М. Н. Кирсанов

ГРАФЫ В MAPLE

Задачи, алгоритмы, программы

Пособие по дискретной математике для студентов университетов

МОСКВА
ФИЗМАТЛИТ
2007

УДК 519.17+681.3.06
ББК 22.213
К 435

К и р с а н о в М. Н. Графы в Maple. Задачи, алгоритмы, программы. — М.: Издательство ФИЗМАТЛИТ, 2007. — 168 с. — ISBN 5-7046-1168-0.

К 435

Изложены решения задач теории графов. Даны описания основных алгоритмов на графах и тексты более 30 программ. Приведены алгоритмы теории искусственного интеллекта (муравьиный алгоритм и метод отжига) для решения задачи коммивояжера. Предметно-именной указатель на 500 терминов и имен может служить справочником по теории графов и командам Maple.

Книга предназначена как для очного, так и для дистанционного обучения.

Для студентов и преподавателей университетов и технических вузов.

УДК 531.3
ББК 22.213

ISBN 5-7046-1168-0

СОДЕРЖАНИЕ

Предисловие

Теория графов — один из разделов современной математики, имеющий большое прикладное значение. Проблемы оптимизации тепловых, газовых и электрических сетей, вопросы совершенствования алгоритмов и создание новых химических соединений связаны с фундаментальными свойствами таких абстрактных математических объектов, как графы. Долгое время задачи теории графов решались вручную, с появлением компьютеров появилась возможность написания специальных программ на алгоритмических языках. Позднее появились пакеты аналитических вычислений Mathematica, MATLAB [9], Mathcad [27] и Maple [8], позволяющие выполнять аналитические символьные преобразования. Для решения задач, объектами которых являются графы, эти пакеты просто незаменимы. В системе Maple есть еще и специализированная библиотека `networks`, составленная из операторов для работы с графами. Таких операторов немного — всего 66, и число это в Maple не меняется от версии к версии [1]. Появилась проблема — описать пакет `networks` и дать примеры решения задач с его помощью. Решению этой проблемы и посвящена данная книга.

В первой части книги даны сведения из теории и решения некоторых основных задач теории графов. Во второй части содержатся готовые программы, разработанные автором как с привлечением операторов и команд пакета `networks`, так и решенные независимым образом на языке Maple. В связи с упоминанием языка представления программ следует уделить немного внимания библиографическому обзору. Дело в том, что во многих книгах и учебниках по дискретной математике даны тексты программ. Однако появилась тенденция печатать их на некотором мертвом условном языке, похожем на Pascal. К сожалению, очень часто такие программы плохо читаются, так как некоторые «команды» понятны только для посвященных (обычно это автор книги). Приятным исключением являются книга Зубова В.С., Шевченко И.В. [11] и отчасти книга Иванова Б.Н. [13]. Книг с программами по дискретной математике на языке Maple нет, и, вероятно, это издание является первым.

[1] На 2006 год последняя версия — Maple 10. В версии Maple 11 добавился пакет по теории графов `GraphTheory`, содержащий 112 команд и операторов.

В книге описаны почти все команды и процедуры пакета теории графов `networks`. Исключение составляют только вероятностные задачи на графах. Кроме того, добавлены некоторые новые собственные процедуры. В частности, введена процедура рисования орграфа. При выборе стиля программирования автор добивался ясности изложения алгоритма, иногда даже в ущерб краткости и скорости работы программы. Совершенствование программ предлагается читателям, тем более, что данная книга — учебник, а учиться лучше всего, программируя самому и убеждаясь, что «моя программа лучше!». Желаем читателям успехов в этом!

Автор благодарит студентов МЭИ(ТУ) Александрова В.А., Ульянова Р.В. и Сутяпова А.В. за программы 15, 16, 19, 20, 28 и 32 сборника.

Архив всех текстов программ из книги можно взять на сайте автора http://vuz.exponenta.ru.

Автор будет благодарен всем, кто пришлет свои замечания о книге по адресу mpei2004@yandex.ru.

Глава 1

НЕОРИЕНТИРОВАННЫЕ ГРАФЫ

Основные определения. *Граф* $G(V, E)$ — совокупность двух множеств: вершин V и ребер E, между которыми определено отношение инцидентности. Каждое ребро e из E инцидентно ровно двум вершинам, v' и v'', которые оно соединяет. При этом вершина v' и ребро e называются *инцидентными* друг другу, а вершины v' и v'' называются *смежными*. Часто пишут v', v'' из G и e из G. Принято обозначение n для числа вершин графа (мощность множества V): $|V(G)| = n$, и m для числа его ребер: $|E(G)| = m$. Говорят, что граф G есть (n, m) граф, где n — *порядок* графа, m — *размер* графа.

Если все ребра (v_1, v_2) графа неориентированные, т.е. пары вершин, определяющие элементы множества E, неупорядочены, то такой граф называется неориентированным графом, или *неографом*.

Маршрут — последовательность ребер, в которой каждые два соседних ребра имеют общую вершину.

Маршрут в неографе, в котором все ребра разные, — *цепь*.

Граф *связен*, если любые две вершины соединены хотя бы одним маршрутом. Число ребер маршрута определяет его длину.

Цепь в графе называется *полуэйлеровой* (эйлеровой), если она содержит все ребра и все вершины графа.

Ребра, инцидентные одной паре вершин, называются параллельными или *кратными*.

Граф с кратными ребрами называется *мультиграфом*.

Ребро (v, v) называется *петлей* (концевые вершины совпадают). Граф, содержащий петли (и кратные ребра), называется *псевдографом*.

Степень $\deg(v)$ вершины — число ребер, инцидентных v. *В неографе сумма степеней всех вершин равна удвоенному числу ребер* (лемма о рукопожатиях):

$$\sum_{i=1}^{n} \deg(v_i) = 2m.$$

Петля дает вклад, равный 2, в степень вершины.

Степенная последовательность — последовательность степеней всех вершин графа, записанная в определенном порядке (по возрастанию или убыванию).

Матрица смежности графа — квадратная матрица A порядка n, где элемент a_{ij} равен числу ребер, соединяющих вершины i и j.

С графом связывают также матрицу *инцидентности* I. Число строк этой матрицы равно числу вершин, число столбцов — числу ребер; $i_{ve} = 1$, если вершина v инцидентна ребру e; в противном случае $i_{ve} = 0$. В каждом столбце матрицы инцидентности простого графа (без петель и без кратных ребер) содержится по две единицы. Число единиц в строке равно степени соответствующей вершины.

Граф $H(V_1, E_1)$ называется *подграфом* графа $G(V, E)$, если $V_1 \subseteq V$, $E_1 \subseteq E$. Если $V_1 = V$, то подграф называется *остовным*.

Компонента связности графа — максимальный по включению вершин и ребер связный подграф.

Ранг графа $\nu^* = n - k$, где k — число компонент связности [1].

Дерево — связный граф, содержащий $n - 1$ ребро [2].

1.1. Радиус и диаметр графа. Эйлерова цепь

Вычисление расстояний и определение маршрутов в графе являются одной из наиболее очевидных и практичных задач на графе. С одной из таких задач началась теория графов [3].

Введем некоторые необходимые определения.

Эксцентриситет вершины графа — расстояние до максимально удаленной от нее вершины.

Радиус графа — минимальный эксцентриситет вершин, а *диаметр* графа — максимальный эксцентриситет вершин.

Центр графа образуют вершины, у которых эксцентриситет равен радиусу. Центр графа может состоять из одной, нескольких или всех вершин графа.

Периферийные вершины имеют эксцентриситет, равный диаметру.

Простая цепь с длиной, равной диаметру графа, называется *диаметральной*.

Теорема 1. *В связном графе диаметр не больше ранга его матрицы смежности.*

Теорема 2 (Жордана [4])**.** *Каждое дерево имеет центр, состоящий из одной или двух смежных вершин.*

Теорема 3. *Если диаметр дерева четный, то дерево имеет единственный центр и все диаметральные цепи проходят через него, если диаметр нечетный, то центров два и все диаметральные цепи содержат ребро, их соединяющее.*

---

[1] Величина $\nu^* = n - k$ называется также коциклическим рангом графа [10]. Часто ранг графа связывается с рангом матрицы смежности ($\operatorname{rank} G = \operatorname{rank} A$). Будем придерживаться терминологии, принятой в Maple: $\operatorname{rank} G = \nu^* = n - k$.

[2] Подробнее см. с. 40.

[3] Задача Л. Эйлера о кенигсбергских мостах (1736 г.) [31].

[4] Jordan C.

Очевидно практическое значение центра графа. Если, например, речь идет о графе дорог с вершинами-городами, то в математическом центре целесообразно размещать административный центр, складские помещения и т.п. [1] В данной задаче расстояние оценивается в числе ребер. Этот же алгоритм можно применять и для взвешенного графа, где расстояния — это веса ребер. В качестве веса можно брать евклидовы расстояния (для *евклидовых* графов с вершинами, являющимися точками на плоскости или в пространстве), время или стоимость передвижения между пунктами. Для несвязных графов в указанном смысле центр не определяется.

Для проверки существования эйлеровой цепи используется известная теорема.

Теорема 4 (Эйлера [2]). *Мультиграф обладает эйлеровой цепью тогда и только тогда, когда он связен и число вершин нечетной степени равно* 0 *или* 2.

Вершины нечетной степени в этой теореме, очевидно, являются началом и концом цепи. Если таких вершин нет, то эйлерова цепь становится *эйлеровым циклом*. Граф, обладающий эйлеровым циклом, называется *эйлеровым*. Заметим, что в задаче о кенигсбергских мостах разыскивался именно эйлеров цикл — по условию требовалось пройти по всем семи мостам города Кенигсберга [3] через реку Преголь по одному разу и вернуться к исходной точке.

Наряду с эйлеровыми циклами [31] представляют интерес гамильтоновы циклы — простые циклы, проходящие через все вершины графа [4].

Задача. Найти радиус r, диаметр d и центр графа (рис. 1.1). Проверить наличие эйлеровой цепи.

а *б* *в*

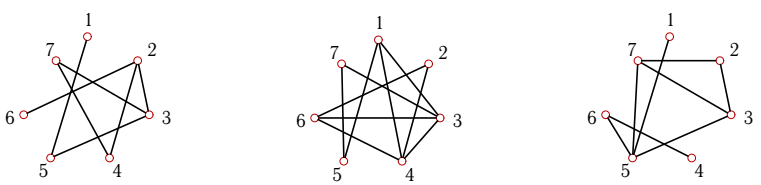


Рис. 1.1

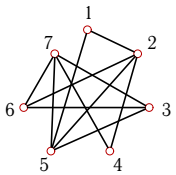
[1] *Кирсанов М.Н.* Математический центр московского метро //Exponenta Pro. Математика в приложениях. 2004. №4(4).

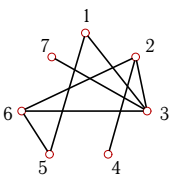
[2] Euler L.

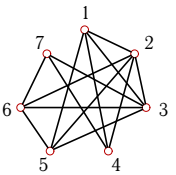
[3] Сейчас это г. Калининград.

[4] Подробнее см. с. 80.

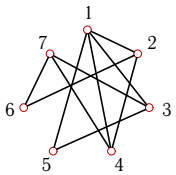
г *д* *е*

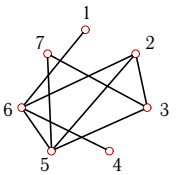






ж *з* *и*





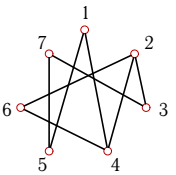


Рис. 1.1 (продолжение)

Ответы

	r	d	Центр	Эйлерова цепь
а	2	4	3	Нет
б	2	3	1, 3, 4	Есть
в	2	4	5	Нет
г	2	2	1, 2, 3, 4, 5, 6, 7	Есть
д	2	3	2, 3, 6	Нет
е	2	2	1, 2, 3, 4, 5, 6, 7	Нет
ж	2	3	1, 2, 3, 4, 7	Нет
з	2	3	2, 5, 6	Нет
и	3	3	1, 2, 3, 4, 5, 6, 7	Есть

Пример. Дан неограф (рис. 1.2). Найти радиус, диаметр и центр графа. Проверить наличие эйлеровой цепи.

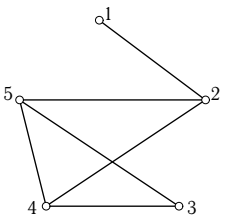


Рис. 1.2

Решение. Радиус и диаметр (способ 1). Находя цепи наименьшей длины, вычислим расстояния между вершинами. Результаты занесем в матрицу расстояний. Для неографа эта матрица симметричная. Для сокращения вычислений находим элементы половины матрицы, заполняя другую половину из условия симметрии:

$$S = \begin{vmatrix} 0 & 1 & 3 & 2 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{vmatrix}. \qquad (1.1)$$

Вычисляем эксцентриситет ε каждой вершины. Эту величину можно определять как максимальный элемент соответствующего столбца (или строки) матрицы расстояний. Получаем

№	ε
1	3
2	2
3	3
4	2
5	2

.

Радиус графа r — минимальный эксцентриситет вершин. В данном случае $r = 2$. Такой эксцентриситет имеют вершины № 2, № 4 и № 5. Эти вершины образуют центр графа. Диаметр графа d — максимальный эксцентриситет вершин. В данном случае $d = 3$. Такой эксцентриситет имеют вершины № 1 и № 3; это периферия графа. В исследованном графе вершины оказались либо центральными, либо периферийными. В графах большего порядка существуют и другие вершины.

Эффективный алгоритм Уоршолла и Флойда для определения кратчайших путей между всеми парами вершин графа, реализованный в Maple, приведен на с. 94.

Радиус и диаметр (способ 2). Эксцентриситеты вершин небольшого графа легко вычислять непосредственным подсчетом по рисунку. Однако не всегда граф задан своим рисунком. Кроме того, граф может иметь большой размер. Поэтому необходим другой способ решения задачи. Известна следующая теорема.

Теорема 5. *Пусть $A = \{\alpha_{ij}\}$ — матрица смежности графа G без петель и $A^k = \{\beta_{ij}\}$, где $k \in N$. Тогда β_{ij} равно числу маршрутов длины k от вершины v_i к вершине v_j* [1].

Решение задач теории графов с помощью различных преобразований матрицы смежности называют алгебраическим методом. Как правило, алгебраические методы годятся для графов небольшого порядка и особенно удобны для применения пакетов символьного вычисления или компьютерных систем со встроенными матричными алгоритмами.

[1] Такие маршруты часто называют (v_i, v_j)-маршрутами.

Построим матрицу смежности графа:

$$A = \begin{vmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{vmatrix}.$$

Будем заполнять матрицу расстояний, рассматривая степени матрицы смежности [1]. В первом приближении единицы матрицы смежности показывают пары вершин, расстояние между которыми равно 1 (т.е. они соединены одним ребром):

$$S_{(1)} = \begin{vmatrix} 0 & 1 & - & - & - \\ 1 & 0 & - & 1 & 1 \\ - & - & 0 & 1 & 1 \\ - & 1 & 1 & 0 & 1 \\ - & 1 & 1 & 1 & 0 \end{vmatrix}.$$

Диагональные элементы матрицы расстояний — нули. Умножаем матрицу смежности на себя:

$$A^2 = \begin{vmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 3 & 2 & 1 & 1 \\ 0 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 3 & 2 \\ 1 & 1 & 1 & 2 & 3 \end{vmatrix}.$$

Согласно теореме между вершинами 2 и 3, 1 и 4 и т. д. имеется некоторое число маршрутов длиной 2 (поскольку степень матрицы 2). Число маршрутов здесь не используется, важен сам факт наличия маршрута и его длина, на что и указывает ненулевой элемент степени матрицы, не совпадающий с элементом, отмеченным при вычислении маршрута меньшей длины. Проставляем 2 в незаполненные элементы матрицы расстояний и получаем следующее приближение:

$$S_{(2)} = \begin{vmatrix} 0 & 1 & - & 2 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ - & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{vmatrix}.$$

Осталось неизвестным расстояние между вершинами 1 и 3. Будем умножать матрицу смежности A саму на себя до тех пор, пока в матрице A^k

[1] Для вещественной симметрической матрицы A справедливо представление $A = B^{-1}CB$, где C — диагональная матрица. Возведение в степень при этом упрощается: $A^k = B^{-1}C^kB$. Диагональ матрицы C совпадает с собственными числами матрицы A (спектр графа).

не появится ненулевой элемент $a_{1,3}$. Тогда соответствующий элемент матрицы расстояний (или ее приближения) равен степени матрицы смежности: $s_{1,3} = k$. Получаем

$$A^3 = \begin{vmatrix} 0 & 3 & 2 & 1 & 1 \\ 3 & 2 & 2 & 6 & 6 \\ 2 & 2 & 2 & 5 & 5 \\ 1 & 6 & 5 & 4 & 5 \\ 1 & 6 & 5 & 5 & 4 \end{vmatrix},$$

следовательно, $s_{1,3} = 3$, и окончательно

$$S_{(3)} = \begin{vmatrix} 0 & 1 & 3 & 2 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{vmatrix}.$$

Полученная матрица совпадает с матрицей расстояний S (1.1), найденной непосредственными вычислениями по рисунку.

Эйлерова цепь. Граф на рис. 1.2 связен. Любые две пары вершин соединены непрерывной последовательностью ребер. Найдем степени вершин графа. Вершина № 1 имеет степень 1 (ей инцидентно одно ребро), вершина № 3 — 2, остальные вершины — степень 3. В данном случае в графе четыре вершины нечетной степени (№ 1, 2, 4, 5). Следовательно, согласно теореме 4, граф эйлеровой цепью не обладает, т.е. нет цепи, содержащей все ребра графа по одному разу.

Рассмотрим для сравнения граф, обладающий эйлеровой цепью. В графе на рис. 1.3 две вершины (№1 и 3) имеют нечетную степень — 3, следовательно, эйлерова цепь есть. Найти эту цепь — другая задача. Беспорядочное блуждание по графу не дает результата. Более того, цепей может быть несколько. Например, цепь 1–2–3–4–1–3 (рис. 1.4) и цепь 1–2–3–1–4–3 (рис. 1.5).

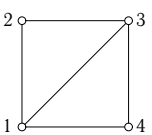


Рис. 1.3

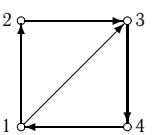


Рис. 1.4

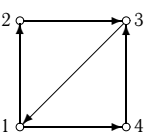


Рис. 1.5

Оценить число эйлеровых цепей можно, используя следствие из теоремы 4. Введем необходимое определение.

Говорят, что реберно-непересекающиеся цепи *покрывают* граф, если каждое ребро графа входит в одну из них [10].

Следствие. *В связном графе, содержащем k вершин нечетной степени, минимальное число покрывающих его реберно-непересекающихся цепей равно $k/2$.*

Если вершины графа удовлетворяют теореме 4, то множество покрывающих реберно-непересекающихся цепей состоит из одной — эйлеровой. В том числе и для $k = 2$ должна существовать одна эйлерова цепь, соединяющая вершины с нечетными степенями. Граф на рис. 1.3 относится к этому случаю.

Три программы нахождения радиуса графа и программа проверки наличия эйлеровой цепи в системе Maple приведены на с. 92–94.

1.2. Реберный граф

Для произвольного графа G реберный [1] граф $L(G)$ определяется следующими условиями:

1) множество вершин реберного графа $L(G)$ совпадает с множеством ребер графа G: $VL(G) = EG$;

2) вершины v_i и v_j смежны в $L(G)$ тогда и только тогда, когда ребра e_i и e_j смежны в G.

Английское название [2] реберного графа — *line graph*, отсюда и обозначение $L(G)$.

Теорема 6. *Если $d_1, d_2, ..., d_n$ — степенная последовательность (n, m) графа G, то $L(G)$ является (m, m_1)-графом, где*

$$m_1 = \frac{1}{2}\sum_{i=1}^{n} d_i^2 - m. \tag{1.2}$$

Задача. По заданному графу (рис. 1.6) построить его реберный граф.

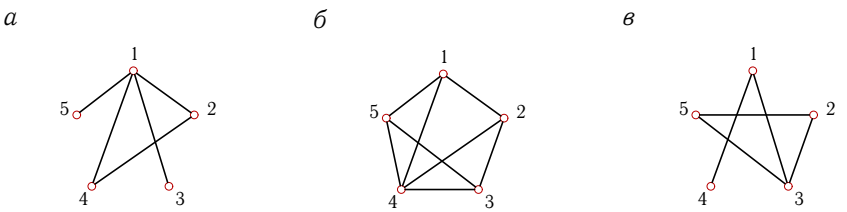
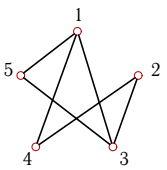


Рис. 1.6

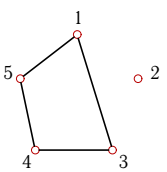
[1] Оре О. называет этот граф графом смежности ребер или *смежностным* графом [26].

[2] Многие английские термины теории графов, принятые в Maple, приведены в предметном указателе книги Н. Кристофидеса [18].

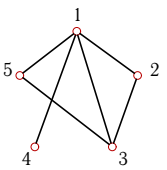
г



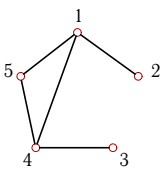
д



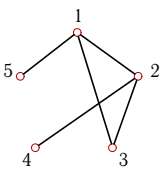
е



ж



з



и

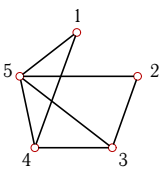
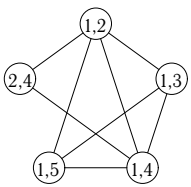


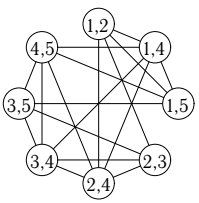
Рис. 1.6 (продолжение)

Ответы

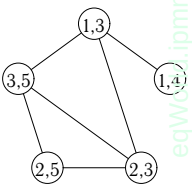
а



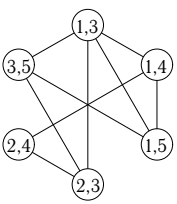
б



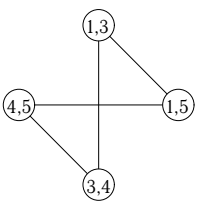
в



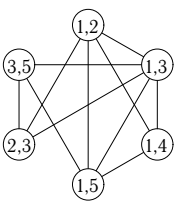
г



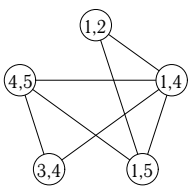
д



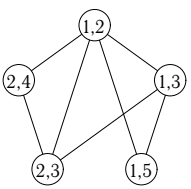
е



ж



з



и

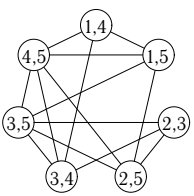


Рис. 1.7

Пример. Построить реберный граф для графа с рис. 1.2.

Решение. Решим задачу графически, с помощью прямого построения реберного графа по определению. В центре каждого ребра исходного графа G поместим вершину будущего реберного графа $L(G)$.

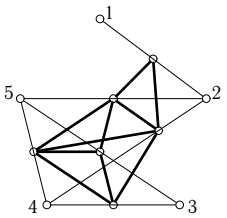


Рис. 1.8

Будем соединять полученные вершины ребрами графа $L(G)$, если ребра графа G, на которых лежат эти вершины, смежны. На рис. 1.8 ребра графа $L(G)$ для наглядности выделены более толстыми линиями. В результате реберный граф получит $m_1 = 10$ ребер и 6 вершин (по числу m ребер графа G).

Число m_1 соответствует значению, которое должно получиться по формуле (1.2). Степенная последовательность графа — 1–3–2–3–3. Находим

$$m_1 = \frac{1}{2}(1^2 + 3^2 + 2^2 + 3^2 + 3^2) - 6 = 16 - 6 = 10.$$

Таким образом, на рисунке совмещены исходный граф G (тонкие линии) и его реберный граф $L(G)$ (толстые линии).

Решение задачи о нахождении реберного графа в системе Maple приведено на с. 96.

1.3. Хроматический полином

Произвольная функция $f(v)$ на множестве вершин графа называется *раскраской* графа. Раскраска называется правильной, если $f(v_1) \neq f(v_2)$ для любых смежных вершин v_1 и v_2. Минимальное число k, при котором граф G является k-раскрашиваемым, называется *хроматическим числом* графа $\chi(G)$. Для хроматического числа имеются оценки.

Теорема 7 (Брукса [1]). *Для любого графа G, не являющегося полным, $\chi(G) \leqslant \Delta(G)$, если $\Delta(G) \geqslant 3$ — максимальная из степеней вершин графа.*

Для определения количества способов раскраски графа в x цветов можно составить *хроматический полином* $P(G, x)$. Значение полинома при некотором конкретном $x = x_0$ равно числу правильных раскрасок графа в x_0 цветов.

[1] Brooks R.L.

Существует лемма, утверждающая, что *хроматический полином графа имеет вид*

$$P(G,x) = P(G_1,x) + P(G_2,x), \tag{1.3}$$

где G_1 — граф, полученный из G добавлением нового ребра (u,v), а граф G_2 получается из G отождествлением вершин u и v.

Другой вариант леммы:

$$P(G,x) = P(G_1,x) - P(G_2,x), \tag{1.4}$$

где G_1 — граф, полученный из G удалением ребра (u,v), а граф G_2 получается из G отождествлением вершин u и v.

Операцию отождествления вершин u и v называют также *стягиванием* ребра (u,v). [1]

Оба варианта леммы составляют основу для хроматической редукции графа. Хроматическая редукция графа — представление графа в виде нескольких пустых или полных графов, сумма хроматических полиномов которых равна хроматическому полиному графа. Очевидно, что хроматический полином пустого графа O_n равен x^n (каждая вершина может быть раскрашена независимо от других), а для полного графа $P(K_n,x) = x(x-1)(x-2),...,(x-n+1)$. Последнее выражение называют *факториальной степенью* [2] переменной x: $P(K_n,x) = x^{(n)}$.

Разложения по пустым и полным графам связаны. Факториальную степень можно представить в виде полинома:

$$x^{(n)} = \sum_{k=0}^{n} s_1(n,k)x^k, \tag{1.5}$$

где $s_1(n,k)$ — числа Стирлинга первого рода, и наоборот, степень x^n можно выразить через факториальные степени:

$$x^n = \sum_{k=0}^{n} s_2(n,k)x^{(k)}, \tag{1.6}$$

[1] Граф G называется *стягиваемым* к графу H, если H получается из G последовательным стягиванием его ребер. *Число Хадвигера* (Hadwiger H.) графа G — максимальный порядок полного графа, к которому стягивается граф G.

[2] Факториальная степень связана с символом $(a)_k$ Похгаммера (Pochgammer L.), который определен как $(a)_k = a(a+1),...,(a+k-1)$. Очевидно, $(a)_k = (a+k-1)^{(k)}$. Для символа Похгаммера имеется большое число форумул (см., например, *Прудников А.П., Брычков Ю.А., Маричев О.И.* Интегралы и ряды. Дополнительные главы. — М.: Наука. 1986).

где $s_2(n,k)$ — числа Стирлинга [1] второго рода, обладающие следующими свойствами:

$$\begin{aligned} &s_2(n,k) = s_2(n-1,k-1) + k s_2(n-1,k) \text{ при } 0 < k < n, \\ &s_2(n,n) = 1 \text{ при } n \geqslant 0, \quad s_2(n,0) = 0 \text{ при } n > 0. \end{aligned} \tag{1.7}$$

При получении хроматического полинома могут быть полезны следующие теоремы [2].

Теорема 8. *Коэффициенты хроматического полинома составляют знакопеременную последовательность.*

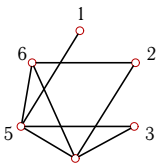
Теорема 9. *Абсолютная величина второго коэффициента хроматического полинома равна числу ребер графа.*

Теорема 10. *Наименьшее число i, для которого отличен от нуля коэффициент при x^i в хроматическом полиноме графа G, равно числу компонент связности графа G.*

Кроме вершинной раскраски, существуют еще реберная раскраска и раскраска граней.

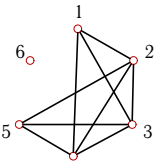
Задача. Найти хроматический полином графа (рис. 1.9) и вычислить количество способов раскраски графа в x_0 цветов.

а



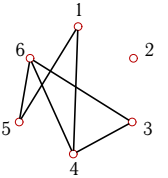
$x_0 = 4$

б



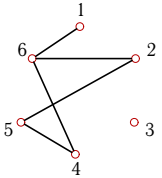
$x_0 = 3$

в



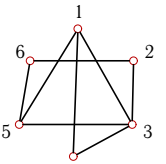
$x_0 = 6$

г



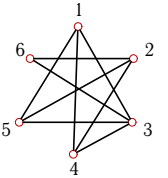
$x_0 = 5$

д



$x_0 = 4$

е

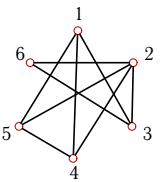


$x_0 = 5$

Рис. 1.9

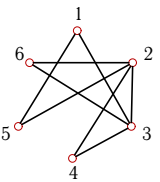
[1] Stirling T.

ж



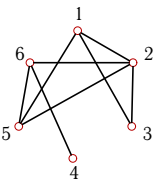
$x_0 = 6$

з



$x_0 = 6$

и



$x_0 = 5$

Рис. 1.9 (продолжение)

Ответы

	$C(x)$	$C(x_0)$
а	$x^6 - 8x^5 + 25x^4 - 38x^3 + 28x^2 - 8x$	288
б	$x^6 - 9x^5 + 29x^4 - 39x^3 + 18x^2$	0
в	$x^6 - 6x^5 + 14x^4 - 15x^3 + 6x^2$	15120
г	$x^6 - 5x^5 + 10x^4 - 9x^3 + 3x^2$	5200
д	$x^6 - 8x^5 + 26x^4 - 43x^3 + 36x^2 - 12x$	336
е	$x^6 - 9x^5 + 34x^4 - 66x^3 + 64x^2 - 24x$	1980
ж	$x^6 - 9x^5 + 33x^4 - 61x^3 + 56x^2 - 20x$	8160
з	$x^6 - 8x^5 + 26x^4 - 43x^3 + 36x^2 - 12x$	10080
и	$x^6 - 8x^5 + 25x^4 - 38x^3 + 28x^2 - 8x$	2160

Пример. Найти хроматический полином графа (рис. 1.10).

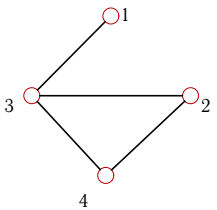
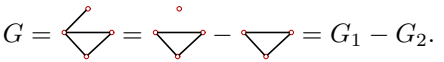


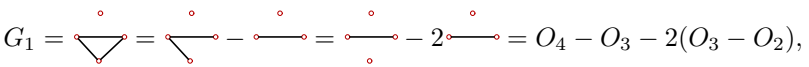
Рис. 1.10

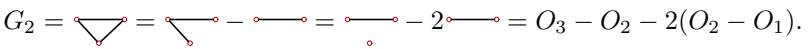
Решение. В зависимости от числа ребер графа можно использовать разложение (1.3) или (1.4). Если граф почти полный, то, добавив несколько ребер по разложению (1.3), получим хроматический полином в виде суммы факториальных степеней. Если же ребер мало и для получения пустого графа требуется удалить только несколько ребер, то следует использовать разложение (1.4) с удалением ребер. Такие действия называются хроматической редукцией.

1. *Хроматическая редукция по пустым графам.* Воспользуемся леммой (1.4). Удаляя ребра и отождествляя соответствующие вершины (стягивая ребра), сведем исходный граф к пустым графам. Сначала разложим граф на два, убрав, а затем стянув ребро 1–3. Выполненное действие запишем в виде условного равенства:



Здесь операция сложения (или вычитания) относится не к самому графу, а к его хроматическому полиному. Таким образом, последнее равенство означает, что $P(G) = P(G_1) - P(G_2)$. Для сокращения записи обозначение $P(...)$ будем опускать. Далее разложим каждый из графов, G_1 и G_2, пользуясь той же леммой:





Приведем подобные члены:

$$G = G_1 - G_2 = O_4 - O_3 - 2(O_3 - O_2) - \\ - (O_3 - O_2 - 2(O_2 - O_1)) = O_4 - 4O_3 + 5O_2 - 2O_1. \tag{1.8}$$

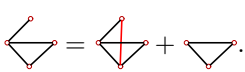
В итоге получим искомый хроматический полином:

$$P(G, x) = x^4 - 4x^3 + 5x^2 - 2x. \tag{1.9}$$

Разложение (1.8) называется хроматической редукцией графа по пустым графам.

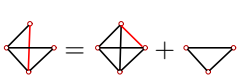
Очевидно, результат соответствует утверждениям теорем 8–10. Коэффициенты в (1.9) образуют знакопеременную последовательность, а коэффициент при x^3 равен четырем — числу ребер. Наименьшая степень x в полиноме равна 1, т.е. числу компонент связности графа.

2. *Хроматическая редукция по полным графам*. Добавив к графу (см. рис. 1.10) ребро 1–4, получим граф с большим числом ребер. Затем в этом же (исходном) графе отождествим вершины 1 и 4. В результате получим два графа:



(1.10)

Отождествление вершин приводит к уменьшению порядка и иногда размера графа. Второй граф — это полный граф K_3, его преобразовывать больше не требуется. К первому графу добавим ребро 1–2 и отождествим вершины 1 и 2:



(1.11)

В итоге

$$= K_4 + 2K_3. \tag{1.12}$$

Хроматический полином примет вид

$$\begin{aligned} P(G,x) = x^{(4)} + 2x^{(3)} = x(x-1)(x-2)(x-3) + \\ + 2x(x-1)(x-2) = x^4 - 4x^3 + 5x^2 - 2x. \end{aligned} \tag{1.13}$$

Разложение (1.12) называется хроматической редукцией графа по полным графам.

Оба способа дали один результат, и из редукции по полным графам легко получить редукцию по пустым. Для этого достаточно раскрыть скобки и привести подобные члены, как в (1.13). Однако обратное действие не очевидно. Для того чтобы полином $x^4 - 4x^3 + 5x^2 - 2x$, полученный из пустых графов, выразить в виде суммы факториальных степеней, необходимы числа Стирлинга 2-го рода. Согласно рекуррентным формулам (1.7) имеем следующие числа:

$$\begin{aligned} &s_2(k,1) = 1, \quad k = 1,2,..., \\ &s_2(3,2) = s_2(2,1) + 2s_2(2,2) = 3, \\ &s_2(4,2) = s_2(3,1) + 2s_2(3,2) = 1 + 2 \cdot 3 = 7, \\ &s_2(4,3) = s_2(3,2) + 3s_2(3,3) = 3 + 3 = 6. \end{aligned}$$

Пользуясь (1.6) и найденными числами Стирлинга 2-го рода, получим

$$\begin{aligned} x^2 &= x^{(1)} + x^{(2)}, \\ x^3 &= x^{(1)} + 3x^{(2)} + x^{(3)}, \\ x^4 &= x^{(1)} + 7x^{(2)} + 6x^{(3)} + x^{(4)}. \end{aligned}$$

Произведем преобразование хроматического полинома:

$$\begin{aligned} &x^4 - 4x^3 + 5x^2 - 2x = x^{(1)} + 7x^{(2)} + 6x^{(3)} + x^{(4)} - \\ &-4(x^{(1)} + 3x^{(2)} + x^{(3)}) + 5(x^{(1)} + x^{(2)}) + 2x^{(1)} = x^{(4)} + 2x^{(3)}. \end{aligned}$$

Хроматическое число $\chi(G)$ графа лучше всего получить, разложив хроматический полином на сомножители:

$$P(G,x) = x(x-1)^2(x-2).$$

Минимальное натуральное число x, при котором $P(G,x)$ не обращается в нуль, равно 3. Отсюда получаем $\chi(G) = 3$. Так как максимальная степень вершин графа $\Delta(G) = 3$, выполняется оценка $\chi(G) \leqslant \Delta(G)$ (см. с. 16).

Maple-программа для нахождения хроматического полинома графа с использованием оператора `chrompoly` приведена на с. 97.

1.4. Ранг-полином графа

Ранг графа определяется как $\nu^* = n - k$, где n — число вершин, k — число компонент связности графа. Коранг графа, или цикломатический ранг, есть $\nu = m - \nu^* = m - n + k$, где m — число ребер.

Ранг-полином [1] графа G имеет вид

$$P_\nu(x, y) = \sum x^{\nu^*_G - \nu^*_H} y^{\nu_H},$$

где $\nu^*_G = n - k$ — ранг графа G, ν_H — коранг остовного (т.е. включающего в себя все вершины графа) подграфа H, а ν^*_H — его ранг. Суммирование ведется по всем остовным подграфам графа G.

Ранг-полином служит для анализа множества остовных подграфов. Так, например, коэффициент при x^{-k} в $P_\nu(x, 1/x)$ есть число подграфов размера k, а значение $P_\nu(0, 1)$ равно числу подграфов (включая несобственный подграф), ранг которых равен рангу самого графа. Другие свойства ранга-полинома приведены в программе 6 на с. 98.

Задача. Найти ранг-полином графа (рис. 1.11).

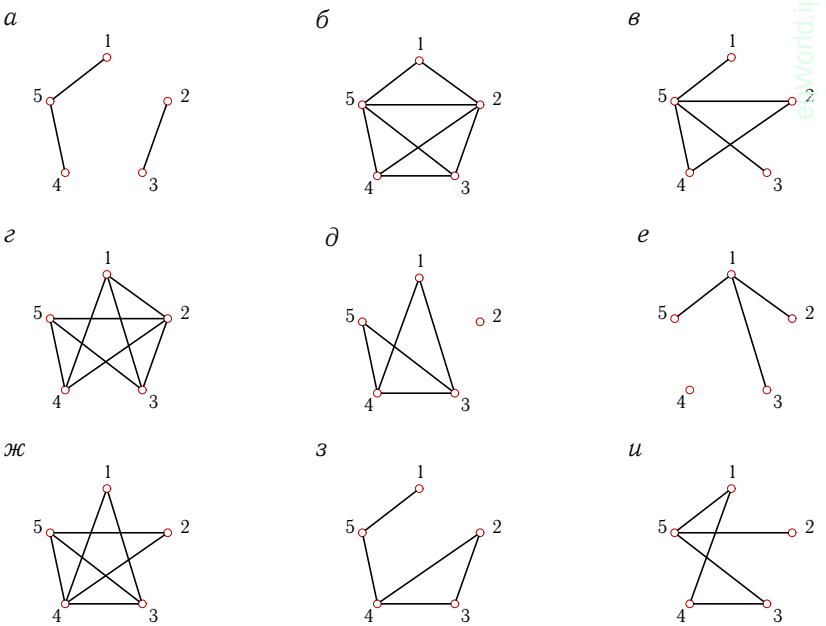


Рис. 1.11

[1] Название этого полинома (rankpoly) заимствовано из Maple.

Ответы

	$P_\nu(x,y)$
а	$x^3+3\,x^2+3\,x+1$
б	$y^4+(x+8)\,y^3+(8\,x+27)\,y^2+(5\,x^2+30\,x+48)\,y+x^4+8\,x^3+$ $+\,28\,x^2+51\,x+40$
в	$(x^2+2\,x+1)\,y+x^4+5\,x^3+10\,x^2+9\,x+3$
г	$y^5+9\,y^4+(3\,x+36)\,y^3+(2\,x^2+21\,x+81)\,y^2+(13\,x^2+58\,x+$ $+\,x^3+105)\,y+x^4+9\,x^3+71\,x+35\,x^2+66$
д	$y^2+(2\,x+5)\,y+x^3+5\,x^2+10\,x+8$
е	$x^3+3\,x^2+3\,x+1$
ж	$y^3+(2\,x+7)\,y^2+(14\,x+3\,x^2+19)\,y+x^4+7\,x^3+21\,x^2+21+32\,x$
з	$(x^2+2\,x+1)\,y+x^4+5\,x^3+10\,x^2+9\,x+3$
и	$(x+1)\,y+x^4+5\,x^3+10\,x^2+10\,x+4$

Пример. Найти ранг-полином графа (рис. 1.12).

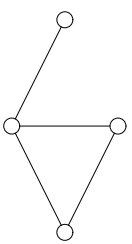


Рис. 1.12

Решение. Найдем все 16 остовных подграфов графа G (рис. 1.13–1.15). Множество представим в виде четырех графов размера 1 (т.е. с одним ребром), шести графов размера 2, четырех графов размера 3 и двух несобственных графов (пустой граф и граф G). Найдем для каждого подграфа ранг (по формуле $\nu^*=4-k$, где k — число компонент подграфа, включая изолированные вершины) и коранг ($\nu=m-$ $-\,\nu^*$, где m — число ребер подграфа).

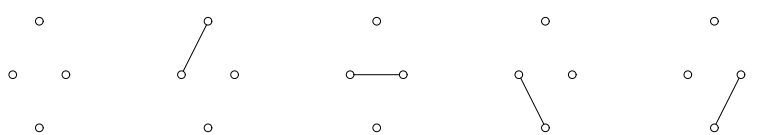


Рис. 1.13

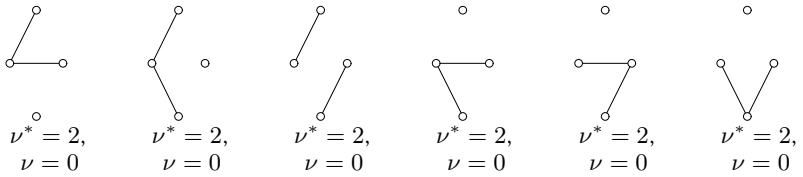


Рис. 1.14

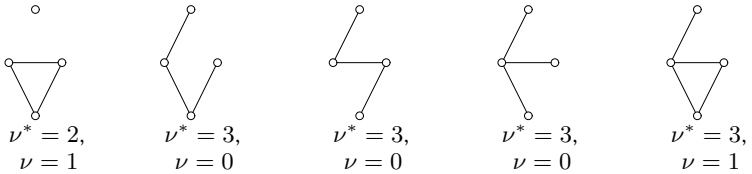


Рис. 1.15

Учитывая, что ранг ν^*_G графа равен 3, получаем сумму

$$\sum x^{\nu^*_G - \nu^*_H} y^{\nu_H} =$$
$$= x^{3-0}y^0 + 4x^{3-1}y^0 + 6x^{3-2}y^0 + 3x^{3-3}y^0 + x^{3-2}y^1 + x^{3-3}y^1 =$$
$$= x^3 + 4\,x^2 + 6\,x + 3 + x\,y + y.$$

Программа нахождения ранга-полинома графа в системе Maple приведена на с. 98.

1.5. Циклы

Маршрут, в котором начало и конец совпадают, — *циклический*. Циклический маршрут называется *циклом*, если он — цепь.

Остовом графа G называют граф, не содержащий циклов и состоящий из ребер графа G и всех его вершин. Остов графа определяется неоднозначно.

Ребра графа, не входящие в остов, называются *хордами*. Цикл, получающийся при добавлении к остову графа его хорды, называется *фундаментальным* относительно этой хорды.

Теорема 11. *Число ребер неографа, которые необходимо удалить для получения остова, не зависит от последовательности их удаления и равно цикломатическому рангу графа.*

Задача. По заданной матрице смежности (рис. 1.16) определить число циклов длины 3 (c_3) и длины 4 (c_4). Записать матрицу инцидентности и матрицу фундаментальных циклов.

а

$$\begin{vmatrix}0&0&0&1&0&0&0&0&0&0\\0&0&1&1&0&1&0&0&0\\0&1&0&0&1&1&0&0&0\\1&1&0&0&1&0&0&1&0\\0&0&1&1&0&1&1&1&1\\0&1&1&0&1&0&0&1&0\\0&0&0&0&1&0&0&1&0\\0&0&0&1&1&1&1&0&1\\0&0&0&0&1&0&0&1&0\end{vmatrix}$$

б

$$\begin{vmatrix}0&1&0&0&0&0&0&0&0&0\\1&0&1&1&1&1&0&0&0\\0&1&0&0&1&1&0&0&0\\0&1&0&0&1&0&0&1&0\\0&1&1&1&0&0&1&1&1\\0&1&1&0&0&0&0&1&1\\0&0&0&0&1&0&0&1&0\\0&0&0&1&1&1&1&0&0\\0&0&0&0&1&1&0&0&0\end{vmatrix}$$

в

$$\begin{vmatrix}0&1&0&1&0&0&0&0&0\\1&0&0&1&0&1&0&0&0\\0&0&0&0&1&1&0&0&0\\1&1&0&0&1&0&1&1&0\\0&0&1&1&0&1&1&0&1\\0&1&1&0&1&0&0&1&1\\0&0&0&1&1&0&0&0&0\\0&0&0&1&0&1&0&0&1\\0&0&0&0&1&1&0&1&0\end{vmatrix}$$

г

$$\begin{vmatrix}0&1&0&0&0&0&0&0&0&0\\1&0&1&1&0&1&0&0&0\\0&1&0&0&1&1&0&0&0\\0&1&0&0&1&0&1&1&0\\0&0&1&1&0&1&1&0&1\\0&1&1&0&1&0&0&1&1\\0&0&0&1&1&0&0&0&0\\0&0&0&1&0&1&0&0&1\\0&0&0&0&1&1&0&1&0\end{vmatrix}$$

д

$$\begin{vmatrix}0&1&0&1&0&0&0&0&0\\1&0&1&1&0&1&0&0&0\\0&1&0&0&1&1&0&0&0\\1&1&0&0&1&0&1&1&0\\0&0&1&1&0&0&1&1&1\\0&1&1&0&0&0&0&1&0\\0&0&0&1&1&0&0&0&0\\0&0&0&1&1&1&0&0&1\\0&0&0&0&1&0&0&1&0\end{vmatrix}$$

е

$$\begin{vmatrix}0&1&0&1&0&0&0&0&0\\1&0&0&1&0&1&0&0&0\\0&0&0&0&1&1&0&0&0\\1&1&0&0&1&0&1&1&0\\0&0&1&1&0&1&1&0&1\\0&1&1&0&1&0&0&1&1\\0&0&0&1&1&0&0&0&0\\0&0&0&1&0&1&0&0&1\\0&0&0&0&1&1&0&1&0\end{vmatrix}$$

ж

$$\begin{vmatrix}0&1&0&0&1&0&0&0&0\\1&0&1&1&1&1&0&0&0\\0&1&0&0&1&1&0&0&0\\0&1&0&0&1&0&1&1&0\\1&1&1&1&0&0&1&1&1\\0&1&1&0&0&0&0&1&0\\0&0&0&1&1&0&0&0&0\\0&0&0&1&1&1&0&0&1\\0&0&0&0&1&0&0&1&0\end{vmatrix}$$

з

$$\begin{vmatrix}0&1&0&0&1&0&0&0&0\\1&0&1&1&1&1&0&0&0\\0&1&0&0&1&0&0&0&0\\0&1&0&0&1&0&1&1&0\\1&1&1&1&0&1&1&0&1\\0&1&0&0&1&0&0&1&1\\0&0&0&1&1&0&0&0&0\\0&0&0&1&0&1&0&0&1\\0&0&0&0&1&1&0&1&0\end{vmatrix}$$

и

$$\begin{vmatrix}0&0&0&1&1&0&0&0&0\\0&0&1&1&1&1&0&0&0\\0&1&0&0&1&0&0&0&0\\1&1&0&0&1&0&1&1&0\\1&1&1&1&0&1&1&1&1\\0&1&0&0&1&0&0&1&1\\0&0&0&1&1&0&0&0&0\\0&0&0&1&1&1&0&0&0\\0&0&0&0&1&1&0&0&0\end{vmatrix}$$

Рис. 1.16

Ответы

№	c_3	c_4	№	c_3	c_4	№	c_3	c_4
а	1	6	г	5	30	ж	6	36
б	4	24	д	4	24	з	6	36
в	5	30	е	5	30	и	6	36

Пример. По заданной матрице смежности:

$$\begin{vmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{vmatrix},$$

определить число циклов длины 3 (c_3) и длины 4 (c_4). Записать матрицу инцидентности и матрицу фундаментальных циклов.

Решение. Матрица смежности данного графа симметричная, поэтому ей соответствует неориентированный граф. Сумма элементов матрицы равна 12, следовательно, по лемме о рукопожатиях (см. с. 7) в графе 6 ребер. Построим этот граф (рис. 1.17). Очевидно, в нем два цикла (3–4–5 и 1–3–5) длиной 3 и один цикл (1–3–4–5) длиной 4. В данной задаче решение получено прямым подсчетом по изображению графа. Для более сложных случаев существует алгоритм решения задачи по матрице смежности.

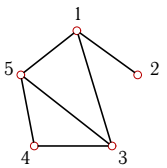


Рис. 1.17

Известно, что след (trace) матрицы смежности, возведенной в k-ю степень, равен числу циклических маршрутов длины k [1]. Это число включает в себя и искомое число циклов. Цикл отличается от циклического маршрута тем, что в нем не повторяются ребра. Кроме того, предполагается, что искомые циклы не помечены, а в след матрицы входят именно помеченные маршруты. Непомеченных циклов длиной 3 в 6 раз меньше, чем помеченных, так как каждый помеченный цикл может отличаться началом, а их в данном случае три, и двумя направлениями обхода (по и против часовой стрелки). Возведем заданную матрицу смежности в третью степень:

$$A^3 = \begin{bmatrix} 2 & 3 & 6 & 2 & 6 \\ 3 & 0 & 1 & 2 & 1 \\ 6 & 1 & 4 & 5 & 5 \\ 2 & 2 & 5 & 2 & 5 \\ 6 & 1 & 5 & 5 & 4 \end{bmatrix}, \tag{1.14}$$

и получим

$$c_3 = \frac{1}{6}\mathrm{trace}A^3 = 2. \tag{1.15}$$

Возведение матрицы в степень лучше выполнить по программе системы Maple: `c3:=A^3`; при этом подключения пакета линейной алгебры

[1] См. теорему 5 на с. 11.

`LinearAlgebra` не требуется. Поскольку циклических маршрутов длиной 3, отличных от циклов длиной 3, не существует, найденное число и есть ответ в поставленной задаче. След матрицы в пакете `LinearAlgebra` системы Maple вычисляется оператором `Trace(c3)`.

С циклами длиной 4 немного сложнее. В след четвертой степени матрицы смежности графа:

$$A^4 = \begin{bmatrix} 15 & 2 & 10 & 12 & 10 \\ 2 & 3 & 6 & 2 & 6 \\ 10 & 6 & 16 & 9 & 15 \\ 12 & 2 & 9 & 10 & 9 \\ 10 & 6 & 15 & 9 & 16 \end{bmatrix}, \tag{1.16}$$

входят не только циклы, но и циклические маршруты с двойным и четырехкратным прохождением ребер. Обозначим количества таких маршрутов через x_2 и x_4 соответственно. Очевидно, число маршрутов с четырехкратным прохождением одного ребра для вершины v_i равно степени этой вершины: $x_4 = \sum\limits_{i=1}^{n} \deg(v_i)$. Число маршрутов с двукратным прохождением ребра складывается из числа x_{2v} маршрутов с висячей вершиной v_i и числа x_{2c} маршрутов с вершиной v_i в центре. Легко заметить, что $x_{2c} = \sum\limits_{i=1}^{n} \deg(v_i)(\deg(v_i) - 1)$. Число x_{2v} зависит от степеней вершин, соседних с v_i:

$$x_{2v} = \sum_{i=1}^{n} \sum_{\{k,i\} \subseteq G} (\deg(v_k) - 1),$$

где $\{k, i\}$ — ребро, инцидентное вершинам i и k.

Для графа на рис. 1.17 получим

$$\begin{aligned} &\operatorname{trace} A^4 = 60, \\ &x_{2v} = 4 + 2 + 5 + 4 + 5 = 20, \\ &x_{2c} = 6 + 0 + 6 + 2 + 6 = 20, \\ &x_4 = 3 + 1 + 3 + 2 + 3 = 12. \end{aligned}$$

С учетом того, что непомеченных циклов длиной 4 в 8 раз меньше, получим

$$c_4 = (\operatorname{trace} A^4 - x_{2c} - x_{2v} - x_4)/8 = (60 - 20 - 20 - 12)/8 = 1.$$

После преобразований формула примет вид

$$c_4 = \frac{1}{8}\left(\operatorname{trace} A^4 - \sum_{i=1}^{n} \sum_{\{k,i\} \subseteq G} (\deg v_k - 1) - \sum_{i=1}^{n} \deg^2 v_i\right) = 1. \tag{1.17}$$

Матрица инцидентности. Число строк матрицы инцидентности I равно числу вершин, число столбцов — числу ребер; $i_{ve} = 1$, если вершина v инцидентна ребру e, в противном случае $i_{ve} = 0$. Матрицу строим по рис. 1.17, ребра нумеруем по рис. 1.18:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Два способа построения матрицы инцидентности в системе Maple рассмотрены в программе 8 на с. 100.

Матрица фундаментальных циклов. Пронумеруем ребра графа, начиная нумерацию с хорд (рис. 1.18).

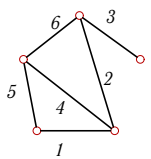


Рис. 1.18

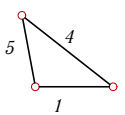


Рис. 1.19

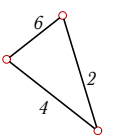


Рис. 1.20

Двум хордам, 1 и 2, соответствуют два фундаментальных цикла: *1–4–5* и *2–4–6* (рисунки 1.19, 1.20). Матрица фундаментальных циклов имеет две строки (число циклов) и шесть столбцов (число ребер):

	1	*2*	*3*	*4*	*5*	*6*
1	1	0	0	1	1	0
2	0	1	0	1	0	1

В первой строке матрицы единицами отмечены столбцы с номерами ребер, входящих в первый цикл, а во второй строке — номера ребер из второго цикла.

Глава 2

ОРИЕНТИРОВАННЫЕ ГРАФЫ

Ребро (v_1, v_2) в графе G может быть ориентированным и иметь начало и конец. Такое ребро называется дугой, а граф, содержащий дуги, называется ориентированным, или орграфом. На рисунке дуга изображается стрелкой, а в Maple дуга вводится как упорядоченная пара вершин: `[v1,v2]`. Многие понятия, введенные для неографов, для орграфов приобретают другое определение. Матрица расстояний для орграфа несимметрична, и эксцентриситет вершины зависит от того, как выбирается максимум. Если максимум ищется в строке, то эксцентриситет вершины называется числом внешнего разделения [18], а если в столбце — числом внутреннего разделения. Соответственно определяются внешний и внутренний центры.

Основание орграфа — неограф с теми же вершинами, но ребрами вместо соответствующих дуг.

Маршрут в орграфе — последовательность вершин, соединенных дугами, направленными в одну сторону.

Маршрут, в котором все дуги разные, есть *путь*.

Путь, в котором начало и конец совпадают, есть *контур*. Длина пути измеряется числом входящих в него дуг, а для взвешенного орграфа это сумма весов дуг.

В орграфе две локальные степени вершины v: $\deg^{\text{in}}(v)$ — число дуг, входящих в v, и $\deg^{\text{out}}(v)$ — число дуг, выходящих из v [1]. Лемма о рукопожатиях (см. с. 7) для орграфа имеет вид

$$\sum \deg^{\text{in}}(v) = \sum \deg^{\text{out}}(v) = m,$$

где суммирование производится по всем вершинам графа.

Множество вершин v, образующих дугу $[v, u]$ с вершиной u, называется множеством $\Gamma^{\text{in}}(u)$ предшественников вершины u, а множество вершин u, образующих дугу $[v, u]$ с вершиной v, называется множеством $\Gamma^{\text{out}}(v)$ преемников вершины v. Мощности этих множеств равны, соответственно, полустепеням вершин: $\Gamma^{\text{in}}(u) = \deg^{\text{in}}(u)$, $\Gamma^{\text{out}}(v) = \deg^{\text{out}}(v)$.

В дальнейшем под графом будем понимать как неограф, так и орграф.

[1] Иногда используются следующие обозначения: d^+ — полустепень исхода и d^- — полустепень захода.

2.1. Маршруты в орграфе

Задачи, связанные с маршрутами в орграфе, имеют большое практическое значение, что и дает стимул к развитию и совершенствованию методов их решения. Наиболее часто встает вопрос о минимальных и максимальных расстояниях, о числе маршрутов определенной длины.

Особенность таких задач для орграфов состоит в том, что несмотря на небольшой порядок графа (в приведенном ниже задании предлагается порядок 5) простое решение неочевидно. В следующей задаче для решения предлагается легко программируемый (особенно в системах компьютерной математики) алгебраический метод. Длина маршрута здесь измеряется в числе дуг, входящих в него. Допускаются замкнутые маршруты.

Задача. Дан орграф порядка 5 (рис. 2.1). Сколько в нем маршрутов длины 3?

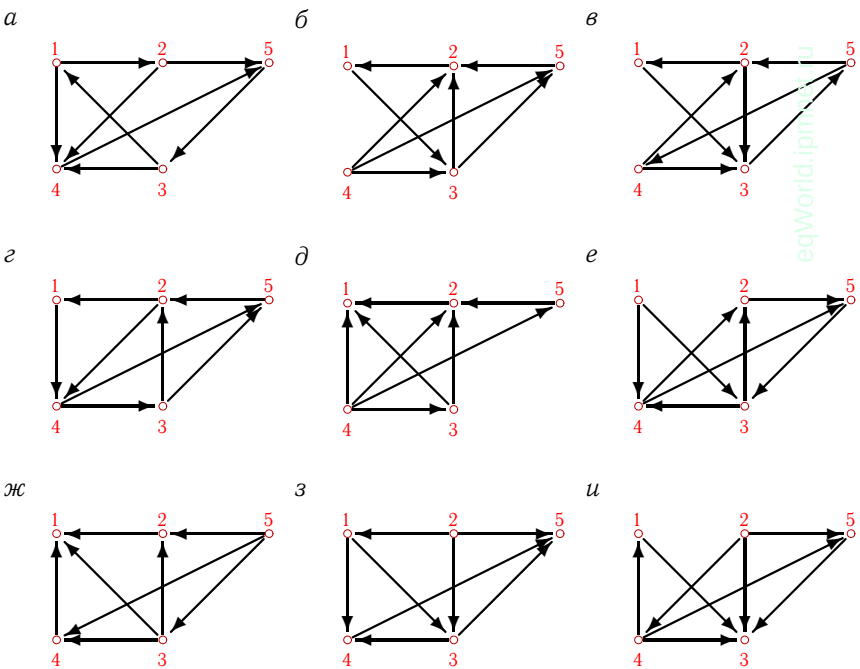


Рис. 2.1

Ответы

	а	б	в	г	д	е	ж	з	и
n_3	15	11	18	21	2	16	2	5	2

Пример. Дан орграф (рис. 2.2). Сколько в нем маршрутов длиной 3?

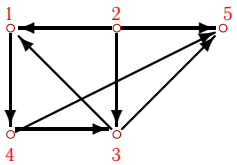


Рис. 2.2

Решение. Используем алгебраический метод решения задачи на основании теоремы 5 (см. с. 11). Запишем матрицу смежности. Матрица смежности орграфа — несимметричная:

$$A := \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Возведем эту матрицу в степень 3:

$$A^3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Суммируя все элементы полученной матрицы, находим, что число маршрутов длиной 3 равно восьми. Три единицы, стоящие по диагонали, показывают, что сюда входит 3 помеченных контура. Очевидно, это контуры 1–4–3, 4–3–1 и 3–1–4.

Проверить наличие контура в орграфе можно также методом топологической сортировки (см. с. 65). Если граф не может быть отсортирован топологически, то в нем есть контуры.

2.2. Транзитивное замыкание

Основные определения. *Прямым* (декартовым) произведением множеств A и B называется множество $A \times B = \{(x, y) |\ x \in A,\ y \in B\}$. *Бинарное отношение* на X — любое подмножество прямого произведения: $\rho \subset X \times X$.

Отношение ρ на X *рефлексивно*, если для любого $x \in X$ пара $(x, x) \in \rho$.

Отношение ρ на X *антирефлексивно*, если для любого $x \in X$ пара $(x, x) \notin \rho$.

Отношение ρ на X *симметрично*, если для любой пары (x, y) из условия $(x, y) \in \rho$ следует $(y, x) \in \rho$.

Отношение ρ на X *антисимметрично*, если из условий $(x, y) \in \rho$ и $(y, x) \in \rho$ следует $x = y$.

Отношение ρ на X *асимметрично*, если для любой пары (x, y) из условия $(x, y) \in \rho$ следует $(y, x) \notin \rho$.

Отношение ρ на X *транзитивно*, если для любых двух пар (x, y) и (y, z) из условий $(x, y) \in \rho$ и $(y, z) \in \rho$ следует $(x, z) \in \rho$.

Отношение $\tilde{\rho}$ называют *замыканием* отношения ρ на свойство A, если $\tilde{\rho}$ обладает свойством A, $\rho \subset \tilde{\rho}$ и для любого отношения σ со свойством A справедливо вложение $\tilde{\rho} \subset \sigma$.

Композицией бинарных отношений $\rho \subset X \times X$ и $\sigma \subset X \times X$ называют отношение $\tau = \sigma \circ \rho$, состоящее из пар (x, z), таких, что $(x, y) \subset \rho$, $(y, z) \subset \sigma$.

Транзитивное замыкание отношения ρ имеет вид $\rho^+ = \bigcup\limits_{k=1}^{\infty} \rho^k$, где $\rho^2 = \rho \circ \rho$, $\rho^k = \rho \circ \rho^{k-1}$.

Теорема 12. *Отношение ρ транзитивно тогда и только тогда, когда $\rho \circ \rho \subset \rho$.*

Граф есть отношение на множестве вершин. Элементами этого отношения являются дуги (или ребра, если отношение симметрично).

Орграф называется *транзитивным*, если для любых его дуг $[v_i, v_k]$, $[v_k, v_j]$ существует замыкающая дуга $[v_i, v_j]$.

Задача. Исследовать отношение, заданное матрицей (рис. 2.3), на симметрию, антисимметрию, асимметрию, рефлексивность, антирефлексивность. Найти транзитивное замыкание отношения. Построить граф отношения ρ и его транзитивного замыкания.

а

$$\begin{vmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

б

$$\begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

в

$$\begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

г

$$\begin{vmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

д

$$\begin{vmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix}$$

е

$$\begin{vmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

ж

$$\begin{vmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

з

$$\begin{vmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

и

$$\begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

Рис. 2.3

Ответы

а
$$\begin{vmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

б
$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

в
$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

г
$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

д
$$\begin{vmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{vmatrix}$$

е
$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

ж
$$\begin{vmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

з
$$\begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

и
$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

Рис. 2.4

Пример. Дано отношение, заданное матрицей

$$A = \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{vmatrix}.$$

Исследовать отношение на симметрию, антисимметрию, асимметрию, рефлексивность, антирефлексивность. Найти транзитивное замыкание отношения. Построить граф отношения ρ и его транзитивного замыкания.

Решение. Исследуем свойства данного отношения.

1. Данное отношение не является симметричным, так как матрица несимметрична. Например, пара (2, 1) принадлежит ρ, а пара (1, 2) ему не принадлежит.

2. Отношение антисимметрично, так как нет ни одной пары $a_{ij} = a_{ji} = 1$, $i \neq j$.

3. Отношение антисимметрично, но не асимметрично, так как на диагонали матрицы имеются элементы, равные 1.

4. Все диагональные элементы матрицы рефлексивного отношения равны 1. Данное отношение не является рефлексивным.

5. Отношение не обладает свойством антирефлексивности, так как диагональ матрицы ненулевая.

6. Данное отношение не является транзитивным, так как, например, пары (1, 4) и (4, 3) принадлежат ρ, а пара (1, 3) ему не принадлежит.

Найдем транзитивное замыкание графа, заданного отношением ρ. Процедура транзитивного замыкания сводится к добавлению в матрицу смежности графа минимального числа единиц, так, чтобы соответствующее отношение обладало свойством транзитивности.

Способ 1.

1. Вычисляем матрицу композиции $\rho^2 = \rho \circ \rho$. Для этого умножаем [1] матрицу саму на себя: $A_1 = AA$.

Для $i, j = 1, ..., 4$ вычисляем

$$a_{1_{ij}} = (a_{i1} \wedge a_{1j}) \vee (a_{i2} \wedge a_{2j}) \vee (a_{i3} \wedge a_{3j}) \vee (a_{i4} \wedge a_{4j}).$$

Получаем

$$A_1 = \begin{Vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{Vmatrix} \begin{Vmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{Vmatrix}.$$

2. Находим логическую сумму (дизъюнкцию) матриц. Поэлементная дизъюнкция матриц дает

$$A_2 = A \vee A_1 = \begin{Vmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{Vmatrix}.$$

3. Сравним A_2 и матрицу A. Если $A_2 = A$, то A_2 — искомая матрица. Если $A_2 \neq A$, то полагаем $A = A_2$, возвращаемся к п. 1 и повторяем всю процедуру для новой матрицы. В данном случае $A_2 \neq \neq A$. Принимаем

$$A = \begin{Vmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{Vmatrix}.$$

[1] Не путать произведение булевых матриц $A = BC$ с поэлементным логическим умножением $B \wedge C$. Очевидно, $a_{ij} = 1$, если хотя бы в одном случае k-й элемент i-й строки первого сомножителя и k-й элемент j-го столбца второго сомножителя одновременно равны 1. В противном случае $a_{ij} = 0$.

1′. Умножаем матрицу саму на себя:

$$A_1 = AA = \left|\begin{matrix}1&0&1&1\\1&0&0&1\\0&0&0&0\\0&0&1&1\end{matrix}\right| \cdot \left|\begin{matrix}1&0&1&1\\1&0&0&1\\0&0&0&0\\0&0&1&1\end{matrix}\right| = \left|\begin{matrix}1&0&1&1\\1&0&1&1\\0&0&0&0\\0&0&1&1\end{matrix}\right|.$$

2′. Находим логическую сумму (дизъюнкцию) матриц:

$$A_2 = A \vee A_1 = \left|\begin{matrix}1&0&1&1\\1&0&1&1\\0&0&0&0\\0&0&1&1\end{matrix}\right|.$$

3′. Сравниваем: $A_2 \neq A$. Полагаем $A = A_2$ и повторяем процедуру еще раз.

1″. Умножаем

$$A_1 = AA = \left|\begin{matrix}1&0&1&1\\1&0&1&1\\0&0&0&0\\0&0&1&1\end{matrix}\right| \cdot \left|\begin{matrix}1&0&1&1\\1&0&1&1\\0&0&0&0\\0&0&1&1\end{matrix}\right| = \left|\begin{matrix}1&0&1&1\\1&0&1&1\\0&0&0&0\\0&0&1&1\end{matrix}\right|.$$

2″. Находим сумму:

$$A_2 = A \vee A_1 = \left|\begin{matrix}1&0&1&1\\1&0&1&1\\0&0&0&0\\0&0&1&1\end{matrix}\right|.$$

3″. Сравниваем: $A_2 = A$. Следовательно, A_2 — матрица транзитивного замыкания заданного отношения.

Способ 2. Алгоритм Уоршелла [1] [28].

Рассматриваем все внедиагональные элементы матрицы. Если $a_{ij} \neq \neq 0$, то i-ю строку заменяем дизъюнкцией i-й и j-й строк.

1. Элемент $a_{14} = 1$. Первую строку заменяем поэлементной дизъюнкцией первой и четвертой строки:

$$A_1 = \left|\begin{matrix}1&0&1&1\\1&0&0&0\\0&0&0&0\\0&0&1&1\end{matrix}\right|.$$

[1] Warshall S.

2. Элемент $a_{21} = 1$. Вторую строку заменяем поэлементной дизъюнкцией второй и первой строки:

$$A_2 = \begin{vmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{vmatrix}.$$

3. Элемент $a_{43} = 1$. Дизъюнкция четвертой и третьей строки не меняет вид матрицы. Таким образом, полученная матрица является матрицей транзитивного замыкания отношения ρ.

Оба способа дают один и тот же результат.

На рисунках 2.5 и 2.6 представлены графы отношения ρ и его транзитивного замыкания. Диагональные элементы матрицы соответствуют петлям на графе. Матрица несимметрична, поэтому граф отношения ориентированный.

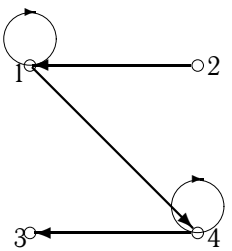


Рис. 2.5

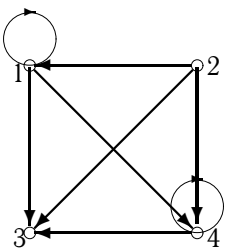


Рис. 2.6

Maple-программа для определения некоторых свойств отношения по его матрице приведена на с. 101. В программе дан также алгоритм транзитивного замыкания.

2.3. Компоненты сильной связности графа

Понятие сильной связности относится только к орграфам.

Основание орграфа — неограф с теми же вершинами, но ребрами вместо соответствующих дуг [1].

Орграф называется *связным*, если связно его основание.

Вершина u *достижима* из вершины v, если существует маршрут из v в u.

[1] Наоборот, каждому неориентированному графу *канонически* соответствует орграф с теми же вершинами, в котором каждое ребро заменено дугами, инцидентными тем же вершинам и имеющими противоположные направления. Кроме того, при вычислении матрицы Кирхгофа удобно вводить *ориентацию* (обычно произвольную) неографа — замену ребер дугами (см. с. 78).

Орграф называется *сильно связным* (или орсвязным), если любая его вершина достижима из любой вершины.

Граф называется *ориентируемым*, если он является основанием сильно связного графа.

Задача. Найти компоненты сильной связности орграфа (рис. 2.7).

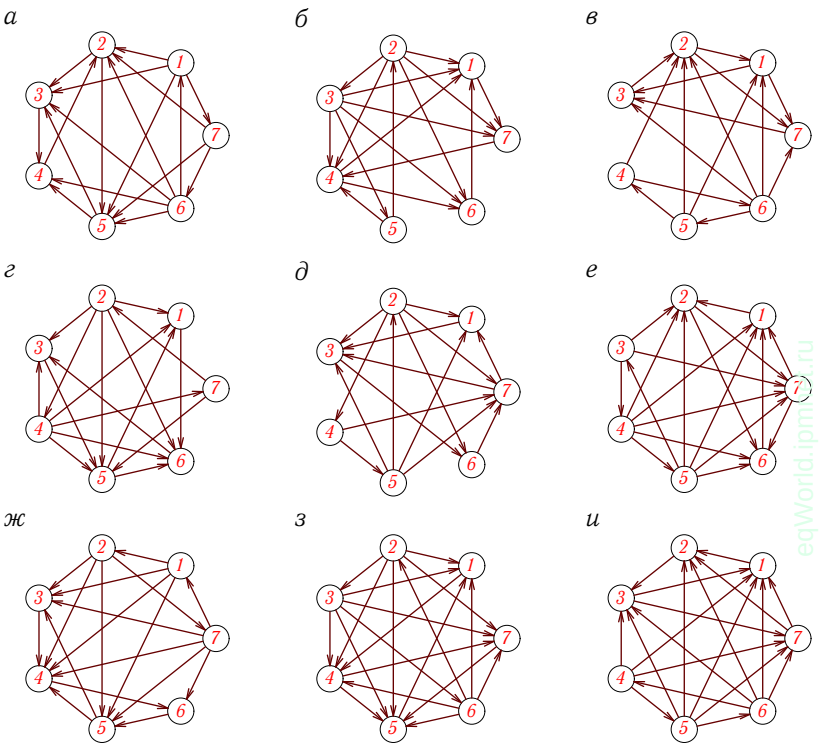


Рис. 2.7

Ответы

	1	2		1	2		1	2
а	2, 5, 3, 4	7, 6, 1	г	5, 1, 6, 3	4, 7, 2	ж	5, 3, 4, 6	7, 1, 2
б	4, 6, 1, 7	2, 3, 5	д	7, 1, 3, 6	2, 4, 5	з	4, 7, 5, 1	6, 2, 3
в	2, 1, 7, 3	5, 4, 6	е	2, 7, 6, 1	3, 4, 5	и	3, 7, 1, 2	4, 5, 6

Пример. Найти компоненты сильной связности графа (рис. 2.8).

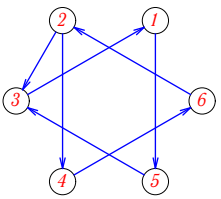


Рис. 2.8

Решение. Матрица смежности графа имеет вид

$$A_1 = A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

В графе 7 дуг, поэтому наибольший путь будет не длиннее семи. Построим матрицу достижимости:

$$A_7 = \sum_{k=1}^{7} A^k = \begin{bmatrix} 2 & 0 & 2 & 0 & 3 & 0 \\ 3 & 2 & 6 & 3 & 3 & 2 \\ 3 & 0 & 2 & 0 & 2 & 0 \\ 3 & 2 & 3 & 2 & 1 & 3 \\ 2 & 0 & 3 & 0 & 2 & 0 \\ 3 & 3 & 3 & 2 & 3 & 2 \end{bmatrix}.$$

Выделим из этой матрицы главные миноры максимального порядка, не содержащие нули. Если граф связен, то в матрице будут строки, не содержащие нулей. Это строки 2, 4, 6:

$$\begin{bmatrix} . & . & . & . & . & . \\ 3 & 2 & 6 & 3 & 3 & 2 \\ . & . & . & . & . & . \\ 3 & 2 & 3 & 2 & 1 & 3 \\ . & . & . & . & . & . \\ 3 & 3 & 3 & 2 & 3 & 2 \end{bmatrix}.$$

Минор со строками и столбцами с этими номерами соответствует одной компоненте связности:

$$A_{(2,4,6)} = \begin{bmatrix} . & . & . & . & . & . \\ . & 2 & . & 3 & . & 2 \\ . & . & . & . & . & . \\ . & 2 & . & 2 & . & 3 \\ . & . & . & . & . & . \\ . & 3 & . & 2 & . & 2 \end{bmatrix}.$$

Удалим из матрицы строки и столбцы с этими номерами. Получим минор, соответствующий второй компоненте связности:

$$A_{(1,3,5)} = \begin{bmatrix} 2 & . & 2 & . & 3 & . \\ . & . & . & . & . & . \\ 3 & . & 2 & . & 2 & . \\ . & . & . & . & . & . \\ 2 & . & 3 & . & 2 & . \\ . & . & . & . & . & . \end{bmatrix}.$$

Итак, в графе две компоненты сильной связности: подграф с вершинами 1, 3, 5 и подграф с вершинами 2, 4, 6. Изобразим обе компоненты сильной связности в виде отдельных графов (рисунки 2.9, 2.10). Общее число ребер в компонентах меньше размера исходного графа. Дуга [2, 3] не вошла ни в одну компоненту.

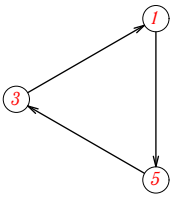


Рис. 2.9

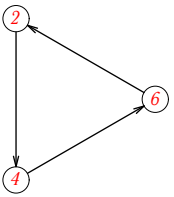


Рис. 2.10

Три варианта решения задачи о нахождении компонент сильной связности графа в системе Maple приведены на с. 104–106.

Глава 3

ДЕРЕВЬЯ

Дерево — связный граф без циклов. *Лес* (или *ациклический* граф) — неограф без циклов. Компонентами леса являются деревья.

Теорема 13. *Для неографа G с n вершинами без петель следующие условия эквивалентны:*

1) *G — дерево;*

2) *G — связный граф, содержащий $n-1$ ребро;*

3) *G — ациклический граф, содержащий $n-1$ ребро;*

4) *любые две несовпадающие вершины графа G соединяет единственная цепь;*

5) *G — ациклический граф, такой, что если в него добавить одно ребро, то в нем появится ровно один цикл.*

Теорема 14. *Неограф G является лесом тогда и только тогда, когда коранг графа $\nu(G) = 0$.*

Висячая вершина в дереве — вершина степени 1. Висячие вершины называются *листьями*, все остальные — *внутренними* вершинами.

Если в дереве особо выделена одна вершина, называемая *корнем*, то такое дерево называется *корневым*, иначе — *свободным*.

Корневое дерево можно считать орграфом с ориентацией дуг из корня или в корень. Очевидно, для любой вершины корневого дерева, кроме корня, $\deg^{\text{in}} = 1$. Для корня $\deg^{\text{in}} = 0$, для листьев $\deg^{\text{out}} = 0$.

Вершины дерева, удаленные на расстояние k (в числе дуг) от корня, образуют k-й *ярус* (уровень) дерева. Наибольшее значение k называется *высотой* дерева.

Если из вершины v корневого дерева выходят дуги, то вершины на концах этих дуг называются сыновьями [1].

3.1. Центроид дерева

Ветвь к вершине v дерева — это максимальный подграф, содержащий v в качестве висячей вершины. *Вес* c_k вершины k — наибольший размер ее ветвей. Центроид [2] дерева $\mathcal{C}$ — множество вершин с наименьшим весом: $\mathcal{C} = \{v | c(v) = c_{\min}\}$ [33].

[1] В английской литературе и Maple — дочери (daughter). Отсюда, вероятно, и термин «дочерние» (а не «сыновние») компании и т.п.

[2] Или *центр масс* дерева [26].

Вес любого листа дерева равен размеру дерева.

Высота дерева с корнем, расположенным в центроиде, не больше наименьшего веса его вершин.

Свободное дерево порядка n с двумя центроидами имеет четное количество вершин, а вес каждого центроида равен $n/2$.

Для центроида дерева существует теорема Жордана, аналогичная его же теореме о центре (см. теорему 2 на с. 8).

Теорема 15 (Жордана)**.** *Каждое дерево имеет центроид, состоящий из одной или двух смежных вершин* [33].

Задача. Найти центроид дерева (рис. 3.1) и наименьший вес его вершин.

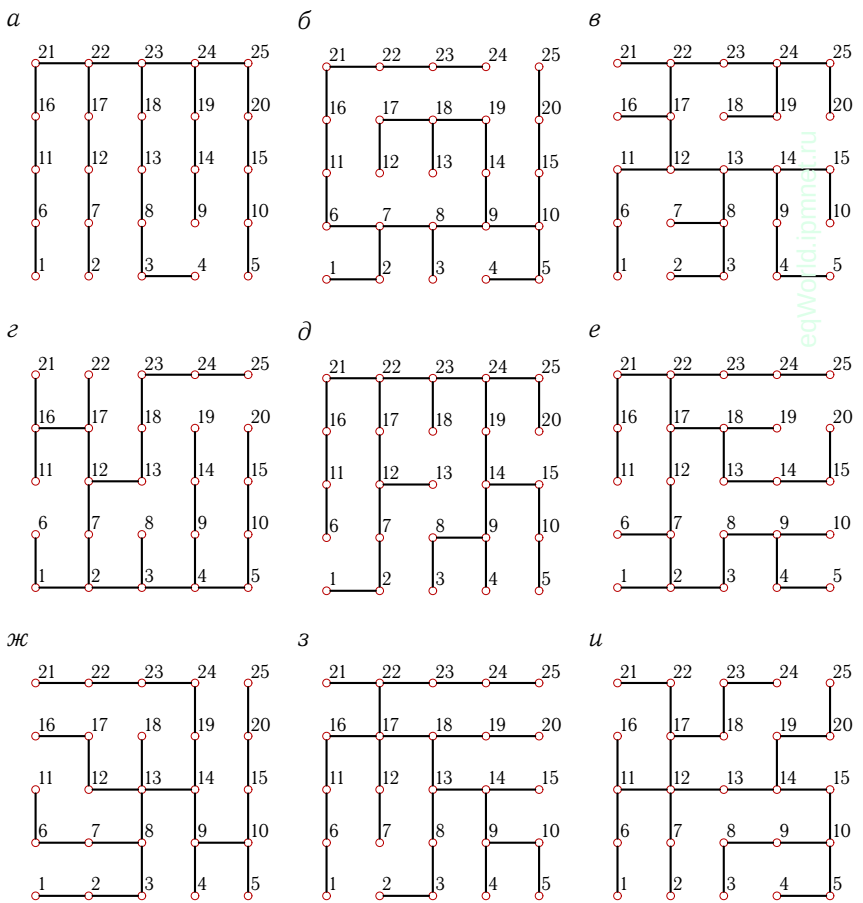


Рис. 3.1

Ответы

	$c_{\min}$	$\mathcal{C}$		$c_{\min}$	$\mathcal{C}$		$c_{\min}$	$\mathcal{C}$
а	10	23	г	12	2	ж	12	14
б	12	9	д	12	23	з	12	18
в	11	12	е	11	17	и	12	12

Пример. Найти наименьший вес вершин дерева (рис. 3.2) и его центроид.

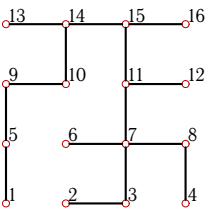


Рис. 3.2

Решение. Очевидно, вес каждой висячей вершины дерева порядка n равен $n-1$. Висячие вершины не могут составить центроид дерева, поэтому исключим из рассмотрения вершины 1, 2, 4, 6, 12, 13 и 16. Для всех остальных вершин найдем их вес, вычисляя длину (размер) их ветвей.

Число ветвей вершины равно ее степени. Размеры ветвей вершин 3, 5 и 8 равны 1 и 14. Следовательно, веса этих вершин равны 14. К вершине 7 подходят четыре ветви размером 1, 2, 2 и 10. Таким образом, ее вес $c_7 = 10$. Аналогично вычисляются веса других вершин: $c_9 = 13$, $c_{10} = c_{14} = 12$, $c_{11} = c_{15} = 8$. Минимальный вес вершин равен 8, следовательно, центроид дерева образуют две вершины с таким весом: 11 и 15.

3.2. Десятичная кодировка

Деревья представляют собой важный вид графов. С помощью деревьев описываются базы данных, деревья моделируют алгоритмы и программы, их используют в электротехнике, химии. Одной из актуальных задач в эпоху компьютерных и телекоммуникационных сетей является задача сжатия информации. Сюда входит и кодировка деревьев. Компактная запись дерева, полностью описывающая его структуру, может существенно упростить как передачу информации о дереве, так и работу с ним. Различные виды кодировки деревьев подробно описаны в [15].

Приведем одну из простейших кодировок помеченных деревьев с выделенным корнем — десятичную.

Кодируя дерево, придерживаемся следующих правил.

1. Кодировка начинается с корня и заканчивается в корне.
2. Каждый шаг на одну дугу от корня кодируется единицей.
3. В узле выбираем направление на вершину с меньшим номером.

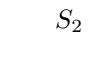
4. Достигнув листа, идем назад, кодируя каждый шаг нулем.

5. При движении назад в узле всегда выбираем направление на непройденную вершину с меньшим номером.

Кодировка в такой форме получается достаточно компактной, однако она не несет в себе информации о номерах вершин дерева. Существуют аналогичные кодировки, где вместо единиц в таком же порядке проставляются номера или названия вершин.

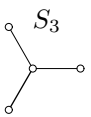
Есть деревья, для которых несложно вывести формулу десятичной кодировки. Рассмотрим, например, графы-звезды $K_{1,n}$, являющиеся полными двудольными графами, одна из долей которых состоит из одной вершины [1]. Другое обозначение звезд — $K_{1,n} = S_n$.

На рисунках 3.3–3.6 приведены звезды и их двоичные и десятичные кодировки. Корень дерева располагается в центральной вершине звезды. Легко получить общую формулу: $Z(S_n) = 2(4^n - 1)/3$.



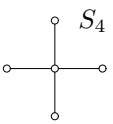
1010
10

Рис. 3.3



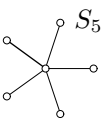
101010
42

Рис. 3.4



10101010
170

Рис. 3.5



1010101010
682

Рис. 3.6

Если корень поместить в любой из висячих вершин, то код Z' такого дерева будет выражаться бóльшим числом. Более того, существует зависимость $Z(S_n) - Z'(S_n) = Z(S_{n-1})$. Аналогично рассматриваются цепи [2] (C_n; рис. 3.7).

В звездах только два варианта расположения корня с различными десятичными кодировками. В цепи же число вариантов кодировок в зависимости от положения корня растет с увеличением n. Рассмотрим самый простой вариант, расположив корень в концевой вершине (листе). Для C_2 получим десятичную кодировку 10 и двоичную 2. Точно так же для остальных цепей: 1100 и 12, 111000 и 56, 11110000

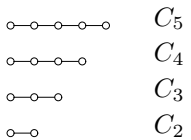


Рис. 3.7

[1] Эта же вершина является центром и центроидом дерева. Наименьший вес вершин звезды равен 1.

[2] У цепей C_{2n} и C_{2n+1} наименьший вес вершин равен n. Центр и центроид цепей совпадают.

и 240. Общая формула для десятичной кодировки цепи с корнем в концевой вершине имеет вид $Z(C_n) = 2^{n-1}(2^{n-1} - 1)$.

Задача. Записать десятичный код дерева (рис. 3.8) с корнем в вершине 7.

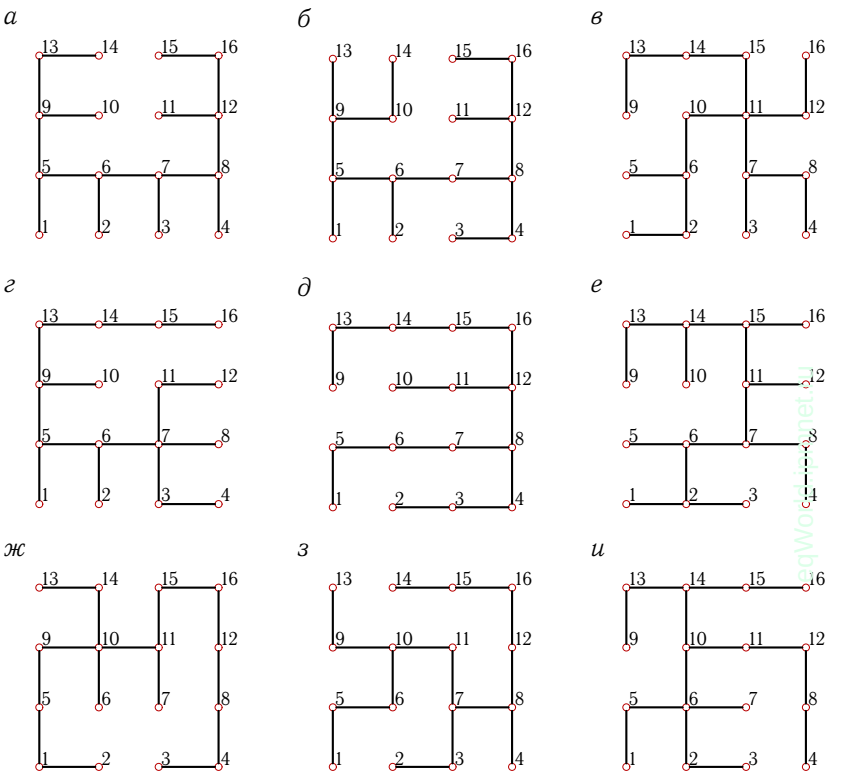


Рис. 3.8

Ответы

	Код(10)		Код(10)		Код(10)
а	766905776	г	862838828	ж	1002643328
б	920926640	д	955371392	з	863518256
в	754522592	е	978115976	и	966725216

Пример. Записать десятичный код дерева (рис. 3.9) с корнем в вершине 3.

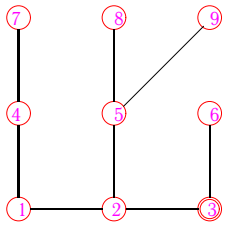


Рис. 3.9

Решение. На основании правила кодировки, двигаясь по дереву, проставим в код единицы и нули. При движении из корня 3 к вершине 7 проходим четыре ребра. В код записываем четыре единицы: 1111. Возвращаясь от вершины 7 к вершине 2 (до ближайшей развилки), проходим три ребра. Записываем в код три нуля: 000. От вершины 2 к 5 и далее к 8 (меньший номер): 11; от 8 назад к 5 и от 5 к 9: 01; от 9 к корню 3: 000.

И наконец, от 3 к 6 и обратно: 10. В итоге, собирая все вместе, получим двоичный код дерева:

$$1\ 111\ 000\ 110\ 100\ 010.$$

Разбивая число на тройки, переводим полученное двоичное представление в восьмеричное [1]. Получаем 1700642_8. Затем переводим это число в десятичное: $2 \cdot 8^0 + 4 \cdot 8^1 + 6 \cdot 8^2 + 0 \cdot 8^3 + 7 \cdot 8^4 + 1 \cdot 8^5 = 61858$.

Можно перевести двоичное число из n цифр в десятичное число непосредственно по формуле

$$\sum_{i=1}^{n} k_i 2^{n-i},$$

где k_i — i-я цифра (0 или 1) в двоичном числе.

Maple-программа для десятичной кодировки приведена на с. 114.

3.3. Кодировка Прюфера

Выбор кодировки дерева зависит от решаемой теоретической или технической задачи. Среди всех возможных кодировок естественно отыскать оптимальные по какому-то качеству решения. Впервые проблемой оптимальности кодировки деревьев занялся А.В. Анисимов (Об оптимальной упаковке деревьев// Кибернетика. — 1976. № 3. С. 89 – 91). Было показано, что существует оптимальный в определенном смысле код дерева — так называемый код Прюфера [2]. Это достаточно редко упоминаемое имя встречается в т. 1 книги Д. Кнута [17] и в книге О. Оре [26] в связи с выводом числа n^{n-2} помеченных деревьев [3]. Наиболее полно кодировка Прюфера и действия с числами (кодами)

[1] Имеем $000_2 = 0_8$, $001_2 = 1_8$, $010_2 = 2_8$, $011_2 = 3_8$, $100_2 = 4_8$, $101_2 = 5_8$, $110_2 = 6_8$, $111_2 = 7_8$.

[2] Prufer, Ernst Paul Heinz.

[3] Теорема Кэли (Cayley A.).

Прюфера рассмотрены в фундаментальном труде В.Н. Касьянова и В.А. Евстигнеева [15].

В частности, было показано, как образуется код Прюфера дерева, полученного склейкой (отождествлением вершин) двух других деревьев. Отметим, что кодировка Прюфера применяется к свободным деревьям (неориентированным деревьям, т.е. деревьям, в которых нет выделенного корня).

Приведем алгоритм кодировки помеченного дерева по Прюферу.

1. Найти висячую вершину с минимальным номером i.
2. Записать в код Прюфера вершину, смежную с i.
3. Удалить вершину i из дерева. Если дерево не пустое, то перейти к п.1.

Задача. Записать код Прюфера [15] дерева (рис. 3.10).

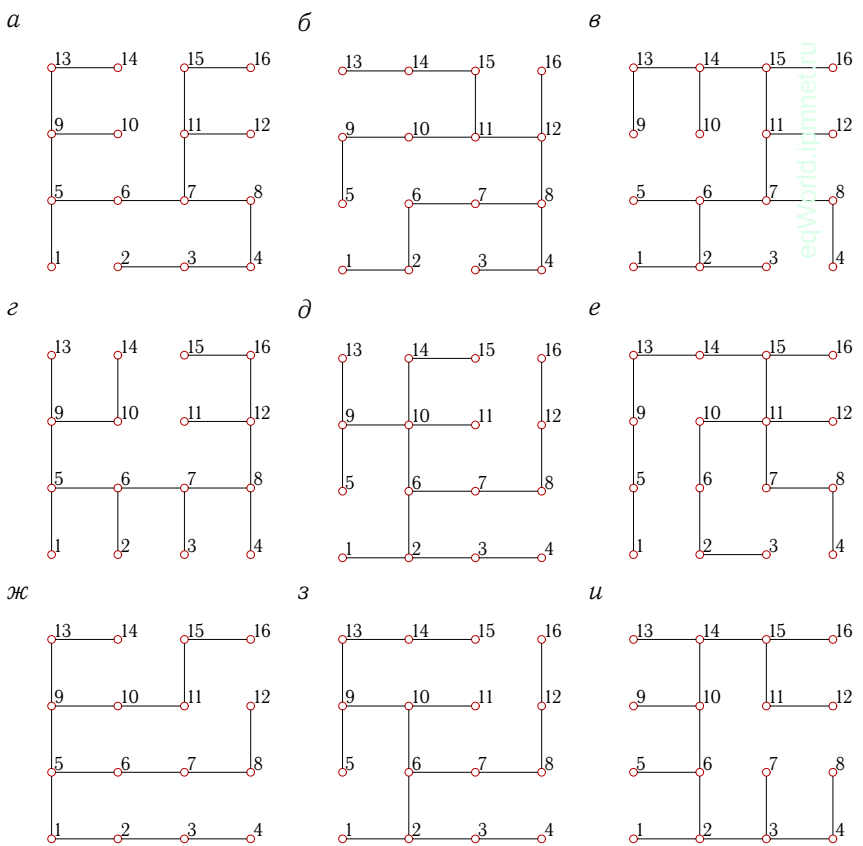


Рис. 3.10

Ответы

	Код Прюфера
а	[5, 3, 4, 8, 7, 9, 11, 13, 9, 5, 6, 7, 11, 15]
б	[2, 6, 4, 8, 9, 7, 8, 12, 10, 11, 14, 15, 11, 12]
в	[2, 2, 6, 8, 6, 7, 7, 11, 13, 14, 11, 15, 14, 15]
г	[5, 6, 7, 8, 12, 9, 10, 9, 5, 6, 7, 8, 12, 16]
д	[2, 3, 2, 6, 9, 10, 9, 10, 14, 10, 6, 7, 8, 12]
е	[5, 2, 6, 8, 9, 10, 7, 11, 13, 11, 11, 15, 14, 15]
ж	[3, 2, 1, 5, 8, 7, 6, 5, 9, 13, 9, 10, 11, 15]
з	[2, 3, 2, 6, 9, 10, 14, 13, 9, 10, 6, 7, 8, 12]
и	[2, 6, 3, 4, 3, 2, 6, 10, 10, 14, 11, 15, 14, 15]

Пример. Записать код Прюфера [15] для дерева (рис. 3.11).

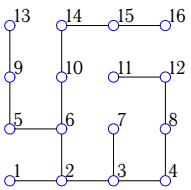
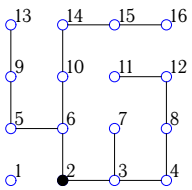
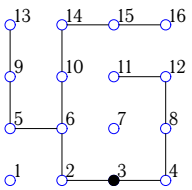


Рис. 3.11

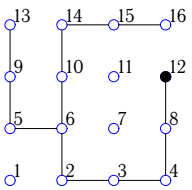
Решение. Находим висячую вершину с минимальным номером, записываем в код Прюфера вершину, смежную с ней, и удаляем ее из дерева. Последовательность определения кода Прюфера для рассматриваемого дерева показана на рисунках 3.12–3.17. Для наглядности изображение удаленной вершины остается на рисунке, а стирается только ребро.



$P = [2]$

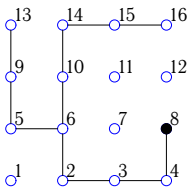


$P = [2, 3]$

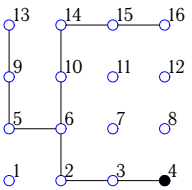


$P = [2, 3, 12]$

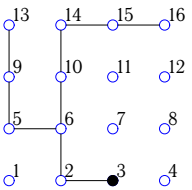
Рис. 3.12



$P = [2, 3, 12, 8]$

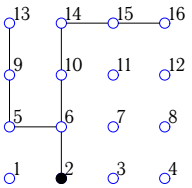


$P = [2, 3, 12, 8, 4]$

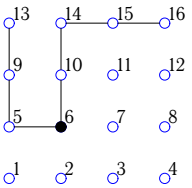


$P = [2, 3, 12, 8, 4, 3]$

Рис. 3.13

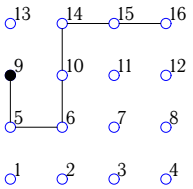


$P = [2, 3, 12, 8, 4, 3, 2]$

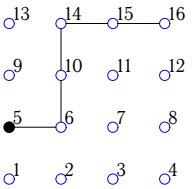


$P = [2, 3, 12, 8, 4, 3, 2, 6]$

Рис. 3.14

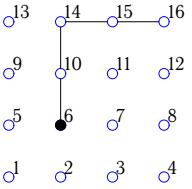


$P = [2, 3, 12, 8,4, 3, 2, 6, 9]$

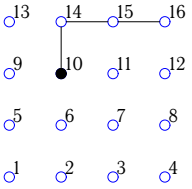


$P = [2, 3, 12, 8, 4, 3, 2, 6, 9, 5]$

Рис. 3.15

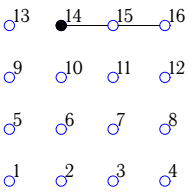


$P = [2, 3, 12, 8, 4, 3, 2, 6, 9, 5, 6]$

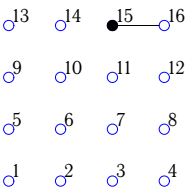


$P = [2, 3, 12, 8, 4, 3, 2, 6, 9, 5, 6, 10]$

Рис. 3.16



$P = [2, 3, 12, 8, 4, 3, 2, 6, 9, 5, 6, 10, 14]$



$P = [2, 3, 12, 8, 4, 3, 2, 6, 9, 5, 6, 10, 14, 15]$

Рис. 3.17

В результате код Прюфера имеет вид

$$P = [2, 3, 12, 8, 4, 3, 2, 6, 9, 5, 6, 10, 14, 15].$$

Вершины в коде Прюфера могут повторяться, более того, в коде Прюфера может быть только одна вершина. Так, если номер центральной вершины звезды на рис. 3.5 равен 5, то код состоит из трех одинаковых цифр — 555.

Две Maple-программы кодировки Прюфера приведены на с. 117.

3.4. Распаковка кода Прюфера

Распаковка кода Прюфера производится по алгоритму, описанному в [15]. Основное требование к алгоритмам кодирования — однозначность восстановления информации.

Задача. По заданному коду Прюфера построить дерево.

а)
[6, 3, 4, 5, 10, 7, 8, 9, 10, 15, 12, 13, 14, 15, 20, 16, 17, 18, 19, 20, 25, 23, 24];

б)
[2, 3, 4, 3, 8, 6, 11, 9, 8, 13, 16, 17, 18, 19, 20, 21, 22, 17, 18, 19, 20, 25, 24];

в)
[2, 7, 4, 4, 9, 11, 8, 9, 9, 14, 12, 17, 14, 21, 18, 22, 23, 18, 19, 14, 15, 20, 25];

г)
[5, 4, 3, 2, 6, 9, 8, 7, 6, 11, 21, 16, 18, 19, 18, 17, 16, 11, 12, 13, 14, 15, 20];

д)
[2, 3, 4, 9, 10, 13, 18, 13, 8, 16, 11, 6, 7, 17, 12, 7, 8, 9, 10, 15, 20, 25, 24];

е)
[6, 2, 7, 9, 10, 11, 7, 12, 14, 15, 12, 17, 17, 22, 18, 15, 14, 13, 18, 23, 22, 23, 24];

ж)
[2, 3, 8, 11, 12, 5, 4, 9, 8, 13, 16, 17, 13, 18, 20, 21, 22, 23, 24, 25, 22, 23, 24];

з)
[6, 7, 8, 4, 9, 9, 11, 14, 15, 14, 16, 17, 18, 17, 16, 11, 6, 7, 8, 9, 14, 19, 24];

и)
[6, 7, 3, 8, 10, 11, 12, 8, 13, 15, 16, 13, 18, 19, 20, 21, 18, 19, 22, 23, 24, 19, 20].

Ответы

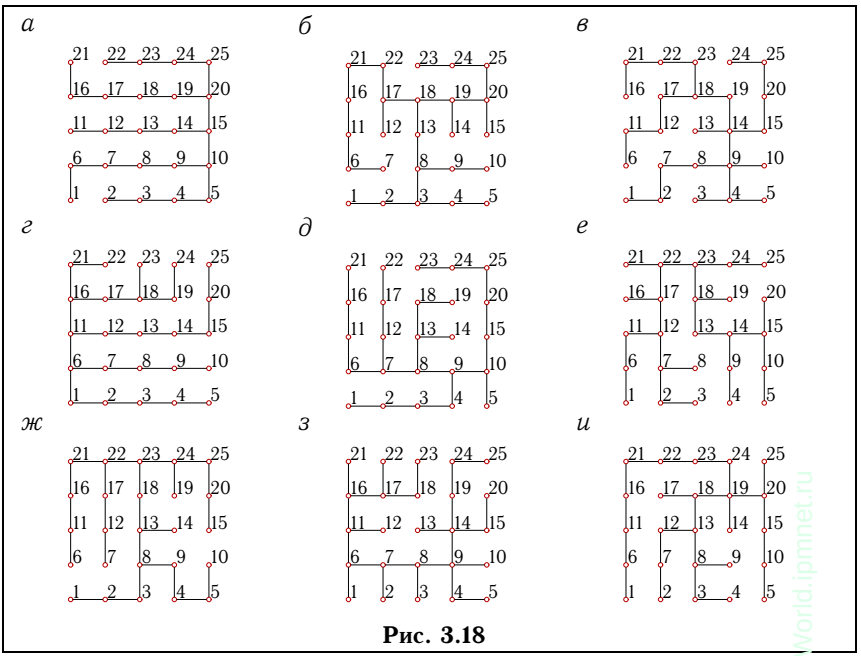


Рис. 3.18

Пример. Построить дерево, соответствующее коду Прюфера $P = [5, 6, 7, 8, 6, 10, 14, 15, 11, 10, 6, 7, 8, 12]$.

Решение. *Алгоритм распаковки кода Прюфера P.*

Введем список (вектор) некоторых элементов $N = [a_1, ..., a_n]$ и операцию укорочения списка на один первый элемент, обозначив ее звездочкой: $N^* = [a_2, a_3, ..., a_n]$. Тогда

1) $A = P$, $N = [1, ..., n]$;
2) $v = \min N$, $v \notin A$, $u = a_1$;
3) вершины u и v соединить ребром;
4) $N = N \setminus v$, $A = A^*$;
5) если $|N| = 2$, то соединить две последние вершины, n_1 и n_2, и завершить процедуру; в противном случае вернуться к п. 2.

Для кода, данного в условии задачи, последовательно получаем

- $A = [5, 6, 7, 8, 6, 10, 14, 15, 11, 10, 6, 7, 8, 12]$,
  $B = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$,
  ребро $(5,\ 1)$;
- $A = [6, 7, 8, 6, 10, 14, 15, 11, 10, 6, 7, 8, 12]$,
  $N = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$,

ребро (6, 2);

- $A = [7, 8, 6, 10, 14, 15, 11, 10, 6, 7, 8, 12]$,
  $N = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$,
  ребро (7, 3);
- $A = [8, 6, 10, 14, 15, 11, 10, 6, 7, 8, 12]$,
  $N = [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$,
  ребро (8, 4);
- $A = [6, 10, 14, 15, 11, 10, 6, 7, 8, 12]$,
  $N = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$,
  ребро (6, 5);
- $A = [10, 14, 15, 11, 10, 6, 7, 8, 12]$,
  $N = [6, 7, 8, \ 9, 10, 11, 12, 13, 14, 15, 16]$.
  Вершины 6, 7, 8 есть в A, поэтому вершину 10 (первая из списка A) соединяем с 9; получаем ребро (10, 9). Укорачиваем A и удаляем вершину 9 из N. Курсивом выделены вершины списка N, присутствующие в A. Далее
- $A = [14, 15, 11, 10, 6, 7, 8, 12]$,
  $N = [6, 7, 8, 10, 11, 12, \ 13, 14, 15, 16]$.
  Вершины 6, 7, 8, 10, 11, 12 из N есть в A, поэтому вершину 14 (первая из A) соединяем с вершиной 13, следующей за списком 6, 7, 8, 10, 11, 12, и получаем ребро (14, 13). Укорачиваем спереди список A, отрезая от него 14, и удаляем вершину 13 из N. Далее
- $A = [15, 11, 10, 6, 7, 8, 12]$,
  $N = [6, 7, 8, 10, 11, 12, \ 14, 15, 16]$. Аналогично получаем ребро (15, 14). Вершину 15 берем из A, вершину 14 — из N.
- $A = [11, 10, 6, 7, 8, 12]$,
  $N = [6, 7, 8, 10, 11, 12, \ 15, 16]$,
  ребро (11, 15);
- $A = [10, 6, 7, 8, 12]$,
  $N = [6, 7, 8, 10, \ 11, 12, 16]$,
  ребро (10, 11);
- $A = [6, 7, 8, 12]$,
  $N = [6, 7, 8, \ 10, 12, 16]$,
  ребро (6, 10);

- $A = [7, 8, 12]$,
  $N = [6, 7, 8, 12, 16]$,
  ребро $(7,\ 6)$;
- $A = [8, 12]$,
  $N = [7, 8, 12, 16]$,
  ребро $(8,\ 7)$;
- $A = [12]$,
  $N = [8, 12, 16]$,
  ребро $(12,\ 8)$.
- На последнем этапе получаем ребро, образованное двумя вершинами из N:
  $A = [\ ]$, $N = [12, 16]$, ребро $(12,\ 16)$.

Дерево, закодированное по Прюферу, — свободное, т.е. оно не имеет корня. Оно может быть изображено, например, в виде, показанном на рис. 3.19 или на рис. 3.20. В последнем случае в дереве искусственно выделен корень 6. Полученное дерево имеет 6 ярусов [1].

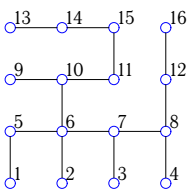


Рис. 3.19

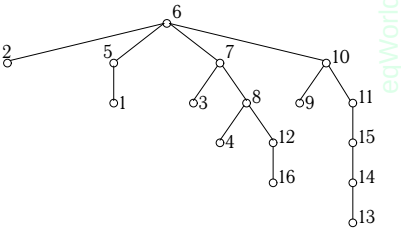


Рис. 3.20

Описанный процесс легко программируется. Соответствующая Maple-программа приведена на с. 119.

3.5. Кодировка Гапта

Для деревьев типа 2–3, т.е. деревьев, каждая не концевая вершина которых имеет по 2 или 3 сына, применяется код Гапта [15]. Без какого-либо изменения алгоритма этот код обобщается и на более сложные случаи. Дерево не обязательно должно быть помечено. Кодировка Гапта (в отличие от кодировки Прюфера) не сохраняет информацию об именах вершин.

[1] Какую вершину свободного дерева надо принять за корень, чтобы высота дерева была минимальной?

Задача. Найти код Гапта дерева (рис. 3.21).

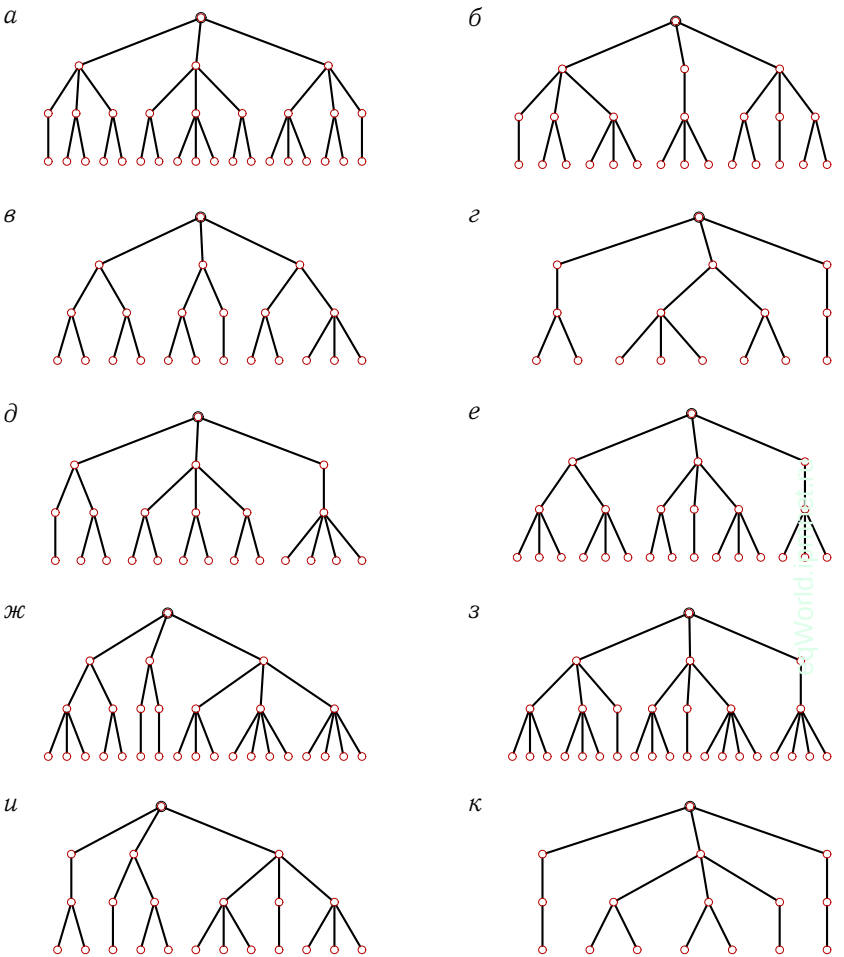


Рис. 3.21

Ответы

	Код		Код
а	[1, 2, 2, 2, 3, 2, 3, 2, 1, 3, 3, 3, 3]	е	[3, 3, 2, 1, 3, 3, 2, 3, 1, 3]
б	[1, 2, 3, 3, 2, 1, 2, 3, 1, 3, 3]	ж	[3, 2, 1, 1, 3, 4, 4, 2, 2, 3, 3]
в	[2, 2, 2, 1, 2, 3, 2, 2, 2, 3]	з	[3, 3, 1, 3, 1, 4, 4, 3, 3, 1, 3]
г	[2, 3, 2, 1, 1, 2, 1, 3]	и	[2, 1, 2, 3, 1, 3, 1, 2, 3, 3]
д	[1, 2, 2, 2, 2, 4, 2, 3, 1, 3]	к	[1, 2, 2, 1, 1, 1, 3, 1, 3]

Пример. Найти код Гапта дерева (рис. 3.22).

Решение. Выберем направление обхода дерева. Пусть код состоит из числа сыновей каждой вершины дерева при обходе дерева *слева направо, снизу вверх*. Висячие вершины (их степень равна 1) не имеют сыновей, поэтому в код дерева порядка n, имеющего n_0 висячих вершин, войдет $n - n_0$ чисел. Для дерева на рис. 3.22 кодировка должна содержать $12-8=4$ числа. Начнем кодировку с самой верхней вершины — A. Она имеет три сына — B, C, D. Следовательно, заносим в код число 3:

$$[--\,-3].$$

Переходим на следующий ярус. Самая правая вершина, B, имеет три сына. Заносим в код число 3:

$$[--\,3,3].$$

Продолжая далее, окончательно получаем код [2, 3, 3, 3].

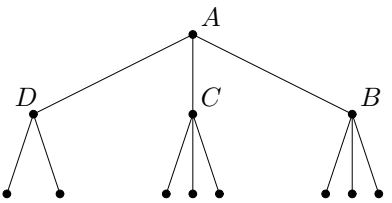


Рис. 3.22

Maple-программа для кодировки Гапта приведена на с. 120.

3.6. Распаковка кода Гапта

Задача. Распаковать код Гапта и построить дерево:

а)
[3, 1, 1, 3, 2, 2, 4, 3, 1, 3, 3];

б)
[2, 2, 2, 3, 2, 4, 4, 2, 3, 3, 2, 3];

в)
[1, 3, 2, 3, 3, 1, 2, 3, 3, 1, 3];

г)
[1, 1, 1, 1, 1, 3, 3, 1, 2, 3];

д)
[2, 3, 2, 2, 3, 3, 4, 2, 2, 3, 3];

е)
[3, 3, 2,1, 1, 2, 2, 1, 3, 3, 2, 3];

ж)
[3, 3, 2, 2, 2, 2, 2, 2, 2, 3];

з)
[1, 1, 2, 2, 3, 1, 2, 2, 3];

и)
[2, 3, 3, 3, 1, 2, 1, 3];

к)
[3, 3, 1, 2, 1, 1, 2, 3, 2, 2, 3].

Ответы

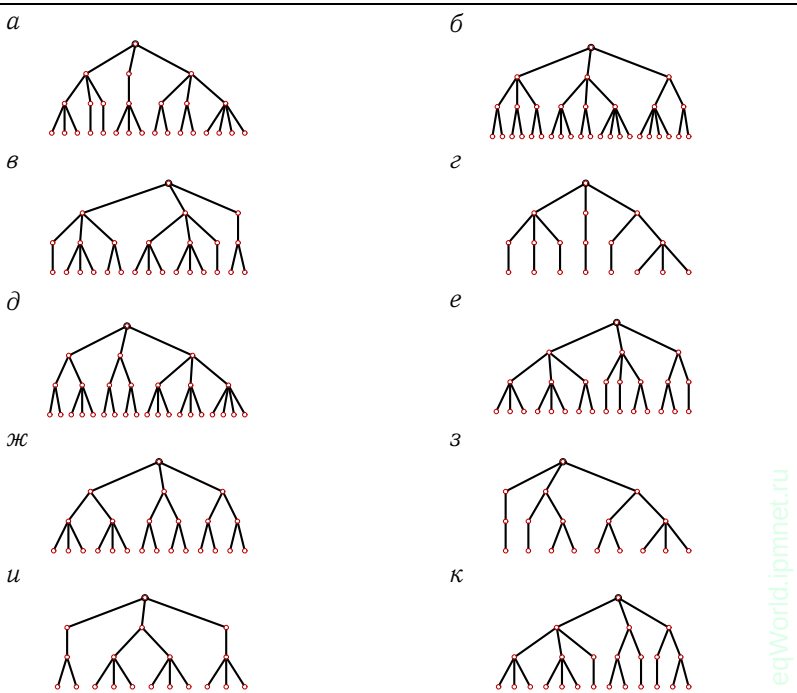


Рис. 3.23

Пример. По заданному коду Гапта $[3,1,1,1,1,4,2,1,2,3,3,3]$, построить дерево.

Решение. Построение начинается с корня. От корня, согласно последнему числу кода, идет три дуги к трем вершинам-сыновьям 2-го яруса. Рассматривая следующие с конца три числа кода, выясняем, что от этих вершин идет 3, 3 и 2 дуги соответственно. Изображаем эту часть дерева и продолжаем строить следующий ярус. В результате получаем рис. 3.24.

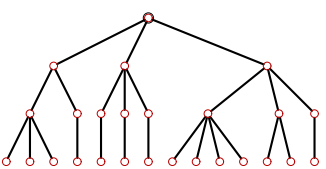


Рис. 3.24

Maple-программа для распаковки кода Гапта приведена на с. 122.

Глава 4

АЛГОРИТМЫ

В этой главе собраны задачи, для решения которых используются либо классические, либо новые алгоритмы, имеющие универсальное применение. Совершенно очевидно, что решение многих задач не является самоцелью. Как правило, это полигон для отладки известных и построения новых методов решения. Большое число алгоритмов на графах с подробным описанием и анализом содержится в [18]. В [11] алгоритмы на графах реализованы в среде Delphi на языке Pascal. Некоторые сведения по теории алгоритмов содержатся в [21].

4.1. Кратчайший путь в орграфе

Дугам во взвешенном орграфе поставлено в соответствие некоторое число (вес). Это может быть евклидово расстояние между вершинами или какая-либо другая числовая характеристика (цена, время, энергия). Ставится задача нахождения кратчайшего пути во взвешенном орграфе от одной заданной вершины до другой. Длина пути — сумма весов дуг пути. Вес дуг может быть положительным, отрицательным, нулевым или бесконечным (что соответствует отсутствию дуги). Здесь мы будем рассматривать положительные веса. Кроме того, предполагается, что решение есть, т.е. указанные вершины соединены путем. Существуют также аналогичные задачи для графов с двойными весами [18].

Задача. Для заданного орграфа (рис. 4.1) найти кратчайший путь от вершины 1 к вершине 12. На рисунке рядом с дугой указан ее вес.

а *б*

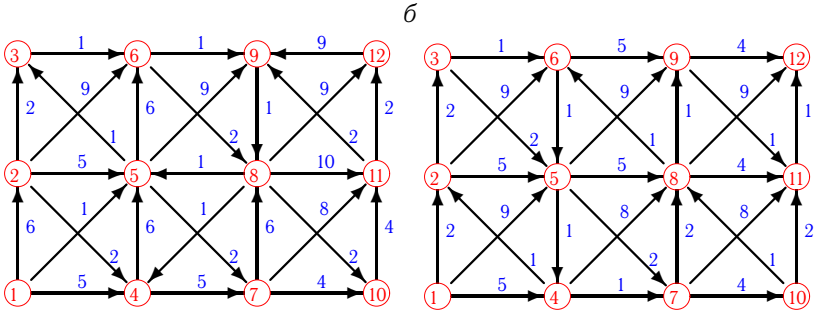
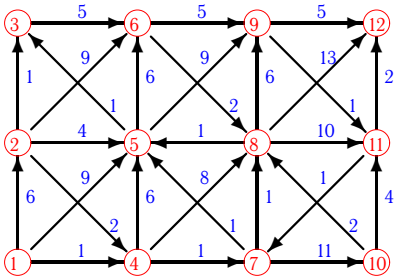
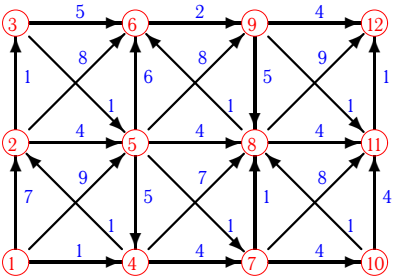


Рис. 4.1

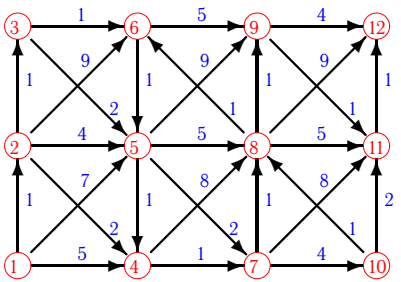
в



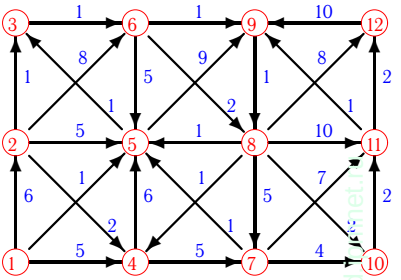
г



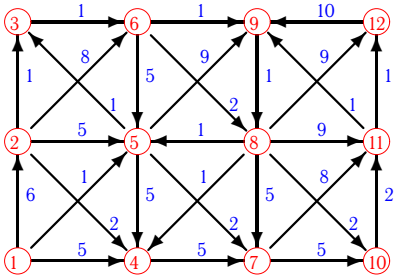
д



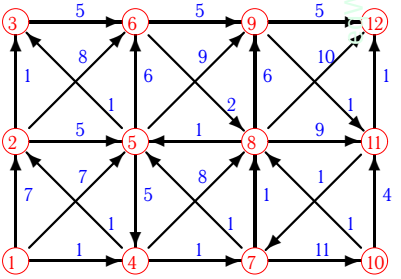
е



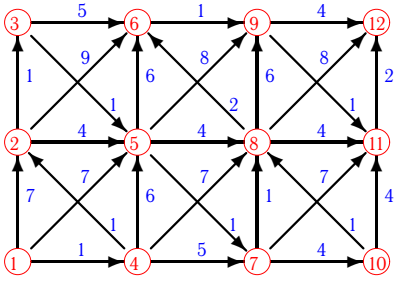
ж



з



и



к

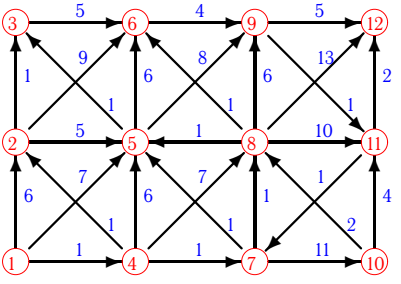


Рис. 4.1 (продолжение)

Ответы

	Путь	Длина		Путь	Длина
а	12 11 7 5 1	13	е	12 11 10 8 6 3 5 1	11
б	12 11 9 8 7 4 1	11	ж	12 11 10 8 6 3 5 1	10
в	12 11 9 8 7 4 1	12	з	12 11 9 8 7 4 1	11
г	12 11 8 7 4 1	11	и	12 11 8 7 5 3 2 4 1	12
д	12 11 9 8 7 4 2 1	8	к	12 11 9 6 8 7 4 1	11

Пример. Дан взвешенный орграф (рис. 4.2). Найти кратчайший путь из вершины A в B.

Решение. Применим алгоритм Е. Дейкстры [1] [10]. Пошаговый алгоритм определения кратчайшего расстояния от вершины A до B состоит в следующем. С каждой вершиной связывается метка. Метка может быть постоянной или временной. Первоначально вершине A приписывается постоянная метка 0, а всем остальным метка ∞. На первом шаге вычисляются расстояния от вершины A с постоянной меткой до всех остальных вершин. Если некоторая вершина не соединена с вершиной с постоянной меткой или дуга направлена в обратную сторону, то расстояние принимается бесконечным. Найденные расстояния являются временными метками вершин. Минимальная из временных меток берется за постоянную. На следующем шаге временные метки всех вершин (кроме тех, у которых постоянные метки) вычисляются как сумма значения последней полученной постоянной метки и расстояния от нее в случае, если это значение не больше предыдущего значения временной метки данной вершины. Таким образом, временная метка вершины может или оставаться прежней, или уменьшаться. Минимальная из временных меток всех вершин опять принимается за постоянную. Процесс продолжается до тех пор, пока вершина B не получит постоянную метку. Значение этой метки и есть кратчайшее расстояние от A до B.

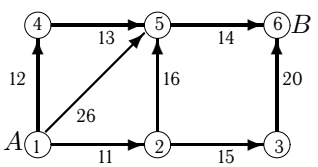


Рис. 4.2

Рассмотрим отдельные шаги решения.

1. Вершина A получает постоянную метку 0, остальные — метку ∞:

1	2	3	4	5	6
0	∞	∞	∞	∞	∞

.

[1] Dijkstra E.W.

2. Вычисляем расстояния от вершины 1 с постоянной меткой 0. Вершины 2, 4 и 5 меняют свои временные метки на 11, 12 и 26. Остальные имеют прежние метки — ∞. Очевидно, наименьшей меткой является 11. Она и становится постоянной:

1	2	3	4	5	6
[0]			∞	∞	∞
	[11]	∞	12	26	∞

3. Вычисляем расстояния от вершины 2 с постоянной меткой 11. Вершины 3 и 5 имеют расстояния 15 и 16 до вершины 2. Суммируя, получаем значения 26 и 27. Для вершины 5 прежнее значение, 26, было меньше нового значения 27. Следовательно, значение метки 5 не меняем; оно остается равным 26. Из трех временных меток — 12, 26 и 26 — наименьшая принадлежит вершине 4. Эта метка и становится постоянной:

1	2	3	4	5	6
[0]		∞		∞	∞
	[11]	∞	[12]	26	∞
		26			∞

4. Вычисляем расстояния от вершины 4 с постоянной меткой 12. Вершина 5 имеет до нее расстояние 13. Суммируя $(13+12)$, получаем значение 25 временной метки вершины 5 вместо прежнего значения 26. Из двух временных меток вершин 3 и 5 наименьшая принадлежит вершине 5. Эта метка и становится постоянной:

1	2	3	4	5	6
[0]		∞			∞
	[11]	∞	[12]		∞
		26			∞
		26		[25]	∞

5. На следующем этапе, вычисляя расстояния от вершины 5 с постоянной меткой 25, приходим к конечной вершине B. Но ее метка $(25+14=39)$ не становится постоянной, так как она не является минимальной. Расстояние от вершины 5 до вершины 3 принято ∞ (они не соединены). Прежнее значение временной метки вершины 3 меньше ∞. Поэтому метка вершины 3 не меняется. Метка

вершины 3 со значением 26, меньшим 39, становится постоянной. На следующем этапе ищем расстояния от нее:

1	2	3	4	5	6
0					∞
	11		12		∞
					∞
				25	∞
		26			39

6. Расстояние от вершины 3 до вершины 6 составляет 20. Так как $26 + 20 > 39$, значение метки 6 не меняем. На этом шаге она остается прежней и единственной временной меткой. Временная метка вершины 6 становится постоянной, что означает конец процесса. Минимальное расстояние от A до B равно 39.

Две Maple-программы для определения кратчайшего пути в орграфе приведены на с. 109 и 111.

4.2. Поток в сети

Сетью называют взвешенный орграф с двумя выделенными вершинами: истоком и стоком. Исток имеет нулевую полустепень захода, а сток — нулевую полустепень исхода. Вес дуги означает ее пропускную способность. Поток — еще одно число, приписанное дуге. Поток дуги не больше ее пропускной способности и может меняться. Поток выходит из истока и без потерь, в том же объеме заходит в сток. Условие равновесия (по объему входа и выхода) выполняется и для каждой вершины сети.

Задача о наибольшем потоке в сети — не единственная, но, вероятно, основная задача для потоков в сети. Очевидна возможность практического применения этой задачи для решения транспортных проблем (пробки на дорогах можно условно связывать с насыщением сети или отдельной ее дуги), проблем транспортировки нефтепродуктов или электроэнергии.

Задача. Задана пропускная способность дуг транспортной сети (рис. 4.3) с началом (истоком) в вершине 1 и концом (стоком) в вершине 14. Используя алгоритм *Форда–Фалкерсона* [13], найти максимальный поток по сети.

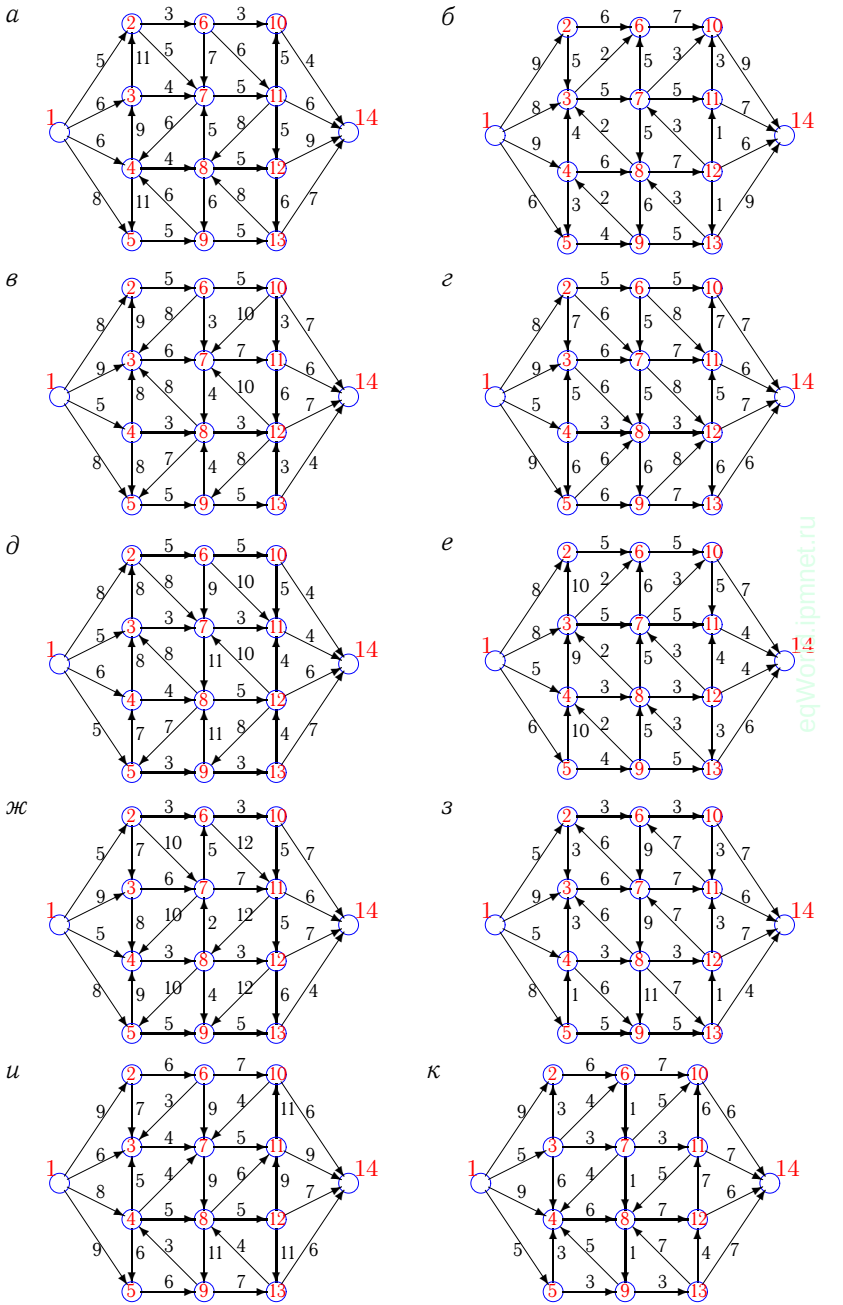


Рис. 4.3

Ответы

	а	б	в	г	д	е	ж	з	и	к
Σ	17	22	19	26	16	17	18	17	24	19

Пример. Задана пропускная способность дуг транспортной сети (рис. 4.4) с началом в вершине 1 и концом в вершине 8. Используя алгоритм *Форда–Фалкерсона*, найти максимальный поток по сети.

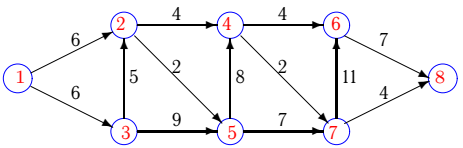


Рис. 4.4

Решение. Алгоритм состоит из двух частей — насыщения потока и его перераспределения. Поток называется насыщенным, если любая цепь из истока в сток содержит насыщенную дугу. В первой части алгоритма разыскиваются цепи из истока в сток и все дуги цепи насыщаются одинаковым возможно большим потоком, определяемым пропускной способностью наиболее «тонкой» дуги или наименьшей разностью между пропускной способностью и потоком в дуге. Различные цепи могут иметь общие дуги. Полученный поток согласован с условием сохранения в узлах (вершинах). Поток, входящий в вершину, равен потоку, выходящему из нее. Поток в сети проходит по *цепям* из истока в сток, т.е. недопустим многократный проход по отдельной дуге. Первая часть задачи считается решенной, если нет ненасыщенных цепей из истока в сток. Первая часть задачи не имеет единственного решения.

Во второй части перераспределение потока выполняется исходя из условия достижения общего по сети максимума потока. Для этого в основании графа (т.е. в графе, в котором снята ориентация дуг) разыскиваются маршруты из истока в сток, состоящие из ребер, соответствующих ненасыщенным дугам, направленным вперед, и непустым дугам, направленным назад. Потоки в дугах прямого направления увеличиваются на величину, на которую уменьшаются потоки в обратных дугах выбранного маршрута. При этом, очевидно, нельзя превышать пропускную способность дуг, направленных вперед, и допускать отрицательных потоков в обратных дугах. В некоторых случаях при удачном выборе цепей в первой части алгоритма перераспределения потока не требуется.

1. **Насыщение потока**. Рассмотрим путь 1–2–4–6–8. Пропустим через этот путь поток, равный 4. При этом дуги [2, 4] и [4, 6] будут насыщенными. Аналогично, путь 1–3–5–7–8 насытим потоком 4. Распределение потока отметим на графе (рис. 4.5). В числителе ставим пропускную способность, в знаменателе — поток. Числитель всегда больше знаменателя, а знаменатель может быть и нулем.

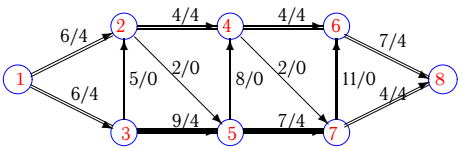


Рис. 4.5

Заметим, что из 1 в 8 есть еще ненасыщенный путь, 1–3–2–5–4–7–6–8,поток в котором можно увеличить на 2. При этом насытятся дуги [1, 3], [2, 5] и [4, 7] (рис. 4.6).

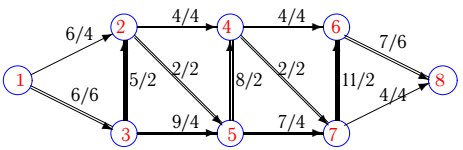


Рис. 4.6

Из 1 в 8 больше нет ненасыщенных путей. По дуге [1,3] двигаться нельзя (она уже насыщена), а движение по дуге [1,2] заканчивается в вершине 2, так как обе выходящие из нее дуги насыщены.

2. **Перераспределение потока.** Найдем последовательность вершин от 1 к 8, такую, что дуги, соединяющие соседние вершины, направленные из 1 в 8, не насыщены, а дуги, направленные в обратном направлении, не пусты. Имеем единственную последовательность: 1–2–3–5–7–6–8. Перераспределяем поток. Поток в дугах прямого направления увеличиваем на 1, а поток в дугах обратного направления уменьшаем на 1. Процесс продолжаем до тех пор, пока одна из прямых дуг не будет насыщена или какая-нибудь обратная дуга не будет пуста.

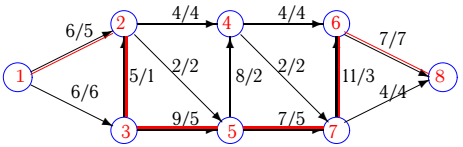


Рис. 4.7

Поток в насыщенной сети можно посчитать по потоку, выходящему из истока 1 или входящему в сток 8. Очевидно, эти числа должны быть равны. Кроме того, для проверки решения следует проверить условие сохранения потока по узлам. Для каждого узла суммарный входящий поток должен быть равен выходящему. В рассматриваемом примере поток равен 11. Распределение потока по дугам при одном и том же суммарном минимальном потоке в сети не единственное.

Maple-программа для определения максимального потока в сети приведена на с. 123.

4.3. Топологическая сортировка сети

Пусть в сети в общем случае имеется несколько истоков и стоков. Стоки и истоки занимают в сети крайние положения. Все остальные вершины могут быть дальше или ближе к ним. Расположение вершины в сети определяется ее *уровнем*. Определим это понятие. Вершины уровня 0 — это истоки; они образуют множество N_0. Если N_i — множество вершин уровня $i \leqslant k$, то N_{k+1} — множество вершин уровня $k+1$ состоящее из тех и только тех вершин, предшественники которых принадлежат любому из множеств $N_0,...,N_k$, причем среди этих множеств нет пустых [3].

Порядковой функцией сети называют отображение, сопоставляющее каждой вершине сети ее уровень.

Задача. Найти порядковую функцию сети (рис. 4.8).

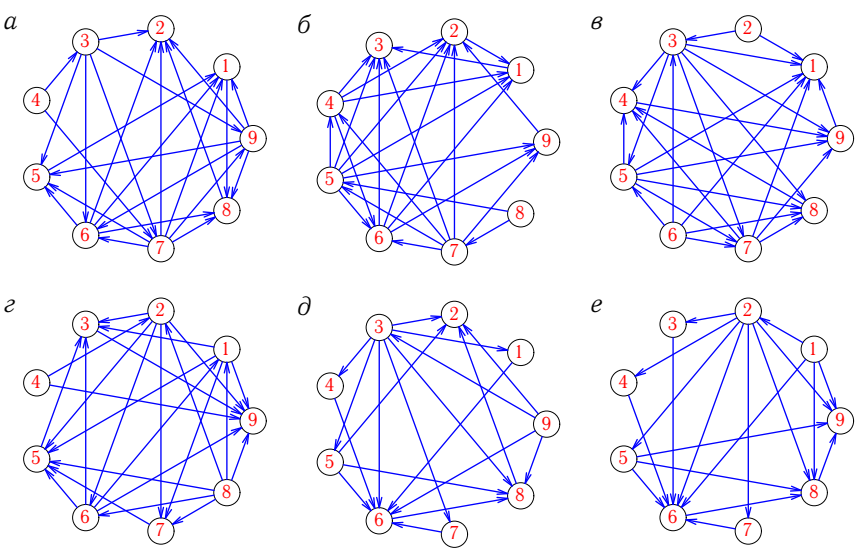


Рис. 4.8

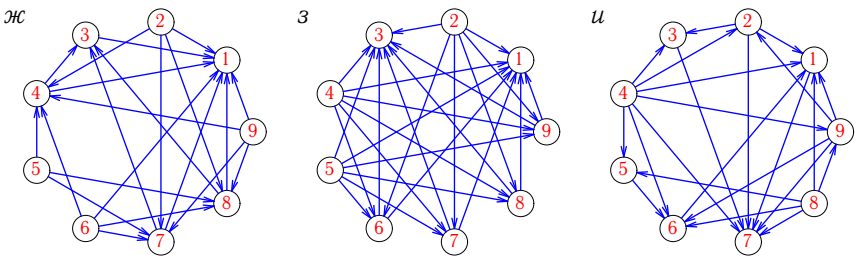


Рис. 4.8 (продолжение)

Ответы

№	Порядковая функция
а	4, 3, 7, 9, 6, 5, 1, 8, 2
б	8, 7, 5, 4, 6, 9, 2, 1, 3
в	(2, 6), 3, 5, 7, 8, 4, 9, 1
г	(4, 8), 2, 6, 1, 7, 5, 3, 9
д	9, 3, (1, 7, 4, 5), 6, 8, 2
е	1, 2, (7, 3, 4, 5), 6, 8, 9
ж	(5, 6, 2, 9), (4, 7, 8), 3, 1
з	(2, 5, 4), (8, 6, 7, 9), (3, 1)
и	(4, 8), (5, 9), (6, 2), (1, 3), 7

Пример. Отсортировать топологически сеть на рис. 4.9.

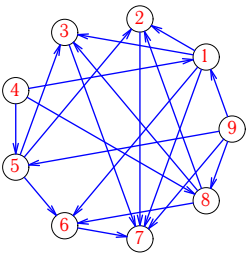


Рис. 4.9

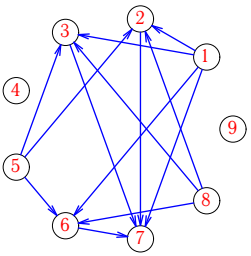


Рис. 4.10

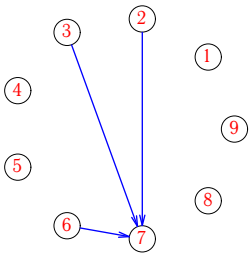


Рис. 4.11

Решение. Находим вершины (4 и 9), полустепень захода которых равна 0. Удаляем эти вершины и дуги, выходящие из них [3]. Удаленные вершины образуют первый уровень упорядоченной сети. Получаем новую сеть (рис. 4.10). В новой сети в вершины 1, 5 и 8 не входят дуги. Удаляем

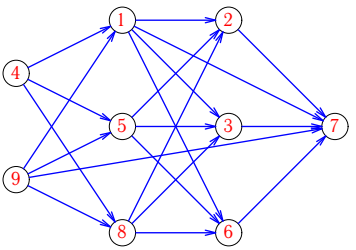


Рис. 4.12

эти вершины (они образуют второй уровень) и получаем сеть, изображенную на рис. 4.11. Очевидно, третий уровень состоит из вершин 2, 3 и 6, а последний (сток сети) — из единственной вершины 7. Изображаем ту же сеть по уровням (рис. 4.12).

Maple-программа для топологической сортировки сети приведена на с. 127.

4.4. Паросочетание в двудольном графе

Граф называется *двудольным*, если существует такое разбиение множества его вершин на две части, что концы каждого ребра принадлежат разным частям (долям).

Теорема 16 (Кенига [1]). *Для двудольности графа необходимо и достаточно, чтобы он не содержал циклов нечетной длины.*

Если любые две вершины двудольного графа, входящие в разные доли, смежны, то граф называется *полным двудольным*. Обозначение для полного двудольного графа с n и m вершинами в долях — $K_{n,m}$.

Так как по определению в двудольном графе вершины одной доли никогда не соединяются ребрами, для избежания ввода ненужной информации матрицу смежности для двудольного графа можно записать в сокращенной форме: элемент a_{ij} матрицы смежности двудольного графа равен 1, если i-я вершина одной доли соединена с j-й вершиной другой доли; в противном случае $a_{ij} = 0$. Если нумерация вершин графа общая, а не по долям, то j-ю вершину другой доли в определении надо заменить на $(j + n)$-ю вершину графа $K_{n,m}$. Очевидно, матрица двудольного графа в общем случае имеет размер $n \times m$, т.е. не является квадратной.

Паросочетанием графа называется граф, ребра которого являются подмножеством ребер графа, а вершины имеют степень 1.

Паросочетание, не являющееся подмножеством другого паросочетания, называется *максимальным*.

Паросочетание, содержащее наибольшее число ребер, называется *наибольшим*. Наибольшее паросочетание является и максимальным.

Паросочетание, порядок которого равен порядку графа, называется *совершенным*. Совершенное паросочетание включает в себя все вершины двудольного графа. Очевидно, совершенное паросочетание является наибольшим и максимальным. Максимальное паросочетание

[1] König D.

может быть и не наибольшим. В одном графе могут иметься различные максимальные паросочетания одного порядка.

Число совершенных паросочетаний в двудольном графе, имеющем одинаковое число вершин в долях, можно определить, вычислив перманент [1] его матрицы смежности.

Матрица смежности двудольного графа на рис. 4.13 имеет вид

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}.$$

Перманент этой матрицы равен 4. Все четыре покрытия графа изображены на рис. 4.14–4.17.

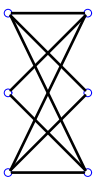


Рис. 4.13

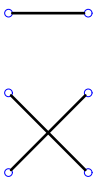


Рис. 4.14

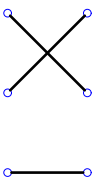


Рис. 4.15

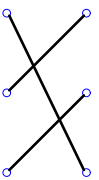


Рис. 4.16

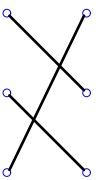


Рис. 4.17

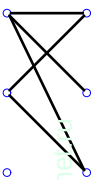


Рис. 4.18

Если граф не имеет совершенного паросочетания, то перманент его матрицы равен нулю. Например, матрица графа с одной изолированной вершиной на рис. 4.18 имеет вид

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{vmatrix}.$$

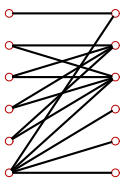
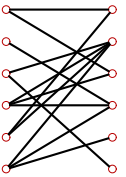
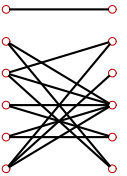
Перманент этой матрицы равен 0 [2].

[1] Перманент матрицы (per) равен сумме всех произведений элементов матрицы по одному из каждой строки в разных столбцах. Перманент [26] квадратной матрицы $a_{ij}(i,j=1,...,n)$ определяется, подобно определителю, рекуррентным образом. При $n=1$ имеем $\mathrm{per}(A_1)=a_{11}$. Для матрицы A_{n+1} размером $n+1$ получим разложение по первой строке: $\mathrm{per}(A_{n+1})=\sum_{k=1}^{n} a_{1k}\mathrm{per}(A_k)$, где $\mathrm{per}(A_k)$ — перманент матрицы размером n, полученной из A_{n+1} вычеркиванием первой строки и k-го столбца. Таким образом, перманент отличается от определителя только тем, что все его слагаемые берутся со знаком плюс. Например, $\mathrm{per}\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = 1\cdot 4 + 3\cdot 2$.

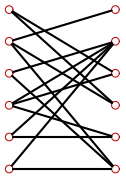
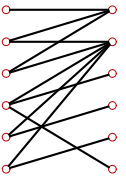
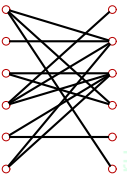
[2] **Теорема 17** [25]. *Перманент матрицы A, состоящей из 0 и 1, порядка n, равен нулю тогда и только тогда, когда в A существует подматрица из нулей размером $s\times t$, где $s+t=n+1$.*

Задача. В заданном двудольном графе (рис. 4.19) найти число совершенных паросочетаний P и одно из наибольших паросочетаний.

а *б* *в*

г *д* *е*

ж *з* *и*

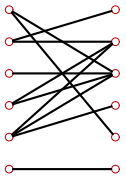
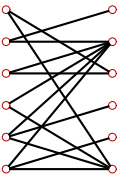
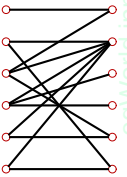
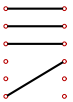




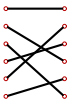
Рис. 4.19

Ответы

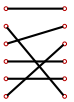
а. $P = 0$



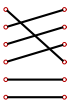
б. $P = 2$



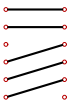
в. $P = 2$



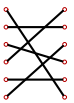
г. $P = 2$



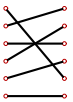
д. $P = 0$



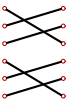
е. $P = 1$



ж. $P = 1$



з. $P = 2$



и. $P = 0$

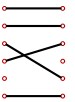


Рис. 4.20

Пример. В заданном двудольном графе (рис. 4.21) найти наибольшее паросочетание.

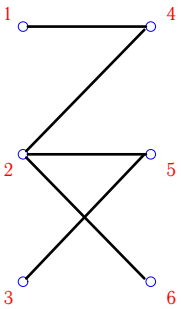


Рис. 4.21

Решение. Применим алгоритм Форда–Фалкерсона (см. с. 62). Образуем из двудольного графа сеть, добавляя исток 7 и сток 8 и ориентируя *ребра* из истока в сток. Пропускную способность всех полученных *дуг* положим равной 1 (рис. 4.22). Максимальный поток по этой сети соответствует искомому паросочетанию графа. Заметим, что решение будет не единственным.

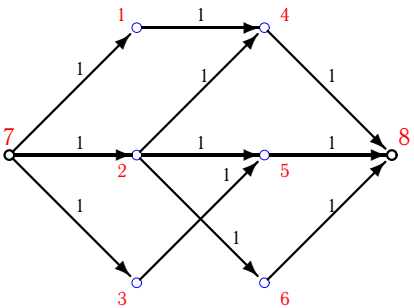


Рис. 4.22

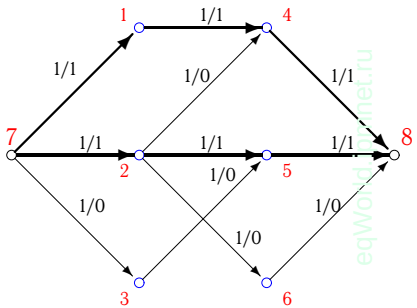


Рис. 4.23

Находим все возможные маршруты из истока в сток и насыщаем их. Таких маршрутов два: 7–1–4–8 и 7–2–5–8 (рис. 4.23). Маршрутов из 7 в 8, по которым можно было бы пропустить дополнительный поток, больше нет [1]. Перераспределяем полученный поток. Находим пути из истока в сток, содержащие ненасыщенные прямые дуги и непустые обратные. В данной задаче это путь 7–3–5–2–6–8. Уменьшаем поток в обратных дугах и увеличиваем на эту же величину поток в прямых дугах (рис. 4.24). В покрытие исходного двудольного графа войдут ребра с потоком, равным 1 (рис. 4.25).

[1] Иногда для фиксации завершения первого этапа алгоритма насыщенные дуги из графа убирают и к следующему этапу переходят, если сеть оказалась разорванной, т.е. исток оторван от стока.

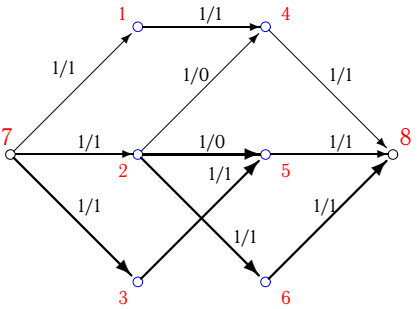


Рис. 4.24

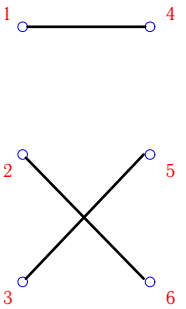


Рис. 4.25

Матрица смежности исходного двудольного графа имеет вид

$$\begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{vmatrix}.$$

Перманент этой матрицы равен 1. Следовательно, нами найдено единственное совершенное покрытие.

Maple-процедура, реализующая другой алгоритм нахождения наибольшего паросочетания двудольного графа, приведена на с. 131. Пример использования этой процедуры дан на с. 134.

4.5. Задача о назначениях

Известно, что совершенное паросочетание в двудольном графе может быть не единственным. Поэтому, если граф взвешенный, т.е. каждое ребро наделено весом, то естественно поставить задачу о поиске паросочетания с наибольшим или наименьшим весом. Очевидна возможность практического применения решения этой задачи. Рассмотрим, например, задачу о назначении группы n сотрудников по n рабочим местам. Каждый из сотрудников может работать на любом из этих мест, но оплата труда везде разная. Обозначим через a_{ij} оплату труда сотрудника i на рабочем месте j. Матрица коэффициентов a_{ij} в общем случае несимметрична. Оптимальное распределение сотрудников соответствует минимальным расходам на производство. Представляя сотрудников вершинами одной доли графа, а рабочие места — вершинами другой доли, получим полный двудольный взвешенный граф. Совершенное паросочетание наименьшего веса является решением задачи.

Задача. Оплата труда работника i на рабочем месте j определяется коэффициентом a_{ij} (рис. 4.26). Найти одно из оптимальных назначений и суммарные затраты Σ на производство.

а

$$A = \begin{vmatrix} 17 & 18 & 16 & 18 & 18 & 18 & 18 \\ 18 & 39 & 18 & 47 & 45 & 48 & 55 \\ 26 & 30 & 18 & 58 & 59 & 62 & 66 \\ 25 & 29 & 18 & 50 & 51 & 54 & 61 \\ 30 & 37 & 18 & 33 & 57 & 60 & 64 \\ 30 & 34 & 18 & 33 & 34 & 60 & 64 \\ 39 & 40 & 18 & 39 & 43 & 40 & 74 \end{vmatrix}$$

б

$$A = \begin{vmatrix} 21 & 21 & 37 & 29 & 30 & 33 & 45 \\ 16 & 16 & 48 & 43 & 38 & 44 & 56 \\ 19 & 16 & 42 & 46 & 44 & 50 & 59 \\ 25 & 16 & 29 & 50 & 48 & 54 & 66 \\ 24 & 16 & 34 & 29 & 42 & 48 & 57 \\ 16 & 15 & 16 & 16 & 16 & 15 & 16 \\ 36 & 16 & 46 & 38 & 36 & 36 & 75 \end{vmatrix}$$

в

$$A = \begin{vmatrix} 20 & 19 & 19 & 16 & 19 & 19 & 19 \\ 19 & 39 & 49 & 19 & 49 & 51 & 53 \\ 28 & 34 & 56 & 19 & 56 & 58 & 57 \\ 25 & 31 & 26 & 19 & 47 & 49 & 48 \\ 29 & 38 & 42 & 19 & 63 & 65 & 64 \\ 34 & 37 & 38 & 19 & 38 & 69 & 68 \\ 36 & 36 & 46 & 19 & 40 & 39 & 66 \end{vmatrix}$$

г

$$A = \begin{vmatrix} 36 & 36 & 47 & 37 & 44 & 41 & 58 \\ 28 & 47 & 43 & 42 & 49 & 49 & 60 \\ 23 & 23 & 36 & 35 & 39 & 42 & 56 \\ 17 & 23 & 23 & 11 & 23 & 23 & 23 \\ 23 & 35 & 34 & 27 & 48 & 48 & 62 \\ 23 & 29 & 25 & 24 & 25 & 42 & 56 \\ 23 & 42 & 47 & 37 & 41 & 35 & 80 \end{vmatrix}$$

д

$$A = \begin{vmatrix} 30 & 32 & 31 & 46 & 40 & 30 & 50 \\ 21 & 23 & 28 & 43 & 40 & 21 & 47 \\ 21 & 17 & 33 & 45 & 45 & 17 & 55 \\ 21 & 17 & 17 & 12 & 17 & 11 & 17 \\ 33 & 32 & 37 & 37 & 57 & 17 & 67 \\ 28 & 24 & 26 & 32 & 29 & 17 & 57 \\ 37 & 39 & 35 & 44 & 38 & 17 & 72 \end{vmatrix}$$

е

$$A = \begin{vmatrix} 24 & 36 & 32 & 24 & 30 & 39 & 43 \\ 24 & 53 & 58 & 24 & 47 & 56 & 63 \\ 24 & 33 & 42 & 24 & 40 & 46 & 56 \\ 28 & 40 & 39 & 24 & 52 & 58 & 65 \\ 25 & 34 & 33 & 24 & 39 & 45 & 52 \\ 24 & 24 & 24 & 19 & 24 & 14 & 24 \\ 32 & 44 & 34 & 24 & 29 & 35 & 63 \end{vmatrix}$$

ж

$$A = \begin{vmatrix} 15 & 15 & 23 & 35 & 36 & 39 & 34 \\ 16 & 15 & 41 & 41 & 45 & 48 & 46 \\ 19 & 15 & 44 & 47 & 51 & 51 & 46 \\ 15 & 13 & 15 & 18 & 15 & 15 & 15 \\ 30 & 15 & 38 & 38 & 63 & 63 & 61 \\ 30 & 15 & 32 & 38 & 36 & 63 & 61 \\ 28 & 15 & 30 & 39 & 34 & 37 & 54 \end{vmatrix}$$

з

$$A = \begin{vmatrix} 30 & 36 & 35 & 42 & 37 & 51 & 42 \\ 24 & 29 & 34 & 41 & 39 & 47 & 41 \\ 13 & 23 & 20 & 23 & 23 & 23 & 23 \\ 24 & 30 & 32 & 44 & 42 & 53 & 47 \\ 23 & 23 & 31 & 26 & 42 & 53 & 47 \\ 23 & 34 & 33 & 34 & 32 & 69 & 63 \\ 23 & 34 & 27 & 31 & 26 & 37 & 48 \end{vmatrix}$$

и

$$A = \begin{vmatrix} 12 & 26 & 26 & 26 & 19 & 23 & 23 \\ 28 & 56 & 49 & 48 & 28 & 54 & 57 \\ 27 & 38 & 59 & 52 & 27 & 52 & 58 \\ 26 & 31 & 30 & 44 & 26 & 50 & 56 \\ 26 & 34 & 36 & 26 & 26 & 43 & 49 \\ 27 & 32 & 31 & 27 & 23 & 45 & 51 \\ 33 & 35 & 43 & 30 & 23 & 27 & 57 \end{vmatrix}$$

к

$$A = \begin{vmatrix} 15 & 18 & 14 & 18 & 18 & 18 & 18 \\ 19 & 30 & 18 & 40 & 44 & 42 & 47 \\ 28 & 33 & 18 & 53 & 54 & 55 & 60 \\ 25 & 30 & 18 & 41 & 45 & 46 & 51 \\ 27 & 29 & 18 & 30 & 57 & 58 & 63 \\ 28 & 30 & 18 & 28 & 32 & 54 & 59 \\ 33 & 41 & 18 & 36 & 37 & 38 & 66 \end{vmatrix}$$

Рис. 4.26

Ответы

№	Σ	Веса назначения	№	Σ	Веса назначения
а	190	18, 18, 18, 29, 33, 34, 40	е	200	43, 24, 33, 24, 33, 14, 29
б	172	30, 16, 16, 29, 29, 16, 36	ж	180	23, 16, 15, 15, 38, 36, 37
в	194	19, 19, 34, 26, 19, 38, 39	з	201	42, 29, 23, 32, 26, 23, 26
г	205	41, 43, 23, 23, 27, 25, 23	и	193	23, 28, 27, 31, 26, 31, 27
д	169	31, 21, 17, 17, 37, 29, 17	к	185	18, 19, 18, 30, 30, 32, 38

Пример. Оплата труда работника i на рабочем месте j определяется коэффициентом a_{ij}:

$$A = \begin{vmatrix} 1 & 7 & 1 & 3 \\ 1 & 6 & 4 & 6 \\ 17 & 1 & 5 & 1 \\ 1 & 6 & 10 & 4 \end{vmatrix}.$$

Найти одно из оптимальных назначений и суммарные затраты на производство.

Решение. Применим для решения венгерский алгоритм [2]. Алгоритм основан на теории чередующихся цепей Дж. Петерсена. Преобразуем матрицу A. Вычтем из каждой строки минимальный элемент. Получим

$$A' = \begin{vmatrix} 0 & 6 & 0 & 2 \\ 0 & 5 & 3 & 5 \\ 16 & 0 & 4 & 0 \\ 0 & 5 & 9 & 3 \end{vmatrix}.$$

Изобразим двудольный граф, в котором ребра соответствуют нулевым элементам матрицы A' (рис. 4.27).

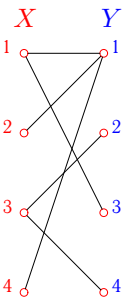


Рис. 4.27

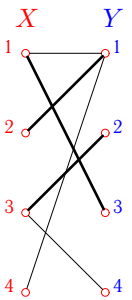


Рис. 4.28

Рассмотрим в этом графе какое-либо максимальное паросочетание. Выделим соответствующие ребра (рис. 4.28).

Чередующейся [1] цепи из X в Y нет. Следовательно, это паросочетание является наибольшим. Выделим множества вершин, не входящие в паросочетание. Это $X_m = \{x_4\}$, $Y_m = \{y_4\}$. Составим множества вершин, которые входят в цепи [2], соединяющие X_m и X:

$$X' = \{x_2, x_4\},\ Y' = \{y_1\}.$$

Преобразуем матрицу A'. Найдем минимальный элемент в строках с номерами элементов множества X' и столбцах с номерами элементов множества $Y \backslash Y'$, т.е в матрице

$$\begin{vmatrix} . & . & . & . \\ . & 5 & 3 & 5 \\ . & . & . & . \\ . & 5 & 9 & 3 \end{vmatrix}.$$

Минимальный элемент равен 3. Вычтем 3 из строк 2 и 4 и добавим 3 к столбцу 1. Получим

$$A'' = \begin{vmatrix} 3 & 6 & 0 & 2 \\ 0 & 2 & 0 & 2 \\ 19 & 0 & 4 & 0 \\ 0 & 2 & 6 & 0 \end{vmatrix}.$$

Соответствующий граф (см. рис. 4.28) изменился. Исчезло ребро (1, 1), и добавились ребра (2, 3) и (4, 4) (рис. 4.29). В этом графе есть ребро (4, 4), не входящее в выделенное паросочетание и не инцидентное его вершинам, но соединяющее X и Y. Паросочетание стало совершенным (рис. 4.30), множества X_m и Y_m пусты, следовательно, задача решена. Таким образом, выбирая элементы исходной матрицы A с номерами ребер полученного паросочетания, найдем наименьшие затраты: $\Sigma = a_{13} + a_{21} + a_{32} + a_{44} = 1 + 1 + 1 + 4 = 7$.

[1] Чередующаяся цепь состоит из нечетного числа чередующихся тонких (из X в Y) и толстых (в обратном направлении) ребер, начиная и кончая тонким ребром. Если бы такая цепь существовала, то число ребер в паросочетании можно было бы увеличить, поменяв толстые и тонкие ребра в этой цепи.

[2] От X к Y можно двигаться по тонким ребрам, в обратном направлении — по толстым.

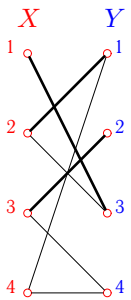


Рис. 4.29

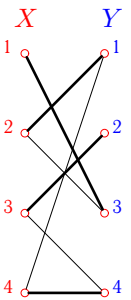


Рис. 4.30

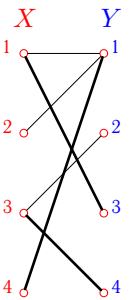


Рис. 4.31

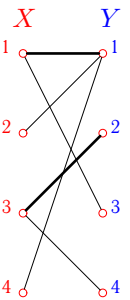


Рис. 4.32

З а м е ч а н и е 1. Паросочетание на рис. 4.28 не единственное. Имеется еще одно наибольшее (3 ребра) паросочетание (рис. 4.31). Множества вершин, не входящие в паросочетание, — это $X_m = \{x_2\}$ и $Y_m = \{y_2\}$. При этом множества X' и Y' не изменятся:

$$X' = \{x_2, x_4\}, \quad Y' = \{y_1\}.$$

З а м е ч а н и е 2. Паросочетание на рис. 4.28 сразу оказалось наибольшим. Однако можно было бы выбрать и не наибольшее, но максимальное паросочетание (рис. 4.32). В этом случае, обратив чередующуюся цепь x_4, y_1, x_1, y_3 (толстые ребра меняются на тонкие, и наоборот), получим наибольшее покрытие (рис. 4.31).

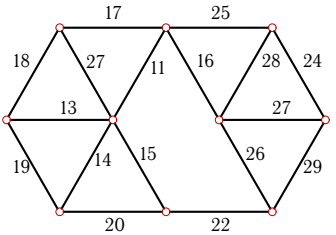
4.6. Остов наименьшего веса

Остовом графа G называют граф, не содержащий циклов и состоящий из ребер графа G и всех его вершин. Таким образом, остов графа является деревом. Число ребер остова равно рангу графа ($\nu^* = n - k$). Число остовов графа можно вычислить по матрице Кирхгофа (см. с. 76).

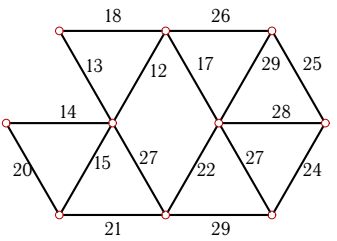
З а д а ч а. Дан взвешенный граф (рис. 4.33). Найти число остовов графа и остов графа минимального веса.

В таблице ответов на с. 76 даны веса ребер минимального остова, суммарный вес остова Σ и число остовов графа n_0.

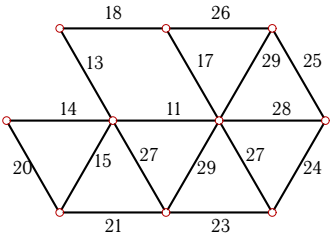
а



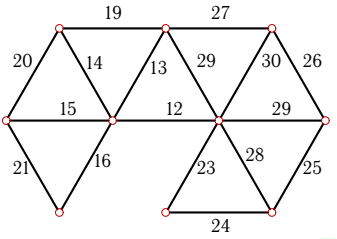
б



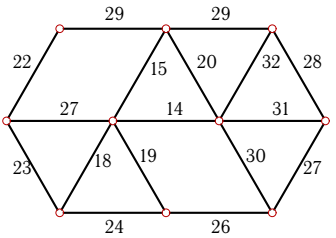
в



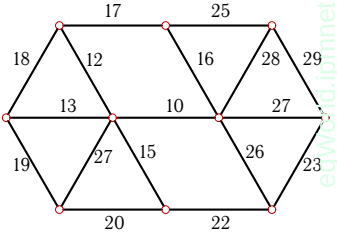
г



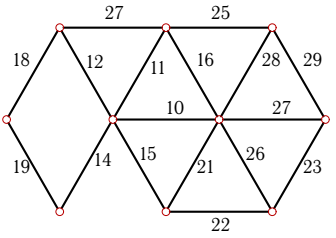
д



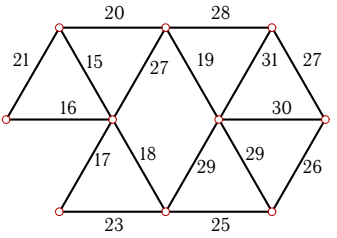
е



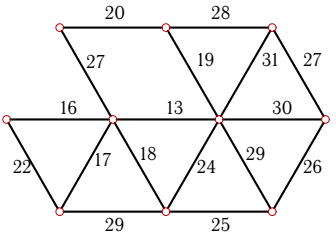
ж



з



и



к

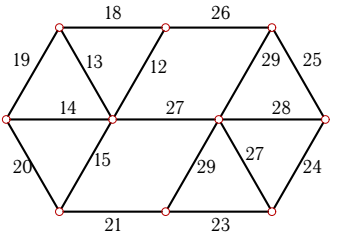


Рис. 4.33

Ответы

№	Веса ребер остова	Σ	n_0
а	16, 11, 13, 14, 15, 17, 22, 25, 24	157	3861
б	17, 12, 13, 14, 15, 21, 26, 25, 24	167	3289
в	11, 13, 14, 15, 17, 21, 23, 24, 25	163	3115
г	12, 13, 14, 15, 16, 23, 24, 25, 26	168	2584
д	14, 15, 18, 19, 23, 22, 26, 27, 28	192	3910
е	10, 12, 13, 15, 16, 19, 22, 23, 25	155	4095
ж	10, 11, 12, 14, 15, 18, 22, 23, 25	150	2944
з	19, 20, 15, 16, 17, 18, 25, 26, 27	183	3289
и	13, 16, 17, 18, 19, 20, 25, 26, 27	181	3115
к	27, 12, 13, 14, 15, 21, 23, 24, 25	174	4095

Пример. Дан взвешенный граф (рис. 4.34).

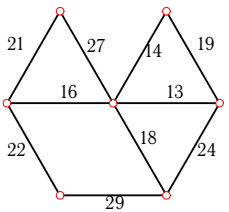


Рис. 4.34

Найти число остовов графа и остов графа минимального веса (экстремальное дерево).

Решение. *Число остовов графа.* Число остовов графа можно вычислить по его матричным характеристикам. Известна следующая теорема.

Теорема 18 (Кирхгофа [1]). *Число остовов графа равно алгебраическому дополнению любого элемента матрицы Кирхгофа.*

Элементы матрицы Кирхгофа B графа G определяются следующим образом. Если вершины i и j графа G смежны, то $b_{ij} = -1$, а если вершины i и j не смежны, то $b_{ij} = 0$. Диагональные элементы матрицы Кирхгофа B равны степеням вершин: $b_{ii} = \deg(i)$. Очевидно свойство: сумма элементов в каждой строке и каждом столбце матрицы Кирхгофа равна 0. Известно также, что алгебраические дополнения всех элементов матрицы Кирхгофа равны, а ранг матрицы Кирхгофа

[1] Kirchhoff G. (1824–1887гг.) — немецкий физик и механик.

неографа порядка n можно вычислить по формуле $\operatorname{rank} B = n - k$, где k — число компонент графа.

Пронумеруем вершины графа (рис. 4.35).

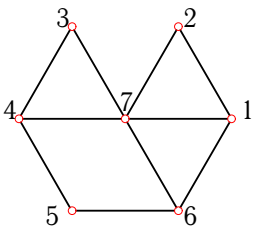


Рис. 4.35

В соответствии с определением запишем матрицу Кирхгофа:

$$B = \begin{bmatrix} 3 & -1 & 0 & 0 & 0 & -1 & -1 \\ -1 & 2 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 2 & -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ -1 & 0 & 0 & 0 & -1 & 3 & -1 \\ -1 & -1 & -1 & -1 & 0 & -1 & 5 \end{bmatrix}.$$

Рассмотрим минор элемента $b_{1,1}$:

$$m_{1,1} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & -1 \\ 0 & 2 & -1 & 0 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 \\ -1 & -1 & -1 & 0 & -1 & 5 \end{bmatrix}.$$

Вычислим определитель: $\det m_{1,1} = 79$. Таким образом, граф имеет 79 остовов, среди которых надо выбрать экстремальное дерево, т.е дерево, обладающее некоторыми эстремальными свойствами, в данным случае — минимальным весом.

Заметим, что матрицу Кирхгофа можно найти, используя формулу

$$B = II^T - 2A, \tag{4.1}$$

где I и A — матрицы инцидентности и смежности графа. Матрицу инцидентности запишем для следующей последовательности номеров ребер: $(3,7)$, $(6,7)$, $(1,2)$, $(6,1)$, $(5,6)$, $(3,4)$, $(2,7)$, $(4,7)$, $(1,7)$, $(4,5)$. Строки матрицы инцидентности соответствуют вершинам, столбцы — ребрам. В неографе матрица инцидентности I состоит из нулей и

единиц, а матрица смежности A симметричная:

$$I = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

Формула (4.1) будет проще, если рассмотреть некоторую *ориентацию графа*. Под ориентацией графа понимают орграф, получающийся заменой каждого ребра дугой произвольного направления. Рассмотрим, например, вариант, приведенный на рис. 4.36.

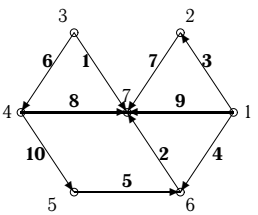


Рис. 4.36

Номера дуг на рисунке указаны полужирным шрифтом в соответствии с номерами ребер исходного графа. Матрица инцидентности I_1 ориентации графа имеет вид

$$I_1 = \begin{bmatrix} 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Сумма элементов в любом столбце этой матрицы всегда равна 0. Матрица Кирхгофа может быть вычислена по формуле

$$B = I_1 I_1^T. \tag{4.2}$$

Алгоритм Дж. Краскала построения остова минимального веса. Строим граф, присоединяя к пустому графу на множестве вершин заданного графа ребро наименьшего веса [10]. К полученному графу последовательно присоединяем остальные ребра, выбирая на каждом шаге ребро наименьшего веса, не образующее цикл с имеющимися

ребрами. В нашем случае начинаем с ребра весом 13 — наименьшего в графе. На рисунках 4.37–4.42 дана последовательность действий. Ребро весом 19 не включается в остов, так как оно образует цикл с ребрами весом 14 и 13.

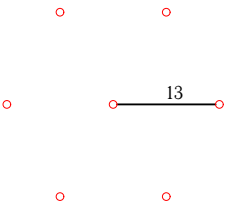


Рис. 4.37

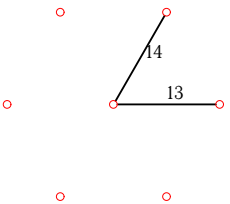


Рис. 4.38

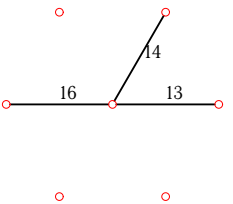


Рис. 4.39

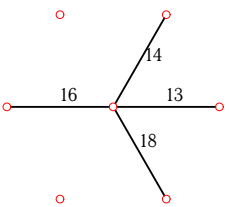


Рис. 4.40

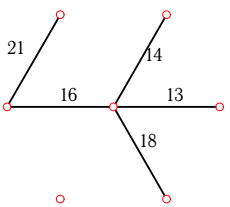


Рис. 4.41

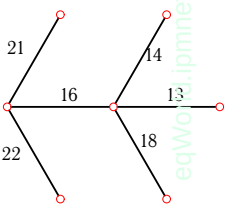


Рис. 4.42

Алгоритм ближайшего соседа построения остова минимального веса. Алгоритм Дж. Краскала требует на каждом шаге проверки на цикличность и предварительной сортировки ребер по весам, что затруднительно для графов с большим числом ребер. Несколько проще следующий алгоритм [13].

1. Отмечаем произвольную вершину графа, с которой начнется построение. Строим ребро наименьшего веса, инцидентное этой вершине.

2. Ищем ребро минимального веса, инцидентное одной из двух полученных вершин. В множество поиска не входит построенное ребро.

3. Продолжаем далее, разыскивая каждый раз ребро наименьшего веса, инцидентное построенным вершинам, не включая в круг поиска все ребра, их соединяющие.

В нашем примере начнем с вершины A. На рисунках 4.43–4.48 дана последовательность действий.

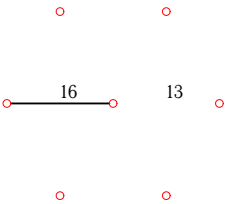


Рис. 4.43

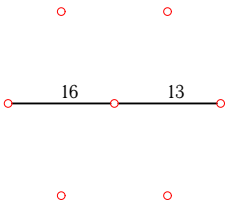


Рис. 4.44

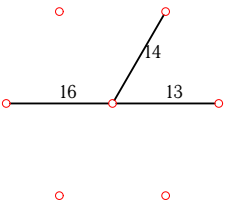


Рис. 4.45

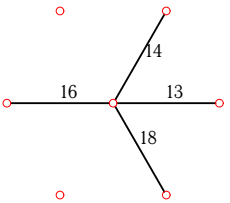


Рис. 4.46

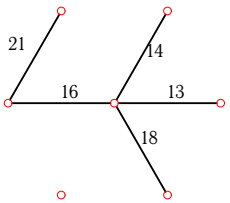


Рис. 4.47

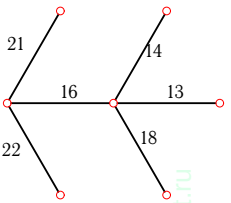


Рис. 4.48

Программа для Maple, использующая встроенные функции, дана на с. 138, программа с визуализацией процесса последовательного построения остова — на с. 141.

4.7. Гамильтоновы циклы

Простой цикл, проходящий через все вершины графа, называется *гамильтоновым* [1]. Простая цепь, проходящая через все вершины графа, называется *гамильтоновой*.

Задача нахождения гамильтоновых циклов получила свое развитие в связи с рядом практических задач. Одной из них является так называемая *задача коммивояжера*, в которой определяется кратчайший гамильтонов цикл (см. с. 84).

В отличие от поиска эйлеровых циклов, проходящих через каждое ребро графа по одному разу, для которых еще Эйлером получено необходимое и достаточное условие существования цикла, для гамильтоновых циклов такого условия не найдено. Существуют, однако, достаточные условия существования гамильтоновых циклов.

Приведем несколько таких признаков [2].

[1] В 1859 г. Уильям Гамильтон (Hamilton W.) предложил математическую игру-головоломку, связанную с обходом вершин додекаэдра, положив тем самым начало одному из самых известных направлений теории графов.

Граф, обладающий гамильтоновым циклом, будем называть *гамильтоновым*.

Теорема 19 (Дирака [1]). *Граф гамильтонов, если степень любой его вершины удовлетворяет неравенству* $\deg(v) \geqslant n/2$.

Теорема 20 (Оре [2]). *Граф гамильтонов, если степени любых двух его несмежных вершин* v *и* u *удовлетворяют неравенству* $\deg v + \deg u \geqslant n$.

Для орграфов также имеется достаточное условие.

Теорема 21 (Гуйя-Ури [3]). *Орсвязный граф обладает гамильтоновым циклом* [4], *если для любой его вершины* v $\deg^{\text{in}}(v) \geqslant n/2$, $\deg^{\text{out}}(v) \geqslant n/2$.

Напомним, что орграф называется *орсвязным*, если любая его вершина достижима из любой вершины.

Орграф называется *полугамильтоновым*, если он содержит гамильтонову цепь.

Среди взвешенных орграфов выделяют орграфы, удовлетворяющие неравенству треугольника:

$$c(u, v) \leqslant c(u, w) + c(w, v). \tag{4.3}$$

Очевидно, этому условию удовлетворяют *евклидовы графы* — графы, вершины которых являются точками плоскости, а веса — евклидовыми расстояниями между ними.

Задача. Найти все гамильтоновы циклы графа (рис. 4.49).

а *б* *в*

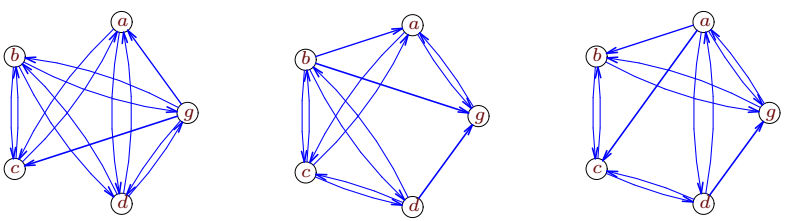


Рис. 4.49

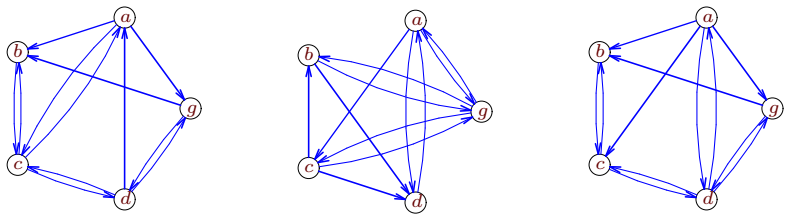
[1] Dirac G.A.

[2] Ore O. [26]

[3] Ghouila-Houri A. [2].

[4] Цикл в орграфе, согласно определению на с. 29, правильнее называть контуром, однако для гамильтоновых циклов, отдавая дань традиции, можно сделать исключение.

г *д* *е*



ж *з* *и*

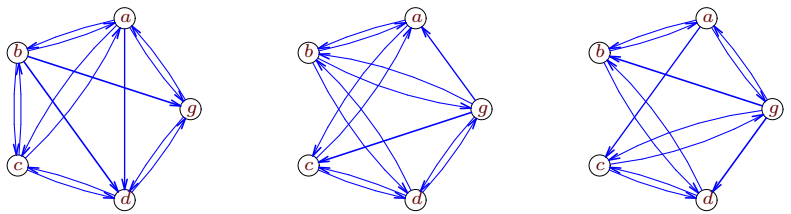


Рис. 4.49 (продолжение)

Ответы

	Гамильтоновы циклы
а	*acbdg*, *adbgc*, *adgbc*, *acbgd*
б	*acdbg*, *acbdg*
в	*agbcd*, *adcbg*, *abcdg*
г	*agbcd*
д	*agcbd*, *acgbd*
е	*agbcd*
ж	*acbdg*, *abgdc*, *agdcb*, *abcdg*, *adcbg*
з	*abdgc*, *acdbg*, *acdgb*, *abgdc*
и	*agcdb*, *abdcg*, *acgdb*

Пример. Дан граф (рис. 4.50). Найти все гамильтоновы циклы графа.

Решение. В [18] описан алгебраический алгоритм [1] нахождения всех гамильтоновых циклов, основанный на теореме 5 (см. с. 11). Эта теорема позволяет найти число маршрутов определенной длины по элементам степени матрицы смежности A графа. Однако эти маршруты остаются безымянными, теорема не дает информации о вершинах в маршрутах. Если немного изменить матрицу смежности и малоинформативные единицы в ней заменить на имена вершин, то возведение такой матрицы в

[1] Yau S.S. (1967 г.), Danielson G.H. (1968 г.), Dhawan V. (1969 г.).

степень даст больше информации о маршрутах. Введем таким образом модифицированную матрицу смежности B по правилу: элемент

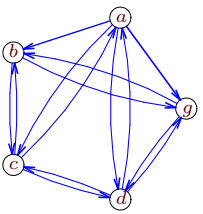


Рис. 4.50

матрицы $\beta_{ij} = v_j$, если существует путь из вершины v_i в вершину v_j.

Для заданного графа получим матрицу смежности A и матрицу B:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & b & c & d & g \\ 0 & 0 & c & 0 & g \\ a & b & 0 & d & 0 \\ a & 0 & c & 0 & g \\ 0 & b & 0 & d & 0 \end{bmatrix}.$$

Далее последовательно находим вспомогательные матрицы P'_k: и P_k

$$P'_{k+1} = B \cdot P_k, \; P_1 = A, \; P_k = \Phi(P'_k).$$

Матрицы B и P_k умножаются по обычному правилу умножения матриц, а под произведением вершин понимается некоммутативная бинарная операция склеивания, заданная на множестве слов. Слово — упорядоченная последовательность вершин (букв). Произведение слова $u_1u_2, ..., u_k$ и слова $v_1v_2, ..., v_m$ есть слово $u_1u_2, ..., u_kv_1v_2, ..., v_m$.

Матрица P'_k преобразуется в матрицу P_k специальным оператором Φ. Оператор Φ, действующий на элементы p_{ij} матрицы, вычеркивает (обнуляет) те ее элементы, в которых содержится вершина v_i или v_j. Такие элементы содержат контуры, замыкающиеся на v_i или v_j. Так как на главной диагонали P_k содержится информация о циклах, для упрощения и ускорения расчетов можно обнулять главную диагональ всех матриц P_k, кроме, разумеется, последней — P_n, поскольку ее диагональ и содержит искомые циклы без начальной и конечной вершины. Эти вершины легко добавить после окончания основной итеративной процедуры.

Для заданного графа получим последовательность матриц P_k (вспомогательные матрицы P'_k не выписаны). Имеем

$$P_2 = \begin{bmatrix} 0 & c+g & b+d & c+g & b+d \\ c & 0 & 0 & c+g & 0 \\ d & a & 0 & a & a+b+d \\ c & a+c+g & a & 0 & a \\ d & 0 & b+d & 0 & 0 \end{bmatrix}.$$

Заметим, что матрица P_2 получена из P'_2 действием оператора Ф: в первом столбце и первой строке нет слов с вершиной a, во втором столбце и второй строке отсутствует вершина b и т.д. Далее

$$P_3 = \begin{bmatrix} 0 & dc+dg & gb+gd & bc+bg & cb+cd \\ cd+gd & 0 & gd & ca & ca+cd \\ 0 & ag+da+dg & 0 & ag+bg & ab+ad+da \\ 0 & ac+ag+ca & ab+gb & 0 & ab+ca+cb \\ bc+dc & da+dc & da & bc & 0 \end{bmatrix},$$

$$P_4 = \begin{bmatrix} 0 & cdg+gdc & bgd+dgb & cbg+gbc & bcd+dcb \\ gdc & 0 & gda & cag & cad+cda \\ bgd & adg+dag & 0 & abg & dab \\ gbc & cag & agb & 0 & acb+cab \\ bcd & dac+dca & dab & bca & 0 \end{bmatrix}.$$

Последняя матрица (P_5) — диагональная. Все ее элементы — нулевые, кроме $P_{5,11} = bgdc + cbgd + dgbc + gbcd$, $P_{5,22} = cadg + cdag + gdac + gdca$, $P_{5,33} = abgd + adgb + bgda + dagb$, $P_{5,44} = acbg + agbc + cabg + gbca$, $P_{5,55} = bcad + bcda + dacb + dcab$. К полученным слагаемым в элементе матрицы P_{kk} добавим в начале или в конце по вершине v_k, где $v_1 = a$, $v_2 = b$, $v_3 = c$, $v_4 = d$, $v_5 = g$, и выполним круговую перестановку так, чтобы вершина a оказалась первой. Многие гамильтоновы циклы повторяются. Множество ответов для данной задачи будет следующим:

$$abgdc,\ adgbc,\ acbgd,\ agbcd.$$

Полный граф порядка n имеет $(n-1)!$ гамильтоновых циклов.

Очевидно, вручную этот алгоритм не реализуется. Вариант программы для Maple, работающей по описанному алгоритму, дан на с. 144.

4.8. Задача коммивояжера

Это образное название устойчиво закрепилось за одной из самых интересных, практически значимых и одновременно сложных задач теории графов. Задача, берущая свое начало из работ Гамильтона, состоит в определении кратчайшего гамильтонова цикла в графе. Ее

решение связано [18] с решением задачи о назначениях (см. с. 70) и с задачей об остове наименьшего веса (см. с. 74).

Задача. Найти кратчайший гамильтонов цикл графа (рис. 4.51).

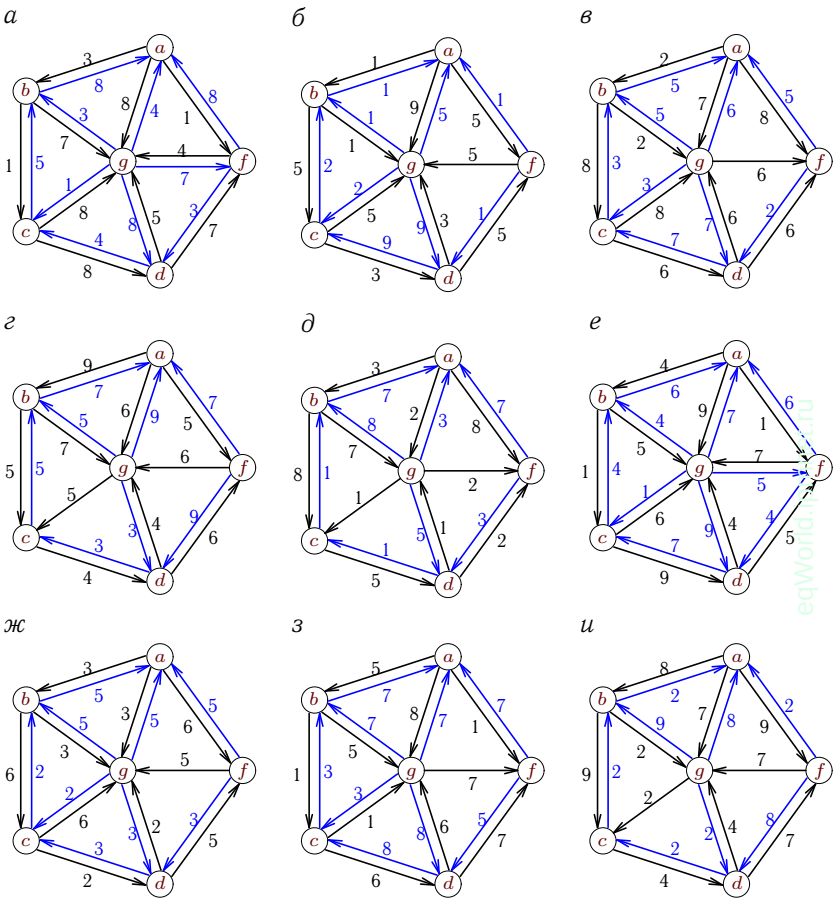


Рис. 4.51

Ответы

	L	Цикл		L	Цикл		L	Цикл
а	23	$afdgcb$	г	29	$afgdcb$	ж	20	$afdgcb$
б	13	$abgcdf$	д	16	$agfdcb$	з	25	$afdgcb$
в	24	$abgcdf$	е	20	$afdgcb$	и	24	$afgdcb$

Пример. Дан взвешенный орграф (рис. 4.52). Найти кратчайший гамильтонов цикл.

Решение. Способ 1. Рассмотрим алгоритм ближайшего соседа (Nva — Nearest vertex add) [1]. Начнем движение по графу из произвольной вершины, например из вершины a. Выбирая из двух возможных дуг, $a \to f$ и $a \to b$, короткую дугу, $a \to b$, длиной 1, далее до вершины c двигаемся без разветвлений: $a \to \to f \to d \to c$. Из c направляемся по более короткой дороге в g, затем в b и возвращаемся в a. Суммарная длина цикла равна $1 + 3 + 8 + 1 + \\ + 3 + 3 = 19$. Алгоритм ближайшего соседа не гарантирует оптимального решения. Решение зависит от выбора начальной вершины и от выбора направления движения при наличии двух и более направлений одинакового веса. Проверим другую вершину в качестве начальной.

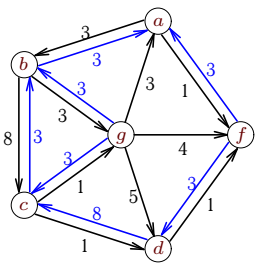


Рис. 4.52

Если начинать движение из b в сторону a по дуге весом 3, то цикл $b \to a \to f \to d \to c \to g \to b$ будет совпадать с найденным, отличаясь только началом отсчета. При выборе направления $b \to g$, также весом 3, получим цикл $b \to g \to c \to d \to f \to a \to b$, имеющий вес $3 + 3 + \\ + 1 + 1 + 3 + 3 = 14$. Этот цикл, наименьший по весу, можно принять за окончательный ответ. В действительности, суммарный вес 14 является наименьшим для этой задачи.

Известна оценка алгоритма ближайшего соседа [2] для графа, веса которого удовлетворяют неравенству треугольника (см. с. 81):

$$c_{\text{Nva}} \leqslant 0,5([\ln n] + 1)c_{\text{opt}},$$

где c_{opt} — вес оптимального маршрута. В нашем случае коэффициент в этой оценке $0,5([\ln n] + 1) = 1$.

Соответствующая программа для Maple дана на с. 145.

Способ 2. Рассмотрим алгоритм ближайшей вставки (Nvi — Nearest insert) [2]. Построение цикла начнем с любой вершины, например c. Присоединим к c вершину, образующую вместе с ней цикл. Из трех возможных вариантов, $c \to d \to c$, $c \to g \to c$ и $c \to b \to c$, возьмем самый короткий, $c \to g \to c$, длиной 4 (рис. 4.53). На следующем шаге можно добавить либо вершину d, вставляя дуги $g \to d$ и $d \to c$ вместо дуги $g \to c$, либо b уже с двумя вариантами циклов: $g \to c \to b \to g$

[1] Аналогичный по стратегии алгоритм использовался для определения остова наименьшего веса (см. с. 79).

и $g \to b \to c \to g$. Выбираем цикл наименьшей длины: $g \to c \to b \to g$ (рис. 4.54). Аналогично последовательно вставляем в цикл вершины d, f, a (рисунки 4.55–4.57). В результате получаем цикл длиной $3 + 3 + 1 + 3 + 8 + 3 = 21$.

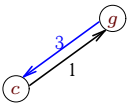


Рис. 4.53

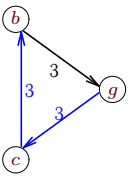


Рис. 4.54

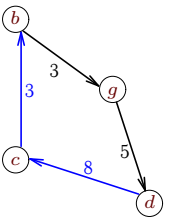


Рис. 4.55

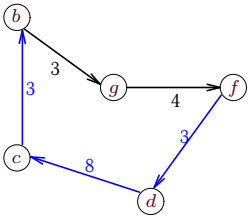


Рис. 4.56

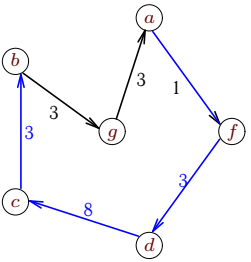


Рис. 4.57

Результат далек от оптимальной длины 14. В рамках этого алгоритма можно добиться меньшей длины, меняя начальную вершину и направления в точках ветвления решения.

Известна оценка алгоритма ближайшей вставки [2], для графа, веса которого удовлетворяют неравенству треугольника (см. с. 81):

$$c_{\text{Nvi}} \leqslant 2c_{\text{opt}},$$

где c_{opt} — вес оптимального маршрута.

Способ 3. Одним из методов искусственного интеллекта является муравьиный алгоритм Марко Дориго [1]. Основная идея алгоритма подсмотрена в природе и имитирует движение колонии муравьев. По форме этот алгоритм похож на жадный Nva (способ 1) и в некоторой степени является его обобщением. Если в алгоритме ближайшего соседа выбор дальнейшего пути производится, исходя из минимального расстояния до очередной вершины, то здесь выбором управляет случайная функция, направляющая движение от текущего положения с большей вероятностью в вершину j, в которой наибольшее значение некоторой

[1] Marco Dorigo [34].

функции $P_{ij,k}$ (где i — номер вершины, в которой производится выбор, k — номер муравья, движущегося по дугам графа). Как и в Nva, во время движения создается список пройденных вершин, что позволяет избежать преждевременного зацикливания. Приведем вид функции, управляющей переходом из данной вершины i в вершину j:

$$P_{ij,k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum\limits_{m}\tau_{im}^{\alpha}\eta_{im}^{\beta}}, \tag{4.4}$$

где τ_{ij} — количество феромона (pheromon), оставленного муравьями на дуге $[i,j]$; η_{ij} — величина, обратная весу (длине) дуги $[i,j]$; α, β — эмпирические коэффициенты. Функция $P_{ij,k}$ подсказывает муравью номер вершины j, в которую он должен направиться. В знаменателе (4.4) стоит нормирующий коэффициент, такой, что $0 \leqslant \leqslant P_{ij,k} \leqslant 1$. Индекс m в сумме пробегает по всем непройденным вершинам, смежным с i. В реальности муравей оставляет след (феромон) во время прохождения пути, и чем чаще он возвращается в исходную точку (а это возможно, если он выбирает оптимальные пути), тем четче след. В математической же модели функция τ_{ij} увеличивается только по завершении маршрута на величину, обратно пропорциональную длине маршрута. При $\alpha = 0$ алгоритм совпадает с Nva — муравей руководствуется только длиной пути. При $\beta = = 0$ основой для выбора пути является только опыт (количество феромона, или «глубина следа») предыдущих муравьев-исследователей. Важно отметить еще одно отличие от алгоритма Nva. Выбор пути производится не по максимуму функции $P_{ij,k}$, а случайным образом, но на случай, конечно, влияет значение $P_{ij,k}$. Поясним это на примере. Пусть муравей k подошел к некоторой вершине 8 и обнаружил, что перед ним 7 возможных путей к семи вершинам (на уже пройденные он внимания не обращает). Куда идти? Муравей доверяется случаю. Он «пускает рулетку» (рис. 4.58). В какой

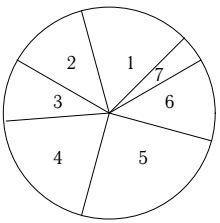


Рис. 4.58

сектор «шарик» закатится, туда и идти. Однако рулетка, размеченная функцией $P_{8j,k}$, $j = 1,...,7$, имеет неравные сектора. Чем ближе вершина и чем глубже туда след, тем больше сектор. Таким образом,

муравей использует и опыт предшественников (τ_{ij}), и здравый смысл (η_{ij}), и случайный фактор, т.е. все как в жизни.

Для того чтобы не пропустить оптимальное решение, в муравьином алгоритме предусмотрено «испарение» следа. Это достигается введением коэффициента p в итеративной формуле $\tau_{ij} = (1-p)\tau_{ij}$, применяющейся после каждого цикла обхода графа.

В алгоритме действует целая колония муравьев. Математически это означает, что в каждом цикле обхода движение производится из разных вершин независимым образом.

Здесь приведен самый простой вариант алгоритма. Алгоритм может быть улучшен. Для ускорения сходимости иногда вводят так называемых *элитных* муравьев [34]. В [7] приведены наилучшие комбинации параметров α и β.

Соответствующая программа для Maple дана на с. 148.

Способ 4. Алгоритм отжига. Этот алгоритм, как и муравьиный алгоритм, относится к вероятностным методам решения. Ключевым моментом в таких подходах является случайный выбор одного из нескольких возможных решений вместо анализа каждого. Это позволяет сократить время счета, а время счета в некоторых задачах на графах имеет принципиальное значение. Простой перебор для задач, сложность которых (время счета) растет по показательному или факториальному закону в зависимости от порядка графа, может даже для небольших графов оказаться недопустимо длительным. Оценка этой характеристики в каждом приложении своя. Например, для мобильного робота [1], выполняющего маневр в поиске оптимального решения, бортовой компьютер должен решать задачу коммивояжера за доли секунды. Прямой же перебор вариантов при пяти-десяти пунктах назначения возможен только за несколько минут, что в данном случае никак не-приемлемо. При оптимизации проектируемых в лабораторных условиях тепловых, газовых, электрических или транспортных сетей, как правило, с сотней или более вершин (узлов), проект может затянуться даже на многие годы. Именно поэтому актуальны вероятностные подходы, дающие, может быть, не самые точные, но вполне допустимые решения. Рассмотрим метод отжига, обычно изучаемый в курсах теории искусственного интеллекта.

Образное название лишь отчасти передает содержание и связывается с процессом образования структуры металла при нагревании и дальнейшем охлаждении. Известно, что только контролируемое охлаждение приводит к желаемой структуре металла. При большей температуре степень свободы частиц (в нашем случае — количество вариантов решения) больше, при меньшей — меньше. Если дать возможность

[1] Мобильные роботы и мехатронные системы: Материалы научной школы – конференции. — М.: Изд-во Моск. ун-та, 2000.

алгоритму случайно выбирать решение, оптимальное на каждом шаге (жадный алгоритм), то можно пропустить ход, не лучший локально, но дающий в результате более оптимальное решение.

В методе отжига очередной порядок следования по маршруту между городами выбирается случайно, небольшим изменением предыдущего решения, предположительно оптимального. Самый простой вариант изменения — перестановка двух случайно выбранных городов в маршруте следования. Если полученный маршрут лучше всех существовавших ранее, то этот маршрут берется за очередной [1]. Если маршрут хуже, то самый простой вариант — это не брать его (зачем ухудшать решение?). Однако так решение может закатиться в один из локальных минимумов, которыми изобилуют подобные задачи (рис. 4.59). Именно

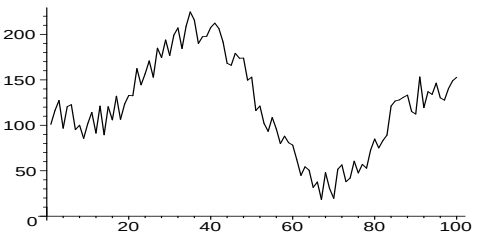


Рис. 4.59

поэтому плохому решению надо дать шанс. Шанс этот (или, правильнее, вероятность) рассчитывается по формуле

$$P = \exp(-\Delta L/T),$$

где ΔL — положительная разность между качеством тестируемого и ранее полученного оптимального решений, T — некоторый постоянно уменьшающийся параметр (условно — температура). В случае задачи коммивояжера качество оценивается длиной маршрута. Очевидно, что чем меньше текущее решение портит минимум и чем больше температура, тем больше вероятность принятия «пробного» выхода из локального минимума.

Снижение температуры обычно производится по формуле $T_{k+1} = \alpha T_k$, где $0 < \alpha < 1$.

Соответствующая программа для Maple дана на с. 152.

[1] Иногда полученный вариант сравнивается с предыдущим.

Глава 5

MAPLE-ПРОГРАММЫ

5.1. Радиус и диаметр графа

В первой части программы задается граф и определяются его радиус, диаметр и центр. Исследуется неориентированный граф, заданный на рис. 1.8 (см. с. 16).

Граф задается оператором `graph(V,E)`, где `V` — множество вершин графа, `E` — множество ребер. В качестве вершин можно использовать произвольные имена в кавычках, например `"my nomer"` (включая пробелы), переменные, значения которых ранее не определяются, или, что проще, просто номера. В данном случае номера вводятся оператором повторения `V:={$1..n}`, заменяющим простое перечисление `V:={1,2,3,4,5}`. Другой оператор повторения, служащий для этих же целей, имеет вид `V:={seq(i,i=1..n)}`.

Для задания графа можно использовать также операторы `new(G)`, `addvertex` и `addedge`.

Оператор вывода рисунка графа на печать имеет три варианта формы вывода вершин: `Concentric`, `Linear` и, по умолчанию, равномерно по окружности. Если в опции `Concentric` задать несколько списков, например `Concentric([1,2,3],[4,5,6,7])`, то вершины первого списка будут на внутренней окружности. Аналогично выводятся вершины графа с опцией `Linear`, пример можно посмотреть на с. 104.

Отметим необходимость использования оператора печати-вывода на экран `print(j)`. Дело в том, что значение в теле условного оператора `if ... then ... end` не выводится на печать, если оператор находится внутри цикла, даже когда везде стоят точки с запятой, а не двоеточие:

```
for j to n do if W[j]=RG then j; end;od;
```

Напротив, оператор `print(j)` выводит результат в любом случае. Для сокращения записи в операторе цикла опускаются начальное значение счетчика цикла, принятое по умолчанию единицей, и шаг цикла, также равный единице: `for j from 1 by 1 to n do`. Перед циклом вставлена запись `Центр;`, которая воспринимается просто как неназначенная переменная или заголовок следующего далее списка вершин. В Maple допускается русский шрифт для переменных.

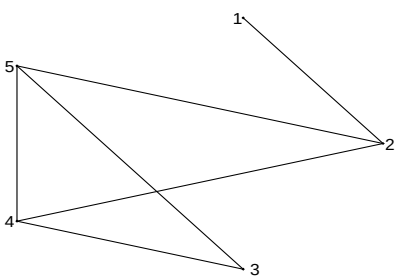
Основным оператором программы является `allpairs`, вычисляющий расстояния между отдельными вершинами графа. Результатом является квадратная таблица расстояний, по форме совпадающая с матрицей расстояний [1]. Эксцентриситет вершины `i` содержится в переменной `W[i]`. Минимальный эксцентриситет равен радиусу графа, максимальный — диаметру. Можно убедиться в справедливости теоремы 1 (см. с. 8) о сравнении диаметра графа и ранга матрицы смежности. Для этого достаточно определить матрицу смежности `A:=adjacency(G)` и вызвать оператор `Rank(A)` вычисления ранга матрицы. В данной задаче ранг матрицы смежности равен пяти, а диаметр графа — трем.

Для вычисления диаметра можно также использовать оператор `diameter`.

Во второй части программы проверяется наличие эйлеровой цепи в графе. Оператор `type` определяет четность степени `i`-й вершины, которая вычисляется функцией `vdegree(i,G)`. Если степень нечетная (`odd`), то счетчик числа таких вершин увеличивается [2].

Программа 1

```
> restart:with(networks): with(LinearAlgebra):
> E:={{1,2},{2,4},{2,5},{3,4},{3,5},{5,4}}: # Ребра
> n:=5:  V:={$1..n}:                  # Вершины
> G :=graph(V,E):                     # Граф
> draw(Concentric([2,1,5,4,3]),G); # Рисунок
```




```
> S:=allpairs(G);
> for i to n do W[i]:=max(seq(S[i,u],u=1..n)):od:
> RG:=min(seq(W[i],i=1..n)); # Радиус
```

[1] В Maple матрица и таблица являются различными типами.

[2] Для четных чисел в Maple введено обозначение `even`.

```
> DG:=max(seq(W[i],i=1..n)); # Диаметр
```

$$RG := 2$$
$$DG := 3$$

```
> Центр;
> for j to n do if W[j]=RG then print(j) end:od:
```

Центр

2
4
5

```
> k:=0: # Счетчик числа вершин нечетной степени
> for i to n do
> if type(vdegree(i,G),odd) then k:=k+1;end;od;
> if k=0 or k=2 then print("Есть эйлерова цепь");fi;
```

В следующей программе матрица расстояний вычисляется по способу 2 (см. с. 11) без оператора `allpairs`. Рассмотрим случай неографа. Матрица расстояний будет при этом симметричной. Степень матрицы смежности накапливается в матрице `A0`, первоначально единичной. Счетчик вычислений `N` для неографа ограничен числом `n(n-1)/2` элементов матрицы над диагональю. Для орграфа это число равно `n(n-1)`.

Приведем только ту часть программы, которая следует за вводом данных (пять первых строчек программы 1). Программа требует подключения пакета линейной алгебры `LinearAlgebra`. Для сравнения результатов, полученных оператором `allpairs` и по программе 2, необходимо привести матрицы расстояний к одному типу. Легче всего таблицу `allpairs` (тип *table*) конвертировать в матрицу с помощью конструктора матриц `Matrix`. Заметим, что оператор `convert` с этим не справляется. В эквивалентности результатов убеждаемся, используя оператор `Equal` пакета `LinearAlgebra`.

Оценить время работы алгоритма можно с помощью оператора `time()`. Для этого перед первым оператором программы надо определить текущее время, которое отсчитывается в секундах с начала сессии Maple: `t1:=time():`. В конце программы время счета можно вывести на экран: `time()-t1`.

Программа 2

```
> A:=adjacency(G);
> A0:=IdentityMatrix(n):
> AllPairs:=Matrix(n,shape=symmetric):
> N:=0:
```

```
> for k while N<>n*(n-1)/2 do
>   A0:=A0.A:
>   for i to n do
>     for j to i-1 do
>       if AllPairs[i,j]=0 and A0[i,j]<>0 then
>       AllPairs[i,j]:= k; N:=N+1:end:
>     od:
>   od:
> od:
> S:=AllPairs:
> S0:=allpairs(G):
> A1:=Matrix(n,n,S0):
> Equal(A1,S);
```

true

Еще одним эффективным алгоритмом для определения кратчайших путей между всеми парами вершин графа является алгоритм Уоршелла и Флойда [20], реализованный в следующей программе. Как и в предыдущей программе, ввод данных совпадает с программой 1 (см. с. 92). Программа 3 записывается вслед за оператором `draw`.

Сначала матрице `AllPairs` присваивается матрица весов (расстояний). В простейшем случае, когда все расстояния равны 1, это матрица смежности. Вместо нулевых элементов в матрице смежности ставятся бесконечные (`infinity`) расстояния. Интересно отметить, что на матрицу, полученную оператором `adjacency`, не действует оператор подстановки `subs(0=infinity,AllPairs)`, поэтому замена нулей на бесконечность произведена поэлементно в двойном цикле. Для сравнения на печать выведены обе матрицы, `allpairs` и `AllPairs`. Они совпадают. Заметим, что `allpairs`, встроенный в пакет `networks`, также использует алгоритм Уоршолла–Флойда.

Программа 3

```
> allpairs_=Matrix(n,n,allpairs(G));
> AllPairs:=adjacency(G):
```

$$allpairs\_ = \begin{bmatrix} 0 & 1 & 3 & 2 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{bmatrix}$$

```
> for i to n do
```

```
>    for j to n do
>     if AllPairs[i,j]=0 then
>     AllPairs[i,j]:=infinity; fi;
>    od:
>  od:
>  for i to n do AllPairs[i,i]:=0; od:
>  for m to n do
>   for i to n do
>    for j to n do
>     AllPairs[i,j]:=min(AllPairs[i,j],
>       AllPairs[i,m]+AllPairs[m,j]);
>    od;
>   od;
>  od;
>  AllPairs_=AllPairs;
```

$$AllPairs\_ = \begin{bmatrix} 0 & 1 & 3 & 2 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{bmatrix}$$

5.2. Реберный граф

Рассмотрим граф на рис. 1.2 (см. с. 10). Вершины графа задаются перечислением имен (в данном случае, номеров) оператором `addvertex`, а ребра — оператором `connect`. При этом можно сразу задать все ребра, инцидентные одной вершине, например ребра (2, 1), (2, 4), (2, 5): `connect(2,{1,4,5},G)`. В операторе `draw` для изображения графа дана опция формы рисунка — `Concentric`. Список вершин в этой опции указывает на порядок изображения вершин равномерно по окружности против часовой стрелки, начиная с нулевого угла, отложенного от горизонтальной оси координат, направленной направо. По умолчанию оператор `draw` также использует эту опцию и на первое место помещает вершину с меньшим номером. Для того чтобы изображение появилось на экране, необходимо двоеточие после оператора `draw` заменить на точку с запятой. Здесь и в некоторых других программах после оператора `draw` ставится двоеточие, чтобы не дублировать имеющийся в тексте рисунок.

По формуле (1.2) (см. с. 14) вычисляется теоретическое число `m1` ребер в реберном графе.

В двойном цикле в соответствии с определением реберного графа формируется множество его ребер. Ребра исходного графа обозначаются в системе Maple как e1, e2.. en. Поэтому для того, чтобы они были доступны, в цикле применяется оператор || «приклеивания» номера к букве e.

Полученное множество ребер E1 позволяет создать и изобразить искомый реберный граф G1.

Программа 4

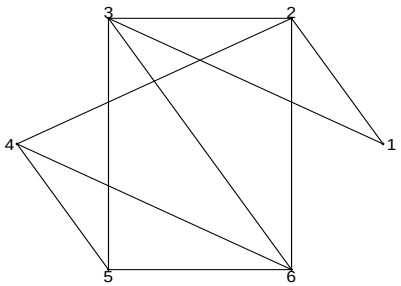
```
> restart:with(networks):
  Исходный граф
> new(G): n:=5: addvertex({$1..n},G):
> connect(2,{1,4,5},G): connect(3,{4,5},G):
> connect(4,5,G):
> draw(Concentric([2,1,5,4,3]),G):
> E:=edges(G):
> m:=nops(E):
  Число ребер реберного графа
> m1:=add(vdegree(i,G)^2,i=1..n)/2-m;
> E1:={}:
> for i to m do
>   for j from i+1 to m do
>     if nops(ends(e||i,G) intersect ends(e||j,G))=1
>      then E1:=E1 union {{i,j}};
>     fi;
>   od;
> od;
> E1; # Множество ребер
```

$$m1 := 10$$

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

```
> new(G1): addvertex({$1..m},G1): # Реберный граф
> addedges(E1,G1):
> m1:=nops(E1); #  Число ребер
> draw(G1);
```

$$m1 := 10$$



5.3. Хроматический полином

Рассматривается граф с рис. 1.10 (см. с. 19). Хроматический полином получается с помощью оператора `chrompoly` пакета `networks`. Вычисление числа раскрасок выполняется при помощи оператора `eval`. Точно так же можно воспользоваться оператором подстановки `subs(x=3,p)`. Хроматическая редукция по пустым графам сразу следует из вида полинома, так как число пустых графов соответствующих порядков равно модулям коэффициентов полинома. Хроматическая редукция по полным графам выполняется с помощью чисел Стирлинга (`stirling2`). Для этого к программе подключен пакет `combinat`.

Программа 5

```
> restart: with(networks): with(combinat):
> new(G):addvertex({$1..4},G):
> addedge([{1,3},{1,4},{2,3},{3,4}],G):
> p:=chrompoly(G,x); # Хроматический полином
```

$$p := x\,(-1+x)^2\,(-2+x)$$

```
> eval(p,x=3); # Число раскрасок в 3 цвета
```

$$12$$

Хроматическая редукция по пустым графам

```
> add(coeff(p,x,i)*O[i],i=1..4);
```

$$-2\,O_1 + 5\,O_2 - 4\,O_3 + O_4$$

```
> for j to 4 do
> a[j]:=add(K[i]*stirling2(j,i),i=1..j); od;
```

$$a_1 := K_1$$

$$a_2 := K_1 + K_2$$
$$a_3 := K_1 + 3\,K_2 + K_3$$
$$a_4 := K_1 + 7\,K_2 + 6\,K_3 + K_4$$

Хроматическая редукция по полным графам

```
> add(coeff(p,x,i)*a[i],i=1..4);
```
$$2\,K_3 + K_4$$

5.4. Ранг-полином графа

Получим ранг-полином графа с рис. 1.12 (см. с. 23). Для этого воспользуемся еще одним способом задания графа, особенно удобным для графа, близкого к полному. Сначала зададим полный (`complete`) граф K_4, затем сотрем (`delete`) в нем два ребра, выбрав последние. Ребра в Maple обозначаются как `e1, e2` и т.п.

После этого можно давать команду для генерации искомого полинома. В качестве переменных возьмем `x` и `y`. Для удобства раскроем скобки оператором `expand`. Значение полинома в некоторых определенных точках (x, y) имеет вполне конкретный смысл, указанный в комментариях к программе. Рассматриваются только остовные подграфы (по четыре вершины в каждом: $n = 4$). Заметим, что в число остовных подграфов входят и несобственные подграфы, т.е. пустой подграф и подграф, совпадающий с самим графом.

Ранг подграфа на подмножестве его ребер `E1` можно вычислить с помощью оператора `rank(E1,G)`. Для вычисления ранга самого графа достаточно взять все ребра графа: `e1,e2,e3,e4`.

Ациклический граф — граф, не содержащий циклов, или граф с нулевым рангом ($\nu = 0$).

Для данного графа оператор `rankpoly(G,0,1)` находит число подграфов ранга 3.

Число остовов определяется также и в программе 30 (см. с. 139).

Программа 6

```
> restart: with(networks):
> G:=complete(4): G:=delete({e5,e6},G):
> expand(rankpoly(G,x,y));
```
$$x^3 + 4\,x^2 + 6\,x + x\,y + 3 + y$$
```
> rank({e1,e2,e3,e4},G);
```
$$3$$
```
> rankpoly(G,1,1), # Число подграфов
```

```
> rankpoly(G,1,0), # Число ациклических подграфов
> rankpoly(G,0,1), # Число подграфов ранга графа G
> rankpoly(G,0,0); # Число остовов
```

$$16,\ 14,\ 4,\ 3$$

5.5. Циклы в неографе

Анализируется граф с рис. 1.17 (см. с. 26). Вычисления производятся по формулам (1.15) и (1.17) (см. с. 27). Производятся возведение в степень матрицы смежности, затем вычисление следа полученной матрицы с помощью оператора `Trace`. Оператор `neighbors(i,G)` находит список вершин неографа, соседних с вершиной `i`. Оператор `vdegree(i,G)` вычисляет степень вершины `i`.

Самый короткий цикл в графе можно найти с помощью оператора `girth(G,short)`. В переменной `short` сохраняются имена ребер этого цикла. Названия вершин, образующих цикл, извлекаются в цикле оператором `ends`. Длина цикла `nc` в данном примере равна 3. Из двух циклов длиной 3 берется один произвольный.

Программа 7

```
> restart:with(networks): with(LinearAlgebra):
> n:=5:
> V:={$1..n}: # Вершины
> E:={{1,2},{1,3},{1,5},{5,3},{4,3},{4,5}}:
> G :=graph(V,E): # Граф
> draw(Concentric([2,1,5,4,3]),G): # Рисунок
> A:=adjacency(G):
> A3:=A^3:
> C3:=Trace(A3)/6; # Количество циклов длиной 3
```

$$C3 := 2$$

```
> A4:=A^4:
> t4:=Trace(A4); # Циклические маршруты длиной 4
```

$$t4 := 60$$

```
> for i to n do
>   s:=0:
>   for j in neighbors(i,G) do
>   s:=s+vdegree(j,G)-1:od:
>   t4:=t4-s-vdegree(i,G)^2:
> od:
```

```
> C4:=t4/4/2;# Количество циклов длиной 4
```

$$C4 := 1$$

```
> nc:=girth(G,short): seq(ends(short[i],G),i=1..nc);
```

$$\{1, 5\}, \{3, 5\}, \{1, 3\}$$

5.6. Матрица инцидентности

Сначала матрица инцидентности для графа с рис. 1.17 (см. с. 26) определяется с помощью оператора `incidence(G)` пакета `networks`. Число строк матрицы инцидентности равно числу вершин, число столбцов — числу ребер. Если граф простой (не мультиграф с кратными ребрами и не псевдограф с петлями), то сумма элементов каждого столбца равна 2, а сумма элементов каждой из строк — степени соответствующей вершины.

Матрицу инцидентности можно построить по матрице смежности и без использования оператора `incidence(G)`. Двойной цикл обхода матрицы смежности по верхнему треугольнику с использованием условного оператора дает искомую матрицу, отличающуюся от предыдущей за счет изменения номеров ребер [1].

Заметим, что запись условного оператора можно завершать как с помощью `fi`, так и с помощью `end if` или просто `end`.

Программа 8

```
> restart: with(networks):
> n:=5: V:={$1..n}: # Вершины
> E:={{4,3},{1,3},{1,2},{5,3},{5,4},{1,5}}:
> m:=nops(E):        # Ребра
> G :=graph(V,E):    # Граф
> A:=adjacency(G):   # Матрица смежности
> incidence(G);      # Матрица инцидентности (1)
```

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

```
> In:=Matrix(n,m):
> k:=0:
```

[1] Одна особенность оператора `incidence`: Maple переставляет столбцы в матрице инцидентности в произвольном порядке, а не по порядку номеров ребер.

```
> for i to n do
>   for j from i to n do
>    if A[i,j]=1 then
>     k:=k+1: In[i,k]:=1: In[j,k]:=1:
>    fi;
>   od;
> od;
> In;                # Матрица инцидентности (2)
```

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

5.7. Транзитивное замыкание

Для определения свойств матрицы графа используется оператор `IsMatrixShape` пакета `LinearAlgebra`. В параметрах оператора нет транзитивности, поэтому данное свойство описывается в специальной процедуре `trnztv`. Процедура работает точно по определению транзитивности (см. с. 32). Аналогично можно ввести проверку всех остальных свойств отношения.

В качестве примера рассмотрен граф с рис. 2.5 (см. с. 36). Первоначальная проверка показывает, что отношение, соответствующее графу, не транзитивно. После применения алгоритма Уоршолла получаем транзитивное замыкание. Для работы алгоритма требуется матрица, элементами которой являются логические константы `true` и `false`. Простая подстановка `subs(1=true,0=false,A)` легко с этим справляется. После получения искомой матрицы необходимо выполнить обратную замену: `subs(true=1,false=0,A)`.

Программа 9

```
> restart: with(LinearAlgebra): n:=4:
> A:=Matrix([[1,0,0,1],[1,0,0,0],[0,0,0,0],
>                                    [0,0,1,1]]);
> `IsMatrixShape/trnztv` := proc(A)
> local i,j,m,S:
> S:=true:
> for i to n do
>   for j to n do
```

```
>     for m to n do
>       if(A[i,m]*A[m,j]=1) then
>                             S:= S and is(A[i,j]=1);fi;
>     od;
>   od;
> od:
> end:
> IsMatrixShape(A,trnztv); # Проверка транзитивности
```

$$A := \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$false$$

```
> if IsMatrixShape(A,symmetric) then
> print("Неограф") else print("Оргграф"); fi;
```

"Оргграф"

```
> if Trace(A)=n then
>      print("Граф рефлексивного отношения");fi;
> if Trace(A)=0 then
>      print("Граф антирефлексивного отношения");fi;
> if IsMatrixShape(A,zero) then
>      print("Пустой граф"); fi;
> A:=subs(1=true,0=false,A);
```

$$A := \begin{bmatrix} true & false & false & true \\ true & false & false & false \\ false & false & false & false \\ false & false & true & true \end{bmatrix}$$

```
> for m to n do # Алгоритм Уоршелла
>   for i to n do
>     for j to n do
>       A[i,j]:=A[i,j] or (A[i,m] and  A[m,j]);
>     od;
>   od;
> od;
> A:=subs(true=1,false=0,A);
> IsMatrixShape(A,trnztv); # Проверка транзитивности
```

$$A := \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

true

5.8. Компоненты сильной связности графа

Приведем три варианта решения задачи о нахождении компонент сильной связности графа. Рассмотрим орграф (рис. 5.1), содержащий, очевидно, три компоненты сильной связности: $(1,2)$, $(3,4)$, $(5,6)$. Дуги задаются списком, например $[1,2]$, а не множеством, как в неографе: {1, 2}. К сожалению, в Maple рисунки орграфа и неографа не отличаются (дуги стрелками не выделяются). Поэтому рисунок, полученный оператором `draw(G)`, непохож на рис. 5.1 (дуги туда и обратно сливаются в одну). В программе определяется матрица достижимости `M`, несимметричная для орграфа. Строки матрицы соответствуют вершинам, от которых идет путь, а столбцы — вершинам, в которые путь приходит. Рассматривая матрицу `M`, замечаем, что из вершин 1 и 2 достижимы все вершины графа (строки 1 и 2 не содержат нулей), сами же вершины 1 и 2 достижимы из множества {1, 2}. Пересечением множеств {1, 2, 3, 4, 5, 6} и {1, 2} является множество {1, 2}. Это множество и является компонентой сильной связности графа. Операция пересечения множеств эквивалентна замене элементов матрицы достижимости произведением элементов и их симметричных образов. Это и выполнено в программе в двойном цикле по `i` и `j`. Начальные значения переменных цикла `from` и шаг `by`, по умолчанию равные 1, по обыкновению, не пишутся. В полученной матрице в последнюю строку дописываются номера вершин. Следует отметить характерную для Maple конструкцию определения и одновременного вызова функции, создающей последовательность номеров вершин: `(j) -> j`. Ориентация вектора номеров создается опцией `row` [1]. Далее, просматривая первую строку получившейся матрицы, заносим в первый список связности номера столбцов с ненулевыми элементами и вычеркиваем соответствующие строки и столбцы. Эта операция повторяется в цикле по `j` до исчерпания матрицы. Используются операторы `DeleteColumn` и `DeleteRow` из пакета `LinearAlgebra`. Эти операторы не могут вычеркивать последние строку и столбец из матрицы, поэтому в программу введен условный оператор `if nV1<>n`.

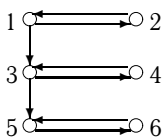
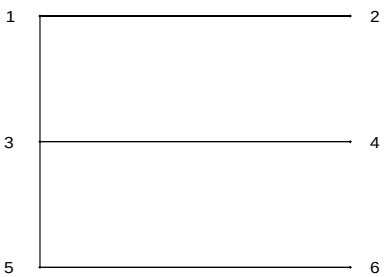


Рис. 5.1

[1] Столбец задается опцией `column`.

Программа 10

```
> restart: with(networks):with(LinearAlgebra):
> n:=6: G:=void(n):                    # Пустой граф
> addedge([[1,2],[2,1],[3,4],[4,3],
> [5,6],[6,5],[1,3],[3,5]],G):         # Дуги
> m:=nops(edges(G)):                   # Число дуг
> draw(Linear([5,3,1],[6,4,2]),G); # Рисунок
```




```
> A:=adjacency(G);                     # Матрица смежности
```

$$A := \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

```
> M:=convert(evalm(add(A^k,k=1..m)),Matrix);
```

$$M := \begin{bmatrix} 4 & 4 & 10 & 10 & 20 & 10 \\ 4 & 4 & 10 & 6 & 10 & 10 \\ 0 & 0 & 4 & 4 & 10 & 10 \\ 0 & 0 & 4 & 4 & 10 & 6 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 4 & 4 \end{bmatrix}$$

```
> for i to n do
>   for j to n do
>    M[i,j]:=M[i,j]*M[j,i];
>   od:
> od:
> ###################################################
```

```
> NM:=Vector[row](n,(j) -> j):# Номера вершин
> M:= << M>,<NM>>; # Номера - в последнюю строку
```

$$M := \begin{bmatrix} 16 & 16 & 0 & 0 & 0 & 0 \\ 64 & 16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 16 & 16 & 0 & 0 \\ 0 & 0 & 64 & 16 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16 & 16 \\ 0 & 0 & 0 & 0 & 64 & 16 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

```
> for j while nV1<>n  do
>   n:=Dimension(M)[2];
>   KMP[j]:={}:          # Компонента связности j
>   V1:={}:  # Номера исключаемых строк и столбцов
>   for i to n do
>     if M[1,i]<>0 then V1:=V1 union {i}:
>     KMP[j]:=KMP[j] union {M[n+1,i]} end;
>   od;
>   V1:=convert(V1,list):
>   nV1:=nops(V1):
>   if nV1<>n then
>   M:=DeleteRow(DeleteColumn(M,V1),V1);fi;
> od:
> ТаблицаКомпонент=op(KMP);
```

$$\text{ТаблицаКомпонент} = ([1 = \{1, 2\}, 2 = \{3, 4\}, 3 = \{5, 6\}])$$

Более простой вариант программы определения компонент сильной связности орграфа основан на функции `components`, применяемой для нахождения компонент связности неографа. Первая часть программы, где вводится орграф, создается матрица достижимости и перемножаются ее симметричные элементы, берется из предыдущей программы. Приведем только те строки, которые идут после закомментированной строки `########`.

Рассмотрим граф с рис. 2.8 (см. с. 38). Список дуг для этого графа имеет вид `addedge([[1,5],[5,3],[3,1],[2,4],[4,6],[6,2],[2,3]],G)`. Матрица `M` определяет неограф `G1`, в котором связность соответствует сильной связности исходного графа. Список ребер создается в двойном цикле при обходе верхней треугольной части матрицы `M`. В отличие от орграфа, ребра теперь представляют собой множества и обозначаются как множества, т.е. фигурными скобками. Далее создается неограф, и остается только воспользоваться функцией `components` для получения результата.

Программа 11

```
> ########################################
> E:={}:
> for i to n do
>    for j from i+1 to n do
>    if M[i,j]<>0 then E:=E union {{i,j}};end;
>    od:
> od:
> G1:=void(n): addedge(E,G1): # Новый неграф
> Компоненты=components(G1);
> ЧислоКомпонент=nops(components(G1));
```

$$\text{Компоненты} = \{\{1, 3, 5\}, \{2, 4, 6\}\}$$

$$\text{ЧислоКомпонент} = 2$$

Еще один способ выявления компонент сильной связности годится для орграфов, у которых число ребер основания совпадает с числом дуг. В пакете `networks` имеется оператор `bicomponents` блоков неографа с выделением мостов между ними. Для того чтобы воспользоваться им, необходимо получить основание графа. Сделать это можно, преобразовав дуги (*списки* вершин) в ребра (*множества* вершин): `E1 := {seq({op(ends(edges(G)[i],G))},i=1..m)}`, где `m` — число дуг исходного графа `G`. Затем обычным образом получаем граф `G1` с этим списком ребер и применяем к нему оператор `bicomponents(G1)`. Для графа (рис. 5.2), представленного дугами `[[1,2],[2,4],[4,1],[3,6],[6,5],[5,3],[1,3]]`, ответ имеет вид `[{e1}, {{e2, e3, e4}, {e5, e6, e7}}]`. При этом в ответе записаны ребра основания: `{{1, 3}, {2, 4}, {1, 2}, {1, 4}, {3, 6}, {5, 6}, {3, 5}}`. Отдельно выделен мост `{e1}` от вершины 3 к вершине 1.

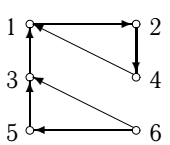


Рис. 5.2

5.9. Пути в орграфе

Для того чтобы вычислить число маршрутов длины 3, необходимо обычным образом задать граф, подключив пакет `networks`, найти его матрицу смежности (`A:=adjacency(G)`), возвести ее в третью степень (`M:=A^3`) и просуммировать все элементы полученной матрицы: `add(add(M[i,j],i=1..n),j=1..n)`.

5.10. Изображение орграфа

Недостатком процедуры `draw` системы Maple [1] является то, что орграфы изображаются, как неографы. Кроме того, подписываются только вершины, а ребра (дуги) остаются безымянными, что затрудняет чтение рисунка. Приведем программу для рисования орграфов с пометкой дуг. Оператором `save` программа записывается на диск в файл `draw.m`. Вызвать программу можно оператором `read`. Для удобства пользования лучше указывать абсолютный адрес файла, тогда Maple может его найти и в том случае, когда файл запускается не из текущей директории. Отметим также, что в Maple не предусмотрено раздельное пользование файлами при чтении, поэтому две программы не могут одновременно пользоваться файлом `draw.m`. Для рисования вершин использован оператор `disk`, рисующий круг и позволяющий использовать заливку цветом. Если применить `circle`, то получится окружность. Стиль стрелок записан в один параметр `q`. Этот параметр регулирует геометрию и цвет стрелок и может быть изменен. Для упрощения программы многие линейные размеры взяты в абсолютных величинах, поэтому для каждого случая использования программы необходимо вручную подбирать некоторые значения. Например, изображения номеров вершин сдвинуты на 4 единицы по оси x и на единицу по оси y: `[x[k]+4,y[k]+1,r[i][j]]`, а номера дуг — на единицу по оси y вверх: `(y1+y2)/2+1`.

Программа 12

```
> restart;
> Draw1:=proc(S,G)
> local q,k,n,n1,m,m1,i,j,t,v,x,y,x1,x2,y1,y2,c,e,E;
> n1:=nops(S);           # Число столбцов вершин
> n:=nops(vertices(G)): # Число вершин графа
> E:=ends(G):           # Список концов дуг
> m1:=nops(E):          # Число дуг
> q:=.5, 2, .05, color=green: # Стиль стрелок
> for i to n1 do
>  m:=nops(S[i]); # Число вершин в столбце i
>   for j to m do
>   k:=S[i][j]:   # Номер вершины
>   x[k]:=100/n1*i; y[k]:=100/(m+1)*j; # Координаты
>   c[k]:=disk([x[k],y[k]],0.5,color=blue): # Вершины
```

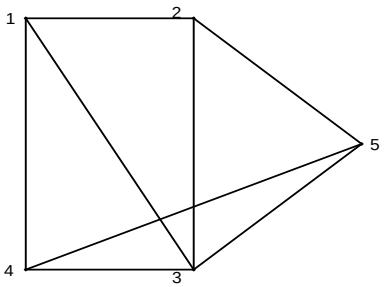
[1] Версии 8, 9, 9.5 и 10.

```
>    t[k]:=textplot([x[k]+4,y[k]+1,r[i][j]], # Подписи
>    align={ABOVE,RIGHT},font=[TIMES,ITALIC,16]);
>   od;
>  od;
>  for i to m1 do
>    x1:=x[E[i][1]]: y1:=y[E[i][1]]: # Координаты
>    x2:=x[E[i][2]]: y2:=y[E[i][2]]:
>    v[i]:=arrow([x1,y1],[x2,y2],q): # Дуги
>    e[i]:=textplot([(x1+x2)/2,(y1+y2)/2+1,e||i],
>    align={ABOVE,RIGHT},# Подписи дуг
>    font=[HELVETICA,OBLIQUE,12],color=red);
>  od;
>  display(seq(c[i],i=1..n),seq(t[i],i=1..n),
>          seq(e[i],i=1..m1),seq(v[i],i=1..m1),
>          axes=none,scaling=constrained);
>  end proc:
>  save Draw1, "C:\\SBDISKR\\MAPLE\\draw.m";
```

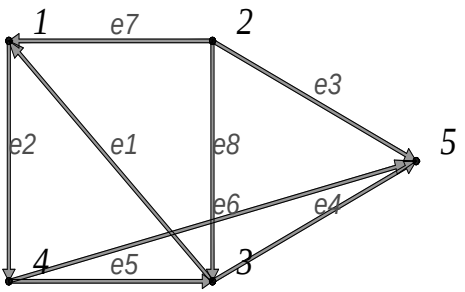

В следующей программе приводится пример использования разработанной процедуры рисования орграфа. В качестве оригинала берется граф с рис. 2.2 (см. с. 31). Сначала для сравнения дается стандартное изображение, встроенное в пакет `networks` с линейной опцией вывода `Linear`, предусматривающей расположение вершин столбцами слева направо.

Программа 13

```
>  restart;
>  with(plots): with(plottools): with(networks):
>  G:=void(5):          # Пустой граф
>  E:=[[1,4],[2,1],[2,3],[3,1],[2,5], # Дуги
>      [3,5],[4,3],[4,5]]:
>  addedge(E,G): r:=[4,1],[3,2],[5]:
>  draw(Linear(r),G);  # Стандартный рисунок
```




```
> read "C:\\SBDISKR\\MAPLE\\draw.m";
> Draw1([r],G); # Рисунок орграфа
```



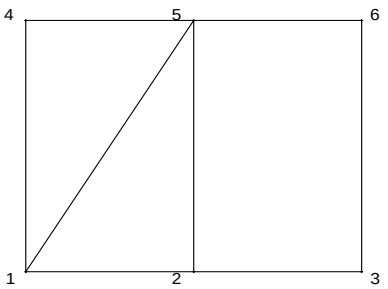
5.11. Кратчайший путь в орграфе

5.11.1. Операторы shortpathtree и allpairs. Рассмотрим граф с рис. 4.2 (см. с. 58). Вычислим кратчайший путь в графе, пользуясь стандартными процедурами Maple. В программе 14 независимо используются оба оператора, `shortpathtree` и `allpairs`.

Программа 14

```
> restart: with(networks):
> new(G):  n:=6:
> addvertex(i$i=1..n,G); # Вершины исходного графа
                          1, 2, 3, 4, 5, 6
> addedge([seq([i,i+3],i=1..3),[1,2],[2,3],[4,5],
```

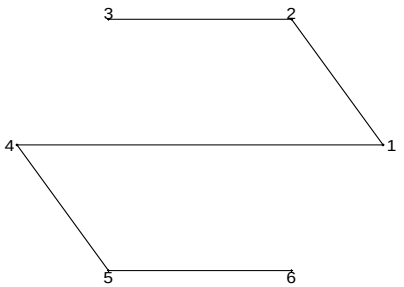
```
> [5,6],[1,5]],weights=[12,16,20,11,15,13,14,26],G):
> draw(Linear([1,4],[2,5],[3,6]),G);
```




```
> T:=shortpathtree(G,1): # Дерево минимальных путей
> W:=vweight(T);  # Таблица весов (расстояний)
```

$$W := \mathrm{table}(sparse, [1 = 0,\ 2 = 11,\ 3 = 26,\ 4 = 12,\ 5 = 25,\ 6 = 39])$$

```
> draw(T);        # Рисунок дерева
```




```
> MinPath:=W[6];   # Ответ. Минимальный путь 1 - 6
```

$$MinPath := 39$$

```
> allpairs(G)[1,6]; # Другой способ решения задачи
```

$$39$$

5.11.2. Вычисление минимальных путей. В прикладных задачах на графах широко применяется вычисление кратчайших путей методом Дейкстры. Алгоритм вычисления приведен на с. 58. В следующей программе реализован этот алгоритм. Ответ заносится в переменную `MinPath`. Алгоритм заканчивает свою работу, когда `flag` принимает

значение `true`, т.е. конечная вершина (target) приобретает постоянную метку. В качестве примера взят граф с рис. 4.2 (см. с. 58).

После завершения работы программы список постоянных меток можно посмотреть, раскрыв переменную V: `evalm(V)`. Список совпадает с результатом, полученным вручную на с. 60.

Программа 15

```
> restart:with(networks):
> new(G):n:=6:
> addvertex(i$i=1..n,G);
                        1, 2, 3, 4, 5, 6
> addedge([seq([i,i+3],i=1..3),[1,2],[2,3],[4,5],
> [5,6],[1,5]],weights=[12,16,20,11,15,13,14,26],G):
> V:=Vector(1..n):
> for i to n do V[i]:=infinity; od:
> s:=1: target:=6: k:=s: V[k]:=0: U:=[0$3*n]:
> flag:=false:
> for i  while not flag  do
>   U[i]:=k:
>   d:=outdegree(k,G):
>   z:=departures(k,G):
>   for j to d do
>     CW1:=eweight(op(edges([k,z[j]],G)),G):
>     if((V[z[j]]=0) or (V[z[j]]>CW1+V[k])) then
>     V[z[j]]:=eweight(op(edges([k,z[j]],G)),G)+V[k]:
>     fi;
>   end do;
>   Next:=n;
>   for j from 2 to n do
>     if not member(j,U) and V[j]<V[Next] then
>     Next:=j;
>     fi;
>   end do:
>   k:=Next;
>   flag:=is(k=target);
> end do:
> evalm(V);
                      [0, 11, 26, 12, 25, 39]
```

```
>    MinPath:=V[6];
```

$$MinPath := 39$$

5.12. Центроид дерева

Пакет `networks` не содержит оператора определения центроида дерева. Напишем специальную программу, рассмотрев в качестве примера граф с рис. 3.2 (см. с. 42). Для иллюстрации всех возможностей `networks` используем еще один способ ввода данных о графе [1]. Применим оператор `void` создания пустого графа с последующим добавлением ребер в виде отдельных ребер или целых ветвей (`Path`). Оператор `nops(edges(G))` задействован для контроля числа введенных ребер. Для дерева число ребер на единицу меньше числа вершин: `m = n - 1`.

В цикле длиной n по числу вершин происходит последовательное их удаление из копии графа. При удалении каждой вершины граф распадается на отдельные деревья (лес). Так как вместе с вершиной удаляется и ребро, число ребер (`nops(edges(induce(K[j],G)))`) в каждом получившемся дереве на единицу меньше реальной длины соответствующей ветви. Число таких деревьев равно степени вершины, вычисленной оператором `nops(K)`.

Программа 16

```
>  restart; with(networks):
>  n:=16:
>  G:=void(n):  # Пустой граф
>  addedges({{13,14},{11,12},{15,16},{6,7},
>  {7,8},{8,4}},G):
>  addedges(Path(1,5,9,10,14,15,11,7,3,2),G):
>  nops(edges(G)):     # Число ребер
>  r:=seq([seq(1+j+4*i,i=0..3)],j=0..3):
>  draw(Linear(r),G):  # Рисунок
>  DL:=maxdegree(G):   # Максимальная степень вершин
>  V:=Vector(DL,[]):
>  for i to n do #### Цикл для расчета весов вершин
>    H:=duplicate(G):   # Удаление вершин из копии G
>    H:=delete(i,H):
>    Di:=vdegree(i,G);  # Степень вершины
```

[1] Два других способа использованы ранее — оператор `new` (см. с. 111) и оператор `graph` (см. с. 99).

```
>    K:=components(H); # Список вершин каждой ветви
>    for j to nops(K) do
>     V[j]:=nops(edges(induce(K[j],G)))+1:# Вес ветви
>     od:
>    W[i]:=max(seq(V[k],k=1..Di)); # Вес вершины
>  od: #### Конец цикла для расчета весов вершин
>  W:=convert(W,array); # Массив весов
```

$$W := [15, 15, 14, 15, 14, 15, 10, 14, 13, 12, 8, 15, 15, 10, 8, 15]$$

Построение центроида

```
>  minW:=min(seq(W[i],i=1..n)); # Наименьший вес
```

$$minW := 10$$

```
>  print("Центроид состоит из следующих вершин:"):
>  C:={}:
>  for i to n do
>    if W[i]=minW then C:=C union {i}; fi;
>  od:
>  print(C);
```

Центроид состоит из следующих вершин:

$$\{11, 15\}$$

5.13. Десятичная кодировка

Программа 17 написана в соответствии с инструкцией кодировки, данной на с. 42. Так как в дереве с n вершинами имеется $n - 1$ ребро, а каждое ребро в процессе кодировки проходится дважды (кодируясь единицей, а затем нулем), число цифр в коде `n2=2*n-2`. В цикле с ограничением на число цифр в коде содержатся две части. Сначала из вершины, помеченной как корень дерева, осуществляется поиск в глубину вплоть до листа, у которого степень `vd1` равна 1 (висячая вершина). Правильный выбор пути на развилках осуществляется благодаря применению оператора `min`. В процессе хода в переменной `S` накапливается множество пройденных вершин и заносятся единицы в список `KOD`. При завершении хода к висячей вершине ее номер сохраняется в переменной `v1`. С этой вершины начинается ход назад с пометкой нулями и удалением всех пройденных ребер, что предупреждает возможность повторного прохождения какого-либо ребра. Движение назад возможно только по тем вершинам, которые уже были пройдены в первой части. За это отвечает дополнительное условие `nops(v1 intersect S)<>0` в условном операторе. Оператор `intersect` вычисляет пересечение множеств. Если возможность для

движения назад исчерпана, переменной `v1` присваивается значение той вершины, откуда была осуществлена последняя попытка найти смежную с ней вершину с подходящими условиями, и цикл досрочно прерывается оператором `break`.

В конце программы полученный двоичный код, имеющий тип `table`, конвертируется в список, а затем в десятичное число.

Аналогичную программу, использующую оператор определения матрицы расстояний `allpairs`, можно найти на сайте http://vuz.exponenta.ru.

В программе 17 рассмотрено дерево с рис. 3.9 (см. с. 45).

Программа 17

```
> restart:with(networks):
> new(G): n:=9:  V:={$1..n}:
> n2:=2*n-2: # Число цифр в коде
> addvertex(V,G):
> addedge(Path(7,4,1,2,3,6),G):
> addedge(Path(2,5,8),G):
> addedge({5,9},G):
> draw(Linear([1,4,7],[2,5,8],[3,6,9]),G):
> v1:=3:    # Корень
> V0:={v1}: # Множество вершин
> i:=0:     # Счетчик элементов кода
> S:={v1}:  # Множество пройденных вершин
> while i<n2 do
  Движение от корня, кодировка единицами
> vd1:=0:
> while vd1<>1 do
>   v1:=min(op(neighbors(v1,G) minus S));
>   vd1:=vdegree(v1,G):
>   i:=i+1;  KOD[i]:=1;
>   S:=S union {v1};
> od:
  Возвращение назад, кодировка нулями
> V0:={v1}:
> while nops(v1)=1 do
>   v10:=v1:
>   v1:=neighbors(v1,G) minus V0;
>   if nops(v1)=1 and nops(v1 intersect S)<>0
```

```
>       then
>       V0:=V0 union {v1[1]}:
>       i:=i+1: KOD[i]:=0:
>       delete(edges({op(v10),op(v1)},G),G):
>       else v1:=v10: break;
>     fi;
>   od;
>   v1:=op(v1);# Начальная вершина
>   od:# Конец цикла i
>   Z2:=convert(KOD,list);
```

$$Z2 := [1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0]$$

Перевод в десятичную систему

```
>   Z10:=add(Z2[i]*2^(n2-i),i=1..n2);
```

$$Z10 := 61858$$

5.14. Распаковка десятичного кода

Сначала десятичный код преобразуется в список нулей и единиц, соответствующий двоичной записи числа справа налево. Затем вычисляется длина списка и список обращается, превращаясь в привычную запись двоичного числа. Переменная `Z` имеет тип последовательности `exprseq` [1].

В процессе работы цикла по всем элементам последовательности различаются два случая: текущий элемент равен 1 или 0. В случае, когда элемент равен 1, в множество ребер графа добавляется ребро `{v1,v2}`, где номер `v2` всякий раз увеличивается, а `v1` принимает каждый следующий раз значение `v2` либо вычисляется при встрече нуля (или нулей) в ряду элементов. При движении вперед, когда список ребер растет, растет и временный список пройденных вершин `F`. При движении назад (к корню) список ребер не меняется, а список пройденных вершин `F` уменьшается посредством удаления последнего элемента. Функцию удаления выполняет присваивание `F:=F[1..-2]` с перечислением элементов от первого до предпоследнего. Последний элемент списка длиной `n` имеет вид `F[n]` или `F[-1]`. Последняя запись удобна в том случае, когда неизвестна длина списка.

Полученный список ребер вносится обычным образом в граф, изображение графа выводится на экран. Как видно из нашего примера, изображение отличается от исходного (см. рис. 3.9 на с. 45). Здесь

[1] Узнать тип переменной можно с помощью оператора `whattype`, например `whattype(Z)`. Проверить тип можно логическим оператором `type(Z,exprseq)`.

проявляется особенность алгоритма десятичной кодировки, в которой не сохраняется нумерация вершин. Для сохранения нумерации вершин дерева необходима другая кодировка, менее компактная и принципиально почти не отличающаяся от десятичной. В такой кодировке вместо единиц нужно ставить номера вершин.

Программа 18

```
> restart:with(networks):
```

Десятичный код

```
> KOD10:=61858:
> Z:=convert(KOD10,base,2):
> n2:=nops(Z):
```

Двоичный код

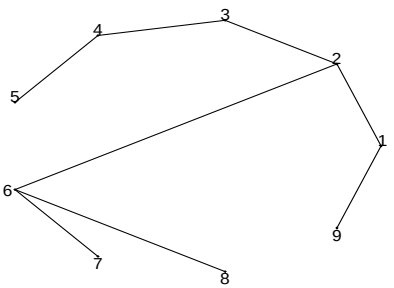
```
> Z:=seq(Z[n2-i+1],i=1..n2);
```

$$Z := 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0$$

```
> E1:={}: # Множество ребер
> v1:=1: v2:=1:
> F:=[1]: # Переменный список вершин
> for i to n2 do
>   if Z[i]=1 then
>     v2:=v2+1:
>     F:=[op(F),v2];
>     E1:=E1 union {{v1,v2}};
>     v1:=v2:
>   fi;
>   if Z[i]=0 then
>     F:=F[1..-2]: # Стираем последний элемент
>     v1:=F[-1];   # Последний элемент списка F
>   fi;
> od;
> E1;
```

$$\{\{1, 2\}, \{2, 3\}, \{2, 6\}, \{3, 4\}, \{4, 5\}, \{6, 7\}, \{6, 8\}, \{1, 9\}\}$$

```
> new(G):
> n:=n2/2+1:  V:={$1..n}:
> addvertex(V,G): addedge(E1,G): draw(G);
```



5.15. Кодировка Прюфера

Условие задачи дано на с. 47. В начале программы формируется заданное дерево (см. рис. 2.11). Особенностью здесь является применение опции `Path` в операторе `addedge`, которая позволяет задавать целые ветви дерева. В параметре `r` формируется список из вертикальных списков вершин, прорисовываемых оператором `draw` с опцией `Linear` линейного рисования.

Код формируется точно по алгоритму. Сначала ищется висячая вершина (вершина с наименьшей степенью в дереве всегда висячая), затем в код `T` записывается номер вершины, смежной с ней (`departures`), а сама вершина удаляется оператором `delete`. Заметим, что скобки для записи конечного значения `(n-2)` переменной цикла в Maple обязательны.

Программа 19

```
> restart;with(networks): new(G):n:=16:
> addvertex($ 1..n,G):                  # Вершины
> addedge(Path(1,2,3,4,8,12,11),G): # Ветви
> addedge(Path(2,6,5,9,13),G):
> addedge(Path(6,10,14,15,16),G):
> addedge({3,7},G):
> r:=seq([seq(1+j+4*i,i=0..3)],j=0..3):
> draw(Linear(r),G):
> T:=[0$(n-2)]:
> for i to (n-2) do
>   mindegree(G,sm[i]):
```

```
>     z:=sm[i]:
>     z1:=departures(z,G):
>     T[i]:=z1[1]:
>     delete({z},G):
>  end do:
>  "Код Прюфера:", T;
```

Код Прюфера:, [2, 3, 12, 8, 4, 3, 2, 6, 9, 5, 6, 10, 14, 15]

Следующая программа служит тем же целям, но обходится без оператора `departures`, используя проверку соответствующего элемента матрицы смежности `A[i,j]=1`. Вместо оператора `mindegree` здесь используется условие `vdegree(i,G)=1`. В программе такой же ввод данных, что и в предыдущей, и записывается она после оператора `draw(Linear(r),G)`.

Программа 20

```
>  k:=0: # Счетчик элементов кода
>  for i  while k<n-2 do
>    A:=adjacency(G):# Матрица смежности
>    if vdegree(i,G)=1 then
>    k:=k+1;
>    for j to n do if A[i,j]=1 then T[k]:=j; fi; od;
>    delete(incident(i,G),G);i:=0;
>    end if;
>  end do:
>  PruferCode:=evalm(Vector(k,T)):
```

5.16. Распаковка кода Прюфера

По заданному коду Прюфера (см. с. 47) программа определяет множество ребер и рисует закодированное дерево.

Сначала оператор `nops` вычисляет число вершин (их на 2 больше числа элементов кода), затем, в соответствии с алгоритмом, в цикле по `i` производится набор ребер. Последнее ребро составляется из двух чисел (номеров вершин), оставшихся во вспомогательном множестве `N`, состоящем изначально из всех вершин дерева. Запись `P:=P[2..-1]` означает, что список `P` укорачивается на один первый элемент. Минус один в качестве последнего элемента в перечислении означает конец списка.

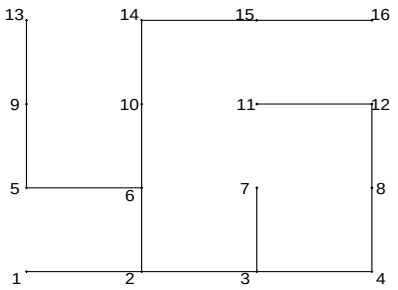
Отдельно рассмотрим строку `x1:=N minus convert(P,set)`. Код Прюфера не является множеством, да и не может быть им, так как в нем есть повторяющиеся элементы. Поэтому перед выполнением вычитания `minus` производится конвертация. Другой вариант этой строки имеет вид `x1:=select(sq,N)`, где введена пользовательская функция проверки наличия аргумента функции в `P`:

```
sq:=x->is(not member(x,T)).
```

Функция `select` производит выбор из `N` всех тех элементов, которые удовлетворяют этому условию. Наблюдается некоторая неестественность использования функции «без аргумента», однако это характерно для Maple в таких операциях.

Программа 21

```
> restart: with(networks):
> P:=[2,3,12,8,4,3,2,6,9,5,6,10,14,15]:# Код Прюфера
> n:=nops(P)+2:# Число вершин графа
> N:={$ 1..n}: # Вспомогательное множество
> i:=1: E:={}: # Пустое множество ребер
> while i<(n-1) do
>    x1:=N minus convert(P,set);
>    E:=E union {{P[1],x1[1]}}:
>    N:=N minus {x1[1]};# Операция N*
>    P:=P[2..-1];       # Операция P*
>    i:=i+1;
> end do:
> E:=E union {N}:
> new(G):
> n:=16:
> r:=seq([seq(1+j+4*i,i=0..3)],j=0..3):
> addvertex($ 1..n,G):# Вершины
> addedge(E,G):       # Ветви
> draw(Linear(r),G);  # Рисунок
```



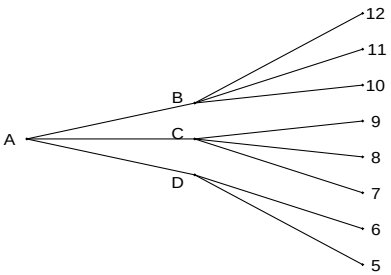
5.17. Код Гапта

В следующей программе решается задача составления кода Гапта для дерева с рис. 3.22 (см. с. 54). Граф задается обычным образом: сначала создается список вершин, затем в операторе `connect` указываются соединения. Для того чтобы не повторять список вершин-листьев, используется обозначение `v:=$5..12`. Некоторой особенностью ввода данных является обозначение части вершин буквами, а части — цифрами. Это вполне допустимо. Более того, буквы в Maple могут выполнять роль идентификаторов, им можно присваивать значения, в том числе даже слова. Например, если в начале программы, сразу после `restart`, задать `A:='корень'`, то программа не воспримет это как ошибку; ответ будет тот же, а на рисунке вершина `A` будет подписана.

Оператор `shortpathtree` алгоритма Дейкстры используется здесь не совсем по назначению. Он только превращает граф в корневое дерево с корнем в вершине `A`. Основой программы являются оператор `daughter`, определяющий список сыновей вершины дерева, и оператор `nops`, вычисляющий длину `z` списка. Эта длина и записывается слева в код Гапта: `kod:=z,kod`.

Программа 22

```
> restart:with(networks):new(G):
> v:=$5..12:
> addvertex(A,B,C,D,v,G):
> connect(A,{B,C,D},G): connect(D,{5,6},G):
> connect(C,{7,8,9},G): connect(B,{10,11,12},G):
> draw(Linear([A],[D,C,B],[v]),G);
```




```
> T := shortpathtree(G,A): z:=A: ss:={A}: # Корень
> kod:=``:                  # "Пустой" код Гапта
> for k while (z<>0) do a:=ss: ss:={}:
>  for i in a do
>   z:=nops(daughter(i,T)):# Число сыновей вершины i
>   if z<>0 then kod:=z,kod; fi:
>   ss:=ss union daughter(i,T)
>  od;
> od:
> kod;                      # Ответ. Код Гапта
                          2, 3, 3, 3,
```

5.18. Распаковка кода Гапта

Для распаковки кода Гапта берется код [3, 1, 1, 1, 1, 4, 2, 1, 2, 3, 3, 3] некоторого неизвестного дерева.

Длину вектора кода определяет оператор `Dimension` [1] пакета `LinearAlgebra`. Он дает значение 12. Использовав оператор `nops`, получаем 4 — мощность множества. Сначала программа дает список ребер `rb` и список вершин `vr`, а затем рисует получившееся дерево. При использовании стандартного линейного изображения с опцией `Linear` изображение получается повернутым на 90°. При необходимости изображение можно повернуть. Для этого в Maple есть оператор `rotate` пакета `plottools`. Если не указывать центр вращения, то достаточно

[1] Этот оператор можно применять и к вектору, и к матрице. В последнем случае он дает число строк и число столбцов. Имеются также операторы вычисления числа строк (`RowDimension`) и столбцов (`ColumnDimension`).
.

записать `rotate(draw(Linear(vr),G),-Pi/2)`. Текстовые подписи при этом не повернутся, т.е. изображение не поворачивается как целое, просто меняются направления осей.

Программа 23

```
> restart;with(LinearAlgebra):with(networks):
> kod:=<3,1,1,1,1,4,2,1,2,3,3,3>:  # Код Гапта
> n1:=Dimension(kod):          # Длина вектора кода
> n:=add(kod[i],i=1..n1)+1: # Число вершин дерева
> new(G):                  # Создание нового графа
> addvertex(seq(i,i=1..n),G):# Добавление  n вершин
> rb:=seq([1,1+i],i=1..kod[n1]):  # Список ребер
> j:=1: t:=1: s:=1:
> for m from 0 while n1-s-j+1>0 do
>   j:=s+j-1:
>   for s to kod[n1-m] do
>    t1:=seq(kod[n1]+i+t,i=1..kod[n1-s-j+1]);
      Увеличение списка ребер
>    rb:=rb,seq([j+s,kod[n1]+i+t],i=1..kod[n1-s-j+1]);
>    t:=t+kod[n1-s-j+1];
>   od:
> od:
> rb; # Ответ. Список ребер
```

[1, 2], [1, 3], [1, 4], [2, 5], [2, 6], [2, 7], [3, 8], [3, 9], [3, 10], [4, 11], [4, 12], [5, 13], [6, 14], [6, 15], [7, 16], [7, 17], [7, 18], [7, 19], [8, 20], [9, 21], [10, 22], [11, 23], [12, 24], [12, 25], [12, 26]

Список вершин для рисунка

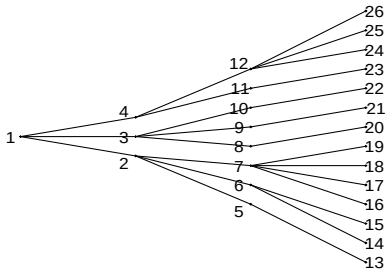
```
> vr:=[1]:
> c[1]:=1: # Одна вершина в ярусе 1
> t:=0: h:=1:
> for z while n1-t>0 do
>    t:=t+c[z]:
```

Число вершин яруса

```
>    c[z+1]:=add(kod[n1-t+i],i=1..c[z]);
>    h:=h+c[z+1]; vr:=vr,[seq(j,j=h+1-c[z+1]..h)];
> od:
> vr;       # Ответ. Список вершин
```

[1], [2, 3, 4], [5, 6, 7, 8, 9, 10, 11, 12],
[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]

```
> addedge([rb],G):    # Добавление к графу ребер
> draw(Linear(vr),G); # Рисунок графа
```



5.19. Поток в сети

В программе одновременно даны два способа вычисления максимального потока в сети (см. рис. 4.4 на с. 62). Оператор `flow(G,v1,v2,edgsatur)` из пакета `networks` возвращает наибольший поток по сети от источника `v1` в сток `v2`, а в переменную `edgsatur` помещает список насыщенных ребер [1].

Программа 24

```
> restart: with(networks):
> new(G):V:=$1..8: addvertex([V],G):
> v1:=1:# Источник
> v2:=8:# Сток
> E:=[[1,3],[3,5],[5,7],[7,8],[1,2],[2,4],[4,6],
```

[1] В случае орграфов оператор `flow` путает ребра и дуги, давая в ответе лишние ребра. Например, для сети с дугами `[[1,3],[3,5],[5,6],[1,2],[2,4],[4,6],[2,5],[3,4]]`, имеющими одинаковую пропускную способность, равную по умолчанию 1, с истоком в вершине 1 и стоком в 6 (рис. 5.3) оператор дает правильный ответ: максимальный поток равен 2. Однако при этом оператор возвращает семь насыщенных ребер (не дуг): {1, 3}, {2, 4}, {4, 6}, {1, 2}, {2, 5}, {5, 6}, {3, 4}, где ребро {2, 4} явно лишнее.

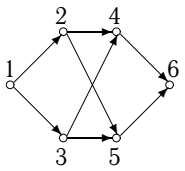


Рис. 5.3

```
> [6,8],[3,2],[2,5],[5,4],[4,7],[7,6]]:
```
Пропускная способность
```
> w:=[6,9,7,4,6,4,4,7,5,2,8,2,11]:
> addedge(E,weights=w,G):
> draw(Linear([1],[3,2],[5,4],[7,6],[8]),G):
> Поток=flow(G,v1,v2,edgsatur);
> edgsatur;                  # Насыщенные дуги
```
$$\text{Поток} = 11$$
$$\{\{1,2\},\{2,5\},\{4,7\},\{7,8\},\{2,4\},\{4,6\},\{6,8\}\}$$
```
> m:=nops(edges(G)):
> H:=duplicate(G):
> potok1:=table([seq(e||i=0,i=1..m)]):# Нач. поток
> while (v1 in vertices(G)) do
> s:=[]: d:={v1}: d2:=v1:
> c1:={v1}: ndep1:=v1:
> while d2<>v2 and ndep1<>0 do
>  d1:=d[1]:              # Начало следующей дуги
>  d:=departures(d1,G):# Множество возможных концов
>  ndep1:=nops(d);
>  if ndep1=0 then delete(d1,G); else
>    d:=d minus c1; # Исключаем пройденные вершины
>    d2:=d[1]:      # Конец дуги
>    nd:=op(edges([d1,d2],G));
>    c1:= c1 union {d2}; # Пополняем список
>    s:=[op(s),nd]:      # Список пройденных дуг
>  fi;
>  od:
> if v2 in c1 then  # Если образовалась цепь
>    n1:=nops(s);   # Длина цепи
>    pt:=[potok1[s[j]]$j=1..n1];
>    sp:=[op(eweight(s,H))];
```
Насыщаем цепь
```
>    potok2:=map(`+`,pt,min(op(sp-pt)));
>    for i to n1 do potok1[s[i]]:=potok2[i];
>     if potok1[s[i]]=eweight(s[i],H) then
>     delete(s[i],G); end;# Удаляем насыщенные дуги
>    od:
> fi:
```

```
>  end:# while
```
Перераспределение
```
>  H2:=duplicate(H):
>  while (v1 in vertices(H2))  do
>  c1:={}:   # Множество пройденных вершин
>  in2:={}:  # Множество входящих дуг
>  out2:={}: # Множество выходящих дуг
>  d1:=v1:   # Первая вершина
>  notupik1:=true;
>  while d1<>v2 and notupik1 do
>  out0:=departures(d1,H2) minus c1;
>  out1:={};
```
Не рассматриваем полные выходящие дуги
```
>  for i in  out0 do
>  nd:=op(edges([d1,i],H2));
>  if eweight(nd,H2)<>potok1[nd] then
>  d2:=i; out1:=edges([d1,i],H2) end;
>  od;
>  out2:=out2 union out1;# Множество прямых дуг цепи
>  in0:=arrivals(d1,H2) minus c1;
>  in1:={};
```
Не рассматриваем пустые входящие дуги
```
>  for i in  in0  do
>    nd:=op(edges([i,d1],H2));
>    if potok1[nd]<>0 then
>    d2:=i; in1:=edges([i,d1],H2); end;
>  od;
>  in2:=in2 union in1;# Множество обратных дуг
>  if nops(in1 union out1)=0 then # Если d1 - тупик
>    delete(d1,H2);   # Удаляем вершину d1
>    notupik1:=false; # Начинаем поиск заново
>    else
```
Присоединяем d1 к пройденным вершинам
```
>    c1:=c1 union {d1};
```
Конец (начало) последней дуги — новая вершина для поиска
```
>    d1:=d2;
>  fi;
>  od;
```

```
> pr1:=(x)->eweight(x,H)-potok1[x]; # Процедура 1
> pr2:=(x)->potok1[x]:              # Процедура 2
> if notupik1 then # Перераспределяем поток
>   m1:=min(op(map(pr1,out2)));
>   m2:=min(op(map(pr2,in2)));
>   ptk:=min(m1,m2);
>   for i in in2 do
>    potok1[i]:=potok1[i]-ptk;
>   od:
>   for i in out2 do
>    potok1[i]:=potok1[i]+ptk; od:
> fi;
> od: # while
> edg2:=incident(v2,H,'In'):# Дуги, входящие в сток
> Поток=map('+',op(eweight([op(edg2)],H)));
> satur1:=[]:
> for x in edges(H) do
>   if pr1(x)=0 then satur1:=[op(satur1),x];fi;
> od;
> satur1;  # Насыщенные дуги
```

Поток = 11

$[e4, e5, e6, e7, e10, e12, e8]$

5.20. Топологическая сортировка сети

Вводятся данные сети с рис. 4.9 (см. с. 65). В алгоритме используется удобная функция `delete`, позволяющая удалять список вершин и соответствующие дуги. Каждый уровень организуется в виде списка для того, чтобы использовать результаты в операторе `Linear` топологически упорядоченной сети. После формирования списка вершин одного уровня эти вершины удаляются из графа. Работа цикла `while NV<>0 do` продолжается до тех пор, пока множество вершин не станет пустым. Счетчик числа вершин — `NV`. Копия графа `H` выводится на экран по полученным уровням.

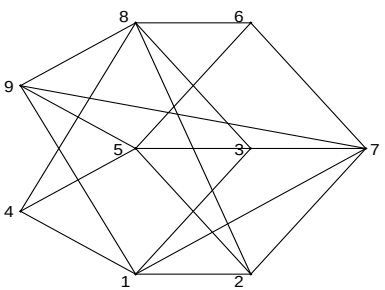
Если граф содержит цикл, то на очередном этапе поиска вершины с нулевой полустепенью захода список уровня окажется пустым и произойдет досрочный выход из цикла с помощью оператора `break`. К графу на рис. 4.9 достаточно добавить, например, дугу `[3,4]`, для того чтобы образовался контур и топологическая сортировка оказалась невозможной.

Программа 25

```
> restart: with(networks):
> new(G):V:=$1..9: addvertex([V],G):
> E:=[[1,2],[1,3],[1,7],[2,7],[3,7],[4,1],[4,5],
> [4,8],[5,2],[5,3],[5,6],[6,7],[8,6],[8,3],[8,2],
> [9,1],[9,5],[9,7],[9,8]]:# Дуги
> addedge(E,G):draw(G):H:=duplicate(G):
> S0:=[]:
> while NV<>0 do
>   S1:=[]:
>   for v in vertices(G) do
>     if indegree(v,G)=0 then
>       S1:=[op(S1),v];
>     fi; # Множество вершин одного уровня
>   od;
>   if nops(S1)=0 then print("Сеть содержит контур");
>     break
>   fi;
>   S0:=[op(S0),S1]; # Добавляем уровень
>   delete(S1,G):    # Удаляем вершины
>   NV:=nops(vertices(G)) # Число вершин
> od:
> S0;
```

$$[[4, 9], [1, 5, 8], [2, 3, 6], [7]]$$

```
> draw(Linear(op(S0)),H);
```



5.21. Паросочетание

Разберем на примере алгоритм решения задачи о паросочетаниях в двудольном графе. Найдем наибольшее паросочетание в графе (рис. 5.4).

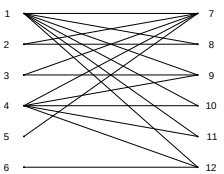


Рис. 5.4

Матрица смежности графа имеет вид

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Выделим отдельные шаги алгоритма.

1. Составим таблицу с размерами матрицы смежности. Пометим недопустимые для паросочетания элементы, проставив какой-либо символ, например звездочку, в тех местах, где стоят нули матрицы. Получим

$$B_1 = \begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline & & * & * & * & * \\ \hline & * & & * & * & * \\ \hline & * & & & & \\ \hline & * & * & * & * & * \\ \hline * & * & * & * & * & \\ \hline \end{array}.$$

2. Двигаясь по матрице B в каком-либо порядке, например слева направо, сверху вниз, проставим на допустимые поля единицы, не допуская более одной единицы в строке и столбце. Так мы получим первый вариант паросочетания. Это будет максимальное (но не обязательно наибольшее) паросочетание:

$$B_2 = \begin{array}{|c|c|c|c|c|c|} \hline 1 & & & & & \\ \hline & 1 & * & * & * & * \\ \hline & * & 1 & * & * & * \\ \hline & * & & 1 & & \\ \hline & * & * & * & * & * \\ \hline * & * & * & * & * & 1 \\ \hline \end{array}.$$

3. Если во всех строках появились единицы, то задача решена, найдено наибольшее паросочетание. В нашем примере в пятой строке нет единицы. Пометим такие строки в специальном столбце, например справа от матрицы:

$$B_3 = \begin{array}{|c|c|c|c|c|c|}\hline 1 & & & & & \\ \hline & 1 & * & * & * & * \\ \hline & * & 1 & * & * & * \\ \hline & * & & 1 & & \\ \hline & * & * & * & * & * \\ \hline * & * & * & * & * & 1 \\ \hline \end{array} \quad \begin{array}{|c|}\hline \\ \hline \\ \hline \\ \hline * \\ \hline \\ \hline \end{array} .$$

4. В помеченной строке (или строках) найдем незапрещенные места в непомеченных столбцах. В специальной строке (ниже матрицы) поместим в этих столбцах номера помеченной строки. Здесь это 5-я строка, а незапрещенное место находится в первом столбце. Поэтому ставим номер 5 в первый столбец в специальную строку. Первый столбец считается помеченным. Получаем

$$B_4 = \begin{array}{|c|c|c|c|c|c|}\hline 1 & & & & & \\ \hline & 1 & * & * & * & * \\ \hline & * & 1 & * & * & * \\ \hline & * & & 1 & & \\ \hline & * & * & * & * & * \\ \hline * & * & * & * & * & 1 \\ \hline \end{array} \quad \begin{array}{|c|}\hline \\ \hline \\ \hline \\ \hline * \\ \hline \\ \hline \end{array} .$$
$$\begin{array}{cccccc} 5 & & & & & \end{array}$$

5. Ищем единицы в помеченных столбцах. Помечаем (справа) строки, в которых найдены единицы:

$$B_5 = \begin{array}{|c|c|c|c|c|c|}\hline 1 & & & & & \\ \hline & 1 & * & * & * & * \\ \hline & * & 1 & * & * & * \\ \hline & * & & 1 & & \\ \hline & * & * & * & * & * \\ \hline * & * & * & * & * & 1 \\ \hline \end{array} \quad \begin{array}{|c|}\hline * \\ \hline \\ \hline \\ \hline * \\ \hline \\ \hline \end{array} .$$
$$\begin{array}{cccccc} 5 & & & & & \end{array}$$

Первая строка оказалась помеченной. Далее возвращаемся к шагу 4.

4′. В первой (помеченной) строке есть пять незапрещенных (без звездочки) мест в непомеченных столбцах. Проставляем ее номер (т.е. 1) в специальной строке:

$$B_6 = \begin{array}{|c|c|c|c|c|c|c|}\hline 1 & & & & & & * \\ \hline & 1 & * & * & * & * & \\ \hline & * & 1 & * & * & * & \\ \hline & * & & 1 & & & \\ \hline & * & * & * & * & * & * \\ \hline * & * & * & * & * & 1 & \\ \hline 5 & 1 & 1 & 1 & 1 & 1 & \\ \end{array} .$$

Далее надо перейти к шагу 5 и искать единицы в помеченных столбцах, помечать строки, в которых найдены единицы, переходить к шагу 4 и т.д. Здесь мы приходим к ситуации, когда в пятом помеченном столбце нет единиц. Это сигнал к следующему этапу решения задачи.

6. Проставляем единицу в найденный столбец в первую строку. При этом обнаруживаем, что в этой строке уже есть единица — в первом столбце. Ее надо куда-то переставить. Указателем для перестановки служит специальная строка внизу. Она дает адрес: 5-я строка этого же столбца. Получаем

$$B_7 = \begin{array}{|c|c|c|c|c|c|c|}\hline & & & & 1 & & * \\ \hline & 1 & * & * & * & * & \\ \hline & * & 1 & * & * & * & \\ \hline & * & & 1 & & & \\ \hline 1 & * & * & * & * & * & * \\ \hline * & * & * & * & * & 1 & \\ \hline 5 & 1 & 1 & 1 & 1 & 1 & \\ \end{array} .$$

Если эта строка тоже занята, то единицу, которая стояла в строке раньше (в другом столбце), двигаем по ее столбцу туда, куда указывает номер в специальной строке внизу. Однако в нашем случае этого нет. Более того, убеждаемся, что мы нашли решение — совершенное (покрывающее все вершины) покрытие. Строим граф, являющийся ответом на поставленную задачу (рис. 5.5).

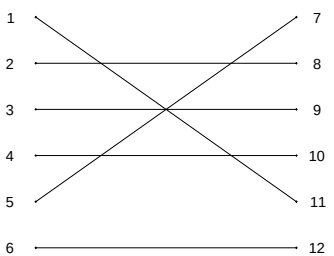


Рис. 5.5

Вычисляя перманент матрицы A, обнаруживаем, что найденное совершенное паросочетание — лишь одно из двух возможных. Очевидно, если на шаге 6 ставить единицу не в первую, а в другую, также свободную, строку 4, то последовательно получим

1					
	1	*	*	*	*
	*	1	*	*	*
	*		1	1	
	*	*	*	*	*
*	*	*	*	*	1

,

1			1		
	1	*	*	*	*
	*	1	*	*	*
	*			1	
	*	*	*	*	*
*	*	*	*	*	1

,

			1		
	1	*	*	*	*
	*	1	*	*	*
	*			1	
1	*	*	*	*	*
*	*	*	*	*	1

.

Сначала передвигаем единицу из четвертого столбца туда, куда указывает специальная строка (здесь она не изображена), т.е. в первую строку. Единицу, стоявшую в первой строке и первом столбце, двигаем опять вниз, на пятую строку. Вновь получаем совершенное паросочетание графа.

Граф может и не иметь совершенного паросочетания. Поэтому в программе предусмотрен выход по условию, когда логическая переменная `Usl` станет ложной, а это произойдет, когда специальный столбец перестанет меняться.

Программа 26

```
> restart: with(networks):with(LinearAlgebra):
  Процедура поиска матрицы паросочетания
> BipartCard:=proc(A)
> global B;
> local i,j,X,Y,nxt,Usl,else1,C,C1,R,j1,nj,nf1,
> nof1r,is1c,cnv1;
  Процедура подсчета 1 в строке x
> nof1r:=proc(x) local i,j;
> i:=0: for j to n do
>         if B[x,j]=1 then i:=i+1 fi:
>       od: i;
> end proc:
  Процедура поиска помеченного столбца без 1
> is1c:=(x)->not is(1 in convert(Column(B,x),set))
>      and R[x]<>0:
  Процедура преобразования в матрицу
> cnv1:=(x)->convert(x,Matrix):
> X:={$1..n}: Y:={$1..n}:
```

```
> for i to n do
>   for j to n do
>    if A[i,j]=0  then B[i,j]:='*': fi;
>    if i in X and j in Y and A[i,j]=1
>    then B[i,j]:=A[i,j]:
>    X:=X minus {i}: Y:=Y minus {j}: fi:
>   od;
> od:
> nxt:=true:
> while nxt do
> C:=[seq(0,i=1..n)]:
> R:=[seq(0,i=1..n)]:
> Usl:=true;
```

Первоначальные метки строк

```
> while Usl do
> for i to n do
>   if nof1r(i)=0 then C[i]:='*'; fi;
> od;
```

Метки столбцов

```
> for i to n do
>   if C[i]<>0 then
>     for j to n do
>       if B[i,j]=0 and R[j]=0 then R[j]:=i; fi;
>     od;
>   fi;
> od;
> C1:=C;
```

Метки строк

```
> for j to n do
>   if R[j]<>0 then
>     for i to n do
>       if B[i,j]=1 and C[i]=0 then C[i]:=j; fi;
>     od;
>   fi;
> od:
```

Проверка зацикливания

```
> Usl:=not Equal(cnv1(C),cnv1(C1));
> od:
```

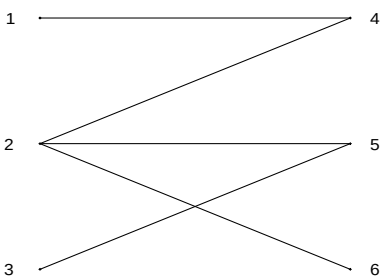
```
> nxt:=false:
  Поиск помеченного столбца без 1
> for j to n do
>   if is1c(j) then nxt:=true: j1:=j; fi;
> od;
> if nxt then
> j:=j1; i:=0; nj:=0; else1:=true;
> while else1 do
> i:=i+1:
> while B[i,j1]<>0 do i:=i+1;od;# Поиск 0 по столбцу
> B[i,j1]:=1; # Вместо 0
> nf1:=nof1r(i);
> while nf1=2 do  # В новой строке находим другую 1
>   j:=1: while B[i,j]<>1 or j=j1 do j:=j+1; od;
>   nj:=R[j];
>   if nj=0 # Если столбец без метки
>     then
>     B[i,j1]:=0;# Вместо ошибочной 1
>     nf1:=1;# Для выхода из цикла
>     else
>     B[nj,j]:=1; # Перенос 1 по адресу из R[j]
>     B[i,j]:=0;  # Вместо 1
>     nf1:=nof1r(nj);
>     i:=nj;
>     else1:=false;
>   fi;
> od;
> od;
> fi;#if nxt
> od:
> B:=subs(`*`=0,B); # Замена * на нули
> end proc:

> save BipartCard, "C:\\SBDISKR\\MAPLE\\bipart.m";
```

Параметр `r` линейного изображения графа записан как функция числа вершин для того, чтобы программу можно было легко перестроить на граф другого порядка.

Программа 27

```
> restart: with(networks):
> new(G):
> n:=3:      # Число вершин в каждой доле
> V:=$1..2*n:# Вершины
> r:=[seq(n+1-i,i=1..n)],[seq(2*n+1-i,i=1..n)]:
> addvertex(V,G):
> addedge([{1,4},{2,4},{2,5},{2,6},{3,5}],G):
> draw(Linear(r),G);
```



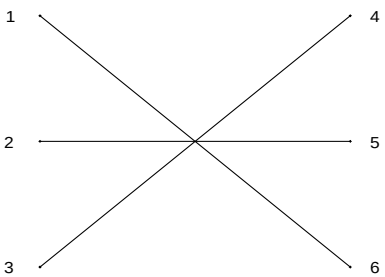

```
> for x in edges(G) do # Матрица двудольного графа
>   A[ends(x,G)[1],ends(x,G)[2]-n]:=1;
> od:
> read "C:\\SBDISKR\\MAPLE\\bipart.m";
> B:=Matrix(n):
> BipartCard(A):
> B_=B;
```

$$B\_ = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

```
> E:={}:# Ребра для графа наибольшего паросочетания
> for i to n do
>   for j to n do
>     if B[i,j]=1 then E:=E union {{i,j+n}} fi:
>   od;
> od;
```

Наибольшее паросочетание

```
> new(G1): addvertex(V,G1):
> addedge(E,G1): draw(Linear(r),G1);
```



Число совершенных паросочетаний

```
> LinearAlgebra[Permanent](A);
```

1

5.22. Задача о назначениях

Одним из простых решений задачи о назначениях является прямой перебор вариантов. В качестве некоторого назначения можно взять такие элементы матрицы, что в каждом столбце и в каждой строке будет содержаться только один элемент. Вложенные циклы перебирают все n^n комбинаций, а условный оператор отбирает $w = n!$ комбинаций, при которых удовлетворяется условие несовпадения. В каждом отобранном случае вычисляется сумма элементов. Минимальная сумма соответствует искомому решению. Отметим конструкцию `i||k`, позволяющую обращаться к индексам `i1`, `i2`, `i3`, `i4` и просто записать сумму оператором `add`. Недостатком программы являются значительное время выполнения, особенно чувствительное для больших n [1], и необходимость менять ее код, вручную добавляя или уменьшая число циклов при изменении n.

Входными данными программы являются данные задачи на с. 72.

Программа 28

```
> restart: n:=4:
> A:=[[1,7,1,3],[1,6,4,6],[17,1,5,1],[1,6,10,4]]:
```

[1] Уже при $n = 10$ время счета становится нереально большим.

```
> w:=0:
> for i1 to n do
>   for i2 to n do
>     for i3 to n do
>       for i4 to n do
>         if((i1<>i2)and(i1<>i3)and(i1<>i4)
>           and(i2<>i3)and(i2<>i4)and(i3<>i4))
>           then w:=w+1;
>           S[w]:=add(A[i||k,k],k=1..n);
>         fi;
>       od;
>     od;
>   od;
> od;
> MinSum:=min(seq(S[i],i=1..w));
```

$$MinSum := 7$$

Одним из самых распространенных алгоритмов решения задачи о назначениях является алгоритм Куна, или венгерский алгоритм, описанный на с. 72.

В следующей программе также решается пример, разобранный на с. 72. Для определения наибольшего паросочетания используется программа BipartCard. Встроенную подпрограмму `flow` применить не удается, так как она часто дает неправильную информацию о насыщенных дугах.

Программа 29

```
> restart: n:=4:
> N:=1..n:
> with(LinearAlgebra):
> A:=Matrix([[1,7,1,3],[1,6,4,6],[17,1,5,1],
>           [1,6,10,4]]):
> A0:=A:
> A1:=Matrix(n):
```

Из каждой строки вычитаем min

```
> for i to n do
>   m:=min(op(convert(Row(A,i),list)));
>   R[i]:=convert(map(`-`,Row(A,i),m),list);
> od:
```

```
> A:=Matrix([seq(R[i],i=N)]):
```

Из каждого столбца вычитаем min

```
> for i to n do
>   m:=min(op(convert(Column(A,i),list)));
>   C[i]:=convert(map('-',Column(A,i),m),list);
> od:
> A:=Transpose(Matrix([seq(C[i],i=N)])):
> n1:=1:
> while n1<>n do # Пока паросочетание несовершенное
>  for i to n do # Матрица A1 двудольного графа
>   for j to n do
>    if A[i,j]=0 then A1[i,j]:=1
>              else A1[i,j]:=0:fi;
>   od:
>  od:
```

Считывание подпрограммы BipartCard из bipart.m (см. с. 133)

```
>  read "C:\\SBDISKR\\MAPLE\\bipart.m";
>  B:=Matrix(n):
>  BipartCard(A1):    # Максимальное паросочетание
>  n1:=nops(op(B)[3]):# Число 1 в матрице B
>  C:=A1-B:      # Матрица графа с дугами от X к Y
>  XM:={}:YM:={}:
```

Множества вершин, не входящих в паросочетание

```
>  for i to n do
>   if not 1 in convert(Row(B,i),list)
>   then XM:=XM union {i} fi;
>   if not 1 in convert(Column(B,i),list)
>   then YM:=YM union {i} fi;
>  od:
```

Множества вершин, достижимых из XM

```
>  Xs:=XM: Ys:={}:
>  L12:=1;
>  while L12<>0 do# Пока не установится процесс
>   L1:=nops(Xs):
>   for k in Xs do
>    for j to n do
```

```
>       for i to n do
>         if C[k,i]=1 then
>         Ys:=Ys union {i};
>         if B[j,i]=1 then Xs:=Xs union {j};fi;
>         fi;
>       od;
>      od;
>    od;
>    L12:=L1-nops(Xs):
>   od:
>   Y0:={$N} minus Ys:
```

Подматрица из строк Xs и столбцов Y0

```
>   sbm:=SubMatrix(A,[op(Xs)],[op(Y0)]):
```

Минимальный элемент подматрицы

```
>   m:=min(op(convert(sbm,set)));
>   for i to n do
>     for j to n do
>       if i in Xs then A[i,j]:=A[i,j]-m; fi;
>       if j in Ys then A[i,j]:=A[i,j]+m; fi;
>     od;
>    od;
>   od:
>   MinSum=add(add(B[i,j]*A0[i,j],i=N),j=N);
```

$$MinSum = 7$$

5.23. Остов наименьшего веса

Приведем две независимые программы решения задачи. В первой из них сначала определяется число остовов, затем для отыскания остова используется оператор из пакета `networks`. Во второй запрограммирован алгоритм ближайшего соседа с наглядной анимацией процесса построения остова. В качестве примера рассматривается граф с рис. 4.34 (см. с. 76).

5.23.1. Число остовов. В специализированном пакете `networks` имеется функция `counttrees(G)` для определения числа остовов графа. Другой способ — вычисление алгебраического дополнения любого элемента матрицы Кирхгофа, полученной по формуле (4.1) со с. 77. Для выполнения операций с матрицами (перемножения, умножения на число, вычитания, транспонирования) необходимо подключить

пакет `LinearAlgebra`. Чтобы не дублировать вывод результатов, уже приведенных на с. 77, операторы вычисления матриц инцидентности (`incidence`) и смежности (`adjacency`) завершаются двоеточием. Перемножить матрицы в Maple можно с помощью точки [1] (`In.Transpose(In)-2*A`), сразу получив матрицу, либо с помощью комбинации знаков `&*` и функции `evalm`:

```
> B:=evalm(In&*Transpose(In)-2*A);
```

В последнем случае результат не имеет тип `Matrix`, в чем можно убедиться, применяя оператор `type(B,Matrix)` проверки типа. Поэтому для дальнейших действий необходима конвертация: `B:=convert(B,Matrix)`. Кроме того, для перемножения матриц `A` и `B` в пакете `LinearAlgebra` есть оператор `MatrixMatrixMultiply(A,B)`.

Если матрица Кирхгофа строится по формуле (4.2), то сначала надо получить какую-либо ориентацию графа. Это делается заменой фигурных скобок { } множества на квадратные скобки [ ] упорядоченной пары в описании ребер. Оператор `op` снимает фигурные скобки, после чего результат заключается в квадратные скобки. Все это выполняется в цикле по `i`: `E1:=seq([op(ends(edges(G)[i],G))],i=1..n)`. Имея список дуг `E1` и вершин, можно создать новый (уже ориентированный) граф и для него получить матрицу инцидентности.

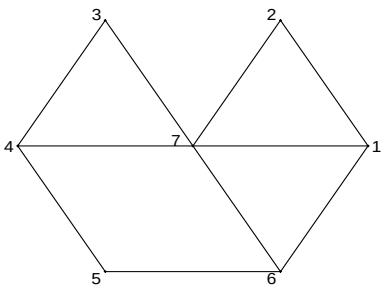
Параметр `r` введен для описания формата вывода рисунка графа в операторе `draw`.

Отметим, что в 10-й версии Maple оператор `Minor` по умолчанию выдает значение определителя, поэтому дополнительного использования `Determinant` не требуется.

Программа 30

```
> restart:
> with(networks):
> with(LinearAlgebra):
> new(G):  # Новый граф
> addvertex(seq(i,i=1..7),G): # Семь вершин
> r:=[4],[5,3],[7],[6,2],[1]:
> addedge([{3,7},{1,2},{6,1},{5,6},
> {3,4},{2,7},{4,7},{1,7},{4,5},{6,7}],
> weights=[27,19,24,29,21,14,16,13,22,18],G);
                e1, e2, e3, e4, e5, e6, e7, e8, e9, e10
> draw(Linear(r),G);  # Рисунок исходного графа
```

[1] Так называемая `dot`-операция.




```
> In:=incidence(G):# Матрица инцидентности
> A:=adjacency(G): # Матрица смежности
> B:=In.Transpose(In)-2*A:# Матрица Кирхгофа
> m:=Minor(B,1,1):
> Determinant(m);          # Число остовов
                          79
```

Число остовов (*number of maximum spanning forests*) графа можно получить также и по полиному Татта (Tutte W.T.), положив оба его аргумента равными 1: `tuttepoly(G,1,1)`, или используя ранг-полином `rankpoly(G,0,0)` с нулевыми аргументами (см. с. 22).

5.23.2. Использование оператора `spantree`. В операторе `spantree` из пакета `networks` заложен алгоритм Прима. Оператор `edges(T)` дает список имен ребер графа, а функция `eweight(Ed[i],T)` обращается к имени `i`-го ребра графа `T` (например, `e1` или `e2`) и дает его вес, заданный ранее в опции `weights` оператора `addedge`.

Обозначение для вывода результата — `MinW`. Заметим, что если в задаче требуется найти только сумму весов остова, то лучше применить оператор `spantree(G,v,MinW)` и сразу получить ответ в переменной `MinW`. Здесь `v` — вершина, начиная с которой строится минимальный остов. Для решения аналогичной задачи в пакете `networks` имеется оператор `shortpathtree`, в котором используется алгоритм Дейкстры. Программа 31 является продолжением программы 30, в которой заданы граф `G` и параметр линейного вывода `r`.

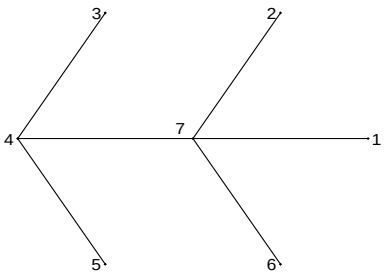
Программа 31

```
> T := spantree(G): # Остов минимального веса
> Ed:=edges(T);     # Ребра остова
```

```
> MinW=add(eweight(Ed[i],T),i=1..6); # Сумма весов
> draw(Linear(r),T);
```

$$Ed := \{e5,\ e6,\ e7,\ e8,\ e9,\ e10\}$$

$$MinW = 104$$



5.23.3. Алгоритм минимального соседа. В качестве начальной можно взять произвольную вершину `w`; в данном примере `w = 1`. Перед началом поиска создается матрица весов `C`. Несуществующим ребрам приписывается бесконечный вес. Оператор `edges({i,j},G)` дает множество ребер, соединяющих вершины `i` и `j` графа `G`. Для того чтобы снять фигурные скобки, обозначающие множество, используется оператор `op`.

Поиск ведется из множества `W`, которое всякий раз после нахождения очередного ребра наименьшего веса уменьшается с помощью операции `minus`.

Логическая функция `member(j,W)` определяет наличие вершины `j` в множестве `W`.

Наряду с решением задачи для наглядности можно показать последовательность построения остова, создав средствами Maple gif анимацию процесса. Для этого на каждом шаге после нахождения очередного ребра оператором `draw` создается отдельный кадр `ris[i]`. Последовательность кадров проигрывается оператором `display` пакета `plots` с опцией `insequence=true`. Полученный анимированный рисунок, иллюстрирующий последовательность нахождения остова минимального веса, может быть сохранен в формате gif.

Программа 32

```
> restart: with(networks): with(plots): new(G):
> n:=7: new(Ost):  addvertex(seq(i,i=1..n),Ost):
```

```
> addvertex(seq(i,i=1..n),G):
> addedge([{3,7},{1,2},{6,1},{5,6},
> {3,4},{2,7},{4,7},{1,7},{4,5},{6,7}],
> weights=[27,19,24,29,21,14,16,13,22,18],G);
> for i to n do # Матрица весов
>   for j to n do
>     if(edges({i,j},G)<>{})
>     then C[i,j]:=eweight(op(edges({i,j},G)),G);
>     else C[i,j]:=infinity; fi;
>   od;
> end do;
> V:=vertices(G): w:=1: W:=V minus {w}: T:={}:
> for v to n do near[v]:=w; d[v]:=C[v,w]; od:
> for i while i<n+1 do # Основной цикл
> addedge(T,Ost):
> ris[i]:=draw(Linear([4],[5,3],[7],[6,2],[1]),Ost);
> dmin:=infinity;
> for j to n do
>     if d[j]<dmin and member(j,W)
>     then v:=j; dmin:=d[j];
>     end if;
> end do;
> T:=T union {{near[v],v}};  W:=W minus {v};
> for u to n do
>     if d[u]>C[u,v] and member(u,W)
>     then near[u]:=v; d[u]:=C[u,v];
>     end if;
> end do;
> end do: # Конец основного цикла
> MinW=add(eweight(op(edges(T[i],G)),G),i=1..n-1);
```

$$MinW = 104$$

```
> display(seq(ris[q],q=1..n),
> insequence=true,thickness=1,axes=none);
```

5.24. Фундаментальные циклы

Для нахождения фундаментальных циклов графа в пакете `networks` имеются операторы `fundcyc({ e },G)` и `cyclebase(G)`. Первый из них определяет цикл графа `G` для заданного подмножества ребер графа в предположении, что это множество соответствует графу с одним циклом. Оператор `cyclebase(G)` дает все множество циклов. Циклы обозначаются в виде множества ребер. Доступ к отдельному циклу можно получить, обратившись к нему по индексу, например `cyclebase(G)[1]`.

Программа 33

```
> n1 := nops(edges(G) minus Ed); # Число хорд
> for i to n1 do
>   e:= op(i,edges(G) minus Ed): #  i-я хорда
>   print(e,fundcyc(Ed union { e }, G)):
> od:
```

$$n1 := 4$$

$$e1, \{e1, e7, e5\}$$

$$e2, \{e2, e8, e6\}$$

$$e4, \{e10, e7, e9, e4\}$$

$$e3, \{e10, e8, e3\}$$

```
> cyclebase(G);
```

$$\{\{e10, e8, e3\}, \{e2, e8, e6\}, \{e10, e7, e9, e4\}, \{e1, e7, e5\}\}$$

5.25. Гамильтоновы циклы

Используем алгебраический метод, основанный на умножении матрицы смежности.

В этой программе вершины впервые обозначены буквами, а не цифрами. Замечено, что Maple после ввода графа переставляет его вершины в алфавитном порядке. В результате циклы оказываются ошибочными. Поэтому лучше задавать вершины сразу в алфавитном порядке.

Для перемножения матриц используется операция некоммутативного умножения (операция обычного умножения коммутативна), так как в циклах вершины могут располагаться произвольно, а не только в алфавитном порядке.

Поэлементное присваивание `C[i,j]:=P[i,j]` пришлось использовать из-за того, что присваивание `C:=P`, предусмотренное для работы с матрицами, работает ненадежно и часто является причиной ошибок.

Программа 34

```
> restart: with(networks): new(G):
> V:=[a,b,c,d,g]:   n:=nops(V):
> addvertex(V,G):
> addedge({{a,c},{a,d},{b,g},{b,c},
>      {c,d},{d,g},[a,g],[a,b]},G):
> B:=Matrix(n):
> A:=adjacency(G):# Матрица смежности
> for i to n do
>   for j to n do
>    if A[i,j]=1 then B[i,j]:=V[j];fi;
>   od:
> od:
> C:=A:
> for k to n-1 do
>   for i to n do
>    for j to n do
>       P[i,j]:=expand(subs(V[i]=0,V[j]=0,
>       add(B[i,m]&*C[m,j],m=1..n)));
>    od;
>   od;
>   for i to n do
>    for j to n do
>     C[i,j]:=P[i,j];
>    od;
>   od;
> od:
> S:={}:
> for i to n do
>   if C[i,i]<>0 then
>   S:=S union {expand(C[i,i]&*V[i])};fi;
> od:
> F:={}:
> for q in S do
>   if whattype(q)=`+` then F:=F union {op(q)}
>   else F:=F union {q}:
>   fi;
```

```
> od;
> S:=map(x -> convert(x,list),F):
> for q in S do # Убираем короткие контуры
>   if nops(q)<>n then S:=S minus {q};fi;
> od;
> k:=nops(S):# Число помеченных контуров
> H:={}:
```

Вершину 1 ставим на первое место в контуре

```
> for j to k do
>   for i to n do #m-е место 1-й вершины в контуре
>     if S[j][i]=V[1] then m:=i;fi;
>   od;
>   for i to n do
>     Z[i]:=S[j][(i+m-2 mod n)+1];
>   od;
>   H:=H union {convert(Z,list)}:
> od:
> H;
```

$$\{[a, b, g, d, c], [a, d, g, b, c], [a, c, b, g, d], [a, g, b, c, d]\}$$

В следующей программе решается задача коммивояжера с помощью алгоритма Nva, описанного на с. 86. Обход графа начинается с произвольной вершины, например `b`. От выбора этой вершины часто зависит результат. Движение по графу происходит по исходящим (опция `Out`) дугам минимального веса с последующим удалением пройденной вершины. Поэтому перед началом процесса необходимо запастись дубликатом графа, из которого затем придется извлекать информацию о длине пройденного маршрута: `G1:=duplicate(G)`.

Оператор `break` досрочного выхода из цикла введен для случая двух или более одинаковых дуг минимального веса, выходящих из текущей вершины.

Несмотря на предельную простоту алгоритма, для небольших графов он дает достаточно близкие к точному решению ответы.

Программа 35

```
> restart: with(networks): new(G):
> V:=[a,b,c,d,f,g]:   n:=nops(V):
> addvertex(V,G): addedge({[a,b],[b,c],[c,d],[d,f],
> [f,a],[a,f],[f,d],[d,c],[c,b],[b,a],
```

```
> [g,a],[g,b],[g,c],[g,d],[g,f],[b,g],[c,g]},
> weights=[3,8,1,1,3, 1,3,8,3,3,
> 3,3,3,5,4,3,1],G):
> G1:=duplicate(G):
> a0:=b:# Начальная вершина
> a1:=a0: sw:=0:
> s:=[a0]:
> for k to n-1 do
>  for v in incident(a1,G1,Out) do
>   if eweight(v,G1)=
>   min(op(eweight([op(incident(a1,G1,Out))],G1)))
>   then u:=v; sw:=sw+eweight(v,G1);break;fi;
>  od;
>  a2:=ends(u,G1)[2];
>  delete(a1,G1):
>  a1:=a2:  s:=[op(s),a2]:
> od:
> sw:=sw+eweight(op(edges([a2,a0],G)),G); # Сумма
> s; # Контур
```

$$sw := 19$$
$$[b,\ a,\ f,\ d,\ c,\ g]$$

Еще один способ решения задачи коммивояжера основан на использовании программы 34, в которой отыскивались все гамильтоновы циклы. Сначала нужно ввести взвешенный граф, вычислить количество `n1:=nops(H)` гамильтоновых циклов и найти минимальный из них. Для этого вводим процедуру вычисления веса дуги `u-v` в цикле `k`:

```
> L:=(u,v,k)->
> eweight(op(edges([H[k][u],H[k][v]],G)),G)
```

и используем стандартную функцию отыскания минимума последовательности:

```
> min(seq(add(L(i,i+1,k),i=1..n-1)+
> L(n,1,k),k=1..n1))
```

Последнее слагаемое, `L(n,1,k)`, есть вес замыкающей дуги. Время вычисления при этом будет в несколько раз больше, и с увеличением порядка графа разность в быстродействии программ будет расти. Несомненное достоинство алгоритма — его точность и простота.

5.26. Муравьиный алгоритм

Рассмотрим вариант программной реализации алгоритма, описанного на с. 87.

Решим задачу коммивояжера для `n` городов, координаты `x[i],y[i]` которых заданы. Для простоты зададим их случайным образом, используя встроенную функцию генератора целых случайных чисел `rand(0..n)` из интервала $[0, n]$. Для инициализации генератора случайных чисел можно применить функцию `randomize()`, привязанную к системным часам. Иначе генератор выдает случайные числа, повторяющиеся при каждой сессии Maple. Здесь эта функция не применяется, поскольку для отладки и анализа программы лучше иметь постоянный набор координат городов.

В начале программы вводятся константы задачи. Эмпирические константы $a = \alpha$, $b = \beta$ и $0 < p < 1$ выбираются произвольно. Улучшение сходимости во многом зависит от их значений. Число `Lmin` требуется для сравнения при выборе минимального маршрута, масштабная константа `Q` порядка длины маршрута выбирается пропорциональной порядку графа. При вычислении расстояний учитывается симметрия матрицы.

Функция `P` вероятности перехода имеет в качестве аргумента список из чисел `x[i]`. Вычисляется сумма `sm`=$\sum x[i]$. Отрезок прямой от 0 до 1 разбивается на n участков `[Beg[i], End[i]]` с длинами `x[i]/sm`, где `Beg[1]=0`, `End[n]=1`. Затем случайное число $0 \geqslant s \geqslant 1$ указывает выигрышный номер — номер вершины для дальнейшего движения.

В цикле, задающем время жизни колонии (для примера — 100 циклов), n муравьев размещаются по вершинам графа, и начинается движение. Первоначально феромон распределен по дугам равномерно, затем, после завершения муравьями своих маршрутов, уровень феромона увеличивается там, где муравьи ходили чаще.

Результат отображается графически. На рис. 5.6 получен цикл длиной `Lmin=62.443` при $a = 8$, $b = 0{,}5$, $p = 0{,}7$. На рис. 5.7 представлен самый короткий цикл — `Lmin=60.855` — при $a = 1$, $b = 3$, $p = 0{,}5$. Наиболее длинный цикл, `Lmin=67.666`, полученный при $a = 1$, $b = 9$, $p = 1$, изображен на рис. 5.8. Толщина линии, где феромона получилось больше, пропорционально увеличена. Масштабный коэффициент — 1/20 — подбирается вручную. Он зависит от числа циклов, времени жизни колонии и от коэффициента испарения. Функция `z` введена только для сокращения записи. Номера вершин выводятся на рисунок со сдвигом по x и y на 0,5 характерным для Maple приемом: `map('+',z(i),0.5)`, т.е. номера вершины i указываются в точке с координатами `x[i]+0.5`, `y[i]+0.5`. Раскрасить пути можно с помощью опции `COLOR(HUE,(Wt[i,j]+Wt[j,i])/10)`. Коэффициент

1/10 подбирается опытным путем. В цветовом режиме HUE [1] параметр меняется от 0 до 1.

При необходимости вершины можно изобразить кружками: s1:=plot([seq([x[i],y[i]],i=1..n)],style=point, symbol=circle).

Вид матрицы W, элементы которой управляют направлением движения муравья, показывает, что в результате случайных блужданий муравьев изначально симметричная матрица становится несимметричной. Следовательно, теперь направление движения имеет значение. Граф приобретает ориентацию. Именно поэтому для изображения путей толщина линий условно берется пропорциональной сумме Wt[i,j]+Wt[j,i]. Очевидно, действительно минимальный путь может (для малого времени жизни колонии) не совпадать с отмеченной толстой линией.

Программа 36

```
> restart;
> with(LinearAlgebra):with(plots):
```
Константы задачи
```
> n:=7: Q:=5*n: Lmin:=1000:
> a:=9: b:=5: p:=0.7:
```
Матрица приращений следа
```
> DWt:=Matrix(n):
```
След
```
> Wt:= Matrix(1..n,1..n,0.1):
```
Координаты вершин
```
> m:=rand(0..20):
> for i to n do
>   x[i]:=m():y[i]:=m():
> od:
```
Симметричная матрица расстояний
```
> W:=Matrix(n,shape=symmetric):
> for i to n do
>  for j from i to n do
>   W[i,j]:=evalf(sqrt((x[i]-x[j])^2+(y[i]-y[j])^2));
>  od;
> od;
> ###
```

[1] Кроме того, существуют режимы RGB и HSV с тремя параметрами, от 0 до 1 каждый.

```
> for i to n do W[i,i]:=infinity:do:
```

Процедура выбора наиболее вероятного направления

```
>   P:=proc(x)
>   local n,sm,i,Beg,End,c,s,m,j;
>   n:=nops(x):
>   sm:=0: for i to n do sm:=sm+x[i]: od:
>   c:=0:
>   for i to n do
>    Beg[i]:=c: End[i]:=c+x[i]/sm; c:=End[i]:
>   od:
>   m:=rand(0..100):
>   s:=m()/101:
>   for i to n do
>    if (Beg[i]<=s) and (End[i]>=s)
>    then j:=i:  fi:
>   od:
>   return j;
>   end proc:
> for k0 to 100 do # Основной цикл
>   for ant to n do # Цикл по муравьям
>    s:={$ 1..n }: # Список непосещенных вершин
>    j:=ant:     # Начальная вершина для муравья ant
>    for j1 to n-1 do
>     s:=s minus {j}; # Tabu list увеличился
>     sp:=[]:k1:=0:
>     for i in s do # Каждой вершине - свой вес
>      sp:=[op(sp),1/W[j,i]^b*Wt[j,i]^a]:
>      k1:=k1+1:  nn[k1]:=i;
>     od:
>     j0:=nn[P(sp)]; # Выбор направления
>     v[j1]:=j0; r[j1]:=W[j,j0];
>     j:=j0: # Начало дуги = конец предыдущей
>    od: # Цикл j1 по всем вершинам для муравья ant
```

Добавляем последнюю дугу

```
>    L:=add(r[i],i=1..n-1)+W[op(s),ant];
>    v[0]:=ant; v[n]:=ant; # Начало и конец пути
>    v1:=seq([v[m],v[m+1]],m=0..n-1):  # Дуги
>    for i to n do # Пометка дуг феромоном
```

```
>       DWt[op(v1[i])]:=DWt[op(v1[i])]+Q/L;
>     od:
>     if L<Lmin then Lmin:=L:fi: # Выбор min
>   od: # ant
>   Wt:=Wt+DWt*p: # Добавление новых следов
>   Wt:=Wt*(1-p): # Испарение феромона
> od: # k0
> Lmin;
```

$$62.44332567$$

```
> z:=i ->[x[i],y[i]]:
> ris:=seq(seq(PLOT(CURVES([z(i),z(j)]),
> THICKNESS(round((Wt[i,j]+Wt[j,i])/20))),i=1..n),
>   j=1..n):
> Ver:=PLOT(seq(TEXT(map(`+`,z(i),0.5),
>   convert(i,symbol)),i=1..n),FONT(TIMES,BOLD,14)):
> display(ris,Ver);
```

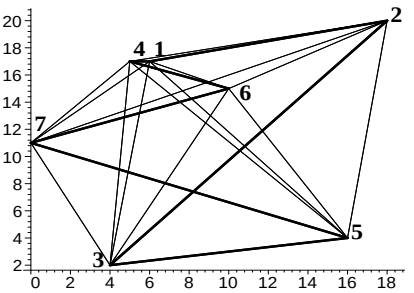


Рис. 5.6

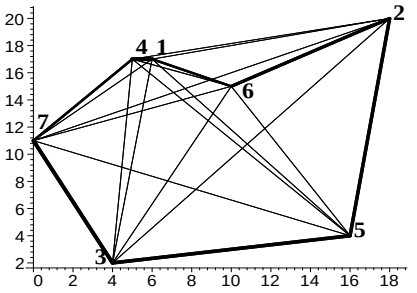


Рис. 5.7

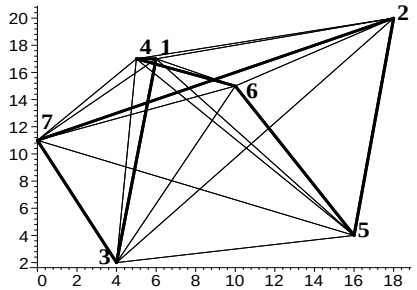


Рис. 5.8

Отметим, что задача коммивояжера является одной из основных тестовых задач для различных алгоритмов. Кроме уже описанных способов, задача может быть решена методами целочисленного линейного программирования [12] и с помощью нейронных сетей Хопфилда [19].

5.27. Алгоритм отжига

Алгоритм отжига является одним из современных методов, применяемых в теории искусственного интеллекта. Его описание дано на с. 89 [7].

Приведем программу, которая решает методом отжига ту же задачу и при тех же условиях, что и предыдущая. Это позволит сопоставить методы. Операторы случайного задания координат городов и вычисления матрицы весов возьмем те же, что и в начале программы 36. Все операторы программы 37 идут после строки-метки `###` программы 36.

Следует обратить внимание на несколько неожиданную операцию поэлементного умножения вектора на 1 `V0:=map('*',Z,1)`. Вместо того, чтобы просто записать `V0:=Z`, пришлось пойти на такое присвоение для снятия зависимости между идентификаторами массивов и их содержимым. Дело в том, что в Maple после выполнения `V0:=Z` изменение массива `Z` повлечет немедленное изменение `V0`, что нежелательно, так как мы хотим запомнить расположение городов в оптимальном маршруте. Другой вариант решения проблемы — классический цикл `for i to n do V0[i]:=Z[i] od` поэлементного присваивания.

Первый маршрут задается случайным образом. Не намного дольше будет работать более простой вариант: `Z[k]:=k`. Поскольку города-вершины занумерованы без определенного правила, такой маршрут тоже в каком-то смысле случаен.

В ответе вектор `V0` транспонируется только для удобства вывода в строку.

Программа 37

```
> for i to n do W[i,i]:=infinity: od: W;
```

$$\begin{bmatrix} \infty & 12.3693 & 15.1327 & 1. & 16.4012 & 4.47214 & 8.48526 \\ 12.3693 & \infty & 22.8036 & 13.3417 & 16.1245 & 9.43398 & 20.1246 \\ 15.1327 & 22.8036 & \infty & 15.0333 & 12.1655 & 14.3178 & 9.84886 \\ 1. & 13.3417 & 15.0333 & \infty & 17.0294 & 5.38516 & 7.81025 \\ 16.4012 & 16.1245 & 12.1655 & 17.0294 & \infty & 12.5300 & 17.4642 \\ 4.47214 & 9.43398 & 14.3178 & 5.38516 & 12.5300 & \infty & 10.7703 \\ 8.48526 & 20.1246 & 9.84886 & 7.81025 & 17.4642 & 10.7703 & \infty \end{bmatrix}$$

Случайные числа для перестановок городов

```
> m:=rand(1..n):
```

Случайные числа для выбора решения

```
> p:=rand(1..100):
```

Генератор первого случайного маршрута

```
> S:={}: k:=1:
> while k<n+1 do
>   Z[k]:=m();
>   if not(Z[k] in S) then
>   S:= S union {Z[k]};k:=k+1; fi;
> od:
```

Начальная длина маршрута и начальная температура

```
> L0:=infinity: T:=100.:
> alpha:=0.9:# Коэффициент снижения температуры
> Z0:=Z: # Начальный маршрут
> k:=0: # Счетчик снижений
> for i to 500 do # Основной цикл
>   i1:=m(): i2:=m(): # Случайные города
>   Z:=Z0:
```

Меняем местами в маршруте города i1 и i2

```
>   z:=Z[i1]:  Z[i1]:=Z[i2]:  Z[i2]:=z:
```

Вычисляем длину получившегося маршрута

```
>   L1:=add(W[Z[i],Z[i+1]],i=1..n-1)+W[Z[n],Z[1]]:
```

Если длина маршрута уменьшилась, выбираем этот маршрут

```
>   if L1<L0 then L0:=L1: Z0:=Z: k:=k+1:TT[k]:=T:
>    L[k]:=L1:  V0:=map(`*`,Z,1);
```

Больший маршрут выбираем в зависимости от P и p( )

```
>    else  P0:=p()/100.: # Случайное число
```

Вероятность выбора худшего решения

```
>     P:=exp(-(L1-L0)/T):
>     if P0<P then Z0:=Z: fi:
>   fi:
```

Через каждые десять циклов уменьшаем температуру

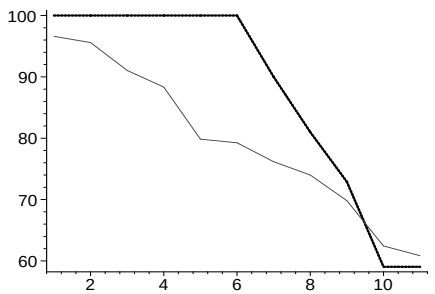
```
>   if (i mod 10) = 0 then
>    T:=alpha*T:
>   fi;
> od:
```

Ответы. Длина цикла и последовательность городов

```
> L[k],Transpose(V0);
```

$$60.85526512,\ [7,\ 3,\ 5,\ 2,\ 6,\ 1,\ 4]$$

```
> plot([[seq([i,L[i]],i=1..k)],
>       [seq([i,TT[i]],i=1..k)]],
>       linestyle=[1,3],thickness=[1,3]);
```



Решение зависит от нескольких параметров. Меняя число циклов, частоту и характер понижения температуры, можно получать различные ответы. При меньшем числе циклов иногда получаются чуть большая длина:

$$62.44332567,\ [3,\ 5,\ 2,\ 6,\ 4,\ 1,\ 7]$$

и другой характер изменения температуры (рис. 5.9).

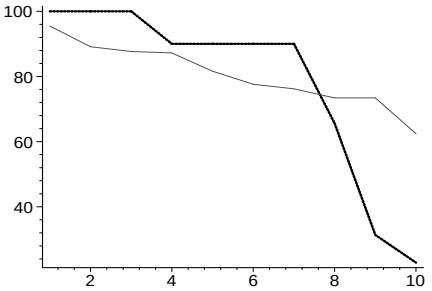


Рис. 5.9

Найденное решение совпадает с одним из решений, полученных с помощью муравьиного алгоритма (см. рис. 5.6 на с. 150).

5.28. Основные функции пакета networks

Вызов пакета производится командой `with(networks)`. Приведем основные его функции:

- `addedge([{1,2},[3,4]],G)` — добавить в граф `G` ребра или дуги. Ребра в неографе задаются как множества — с вершинами в фигурных скобках, а дуги в орграфе — списком вершин в квадратных скобках. Если требуется задать вес дуги в орграфе, то дуги надо обозначать только квадратными скобками. Если вес можно не указывать, то для сокращения ввода две дуги, соединяющие вершины в разных направлениях, проще ввести как ребро;

- `addvertex(seq(i,i=1..n),G)` — добавить к графу `G` вершины `1,2 ,..., n`. В опции `weights` можно в списке указать веса вершин. По умолчанию веса нулевые;

- `adjacency(G)` — матрица смежности графа `G`. Данный оператор правильно работает в случае мультиграфов, но для псевдографов он не пригоден. Главная диагональ этой матрицы нулевая;

- `allpairs(G)` — матрица пар расстояний между вершинами графа `G`;

- `ancestor(n,T)` — вершина-отец вершины `n` в ориентированном дереве `T`;

- `arrivals(v,G)` — множество ребер, входящих в вершину `v` орграфа `G`;

- `bicomponents(G)` — выделение множеств ребер, составляющих блоки [1] неографа `G`. Мосты помещаются в отдельное множество впереди компонент;
- `charpoly(G, x)` — характеристический полином графа `G` (переменная `x`);
- `chrompoly(G,x)` — хроматический полином графа `G` (переменная `x`);
- `complement(G)` — дополнение графа `G`. Например, дополнением к графу на рис. 1.2 (см. с. 10) является граф на рис. 5.10;

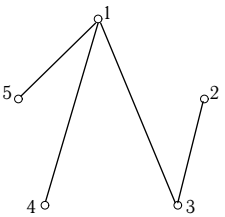


Рис. 5.10

- `complete(n)` — полный граф K_n;
- `complete(n,m)` — полный двудольный граф $K_{n,m}$;
- `components(G)` — список вершин в связных компонентах графа;
- `connect({u,v},'weights'=L1,'names'=L2,'directed', G)` — соединить две вершины — `u` и `v`. Можно указать вес, имя и выбрать орграф (`directed`) или неограф. Оператор позволяет соединить сразу несколько вершин с одной, например `connect(1,{2,3,4},G)`;
- `connectivity(G)` — число $\lambda(G)$ реберной связности (наименьшее число ребер графа `G`, удаление которых приводит к несвязному графу);
- `contract(u,v,G)` — совмещение вершин, или стягивание ребра, соединяющего вершины `u` и `v` графа `G`;
- `counttrees(G)` — вычисление количество остовов графа `G`;

[1] Вершина называется точкой сочленения графа, если ее удаление приводит к увеличению числа компонент графа. Максимальный по включению связный подграф, не содержащий точек сочленения, называется блоком.

- `cube(n)` — n-мерный куб Q_n [10]. Нумерация вершин по вложенным квадратам принимается в порядке $[2k, 2k+1, 2k+3, 2k+2]$, $k = 0, ..., 2^{n-2}-1$. Например, для `G:=cube(3)` имеем следующее изображение: `draw(Concentric([0,1,3,2],[4,5,7,6]),G)`;

- `cycle(n)` — циклический граф с n вершинами;

- `cyclebase(G)` — множество фундаментальных циклов графа `G`;

- `daughter(n,T)` — сын [1] вершины `n` в ориентированном дереве;

- `degreeseq(G)` — степенная последовательность графа `G`;

- `delete(z,G)` — удаление ребра или вершины `z` из графа `G`;

- `departures(v,G)` — множество ребер, выходящих из вершины `v` орграфа `G`;

- `diameter(G)` — диаметр графа `G`;

- `dinic(G,1,2,eset,comp)` — определение максимального потока в сети (см. `flow`);

- `djspantree(G)` — разбиение графа на реберно-непересекающиеся деревья, так, чтобы большинство из них было остовами (алгоритм Эдмондса [2]);

- `dodecahedron()` — создание додекаэдра, т.е. регулярного (однородного) графа порядка 20 степени 3;

- `draw(G)` — рисунок графа `G`;

- `draw3d(G)` — трехмерное изображение графа `G`. Координаты вершин принимаются в соответствии с собственными векторами матрицы смежности;

- `duplicate()` — создание копии графа. Изменение оригинала не влияет на копию;

- `edges(G)` — множество ребер графа `G`;

- `ends(G)` — множество пар концов ребер графа `G` (или `ends(e1,G)` — концы ребра `e1`);

[1] Дословно — дочь.
[2] Edmonds J.

- `flow(G, s, t, eset, comp, maxflow=f)` — определение максимального потока в сети из вершины `s` (источник) в вершину `t` (сток) графа `G`. После выполнения процедуры список насыщенных дуг помещается в переменную `eset`, а в переменную `comp` помещаются вершины, входящие в насыщенную сеть. Необязательная опция `maxflow=f` ограничивает поток значением `f`. Замечена ошибка Maple: в переменную `eset` часто помещаются лишние дуги;
- `fundcyc(e,G)` — определение цикла из подмножества `e` ребер графа `G`. Предполагается, что `e` содержит только один цикл;
- `getlabel(G)` — определение порядкового номера графа;
- `girth(G,scyc)` — определение длины кратчайшего цикла в графе `G` или возвращение «бесконечности», если циклов нет. Номера ребер цикла помещаются в переменную `scyc`;
- `graph(V,E)` — граф, заданный списком вершин `V` и ребер `E`;
- `graphical([sp]` — проверка графичности степенной последовательности `sp`. Если заданной последовательности соответствует граф, то оператор создает список ребер. Можно задать мультиграф: `graphical([sp],'MULTI')`;
- `gsimp(G)` — создание простого графа из псевдографа или мультиграфа `G`. Удаление кратных ребер и петель;
- `gunion(G,J,'SIMPLE')` — объединение двух графов. В полученном графе множество вершин является объединением множеств вершин графов `G` и `J`. Если опустить опцию `SIMPLE`, то в полученном графе допускаются кратные ребра (мультиграф);
- `head(e1,G)` — конец дуги `e1` (направленного ребра) орграфа `G`;
- `icosahedron` — создание икосаэдра, т.е. регулярного (однородного) графа порядка 12 степени 5;
- `incidence(G)` — матрица инцидентности графа `G` (строки — вершины, столбцы — ребра);
- `incident(v,G,In)` — множество ребер, инцидентных вершине `v`. В орграфе можно уточнить: `incident(v,G,In)` — входящие дуги, `incident(v,G,Out)` — исходящие дуги;
- `indegree(v,G)` — полустепень захода вершины `v`;
- `induce(Eset, G)` — создание подграфа по данному множеству вершин или ребер графа `G`;

- `isplanar(G)` — проверка планарности графа `G`;
- `maxdegree(G)` — максимальная степень вершин графа `G`;
- `mincut(G, s, t, vf)` — минимальный разрез, отделяющий вершины `s` и `t`. В переменной `vf` — величина потока;
- `mindegree(G)` — минимальная степень вершин графа `G`;
- `neighbors(v,G)` — множество вершин графа `G`, соседних с вершиной `v`;
- `new(G)` — создание нового графа `G`;
- `nops(edges(G))` — число ребер графа;
- `octahedron()` — создание октаэдра, т.е. регулярного (однородного) графа порядка 6 степени 4;
- `outdegree(v,G)` — полустепень исхода вершины `v`;
- `petersen()` — граф Петерсена, [1] т.е. регулярный (однородный) граф порядка 10 степени 3 (рис. 5.11);

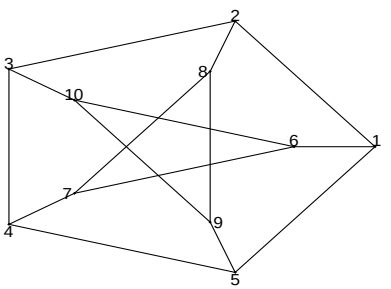


Рис. 5.11

- `random(n)` — создание случайного графа с `n` вершинами;
- `rank(E1,G)` — ранг подграфа `G` с подмножеством ребер `E1`, т.е. $n-k$, где $n=|V|$ — порядок графа `G(V,E)`, k — число компонент связности графа `G(V,E1)`;
- `rankpoly(G, x, y)` — ранг-полином графа `G` с переменными `x` и `y`;

[1] Petersen J.

- `shortpathtree(G,v)` — выделение из графа `G` дерева минимальных путей из вершины `v`;
- `show(G)` — информация о графе: названия вершин, ребер, таблица весов ребер и т.д.;
- `shrink(vset,G,u)` — стягивание множества вершин `vset` графа `G` с образованием из них новой вершины (узла) `u`;
- `span(eset,G)` — определение множества ребер графа `G`, концы которых принадлежат множеству `eset` концов ребер графа `G`;
- `spantree(G,s,w)` — определение остова наименьшего веса графа `G`. Корень дерева — в вершине `s`, суммарный вес остова — в переменной `w`;
- `tail(e1,G)` — начало дуги `e1` орграфа `G`;
- `tetrahedron()` — создание тетраэдра — регулярного (однородного) графа порядка 4 степени 3, или, что то же самое, полного графа K_4;
- `tuttepoly(G,i,j)` — создание полинома Татта по `i` и `j`;
- `vdegree(v,G)` — степень вершины;
- `vertices(G)` — множество вершин графа `G`;
- `void(n)` — создание пустого графа с `n` вершинами. Возможно обращение по списку вершин, например `void({$1..n})` или `void({a,b,c1,c2})`;
- `vweight(G)` — определение весов вершин графа. Возможно обращение по списку вершин, например `vweight([1,2],G)`, или обращение к отдельной вершине. Здесь под весом вершины понимается число, введенное оператором `addvertex` в опции `weights`, а не вес вершины дерева при определении центроида (см. с. 40).

Список литературы

1. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.
2. *Асанов М.О., Баранский В.А., Расин В.В.* Дискретная математика: графы, матроиды, алгоритмы. — Ижевск: НИЦ «Регулярная и хаотическая динамика», 2001.
3. *Белоусов А.И., Ткачев С.Б.* Дискретная математика. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.
4. *Берж К.* Теория графов и ее применения. — М.: Изд-во иностранной литературы, 1962.
5. *Говорухин В.Н., Цибулин В.Г.* Компьютер в математическом исследовании. Учебный курс. — СПб.: Питер, 2001.
6. *Горбатов В.А., Горбатов А.В., Горбатова М.В.* Дискретная математика. — М.: АСТ, 2003.
7. *Джонс М.Т.* Программирование искусственного интеллекта в приложениях. — М.: ДМК Пресс, 2004.
8. *Дьяконов В.П.* MAPLE 6: учебный курс. — СПб.: Питер, 2001.
9. *Дьяконов В.П.* MATLAB: учебный курс. — СПб.: Питер, 2001.
10. *Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И.* Лекции по теории графов. — М.: Наука, 1990.
11. *Зубов В.С., Шевченко И.В.* Структуры и методы обработки данных. Практикум в среде Delphi. — М.: Филинъ, 2004.
12. *Иглин С.П.* Решение некоторых задач теории графов в MATLAB// Exponenta Pro. Математика в приложениях. 2004. №4(4).
13. *Иванов Б.Н.* Дискретная математика. Алгоритмы и программы.– М.: Лаборатория базовых знаний, 2002.
14. *Иванов Б.Н.* Дискретная математика. Алгоритмы и программы. Полный курс.– М.: Физматлит, 2007.
15. *Касьянов В.Н., Евстигнеев В.А.* Графы в программировании: обработка, визуализация и применение. — СПб.: БХВ-Петербург, 2003.
16. *Кирсанов М.Н.* Решебник. Теоретическая механика/ Под ред. А.И. Кириллова. — М.: Физматлит, 2002.
17. *Кнут Д.Э.* Искусство программирования. — М.: Издательский дом «Вильямс», 2003. Т 1, 3.
18. *Кристофидес Н.* Теория графов. Алгоритмический подход. — М.: Мир, 1978.

19. *Круг П.Г.* Нейронные сети и нейрокомпьютеры. — М.: Изд-во МЭИ, 2002.
20. *Липский В.* Комбинаторика для программистов. — М.: Мир, 1988.
21. *Макконнелл Дж.* Основы современных алгоритмов. — М.: Техносфера, 2004.
22. *Манзон Б.М.* Maple V Power Edition. — М.: ИИД «Филинъ», 1998.
23. *Матросов А.* Maple 6. Решение задач высшей математики и механики. — СПб.: БХВ-Петербург, 2001.
24. *Москинова Г.И.* Дискретная математика. Математика для менеджера. — М.: Логос, 2000.
25. *Носов В.А.* Комбинаторика и теория графов. — М.: МГУ, 1999.
26. *Оре О.* Теория графов. — М.: Наука, 1980.
27. *Очков В.Ф.* Mathcad 12 для студентов и инженеров. — СПб.: БХВ-Петербург, 2005.
28. *Показеев В.В., Матяш В.И., Черкесова Г.В., Кирсанов М.Н.* Элементы дискретной математики. Курс лекций. — М.: МГТУ «МАМИ», 2004.
29. *Редькин Н.П.* Дискретная математика. — СПб.: «Лань», 2003.
30. *Судоплатов С.В., Овчинникова Е.В.* Элементы дискретной математики. — М.: ИНФРА-М, Новосибирск: Изд-во НГТУ, 2002.
31. *Фляйшнер Г.* Эйлеровы графы и смежные вопросы. — М.: Мир, 2002.
32. *Хаггарти Р.* Дискретная математика для программистов. — М.: Техносфера, 2003.
33. *Харари Ф.* Теория графов. — М.: Едиториал УРСС, 2003.
34. *Штовба С.Д.* Муравьиные алгоритмы// Exponenta Pro. Математика в приложениях. 2004.№4(4)

Предметный и именной указатель

Учебное издание

КИРСАНОВ Михаил Николаевич

Графы в Maple

Задачи, алгоритмы, программы

Редактор Константинова О. А.
Оригинал-макет автора

Оформление переплета: *А.А. Логунов*

ЛР № 020528 от 05.06.97.

Подписано в печать с оригинал-макета ***** Формат 60×84/16.
Бумага офсетная. Печать офсетная.
Усл.печ. л. 10,5 Тираж 5000 экз. Заказ