \setminus \{m - 1, b^{(m-1)}\}$ появляется дважды, причём расстояние между этими вхождениями нечётное.

Доказать, что в этом случае процедура $PERM(n)$ генерирует все перестановки элементов $P[1], \dots, P[n]$, причём φ_n (см. доказательство корректности алгоритма 1.11) является транспозицией для каждого нечётного $n \geq 3$ и циклом длины n для каждого чётного n (ср. [44]).

18. Пользуясь предыдущим заданием, доказать, что следующие варианты процедуры $B(m, i)$ в алгоритме 1.11 приводят к правильному алгоритму генерирования всех перестановок:

- (1) **if** ($m \bmod 2 = 0$) **then**
 if $i > 1$ **then** $B := m - i$
 else $B := m - 2$
 else $B := m - 1$
- (2) **if** $m \bmod 2 = 0$ **then**
 if ($i = 1$) **or** ($i = m - 1$) **then** $B := m - 1$
 else $B := i$
 else $B := 1$
- (3) **if** ($m \bmod 2 = 0$) **and** ($i > 2$) **then** $B := m - i$
 else $B := m - 1$ (*Wells[71]*)
- (4) **if** $m \bmod 2 = 0$ **then** $B := i$
 else $B := 1$ (*Heap[32]*)

(5) **if** $m \bmod 2 = 0$ **then** $B := i$
else $B := m - 2$

Какой вид имеет перестановка φ_n в каждом из этих случаев? Дать другие возможные варианты этой процедуры.

19. Доказать, что среднее число шагов, необходимое для генерирования каждой следующей перестановки в алгоритме 1.12, ограничено константой, не зависящей от n . Представить такой вариант алгоритма, в котором это замечание справедливо также для числа шагов для каждой конкретной перестановки (ср. с задачей 1.15).
20. Алгоритмы 1.11 и 1.12 генерируют последовательности, содержащие попеременно чётные и нечётные перестановки (почему?). Верно ли это замечание для алгоритма 1.10?
21. Доказать, что среднее число шагов, необходимое для генерирования каждого следующего подмножества с помощью алгоритма 1.13, ограничено константой, не зависящей от n . Предложить вариант алгоритма, в котором это верно также для каждого конкретного подмножества (см. задачу 1.15).
22. Предложить нерекурсивную версию алгоритма, генерирующего все подмножества с повторениями.
23. Использовать алгоритм 1.13 для решения следующей задачи: даны $n + 1$ d -мерных векторов a_1, \dots, a_n, s с целыми координатами. Проверить, существует ли подмножество $J \subseteq \{1, \dots, n\}$, такое что $\sum_{j \in J} a_j = s$.
24. Доказать тождества

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n-1}{k-1} + \binom{n-2}{k-2} + \dots + \binom{n-k}{0},$$

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n-1}{k} + \binom{n-2}{k} + \dots + \binom{n-k}{k}$$

двумя способами: многократно применяя формулу (1.17) и придавая правой части некоторую комбинаторную интерпретацию (например, в первом тождестве $\binom{n-j}{k-j}$ есть число k -элементных подмножеств $A \in \{0, \dots, n\}$, таких что $\min\{i : i \notin A\} = j$).

25. Доказать тождество $n \binom{n-1}{k-1} = k \binom{n}{k}$, подсчитав двумя способами пары $\langle a, K \rangle$, где $a \in K$ и K является k -элементным подмножеством фиксированного n -элементного множества X . Использовать это тождество для вывода формулы (1.21)
26. Доказать, что произведение k произвольных последовательных натуральных чисел делится на $k!$. (*Указание:* рассмотреть биномиальный коэффициент $\binom{n+k}{k}$).
27. Доказать, что для произвольных натуральных чисел p, q, n

$$[p + q]_n = \sum_{k=0}^n \binom{n}{k} [p]_k [q]_{n-k},$$

$$[p + q]^n = \sum_{k=0}^n \binom{n}{k} [p]^k [q]^{n-k}.$$

(*Указание:* для доказательства первого тождества подсчитайте двумя способами число всех взаимно однозначных функций $f : X \rightarrow P \cup Q$, где $|X| = n$, $|P| = p$, $|Q| = q$, $P \cap Q = \emptyset$. Для второго тождества найти подобную интерпретацию в терминах размещений, упорядоченных в $p + q$ ячейках.)

28. Доказать, что

$$(x_1 + x_2 + \dots + x_p)^n = \sum_{n_1 + \dots + n_p = n} \binom{n}{n_1 n_2 \dots n_p} x_1^{n_1} \dots x_p^{n_p},$$

где $\binom{n}{n_1 n_2 \dots n_p} = \frac{n!}{n_1! n_2! \dots n_p!}$. Дайте комбинаторную интерпретацию коэффициентов. Чему равно $\binom{n}{kn-k}$?

29. Дать нерекурсивную версию алгоритма генерирования k -элементных подмножеств, основанную на формуле (1.25). Показать, что рекуррентная зависимость

$$G(n, k) = G(n-1, k), \\ G(n-2, k-2) \cup \{n-1, n\}, G^*(n-2, k) \cup \{n\}$$

определяет также некоторый список k -элементных подмножеств, в котором соседние подмножества мало отличаются друг от друга. Дать нерекурсивный алгоритм, основанный на этой зависимости.

30. Показать, что решётка разбиений не является дистрибутивной, т.е. $\pi \wedge (\sigma \vee \mu) \neq (\pi \wedge \sigma) \vee (\pi \wedge \mu)$. Указать соответствующие разбиения π , σ , μ .
31. Доказать, что число последовательностей длины n с элементами из k -элементного множества, содержащих каждый элемент этого множества по крайней мере один раз, равно $k!S(n, k)$.
32. Разбиение π n -элементного множества X имеет тип $1^{\lambda_1} 2^{\lambda_2} \dots n^{\lambda_n}$ (где $\lambda_1 + 2\lambda_2 + \dots + n\lambda_n = n$), если оно содержит λ_i i -элементных блоков для $i = 1, 2, \dots, n$. Доказать, что число разбиений типа $1^{\lambda_1} 2^{\lambda_2} \dots n^{\lambda_n}$ равно $\frac{n!}{(\lambda_1! \dots \lambda_n! (1!)^{\lambda_1} \dots (n!)^{\lambda_n})}$.
33. Обозначим для каждого $k \geq 0$

$$g^{(k)} = \frac{d^k}{dx^k} g(x), \quad f^{(k)} = \frac{d^k}{dy^k} f(y) \Big|_{y=g(x)}$$

Доказать, что

$$\frac{d^n}{dx^n} f(g(x)) = \sum_{j=0}^n \sum_{\substack{k_1+k_2+\dots+k_n=j \\ k_1+2k_2+\dots+nk_n=n \\ k_1, k_2, \dots, k_n \geq 0}} f^{(j)} \frac{n! (g^{(1)})^{k_1} \dots (g^{(n)})^{k_n}}{k_1! (1!)^{k_1} \dots k_n! (n!)^{k_n}}$$

Показать, что сумма коэффициентов при всех слагаемых вида $f^{(j)} (g^{(1)})^{k_1} \dots (g^{(n)})^{k_n}$ равна числу Белла B_n .

34. Доказать, что для каждого $m \geq 0$ матрица чисел Стирлинга первого рода $[s(n, k)]_{0 \leq n, k \leq m}$ есть обратная матрица для матрицы чисел Стирлинга второго рода $[S(n, k)]_{0 \leq n, k \leq m}$.
35. Доказать, что
- $$[x]^n = \sum_{k=0}^n |s(n, k)| x^k$$
36. Доказать, что $|s(n, k)|$ есть число тех перестановок n -элементного множества, которые имеют в разбиении на циклы в точности k циклов.
37. Рассмотрим следующую программу:

```

min := ∞
for i:=1 to n do
  if P[i] < min then min:=P[i];

```

Доказать, что вероятность того, что для произвольно выбранной перестановки $P[1], \dots, P[n]$ команда $min := P[i]$ будет выполняться в точности k раз, составляет $|s(n, k)|/n!$. Доказать, что среднее число выполнений указанной команды для произвольно выбранной перестановки есть $O(\log n)$, точное значение равно $\sum_{k=1}^n \frac{1}{k!}$. (Указание: подсчитать число циклов во всех $n!$ перестановках.)

38. Доказать, что среднее число шагов, необходимых для построения каждого следующего разбиения в алгоритме 1.19, ограничено константой, не зависящей от n . Дать модификацию алгоритма, гарантирующую ограниченность этого числа константой для каждого порождённого разбиения (ср. с задачей 1.15).
39. Доказать, что число разбиений числа n , в котором ни одно из слагаемых не превосходит k , равно числу разбиений числа $n + k$ на k слагаемых, т.е. $P(n + k, k)$.
40. Доказать, что число *самоспряжённых* разбиений числа n (т.е. равных сопряжённому разбиению) равно числу разбиений числа n на попарно различные нечётные слагаемые.
41. Обозначим через $E(n)$ и $O(n)$ соответственно число разбиений числа n на попарно различные чётные слагаемые и на попарно различные нечётные слагаемые. Доказать, что

$$E(n) - O(n) = \begin{cases} (-1)^k, & \text{если } n \text{ имеет вид } (3k^2 \pm k)/2 \\ 0 & \text{в противном случае} \end{cases}$$

42. Разработать алгоритм генерирования всех $P(n, k)$ разбиений числа n на k слагаемых в порядке, обратном лексикографическому, такой что число шагов, необходимых для построения каждого следующего разбиения, ограничено константой, не зависящей от n .
43. Ряд

$$\sum_{k=0}^{\infty} \frac{c_k}{k!} x^k$$

будем называть *экспоненциальной производящей функцией* для последовательности c_0, c_1, \dots . Пусть c_k — число различных последовательностей длины k с элементами из множества $Y = \{y_1, \dots, y_n\}$, в которых число появлений элемента y_i принадлежит множеству $\{m_{i1}, m_{i2}, \dots\}$ ($m_{i1} < m_{i2} < \dots$). Доказать, что экспоненциальная производящая функция для последовательности c_0, c_1, \dots есть

$$c(x) = \left(\frac{x^{m_{11}}}{m_{11}!} + \frac{x^{m_{12}}}{m_{12}!} + \dots \right) \left(\frac{x^{m_{21}}}{m_{21}!} + \frac{x^{m_{22}}}{m_{22}!} + \dots \right) \dots \left(\frac{x^{m_{n1}}}{m_{n1}!} + \frac{x^{m_{n2}}}{m_{n2}!} + \dots \right)$$

44. Доказать, что

$$F_{n+1} = \sum_{k=0}^n \binom{n-k+1}{k} = \sum_{k=0}^{\lfloor (n+1)/2 \rfloor} \binom{n-k+1}{k}$$

45. Доказать, что число бинарных последовательностей длины n , не содержащих единиц ни на каких двух соседних позициях, равно числу Фибоначчи F_{n+1} .

46. Описать метод решения возвратных уравнений вида

$$C_0 a_n + C_1 a_{n-1} + \dots + C_r a_{n-r} = 0$$

(с начальными условиями, определяющими значения a_0, a_1, \dots, a_{n-1}) при помощи производящих функций, аналогичный методу, использованному для нахождения формулы (1.65), определяющей числа Фибоначчи.

47. Пусть p_n — число всевозможных расположений скобок в произведении $x_0 \dots x_n$ (например, $p_2 = 2$, так как имеется два способа расстановки скобок: $(x_0 x_1) x_2, x_0 (x_1 x_2)$). Доказать, что p_n равно числу Каталана c_n .

48. Доказать, что число способов, которыми можно разбить плоский выпуклый $(n+2)$ -угольник на различные треугольники с помощью $n-1$ диагоналей, не пересекающихся внутри этого $(n+2)$ -угольника, равно числу Каталана c_n .

49. Обозначим через $\pi(x)$ количество простых чисел, не превосходящих x . Доказать, что

$$\pi(n) - \pi(\sqrt{n}) = n - 1 - \sum_{1 \leq i \leq k} \left\lfloor \frac{n}{p_i} \right\rfloor + \sum_{1 \leq i < j \leq k} \left\lfloor \frac{n}{p_i p_j} \right\rfloor - \dots + (-1)^k \left\lfloor \frac{n}{p_1 p_2 \dots p_k} \right\rfloor,$$

где $k = \pi(\sqrt{n})$

50. Обозначим через $\varphi(n)$ количество натуральных чисел, не превосходящих n и взаимно простых с n (т.е. не имеющих общего делителя, большего единицы). Доказать, что

$$\varphi(n) = n \left(1 - \sum_{i=1}^m \frac{n}{q_i} + \sum_{1 \leq i < j \leq m} \frac{n}{q_i q_j} - \dots + (-1)^m \frac{1}{q_1 q_2 \dots q_m} \right) = n \prod_{i=1}^m \left(1 - \frac{1}{q_i} \right),$$

где q_1, \dots, q_m являются различными простыми делителями числа n .

Глава 2

Алгоритмы на графах

2.1 Машинное представление графов

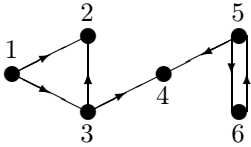
Очевидно, что наиболее понятный и полезный для человека способ представления графа — изображение графа на плоскости в виде точек и соединяющих их линий — будет совершенно бесполезным, если мы захотим решать с помощью ЭВМ задачи, связанные с графами. Выбор соответствующей структуры данных для представления графов имеет принципиальное влияние на эффективность алгоритмов, поэтому мы подробнее остановимся на этой проблеме. Мы покажем несколько различных способов представления и кратко разберем их основные достоинства и недостатки.

Мы будем рассматривать как ориентированные, так и неориентированные графы. Граф мы будем всегда обозначать $G = \langle V, E \rangle$, где V обозначает множество вершин, а E — множество ребер¹, причем $E \subseteq V \times V$ для ориентированного графа и $E \subseteq \{\{x, y\} : x, y \in V \wedge x \neq y\}$ для неориентированного графа. Будем также использовать обозначения $|V| = n$ и $|E| = m$.

В теории графов классическим способом представления графа служит *матрица инцидентий*. Это матрица A с n строками, соответствующими вершинам, и m столбцами, соответствующими ребрам. Для ориентированного графа столбец, соответствующий дуге $\langle x, y \rangle \in E$, содержит -1 в строке, соответствующей вершине x , 1 в строке, соответствующей вершине y , и нули во всех остальных строках (петлю, т.е. дугу вида $\langle x, x \rangle$, удобно представлять иным значением в строке x , например, 2). В случае неориентированного графа столбец, соответ-

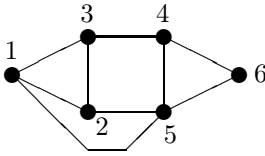
¹В отечественной литературе ребра ориентированного графа принято называть *дугами*. В дальнейшем мы будем употреблять этот термин везде, где речь идет об орграфах, используя термин «ребро» как более общий. — *Прим. перев.*

(a)



$$\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

(б)



$$\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Рис. 2.1: а) ориентированный граф и его матрица инцидентий; б) неориентированный граф и его матрица инцидентий

ствующий ребру $\{x, y\}$, содержит 1 в строках, соответствующих x и y , и нули в остальных строках. Это проиллюстрировано на рис. 2.1. С алгоритмической точки зрения матрица инцидентий является, вероятно, самым худшим способом представления графа, который только можно себе представить. Во-первых, он требует mn ячеек памяти, причем большинство этих ячеек вообще занято нулями. Неудобен также доступ к информации. Ответ на элементарные вопросы типа «существует ли дуга $\langle x, y \rangle$ », «к каким вершинам ведут ребра из x ?» требует в худшем случае перебора всех столбцов матрицы, а следовательно, m шагов.

Лучшим способом представления графа является *матрица смежности*, определяемая как матрица $B = [b_{ij}]$ размера $n \times n$, где $b_{ij} = 1$, если существует ребро, идущее из вершины x в вершину y , и $b_{ij} = 0$ в противном случае. Здесь

(а)	1	2	3	4	5	6	(б)	1	2	3	4	5	6
.	1	2	3	4	5	6	.	1	2	3	4	5	6
1	0	1	1	0	0	0	1	0	1	1	0	1	0
2	0	0	0	0	0	0	2	1	0	1	0	1	0
3	0	1	0	1	0	0	3	1	1	0	1	0	0
4	0	0	0	0	0	0	4	0	0	1	0	1	1
5	0	0	0	1	0	1	5	1	1	0	1	0	1
6	0	0	0	0	1	0	6	0	0	0	1	1	0

Рис. 2.2: Матрицы смежности для графов на рис. 2.1.

мы подразумеваем, что ребро $\{x, y\}$ неориентированного графа идет как от x к y , так и от y к x , так что матрица смежности такого графа всегда является симметричной. Это проиллюстрировано на рис. 2.2.

Основным преимуществом матрицы смежности является тот факт, что за один шаг можно получить ответ на вопрос «существует ли ребро из x в y ?». Недостатком же является тот факт, что независимо от числа ребер объем занятой памяти составляет n^2 . На практике это неудобство можно иногда уменьшить, храня целую строку (столбец) матрицы в одном машинном слове — это возможно для малых n .

В качестве еще одного аргумента против использования матрицы смежности приведем теорему о числе шагов, которое должен выполнить алгоритм, проверяющий на основе матрицы смежности некоторое свойство графа. Пусть P — некоторое свойство графа ($P(G) = 0$ или $P(G) = 1$ в зависимости от того, обладает или не обладает G нашим свойством). Предположим, что свойство P удовлетворяет следующим трем условиям:

- (а) $P(G) = P(G')$, если графы G и G' изоморфны;
- (б) $P(G) = 0$ для произвольного пустого графа $\langle V, \emptyset \rangle$ и $P(0) = 1$ для произвольного полного графа $\langle V, P_2(V) \rangle$ с достаточно большим числом вершин;
- (в) добавление ребра не нарушает свойства P , т.е. $P(G) \leq P(G')$ для произвольных графов $G = \langle V, E \rangle$ и $G' = \langle V, E' \rangle$, таких что $E \subseteq E'$.

Примером такого свойства P является существование цикла (в графе, имеющем по крайней мере три вершины).

Теорема 2.1. *Если P — свойство графа, отвечающее условиям (а), (б), (в), то каждый алгоритм, проверяющий свойство P (т.е. вычисляющий значение $P(G)$ для данного графа G) на основе матрицы смежности, выполняет в худшем случае $\Omega(n^2)$ шагов, где n — число вершин графа.*

(а)								
1	1	3	3	5	5	4		
2	3	2	4	4	6	5		

(б)								
1	1	1	2	2	3	4	4	5
2	3	5	3	5	4	5	6	6

Рис. 2.3: Списки рёбер, соответствующие графам на рис. 2.1.

Эта теорема справедлива также для ориентированных графов и для свойств, не зависящих от петель графа, т.е. отвечающих дополнительному условию

- (г) $P(G) = P(G')$ для произвольных ориентированных графов $G = \langle V, E \rangle$, $G' = \langle V, E \cup \langle v, v \rangle \rangle$, $v \in V$.

Доказательство теоремы читатель может найти в работах [5] и [59].

Более экономным в отношении памяти (особенно в случае неплотных графов, когда m гораздо меньше n^2) является метод представления графа с помощью списка пар, соответствующих его ребрам. Пара $\langle x, y \rangle$ соответствует дуге $\langle x, y \rangle$, если граф ориентированный, и ребру $\{x, y\}$ в случае неориентированного графа (рис. 2.3). Очевидно, что объем памяти в этом случае составляет $2m$. Неудобством является большое число шагов — порядка m в худшем случае, — необходимое для получения множества вершин, к которым ведут ребра из данной вершины.

Ситуацию можно значительно улучшить, упорядочив множество пар лексикографически и применяя двоичный поиск, но лучшим решением во многих случаях оказывается структура данных, которую мы будем называть списками инцидентности. Она содержит для каждой вершины $v \in V$ список вершин u , таких что $v \rightarrow u$ (или $v - u$ в случае неориентированного графа). Точнее, каждый элемент такого списка является записью r , содержащей вершину r . строка и указатель r .след на следующую запись в списке (r .след = **nil** для последней записи в списке). Начало каждого списка хранится в таблице НАЧАЛО, точнее, НАЧАЛО[v] является указателем на начало списка, содержащего вершины из множества $\{u : v \rightarrow u\}$ ($\{u : v - u\}$ для неориентированного графа). Весь такой список обычно неформально будем обозначать ЗАПИСЬ[u], а цикл, выполняющий определенную операцию для каждого элемента u из этого списка в произвольной, но четко установленной последовательности, соответствующей очередности элементов в списке, будем записывать «**for** $u \in$ ЗАПИСЬ[v] **do** ...».

Отметим, что для неориентированных графов каждое ребро $\{u, v\}$ представлено дважды: через вершину v в списке ЗАПИСЬ[u] и через вершину u в списке

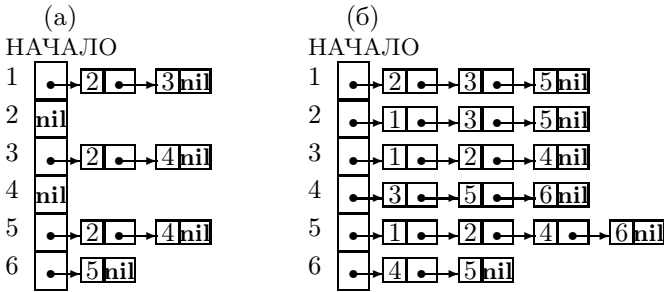


Рис. 2.4: Списки инцидентности ЗАПИСЬ[v], $v \in V$, соответствующие графам на рис. 2.1.

ЗАПИСЬ[v]. Во многих алгоритмах структура графа динамически модифицируется добавлением и удалением ребер. В таких случаях полагаем, что в наших списках инцидентности элемент списка ЗАПИСЬ[u], содержащий вершину v , снабжен указателем на элемент списка ЗАПИСЬ[u], содержащий вершину u , и что каждый элемент списка содержит указатель не только к следующему элементу, но и к предыдущему. Тогда удаляя некоторый элемент из списка, мы можем легко за число шагов, ограниченное константой, удалить другой элемент, представляющий то же самое ребро, не просматривая список, содержащий этот элемент (см. задачу 2.3).

Аналогичным способом определяем для каждой вершины v неориентированного графа список ПРЕДШ[v], содержащий вершины из множества $\{u : u \rightarrow v\}$. Число ячеек памяти, необходимое для представления графа с помощью списков инцидентности, будет, очевидно, иметь порядок $m+n$. На рис. 2.4 представлены списки инцидентности, соответствующие графам на рис. 2.1.

2.2 Поиск в глубину в графе

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой что каждая вершина просматривается в точности один раз. Поэтому важной задачей является нахождение хороших методов поиска в графе. Вообще говоря, метод поиска «хорош», если

- (а) он позволяет алгоритму решения интересующей нас задачи легко «погрузиться» в этот метод и
- (б) каждое ребро графа анализируется не более одного раза (или, что существенно не меняет ситуации, число раз, ограниченное константой).

Опишем теперь такой метод поиска в неориентированном графе, который стал одной из основных методик проектирования графовых алгоритмов. Этот метод называется *поиском в глубину* (англ. depth first search [66]) по причинам, которые, как мы надеемся, вскоре станут ясными.

Общая идея этого метода состоит в следующем. Мы начинаем поиск с некоторой фиксированной вершины v_0 . затем выбираем произвольную вершину u , смежную с v_0 , и повторяем наш процесс от u . В общем случае предположим, что мы находимся в вершине v . Если существует *новая* (еще непросмотренная) вершина u , $u - v$, то мы рассматриваем эту вершину (она перестает быть новой), и, начиная с нее, продолжаем поиск. Если же не существует ни одной новой вершины, смежной с v , то мы говорим, что вершина v *использована*, возвращаемся в вершину, из которой мы попали в v , и продолжаем процесс (если $v = v_0$, то поиск закончен). Другими словами, поиск в глубину из вершины v основывается на поиске в глубину из всех новых вершин, смежных с v . Это можно легко записать с помощью следующей рекурсивной процедуры:

```

1 procedure  $WG(v)$  (*поиск в глубину из вершины  $v$ ,
переменные НОВЫЙ, ЗАПИСЬ глобальные *)
2 begin рассмотреть  $v$ ; НОВЫЙ[ $v$ ]:=ложь;
3   for  $u \in$  ЗАПИСЬ[ $v$ ] do
4     if НОВЫЙ[ $u$ ] then  $WG(u)$ 
5 end (*вершина  $v$  использована*)

```

Поиск в глубину в произвольном, необязательно связном графе проводится согласно следующему алгоритму:

```

1. begin
2.   for  $v \in V$  do НОВЫЙ[ $v$ ]:=истина;
3.   for  $v \in V$  do
4.     if НОВЫЙ[ $u$ ] then  $WG(v)$ 
5. end

```

Покажем теперь, что этот алгоритм просматривает каждую вершину в точности один раз и его сложность порядка $O(m + n)$. Отметим сначала, что вызов $WG(v)$ влечет за собой просмотр всех вершин связной компоненты графа, содержащей v (если НОВЫЙ[u]=истина для каждой вершины u этой компоненты). Это непосредственно следует из структуры процедуры WG : после посещения вершины (строка 2) следует вызов процедуры WG для всех ее новых соседей. Отметим также, что каждая вершина просматривается не более одного раза, так как просматриваться может только вершина u , для которой НОВЫЙ[v]=истина, сразу же после посещения этой вершины исполняется команда НОВЫЙ[u]:=ложь (строка 2).

Наш алгоритм начинает поиск поочередно от каждой еще не просмотренной вершины, следовательно, просматриваются все вершины графа (необязательно связного).

Чтобы оценить вычислительную сложность алгоритма, отметим сначала, что число шагов в обоих циклах (строки 2 и 3) порядка n , не считая шагов, выполнение которых инициировано вызовом процедуры WG . Эта процедура выполняется не более n раз во втором цикле сразу после посещения каждой из вершин для каждого из ее новых соседей, итого суммарно $O(n + m)$ раз. Полное число шагов, выполняемых циклом в строке 3 процедуры WG (не считая шагов, выполняемых $WG(n)$), для всех вызовов этой процедуры будет порядка m , где m - число ребер. Это дает общую сложность алгоритма $O(m + n)$.

Отметим, что алгоритм поиска в глубину в произвольном графе можно легко модифицировать так, чтобы он вычислял связные компоненты этого графа. Зафиксируем этот факт:

Следствие 2.2. *Связные компоненты произвольного графа, представленного списками инцидентности, можно вычислить за время $O(n + m)$.*

В связи с тем, что поиск в глубину играет важную роль в проектировании алгоритмов на графах, представим также нерекурсивную версию процедуры WG . Рекурсия удаляется стандартным способом при помощи стека (см., например, [1]). Каждая просмотренная вершина помещается в стек и удаляется из стека после использования.

1. **procedure** $WG1(v)$; (* поиск в глубину в графе, начиная с вершины v — нерекурсивная версия процедуры WG ; предполагаем, что в начале процесса поиска $P[u]$ = указатель на первую запись списка ЗАПИСЬ[u] для каждой вершины u ; массивы P , НОВЫЙ — глобальные*)
2. **begin** СТЕК := \emptyset ; СТЕК $\leftarrow v$; рассмотреть v ; НОВЫЙ[v]:=ложь;
3. **while** СТЕК $\neq \emptyset$ **do**
4. **begin** $t := top(СТЕК)$; (* t — верхний элемент стека*)
5. (*найти первую новую вершину в списке ЗАПИСЬ*)
6. **if** $P[t]=nil$ **then** $b:=ложь$
7. **else** $b:=not$ НОВЫЙ[$P(t) \uparrow$.строка];
8. **while** b **do**
9. **begin** $P[t] := P[t].след$;
10. **if** $p[t]=nil$ **then** $b:=ложь$
11. **else** $b:= not$ НОВЫЙ[$P[t] \uparrow$.строка]
12. **end**;
13. **if** $p[t] = nil$ **then** (*найдена новая вершина *)
14. **begin** $t := P[t] \uparrow$.строка; СТЕК $\leftarrow t$;
15. рассмотреть t ; НОВЫЙ[t]:= ложь
16. **end**

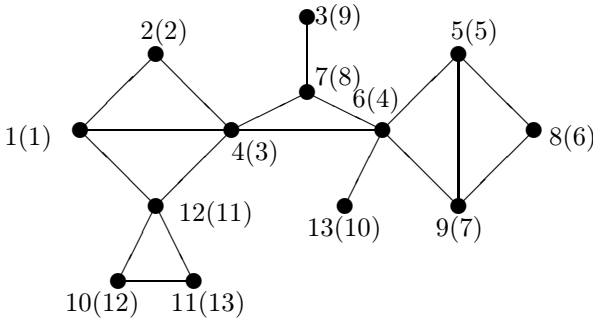


Рис. 2.5: Нумерация вершин графа (в скобках), соответствующая очередности, в которой они просматриваются в процессе поиска в глубину.

```

16.         else (*вершина t использована*)
17.           t ← СТЕК (*удалить верхний элемент стека*)
18.     end
19. end

```

Корректность этой процедуры можно доказать путем незначительной модификации анализа процедуры WG . Мы оставляем это читателю. На рис. 2.5 показан граф, вершины которого перенумерованы в соответствии с тем порядком, в котором они просматриваются в процессе поиска в глубину (мы отождествляем эти вершины с числами $1, \dots, 10$ и полагаем, что в списке $\text{ЗАПИСЬ}[u]$ вершины упорядочены по возрастанию).

Методика поиска в глубину очевидным образом переносится на ориентированные графы. Нетрудно проверить, что в случае ориентированного графа результатом вызова процедуры $WG(u)$, а также $WG1(v)$ будет посещение за $O(n + m)$ шагов всех вершин u , таких что существует путь из v в u (см. задачу 2.11).

2.3 Поиск в ширину в графе

Теперь рассмотрим несколько иной метод поиска в графе, называемый *поиском в ширину* (англ.: breadth first search). Прежде чем описать его, отметим, что при поиске в глубину чем позднее будет посещена вершина, тем раньше она будет использована — точнее, так будет при допущении, что вторая вершина посещается перед использованием первой. Это прямое следствие того факта,

что просмотренные, но еще не использованные вершины скапливаются в стеке. Поиск в ширину, грубо говоря, основывается на замене стека очередью. После такой модификации чем раньше посещается вершина (помещается в очередь), тем раньше она используется (удаляется из очереди). Использование вершины происходит с помощью просмотра сразу всех еще непросмотренных соседей этой вершины. Вся процедура представлена ниже

```

1. procedure  $WS(v)$ ; (* поиск в ширину в графе с началом в вершине  $v$ ;
   переменные НОВЫЙ, ЗАПИСЬ — глобальные*)
2. begin
3.   ОЧЕРЕДЬ :=  $\emptyset$ ; ОЧЕРЕДЬ  $\leftarrow v$ ; НОВЫЙ[ $v$ ]:=ложь
4.   while ОЧЕРЕДЬ  $\neq \emptyset$  do
5.     begin  $p \leftarrow$  ОЧЕРЕДЬ; посетить  $p$ ;
6.     for  $u \in$  ЗАПИСЬ[ $p$ ] do
7.       if НОВЫЙ[ $u$ ] then
8.         begin ОЧЕРЕДЬ  $\leftarrow u$ ; НОВЫЙ[ $u$ ]:=ложь
9.         end
10.    end
11. end

```

Способом, аналогичным тому, какой был применен для поиска в глубину, можно легко показать, что вызов процедуры $WS(u)$ приводит к посещению всех вершин связной компоненты графа, содержащей вершину v , причем каждая вершина просматривается в точности один раз. Вычислительная сложность поиска в ширину также имеет порядок $m + n$, так как каждая вершина помещается в очередь и удаляется из очереди в точности один раз, а число итераций цикла δ , очевидно, будет иметь порядок числа ребер графа.

Оба вида поиска в графе — в глубину и в ширину могут быть использованы для нахождения пути между фиксированными вершинами v и u . Достаточно начать поиск в графе с вершины v и вести его до момента посещения вершины u . Преимуществом поиска в глубину является тот факт, что в момент посещения вершины u стек содержит последовательность вершин, определяющую путь из v в u . Это становится очевидным, если отметить, что каждая вершина, помещаемая в стек, является соседом верхнего элемента в стеке. Однако недостатком поиска в глубину является то, что полученный таким образом путь в общем случае не будет кратчайшим путем из v в u .

От этого недостатка свободен метод нахождения пути, основанный на поиске в ширину. Модифицируем процедуру WS , заменяя строки 7–9 на

```

if НОВЫЙ[ $u$ ] then
  begin ОЧЕРЕДЬ  $\leftarrow u$ ; НОВЫЙ[ $u$ ]:=ложь; ПРЕДЫДУЩИЙ[ $u$ ] :=  $p$ 
  end

```

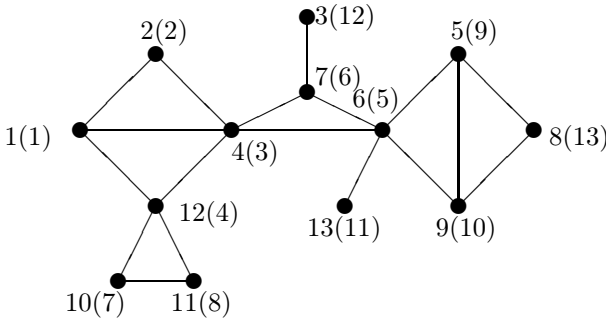


Рис. 2.6: Нумерация вершин графа на рис. 2.5 (в скобках), соответствующая очередности, в которой они просматриваются в процессе поиска в ширину.

По окончании работы модифицированной таким образом процедуры таблица ПРЕДЫДУЩИЙ содержит для каждой просмотренной вершины u вершину ПРЕДЫДУЩИЙ[u], из которой мы попали в u . Отметим, что кратчайший путь из u в v обозначается последовательностью вершин $u = u_1, u_2, \dots, u_k = v$, где $u_{i+1} = \text{ПРЕДЫДУЩИЙ}[u_i]$ для $1 \leq i < k$ и k является первым индексом i , для которого $u_i = v$. Действительно, в очереди помещены сначала вершины, находящиеся на расстоянии 0 от v (т.е. сама вершина v), затем поочередно все новые вершины, достижимые из v , т.е. вершины, находящиеся на расстоянии 1 от v , и т.д. Под расстоянием здесь мы понимаем длину кратчайшего пути. Предположим теперь, что мы уже рассмотрели все вершины, находящиеся на расстоянии, не превосходящем r , от v , что очередь содержит все вершины, находящиеся от v на расстоянии r , и только эти вершины и что таблица ПРЕДЫДУЩИЙ правильно определяет кратчайший путь от каждой, уже просмотренной вершины до u способом, описанным выше. Используя каждую вершину p , находящуюся в очереди, наша процедура просматривает некоторые новые вершины, и каждая такая новая вершина u находится, очевидно, на расстоянии $r + 1$ от v , причем, определяя ПРЕДЫДУЩИЙ[u] := p , мы продлеваем кратчайший путь от p до v до кратчайшего пути от u до v . После использования всех вершин из очереди, находящихся на расстоянии r от v , она (очередь), очевидно, содержит множество вершин, находящихся на расстоянии $r + 1$ от v , и легко заметить, что условие индукции выполняется и для расстояния $r + 1$.

На рис. 2.6 представлен граф, вершины которого занумерованы согласно очередности, в которой они посещаются в процессе поиска в глубину.

Как и в случае поиска в глубину, процедуру WS можно использовать без всяких модификаций и тогда, когда списки инцидентности $ЗАПИСЬ[v]$, $v \in V$, определяют некоторый ориентированный граф. Очевидно, что тогда посещаются только те вершины, до которых существует путь от вершины, с которой мы начинаем поиск (см. задачу 2.12).

2.4 Стягивающие деревья (каркасы)

Деревом называется произвольный неориентированный связный граф без циклов. Для произвольного связного неориентированного графа $G = \langle V, E \rangle$ каждое дерево $\langle V, T \rangle$, где $T \subseteq E$, будем называть *стягивающим деревом*² графа G . Ребра такого дерева будем называть *ветвями*, а все остальные ребра графа будем называть *хордами* (очевидно, что о ветвях и хордах в графе мы можем говорить только в контексте фиксированного стягивающего дерева).

Отметим, что каждое дерево с n вершинами имеет $n - 1$ ребер. Этот факт можно просто доказать, применяя индукцию относительно n . Очевидно, что это справедливо для $n = 1$. Если $n > 1$, то отметим сначала, что в каждом дереве с n вершинами существует «висячая» вершина, т.е. вершина степени 1. Действительно, рассмотрим в таком дереве произвольный путь $v_1 - v_2 - \dots - v_k$ с максимальной длиной ($v_i \neq v_j$ для $i \neq j$). Вершины v_i, v_k «висячие», так как ни от одной из них нельзя провести второго ребра ни к одной из вершин v_1, \dots, v_n (дерево не содержит циклов), ни к любой другой вершине (путь максимальный). Удаляя из нашего дерева вершину v_k и ребро $\{v_{k-1}, v_k\}$ (либо вершину v_1 и ребро $\{v_1, v_2\}$), получаем, очевидно, дерево с $n - 1$ вершинами, которое в силу индуктивного предположения имеет $n - 2$ ребер. Следовательно, наше первоначальное дерево имело $n - 2 + 1 = n - 1$ ребер.

Процедуры поиска в глубину и в ширину можно простым способом использовать для нахождения стягивающих деревьев. В обоих случаях достижение новой вершины u из вершины v вызывает включение в дерево ребра $\{u, v\}$.

Алгоритм 2.3. (Нахождение стягивающего дерева связного графа методом поиска в глубину.)

Данные: Связный граф $G = \langle V, E \rangle$, заданный списками инцидентности $ЗАПИСЬ[v]$, $v \in V$.

Результаты: Стягивающее дерево $\langle V, T \rangle$ графа G .

1 **procedure** $WGD(v)$;

(*поиск в глубину, соединенный с нахождением ребра дерева;
переменные **НОВЫЙ**, **ЗАПИСЬ**, T — глобальные*)

²В литературе по теории графов широко распространены также термины *каркас* и *остов* — *Прим. перев.*

```

2 begin НОВЫЙ[v]:=ложь;
3   for  $u \in \text{ЗАПИСЬ}[v]$  do
4     if НОВЫЙ[u] then (*{v, u} — новая ветвь*)
5       begin  $T := T \cup \{v, u\}$ ;  $WGD(u)$ 
6     end
7 end; (* WGD*)
8 begin (* главная программа*)
9   for  $u \in V$  do НОВЫЙ[u]:=истина; (*инициализация*)
10   $T := \emptyset$ ; (*T=множество найденных к этому моменту ветвей*)
11   $WGD(r)$  (*r — произвольная вершина графа*)
12 end

```

Для доказательства того, что алгоритм 2.3 корректно строит стягивающее дерево произвольного связного графа, достаточно отметить следующие три факта. Во-первых, в момент добавления к множеству T новой ветви $\{v, u\}$ (строка 5) в $\langle V, E \rangle$ существует путь из r в v (этот факт мы доказываем по индукции). Таким образом, алгоритм строит связный граф. Во-вторых, каждая новая ветвь $\{u, v\}$, добавляемая к множеству T , соединяет уже рассмотренную вершину v (т.е. $\text{НОВЫЙ}[v]=\text{ложь}$) с новой вершиной u . Отсюда следует, что построенный граф $\langle V, T \rangle$ не содержит циклов. Действительно, последняя ветвь, «закрывающая» цикл, должна была бы соединить две уже рассмотренные вершины. И наконец, в-третьих, из свойства поиска в глубину следует, что процедура WGD просматривает все вершины связного графа. Следовательно, граф $\langle V, T \rangle$, построенный нашим алгоритмом, есть стягивающее дерево графа G . Вычислительная сложность алгоритма есть, очевидно, $O(n + m)$, т.е., того же порядка, что и поиск в глубину. Пример стягивающего дерева, построенного алгоритмом 2.3, приведен на рис. 2.7, а. Каждое стягивающее дерево, построенное указанным алгоритмом, имеет любопытное свойство, которое мы сейчас опишем. Вершину r , с которой мы начинаем поиск в графе, назовем *корнем* стягивающего дерева $\langle V, T \rangle$. Очевидно, что в дереве $\langle V, T \rangle$ существует в точности один путь от произвольной вершины к корню (см. задачу 10).

Для двух различных вершин v и u дерева $\langle V, T \rangle$ будем говорить, что u является *потомком* вершины v , если v лежит на пути (в дереве $\langle V, T \rangle$) из u в корень. Если при этом имеет место $v - u$, то будем говорить, что u — *сын* вершины v , а v — *отец* вершины u .

Теорема 2.4. Пусть $\langle V, T \rangle$ — стягивающее дерево связного графа $G = \langle V, E \rangle$, построенное алгоритмом 2.3, и пусть $\{u, v\} \in E$. Тогда либо u — потомок v , либо v — потомок u .

Доказательство. Предположим без ограничения общности, что вершина v будет просмотрена раньше, чем u . Рассмотрим процесс поиска в глубину, начиная с вершины v . Очевидно, что по окончании его должно быть $\text{НОВЫЙ}[u]=\text{ложь}$,

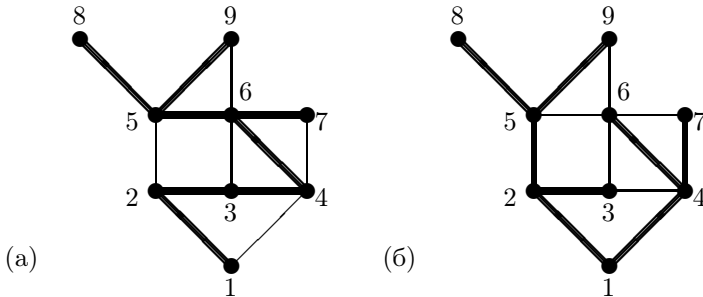


Рис. 2.7: Стягивающие деревья, построенные с помощью алгоритма 2.3 (а) и алгоритма 2.5 (б).

ибо $v - u$. Но это означает, что ребра, добавленные в множество T в течение этого процесса, содержат путь из v в u , откуда следует, что v лежит на пути из u в корень.

Подобным же способом можно построить стягивающее дерево, используя метод поиска в ширину.

Алгоритм 2.5. (Нахождение стягивающего дерева связного графа методом поиска в ширину.)

Данные: Связный граф $G = \langle V, E \rangle$, представленный списками инцидентности ЗАПИСЬ[v], $v \in V$.

Результаты: Стягивающее дерево $\langle V, T \rangle$ графа G .

```

1 begin
2   for  $u \in V$  do НОВЫЙ[ $u$ ]:=истина; (*инициализация*)
3    $T := \emptyset$ ; (* $T$  — множество найденных к этому моменту ветвей*)
4   ОЧЕРЕДЬ :=  $\emptyset$ ; ОЧЕРЕДЬ  $\leftarrow r$ ;
5   НОВЫЙ[ $r$ ]:=ложь; (*корень стягивающего дерева*)
6   while ОЧЕРЕДЬ  $\neq \emptyset$  do
7     begin  $v \leftarrow$  ОЧЕРЕДЬ;
8       for  $u \in$  ЗАПИСЬ[ $v$ ] do
9         if НОВЫЙ[ $u$ ] then (* $\{v, u\}$  = новая ветвь*)
10          begin ОЧЕРЕДЬ  $\leftarrow u$ ; НОВЫЙ[ $u$ ]:=ложь;
11               $T := T \cup \{v, u\}$ 
12          end
13   end

```

Способом, аналогичным использованному для алгоритма 2.3, можно доказать, что данный алгоритм корректно строит стягивающее дерево для произвольного связного графа за $O(m + n)$ шагов.

На рис. 2.7, б дан пример стягивающего дерева, построенного алгоритмом 2.5.

Рассуждения разд. 2.3 приводят непосредственно к следующей теореме:

Теорема 2.6. Пусть $\langle V, T \rangle$ — стягивающее дерево связного графа $G = \langle V, E \rangle$, построенное при помощи алгоритма 2.5. Тогда путь в $\langle V, T \rangle$ из произвольной вершины u до корня r является кратчайшим путем из u в r в графе G .

В гл. 3 особо обсуждается более общая задача отыскания кратчайших путей в графе, ребрам которого приписаны «длины», не обязательно равные единице.

Рассуждения данного раздела легко переносятся на произвольные графы, не обязательно связные. Максимальный подграф без циклов произвольного графа G называется *стягивающим лесом* графа G . Очевидно, что стягивающий лес графа с k компонентами связности определяется через стягивающие деревья этих компонент и, следовательно, содержит $n - k - 1$ ребер.

2.5 Отыскание фундаментального множества циклов в графе

Тесно связана с задачей нахождения стягивающего дерева задача построения фундаментального множества циклов. Если к стягивающему дереву $\langle V, T \rangle$ графа $G = \langle V, E \rangle$ мы добавим произвольную хорду $e \in E \setminus T$, то нетрудно отметить, что возникший при этом подграф $\langle V, T \cup \{e\} \rangle$ содержит в точности один цикл³, который мы будем обозначать через C_e . Очевидно, что C_e содержит ребро e . Множество $\mathcal{C} = \{C_e : e \in E \setminus T\}$ будем называть *фундаментальным множеством циклов* графа G (относительно стягивающего дерева $\langle V, T \rangle$). Название «фундаментальный» связано с тем фактом, что, как мы докажем позднее, каждый цикл графа G можно некоторым естественным способом получить из циклов множества \mathcal{C} .

Введем для произвольных множеств A и B операцию

$$A \oplus B = (A \cup B) \setminus (A \cap B).$$

Множество $A \oplus B$ будем называть *симметрической разностью* множеств A и B .

Отметим, что симметрическая разность множеств A_1, \dots, A_k содержит независимо от расстановки скобок в точности те элементы, которые появляются в

³В этом разделе под циклом мы всегда будем понимать элементарный цикл

нечетном числе множеств A_i . Действительно, это нетрудно доказать индукцией по k . Для $k = 1$ и $k = 2$ это, очевидно, так. Для $k > 2$ наша симметрическая разность имеет вид $A \oplus B$, где A является симметрической разностью множеств A_1, \dots, A_p и B — симметрической разностью множеств A_{p+1}, \dots, A_{p+q} , где $p, q < k$ и $p+q = k$. Множество $A \oplus B$ содержит в точности те элементы, которые появляются только в одном из множеств A или B . Пользуясь индуктивным предположением, сделаем вывод, что $A \oplus B$ является множеством тех элементов, которые появляются в нечетном числе множеств из A_1, \dots, A_p и в четном числе из A_{p+1}, \dots, A_{p+q} , или же в нечетном числе множеств из A_{p+1}, \dots, A_{p+q} и в четном числе множеств из A_1, \dots, A_p . А это в точности множество тех элементов, которые появляются в нечетном числе множеств из A_1, \dots, A_k .

Из доказанного выше свойства следует, что мы можем опустить скобки в симметрической разности произвольного числа множеств и писать $A_1 \oplus A_2 \oplus \dots \oplus A_k$. Множество C ребер графа называется *псевдоциклом*, если каждая вершина графа $\langle V, C \rangle$ имеет четную степень. Примером псевдоцикла является пустое множество и произвольный цикл графа.

Лемма 2.7. *Симметрическая разность произвольного числа псевдоциклов является псевдоциклом.*

Доказательство. Очевидно, что достаточно рассмотреть случай двух псевдоциклов C_1 и C_2 . Обозначим для произвольной вершины v через $S_1(v)$ и $S_2(v)$ множество ребер соответственно из C_1 и C_2 , инцидентных с v . Множества $S_1(v)$ и $S_2(v)$ имеют четную мощность, четной является также мощность множества ребер из $C_1 \oplus C_2$, инцидентных с v , так как $|S_1(v) \oplus S_2(v)| = |S_1(v)| + |S_2(v)| - 2|S_1(v) \cap S_2(v)|$

Теперь мы можем доказать теорему о фундаментальном множестве циклов.

Теорема 2.8. *Пусть $G = \langle V, E \rangle$ — связный неориентированный граф, а $\langle V, T \rangle$ — его стягивающее дерево. Произвольный цикл графа G можно однозначно представить как симметрическую разность некоторого числа фундаментальных циклов. В общем случае произвольный псевдоцикл C графа G можно однозначно выразить как*

$$C = \bigoplus_{e \in C \setminus T} C_e. \tag{2.1}$$

Доказательство. Симметрическая разность $\bigoplus_{e \in C \setminus T} C_e$, являющаяся псевдоциклом в силу предыдущей леммы, состоит из множества хорд $C \setminus T$ и некоторых ветвей. Это вытекает из того факта, что каждая хорда $e' \in C \setminus T$ принадлежит в точности к одному фундаментальному циклу, а именно к C_e . Множество

$$C \oplus \bigoplus_{e \in C \setminus T} C_e. \tag{2.2}$$

является также в силу предыдущей леммы псевдоциклом, причем он может содержать только ветви. Однако ни один непустой псевдоцикл не может содержаться в T , так как каждый непустой подграф без циклов содержит вершину степени 1 («висячую»). Отсюда следует, что множество (2.2) пустое, что эквивалентно равенству (2.1). Однозначность представления (2.1), легко следует из того, что цикл C_e является единственным фундаментальным циклом, содержащим хорду e ($e \in E \setminus T$)

Теорема, которую мы доказали, имеет очень простую интерпретацию в терминах линейных пространств. Пусть $E = \{e_1, \dots, e_m\}$ и каждому псевдоциклу C поставлен в соответствие вектор $\langle a_1, \dots, a_n \rangle$, где

$$a_i = \begin{cases} 1, & \text{если } e_i \in C, \\ 0 & \text{в противном случае} \end{cases}$$

Сумме таких векторов в предположении, что суммы координат берутся по модулю 2, соответствует симметрическая разность некоторых псевдоциклов. Лемма 2.7 утверждает, что векторы, поставленные в соответствие псевдоциклам, образуют подпространство m -мерного линейного пространства над двухэлементным полем (состоящим из 0 и 1 с операциями сложения по модулю 2 и умножения). Отметим, что произвольная линейная комбинация в пространстве над двухэлементным полем соответствует симметрической разности, так как единственным ненулевым элементом (коэффициентом) может быть только 1. В свою очередь теорема 2.8 говорит, что фундаментальные циклы определяют базис нашего подпространства.

Нахождение фундаментального множества циклов имеет существенное значение при анализе электрических цепей. Точнее говоря, каждому фундаментальному циклу в графе, соответствующему данной электрической цепи, мы можем сопоставить уравнение, определяющее закон Кирхгофа для напряжений (см., например, [61]): сумма падения напряжений вдоль цикла равна нулю. Тогда ни одно из этих уравнений не зависит от остальных, от них же зависит произвольное уравнение, определяющее закон Кирхгофа для произвольного цикла графа.

Опишем теперь простой алгоритм нахождения множества фундаментальных циклов. Этот алгоритм основывается на поиске в глубину и имеет структуру, аналогичную рекурсивному алгоритму нахождения стягивающего дерева (алгоритм 2.3). Каждая новая вершина, встречающаяся в процессе поиска, помещается в стек, представленный таблицей СТЕК, и удаляется из стека после использования. Очевидно, что стек всегда содержит последовательность вершин с рассматриваемой в данный момент вершины v до корня. Поэтому же если анализируемое нами ребро $\{v, u\}$ замыкает цикл (т.е. $WGN[v] > WGN[u] > 0$ и u не находится непосредственно под верхним элементом стека), то вершина u —

также в силу теоремы 2.4 — находится в стеке и цикл, замыкаемый ребром $\{v, u\}$ представлен верхней группой элементов стека, начиная с v и кончая вершиной u .

Алгоритм 2.9. (Нахождение множества элементарных циклов графа.)

Данные: Граф $G = \langle V, E \rangle$, представленный списками инцидентности ЗАПИСЬ $[v]$, $v \in V$.

Результаты: Множество элементарных циклов графа G .

```

1. procedure ЦИКЛ( $u$ );
   (* нахождение фундаментального множества циклов
   для компоненты связности, содержащей вершину  $u$ ,
   переменные  $d$ ,  $num$ , СТЕК, ЗАПИСЬ,  $WGN$  — глобальные*)
2. begin  $d := d + 1$ ; СТЕК $[d] := v$ ;  $num := num + 1$ ;  $WGN[v] := num$ ;
3.   for  $u \in$  ЗАПИСЬ $[v]$  do
4.     if  $WGN[u] = 0$  then ЦИКЛ( $u$ )
5.     else if ( $u \neq$  СТЕК $[d - 1]$ ) and ( $WGN[v] > WGN[u]$ ) then
       (*  $\{v, u\}$  замыкает новый цикл *)
6.       выписать цикл с вершинами
7.       СТЕК $[d]$ , СТЕК $[d - 1], \dots$ , СТЕК $[c]$ , где СТЕК $[c] = u$ ;
8.        $d := d - 1$  (*использованная вершина  $v$  удаляется из стека *)
9. end; (*ЦИКЛ*)
10. begin (* главная программа*)
11.   for  $v \in V$  do  $WGN[v] := 0$ ;  $num := 0$ ; (*инициализация*)
12.      $d := 0$ ; СТЕК $[0] := 0$ ; (* $d$ =число элементов в стеке*)
13.     for  $v \in V$  do
14.       if  $WGN[v] = 0$  then ЦИКЛ( $u$ )
15. end

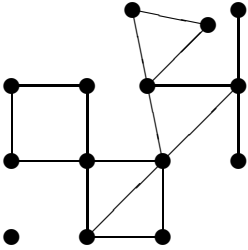
```

Оценим теперь вычислительную сложность этого алгоритма. Отметим сначала, что общее число шагов, не считая выписывания циклов (строки 6, 7) — как и во всех алгоритмах, основанных на поиске в глубину, — имеет порядок $O(n + m)$. К этому следует прибавить суммарную длину всех циклов. Эта длина не превосходит $(m - n + 1)n$, что дает общую сложность алгоритма $O(nm + n)$ (или $O(nm)$, если отбросить вырожденный случай $m = 0$).

2.6 Нахождение компонент двусвязности

Вершину a неориентированного графа $G = \langle V, E \rangle$ будем называть *точкой сочленения*, если удаление этой вершины и всех инцидентных ей ребер ведет к увеличению числа компонент связности графа. Равнозначное утверждение состоит в том, что a является точкой сочленения, если существуют вершины v

(а)



б)

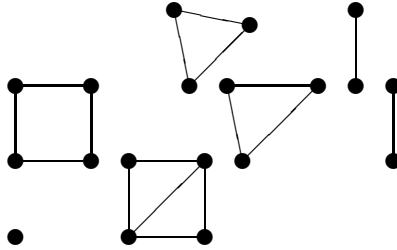


Рис. 2.8: а) Граф с точками сочленения; б) Блоки этого графа

и u , отличные от a , такие, что каждый путь из u в v , а мы предполагаем, что существует по крайней мере один такой путь, проходит через вершину a . Неориентированный граф называется *двусвязным*, если он связный и не содержит точек сочленения. Произвольный максимальный двусвязный подграф графа G называется *компонентой двусвязности* или *блоком* этого графа.

Двусвязность графа — очень желательный признак для некоторых приложений. Представим себе, что вершины графа изображают узлы некоторой информационной сети, а ребра соответствуют линиям передачи. Если наш граф двусвязный, то выход из строя отдельного узла w никогда не приведет к потере соединения между любыми двумя узлами, отличными от w . Знание блоков графа также очень важно, если принять во внимание то, что многие графовые задачи, такие как нахождение всех элементарных циклов или установление факта планарности графа (граф называется планарным, если его можно так начертить на плоскости, чтобы никакие два ребра не пересекались), приводят естественным путем к аналогичным задачам для блоков данного графа.

Отметим, что если $\langle V_1, B_1 \rangle, \langle V_2, B_2 \rangle$ — два разных блока графа G , то $V_1 \cap V_2 = \emptyset$ или $V_1 \cap V_2 = \{a\}$, где a — точка сочленения графа G . Действительно, рассмотрим подграф $\langle V_1 \cup V_2, B_1 \cup B_2 \rangle$. Если было бы $|V_1 \cap V_2| \geq 2$, этот подграф был бы двусвязным вопреки предположению о максимальности $\langle V_1, B_1 \rangle$ и $\langle V_2, B_2 \rangle$. Невозможен также случай $V_1 \cap V_2 = \{a\}$, где a не является точкой сочленения графа G . Ибо тогда в G существует путь между вершинами $u \in V_1$, $u \neq a$ и $v \in V_2$, $v \neq a$, не включающий вершину a . Нетрудно отметить, что подграф,

возникающий из $\langle V_1 \cup V_2, B_1 \cup B_2 \rangle$ добавлением вершин и ребер этого пути, будет двусвязным, что противоречит предположению о максимальности $\langle V_1, B_1 \rangle$ и $\langle V_2, B_2 \rangle$. На рис. 2.8 дан пример графа, имеющего точки сочленения и блоки.

Нахождение точек сочленения и блоков графа является классической задачей, которую можно эффективно решить погружением в процедуру поиска в глубину (см. Тарьян [66]). Прежде чем представить частный алгоритм, сделаем предварительно несколько замечаний.

Теорема 2.10. Пусть $D = \langle V, T \rangle$ — стягивающее дерево с корнем r связного графа $G = \langle V, E \rangle$, построенное с помощью поиска в глубину (алгоритм 2.3). Вершина $v \in V$ является точкой сочленения графа G тогда и только тогда, когда либо $v = r$ и r имеет по крайней мере двух сыновей в D , либо $v \neq r$ и существует сын w вершины u , такой что ни w , ни какой-либо его потомок не связаны ребром ни с одним предком вершины u .

Доказательство. Рассмотрим сначала случай $v = r$. Если вершина v имеет только одного сына (либо является изолированной вершиной), то ее устранение не увеличит числа компонент связности, ибо это не нарушает связности дерева. Если она имеет по меньшей мере двух сыновей, то после удаления вершины v эти сыновья лежат в разных компонентах связности. Действительно, из теоремы 2.4 вытекает, что каждый путь между двумя различными сыновьями должен проходить через корень — в противном случае он содержал бы хорду $\{u, t\}$, где ни u не является потомком t , ни t не является потомком u . Аналогично, для случая $v \neq r$. Если после устранения вершины v существует путь от ее сына w до корня, то, как и в предыдущем случае из теоремы 2.4, следует, что этот путь должен содержать ребро, соединяющее w или некоторого потомка вершины w с некоторым предком вершины u .

Идея алгоритма следующая: ведем поиск в глубину в графе, начиная с некоторой вершины r , вычисляя для каждой вершины u два параметра: $WGN[v]$ и $L[v]$. Первый из них — это просто номер вершины v в порядке, в котором вершины посещаются при поиске в глубину, начиная с вершины r . Если через $D = \langle V, T \rangle$ мы обозначим дерево, соответствующее нашему поиску в глубину, то другой параметр определяет наименьшее значение $WGN[u]$, где $u = v$ или вершина u связана хордой с вершиной v или ее произвольным потомком в D . Параметр $L[v]$ легко вычислить по индукции относительно дерева D , если мы знаем $L[w]$ для всех сыновей w вершины v . Обозначив

$$A = \min\{L[w] : w \text{ — сын вершины } v\},$$

$$B = \min\{WGN[u] : \{u, v\} \in E \setminus T\},$$

имеем $L[v] = \min\{WGN[v], A, B\}$.

Из теоремы 2.10 следует, что v является точкой сочленения или корнем тогда и только тогда, когда $L[w] \geq WGN[v]$ для некоторого сына u вершины v

(полагаем, что $n > 1$).

Алгоритм 2.11. (Нахождение компонент двусвязности графа).

Данные: Граф $G = \langle V, E \rangle$ без изолированных вершин, представленный списками инцидентности ЗАПИСЬ $[v]$, $v \in V$.

Результаты: Множества ребер всех компонент двусвязности.

1. **procedure** ДВУСВ(v, p);

(* поиск в глубину, начиная с вершины v и полагая, что p является отцом вершины u в этом процессе;

параллельно выписываются ребра найденных компонент двусвязности; переменные num , L , WGN , СТЕК — глобальные*)

2. **begin** $num := num + 1$; $WGN[v] := num$;

3. $L[v] := WGN[v]$;

4. **for** $u \in \text{ЗАПИСЬ}[v]$ **do**

5. **if** $WGN[u] = 0$ **then** (*вершина u — новая, $\{v, u\}$ — ветвь*)

6. **begin** СТЕК $\leftarrow \{v, u\}$; ДВУСВ(u, v);

7. $L[v] := \min(L[v], L[u])$;

8. **if** $L[u] \geq WGN[v]$ **then** (* v есть корень или точка сочленения, а верхняя часть стека до $\{v, u\}$ включительно содержит компоненту двусвязности*)

9. **begin** (*выписать ребра компоненты двусвязности*)

10. **repeat** $e \leftarrow \text{СТЕК}$; $write(e)$

11. **until** $e = \{v, u\}$;

12. $write(';')$ (*знак конца компоненты двусвязности*)

13. **end**

14. **end**

15. **else** (* $WGN[u] > 0$, т.е. u уже была посещена*)

16. **if** ($u \neq p$) **and** ($WGN[u] < WGN[u]$) **then** ⁴

(* ребро $\{v, u\}$ — хорда и не включается в стек*)

17. **begin** СТЕК $\leftarrow \{v, u\}$;

18. $L[v] := \min\{L[v], WGN[u]\}$

19. **end**

20. **end**; (*ДВУСВ*)

21. **begin** (*главная программа*)

22. **for** $u \in V$ **do** $WGN[u] := 0$; (*инициализация*)

23. $СТЕК := \emptyset$; $num := 0$;

24. **for** $u \in V$ **do**

25. **if** $WGN[u] = 0$ **then** ДВУСВ($u, 0$)

26. **end**

Покажем теперь, что вызов процедуры ДВУСВ($v, 0$) (строка 25) для еще не

рассматривавшейся вершины v влечет за собой выделение и фиксирование всех блоков компоненты связности графа, содержащей вершину v . Доказательство проводится методом индукции по числу блоков этой компоненты. Если компонента связности не содержит точек сочленения, то нетрудно отметить, что вызов $\text{ДВУСВ}(v, 0)$ приводит к засылке в стек всех ребер этой компоненты, а затем (см. цикл 10) запись этих ребер. Предположим теперь, что наша компонента содержит $k > 1$ блоков и что алгоритм работает корректно для произвольной компоненты, содержащей менее k блоков. Рассмотрим первое встретившееся ребро $\{v, u\}$, такое что $L[u] \geq \text{WGN}[v]$ в строке 8. Согласно нашим предыдущим рассуждениям это неравенство означает, что v — корень или точка сочленения. Ни один из потомков вершины u (в дереве поиска в глубину, реализованном соответствующим алгоритмом) не является точкой сочленения, и в результате ребра верхней части стека до $\{v, u\}$ включительно являются ребрами графа, соединяющими потомков вершины v вместе с самой u , и тем самым создают блок, который выписывается в цикле 10. Модифицируем теперь нашу компоненту, удалив описанный выше блок (не удаляя вершину v). Модифицированная таким образом компонента имеет $k - 1$ блок, и выполнение для нее процедуры $\text{ДВУСВ}(v, 0)$ вызывает в силу индуктивного предположения корректную запись этих блоков. Действия алгоритма для модифицированной компоненты отличаются от действий в случае с исходной компонентой только тем, что для первой встреченной точки сочленения (исходной компоненты) v цикл 4 не выполняется для вершин u , принадлежащих удаленному блоку. Отсюда легко следует, что $\text{ДВУСВ}(u, 0)$ корректно определяет все k блоков нашей компоненты связности, а тем самым весь алгоритм корректно определяет все блоки графа.

Оценим теперь вычислительную сложность алгоритма. Циклы 22 и 25 требуют $O(n)$ шагов, причем для второго цикла не учитываются шаги, выполняемые при вызове $\text{ДВУСВ}(v, 0)$ для каждой еще нерассмотренной вершины. Такой вызов для компоненты связности с n_i вершинами и m_i ребрами требует $O(n_i + m_i)$ шагов, не считая шагов в цикле 10, так как такая процедура ведет в компоненте поиск в глубину, требуя число шагов, ограниченное константой для каждого просматриваемого ребра. Каждое ребро удаляется из стека и попадает в список ребер блока в точности один раз, что дает в сумме $O(m)$ шагов, совершаемых циклом 10 во время его выполнения всего алгоритма. Суммируя все эти слагаемые, получаем в итоге общую сложность алгоритма, равную $O(n + m)$.

2.7 Эйлеровы пути

Эйлеровым путем в графе называется произвольный путь, проходящий через каждое ребро графа в точности один раз, т.е. путь v_1, \dots, v_{m+1} , такой что каждое ребро $e \in E$ появляется в последовательности v_1, \dots, v_{m+1} в точности один

раз как $e = \{v_i, v_{i+1}\}$ для некоторого i . Если $v_i = v_{m+1}$, то такой путь называется *эйлеровым циклом*. Задача существования эйлерова пути в заданном графе была решена Эйлером в 1736 г., и представленное им необходимое и достаточное условие существования такого пути (см. теорему 2.12) считается первой в истории теоремой теории графов.

Теорема 2.12. *Эйлеров путь в графе существует тогда и только тогда, когда граф связный и содержит не более чем две вершины нечетной степени.*

Доказательство. Доказательство достаточности условия теоремы будет следствием анализа алгоритма нахождения эйлерова пути, который мы опишем в данном разделе. Необходимость условия очевидна, так как если вершина v , отличная от v_1 и v_{m+1} , появляется в эйлеровом пути v_1, \dots, v_{m+1} k раз, то это означает, что степень этой вершины в графе составляет $2k$. Отсюда следует, что вершины нечетной степени, если они существуют, являются концами эйлерова пути. Здесь следует отметить, что не существует графов с одной только вершиной нечетной степени. Действительно, обозначая степень вершины v через $d(v)$, имеем

$$\sum_{v \in V} d(v) = 2m,$$

ибо в указанной выше сумме каждое ребро $\{v, u\}$ считается дважды: один раз в $d(u)$ и один раз в $d(v)$. Отсюда следует, что число вершин нечетной степени всегда четно.

Если в связном графе нет вершин нечетной степени, то каждый эйлеров путь является циклом, так как концы эйлерова пути, не являющегося циклом, всегда вершины нечетной степени. Предположим, что u и v — единственные вершины нечетной степени связного графа $G = \langle V, E \rangle$, и образуем граф G^* добавлением дополнительной вершины t и ребер $\{u, t\}$ и $\{v, t\}$ (или просто $\{u, v\}$, если $\{u, v\} \notin E$). Тогда G^* — связный граф без вершин нечетной степени, а эйлеровы пути в G будут в очевидном взаимно однозначном соответствии с эйлеровыми циклами в G^* . В силу этого в оставшейся части данного раздела мы будем заниматься только эйлеровыми циклами.

Алгоритм 2.13 (Нахождение эйлерового цикла).

Данные: Связный граф $G = \langle V, E \rangle$ без вершин нечетной степени, представленный списками ЗАПИСЬ[v], $v \in V$.

Результаты: Эйлеров цикл, представленный последовательностью вершин в стеке СЕ.

1. begin
2. СТЕК := \emptyset ; СЕ := \emptyset ;
3. v := произвольная вершина графа;
4. СТЕК $\leftarrow v$;

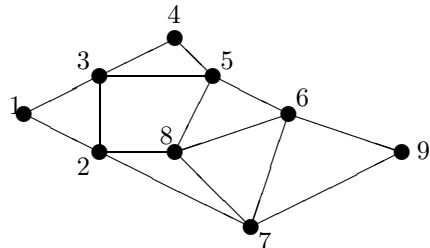
```

5.   while СТЕК  $\neq \emptyset$  do
6.       begin  $v := \text{top}(\text{СТЕК});$ (* $v$ =верхний элемент стека*)
7.       if ЗАПИСЬ[ $v$ ]  $\neq \emptyset$  then
8.           begin  $u :=$ первая вершина списка ЗАПИСЬ[ $v$ ];
9.               СТЕК  $\leftarrow u$ ;
10.              (*удалить ребро  $\{v, u\}$  из графа*)
11.              ЗАПИСЬ[ $v$ ] := ЗАПИСЬ[ $v$ ]\{ $u$ \};
12.              ЗАПИСЬ[ $u$ ] := ЗАПИСЬ[ $u$ ]\{ $v$ \};
13.               $v := u$ 
14.          end
15.       else (*ЗАПИСЬ[ $v$ ] =  $\emptyset$ *)
16.           begin  $v \leftarrow$  СТЕК,  $CE \leftarrow u$ 
17.           end
18.   end

```

Принципы действия алгоритма можно объяснить следующим образом: пусть v_0 — вершина, выбранная в строке 3. Цикл 5 начинает строить путь с началом в v_0 , причем вершины этого пути помещаются в СТЕК, а ребра удаляются из графа. Эти действия продолжают вплоть до того момента, когда путь нельзя удлинить, включив в него новую вершину, т.е. когда ЗАПИСЬ[v] = \emptyset в строке 7. Отметим, что тогда должно быть $v = v_0$, так как в любом другом случае это означало бы, что степень вер-

шины v нечетная. Таким образом, из нашего графа был удален цикл, а вершины этого цикла находятся в стеке СТЕК. Отметим, что в графе, модифицированном таким способом, степень произвольной вершины останется четной. Вершина $v = v_0$ переносится теперь из стека СТЕК в стек СЕ, а «очередной» вершиной u становится верхний элемент стека СТЕК. Процесс повторяется, начиная с этой вершины (если ЗАПИСЬ[v] $\neq \emptyset$), в результате чего вследствие четности степени всех вершин находится и помещается в стек СТЕК, некоторый цикл, проходящий через вершину v . Это продолжается до того момента, когда СТЕК не станет пустым. Очевидно, что вершины, помещаемые в стек СЕ, образуют некоторый путь, причем вершина v переносится в стек СЕ только тогда, когда ЗАПИСЬ[v] = \emptyset , т.е. когда все ребра, инцидентные с этой вершиной, представ-



1, 2, 3, 4, 5, 6, 7, 2, 8, 6, 9, 7, 8, 5, 3, 1

Рис. 2.9: Граф и эйлеров цикл в этом графе, найденный с помощью алгоритма 2.13.

лены (парами соседних вершин) в одном из стеков. Отсюда легко следует, что по окончании алгоритма стек SE содержит эйлеров цикл.

Оценим теперь вычислительную сложность нашего алгоритма. Для этого отметим, что каждая итерация главного цикла (строка 5) либо помещает вершину в стек СТЕК и удаляет ребро из графа, либо переносит вершину из стека СТЕК в стек SE. Таким образом, число итераций этого цикла — $O(m)$. В свою очередь число шагов в каждой итерации ограничено константой.

Мы предполагаем здесь, что каждый из списков инцидентности ЗАПИСЬ $[v]$, $v \in V$, реализован таким образом, что каждая вершина в этом списке содержит указатели на предыдущую и последующую вершины, и вершина u в списке ЗАПИСЬ $[v]$ содержит указатель на вершину v в списке ЗАПИСЬ $[u]$. Это дает возможность устранить ребро $\{v, u\}$ (строка 10) за время, ограниченное константой (см. разд. 2.1). Из приведенных выше рассуждений можно сделать вывод, что общая сложность алгоритма есть $O(m)$. Очевидно, что этот алгоритм оптимальный, так как уже одно выписывание эйлерова цикла требует $\Omega(m)$ шагов.

На рис. 2.9 представлен граф и некоторый эйлеров цикл, найденный алгоритмом 2.13.

2.8 Алгоритмы с возвратом (back-tracking)

Рассмотрим теперь задачу, рассмотренную в предыдущем параграфе, с той разницей, что на этот раз нас будут интересовать пути, проходящие в точности один раз через каждую вершину (а не каждое ребро) данного графа. Эта небольшая, как могло бы показаться, модификация приводит, как мы убедимся, к значительному изменению характера проблемы. Вспомним, что путь с указанным свойством называется *гамильтоновым путем* (*гамильтонов цикл* определяется очевидным образом). Пример гамильтонова пути в графе и графа, в котором такого пути не существует, показан на рис. 2.10.

В отличие от эйлеровых путей не известно ни одного простого необходимого и достаточного условия для существования гамильтоновых путей и это несмотря на то, что эта задача — одна из центральных в теории графов. Не известен также алгоритм, который проверял бы существование гамильтонова пути в произвольном графе, используя число шагов, ограниченное многочленом от переменной n (числа вершин в графе). Имеются некоторые основания, но нет математического доказательства того, чтобы предполагать, что такое положение вещей вызвано не столько нашим незнанием, а скорее тем фактом, что такого алгоритма не существует. Проблема существования гамильтонова пути принадлежит к классу так называемых *NP-полных задач* [27]. Детальное обсуждение этих задач выходит за рамки этой книги. Отметим здесь только, что это — широкий класс задач, включающий фундаментальные задачи из теории

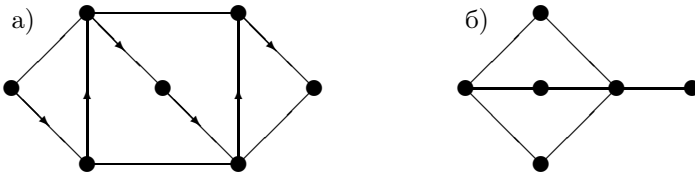


Рис. 2.10: а) Гамильтонов путь в графе; б) Граф, в котором не существует гамильтонова пути.

графов, логики, теории чисел, дискретной оптимизации и других дисциплин, ни для одной из которых неизвестен полиномиальный алгоритм (т.е. с числом шагов, ограниченным полиномом от размерности задачи), причем существование полиномиального алгоритма для хотя бы одной из них автоматически влекло бы за собой существование полиномиальных алгоритмов для всех этих задач. Именно факт фундаментальности многих NP-полных задач в различных областях и то, что, несмотря на независимые друг от друга усилия специалистов в этих областях, не удалось найти полиномиального алгоритма ни для одной из этих задач, склоняет к предположению, что такого алгоритма не существует.

Вернемся к конкретной задаче существования гамильтонова пути. Очевидный алгоритм, который мы можем применить, это «полный перебор всех возможностей»: генерируем все $n!$ различных последовательностей вершин и для каждой из них проверяем, определяет ли она гамильтонов путь. Такие действия требуют по меньшей мере $n!n$ шагов, но функция от n подобного вида растет быстрее, чем произвольный многочлен, и даже быстрее, чем произвольная экспоненциальная функция вида a^n , $a > 1$, ибо

$$n!n > \lfloor n/2 \rfloor^{\lfloor n/2 \rfloor} = a^{\lfloor n/2 \rfloor \log_a \lfloor n/2 \rfloor}$$

Опишем теперь общий метод, позволяющий значительно сократить число шагов в алгоритмах типа полного перебора всех возможностей. Чтобы применить этот метод, искомое решение должно иметь вид последовательности $\langle x_1, \dots, x_n \rangle$. Основная идея метода состоит в том, что мы строим наше решение последовательно, начиная с пустой последовательности ε (длины 0). Вообще, имея данное частичное решение $\langle x_1, \dots, x_i \rangle$, мы стараемся найти такое допустимое значение x_{i+1} , относительно которого мы не можем сразу заключить, что $\langle x_1, \dots, x_i, x_{i+1} \rangle$ можно расширить до некоторого решения (либо $\langle x_1, \dots, x_{i+1} \rangle$ уже является решением). Если такое предполагаемое, но еще не использованное значение x_{i+1} существует, то мы добавляем его к нашему частичному решению и продолжаем процесс для последовательности $\langle x_1, \dots, x_i, x_{i+1} \rangle$. Если его не существует, то мы возвращаемся к нашему частичному решению $\langle x_1, \dots, x_{i-1} \rangle$ и

продолжаем наш процесс, отыскивая новое, еще не использованное допустимое значение x'_i — отсюда название «алгоритм с возвратом» (англ.: backtracking).

Точнее говоря, мы предполагаем, что для каждого $k > 0$ существует некоторое множество A_k , из которого мы будем выбирать кандидатов для k -й координаты частичного решения. Очевидно, что множества A_k должны быть определены таким образом, что для каждого целочисленного решения $\langle x_1, \dots, x_n \rangle$ нашей задачи и для каждого $k \leq n$ множество A_k содержало элемент x_k (на практике мы не можем вообще исключить ситуацию, когда множество A_k содержит некоторые «лишние» элементы, не появляющиеся в k -й координате ни одного целочисленного решения). Мы предполагаем также, что существует некоторая простая функция, которая произвольному частичному решению $\langle x_1, \dots, x_i \rangle$ ставит в соответствие значение $P(x_1, \dots, x_i)$ (**истина** либо **ложь**) таким образом, что если $P(x_1, \dots, x_i) = \text{ложь}$, то последовательность $\langle x_1, \dots, x_i \rangle$ несомненно нельзя расширить до решения. Если $P(x_1, \dots, x_i) = \text{истина}$, то мы говорим, что значение x_i *допустимо* (для частичного решения $\langle x_1, \dots, x_{i-1} \rangle$), но это отнюдь не означает, что $\langle x_1, \dots, x_{i-1} \rangle$ обязательно расширяется до полного решения. Этот процесс можно записать в виде следующей схемы:

```

1 begin
2    $k := 1$ ;
3   while  $k > 0$  do
4     if существует еще не использованный элемент  $y \in A_k$ ,
       такой что  $P(X[1], \dots, X[k-1], y)$ 
       then
5       begin  $X[k] := y$ ; (* элемент  $y$  использован*)
6         if  $\langle X[1], \dots, X[k] \rangle$  является целочисленным решением then
7           write( $X[1], \dots, X[k]$ );
8            $k := k + 1$ 
9         end
10    else (* возврат на более короткое частичное решение;
          все элементы множества  $A_k$  вновь становятся неиспользованными *)
11       $k := k - 1$ 
12    end

```

Этот алгоритм находит все решения в предположении, что множества A_k конечные и что существует n , такое что $P(x_1, \dots, x_n) = \text{ложь}$ для всех $x_1 \in A_1, \dots, x_n \in A_n$ (последнее условие означает, что все решения имеют длину меньше n). Покажем наличие более общего свойства:

Пусть $s > 0$, и пусть $\langle x_1, \dots, x_{s-1} \rangle$ — некоторое частичное решение, построенное алгоритмом. Рассмотрим первую итерацию цикла

3, для которой $k = s$, $X[i] = x_i$, $1 \leq i \leq s$. Начиная с этой итерации, алгоритм генерирует все целочисленные решения, являющиеся расширением последовательности (x_1, \dots, x_{s-1}) , и приходит к состоянию, когда $k = s - 1$.

Очевидно, что для $s = 1$ приведенное выше свойство означает просто корректность алгоритма. Доказательство этого свойства в общем виде проводится по индукции «назад» относительно s . Оно имеет место для $s = n$, потому что не существует ни одного допустимого элемента для $\langle x_1, \dots, x_{s-1} \rangle$ и сразу же выполняется второе условие условного оператора в строке 10, т.е. переменная k принимает значение $s - 1$. Предположим теперь правильность нашего свойства для некоторого $s > 1$. Покажем его правильность для $s - 1$. Пусть $\langle x_1, \dots, x_{s-2} \rangle$ — произвольное частичное решение, построенное нашим алгоритмом; рассмотрим первую итерацию цикла 3, для которой $k = s - 1$, $X[i] = X_i$, $1 \leq i \leq s - 1$. Если существует элемент y , допустимый для $\langle x_1, \dots, x_{s-2} \rangle$, то построение дает частичное решение $\langle x_1, \dots, x_{s-2}, y \rangle$ (строка 5) и переменная k принимает значение s (строка 8). Затем согласно нашему индуктивному предположению будут сгенерированы все решения, являющиеся расширением последовательности $\langle x_1, \dots, x_{s-1}, y \rangle$, и мы приходим к состоянию, когда $k = s - 1$, после чего процесс повторяется для следующего неиспользованного элемента y , допустимого для $\langle x_1, \dots, x_{s-2} \rangle$, пока не будут использованы все такие элементы (такая ситуация может появиться сразу с самого начала). Переменная k уменьшается тогда на 1 и принимает значение $s - 2$ (строка 11). Из приведенных выше рассуждений следует, что алгоритм правильно порождает все решения, являющиеся расширением последовательности $\langle x_1, \dots, x_{s-1} \rangle$, что завершает доказательство шага индукции.

Доказанное нами свойство приводит непосредственно к следующему простому и очевидному рекурсивному варианту схемы алгоритма с возвратом:

```

1 procedure  $AP(k)$ ; (*генерирование всех решений,
являющихся расширением последовательности  $X[1], \dots, X[k - 1]$ ;
массив  $X$  — глобальный *)
2 begin
3   for  $y \in A_k$  такого, что  $P(X[1], \dots, X[k - 1], y)$  do
4     begin  $X[k] := y$ ;
5       if  $X[1], \dots, X[k]$  есть целочисленное решение then
6          $write(X[1], \dots, X[k])$ ;
7          $AP(k + 1)$ 
8     end
9 end

```

Генерирование всех целочисленных решений можно вызвать вызовом $AP(1)$.

Представление алгоритма с возвратом мы начали с несколько более сложного нерекурсивного варианта только потому, что в рекурсивном варианте «возврат» не появляется в явном виде, будучи частью реализации механизма рекурсии.

Применим теперь алгоритм с возвратом для нахождения гамильтонова цикла в графе $G = \langle V, E \rangle$. Каждый такой цикл можно представить последовательностью $\langle x_1, x_2, \dots, x_{n+1} \rangle$, причем $x_1 = x_{n+1} = v_0$, где v_0 — некоторая фиксированная вершина графа, $x_i = x_{i+1}$ для $1 \leq i \leq n$ и $x_i \neq x_j$ для $1 \leq i < j \leq n$. Согласно этим условиям мы можем положить:

$$A_k = V, \\ P(x_1, \dots, x_k - 1, y) \Leftrightarrow y \in \text{ЗАПИСЬ}[x_k - 1] \wedge y \notin \{x_1, \dots, x_{k-1}\}.$$

Алгоритм 2.14. (Нахождение всех гамильтоновых циклов в графе.)

Данные: Граф $G = \langle V, E \rangle$, представленный списками инцидентности $\text{ЗАПИСЬ}[v]$, $v \in V$.

Результаты: Список всех гамильтоновых циклов графа G .

1 **procedure** ГАМИЛЬТ(k) (* генерирование всех гамильтоновых циклов, являющихся расширением последовательности $\langle X[1], \dots, X[k-1] \rangle$; массивы X , DOP — глобальные *)

2 **begin**

3 **for** $y \in \text{ЗАПИСЬ}[X[k-1]]$ **do**

4 **if** $(k = n + 1)$ **and** $(y = v_0)$ **then** $\text{write}(X[1], \dots, X[n], v_0)$

5 **else if** $DOP[y]$ **then**

6 **begin** $X[k] := y$; $DOP[y] := \text{ложь}$;

7 ГАМИЛЬТ($k + 1$);

8 $DOP[y] := \text{истина}$

9 **end**

10 **end**; (*ГАМИЛЬТ*)

11 **begin** (*главная программа*)

12 **for** $v \in V$ **do** $DOP[v] := \text{истина}$; (* инициализация*)

13 $X[1] := v_0$; (* v_0 -произвольная фиксированная вершина графа*)

14 $DOP[v_0] := \text{ложь}$;

15 ГАМИЛЬТ(2);

16 **end**

Работу этого алгоритма, так же как и произвольного алгоритма с возвратом, можно проиллюстрировать процессом поиска в некотором дереве. Каждая вершина дерева естественным образом соответствует некоторой последовательности $\langle x_1, \dots, x_k \rangle$, причем вершины, соответствующие последовательностям вида $\langle x_1, \dots, x_k, y \rangle$, являются сыновьями этой вершины (корень соответствует пустой последовательности ε). Рассмотрим полное дерево D , состоящее из всех

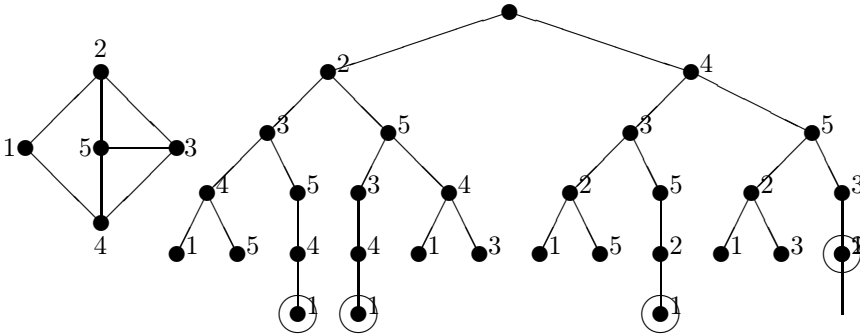


Рис. 2.11: Граф и дерево, иллюстрирующие алгоритм с возвратом нахождения всех гамильтоновых циклов в этом графе.

возможных последовательностей вида $\langle x_1, \dots, x_k \rangle$, где $0 \leq k \leq n$ и $x_i \in A_i$ для $1 \leq i \leq k$, и временно допустим, что каждый элемент $y \in A$ является допустимым для $\langle x_1, \dots, x_{k-1} \rangle$, если $k \leq n$, и ни один элемент не является допустимым для $\langle x_1, \dots, x_{k-1} \rangle$, если $k > n$; другими словами,

$$P(x_1, \dots, x_{k-1}, k_k) = \begin{cases} \text{истина,} & \text{если } k \leq n \\ \text{ложь,} & \text{если } k > n \end{cases}$$

($x_i \in A_i$ для $1 \leq i \leq k$). Тогда нетрудно отметить, что вызов $AP(1)$ вызывает поиск в глубину во всем дереве D (начиная от корня ε). Для случая менее тривиальной функции P , определяющей допустимость вершин, процесс поиска «опускает» рассмотрение вершин поддерева с корнем в каждой встреченной «недопустимой» вершине (т.е. вершине, соответствующей последовательности $\langle x_1, \dots, x_k \rangle$, для которой $P(x_1, \dots, x_k) = \text{ложь}$). В этом, собственно говоря, заключается эффективность алгоритма с возвратом в отношении полного перебора всех возможностей. Для алгоритма 2.14 это иллюстрируется рис. 2.11.

Следует отметить, что для большинства приложений число шагов алгоритма с возвратом хотя и будет меньше, чем в случае полного перебора, однако же в наихудшем случае растет по экспоненте с возрастанием размерности задачи. Это справедливо и для случая, когда мы ищем только одно, а не все решения (тогда мы просто прерываем выполнение алгоритма после получения первого решения; отметим, что, когда задача не имеет решения, эта модификация не влияет на ход выполнения алгоритма).

2.9 Задачи

2.1. Доказать, что для произвольного неориентированного графа матрица смежности B выражается через матрицу инцидентности A следующим образом:

$$B = A \cdot A^T - \text{diag}[d_1, \dots, d_n],$$

где A^T есть транспонированная матрица A , d_i — степень i -й вершины, $\text{diag}[d_1, \dots, d_n]$ — матрица размера $n \times n$ с элементами d_1, \dots, d_n на главной диагонали.

2.2. Для произвольного неориентированного графа определим матрицу соседства ребер $C = [c_{ij}]$, где $c_{ij} = 1$, если i -е и j -е ребра инцидентны с общей вершиной, и $c_{ij} = 0$ в противном случае (принимаем $c_{ii} = 0$). Как можно выразить матрицу C через матрицу инцидентности? Определяет ли матрица соседства ребер граф с точностью до изоморфизма?

2.3. Представить детальную реализацию списков инцидентности и процедур удаления и добавления ребер за время, ограниченное константой (следует помнить, что при удалении ребра u, v необходимо удалить соответствующие элементы из обоих списков ЗАПИСЬ[u] и ЗАПИСЬ[v], мы предполагаем, что указатель на один из этих элементов представлен как параметр процедуры).

2.4. Написать процедуры перехода между каждыми двумя способами представления графа, описанными в разд. 2.1, и оценить вычислительную сложность указанных процедур.

2.5. Некоторые графовые алгоритмы используют только $O(n)$ или $O(n + m)$ из n^2 элементов матрицы смежности B . В то же время в начале алгоритма следует инициализация всех элементов матрицы B , в результате чего сложность алгоритма автоматически становится равной $\Omega(n^2)$. Предложить способ, позволяющий избежать столь дорогую инициализацию. *Указание:* для каждого элемента b_{ij} , использованного алгоритмом первый раз, помещаем в стек указатель на этот элемент, а при b_{ij} размещаем указатель на созданный таким образом новый элемент стека. Чтобы проверить, был ли данный элемент инициализирован ранее, достаточно посмотреть, ведет ли указатель при b_{ij} к указателю в стеке, определяющему возврат к b_{ij} [1].

2.6. Предложить алгоритм сложности $O(n)$ проверки, содержит ли данный ориентированный граф, заданный матрицей смежности, вершину, к которой примыкают дуги от остальных $n - 1$ вершин и из которой не исходит ни одна дуга (такой алгоритм показывает, что условие (в) в теореме 2.1 существенно).

2.7. Исследовать метод, отличающийся от поиска в ширину только тем, что вновь достигнутая вершина помещается в стек (такой алгоритм описывается в точности процедурой WS , в которой ОЧЕРЕДЬ заменена на СТЕК) [35].

2.8. Исследовать метод поиска в графе, при котором посещенные, но еще не использованные вершины помещаются в очередь, и который точнее можно

описать следующим способом: рассматриваем последнюю (позже всех помещенную) вершину u из очереди (вначале это будет произвольная вершина графа). Если существует новая вершина $u \in \text{ЗАПИСЬ}[v]$, то мы помещаем ее в конце очереди и повторяем процесс до тех пор, пока помещенная в очередь вершина не имеет новых соседей (этот фрагмент напоминает поиск в глубину). Затем исследуем первую (раньше всего помещенную) вершину из очереди. Если у этой вершины существует новый сосед, то мы ставим его в конец очереди и повторяем для этой вершины описанный выше процесс поиска в глубину. В противном случае первая вершина из очереди использована, и мы убираем ее из очереди, а затем рассматриваем следующую за ней вершину. Весь процесс повторяется до тех пор, пока очередь не будет пустой. Представить детальную реализацию данного метода.

2.9. Доказать, что приведенная ниже процедура просматривает каждую вершину графа в точности один раз.

procedure ПОИСК(r);

(*вначале СТАТУС[v]=новый для каждой $v \in V^*$

begin

 посетить r ; СТАТУС[r]:=посещенный;

while существует в V посещенная вершина **do**

begin u :=произвольная посещенная вершина;

if существует новая $u \in \text{ЗАПИСЬ}[v]$ **then**

begin u :=первая новая вершина списка ЗАПИСЬ[v];

 посетить u ; СТАТУС[u]:=посещенная

end

else СТАТУС[v]:=использованная

end

end

Показать, что поиск в глубину и в ширину, а также методы, описанные в задачах 2.7 и 2.8, можно рассматривать как частные случаи этой процедуры.

2.10. Пусть $D = \langle V, T \rangle$ — подграф связного графа $G = \langle V, E \rangle$. Доказать, что следующие условия эквивалентны:

- (а) D является стягивающим деревом графа G .
- (б) $|T| = |E| - 1$ и D не содержит циклов.
- (в) Для каждой пары вершин $u, v \in V$ существует в D в точности один путь из u в v .
- (г) $|T| = |E| - 1$ и для каждой пары вершин $u, v \in V$ существует в D хотя бы один путь из u в v .

- (д) $|T| = |E| - 1$ и для каждой пары вершин $u, v \in V$ существует в D не более чем один путь из u в v .

2.11. Рассмотреть детальнее метод поиска в глубину для ориентированных графов, упомянутый в конце §2.2. Показать, что если в ориентированном графе G существует путь из вершины r к каждой другой вершине, то алгоритм 2.3 (измененный так, что в строке 5 добавляется дуга $\langle v, u \rangle$) строит подграф $\langle V, T \rangle$, который при игнорировании ориентации дуг является стягивающим деревом графа G . Показать, что в $\langle V, T \rangle$ существует в точности один путь из r к каждой вершине $v \in V$. Можно ли перенести теорему 2.4 на ориентированные графы? Как следовало бы ее тогда изменить?

2.12. Рассмотреть детальнее метод поиска в ширину в ориентированном графе. Показать, что если в ориентированном графе G существует путь из вершины r к каждой другой вершине, то алгоритм 2.5 находит подграф $\langle V, T \rangle$, содержащий в точности один путь из r к каждой другой вершине, причем это кратчайший путь в G .

2.13. Используйте поиск в глубину для нахождения в связном графе неориентированного цикла (с повторяющимися ребрами и вершинами), проходящего каждое ребро в каждом направлении в точности один раз.

2.14. *Мостом* графа G называется каждое ребро, удаление которого приводит к увеличению числа связных компонент графа. Представить алгоритм нахождения всех мостов графа за время $O(m + n)$. (*Указание:* используйте поиск в глубину как в случае отыскания точек сочленения.)

2.15. Граф $G = \langle V, E \rangle$ называется *двудольным*, если существует разбиение $V = A \cup B$, $A \cap B = \emptyset$, такое что каждое ребро $e \in E$ имеет вид $e = \{a, b\}$, $a \in A$, $b \in B$. Доказать, что граф является двудольным тогда и только тогда, когда он не содержит простых циклов нечетной длины. Предложить два алгоритма, проверяющих, будет ли данный граф двудольным, за время $O(m + n)$ — один, основанный на поиске в глубину, другой — на поиске в ширину.

2.16. Написать алгоритм, проверяющий за время $O(m + n)$ ацикличность данного ориентированного графа.

2.17. Ориентированный граф называется *сильно связным*, если для каждой пары вершин u и v существует путь из u в v . Под *компонентой сильной связности* понимается произвольной максимальной сильно связный подграф. Показать, что компоненты сильной связности определяют разбиение множества вершин на непересекающиеся непустые подмножества. Написать алгоритм нахождения компонент сильной связности за время $O(m + n)$. (*Указание:* использовать поиск в глубину.)

2.18. Написать алгоритм нахождения за время $O(c(m + n))$ всех простых контуров ориентированного графа, где c — число контуров. (*Указание:* использовать поиск в глубину.)

2.19. Предложить необходимое и достаточное условие существования эйлеровых контуров в ориентированном графе (идентичное условию для неориентированных графов). Показать, что алгоритм 2.13 (после удаления строки 9) правильно строит эйлеров контур для ориентированного графа.

2.20. Циклом де Брёйна порядка n называется циклическая бинарная последовательность длины $2n$, в которой каждая из $2n$ бинарных последовательностей длины n выступает как подпоследовательность n последовательных членов. Доказать существование цикла де Брёйна порядка n для каждого $n \geq 1$. (Указание: рассмотреть граф, вершины которого соответствуют бинарным последовательностям длины $n - 1$, а каждое ребро соответствует бинарной последовательности длины n , причем ребро $b_1 \dots b_n$ ведет от $b_1 \dots b_{n-1}$ к $b_2 \dots b_n$. Доказать существование эйлерова пути в таком графе [8].)

2.21. Изменить представленную в §2.8 схему алгоритма с возвратом так, чтобы он определял только одно решение, а не все.

2.22. Доказать, представив соответствующие графы, что число шагов в алгоритме 2.14 (в наихудшем случае) растет экспоненциально с ростом n .

2.23. Использовать алгоритм с возвратом для решения следующих задач:

- (а) Нахождение размещения 8 взаимно не нападающих друг на друга ферзей на шахматной доске.
- (б) Нахождение клики наибольшей мощности в неориентированном графе. (Кликкой называется произвольное подмножество вершин, в котором каждая пара различных вершин соединена ребром графа.)
- (в) Нахождение раскраски вершин графа минимальным числом цветов так, чтобы ни одно ребро не соединяло двух вершин одного цвета.
- (г) Для данных целых чисел a_1, \dots, a_n, b нахождение множества индексов $J \subseteq \{1, \dots, n\}$, такого что $\sum_{j \in J} a_j = b$, если такое множество J существует.
- (д) Установление изоморфизма двух графов.

Глава 3

Нахождение кратчайших путей в графе

3.1 Начальные понятия

В этой главе мы будем рассматривать ориентированные графы $G = \langle V, E \rangle$, дугам которых приписаны веса. Это означает, что каждой дуге $\langle v, u \rangle \in E$ поставлено в соответствие некоторое вещественное число $a(v, u)$, называемое *весом* данной дуги. Полагаем, кроме того, что $a(v, v) = \infty$, если $v \not\rightarrow v$. Если последовательность вершин

$$v_0, v_1, \dots, v_p \tag{3.1}$$

определяет путь в G , то его длина определяется как сумма

$$\sum_{i=1}^p a(v_{i-1}, v_i)$$

(Отметим, что если в произвольном графе мы примем вес каждой дуги равным единице, то мы получим обычное определение длины пути как числа дуг; как и ранее, принимаем, что длина пути равна 0 при $p = 0$.) Нас будет интересовать нахождение кратчайшего пути между фиксированными вершинами $s, t \in V$. Длину такого кратчайшего пути мы будем обозначать $d(s, t)$ и называть *расстоянием* от s до t (расстояние, определенное таким образом, может быть отрицательным). Если не существует ни одного пути из s в t , то полагаем $d(s, t) = \infty$. Отметим, что если каждый контур нашего графа имеет положительную длину, то кратчайший путь будет всегда элементарным путем, т.е.

в последовательности (3.1) не будет повторов. С другой стороны, если в графе существует контур отрицательной длины, то расстояние между некоторыми парами вершин становится неопределенным, потому что, обходя этот контур достаточное число раз, мы можем показать путь между этими вершинами с длиной, меньшей произвольного вещественного числа. В таком случае можно было бы говорить о длине кратчайшего элементарного пути, однако задача, поставленная таким образом, вероятно будет значительно более сложной, так как, в частности, она содержит в себе задачу существования гамильтонова пути (см. задачу 3.1).

Можно дать много практических интерпретаций задачи о кратчайших путях. Например, вершины могут соответствовать городам, а каждая дуга — некоторому пути, длина которого представлена весом дуги. Мы ищем затем кратчайшие пути между городами. Вес дуги также может соответствовать стоимости (или времени) передачи информации между вершинами. В таком случае мы ищем самый дешевый (или самый скорый) путь передачи информации. Еще одну ситуацию мы получаем, когда вес дуги $\langle u, v \rangle$ равен вероятности $p(u, v)$ безаварийной работы канала передачи информации. Если предположить, что аварии каналов не зависят друг от друга, то вероятность исправности пути передачи информации равна произведению вероятностей составляющих его дуг. Задачу нахождения наиболее надежного пути легко можно свести к задаче о кратчайшем пути, заменяя веса $p(u, v)$ на

$$a(u, v) = -\log p(u, v).$$

Еще с одним приложением задачи о кратчайшем пути мы познакомимся в разд. 3.4.

В данном разделе мы дадим алгоритмы нахождения расстояния между вершинами, а не самих путей. Однако, зная расстояние, мы можем при условии положительной длины всех контуров легко определить кратчайшие пути. Для этого достаточно отметить, что для произвольных $s, t \in V$ ($s \neq t$) существует вершина v , такая что

$$d(s, t) = d(s, v) + a(v, t).$$

Действительно, таким свойством обладает предпоследняя вершина произвольного кратчайшего пути из s в t . Далее мы можем найти вершину u , для которой $d(s, v) = d(s, u) + a(u, v)$, и т.д. Из положительности длины всех контуров легко следует, что созданная таким образом последовательность t, u, v, \dots не содержит повторов и оканчивается вершиной s . Очевидно, что она определяет (при обращении очередности) кратчайший путь из s в t . Таким образом, мы получаем следующий алгоритм:

Алгоритм 3.1. (Нахождение кратчайшего пути).

Данные: расстояния $D[v]$ от фиксированной вершины s до всех остальных вершин $v \in V$, фиксированная вершина t , матрица весов ребер $A[u, v]$, $u, v \in V$.

Результаты: СТЕК содержит последовательность вершин, определяющую кратчайший путь из s в t .

```

1 begin
2     СТЕК := ∅; СТЕК ← t; v := t;
3     while v ≠ s do
4         begin
5             u := вершина, для которой D[v] = D[u] + A[u, v];
6             СТЕК ← u;
7             v := u;
8         end
9 end

```

Пусть $\langle V, E \rangle$ — ориентированный граф, $|V| = n$, $|E| = m$. Если выбор вершины u в строке 5 происходит в результате просмотра всех вершин, то сложность нашего алгоритма — $O(n^2)$. Если мы просматриваем только список ПРЕДШ[v], содержащий все вершины u , такие что $u \rightarrow v$, то в этом случае сложность будет $O(m)$.

Отметим, что в случае положительных весов ребер задача о кратчайшем пути в неориентированном графе легко сводится к аналогичной задаче для некоторого ориентированного графа. С этой целью достаточно заменить каждое ребро $\{u, v\}$ двумя дугами $\langle u, v \rangle$ и $\langle v, u \rangle$, каждая с таким же весом, что и $\{u, v\}$. Однако в случае неположительных весов это приводит к возникновению контуров с неположительной длиной. В данной главе мы всегда будем предполагать, что $G = \langle V, E \rangle$ является ориентированным графом, $n = |V|$, $m = |E|$. В целях упрощения изложения и избежания вырожденных случаев при оценке сложности алгоритмов будем принимать $m = \Omega(n)$ (т.е. $n = O(m)$). Это исключает ситуации, при которых «большинство» вершин изолированы. Будем предполагать также, что веса дуг запоминаются в массиве $A[u, v]$, $u, v \in V$ ($A[u, v]$ содержит вес $a(u, v)$).

3.2 Кратчайшие пути от фиксированной вершины

Большинство известных алгоритмов нахождения расстояния между двумя фиксированными вершинами s и t опирается на действия, которые в общих чертах можно представить следующим образом: при данной матрице весов дуг $A[u, v]$, $u, v \in V$, вычисляет некоторые верхние ограничения $D[u]$ на расстояния от s до

всех вершин $v \in V$. Каждый раз, когда мы устанавливаем, что

$$D[u] + A[u, v] < D[v], \quad (3.2)$$

оценку $D[v]$ улучшаем: $D[v] := D[u] + A[u, v]$.

Процесс прерывается, когда дальнейшее улучшение ни одного из ограниченных невозможно. Легко можно показать, что значение каждой из переменных $D[u]$ равно тогда $d(s, u)$ — расстоянию от s до v . Заметим, что, для того чтобы определить расстояние от s до t , мы вычисляем здесь расстояния от s до всех вершин графа. Не известен ни один алгоритм нахождения расстояния между двумя фиксированными вершинами, который был бы существенным образом более эффективным, нежели известные алгоритмы определения расстояния от фиксированной вершины до всех остальных.

Описанная нами общая схема является неполной, так как она не определяет очередности, в которой выбираются вершины u и v для проверки условия (3.2). Эта очередность оказывает, как будет доказано, очень сильное влияние на эффективность алгоритма. Опишем теперь более детально методы нахождения расстояния от фиксированной вершины, называемой *источником*, его всегда будем обозначать через s , до всех остальных вершин графа.

Сначала представим алгоритм для общего случая, в котором предполагаем только отсутствие контуров с отрицательной длиной. С этим алгоритмом обычно связывают имена Л.Р. Форда [22] и Р.Е. Беллмана [3].

Алгоритм 3.2. (Нахождение расстояния от источника до всех вершин — метод Форда–Беллмана.)

Данные: Ориентированный граф $\langle V, E \rangle$ с n вершинами с выделенным источником $s \in V$, матрица весов дуг $A[u, v]$, $u, v \in V$ (граф не содержит контуров отрицательной длины).

Результаты: Расстояния от источника до всех вершин графа: $D[v] = d(s, v)$, $v \in V$.

1. **begin**
2. **for** $v \in V$ **do** $D[v] := A[s, v]$; $D[s] := 0$;
3. **for** $k := 1$ **to** $n - 2$ **do**
4. **for** $v \in V \setminus \{s\}$ **do**
5. **for** $u \in V$ **do** $D[u] := \min(D[v], D[u] + A[u, v])$
6. **end**

Докажем правильность этого алгоритма. Для этого обозначим через $d^{(m)}(v)$ длину кратчайшего из путей из s в v , содержащих не более m дуг ($d^{(m)}(v) = \infty$, если не существует ни одного такого пути). Тогда имеем

$$d^{(m+1)}(v) = \min\{d^{(m)}(u) + a(u, v) : u \in V\}, \quad v \in V. \quad (3.3)$$

В самом деле, для каждого $u \in V$ очевидно имеем $d^{(m+1)}(v) \leq d^{(m)}(u) + a(u, v)$, причем равенство появляется, когда u является предпоследней вершиной произвольного кратчайшего пути из s в v .

Покажем теперь, что если при входе в очередную итерацию

$$d(s, v) \leq D[v] \leq d^{(m)}(v) \quad \forall v \in V, \quad (3.4)$$

то по окончании этой итерации

$$d(s, v) \leq D[v] \leq d^{(m+1)}(v) \quad \forall v \in V. \quad (3.5)$$

В самом деле, предполагая выполнение условия (3.4) и анализируя действие оператора в строке 5, мы видим, что по окончании нашей итерации цикла 3 имеем

$$d(s, v) \leq D[v] \leq \min \left\{ d^{(m)}(u) + a(u, v) : u \in V \right\},$$

что, принимая во внимание равенство (3.3), дает условие (3.5).

Отметим, что при входе в цикл 3 имеем $D[v] = d^1(v)$, $v \in V$, следовательно, после выполнения $n - 2$ итераций этого цикла будут выполняться неравенства $d(s, v) \leq D[v] \leq d^{(m+1)}(v)$, $v \in V$. Теперь достаточно показать, что $d^{(n-1)}(v) = d(s, v)$. Это справедливо, поскольку каждый путь более чем с $n - 1$ дугами содержит контур, устранение которого не увеличивает длины пути (ведь мы предполагаем отсутствие контуров отрицательной длины). Тем самым закончено доказательство корректности алгоритма. Очевидно, что временная сложность алгоритма есть $O(n^3)$. Мы можем, конечно, закончить вычисления, когда выполнение цикла 4 не вызывает изменения ни одной из переменных $D[v]$, $v \in V$. Это может наступить для $k < n - 2$, однако такая модификация алгоритма не изменяет существенным образом его сложности. Для редких графов ($m \ll n^2$) удобнее представлять граф списками инцидентности ПРЭДШ[u], $v \in V$. Заменяя строку 5 на

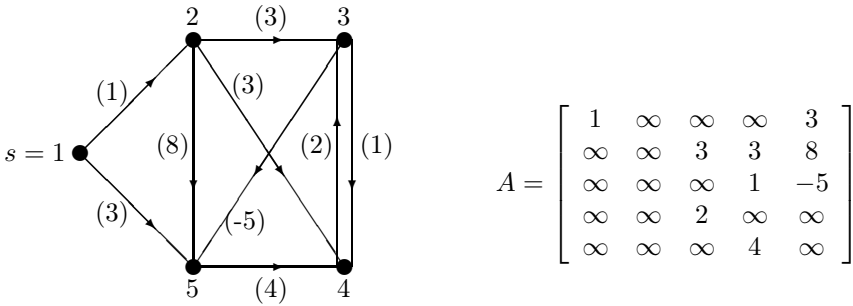
for $u \in$ ПРЭДШ[v] **do** $D[v] := \min(D[v], D[u] + A[a, v])$,

получаем алгоритм со сложностью $O(nm)$.

Работа алгоритма Форда–Беллмана проиллюстрирована на рис. 3.1 ($V = \{1, \dots, 5\}$, веса дуг даны числами в скобках, циклы 4 и 5 выполняются в порядке возрастания номеров вершин).

3.3 Случай неотрицательных весов — алгоритм Дейкстры

Известны более эффективные алгоритмы для двух важных случаев, а именно: когда веса всех дуг неотрицательны или когда граф бесконтурный. Сначала



k	D[1]	D[1]	D[2]	D[3]	D[4]
	0	1	∞	∞	3
1	0	1	4	4	-1
2	0	1	4	3	-1
3	0	1	4	3	-1

Рис. 3.1: Работа алгоритма Форда–Беллмана

опишем алгоритм для первого случая — алгоритм Дейкстры [11]. Вторым случаем займемся в следующем параграфе.

Алгоритм 3.3. (Нахождение расстояния от источника до всех остальных вершин в графе с неотрицательными весами дуг — метод Дейкстры.)

Данные: Ориентированный граф $\langle V, E \rangle$ с выделенным источником $s \in V$, матрица весов дуг $A[u, v]$, $u, v \in V$ (все веса неотрицательны).

Результаты: Расстояния от источника до всех вершин графа $D[v] = d(s, v)$, $v \in V$.

```

1. begin
2.   for  $v \in V$  do  $D[v] := A[s, v]$ ;
3.    $D[s] := 0$ ;  $T := V \setminus \{s\}$ ;
4.   while  $T \neq \emptyset$  do
5.     begin
6.        $u :=$  произвольная вершина  $r \in T$ , такая, что
            $D[r] = \min\{D[p] : p \in T\}$ ;
7.        $T := T \setminus \{u\}$ ;
8.       for  $v \in T$  do  $D[v] := \min(D[v], D[u] + A[u, v])$ 
9.     end
10. end

```

Чтобы понять действие алгоритма, покажем, что следующее условие явля-

ется инвариантом цикла 4:

$$\forall v \in V \setminus T \quad D[v] = d(s, v),$$

$$\forall v \in T \quad D[v] = \text{длине кратчайшего из тех путей из } s \text{ в } v, \quad (3.6)$$

для которых предпоследняя вершина принадлежит множеству $V \setminus T$.

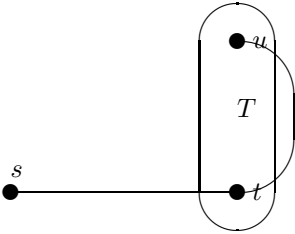


Рис. 3.2: Обоснование алгоритма Дейкстры

В самом деле, в строке 5 мы находим вершину $u \in T$, такую что значение $D[u]$ является минимальным (из всех) значением $D[t]$, для $t \in T$. Покажем, что $D[u] = d(s, u)$. Это именно так, потому что если кратчайший путь из s в u имеет длину меньше $D[u]$, то в силу второй части условия (3.6) его предпоследняя вершина принадлежит множеству T . Пусть t будет первой вершиной пути, принадлежащей множеству T (см. рис. 3.2). Начальный отрезок нашего пути из s в t составляет кратчайший путь из s в t , причем его предпоследняя вершина не принадлежит множеству T . По второй части условия (3.6) имеем $D[t] = d(s, t)$. Используя предположение о неотрицательности весов, получаем

$$D[t] = d(s, t) \leq d(s, u) < D[u]$$

вопреки принципу, по которому была выбрана вершина u .

Таким образом, $D[u] = d(s, u)$ и мы можем в строке 7 удалить u из множества T , не нарушая первой части условия (3.3). Чтобы обеспечить выполнение также и второй части этого условия, следует еще проверить пути из s в $v \in T$, предпоследняя вершина в которых есть u , и выполнить актуализацию переменных $D[v]$, $v \in T$. Именно это выполняет цикл 8.

Очевидно, что условие (3.6) выполняется при входе в цикл 4. По окончании действия алгоритма $T = \emptyset$, а следовательно, согласно условию (3.6), $D[u] = d(s, v)$, $v \in V$.

Перейдем теперь к оценке сложности алгоритма Дейкстры. Цикл 4 выполняется $n - 1$ раз, причем каждое его выполнение требует $O(n)$ шагов: $O(n)$ шагов для нахождения вершины u в строке 6 (предполагаем, что множество T представлено списком) и $O(n)$ шагов для выполнения цикла 8. Таким образом, сложность алгоритма есть $O(n^2)$.

Тщательно подбирая структуры данных, можно получить вариант алгоритма со сложностью $O(m \log n)$. Для этого множество T нужно представить бинарным деревом с высотой $O(\log n)$ и с таким свойством, что для произвольных

его вершин u и v

$$\text{если } u \text{ — сын } v, \text{ то } D[u] \leq D[v] \quad (3.7)$$

(подобная структура данных используется в алгоритме сортировки Heapsort, см. [21], [1]). Вершина u , для которой значение $D[u]$ минимально, является тогда корнем дерева. Этот корень можно устранить за $O(\log n)$ шагов, сохраняя свойство уменьшения значения $D[j]$ на каждом пути до корня. Достаточно сменить на место корня его сына s с большим (или равным) значением $D[j]$, затем на освободившееся место передвинуть сына вершины s с большим значением $D[j]$ и т.д. Если граф представлен списками ЗАПИСЬ[u], $u \in V$, то строку 8 можно заменить на

```

for  $v \in$  ЗАПИСЬ[ $u$ ] do
  if  $D[u] + A[u, v] < D[v]$  then
    begin
       $D[v] := D[u] + A[u, v]$ ;
      передвинуть вершину в дереве в направлении корня так,
      чтобы сохранить условие (3.7)
    end

```

Если предположить существование таблицы указателей на вершины нашего дерева, то передвижение вершины v , о которой идет речь в данной части раздела, может быть осуществлено за $O(\log n)$ шагов. Достаточно заменять v поочередно вершинами, находящимися непосредственно над ней.

В алгоритме, модифицированном таким способом, каждая дуга графа анализируется в точности один раз, причем с этим связано $O(\log n)$ шагов на передвижение соответствующей вершины в дереве, представляющем множество T . Это дает в сумме $O(m \log n)$ шагов. Сюда нужно добавить $O(n \log n)$ шагов, необходимых для построения нашего дерева и для устранения $n - 1$ раз из него корня. Общая сложность алгоритма есть $O(m \log n)$ (см. задачу 3.6).

Неизвестно, существует ли алгоритм сложности $O(m)$ нахождения расстояния от фиксированной вершины до всех остальных вершин графа с неотрицательными весами всех дуг. Можно показать, что существует константа C , такая что эта задача для произвольного $k > 0$ может быть решена за время $Ck(m + n^{1+1/k})$ (см. [37]).

Работа алгоритма Дейкстры проиллюстрирована на рис. 3.3 ($V = \{1, \dots, 6\}$, веса дуг даны в скобках, значения $D[v]$, $v \in T$, обозначены кружками, минимальные значения — двойными кружками).

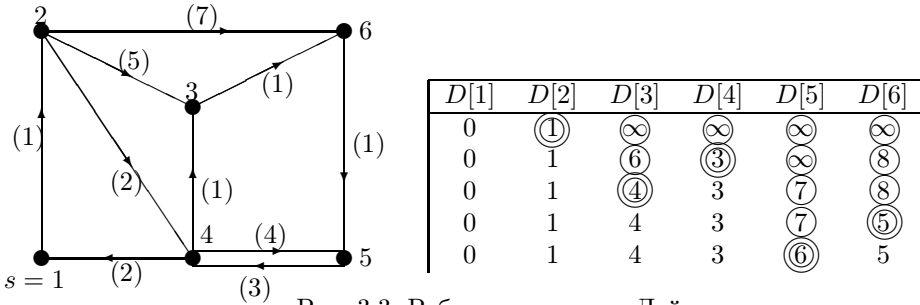


Рис. 3.3: Работа алгоритма Дейкстры

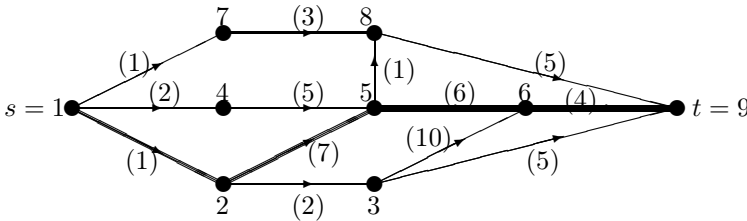


Рис. 3.4: Самый длинный путь в бесконтурном графе

3.4 Пути в бесконтурном орграфе

Займемся теперь вторым случаем, для которого известен алгоритм нахождения расстояний от фиксированной вершины за время $O(n^2)$, а именно случае, когда граф является бесконтурным (веса дуг могут быть произвольными). Сначала докажем следующую лемму.

Лемма 3.4. *В произвольном бесконтурном графе вершины можно перенумеровать так, что каждая дуга будет иметь вид $\langle v_i, v_j \rangle$, где $i < j$.*

Пример такой нумерации приведен на рис. 3.4.

Для доказательства предложим алгоритм, который конструирует такую нумерацию.

Алгоритм 3.5. (Нумерация вершин бесконтурного графа.)

Данные: Ориентированный бесконтурный граф $\langle V, E \rangle$, определяемый списками инцидентности ЗАПИСЬ[v], $v \in V$.

Результаты: Для каждой вершины $v \in V$ номер $NR[v]$, такой что для произвольной дуги $\langle u, v \rangle \in E$ выполняется неравенство $NR[u] < NR[v]$.

1. begin


```

2.   for  $v \in V$  do ЧЗАХ[ $v$ ] := 0; (*ЧЗАХ[ $v$ ]=число дуг, заходящих в  $v$ *)
3.   for  $u \in V$  do
4.       for  $v \in$  ЗАПИСЬ[ $u$ ] do ЧЗАХ[ $u$ ] := ЧЗАХ[ $u$ ] + 1;
5.   СТЕК :=  $\emptyset$ ;
6.   for  $v \in V$  do
7.       if ЧЗАХ[ $v$ ] = 0 then СТЕК  $\leftarrow v$ ;
8.   num := 0;
9.   while СТЕК  $\neq \emptyset$  do
10.      begin  $u \leftarrow$  СТЕК;
11.          num := num + 1; NR[ $u$ ] := num;
12.          for  $v \in$  ЗАПИСЬ[ $u$ ] do
13.              begin ЧЗАХ[ $v$ ] := ЧЗАХ[ $v$ ] - 1;
14.                  if ЧЗАХ[ $v$ ] = 0 then СТЕК  $\leftarrow v$ 
15.              end
16.          end
17. end

```

Алгоритм основывается на следующем простом факте: в произвольном (непустом) бесконтурном графе существует вершина, в которую не заходит ни одна дуга. Чтобы убедиться в этом, выберем произвольную вершину w_1 графа, затем некоторую вершину w_2 , такую что $w_2 \rightarrow w_1$, затем вершину w_3 , такую что $w_3 \rightarrow w_2$, и т.д. Через конечное число шагов мы должны прийти до некоторой вершины w_i , в которую не заходит ни одна дуга, ибо в силу бесконтурности ни одна вершина не может повторяться в последовательности w_1, w_2, \dots . В нашем алгоритме вершины, в которые не заходит ни одна дуга, накапливаются в стеке. В строке 10 выбирается верхний элемент стека u (это мог бы быть произвольный элемент стека), и этой вершине присваивается самый маленький из еще неиспользованных номеров. Таким образом, мы гарантируем, то что все дуги, выходящие из этой вершины, ведут к вершинам с большими номерами. Затем вершина u вместе с выходящими из нее дугами удаляется из графа. Это приводит к уменьшению на единицу числа дуг, заходящих в каждую вершину v , такую что $u \rightarrow v$, это число запоминается в ЧЗАХ[v]. Если для какой-нибудь из вершин v это число сводится к нулю, то v помещается в стек. В силу бесконтурности графа и наших предыдущих замечаний полное опустошение стека, вызывающее окончание выполнения алгоритма (см. цикл 9), наступает не раньше, чем после присвоения номеров всем вершинам графа. Легко заметить, что каждая дуга анализируется алгоритмом один раз в строке 4 и один раз в строке 12. Таким образом, сложность алгоритма есть $O(m)$ (вспомним, что остается в силе предположение $m = \Omega(n)$), в противном случае следовало бы написать $O(m + n)$.

Согласно алгоритму 3.5 при описании алгоритма нахождения путей в бес-

контурном графе мы можем предположить, что каждая дуга идет из вершины с меньшим номером в вершину с большим номером.

Алгоритм 3.6. (Нахождение расстояний от источника до всех вершин в бесконтурном графе.)

Данные: Ориентированный граф $\langle V, E \rangle$, где $V = \{v_1, \dots, v_n\}$, и для произвольной дуги $\langle v_i, v_j \rangle \in E$ имеем $i < j$. Этот граф определен списками инцидентности ПРЕДШ[u], $v \in V$.

Результаты: Расстояния от v_1 до всех вершин графа: $D[v_i] = d(v_1, v_i)$, $i = 1, \dots, n$.

```

1. begin
2.    $D[v_1] := 0$ ;
3.   for  $j := 2$  to  $n$  do  $D[v_j] := \infty$ ;
4.   for  $j := 2$  to  $n$  do
5.     for  $v_i \in \text{ПРЕДШ}[v_j]$  do  $D[v_j] := \min(D[v_j], D[v_i] + A[v_i, v_j])$ 
6. end

```

Нетрудно доказать индукцией по j , что после выполнения цикла 4 для некоторого значения j выполняется равенство $D[v_i] = d(v_1, v_i)$, $i = 1, \dots, j$. Достаточно воспользоваться тем фактом, что все промежуточные вершины кратчайшего пути из v_1 в v_j имеют номера меньше j .

Сложность алгоритма есть $O(m)$, так как каждая дуга $\langle v_i, v_j \rangle$ анализируется в строке 5 в точности один раз.

Алгоритмы 3.5 и 3.6 могут найти применение в методах управления выполнением проекта, называемых *PERT* (Project Evaluation and Review Technique) или *CPM* (Critical Path Method). Эти методы основываются на построении графа (*сети PERT* или *сети CPM*), дуги которого соответствуют некоторым элементарным задачам, составляющим проект, а их веса указывают на время, необходимое для решения отдельных задач. Кроме этого, мы предполагаем, что для произвольных дуг этого графа $\langle u, v \rangle$ и $\langle v, t \rangle$ задача, изображаемая дугой $\langle u, v \rangle$, должна быть закончена перед началом решения задачи, изображаемой дугой $\langle v, t \rangle$. Легко заметить, что такой граф должен быть бесконтурным. Нашей задачей является нахождение самого длинного пути из вершины s , соответствующей началу проекта, до вершины t , соответствующей его окончанию (см. рис. 3.4). Такой путь называется *критическим путем*. Его длина определяет время, необходимое для реализации всего проекта. Очевидно, что эта задача сводится к задаче о кратчайшем пути путем изменения знака каждого веса $a(u, v)$, где $u \rightarrow v$, на обратный.

3.5 Кратчайшие пути между всеми парами вершин, транзитивное замыкание отношения

Очевидно, что задачу определения расстояния между всеми парами вершин можно решить, используя n раз (поочередно для каждой вершины) один из описанных ранее методов нахождения расстояний от фиксированной вершины. Таким образом, мы получаем алгоритм со сложностью $O(n^4)$ (при использовании метода Форда–Беллмана) или $O(n^3)$ для бесконтурных графов или неотрицательных весов. Однако оказывается, что в общем случае n -кратное использование метода Форда–Беллмана не является наилучшим методом. Покажем теперь два более эффективных метода.

Для этого рассмотрим ориентированный граф $G = \langle V, E \rangle$, где $V = \{v_1, \dots, v_n\}$, и предположим, что $A = [a_{ij}]$ есть матрица весов ($a_{ij} = a(v_i, v_j)$). Обозначив через $d_{ij}^{(m)}$ длину кратчайшего пути из v_i и v_j , содержащего не более m дуг, получаем следующие очевидные уравнения (ср. с (3.3)):

$$d_{ij}^{(0)} = \begin{cases} 0, & \text{если } i = j \\ \infty, & \text{если } i \neq j \end{cases} \quad (3.8)$$

$$d_{ij}^{(m+1)} = \min \left\{ d_{ik}^{(m)} + a_{kj} : 1 \leq k \leq n \right\}. \quad (3.9)$$

Отметим, что уравнение (3.9) обнаруживает некоторое сходство с определением произведения двух квадратных матриц. Если операцию \min трактовать как «сумму», операцию «+» — как «произведение», то матрица $[d_{ij}^{(m+1)}]$ является «произведением» матриц $[d_{ij}^{(m)}]$ и $A = [a_{ij}]$. Обозначим такое «произведение» двух матриц A и B через $A * B$ и отметим, что для этой операции единичным элементом служит матрица

$$U = \begin{bmatrix} 0 & \infty & \infty & \cdots & \infty \\ \infty & 0 & \infty & \cdots & \infty \\ \infty & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \vdots & & \vdots \\ \infty & \infty & \infty & \cdots & 0 \end{bmatrix}$$

Из уравнений (3.8) и (3.9) теперь легко следует, что $[d_{ij}^{(0)}] = U$ и

$$d_{ij}^{(m)} = \underbrace{((\dots ((A * A) * A) \dots) * A)}_{m \text{ раз}} \quad (m \geq 1) \quad (3.10)$$

Возможен один из двух следующих случаев:

1. $d_{ij}^{(n-1)} = d_{ij}^{(n)}$ и в результате $d_{ij}^{(m)} = d_{ij}^{(n-1)}$ для каждого $m \geq n$. Тогда очевидно $d_{ij}^{(n-1)} = d(v_i, v_j)$.
2. $d_{ij}^{(n-1)} \neq d_{ij}^{(n)}$. Это означает, что существует контур отрицательной длины.

Произведение $A * B$ двух матриц размерности $n \times n$ можно вычислить за время $O(n^3)$ (n сложений и $n - 1$ сравнений на каждый из n^2 элементов произведения $A * B$). Следовательно, матрицу $[d_{ij}^{(n-1)}]$ и тем самым расстояние между всеми парами вершин можно вычислить за время $O(n^4)$.

Пока сложность этого алгоритма такая же, как и для случая n -кратного использования алгоритма Форда–Беллмана. Однако мы можем ее снизить, если заметим, что операция «*» ассоциативна (т.е. $(A * B) * C = A * (B * C)$, см. задачу 3.8). Этот факт позволяет вычислять произведение (3.10), поочередно возводя матрицу A в квадрат и тем самым заменяя $n - 1$ умножение матрицы $\lceil \log n \rceil$ умножениями. Таким образом, мы получаем алгоритм сложности $O(n^3 \log n)$, отыскивающий расстояния между всеми парами вершин в графе без контуров отрицательной длины.

Авторами еще более эффективного алгоритма являются Уоршалл [70] и Флойд [20]. Идея этого алгоритма следующая. Обозначим через $d_{ij}^{(m)}$ длину кратчайшего из путей из v_i в v_j с промежуточными вершинами в множестве $\{v_1, \dots, v_m\}$. Тогда имеем следующие уравнения:

$$d_{ij}^0 = a_{ij}, \quad (3.11)$$

$$d_{ij}^{(m+1)} = \min \left(d_{ij}^{(m)}, d_{im}^{(m)} + d_{mj}^{(m)} \right). \quad (3.12)$$

Обоснование второго уравнения достаточно простое. Рассмотрим кратчайший путь из v_i в v_j с промежуточными вершинами из множества $\{v_1, \dots, v_m, v_{m+1}\}$. Если этот путь не содержит v_{m+1} , то $d_{ij}^{(m+1)} = d_{ij}^{(m)}$. Если же он содержит v_{m+1} , то, деля путь на отрезки от v_i до v_{m+1} и от v_{m+1} до v_j , получаем равенство $d_{ij}^{(m+1)} = d_{im}^{(m)} + d_{mj}^{(m)}$.

Уравнения (3.11) и (3.12) дают возможность легко вычислить расстояния $d(v_i, v_j) = d_{ij}^{(n)}$, $1 \leq i, j \leq n$.

Алгоритм 3.7. (Вычисление расстояний между всеми парами вершин — метод Флойда.)

Данные: Матрица весов дуг $A[i, j]$, $1 \leq i, j \leq n$ ориентированного графа без контуров отрицательной длины.

Результаты: Расстояния между всеми парами вершин $D[i, j] = d(v_i, v_j)$.

```

1. begin
2.   for  $i := 1$  to  $n$  do
3.     for  $j := 1$  to  $n$  do  $D[i, j] := A[i, j]$ ;
4.   for  $i := 1$  to  $n$  do  $D[i, i] := 0$ ;
5.   for  $m := 1$  to  $n$  do
6.     for  $i := 1$  to  $n$  do
7.       for  $j := 1$  to  $n$  do
8.          $D[i, j] := \min(D[i, j], D[i, m] + D[m, j])$ 
9. end

```

Обоснование корректности алгоритма предоставляем читателю.

Очевидно, что сложность этого алгоритма есть $O(n^3)$. Стоит отметить, что такую же сложность имел алгоритм Форда–Беллмана нахождения расстояний от фиксированной вершины до всех остальных вершин. Любопытно, что для общего случая (т.е. без предположения о неотрицательности весов либо о бесконечности графа) не известен ни один алгоритм нахождения расстояния между одной фиксированной парой вершин, который был бы значительно эффективнее алгоритма нахождения расстояний между всеми парами вершин.

С задачей определения кратчайших путей в графе тесно связана задача транзитивного замыкания бинарного отношения. Вспомним, что под бинарным отношением на множестве V мы понимаем произвольное подмножество $E \subseteq V \times V$. Такое отношение является транзитивным, если удовлетворяется условие

$$(x, y) \in E \wedge (y, z) \in E \Rightarrow (x, z) \in E$$

$\forall x, y, z \in E$. Отметим, что бинарное отношение $E \subseteq V \times V$ можно однозначно представить ориентированным графом $G = \langle V, E \rangle$. Для произвольного такого отношения мы определяем

$$E^* = \{(x, y) : \text{в } \langle V, E \rangle \text{ существует путь ненулевой длины из } x \text{ в } y\}.$$

Нетрудно заметить, что E^* — транзитивное отношение на множестве V и $E \subseteq E^*$. Более того, E^* является наименьшим транзитивным отношением, содержащим E , т.е. для произвольного транзитивного отношения $F \supseteq E$ выполняется включение $E^* \subseteq F$. Отношение E^* называется *транзитивным замыканием* отношения E .

Если отношение E представить в виде графа $\langle V, E \rangle$, в котором каждая дуга имеет вес 1, то транзитивное замыкание E^* можно вычислить с помощью алгоритма 3.7 за время $O(n^3)$; после завершения работы алгоритма имеем

$$\langle v_i, v_j \rangle \in E^* \Leftrightarrow D[i, j] < \infty$$

При вычислении транзитивного замыкания удобно принять

$$A[i, j] = \begin{cases} 0, & \text{если } \langle v_i, v_j \rangle \notin E \\ 1, & \text{если } \langle v_i, v_j \rangle \in E \end{cases} \quad (\text{вместо } \infty) \quad (3.13)$$

Тогда строку 7 алгоритма 3.7 мы можем заменить на

$$D[i, j] := D[i, j] \vee (D[i, m] \wedge D[m, j]),$$

где \vee и \wedge — обычные булевы операции:

$$\begin{array}{c|cc} \vee & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 1 \end{array} \quad \begin{array}{c|cc} \wedge & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

После завершения работы алгоритма, модифицированного таким образом (принадлежащего Уоршаллу [70]), очевидно, имеем

$$D[i, j] = \begin{cases} 1, & \text{если } \langle v_i, v_j \rangle \in E^*, \\ 0, & \text{если } \langle v_i, v_j \rangle \notin E^*. \end{cases}$$

Здесь следует отметить, что известен алгоритм построения транзитивного замыкания, более эффективный, чем алгоритм Уоршалла (см. [53]). Он использует связь этой задачи с умножением матриц, обсуждавшуюся в начале этого параграфа, эта связь для матриц A , определяемых равенством (3.13), приводит к обычному умножению булевых матриц по формуле

$$c_{ij} = \bigvee_{k=1}^n (a_{ik} \wedge b_{kj})$$

Оказывается (см. [64]), что такое умножение можно выполнить за время $O(n^{\log 7})^1$, что дает алгоритм построения транзитивного замыкания, имеющего сложность $O(n^{\log 7} \log n)$. Такой алгоритм имеет скорее теоретическое значение, поскольку метод умножения матриц за время $O(n^{\log 7})$ является довольно сложным и, следовательно, обнаруживает свое преимущество перед обычным «школьным» методом только при очень больших значениях n . На практике обычно самым эффективным оказывается алгоритм Уоршалла, соответствующим образом запрограммированный (см. [65] и задачу 3.9).

Другим способом построения транзитивного замыкания отношения E является применение поиска в глубину (или в ширину) в графе $\langle V, E \rangle$. Этот способ особенно выгоден, когда отношение E симметрично; очевидно, что тогда транзитивное замыкание строится отдельно для каждой связной компоненты, на которые разбивается неориентированный граф, определяемый отношением E , и это построение может быть получено за время $O(m + n)$. Детали оставляем читателю (см. задачу 3.12).

¹Показатель $\log 7$ может быть немного уменьшен (см. [55])

3.6 Задачи

3.1. Пусть $G = \langle V, E \rangle$ — произвольный ориентированный граф. Зафиксируем некоторую вершину $s \in V$, добавим новую вершину t , заменим каждую дугу $\langle v, s \rangle \in E$ на дугу $\langle v, t \rangle$ и поставим в соответствие каждой дуге полученного графа вес -1 . Доказать, что кратчайший элементарный путь из s в t имеет длину $-n$ тогда и только тогда, когда в G существует гамильтонов контур.

3.2. Модифицировать алгоритм 3.1 так, чтобы он определял

- (а) кратчайший элементарный путь из s в t ;
- (б) все кратчайшие пути из s в t в предположении, что граф не содержит контуров отрицательной длины (может содержать контуры нулевой длины).

Провести анализ предложенных алгоритмов.

3.3. Изменить алгоритмы 3.2, 3.3 и 3.6 так, чтобы они определяли кратчайшие пути из s до остальных вершин, не используя алгоритм 3.1. (*Указание:* для каждой вершины $v \in V$ следует запоминать и активизировать во время работы алгоритма такую вершину $PRE[v]$, что

$$D[v] = D[PRE[v]] + A[PRE[v], v].)$$

3.4. Можно ли условия (3.4) и (3.5) заменить равенствами

$$D[v] = d(m)(v), \quad D[v] = d(m+1)(v) \quad ?$$

3.5. Доказать, что в графе без контуров с неположительной длиной уравнения

$$\begin{aligned} d(s, s) &= 0, \\ d(s, v) &= \min \{d(s, u) + a(u, v) : u \in V \setminus \{v\}\} \end{aligned}$$

однозначно определяют расстояния $d(s, v)$, $v \in V$. Дать пример графа с циклом (контуром) длины 0, для которого эта система уравнений имеет бесконечно много решений.

3.6. Дополнить детали описания и анализа варианта алгоритма Дейкстры со сложностью $O(m \log n)$.

3.7. Определить кратчайший путь из s в t в графе на рис. 3.4.

3.8. Доказать, что операция «*», определенная для квадратных матриц $A = [a_{ij}]$ и $B = [b_{ij}]$ размерности $n \times n$ формулой $A * B = [c_{ij}]$, где $c_{ij} = \min \{a_{ik} + b_{kj} : 1 \leq k \leq n\}$, является ассоциативной.

3.9. Доказать, что следующая модификация алгоритма Уоршалла стоит транзитивное замыкание отношения [65]:

```

begin (*отношение  $E$  определено матрицей  $A$ , см. (3.13)*)
  for  $m := 1$  to  $n$  do
    for  $i := 1$  to  $n$  do
      if  $A[i, m] = 1$  then
        for  $j := 1$  to  $n$  do  $A[i, j] := A[i, j] \vee A[m, j]$ 
    end
  end (* $A[i, j] = 1 \Leftrightarrow \langle v_i, v_j \rangle \in E^*$ *)

```

Предположим, что строка матрицы A помещается в машинном слове и цикл 5 мы трактуем как одну элементарную операцию (логическое сложение m -й строки с 1-й). Какова тогда сложность алгоритма?

3.10. Модифицировать алгоритм Флойда так, чтобы кроме расстояний $D[i, j]$ он определял матрицу $M[i, j]$, где $M[i, j]$ есть наибольший номер вершины некоторого кратчайшего пути из v_i в v_j ($M[i, j] = 0$, если этот путь не содержит промежуточных вершин). Доказать, что рекурсивная процедура

```

procedure ВЫПИСАТЬПУТЬ( $i, j$ );
begin
  if  $M[i, j] = 0$  then if  $i = j$  then write( $i$ )
    else write( $i, j$ )
  else begin
    ВЫПИСАТЬПУТЬ( $i, M[i, j]$ );
    ВЫПИСАТЬПУТЬ( $M[1, j], j$ )
  end
end

```

выписывает кратчайший путь из v_i в v_j за время $O(n)$. Написать нерекурсивный вариант этой процедуры.

3.11. Под *пропускной способностью пути* будем понимать наименьший вес дуги этого пути. Написать алгоритм, определяющий наибольшие пропускные способности путей между

- (а) фиксированной парой вершин;
- (б) всеми парами вершин.

3.12. Опираясь на метод поиска в глубину, написать алгоритм, строящий транзитивное замыкание произвольного отношения $E \subseteq V \times V$ за время $O(n^2 + mn)$, и алгоритм сложности $O(n^2)$ для симметричных отношений.

Глава 4

Потоки в сетях и родственные задачи

4.1 Максимальный поток в сети

Под *сетью* мы будем понимать пару $S = \langle G, c \rangle$, где $G = \langle V, E \rangle$ — произвольный ориентированный граф, а $c : E \rightarrow \mathbb{R}$ — функция, которая каждой дуге $\langle v, u \rangle$ ставит в соответствие неотрицательное вещественное число $c(u, v)$, называемое *пропускной способностью* этой дуги. Множества V и E называются соответственно множеством *вершин* и множеством *дуг* сети S . Для произвольной функции

$$c : E \rightarrow \mathbb{R} \tag{4.1}$$

и произвольной вершины u сети S рассмотрим величину

$$\text{Div}_f(v) = \sum_{u:v \rightarrow u} f(u, v) - \sum_{u:u \rightarrow v} f(u, v)$$

Если $f(u, v)$ мы интерпретируем как поток из u в v , то величина $\text{Div}_f(v)$ определяет «количество потока», выходящего из вершины v . Эта величина может быть положительной (если из вершины v больше выходит, чем входит в нее), отрицательной (если в вершину входит больше, чем выходит из нее, т.е. наступает «накопление» потока в вершине u) и, наконец, равной нулю. Последний случай нас будет интересовать более всего. Выделим в нашей сети две вершины — источник s и сток t ($s \neq t$). Под *потоком* из s в t в сети S мы будем понимать произвольную функцию вида (4.1), для которой выполняются условия

$$0 \leq f(u, v) \leq c(u, v) \text{ для каждой дуги } (u, v) \in E \tag{4.2}$$

и

$$\operatorname{Div}_f(v) = 0 \text{ для каждой вершины } v \in V \setminus \{s, t\}. \quad (4.3)$$

Величину

$$W(f) = \operatorname{Div}_f(s)$$

будем называть *величиной потока* f .

Согласно описанному выше, мы имеем здесь дело с потоком, который не возникает и не накапливается ни в одной из вершин, отличных от s , t , и который удовлетворяет следующему условию: через дугу $\langle u, v \rangle$ мы можем пропустить не более чем $c(u, v)$ единиц потока. Такой поток может описывать поведение газа или жидкости в трубопроводе, потоки автомобилей в сети автострад, пересылку товаров по железной дороге (без хранения их на промежуточных станциях), передачу информации в информационной сети и т.д.

Мы будем интересоваться главным образом нахождением максимального потока (т.е. потока с максимальной величиной) в заданной сети. Докажем сначала несколько простых и интуитивно очевидных фактов. Под *разрезом* $P(A)$ сети S , соответствующим подмножеству $A \subseteq V$ ($A \neq \emptyset$, $A \neq V$), мы понимаем множество дуг $\langle u, v \rangle \in E$, таких что $u \in A$ и $v \in V \setminus A$, т.е.

$$P(A) = E \cap (A \times (V \setminus A)).$$

Для произвольного потока f в сети S поток через разрез $P(A)$ определяется естественным образом:

$$f(A, V \setminus A) = \sum_{e \in P(A)} f(e)$$

Лемма 4.1. Если $s \in A$ и $t \in V \setminus A$, то для произвольного потока f из s в t

$$W(f) = f(A, V \setminus A) - f(V \setminus A, A). \quad (4.4)$$

Доказательство. Суммируем уравнения $\operatorname{Div}_f(v) = 0$ (4.3) для всех $v \in A$. Эта сумма складывается из некоторого числа слагаемых $f(u, v)$, снабженных знаком плюс и минус, причем хотя бы одна из вершин u, v принадлежит множеству A . Если обе вершины принадлежат A , то $f(u, v)$ появляется со знаком плюс в $\operatorname{Div}_f(u)$ и со знаком минус в $\operatorname{Div}_f(v)$ (и не появляется в выражении $\operatorname{Div}_f(r)$ ни для одного r , отличного от u и v), что в сумме дает 0. Каждое из слагаемых $f(u, v)$, $u \in V \setminus A$, $v \in A$ появляется в точности один раз со знаком плюс в $\operatorname{Div}_f(u)$, что в сумме дает $f(A, V \setminus A)$. Аналогичные слагаемые $f(u, v)$, $u \in V \setminus A$, $v \in A$ отвечают за слагаемое $f(V \setminus A, A)$ в (4.4). С другой стороны наша сумма равна $\operatorname{Div}_f(s) = W(f)$, ибо $\operatorname{Div}_f(u) = 0$ для каждого $u \in A \setminus \{s\}$.

Принимая $A = V \setminus \{t\}$, получаем в этом частном случае из леммы 4.1

$$\begin{aligned} \operatorname{Div}_f(s) = W(f) &= f(V \setminus \{t\}, \{t\}) - f(\{t\}, V \setminus \{t\}) = \\ &= -(f(V \setminus \{t\}, \{t\}) - f(\{t\}, V \setminus \{t\})) = -\operatorname{Div}_f(f), \end{aligned}$$

что выражает интуитивно понятный факт: в сток входит в точности такое количество потока, какое выходит из источника. В общем виде лемма 4.1 говорит, что общее количество потока можно измерить в произвольном разрезе, отделяющем s от t .

Определим *пропускную способность* разреза $P(A)$ следующим образом:

$$c(A, V \setminus A) = \sum_{e \in P(A)} c(e).$$

Под *минимальным разрезом*, разделяющим s и t , мы будем понимать произвольный разрез $P(A)$, $s \in A$, $t \in V \setminus A$ с минимальной пропускной способностью.

Одним из фундаментальных фактов теории потоков в сетях является классическая теорема о максимальном потоке и минимальном разрезе.

Теорема 4.2 (Форд и Фалкерсон [23]). *Величина каждого потока из s в t не превосходит пропускной способности минимального разреза, разделяющего s и t , причем существует поток, достигающий этого значения.*

Доказательство. Пусть $P(A)$ — минимальный разрез. В силу леммы 4.1 для произвольного потока f имеем

$$\begin{aligned} W(f) &= f(A, V \setminus A) - f(V \setminus A, A) \leq f(A, V \setminus A) = \\ &= \sum_{e \in P(A)} c(e) = c(A, V \setminus A). \end{aligned} \quad (4.5)$$

Существование потока, для которого указанное неравенство переходит в равенство (такой поток, очевидно, максимален), — более глубокий факт. Мы получаем его как следствие анализа алгоритма, представленного в следующем разделе.

Все известные алгоритмы построения максимального потока основываются на последовательном увеличении потока, причем модификация потока, увеличивающая его величину, чаще всего опирается на *метод увеличивающих цепей*.

Будем говорить, что дуга e сети S является *допустимой дугой* из u в v относительно потока f , если

$$e = \langle u, v \rangle \text{ и } f(e) < c(e) \quad (4.6)$$

или

$$e = \langle v, u \rangle \text{ и } f(e) > 0. \quad (4.7)$$

В зависимости от того, какое из приведенных условий имеет место, первое или второе, будем говорить соответственно о *согласованной* или *несогласованной* допустимой дуге из u в v . *Увеличивающей цепью* (длины l) для данного потока f из s в t называется произвольная знакопеременная последовательность (попарно различных) вершин и дуг

$$v_0, e_1, v_1, e_2, v_2, \dots, v_{l-1}, e_l, v_l, \quad (4.8)$$

такая что $v_0 = s$, $v_l = t$, и для каждого $i \leq l$ дуга e_i допустима из v_{i-1} в v_i относительно потока f . Знание увеличивающей цепи вида (4.8) позволяет легко увеличить величину потока f на

$$\delta = \min \{ \Delta(e_i) : 1 \leq i \leq l \},$$

где

$$\Delta(e_i) = \begin{cases} c(e_i) - f(e_i), & \text{если дуга } e_i \text{ согласована,} \\ f(e_i), & \text{если дуга } e_i \text{ не согласована.} \end{cases}$$

Для этого достаточно увеличить на δ поток по каждой согласованной дуге e_i :

$$f'(e_i) = f(e_i) + \delta,$$

а также уменьшить на δ поток по каждой несогласованной дуге e_i :

$$f'(e_i) = f(e_i) - \delta.$$

Определенная таким образом функция f' (мы полагаем $f'(e) = f(e)$ для дуг e , не принадлежащих цепи (4.8)) является потоком. Действительно, изменение величины $\text{Div}_f(v_i)$, $1 \leq i \leq l-1$, основывается на ее увеличении на δ , связанном с изменением потока по e_{i+1} , и уменьшении ее на δ , связанном с изменением потока по e_i — это не зависит от типа дуги e_i , e_{i+1} . В сумме эти изменения компенсируются и в результате $\text{Div}'_f(v_i) = \text{Div}_f(v_i) = 0$ ($1 \leq i \leq l-1$). Выполняется также условие $0 \leq f'(e) \leq c(e)$ для каждой дуги e . Величина же потока увеличивается на δ :

$$W(f') = \text{Div}'_f(s) = \text{Div}_f(s) + \delta = W(f) + \delta.$$

На рис. 4.1 показан процесс увеличения потока вдоль увеличивающейся цепи. Поток $f(e)$ по каждой дуге e указан возле соответствующих дуг, в скобках указана пропускная способность дуг $c(e)$. Увеличивающая цепь имеет вид

$$v_1, \langle v_1, v_3 \rangle, v_3, \langle v_2, v_3 \rangle, v_2, \langle v_5, v_2 \rangle, v_5, \langle v_5, v_6 \rangle, v_6, \langle v_7, v_6 \rangle, v_7, \langle v_7, v_8 \rangle, v_8,$$

что дает увеличение потока на $\delta = \Delta(v_4, v_5) = 1$ с 10 до 11.

Теорема 4.3. *Следующие три условия эквивалентны:*

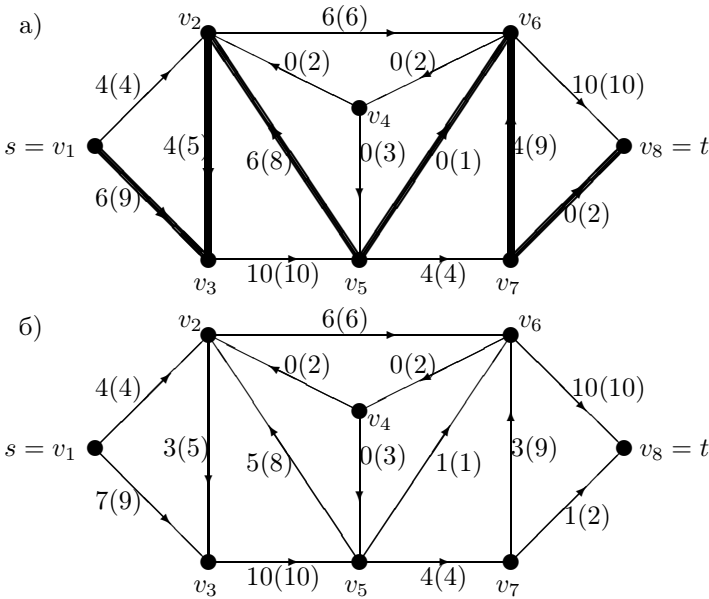


Рис. 4.1: Увеличение потока вдоль увеличивающей цепи: а) перед модификацией, б) после модификации.

- (а) Поток из s в t максимален.
- (б) Не существует увеличивающей цепи для f .
- (в) $W(f) = c(A, V \setminus A)$ для некоторого $A \subseteq V$, такого что $s \in A$, $t \notin A$.

Доказательство. (а) \Rightarrow (б). Если поток максимален, то очевидно, что для него не существует увеличивающей цепи, ибо существование такой цепи делало бы возможным увеличение потока.

(б) \Rightarrow (в) Предположим, что для некоторого потока f не существует увеличивающей цепи. Определим множество $A \subseteq V$ как множество вершин v , для которых существует цепь

$$v_0, e_1, v_1, e_2, v_2, \dots, v_k - 1, e_k, v_k$$

такая что $k \geq 0$, $v_0 = s$, $v_k = v$ и e_i — дуга, допустимая из v_{i-1} в v_i , $1 \leq i \leq k$. Очевидно, что $s \in A$, а $t \notin A$, поскольку это означало бы существование увеличивающей цепи для f . Рассмотрим разрез $P(A)$. Для каждой дуги $e \in P(A)$

должно выполняться равенство $f(e) = c(e)$ согласно определению множества A (см. условие (4.6)), таким образом, $f(A, V \setminus A) = c(A, V \setminus A)$. Пользуясь определением множества A , аналогичным образом получаем $f(V \setminus A, A) = 0$ (см. условие (4.7)). В силу леммы 4.1 отсюда вытекает, что

$$w(f) = f(A, V \setminus A) - f(V \setminus A, A) = c(A, V \setminus A).$$

(в) \Rightarrow (а) следует из уже доказанного факта, что величина произвольного потока не превосходит $c(A, V \setminus A)$ (см. (4.5)).

Нетрудно проверить, что поток на рис. 4.1, б — максимальный. «Насыщенный» разрез $P(A)$, появляющийся в доказательстве теоремы 4.3, обозначен через $A = \{v_1, v_2, v_3, v_5\}$. Для того чтобы из теоремы 4.3 вывести теорему о максимальном потоке и минимальном разрезе, нам еще не хватает одной, довольно тонкой детали. (Прежде чем читать дальше, предлагаем читателю догадаться, какой именно.) Этим недостающим фактом является просто существование максимального потока в произвольной сети. Другими словами, нужно показать, что невозможна ситуация, в которой, например, для каждого $\varepsilon > 0$ существует поток с величиной, превосходящей $7 - \varepsilon$, но не существует потока с величиной, равной 7 (стоит отметить, что подобной проблемы для минимального разреза не существует — число различных разрезов в отличие от числа различных потоков является конечным). Этот простой факт, являющийся в сущности следствием нестрогости неравенства в условии (4.2), можно доказать многими способами. Мы докажем его в следующем разделе путем анализа некоторого алгоритма построения максимального потока.

4.2 Алгоритм построения максимального потока

Рассуждения предыдущего раздела подсказывают идею следующего простого алгоритма построения максимального потока. Начиная с произвольного потока (например, $f(e) = 0$ для каждой дуги $e \in E$), мы ищем увеличивающие цепи и увеличиваем вдоль них фактический поток.

Однако здесь возникают два вопроса. Во-первых, закончится ли работа такого алгоритма через конечное число шагов и, во-вторых, будет ли полученный поток максимальным. Ответ на второй вопрос в предположении, что алгоритм остановится через конечное число шагов из-за невозможности отыскать новые увеличивающие цепи, будет утвердительным, что непосредственно следует из теоремы 4.3. С другой стороны, Форд и Фалкерсон в работе [23] дали в некотором смысле отрицательный ответ на первый вопрос. А именно, они привели пример сети, в которой можно так «злоумышленно» подбирать очередные рассматриваемые увеличивающие цепи, что процесс никогда не кончится; более

того, величина потока в течение всего времени будет меньше одной четверти величины максимального потока.

Эдмондс и Карп (см. [17]) показали, что если мы увеличиваем фактический поток всегда вдоль кратчайшей увеличивающей цепи, то максимальный поток строится с использованием не более $mn/2$ цепей (как обычно, $n = |V|$) и $m = |E|$). Отыскание кратчайшей цепи легко реализовать при помощи алгоритма, аналогичного поиску в ширину (см. разд. 2.3). Точнее говоря, начиная от источника s , мы ведем поиск в ширину в графе G_f , состоящем из дуг $\langle u, v \rangle$, таких что в нашей сети существует дуга, допустимая от u до v относительно фактического потока f , вплоть до момента достижения стока t . Очевидно, что найденный в результате этого процесса кратчайший путь из s в t соответствует кратчайшей увеличивающей цепи из s в t в исходной сети. Принимая во внимание, что процесс поиска в ширину мы можем выполнить за время $O(m + n)$, мы можем оценить сложность всего алгоритма построения максимального потока как $O(mn(m + n))$. Не будем заниматься здесь деталями этого алгоритма, так как ниже в данном параграфе мы представим более эффективный метод, имеющий сложность $O(n^2)$. Любопытный метод использовал для построения максимального потока Диниц в работе [13]. Этот метод основан на построении некоторой вспомогательной бесконтурной сети (т.е. с графом, не содержащим контуров), структура которой точно отображает все кратчайшие увеличивающие цепи из s в t относительно фактического потока f . Обозначим такую сеть через S_f . Мы строим ее при помощи поиска в ширину в уже упомянутом графе G_f с дугами, определяемыми допустимыми относительно фактического потока f дугами в исходной сети. Сеть S_f содержит источник s , сток t и дуги графа G_f вида $\langle v, u \rangle$, где u находится на расстоянии d , а v на расстоянии $d + 1$ от s , $0 \leq d \leq l$, где l есть длина сети S_f , т.е. расстояние от s до t в графе G_f . Пропускную способность $c_f(u, v)$ мы определяем как $c(u, v) - f(u, v)$ или как $f(v, u)$ в зависимости от того, представляет ли дуга $\langle u, v \rangle$ согласованную дугу или нет (или сумму этих значений), если одновременно существуют согласованная и несогласованная дуги, допустимые от u к v .

Ниже представлена процедура PSA , строящая сеть S_f . Исходная сеть S представлена списками инцидентности $ЗАПИСЬ[v]$, $ПРЕДШ[v]$, $v \in V$, и массивом пропускных способностей дуг $C[u, v]$, $u, v \in V$, а фактический поток f определяется массивом $F[u, v]$, $u, v \in V$. Переменные, относящиеся к построенной вспомогательной сети S_f , отличаются от переменных, относящихся к исходной сети S , первой буквой X : имеем множество вершин XV , списки инцидентности $XЗАПИСЬ[v]$, $XПРЕДШ[v]$, $x \in XV$, и массивы пропускных способностей XC и потоков XF . Расстояние до вершины от источника в сети S_f запоминается в массиве $ДЛИНА$. Структура процедуры PSA следующая: за инициализацией в строках 3–7 следует поиск в ширину в графе G_f , начиная от источника

s (см. главный цикл в строке 10). Вершины v , последовательно посещаемые в процессе поиска, помещаются в очередь (строки 13 и 21), каждая вершина u , выбираемая из очереди (строка 11), используется в двух циклах (строки 12 и 20). Первый из них ищет согласованные допустимые дуги $\langle u, v \rangle$, второй — несогласованные допустимые дуги $\langle v, u \rangle$. Отметим, что сеть может одновременно содержать согласованную допустимую дугу $\langle u, v \rangle$ и несогласованную допустимую дугу $\langle v, u \rangle$. Дуга $\langle v, u \rangle$, включаемая в S_f , имеет в таком случае пропускную способность, равную сумме пропускных способностей, вносимых этими дугами (см. строки 24 и 28).

1. **procedure** *PSA*; (*построение вспомогательной бесконтурной сети S_f ; переменные $V, XV, \text{ПРЕДШ}, \text{ЗАПИСЬ}, \text{ХПРЕДШ}, \text{ХЗАПИСЬ}, \text{ДЛИНА}, C, F, XC, XF, s, t$ — глобальные*)

2. **begin**

3. **for** $u \in V$ **do** (*инициализация*)

4. **begin** $\text{ДЛИНА}[u] := \infty; \text{ХПРЕДШ}[u] := \emptyset; \text{ХЗАПИСЬ}[u] := \emptyset;$

5. **for** $v \in V$ **do** $XC[u, v] := 0;$

6. **for** $v \in V$ **do** $XF[u, v] := 0;$ (*инициализация нулевого потока*)

7. **end;**

8. $\text{ОЧЕРЕДЬ} := \emptyset; XV := \emptyset; \text{ДЛИНА}[s] := 0;$

(*поиск в ширину, начиная от источника s *)

9. $\text{ОЧЕРЕДЬ} \leftarrow s;$

10. **while** $\text{ОЧЕРЕДЬ} \neq \emptyset$ **do**

11. **begin** $u \leftarrow \text{ОЧЕРЕДЬ}; XV := XV \cup \{u\};$

12. **for** $v \in \text{ЗАПИСЬ}[u]$ **do** (*поиск согласованных дуг*)

13. **if** ($\text{ДЛИНА}[u] < \text{ДЛИНА}[v] \leq \text{ДЛИНА}[t]$)

and ($F[u, v] < C[u, v]$) **then**

14. **begin if** $\text{ДЛИНА}[v] = \infty$

then $\text{ОЧЕРЕДЬ} \leftarrow v;$ (* v — новая*)

15. $\text{ДЛИНА}[v] := \text{ДЛИНА}[u] + 1;$ (*добавить $\langle u, v \rangle$ к сети S_f *)

16. $\text{ХЗАПИСЬ}[u] := \text{ХЗАПИСЬ}[u] \cup \{v\};$

17. $\text{ХПРЕДШ}[v] := \text{ХПРЕДШ}[v] \cup \{u\};$

18. $XC[u, v] := C[u, v] - F[u, v]$

19. **end;**

20. **for** $v \in \text{ПРЕДШ}[u]$ **do** (*поиск несогласованных дуг*)

21. **if** ($\text{ДЛИНА}[u] < \text{ДЛИНА}[v] \leq \text{ДЛИНА}[t]$)

and ($F[v, u] > 0$) **then**

22. **begin if** $\text{ДЛИНА}[v] = \infty$ **then** $\text{ОЧЕРЕДЬ} \leftarrow v$ (*новый*)

23. $\text{ДЛИНА}[v] := \text{ДЛИНА}[u] + 1;$

24. **if** $XC[u, v] = 0$ **then**


```

25.      (*добавить  $\langle u, v \rangle$  к сети  $S_f$  *)
26.      begin ХЗАПИСЬ[ $u$ ] := ХЗАПИСЬ[ $u$ ]  $\cup$  { $v$ };
27.      ХПРЕДШ[ $v$ ] := ХПРЕДШ[ $v$ ]  $\cup$  { $u$ }
28.      end;
29.      ХС[ $u, v$ ] := ХС[ $u, v$ ] +  $F[v, u]$ 
30.      end
31. end

```

Пусть по окончании работы процедуры ДЛИНА[t] = l . Если $l = \infty$, то это означает, что мы в нашем процессе не достигли стока t , т.е. в нашей сети не существует увеличивающей цепи от s к t . В силу теоремы 4.3 фактический поток в сети S максимальный. Если $l < \infty$, то кратчайшая увеличивающая цепь из s в t имеет длину l , более того, легко отметить, что пути из s в t во вспомогательной сети взаимно однозначно соответствуют увеличивающим цепям длины l в исходной сети (строго говоря, некоторые из этих «увеличивающих цепей» могут использовать одновременно согласованную и несогласованные дуги, допустимые от u до v). Стоит отметить, что после достижения стока, когда ДЛИНА[t] $< \infty$, вновь встречаемые вершины уже не рассматриваются в процедуре PSA (см. условия в строках 13 и 21), так как ни одна такая вершина, находясь на расстоянии l от s , не может быть вершиной цепи длины l из s в t .

Нетрудно оценить сложность процедуры PSA . Каждая вершина u помещается в очередь и удаляется из нее не более одного раза. Каждая дуга, инцидентная с v , анализируется в точности один раз (в циклах 12 или 20), причем число шагов для каждой такой дуги ограничено константой. Таким образом, общее число шагов будет иметь порядок $n + m$, что доминируется n^2 шагами инициализации массивов XC и XF (строки 5 и 6). Таким образом, вспомогательную бесконтурную сеть можно построить за время $O(n^2)$ (можно было бы избежать дорогостоящей инициализации массивов XC и XF , однако применение такой модифицированной процедуры со сложностью $O(n + m)$ не изменило бы оценки $O(n^3)$ для всего алгоритма построения максимального потока).

Идея Диница основывается на разбиении процесса увеличения потока вдоль увеличивающих цепей на фазы, соответствующие использованию кратчайших цепей определенной длины. Фаза начинается с построения вспомогательной бесконтурной сети, затем во вспомогательной сети находится так называемый псевдомаксимальный поток. *Псевдомаксимальным потоком* в сети S_f длины l мы называем произвольный поток f^* в S_f , такой что в S_f не существует увеличивающей цепи длины l относительно потока f^* , другими словами, для произвольного пути из s в t в S_f , определенного последовательностью вершин

$$v_0, v_1, \dots, v_l \quad (v_0 = s, v_l = t),$$

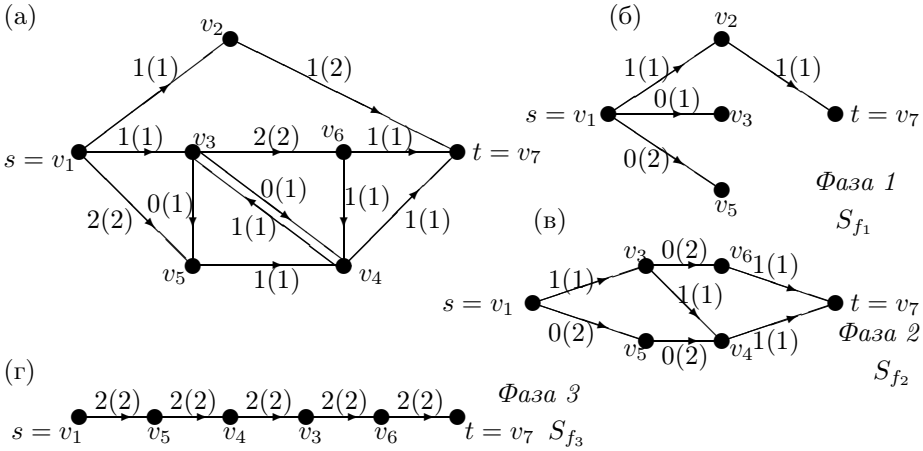


Рис. 4.2: а) Сеть S и максимальный поток в этой сети; б), в), г) Вспомогательные бесконтурные сети, соответствующие последовательным фазам алгоритма Диница с псевдомаксимальными потоками в каждой из них.

существует дуга $\langle v_i, v_{i+1} \rangle$, $0 \leq i < l$, такая что $f^*(v_i, v_{i+1}) = c_f(v_i, v_{i+1})$. Псевдомаксимальный поток затем «переносится» из вспомогательной сети в первоначальную сеть: поток $f^*(u, v)$ складывается с $f(u, v)$, а если это вызывает переполнение дуги, т.е. превышение потока над пропускной способностью дуги $c(u, v)$, то этот «избыток» ликвидируется соответствующим уменьшением потока $f(v, u)$ (таким образом, результатом нашей модификации всегда является увеличение $\text{Div}_f(u)$ на $f^*(u, v)$ и уменьшение $\text{Div}_f(v)$ на $f^*(u, v)$). Нетрудно проверить, что такая модификация потока, проведенная для всех дуг $\langle u, v \rangle$ вспомогательной сети, определяет некоторый новый поток f' , такой что $W(f') = W(f) + W(f^*)$. Фазу считаем законченной. Метод Диница основывается на выполнении последовательных фаз, начиная с нулевого потока, вплоть до момента, пока поток в нашей сети не станет максимальным.

Это показано на рис. 4.2 (как обычно, при каждой дуге указаны поток по этой дуге и в скобках ее пропускная способность). Начиная с нулевого потока f_0 в сети S , мы получаем вспомогательную бесконтурную сеть S_f (рис. 4.2, б). Мы находим в ней псевдомаксимальный поток (метод эффективного построения такого потока будет разобран далее) и переносим его в сеть S , получая поток f_1 .

Затем мы строим сеть S_{f_1} (рис. 4.2, в) и найденный в ней псевдомаксимальный поток переносим в сеть S , что дает поток f_2 . В последней фазе мы строим

сеть S_{f_2} (рис. 4.2, г). После перенесения псевдомаксимального потока из S_{f_2} в S получаем в S поток f (показанный на рис. 4.2, а), который является максимальным, ибо $W(f) = 4 = c(\{s\}, V \setminus \{s\})$ (см. теорему 4.3). Рис. 4.2, в хорошо иллюстрирует тот факт, что псевдомаксимальный поток не должен быть максимальным. Действительно, псевдомаксимальный поток, показанный на этом рисунке, мы можем увеличить вдоль следующей увеличивающей цепи длины 5:

$$v_1, \langle v_1, v_5 \rangle, v_5, \langle v_5, v_4 \rangle, v_4, \langle v_3, v_4 \rangle, v_3, \langle v_3, v_6 \rangle, v_6, \langle v_6, v_7 \rangle, v_7$$

Отметим еще, что дуга $\langle v_4, v_3 \rangle$ сети S_{f_2} (рис. 4.2, г) возникла из двух противоположно направленных дуг сети S .

Метод Диница эффективен в основном из-за того, что независимо от пропускной способности дуг число фаз не превышает n . Для доказательства этого факта обозначим через S_k вспомогательную бесконтурную сеть, построенную в k -й фазе, а через l_k — длину этой сети. (Внимание: последний вызов процедуры PSA , в котором не достигается сток, за фазу не считается.)

Лемма 4.4. *Если число фаз превышает k , то $l_k < l_{k+1}$*

Доказательство. Обозначим через $d_k(v)$ расстояние в S_k от s до v . Нужно доказать, что $d_{k+1}(t) > d_k(t)$. Рассмотрим в S_{k+1} произвольный путь $v_0, v_1, \dots, v_{l_{k+1}}$ из $s = v_0$ в $t = v_{l_{k+1}}$.

Предположим, что для некоторого $p \leq l_{k+1}$ все вершины v_i , $0 \leq i \leq p$, принадлежат сети S_k . Покажем сначала, что тогда

$$d_k(v_{i+1}) \leq d_k(v_i) + 1, \quad 0 \leq i \leq p. \quad (4.9)$$

Допустим, что это не так, т.е. что $d_k(v_{i+1}) > d_k(v_i) + 1$ для некоторого i . Согласно правилам построения сети S_k это означает, что S_k не содержит дуг $\langle v_i, v_{i+1} \rangle$ (и, очевидно, не содержит также дуг $\langle v_{i+1}, v_i \rangle$). Таким образом, во время k -й фазы в S не модифицируется ни поток из v_i в v_{i+1} , ни из v_{i+1} в v_i . Однако это противоречит существованию в начале $(k+1)$ -й фазы дуги в S , допустимой от v_i до v_{i+1} , которой не было в начале k -й фазы. Это противоречие доказывает неравенства (4.9). Из этих неравенств, учитывая, что $d_k(v_0) = 0$, индукцией по i выводим, что

$$d_k(v_i) \leq i = d_{k+1}(v_i), \quad 0 \leq i \leq p \quad (4.10)$$

Если все вершины пути $v_0, v_1, \dots, v_{l_{k+1}}$ принадлежат сети S_k , то отсюда следует неравенство $d_k(t) \leq d_{k+1}(t)$. Равенство $d_k(t) = d_{k+1}(t)$ было бы возможно, только когда $d_k(v_i) = d_{k+1}(v_i)$, $0 \leq i \leq l_{k+1}$, но легко заметить, что повторение пути в S_k и S_{k+1} противоречило бы псевдомаксимальности потока, найденного в S_k . Следовательно, должно быть $d_{k+1}(t) > d_k(t)$.

Рассмотрим теперь случай, когда существует вершина пути $v_0, v_1, \dots, v_{l_{k+1}}$, не принадлежащая S_k . Пусть v_{p+1} — первая такая вершина (очевидно, что $1 \leq$

$p+1 \leq l_{k+1}-1$). Рассуждая аналогично предыдущему, мы можем сделать вывод, что появление в S_{k+1} дуги (v_p, v_{p+1}) , которой не было в S_k , может быть вызвано только тем, что $d_k(v_p) = d_k(t) - 1$, и до вершины v_{p+1} мы доходим уже после достижения стока t (так что она не включается в S_k). Учитывая (4.10) и тот факт, что вершины v_0, v_1, \dots, v_{p-1} принадлежат сети S_k , получаем

$$l_k = d_k(t) = d_k(v_p) + 1 < d_{k+1}(v_p) + 1 = p + 1 \leq l_{k+1}.$$

Принимая во внимание то, что $l_1 \geq 1$ и $l_k \leq n - 1$ для произвольного k , получаем

Следствие 4.5. Число фаз в методе Диница не превышает $n - 1$.

Нам осталось только рассмотреть эффективный алгоритм построения псевдомаксимального потока во вспомогательной бесконтурной сети. Авторами метода, который мы опишем, являются Малхотри, Кумара и Махешвари [48], метод реализуется в процедуре *MAXPSA*, описанной ниже. Чтобы описать эту процедуру, введем понятие *потенциала вершины сети*, т.е. максимального количества потока, который можно «пропустить» через эту вершину. Он определяется как минимум двух величин: максимального возможного потока в вершину v (*входной потенциал*) и максимального возможного потока из вершины v (*выходной потенциал*). Точнее, *потенциал* вершины v произвольной сети $S = \langle G, c \rangle$ с источником s и стоком t равен

$$\Pi(v) = \min \{ \text{ПВХ}(v), \text{ПВЫХ}(v) \},$$

где

$$\text{ПВХ}(v) = \begin{cases} \sum_{u:u \rightarrow v} c(u, v), & \text{если } v \neq s, \\ \infty, & \text{если } v = s. \end{cases}$$

и

$$\text{ПВЫХ}(v) = \begin{cases} \sum_{u:v \rightarrow u} c(u, v), & \text{если } v \neq t, \\ \infty, & \text{если } v = t. \end{cases}$$

Процедура *MAXPSA* сначала вычисляет потенциалы всех вершин во вспомогательной бесконтурной сети (строки 3–12), помещая вершины с нулевым потенциалом в стек СТЕК. Очевидно, что вершину с нулевым потенциалом можно удалить из сети вместе со всеми инцидентными ей дугами, не оказывая влияния на какой-либо поток в этой сети. Удаление такой вершины, а точнее, инцидентных ей дуг, может вызвать обращение в нуль потенциалов некоторых соседних вершин, удаление которых обращает в нуль потенциалы других вершин и т.д. Этот процесс реализуется в цикле 17. После выхода из этого цикла (когда $\text{СТЕК} = \emptyset$) множество XN содержит вершины вспомогательной бесконтурной сети с ненулевым потенциалом (вычисляемым в сети, модифицированной последовательным удалением вершин с нулевым потенциалом). Если $XN \neq \emptyset$, то

наступает важный момент: в XN ищется вершина r с минимальным потенциалом $p = \Pi(r)$ (строки 38–41). Затем мы находим в нашей бесконтурной сети поток из r в t величины p (строки 42–65) и поток из s в r величины p (строки 66–91); в сумме это дает поток из s в t величины p . Опишем первую часть — поток из r в t ; вторая часть полностью аналогична. Наш поток будет использовать только согласованные во вспомогательной бесконтурной сети допустимые дуги, а метод этого вычисления интуитивно лучше всего объяснить следующим образом. Представим себе, что в вершине r мы поместили «груз» величины $Q[r] = p$ и хотим этот груз «протолкнуть» до стока t так, чтобы количество груза, перемещаемое по любой дуге, не превышало ее пропускной способности. Для этого мы ведем в нашей сети поиск в ширину, начиная с вершины r . Как обычно при поиске в ширину достигнутые, но еще не использованные вершины запоминаются в очереди. Каждая вершина u , выбираемая из очереди (строка 43) и не являющаяся стоком, «разгружается» (строки 49–64). Это заключается в перенесении груза $Q[v]$ в вершины u , к которым ведут дуги из v . Этот груз переносится в несколько первых вершин в списке $X\text{ЗАПИСЬ}[v]$, причем груз, перенесенный в каждую такую вершину, за исключением, может быть, последней, равен пропускной способности $XC[v, u]$. Таким же способом на всех использованных дугах $\langle v, u \rangle$, за исключением, может быть, последней, устанавливается поток $XF[v, u] = XC[u, v]$, и эти «насыщенные» дуги мы можем устранить из сети (строки 57–60; вспомним, что мы ищем псевдомаксимальный поток, следовательно, можно ограничиться согласованными допустимыми дугами вспомогательной бесконтурной сети). Устранение дуг сопровождается соответствующим уменьшением потенциалов (строки 44–46). Вершины сети посещаются согласно схеме поиска в ширину, а следовательно, если $d = \text{ДЛИНА}[r]$, весь груз из вершины r переносится в вершины, находящиеся на расстоянии $d + 1$ от s , и далее в вершины, находящиеся на расстоянии $d + 2$ от s и т.д., пока весь груз не достигнет стока t . Отметим, что груз, достигающий каждой вершины u , не превышает p , и эту вершину мы можем разгрузить (если $v \neq t$), так как p , будучи равным минимальному потенциалу, несомненно не превышает выходного потенциала вершины v . После построения аналогичным способом потока величины p из s в r закончена первая итерация главного цикла (строка 15), состоящая в удалении вершин с нулевым потенциалом и в увеличении потока на величину минимального в данный момент потенциала. Дальнейшие итерации следуют вплоть до момента, когда уже нет вершин с ненулевым потенциалом (т.е. $XN = \emptyset$). Очевидно, что найденный тогда в нашей сети поток является псевдомаксимальным.

1. **procedure** *MAXPSA*; (*построение псевдомаксимального потока во вспомогательной бесконтурной сети методом Малхотри, Кумара и Махешвари; переменные XV , XC , XF , $XПРЕДШ$, $XЗАПИСЬ$, s , r — глобальные *)

2. **begin**

3. $СТЕК := \emptyset$;

(*СТЕК будет содержать вершины с нулевым потенциалом*)

4. **for** $v \in XV$ **do** (*определение потенциала вершины v *)

5. **begin** $ПВX[v] := 0$; $ПВЫX[v] := 0$;

6. **if** $v = s$ **then** $ПВX[u] := \infty$

7. **else for** $u \in XПРЕДШ[v]$ **do**

$ПВX[v] := ПВX[v] + XC[u, v]$;

8. **if** $v = t$ **then** $ПВЫX[v] := \infty$

9. **else for** $u \in XЗАПИСЬ[v]$ **do**

$ПВЫX[u] := ПВЫX[v] + XC[v, u]$;

10. $\Pi[v] := \min(ПВX[v], ПВЫX[v])$;

11. **if** $\Pi[v] = 0$ **then** $СТЕК \leftarrow v$

12. **end**;

13. **for** $v \in XV$ **do** $Q[v] := 0$; (*инициализация грузов в вершинах*)

14. $XN := XV$;

(* XN будет содержать вершины с ненулевым потенциалом*)

15. **while** $XN \neq \emptyset$ **do** (* главный цикл *)

16. **begin** (* удаление вершин с нулевым потенциалом*)

17. **while** $СТЕК \neq \emptyset$ **do**

18. **begin** $v \leftarrow СТЕК$; $XN := XN \setminus \{v\}$;

 (*удаление дуг, входящих в v *)

19. **for** $u \in XПРЕДШ[v]$ **do** (*удаление дуги $\langle u, v \rangle$ *)

20. **begin** $ПВЫX[u] := ПВЫX[u] - (XC[u, v] - XF[u, v])$;

21. $XЗАПИСЬ[u] := XЗАПИСЬ[u] \setminus \{v\}$;

$XПРЕДШ[v] := XПРЕДШ[v] \setminus \{u\}$;

22. **if** $\Pi[u] \neq 0$ **then** (*модификация $\Pi[u]$ *)

23. **begin** $\Pi[u] := \min(ПВX[u], ПВЫX[u])$;

24. **if** $\Pi[u] = 0$ **then** $СТЕК \leftarrow u$

25. **end**

26. **end**

(*удаление дуг, исходящих из v *)

27. **for** $u \in XЗАПИСЬ[v]$ **do** (*удаление дуги $\langle v, u \rangle$ *)

28. **begin** $ПВX[u] := ПВX[u] - (XC[v, u] - XF[v, u])$;

29. $XПРЕДШ[u] := XПРЕДШ[u] \setminus \{v\}$;

$XЗАПИСЬ[v] := XЗАПИСЬ[v] \setminus \{u\}$;

```

30.           if  $\Pi[u] \neq 0$  then (*модификация  $\Pi[u]^*$ )
31.             begin  $\Pi[u] := \min(\text{ПВХ}[u], \text{ПВЫХ}[u]);$ 
32.               if  $\Pi[u] = 0$  then  $\text{СТЕК} \leftarrow u$ 
33.             end
34.           end
35.         end;
(* $XN$  является множеством вершин с ненулевым потенциалом*)
36. if  $XN \neq \emptyset$  then (*поток еще не является максимальным*)
37.   begin
(*нахождение вершины  $r$  с минимальным потенциалом*)
38.      $p := \infty;$ 
39.     for  $v \in VN$  do if  $\Pi[v] < p$  then
40.       begin  $r := v; p := \Pi[r]$ 
41.       end;
(*построение потока величины  $p$  из  $r$  в  $t^*$ )
42.      $\text{ОЧЕРЕДЬ} := \emptyset; \text{ОЧЕРЕДЬ} \leftarrow r; Q[r] := p;$ 
43.     repeat  $v \leftarrow \text{ОЧЕРЕДЬ};$ 
44.        $\text{ПВХ}[v] := \text{ПВХ}[v] - Q[v];$ 
45.        $\text{ПВЫХ}[v] := \text{ПВЫХ}[v] - Q[v];$ 
46.        $\Pi[v] := \Pi[v] - Q[v];$ 
47.       if  $\Pi[v] = 0$  then  $\text{СТЕК} \leftarrow v;$ 
48.       if  $v = t$  then  $Q[v] := p$ 
49.     else
50.       begin (*разгрузка вершины  $v^*$ )
51.          $u :=$  первая вершина в списке  $X\text{ЗАПИСЬ}[v];$ 
52.         while  $Q[v] > 0$  do
53.           begin if  $Q[u] = 0$  then  $\text{ОЧЕРЕДЬ} \leftarrow u;$ 
54.              $\text{delta} := \min(Q[v], XC[v, u] - XF[v, u]);$ 
55.              $XF[u, v] := XF[u, v] + \text{delta};$ 
56.              $Q[v] := Q[v] - \text{delta};$ 
57.              $Q[u] := Q[u] + \text{delta};$ 
58.           if  $XF[v, u] = XC[v, u]$  then (*удаление  $\langle v, u \rangle^*$ )
59.             begin  $X\text{ЗАПИСЬ}[v] := X\text{ЗАПИСЬ}[v] \setminus \{u\};$ 
60.                $X\text{ПРЕДШ}[u] := X\text{ПРЕДШ}[u] \setminus \{v\}$ 
61.             end;

```

```

62.         if  $Q[v] > 0$  then
63.              $u :=$  следующая после  $u$  вершина
                списка  $XЗАПИСЬ[v]$ 
64.         end
65.         end (*конец разгрузки вершины  $v^*$ )
66.     until  $v = t$ ; (*поток из  $r$  в  $t$  найден*)
        (*построение потока величины  $p$  из  $s$  в  $r^*$ )
67.     ОЧЕРЕДЬ :=  $\emptyset$ ; ОЧЕРЕДЬ  $\leftarrow r$ ;  $Q[r] := p$ ;
68.     repeat  $v \leftarrow$  ОЧЕРЕДЬ;
69.         if  $v \neq r$  then (* $\Pi[u]$  еще не уменьшен*)
70.             begin  $PВХ[v] := PВХ[v] - Q[v]$ ;
71.                  $PВЫХ[v] := PВЫХ[v] - Q[v]$ ;
72.                  $\Pi[v] := \Pi[v] - Q[v]$ ;
73.                 if  $\Pi[v] = 0$  then  $СТЕК \leftarrow v$ 
74.             end;
75.         if  $v = s$  then  $Q[v] := 0$ 
76.         else
77.             begin (*разгрузка вершины  $v^*$ )
78.                  $u :=$  первая вершина в списке  $XПРЕДШ[v]$ ;
79.                 while  $Q[v] > 0$  do
80.                     begin if  $Q[u] = 0$  then  $ОЧЕРЕДЬ \leftarrow u$ ;
81.                          $delta := \min(Q[v], XC[u, v] - XF[u, v])$ ;
82.                          $XF[u, v] := XF[u, v] + delta$ ;
83.                          $Q[v] := Q[v] - delta$ ;
84.                          $Q[u] := Q[u] + delta$ ;
85.                         if  $XF[u, v] = XC[u, v]$  then (*удаление  $\langle u, v \rangle^*$ )
86.                             begin  $XЗАПИСЬ[u] := XЗАПИСЬ[u] \setminus \{v\}$ ;
87.                                  $XПРЕДШ[v] := XПРЕДШ[v] \setminus \{u\}$ 
88.                             end;
89.                         if  $Q[v] > 0$  then
90.                              $u :=$  следующая после  $u$  вершина
                                    в списке  $XПРЕДШ[v]$ 
91.                         end
92.                     end (*конец разгрузки вершины  $v^*$ )
93.                 until  $v = s$  (*найден поток из  $s$  в  $r^*$ )
94.             end (*конец увеличения потока вдоль единичной вершины*)
95.         end (*конец главного цикла*)
96.     end (*конец процедуры  $MAXPSA^*$ )

```

Оценим теперь число шагов, выполняемых процедурой $MAXPSA$. Определение потенциалов (строки 3–12) требует $O(n+m)$ шагов, поскольку каждая ду-

га $\langle u, v \rangle$ анализируется не более двух раз: при вычислении ПВЫХ[u] и ПВХ[v]. Общее число шагов во всех итерациях главного цикла, выполняемых фрагментом процедуры, удаляющим вершины с нулевым потенциалом (строки 17–35), также порядка $O(n+m)$, так как каждая вершина заносится в СТЕК в точности один раз, а при удалении этой вершины из стека удаляются все инцидентные ей дуги (циклы 19 и 27), так что каждая дуга удаляется не более одного раза. Каждая итерация главного цикла вызывает обращение в нуль потенциала хотя бы одной вершины, а именно вершины r . Таким образом, число итераций этого цикла не превосходит n . Каждая итерация, кроме проанализированного уже процесса удаления вершин с нулевым потенциалом, содержит две части, являющиеся в основном процессами поиска в ширину — от r до t и от s до r . Сложность этих частей порядка числа анализируемых дуг. Анализ дуги может быть «уничтожающим», когда анализируемая дуга насыщается потоком и удаляется из сети, или «неуничтожающим», когда дуга остается в сети и может быть проанализирована в последующих итерациях главного цикла. Очевидно, что в течение всех итераций главного цикла мы анализируем уничтожающим способом $O(m)$ дуг, а во время каждой итерации неуничтожающим способом анализируем не более n дуг, по одной на каждую посещаемую вершину. В сумме это дает число шагов во всех итерациях главного цикла порядка $m+n^2$, т.е. $O(n^2)$. Отсюда мы делаем вывод, что общая вычислительная сложность процедуры *MAXPSA* равна $O(n^2)$. Во время нашего анализа мы предполагали, что удаление дуг из нашей сети (строки 19, 27, 58 и 85) можно выполнить за время, ограниченное константой. Это возможно, если появление вершины v в списке *XЗАПИСЬ*[u] и вершины u в списке *XПРЕДШ*[v] взаимно связаны указателями и каждый элемент списка содержит указатель как на предыдущий, так и на последующий элементы списка.

Теперь мы можем собрать уже описанные процедуры *PSA* и *MAXPSA* в полный алгоритм построения максимального потока.

Алгоритм 4.6. (Построение максимального потока в сети.)

Данные: Сеть, представленная списками инцидентности *ПРЕДШ*[v], *ЗАПИСЬ*[v], $v \in V$, пропускные способности $c[u, v]$, $u, v \in V$.

Результаты: Максимальный поток $F[u, v]$, $u, v \in V$.

1. **begin**
2. **for** $u \in V$ **do**
3. **for** $v \in V$ **do** $F[u, v] := 0$; (*нулевой начальный поток*)
4. **repeat** *PSA*; (*построение вспомогательной бесконтурной сети*)
5. **if** ДЛИНА[t] $\neq \infty$ **then** (*поток F не является максимальным*)
6. **begin** *MAXPSA*;
 (*построение максимального потока во вспомогательной сети*)
 (*перенести поток из вспомогательной сети в главную*)

```

7.      for  $u \in XV$  do (* $XV$  = множество вершин вспомогательной сети*)
8.          for  $v \in XV$  do
9.              begin  $F[u, v] := F[u, v] + XF[u, v]$ ;
10.             if  $F[u, v] > C[u, v]$  then
11.                 begin  $F[v, u] := F[v, u] - (F[u, v] - C[u, v])$ ;
12.                     $F[u, v] := C[u, v]$ 
13.                end
14.            end
15.        end (*конец фазы*)
16.    until ДЛИНА[ $t$ ] =  $\infty$  (*поток  $F$  — максимальный*)
17. end

```

Исходя из проведенного уже анализа процедур *PSA* и *MAXPSA*, а также из того, что число фаз не превышает n (см. следствие 4.5), получаем

Следствие 4.7. *Алгоритм 4.6 правильно строит максимальный поток за время $O(n^3)$.*

Отметим, что анализом этого алгоритма мы доказали существование максимального потока в произвольной сети и тем самым предоставили недостающий фрагмент для доказательства фундаментальной теоремы о максимальном потоке и минимальном разрезе (теорема 4.2).

Стоит отметить, что алгоритм 4.6 не подходит непосредственно под схему последовательного увеличения потока вдоль увеличивающих цепей, однако метод, использованный в этом алгоритме, можно считать обобщением методики увеличивающих цепей. Увеличение потока (на значение, равное минимальному потенциалу) происходит вдоль несколько более общей структуры, чем единичная увеличивающая цепь от s до t . Эту структуру можно было бы назвать «кратной увеличивающей цепью» от s до t , проходящей через вершину r (с минимальным потенциалом). Использование такой кратной цепи приводит к удалению из сети хотя бы одной вершины, а не одной дуги, как в случае единичной увеличивающей цепи — отсюда меньшее число итераций и тем самым лучшая оценка сложности алгоритма.

В заключение сформулируем следующее очевидное, но важное свойство алгоритма 4.6.

Теорема 4.7. *Если все пропускные способности дуг являются целыми числами, то максимальный поток f , построенный алгоритмом 4.6 — целочисленный, т.е. $f(e)$ является целым числом для каждой дуги e .*

4.3 Наибольшие паросочетания в двудольных графах

Паросочетанием в неориентированном графе $G = \langle V, E \rangle$ называется произвольное множество ребер $M \subseteq E$, такое что никакие два ребра из M не инцидентны одной вершине. Другими словами, M есть паросочетание, если для любых двух ребер $e_1, e_2 \in M$ имеем либо $e_1 = e_2$, либо $e_1 \cap e_2 = \emptyset$. Для каждого ребра $\langle u, v \rangle \in M$ мы говорим, что M *сочетает* u с v , а каждую вершину, не принадлежащую ни одному ребру паросочетания, будем называть *свободной*. Особенно интересны задачи, связанные с паросочетаниями в *двудольных* графах, т.е. в неориентированных графах $G = \langle V, E \rangle$, таких что множества их вершин можно разбить на непересекающиеся множества $V = X \cup Y$, $X \cap Y = \emptyset$, причем каждое ребро $e \in E$ имеет вид $e = \langle x, y \rangle$, где $x \in X$, $y \in Y$ (см. задачу 4.15). Двудольный граф будем обозначать $\langle X, Y, E \rangle$ (это обозначение предполагает фиксацию некоторого разбиения множества вершин на множества X и Y — оно не определяется однозначно через множества V и E).

В этом разделе мы займемся задачей нахождения наибольшего паросочетания в заданном двудольном графе. Это классическая задача комбинаторики, известная также под названием «задачи о супружеских парах». Такое название связано со следующей интерпретацией. Пусть X и Y — соответственно множества юношей и девушек, и пусть $\langle x, y \rangle \in E$ означает, что юноша x знаком с девушкой y . Тогда каждое паросочетание M соответствует возможному множеству супружеских пар, в котором каждая пара образована из юноши и девушки, знакомых между собой, причем каждый человек участвует не более чем в одной паре.

Оказывается, что задачу нахождения наибольшего паросочетания в двудольном графе можно простым способом свести к построению максимального потока в некоторой сети. Пусть $H = \langle X, Y, E \rangle$ — произвольный двудольный граф. Построим сеть $S(H)$ с источником s , стоком t ($s \neq t$ и $s, t \notin X \cup Y$), множеством вершин

$$V^* = \{s, t\} \cup X \cup Y,$$

множеством дуг

$$E^* = \{\langle s, x \rangle : x \in X\} \cup \{\langle y, t \rangle : y \in Y\} \cup \{\langle x, y \rangle : x \in X \wedge y \in Y \wedge \langle x, y \rangle \in E\}$$

и пропускной способностью $c(u, v) = 1$ для каждой дуги $\langle u, v \rangle \in E^*$.

На рис. 4.3 показано построение сети $S(H)$ для некоторого двудольного графа.

Отметим, что согласно теореме 4.7 в $S(H)$ существует максимальный нуль-единичный поток, т.е. максимальный поток f , такой что $f(e) = 0$ или $f(e) = 1$ для каждой дуги $e \in E^*$.

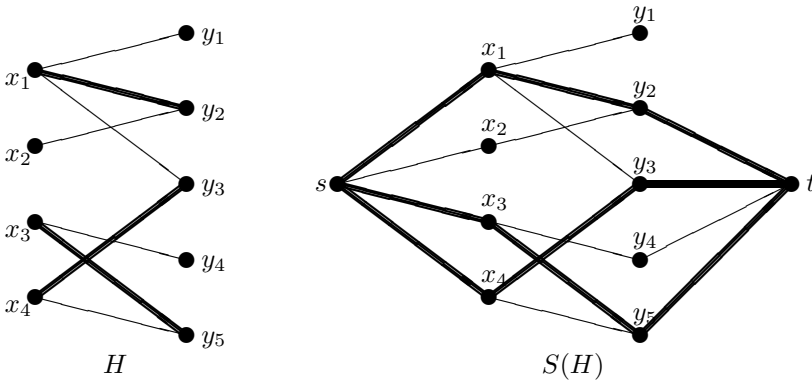


Рис. 4.3: Построение сети $S(H)$ для двудольного графа H и нуль-единичный поток, соответствующий паросочетанию $M = \{\{x_1, y_2\}, \{x_3, y_5\}, \{x_4, y_3\}\}$.

Теорема 4.8. *Существует взаимно однозначное соответствие между паросочетаниями в H и нуль-единичными потоками в $S(H)$, при котором паросочетанию $M = \{\{x_1, y_1\}, \dots, \{x_k, y_k\}\}$ мощности k ($x_i \in X$, $y_i \in Y$ для $1 \leq i \leq k$) соответствует поток f_M величины k , определяемый следующим образом:*

$$f_M(s, x_i) = f_M(x_i, y_i) = f_M(y_i, t) = 1, \quad 1 < i < k. \quad (4.11)$$

и $f_M(e) = 0$ для остальных дуг e сети $S(H)$; потоку f величины k соответствует паросочетание M_f , $|M_f| = k$, определяемое следующим образом:

$$M_f = \{\{x, y\} : x \in X \wedge y \in Y \wedge f(x, y) = 1\}. \quad (4.12)$$

Доказательство. Если $M = \{\{x_1, y_1\}, \dots, \{x_k, y_k\}\}$ — паросочетание мощности k , то вершины x_1, \dots, x_k , а также y_1, \dots, y_k попарно различны. Отсюда сразу же следует, что $\text{Div}_{f_M}(x_i) = \text{Div}_{f_M}(y_i) = 0$, $0 \leq i \leq k$, т.е. функция f_M , определяемая формулой (4.11), является потоком из s в t в $S(H)$. Легко заметить также, что разным паросочетаниям M соответствуют разные потоки f_M . С другой стороны, если f — нуль-единичный поток в $S(H)$, то количество потока, прибывающего в (а следовательно, и вытекающего из) каждую вершину $x \in X$, не превосходит единицы (единственная дуга, входящая в x , — это дуга $\langle s, x \rangle$ с пропускной способностью, равной 1). Отсюда следует, что в множестве M_f , определяемом (4.12), нет ребер вида $\langle x, y_1 \rangle$, $\langle x, y_2 \rangle$, $y_1 \neq y_2$. По аналогичным причинам в M_f нет ребер вида $\langle x_1, y \rangle$, $\langle x_2, y \rangle$, $x_1 \neq x_2$. Следовательно, M_f является паросочетанием в H . Нетрудно также отметить, что отображение $f \rightarrow M_f$ является обратным к отображению $M \rightarrow f_M$, т.е. $f_{M_f} = f$ и $M_{f_M} = M$.

Теорема 4.8 позволяет использовать произвольный алгоритм построения максимального потока (целочисленного) для определения максимального паросочетания. Используя алгоритм 4.6, мы находим наибольшее паросочетание за $O(n^3)$ шагов. Оказывается, однако, что особая форма сети $S(H)$ позволяет построить более эффективный алгоритм. Этот алгоритм, авторами которого являются Хопкрофт и Карп [33], использует общую схему Диница, т.е. состоит из фаз, соответствующих увеличению потока вдоль кратчайших увеличивающихся цепей определенной длины. Особый вид сети позволяет, однако, как ограничить число фаз, так и построить более эффективный алгоритм построения псевдо-максимального потока в каждой фазе.

Отметим сначала, что каждая увеличивающая цепь в $S(H)$ будет нечетной длины и будет состоять после отбрасывания первой и последней дуг из последовательности попеременно чередующихся согласованных и несогласованных дуг (начинающейся и заканчивающейся согласованной дугой). Для данного паросочетания M мы будем называть *чередующейся цепью* (длины $l = 2k + 1$ из X в Y) относительно M произвольное множество дуг $P \subseteq E$ вида

$$P = \{\{x_0, y_1\}, \{y_1, x_1\}, \{x_1, y_2\}, \dots, \{y_k, x_k\}, \{x_k, y_{k+1}\}\}, \quad (4.13)$$

где $k > 0$, все вершины $x_0, \dots, x_k, y_1, \dots, y_{k+1}$ различны, x_0 — свободная вершина в X , y_{k+1} — свободная вершина в Y , каждая вторая дуга принадлежит M , т.е.

$$P \cap M = \{\{y_1, x_1\}, \{y_2, x_2\}, \dots, \{y_k, x_k\}\}$$

Очевидно, что чередующуюся цепь можно однозначно определить последовательностью $x_0, y_1, x_1, y_2, \dots, y_k, x_k, y_{k+1}$. Разумеется, что при описанном в теореме 4.8 соответствии между паросочетаниями в H и нуль-единичными потоками в сети $S(H)$ чередующаяся цепь (4.13) естественным образом соответствует некоторой увеличивающей цепи относительно потока f_M в $S(H)$, а именно цепи

$$s, \langle s, x_0 \rangle, x_0, \langle x_0, y_1 \rangle, y_1, \langle y_1, x_1 \rangle, x_1, \langle x_1, y_2 \rangle, y_2, \dots, \\ y_k, \langle y_k, x_k \rangle, x_k, \langle x_k, y_{k+1} \rangle, y_{k+1}, \langle y_{k+1}, t \rangle, t,$$

причем увеличение потока f_M вдоль этой цепи дает поток, соответствующий паросочетанию, определенному симметрической разностью $M \oplus P$. Это проиллюстрировано на рис. 4.4.

Из приведенных выше рассуждений непосредственно вытекает следующая теорема.

Теорема 4.9. (Берж). *Паросочетание M в двудольном графе H наибольшее тогда и только тогда, когда в H не существует чередующейся цепи относительно M .*

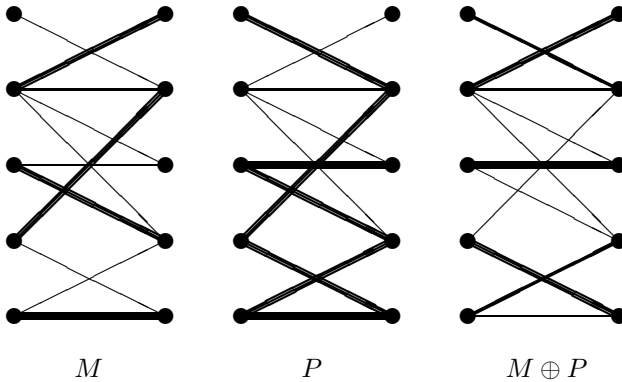


Рис. 4.4: Увеличение паросочетания M вдоль чередующейся цепи P .

Рассмотрим, как выглядит псевдомаксимальный поток во вспомогательной бесконтурной сети, построенной для сети $S(H)$ и некоторого целочисленного, а следовательно, нуль-единичного потока в этой сети. Потенциал каждой вершины, отличной от s и t , составляет 0 или 1, так как входной потенциал каждой вершины $x \in X$ и выходной потенциал каждой вершины $y \in Y$, во вспомогательной сети равен единице. Нетрудно заметить, что «увеличивающая кратная цепь», найденная на каждой итерации главного цикла процедуры $MAXPSA$ (см. предыдущий раздел), имеет вид одиночной увеличивающей цепи. Более того, все промежуточные вершины (т.е. отличные от s и t) такой цепи имеют потенциал, равный единице, а, следовательно, при увеличении потока вдоль этой цепи их потенциал убывает до нуля и они удаляются из сети. Теперь ясно, что во вспомогательной бесконтурной сети длины $l + 2$ поток идет вдоль максимального множества путей длины $l + 2$ из s в t с попарно непересекающимися множествами промежуточных вершин. Этому множеству естественным образом соответствует максимальное множество чередующихся цепей длины l с попарно непересекающимися множествами вершин (под «максимальным» мы подразумеваем такое множество, которое нельзя увеличить на дополнительную чередующуюся цепь длины l и на множество вершин, не содержащее ни одной вершины прежних цепей).

Ниже детально описан алгоритм 4.10, носящий имя Хопкрофта–Карпа. В этом алгоритме вспомогательная бесконтурная сеть, а точнее вспомогательный бесконтурный граф, поскольку пропускные способности всех ребер равны единице, строится при помощи процедуры PGA . Чтобы объяснить ее действие, удобно ввести в рассмотрение для данного двудольного графа $H = \langle X, Y, E \rangle$ и паросочетания M в H ориентированный граф H_M посредством ориентации

всех ребер $e \in M$ от Y к X и всех ребер $e \in E \setminus M$ от X к Y (пути в H_M от свободной вершины в X до свободной вершины в Y в точности соответствуют чередующимся цепям относительно M). Процедура *PGA* сначала строит дуги от источника s до всех свободных вершин $x \in X$ (строки 6–10). Далее проводится поиск в ширину в графе H_M , начиная от свободных вершин в X . При нахождении каждой свободной вершины $y \in Y$ добавляем во вспомогательный бесконтурный граф дугу $\langle y, t \rangle$ (строки 14–17), причем с момента добавления первой такой дуги мы уже в процессе поиска не рассматриваем вершин, находящихся на расстоянии, большем или равном расстоянию от s до t (см. условие в строке 20). Увеличение существующего паросочетания вдоль максимального множества кратчайших увеличивающих цепей с попарно различными множествами вершин реализуется процедурой ФАЗА. Она ведет поиск в глубину во вспомогательном бесконтурном графе, начиная с s . Главный цикл в строке 40 напоминает рекурсивную процедуру поиска в глубину *WG1* (см. 2.2). Каждый раз, когда в нашем процессе мы достигаем стока t (см. строку 50), содержимым стека СТЕК является последовательность вершин, представляющая некоторую чередующуюся цепь (отметим, что сток t не помещается в СТЕК, тем самым все время $\text{НОВЫЙ}[t] = \text{истина}$ и, следовательно, t может посещаться многократно). Цикл 52 выполняет модификацию массива СОЧЕТ, соответствующего увеличению паросочетания вдоль такой цепи. Отметим, что если во время выполнения нашей процедуры некоторой вершине $v \in V$ приписывается значение $\text{НОВЫЙ}[v] = \text{ложь}$, то эта процедура либо находит затем чередующуюся цепь, проходящую через эту вершину, либо устанавливает, что не существует ни одной цепи, проходящей через эту вершину и не пересекающей ни одной из ранее найденных цепей. Отсюда легко следует, что процедура ФАЗА находит максимальное множество непересекающихся кратчайших чередующихся цепей. Корректность всего алгоритма следует из анализа метода Диница, проведенного в предыдущем разделе.

Алгоритм 4.10. (Нахождение наибольшего паросочетания методом Хопкрофта–Карпа.)

Данные: Двудольный граф $H = \langle X, Y, E \rangle$, представленный списками инцидентности ЗАПИСЬ $[x]$, $x \in X$.

Результаты: Наибольшее паросочетание, представленное массивом СОЧЕТ (СОЧЕТ $[v]$ есть вершина, сочетающаяся с v (а если v — свободная вершина, СОЧЕТ $[v] = 0$)).

1. **procedure** *PGA*; (*построение вспомогательного бесконтурного графа; переменные $V, XV, X, Y, \text{СОЧЕТ}, \text{ЗАПИСЬ}, \text{ХЗАПИСЬ}, s, t$ — глобальные*)

2. **begin**

3. **for** $u \in V \cup \{s, t\}$ **do** (*инициализация*)

4. **begin** ДЛИНА[u] := ∞ ; ХЗАПИСЬ[u] := \emptyset

5. **end**;

(*поместить свободные вершины $x \in X$ в очередь и в список ХЗАПИСЬ[s] *)

6. ОЧЕРЕДЬ := \emptyset ; $XV := \{s\}$; ДЛИНА[s] := 0;

7. **for** $x \in X$ **do**

8. **if** СОЧЕТ[x] = 0 **then** (* x — свободная*)

9. **begin** ОЧЕРЕДЬ $\leftarrow x$;

 ХЗАПИСЬ[s] := ХЗАПИСЬ[s] $\cup \{x\}$; ДЛИНА[x] := 1

10. **end**;

(*поиск в ширину, начиная со свободных вершин в X *)

11. **while** ОЧЕРЕДЬ $\neq \emptyset$ **do**

12. **begin** $u \leftarrow$ ОЧЕРЕДЬ; $XV := XV \cup \{u\}$;

13. **if** $u \in Y$ **then**

14. **if** СОЧЕТ[u] = 0 **then** (* u — свободная, добавить дугу $\langle u, t \rangle$ *)

15. **begin** ХЗАПИСЬ[u] := ХЗАПИСЬ[u] $\cup \{t\}$; $XV := XV \cup \{t\}$;

16. ДЛИНА[t] := ДЛИНА[u] + 1

17. **end**

18. **else** (*СОЧЕТ[u] $\neq 0$ *)

19. **begin** $x :=$ СОЧЕТ[u];

20. **if** ДЛИНА[t] = ∞ **then** (*добавить дугу $\langle u, x \rangle$ *)

21. **begin** ОЧЕРЕДЬ $\leftarrow x$;

 ХЗАПИСЬ[u] := ХЗАПИСЬ[u] $\cup \{x\}$;

22. ДЛИНА[x] := ДЛИНА[u] + 1

23. **end**

24. **end**

25. **else** (* $u \in X$ *)

26. **for** $y \in$ ЗАПИСЬ[u] **do**

27. **if** ДЛИНА[u] < ДЛИНА[y] **then**

 (*ДЛИНА[y] = ДЛИНА[u] + 1 или ∞ *)

28. **begin** **if** ДЛИНА[y] = ∞ **then** ОЧЕРЕДЬ $\leftarrow y$; (* y — новая*)

29. ХЗАПИСЬ[u] := ХЗАПИСЬ[u] $\cup \{y\}$;

30. ДЛИНА[y] := ДЛИНА[u] + 1

31. **end**

32. **end**

(*PGA*)

33. procedure ФАЗА

(*увеличение существующего паросочетания вдоль максимального множества путей из s в t с попарно различными множествами промежуточных вершин во вспомогательном бесконтурном графе; переменные XV , НАЧАЛО, ХЗАПИСЬ, СОЧЕТ, s , t — глобальные*)

34. **begin**

35. **for** $u \in XV$ **do** (*инициализация*)

36. **begin** НОВЫЙ[u]:=истина;

37. $P[u] :=$ НАЧАЛО[u]

(*НАЧАЛО[u] — указатель на начало списка ХЗАПИСЬ[u];

$P[u]$ =указатель на фактически анализируемый элемент списка ХЗАПИСЬ[u]*)

38. **end**;

(*поиск в глубину во вспомогательном бесконтурном графе, начиная с источника s *)

39. $СТЕК := \emptyset$; $СТЕК \leftarrow s$; НОВЫЙ[s] :=ложь;

40. **while** $СТЕК \neq \emptyset$ **do** (*главный цикл*)

41. **begin** $u := top(СТЕК)$; (* u — верхний элемент стека*)

42. (*отыскание первой новой вершины в списке ХЗАПИСЬ[u]*)

43. **if** $P[u]=nil$ **then** $b:=ложь$

else $b:=not$ НОВЫЙ[$P[u] \uparrow$.верш];

44. **while** b **do**

45. **begin** $P[u] := P[u] \uparrow$.след;

46. **if** $P[u]=nil$ **then** $b:=ложь$

47. **else** $b:=not$ НОВЫЙ[$P[u] \uparrow$.верш]

48. **end**;

49. **if** $P[u] \neq nil$ **then** (*найдена новая вершина*)

50. **if** $P[u] \uparrow$.верш = t **then** (*найдена чередующаяся цепь*)

51. (*увеличение существующего паросочетания вдоль чередующейся цепи, хранящейся в стеке*)

52. **while** $top(СТЕК) \neq s$ **do**

53. **begin** $y \leftarrow$ СТЕК; $x \leftarrow$ СТЕК;

54. $СОЧЕТ[x] := y$; $СОЧЕТ[y] := x$

55. **end**

56. (*в стеке остался только источник s *)

57. **else** (* $P[u] \uparrow$.верш $\neq t$ *)

58. **begin** $v := P[u]$.верш; $СТЕК \leftarrow v$; НОВЫЙ[v] :=ложь

59. **end**

60. **else** (* $P[u]=nil$, т.е. в списке ХЗАПИСЬ[u] уже нет новых вершин*)

61. $u \leftarrow$ СТЕК (*удалить верхний элемент стека*)

62. **end**

63. **end**; (*ФАЗА*)

64. **begin** (*главная программа*)

65. **for** $v \in V$ **do** СОЧЕТ[v] := 0; (*инициализация; паросочетание — пустое*)

66. **PGA**; (*построение вспомогательного бесконтурного графа*)

67. **repeat** ФАЗА; **PGA**

68. **until not** $XV[t]$

69. **end**

Большая эффективность алгоритма Хопкрофта–Карпа объясняется тем, что порядок числа фаз есть \sqrt{n} . Для доказательства ограниченности числа фаз нам будет необходима следующая лемма.

Лемма 4.11. Пусть M и N — два паросочетания в двудольном графе $H = \langle X, Y, E \rangle$, и пусть $|M| = r < s = |N|$. Тогда симметрическая разность $M \oplus N$ содержит не менее $s - r$ чередующихся цепей относительно M с попарно различными множествами вершин.

Доказательство. Рассмотрим граф $H^* = \langle X, Y, M \oplus N \rangle$ и обозначим через C_1, \dots, C_p компоненты связности этого графа. Каждая вершина графа H^* принадлежит не более чем одному ребру из $M \setminus N$ и не более чем одному ребру из $N \setminus M$ (так как M и N — паросочетания). Отсюда следует, что каждая компонента связности C_i имеет один из трех следующих видов:

1. изолированная вершина;
2. цикл четной длины с ребрами попеременно из $M \setminus N$ и $N \setminus M$;
3. цепь с ребрами попеременно из $M \setminus N$ и $N \setminus M$ (это не обязательно чередующаяся цепь: оба ее конца могут принадлежать X или оба Y).

Обозначим через E_i множество ребер компоненты C_i и рассмотрим величину $\delta_i = |E_i \cap N| - |E_i \cap M|$. Имеем $\delta_i \in \{-1, 0, 1\}$, причем $\delta_i = 1$ тогда и только тогда, когда C_i — чередующаяся цепь относительно M . Более того, $\delta_i = 1$ для не менее чем $s - r$ указателей, так как

$$\begin{aligned} \sum_{i=1}^p \delta_i &= \sum_{i=1}^p (|E_i \cap N| - |E_i \cap M|) = \sum_{i=1}^p |E_i \cap N| - \sum_{i=1}^p |E_i \cap M| = \\ &= |N \setminus M| - |M \setminus N| = |N| - |M| = s - r. \end{aligned}$$

Этим доказательство леммы закончено.

Теперь мы можем приступить к доказательству существования объявленно ограниченного на число фаз алгоритма Хопкрофта–Карпа.

Теорема 4.12. Число фаз алгоритма Хопкрофта–Карна не превышает $2\lfloor\sqrt{s}\rfloor + 1$, где s — наибольшая мощность паросочетания в данном графе.

Доказательство. Обозначим через P_0, P_1, \dots, P_{s-1} последовательные чередующиеся цепи, построенные алгоритмом, и определим $M_0 = \emptyset$, $M_i = M_{i-1} \oplus P_{i-1}$ для $1 \leq i \leq s$. Согласно алгоритму P_i является чередующейся цепью относительно M_i , и мы уже знаем, что $|P_0| \leq |P_1| \leq \dots \leq |P_s - 1|$ (см. лемму 4.4). Число фаз — это не что иное, как количество различных чисел в последовательности $|P_0|, |P_1|, \dots, |P_{s-1}|$. Обозначим $r = \lfloor s - \sqrt{s} \rfloor$, и пусть r^* — такой наименьший индекс, что $|P_{r^*}| = |P_r|$. Согласно построению вспомогательного бесконтурного графа P_{r^*} — кратчайшая чередующаяся цепь относительно M_{r^*} (фактически P_i — это кратчайшая чередующаяся цепь относительно M_i для каждого i , но этот факт требует доказательства, см. задачу 4.11). По лемме 4.11 существует по крайней мере $|M_s| - |M_{r^*}| = s - r^*$ непересекающихся чередующихся цепей относительно M_{r^*} , следовательно, кратчайшая чередующаяся цепь относительно M_{r^*} содержит не более чем $r^*/(s - r^*)$ ребер паросочетания M_{r^*} .

Отсюда

$$\begin{aligned} |P_r| &= |P_{r^*}| \leq \frac{2r^*}{s-r^*} + 1 \leq \frac{2r}{s-r} + 1 = \frac{2\lfloor s-\sqrt{s} \rfloor}{\lfloor \sqrt{s} \rfloor} + 1 \\ &= \frac{2s}{\lfloor \sqrt{s} \rfloor} - 1 \leq 2\sqrt{s} - 1 \leq 2\lfloor \sqrt{s} \rfloor + 1. \end{aligned}$$

Длина каждой цепи нечетная, а, следовательно, последовательность $|P_0|, \dots, |P_r|$ содержит не более чем $\lfloor \sqrt{s} \rfloor + 1$ различных чисел. Последовательность $|P_{r+1}|, \dots, |P_{s-1}|$ может содержать не более чем $(s-1) - (r+1) + 1 = s - r - 1 = \lfloor \sqrt{s} \rfloor - 1 \leq \lfloor s \rfloor$ других чисел, что в сумме дает требуемое ограничение $2\lfloor \sqrt{s} \rfloor + 1$.

Теперь легко оценить сложность всего алгоритма 4.10. Как поиск в ширину в процедуре *PGA*, так и поиск в глубину в процедуре *ФАЗА* требуют $O(m+n)$ шагов, что при числе фаз порядка \sqrt{n} дает общую сложность $O((m+n)\sqrt{n})$ или, если использовать только число вершин в качестве размерности задачи, $O(n^{5/2})$.

Стоит отметить, что известен алгоритм построения наибольшего паросочетания в произвольном необязательно двудольном графе со сложностью $O(n^{5/2})$ (см. [19]). Однако этот алгоритм несравненно более сложный. Алгоритм для двудольного графа, отличный от представленного здесь, предложил Галил в [26].

4.4 Системы различных представителей

Пусть $\langle A_1, \dots, A_n \rangle$ — произвольная последовательность множеств (необязательно непересекающихся и необязательно различных). *Системой различных представителей* для $\langle A_1, \dots, A_n \rangle$ будем называть такую произвольную последовательность $\langle a_1, \dots, a_n \rangle$, что $a_i \in A_i$, $1 \leq i \leq n$, и $a_i \neq a_j$ для $i \neq j$. Мы будем говорить, что в такой системе различных представителей элемент a_i *представляет* множество A_i . Проблема существования и построения системы различных представителей известна во многих неформальных постановках. Одна из них — это так называемая «задача о комиссиях». Имеется n комиссий, причем A_i — множество членов i -й комиссии. Нужно в каждой комиссии выбрать председателя так, чтобы ни один человек не был председателем более чем в одной комиссии. Нетрудно заметить, что задача о комиссиях сводится к частному случаю задачи о супружеских парах. Действительно, создадим множества

$$X = A_1 \cup \dots \cup A_n = \{x_1, \dots, x_m\} \quad (4.14)$$

$$Y = \{y_1, \dots, y_n\} \quad (4.15)$$

(элементы $x_1, \dots, x_m, y_1, \dots, y_n$ попарно различны) и

$$E = \{\{x_i, y_j\} : 1 \leq i \leq m \wedge 1 \leq j \leq n \wedge x_i \in A_j\}. \quad (4.16)$$

Очевидно, что каждая система различных представителей $\langle a_1, \dots, a_n \rangle$ однозначно соответствует паросочетанию мощности n в двудольном графе $H = \langle X, Y, E \rangle$, а именно паросочетанию $\{\{a_1, y_1\}, \dots, \{a_n, y_n\}\}$.

Принимая во внимание тот факт, что граф H имеет $m + n$ вершин и $\sum_{j=1}^n |A_j|$ ребер, получаем

Следствие 4.13. *Для данной последовательности множеств $\langle A_1, \dots, A_n \rangle$ можно найти систему различных представителей или установить, что такой системы не существует, за время $O\left(\sqrt{n} \sum_{j=1}^n |A_j|\right)$.*

Доказательство. Число фаз в алгоритме Хопкрофта–Карпа, примененном к графу H , не превышает $2\lfloor\sqrt{n}\rfloor + 1$ (см. теорему 4.12). Отсюда число шагов алгоритма имеет порядок $O\left(\sqrt{n}\left(n + \sum_{j=1}^n |A_j|\right)\right)$. Можно предполагать, что все множества A_i непустые, в противном случае очевидно, что системы различных представителей не существует. При таком предположении $n + \sum_{j=1}^n |A_j| \leq 2\sum_{j=1}^n |A_j|$, что дает требуемую оценку сложности алгоритма.

Докажем теперь классическую теорему Ф. Холла [30], дающую необходимое и достаточное условие существования системы различных представителей для данной последовательности множеств.

Теорема 4.14. Система различных представителей для последовательности $\langle A_1, \dots, A_n \rangle$ существует тогда и только тогда, когда

$$\left| \bigcup_{j \in J} A_j \right| \geq |J| \text{ для каждого } J \subseteq \{1, \dots, n\}. \quad (4.17)$$

Доказательство. Если существует система различных представителей $\langle a_1, \dots, a_n \rangle$, то для каждого J имеем

$$\left| \bigcup_{j \in J} A_j \right| \geq \left| \bigcup_{j \in J} \{a_j\} \right| = |J|$$

Предположим теперь, что для $\langle A_1, \dots, A_n \rangle$ не существует системы различных представителей, и рассмотрим наибольшее паросочетание M в графе $H = \langle X, Y, E \rangle$, соответствующем последовательности $\langle A_1, \dots, A_n \rangle$ (см. формулы (4.14), (4.15), (4.16)). Очевидно, что $|M| < n$, а следовательно, в Y существует свободная вершина, допустим b_0 . Обозначим через X_0 множество тех вершин $x \in X$, до которых существует «частичная чередующаяся цепь» из b_0 вида

$$b_0, a_1, b_1, \dots, b_{k-1}, a_k,$$

где $k \geq 1$, $a_k = x$, $\{a_i, b_i\} \in M$ для $1 \leq i \leq k-1$ и $\{b_{i-1}, a_i\} \in E \setminus M$ для $1 \leq i \leq n$. Очевидно, что X_0 не содержит свободных вершин, поскольку такая вершина соответствовала бы чередующейся цепи вопреки предположению, что M — наибольшее паросочетание. Пусть Y_0 будет множеством вершин, сочетаемых посредством M с вершинами из X_0 . Обозначим $J = \{j : 1 \leq j \leq n \wedge y_j \in Y_0 \cup \{b_0\}\}$. Нетрудно заметить, что

$$\bigcup_{j \in J} A_j = X_0$$

и, следовательно,

$$\left| \bigcup_{j \in J} A_j \right| = |X_0| = |Y_0| = |J| - 1 < |J|,$$

т.е. условие (4.17) не выполняется.

Очевидно, что теорема Холла не имеет большого значения с алгоритмической точки зрения, так как проверка условия (4.17) требует рассмотрения всех 2^n возможных множеств $J \subseteq \{1, \dots, n\}$.

Докажем еще одну теорему, связанную с паросочетаниями в двудольном графе. Назовем *вершинным покрытием* в графе $G = \langle V, E \rangle$ такое произвольное

множество $P \subseteq V$, что каждое ребро $e \in E$ инцидентно с некоторой вершиной из множества P . Отметим, что P является вершинным покрытием тогда и только тогда, когда $V \setminus P$ есть независимое множество вершин, т.е. множество вершин, в котором никакие две не связаны ребром. В дальнейшем нас будут интересовать наименьшие по мощности вершинные покрытия в двудольных графах. Пусть M — наибольшее паросочетание в двудольном графе $H = \langle X, Y, E \rangle$, и пусть A — множество вершин $a \in X \cup Y$, достижимых из свободных вершин в X при помощи «частичных чередующихся цепей», определенных аналогично тому, как это сделано в доказательстве теоремы Холла.

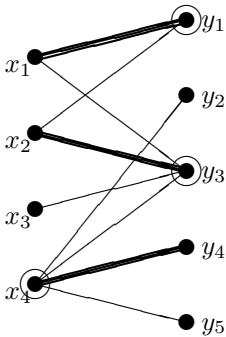
Лемма 4.15. *Если граф H не содержит изолированных вершин, то $P = (X \setminus A) \cup (Y \cap A)$ является минимальным вершинным покрытием, а $N = (X \cap A) \cup (Y \setminus A)$ — максимальным независимым множеством вершин.*

Доказательство. Сначала покажем, что P является вершинным покрытием. Предположим, что некоторое ребро $\{x, y\} \in E$, $x \in X$, $y \in Y$, не будет инцидентным ни с одной из вершин множества P . Это означает, что $x \in A$ и $y \notin A$. Включение $\{x, y\} \in M$ не может иметь места, ибо в таком случае вершина x может быть достижима только с помощью цепи, проходящей через y , что вызвало бы $y \in A$. Однако и ситуация, в которой $\{x, y\} \in E \setminus M$, также невозможна, так как цепь, достигающую x , можно было бы увеличить на ребро $\{x, y\}$, что снова вызвало бы $y \in A$. Полученное противоречие доказывает, что P является вершинным покрытием. Отметим, что ни одна вершина в P не является свободной. Для вершин из $X \setminus A$ это вытекает непосредственно из определения множества A , а для вершин из $Y \cap A$ из того, что свободная вершина в $Y \cap A$ соответствовала бы чередующейся цепи относительно M вопреки нашему предположению о максимальной $|M|$. Подобным же образом доказываем, что не более одной вершины произвольного ребра $\{x, y\} \in M$ принадлежит множеству P : если $y \in Y \cap A$, то $x \in A$ и тем самым $x \notin P$. Отсюда следует, что $|P| \leq |M|$. Однако каждое вершинное покрытие по определению должно содержать не менее одной вершины каждого ребра, а для ребер паросочетания очевидно, что эти вершины попарно различны. Следовательно, всегда должно выполняться $|P| \geq |M|$, что доказывает минимальность нашего вершинного покрытия и равенство $|P| = |M|$. Вторая часть леммы вытекает из первой в силу наших предыдущих замечаний и из того, что $N = (X \cup Y) \setminus P$.

Отметим следствие из приведенного выше доказательства; заметим, что изолированные вершины не оказывают никакого влияния ни на паросочетания, ни на вершинные покрытия.

Следствие 4.16. *В произвольном двудольном графе минимальная мощность вершинного покрытия равна максимальной мощности паросочетания.*

Следует дать здесь другую формулировку, эквивалентную приведенной выше, — формулировку в терминах 0-1-матриц. Под *линией* такой матрицы мы



	y_1	y_2	y_3	y_4	y_5
x_1	1	0	1	0	0
x_2	1	0	1	0	0
x_3	0	0	1	0	0
x_4	0	1	1	1	1

Рис. 4.5: Иллюстрация венгерской теоремы.

будем понимать ее строку или столбец. Теорема, приводимая ниже, часто называется также «венгерской теоремой».

Теорема 4.17. (Кениг и Эгервари). *Для произвольной 0-1-матрицы максимальная мощность множества (появлений) единиц, из которых никакие две не лежат на одной линии, равна минимальному числу линий, которыми можно покрыть все единицы.*

Доказательство. Для данной 0-1-матрицы $A = [a_{ij}]$ с n строками и m столбцами построим двудольный граф $H = \langle X, Y, E \rangle$, где $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_m\}$ и

$$E = \{\{x_i, y_j\} : 1 \leq i \leq n \wedge 1 \leq j \leq m \wedge a_{ij} = 1\}$$

Легко отметить, что множества единиц в A , никакие две из которых не лежат на одной линии, в точности соответствуют паросочетанию в H , а каждое множество линий в A , покрывающих все единицы, представляет вершинное покрытие в H . Следовательно, наша теорема является просто другой формулировкой следствия 4.16.

На рис. 4.5 представлен двудольный граф с максимальным паросочетанием и минимальным вершинным покрытием, а также представлена соответствующая 0-1-матрица, иллюстрирующая венгерскую теорему.

Отметим, что множество A , появляющееся в лемме 4.15, можно найти за время $O(|X| + |Y| + |E|)$, используя поиск в глубину (или в ширину). Если мы воспользуемся алгоритмом Хопкрофта–Карпа и заметим, что каждую изолированную вершину можно удалить из произвольного вершинного покрытия и можно добавить к каждому независимому множеству вершин, то получим следствие:

Следствие 4.18. *Как минимальное вершинное покрытие, так и максимальное независимое множество вершин в произвольном двудольном графе $H = \langle X, Y, E \rangle$ могут быть найдены за время $O((m+n)\sqrt{n})$, где $n = |X \cup Y|$ и $m = |E|$.*

Следует отметить, что для задач, приведенных выше, не известен ни один полиномиальный алгоритм для случая произвольных, не обязательно двудольных графов.

В заключение этого параграфа займемся задачей существования общей системы различных представителей для двух последовательностей множеств $\langle A_1, \dots, A_n \rangle$ и $\langle B_1, \dots, B_n \rangle$.

Рассмотрение такой системы как просто системы различных представителей как для первой, так и для второй последовательности не дает ничего интересного: ситуация сводится просто к рассмотрению последовательности $\langle A_1 \cap B_1, \dots, A_n \cap B_n \rangle$. Поэтому мы определим *общую систему различных представителей* для наших последовательностей как произвольную последовательность $\langle a_1, \dots, a_n \rangle$, обладающую тем свойством, что существует перестановка σ множества $\{1, \dots, n\}$, такая что $\langle a_1, \dots, a_n \rangle$ является системой различных представителей для $\langle A_1, \dots, A_n \rangle$ и для $\langle B_{\sigma(1)}, \dots, B_{\sigma(n)} \rangle$. Задача существования и построения такой системы легко сводится (см. [23]) к построению максимального нуль-единичного потока тем же способом, что и для нахождения паросочетания. Соответствующая сеть имеет множество вершин

$$V = \{s, t\} \cup \{x_1, \dots, x_n\} \cup \{u_1, \dots, u_m\} \cup \{v_1, \dots, v_m\} \cup \{y_1, \dots, y_n\},$$

где $A_1 \cup \dots \cup A_n \cup B_1 \cup \dots \cup B_n = \{z_1, \dots, z_m\}$. Вершина x_i соответствует множеству A_i , вершина y_i — множеству B_i , а вершина u_j , так же как и вершина v_j , — элементу z_j . Множество дуг сети определяется следующим образом:

$$\begin{aligned} E = & \{ \langle s, x_i \rangle : 1 \leq i \leq n \} \cup \{ \langle x_i, u_j \rangle : 1 \leq i \leq n \wedge 1 \leq j \leq m \wedge z_j \in A_i \} \\ & \cup \{ \langle u_j, v_j \rangle : 1 \leq j \leq m \} \cup \{ \langle v_j, y_i \rangle : 1 \leq j \leq m \wedge 1 \leq i \leq n \wedge z_j \in B_i \} \\ & \cup \{ \langle y_j, t \rangle : 1 \leq j \leq n \}, \end{aligned}$$

причем все дуги имеют пропускную способность, равную единице. Предоставляем читателю доказательство того, что общая система различных представителей существует тогда и только тогда, когда в описанной выше сети существует поток величины n , а также уточнение деталей алгоритма нахождения такой системы со сложностью $O\left(\sqrt{n} \sum_{j=1}^n (|A_j| + |B_j|)\right)$ (см. задачу 4.19).

Очень прост случай, когда как $\langle A_1, \dots, A_n \rangle$ так и $\langle B_1, \dots, B_n \rangle$ определяет разбиение некоторого множества на блоки одинаковой мощности.

Теорема 4.19. *Пусть $A_1 \cup \dots \cup A_n = B_1 \cup \dots \cup B_n = X$, $|A_i| = |B_i| = k$ и $|X| = nk$. Тогда существует k общих систем различных представителей, которые в сумме исчерпывают все элементы множества X .*

Доказательство. Построим двудольный граф $H = \langle X, Y, E \rangle$, где $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_m\}$,

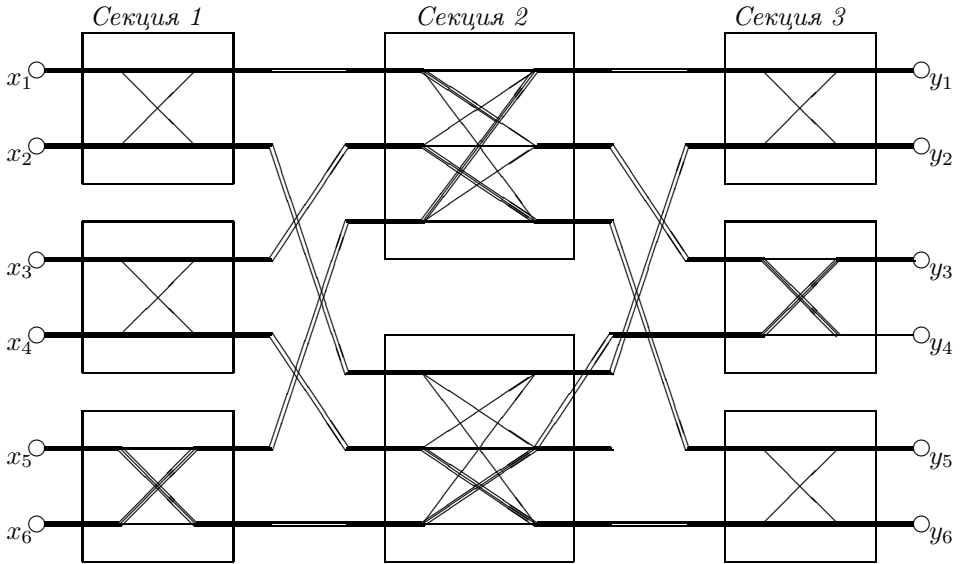
$$E = \{\{x_i, y_j\} : 1 \leq i, j \leq n \wedge A_i \cap B_j \neq \emptyset\}$$

Покажем, что в графе H существует паросочетание мощности n . Действительно, для произвольного $J \subseteq \{1, \dots, n\}$ сумма $\bigcup_{j \in J} A_j$ содержит в точности $|J|k$ элементов и в результате пересекается по крайней мере с $|J|$ из множеств B_1, \dots, B_n . Следовательно, существование заданного паросочетания вытекает из теоремы Холла (см. теорему 4.14). Пусть наше паросочетание сочетает x_i с $y_{\sigma(i)}$, $1 \leq i \leq n$. Множества $C_i = A_i \cap B_{\sigma(i)}$, $1 \leq i \leq n$, непусты и не пересекаются, а следовательно, выбирая произвольные элементы $a_i \in C_i$, $1 \leq i \leq n$, получаем общую систему различных представителей $\langle a_1, \dots, a_n \rangle$. Удалив элементы a_i , $1 \leq i \leq n$, из множеств A_i и B_i , $1 \leq t \leq n$, получаем ситуацию, идентичную первоначальной, с той разницей, что мощность всех множеств уменьшена до $k - 1$. Повторив приведенное выше построение k раз, получаем k систем различных представителей, о которых речь идет в теореме.

Покажем теперь применение этой теоремы к теории коммутационных сетей, используемых в телефонной связи. Предположим, что даны два множества «абонентов» X и Y , и пусть $|X| = |Y| = N$. Наша задача — составить проект установки, состоящей из одиночных переключателей, которая могла бы реализовать систему связей, определенную взаимно однозначным отображением $\varphi : X \rightarrow Y$. Такую установку мы будем называть *перестраиваемой сетью размерности $N \times N$* . Предположим, что каждый переключатель может находиться в одном из двух состояний: замкнут и разомкнут. Самым простым решением, требующим однако N^2 переключателей, является использование отдельного переключателя между каждой парой абонентов $\langle x, y \rangle \in X \times Y$. Реализация отображения φ основывается на замыкании переключателей для пар $\langle x, \varphi(x) \rangle$, $x \in X$, при размыкании всех остальных. Такую установку будем называть *коммутатором размерности $N \times N$* .

Отметим, что каждая перестраиваемая сеть может реализовывать все $N!$ (взаимно однозначных) отображений X на Y , следовательно, она должна иметь не менее чем $N!$ возможных состояний. Поскольку состояние определяется комбинацией частных (с двумя состояниями) переключателей, то число переключателей должно быть не менее чем $\log N! = O(N \log N)$. Далее будет показано, что существует перестраиваемая сеть, содержащая $O(N \log N)$ переключателей.

Предположим, что $N = nk$, где n и k — целые числа, большие единицы. *Трехсекционная сеть Клоса* (см. [4], [7], [50]) типа $\langle n, k \rangle$ мы строим из трех секций: первой, состоящей из n коммутаторов размерности $k \times k$, второй, состоящей из k коммутаторов размерности $n \times n$, и третьей, состоящей, как и первая, из n

Рис. 4.6: Трехсекционная сеть Клоса типа $\langle 3, 2 \rangle$

коммутаторов размерности $k \times k$. Эти секции соединены между собой следующим образом: i -й выход j -го коммутатора первой секции соединен постоянно с j -м входом i -го коммутатора второй секции, а i -й вход j -го коммутатора третьей секции соединен с j -м выходом i -го коммутатора второй секции для $1 \leq i \leq n$, $1 \leq j \leq k$. Трехсекционная сеть Клоса типа $\langle 3, 2 \rangle$ показана на рис. 4.6.

Теорема 4.20. (Слепьян [62], Дигид [10]). *Трехсекционная сеть Клоса является перестраиваемой сетью.*

Доказательство. Пусть φ — произвольное взаимно однозначное отображение X на Y . Обозначим через A_i множество входов i -го коммутатора первой секции, через C_i — множество выходов i -го коммутатора третьей секции и

$$B_i = \{x \in X : \varphi(x) \in C_i\}.$$

Очевидно, что множества A_1, \dots, A_n , как и множества B_1, \dots, B_n , определяют два разбиения множества X на блоки мощностью k каждый. В силу теоремы 4.19 множество X можно представить в виде прямой суммы $X = X_1 \cup \dots \cup X_k$, где каждое из множеств X_j является множеством элементов некоторой общей системы различных представителей для $\langle A_1, \dots, A_n \rangle$ и $\langle B_1, \dots, B_n \rangle$. Отметим, что для $j = 1, \dots, k$ «абонентов» из множества X_j мы можем соединить с соответствующими «абонентами» в Y через j -й коммутатор центральной секции. Дей-

ствительно, пусть $X_j = \{a_1, \dots, a_n\}$, где $a_i \in A_i$, и $a_j \in B_{\sigma(i)}$, т.е. $\varphi(a_i) = C_{\sigma(i)}$ для $1 \leq i \leq n$ и некоторой перестановки σ множества $\{1, \dots, n\}$. Коммутаторы первой секции, содержащие a_1, \dots, a_n , как и коммутаторы третьей секции, содержащие $\varphi(a_1), \dots, \varphi(a_n)$, попарно различны, а следовательно, для $i = 1, \dots, n$ мы можем построить цепь, реализующую соединение a_i с $\varphi(a_i)$, соединяя a_i с j -м выходом i -го коммутатора первой секции, затем i -й вход с $\sigma(i)$ -м выходом j -го коммутатора второй секции и, наконец, j -й вход $\sigma(i)$ -го коммутатора третьей секции с $\varphi(a_i)$.

На рис. 4.6 жирной линией показано соединение, реализующее следующее соответствие φ :

$$\varphi(x_1) = y_4, \varphi(x_2) = y_2, \varphi(x_3) = y_5, \varphi(x_4) = y_6, \varphi(x_5) = y_3, \varphi(x_6) = y_1, \\ (X_1 = \{x_1, x_3, x_6\}, X_2 = \{x_2, x_4, x_5\}).$$

Предположим теперь, что N имеет вид 2^p (если это не так, увеличиваем N до ближайшей степени двойки, т.е. до $2^{\lceil \log N \rceil}$). Построим трехсекционную сеть Клоса типа $\langle N/2, 2 \rangle$. Если каждый из двух коммутаторов средней секции мы заменим затем трехсекционной сетью Клоса типа $\langle N/4, 2 \rangle$, то очевидно, что полученная сеть (с пятью секциями) останется перестраиваемой. Аналогичные действия мы можем продолжать до тех пор, пока все коммутаторы не будут размерности 2×2 . Нетрудно заметить, что полученная перестраиваемая сеть состоит из $2p + 1$ секций, каждая из которых включает $N/2$ коммутаторов размерности 2×2 . Следовательно, общее число переключателей равно

$$(2p + 1) \cdot (N/2) \cdot 4 = 2N \cdot (2p + 1) = 2N(2 \log N + 1) = O(N \log N).$$

Другим применением теоремы 4.19 является задача составления расписания занятий. В простейшей наиболее идеализированной формулировке эта задача представлена некоторым множеством занятий X мощности nk и двумя разбиениями $X = W_1 \cup \dots \cup W_n$ и $X = S_1 \cup \dots \cup S_n$, где W_i — множество занятий, проводимых i -м преподавателем, а S_i — множество занятий, проводимых в i -й аудитории, причем предполагаем, что $|W_i| = |S_i| = k$, $1 \leq i \leq n$. Построение, проведенное в доказательстве теоремы 4.19, дает k общих систем различных представителей для последовательностей $\langle W_1, \dots, W_n \rangle$ и $\langle S_1, \dots, S_n \rangle$, причем эти системы в сумме исчерпывают все множество занятий X . Если, скажем, каждое занятие длится 1 час, то, проводя занятия, определенные j -й системой в течение j -го часа, $j = 1, \dots, k$, мы получаем расписание занятий, в котором все занятия проводятся в течение k часов, причем все это время заняты каждая аудитория и каждый преподаватель. На практике составление расписания занятий является более сложным, потому что появляются дополнительные ограничения (часто противоречивые), возникающие, например, из-за того, что

некоторые учащиеся хотят быть слушателями многих курсов, которые тем самым не могут проходить одновременно.

4.5 Разложение на цепи

Семейство \mathcal{C} подмножеств конечного множества X называется *цепью*, если для произвольных $A, B \in \mathcal{C}$ имеем $A \subseteq B$ или $B \subseteq A$. Другими словами, \mathcal{C} является цепью, если это семейство можно представить как $\mathcal{C} = \{C_1, \dots, C_k\}$, где $k = |\mathcal{C}|$ и $C_1 \subset C_2 \subset \dots \subset C_k$. В данном параграфе мы займемся разложением семейства $\wp(X)$ всех подмножеств множества X на минимальное число цепей.

Пусть $n = |X|$. Цепь W будем называть *симметричной цепью* в $\wp(X)$, если она имеет вид

$$C_{\lfloor n/2 \rfloor - j} \subset C_{\lfloor n/2 \rfloor - j + 1} \subset \dots \subset C_{\lfloor n/2 \rfloor + j},$$

где $0 \leq j \leq \lfloor n/2 \rfloor$ и $|C_i| = i$ для $\lfloor n/2 \rfloor - j \leq i \leq \lfloor n/2 \rfloor + j$.

Покажем теперь построение, которое из разбиения семейства $\wp(X)$ на симметричные цепи строит разбиение семейства $\wp(X \cup \{a\})$ ($a \notin X$) на симметричные цепи. Для этого каждую симметричную цепь $A_1 \subset A_2 \subset \dots \subset A_k$ в $\wp(X)$, принадлежащую нашему разбиению, заменим сначала следующими двумя цепями в $\wp(X \cup \{a\})$:

$$A_1 \subset A_2 \subset \dots \subset A_k, \quad (4.18)$$

$$A_1 \cup \{a\} \subset A_2 \cup \{a\} \subset \dots \subset A_k \cup \{a\} \quad (4.19)$$

Очевидно, что таким образом мы получаем некоторое разбиение семейства $\wp(X \cup \{a\})$ на цепи. Цепи (4.18) и (4.19) не будут симметричными в $\wp(X \cup \{a\})$, но нетрудно заметить, что их можно сделать такими, перенеся последнее множество из второй цепи в первую:

$$A_1 \subset A_2 \subset \dots \subset A_k \subset A_k \cup \{a\}, \quad (4.20)$$

$$A_1 \cup \{a\} \subset A_2 \cup \{a\} \subset \dots \subset A_{k-1} \cup \{a\} \quad (4.21)$$

Если $k = 1$, то вторая из приведенных выше цепей исчезает, т.е. все подмножества, появляющиеся в (4.18) и (4.19), мы помещаем в одну симметричную цепь. Начиная с семейства $\wp(\emptyset) = \{\emptyset\}$, которое само является симметричной цепью, и повторяя описанное выше построение n раз, получаем разбиение семейства $\wp(X)$ на симметричные цепи.

Антицепью в $\wp(X)$ будем называть такое произвольное семейство $\mathcal{A} \subseteq \wp(X)$, что для произвольных $A, B \in \mathcal{A}$, $A \neq B$, имеем $A \not\subseteq B$ и $B \not\subseteq A$. Примером антицепи является $\wp_k(X)$, семейство всех k -элементных подмножеств множества X ($0 \leq k \leq |X|$). Пусть $\wp(X) = \mathcal{C}_1 \cup \dots \cup \mathcal{C}_p$ — произвольное разбиение семейства

$\wp(X)$ на непересекающиеся цепи, и пусть A — произвольная антицепь в $\wp(X)$. Ясно, что не более одного множества $A \in \mathcal{A}$ попадает в каждую из цепей C_i и что отсюда вытекает $p \geq |\mathcal{A}|$. Следовательно, имеем

Следствие 4.21. *Мощность каждой антицепи в $\wp(X)$ не превышает числа блоков в произвольном разбиении семейства $\wp(X)$ на цепи.*

Если бы нам удалось указать такую антицепь \mathcal{A} в $\wp(X)$ и такое разбиение на цепи $\wp(X) = C_1 \cup \dots \cup C_p$, при котором $|\mathcal{A}| = p$, то очевидно, что \mathcal{A} было бы максимальной антицепью в $\wp(X)$, а наше разбиение было бы разбиением семейства $\wp(X)$ на минимальное число цепей. В сущности легко убедиться, что такие антицепь и разбиения существуют. Достаточно принять $\mathcal{A} = \wp_{\lfloor n/2 \rfloor}(X)$ ($n = |X|$), а в качестве разбиения рассмотреть произвольное разбиение семейства $\wp(X)$ на симметричные цепи, например разбиение, полученное применением описанного рекурсивного построения. Это следует из того, что каждая симметричная цепь содержит в точности одно $\lfloor n/2 \rfloor$ -элементное множество. Поскольку цепи не пересекаются и покрывают $\wp(X)$, то их должно быть в точности столько, сколько имеется $\lfloor n/2 \rfloor$ -элементных подмножеств. Вспомним, что число k -элементных подмножеств n -элементного множества равно биномиальному коэффициенту $\binom{n}{k}$ (см. разд. 1.6), и упорядочим полученные факты.

Теорема 4.22 (Шпернер [63]). *Мощность произвольной антицепи в $\wp(X)$, $|X| = n$, не превышает $\binom{n}{\lfloor n/2 \rfloor}$, т.е. $\wp_{\lfloor n/2 \rfloor}(X)$ — максимальная антицепь.*

Теорема 4.23. *Каждое разбиение семейства $\wp(X)$ на симметричные цепи состоит из $\binom{n}{\lfloor n/2 \rfloor}$ цепей, где $n = |X|$, и является разбиением семейства $\wp(X)$ на минимальное возможное число цепей.*

Опишем теперь детальную реализацию предложенного рекурсивного метода построения разбиения семейства $\wp(X)$ на цепи. Примем $X = \{1, \dots, n\}$ и представим каждую симметричную цепь C в $\wp(X)$ записью, содержащей массив $P[1..n]$ и переменные нач и конец, такие что

$$C = \{ \{ P[1], P[2], \dots, P[i] \} : \text{нач} \leq i \leq \text{конец} \}.$$

Кроме того, такая запись содержит указатель след на запись, представляющую следующую цепь разбиения.

Алгоритм 4.24. (Нахождение разбиения семейства всех подмножеств множества $\{1, \dots, n\}$ на симметричные цепи.)

Данные: n .

Результаты: Список записей, каждая из которых представляет блок разбиения семейства $\wp(\{1, \dots, n\})$ на симметричные цепи (разб является указателем на начало этого списка).

1 **function** РАЗБИЕНИЕ(k); (*значением этой функции является указатель на начало списка записей, каждая из которых представляет собой

симметричную цепь $\wp(X)$, $X = \{1, \dots, k\}$; все записи в списке определяют разбиение семейства $\wp(X)$ на симметричные цепи*

```

2 begin
3   if  $k = 0$  then (*построение разбиения семейства  $\wp(\emptyset) = \{\emptyset\}$ 
   на симметричные цепи — единственной цепью является  $\{\emptyset\}$ *)
4     begin  $new(w)$ ; (* $w$  = указатель на запись, представляющую цепь*)
5       with  $w \uparrow$  do
6         begin  $нач := 0$ ;  $конец := 0$ ;  $след := nil$ 
7         end
8       end
9     else (* $k > 0$ *)
10    begin  $w := \text{РАЗБИЕНИЕ}(k - 1)$ ;
11       $pp := w$ ;
12    while  $pp \neq nil$  do
13      with  $pp \uparrow$  do
14        begin (*симметричная цепь  $A_1 \subset \dots \subset A_r$ ,
   представленная записью  $pp \uparrow$ , заменяется двумя цепями:
    $A_1 \subset A_2 \subset \dots \subset A_k \subset A_k \cup \{a\}$  и  $A_1 \cup \{a\} \subset A_2 \cup \{a\} \subset \dots \subset A_{k-1} \cup \{a\}$ ,
   причем вторая опускается, если  $r = 1$ , т.е. когда  $pp \uparrow .нач = pp \uparrow .конец$ *)
15           $послед := след$ ;
16        if  $нач < конец$  then (*цепь длины  $r > 1$ *)
17          begin (*добавить новую цепь  $A_1 \cup \{k\} \subset \dots \subset A_{r-1} \cup \{k\}$ *)
18             $new(pn)$ ;  $pn \uparrow .след := след$ ;  $след := pn$ ;
19             $pn \uparrow .нач := нач + 1$ ;  $pn \uparrow .конец := конец$ ;
20             $pn \uparrow .P[1] := k$ ;
21            for  $i := 2$  to  $конец$  do  $pn \uparrow .P[i] := P[i - 1]$ 
22          end;
   (*добавить  $A_r \cup \{k\}$  к цепи, представленной записью  $pp \uparrow$ *)
23           $конец := конец + 1$ ;  $P[конец] := k$ ;
24           $pp := послед$ 
25        end
26      end;
27     $\text{РАЗБИЕНИЕ} := w$ 
28 end; (*РАЗБИЕНИЕ*)

29 begin (*главная программа*)
30    $разб := \text{РАЗБИЕНИЕ}(n)$ 
31 end

```

Предоставляем читателю проверить, что число шагов, выполняемых алгоритмом, имеет порядок суммы длин всех полученных цепей разбиения, т.е.

$O(2^n)$. На рис. 4.7 показан список симметричных цепей, полученных при помощи этого алгоритма для $n = 5$.

	1	2	3	4	5]
[5	1	2	3]	
[4	1	2	5]	
	5	[4	1]	
[3	1	4	5]	
	5	[3	1]	
	4	[3	5]	
[2	3	4	5]	
	5	[2	3]	
	4	[2	5]	

Отметим, что в каждой из записей списка, составленного алгоритмом 4.24, элементы $P[i]$ для $i >$ конец не определены. Если их определить произвольным способом так, чтобы последовательность $P[1], \dots, P[n]$ была перестановкой, то мы получим список перестановок $\varphi_1, \varphi_2, \dots, \varphi_m$, $m = \binom{n}{\lfloor n/2 \rfloor}$, с очень любопытным свойством. Для каждого подмножества $A \subseteq \{1, \dots, n\}$ существует такая перестановка φ , что множество первых $|A|$ членов этой перестановки равно A (перестановку мы трактуем здесь как последовательность

Рис. 4.7: Список симметричных цепей, построенных алгоритмом 4.24 для $n = 5$ («[» предшествует $P[\text{нач}]$ а «]» следует за $P[\text{конец}]$, нач = 0 для первой цепи).

длины n). Отметим, что система меньшего числа перестановок с приведенным выше свойством не может существовать, поскольку каждое из $m = \binom{n}{\lfloor n/2 \rfloor} \lfloor n/2 \rfloor$ -элементных подмножеств должно появляться как начальный участок в какой-либо перестановке.

Покажем теперь применение такой системы перестановок для организации картотек, автором которой является Лум (см. [47]). Для этого рассмотрим ситуацию, когда запись в некоторой картотеке описывается с помощью n атрибутов. Предположим, что i -й атрибут для каждой конкретной записи может принимать одно из n_i значений; для простоты обозначим эти значения $1, \dots, n_i$. Пусть R будет множеством записей в нашей картотеке. Тогда каждая запись r описывается последовательностью $\langle w_1(r), \dots, w_n(r) \rangle$, где $w_i(r)$ — значение i -го атрибута для r ($1 \leq w_i(r) \leq n_i$). Мы можем рассортировать нашу картотеку «лексикографически» многими способами, если принять во внимание то, что понятие лексикографического порядка предполагает некоторое упорядочение координат — от координат, значения которых «изменяются медленно», до координат, значения которых «изменяются быстрее всего». Точнее, для произвольной перестановки φ множества $\{1, \dots, n\}$ мы определяем порядок \leq_φ следующим образом:

$$\langle a_1, \dots, a_n \rangle \leq_\varphi \langle b_1, \dots, b_n \rangle \Leftrightarrow \langle a_1, \dots, a_n \rangle = \langle b_1, \dots, b_n \rangle \text{ или } \exists k, 1 \leq k \leq n, \text{ такое что } a_{\varphi(i)} = b_{\varphi(i)} \text{ для } 1 \leq i < k \text{ и } a_{\varphi(k)} = b_{\varphi(k)}.$$

В методе Лума используется $\binom{n}{\lfloor n/2 \rfloor}$ копий картотек, из которых i -я, обозначим ее K_i , сортируется в соответствии с порядком на последовательностях

$\langle w_1(r), \dots, w_n(r) \rangle$. Предположим, что каждый запрос, направленный в картотеку, имеет вид $Q = \langle x_1, \dots, x_n \rangle$, где каждый член x_n является либо некоторым значением i -го атрибута, $1 \leq x_i \leq n_i$, либо $x_i = *$, что означает, что запрос не налагает никаких условий на значение i -го атрибута. Ответ на запрос $\langle x_1, \dots, x_n \rangle$ обозначим $\|\langle x_1, \dots, x_n \rangle\|$ и определим следующим образом:

$$\|\langle x_1, \dots, x_n \rangle\| = \{r \in R : w_i(r) = x_i \text{ или } w_i(r) = * \text{ для } 1 \leq i \leq n\}.$$

Суть метода Лума состоит в том, что для произвольного запроса Q существует такое i , что ответ состоит из отрезка последовательных (относительно порядка \leq_φ записей в картотеке K_i). Чтобы доказать это свойство, рассмотрим произвольный запрос $Q = \langle x_1, \dots, x_n \rangle$. Согласно основному свойству множества перестановок $\varphi_1, \dots, \varphi_m$ среди них найдется такая перестановка φ_i , которая переводит в начало Q все координаты x_i , отличные от $*$, т.е. $x_{\varphi_i(j)} \leq *$ для $1 \leq j \leq k$ и $x_{\varphi_i(j)} = *$ для $k < i \leq n$.

Ясно, что в K_i ответ $\|Q\|$ есть отрезок последовательных записей, начинающийся с записи r , такой что

$$\langle w_{\varphi_i(1)}(r), \dots, w_{\varphi_i(n)}(r) \rangle = \langle x_{\varphi_i(1)}, \dots, x_{\varphi_i(k)}, 1, \dots, 1 \rangle,$$

и кончающийся такой записью r' , что

$$\langle w_{\varphi_i(1)}(r'), \dots, w_{\varphi_i(n)}(r') \rangle = \langle x_{\varphi_i(1)}, \dots, x_{\varphi_i(k)}, n_{\varphi_i(k+1)}, \dots, n_{\varphi_i(n)} \rangle,$$

(ситуация существенно не меняется, когда в нашей картотеке нет записи r или r').

Если наши картотеки K_i запоминаются на носителе информации с линейной структурой, то тот факт, что все записи, которые мы хотим найти, составляют связный отрезок, значительно облегчает процесс поиска. И очевидно, что главным неудобством, особенно для больших n , является большое число копий нашей картотеки, которые необходимо хранить в памяти.

В заключение данного раздела стоит отметить, что большинство рассмотренных нами теорем имело минимальный характер: в них говорилось, что минимум некоторой величины равен максимуму некоторой другой величины. Примерами могут служить теорема о максимальном потоке и минимальном разрезе, венгерская теорема и теоремы 4.22 и 4.23. Оказывается, что последний пример есть частный случай одной значительно более общей теоремы Дилворта (см. [12]), касающейся частично упорядоченного конечного множества. Так же как и для случая частично упорядоченного множества $\langle \varphi(X), \subseteq \rangle$, для произвольного частично упорядоченного множества $\langle P, \leq \rangle$ цепь определяется как такое произвольное подмножество $L \subseteq P$, что $x \leq y$ или $y \leq x$ для произвольных $x, y \in L$,

а антицепь определяется как такое произвольное подмножество $A \subseteq P$, что для произвольных $x, y \in A$

$$x \leq y \Rightarrow x = y.$$

А это обещанная теорема Дилворта.

Теорема 4.25. *Для произвольного конечного частично упорядоченного множества $\langle P, \leq \rangle$ минимальное число цепей, которые в сумме покрывают P , равно максимальной мощности антицепи.*

Доказательство. Для доказательства воспользуемся индукцией относительно мощности множества P . Если $|P| \leq 1$, то очевидно, что теорема верна. Пусть $|P| > 1$, L — произвольная максимальная цепь (т.е. цепь, не являющаяся подмножеством никакой большей цепи), и пусть m — максимальная мощность антицепи в $\langle P, \leq \rangle$. Покажем, что наше множество P можно покрыть m цепями. Если максимальная мощность антицепи в $\langle P \setminus L, \leq \rangle$ равна $m - 1$ (очевидно, что она не может быть меньше $m - 1$), то в силу индуктивного предположения существует разбиение на цепи $P \setminus L = L_1 \cup \dots \cup L_{m-1}$, а отсюда получаем требуемое разбиение $P = L_1 \cup \dots \cup L_{m-1} \cup L$. Предположим, что в $\langle P \setminus L, \leq \rangle$ существует m -элементная антицепь $A = \{a_1, \dots, a_m\}$. Построим множества элементов, находящихся «над» A :

$$G = \{x \in P : x > a \text{ для некоторого } a \in A\}$$

и «под» A :

$$D = \{x \in P : x < a \text{ для некоторого } a \in A\}$$

Предположим, что L имеет вид $l_1 < \dots < l_k$. Тогда $b_1 \notin G$, ибо в противном случае цепь L можно было бы увеличить на некоторый элемент $a < l_1$, $a \in A$, вопреки предположению о максимальной цепи L . Аналогичным образом доказывается, что $l_k \notin D$. Следовательно, $|G| < |P|$, $|D| < |P|$ и в силу индуктивного предположения, примененного к $\langle G, \leq \rangle$ и $\langle D, \leq \rangle$, существуют разбиения на цепи

$$G = G_1 \cup \dots \cup G_m \text{ и } D = D_1 \cup \dots \cup D_m.$$

После возможной перенумерации множеств цепей G_1, \dots, G_m и D_1, \dots, D_m мы можем предполагать, что $a_i \in G_i \cap D_i$, $1 \leq i \leq m$. Но $G \cup D = P$, поскольку существование элемента $b \in P \setminus (G \cup D)$ противоречило бы максимальной мощности антицепи A . Следовательно, получаем требуемое разбиение на цепи

$$P = L_1 \cup \dots \cup L_n,$$

полагая $L_i = G_i \cup D_i$, $i = 1, \dots, m$.

4.6 Задачи

4.1. Доказать, что каждый поток f из s в t можно представить в виде суммы $f = f_1 + \dots + f_k$, $k \leq m$ (т.е. $f(e) = f_1(e) + \dots + f_k(e)$ для каждой дуги $e \in E$), где f_i — поток вдоль отдельного пути из s в t , $i = 1, \dots, k$. Вывести отсюда, что можно получить максимальный поток, поочередно увеличивая поток вдоль соответственно выбранных увеличивающих цепей со всеми согласованными дугами, начав с нулевого потока. Построить сеть, в которой $k = \Omega(m)$ для каждого такого разбиения $f = f_1 + \dots + f_k$.

4.2. Доказать, что если каждое увеличение потока происходит вдоль кратчайшей увеличивающей цепи относительно существующего потока, то длины последовательных цепей образуют неубывающую последовательность. (Следует отметить, что из этого свойства легко вытекает лемма 4.4, однако обратное следование не является таким очевидным.)

4.3. Доказать, что если каждое увеличение потока происходит вдоль кратчайшей увеличивающей цепи, то, начиная с произвольного потока, мы получаем максимальный поток с использованием не более чем $mn/2$ увеличивающих цепей (см. [17]).

4.4. Показать, что для сети с пропускной способностью каждой дуги, равной нулю или единице, алгоритм Диница можно реализовать за время $O(n^{2/3}m)$ (см. [18]).

4.5. Показать, что если в сети с целочисленными пропускными способностями потенциал каждой вершины, отличной от s и t , равен нулю или единице, то алгоритм Диница может быть реализован за время $O(n^{1/2}m)$ (см. [18]).

4.6. Рассмотреть сеть со многими выделенными источниками s_1, \dots, s_p и стоками t_1, \dots, t_q и заданными *продуктивностями* источников $a_1, \dots, a_p \geq 0$ и *спросами* стоков $b_1, \dots, b_q \geq 0$, такими что $a_1 + \dots + a_p = b_1 + \dots + b_q$. Представить метод построения в такой сети потока f , для которого $\text{Div}_f(s_i) = a_i$ для $1 \leq i \leq p$, $\text{Div}_f(t_j) = -b_j$ для $1 \leq j \leq q$ и $\text{Div}_f(u) = 0$ для всех остальных вершин сети, если такой поток существует. (*Указание:* добавить к сети дополнительные дуги $\langle s, s_i \rangle$ с пропускными способностями a_i для $i = 1, \dots, p$ и дуги $\langle t_j, t \rangle$ с пропускными способностями b_j , для $j = 1, \dots, q$).

4.7. Добавить к задаче о потоке в сети дополнительное ограничение, состоящее в том, что для каждой вершины v , отличной от s и t , количество потока, втекающего в v и вытекающего из v , не может превышать пропускной способности $g(v)$ этой вершины. Как свести задачу, измененную таким способом, к первоначальной задаче без пропускных способностей вершин? (*Указание:* для каждой вершины v добавить новую вершину v^* , дугу $\langle v, v^* \rangle$ с пропускной способностью $g(v)$ и заменить каждую дугу вида $\langle v, u \rangle$ на $\langle v^*, u \rangle$).

4.8. Пусть G — орграф, не содержащий дуги $\langle s, t \rangle$. Доказать, что максималь-

ное число путей в G из s в t с попарно различными промежуточными вершинами (т.е. отличными от s и t) равно минимальному числу промежуточных вершин, после удаления которых в графе не найдется ни одного пути из s в t (см. [52]). (Указание: рассмотреть G как сеть с пропускной способностью каждой вершины, равной единице — ср. с предыдущей задачей, — и воспользоваться теоремой о максимальном потоке и минимальном разрезе.)

4.9. Написать алгоритм нахождения дуг с таким свойством, что увеличение пропускной способности дуги приводит к увеличению потока. Всегда ли существует такая дуга? (Указание: найти максимальный поток f , а затем множество A вершин, достижимых с помощью частичных увеличивающих цепей из s , и множество вершин B , из которых аналогичным образом можно достичь t . Требуемое множество — это $(A \times B) \cap E$.)

4.10. Рассмотреть сеть, в которой каждой дуге $e \in E$ приписана, кроме пропускной способности, стоимость $h(e)$ посылки единицы потока через e . Определить стоимость увеличивающей цепи как сумму стоимостей согласованных дуг минус сумму стоимостей несогласованных дуг этой цепи. Доказать, что поток f величины $w = W(f)$ имеет стоимость, наименьшую из всех потоков величины w , тогда и только тогда, когда не существует увеличивающего цикла (замкнутой цепи) относительно f с отрицательной стоимостью. Показать, что увеличение на δ потока величины w наименьшей стоимости вдоль увеличивающей цепи с наименьшей стоимостью приводит к потоку с наименьшей стоимостью величины $w + \delta$.

4.11. Доказать, что если P — кратчайшая чередующаяся цепь относительно паросочетания M в двудольном графе, P' — кратчайшая чередующаяся цепь относительно $M \oplus P$, то $|P'| \geq |P| + |P \cap P'|$ (см. [33]).

4.12. Доказать, что если M и N — паросочетания в двудольном графе H , то существует такое паросочетание R , что вершина графа H будет свободной относительно R тогда и только тогда, когда она свободна относительно как M , так и N (см. [51], см. также лемму 4.11).

4.13. Двудольный граф $H = \langle X, Y, E \rangle$ называется *выпуклым* на X , если множество X можно упорядочить в последовательность x_1, \dots, x_p так, чтобы для каждого $y \in Y$ множество $A(y) = \{x \in X : x - y\}$ образовывало отрезок вида $\{x_{P(y)}, x_{P(y)+1}, \dots, x_{K(y)}\}$. Подобным же образом определяем выпуклость на Y . Доказать, что если граф H выпуклый на X , то наибольшее паросочетание можно получить, просматривая последовательность x_1, \dots, x_p и сочетая вершину x_i с той из свободных еще вершин $y \in Y$, $x - y$, для которой значение $K(y)$ минимальное (или оставляя вершину x_i свободной, если такой вершины y не существует) (см. [28]).

4.14. Показать, что наибольшее паросочетание в двудольном графе $H = \langle X, Y, E \rangle$, выпуклом на X (см. предыдущую задачу), можно построить за время

$O(|X| + |Y| \cdot \log |Y|)$ и за время $O(|X| + |Y|)$, если H — выпуклый как на X , так и на Y . (*Внимание:* предполагаем, что граф представлен массивами значений $P(y)$ и $K(y)$, $y \in Y$; здесь следует отметить, что оценку $O(|X| + |Y| \cdot \log |Y|)$ можно значительно улучшить, как это показано в работе [46], пользуясь некоторой общей методикой, изложенной в [67] и автором которой является Р. Тарьян.)

4.15. Привести пример графа (недвудольного), для которого не выполнено свойство, выделенное в следствии 4.16.

4.16. Доказать следующую теорему, двойственную по отношению к венгерской теореме: для произвольной 0-1-матрицы максимальное число единиц в линии равно минимальному числу рассеянных множеств единиц, покрывающих все единицы матрицы (множество единиц называется *рассеянным*, если никакие две единицы не лежат на одной линии). Сформулировать теорему в терминах паросочетаний в двудольном графе.

4.17. *Бистохастической матрицей* называется матрица размерности $n \times n$ с вещественными неотрицательными элементами, в которой сумма элементов в каждой строке и в каждом столбце равна единице. *Перестановочной матрицей* называется произвольная бистохастическая 0-1-матрица. Доказать, что каждую бистохастическую матрицу B можно представить в виде комбинации $B = \mu_1 P_1 + \dots + \mu_k P_k$, где P_1, \dots, P_k — перестановочные матрицы, $\mu_1, \dots, \mu_k > 0$ и $\mu_1 + \dots + \mu_k = 1$ (см. [6]).

4.18. *Перманент* матрицы $A = [a_{ij}]$ размерности $n \times n$ определяется следующим образом:

$$\text{per } A = \sum a_{1,\sigma(1)}, a_{2,\sigma(2)}, \dots, a_{n,\sigma(n)},$$

где суммирование ведется по всем перестановкам s множества $\{1, \dots, n\}$. Доказать, что перманент 0-1-матрицы A размерности $n \times n$ равен нулю тогда и только тогда, когда A содержит нулевую подматрицу размерности $p \times p$, $p + r = n + 1$ (см. [24]).

4.19. Уточнить детали алгоритма нахождения общей системы различных представителей для последовательностей $\langle A_1, \dots, A_n \rangle$ и $\langle B_1, \dots, B_n \rangle$ со сложностью $O\left(\sqrt{n} \sum_{j=1}^n (|A_j| + |B_j|)\right)$, приведенного в разд. 4.4 (см. задачу 4.5).

4.20. Привести построение последовательности с элементами из множества $\{1, \dots, n\}$ длины $l_n \approx \frac{2}{\pi} 2^n$ с таким свойством, что каждое подмножество $A \subseteq \{1, \dots, n\}$ появляется в этой последовательности как подпоследовательность из $|A|$ последовательных членов [43]. *Указание:* построить $\binom{k}{\lfloor k/2 \rfloor}$ перестановок множества $\{1, \dots, k\}$, $k = \lfloor n/2 \rfloor$, таких что каждое подмножество $K \subseteq \{1, \dots, k\}$ появляется как конечный отрезок некоторой из этих последовательностей, и $\binom{p}{\lfloor p/2 \rfloor}$, $p = \lfloor n/2 \rfloor$, перестановок множества $\{k + 1, \dots, n\}$, таких что каждое подмножество $L \subseteq \{k + 1, \dots, n\}$ появляется как начальный отрезок некоторой из

этих перестановок. Далее следует соединить эти перестановки соответствующим образом. *Внимание:* в силу формулы Стирлинга

$$\binom{k}{\lfloor k/2 \rfloor} \approx \sqrt{\frac{2}{\pi k}} 2^k.$$

Глава 5

Матроиды

5.1 Жадный алгоритм решения оптимизационных задач

Рассмотрим следующую матрицу с действительными неотрицательными коэффициентами:

$$A = \begin{bmatrix} 7 & 5 & 1 \\ 3 & 4 & 3 \\ 2 & 3 & 1 \end{bmatrix}$$

Займемся решением следующей оптимизационной задачи.

Задача 1. Найти такое подмножество элементов матрицы, что

- (а) в каждом столбце находится не более одного выбранного элемента
- (б) сумма выбранных элементов является наибольшей из возможных.

Попробуем решить эту задачу следующим образом: будем выбирать элементы последовательно, причем каждый раз будем выбирать наибольший из элементов, которые мы можем добавить, не нарушая условия (а). Будем действовать так вплоть до момента, пока добавление произвольного элемента не нарушит условия (а). Алгоритм такого типа мы будем называть *жадным*.

Для задачи 1 и матрицы A жадный алгоритм находит подмножество

$$\begin{bmatrix} \textcircled{7} & \textcircled{5} & 1 \\ 3 & 4 & \textcircled{3} \\ 2 & 3 & 1 \end{bmatrix}$$

которое действительно дает наибольшую возможную сумму. Нетрудно отметить, что с помощью жадного алгоритма правильно находится решение задачи

1 для произвольной вещественной матрицы с неотрицательными элементами. Рассмотрим несколько иную задачу.

Задача 2. Найти такое подмножество элементов матрицы, что

(а) в каждом столбце и в каждой строке находится не более одного выбранного элемента

(б) сумма выбранных элементов является наибольшей из возможных.

На этот раз применение жадного алгоритма к задаче 2 и матрице A дает подмножество

$$\begin{bmatrix} \textcircled{7} & 5 & 1 \\ 3 & \textcircled{4} & 3 \\ 2 & 3 & \textcircled{1} \end{bmatrix}$$

с суммой 12, что не является правильным решением, поскольку подмножество

$$\begin{bmatrix} \textcircled{7} & 5 & 1 \\ 3 & 4 & \textcircled{3} \\ 2 & \textcircled{3} & 1 \end{bmatrix}$$

имеет сумму 13. Следовательно, на втором шаге не следовало бы быть жадным, мы выигрываем в конечном результате, выбирая несколько меньший элемент (3 вместо 4).

Возникает вопрос: когда выгодно быть жадным? Сформулируем это более строго.

Мы будем рассматривать оптимизационные задачи следующего типа.

Задача 3. Даны конечное множество E , семейство его подмножеств $\mathfrak{J} \subseteq \wp(E)$ и функция $w : E \rightarrow \mathbb{R}^+$, где \mathbb{R}^+ обозначает множество вещественных неотрицательных чисел. Найти подмножество $S \in \mathfrak{J}$ с наибольшей суммой $\sum_{e \in S} w(e)$.

Как задача 1, так и задача 2 являются частными случаями задачи 3. В обоих случаях E есть множество позиций матрицы, а w ставит в соответствие позиции $\langle i, j \rangle$ матрицы $[a_{ij}]$ число a_{ij} .

Для задачи 1

$$S \in \mathfrak{J} \Leftrightarrow \text{каждый столбец содержит не более одной позиции из множества } S.$$

а в случае задачи 2

$$S \in \mathfrak{J} \Leftrightarrow \text{каждый столбец и каждая строка содержит не более одной позиции из множества } S.$$

Теперь мы можем сформулировать наш вопрос следующим образом: при каких условиях относительно семейства \mathfrak{J} жадный алгоритм правильно решает задачу 3 для произвольной функции w ?

Оказывается, что на этот вопрос можно найти простой ответ. А именно, достаточно, чтобы пара $\langle E, \mathcal{I} \rangle$ образовывала так называемый матроид. Следующий раздел данной главы посвящен этим важным комбинаторным объектам (читателя, заинтересованного в более глубоком изучении теории матроидов, отсылаем к [42], [72]).

5.2 Матроиды и их основные свойства

Матроиды были введены Уитни в работе [73] в совершенно ином контексте, нежели жадные алгоритмы, а именно в исследованиях абстрактной теории линейной зависимости. Существует много эквивалентных определений матроида. Для нас наиболее удобным будет следующее определение:

Матроидом мы будем называть произвольную пару $M = \langle E, \mathcal{I} \rangle$, где E — конечное множество, а $\mathcal{I} \subseteq \wp(E)$ — семейство, удовлетворяющее условиям:

- M1 $\emptyset \in \mathcal{I}$ и если $A \in \mathcal{I}$ и $B \subseteq A$, то $B \in \mathcal{I}$.
- M2 Для произвольных $A, B \in \mathcal{I}$, таких что $|B| = |A| + 1$, существует элемент $e \in B \setminus A$, такой что $A \cup \{e\} \in \mathcal{I}$.

(Условие $\emptyset \in \mathcal{I}$ исключает вырожденный случай $\mathcal{I} = \emptyset$.)

Множества семейства \mathcal{I} мы будем называть *независимыми множествами*, а остальные подмножества из $\wp(E) \setminus \mathcal{I}$ — *зависимыми множествами* матроида M . В этом проявляется уже упомянутая связь матроидов с теорией линейной зависимости: аксиомы матроидов выбраны таким образом, чтобы они отражали наиболее характерные свойства независимых множеств линейного пространства, точнее говоря, независимых подмножеств, содержащихся в некотором конечном подмножестве этого пространства (вспомним, что подмножество $\{e_1, \dots, e_n\}$ линейного пространства называется независимым, если не существует набора скаляров $\lambda_1, \dots, \lambda_n$, не всех равных нулю, такого что $\lambda_1 e_1 + \dots + \lambda_n e_n = 0$). В самом деле, очевидно, что произвольное подмножество независимого множества линейного пространства независимо. Если $|B| = |A| + 1$ для линейно независимых подмножеств A, B линейного пространства, то A порождает пространство размерности $|A|$, которое может содержать не более чем $|A|$ элементов множества B . Следовательно, существует элемент $e \in B \setminus A$, не принадлежащий этому подпространству. Множество $A \cup \{e\}$ порождает подпространство размерности $|A| + 1$ и, следовательно, является линейно независимым.

Будем говорить, что два матроида $\langle E, \mathcal{I} \rangle$ и $\langle E', \mathcal{I}' \rangle$ *изоморфны*, если существует взаимно однозначное отображение f множества E на множество E' , такое что $A \in \mathcal{I}$ тогда и только тогда, когда $f(A) \in \mathcal{I}'$. Часто мы не будем делать различия между изоморфными матроидами.

Для произвольного подмножества $C \subseteq E$ мы будем изучать его максимальные независимые подмножества, т.е. независимые подмножества $A \subseteq C$, обладающие таким свойством: не существует независимого подмножества B , такого что $A \subset B \subseteq C$.

Теорема 5.1. Пусть E — конечное множество, \mathcal{I} — семейство его подмножеств, удовлетворяющих условию М1. При этих предположениях $M = \langle E, \mathcal{I} \rangle$ является матроидом тогда и только тогда, когда удовлетворяется условие

- М3 Для произвольного подмножества $C \subseteq E$ каждые два максимальных подмножества множества C имеют одинаковую мощность.

Доказательство. Предположим, что $M = \langle E, \mathcal{I} \rangle$ является матроидом и для некоторого $C \subseteq E$ существуют два максимальных независимых подмножества $A, B \subseteq C$, такие что $|B| > |A|$. Выберем произвольное подмножество $B' \subseteq B$ (независимое по условию М1), такое что $|B'| = |A| + 1$. В силу М2 существует такой элемент $e \in B' \setminus A \subseteq C$, что $A \cup \{e\} \in \mathcal{I}$ вопреки максимальнойности множества A .

Напротив, предположим, что выполняются условия М1 и М3. Выберем такие произвольные подмножества $A, B \in \mathcal{I}$, что $|B| = |A| + 1$, и обозначим $C = A \cup B$. Допустим, что не существует такого элемента $e \in B \setminus A$, что $A \cup \{e\} \in \mathcal{I}$. Это означало бы, что A является максимальным независимым подмножеством множества C . Расширяя B до максимального независимого подмножества $B^* \subseteq C$, мы имели бы $|A| < |B^*|$ вопреки условию М3. Следовательно, условия М1 и М3 влекут за собой условие М2.

Из теоремы 5.1 следует, что условия М1 и М3 образуют эквивалентную систему аксиом для матроидов.

Мощность максимального подмножества множества $C \subseteq E$ называется *рангом* этого множества и обозначается через $r(C)$:

$$r(C) = \max\{|A| : A \in \mathcal{I} \wedge A \subseteq C\}.$$

Очевидно, что подмножество $C \subseteq E$ является независимым тогда и только тогда, когда $r(C) = |C|$. Каждое максимальное независимое множество матроида $M = \langle E, \mathcal{I} \rangle$ будем называть (по аналогии с линейными пространствами) *базой* этого матроида, а ранг $r(E)$, являющийся аналогом размерности линейного пространства, рангом матроида. Отметим важное следствие теоремы 5.1.

Следствие 5.2. *Каждые две базы матроида имеют одинаковое число элементов.*

Отметим также, что каждое независимое множество $C \in \mathcal{I}$ можно расширить до базы $B \supseteq C$; достаточно поочередно добавлять в C новые элементы, присоединение которых не нарушает независимости, вплоть до момента, когда таких элементов больше не существует. Полученное множество будет максимальным

независимым множеством, а следовательно, базой. Аналогично, произвольное независимое множество $A \subseteq C$ можно расширить до максимального независимого подмножества множества C . Отметим некоторые свойства рангов.

Теорема 5.3. *Для произвольных $A, B \subseteq E$ и $e, f \in E$ имеем*

- R1 $0 \leq r(A) \leq |A|$,
- R2 *если $A \subseteq B$, то $r(A) \leq r(B)$,*
- R3 $r(A \cup B) + r(A \cap B) \leq r(A) + r(B)$,
- R4 $r(A) \leq r(A \cup \{e\}) \leq r(A) + 1$,
- R5 *если $r(A \cup \{e\}) = r(A \cup \{f\}) = r(A)$, то $r(A \cup \{e, f\}) = r(A)$.*

Доказательство. Свойства R1 и R2 очевидны. Докажем R3. Пусть $\{e_1, \dots, e_p\}$ — максимальное независимое множество в $A \cap B$. Расширим его до максимального независимого множества $e_1, \dots, e_p, f_1, \dots, f_q \subseteq A$, а затем до максимального независимого множества $\{e_1, \dots, e_p, f_1, \dots, f_q, g_1, \dots, g_r\} \subseteq A \cup B$. Имеем $p = r(A \cap B)$, $p + q = r(A)$, $p + r \leq p + q + r = r(A \cup B)$, а следовательно,

$$r(A \cup B) + r(A \cap B) = (p + q + r) + p = (p + q) + (p + r) \leq r(A) + r(B).$$

Условие R4 очевидно. R5 вытекает из R3:

$$\begin{aligned} r(A) &\leq r(A \cup \{e, f\}) = r((A \cup \{e\}) \cup \{f\}) \leq \\ &\leq r(A \cup \{e\}) + r(A \cup \{f\}) - r((A \cup \{e\}) \cap (A \cup \{f\})) = \\ &= r(A) + r(A) - r(A) = r(A). \end{aligned}$$

Продолжая аналогию с линейными пространствами, будем говорить, что элемент e *зависим* от множества A , если $r(A \cup \{e\}) = r(A)$, и будем обозначать через $\text{sp}(A)$ множество всех элементов, зависящих от A :

$$\text{sp}(A) = \{e \in E : r(A \cup \{e\}) = r(A)\}.$$

Множество $A \in E$ называется *подпространством* матроида, если $A = \text{sp}(A)$, т.е. если $r(A \cup \{e\}) = r(A) + 1$ для произвольного $e \in E \setminus A$.

Теорема 5.4. *Для произвольных $A, B \subseteq E$ и $e, f \in E$ имеем*

- S1 $A \subseteq \text{sp}(A)$,
- S2 *если $A \subseteq B$, то $\text{sp}(A) \subseteq \text{sp}(B)$,*

- S3 $\text{sp}(\text{sp}(A)) = \text{sp}(A)$,
- S4 если $f \notin \text{sp}(A)$ и $f \in \text{sp}(A \cup \{e\})$, то $e \in \text{sp}(A \cup \{f\})$.

Доказательство. Условие S1 очевидно: если $e \in A$, то $r(A \cup \{e\}) = r(A)$, т.е. $e \in \text{sp}(A)$. Для доказательства S2 воспользуемся R3. Предположим, что $A \subseteq B$ и $e \in \text{sp}(A)$. Тогда

$$\begin{aligned} r(B) &\leq r(B \cup \{e\}) = r((A \cup \{e\}) \cup B) \leq \\ &\leq r(A \cup \{e\}) + r(B) - r(A \cup (B \cap \{e\})) = \\ &= r(A) + r(B) - r(A) = r(B) \end{aligned}$$

и, следовательно, $e \in \text{sp}(B)$.

Из условий S1 и S2 следует, что $\text{sp}(A) \subseteq \text{sp}(\text{sp}(A))$. Докажем теперь противоположное включение. Для этого нам будет необходимо равенство

$$r(\text{sp}(A)) = r(A). \quad (5.1)$$

Чтобы доказать его, предположим, что $\text{sp}(A) \setminus A = \{e_1, \dots, e_k\}$. Из определения $\text{sp}(A)$ имеем $r(A \cup \{e_i\}) = r(A)$, $i = 1, \dots, k$. Предположим, что для некоторого $i < k$ имеет место $r(A \cup \{e_1, \dots, e_i\}) = r(A)$. Тогда из R3 получаем

$$\begin{aligned} r(A \cup \{e_1, \dots, e_i, e_i + 1\}) &= r((A \cup \{e_1, \dots, e_i\}) \cup (A \cup \{e_i + 1\})) \leq \\ &\leq r(A \cup \{e_1, \dots, e_i\}) + r(A \cup \{e_i + 1\}) - r(A) = \\ &= r(A) + r(A) - r(A) = r(A). \end{aligned}$$

Отсюда индукцией по i получаем требуемое равенство (5.1). Доказательство включения $\text{sp}(\text{sp}(A)) \subseteq \text{sp}(A)$ теперь просто. Предполагая $e \in \text{sp}(\text{sp}(A))$, т.е. $r(\text{sp}(A) \cup \{e\}) = r(\text{sp}(A))$, имеем $r(A) \leq r(A \cup \{e\}) \leq r(\text{sp}(A) \cup \{e\}) = r(\text{sp}(A)) = r(A)$, т.е. $e \in \text{sp}(A)$.

Наконец, свойство S4 вытекает из определения зависимости элемента от множества: предположив $f \notin \text{sp}(A)$ и $f \in \text{sp}(A \cup \{e\})$, имеем

$$r(A) + 1 = r(A \cup \{f\}) < r(A \cup \{e, f\}) = r(A \cup \{e\}) < r(A) + 1,$$

а отсюда $r(A \cup \{f\}) = r(A \cup \{e, f\})$, т.е. $e \in \text{sp}(A \cup \{f\})$.

Отметим, что из свойства S3 следует, что $\text{sp}(A)$ является подпространством матроида. Будем называть его *подпространством, натянутым на множество A* .

Последним важным понятием данного раздела является понятие цикла. *Циклом* матроида мы будем называть каждое минимальное зависимое множество, т.е. такое зависимое множество C , что $C \setminus \{e\}$ является независимым для произвольного $e \in C$. Понятие цикла имеет особенно ясную интерпретацию для случая графовых матроидов, о которых речь пойдет в разд. 5.5.

Теорема 5.5. *Для произвольных циклов C, D выполняются условия:*

- C1 Если $C \subseteq D$, то $C = D$,
- C2 Если $C \neq D$ и $e \in C \cap D$, то существует цикл $F \subseteq (C \cup D) \setminus \{e\}$.

Доказательство. Условие C1 выражает просто свойство минимальности, содержащееся в определении цикла. Докажем теперь C2. Множества $C \setminus \{e\}$ и $D \setminus \{e\}$ независимы, а следовательно, $r(C \setminus \{e\}) = |C| - 1$ и $r(D \setminus \{e\}) = |D| - 1$. В силу условия R3

$$r(C \cup D) + r(C \cap D) \leq r(C) + r(D) = |C| + |D| - 2 = |C \cup D| + |C \cap D| - 2. \quad (5.2)$$

Множество $C \cap D$ независимое, так как оно является собственным подмножеством обоих циклов. Отсюда $r(C \cap D) = |C \cap D|$ и неравенство (5.2) можно переписать как

$$r(C \cup D) \leq |C \cup D| - 2. \quad (5.3)$$

Предположим теперь, что не существует цикла F , о котором идет речь в условии C2. Тогда множество $(C \cup D) \setminus \{e\}$ независимое, $r((C \cup D) \setminus \{e\}) = |C \cup D| - 1$ и в результате $r(C \cup D) \geq |C \cup D| - 1$ вопреки неравенству (5.3).

Отметим важное следствие из свойства C2.

Следствие 5.6. Если A — независимое множество, то для произвольного $e \in E$ множество $A \cup \{e\}$ содержит не более одного цикла.

Доказательство. Если бы существовали два различных цикла $C, D \subseteq A \cup \{e\}$, то очевидно, что мы имели бы $e \in C \cap D$, и вследствие свойства C2 существовал бы цикл $F \subseteq (C \cup D) \setminus \{e\} \subseteq A$ вопреки предположению о независимости множества A .

Свойства матроидов, рассмотренные выше, связаны с линейно независимыми множествами векторов линейного пространства. Приведем два других примера матроидов.

Для произвольного конечного множества E пара $\langle E, \wp(E) \rangle$, очевидно, удовлетворяет условиям M1 и M2. Будем называть ее свободным матроидом на множестве E . В таком матроиде каждое множество $A \subseteq E$ независимое, и в результате $r(A) = |A|$.

Другой пример получаем, рассматривая произвольное разбиение $\pi = \{D_1, \dots, D_k\}$ множества E и определяя

$$\mathcal{I} = \{A \subseteq E : |A \cap D_i| \leq 1 \text{ для } i = 1, \dots, k\}.$$

Отметим, что если $A, B \in \mathcal{I}$ и $|B| = |A| + 1$, то должен существовать такой индекс i , что $D_i \cap A = \emptyset$, но $D_i \cap B \neq \emptyset$. Элемент $e \in D_i \cap B$ можно «перенести» в A и получить множество $A \cup \{e\} \in \mathcal{I}$. Это доказывает справедливость условия M2. Матроид $M = \langle E, \mathcal{I} \rangle$, определенный таким образом, будем называть *матроидом разбиений*. Более интересные примеры матроидов будут приведены в разд. 5.4, 5.5 и 5.6.

5.3 Теорема Рамо-Эдмондса

Вернемся к жадным алгоритмам, о которых шла речь в первом разделе. Сформулируем сначала общую схему алгоритмов этого типа. Мы будем рассматривать конечное множество E , функцию $w: E \rightarrow \mathbb{R}^+$ и семейство $\mathcal{I} \subseteq \wp(E)$. Значение $w(e)$ называется весом элемента e . Вес подмножества $A \subseteq E$ определяется следующим образом:

$$w(A) = \sum_{e \in A} w(e)$$

Алгоритм 5.7. (жадный алгоритм.)

1 **begin**

2 упорядочить множество E по убыванию весов так, чтобы
 $E = \{e_1, \dots, e_n\}$, где $w(e_1) \geq w(e_2) \geq \dots \geq w(e_n)$;

3 $S := \emptyset$;

4 **for** $i := 1$ **to** n **do**

5 **if** $S \cup \{e_i\} \in \mathcal{I}$ **then** $S := S \cup \{e_i\}$

6 **end**

Теорема 5.8. (Радо, Эдмондс, см. [15] и [56]). *Если $M = \langle E, \mathcal{I} \rangle$ есть матроид, то множество S , найденное жадным алгоритмом, является независимым множеством с наибольшим весом. Напротив, если $M = \langle E, \mathcal{I} \rangle$ не является матроидом, то существует такая функция $w: E \rightarrow \mathbb{R}^+$, что S не будет независимым множеством с наибольшим весом.*

Доказательство. Предположим, что $M = \langle E, \mathcal{I} \rangle$ является матроидом, и пусть $S = \{s_1, \dots, s_k\}$ — множество, построенное жадным алгоритмом, причем $w(s_1) \geq w(s_2) \geq \dots \geq w(s_k)$. Рассмотрим произвольное независимое множество $T = \{t_1, \dots, t_m\}$, где $w(t_1) \geq w(t_2) \geq \dots \geq w(t_m)$. Отметим сначала, что должно выполняться $m \leq k$, так как выбранное жадным алгоритмом множество S является базой матроида: каждый элемент $e_i \notin S$, «отвергнутый» на некотором шаге алгоритма, является зависимым от множества элементов, выбранных в предыдущий раз, а следовательно, от целого множества S . Докажем теперь, что $w(T) \leq w(S)$, точнее, что для произвольного $i \leq m$ имеем $w(t_i) \leq w(s_i)$. Предположим, что $w(t_i) > w(s_i)$, и рассмотрим независимые множества

$$\begin{aligned} A &= \{s_1, \dots, s_{i-1}\}, \\ B &= \{t_1, \dots, t_{i-1}, t_i\}. \end{aligned}$$

Согласно условию M2 существует элемент t_j , $j \leq i$, такой что множество $\{s_1, \dots, s_{i-1}, t_j\}$ независимое. Имеем $w(t_j) \geq w(t_i) \geq w(s_i)$, следовательно, существует такой индекс $p \leq i$, что $w(s_1) \geq \dots \geq w(s_{p-1}) \geq w(t_j) \geq w(s_p)$

вопреки тому, что s_p — элемент с наибольшим весом, добавление которого к $\{s_1, \dots, s_{p-1}\}$ не нарушает его независимости. Полученное противоречие доказывает, что $w(t_i) \leq w(s_i)$, $1 \leq i \leq m$.

Предположим теперь, что $M = (E, \mathcal{J})$ не является матроидом. Если не выполняется условие M1, т.е. если существуют такие множества $A, B \subseteq E$, что $A \subseteq B \in \mathcal{J}$, $A \notin \mathcal{J}$, то определим

$$w(e) = \begin{cases} 1, & \text{если } e \in A, \\ 0, & \text{если } e \in E \setminus A. \end{cases}$$

Легко увидеть, что множество A не содержится тогда во множестве S , выбранном жадным алгоритмом. В результате $w(S) < w(B) = w(A)$. Если же условие M1 выполняется, но не выполняется условие M2, то существуют независимые множества A, B , такие что $|A| = k$, $|B| = k + 1$, и для каждого $e \in B \setminus A$ множество $A \cup \{e\}$ зависимое. Обозначим $p = |A \cap B|$ (очевидно, что $p < k$ и пусть $0 < \varepsilon < 1/(k - p)$). Определим

$$w(e) = \begin{cases} 1 + \varepsilon, & \text{если } e \in A, \\ 1, & \text{если } e \in B \setminus A, \\ 0, & \text{в остальных случаях.} \end{cases}$$

Заметим, что при весах, определенных таким образом, жадный алгоритм сначала выберет все элементы множества A , а затем отбросит все элементы $e \in B \setminus A$. В результате будет выбрано множество S с весом, меньшим веса множества B :

$$\begin{aligned} w(S) &= w(A) = k(1 + \varepsilon) = (k - p)(1 + \varepsilon) + p(1 + \varepsilon) \leq \\ &\leq (k - p)\frac{k+1-p}{k-p} + p(1 + \varepsilon) = (k + 1 - p) + p(1 + \varepsilon) = w(B). \end{aligned}$$

Следует обратить внимание на один довольно удивительный факт. Поскольку мы определили вес подмножества как сумму весов его элементов, то могло бы казаться, что оптимальное множество S будет существенным образом зависеть от численных значений весов отдельных элементов. Однако же эта зависимость очень слаба: множество S зависит только от упорядочения весов отдельных элементов. В жадном алгоритме после сортировки элементов веса совершенно перестают нас интересовать.

Стоит отметить также, что в доказательстве теоремы 5.8 мы показали, что множество S оптимальное в очень сильном смысле. У него не только максимальна сумма элементов, но также в любом независимом множестве T вес i -го по величине элемента не больше веса i -го по величине элемента в S . Этот факт мы будем называть *оптимальностью по Гейлу* (см. [25]). Таким образом, в матроиде нельзя выбрать независимое множество, состоящее из «меньшего числа больших по величине (но весу) элементов».

Отметим также, что при обращении упорядочения элементов жадный алгоритм выберет множество S , которое не только имеет наименьший вес, но и i -й по величине элемент (считая от наименьшего) будет не больше i -го по величине элемента произвольной базы. Теперь мы можем посмотреть на применение жадного алгоритма к задаче 1 в начале этой главы с общей точки зрения: мы имели там дело с матроидом разбиений, порожденным разбиением позиций матрицы на столбцы.

Далее мы познакомимся с применением жадных алгоритмов к другим, менее банальным матроидам. Покажем также, как в конкретных случаях эффективно проверять условие $S \cup \{e_i\} \in \mathcal{I}$, появляющееся в строке 6 жадного алгоритма. Во всех случаях мы получим оценку сложности алгоритма в виде многочлена относительно размерности задачи, что отнюдь не является типичной ситуацией для оптимизационных задач. В большой степени эффективность жадных алгоритмов вызывается тем, что элемент, один раз включенный в оптимальное решение, остается в нем до конца. Здесь не бывает «проверки всех возможностей», характерной для алгоритмов с возвратом (см. п. 2.8), что обычно приводит к экспоненциальному росту числа шагов при росте размерности задачи.

5.4 Матричные матроиды

Матричные матроиды отличаются от матроидов, определяемых независимыми подмножествами линейного пространства, только видом. Пусть $\{v_1, \dots, v_n\}$ — элементы некоторого линейного пространства размерности m , и пусть $\{b_1, \dots, b_m\}$ — базис этого пространства. Векторы v_1, \dots, v_n имеют однозначно определенное разложение относительно базиса b_1, \dots, b_m .

$$\begin{aligned} v_1 &= a_{11}b_1 + a_{21}b_2 + \dots + a_{m1}b_m \\ v_2 &= a_{12}b_1 + a_{22}b_2 + \dots + a_{m2}b_m \\ &\vdots \\ v_n &= a_{1n}b_1 + a_{2n}b_2 + \dots + a_{mn}b_m \end{aligned}$$

Рассмотрим матрицу

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}. \quad (5.4)$$

Ее столбцы, обозначим их e_1, \dots, e_n , соответствуют векторам v_1, \dots, v_n , причем подмножество векторов линейно независимо тогда и только тогда, когда

соответствующее им множество столбцов линейно независимо. Обратное, столбцы произвольной матрицы A вида (5.4) можно трактовать как векторы некоторого m -мерного линейного пространства. Каждая такая матрица определяет матроид $M(A) = \langle E, \mathcal{I} \rangle$, где E есть множество ее столбцов, а $B \in \mathcal{I}$ тогда и только тогда, когда множество столбцов B линейно независимо. Матроид, определенный таким образом, будем называть *матроидом матрицы A* . Матроид называется *матричным*, если он является (изоморфен с) матроидом некоторой матрицы.

Алгоритм 5.9. (Жадный алгоритм для матричного матроида.)

Данные: Матрица A с m строками и n столбцами, столбцы которой упорядочены по невозрастанию весов (веса неотрицательны).

Результаты: Независимое множество столбцов с наибольшей суммой весов (S содержит номера этих столбцов).

```

1. begin
2.    $S := \emptyset$ ;
3.   for  $j := 1$  to  $n$  do
4.     begin  $i := 0$ ;
5.       while  $(A[i, j] = 0)$  and  $(i < m)$  do  $i := i + 1$ ;
6.       if  $A[i, j] \neq 0$  then (*  $j$ -й столбец ненулевой *)
7.         begin  $S := S \cup \{j\}$ ;
8.           for  $k := j + 1$  to  $n$  do
9.             for  $l := 1$  to  $m$  do
10.               $A[l, k] := A[l, k] - A[l, j] * A[i, k] / A[i, j]$ 
11.            end (*  $A[i, k] = 0$  для  $j + 1 \leq k \leq n^*$  *)
12.          end
13.        end

```

Процесс преобразования матрицы A , реализуемый этим алгоритмом, — это не что иное, как известный из численного анализа метод исключения Гаусса. В каждой итерации цикла 3 алгоритм проверяет (строка 5), состоит ли j -й столбец из одних нулей. Если да ($A[i, j] = 0$ в строке 6), то очевидно, что j -й столбец не принадлежит ни к одному линейно независимому множеству столбцов. Если нет ($A[i, j] \neq 0$ в строке 6), то мы включаем j -й столбец в искомое множество линейно независимых столбцов и вычитаем для всех $k > j$ из k -го столбца j -й столбец, умноженный на $A[i, k] / A[i, j]$. Это не нарушает линейной зависимости столбцов, но приводит к обращению в нуль всех элементов i -й строки справа от $A[i, j]$ (легко увидеть, что последующие итерации цикла 3 не меняют положения дел). По окончании работы алгоритма множество S содержит номера ненулевых столбцов. Эти столбцы линейно независимы, так как после соответствующей перестановки строк они содержат подматрицу размером $|S| \times |S|$ с нулями выше

главной диагонали и ненулевыми элементами на диагонали. Сложность алгоритма можно легко оценить, если заметить, что доминирующей частью ($O(mn)$ шагов) блока 4 является цикл 8. Этот блок выполняется n раз, что в сумме дает $O(n^2m)$ шагов.

Покажем теперь пример применения алгоритма 5.9, связанный с планированием экспериментов. Вообще говоря, будем рассматривать эксперименты, в которых некоторый объект одновременно подвергается действию многих независимых факторов, причем можно количественно измерить суммарное изменение объекта. Нашей задачей будет определение влияния отдельных факторов на изменение объекта. Примем линейную модель, в которой изменение объекта выражается формулой

$$b = c_1x_1 + c_2x_2 + \dots + c_mx_m,$$

где x_i есть интенсивность i -го фактора, а c_1, \dots, c_m — коэффициенты, которые нужно определить. Проще всего было бы совершить m экспериментов, подвергая объект в i -м эксперименте действию только i -го фактора с единичной интенсивностью; тогда $x_i = 1$, $x_k = 0$ для $k \neq i$, что дает возможность непосредственно определить c_i . Однако такое решение часто невозможно по чисто техническим причинам. Предположим, например, что мы наблюдаем влияние содержания различных минералов в почве на рост урожая некоторой культуры, причем мы имеем в своем распоряжении n ($n \geq m$) удобрений, являющихся смесью этих минералов в строго определенных пропорциях. Уравнения, соответствующие возможным экспериментам, можно представить следующим образом:

$$\begin{array}{ccccccc} c_1a_{11} + & c_2a_{21} + & \dots + & c_ma_{m1} = & b_1 \\ c_1a_{12} + & c_2a_{22} + & \dots + & c_ma_{m2} = & b_2 \\ \vdots & \vdots & & \vdots & \\ c_1a_{1n} + & c_2a_{2n} + & \dots + & c_ma_{mn} = & b_n \end{array} \quad (5.5)$$

где a_{ij} обозначает количество i -го минерала в j -м эксперименте (мы предполагаем использование стандартных количеств удобрений и площадей культур). Рассмотрим матрицу $A = [a_{ij}]$. Ее столбцы соответствуют экспериментам (удобрениям), а строки — минералам. Каждое линейно независимое множество из m столбцов определяет множество экспериментов, которые нужно провести, чтобы вычислить коэффициенты c_1, \dots, c_m из системы уравнений (5.5). Если упорядочить столбцы по убыванию стоимости соответствующих им экспериментов, то жадный алгоритм определит самую дешевую систему экспериментов.

5.5 Графовые матроиды

Пусть $G = \langle V, E \rangle$ — неориентированный граф. Определим $M(G) = \langle E, \mathcal{J} \rangle$, где $\mathcal{J} = \{A \subseteq E : \text{граф } \langle V, A \rangle \text{ не содержит циклов}\}$.

Теорема 5.10. *$M(G)$ является матроидом для произвольного графа G .*

Доказательство. Очевидно, что условие М1 выполняется. В силу теоремы 5.1 достаточно показать, что выполняется также условие М3. И это действительно так, поскольку каждое максимальное подмножество $A \in \mathcal{J}$, содержащееся в множестве $B \subseteq E$, является не чем иным, как стягивающим лесом графа $\langle V, B \rangle$ (см. разд. 2.4). Количество элементов каждого такого множества A составляет, таким образом,

$$|V| - (\text{число компонент связности графа } \langle V, B \rangle).$$

Матроид $M(G)$ будем называть *матроидом графа G* . Произвольный матроид называется *графовым*, если он является (изоморфен с) матроидом некоторого графа. Отметим, что циклы матроида $M(G)$ — это не что иное, как множества ребер элементарных циклов; это объясняет происхождение названия «цикл» в случае произвольных матроидов. Оказывается, что каждый графовый матроид $M(G)$ можно трактовать как матричный матроид, соответствующий матрице инцидентности графа G , рассматриваемой как матрица с элементами из двухэлементного поля $Z_2 = \{0, 1\}$. Сложение в таком поле выполняется по модулю 2, а умножение — как обычное умножение целых чисел:

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 1 \end{array} \quad \begin{array}{c|cc} \cdot & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

Вспомним, что столбцы матрицы инцидентности соответствуют ребрам, а строки — вершинам графа, причем столбец, соответствующий ребру $\{u, v\}$, содержит единицы в строках, соответствующих вершинам u и v , и нули в остальных позициях.

Теорема 5.11. *Пусть $G = \langle V, E \rangle$ — произвольный граф и A — его матрица инцидентности. Подмножество столбцов матрицы A является линейно зависимым над полем Z_2 тогда и только тогда, когда соответствующее ему подмножество ребер содержит цикл.*

Доказательство. Предположим, что множество $C \in E$ является циклом, и рассмотрим соответствующее ему множество столбцов матрицы A . Это множество содержит в каждой ненулевой строке две единицы, следовательно, является линейно зависимым над полем Z_2 ; его сумма, вычисленная по модулю 2, дает нулевой столбец. Напротив, предположим, что некоторое непустое множество столбцов матрицы A является линейно зависимым над Z_2 . Это означает,

что оно содержит непустое множество столбцов с суммой, являющейся нулевым столбцом (отметим, что в случае поля Z_2 линейная комбинация с ненулевыми коэффициентами — это просто сумма). Пусть $B \subseteq E$ — соответствующее ему множество ребер. $\langle V, B \rangle$ является тогда графом с четной степенью каждой вершины. Этот граф содержит цикл, поскольку каждый непустой граф без циклов содержит по крайней мере одну вершину степени 1.

В силу этой теоремы мы можем использовать алгоритм 5.9 для отыскания базы матроида $M(G)$ с наименьшим весом — другими словами, минимального стягивающего дерева (в общем случае леса, т.е. графа, все компоненты связности которого являются деревьями). Сложность такого решения составляет $O(m^2n)$, где $n = |V|$ и $m = |E|$.

Существуют однако значительно более эффективные алгоритмы. Опишем один из них — алгоритм Краскала [40]. Следует отметить, что этот алгоритм сыграл большую роль в развитии теории, описанной в данной главе; алгоритм 5.7 был сформулирован как обобщение алгоритма Краскала с графового матроида на произвольные матроиды.

Алгоритм 5.12. (Краскал, см. [40]).

Данные: Граф $G = \langle V, E \rangle$ в виде списка ребер e_1, \dots, e_m , упорядоченного по неубыванию весов.

Результаты: Минимальный стягивающий лес (S содержит множество его ребер).

```

1. begin
2.    $S := \emptyset$ ;
3.   РАЗБ :=  $\{\{v\} : v \in V\}$ ;
4.   for  $i := 1$  to  $m$  do
5.     if  $e_i = \{u, v\}$ , где  $u \in A, v \in B, A, B \in \text{РАЗБ}, A \neq B$  then
6.       begin  $S := S \cup \{e_i\}$ ;
7.         РАЗБ :=  $(\text{РАЗБ} \setminus \{A, B\}) \cup \{A \cup B\}$ ;
8.       end
9. end

```

В этом алгоритме, являющемся вариантом жадного алгоритма для матроида $M(G)$, мы строим стягивающий лес графа G , шаг за шагом добавляя последовательно к S ребра, не вызывающие замыкания цикла. РАЗБ содержит разбиение множества V на множества вершин компонент связности графа, определяемого текущим содержанием множества S . Ребро e_i , анализируемое в строке 5, добавляем к S тогда и только тогда, когда оно имеет оба конца в различных блоках разбиения РАЗБ, в противном случае его добавление вызвало бы образование цикла. После добавления e_i к S следует заменить блоки $A, B \in \text{РАЗБ}$, содержащие концы ребер e_i , их суммой $A \cup B$. Сложность алгоритма Краскала

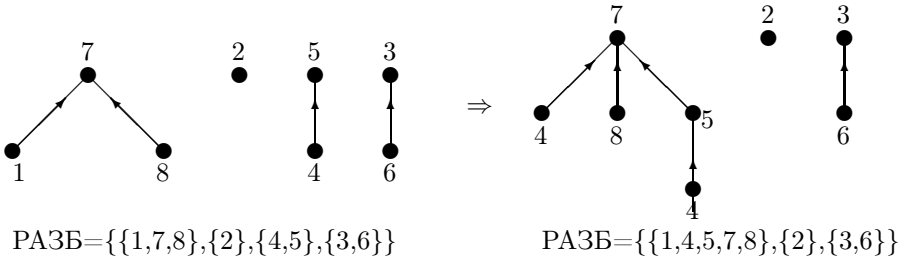


Рис. 5.1: Слияние блоков $\{1, 7, 8\}$ и $\{4, 5\}$ разбиения РАЗБ. Пример минимального стягивающего дерева, построенного алгоритмом Краскала, представлен на рис. 5.2 (в скобках даны веса ребер).

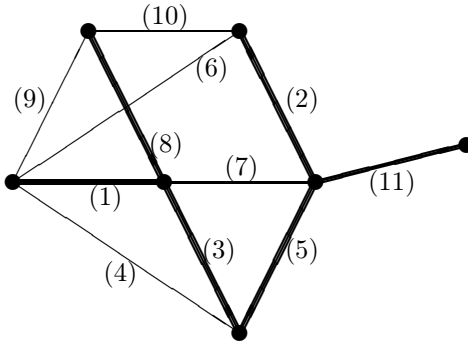


Рис. 5.2: Минимальное стягивающее дерево, построенное алгоритмом Краскала.

существенным образом зависит от способа представления разбиения РАЗБ и способа, каким выполняется слияние его блоков. Существуют очень эффективные методы решения последней задачи (см. [67]). Для наших целей достаточно следующего метода: каждый блок разбиения представляется ориентированным деревом, в котором из каждой вершины существует путь до корня. Корень идентифицирует данный блок. Такое дерево содержит также информацию о своей высоте. Идентификация блока, содержащего данную вершину, происходит прохождением пути из этой вершины до корня. Слияние блоков A, B происходит путем добавления дуг от корня дерева, представляющего один блок, к корню дерева, представляющего другой блок (рис. 5.1). Если при слиянии добавленная дуга всегда идет от корня дерева с меньшей (или равной) высотой, то в ходе алгоритма высота каждого дерева не больше логарифма его вершин, а тем самым не больше $\log n$ (см. задачу 5.13).

5.6 Матроиды трансверсалей

Пусть $\mathcal{A} = (A_1, \dots, A_n)$ — семейство подмножеств (не обязательно различных) некоторого конечного множества E . Будем говорить, что множество $S \subseteq E$ является *частичной трансверсалью* семейства \mathcal{A} , если существует такое инъективное отображение $\varphi : S \rightarrow \{1, \dots, n\}$, что $e \in A_{\varphi(e)}$ для каждого $e \in S$. Это эквивалентно следующему утверждению: множество $S = \{e_1, \dots, e_k\}$ является частичной трансверсалью семейства \mathcal{A} , если для некоторой перестановки f множества $\{1, \dots, k\}$ и для некоторой последовательности индексов $1 \leq i_1 \leq \dots \leq i_k \leq n$ последовательность $\langle e_{f(1)}, \dots, e_{f(k)} \rangle$ является системой различных представителей для последовательности $\langle A_{i_1}, \dots, A_{i_k} \rangle$ (см. п. 4.4). Отметим, что для частичной трансверсали S семейства (A_1, \dots, A_n) существует, вообще говоря, много различных взаимно однозначных функций $\varphi : S \rightarrow \{1, \dots, n\}$, удовлетворяющих условию $e \in A_{\varphi(e)}$, $e \in S$.

Основным результатом данного раздела является следующая теорема Эдмондса и Фалкерсона (см. [16]).

Теорема 5.13. Пусть $\mathcal{A} = (A_1, \dots, A_n)$ — семейство подмножеств конечного множества E , и пусть \mathfrak{I} — семейство частичных трансверсалей семейства \mathcal{A} . Тогда $M(\mathcal{A}) = \langle E, \mathfrak{I} \rangle$ является матроидом.

Доказательство. Очевидно, что условие М1 выполняется. Чтобы доказать выполнение условия М2, предположим, что $A, B \in \mathfrak{I}$, $|A| = k$, $|B| = k + 1$. Мы можем предполагать, что $A = \{a_1, \dots, a_k\}$ и $B = \{b_1, \dots, b_{k+1}\}$, где $a_i = b_i$ для $1 \leq i \leq r = |A \cap B|$. Если $r = k$, т.е. $A \subseteq B$, то $A \cup \{b_{k+1}\} \in \mathfrak{I}$ и условие М2 выполняется. Пусть $r < k$, и предположим, что $\langle a_1, \dots, a_k \rangle$ является системой различных представителей для A_{i_1}, \dots, A_{i_k} , а $\langle b_1, \dots, b_{k+1} \rangle$ является системой различных представителей для $B_{j_1}, \dots, B_{j_{k+1}}$, (индексы i_1, \dots, i_k попарно различны, как и j_1, \dots, j_{k+1}). Рассмотрим следующее построение, которое либо находит такой элемент $b \in B \setminus A$, что $A \cup \{b\} \in \mathfrak{I}$, либо заменяет последовательность A_{i_1}, \dots, A_{i_k} некоторой новой последовательностью, для которой $\langle a_1, \dots, a_k \rangle$ также является системой различных представителей.

Построение

Выберем такой индекс $p \leq k+1$, что $j_p \notin \{i_1, \dots, i_k\}$. Если можно так выбрать этот индекс, чтобы $r + 1 \leq p \leq k + 1$, то последовательность $\langle a_1, \dots, a_k, b_p \rangle$ является системой различных представителей для $A_{i_1}, \dots, A_{i_k}, A_{j_p}$, а из этого следует, что $A \cup \{b_p\} \in \mathfrak{I}$. В противном случае, т.е. когда $1 \leq p \leq r$, имеем $a_p = b_p$ и последовательность $\langle a_1, \dots, a_k \rangle$ является системой различных представителей для $A_{i_1}, \dots, A_{i_{p-1}}, A_{i_{p+1}}, \dots, A_{i_k}$.

Рассмотрим подробнее эту последовательность. Если последовательности $\langle i_1, \dots, i_r \rangle$ и $\langle j_1, \dots, j_r \rangle$ совпадают на $q < r$ позициях, то последовательности $\langle i_1, \dots, i_{p-1}, j_p, i_{p+1}, \dots, i_r \rangle$ и $\langle j_1, \dots, j_r \rangle$ совпадают на $p + 1$ позициях, так

как $j_p \neq i_p$. Повторяя наше построение, мы либо находим требуемую частичную трансверсаль $A \cup \{b\}$, $b \in B \setminus A$ либо приходим к ситуации, когда $\langle i_1, \dots, i_r \rangle = \langle j_1, \dots, j_r \rangle$. Однако в последнем случае $j_p \notin \{i_1, \dots, i_k\}$ для некоторого p , $r+1 \leq p \leq k+1$ (ведь не может быть $\{j_{r+1}, \dots, j_{k+1}\} \subseteq \{i_{r+1}, \dots, i_k\}$). Выполнение нашего построения еще раз приводит к нахождению такого элемента $b_p \in B \setminus A$, что $A \cup \{b_p\} \in \mathcal{I}$.

Матроид $M(\mathcal{A})$ называется *матроидом трансверсалей* семейства \mathcal{A} .

Опишем теперь жадный алгоритм для матроидов трансверсалей. Этот алгоритм является модификацией метода нахождения систем различных представителей (а точнее, наибольшего паросочетания в двудольном графе), основанного на методике чередующихся цепей, описанной в гл. 4. Вспомним, что семейство (A_1, \dots, A_n) подмножеств множества $E = \{e_1, \dots, e_m\}$ мы изображаем двудольным графом H с вершинами $u_1, \dots, u_m, v_1, \dots, v_n$ и ребрами вида $\{u_i, v_j\}$, где $e_i \in A_j$. Каждой системе различных представителей $\langle e_{i_1}, \dots, e_{i_k} \rangle$ последовательности $\langle A_{j_1}, \dots, A_{j_k} \rangle$ (индексы j_1, \dots, j_k попарно различны) соответствует независимое множество ребер $M = \{\{u_{i_1}, v_{j_1}\}, \dots, \{u_{i_k}, v_{j_k}\}\}$ в графе H .

Алгоритм 5.14. (Жадный алгоритм для матроида трансверсалей.)

Данные: Семейство $\mathcal{A} = (A_1, \dots, A_n)$ подмножеств множества $E = \{e_1, \dots, e_m\}$, представленное с помощью двудольного графа G . Элементы e_1, \dots, e_m заиндексированы в порядке невозрастания весов (веса неотрицательные).

Результаты: частичная трансверсаль S с наибольшим весом для семейства \mathcal{A} (и паросочетание M графа H , определяющее функцию $\varphi : S \rightarrow \{1, \dots, n\}$, о которой идет речь в определении частичной трансверсали).

1. **begin**

2. $S := \emptyset; M := \emptyset;$

3. **for** $i := 1$ **to** m **do**

4. **if** существует в G чередующаяся цепь P относительно M с началом в e_i **then**

5. **begin** $S := S \cup \{e_i\}; M := P \otimes M$

6. **end**

7. **end**

Очевидно, что сложность этого алгоритма зависит от метода поиска чередующейся цепи P (строки 4 и 5). Если мы используем метод поиска в глубину так, как это сделано в разд. 2.2, то число шагов, необходимое для отыскания цепи, будет порядка числа ребер в графе G , т.е. $O\left(\sum_{i=1}^n |A_i|\right)$. Тогда общая сложность алгоритма 5.14 будет $O\left(m \sum_{i=1}^n |A_i|\right)$.

Приведем теперь две задачи из практики, которые можно решать с помощью этого алгоритма.

Предположим, что некий предприниматель принял на работу n человек, причем i -й сотрудник имеет квалификацию, позволяющую выполнять любую работу из некоторого множества A_i .

Обозначим $\bigcup_{i=1}^n A_i = E = \{e_1, \dots, e_m\}$ и предположим, что прибыль предпринимателя от выполнения работы e_j составляет $w(e_j)$ и что каждый из принятых на работу может выполнять не более одной работы. Нужно найти такое соответствие работ и людей, могущих их выполнять, чтобы прибыль была наибольшей. На языке матроидов задача заключается в нахождении базиса с наибольшим весом матроида трансверсалей $M(A)$, где $A = (A_1, \dots, A_n)$. Точнее, нужно не только найти частичную трансверсаль S с наибольшим весом, но также и взаимно однозначную функцию $\varphi : S \rightarrow \{1, \dots, n\}$, для которой $e \in A_{\varphi(e)}$, $e \in S$. Именно это и делает алгоритм 5.14.

Стоит отметить здесь, что, так же как и во всех жадных алгоритмах, представленных в данной главе, база с наименьшим (наибольшим) весом зависит только от упорядочения элементов e_1, \dots, e_m по неубыванию весов. Обозначим через $e_i \leq e_j$ тот факт, что $w(e_i) \leq w(e_j)$ («работа e_j оплачивается не меньше, чем работа e_i »). Мы знаем, что множество работ $S = \{a_1, \dots, a_k\}$ (где $a_1 \geq \dots \geq a_k$) является оптимальным по Гейлу, а следовательно, для каждого другого множества работ $T = \{b_1, \dots, b_l\}$ (где $b_1 \geq \dots \geq b_l$), которое может быть выполнено нанятыми на работу, имеем $l \leq k$ и $b_i \leq a_i$ для $1 \leq i \leq l$.

Второй пример касается нахождения оптимальной очередности выполнения работ. Предположим, что дано n работ e_1, \dots, e_m , причем выполнение каждой из них требует одного и того же количества времени, скажем одного часа. Работы мы выполняем по очереди, не прерывая ни одной начатой работы, причем в каждый момент времени может выполняться не более одной работы. Для каждого i , $1 \leq i \leq m$, дан также срок d_i выполнения работы e_i (отсчитываемый от нулевого момента) и штраф $w(e_i)$, который выплачивается за невыполнение задания в срок (предполагаем, что штраф не зависит от того, на сколько часов был превышен срок). Нужно найти очередность выполнения работ, минимизирующую сумму штрафов. Эту же задачу мы можем сформулировать как нахождение выполнимого в срок множества работ с наибольшей суммой штрафов (тут мы максимизируем сумму штрафов, «которых избежали»). При этом нужно найти также очередность выполнения работ из этого оптимального множества. Связь этой задачи с матроидами трансверсалей следующая. Определим $E = \{e_1, \dots, e_m\}$, $n = \max_{1 \leq i \leq m} d_i$ и $A_i = \{e_j \in E : d_j \geq i\}$ для $1 \leq i \leq n$. При таких обозначениях произвольная частичная трансверсаль S семейства $A = (A_1, \dots, A_n)$ определяет выполнимое в срок множество работ, причем взаимно однозначная функция $\varphi : S \rightarrow \{1, \dots, n\}$, такая что $e \in A_{\varphi(e)}$, $e \in S$, определяет для каждой работы $e \in S$ время $\varphi(e)$, в которое эта работа

должна быть выполнена. Следовательно, оптимальное решение S является базисом с наибольшим весом матроида трансверселей $M(\mathcal{A})$ и его можно найти, используя алгоритм 5.14.

5.7 Задачи

5.1. Доказать, что семейство $\mathcal{B} \subseteq \wp(E)$ является базой некоторого (однозначно определенного семейством \mathcal{B}) матроида тогда и только тогда, когда выполняется условие

- В1 Для произвольных $B_1, B_2 \in \mathcal{B}$ и $e \in B_1 \setminus B_2$ существует $f \in B_2 \setminus B_1$, такое что $(B_1 \setminus \{e\}) \cup \{f\} \in \mathcal{B}$.

5.2. Для следующих понятий, связанных с матроидами:

- (а) семейство \mathcal{I} независимых множеств матроида,
- (б) семейство \mathcal{B} баз матроида,
- (в) семейство \mathcal{C} циклов матроида,
- (г) семейство \mathcal{F} подпространств матроида,
- (д) функция ранга r ,
- (е) операция порождения подпространства sp

выразить каждое из них в терминах каждого из оставшихся.

5.3. Доказать, что семейство $\mathcal{C} \subseteq \wp(E)$ является семейством циклов некоторого матроида тогда и только тогда, когда удовлетворяются условия С1 и С2.

5.4. Доказать, что операция $\text{sp} : \wp(E) \rightarrow \wp(E)$ является операцией порождения подпространства в некотором матроиде тогда и только тогда, когда выполняются условия S1, S2, S3, S4.

5.5. Доказать, что функция r является функцией ранга в некотором матроиде тогда и только тогда, когда выполняются условия R1, R2, R3 (или R1, R4, R5).

5.6. Доказать, что если элемент принадлежит каждой базе матроида, то он не принадлежит ни одному циклу этого матроида.

5.7. Проверить, что для произвольного конечного множества E и произвольного $k \leq |E|$ пара $M = \langle E, \mathcal{I} \rangle$, где $\mathcal{I} = \{A \subseteq E : |A|_k\}$ является матроидом. Определить для этого матроида понятия (а)–(е) из задачи 5.2.

5.8. Доказать, что в произвольном матроиде $e \in \text{sp}(A)$ тогда и только тогда, когда существует цикл C , такой что $C \setminus A = \{e\}$.

5.9. Доказать, что если C_1 и C_2 — два произвольных различных цикла матроида и $f \in C_1 \setminus C_2$, то для каждого $e \in C_1 \cap C_2$ найдется такой цикл C_3 , что $f \in C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$.

5.10. Доказать, что если \mathcal{B} — семейство баз матроида $M = \langle E, \mathcal{I} \rangle$, то $\mathcal{B}^* = \{E \setminus B : B \in \mathcal{B}\}$ — семейство баз некоторого матроида M^* (M^* называется матроидом, двойственным M). Доказать, что функция ранга двойственного матроида выражается формулой $r^*(A) = |A| + r(E \setminus A) - r(E)$.

5.11. Доказать, что для произвольного матроида $M = \langle E, \mathcal{I} \rangle$ и произвольного множества $S \subseteq E$ пара $\langle S, \mathcal{I}' \rangle$, где $\mathcal{I}' = \{A \in \mathcal{I} : A \subseteq S\}$, является матроидом. Всегда ли будет матроидом $\langle S, \mathcal{I}'' \rangle$, где $\mathcal{I}'' = \{S \cap A : A \in \mathcal{I}\}$?

5.12. Применить алгоритм 5.11 к матрице

$$A = \begin{bmatrix} 1 & 4 & 2 & 1 & 3 & 6 \\ 2 & 8 & 4 & 1 & 7 & 2 \\ -7 & 13 & -14 & 1 & 1 & 6 \\ 0 & 6 & 0 & 1 & 12 & 2 \end{bmatrix}.$$

(столбцы упорядочены по неубыванию весов).

5.13. Доказать, что при описанной в разд. 5.5 реализации алгоритма Краскала высота дерева, представляющего произвольный блок B разбиения РАЗБ, не превышает $\log |B|$.

5.14. Доказать, что каждый матроид разбиения является частным случаем матроида трансверсалей.

5.15. Доказать, что если существует система различных представителей для $\langle A_1, \dots, A_n \rangle$ и $\langle x_1, \dots, x_k \rangle$ является системой различных представителей для $\langle A_1, \dots, A_k \rangle$ ($k < n$), то существует элемент x_{k+1} и перестановка f множества $\{1, \dots, k+1\}$, такая что $\langle x_{f(1)}, \dots, x_{f(k+1)} \rangle$ является системой различных представителей для $\langle A_1, \dots, A_{k+1} \rangle$.

Предметный указатель

- алгоритм , 9
 - генерирования
 - всех перестановок, 23, 27
 - всех подмножеств k -элементного множества, 30
 - всех последовательностей, 21
 - Дейкстры, 117
 - Диница, 137, 138
 - жадный , 174, 181
 - для матричного матроида, 184
 - для матроида трансверсалей, 190
 - Краскала, 187
 - нахождения
 - всех гамильтоновых циклов, 106
 - всех разбиений числа, 57
 - кратчайшего пути, 113
 - множества элементарных циклов графа, 95
 - разбиения семейства всех подмножеств множества на симметричные цепи, 165
 - расстояний от источника до всех вершин в бесконтурном графе, 122
 - эйлерового цикла, 100
 - нумерации вершин бесконтурного графа, 120
 - определения знака перестановки, 18
 - построения максимального потока в сети, 145
 - с возвратом, 104, 105
 - Уоршалла, 126
 - Флойда, 124
 - Хопкрофта–Карпа, 151
- антицепь, 164
- Беллман Р.Е., 115
- бинарный код Грэя порядка n , 30
- блок
 - графа, 96
 - разбиения, 43
- вектор инверсивный, 70
- величина потока, 130
- вершина графа , 7
 - изолированная, 7
 - отец, 90
 - потомок, 90
 - свободная, 147
 - сын, 90
- вес
 - дуги, 112
 - элемента матроида, 181
- ветвь, 89
- Гамильтон Р.В., 28, 33, 41, 102
- Гейл Д., 182
- генерирование

- множества перестановок, 20
- подмножеств
 - k -элементных, 40
- разбиений числа, 56
- граница
 - верхняя, 44
 - нижняя, 44
- граф
 - бесконтурный, 120
 - выпуклый, 171
 - двудольный, 110, 147
 - двусвязный, 96
 - неориентированный, 7
 - ориентированный, 7
 - связный, 8, 89
 - сильно связный, 110
- графы изоморфные, 8
- группа перестановок степени n , 15
- группа симметрическая степени n , 15

- Дейкстра Э.В., 117
- Джонсон С.М., 25
- Дилворт Р.П., 169
- Диниц Е.А., 135, 137

- двоичный n -мерный куб, 33
- де Брёйн Н.Д., 111
- дерево , 89
 - бинарное, 64
 - стягивающее графа, 89
- деревья изоморфные, 64
- диаграмма Феррерса, 55
- длина
 - пути, 8, 112
 - сети, 135
 - увеличивающей цепи, 132
- дуга, 7

- задача NP-полная, 103
- закон Кирхгофа, 94

- замыкание отношения транзитивное, 125
- знак перестановки, 16

- измельчение разбиения, 43
- инверсия, 69
- инволюция, 70
- источник, 129

- Карп Р.М., 149
- Кениг Д., 159
- Клос С., 161
- Краскал Д.В., 187

- класс эквивалентности, 43
- клика, 111
- комбинация без повторов, 35
- коммутатор размерности $N \times N$, 161
- компонента
 - двусвязности, 96
 - связности, 8
 - сильной связности, 110
- конец пути, 8
- контур, 8
- корень дерева , 64
 - стягивающего, 90
- коэффициент биномиальный, 35
- кратность элемента, 33

- лес стягивающий, 92
- линия матрицы, 159

- матрица
 - бистохастическая, 172
 - инциденций, 79
 - перестановочная, 172
 - смежности, 80
 - соседства ребер, 108
- матроид , 176
 - база, 177
 - графа, 186

- графовый, 186
- двойственный, 193
- матричный, 184
- ранг, 177
- свободный, 180
- трансверсалией, 190
- матроиды изоморфные, 176
- метод
 - Лума, 167
 - Флойда, 124
- метод увеличивающих цепей, 131
- метод Форда–Беллмана, 115
- множество
 - матроида
 - зависимое, 176
 - независимое, 176
 - с повторениями, 33
 - частично упорядоченное, 6
- мост графа, 110
- мощность множества, 5
- начало пути, 8
- область
 - левая, 6
 - правая, 6
- оптимальность по Гейлу, 182
- отец вершины, 90
- отношение , 6
 - антисимметричное, 6
 - бинарное , 6, 125
 - транзитивное замыкание, 125
 - симметричное, 6
 - транзитивное, 6
 - эквивалентности, 6
- очередь, 9
- палиндром, 70
- пара упорядоченная, 5
- паросочетание, 147
- паскаль (язык), 9
- перестановка , 12
 - нечетная, 16
 - обратная, 14
 - тождественная, 14
 - четная, 16
- перманент матрицы, 172
- петля, 80
- подграф, 8
- подмножество , 5, 29
 - k -элементное, 35
 - собственное, 5
- подпространство матроида , 178
 - натянутое на множество, 179
- поиск
 - в глубину, 84
 - в ширину, 86
- покрытие вершинное, 157
- порядок
 - антилексикографический, 20
 - лексикографический, 20
 - частичный, 6
- потенциал вершины сети, 140
- поток, 129
- поток
 - максимальный, 130
 - псевдомаксимальный, 137
- потомок вершины, 90
- принцип включения–исключения, 66
- произведение
 - декартово, 5
 - Коши рядов, 59
- пропускная способность
 - вершины, 170
 - дуги, 129
 - разреза, 131
- псевдоцикл, 93
- путь
 - в графе, 7
 - гамильтонов, 33, 34, 41, 102
 - длина, 8

- конец, 8
 - критический, 122
 - начало, 8
 - эйлеров, 99
- Радо, 181
- разбиение
- множества, 43
 - числа , 55
 - самосопряженное, 76
 - сопряженное, 55
 - разложение на циклы, 15
 - размещение объектов по ящикам упорядоченное, 12
 - разность симметрическая, 92
 - разрез сети , 130
 - минимальный, 131
 - ранг матроида, 177
 - расстояние, 112
 - ребро, 7
 - решетка, 43
 - ряд
 - Маклорена, 59
 - формальный, 58
- секция коммутатора, 161
- сеть, 129
- сеть
- CPM, 122
 - PERT, 122
 - перестраиваемая размерности $N \times N$, 161
 - трехсекционная Клоса, 161
- система различных представителей, 156
- система различных представителей
- общая, 160
- сложение рядов, 58
- сложность
- временная, 10
 - вычислительная, 9
 - по памяти, 10
- список инцидентности, 82
- стек, 9
- степень вершины, 7
- сток, 129
- суперпозиция перестановок, 14
- сын вершины, 90
- Троттер Х.Ф., 25
- теорема
- венгерская, 159
 - Дилворта, 169
 - Радо-Эдмондса, 181
 - Холла, 156
- тождество Коши, 36
- точка сочленения, 95
- транзитивное замыкание отношения, 125
- трансверсаль частичная, 189
- транспозиция, 16
- треугольник Паскаля, 36, 45
- Уоршалл С., 124, 126
- увеличивающая цепь , 131
- длина, 132
- Феррерс Н.М., 55
- Флойд Р.В., 124
- Форд Л.Р., 115
- фундаментальное множество циклов, 92
- функция , 6
- взаимно однозначная, 7
 - на, 7
 - производящая , 58
 - экспоненциальная, 77
 - ядро, 47

Холл Ф., 156
Хопкрофт Дж., 149

хорда, 89

цепь, 164

цепь

 симметричная, 164

 чередующаяся, 149

цикл, 8, 92

 гамильтонов, 102

 де Брёйна порядка n , 111

 матроида, 179

 разложения перестановки, 15

 эйлеров, 99

 элементарный, 8

число

 Белла, 47

 Стирлинга второго рода, 45, 49

 Фибоначчи, 63

шаг алгоритма, 10

Эгервари Э., 159

Эдмондс Дж., 135, 181

экспоненциальная производящая функция, 77

элемент зависимый от множества, 178

ядро функции, 47

Оглавление

0.1	Предисловие редактора перевода	2
0.2	От автора	3
1	Введение в комбинаторику	5
1.1	Основные понятия	5
1.2	Функции и размещения	10
1.3	Перестановки: разложение на циклы, знак перестановки	13
1.4	Генерирование перестановок	19
1.5	Подмножества множества, множества с повторениями,	29
1.6	k -элементные подмножества, биномиальные коэффициенты	34
1.7	Генерирование k -элементных подмножеств	39
1.8	Разбиения множества	43
1.9	Числа Стирлинга второго и первого рода	45
1.10	Генерирование разбиений множества	50
1.11	Разбиения чисел	54
1.12	Производящие функции	58
1.13	Принцип включения и исключения	66
1.14	Задачи	70
2	Алгоритмы на графах	79
2.1	Машинное представление графов	79
2.2	Поиск в глубину в графе	83
2.3	Поиск в ширину в графе	86
2.4	Стягивающие деревья (каркасы)	89
2.5	Отыскание фундаментального множества циклов в графе	92
2.6	Нахождение компонент двусвязности	95
2.7	Эйлеровы пути	99
2.8	Алгоритмы с возвратом (back-tracking)	102
2.9	Задачи	108

3	Нахождение кратчайших путей в графе	112
3.1	Начальные понятия	112
3.2	Кратчайшие пути от фиксированной вершины	114
3.3	Случай неотрицательных весов — алгоритм Дейкстры	116
3.4	Пути в бесконтурном орграфе	120
3.5	Кратчайшие пути между всеми парами вершин, транз. замыкание	123
3.6	Задачи	127
4	Потоки в сетях и родственные задачи	129
4.1	Максимальный поток в сети	129
4.2	Алгоритм построения максимального потока	134
4.3	Наибольшие паросочетания в двудольных графах	147
4.4	Системы различных представителей	156
4.5	Разложение на цепи	164
4.6	Задачи	170
5	Матроиды	174
5.1	Жадный алгоритм решения оптимизационных задач	174
5.2	Матроиды и их основные свойства	176
5.3	Теорема Рамо-Эдмондса	181
5.4	Матричные матроиды	183
5.5	Графовые матроиды	186
5.6	Матроиды трансверсалей	189
5.7	Задачи	192