

ГРАФЫ

(Лекции по дискретной математике)

М.И. Дехтярь

1 Основные понятия

Мы часто сталкиваемся с задачами, в условиях которых заданы некоторые объекты и между некоторыми их парами имеются определенные связи. Если объекты изобразить точками (вершинами), а связи — линиями (ребрами), соединяющими соответствующие пары точек, то получится рисунок, называемый *графом*. Историю теории графов принято исчислять с 1736 г., когда Эйлер исследовал “задачу о кенигсбергских мостах”: построить в графе циклический путь, проходящий по одному разу через каждое ребро. В середине 19-го века Гамильтон заинтересовался задачей построения циклического пути, проходящего по одному разу через каждую вершину графа¹. К тому же времени относится использование графов для анализа электрических цепей (Кирхгоф) и химических молекул (Кэли). Развитие современной теории графов относится к 30-м годам 20-го столетия. Они нашли многочисленные применения в электротехнике, электронике, биологии, экономике, программировании и в других областях. В этой главе мы рассмотрим основные понятия теории графов и несколько широко используемых в различных приложениях типовых задач таких, как представления графов, отношение достижимости и транзитивное замыкание графа, компоненты сильной связности ориентированного графа, деревья и их обходы, минимальные остовные деревья, поиск в глубину и задача о кратчайших путях. Основное внимание уделено алгоритмическим процедурам, решающим указанные задачи.² Некоторый дополнительный материал помещен в конце этой главы в разделе “Задачи”. В последующих двух главах будут приведены примеры применения графов для реализации булевых функций и для представления конечных автоматов.

Приведем основные определения.

Определение 1. **Ориентированный граф** — это пара (V, E) , где V — конечное множество вершин (узлов, точек) графа, а E — некоторое множество пар вершин, т.е. подмножество множества $V \times V$ или бинарное отношение на V . Элементы E называют **ребрами** (дугами, стрелками, связями). Для ребра $e = (u, v) \in E$ вершина u называется началом e , а вершина v — концом e , говорят, что ребро e ведет из u в v .

Неориентированный граф $G = (V, E)$ — это ориентированный граф, у которого для каждого ребра $(u, v) \in E$ имеется противоположное ребро $(v, u) \in E$, т.е. отношение E симметрично. Такая пара $(u, v), (v, u)$ называется неориентированным ребром. Для его задания можно использовать обозначение для множества концов: $\{u, v\}$, но чаще используется указание одной из пар в круглых скобках. Если $e = (u, v) \in E$, то вершины u и v называются смежными в G , а ребро e и эти вершины называются инцидентными. Степенью вершины в неориентированном графе называется число смежных с ней вершин. Вершина степени 0 называется изолированной.

В ориентированном графе полустепень исхода вершины — это число исходящих из нее ребер, а полустепень захода — это число входящих в данную вершину ребер.

Заметим, что в ориентированном графе может быть ребро вида (u, u) , называемое *петлей*, а в неориентированном петель не бывает.

Пример 1. На рис.1 приведены примеры ориентированного графа $G_1 = (V_1, E_1)$ и неориентированного графа $G_2 = (V_2, E_2)$. Здесь $V_1 = \{a, b, c, d\}$, $E_1 = \{(a, b), (a, c), (b, b), (b, d), (d, a)\}$,

¹Интересно, что несмотря на внешнюю похожесть задача Эйлера имеет простое эффективное решение (см. задачу 12), а задача Гамильтона в общем случае эффективно не решается.

²В качестве пособия по теории графов, содержащего часть материала этой главы и некоторые дополнительные сведения, можно использовать брошюру М.А. Тайцлина “Графы”, изданную в 2000г. в ТвГУ.

$V_2 = \{a, b, c, d\}$, $E_2 = \{(a, b), (a, c), (a, d), (b, d)\}$. В графе G_1 ребро (b, b) является петлей, полустепень исхода вершины a равна 2, а полустепень захода для нее равна 1. В графе G_2 степень вершины a равна 3, вершин b и d — 2, вершины c — 1, а вершины e — 0, т.е. вершина e является изолированной,

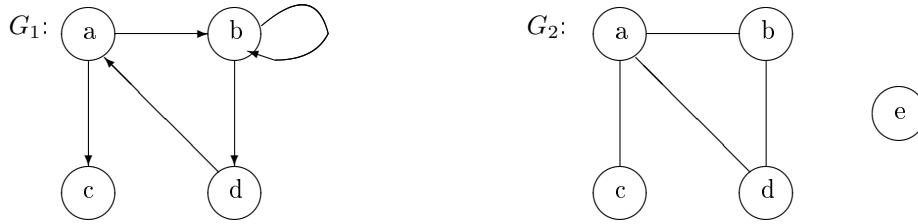


Рис. 1: Ориентированный граф G_1 и неориентированный граф G_2

Во многих приложениях с вершинами и ребрами графов связывается некоторая дополнительная информация. Обычно она представляется с помощью функций разметки вершин и ребер.

Определение 2. Размеченный граф — это ориентированный или неориентированный граф $G = (V, E)$, снабженный одной или двумя функциями разметки вида: $l : V \rightarrow M$ и $s : E \rightarrow L$, где M и L — множества меток вершин и ребер, соответственно.

Упорядоченный граф — это размеченный граф $G = (V, E)$, в котором ребра, выходящие из каждой вершины $v \in V$, упорядочены, т.е. помечены номерами $1, \dots, k_v$, где k_v — полустепень исхода v , т.е. $k_v = |\{w \mid (v, w) \in E\}|$.

В качестве множества меток ребер L часто выступают числа, задающие “веса”, “длины”, “стоимости” ребер. Графы с такой разметкой часто называют *взвешенными*.

Во многих случаях естественно не различать графы, отличающиеся лишь именами (порядком) вершин.

Определение 3. Изоморфизм графов. Два графа $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ называются *изоморфными*, если между их вершинами существует взаимно однозначное соответствие $\varphi : V_1 \rightarrow V_2$ такое, что для любой пары вершин u, v из V_1 ребро $(u, v) \in E_1 \Leftrightarrow$ ребро $(\varphi(u), \varphi(v)) \in E_2$.

Для изоморфизма размеченных графов требуется также совпадение меток соответствующих вершин: $l_1(v) = l_2(\varphi(v))$ и/или ребер: $c_1((u, v)) = c_2((\varphi(u), \varphi(v)))$.

Многие приложения графов связаны с изучением путей между их вершинами.

Определение 4. Путь в ориентированном или неориентированном графе — это последовательность ребер вида $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$. Этот путь ведет из начальной вершины v_1 в конечную вершину v_n и имеет длину $n-1$. В этом случае будем говорить, что v_n **достижима** из v_1 . Будем считать, что каждая вершина достижима сама из себя путем длины 0. Путь можно также определять, как соответствующую последовательность вершин: $(v_1, v_2, v_3, \dots, v_{n-1}, v_n)$, где $(v_i, v_{i+1}) \in E$ при $i = 1, 2, \dots, n-1$.

Путь называется **простым**, если все ребра и все вершины на нем, кроме, быть может, первой и последней, различны.

Циклом в ориентированном графе называется путь, в котором начальная вершина совпадает с конечной и который содержит хотя бы одно ребро. Цикл $(v_1, v_2, \dots, v_{n-1}, v_n = v_1)$ называется **простым**, если в нем нет одинаковых вершин, кроме первой и последней, т.е. если все вершины v_2, \dots, v_{n-1} различны.

В неориентированном графе путь $(v_1, v_2, \dots, v_{n-1}, v_n = v_1)$ называется **циклом**, если $n \geq 4$ и все ребра (v_i, v_{i+1}) различны.

Из последнего определения следует, что длина цикла в неориентированном графе не меньше 3. Следующее утверждение непосредственно следует из определений.

Лемма 1. Если в графе G (ориентированном или неориентированном) имеется путь из u в v , то в нем имеется и простой путь из u в v .

Неориентированный граф называется *связным*, если любая пара вершин в нем соединена путем. При выполнении такого же условия ориентированный граф называется *сильно связным*.

2 Представления графов

Из определения графа следует, что каждый граф $G = (V, E)$ можно задать, непосредственно перечислив его множество вершин V и множество ребер E . Однако такое представление неудобно для решения многих задач о графах. Например, чтобы проверить наличие ребра между двумя вершинами, придется, вообще говоря, просмотреть все множество E . Хорошее представление, по крайней мере, должно позволить легко переходить от вершины к ее соседу и перечислять всех ее соседей.

В этом параграфе мы рассмотрим три разных способа представления графов, которые более эффективны при решении типичных для теории графов задач.

2.1 Матрица (таблица) смежности

Определение 5. Матрицей смежности ориентированного (или неориентированного) графа $G = (V, E)$ с n вершинами $V = \{v_1, \dots, v_n\}$ называется булева матрица A_G размера $n \times n$ с элементами

$$a_{ij} = \begin{cases} 1, & \text{если } (v_i, v_j) \in E \\ 0 & \text{в противном случае} \end{cases}$$

Это представление позволяет легко проверять наличие ребер между заданными парами вершин. Для поиска всех соседей, в которые ведут ребра из вершины v_i , необходимо просмотреть соответствующую ей i -ю строку матрицы A_G , а чтобы найти вершины, из которых ребра идут в v_i , необходимо просмотреть ее i -ый столбец. Требуемая для A_G память — по порядку n^2 битов — не может быть уменьшена для графов, у которых “много” ребер. Но для разреженных графов с числом ребер существенно меньшим по порядку n^2 в матрице смежности много “ненужных” нулей. Для таких графов более эффективными могут оказаться другие представления.

2.2 Матрица (таблица) инцидентности

Определение 6. Матрицей инцидентности ориентированного (или неориентированного) графа $G = (V, E)$ с n вершинами $V = \{v_1, \dots, v_n\}$ и t ребрами $E = \{e_1, \dots, e_t\}$ называется матрица B_G размера $n \times t$ с элементами

$$b_{ij} = \begin{cases} 1, & \text{если для некоторого } k \text{ ребро } e_j = (v_i, v_k), \\ -1, & \text{если для некоторого } k \text{ ребро } e_j = (v_k, v_i), \\ 2 & \text{если ребро } e_j = (v_i, v_i), \\ 0 & \text{в противном случае} \end{cases}$$

Таким образом, в матрице инцидентности B_G любому ребру $e_j = (v_i, v_k)$ соответствует j -ый столбец, в котором в i -ой строке стоит 1, а в k -ой — -1. Ребра-петли выделяются числом 2. Для проверки наличия ребра между двумя вершинами v_i и v_k требуется просмотреть i -ю и k -ую строки B_G , поиск всех соседей вершины требует просмотра соответствующей строки. Если $t \gg n$, то это требует существенно больше времени, чем при использовании матрицы смежности. Поэтому при практическом решении задач на графах матрица инцидентности почти не используется.

2.3 Списки смежности

Определение 7. Пусть $G = (V, E)$ — ориентированный граф, v — вершина из V . Список смежности L_v для вершины v включает все смежные с ней вершины, т.е. $L_v = w_1, \dots, w_k$, где $\{w_1, \dots, w_k\} = \{w \mid (v, w) \in E\}$.

Представление графа $G = (V, E)$ с n вершинами $V = \{v_1, \dots, v_n\}$ с помощью списков смежности состоит из списков смежности всех вершин: $L_{v_1}, L_{v_2}, \dots, L_{v_n}$.

Размер этого представления сравним с суммой числа вершин и ребер графа. Оно позволяет легко переходить по ребрам от вершины к ее соседям. В программах списки смежности представляются списковыми структурами, которые легко реализуются во всех языках программирования.

Пример 2. Рассмотрим следующий граф $G = (V, E)$:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\},$$

$$E = \{e_1 = (v_1, v_3), e_2 = (v_3, v_2), e_3 = (v_4, v_1), e_4 = (v_4, v_5), e_5 = (v_5, v_3), e_6 = (v_6, v_5), e_7 = (v_6, v_6), e_8 = (v_2, v_1)\}.$$
 Он показан на рис.2.

Построим для него определенные выше представления.

1) Матрица смежности.

$$A_G = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

2) Матрица инцидентности.

$$B_G = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \end{pmatrix}.$$

3) Списки смежности.

$$\begin{aligned} L_{v_1} &: v_3; & L_{v_4} &: v_1, v_5; \\ L_{v_2} &: v_1; & L_{v_5} &: v_3, v_6; \\ L_{v_3} &: v_2; & L_{v_6} &: v_6. \end{aligned}$$

3 Достижимость и связность

Один из первых вопросов, возникающих при изучении графов, это вопрос о существовании путей между заданными или всеми парами вершин. Ответом на этот вопрос является введенное выше отношение достижимости на вершинах графа $G = (V, E)$: вершина w достижима из вершины v , если $v = w$ или в G есть путь из v в w . Иначе говоря, отношение достижимости является рефлексивным и транзитивным замыканием отношения E . Для неориентированных графов это отношение является также симметричным и, следовательно, является отношением эквивалентности на множестве вершин V . В неориентированном графе классы эквивалентности по отношению достижимости называются *связными компонентами*. Для ориентированных графов достижимость, вообще говоря, не является симметричным отношением. Симметричной является взаимная достижимость.

Определение 8. Вершины v и w ориентированного графа $G = (V, E)$ называются *взаимно достижимыми*, если в G есть путь из v в w и путь из w в v .

Ясно, что отношение взаимной достижимости является рефлексивным, симметричным и транзитивным и, следовательно, эквивалентностью на множестве вершин графа. Классы эквивалентности по отношению взаимной достижимости называются *двусвязными компонентами* графа.

Рассмотрим вначале вопрос о построении отношения достижимости.

3.1 Граф достижимости (транзитивного замыкания)

Определим для каждого графа его граф достижимости, ребра которого соответствуют путям исходного графа.

Определение 9. Пусть $G = (V, E)$ — ориентированный граф. **Граф достижимости** $G^* = (V, E^*)$ для G имеет то же множество вершин V и следующее множество ребер $E^* = \{(u, v) \mid \text{в графе } G \text{ вершина } v \text{ достижима из вершины } u\}$.

Пример 3. Рассмотрим граф G из примера 2. Он показан на следующем рисунке:

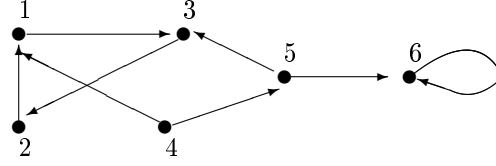


Рис. 2: Граф G

Тогда можно проверить, что граф достижимости G^* для G выглядит так (новые ребра-петли при каждой из вершин 1-5 не показаны):

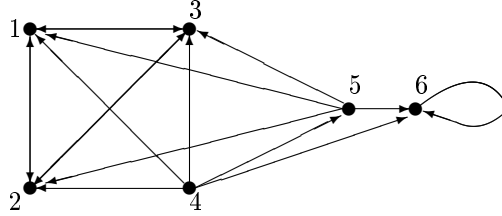


Рис. 3: Граф G^*

Каким образом по графу G можно построить граф G^* ? Один способ заключается в том, чтобы для каждой вершины графа G определить множество достижимых из нее вершин, последовательно добавляя в него вершины, достижимые из нее путями длины 0, 1, 2 и т.д.

Мы рассмотрим другой способ, основанный на использовании матрицы смежности A_G графа G и булевых операций. Пусть множество вершин $V = \{v_1, \dots, v_n\}$. Тогда матрица A_G — это булева матрица размера $n \times n$.

Ниже для сохранения сходства с обычными операциями над матрицами мы будем использовать “арифметические” обозначения для булевых операций: через $+$ будем обозначать дизъюнкцию \vee , а через \cdot — конъюнкцию \wedge .

Обозначим через E_n единичную матрицу размера $n \times n$. Положим $\tilde{A} = A_G + E_n$. Пусть $\tilde{A}^0 = E_n, \tilde{A}^1 = \tilde{A}, \dots, \tilde{A}^{k+1} = \tilde{A}^k \cdot \tilde{A}$. Наша процедура построения G^* основана на следующем утверждении.

Лемма 2. Пусть $\tilde{A}^k = (a_{ij}^{(k)})$. Тогда

$$a_{ij}^{(k)} = \begin{cases} 1, & \text{в графе } G \text{ из } v_i \text{ в } v_j \text{ имеется путь длины } \leq k \\ 0 & \text{в противном случае} \end{cases}$$

Доказательство проведем индукцией по k .

Базис. При $k = 0$ и $k = 1$ утверждение справедливо по определению \tilde{A}^0 и \tilde{A}^1 .

Индукционный шаг. Пусть лемма справедлива для k . Покажем, что она остается справедливой и для $k + 1$. По определению \tilde{A}^{k+1} имеем:

$$a_{ij}^{(k+1)} = a_{i1}^{(k)} a_{1j}^{(1)} + \dots + a_{ir}^{(k)} a_{rj}^{(1)} + \dots + a_{in}^{(k)} a_{nj}^{(1)}.$$

Предположим, что в графе G из v_i в v_j имеется путь длины $\leq k + 1$. Рассмотрим кратчайший из таких путей. Если его длина $\leq k$, то по предположению индукции $a_{ij}^{(k)} = 1$. Кроме того, $a_{jj}^{(1)} = 1$. Поэтому $a_{ij}^{(k)} a_{jj}^{(1)} = 1$ и $a_{ij}^{(k+1)} = 1$. Если длина кратчайшего пути из v_i в v_j равна $k + 1$, то пусть v_r — его предпоследняя вершина. Тогда из v_i в v_r имеется путь длины k и по предположению индукции $a_{ir}^{(k)} = 1$. Так как $(v_r, v_j) \in E$, то $a_{rj}^{(1)} = 1$. Поэтому $a_{ir}^{(k)} a_{rj}^{(1)} = 1$ и $a_{ij}^{(k+1)} = 1$.

Обратно, если $a_{ij}^{(k+1)} = 1$, то хотя бы для одного r слагаемое $a_{ir}^{(k)} a_{rj}^{(1)}$ в сумме равно 1. Если это $r = j$, то $a_{ij}^{(k)} = 1$ и по индуктивному предположению в G имеется путь из v_i в v_j длины $\leq k$. Если же $r \neq j$, то $a_{ir}^{(k)} = 1$ и $a_{rj}^{(1)} = 1$. Это означает, что в G имеется путь из v_i в v_r длины $\leq k$ и ребро $(v_r, v_j) \in E$. Объединив их, получаем путь из v_i в v_j длины $\leq k + 1$. \square

Из лемм 1 и 2 непосредственно получаем

Следствие 1. Пусть $G = (V, E)$ — ориентированный граф с n вершинами, а G^* — его граф достижимости. Тогда $A_{G^*} = \tilde{A}^{n-1}$.

Доказательство. Из леммы 5.1 следует, что, если в G имеется путь из u в $v \neq u$, то в нем имеется и простой путь из u в v длины $\leq n - 1$. А по лемме 5.2 все такие пути представлены в матрице \tilde{A}^{n-1} . \square

Таким образом процедура построения матрицы смежности A_{G^*} графа достижимости для G сводится к возведению матрицы \tilde{A} в степень $n - 1$. Сделаем несколько замечаний, позволяющих упростить эту процедуру.

1) Для возведения матрицы \tilde{A} в произвольную степень n достаточно выполнить $\lceil \log n \rceil$ возведений в квадрат:

$$\tilde{A} \Rightarrow \tilde{A}^2 \Rightarrow \tilde{A}^{2^2} \Rightarrow \dots \Rightarrow \tilde{A}^{2^k},$$

где k — это наименьшее число такое, что $2^k \geq n$.

2) Так как на диагонали в матрице \tilde{A} стоят единицы, то для любых $i < j$ все единицы матрицы \tilde{A}^i сохраняются в матрице \tilde{A}^j , в частности, и в матрице $(\tilde{A}^i)^2$.

3) Если при вычислении элемента $a_{ij}^{(2)}$ матрицы $(\tilde{A}^k)^2$ по стандартной формуле

$$a_{ij}^{(2)} = a_{i1}a_{1j} + a_{i2}a_{2j} + \dots + a_{ir}a_{rj} + \dots + a_{in}a_{nj}$$

обнаруживается такое r , что $a_{ir} = 1$ и $a_{rj} = 1$, то и вся сумма $a_{ij}^{(2)} = 1$. Поэтому остальные слагаемые можно не рассматривать.

Пример 4. Рассмотрим в качестве примера вычисление матрицы графа достижимости A_{G^*} для графа G , представленного на рис. 2. В этом случае

$$\tilde{A} = A_G + E_6 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Так как у G имеется 6 вершин, то $A_{G^*} = \tilde{A}^5$. Вычислим эту матрицу:

$$\tilde{A}^2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \tilde{A}^4 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \tilde{A}^5 = \tilde{A}^4 \times \tilde{A} = \tilde{A}^4.$$

(последнее равенство нетрудно проверить). Таким образом,

$$A_{G^*} = \tilde{A}^4 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Как видим, эта матрица действительно задает граф G^* , представленный на рис.3.

3.2 Взаимная достижимость, компоненты сильной связности и базы графа

По аналогии с графом достижимости определим граф сильной достижимости.

Определение 10. Пусть $G = (V, E)$ — ориентированный граф. **Граф сильной достижимости** $G_*^* = (V, E_*^*)$ для G имеет то же множество вершин V и следующее множество ребер $E_*^* = \{(u, v) \mid \text{в графе } G \text{ вершины } u \text{ и } v \text{ взаимно достижимы}\}$.

По матрице графа достижимости A_G легко построить матрицу $A_{G_*^*}$ графа сильной достижимости. Действительно из определений достижимости и сильной достижимости непосредственно следует, то для всех пар $(i, j), 1 \leq i, j \leq n$, значение элемента $A_{G_*^*}(i, j)$ равно 1 тогда и только тогда, когда оба элемента $A_G(i, j)$ и $A_G(j, i)$ равны 1, т.е.

$$A_{G_*^*}(i, j) = A_G(i, j) \wedge A_G(j, i).$$

По матрице $A_{G_*^*}$ можно выделить двусвязные компоненты графа G следующим образом.

- (1) Поместим в компоненту K_1 вершину v_1 и все такие вершины v_j , что $A_{G_*^*}(1, j) = 1$.
- (2) Пусть уже построены компоненты K_1, \dots, K_i и v_k — это вершина с минимальным номером, еще не попавшая в компоненты. Тогда поместим в компоненту K_{i+1} вершину v_k и все такие вершины v_j , что $A_{G_*^*}(k, j) = 1$.

Повторяем шаг (2) до тех пор, пока все вершины не будут распределены по компонентам.

В нашем примере для графа G на рис.2 по матрице A_G получаем следующую матрицу графа сильной достижимости

$$A_{G_*^*} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Используя описанную выше процедуру, находим, что вершины графа G разбиваются на 4 компоненты сильной связности: $K_1 = \{v_1, v_2, v_3\}$, $K_2 = \{v_4\}$, $K_3 = \{v_5\}$, $K_4 = \{v_6\}$.

На множестве компонент сильной связности также определим отношение достижимости.

Определение 11. Пусть K и K' — компоненты сильной связности графа G . Компонента K достижима из компоненты K' , если $K = K'$ или существуют такие две вершины $u \in K$ и $v \in K'$, что u достижима из v . K строго достижима из K' , если $K \neq K'$ и K достижима из K' .

Компонента K называется минимальной, если она не является строго достижимой ни из какой компоненты.

Так как все вершины в одной компоненте взаимно достижимы, то нетрудно понять, что отношения достижимости и строгой достижимости на компонентах не зависят от выбора вершин $u \in K$ и $v \in K'$.

Из определения легко выводится следующая характеристика строгой достижимости.

Лемма 3. Отношение строгой достижимости является отношением частичного порядка, т.е. оно антирефлексивно, антисимметрично и транзитивно.

Это отношение можно представлять в виде ориентированного графа, вершинами которого являются компоненты, а ребро (K', K) означает, что K строго достижима из K' . На рис. 4 показан этот граф компонент для рассматриваемого выше графа G .

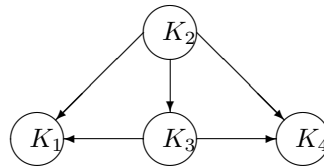


Рис. 4: Граф отношения достижимости на компонентах G

В данном случае имеется одна минимальная компонента K_2 .

Во многих приложениях ориентированный граф представляет собой сеть распространения некоторого ресурса: продукта, товара, информации и т.п. В таких случаях естественно возникает задача поиска минимального числа таких точек (вершин), из которых этот ресурс может быть доставлен в любую точку сети.

Определение 12. Пусть $G = (V, E)$ — ориентированный граф. Подмножество вершин $W \subseteq V$ называется порождающим, если из вершин W можно достичь любую вершину графа. Подмножество вершин $W \subseteq V$ называется базой графа, если оно является порождающим, но никакое его собственное подмножество порождающим не является.

Следующая теорема позволяет эффективно находить все базы графа.

Теорема 1. Пусть $G = (V, E)$ — ориентированный граф. Подмножество вершин $W \subseteq V$ является базой G тогда и только тогда, когда содержит по одной вершине из каждой минимальной двусвязной компоненты G и не содержит никаких других вершин.

Доказательство. Заметим вначале, что каждая вершина графа достижима из вершины, принадлежащей некоторой минимальной компоненте. Поэтому множество вершин W , содержащих по одной вершине из каждой минимальной компоненты, является порождающим а при удалении из него любой вершины перестает быть таковым, так как вершины из соответствующей минимальной компоненты становятся недостижимыми. Поэтому W является базой.

Обратно, если W является базой, то оно обязано включать хотя бы по одной вершине из каждой минимальной компоненты, иначе вершины такой минимальной компоненты окажутся недоступны. Никаких других вершин W содержать не может, так как каждая из них достижима из уже включенных вершин. \square

Из этой теоремы вытекает следующая процедура построения одной или перечисления всех баз графа G .

- 1) Найти все двусвязные компоненты G .
- 2) Определить порядок на них и выделить минимальные относительно этого порядка компоненты.
- 3) Порождать одну или все базы графа, выбирая по одной вершине из каждой минимальной компоненты.

Пример 5. Определим все базы ориентированного графа G , показанного на рис. 5.

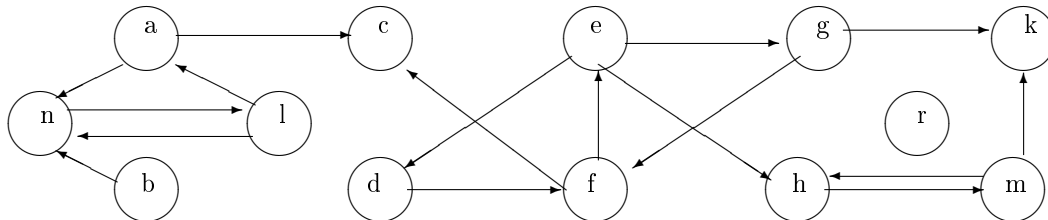


Рис. 5: Граф G

На первом этапе находим двусвязные компоненты G :

$K_1 = \{a, n, l\}$, $K_2 = \{b\}$, $K_3 = \{c\}$, $K_4 = \{d, e, f, g\}$, $K_5 = \{h, m\}$, $K_6 = \{k\}$, $K_7 = \{r\}$.

На втором этапе строим граф строгой достижимости на этих компонентах.

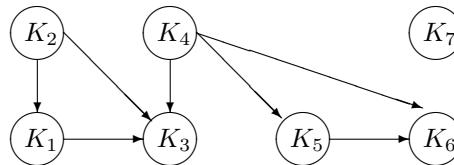


Рис. 6: Граф отношения достижимости на компонентах G

Определяем минимальные компоненты: $K_2 = \{b\}$, $K_4 = \{d, e, f, g\}$ и $K_7 = \{r\}$.

Наконец перечисляем все четыре базы G : $B_1 = \{b, d, r\}$, $B_2 = \{b, e, r\}$, $B_3 = \{b, f, r\}$ и $B_4 = \{b, g, r\}$.

3.3 Задачи

Задача 1. Докажите, что сумма степеней всех вершин произвольного неориентированного графа четна.

У этой задачи имеется популярная интерпретация: доказать, что общее число рукопожатий, которыми обменялись люди, пришедшие на вечеринку, всегда четно.

Задача 2. Определите, что представляет из себя граф достижимости для

а) графа с n вершинами и пустым множеством ребер;

б) графа с n вершинами: $V = \{v_1, \dots, v_n\}$, ребра которого образуют цикл:

$E = \{(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n), (v_n, v_1)\}$.

Задача 3. Вычислите матрицу графа достижимости A_{G^*} для графа

$G = (\{a, b, c, d, e, f, g\}, \{(a, b), (b, a), (a, c), (b, d), (e, d), (d, f), (f, c), (c, f), (g, e)\})$

и постройте соответствующий ей граф достижимости.

Найдите все базы графа G .

Задача 4. Построить для заданного на рис. 7 ориентированного графа $G_1 = (V, E)$ его матрицу смежности A_{G_1} , матрицу инцидентности B_{G_1} и списки смежности. Вычислить матрицу достижимости $A_{G_1^*}$ и построить соответствующий граф достижимости G_1^* .

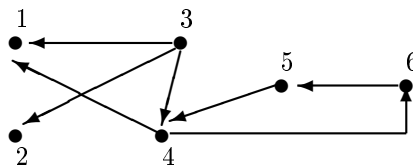


Рис. 7: Граф G_1

Задача 5. Перечислите все неизоморфные неориентированные графы, у которых не более четырех вершин.

4 Деревья

4.1 Неориентированные и ориентированные деревья

Деревья являются одним из интереснейших классов графов, используемых для представления различного рода иерархических структур.

Определение 13. Неориентированный граф называется деревом, если он связный и в нем нет циклов.

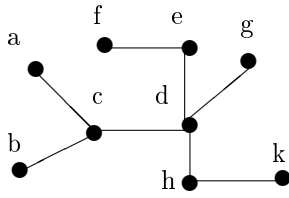
Определение 14. Ориентированный граф $G = (V, E)$ называется (ориентированным) деревом, если

- 1) в нем есть одна вершина $r \in V$, в которую не входят ребра; она называется корнем дерева;
- 2) в каждую из остальных вершин входит ровно по одному ребру;
- 3) все вершины достижимы из корня.

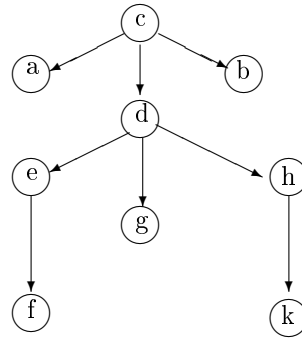
На рисунке 8 показаны примеры неориентированного дерева G_1 и ориентированного дерева G_2 . Обратите внимание на то, что дерево G_2 получено из G_1 с помощью выбора вершины c в качестве корня и ориентации всех ребер в направлении «от корня».

Это не случайно. Докажите самостоятельно следующее утверждение о связи между неориентированными и ориентированными деревьями.

Лемма 4. Если в любом неориентированном дереве $G = (V, E)$ выбрать произвольную вершину $v \in V$ в качестве корня и сорентировать все ребра в направлении «от корня», т.е. сделать v началом всех инцидентных ей ребер, вершины, смежные с v — началами всех инцидентных им еще не сорентированных ребер и т.д., то полученный в результате ориентированный граф G' будет ориентированным деревом.



Неориентированное дерево G_1



Ориентированное дерево G_2

Рис. 8: Неориентированное и ориентированное деревья

Неориентированные и ориентированные деревья имеют много эквивалентных характеристик.

Теорема 2. Пусть $G = (V, E)$ — неориентированный граф. Тогда следующие условия эквивалентны.

- (1) G является деревом.
- (2) Для любых двух вершин в G имеется единственный соединяющий их путь.
- (3) G связен, но при удалении из E любого ребра перестает быть связным.
- (4) G связен и $|E| = |V| - 1$.
- (5) G ациклический и $|E| = |V| - 1$.
- (6) G ациклический, но добавление любого ребра к E порождает цикл.

Доказательство. (1) \Rightarrow (2): Если бы в G некоторые две вершины соединялись двумя путями, то, очевидно, в G имелся бы цикл. Но это противоречит определению дерева в (1).

(2) \Rightarrow (3): Если G связен, но при удалении некоторого ребра $(u, v) \in E$ не теряет связности, то между u и v имеется путь, не содержащий это ребро. Но тогда в G имеются не менее двух путей, соединяющих u и v , что противоречит условию (2).

(3) \Rightarrow (4): Предоставляется читателю (см. задачу 8).

(4) \Rightarrow (5): Если G содержит цикл и является связным, то при удалении любого ребра из цикла связность не должна нарушиться, но ребер останется $|E| = V - 2$, а по задаче 8(а) в связном графе должно быть не менее $V - 1$ ребер. Полученное противоречие показывает, что циклов в G нет и выполнено условие (5).

(5) \Rightarrow (6): Предположим, что добавление ребра (u, v) к E не привело к появлению цикла. Тогда в G вершины u и v находятся в разных компонентах связности. Так как $|E| = V - 1$, то в одной из этих компонент, пусть это (V_1, E_1) , число ребер и число вершин совпадают: $|E_1| = |V_1|$. Но тогда в ней имеется цикл (см. задачу 8 (б)), что противоречит ациклическости G .

(6) \Rightarrow (1): Если бы G не был связным, то нашлись бы две вершины u и v из разных компонент связности. Тогда добавление ребра (u, v) к E не привело бы к появлению цикла, что противоречит (6). Следовательно, G связен и является деревом. \square

Для ориентированных деревьев часто удобно использовать следующее индуктивное определение.

Определение 15. Определим по индукции класс ориентированных графов \mathcal{D} , называемых деревьями. Одновременно для каждого из них определим выделенную вершину — корень.

1) Граф $T_0 = (V, E)$, с единственной вершиной $V = \{v\}$ и пустым множеством ребер $E = \emptyset$ является деревом (входит в \mathcal{D}). Вершина v называется корнем этого дерева.

2) Пусть графы $T_1 = (V_1, E_1), \dots, T_k = (V_k, E_k)$ с корнями $r_1 \in V_1, \dots, r_k \in V_k$ принадлежат \mathcal{D} , а r_0 — новая вершина, т.е. $r_0 \notin \bigcup_{i=1}^k V_i$. Тогда классу \mathcal{D} принадлежит также следующий граф $T = (V, E)$, где $V = \{r_0\} \cup \bigcup_{i=1}^k V_i$, $E = \{(r_0, r_i) \mid i = 1, \dots, k\} \cup \bigcup_{i=1}^k E_i$. Корнем этого дерева является вершина r_0 .

3) Других графов в классе \mathcal{D} нет.

Рисунок 9 иллюстрирует это определение.

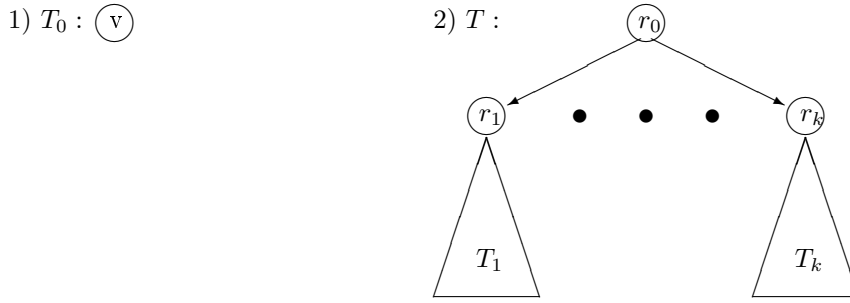


Рис. 9: Индуктивное определение ориентированных деревьев

Теорема 3. *Определения ориентированных деревьев 14 и 15 эквивалентны.*

Доказательство. \Rightarrow Пусть граф $G = (V, E)$ удовлетворяет условиям определения 14. Покажем индукцией по числу вершин $|V|$, что $G \in \mathcal{D}$.

Если $|V| = 1$, то единственная вершина $v \in V$ является по свойству (1) корнем дерева, т.е. в этом графе ребер нет: $E = \emptyset$. Тогда $G = T_0 \in \mathcal{D}$.

Предположим, что всякий граф с $\leq n$ вершинами, удовлетворяющий определению 14 входит в \mathcal{D} . Пусть граф $G = (V, E)$ с $(n + 1)$ -й вершиной удовлетворяет условиям определения 14. По условию (1) в нем имеется вершина-корень r_0 . Пусть из r_0 выходит k ребер и они ведут в вершины r_1, \dots, r_k ($k \geq 1$). Обозначим через G_i , ($i = 1, \dots, k$) граф, включающий вершины $V_i = \{v \in V \mid v \text{ достижима из } r_i\}$ и соединяющие их ребра $E_i \subseteq E$. Легко понять, что G_i удовлетворяет условиям определения 14. Действительно, в r_i не входят ребра, т.е. эта вершина — корень G_i . В каждую из остальных вершин из V_i входит по одному ребру как и в G . Если $v \in V_i$, то она достижима из корня r_i по определению графа G_i . Так как $|V_i| \leq n$, то по индуктивному предположению $G_i \in \mathcal{D}$. Тогда граф G получен по индуктивному правилу (2) определения 14 из деревьев G_1, \dots, G_k и поэтому принадлежит классу \mathcal{D} .

\Leftarrow Если некоторый граф $G = (V, E)$ входит в класс \mathcal{D} , то выполнение условий (1)-(3) определения 14 для него легко установить индукцией по определению 15. Предоставляем это читателю в качестве упражнения. \square

С ориентированными деревьями связана богатая терминология, пришедшая из двух источников: ботаники и области семейных отношений.

Корень — это единственная вершина, в которую не входят ребра, *листья* — это вершины, из которых не выходят ребра. Путь из корня в лист называется *ветвью* дерева. *Высота дерева* — это максимальная из длин его ветвей. *Глубина вершины* — это длина пути из корня в эту вершину. Для вершины $v \in V$, подграф дерева $T = (V, E)$, включающий все достижимые из v вершины и соединяющие их ребра из E , образует *поддерево* T_v дерева T с корнем v (см. задачу 23). *Высота вершины* v — это высота дерева T_v . Граф, являющийся объединением нескольких непересекающихся деревьев называется *лесом*.

Если из вершины v ведет ребро в вершину w , то v называется *отцом* w , а w — *сыном* v (в последнее время в англоязычной литературе употребляется асексуальная пара терминов: родитель - ребенок). Из определения дерева непосредственно следует, что у каждой вершины кроме корня имеется единственный отец. Если из вершины v ведет путь в вершину w , то v называется *предком* w , а w — *потомком* v . Вершины, у которых общий отец называются *братьями* или *сестрами*.

Выделим еще один класс графов, обобщающий ориентированные деревья — *ориентированные графы без циклов*. Два вида таких размеченных графов будут использованы в главе 6 для представления булевых функций. У этих графов может быть несколько корней - вершин, в которые не входят ребра, и в каждую вершину может входить несколько ребер, а не одно как у деревьев.

4.2 Деревья и формулы (выражения)

Напомним, что в главе 2 было введено общее понятие *формулы* над системой функций \mathcal{B} , которое применимо для произвольных функций, а не только булевых. В главе 4, аналогичные синтаксические объекты для логики предикатов названы *термами*, а в языках программирования такие конструкции часто называются *выражениями*.

Итак, пусть формула над множеством функций \mathbf{F} , множеством констант \mathbf{C} и множеством переменных \mathbf{Var} определяется индуктивно по следующим правилам:

- Переменная из \mathbf{Var} есть формула.
- Константа из \mathbf{C} есть формула.
- Если g_1, \dots, g_k — формулы, а $f^{(k)}$ — k -местная функция из \mathbf{F} , то $f(g_1, \dots, g_k)$ — это формула.

Обозначим множество всех таких формул через $\mathcal{F}(\mathbf{F}, \mathbf{C}, \mathbf{Var})$.

Рассмотрим класс упорядоченных размеченных деревьев $\mathcal{T}(\mathbf{F}, \mathbf{C}, \mathbf{Var})$, листья которых помечены элементами из $(\mathbf{C} \cup \mathbf{Var})$, а внутренние вершины — функциями из \mathbf{F} , причем, если вершина помечена символом k -местной функции из \mathbf{F} , то у нее имеется k сыновей.

Предложение 1. Между множеством формул $\mathcal{F}(\mathbf{F}, \mathbf{C}, \mathbf{Var})$ и множеством деревьев $\mathcal{T}(\mathbf{F}, \mathbf{C}, \mathbf{Var})$ имеется взаимно однозначное соответствие.

Доказательство. Это соответствие легко устанавливается индукцией по определениям формул и деревьев. Оно показано на рис. 10.

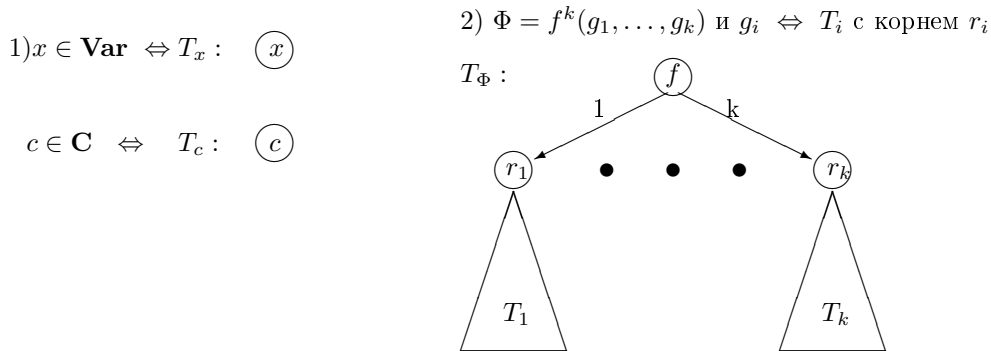


Рис. 10: Индуктивное определение связи между формулами и деревьями

Пример 6. Рассмотрим, например, класс обычных арифметических формул над множеством функций $\mathbf{F} = \{+, -, *, :\}$, целочисленных констант $\mathbf{C} = \{0, 1, 2, \dots\}$ и переменных $\mathbf{Var} = \{x, y, z, \dots\}$. Пусть формула $\Phi = +(* (5, (x - 7)), (: (y, +(x, 2)))$ (ее обычное представление $\Phi = 5 * (x - 7) + y : (x + 2)$)

Тогда в соответствии с предложением 1 эта формула представляется деревом T_Φ , изображенном на рис. 11.

4.3 Обходы деревьев

Часто требуется при обработке представленной в дереве информации обойти некоторым регулярным способом все его вершины. Имеется два естественных стандартных способа обхода деревьев. Каждый из них позволяет линейно упорядочить вершины дерева и тем самым представить его “двумерную” структуру” в виде линейной последовательности вершин.

Прямой (префиксный) обход дерева основан на принципе: “сначала родитель, затем дети”. Определим индукцией по построению дерева T в определении 15 его прямое представление $PR(T)$ следующим образом.

- 1) Если $T_0 = (\{v\}, \emptyset)$, то $PR(T_0) = v$.

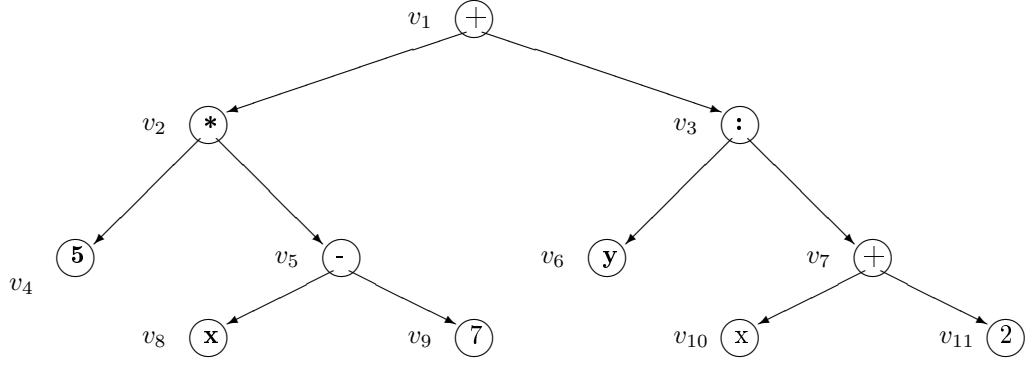


Рис. 11: Дерево T_Φ

2) Если T получено из деревьев T_1, \dots, T_k и нового корня r_0 по пункту (2) определения 15 то $PP(T) = r_0 PP(T_1) \dots PP(T_k)$.

Обратный (суффиксный) обход дерева основан на противоположном принципе: “сначала дети, затем родитель”. Вот его индуктивное определение.

1) Если $T_0 = (\{v\}, \emptyset)$, то $ОБР(T_0) = v$.

2) Если T получено из деревьев T_1, \dots, T_k и нового корня r_0 по пункту (2) определения 15, то $ОБР(T) = ОБР(T_1) \dots ОБР(T_k) r_0$.

Для бинарных деревьев, внутренние вершины которых имеют не более 2-х сыновей, помеченных как “левый” и “правый”, можно естественно определить еще один способ обхода — *инфиксный (внутренний) обход*, основанный на принципе: “сначала левый сын, затем родитель, а затем правый сын”. Он определяется следующим образом.

1) Если $T_0 = (\{v\}, \emptyset)$, то $ИНФ(T_0) = v$.

2) Если T получено из деревьев T_1, T_2 и нового корня r_0 по пункту (2) определения 15, то $ИНФ(T) = ИНФ(T_1)r_0ИНФ(T_2)$

(Если одно из деревьев T_1, T_2 пусто, то соответствующее ему инфиксное представление тоже пусто).

Пример 7. Построим в соответствии с этими определениями три разных обхода бинарного дерева T_Φ , изображенного на рис. 11 (в скобках после вершины указана ее метка).

$$PP(T_\Phi) = v_1(+)v_2(*)v_4(5)v_5(-)v_8(x)v_9(7)v_3(:)v_6(y)v_7(+)v_{10}(x)v_{11}(2).$$

$$ОБР(T_\Phi) = v_4(5)v_8(x)v_9(7)v_5(-)v_2(*)v_6(y)v_{10}(x)v_{11}(2)v_7(+)v_3(:)v_1(+).$$

$$ИНФ(T_\Phi) = v_4(5)v_2(*)v_8(x)v_5(-)v_9(7)v_1(+)v_6(y)v_3(:)v_{10}(x)v_7(+)v_{11}(2).$$

Замечание. Для упорядоченного размеченного дерева T из класса $\mathcal{T}(\mathbf{F}, \mathbf{C}, \mathbf{Var})$ по любому из указанных обходов $PP(T)$, $ОБР(T)$ и, если дерево бинарное, — $ИНФ(T)$ можно однозначно восстановить само дерево T .

5 Построение минимального остова

Во многих задачах в заданном графе нужно выделить некоторую часть, обладающую тем или иным свойством.

Определение 16. Граф $G_1 = (V_1, E_1)$ называется *подграфом* графа $G = (V, E)$, если $V_1 \subseteq V$ и $E_1 \subseteq E$.

Для неориентированных связных графов одним из интересных классов подграфов являются деревья, сохраняющие связность вершин. Они называются *остовами*, *остовными деревьями*, *каркасами* или *скелетами* графа.

Определение 17. *Остовом* (неориентированного) связного графа $G = (V, E)$ называется его подграф $S = (V, T)$, являющийся деревом.

Пусть задана функция $c : E \rightarrow R$, приписывающая каждому ребру $e \in E$ его стоимость (вес, длину) $c(e) \in R$ (R — множество вещественных чисел). Тогда стоимость $c(S)$ дерева S определяется как сумма стоимостей всех его ребер, т.е. $c(S) = \sum_{e \in T} c(e)$.

Минимальным остовом называется остов минимальной стоимости.

Таким образом, минимальный остов — это самая дешевая (короткая) система путей, связывающая все вершины G .

Опишем процедуру построения минимального остова, предложенную Дж. Крускалом в 1956г.

Алгоритм МинОстов

Вход: связный граф $G = (V, E)$ и функция стоимости ребер $c : E \rightarrow R$.

Выход: минимальный остов $S = (V, T)$.

Этап 1. Пусть E содержит m ребер. Упорядочим их по возрастанию стоимостей: $E = \{e_1, e_2, \dots, e_i, \dots, e_m\}$ так, что $c(e_1) \leq c(e_2) \leq \dots \leq c(e_i) \leq \dots \leq c(e_m)$.

Этап 2. Последовательно для каждого $i = 1, \dots, m$ определим множество ребер T_i :

$$T_1 = \{e_1\};$$

...

$$T_i = \begin{cases} T_{i-1} \cup \{e_i\}, & \text{если во множестве } T_{i-1} \cup \{e_i\} \text{ нет циклов} \\ T_{i-1}, & \text{в противном случае} \end{cases}$$

...

Положим $T = T_m$.

Выдать в качестве результата граф $S = (V, T)$.

Докажем, что этот алгоритм корректен.

Теорема 4. Алгоритм МинОстов строит минимальный остов входного графа $G = (V, E)$.

Доказательство. Пусть результатом работы МинОстов на графе $G = (V, E)$ является граф $S = (V, T)$. Отметим вначале, что S является деревом. Действительно, отсутствие циклов следует из определения множеств T_i . Предположим, что S не является связным. Тогда должны существовать две вершины $u, v \in V$, которые не достижимы друг из друга в S . Но граф G связен, поэтому в нем есть путь из u в v . Тогда на этом пути обязательно имеется такое ребро $e_i = (a, b) \in (E \setminus T)$, у которого один конец a соединен путем с u в графе S , а второй конец b — нет. Но тогда на шаге i ребро e_i должно попасть в T_i , так как его добавление не образует цикла. Следовательно, граф S связен.

Покажем теперь, что дерево S имеет минимальную стоимость. Пусть $T = \{d_1, \dots, d_k, \dots, d_{n-1}\}$ — упорядочение всех ребер T по стоимости.

Покажем индукцией по $k = 1, \dots, (n-1)$, что существует минимальный остов, включающий ребра d_1, \dots, d_k .

Пусть $S' = (V, T')$ — минимальный остов, у которого $(k-1)$ наименьших по стоимости ребер совпадают с ребрами T , т.е. упорядочение всех его ребер имеет вид: $T' = \{f_1 = d_1, \dots, f_{k-1} = d_{k-1}, f_k, \dots, f_{n-1}\}$ и $f_k \neq d_k$. Пусть $d_k = (u, v)$. В T' имеется некоторый путь p из u в v , который не содержит ребро d_k . На этом пути обязательно есть некоторое ребро $f = (u', v')$, не попавшее в T , иначе в T образовался бы цикл. Из построения T следует, что $c(d_k) \leq c(f)$. Рассмотрим граф $S'' = (V, T' \setminus \{f\}) \cup \{d_k\}$. Очевидно, что этот граф является остовным деревом. Действительно, связь между u' и v' сохранилась, так как в S'' имеется путь: $u' \leftrightarrow \dots \leftrightarrow u \xrightarrow{d_k} v \leftrightarrow \dots \leftrightarrow v'$. Поэтому S'' — связный граф. Если бы в S'' был цикл, то он обязательно включал бы ребро d_k . Но тогда часть этого цикла без ребра d_k образовывала бы путь p' между u и v , не совпадающий с путем p . Следовательно, в дереве T' было бы два разных пути между u и v , что невозможно, так как тогда в T' был бы цикл. Отсюда заключаем, что в S'' циклов нет и S'' — остовное дерево. Его стоимость $c(S'') = c(S') - c(f) + c(d_k) \leq c(S')$. Так как S' — минимальный остов, то $c(S'') = c(S')$ и S'' — тоже минимальный остов, у которого с S имеется k общих ребер: d_1, \dots, d_k .

Тогда при $k = n-1$ получаем, что S — минимальный остов.

□

Пример 8. Рассмотрим нагруженный граф G , показанный на рис. 12.

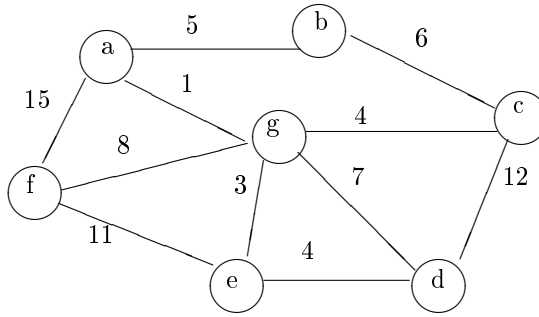


Рис. 12: Граф G

Применим к нему алгоритм **МинОстов**. На первом этапе упорядочим все ребра, а на втором — рядом с каждым из них определим соответствующее множество T_i . Ребра, попавшие в T будем по ходу вычисления отмечать знаком '+', а не попавшие — знаком '-'.

E	$c(e)$	входит	T_i
(a, g)	1	+	$T_1 = \{(a, g)\}$
(g, e)	3	+	$T_2 = \{(a, g), (g, e)\}$
(g, c)	4	+	$T_3 = \{(a, g), (g, e), (g, c)\}$
(e, d)	4	+	$T_4 = \{(a, g), (g, e), (g, c), (e, d)\}$
(a, b)	5	+	$T_5 = \{(a, g), (g, e), (g, c), (e, d), (a, b)\}$
(b, c)	6	-	$T_6 = T_5$
(d, g)	7	-	$T_7 = T_6$
(f, g)	8	+	$T_8 = \{(a, g), (g, e), (g, c), (e, d), (a, b), (f, g)\}$
(e, f)	11	-	$T_9 = T_8$
(c, d)	12	-	$T_{10} = T_9$
(a, f)	15	-	$T_{11} = T_{10}$

Таким образом, мы построили для G минимальный остов $S = (V, T)$, где $T = T_8 = \{(a, g), (g, e), (g, c), (e, d), (a, b), (f, g)\}$. Он показан на рис. 13. Стоимость этого остова $c(S) = 25$.

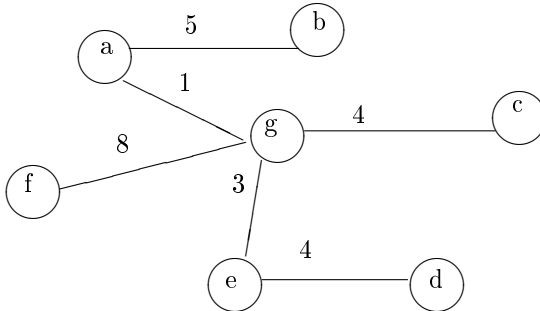


Рис. 13: Минимальный остов $S = (V, T)$ для графа G

Замечание. Так как дерево с n вершинами содержит в точности $(n - 1)$ ребер, то работу алгоритма **МинОстов** можно прекращать после такого шага i , на котором в T_i окажется $|V| - 1$ ребер. В нашем примере $|V| = 7$ и алгоритм мог остановиться после 8-го шага.

6 Поиск в глубину и задача о лабиринте

Задача поиска выхода из лабиринта известна с древних времен. В терминах графов ее можно формализовать так: лабиринт — это неориентированный граф, вершины которого представляют “перекрестки” лабиринта, а ребра — дорожки между соседними перекрестками. Одна или несколько вершин отмечены как выходы. Задача состоит в построении пути из некоторой исходной вершины в вершину-выход.

В этом разделе мы рассмотрим метод обхода всех вершин графа, называемый *поиском в глубину*. Его идею кратко можно описать так: находясь в некоторой вершине v , идем из нее в произвольную еще не посещенную смежную вершину w , если такой вершины нет, то возвращаемся в вершину, из которой мы пришли в v .

Алгоритм поиска в глубину

Вход: $G = (V, E)$ — неориентированный граф, представленный списками смежностей: для каждой $v \in V$ список L_v содержит перечень всех смежных с v вершин.

Выход: $NUM[v]$ — массив с номерами вершин в порядке их прохождения и множество (древесных) ребер $T \subseteq E$, по которым осуществляется обход.

Алгоритм ПОГ

1. $T = \{\}$; $NOMER = 1$;
2. для каждой вершины $v \in V$ положим $NUM[v] = 0$ и пометим v как “новую”;
3. **ПОКА** существует “новая” вершина $v \in V$
4. **ВЫПОЛНЯТЬ** ПОИСК(v).

Основную роль в этом алгоритме играет следующая рекурсивная процедура.

Алгоритм ПОИСК(v):

1. пометить v как “старую”;
2. $NUM[v] = NOMER$; $NOMER = NOMER + 1$;
3. **ДЛЯ КАЖДОЙ** $w \in L_v$ **ВЫПОЛНЯТЬ**
4. **ЕСЛИ** вершина w “новая”
5. **ТО**
6. { добавить (v, w) к T ;
7. ПОИСК(w);
8. }

Теорема 5. Алгоритм ПОГ обходит (нумерует) все вершины графа $G = (V, E)$. Если G — связный граф, то $S = (V, T)$ — это остов G , если граф G не является связным, то $S = (V, T)$ — это остовный лес для G , т.е. объединение остовных деревьев для каждой из компонент связности G .

Доказательство. Первое утверждение следует из того, что по окончании алгоритма ПОГ все вершины графа старые, а это значит, что для каждой из них вызывалась процедура ПОИСК, которая в стр.2 присвоила номер.

Заметим теперь, что если ребро (v, w) попадает в T , то вызов процедуры ПОИСК(w) происходит после вызова ПОИСК(v) и поэтому $NUM[v] < NUM[w]$. Существование цикла в S означало бы, что для некоторого ребра из T это свойство нарушено (почему?). Следовательно, в S циклов нет.

Пусть $G_1 = (V_1, E_1)$ — связная компонента G и $v_1 \in V_1$ — первая ее вершина, для которой вызывается процедура ПОИСК. Тогда для каждой вершины $w \in V_1$ внутри вызова ПОИСК(v_1) произойдет вызов ПОИСК(w).

Это утверждение доказывается индукцией по расстоянию (длине кратчайшего пути) от v_1 до w .

Если это расстояние равно 1, то $w \in L_{v_1}$ и рассматривается в вызове ПОИСК(v_1) в стр.3. Если w в этот момент “старая”, то значит ПОИСК(w) уже вызывался. Если же w “новая”, то в стр. 7 происходит вызов ПОИСК(w).

Предположим теперь, что ПОИСК(u) вызывается для всех вершин u , находящихся на расстоянии $k \geq 1$ от v_1 , и пусть вершина $w \in L_u$ находится на расстоянии $(k + 1)$ от v_1 . Тогда имеется путь длины $(k + 1)$ от v_1 до w . Пусть u — это предпоследняя вершина на этом пути. Тогда расстояние от v_1 до u равно k и по нашему предположению в некоторый момент выполняется вызов ПОИСК(u). Так как $w \in L_u$, то в этом вызове вершина w в некоторый момент рассматривается в цикле в стр.3. Как и выше, если она в этот момент “старая”, то ПОИСК(w) уже вызывался. Если же w еще “новая”, то в стр.7 происходит вызов ПОИСК(w).

Также по индукции замечаем, что если вызов ПОИСК(w) произошел внутри вызова ПОИСК(v), то в T имеется путь из v в w . Следовательно, граф $S_1 = (V_1, T_1)$, построенный в процессе вы-

зова $\text{ПОИСК}(v)$ является деревом с корнем v . \square

Дерево $S = (V, T)$, которое строится алгоритмом **ПОГ** называется *глубинным остовным деревом* графа G . Ребра, попавшие в множество T , называются *прямыми*, а не попавшие в это множество ребра из множества $(E \setminus T)$ — *обратными*. Каждое обратное ребро определяет некоторый цикл в исходном графе G (почему?). Поэтому алгоритм **ПОГ** можно использовать для проверки наличия циклов в G .

Пример 9. Применим алгоритм **ПОГ** к графу G_2 , изображенному на рисунке 14.

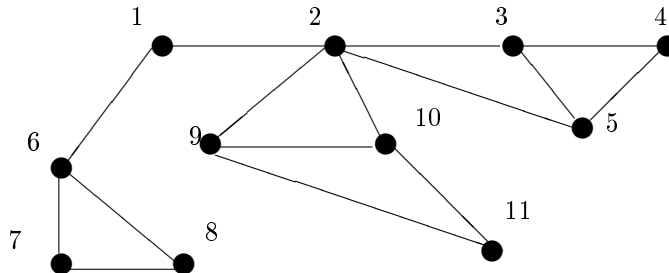


Рис. 14: Граф G_2

Его представление в виде списков смежности имеет следующий вид:

$L_1 : 6, 2$	$L_7 : 6, 8$
$L_2 : 1, 9, 10, 3$	$L_8 : 6, 7$
$L_3 : 2, 5, 4$	$L_9 : 2, 10, 11$
$L_4 : 3, 5$	$L_{10} : 2, 9, 11$
$L_5 : 2, 3, 4$	$L_{11} : 9, 10$
$L_6 : 1, 7, 8$	

Алгоритм **ПОГ** вызовет процедуру $\text{ПОИСК}(1)$. Эта процедура рекурсивно вызовет $\text{ПОИСК}(6)$ и т.д. Вот структура всех получающихся вызовов процедуры ПОИСК :

$\text{ПОИСК}(1) \Rightarrow \text{ПОИСК}(6) \Rightarrow \text{ПОИСК}(7) \Rightarrow \text{ПОИСК}(8)$
 \Downarrow
 $\text{ПОИСК}(2) \Rightarrow \text{ПОИСК}(9) \Rightarrow \text{ПОИСК}(10) \Rightarrow \text{ПОИСК}(11)$
 \Downarrow
 $\text{ПОИСК}(3) \Rightarrow \text{ПОИСК}(5) \Rightarrow \text{ПОИСК}(4)$

Вначале идут “горизонтальные” вызовы, затем возвраты справа налево и вызовы “по вертикали”. В результате вершины G_2 получают следующие номера, отражающие порядок их прохождения:

$V :$	1	2	3	4	5	6	7	8	9	10	11
$NUM :$	1	5	9	11	10	2	3	4	6	7	8

Ребра остова T , построенные в процессе обхода графа, показаны на рисунке 15. Стрелки указывают направление обхода. В скобках рядом с номером вершины указан ее номер в массиве NUM , т.е. номер в порядке обхода алгоритмом **ПОГ**.

В процессе построения этого дерева были определены следующие обратные ребра: $(8, 6)$, $(10, 2)$, $(11, 9)$, $(5, 2)$ и $(4, 3)$. Нетрудно проверить, что добавление любого из этих ребер к T приводит к образованию простого цикла.

Алгоритм поиска в глубину часто используется как основа для различных алгоритмов обработки графов. Вместо строки 6, в которой вершина v получает номер $NUM(v)$, можно вставить

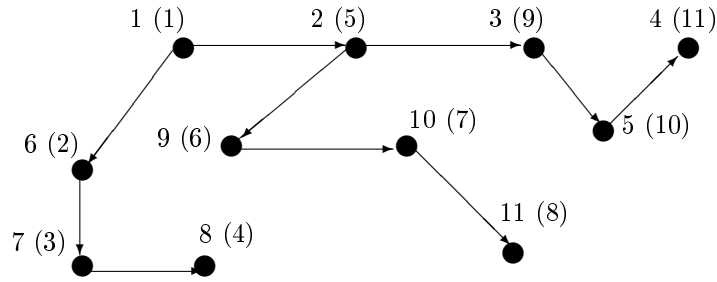


Рис. 15: Остовное “глубинное” дерево $S = (V, T)$ графа G_2

вызов любой процедуры, обрабатывающей информацию, связанную с этой вершиной (например, для задачи о лабиринте это может быть проверка того, что v является выходом из лабиринта). И тогда полученный вариант алгоритма обеспечит обработку всех вершин графа.

7 Задача о кратчайших путях из одного источника

Пусть $G = (V, E)$ — ориентированный граф, для каждого ребра $e \in E$ которого указана его (неотрицательная) длина: $c(e) \geq 0$. Тогда длина пути $p = v_1, v_2, \dots, v_{k+1}$ определяется как сумма длин ребер, входящих в этот путь: $c(p) = \sum_{i=1}^k c(v_i, v_{i+1})$. Если в G имеется путь из вершины a в вершину b , то имеется и такой путь минимальной длины. Он называется *кратчайшим путем из a в b* . Конечно, в графе может оказаться несколько различных кратчайших путей из a в b .

Естественно спросить, как узнать длину кратчайшего пути из a в b и построить его? Лучшие известные на сегодняшний день алгоритмы, отвечающие на этот вопрос, решают, на самом деле, более общую задачу построения всех кратчайших путей из одного источника: *по вершине a найти длины кратчайших путей из a во все достижимые из нее вершины и построить для каждой из таких вершин некоторый кратчайший путь из a* . Если для каждой вершины $v \in V$, достижимой из a , зафиксировать один кратчайший путь из a в v , то получившийся граф будет представлять ориентированное дерево с корнем a (докажите это!). Это дерево называется *деревом кратчайших путей из a* .

Мы рассмотрим алгоритм построения дерева кратчайших путей и определения их длин, предложенный в 1959г. Е. Дейкстрой. Его идея следующая: перед каждым этапом известно множество *отмеченных вершин* S , для которых кратчайшие пути найдены ранее; тогда на очередном этапе к нему добавляется вершина w , с самым коротким путем из a , проходящим по множеству S ; после этого пересчитываются длины кратчайших путей из a в оставшиеся вершины из $V \setminus S$ с учетом новой вершины w . Длина текущего кратчайшего пути из a в v , проходящего по множеству S , заносится в ячейку $D[v]$ массива D . В конце работы в этом массиве находятся длины соответствующих кратчайших путей. Для определения дерева кратчайших путей служит массив ОТЕЦ, его элемент ОТЕЦ[v] содержит ссылку на вершину, из которой кратчайший путь приходит в v .

Алгоритм Дейкстры

Вход: $G = (V, E)$ — ориентированный граф, $c(u, v) \geq 0$ — длина ребра $(u, v) \in E$ (если $(u, v) \notin E$, то считаем, что $c(u, v) = \infty$) и исходная вершина $a \in V$.

’ === ИНИЦИАЛИЗАЦИЯ ===

1. $S := \{a\}$; ’ отметить a
2. $D[a] := 0$; ’ расстояние от a до a
3. **ДЛЯ КАЖДОЙ** $v \in V, v \neq a$ **ВЫПОЛНЯТЬ**
4. $\{D[v] := c(a, v)$; ’ расстояние от a до v через a
5. **ЕСЛИ** $c(a, v) < \infty$ **ТО** ОТЕЦ[v] := a **ИНАЧЕ** ОТЕЦ[v] := $-$ };

’ === ОСНОВНОЙ ЦИКЛ ===

6. **ПОКА** $V \setminus S \neq \emptyset$ **ВЫПОЛНЯТЬ** ' есть неотмеченные вершины
7. { выбрать неотмеченную вершину w с минимальным $D[w]$;
8. $S := S \cup \{w\}$; ' отметить w
9. **ДЛЯ КАЖДОЙ** (неотмеченной) $u \in V \setminus S$ **ВЫПОЛНЯТЬ**
10. **ЕСЛИ** $D[u] > D[w] + c(w, u)$
11. **ТО** { $D[u] := D[w] + c(w, u)$;
12. ОТЕЦ[u] := w }
13. }

Пример 10. Рассмотрим работу этого алгоритма на нагруженном графе $G = (V = \{a, b, c, d, e, f\}, E)$ и выделенной вершине $a \in V$. Зададим длины ребер матрицей $C = (c_{uv})$, где элемент $c_{uv} = c(u, v)$:

$$\begin{pmatrix} & a & b & c & d & e & f \\ a & 0 & 25 & 5 & 30 & \infty & 75 \\ b & 12 & 0 & \infty & \infty & 120 & 20 \\ c & \infty & 15 & 0 & 20 & 45 & 60 \\ d & \infty & \infty & \infty & 0 & 23 & 20 \\ e & \infty & \infty & 75 & 20 & 0 & 20 \\ f & 40 & 15 & 15 & 26 & \infty & 0 \end{pmatrix}$$

Поэтапную работу алгоритма Дейкстры удобно представлять в виде таблицы, строки которой соответствуют его этапам. Первый столбец — номер этапа, второй показывает изменение множества отмеченных вершин S , третий — вершину w , добавляемую к S на текущем шаге, четвертый — длину кратчайшего пути из a в w , затем идут столбцы со значениями элементов массивов D и ОТЕЦ.

N	S	w	$D[w]$	D					О Т Е Ц				
				b	c	d	e	f	b	c	d	e	f
1.	a	c	5	25	5	30	∞	75	a	a	a	-	a
2.	a, c	b	20	20	-	25	50	65	c	a	c	c	c
3.	a, c, b	d	25	-	-	25	50	40	c	a	c	c	b
4.	a, c, b, d	f	45	-	-	-	48	45	c	a	c	d	b
5.	a, c, b, d	e	45	-	-	-	-	45	c	a	c	d	b

Таблица 1: Алгоритм Дейкстры на графе G .

Дерево кратчайших путей из вершины a задается массивом ОТЕЦ. Оно представлено на рисунке 16.

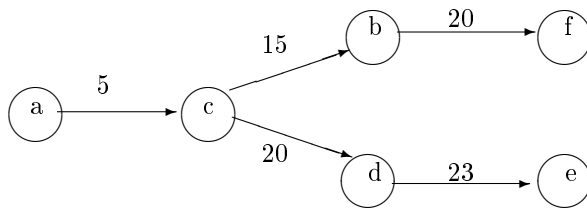


Рис. 16: Дерево кратчайших путей из вершины a в графе G

Теорема 6. (о корректности алгоритма Дейкстры)

Алгоритм Дейкстры строит дерево кратчайших путей из вершины a во все достижимые из нее вершины u и для каждой такой вершины v определяет длину $D[v]$ кратчайшего пути в нее из a .

Доказательство. Докажем по индукции, что после каждого этапа алгоритма выполнены следующие условия:

а) для любой вершины $v \in S$ величина $D[v]$ равна длине кратчайшего пути из a в v ;

- б) для любой вершины $v \in V \setminus S$ величина $D[v]$ равна длине кратчайшего пути из a в v , проходящего по множеству S ;
- в) для каждой вершины v дерева T , задаваемого массивом ОТЕЦ, длина пути из корня a в v равна $D[v]$.

Эти три условия, очевидно, выполняются после инициализации в строках 1- 5.

Предположим теперь что они выполнены перед началом k -го этапа. Пусть w — вершина, добавляемая к S на k -ом этапе. По предположению, $D[w]$ — длина кратчайшего пути из a в w , все вершины которого, кроме w , входят в S . Предположим, что есть другой более короткий путь p из a в w . Зафиксируем на этом пути первую вершину u , не входящую в S . По выбору p и $u \neq w$. Поэтому путь p разбивается на две непустые части: путь p_1 из a в u и путь p_2 из u в w . Но по выбору w мы имеем, что длина $p_1 \geq D[u] \geq D[w]$. Так как длина p_2 неотрицательна, то длина $p \geq D[w]$, т.е. этот путь не короче пути, представленного в дереве T . Таким образом, $D[w]$ — это длина кратчайшего пути из a в w . Следовательно, условие (а) выполнено и после k -го этапа.

Рассмотрим теперь произвольную вершину $u \in V \setminus (S \cup \{w\})$. Кратчайший путь p из a в u , проходящий по множеству $S \cup \{w\}$, либо не включает вершину w и в этом случае его длина равна $D[u]$ и он имеется в текущем дереве T , либо он проходит через w и составлен из кратчайшего пути из a в w через S , продолженного ребром (w, u) . В последнем случае длина пути равна $D[w] + c(w, u)$. Но в 10-ой строке алгоритма эти величины сравниваются и, если путь через w короче, то его длина становится новым значением $D[u]$ (строка 11) и он фиксируется в дереве T (строка 12). Следовательно, условия (б) и (в) также выполнены после k -го этапа.

Так как после завершения алгоритма $S = V$, то в завершающем дереве T представлены кратчайшие пути из a во все достижимые из нее вершины, а массив D содержит длины этих путей. Значение $D[u] = \infty$ указывает на то, что вершина u не достижима из вершины a .

□

Замечание о сложности. На каждом этапе (исполнении тела основного цикла в стр. 6 - 13) одна вершина добавляется во множество S . Поэтому таких этапов не более $|V|$. Чтобы выбрать в массиве D вершину w с минимальным $D[w]$ (стр. 7) требуется не более $|V|$ шагов. Перевычисление $D[u]$ для каждой из вершин $u \in (V \setminus S)$ требует константного числа операций, поэтому весь цикл в стр. 9 - 12 потребует не более $c|V|$ шагов. Отсюда получаем, что для некоторой константы c время выполнения алгоритма Дейкстры не превышает $c|V|^2$. Поскольку размер любого представления исходного графа не меньше $|V|$, то алгоритм работает в *квадратичное время* (от размера входа).

8 Задачи

Задача 6. Докажите, что неориентированный связный граф остается связным после удаления некоторого ребра \leftrightarrow это ребро принадлежит некоторому циклу.

Задача 7. Докажите, что если в связном неориентированном графе число вершин равно числу ребер, то можно выбросить одно из ребер так, что после этого граф станет деревом.

Задача 8. Докажите, что неориентированный связный граф с n вершинами

- содержит не менее $n - 1$ ребер,
- если содержит больше $n - 1$ ребер, то имеет по крайней мере хоть один цикл.

Задача 9. Докажите, что в любой группе из 6 человек есть трое попарно знакомых или трое попарно незнакомых.

Задача 10. Докажите, что неориентированный граф $G = (V, E)$ связан \leftrightarrow для каждого разбиения $V = V_1 \cup V_2$ с непустыми V_1 и V_2 существует ребро, соединяющее V_1 с V_2 .

Задача 11. Докажите, что, если в неориентированный графе без петель имеется ровно две вершины нечетной степени, то они связаны путем.

Задача 12. Цикл в связном неориентированном графе называется Эйлеровым, если он проходит по одному разу через каждое ребро графа. Докажите, что в таком графе имеется Эйлеров цикл тогда и только тогда, когда степени всех его вершин четны (такие графы называются четными).

Указание. Достаточность можно установить, доказав правильность следующей процедуры

построения Эйлера цикла.

1) Выбрать произвольную вершину a и построить цикл, начинающийся и заканчивающийся в a , следуя правилу (*): прийдя в некоторую вершину, выйти из нее по произвольному ребру, еще не включенному в цикл.

2) Если построенный цикл содержит не все ребра, то выбрать среди вершин, через которые он проходит, произвольную вершину b , инцидентную еще не пройденному ребру (почему такая вершина найдется?), и построить, начиная с нее, цикл, следуя тому же правилу (*). Объединить этот цикл с построенным ранее.

3) Повторять пункт (2) до тех пор, пока построенный цикл не станет Эйлеровым.

Задача 13. Используя результаты предыдущей задачи, определить по неориентированному графу $G = (V, E)$ четный ли он. Если он не является четным, то удалить из него минимальное число ребер, чтобы он стал четным. Построить в исходном или в получившемся после удаления ребер четном графе Эйлера цикл.

$V = \{a, b, c, e, f, g, h, k, m, n\}$, $E = \{(a, c), (a, h), (a, m), (a, k), (b, c), (b, k), (b, f), (b, m), (c, k), (c, m), (e, f), (e, g), (f, k), (f, n), (g, m), (g, h), (h, k), (h, m), (k, n)\}$.

Задача 14. Неориентированный граф $G = (V, E)$ называется двудольным, если его вершины можно разбить на две непересекающиеся части X и Y ($V = X \cup Y, X \cap Y = \emptyset$) так, что каждое ребро из $e \in E$ соединяет вершину из X с вершиной из Y . Такой граф также называется бихроматическим, так как его вершины можно раскрасить в два цвета так, что соседние вершины будут окрашены в разные цвета.

Докажите, что граф является двудольным тогда и только тогда, когда в нем нет циклов нечетной длины.

Указание. Достаточность можно установить, доказав правильность следующей процедуры разбиения V на X и Y :

1) Поместить произвольную вершину $v \in V$ в X и отметить ее знаком $+$.

2) ПОКА имеются неотмеченные вершины с отмеченными соседями

ВЫПОЛНЯТЬ {

2.1) Поместить все неотмеченные вершины с соседями из X в Y и отметить их знаком $-$;

2.2) Поместить все неотмеченные вершины с соседями из Y в X и отметить их знаком $+$ };

3) ЕСЛИ после завершения цикла 2 в V остались неотмеченные вершины

ТО поместить произвольную такую вершину v в X , отметить ее знаком $+$ и снова повторить цикл 2

ИНАЧЕ выдать в качестве результата полученные множества: X – вершины, отмеченные $+$, и Y – вершины, отмеченные $-$.

Для доказательства корректности этой процедуры установите, что в процессе разметки ни одна вершина не получит соседа с той же меткой.

Задача 15. Используя результаты предыдущей задачи, определить является ли заданный ниже неориентированный граф $G = (V, E)$ двудольным. Если он не двудольный, то каково минимальное число ребер, которые нужно из него удалить, чтобы он стал двудольным? Приведите обоснование ответа.

$V = \{a, b, c, e, f, g, h, k, m, n\}$, $E = \{(a, h), (a, n), (a, k), (b, k), (b, f), (b, m), (c, k), (c, h), (e, f), (e, g), (f, a), (f, m), (g, m), (m, n)\}$.

Задача 16. Пусть $G = (V, E)$ неориентированный граф с $|E| < |V| - 1$. Докажите, что тогда G несвязный граф.

Задача 17. Докажите, что в связном неориентированном графе любые два простых пути максимальной длины имеют общую вершину.

Задача 18. Пусть неориентированный граф без петель $G = (V, E)$ имеет k компонент связности. Доказать, что тогда

$$|E| \leq (|V| - k)(|V| - k + 1)/2.$$

Задача 19. Пусть $G = (V, E)$ — неориентированное дерево и $v \in V$ — произвольная вершина. Докажите, что если для каждого ребра $(u, w) \in E$ выбрать ориентацию от u к w , если им заканчивается путь из v в w , и ориентацию от w к u , если им заканчивается путь из v в u , то полученный ориентированный граф будет ориентированным деревом с корнем v . Используйте это утверждение для доказательства следующего факта: если в неориентированном дереве $G = (V, E)$ имеется вершина степени $d > 1$, то в нем имеется по крайней мере d вершин степени 1.

Задача 20. Для каждого из обходов деревьев $PP(T)$, $ОБР(T)$ и $ИНФ(T)$ предложите процедуру, восстановления соответствующего дерева $T \in \mathcal{T}(\mathbf{F}, \mathbf{C}, \mathbf{Var})$.

Задача 21. Постройте дерево, представляющее следующую логическую формулу $\Psi = ((X \vee \neg Y) \wedge \neg(Z \rightarrow (X \wedge Y))) \vee (\neg Z + Y)$.

Для полученного дерева определите прямой, обратный и инфиксный обходы.

Задача 22. Ориентированное дерево называется двоичным (или бинарным), если степень исхода каждой его вершины не больше двух. Докажите по индукции, что в любом двоичном дереве число вершин степени 2 на единицу меньше числа листьев.

Задача 23. Пусть $T = (V, E)$ — это ориентированное дерево с корнем $v_0 \in V$. определим для каждой вершины $v \in V$ подграф $T_v = (V_v, E_v)$ следующим образом: V_v — это множество вершин, достижимых из v в T , а E_v — это множество ребер из E , оба конца которых входят в V_v . Доказать, что

- T_v является деревом с корнем v ;
- если две разные вершины v и u имеют одинаковую глубину, то деревья T_v и T_u не пересекаются.

Задача 24. Пусть $G = (V, E)$ — ориентированный граф с $n > 1$ вершинами. Докажите, что G является (ориентированным) деревом тогда и только тогда, когда в G нет циклов, имеется одна вершина r , в которую не входят ребра, а в каждую из остальных вершин $v \in V \setminus \{r\}$ входит ровно одно ребро.

Задача 25. Измените алгоритм обхода “в глубину” так, чтобы он позволил перечислить все связные компоненты неориентированного графа.

Задача 26. Найти минимальное остовное дерево для неориентированного графа $G = (V, E)$, где $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$,
 $E = \{(v_1, v_2, 18), (v_1, v_3, 2), (v_3, v_2, 4), (v_3, v_4, 6), (v_3, v_5, 8), (v_4, v_6, 5), (v_5, v_4, 4), (v_6, v_1, 7), (v_6, v_8, 4), (v_6, v_7, 3), (v_7, v_9, v_1, 1)\}$ (третий параметр в скобках — стоимость ребра).

Задача 27. Пусть $S = (V, T)$ — остовное дерево наименьшей стоимости, построенное алгоритмом МИНОД для нагруженного неориентированного графа $G = (V, E)$ с n вершинами. Пусть $c_1 \leq c_2 \leq \dots \leq c_{n-1}$ — это последовательность длин ребер из T , упорядоченных по возрастанию. Пусть S' — произвольное остовное дерево для G с длинами ребер $d_1 \leq d_2 \leq \dots \leq d_{n-1}$. Показать, что $c_i \leq d_i$ для всех $i: 1 \leq i \leq n-1$.

Задача 28. Пусть e — ребро максимального веса в некотором цикле графа $G = (V, E)$. Докажите, что существует минимальный остов графа $G' = (V, E \setminus \{e\})$, который является также минимальным остовом графа G .

Задача 29. Пусть $D = (V, T)$ — это остовное дерево, построенное алгоритмом обхода “в глубину” для графа $G = (V, E)$. Докажите, что для каждого ребра (u, v) из E , не попавшего в T (такие ребра называются обратными), либо u является предком v в D , либо v является предком u в D .

Задача 30. Обойти (занумеровать) вершины заданного неориентированного графа G с помощью алгоритма обхода “в глубину” и построить дерево этого обхода.

$G = (V, E)$, где $V = \{v_1, v_2, v_3, v_4, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$,
 $E = \{(v_1, v_2), (v_1, v_4), (v_1, v_8), (v_7, v_8), (v_2, v_9), (v_9, v_{11}), (v_3, v_8), (v_6, v_3), (v_3, v_7), (v_6, v_7), (v_{10}, v_9)\}$.

Какое обратное ребро $e \in E \setminus T$ и цикл в G обнаружили в этом обходе первыми?

Задача 31. Определить для заданного нагруженного графа $G = (V, E)$ и выделенной вершины $a \in V$ длины кратчайших путей из этой вершины в остальные вершины G и построить дерево этих путей.

$V = \{a, b, c, d, e, f\}$, $E = \{(a, b; 154), (a, c; 17), (a, d; 214), (a, e; 63), (b, d; 25), (c, e; 33), (c, d; 192), (c, b; 123), (d, f; 10), (e, f; 10)\}$ (здесь каждая скобка $(u, v; D)$ задает ребро $(u, v) \in E$ и его “вес” $c(u, v) = D$).

Задача 32. Где в доказательстве правильности алгоритма Дейкстры используется неотрицательность весов ребер? Приведите пример графа (с отрицательными весами), для которого алгоритм Дейкстры дает неверный ответ.

Задача 33. Докажите, что на каждом шаге алгоритма Дейкстры кратчайший путь из исходной вершины в любую вершину множества S проходит только через вершины множества S .

Задача 34. Сколько раз может меняться для одной вершины v значение $D[v]$ в ходе работы алгоритма Дейкстры для графа с b вершинами. Привести пример на каждый возможный случай.

Задача 35. Доказать, что на каждом шаге алгоритма Дейкстры для любой вершины $x \in S$ и любой вершины $y \in V \setminus S$ выполнено неравенство $D[x] \leq D[y]$.