Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Владимирский государственный университет

М.Ю. ЗВЯГИН
М.С. БЕСПАЛОВ
А.В. АЛЕКСАНДРОВ

# ПРИКЛАДНЫЕ АЛГОРИТМЫ НА ГРАФАХ

Учебное пособие

Владимир 2005

## *ОГЛАВЛЕНИЕ*

# *ПРЕДИСЛОВИЕ*

Растущий интерес к дискретной математике вполне понятен. Это молодая, стремительно развивающаяся отрасль математического знания. Дискретная математика создает фундаментальные предпосылки для развития идей алгоритмизации, программирования, информатизации. С другой стороны, развитие вычислительной техники, информационных сетей и технологий постоянно приводит к появлению целого ряда новых задач в дискретной математике. Круг таких задач настолько богат и разнообразен, что в рамках дискретной математики уже сейчас можно выделить множество разделов: комбинаторика, теория множеств, теория графов, теория конечных автоматов и управляемых процессов, теория информации и кодирования, теория алгоритмов, теория сложности алгоритмов. Каждый из перечисленных разделов заслуживает самостоятельного учебного курса, а одно учебное пособие не может их заменить.

В настоящем учебном пособии "Прикладные алгоритмы на графах" рассмотрены практические приложения теории графов и орграфов в широком круге задач. Задачи о кратчайшем или наиболее выгодном пути с определенной точки зрения возникают в вопросах безопасности и эффективности работы информационных, транспортных, экономических, производственных сетей и сообщений. Множество более сложных задач можно свести к повторительных схемам, в которых работают алгоритмически решенные задачи теории графов.

Мы не ставили целью сделать изложение материала учебного пособия полным и математически обоснованным. Наибольшее внимание было уделено матричным способам представления графов, постановкам поисковых задач на графах и методам их алгоритмической реализации, как раз тем прикладным вопросам, с которыми приходится сталкиваться любому программисту.

Авторы благодарят студентов специальности "Комплексная защита объектов информатизации" гр. КЗИ-102, 202, принявших активное участие в обсуждении и подготовке данного пособия, а также М.М. Звягину за помощь в электронной верстке пособия и других технических вопросах.

# ГРАФЫ И ОРГРАФЫ

## Определение графа

*Графом* $G=(V, E)$ называется совокупность двух множеств: конечного непустого множества $V$ (*множество вершин*) и множества $E$, состоящего из неупорядоченных пар различных элементов множества $V$ ($E=\{\{u,v\}\mid u,v\in V,\ u\neq v\}$ - *множество рёбер*). Число вершин графа $G$ называется его *порядком* и обозначается $|G|$ или $|V|$.

Приведено определение конечного простого графа $G(V, E)$ (знак равенства опускаем). Существуют также другие определения графов, которые имеют свои названия.

Граф называется *псевдографом* (или графом с петлями), если допускаются рёбра вида $(v, v)$, представляющие собой пару одинаковых элементов множества вершин $V$.

Граф называется *мультиграфом* (или графом с кратными рёбрами), если $E$ является не множеством, а набором неупорядоченных пар элементов множества $V$ (в наборах допускается повторение одного и того же элемента, которые, в отличие от случая множества, считаются различными).

Граф называется *орграфом* (ориентированным графом) и обозначается $D(V, E)$, если множество $E$ рассматривается как множество упорядоченных пар различных элементов множества $V$ ($E\subseteq V\times V=\{(u,v)\mid u,v\in V\}$).

В орграфе вершины называются *узлами*, а вместо термина ребро используется термин *дуга*. Отметим, что $(u,v)\neq(v,u)$, если $u\neq v$. Иногда данную конструкцию называют псевдоорграф, а орграфом называют орграф без петель в нашей терминологии.

Для любого орграфа $D(V, E)$ можно построить *ассоциированный* с ним мультиграф (псевдомультиграф) $G(V, E_1)$, если каждую упорядоченную пару $(u, v)$ из множества дуг $E$ заменить на аналогичную неупорядоченную пару $\{u, v\}$, которые будут составлять набор (а не множество) ребер $E_1$. Для любого орграфа без петель $D(V, E)$ можно построить *ассоциированный с ним простой граф* $G(V, E_1)$, если каждую упорядоченную пару $(u, v)$ из множества дуг $E$ заменить на аналогичную неупорядоченную пару $\{u, v\}$, которые будут составлять множество ребер $E_1$ (то есть ребра $\{u, v\}$ и $\{v, u\}$ отождествили).

Встречаются смешанные конструкции графа, где множество $E$ есть объединение ребер и дуг.

## Диаграммы

Графы удобно изображать в виде рисунка (диаграммы), где вершины изображают в виде точек или кружочков, а рёбра – как соединяющие эти вершины линии (в случае орграфа дуги показываются в виде стрелок, которые позволяют определить начало и конец дуги). Вершины графа (элементы множества $V$) обозначают буквами с индексами ($v_1$, $v_2$, …) или без индексов ($v$, $u$, $w$, …), или числами начала натурального ряда (1, 2, 3, …). В этом случае граф называется *помеченным*. Орграфы, как правило, рассматриваются помеченными. Рисунок, где вершины не обозначены, называется диаграммой (непомеченного) графа. Итак, термин «граф» будем использовать для неориентированного, непомеченного графа без петель и кратных рёбер.

***Упражнение 1.*** Привести примеры диаграмм: графа, помеченного графа, псевдографа, помеченного псевдографа, мультиграфа, помеченного мультиграфа, орграфа, помеченного орграфа, помеченного орграфа без петель.

***Упражнение 2.*** Для подстановок $\tau = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$; $\rho = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$ построить орграфы (в виде диаграмм). Орграф какой подстановки без петель? С помощью орграфов вычислить подстановки

$$\tau^2, \rho^2, \tau \circ \rho, \rho \circ \tau, \tau^{-1}, \rho^{-1}, \tau \circ \rho^{-1}, \rho^{-1} \circ \tau^{-1}.$$

## Смежность

Если $e = \{u, v\}$ – ребро графа $G$ ($\{u, v\} \in E$), то вершины $u$, $v$ называются концами ребра $e$, а ребро $e$ соединяет вершины $u$, $v$. Тогда ребро $e$ и вершина $u$ (аналогично ребро $e$ и вершина $v$) называются *инцидентными*.

Два ребра, инцидентные одной вершине, называются *смежными*. Две вершины, инцидентные одному ребру, называются *смежными*. Множество вершин, смежных с вершиной $u$, называется *окружением* вершины $u$ (оно обозначается $N^+(u)$). *Множеством смежности* вершины $u$ называется множество $N(u) = N^+(u) \cup \{u\}$. Множеством смежности подмножества вершин $A$ ($A \subseteq V$) называется $N(A) = \bigcup_{u \in A} N(u)$. *Окружением* подмножества вершин $A$ ($A \subseteq V$) в графе называется $\Gamma(A) = N(A) \setminus A$.

Списком смежности графа назовем список $u - N^+(u)$, где для каждой вершины указано ее окружение.

*Степенью* вершины называется число рёбер, инцидентных вершине (другое название - валентность): $d(u) = |N^+(u)|$. Вершина степени 0 называ-

ется *изолированной*, а степени 1 – *висячей*. Ребро, инцидентное висячей вершине, называется *концевым*.

*Теорема Эйлера (лемма о рукопожатиях)*

Сумма степеней всех вершин графа равна удвоенному числу ребер.

*Следствие.* Число вершин нечетной степени в графе четно.

Граф называется *регулярным* степени $k$, если степень каждой вершины равна $k$.

*Замечание.* В случае псевдографа вклад каждой петли, инцидентной вершине $u$, в $d(u)$ равен 2. Тогда лемма о рукопожатиях верна не только для графов и мультиграфов, но и для псевдографов.

Для орграфа есть некоторые отличия. Если $e = (u, v)$ – дуга орграфа, то $u$ называется *началом дуги* $e$, $v$ *– концом дуги* $e$; дуга $e$ исходит из узла $u$, а заходит в узел $v$. *Полустепенью исхода* узла $u$ (обозначается $d^-(u)$) называется число дуг, исходящих из $u$. *Полустепенью захода* узла $v$ (обозначается $d^+(v)$) называется число дуг, заходящих в узел $v$. *Степенью* (валентностью) узла называется сумма его полустепеней:

$$d(u) = d^+(u) + d^-(u).$$

*Теорема Эйлера для орграфа*

$$\sum_{u\in V} d^+(u) = \sum_{u\in V} d^-(u) = q(G); \quad \sum_{u\in V} d(u) = 2q(G).$$

Орграф называется *регулярным*, если все полустепени всех узлов совпадают.

Если в орграфе полустепень захода некоторой вершины равна нулю (то есть $d^+(u) = 0$), то такая вершина называется *источником*, если же нулю равна полустепень исхода (то есть $d^-(u) = 0$ ), то такая вершина называется *стоком*. Направленный орграф с одним источником и одним стоком называется *сетью*. Есть более широкое толкование сети как связного орграфа.

***Упражнение 3.*** Доказать лемму о рукопожатиях для орграфа и для графа.

***Упражнение 4.*** Привести примеры регулярных графов степени 1, 2, 3. Привести пример регулярного орграфа.

***Упражнение 5.*** Можно ли построить граф (псевдограф) такой, что степени всех его вершин различны?

***Упражнение 6.*** Построить граф порядка 4, у которого множества смежности всех вершин совпадают.

## Изоморфизм

Два помеченных графа $G_1(V_1,E_1)$ и $G_2(V_2,E_2)$ называются изоморфными, если существует биективное отображение вершин $f$: $V_1 \to V_2$, сохраняющее смежность: $\{u,v\} \in E_1$ тогда и только тогда, когда $\{f(u),f(v)\} \in E_2$.

Два орграфа $D_1(V_1, E_1)$ и $D_2(V_2, E_2)$ называются изоморфными, если существует биективное отображение узлов $f : V_1 \to V_2$ такое, что $(u,v) \in E_1$ тогда и только тогда, когда $(f(u), f(v)) \in E_2$ (то есть индуцирующее биективное отображение дуг).

Аналогично определяется изоморфизм псевдографов. Класс изоморфных помеченных графов есть непомеченный граф, так как отношение изоморфизма графов есть отношение эквивалентности.

***Упражнение 7.*** Сколько существует изоморфных помеченных графов третьего порядка без изолированных вершин с одинаковым множеством $V$.

***Упражнение 8.*** Привести пример биективного отображения вершин графов, которое не индуцирует биективное отображение рёбер.

## Основные примеры графов. Маршруты, подграфы, связность

Граф называется *пустым*, если в нём нет рёбер ($E = \varnothing$). Пустой граф порядка $n$ обозначается $O_n$. Граф $O_1$ называется тривиальным.

Граф называется *полным* ($K_n$ – полный граф порядка $n$), если любые две вершины графа смежны.

Если в полном графе ориентировать все ребра (заменить каждое ребро на дугу, выбрав одно из возможных направлений), то получим орграф, который называется *турнир*. Он соответствует результатам спортивного соревнования по круговой системе без возможных ничьих.

*Маршрутом* $M$ в графе (псевдографе, мультиграфе) называется чередующаяся последовательность вершин и рёбер, в которой два соседних элемента инцидентны. Маршрут $M = u_0, e_1, u_1, e_2, u_2, e_3, \ldots, u_{n-1}, e_n, u_n$ начинается и оканчивается вершиной ($u_0$ и $u_n$ соответственно, остальные вершины внутренние).

*Длиной маршрута* $d(M)$ называется число рёбер ($d(M) = n$) маршрута.

***Замечание***. Аналогично определение *пути* (вместо термина «маршрут») от $u_0$ к $u_n$ в орграфе, если потребовать, что $e_i = (u_{i-1}, u_i) \in E$. Путь с началом в $u_0$ и концом в $u_n$ будем обозначать $M(u_0, u_n)$.

***Замечание.*** Для графа, псевдографа (и орграфа), в отличие от мультиграфа, в записи маршрута (пути) можно не указывать ребра (дуги).

Маршрут (путь в случае орграфа), все рёбра (дуги) которого различны, называется цепью. Маршрут (путь), все вершины (узлы) которого различны (а следовательно, и все рёбра (дуги)), называется простой цепью. Ана-

логично граф, все вершины и рёбра которого упомянуты в простой цепи, называется *простой цепью* и обозначается $P_n$, где $n$ – число вершин. Маршрут, у которого $u_0 = u_n$ (начало и конец маршрута совпадают), называется циклом.

В случае орграфа применяется термин «контур» для маршрута, который в ассоциированном простом графе является циклом, но не является циклом как путь в исходном орграфе. Для орграфа простым контуром называется контур, в котором все узлы различны.

Маршрут, у которого все вершины, кроме крайних ($u_0 = u_n$), различны, называется простым циклом. Аналогично определяется граф с названием *простой цикл* (обозначается $C_n$) как граф, все вершины которого упомянуты в данном маршруте.

Граф называется *двудольным* и обозначается $K\ \{n, m\}$, если множество вершин $V$ разбито на две доли ($V = V_1 \cup V_2$; $V_1 \cap V_2 = \varnothing$), такие что $|V_1| = n$, $|V_2| = m$, и любые две вершины разных долей смежные, а одной доли – не смежные.

*Теорема (критерий двудольности).* Граф является двудольным тогда и только тогда, когда все его простые циклы имеют четную длину.

*Подграфом* графа $G(V, E)$ называется граф $G_1(V_1, E_1)$ такой, что $V_1 \subseteq V$, $E_1 \subseteq E$. Подграф графа $G$, отличный от графа $G$, называется *собственным подграфом*.

*Лемма о простом пути*. Для любого пути $M$ $(u, v)$ от $u$ к $v$ в орграфе $D(V, E)$ найдется простая цепь $M_1$ $(u, v)$ от $u$ к $v$; причем $d(M_1) \leq d(M)$.

Граф называется *связным*, если для любых двух вершин графа найдётся маршрут, их соединяющий. Граф $G$ $(V, E)$ называется двусвязным, если он не связан, но существуют два непересекающихся связных подграфа $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ графа $G$ таких, что $V_1 \cup V_2 = V$, $E_1 \cup E_2 = E$, $V_1 \cap V_2 = \varnothing$. При этом $G_1$ и $G_2$ называются связными компонентами графа $G$.

***Упражнение 9.*** Найти число ребер для графов $O_n$, $P_n$, $C_n$, $K_n$, $K_{n,m}$.

***Упражнение 10.*** Построить графы $O_3$, $P_3$, $C_3$, $K_3$, $K_{2,1}$. Какие из этих графов изоморфны?

***Упражнение 11.*** Построить графы $P_4$, $C_4$, $K_4$, $K_{2,2}$, $K_{3,1}$. Какие из этих графов изоморфны? Какие из этих графов являются собственными подграфами графа $C_4$ (графа $K_4$; графа $C_5$)?

***Упражнение 12.*** Найти простой цикл (маршрут) в графе $K_{2,2}$ и вычислить его длину. Какому условию должен удовлетворять двудольный граф, имеющий маршрут в виде простого цикла? Какое утверждение можно сформулировать о длине этого маршрута?

***Упражнение 13.*** Привести пример графа и следующих маршрутов в нём: не цепь; цепь, но не простая цепь; простая цепь; цикл, но не простой цикл; простой цикл.

***Упражнение 14.*** Привести пример орграфа без петель, в котором можно указать: контур, но не простой контур; простой контур, но не простой цикл; простой цикл.

***Упражнение 15.*** Доказать лемму о простом пути в орграфе.

***Упражнение 16.*** Сформулировать и доказать лемму о простом маршруте в графе.

***Упражнение 17.*** Доказать критерий двудольности графа.

## Расстояния в графе

*Расстоянием* между вершинами $u$ и $v$ в графе $G$ (обозначается $d(u, v)$) называется длина кратчайшего маршрута между ними. Полагаем по определению $d(u,v) = +\infty$, если маршрута между $u$ и $v$ в графе нет. *Диаметром* графа (обозначается $d(G)$) называется расстояние между наиболее удаленными вершинами: $d(G) = \max_{u,v} d(u,v)$.

Множество вершин, находящихся на одинаковом расстоянии $n$ от вершины $u$ (обозначается $d_n(u)$), называется *ярусом*.

В дополнение к классификации графов, приведенной в разделе «Определение графа», отметим, что существуют *нагруженные графы* $G$ ($V$, $E$, $r$), где добавлена функция расстояния $r = r(e)$, заданная на множестве ребер $E$ и принимающая натуральные или действительные значения. Примерами нагруженного графа служат: атлас автомобильных дорог с указанием расстояний, схема перевозок груза с указанием стоимости каждой перевозки. Граф можно считать частным случаем нагруженного графа, где длина каждого ребра равна 1. Для нагруженного графа приведенные выше понятия расстояния между вершинами, диаметра графа и яруса вычисляются с учетом функции $r$ расстояния.

В орграфе учитываются только разрешенные направления. *Расстоянием* от узла $u$ до узла $v$ в орграфе $D$ (обозначается $d(u,v)$) называется длина кратчайшего пути от узла $u$ до узла $v$. Полагаем по определению $d(u,v) = +\infty$, если пути от $u$ к $v$ в орграфе нет. Диаметр орграфа в общем случае не принято рассматривать. *Диаметром* сети $D$ ($V$, $E$) (обозначается $d(D)$) называется кратчайший путь от источника до стока.

Аналогичные изменения с учетом функции расстояния производят в *нагруженном орграфе* и в *нагруженной сети.*

***Упражнение 18.*** Вычислить диаметры графов $O_n$, $P_n$, $C_n$, $K_n$, $K_{n,m}$ и первые ярусы каждой вершины графов $K_n$, $K_{n,m}$.

## Операции над графами

Сначала рассмотрим операции над помеченными графами.

*Объединением* помеченных графов $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ (обозначение $G_1(V_1, E_1) \cup G_2(V_2, E_2)$) называется граф $G(V, E)$, где $V = V_1 \cup V_2$, $E = E_1 \cup E_2$.

*Пересечением* помеченных графов $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ (обозначение $G_1(V_1,E_1) \cap G_2(V_2,E_2)$) называется граф $G(V, E)$, где $V = V_1 \cap V_2$, $E = E_1 \cap E_2$.

*Дополнением* графа $G = G(V, E)$ называется граф $\overline{G} = G(V, \overline{E})$, где $E \cap \overline{E} = \emptyset$, $G(V, E \cap \overline{E})$ – полный граф.

*Соединением* помеченных графов $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ (обозначение $G_1(V_1, E_1) + G_2(V_2, E_2)$, которое возможно только при условии $V_1 \cap V_2 = \emptyset$, $E_1 \cap E_2 = \emptyset$, называется граф $G(V, E)$, где

$V = V_1 \cup V_2$, $E = E_1 \cup E_2 \cup \{e = \{v_1, v_2\} \mid v_1 \in V_1, v_2 \in V_2\}$.

Вводится понятие *степени* графа. Для графа $G(V, E)$ назовем $k$-й степенью такой граф $G^k = G_1(V_1, E_1)$, где $V = V_1$, $e = (u, v) \in E_1$ для всех таких вершин $u, v \in V$, что $d(u, v) \le k$ в графе $G(V, E)$.

Операция *удаления вершины* $u$ из графа $G(V, E)$ (обозначение $G(V, E) - u$) дает граф $G_1(V_1, E_1)$ без этой вершины и всех смежных с ней ребер:

$V_1 = V \setminus \{u\}$, $E_1 = E \setminus \{e = \{u, v\}\}$.

Под операцией удаления вершины из графа (орграфа) будем понимать операцию, заключающуюся в одновременном удалении из множества $V$ данной вершины (узла) и из множества $E$ всех инцидентных ей ребер (дуг).

Операция *удаления ребра* $e$ из графа $G(V, E)$ (обозначение $G(V, E) - e$) дает граф без этого ребра с тем же множеством вершин.

Операция *добавления вершины* $u$ в граф $G(V, E)$ ( обозначение $G(V, E) + u$, при условии $u \notin V$) дает граф $G_1(V_1, E)$ с добавленной вершиной и тем же множеством ребер: $V_1 = V \cup \{u\}$.

Операция *добавления ребра* $e = \{u, v\}$ в граф $G(V, E)$ (обозначение $G(V, E) + e$, возможно при условии $u, v \in V$, $e \notin E$) дает граф с добавленным ребром с тем же множеством вершин.

Операция *стягивания подграфа* $G_1(V_1, E_1)$ графа $G(V, E)$ (обозначение $G(V, E) / G_1(V_1, E_1)$) дает граф $G_2(V_2, E_2)$, который получается заменой подграфа на новую вершину $u$, где

$V_2 = V \cup \{u\} \setminus V_1$,

$E_2 = E \cup \{e = \{u, v\} \mid v \in \Gamma(V_1)\} \setminus (E_1 \cup \{e = \{v, w\} \mid v \in \Gamma(V_1), w \in V_1\})$.

Операция *размножения вершины* $u$ в графе $G(V, E)$ (обозначение $G(V, E) \uparrow u$, при условии $u \in V$, $u' \notin V$) дает граф $G_1(V_1, E)$ с добавленной вершиной и большим множеством ребер:

$V_1 = V \cup \{u'\}$, $E_1 = E \cup \{e = \{u', v\} \mid v \in N(u)\}$.

## Матричное задание графов

*Матрицей смежности* помеченного графа $G = (V, E)$ называется квадратная матрица $A(G) = [a_{ij}]$ порядка $n$, где $V = \{u_1, u_2, \ldots, u_n\}$, у которой

$$a_{ij} = \begin{cases} 1, \text{если } \{u_i, u_j\} \in E, \\ 0, \text{если } \{u_i, u_j\} \notin E. \end{cases}$$

*Матрицей инцидентности* помеченного графа $G = (V, E)$ называется матрица $B(G) = [b_{ij}]$ размера $n \times m$, где $n$ - порядок графа $G$ ($|V| = n$), а $m$ – число ребер графа $G$ ($|E| = m$), у которой

$$b_{ij} = \begin{cases} 1, \text{если вершина } u_i \text{ инцидентна ребру } e_j, \\ 0, \text{если вершина } u_i \text{ неинцидентна ребру } e_j. \end{cases}$$

Здесь $E = \{e_1, e_2, \ldots, e_m\}$.

*Матрицей Кирхгофа* помеченного графа $G = (V, E)$ называется матрица $C(G) = [c_{ij}]$ порядка $n$ такая, что

$$c_{ij} = \begin{cases} -1, \text{если вершины } u_i, u_j \text{ - смежные}, \\ 0, \text{если вершины } \{u_i, u_j\} \notin E, i \neq j, \\ d(u_i), \text{если } i = j. \end{cases}$$

В определение матрицы смежности для псевдографа следует добавить, что $a_{ii} = 2$, если петля $(u_i, u_i)$ встречается в графе, а для мультиграфа $a_{ij}$ равно числу ребер, соединяющих вершины $u_i$, $u_j$.

Определение матрицы инцидентности для мультиграфа сохраняется, а для псевдографа дополнительно полагаем $b_{ij} = 2$, если $e_j = (u_i, u_i)$. Матрицы Кирхгофа для мультиграфа и псевдографа не определяются.

***Замечание***. Встречается трактовка, что $a_{ij} = 1$ и $b_{ij} = 1$, если $(u_i, u_i) = e_j \in E$ для псевдографа. Это позволяет задавать матрицы смежности и инцидентности и для псевдографов в рамках булевых матриц.

Приведем аналогичные определения для орграфа $D = (V, E)$, где $V = \{u_1, u_2, \ldots, u_n\}$ – множество узлов, $E = \{e_1, e_2, \ldots, e_m\}$ – множество дуг.

Для матрицы смежности $A(D) = [a_{ij}]$:

$$a_{ij} = \begin{cases} 1, \text{если } \{u_i, u_j\} \in E, \\ 0, \text{если } \{u_i, u_j\} \notin E. \end{cases}$$

Для матрицы инцидентности $B(D) = [b_{ij}]$:

$$b_{ij} = \begin{cases} 1, \text{если узел } u_i \text{ - конец дуги } e_j, \\ -1, \text{если узел } u_i \text{ - начало дуги } e_j, \\ 0, \text{если узел } u_i \text{ неинцидентен дуге } e_j. \end{cases}$$

Вместо матрицы Кирхгофа для орграфа можно ввести две аналогичные матрицы: матрицу исходов $C^+(D)$ и матрицу заходов $C^-(D)$.

Свойства матриц.

1. Матрицы $A(G)$, $C(G)$ для графов (псевдографов, мультиграфов) симметричны.

2. Сумма элементов $i$-й строки матрицы $A(G)$ графа (псевдографа, мультиграфа) $G$ равна $d(u_i)$ степени вершины $u_i$.

3. Сумма элементов $i$-й строки ($i$-го столбца) матрицы $A(D)$ орграфа $D$ равна $d^+(u_i)$ (соответственно $d^-(u_i)$).

4. Покоординатная сумма всех строк матрицы $B(G)$ для графа (псевдографа, мультиграфа) равна строке, все элементы которой 2. Отсюда вытекает, что любая строка матрицы $B(G)$ равна сумме отдельных строк по модулю 2.

Аналогичное утверждение справедливо для столбцов матрицы $B(G)$, которые соответствуют ребрам, составляющим простой цикл в графе $G$.

5. Покоординатная сумма всех строк матрицы $B(D)$ орграфа $D$ без петель есть нулевая строка.

6. Для любого простого контура орграфа без петель сумма столбцов матрицы $B(D)$, соответствующих дугам этого контура, равна нулевому столбцу.

## Матрицы связности. Слабая и сильная связность

Вершина (для орграфа узел) $u_j$ называется достижимой из вершины (из узла) $u_i$, если в графе (орграфе) существует маршрут (путь), в котором $u_i$ и $u_j$ – концевые вершины (который начинается в узле $u_i$ и оканчивается в узле $u_j$). *Матрицей достижимости* $T(D) = [t_{ij}]$ орграфа $D$ называется квадратная матрица порядка $n$, где $n$ – порядок графа, у которой

$$t_{ij} = \begin{cases} 1, \text{если } u_j \text{ достижимо из } u_i, \\ 0, \text{если } u_j \text{ недостижимо из } u_i. \end{cases}$$

Заметим, что матрица $T(D)$ в общем случае несимметрична. Орграф называется односторонне связным, если для любых двух узлов хотя бы один достигается из другого. Для любого орграфа (псевдографа) $D = (V, E)$ можно построить ассоциированный с ним мультиграф (псевдомультиграф) $G = (V, E_1)$, если каждую упорядоченную пару $(u, v)$ из множества дуг $E$ заменить на аналогичную неупорядоченную пару $\{u, v\}$, которые будут составлять множество ребер $E_1$. Орграф называется слабо связанным, если связан ассоциированный с ним мультиграф. Орграф называется сильно связанным, если для любых двух узлов $u_i$ и $u_j$ существуют оба маршрута: как из $u_i$ в $u_j$, так и из $u_j$ в $u_i$. *Матрицей сильной связности* $S(D) = [s_{ij}]$ орграфа $D$ называется симметричная (квадратная) матрица порядка $n$, у которой

$$s_{ij} = \begin{cases} 1, \text{если } u_i \text{ достижимо из } u_j \text{ и } u_j \text{ достижимо из } u_i, \\ 0 - \text{ в противном случае.} \end{cases}$$

Аналогично определяется матрица сильной связности $S(G)$ графа $G$. Пусть $\rho_1$ есть отношение достижимости на множестве узлов $V$ орграфа $D = (V, E)$: $u\ \rho_1\ v$ тогда и только тогда, когда узел $v$ достижим из $u$.

Отношение $\rho_1^{-1}$ есть отношение обратной достижимости: $u\ \rho_1^{-1}\ v$ тогда и только тогда, когда $u\ \rho_1\ v$.

Определим $\rho = \rho_1 \cap \rho_1^{-1}$ - отношение двусторонней достижимости.

***Утверждение***. Отношение $\rho_1$ рефлексивно и транзитивно на $V$. Отношение $\rho$ есть отношение эквивалентности на $V$.

Два узла $u$, $v$ принадлежат одной компоненте сильной связности $D = (V_1, E_1)$, если $u\ \rho\ v$, а также $u, v \in V_1$.

Отношение $\rho$ разбивает любой граф на классы эквивалентности, каковыми являются компоненты сильной связности: $D = D_1 \cup D_2 \cup \ldots \cup D_m$, где $D_i$ - компоненты сильной связности, то есть

$D = (V, E)$,

$D_i = (V_i, E_i)$,

$V = V_1 \cup V_2 \cup \ldots \cup V_m$,

$E = E_1 \cup E_2 \cup \ldots \cup E_m$,

$V_i \cap V_j = \varnothing$, $E_i \cap E_j = \varnothing$ при $i \neq j$.

В этом случае орграф $D$ называется $m$-сильно связанным.

Вершина графа, удаление которой увеличивает число компонент связности (для орграфа - компонент сильной связности), называется разделяющей или точкой сочленения.

# АЛГОРИТМЫ НА ГРАФАХ

## Обход графов в ширину и глубину

### Описание алгоритма

Пусть нам необходимо обойти граф $G$ ($V$, $E$), который представлен списком смежности $Г$. Обход графа подразумевает некоторое систематическое перечисление его вершин. Для этого используются следующие вспомогательные структуры данных:

***Структура данных T*** является своего рода вспомогательным буфером, в который временно помещаются обойденные вершины (это необходимо для обхода смежных с ними вершин). Данная структура может являться стеком (в случае поиска в глубину) или очередью (в случае поиска в ширину).

*Стек* – это структура данных, в которой первый помещенный в нее элемент извлекается последним. *Очередь* – это структура данных, в которой первый помещенный в нее элемент извлекается первым.

***Массив X***, длина которого равна числу вершин, содержит данные о том, была ли отмечена (пройдена) вершина. Каждый элемент массива соответствует одной вершине графа и может принимать два значения:

1 – вершина отмечена (пройдена);

0 – вершина не отмечена.

Рассмотрим, как осуществляется обход поэтапно:

1) Обнуление массива *X*. До начала обхода все вершины являются неотмеченными.
2) Выбирается некоторая произвольная вершина *v*, с которой будет начинаться обход.
3) Вершина *v* помещается в структуру *T* и отмечается в массиве *X* как пройденная (*X* [*v*] := 1).
4) Из структуры *T* извлекается вершина (обозначим её *u*). Эта вершина является пройденной. Таким образом, именно на этом этапе мы выделяем обойденные вершины.
5) По списку смежности *Г* поочередно выбираются все вершины смежные с *u* (обозначим *w* вершину, смежную с *u*), и если они не были раннее отмечены (то есть если *X* [*w*] = 0), то они помещаются в структуру *T* и отмечаются.

Если в структуре *T* находятся какие-либо вершины, то осуществляется переход к п. 4. Если же нет, то обход графа *G* (*V*, *E*) закончен.

## Пример

### *Обход графа в глубину:*

Рассмотрим на примере обход графа *G* (*V*, *E*) (рис. 1).

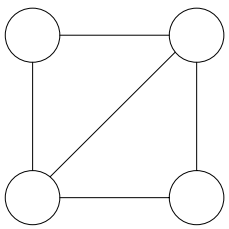


*Рис. 1*

**Список смежности ( *Г* )**

1 – 2, 4
2 – 1, 3, 4
3 – 2, 4
4 – 1, 2, 3

1) Обнулим массив $X$.

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |

2) Начнем обход с первой вершины ($v = 1$).
3) Поместим вершину 1 ($v = 1$) в структуру $T$ и отметим ее в массиве $X$ как пройденную.

$T$:

| 1 | | | |
|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 0 | 0 | 0 |

4) Извлекаем из структуры $T$ вершину 1 ($v = 1$), т.е. вершина 1 пройдена.

$T$:

| | | | |
|---|---|---|---|

5) Затем помещаем вершины, смежные с первой ($v = 1$) и еще не отмеченные в массиве $X$, в структуру $T$ и отмечаем их в массиве $X$.

$T$:

| 2 | 4 | | |
|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 1 | 0 | 1 |

6) Извлекаем из структуры $T$ вершину 4 ($v = 4$), т.е. вершина 4 пройдена.

$T$:

| 2 | | | |
|---|---|---|---|

7) Затем помещаем вершины, смежные с четвертой ($v = 4$) и еще не отмеченные в массиве $X$ ($v = 2$ и $v = 3$), в структуру $T$ и отмечаем их в массиве $X$.

$T$:

| 2 | 3 | | |
|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |

8) Извлекаем из структуры $T$ вершину 3 ($v = 3$), т.е. вершина 3 пройдена.

$T$:

| 2 | | | |
|---|---|---|---|

9) Все вершины, смежные с третьей, уже отмечены в массиве $X$, поэтому ничего не помещаем в структуру $T$.
10) Извлекаем из структуры $T$ вершину 2 ($v = 2$), т.е. вершина 2 пройдена.

$T$:

| | | | |
|---|---|---|---|

11) Все вершины, смежные со второй, уже отмечены в массиве $X$, поэтому ничего не помещаем в структуру $T$.
12) Структура $T$ пуста. Обход вершин графа закончен: 1 – 4 – 3 – 2.

*Обход графа в ширину:*

Рассмотрим на примере обход графа $G(V, E)$ (рис. 2).

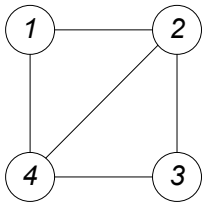



*Рис. 2*

**Список смежности ( $\Gamma$ )**

1 – 2, 4
2 – 1, 3, 4
3 – 2, 4
4 – 1, 2, 3

1) Обнулим массив $X$.

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |

2) Начнем обход с первой вершины ($v = 1$).
3) Поместим вершину 1 ($v = 1$) в структуру $T$ и отметим ее в массиве $X$ как пройденную.

$T$:

| 1 | | | |
|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 0 | 0 | 0 |

4) Извлекаем из структуры $T$ вершину 1 ($v = 1$), т.е. вершина 1 пройдена.

$T$:

| | | | |
|---|---|---|---|

5) Затем помещаем вершины, смежные с первой ($v = 1$) и еще не отмеченные в массиве $X$, в структуру $T$ и отмечаем их в массиве $X$.

$T$:

| 2 | 4 | | |
|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 1 | 0 | 1 |

6) Извлекаем из структуры $T$ вершину 2 ($v = 2$), т.е. вершина 2 пройдена.

$T$:

| 4 | | | |
|---|---|---|---|

7) Затем помещаем вершины, смежные со второй ($v = 2$) и еще не отмеченные в массиве $X$ ($v = 3$), в структуру $T$ и отмечаем их в массиве $X$.

$T$:

| 4 | 3 | | |
|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |

8) Извлекаем из структуры $T$ вершину 4 ($v = 4$), т.е. вершина 4 пройдена.

$T$:

| 3 | | | |
|---|---|---|---|

9) Все вершины, смежные с четвертой, уже отмечены в массиве $X$, поэтому ничего не помещаем в структуру $T$.
10) Извлекаем из структуры $T$ вершину 3 ($v = 3$), т.е. вершина 3 пройдена.

$T$:

| | | | |
|---|---|---|---|

11) Все вершины, смежные с третьей, уже отмечены в массиве $X$, поэтому ничего не помещаем в структуру $T$.
12) Структура $T$ пуста. Обход вершин графа закончен: 1 – 2 – 4 – 3.

На практике обход в глубину можно использовать, в частности, для получения ориентированного дерева из неориентированного. Рассмотрим это на примере.

### *Пример получения ориентированного дерева*

Исходное дерево представлено на рис. 3. Наша цель – посредством обхода исходного дерева в глубину получить ориентированное дерево с корнем $v$ и списком смежности $\Gamma'$.

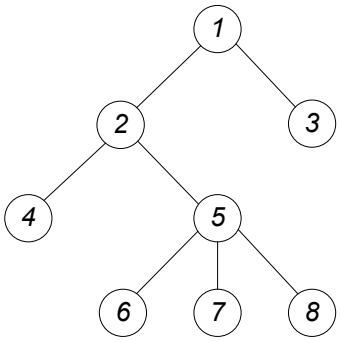



*Рис. 3*

**Список смежности ( $\Gamma$ )**

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

1) Обнулим массив $X$.

$X$:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

2) Пусть корнем будет вершина 5 ($v = 5$). Поместим вершину 5 в структуру $T$ и пометим ее как пройденную в массиве $X$.

$T$:

| 5 | | | | | | | |
|---|---|---|---|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

3) Затем извлекаем из структуры $T$ вершину 5 ($v = 5$) и помещаем смежные с ней вершины (ранее не пройденные) в $T$, отметим их в массиве $X$.

$T$:

| 2 | 6 | 7 | 8 | | | | |
|---|---|---|---|---|---|---|---|

$X$:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

4) Начинаем формировать список смежности ориентированного дерева ($\Gamma'$). Для этого берем вершину $v$ и выписываем из списка смежности исходного графа ($\Gamma$) вершины, совпадающие с вершинами, находящимися в структуре $T$ (это будут вершины, смежные с $v$), то есть

$\Gamma'$: 5 – 2, 6, 7, 8;

5) Затем извлекаем из структуры $T$ вершину 8 ($v = 8$). Дополняем список смежности $\Gamma'$ восьмой вершиной. У нее нет смежных вершин.

*T:*

| 2 | 6 | 7 | | | | | |
|---|---|---|---|---|---|---|---|

*Г'*: 5 – 2, 6, 7, 8; 8 – ;

6) Затем извлекаем из структуры *T* вершину 7 ($v$ = 7). Дополняем список смежности *Г'*.

*T:*

| 2 | 6 | | | | | | |
|---|---|---|---|---|---|---|---|

*Г'*: 5 – 2, 6, 7, 8; 8 – ; 7 – ;

7) Затем извлекаем из структуры *T* вершину 6 ($v$ = 6). Дополняем список смежности *Г'*.

*T:*

| 2 | | | | | | | |
|---|---|---|---|---|---|---|---|

*Г'*: 5 – 2, 6, 7, 8; 8 – ; 7 – ; 6 – ;

8) Затем извлекаем из структуры *T* вершину 2 ($v$ = 2). В структуру *T* вносим номера вершин, смежных с вершиной 2 (ранее не пройденных). Дополняем список смежности *Г'*.

*T:*

| 1 | 4 | | | | | | |
|---|---|---|---|---|---|---|---|

*X:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

*Г'*: 5 – 2, 6, 7, 8; 8 – ; 7 – ; 6 – ; 2 – 1, 4;

9) Извлекаем из структуры *T* вершину 4 ($v$ = 4). Дополняем список смежности *Г'*.

*T:*

| 1 | | | | | | | |
|---|---|---|---|---|---|---|---|

*Г'*: 5 – 2, 6, 7, 8; 8 – ; 7 – ; 6 – ; 2 – 1, 4; 4 – ;

10) Извлекаем из структуры *T* вершину 1 ($v$ = 1). Дополняем список смежности *Г'*. Помещаем смежные с ней вершины (ранее не пройденные) в *T*.

*T:*

| 3 | | | | | | | |
|---|---|---|---|---|---|---|---|

*X:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*Г'*: 5 – 2, 6, 7, 8; 8 – ; 7 – ; 6 – ; 2 – 1, 4; 4 – ; 1 – 3;

11) Извлекаем из структуры *T* вершину 3 ($v$ = 3). Дополняем список смежности *Г'*.

*T:*

| | | | | | | | |
|---|---|---|---|---|---|---|---|

*Г'*: 5 – 2, 6, 7, 8; 8 – ; 7 – ; 6 – ; 2 – 1, 4; 4 – ; 1 – 3; 3 – ;

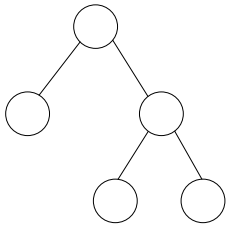
12) Структура *T* пуста. Обход графа закончен. Получили следующий список смежности ориентированного дерева (*Г'*):

1 – 3; 2 – 1, 4; 3 – ; 4 – ; 5 – 2, 6, 7,8; 6 – ; 7 – ; 8 – .

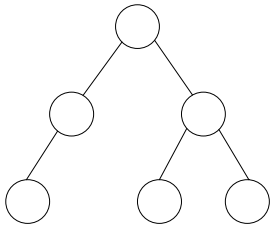
**Задания**

*Задание 1.* Построить из исходного неориентированного дерева ориентированное методом обхода в глубину из первой вершины.

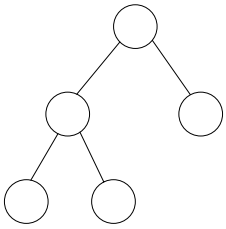
1.



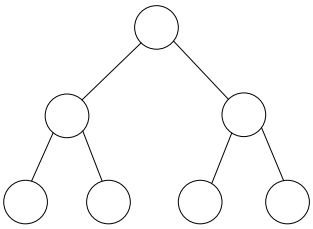
3.



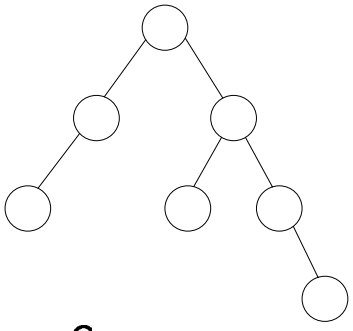
5.



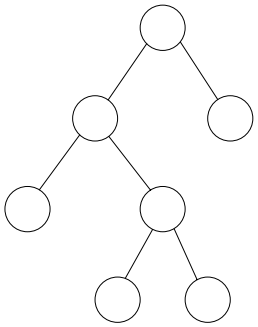
2.




4.

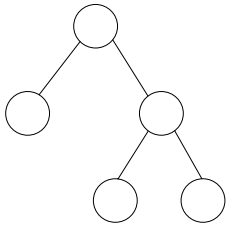



6.

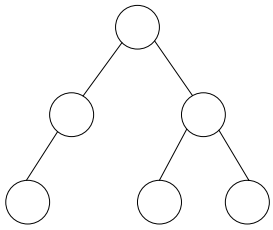



*Задание 2.* Построить из исходного неориентированного дерева ориентированное методом обхода в глубину из третьей вершины.

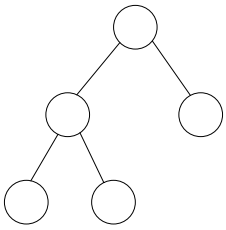
7.



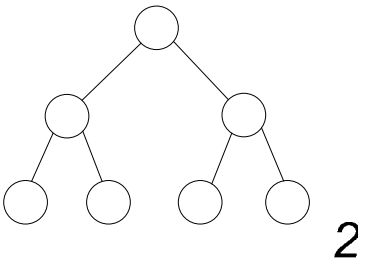
9.



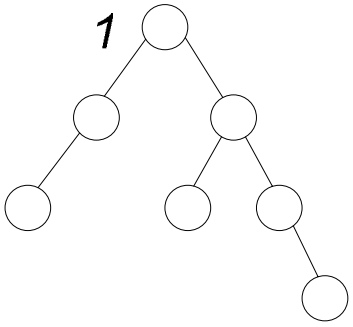

11.



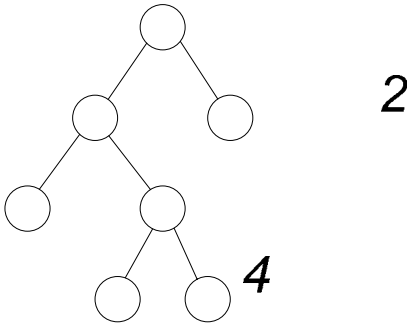

8.




10.

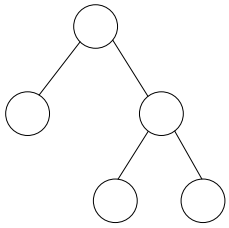



12.

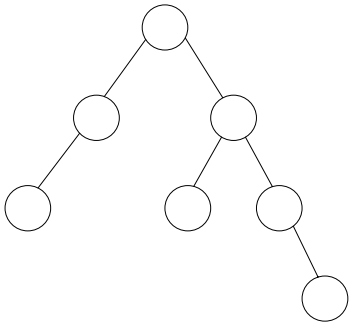


*Задание 3.* Построить из исходного неориентированного дерева ориентированное методом обхода в глубину из второй вершины.

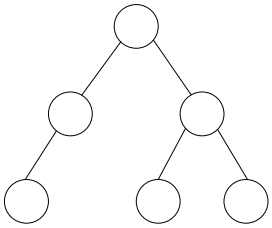
13.



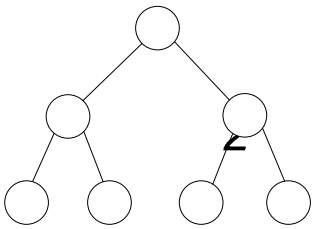
15.



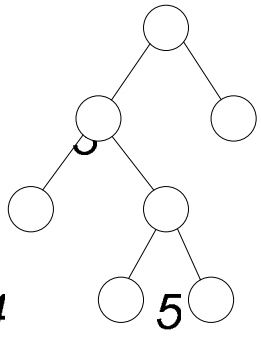
17.



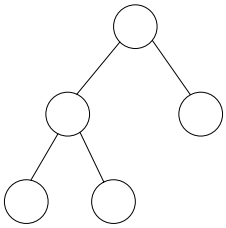
14.



16.

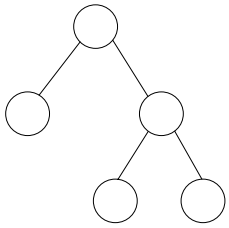


18.

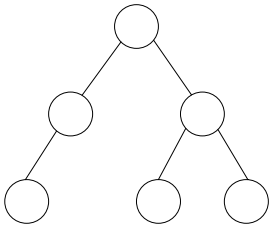


*Задание 4.* Построить из исходного неориентированного дерева ориентированное методом обхода в глубину из четвертой вершины.

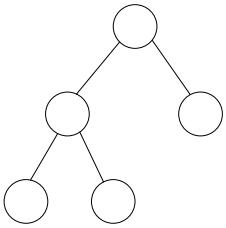
19.



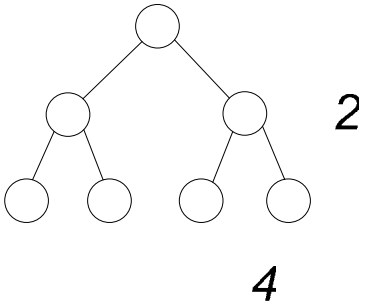
21.



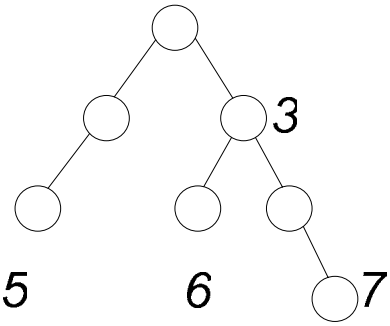
23.



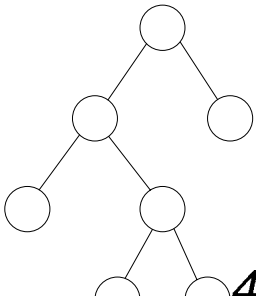
20.



22.

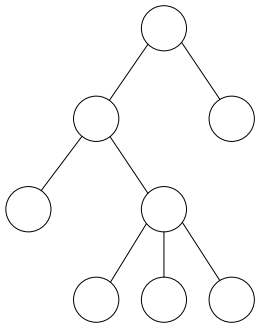


24.

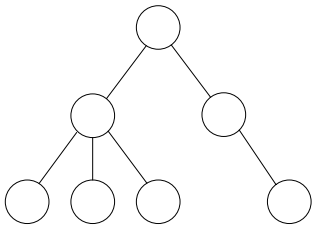


*Задание 5.* Построить из исходного неориентированного дерева ориентированное методом обхода в глубину из первой вершины.

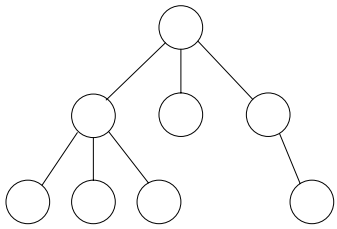
25.



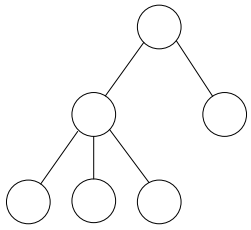
27.



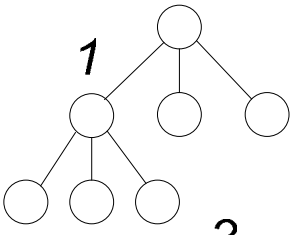
29.



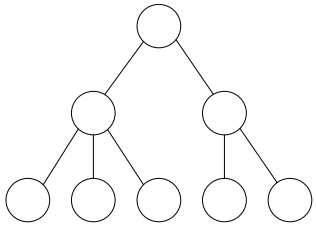
26.



28.



30.



## *Алгоритм Тэрри*

### Описание алгоритма

Алгоритм Тэрри – алгоритм поиска маршрута в связном графе $G$ ($V$,$E$), соединяющего заданные вершины $v$ и $w$. Исходя из вершины $v$ и осуществляя последовательный переход от каждой достигнутой вершины к смежной ей вершине, всегда можно найти маршрут в связном графе $G$ ($V$,$E$), соединяющий заданные вершины $v$ и $w$. Переход от вершины к вершине согласно алгоритму осуществляется по следующим правилам:

1) при проходе ребра необходимо всякий раз отмечать направление, в котором оно было пройдено;
2) исходя из некоторой вершины $v'$, нужно всегда следовать по тому ребру, которое не было пройдено или было пройдено в противоположном направлении;
3) для всякой вершины $v'$, отличной от $v$, необходимо отмечать пометкой «х» первое заходящее ребро, если вершина $v'$ встречается первый раз;
4) исходя из вершины $v'$, отличной от $v$, по первому заходящему в $v'$ ребру нужно идти лишь тогда, когда нет других возможностей.

**Пример**

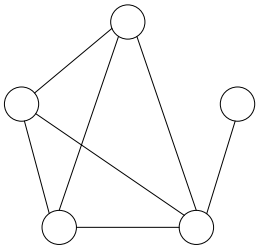


*Рис. 4*

Рассмотрим алгоритм Тэрри на примере поиска маршрута между вершинами 1 и 4 графа $G$ ($V$, $E$) (рис. 4). Если это не противоречит правилам алгоритма, то переходить от одной вершины будем к смежной ей с меньшим номером.

1) Переходим из вершины 1 в вершину 2, отмечаем направление прохода ребра (рис. 5).
2) Из вершины 2 можем перейти к вершинам 3 и 5, а также к вершине 1, но лишь в том случае, если нет других вариантов. Переходим к вершине 3. Отмечаем направление перехода (рис. 6).

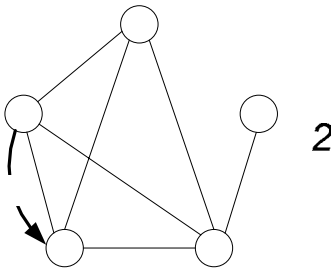


*Рис. 5*

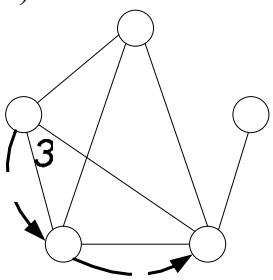


*Рис. 6*

3) Из третьей вершины переходим в первую, так как она имеет наименьший номер среди всех возможных вариантов перехода (рис. 7).

4) Первая вершина смежная с вершинами 2, 3 и 5. В вершину 2 мы не можем осуществить переход, так как это направление уже отмечено. В вершину 3 осуществлять переход не будем, так как ребро 1-3 уже отмечено как заходящее в вершину 1 и существует альтернатива перехода к вершине 5. Следовательно, переходим к вершине 5. Отмечаем направление перехода (рис. 8).

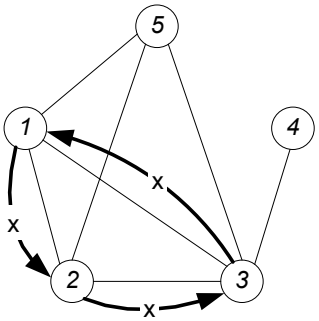



*Рис. 7*

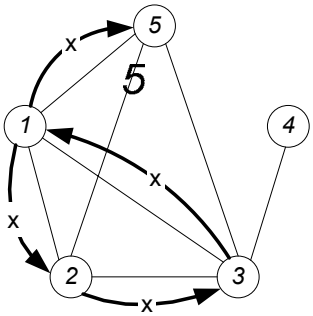



*Рис. 8*

5) Из вершины 5 переходим к вершине 2, так как она имеет наименьший номер среди всех возможных вариантов перехода (рис. 9).

6) Из вершины 2 переходим к вершине 1, так как ребро, связывающее эти вершины, отмечено как заходящее в вершину 2, а неотмеченных ребер при рассматриваемой вершине нет. Отмечаем направление перехода (рис. 10).

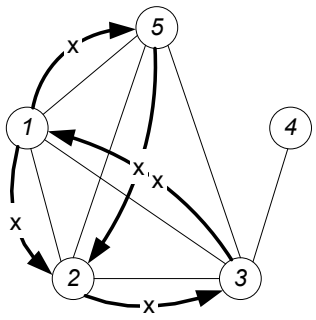



*Рис. 9*

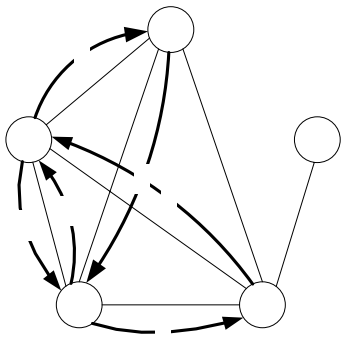


*Рис. 10*

7) Из вершины 1 можно перейти только к вершине 3 (рис. 11).

8) Из вершины 3 осуществляем переход к вершине 4, так как она имеет наименьший номер среди всех альтернативных вариантов (рис. 12). Поиск маршрута окончен.

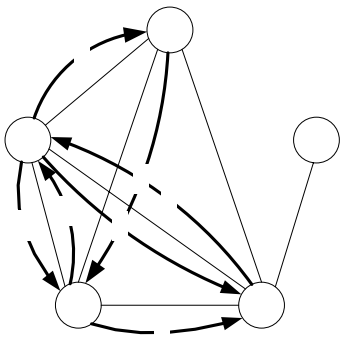


*Рис. 11*

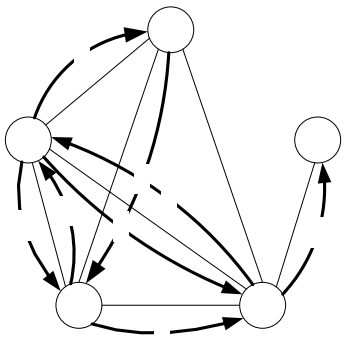


*Рис. 12*

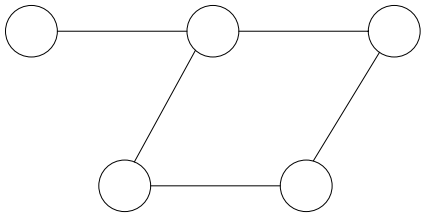
Маршрут из вершины 1 в вершину 4 в соответствии с алгоритмом Тэрри имеет вид:

**1 – 2 – 3 – 1 – 5 – 2 – 1 – 3 – 4**

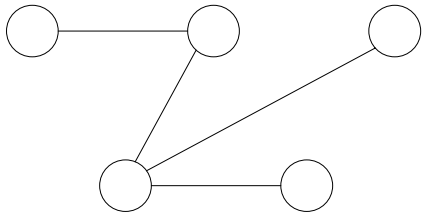
## Задания

Определить путь обхода графов.

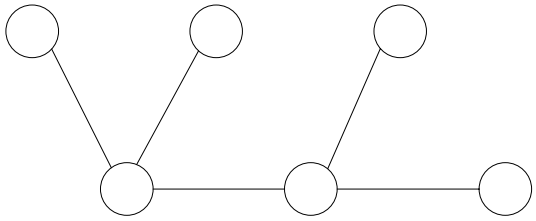
**1.** Обход в ширину из точки 3.
**2.** Обход в ширину из точки 1.
**3.** Обход в ширину из точки 5.
**4.** Обход в глубину из точки 1.
**5.** Обход в глубину из точки 2.



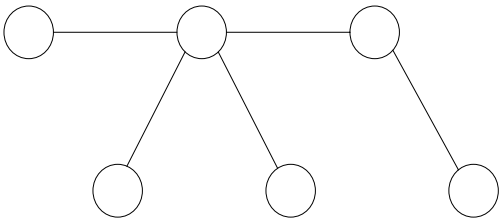
**6.** Обход в ширину из точки 2.
**7.** Обход в ширину из точки 4.
**8.** Обход в глубину из точки 1.
**9.** Обход в глубину из точки 3.
**10.** Обход в глубину из точки 5.



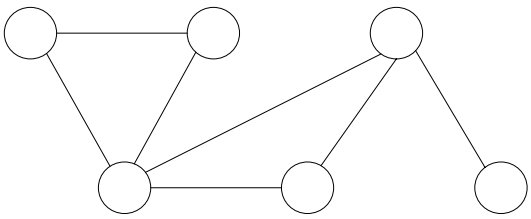
**11.** Обход в ширину из точки 1.
**12.** Обход в ширину из точки 4.
**13.** Обход в ширину из точки 5.
**14.** Обход в глубину из точки 2.
**15.** Обход в глубину из точки 6.



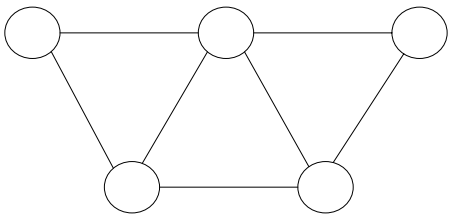
**16.** Обход в ширину из точки 2.
**17.** Обход в ширину из точки 6.
**18.** Обход в глубину из точки 3.
**19.** Обход в глубину из точки 5.
**20.** Обход в глубину из точки 1.



**21.** Обход в ширину из точки 1.
**22.** Обход в ширину из точки 3.
**23.** Обход в ширину из точки 4.
**24.** Обход в глубину из точки 6.
**25.** Обход в глубину из точки 2.



**26.** Обход в ширину из точки 2.
**27.** Обход в ширину из точки 5.
**28.** Обход в глубину из точки 1.
**29.** Обход в глубину из точки 3.
**30.** Обход в глубину из точки 4.



## *Матроиды. Жадные алгоритмы. Алгоритм Краскала*

### Описание алгоритма

*Матроидом* $M = < E, \varepsilon >$ называется конечное множество $E$, число элементов которого равно $n$ ($|E| = n$), и семейство его подмножеств $\varepsilon$, которое содержится в множестве всех подмножеств множества $E$ ($\varepsilon \subset 2^E$), такое, что выполняются следующие три аксиомы:

$M_1$: пустое множество принадлежит подмножеству $\varepsilon$ ( $\varnothing \in \varepsilon$ );

$M_2$: если множество $A$ принадлежит подмножеству $\varepsilon$, то и множество $B$, которое содержится в множестве $A$, тоже принадлежит подмножеству $\varepsilon$.

($A \in \varepsilon \ \& \ B \subset A \ \Rightarrow B \in \varepsilon$);

$M_3$: если множества $A$ и $B$ принадлежат подмножеству $\varepsilon$ и количество элементов множества $A$ на один элемент меньше, чем в множестве $B$, то существует элемент, принадлежащий разности множеств $B$ и $A$, такой, что объединение его с множеством $A$ будет принадлежать подмножеству $\varepsilon$.

($A,B \in \varepsilon \ \& \ |B| = |A| + 1 \ \Rightarrow \ \exists \ e \in B \backslash A, \ A \cup \{e\} \in \varepsilon$ ).

Элементы множества $\varepsilon$ называются независимыми, а остальные подмножества $E$ ($2^E \setminus \varepsilon$) – зависимыми. Пусть множество $X$ (произвольное множество) содержится в множестве $E$. Максимальным независимым подмножеством множества $X$ называется множество $Y$ такое, что если множество $Y$, принадлежащее независимому множеству $\varepsilon$, содержится в множестве $X$ и $Z$, принадлежащее независимому множеству $\varepsilon$, содержится в множестве $X$, то множество $Z$ содержится в множестве $Y$ ($Y \subset X \ \& \ Y \in \varepsilon \ \& \ Z \subset X \Rightarrow Z \subset Y$) . Множество максимальных независимых подмножеств множества $X$ обозначим $\underline{X}$.

Рассмотрим следующее утверждение, справедливое для любого $X$:

$M_4$: максимальные независимые подмножества данного множества равномощны ( $Y \in \underline{X} \ \& \ Z \in \underline{X} \Rightarrow |Y| = |Z|$ ).

Пусть $M = < E, \varepsilon >$ и выполнены аксиомы $M_1$ и $M_2$ . Тогда аксиома $M_3$ и утверждение $M_4$ эквивалентны, то есть $M_1$, $M_2$, $M_4$ - эквивалентная система аксиом матроида.

Пусть имеются конечное множество $E$, весовая функция $\omega$ и семейство $\varepsilon \subset 2^E$. Необходимо выбрать в указанном семействе $\varepsilon$ подмножество $X$ наибольшего веса. Для решения этой задачи будем считать, что множество $E$ упорядочено в порядке убывания весов элементов. Вначале множество $X$ пусто. Затем проверяем каждый элемент множества $E$, начиная с первого: если объединение его с множеством $X$ принадлежит указанному семейству $\varepsilon$, то добавляем его в множество $X$, если не принадлежит семейству $\varepsilon$, то рассматриваем следующий элемент, и так до $n$-го.

Алгоритм такого типа называется *жадным*. Очевидно, что жадный алгоритм является очень эффективным, он за наименьшее количество шагов находит искомое подмножество. Жадные алгоритмы и их свойства исследованы сравнительно недавно, но их значение в практике программирования чрезвычайно велико. Если удается свести конкретную экстремальную задачу к такой постановке, где множество допустимых вариантов (из которых необходимо выбрать наилучший) является матроидом, то в большинстве случаев следует сразу применять жадный алгоритм, поскольку он достаточно эффективен в практическом смысле. Если же наоборот, оказывается, что множество допустимых вариантов не образует матроида, то это «плохой признак». Скорее всего, данная задача окажется труднорешаемой. В этом случае целесообразно тщательно исследовать задачу для предварительного получения теоретических оценок сложности, чтобы избежать бесплодных попыток «изобрести» эффективный алгоритм там, где это на самом деле невозможно.

Другими словами, если $M = < E, \varepsilon >$ - матроид, то для любой весовой функции $\omega$ жадный алгоритм находит независимое множество $X$ с наибольшим весом; если же $M = < E, \varepsilon >$ не является матроидом, то существует такая функция $\omega$, что множество $X$, найденное жадным алгоритмом, не будет максимальным.

Пусть $G$ ($V,E$) – граф. Остовной подграф $G$ ($V,E$) – это подграф, содержащий все вершины. Остовной подграф, являющийся деревом, называется *остовом*. Несвязный граф не имеет остова, связный граф может иметь много остовов. Если задать длины рёбер, то можно поставить задачу нахождения кратчайшего остова. Существует множество различных способов найти какой-то остов графа. Множество кратчайших путей из заданной вершины ко всем остальным тоже образует остов. Однако этот остов может не быть кратчайшим. На рис. 13 показаны диаграмма графа, дерево кратчайших путей из вершины один с суммарным весом 5 и два кратчайших остова этого графа.

Одним из способов нахождения кратчайшего остова в связном графе является алгоритм Краскала. Он является частным случаем жадного алгоритма. Заметим, что множество подмножеств множества рёбер, не содержащих циклов, образует матроид.

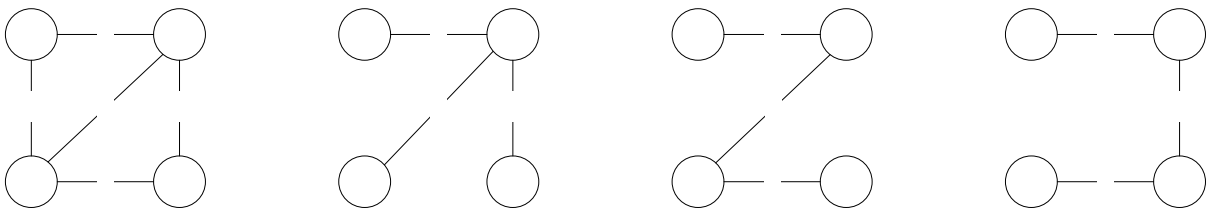


*Рис. 13*

### *Алгоритм Краскала*

Алгоритм Краскала основан на выборе ребер графа, составляющих его наикратчайший остов. Изначально известен список ребер графа с их длинами. Выбор происходит по двум критериям:

1) так как остов должен быть кратчайшим, выбор начинается с ребер минимальной длины;

2) так как остов – это дерево (структура, не содержащая циклов), каждое выбранное ребро проверяется: не составляет ли оно цикла с ранее выбранными ребрами.

Цикл проходит $n = v - 1$ раз, где $v$ – количество вершин графа.

Таким образом, при совершении нужного количества вышеуказанных действий мы будем иметь множество ребер, составляющих наикратчайший остов графа. Рассмотрим этот алгоритм на примере.

**Пример**

На рис. 14 задан граф. Необходимо найти его минимальный остов при помощи алгоритма Краскала.

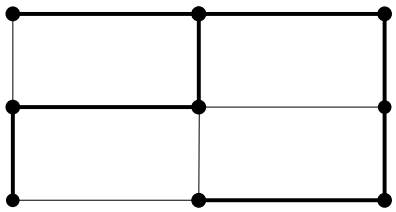


*Рис. 14*

*Решение*:

Представим список ребер исходного графа с указанием длины каждого из них. Затем отсортируем ребра в порядке возрастания их длин.

| **Исходный список** | **Отсортированный список** |
|---|---|
| (1,2) – 2 | (2,3) - 1 |
| (1,4) – 3 | (4,5) - 1 |
| (2,3) – 1 | (8,9) - 1 |
| (2,5) – 2 | (1,2) - 2 |
| (3,6) – 3 | (2,5) - 2 |
| (4,5) – 1 | (6,9) - 2 |
| (4,7) – 4 | (1,4) - 3 |
| (5,6) – 4 | (3,6) - 3 |
| (5,8) – 3 | (5,8) - 3 |
| (6,9) – 2 | (4,7) - 4 |
| (7,8) – 4 | (5,6) - 4 |
| (8,9) – 1 | (7,8) - 4 |

Для решения задачи с помощью алгоритма Краскала составим таблицу, отражающую шаги выполнения алгоритма для рассматриваемого примера.

| **Шаг** | **Массив ребер** | **Компонента связности** |
|---|---|---|
| | | |

Во втором столбце формируем массив ребер по принципу:

Шаг 1. Помещаем в первый столбец первое ребро отсортированного списка. Компонента связности данного ребра будет соответствовать вершинам, которые данное ребро связывает.

Шаг 2…Шаг *N*. Рассматриваем *n*-е ребро отсортированного списка:

1) добавляем ребро в массив, если оно удовлетворяет условию ацикличности;
2) пропускаем ребро в противном случае.

И так далее до конца списка ребер.

Проверка ацикличности формируемого массива производится при помощи третьего столбца таблицы, в который помещаются компоненты связности «леса», разрастающегося во втором столбце. Компоненты связности формируются следующим образом:

1) Если вершины добавляемого ребра не входят ни в одну из уже существующих компонент связности, то формируется новая компонента связности из двух вершин.

| Шаг | Массив ребер | Компонента связности |
|---|---|---|
| 1 | (2,3) | |
| 2 | (2,3); (4,5) | |
| 3 | (2,3); (4,5); (8,9) | |

2) Если одна из вершин добавляемого ребра входит в одну из имеющихся компонент связности, то ребро заносится в массив, соответствующий компоненте связности, а вторая вершина добавляемого ребра присоединяется к имеющейся компоненте.

| Шаг | Массив ребер | Компонента связности |
|---|---|---|
| 4 | (2,3); (4,5); (8,9); (1,2) | |

3) Если одна вершина рассматриваемого ребра входит в одну из имеющихся компонент связности, а вторая – в другую, то компоненты связности и массивы ребер объединяются.

| Шаг | Массив ребер | Компонента связности |
|---|---|---|
| 5 | (2,3); (4,5); (8,9); (1,2); (2,5) | |

4) Если обе вершины рассматриваемого ребра принадлежат одной компоненте связности, то ребро исключается из рассмотрения, в таблицу изменения не вносятся. В рассматриваемом примере такими ребрами являются (1,4), (5,8), (5,6) и (7,8).

Продолжим решение задачи при помощи алгоритма Краскала, учитывая перечисленные условия.

| Шаг | Массив ребер | Компонента связности |
|---|---|---|
| 6 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9) | 2, 3, 1, 4, 5 |
| 7 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9) [(1,4)] | 2, 3, 1, 4, 5 |

| Шаг | Массив ребер | Компонента связности |
|---|---|---|
| 8 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9); (3,6) | |
| 9 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9); (3,6) [(5,8)] | |
| 10 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9); (3,6); (4,7) | |
| 11 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9); (3,6); (4,7) [(5,6)] | |
| 12 | (2,3); (4,5); (8,9); (1,2); (2,5); (6,9); (3,6); (4,7) [(7,8)] | |

*Примечание*: в квадратных скобках указано рассматриваемое на данном шаге ребро, но не включаемое в массив ребер в силу выполнения условия 4.

В результате рассмотрения всех ребер отсортированного списка получим строку таблицы, определяющую минимальный остов графа (шаг 12).

Таким образом, минимальным остовом исходного графа является остов, состоящий из ребер (2,3); (1,2); (2,5); (4,5); (8,9); (6,9); (3,6); (4,7).

### Задания

Найти минимальный остов графа, заданного списком ребер, при помощи алгоритма Краскала.

**1.** (1,2) – 3; (1,3) – 6; (1,4) – 2; (1,5) – 5; (2,3) – 4; (2,4) – 3; (2,5) – 1; (3,4) – 2; (3,5) – 5; (4,5) – 2.

**2.** (1,2) – 6; (1,6) – 5; (1,8) – 5; (2,3) – 4; (3,4) – 1; (3,8) – 3; (4,5) – 1; (4,7) – 2; (5,6) – 2; (6,7) – 4; (7,8) – 1.

**3.** (1,2) – 5; (1,3) – 3; (1,6) – 4; (2,3) – 6; (2,6) – 1; (3,4) – 2; (3,5) – 5; (4,5) – 1; (4,6) – 1; (5,6) – 4.

**4.** (1,2) – 1; (1,3) – 7; (1,6) – 5; (2,3) – 4; (2,6) – 2; (3,4) – 3; (3,5) – 6; (3,6) – 1; (4,5) – 2; (4,6) – 5; (5,6) – 3.

**5.** (1,2) – 1; (1,8) – 2; (2,3) – 2; (2,4) – 7; (2,8) – 3; (3,4) – 5; (4,5) – 5; (4,6) – 6; (5,6) – 3; (6,7) – 4; (6,8) – 1; (7,8) – 1.

**6.** (1,2) – 4; (1,3) – 5; (1,4) – 2; (1,5) – 3; (2,3) – 1; (2,4) – 2; (2,5) – 3; (3,4) – 5; (3,5) – 6; (4,5) – 7.

**7.** (1,2) – 4; (1,6) – 1; (1,8) – 6; (2,3) – 5; (3,4) – 2; (3,8) – 7; (4,5) – 2; (4,7) – 3; (5,6) – 3; (6,7) – 5; (7,8) – 5.

**8.** (1,2) – 1; (1,3) – 7; (1,6) – 5; (2,3) – 4; (2,6) – 2; (3,4) – 3; (3,5) – 6; (4,5) – 2; (4,6) – 5; (5,6) – 3.

**9.** (1,2) – 5; (1,3) – 3; (1,6) – 4; (2,3) – 6; (2,6) – 1; (3,4) – 2; (3,5) – 5; (3,6) – 2; (4,5) – 1; (4,6) – 1; (5,6) – 4.

**10.** (1,2) – 5; (1,8) – 1; (2,3) – 1; (2,4) – 3; (2,8) – 2; (3,4) – 1; (4,5) – 4; (4,6) – 5; (5,6) – 4; (6,7) – 6; (6,8) – 2; (7,8) – 2.

**11.** (1,2) – 5; (1,3) – 3; (1,4) – 7; (1,5) – 2; (2,3) – 2; (2,4) – 1; (2,5) – 2; (3,4) – 4; (3,5) – 5; (4,5) – 2.

**12.** (1,2) – 5; (1,6) – 2; (1,8) – 5; (2,3) – 3; (3,4) – 1; (3,8) – 2; (4,5) – 7; (4,7) – 2; (5,6) – 2; (6,7) – 4; (7,8) – 1.

**13.** (1,2) – 2; (1,3) – 2; (1,6) – 4; (2,3) – 5; (2,6) – 1; (3,4) – 2; (3,5) – 5; (4,5) – 7; (4,6) – 1; (5,6) – 3.

**14.** (1,2) – 3; (1,3) – 1; (1,6) – 4; (2,3) – 7; (2,6) – 2; (3,4) – 5; (3,5) – 4; (3,6) – 1; (4,5) – 3; (4,6) – 2; (5,6) – 1.

**15.** (1,2) – 3; (1,8) – 2; (2,3) – 3; (2,4) – 1; (2,8) – 5; (3,4) – 2; (4,5) – 4; (4,6) – 4; (5,6) – 1; (6,7) – 7; (6,8) – 1; (7,8) – 5.

**16.** (1,2) – 2; (1,3) – 4; (1,4) – 1; (1,5) – 1; (2,3) – 3; (2,4) – 5; (2,5) – 2; (3,4) – 3; (3,5) – 4; (4,5) – 6.

**17.** (1,2) – 2; (1,6) – 3; (1,8) – 4; (2,3) – 4; (3,4) – 5; (3,8) – 6; (4,5) – 1; (4,7) – 2; (5,6) – 1; (6,7) – 3; (7,8) – 1.

**18.** (1,2) – 3; (1,3) – 6; (1,6) – 3; (2,3) – 2; (2,6) – 5; (3,4) – 1; (3,5) – 4; (4,5) – 1; (4,6) – 1; (5,6) – 2.

**19.** (1,2) – 2; (1,3) – 2; (1,6) – 4; (2,3) – 5; (2,6) – 1; (3,4) – 2; (3,5) – 5; (3,6) – 3; (4,5) – 7; (4,6) – 1; (5,6) – 3.

**20.** (1,2) – 2; (1,8) – 1; (2,3) – 7; (2,4) – 2; (2,8) – 2; (3,4) – 1; (4,5) – 4; (4,6) – 5; (5,6) – 3; (6,7) – 5; (6,8) – 3; (7,8) – 4.

**21.** (1,2) – 7; (1,3) – 1; (1,4) – 3; (1,5) – 5; (2,3) – 3; (2,4) – 2; (2,5) – 2; (3,4) – 4; (3,5) – 4; (4,5) – 1.

**22.** (1,2) – 7; (1,6) – 3; (1,8) – 4; (2,3) – 1; (3,4) – 2; (3,8) – 1; (4,5) – 3; (4,7) – 2; (5,6) – 5; (6,7) – 4; (7,8) – 2.

**23.** (1,2) – 3; (1,3) – 1; (1,6) – 4; (2,3) – 7; (2,6) – 2; (3,4) – 5; (3,5) – 4; (4,5) – 3; (4,6) – 2; (5,6) – 1.

**24.** (1,2) – 2; (1,3) – 7; (1,6) – 4; (2,3) – 1; (2,6) – 3; (3,4) – 5; (3,5) – 6; (3,6) – 5; (4,5) – 4; (4,6) – 7; (5,6) – 2.

**25.** (1,2) – 2; (1,8) – 3; (2,3) – 4; (2,4) – 7; (2,8) – 5; (3,4) – 7; (4,5) – 4; (4,6) – 6; (5,6) – 2; (6,7) – 1; (6,8) – 5; (7,8) – 5.

**26.** (1,2) – 1; (1,3) – 2; (1,4) – 4; (1,5) – 5; (2,3) – 2; (2,4) – 3; (2,5) – 1; (3,4) – 4; (3,5) – 6; (4,5) – 7.

**27.** (1,2) – 1; (1,6) – 2; (1,8) – 6; (2,3) – 2; (3,4) – 3; (3,8) – 7; (4,5) – 4; (4,7) – 1; (5,6) – 5; (6,7) – 4; (7,8) – 7.

**28.** (1,2) – 2; (1,3) – 7; (1,6) – 4; (2,3) – 1; (2,6) – 3; (3,4) – 5; (3,5) – 6; (4,5) – 4; (4,6) – 7; (5,6) – 2.

**29.** (1,2) – 3; (1,3) – 1; (1,6) – 4; (2,3) – 7; (2,6) – 2; (3,4) – 5; (3,5) – 4; (3,6) – 5; (4,5) – 3; (4,6) – 2; (5,6) – 1.

**30.** (1,2) – 3; (1,8) – 2; (2,3) – 3; (2,4) – 1; (2,8) – 5; (3,4) – 2; (4,5) – 4; (4,6) – 4; (5,6) – 1; (6,7) – 7; (6,8) – 5; (7,8) – 5.

## *Нахождение кратчайшего пути в сети без контуров*

Задача о нахождении кратчайшего пути имеет столько практических применений и интерпретаций, что важность её не нуждается в обсуждении. Ниже рассматриваются классические алгоритмы, которые должен знать каждый программист.

В дальнейшем мы используем понятие сети, оно нуждается в некотором уточнении, так как нет единого подхода в его понимании. Под *сетью* подразумевается просто связный ориентированный граф $D$ ($V$, $E$), в котором, возможно, выделены вход и выход. Более узкое толкование термина «сеть» предполагает существование одного источника и одного стока, об этом по мере необходимости мы будем говорить отдельно. Нужно отметить, что существуют и другие толкования термина. Отметим также, что в ориентированном графе $D$ ($V$, $E$), как и в неориентированном $G$ ($V$, $E$), используется название «вершина», а не узел.

### Описание алгоритма

Этот алгоритм используется для решения задачи о нахождении кратчайшего пути в сети без контуров. Пусть задана сеть из $n + 1$ вершин, то есть ориентированный граф. В нем выделены две вершины – вход (нулевая вершина) и выход (вершина с номером $n$). Для каждой дуги заданы числа, называемые *длинами* дуг. *Длиной пути (контура)* называется сумма длин входящих в него дуг (если длины дуг не заданы, то длина пути (контура) определяется как число входящих в него дуг).

Задача заключается в поиске кратчайшего пути (пути минимальной длины) от входа до выхода сети. Будем предполагать, что в любую вершину сети можно попасть из входа и из любой вершины можно попасть в выход (вершины, не удовлетворяющие этому требованию, можно удалить).

Известно, что для существования кратчайшего пути необходимо и достаточно отсутствия в сети контуров отрицательной длины.

Предположим, что в сети нет контуров. Тогда всегда можно пронумеровать вершины таким образом, что для любой дуги $(i, j)$ имеет место $j > i$. Такая нумерация называется правильной. В сети без контуров всегда существует правильная нумерация.

Если изначально в задаче дается неправильная (произвольная) нумерация вершин, то, прежде чем использовать метод потенциалов, нужно определить правильную нумерацию, используя алгоритм топологической сортировки и алгоритм Уоршалла.

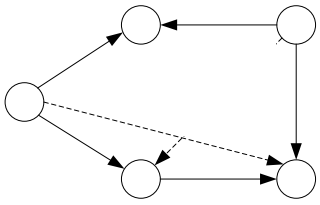


*Рис. 15*

***Алгоритм Уоршалла*** заключается в вычислении транзитивного замыкания. Для того чтобы понять принцип алгоритма Уоршалла, рассмотрим пример. Пусть задан граф с произвольной нумерацией вершин (рис. 15). Составим для этого графа две матрицы смежности.

**Матрица смежности для первоначального графа**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 |

**Матрица смежности с транзитивным замыканием**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 |

Номер строки в матрице соответствует номеру исходящей вершины, а номер столбца – входящей. Единица на пересечении $i$-й строки и $j$-го столбца ставится в том случае, если есть путь, выходящий из $i$-й вершины и входящий в $j$-ю. Во все остальные ячейки матрицы заносятся нули.

Алгоритм Уоршалла, например, может быть реализован циклом по $k$ ($k = 1, \ldots, p$). На каждом шаге $k$ цикла в позицию $(i, j)$, где $i, j$ отличны от $k$, заносят 1, если в $(i, k)$, $(k, j)$ одновременно находятся единицы, и оставляют без изменений в противном случае.

С помощью полученной матрицы смежности с транзитивным замыканием можно достроить на графе дополнительные дуги: добавленные в матрицу единицы обозначают новые дуги в графе (на рис. 15 они обозначены пунктиром).

***Алгоритм топологической сортировки*** – это алгоритм дополнения частичного порядка на конечном множестве. За основу берется матрица смежности, полученная алгоритмом Уоршалла. Рассматриваем в ней столбцы. Выбираем столбец, содержащий только нулевые элементы, и условно его вычеркиваем вместе со строкой этого же номера. Номер этого столбца записываем, причем вариантов выкидывания может быть несколько из-за того, что может оказаться несколько таких столбцов. Остается матрица, меньшая предыдущей на столбец и строку, с ней проделывют ту же операцию, и так до тех пор, пока не останется один элемент, номер которого также записываем.

Каждый раз записывая номер вычеркнутого столбца, получаем последовательность номеров вершин, которая идет не по порядку. Под ней выписываем номера по порядку.

**1 – 3 – 2 – 5 – 4**
**1 – 2 – 3 – 4 – 5**

Это означает, что номеру, произвольно поставленному у вершины, соответствует другой номер, полученный в результате топологической сортировки. Чтобы теперь получить правильную нумерацию, необходимо заменить первоначальные номера вершин на правильные (на рисунках правильная нумерация дается в квадратных скобках).

*Замечание*: алгоритм Уоршалла и топологическая сортировка выполнены верно, если после проставления правильной нумерации на графе, номер вершины, из которой идет дуга, меньше номера вершины, в которую она входит.

После того как будет выбрана правильная нумерация вершин, можно использовать метод потенциалов для нахождения кратчайшего пути на графе. Он заключается в следующих шагах:

Шаг 0: помечаем нулевую вершину (вход) индексом $\lambda_0 = 0$.

Шаг $k$: помечаем вершину $k$ индексом $\lambda_k = \min ( \lambda_i + l_{ik} )$.

Здесь используются следующие обозначения: $l_{ij}$ – длина дуги $(i, j)$; $\lambda_n$ – потенциал вершины (индекс выхода), равный длине кратчайшего пути.

Кратчайший путь здесь ищется по принципу оптимальности Беллмана. Он формулируется так: если ищется кратчайший путь между двумя точками, то длина пути между любыми двумя точками кратчайшего пути также должна быть минимальна. Когда потенциалы установлены, кратчайший путь определяется методом обратного хода от выхода к входу, то есть кратчайшим является путь $\mu = (0; i_1; i_2; \ldots; i_{n-1}; n)$, такой, что $l_{in;1n} = \lambda_n - \lambda_{n-1}$ и т.д.

### Пример

На рис. 16 приведен пример применения метода потенциалов для определения кратчайшего пути.

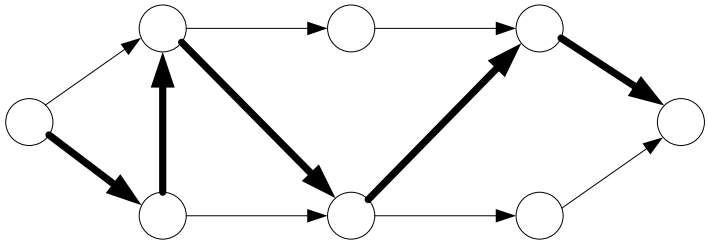


*Рис. 16*

Задана сеть из 8 вершин, входом является вершина с номером 0, а выходом – вершина с номером 7. Числа у дуг равны длинам дуг. Нумерация вершин является правильной, т.к. для любой дуги $(i, j)$ имеет место $j > i$.

Помечаем первую вершину (вход) индексом 0, т.е. потенциал этой вершины равен 0. Индексы вершин – потенциалы – на рисунке помещены в квадратные скобки.

Далее определяем потенциал для первой вершины. В первую вершину можно попасть только из нулевой, поэтому складываем потенциал этой вершины с длиной дуги между ними (то есть [0]+3=[3]). Потенциал первой вершины равен 3.

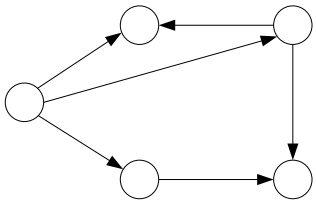
В отличие от первой во вторую вершину можно попасть либо из нулевой вершины, либо из первой. Определяем длины этих путей. В первом случае он будет равен 9 ([0]+9), а во втором – 8 (потенциал первой вершины + длина дуги между первой и второй вершиной, т.е. [3]+5). Из двух путей выбираем наименьший. Таким образом, потенциал второй вершины будет равен 8.

Определяя потенциал третьей вершины, рассматриваем пути только от ближайших вершин, то есть от первой и второй вершин. В первом случае путь равен 11, а во втором – 10. Соответственно потенциал 3-й вершины равен 10.

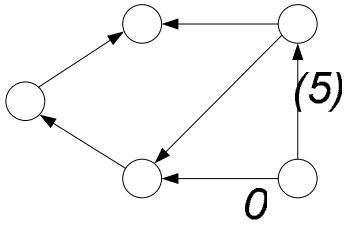
Далее аналогичным образом расставляем потенциалы всех остальных вершин в графе. После того как все потенциалы расставлены, методом обратного хода устанавливаем кратчайший путь. На рис. 16 он выделен жирными линиями.

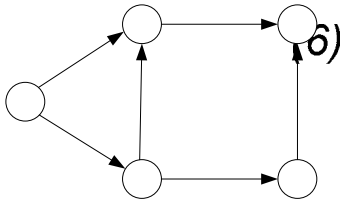
### Задания

Определить кратчайший путь в графе.

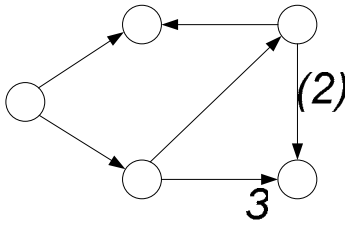


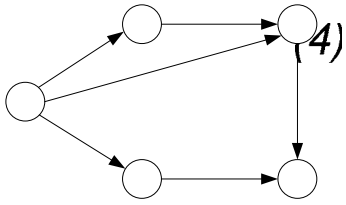
1.



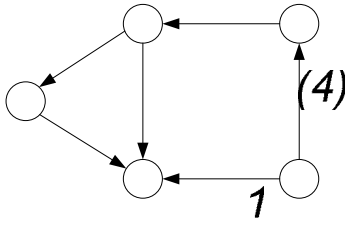




2.

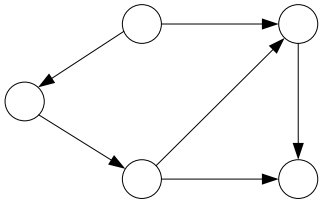


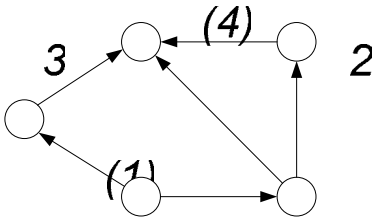



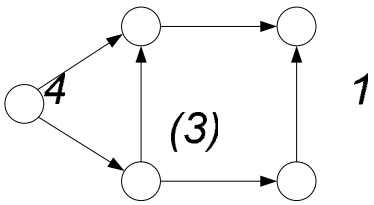

3.

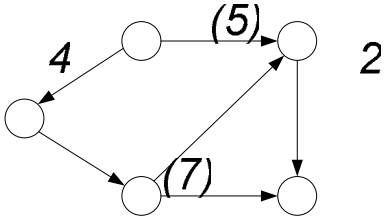


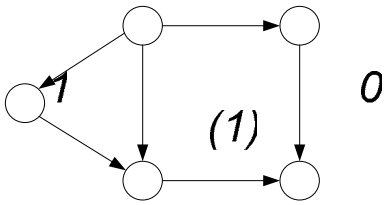

(8)

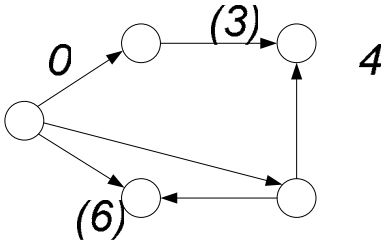




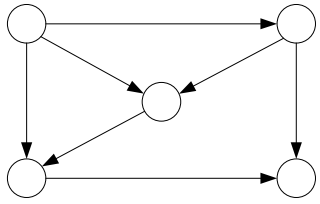

3 (5)

4.

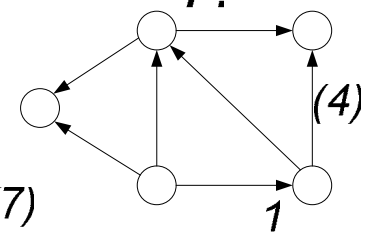
(4)

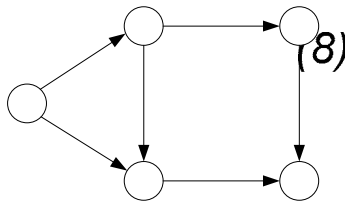
4 3

(3)

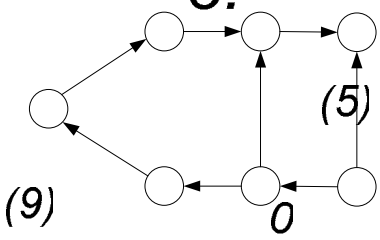


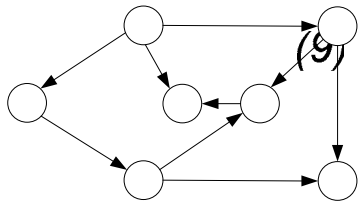
7.



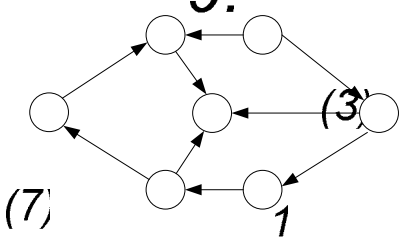




8.




9.



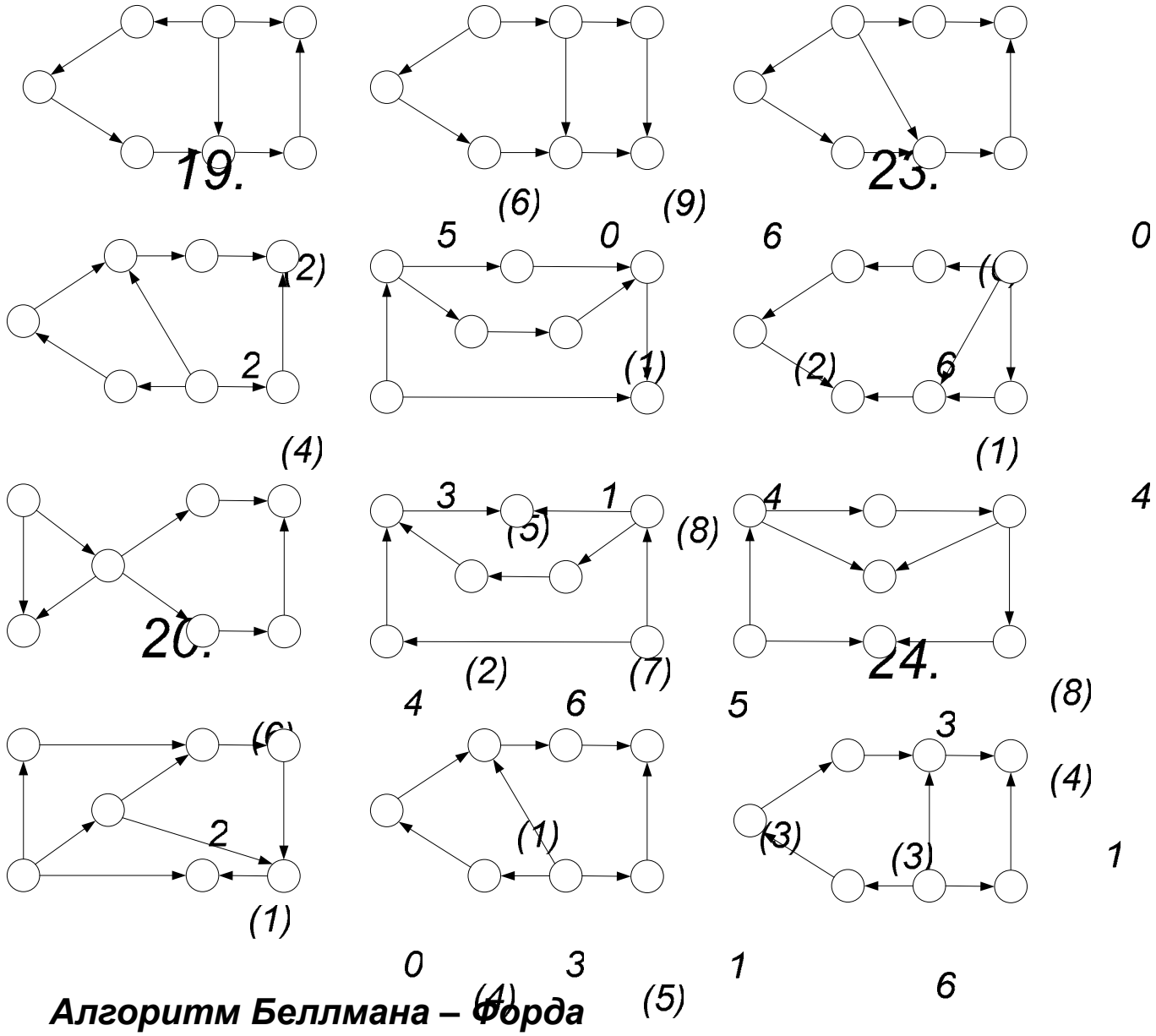

(7)

2 3

10.

1

(5)



## *Алгоритм Беллмана – Форда*

### Описание алгоритма

Предположим, что вершина 1 является вершиной-входом, и требуется найти длины кратчайших путей от вершины 1 до каждой другой вершины графа. Для этого алгоритма дуговые расстояния могут быть как положительными, так и отрицательными, но не должно быть циклов отрицательной длины. Предположим, что если в графе отсутствует та или иная дуга, то её вес равен бесконечно большому числу. Основная идея алгоритма Беллмана – Форда состоит в том, чтобы сначала найти длины кратчайших путей, при условии, что пути содержат не более двух дуг, не более трех дуг и т.д. Кратчайший путь, при условии, что он содержит не более *h* дуг, будем называть *h*-кратчайшим путем в графе. Согласно идее Беллмана -

Форда, Р. Прим предложил алгоритм нахождения длин кратчайших путей, ведущих из произвольной вершины графа во все остальные его вершины (в которые есть пути из вершины-входа), состоящий в присвоении вершинам некоторых потенциалов. Признаком окончания алгоритма является остановка процесса изменения потенциалов.

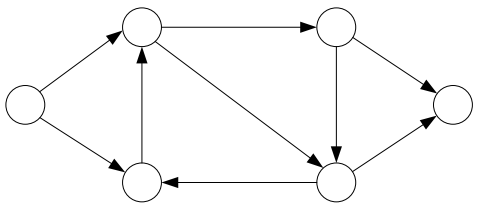
Алгоритм, предложенный Р. Примом, состоит в выполнении следующих операций:

1. Вершине-входу присваиваем потенциал $\varphi_1 = 0$; остальным вершинам – потенциал $\varphi_i = \infty$, $i = 1 \ldots n$, где $n$ - число вершин в рассматриваемом графе.
2. Для каждой вершины $i$, смежной с вершиной $j$ из массива вершин $L$, находим $\varphi_j + l(j, i)$ и проверяем неравенство
$$\varphi_j + l(j, i) < \varphi_i.$$
Если неравенство выполняется, то
$$\varphi_i := \varphi_j + l(j,i).$$
Вершину $i$ заносим в массив $L_1$.
3. Проверяем массив $L_1 \neq$ Ш. Если неравенство выполняется $L := L_1$, осуществляем переход к п. 2.
4. Конец алгоритма нахождения кратчайших путей.

Рассмотрим работу алгоритма на примере.

## Пример

Допустим, имеется граф (рис. 17), представленный списком дуг:



(1,2) – 3
(1,3) – 9
(2,3) – 5
(3,4) – 1
(4,5) – 2
(4,6) – 6
(3,5) – 2
(5,2) – -1
(5,6) – 4
(6,0) – 0

*Рис. 17*

*Решение*:

Результаты работы алгоритма по шагам вычислений сведем в таблицу.

| **Шаг вычислений** | **Массив *L*** | **Потенциалы вершин** | | | | | | **Массив $L_1$** |
|---|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** | |
| 0 | Ш | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 1 |
| 1 | 1 | 0 | 3 | 9 | ∞ | ∞ | ∞ | 2, 3 |
| 2 | 2, 3 | 0 | 3 | 8 | 10 | 11 | ∞ | 3, 4, 5 |
| 3 | 3, 4, 5 | 0 | 3 | 8 | 9 | 10 | 15 | 4, 5, 6 |
| 4 | 4, 5, 6 | 0 | 3 | 8 | 9 | 10 | 14 | 6 |
| 5 | 6 | 0 | 3 | 8 | 9 | 10 | 14 | Ш |

На нулевом шаге производится инициализация согласно п.1 алгоритма. Далее на первом шаге вычислений рассматриваются вершины, смежные с вершиной-входом, потенциалы рассматриваемых вершин становятся равными длине соединяющей их дуги, если таковое существует, в противном случае – потенциал вершины остается равным ∞.

На втором шаге рассматриваем вершины, достижимые из вершины-входа при помощи пути, состоящего из двух дуг. Таковыми являются:

1) 1 – 2 – 3: потенциал вершины 3 изменяется $\varphi_3 = 8$;
2) 1 – 3 – 4: потенциал вершины 4 изменяется $\varphi_4 = 10$;
3) 1 – 3 – 5: потенциал вершины 5 изменяется $\varphi_5 = 11$.

Третий шаг:

1) 1 – 3 – 4 – 6: потенциал вершины 6 изменяется $\varphi_6 = 16$;
2) 1 – 2 – 3 – 5: потенциал вершины 5 изменяется $\varphi_5 = 10$;
3) 1 – 3 – 5 – 2: потенциал вершины 2 не изменится ($\varphi_2 < 10$);
4) 1 – 3 – 4 – 5: потенциал вершины 5 не изменится ($\varphi_5 < 12$);
5) 1 – 3 – 5 – 6: потенциал вершины 6 изменится $\varphi_6 = 15$.

Четвертый шаг:

1) 1 – 2 – 3 – 5 – 6: потенциал вершины 6 изменится $\varphi_6 = 14$;
2) 1 – 2 – 3 – 4 – 5: потенциал вершины 5 не изменится ($\varphi_5 < 11$);
3) 1 – 2 – 3 – 5 – 2: потенциал вершины 2 не изменится ($\varphi_2 < 9$);
4) 1 – 3 – 5 – 2 – 3: потенциал вершины 3 не изменится ($\varphi_3 < 15$);
5) 1 – 3 – 4 – 5 – 2: потенциал вершины 2 не изменится ($\varphi_2 < 11$).

Пятый шаг:

1) 1 – 3 – 5 – 2 – 3 – 5: потенциал вершины 5 не изменится ($\varphi_5 < 17$);
2) 1 – 3 – 5 – 2 – 3 – 4: потенциал вершины 4 не изменится ($\varphi_4 < 16$);
3) 1 – 3 – 4 – 5 – 2 – 3: потенциал вершины 3 не изменится ($\varphi_3 < 14$);
4) 1 – 2 – 3 – 4 – 5 – 2: потенциал вершины 2 не изменится ($\varphi_2 < 10$);
5) 1 – 2 – 3 – 4 – 5 – 6: потенциал вершины 6 не изменится ($\varphi_6 = 14$);
6) 1 – 2 – 3 – 5 – 2 – 3: потенциал вершины 3 не изменится ($\varphi_3 < 14$).

На пятом шаге вычислений не произошло изменений ни в одном из потенциалов вершин графа, делаем вывод об окончании работы алгоритма.

Восстановим кратчайший путь из вершины 6 в вершину 1. Для этого для каждой вершины $j$ определим такую вершину $i$, что будет выполняться правило

$$\varphi_i := \varphi_j + l(j,i)$$

Следуя правилу, последовательно находим кратчайший путь:

вершина 6: $i = 5$;

вершина 5: $i = 3$;

вершина 3: $i = 2$;

вершина 2: $i = 1$.

Кратчайшим путем (рис. 18) является путь

1 – 2 – 3 – 5 – 6.

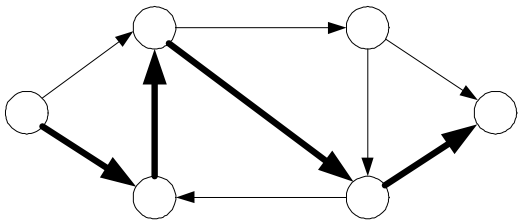


*Рис. 18*

## Задания

Дан список дуг с указанием их длин. Составьте по нему рисунок ориентированного графа. Найдите для этого графа наименьший путь от вершины-входа до вершины с максимальным номером.

**1.** (0;1) – 3, (0;2) – 9, (1;2) – 5, (2;4) – 1, (1;3) – 8, (2;3) – 2, (3;5) – 4, (4;5) – 6.

**2.** (0;1) – 4, (0;2) – 5, (1;2) – 8, (2;4) – 3, (1;3) – 11, (2;3) – 5, (3;5) – 3, (4;5) – 6.

**3.** (0;1) – 3, (0;2) – 9, (1;2) – 12, (2;4) – 1, (1;3) – 2, (2;3) – 3, (3;5) – 10, (4;5) – 5.

**4.** (0;1) – 6, (0;2) – 2, (2;1) – 3, (2;4) – 6, (1;3) – 1, (2;3) – 5, (3;5) – 8, (4;5) – 7.

**5.** (0;1) – 6, (0;2) – 5, (1;2) – 1, (2;4) – 6, (1;3) – 7, (2;3) – 6, (3;5) – 8, (4;5) – 7.

**6.** (0;1) – 3, (0;2) – 2, (2;1) – 1, (2;5) – 3, (1;5) – 4, (5;4) – 8, (5;3) – 5, (3;4) – 3, (4;6) – 2, (3;6) – 4.

**7.** (0;1) – 10, (0;2) – 5, (2;1) – 4, (2;5) – 8, (1;5) – 3, (5;4) – 4, (5;3) – 2, (3;4) – 1, (4;6) – 5, (3;6) – 7.

**8.** (0;1) – 3, (0;2) – 2, (2;1) – 2, (2;5) – 12, (1;5) – 8, (5;4) – 2, (5;3) – 6, (3;4) – 1, (4;6) – 8, (3;6) – 3.

**9.** (0;1) – 2, (0;2) – 7, (2;1) – 1, (2;5) – 6, (1;5) – 12, (5;4) – 10, (5;3) – 5, (3;4) – 4, (4;6) – 2, (3;6) – 7.

**10.** (0;1) – 4, (0;2) – 2, (2;1) – 1, (2;5) – 7, (1;5) – 5, (5;4) – 4, (5;3) – 1, (3;4) – 4, (4;6) – 3, (3;6) – 7.

**11.** (0;2) – 2, (0;1) – 7, (2;1) – 4, (2;4) – 9, (1;3) – 3, (3;4) – 1, (4;6) – 2, (3;5) – 8, (6;5) – 4, (6;7) – 10, (5;7) – 5.

**12.** (0;2) – 10, (0;1) – 5, (2;1) – 1, (2;4) – 4, (1;3) – 3, (3;4) – 5, (4;6) – 3, (3;5) – 10, (6;5) – 10, (6;7) – 5, (5;7) – 1.

**13.** (0;2) – 4, (0;1) – 6, (2;1) – 4, (2;4) – 6, (1;3) – 3, (3;4) – 2, (4;6) – 4, (3;5) – 7, (6;5) – 3, (6;7) – 8, (5;7) – 5.

**14.** (0;2) – 3, (0;1) – 1, (2;1) – 4, (2;4) – 2, (1;3) – 6, (3;4) – 4, (4;6) – 3, (3;5) – 6, (6;5) – 4, (6;7) – 12, (5;7) – 7.

**15.** (0;2) – 8, (0;1) – 12, (2;1) – 3, (2;4) – 6, (1;3) – 5, (3;4) – 4, (4;6) – 10, (3;5) – 4, (6;5) – 6, (6;7) – 10, (5;7) – 6.

**16.** (0;1) – 3, (0;2) – 3, (1;4) – 3, (2;5) – 3, (1;3) – 2, (0;3) – 4, (2;3) – 2, (3;4) – 2, (3;6) – 4, (3;5) – 2, (4;6) – 3, (5;6) – 3.

**17.** (0;1) – 3, (0;2) – 2, (1;4) – 13, (2;5) – 13, (1;3) – 7, (0;3) – 11, (2;3) – 9, (3;4) – 5, (3;6) – 10, (3;5) – 3, (4;6) – 4, (5;6) – 7.

**18.** (0;1) – 4, (0;2) – 5, (1;4) – 7, (2;5) – 7, (1;3) – 5, (0;3) – 8, (2;3) – 4, (3;4) – 2, (3;6) – 7, (3;5) – 1, (4;6) – 4, (5;6) – 6.

**19.** (0;1) – 5, (0;2) – 5, (1;4) – 4, (2;5) – 5, (1;3) – 3, (0;3) – 10, (2;3) – 3, (3;4) – 3, (3; 6) – 10, (3;5) – 3, (4;6) – 7, (5;6) – 5.

**20.** (0;1) – 6, (0;2) – 7, (1;4) – 18, (2;5) – 19, (1;3) – 8, (0;3) – 14, (2;3) – 6, (3;4) – 8, (3;6) – 14, (3;5) – 5, (4;6) – 5, (5;6) – 9.

**21.** (0;1) – 2, (1;2) – 3, (0;2) – 6, (2;3) – 4, (3;4) – 2, (2;4) – 7, (4;5) – 5, (5;6) – 3, (4;6) – 9.

**22.** (0;1) – 3, (1;2) – 5, (0;2) – 7, (2;3) – 6, (3;4) – 5, (2;4) – 12, (4;5) – 3, (5;6) – 2, (4;6) – 4.

**23.** (0;1) – 5, (1;2) – 4, (0;2) – 10, (2;3) – 4, (3;4) – 5, (2;4) – 8, (4;5) – 7, (5;6) – 6, (4;6) – 14.

**24.** (0;1) – 2, (1;2) – 4, (0;2) – 5, (2;3) – 4, (3;4) – 5, (2;4) – 8, (4;5) – 3, (5;6) – 2, (4;6) – 4.

**25.** (0;1) – 3, (1;2) – 2, (0;2) – 5, (2;3) – 1, (3;4) – 1, (2;4) – 3, (4;5) – 2, (5;6) – 2, (4;6) – 3.

**26.** (0;1) – 10, (0;2) – 4, (1;4) – 8, (2;5) – 20, (3;1) – 5, (4;3) – 3, (2;3) – 4, (3;5) – 17, (4;6) – 7, (5;6) – 5.

**27.** (0;1) – 6, (0;2) – 2, (1;4) – 4, (2;5) – 15, (3;1) – 2, (4;3) – 3, (2;3) – 13, (3;5) – 2, (4;6) – 7, (5;6) – 1.

**28.** (0;1) – 2, (0;2) – 3, (1;4) – 1, (2;5) – 10, (3;1) – 1, (4;3) – 6, (2;3) – 4, (3;5) – 5, (4;6) – 20, (5;6) – 6.

**29.** (0;1) – 2, (0;2) – 3, (1;4) – 7, (2;5) – 6, (3;1) – 1, (4;3) – 2, (2;3) – 2, (3;5) – 3, (4;6) – 8, (5;6) – 10.

**30.** (0;1) – 3, (0;2) – 7, (1;4) – 8, (2;5) – 3, (3;1) – 1, (4;3) – 4, (2;3) – 2, (3;5) – 2, (4;6) – 7, (5;6) – 6.

## *Алгоритм Дейкстра*

### Описание алгоритма

Алгоритм Дейкстра требует, чтобы длины всех дуг были положительны. Объем вычислений в худшем случае для этого алгоритма значительно меньше, чем у алгоритма Беллмана – Форда. Основная его идея состоит в том, чтобы отыскивать кратчайшие пути в порядке возрастания длины пути. Кратчайшим среди всех кратчайших путей от вершины-входа является путь, состоящий из одной дуги, соединяющий вершину-вход с ближайшей соседней вершиной, так как любой путь, состоящий из нескольких дуг, будет всегда длиннее первой дуги вследствие предположения о положительности всех дуговых длин. Следующим кратчайшим среди кратчайших путей должен быть либо путь из одной дуги к следующему ближайшему со-

седу вершины-входа, либо кратчайший путь из двух дуг, проходящий через вершину, выбранный на первом шаге и т.д. Алгоритм Дейкстра состоит в выполнении следующих операций:

Шаг 0. Помечаем нулевую вершину индексом $\lambda_0 = 0$.

Шаг $k$. Пусть уже помечено некоторое количество вершин. Обозначим $Q$ множество непомеченных вершин, смежных с помеченными. Для каждой вершины $k$, принадлежащей $Q$, вычисляем величину $\xi_k = \min ( \lambda_k + l_{ki} )$, где минимум берется по всем помеченным вершинам $i$, смежным с вершиной $k$. Помечаем вершину $k$, для которой величина $\xi_k$ минимальна, индексом $\lambda_k = \xi_k$ . Подобную процедуру повторяем до тех пор, пока не будет помечена вершина $n$. Длина кратчайшего пути равна $\lambda_n$, а сам кратчайший путь определяется так, как это было описано выше.

### Пример

Найдем кратчайший путь из вершины 1 в вершину 7 (рис. 19).

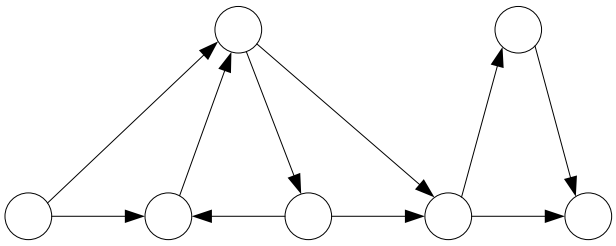


*Рис. 19*

1) Пометим вершину 1 индексом $\lambda_1 = 0$.
2) Смежными с 1-й являются вершины 2 и 3. $\xi_2 = 0 + 3 = 3$. $\xi_3 = 0 + 1 = 1$. Величина $\xi_3$ минимальна. Помечаем вершину 3: $\lambda_3 = 1$. Помечены вершины 1 и 3.
3) Смежными с помеченными вершинами (1 и 3) являются вершины 2, 4 и 5. $\xi_2 = 0 + 3 = 3$. $\xi_4 = 1 + 1 = 2$. $\xi_5 = 1 + 6 = 7$. Величина $\xi_4$ минимальна. Помечаем вершину 4 индексом: $\lambda_4 = 2$. Помечены вершины 1, 3 и 4.
4) Смежными с помеченными вершинами (1, 3 и 4) являются вершины 2 и 5. $\xi_5 = 2 + 4 = 6$. $\xi_5 = 1 + 6 = 7$. $\xi_2 = 0 + 3 = 3$. $\xi_2 = 2 + 2 = 4$. Помечаем вершину 2: $\lambda_2 = 3$. Помечены вершины 1, 2, 3, 4.
5) Смежной с помеченными вершинами (1, 2, 3, 4) является вершина 5. $\xi_5 = 2 + 4 = 6$. $\xi_5 = 1 + 6 = 7$. Помечаем вершину 5 индексом: $\lambda_5 = 6$. Помечены вершины 1, 2, 3, 4 и 5.
6) Смежными с помеченными вершинами (1, 2, 3, 4, 5) являются вершины 6 и 7. $\xi_6 = 6 + 2 = 8$. $\xi_7 = 6 + 9 = 15$. Величина $\xi_6$ минимальна для вершины 6. Помечаем вершину 6: $\lambda_6 = 8$.
7) Смежная с помеченными вершинами (1, 2, 3, 4, 5, 6) вершина 7. $\xi_7 = 6 + 9 = 15$. $\xi_7 = 8 + 1 = 9$. Помечаем вершину 7 индексом: $\lambda_7 = 9$.

8) Для вершины 6 смежной является вершина 7. $\xi_7 = 8 + 1 = 9$. Величина $\xi_7$ минимальна. Помечаем вершину 7 индексом $\lambda_7 = 9$.

С помощью алгоритма Дейкстра мы получили длину кратчайшего пути из вершины 1 в вершину 7, которая составит 9. Кратчайшим путем является путь 1 – 3 – 4 – 5 – 6 – 7 (рис. 20).

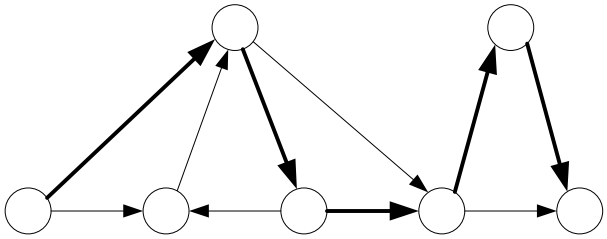


*Рис. 20*

## Задания

Для орграфа дана матрица весов. В позиции ( *i, j* ) записана длина дуги из вершины *i* в вершину *j* (ноль означает, что пути из *i* в *j* не существует). Если согласно матрице путь $i \to j$ существует, то путь $j \to i$ не существует. Задание: найти кратчайший путь из вершины 1 в вершину 5.

1. $\begin{pmatrix} 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

2. $\begin{pmatrix} 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$

3. $\begin{pmatrix} 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$

4. $\begin{pmatrix} 0 & 5 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$

5. $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

6. $\begin{pmatrix} 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$

7. $\begin{pmatrix} 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 2 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$

8. $\begin{pmatrix} 0 & 0 & 2 & 3 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$

9. $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 3 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

10. $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

11. $\begin{pmatrix} 0 & 0 & 0 & 3 & 9 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

12. $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{pmatrix}$