ГРАФЫ

Введение

Графы возникли в восемнадцатом веке, когда известный математик Леонард Эйлер пытался решить теперь уже классическую задачу о Кенигсбергских мостах. В то время в городе Кенигсберге было два острова, соединенных семью мостами с берегами реки Преголь и друг с другом так, как показано на рис. 7.1.

Задача состоит в следующем: осуществить прогулку по городу таким образом, чтобы, пройдя ровно по одному разу по каждому мосту, вернуться в то же место, откуда начиналась прогулка. Решая эту задачу, Эйлер изобразил Кенигсберг в виде графа, отождествив его вершины с частями города, а ребра — с мостами, которыми связаны эти части.

Как мы увидим, Эйлеру удалось доказать, что искомого маршрута обхода города не существует.

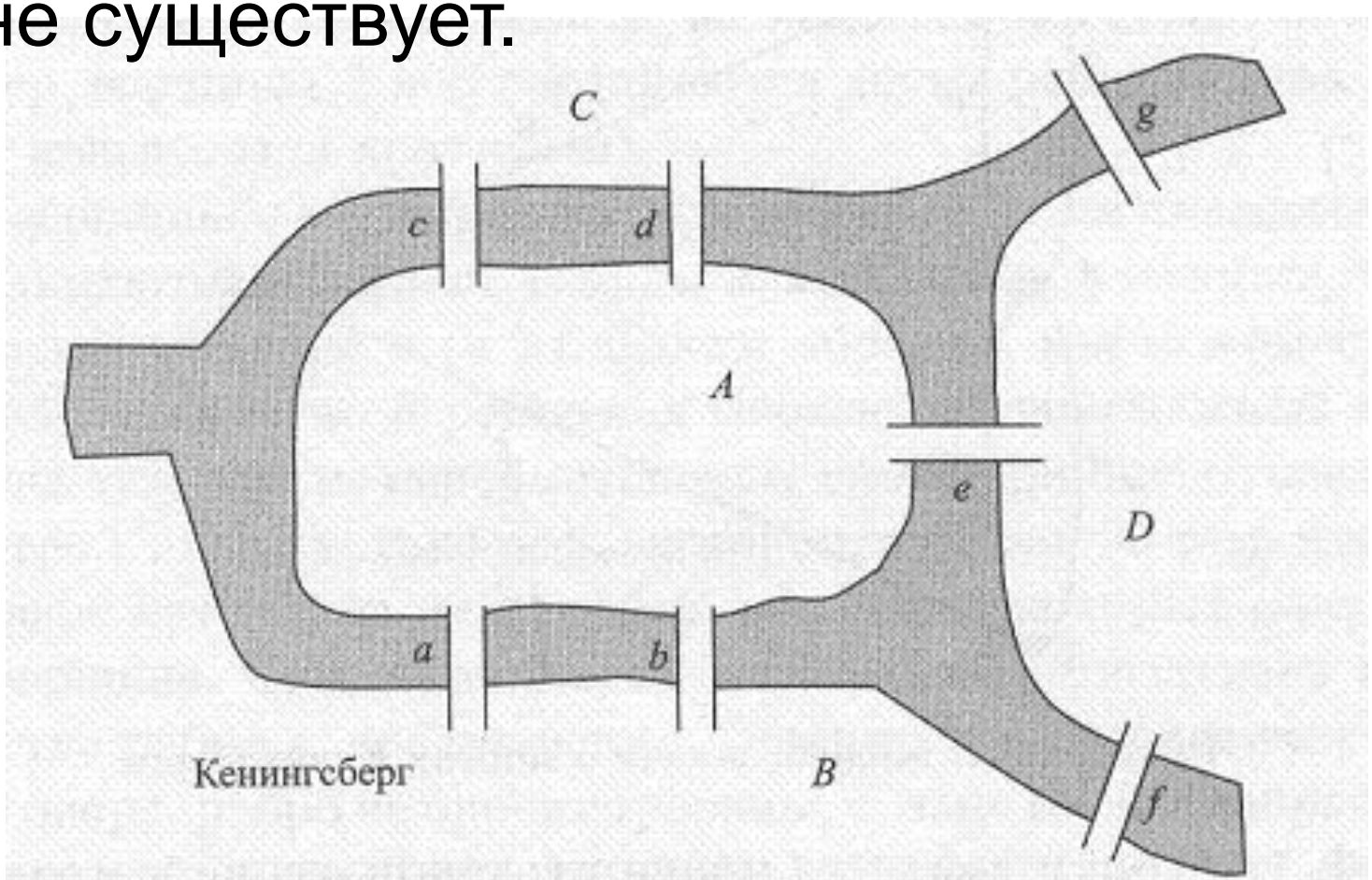

Рисунок 7.1. Схема старого Кенигсберга

В этом разделе мы введем стандартную терминологию, используемую в теории графов, и разберем несколько конкретных задач, решаемых с помощью графов. В частности, мы познакомимся с классом графов, называемым деревьями.

Деревья — естественная модель, представляющая данные, организованные в иерархическую систему. Поиск по дереву для выделения отдельных предметов и сортировка данных в дереве представляют собой важные точки приложения усилий в информатике. В приложении к этому разделу мы займемся *сортировкой и поиском* данных, организованных в деревья.

Графы. Терминология

На рис. 7.1 изображены семь мостов Кенигсберга так, как они были расположены в восемнадцатом веке. В задаче, к которой обратился Эйлер, спрашивается: можно ли найти маршрут прогулки, который проходит ровно один раз по каждому из мостов и начинается и заканчивается в одном и том же месте города?

Модель задачи — это *граф*, состоящий из множества *вершин* и множества *ребер*, соединяющих вершины. Вершины *A*, *B*, *C* и *D* символизируют берега реки и острова, а ребра *a*, *b*, *c, d*, *e*, *f* и *g* обозначают семь мостов (см. рис. 7.2). Искомый маршрут (если он существует) соответствует обходу ребер графа таким образом, что каждое из них проходится только один раз.

Проход ребра, очевидно, соответствует пересечению реки по мосту.

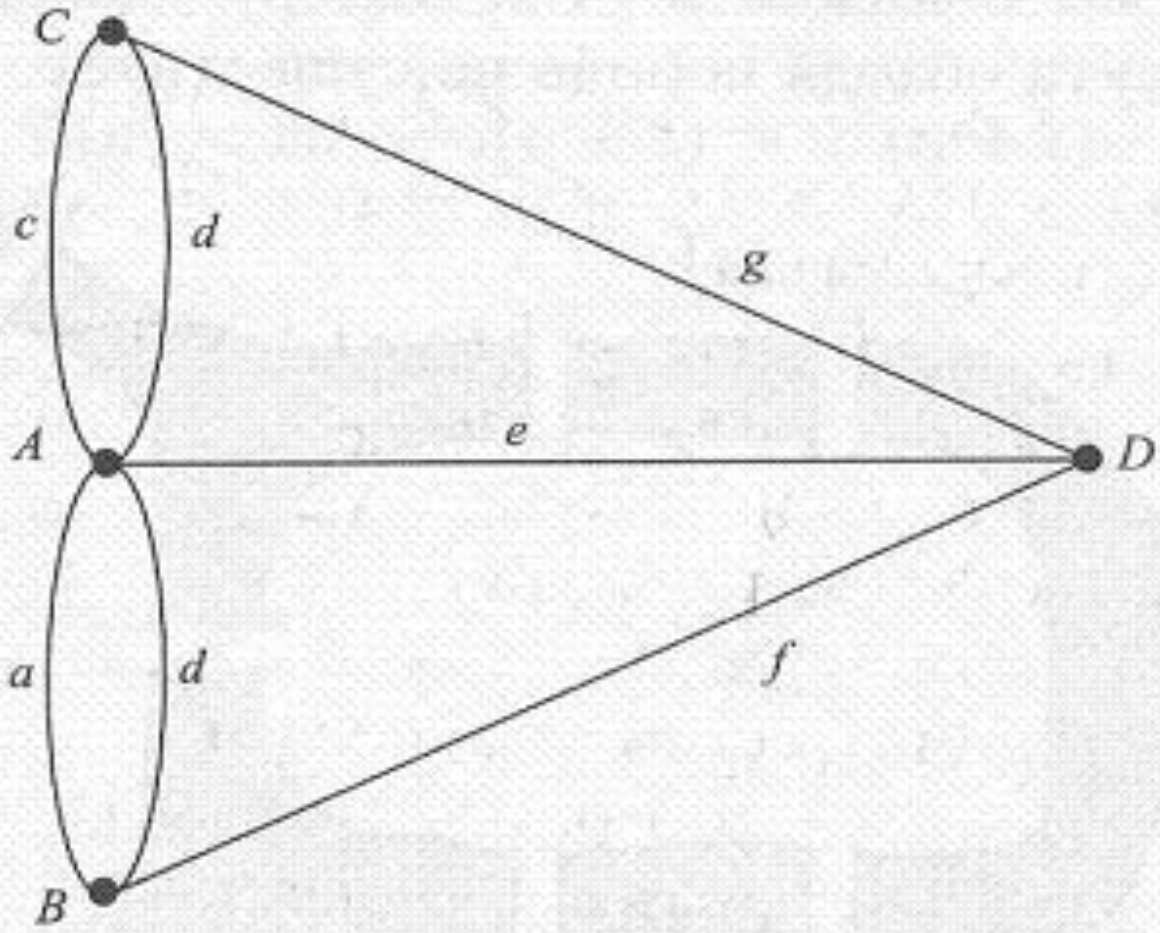

Рисунок 7.2. Модель задачи о мостах Кенигсберга

Граф, в котором найдется маршрут, начинающийся и заканчивающийся в одной вершине, и проходящий по всем ребрам графа ровно один раз, называется *эйлеровым графом*. Последовательность вершин (может быть и с повторениями), через которые проходит искомый маршрут, как и сам маршрут, называется *эйлеровым циклом*.

Эйлер заметил, что если в графе есть эйлеров цикл, то для каждого ребра, ведущего в какую-то вершину, должно найтись другое ребро, выходящее из этой вершины, и получил из этого простого наблюдения такой вывод: если в данном графе существует эйлеров цикл, то к каждой вершине должно подходить четное число ребер.

Кроме того, Эйлеру удалось доказать и противоположное утверждение, так что граф, в котором любая пара вершин связана некоторой последовательностью ребер, является Эйлеровым тогда и только тогда, когда все его вершины имеют четную степень. *Степенью* вершины v называется число $\delta(v)$ ребер, ей *инцидентных*.

Если вершина *v* графа является концом ребра *x*, то говорят, что *v* и *x* инцидентны.

Теперь совершенно очевидно, что в графе, моделирующем задачу о мостах Кенигсберга, эйлерова цикла найти нельзя.

Действительно, степени всех его вершин нечетны: $\delta(B) = \delta(C) = \delta(D) = 3$ и $\delta(A) = 5$.

С легкой руки Эйлера графы, подобные тому, который мы исследовали при решении задачи о мостах, стали использоваться при решении многих практических задач, а их изучение выросло в значительную область математики.

Простой граф определяется как пара $G = (V, E)$, где V — конечное множество вершин, а E — конечное множество ребер, причем G не может содержать *петель* (ребер, начинающихся и заканчивающихся в одной вершине) и *кратных ребер* (кратными называются несколько ребер, соединяющих одну и ту же пару вершин).

Граф, изображенный на рис. 7.2, не является простым, поскольку, например, вершины *A* и *B* соединяются двумя ребрами (как раз эти ребра и называются кратными).

Две вершины *u* и *v* в простом графе называются *смежными,* если они соединяются каким-то ребром *e,* про которое говорят, что оно *инцидентно* вершине *u* (и вершине *v*). Таким образом, мы можем представлять себе множество *E* ребер как множество пар смежных вершин, определяя тем самым нерефлексивное, симметричное отношение на множестве *V*.

Отсутствие рефлексивности связано с тем, что в простом графе нет петель, т. е. ребер, оба конца которых находятся в одной вершине. Симметричность же отношения вытекает из того факта, что ребро *e*, соединяющее вершину *u* с *v,* соединяет и *v* с *u* (иначе говоря, ребра не ориентированы, т. е. не имеют направления). Единственное ребро простого графа, соединяющее пару вершин *u* и *v,* мы будем обозначать как *uv* (или *vu*).

Логическая матрица отношения на множестве вершин графа, которое задается его ребрами, называется *матрицей смежности*.

Симметричность отношения в терминах матрицы смежности *M* означает, что *M* симметрична относительно главной диагонали. А из-за нерефлексивности этого отношения на главной диагонали матрицы *M* стоит символ «Л» - ложь.

Пример 7.1. Нарисуйте граф $G(V, E)$ с множеством вершин $V = \{a, b, c, d, e\}$ и множеством ребер $E = \{ab, ae, bc, bd, ce, de\}$. Выпишите его матрицу смежности.

Решение. Граф G показан на рис. 7.3.

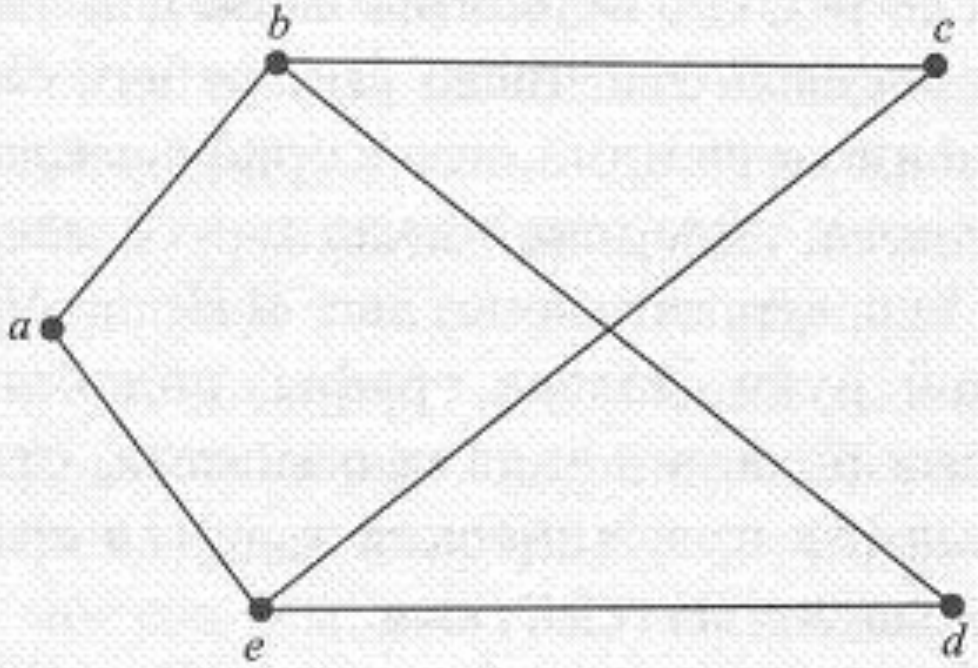

Рисунок 7.3.

Его матрица смежности имеет вид:

	a	b	c	d	e
a	Л	И	Л	Л	И
b	И	Л	И	И	Л
c	Л	И	Л	Л	И
d	Л	И	Л	Л	И
e	И	Л	И	И	Л

Для восстановления графа нам достаточно только тех элементов матрицы смежности, которые стоят над главной диагональю.

Подграфом графа $G = (V, E)$ называется граф $G' = (V', E')$, для которого $E' \subset E$ и $V' \subset V$.

Пример 7.2. Найдите среди графов *H*, *K* и *L*, изображенных на рис. 7.4, подграфы графа *G*.

Решение. Обозначим вершины графов *G*, *H* и *K* как показано на рис. 7.5. Графы *H* и *K* — подграфы *G*, как видно из обозначений. Сразу можно сказать - граф *L* не является подграфом *G*, поскольку видно, что у него есть вершина степени 4, а у графа *G* такой нет.

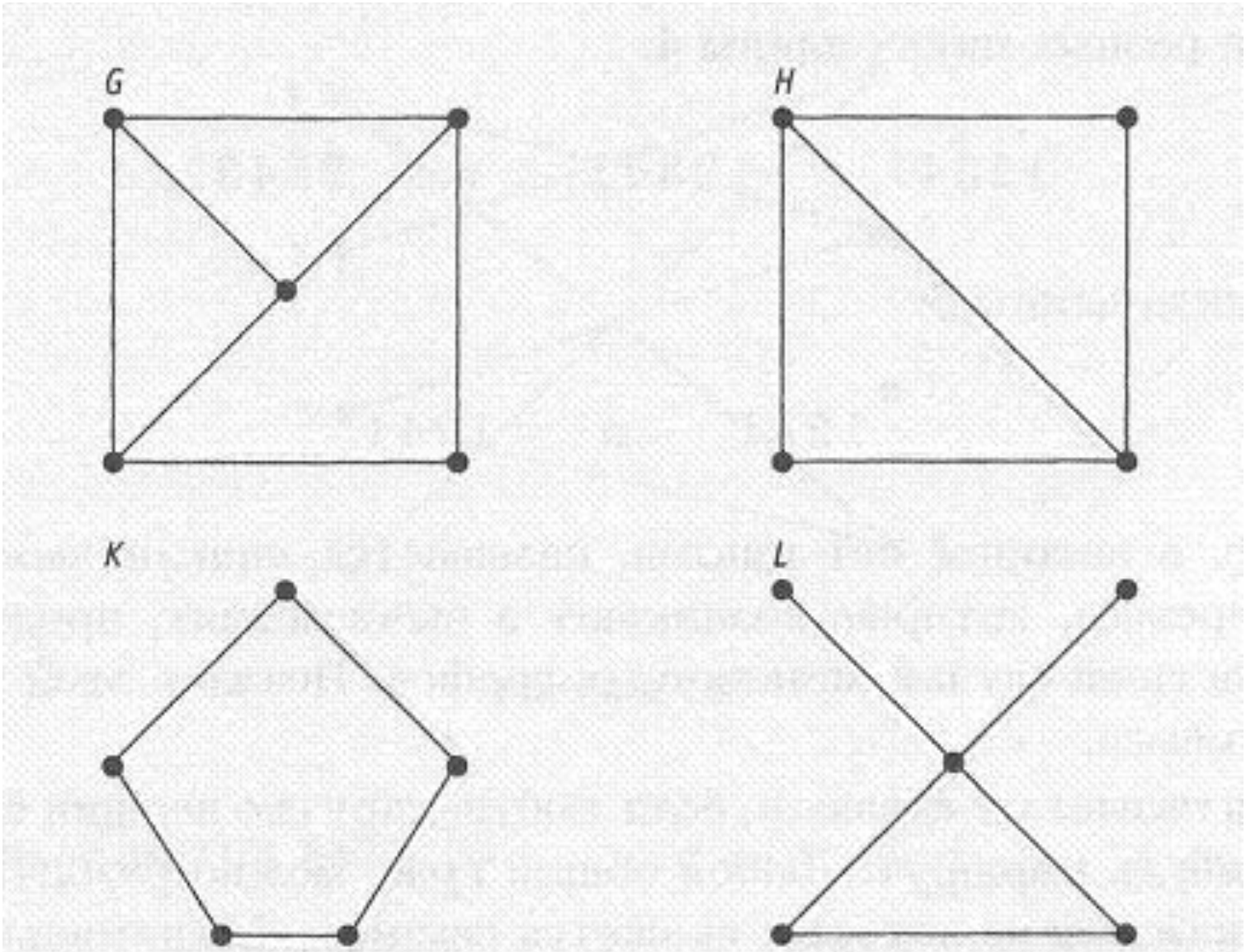

Рисунок 7.4.

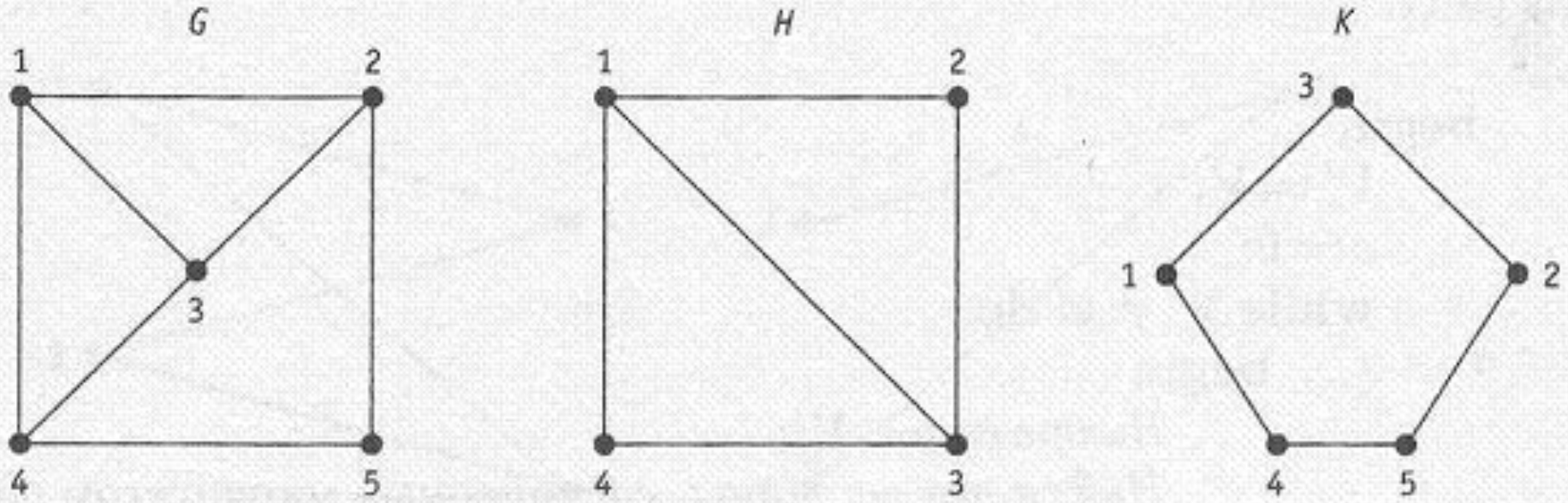

Рисунок 7.5.

Маршрутом длины k в графе G называется такая последовательность вершин v_0, v_1, ..., v_k, что для каждого $i = 1, ..., k$ пара v_{i-1} v_i образует ребро графа.

Такой маршрут обозначается v_0 v_1 ... v_k.

Например, 1 4 3 2 5 — это маршрут длины 4 в графе G из примера 7.2.

Циклом в графе называется последовательность вершин v_0, v_1, ..., v_k, каждая пара которых является концами одного ребра, причем $v_0 = v_k$, а остальные вершины (и ребра) не повторяются.

Иными словами, цикл — это замкнутый маршрут, проходящий через каждую свою вершину и ребро только один раз.

Пример 7.3. Найдите циклы в графе *G* из примера 7.2.

Решение. В этом графе есть два разных цикла длины 5 (см. рис.7.5):

1 3 2 5 4 1 и 1 2 5 4 3 1.

Мы можем пройти эти циклы как в одном направлении, так и в другом, начиная с произвольной вершины цикла. Кроме того, в графе есть три разных цикла длины 4:

1 2 5 4 1, 1 2 3 4 1 и 2 5 4 3 2,

и два цикла длины 3:

1 2 3 1 и 1 3 4 1.

Граф, в котором нет циклов, называется *ацикличным* (или *ациклическим*).

Частным случаем ациклических графов являются деревья.

Граф называется *связным*, если любую пару его вершин соединяет какой-нибудь маршрут.

Любой граф можно разбить на связные подграфы.

Минимальное число таких связных компонент называется *числом связности* графа и обозначается через $c(G)$.

Вопросы связности имеют важное значение в приложениях теории графов к компьютерным сетям.

Следующий алгоритм применяется для определения числа связности графа.

Алгоритм связности

Пусть $G = (V, E)$ — граф. Алгоритм предназначен для вычисления значения $c = c(G)$, т.е. числа компонент связности данного графа G.

begin

$V' := V$;

$c := 0$;

while $V' \neq \varnothing$ **do begin**

выбрать $y \in V'$;

найти все вершины, соединенные маршрутом с *y;* удалить вершину *y* из *V' u*
соответствующие ребра из *E*;
c := *c* + 1;
end
end

Пример 7.4. Проследите за работой алгоритма связности на графе, изображенном на рис. 7.6.

Решение. См. табл. 7.1:

Таблица 7.1

	V'	c
Исходные значения	$\{1, 2, 3, 4, 5, 6, 7, 8\}$	0
Выбор $y = 1$	$\{2, 4, 5, 7\}$	1
Выбор $y = 2$	$\{7\}$	2
Выбор $y = 7$	$\varnothing$	3

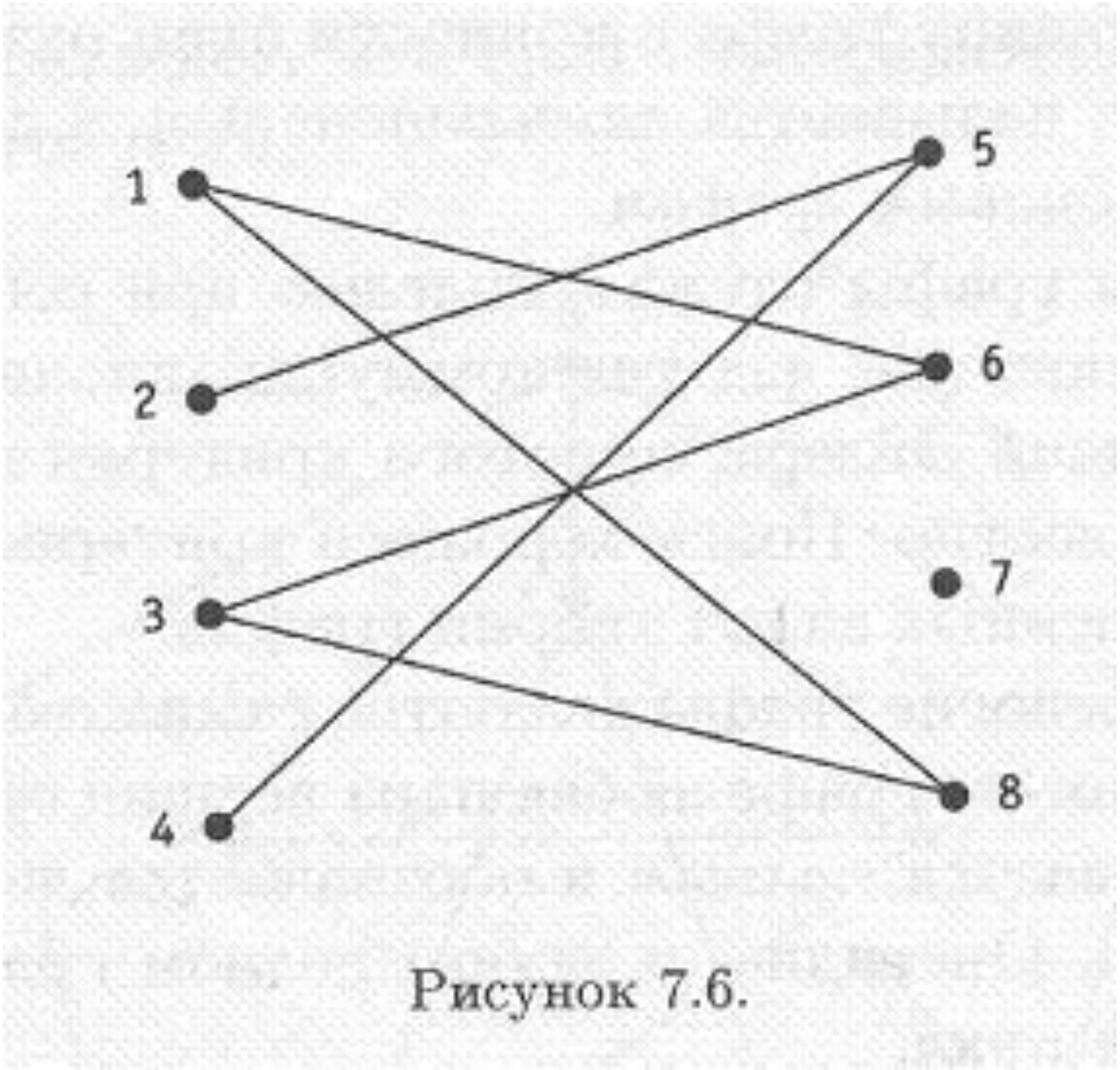

Рисунок 7.6.

На первом шаге удалены вершины 1,6,3,8. В итоге, $c(G) = 3$. Соответствующие компоненты связности приведены на рис. 7.7.

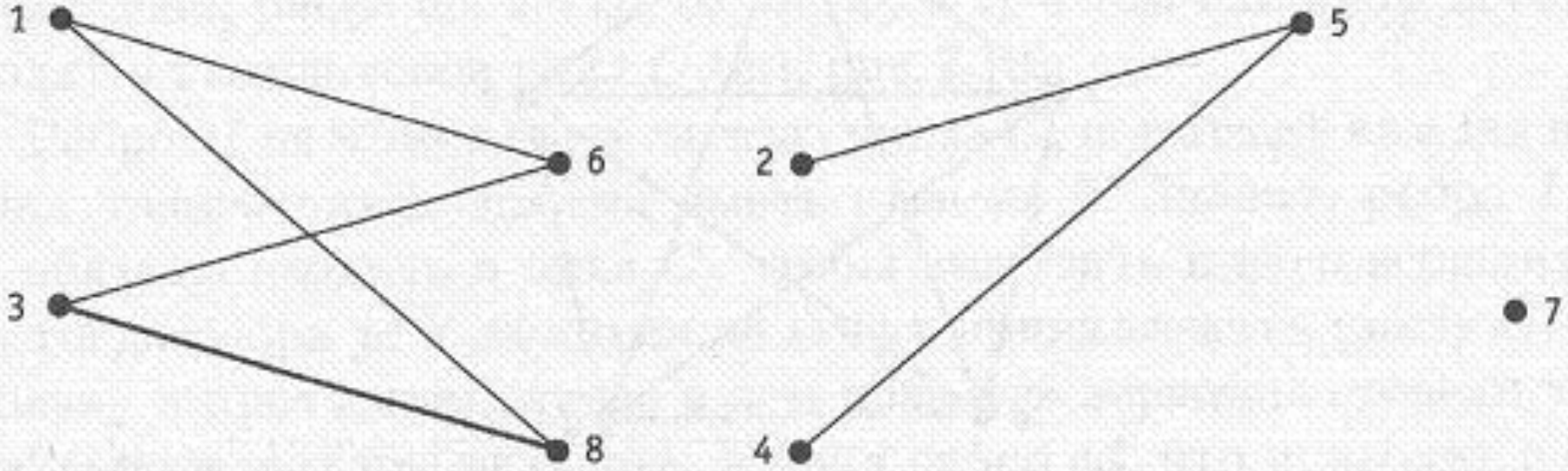

Рисунок 7.7.

Гамильтоновы графы

Мы начали этот раздел с изучения эйлеровых графов, обладающих замкнутым маршрутом, который проходит по всем ребрам графа ровно один раз. Похожая задача состоит в поиске цикла, проходящего через каждую вершину графа в точности один раз. Такой цикл, если он существует, называется *гамильтоновым*, а соответствующий граф — *гамильтоновым графом*.

Гамильтоновы графы служат моделью при составлении расписания движения поездов, для телекоммуникационных сетей и т.д.

В отличие от задачи Эйлера, простого критерия гамильтоновости графа пока не известно.

Поиск хорошего критерия остается одной из главных нерешенных задач теории графов.

Тем не менее, многие графы являются гамильтоновыми.

Если в графе ***любая*** пара вершин соединена ребром, то такой граф называется ***полным*** и обозначается через K_n, где n — число его вершин.

Очевидно, в любом полном графе можно найти гамильтонов цикл.

Полный граф K_5 изображен на рис. 7.8.

Его цикл *a b c d e a,* очевидно, является гамильтоновым.

В нем есть и другие гамильтоновы циклы.

Поскольку каждая вершина смежна с остальными, то начиная с вершины *a*, в качестве второй вершины цикла можно выбрать любую из четырех оставшихся вершин.

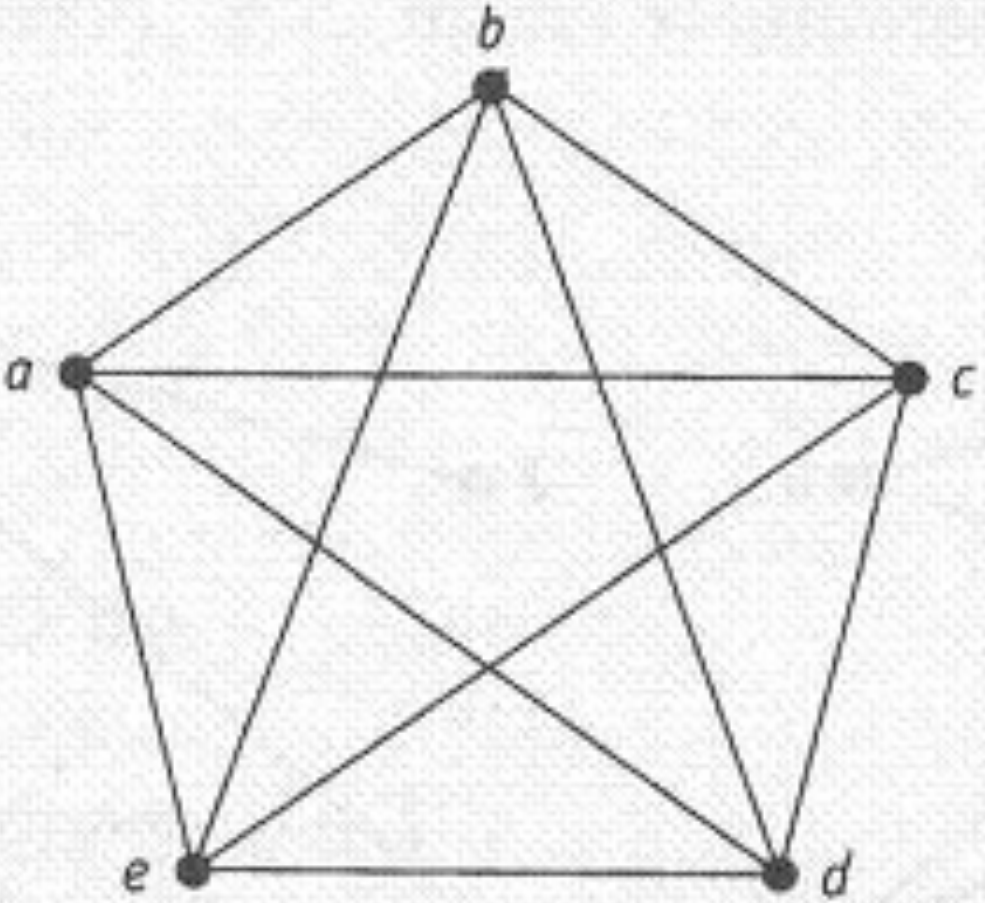

Рисунок 7.8. Полный граф K_5

Далее будет три варианта для выбора третей вершины и два для четвертой, после чего мы вернемся в вершину *a.* Таким образом, у нас есть $4 \cdot 3 \cdot 2 = 24$ цикла.

Поскольку каждый цикл можно проходить как в одном направлении, так и в другом, то реально в графе K_5 есть только 12 разных гамильтоновых циклов.

Поиск гамильтонова цикла (если он существует) в произвольном (связном) графе — задача далеко не всегда простая.

Ответ на вопрос о гамильтоновости графа может оказаться довольно трудоемким.

Пример 7.5. Покажите, что граф, изображенный на рис. 7.9, не является гамильтоновым.

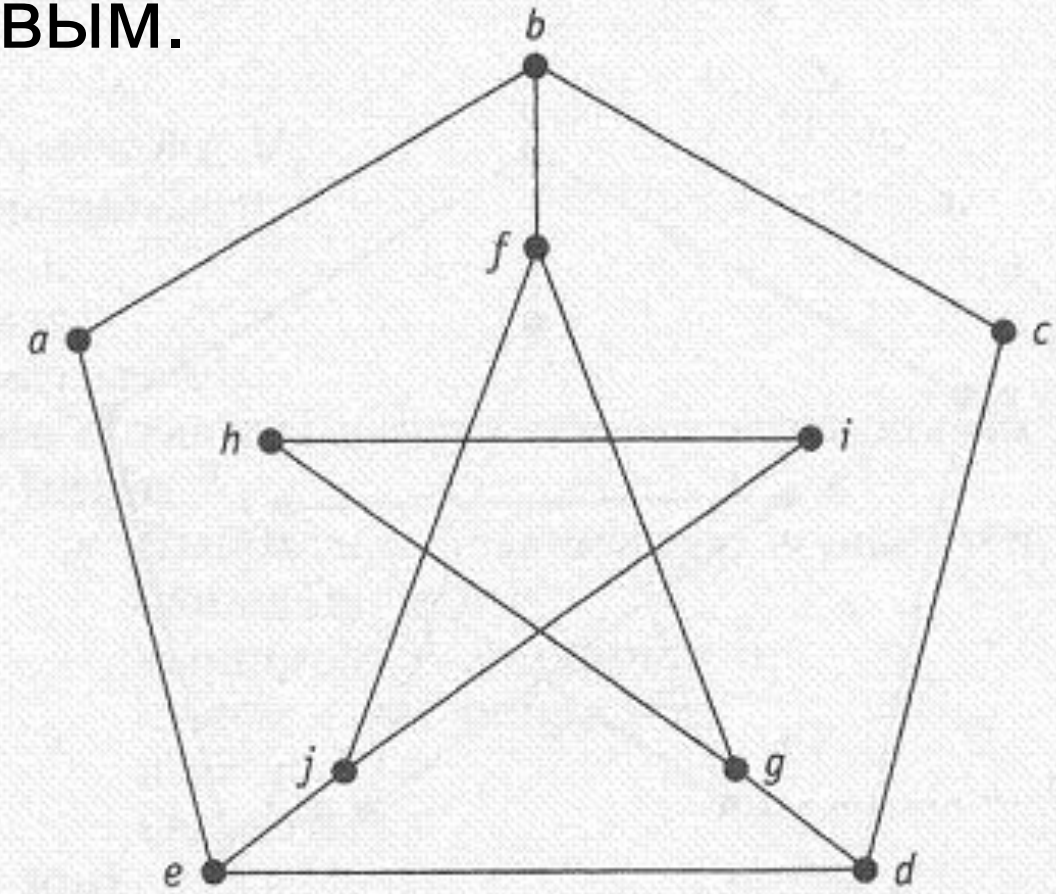

Рисунок 7.9. Пример не гамильтонова графа

Решение. Предположим, что в связном графе найдется гамильтонов цикл.

Каждая вершина *v* включается в гамильтонов цикл *C* выбором двух инцидентных с ней ребер, а значит, степень каждой вершины в гамильтоновом цикле (после удаления лишних ребер) равна 2.

Все степени вершин данного графа — только 2 или 3.

Вершины степени 2 должны входить в возможный цикл вместе с обоими инцидентными с ними ребрами.

Следовательно, ребра *ab*, *ae*, *cd*, *cb*, *hi*, *hg* и *ij* в том или ином порядке могут входить в возможный гамильтонов цикл *C,* если, конечно, он существует (см. рис. 7.10).

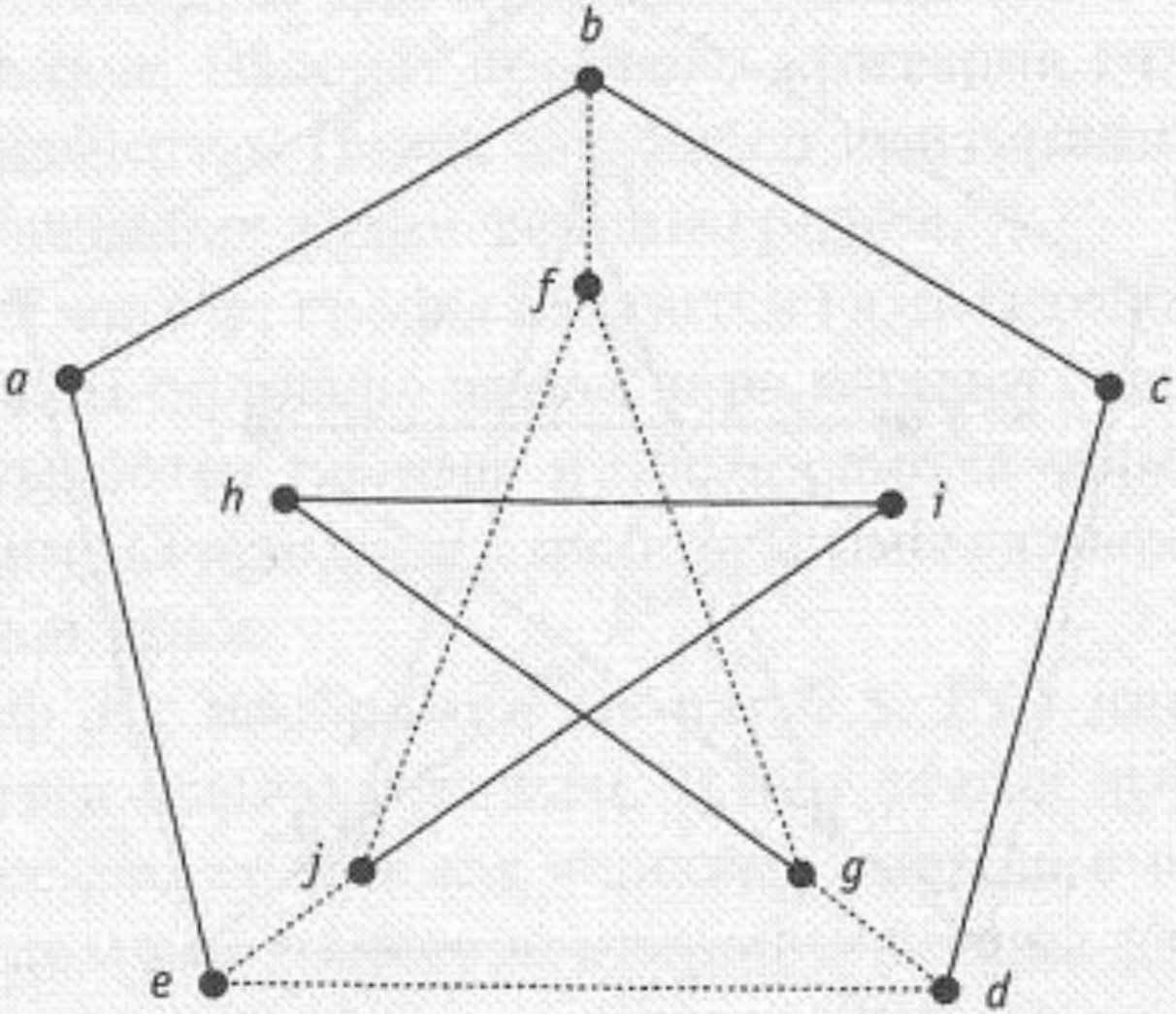

Рисунок 7.10. Ребра, входящие в гамильтонов цикл C

Ребро *bf* не может быть частью цикла *C*, поскольку каждая вершина такого цикла должна иметь степень 2 (а у вершины *b* тогда будет степень 3).

Значит, ребра *fj* и *fg* обязаны входить в цикл *C*, чтобы включить в этот цикл вершину *f*.

Но тогда ребра *je* и *gd* никак не могут принадлежать циклу *C*, поскольку в противном случае в нем появятся вершины (*d* и *e*) степени 3.

Это вынуждает нас включить в цикл ребро *ed*, что приводит к противоречию:

ребра, которые мы были вынуждены выбрать, образуют *два* несвязных цикла, а не *один*, существование которого мы предполагали.

Вывод: граф, изображенный на рис. 7.10, не является гамильтоновым.

Гамильтоновы графы применяются для моделирования многих практических задач.

Основой всех таких задач служит классическая *задача коммивояжера*, формулировка которой приведена ниже.

Коммивояжер должен совершить поездку по городам и вернуться обратно, побывав в каждом городе ровно один раз, сведя при этом затраты на передвижения к минимуму.

Графическая модель задачи коммивояжера состоит из гамильтонова графа, вершины которого изображают города, а ребра — связывающие их дороги.

Кроме того, каждое ребро оснащено *весом,* обозначающим транспортные затраты, необходимые для путешествия по соответствующей дороге, такие, как, например, расстояние между городами, стоимость проезда или время движения по дороге.

Граф, каждое ребро которого снабжено весом, называется *нагруженным*.

Для решения задачи нам необходимо найти гамильтонов цикл *минимального общего веса*.

К сожалению, эффективный алгоритм решения данной задачи пока не известен.

Для сложных сетей число гамильтоновых циклов, которые необходимо просмотреть для выделения минимального, непомерно огромно.

Однако существуют алгоритмы поиска *субоптимального решения*.

Субоптимальное решение необязательно даст цикл минимального общего веса, но найденный цикл будет, как правило, значительно меньшего веса, чем большинство произвольных гамильтоновых циклов.

Один из таких алгоритмов мы сейчас изучим.

Алгоритм ближайшего соседа

Этот алгоритм выдает субоптимальное решение задачи коммивояжера, генерируя гамильтоновы циклы в нагруженном графе с множеством вершин *V*.

Цикл, полученный в результате работы алгоритма, будет совпадать с конечным значением переменной *маршрут,* а его общая длина — конечное значение переменной *w*.

begin

выбрать $v \in V$;

маршрут := *v*;

w := *0*;

v' :=*v*;

отметить *v'*;

while остаются неотмеченные вершины **do**

begin

выбрать неотмеченную вершину *u*,

ближайшую к *v'*;

маршрут := маршрут u;

w : = w + **вес ребра** *v'u;*

v' := u;

отметить *v';*

end

маршрут := маршрут v;

w := w + **вес ребра** *v'v;*

end

Пример 7.6. Примените алгоритм ближайшего соседа к графу, изображенному на рис. 7.11. За исходную вершину возьмите вершину *D*.

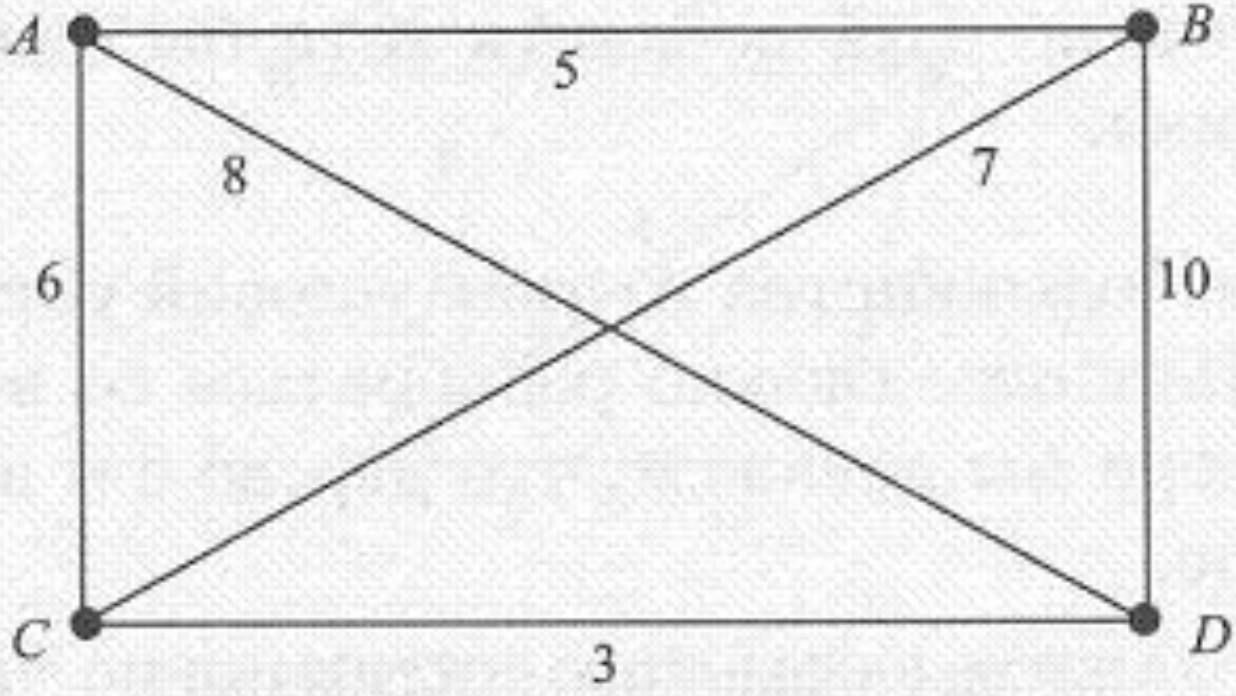

Рисунок 7.11.

Решение. См. табл. 7.2.

Таблица 7.2

	u	*маршрут*	*w*	*v'*
Исходные значения	-	*D*	0	*D*
	C	*DC*	3	*C*
	A	*DCA*	9	*A*
	B	*DCAB*	14	*B*
Последний проход	*B*	*DCABD*	24	*B*

В результате работы алгоритма был найден гамильтонов цикл *D C A B D* общего веса 24.

Делая полный перебор всех циклов в этом маленьком графе, можно обнаружить еще два других гамильтоновых цикла: *A B C D A* общего веса 23 и *A C B D A* общего веса 31.

В полном графе с двадцатью вершинами существует приблизительно $6{,}1 \cdot 10^{16}$ гамильтоновых циклов, перечисление которых требует чрезвычайно много машинной памяти и времени.

Деревья

Как упоминалось ранее в этом разделе, есть класс графов, называемых *деревьями*, которые особенно интенсивно используются в вычислительных приложениях.

Граф $G = (V, E)$ называется *деревом,* если он связен и ацикличен (т.е. не содержит циклов).

Пусть $G = (V, E)$ — граф с n вершинами и m ребрами. Можно сформулировать несколько необходимых и достаточных условий, при которых граф G является деревом:

- любая пара вершин в графе G соединена единственным путем;
- граф G связен и $m = n - 1$;
- граф G связен, а удаление хотя бы одного его ребра нарушает связность графа;
- граф G ацикличен, но если добавить хотя бы одно ребро, то в G появится цикл.

Эквивалентность большинства из этих условий устанавливается без особого труда.

Наиболее сложно разобраться со вторым из них.

В следующем примере мы докажем, что дерево с n вершинами имеет ровно n - 1 ребро.

Пример 7.7. Докажите с помощью индукции по числу вершин, что для дерева T с n вершинами и m ребрами выполнено соотношение: $m = n - 1$.

Решение. Поскольку дерево с единственной вершиной вообще не содержит ребер, то доказываемое утверждение справедливо при $n = 1$.

Рассмотрим дерево T с n вершинами (и m ребрами), где $n > 1$ и предположим, что любое дерево с $k < n$ вершинами имеет ровно $k - 1$ ребро.

Удалим ребро из T. По третьему свойству дерево T после этой процедуры превратится в несвязный граф.

Получится ровно две компоненты связности, ни одна из которых не имеет циклов (в противном случае исходный граф T тоже содержал бы циклы и не мог бы быть деревом).

Таким образом, полученные компоненты связности — тоже деревья.

Обозначим новые деревья через T_1 и T_2. Пусть n_1 — количество вершин у дерева T_1, а n_2 — у T_2. Поскольку $n_1 + n_2 = n$, то $n_1 < n$ и $n_2 < n$.

По предположению индукции дерево T_1 имеет $n_1 - 1$ ребро, а T_2 — $n_2 - 1$.

Следовательно, исходное дерево T насчитывало (с учетом одного удаленного) $(n_1 - 1) + (n_2 - 1) + 1 = n - 1$ ребро, что и требовалось доказать.

Несложно доказать, что в любом связном графе найдется подграф, являющийся деревом.

Подграф графа *G*, являющийся деревом и включающий в себя все вершины графа *G*, называется *остовным деревом*.

Остовное дерево в графе *G* строится просто: выбираем произвольное его ребро и последовательно добавляем другие ребра, не создавая при этом циклов, до тех пор, пока нельзя будет добавить никакого ребра, не получив при этом цикла.

Благодаря примеру 7.7, мы знаем, что для построения остовного дерева в графе из *n* вершин необходимо выбрать ровно *n* - 1 ребро.

Пример 7.8. Найдите два разных остовных дерева в графе, изображенном на рис. 7.12.

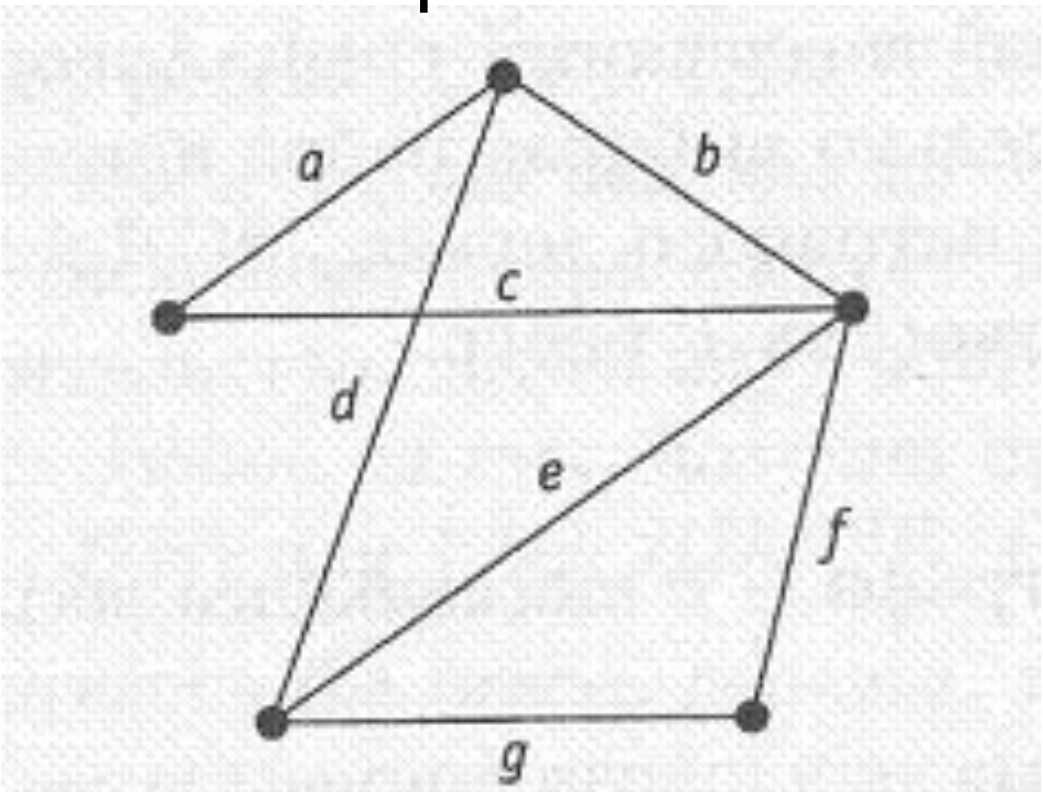

Рисунок 7.12. Связный граф G

Решение. В этом графе существует несколько остовных деревьев. Одно из них получается последовательным выбором ребер: a, *b*, *d* и *f*. Другое — *b*, *c*, *e* и *d*. Названные деревья показаны на рис. 7.13.

Процесс, описанный в примере 7.8, можно приспособить для решения *задачи поиска кратчайшего соединения*:

Нужно построить железнодорожную сеть, связывающую некоторое число городов. Известна стоимость строительства отрезка путей между любой парой городов. Требуется найти сеть минимальной стоимости.

На языке теории графов нам нужно в нагруженном графе найти остовное дерево наименьшего общего веса. Такое дерево принято называть *минимальным остовным деревом* или, сокращенно, *МОД*.

В отличие от задачи коммивояжера, здесь есть эффективный алгоритм, находящий действительно минимальное остовное дерево. Он похож на алгоритм Прима, с которым мы познакомились в главе 1 при решении задачи поиска кратчайшего соединения для набора из шести шотландских городов.

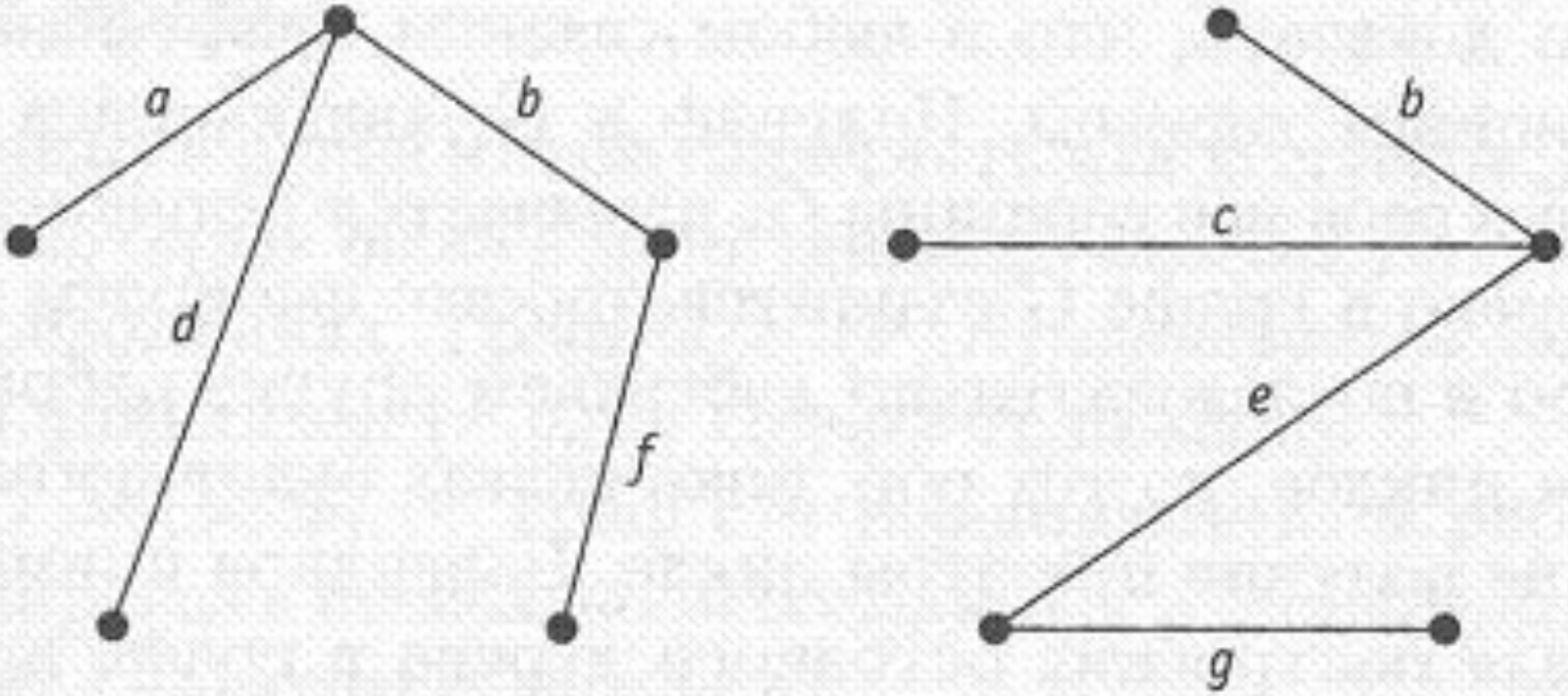

Рисунок 7.13. Остовные деревья графа G

Алгоритм поиска минимального остовного дерева

Пусть

$G = (V, E)$ — связный взвешенный граф. Алгоритм строит МОД в графе G, последовательно выбирая ребра наименьшего возможного веса до образования остовного дерева. МОД в памяти компьютера хранится в виде множества T ребер.

begin

e : = *ребро графа G с наименьшим весом*;

T := {*e*};

E′ := *E* \ {*e*}

while *E′* $\neq \varnothing$ **begin**

e': = *ребро из E' наименьшего веса*;
T:=*T* ∪ {*e*};

E':= *множество ребер из E'* \ {*T*}, *чье добавление к T не ведет к образованию циклов;*

end

end

Пример 7.9. В таблице 7.3 дано расстояние (в милях) между пятью деревнями A, *B*, *C*, *D* и *E*. Найдите минимальное остовное дерево.

Таблица 7.3

	A	*B*	C	*D*	*E*
A	—	13	3	9	9
B	13	—	11	11	13
C	3	11	—	9	7
D	9	11	9	—	2
E	9	13	7	2	—

Решение. Ребра выбираются следующим образом: первое — ребро *DE* веса 2; второе — *AC* веса 3; третье — *CE* веса 7. На этой стадии строящееся дерево выглядит так, как на рис. 7.14

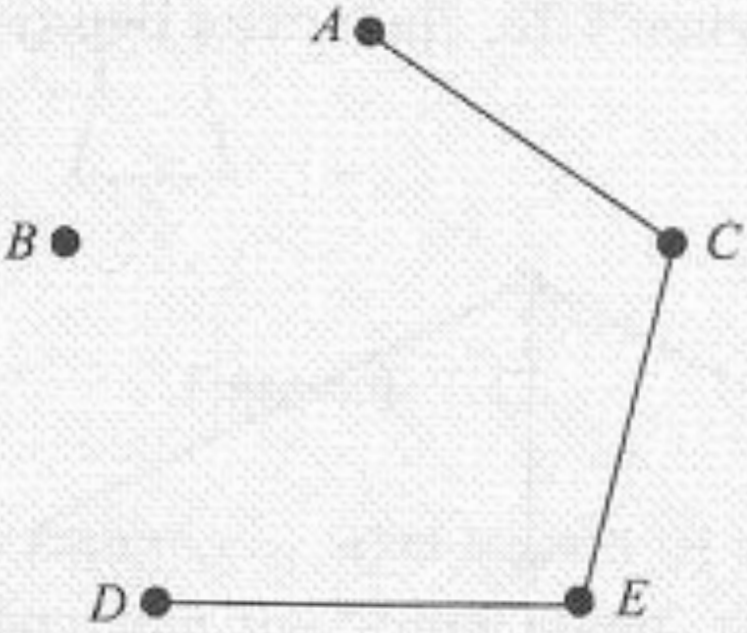

Рисунок 7.14. Вид дерева после трех шагов

Следующие по весу ребра — *AD*, *AE* и *CD*, каждое из которых имеет вес 9. Однако какое бы из них мы ни добавили, получится цикл. Поэтому перечисленные ребра следует исключить из числа доступных для строительства. Далее идут ребра *BC* и *BD* веса 11. Можно присоединить любое из них, получив при этом два разных МОД: {*AC, BC, CE, DE*} или {*AC, BD, CE, DE*} веса 23 каждое.

Зачастую нам хотелось бы иметь деревья, представляющие информацию с учетом естественной иерархической структуры, такие, как, например, генеалогическое древо (рис. 7.15). На нем показаны некоторые члены семьи Бернулли, каждый из которых был известным швейцарским математиком.

Генеалогическое древо можно изобразить и более сжато. Схема, приведенная на рис. 7.16, представляет собой пример так называемого дерева с корнем. *Деревом с корнем* называется дерево с одной выделенной вершиной. Именно эта выделенная вершина и является *корнем* дерева.

Корень, в некотором смысле, можно назвать «величайшей» вершиной (например, родоначальник математиков Бернулли). Вершины дерева, лежащие непосредственно под данной, называются *сыновьями*. С другой стороны, вершина, стоящая непосредственно перед сыном, называется ее *отцом*.

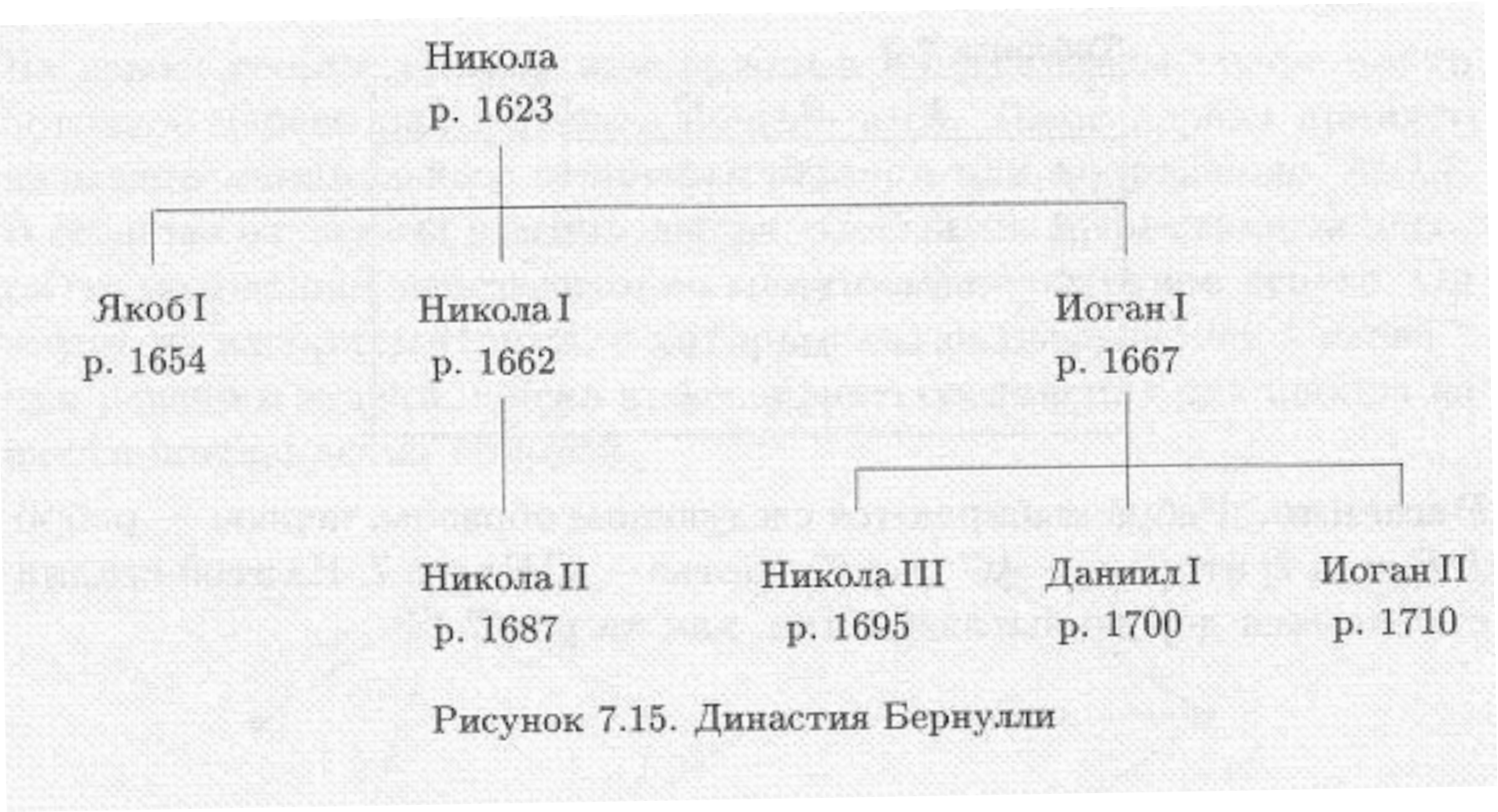

Рисунок 7.15. Династия Бернулли

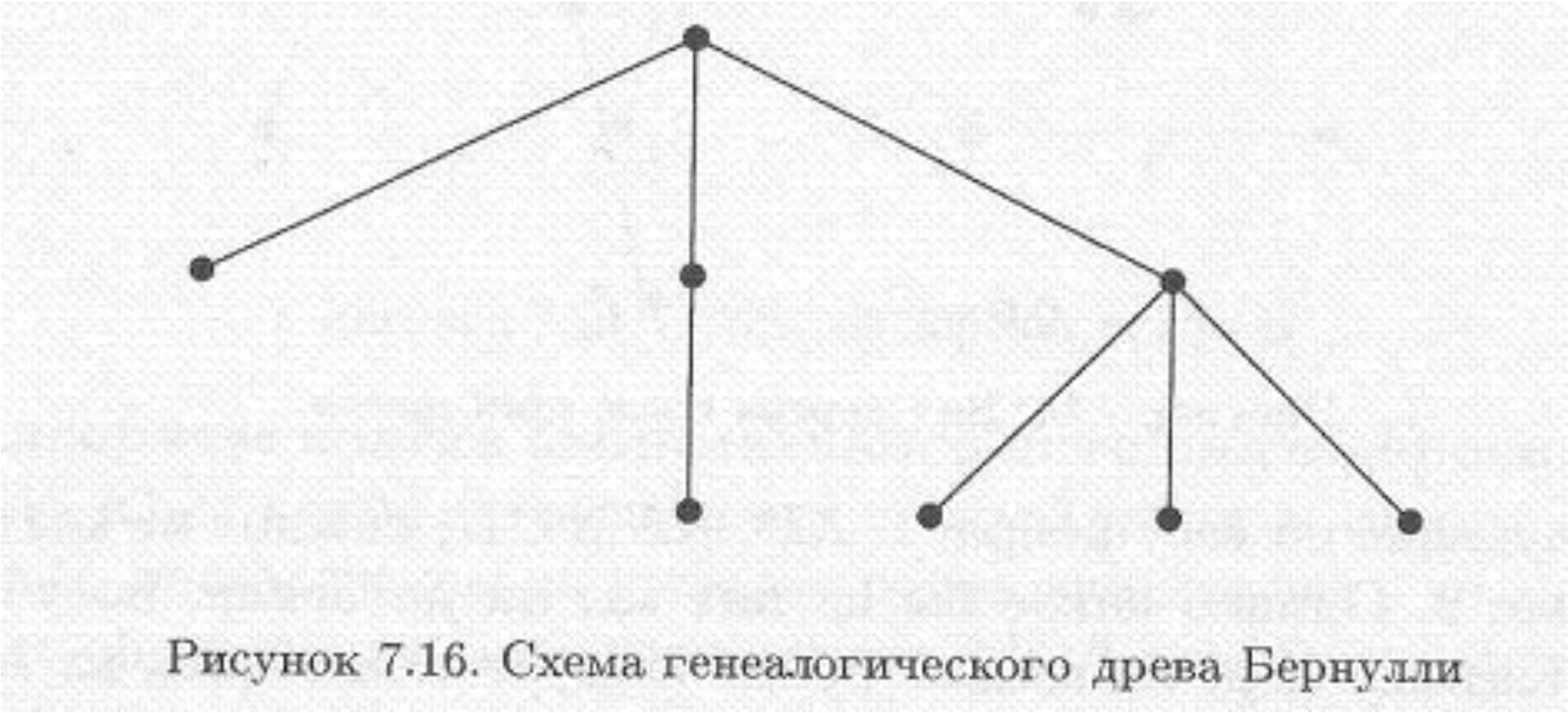

Рисунок 7.16. Схема генеалогического древа Бернулли

Дерево с корнем можно определить рекуррентным образом. Отдельная вершина является деревом с корнем (она же служит и корнем такого дерева). Если T_1, T_2, …, T_k — несвязанные друг с другом деревья с корнями v_1, v_2, …, v_k, то граф, получающийся присоединением новой вершины v к каждой из вершин v_1, v_2, …, v_k отдельным ребром, является деревом T с корнем v.

Вершины $v_1,\ldots, v_k$ графа T — это сыновья корня v. Мы изображаем такое дерево с корнем, расположенным наверху, и сыновьями, стоящими ниже, непосредственно под корнем (см. рис. 7.17). Каждую вершину дерева с корнем можно рассматривать как корень другого дерева, которое «растет» из него. Мы будем называть его *поддеревом* дерева T.

Как мы уже говорили, вершина на самом верху дерева — его корень, а вот те, которые находятся в самом низу дерева (и не имеют сыновей) принято называть *листьями.* Остальные вершины, отличные от корня и листьев, называют *внутренними.*

Область применения деревьев с корнем обширна. Это, например, и информатика, и биология, и менеджмент. Для приложения к информатике наиболее важны так называемые *двоичные* или *бинарные* деревья с корнем. Двоичное дерево отличает от остальных то, что каждая его вершина имеет не более двух сыновей. В двоичном дереве с корнем вниз от каждой вершины идет не более двух ребер.

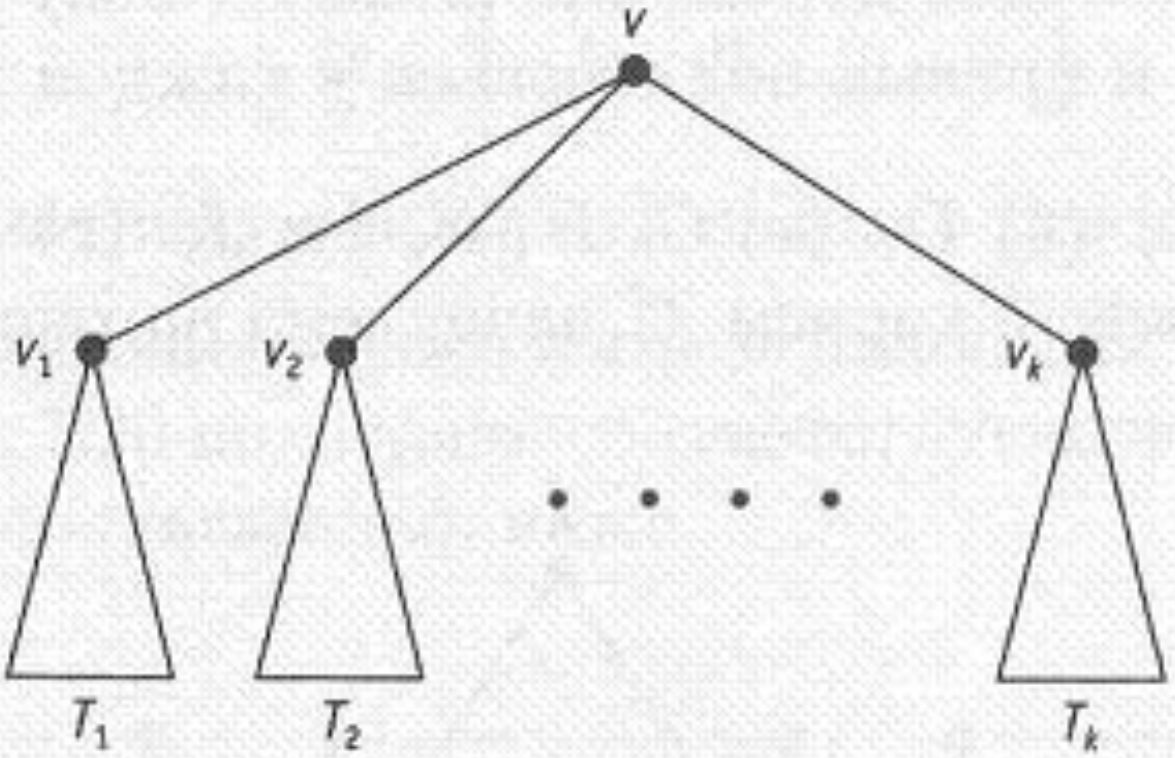

Рисунок 7.17.

Таким образом, можно сказать, что каждой вершине двоичного дерева с корнем соответствует не более, чем два поддерева, которые принято называть *левым* и *правым* поддеревьями этой вершины.

Если оказалось, что у какой-то вершины дерева отсутствует потомок слева, то ее левое поддерево называют *нулевым, деревом* (т.е. нулевое дерево — это дерево без единой вершины). Аналогично, если у вершины отсутствует правый потомок, то ее правое поддерево будет нулевым.

Пример 7.10. Пусть T — двоичное дерево с корнем, изображенное на рис. 7.18.

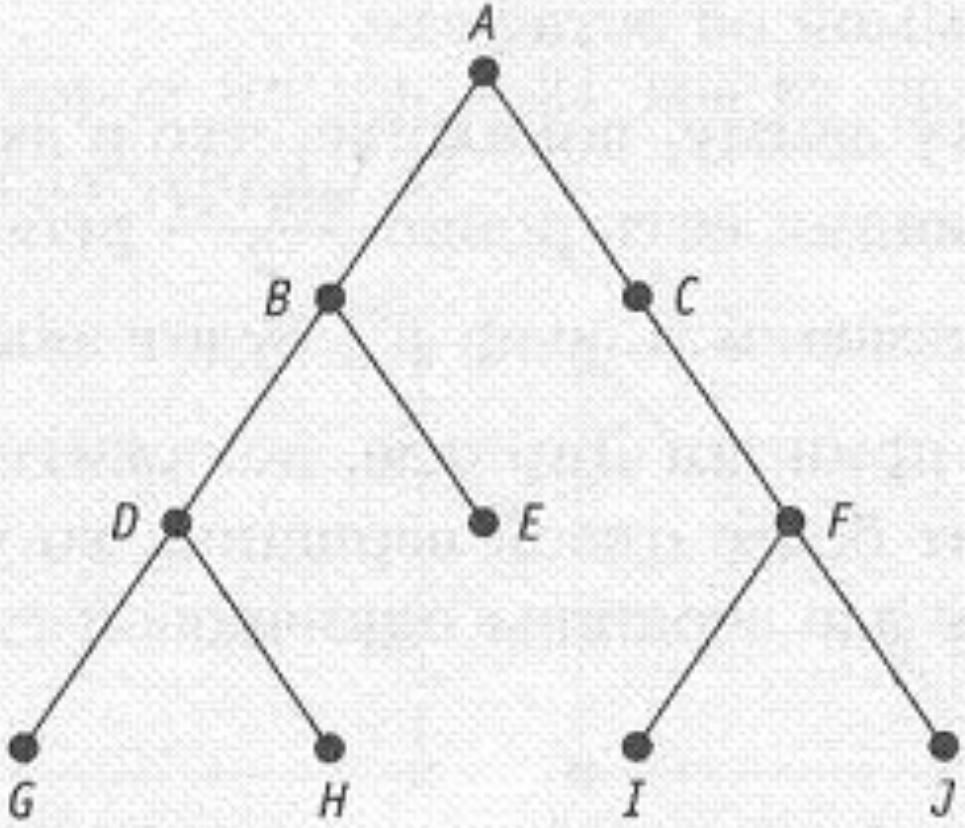

Рисунок 7.18. Двоичное дерево с корнем T

Определите

- (а) корень *T*;
- (б) корень левого поддерева вершины *B*;
- (в) листья *T*;
- (г) сыновей вершины *C.*

Нарисуйте двоичное дерево с корнем *T′*, полученное из *T* перестановкой левых и правых поддеревьев у каждой вершины.

Решение.(а) *A*; (б) *D*; (в) G, *H*, *E*, *I* и *J*; (г) *F*.

Двоичное дерево с корнем *T′* начерчено на рис. 7.19.

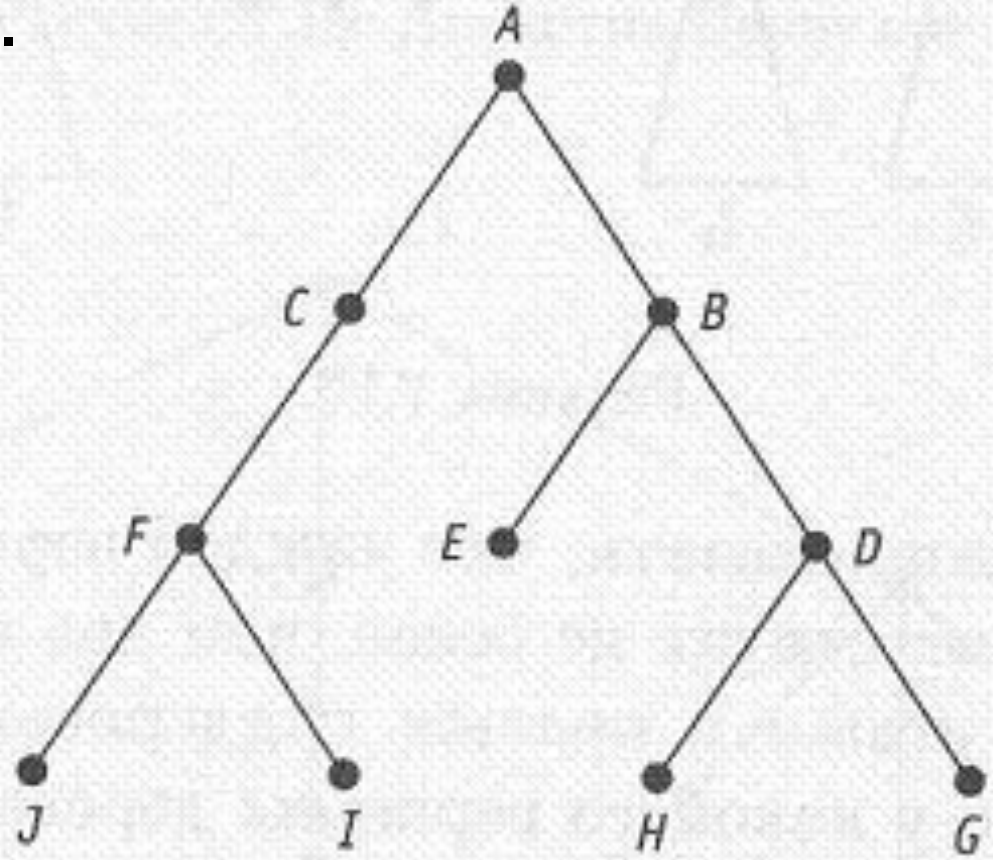

Рисунок 7.19. Двоичное дерево с корнем T'

Краткое содержание главы

Граф $G = (V, E)$ состоит из множества V, чьи элементы называют **вершинами** графа, и множества E его ребер, соединяющих некоторые пары вершин.

Вершины u и v графа называют **смежными,** если они соединены каким-то ребром e, про которое говорят, что оно инцидентно вершинам u и v.

Степенью вершины v считают число $\delta(v)$ ребер графа, инцидентных V.

Граф, в котором существует маршрут (называемый **эйлеровым**), начинающийся и заканчивающийся в одной и той же вершине и проходящий по каждому ребру графа ровно один раз, называется Эйлеровым графом. Связный граф с двумя или более вершинами является эйлеровым тогда и только тогда, когда каждая его вершина имеет четную степень.

Лемма об эстафете утверждает, что сумма степеней вершин произвольного графа $G = (V, E)$ равна удвоенному числу его ребер.

Простым принято называть граф $G = (V, E)$ с конечным множеством вершин V и конечным множеством ребер E, в котором нет петель и кратных ребер.

Логическая матрица отношения на множестве вершин простого графа G, которое задается его ребрами, называется **матрицей** смежности.

Подграфом графа $G = (V, E)$ называют граф $G' = (V', E')$, в котором $E' \subset E$ и $V' \subset V$.

Маршрутом длины k в графе называют такую последовательность различных вершин v_0, v_1, ..., v_k, в которой каждая пара соседних вершин v_{i-1} v_i соединена ребром.

Циклом в графе является замкнутый маршрут v_0, v_1, ..., v_0, у которого все вершины, кроме первой и последней, различны.

Граф, не содержащий циклов, называют **ацикличным.**

Связным является тот граф, в котором каждая пара вершин соединена маршрутом.

Количество компонент связности графа можно подсчитать с помощью **алгоритма связности.**

Гамильтоновым называют такой цикл в графе, который проходит через каждую вершину графа, причем только один раз. Граф, в котором существует гамильтонов цикл, называют **гамильтоновым.**

Задача коммивояжера состоит в поиске гамильтонова цикла минимального общего веса в нагруженном графе. **Алгоритм ближайшего соседа** позволяет найти субоптимальное решение задачи коммивояжера.

Связный ациклический граф $G = (V, E)$ является **деревом.** Следующие утверждения о связном графе $G = (V, E)$ с n вершинами и m ребрами эквивалентны:

- (а) G — дерево;
- (б) любую пару вершин G связывает единственный путь;
- (в) G связен и $m = n - 1$;
- (г) G связен, а удаление любого его ребра нарушает это свойство;
- (д) G ациклический, но соединяя любую пару вершин новым ребром, мы получаем цикл.

Остовным деревом графа *G* называют такой его подграф, который является деревом и содержит все вершины графа *G*. **Алгоритм поиска минимального остовного дерева** позволяет найти остовное дерево минимального общего веса в нагруженном графе и может быть использован для решения **задачи поиска кратчайшего соединения.**

Дерево с одной выделенной вершиной называют **деревом с корнем,** а выделенную вершину — его **корнем.** Вершины, стоящие непосредственно под вершиной *v* (и соединенные с ней ребрами), называются **сыновьями** вершины *v.*

Вершины, расположенные в самом низу дерева (они не имеют сыновей), называются **листьями.** Вершины, отличные от корня и листьев, называют **внутренними** вершинами графа. **Нулевое дерево** — это дерево, не имеющее ни одной вершины.

Каждая вершина дерева с корнем *T* является корнем какого-то другого дерева, называемого **поддеревом** *T*. В **двоичном дереве с корнем** каждая вершина имеет не более двух сыновей, а два поддерева вершины *v* называют **левым** и **правым** поддеревьями, ассоциированными с *v*. Двоичное дерево с корнем называют **полным**, если каждая его вершина, за исключением листьев, имеет ровно по два сына.

Глубиной вершины *v* дерева с корнем *T* принято считать длину единственного маршрута, соединяющего ее с корнем. **Глубиной графа** T называют максимальную глубину его вершин.

Сортировка и поиск

Двоичные деревья с корнем очень полезны при решении задач о выборе, в частности, о выборе такого сорта, при котором нужно классифицировать упорядоченные данные или вести в них поиск.

Упорядоченные данные, такие как множество чисел, упорядоченных по величине или множество строк литер, упорядоченных лексикографически (в алфавитном порядке), можно организовать в виде вершин двоичного дерева с корнем в соответствии с их порядком.

При этом мы стремимся к тому, чтобы данные, стоящие в левом поддереве данной вершины *v* были бы меньше данных, соответствующих этой вершине, а данные, расположенные в правом ее поддереве — больше. Дерево данных, удовлетворяющее указанному условию, называют *двоичным деревом поиска.*

Например, в дереве двоичного поиска, приведенном на рис. 7.26, слова фразы «У МОЕГО КОМПЬЮТЕРА ЕСТЬ ЧИП НА МАТЕРИНСКОЙ ПЛАТЕ» организованы именно таким образом.

Заметим, что каждое слово в левом поддереве любой вершины предшествует (относительно алфавитного порядка) слову, стоящему в этой вершине, а каждое слово ее правого поддерева следует за словом выбранной вершины.

Рисунок 7.26. Дерево двоичного поиска

Преимущество организации упорядоченных данных в виде двоичного дерева поиска заключается в возможности создания эффективного алгоритма поиска каких-то конкретных данных, включения новых данных в дерево и печати всей информации, содержащейся в дереве в виде упорядоченного списка.

Предположим, что в университете хранится список студентов (упорядоченный в алфавитном порядке), в котором кроме фамилий и имен имеются дополнительные важные сведения о студентах.

Допустим также, что возникла необходимость найти какую-то информацию в списке или добавить новые записи к списку. Мы сейчас познакомимся с алгоритмами, которые осуществляют поиск конкретной информации, добавляют новых студентов к списку и выводят на печать все записи в алфавитном порядке.

Записи о студентах организованы в двоичное дерево поиска (каждая запись соответствует одной вершине), и наши алгоритмы будут исследовать вершины этого дерева. Поскольку каждая вершина является также и корнем некоторого двоичного дерева поиска, алгоритмы будут последовательно проверять левые и правые поддеревья вершин.

Чтобы это осуществить, необходимо приписать каждой вершине некоторый *ключ* для идентификации и ссылок на ее левое и правое поддеревья (в структурах данных для этих целей используются так называемые дважды связанные списки).

Из всех ключей организуется линейно упорядоченное множество (в нашей ситуации оно упорядочено лексикографически).

Алгоритм *поиска* определяет, является ли данная запись *(ключ поиска)* вершиной в двоичном дереве поиска, сравнивая ключ поиска с ключом корня дерева, и, при необходимости, осуществляет аналогичные сравнения в левом или правом поддеревьях.

Поиск (дерево)

```
begin
    if дерево нулевое  then
        поиск := ложь;
    else
        if ключ поиска = ключ корня  then
            поиск := истина;
        else
```

```
if ключ поиска < ключ корня  then
        поиск := поиск (левое поддерево);
    else
        поиск := поиск (правое поддерево);
    end
```

Задача 1. Проследите за работой алгоритма над двоичным деревом поиска, изображенном на рис. 7.27. Известно, что ключ поиска — буква Л, а ключи вершин упорядочены лексикографически.

Решение. Поскольку $R > K$, то поиск продолжается в правом поддереве вершины *К*. Так как $R < T$, процесс поиска переключается на левое поддерево вершины Г. Наконец, ввиду неравенства $R \neq M$ и отсутствия поддеревьев у вершины *М*. алгоритм заканчивается и сообщает, что искомая вершина не была найдена.

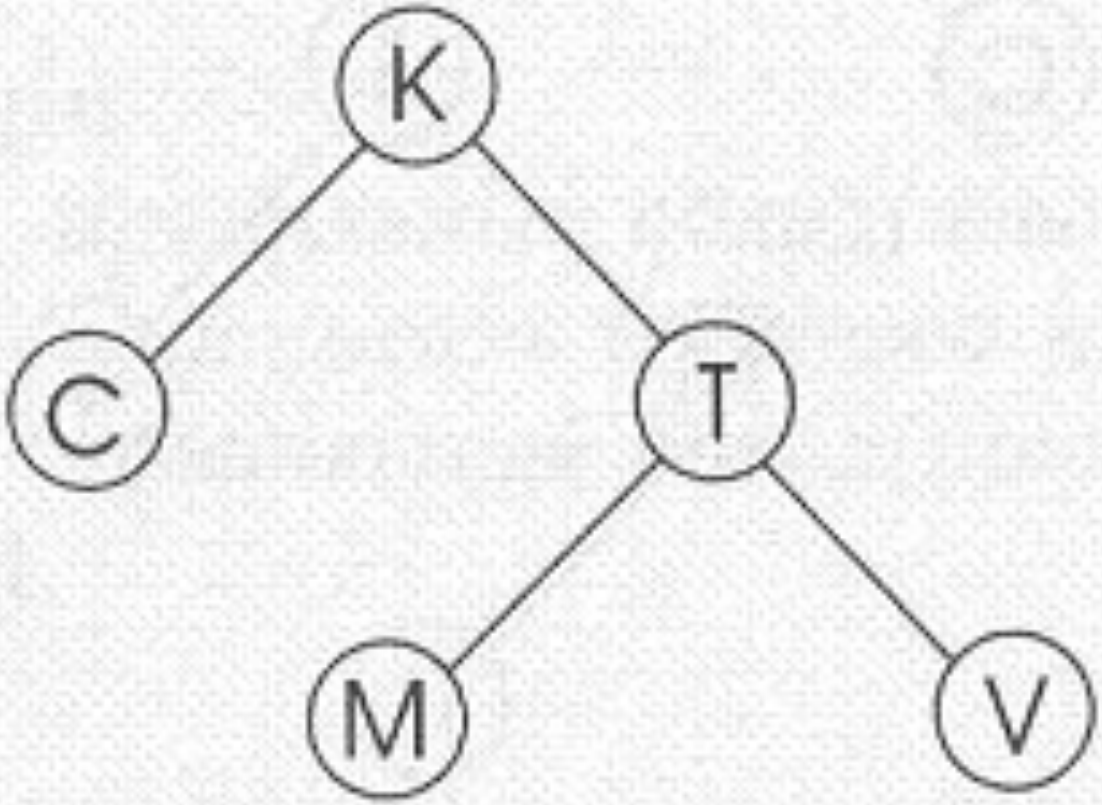

Рисунок 7.27.

Алгоритм вставки вставляет новые вершины *(ключи вставок)* в двоичное дерево поиска, создавая при этом новую вершину слева или справа от уже существующей. Это делается таким образом, чтобы все ключи вершин в получившемся дереве подчинялись установившемуся порядку.

Вставка (запись, дерево)

```
begin
    if дерево нулевое  then
        добавить новую вершину;
    else
        if ключ вставки = ключ корня then
            вывести на печать:
            «запись содержится в дереве»;
        else
```

if *ключ вставки* < *ключ корня* **then**

вставка := *вставка* (*запись, левое поддерево*);

else

вставка := *вставка* (*запись, правое поддерево*);

end

Задача 2. Проследите за работой алгоритма вставки на примере вершин *R, A* и *L* в дерево из задачи 1.

Решение. Поскольку $R > K$, мы применяем алгоритм вставки к правому поддереву вершины *K*. Далее мы видим, что $R < T$. Значит, алгоритм вставки переключается на левое поддерево вершины *T*. Так как $R > M$ и правое поддерево вершины *M* нулевое, то мы ставим вершину *R* справа от *M* и получаем дерево, изображенное на рис. 7.28. Теперь вставим *A* и *L*, построив дерево, показанное на рис. 7.29.

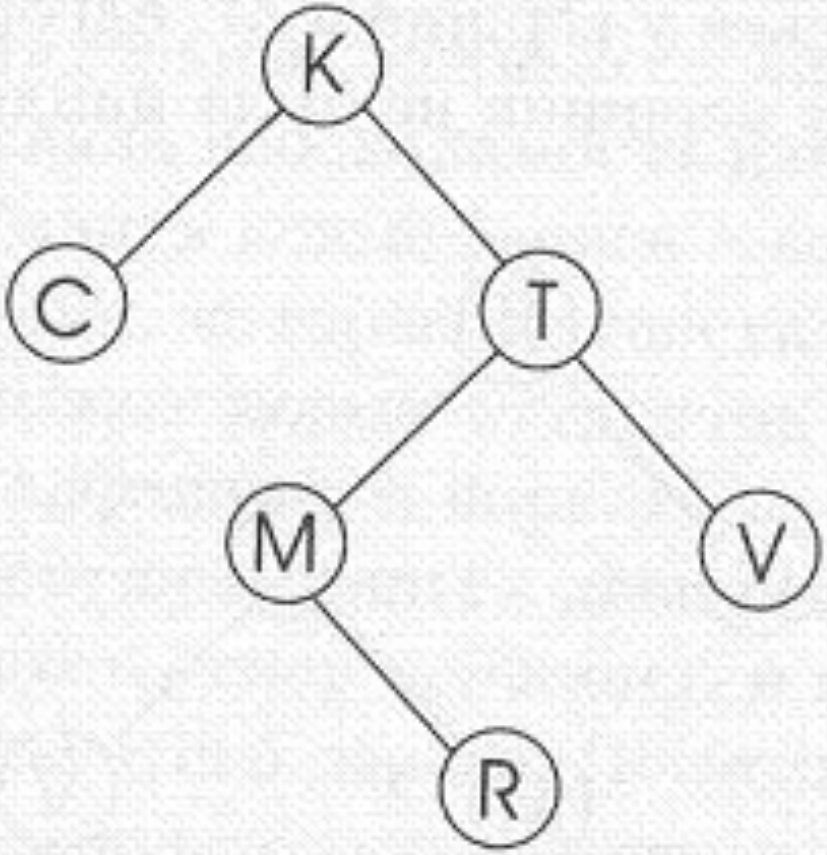

Рисунок 7.28.

Алгоритм вставки можно использовать для создания двоичного дерева поиска, начиная с нулевого дерева и последовательно добавляя новые данные в удобном для нас порядке.

Например, двоичное дерево поиска на рис. 7.26 является результатом применения алгоритма вставки к нулевому дереву в процессе добавления слов фразы «У МОЕГО КОМПЬЮТЕРА ЕСТЬ ЧИП НА МАТЕРИНСКОЙ ПЛАТЕ» в том порядке, в котором они в ней записаны.

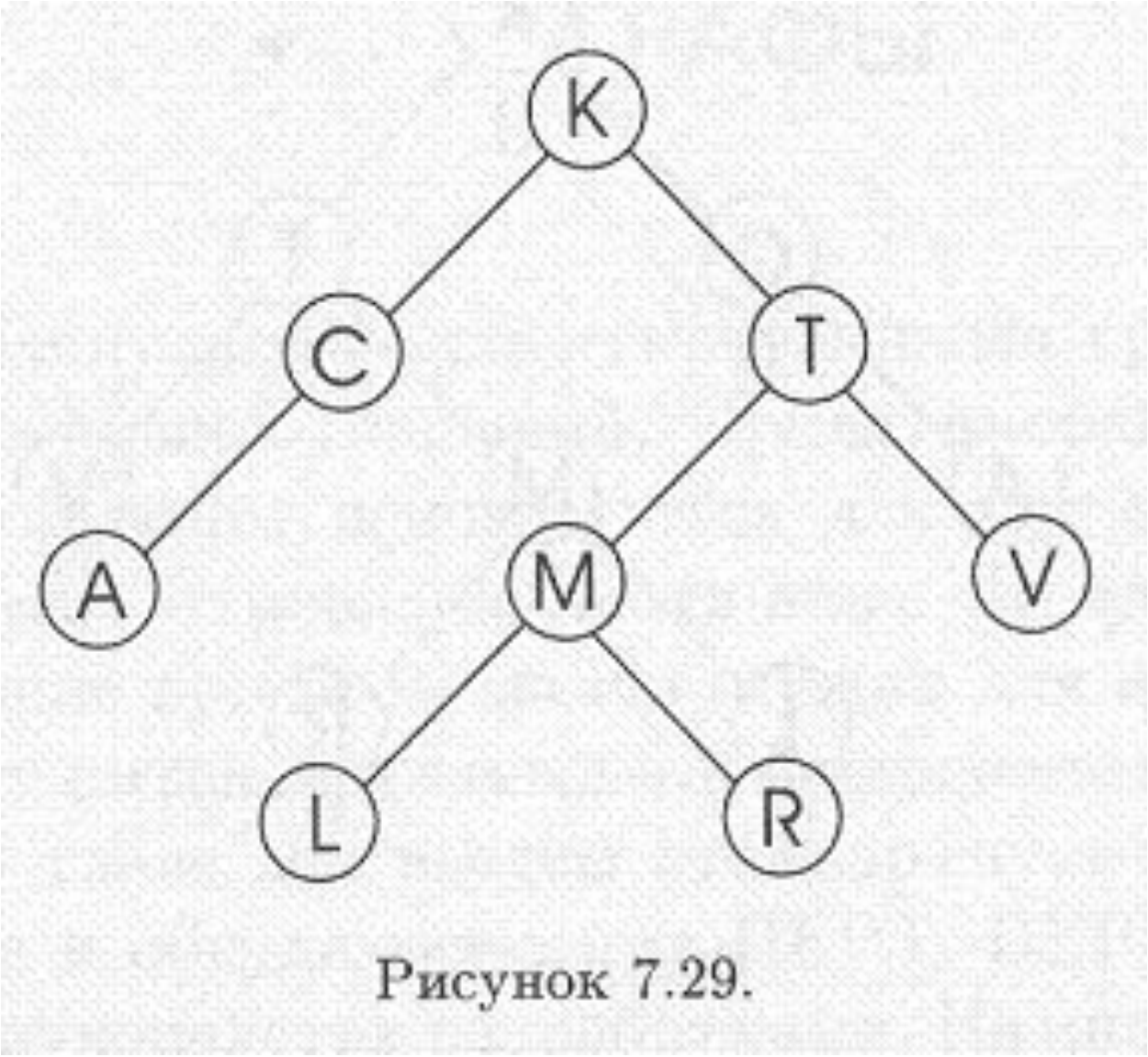

Рисунок 7.29.

Алгоритм правильного обхода выводит на печать всю информацию, содержащуюся в двоичном дереве поиска, в надлежащем порядке. При этом все вершины дерева осматриваются в определенном порядке. Алгоритм работает следующим образом.

Для каждой вершины, начиная с корня, печатается вся информация, содержащаяся в вершинах левого поддерева. Затем выводится информация, хранящаяся в этой вершине, и наконец, информация, соответствующая вершинам правого поддерева.

Правильный обход(дерево)
begin
if *дерево нулевое* **then**
ничего не делать;
else
begin
правильный обход (*левое поддерево*);
напечатать корневой ключ;
правильный обход (*правое поддерево*);
end
end

Задача 3. Примените алгоритм правильного обхода к дереву, полученному в задаче 2 после вставки *R*, *A* и *L*.

Решение. После работы алгоритма над указанным деревом получается список: *A*, *C*, *K*, *L*, *M*, *R*, *T*, *V*.

Он соответствует обходу дерева против часовой стрелки (рис. 7.30) и печати информации, содержащейся в вершинах, как только Вы прошли *под* вершиной.

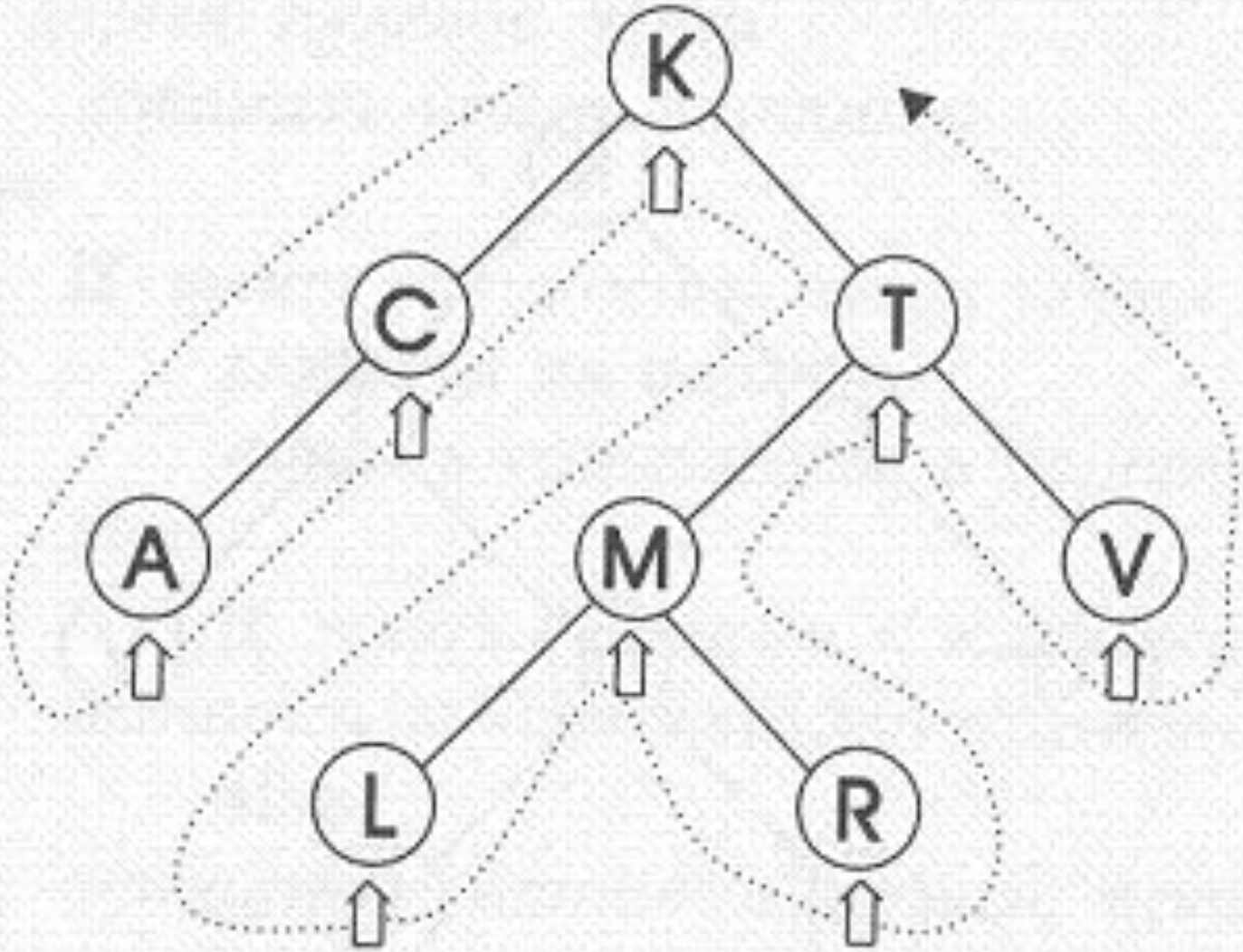

Рисунок 7.30.