

# ЭЛЕМЕНТЫ ТЕОРИИ ГРАФОВ

## Введение в теорию графов

Основу теории графов составляет совокупность методов и представлений, сформировавшихся при решении конкретных прикладных задач. Граф есть совокупность точек и линий, соединяющих эти точки, причем соединения могут обладать различными характеристиками. В этой связи граф можно рассматривать как математическую модель для всякой системы, содержащей бинарные отношения, поэтому как теоретическая дисциплина теория графов является разделом дискретной математики, исследующим отношения между конечными множествами объектов. Можно выделить два основных типа задач в рамках теории графов. В первом случае требуется ответить на вопрос, существуют ли графы, обладающие определенным свойством, и если да, то сколько их или каково их максимальное количество. В другом случае нужно определить, как построить граф или подграф, обладающий некоторым заданным свойством.

Как прикладная дисциплина теория графов является замечательным инструментом для формализации целого ряда задач, связанных с дискретным размещением объектов. К ним, в частности, относятся: проектирование и исследование сетей связи, электрических и монтажных схем, программных систем; исследование автоматов, анализ и синтез логических цепей; задачи календарного планирования; поиск информации; разработка стратегий инвестиций, анализ качества, исследование движения транспорта, размещение предприятий коммунального обслуживания и логистических центров, исследование поведения индивидуумов.

## Основные понятия и определения

Пусть задано некоторое конечное множество  $X$ , элементы которого будем называть вершинами, и множество  $U$ , состоящее из пар элементов  $(x_i, x_j) \in X^2$ . Упорядоченная пара множеств  $G = (X, U)$  называется *графом*. Если в определении графа существенно в каком порядке выбираются вершины, то есть пара  $(x_i, x_j)$  отлична от пары  $(x_j, x_i)$ , то такой граф называют ориентированным или *орграфом*, а пару  $(x_i, x_j)$  – *дугой*, при этом считается, что  $x_i$  – начальная, а  $x_j$  – конечная вершины для данной дуги. Если начало и конец дуги совпадают, то она называется *петлей*. В геометрической интерпретации дуге соответствует направленный отрезок (рис. 1). Если в определении графа не существует порядок вершин при образовании пары  $(x_i, x_j)$ , то граф называют неориентированным или *неографом* (рис. 2), а пару  $(x_i, x_j)$  – *ребром*. Неограф, полученный из орграфа заменой каждой дуги на ребро, называется *основанием* орграфа.

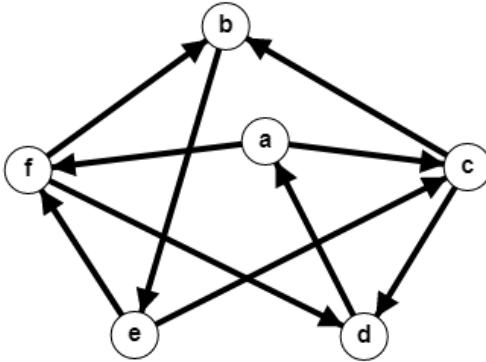


Рисунок 1 – Орграф

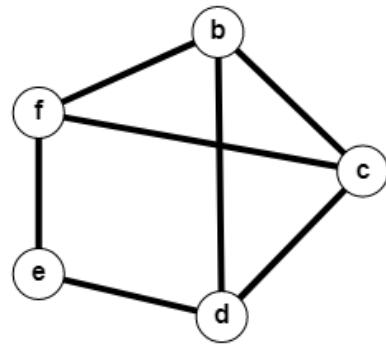


Рисунок 2 – Неорграф

Две вершины  $x_i$  и  $x_j$  называются *смежными*, если существует соединяющее их ребро (дуга), при этом вершины называются *инцидентными* этому ребру (дуге), а ребро (дуга) – *инцидентным(-ой)* этим вершинам. Аналогично, два различных ребра (дуги) называются *смежными*, если они имеют, по крайней мере, одну общую вершину. Вершина, для которой не существует инцидентных ей ребер, называется *изолированной*. Вершина, для которой существует только одно инцидентное ей ребро, называется *висячей*.

Если вершины графа соединены более чем одним ребром (дугой), то такой граф называется *мультиграфом*, а ребра (дуги) – *кратными*. Мультиграф не допускает петель. Граф, не содержащий петель и кратных ребер (дуг), называется *простым*.

Заметим, что всякий граф  $G = (X, U)$  определяет некоторое бинарное отношение на множестве  $X$ . Справедливо и обратное предложение: если  $X$  – конечное множество, то всякое бинарное отношение на  $X$  определяет граф, у которого множество вершин есть  $X$ . Из этого факта следует, что многие понятия теории бинарных отношений распространяются на графы. В частности, операции, вводимые для отношений, а также свойства отношений порождают аналогичные определения для графов.

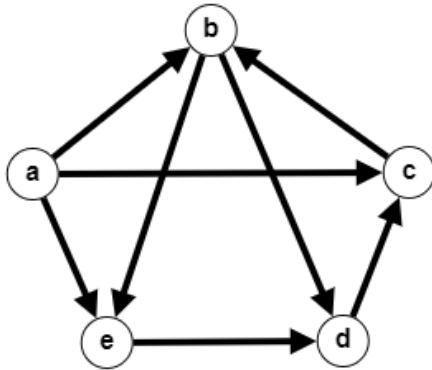
Граф  $G = (X, U)$  является *симметричным*, если в  $U$  для любой дуги  $(x_i, x_j)$  существует противоположно ориентированная дуга  $(x_j, x_i)$ . Предположив, что ребро равноценно двум противоположно направленным дугам, можно утверждать, что всякий неорграф является симметричным.

Орграф называется *антисимметричным*, если каждая пара смежных вершин соединена только в одном направлении и петли отсутствуют.

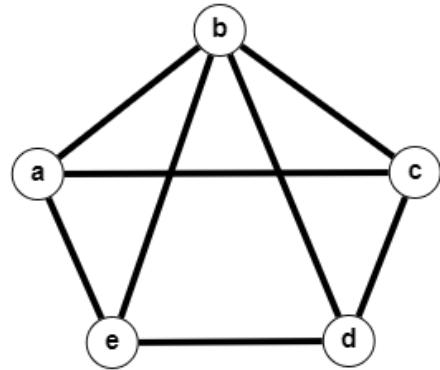
*Рефлексивный* граф соответствует рефлексивному бинарному отношению.

Для орграфа можно рассматривать свойство транзитивности. Граф называется *транзитивным*, если для каждой пары дуг  $(x_i, x_j)$  и  $(x_j, x_k)$  существует дуга  $(x_i, x_k)$ , которая называется *транзитивно замыкающей*.

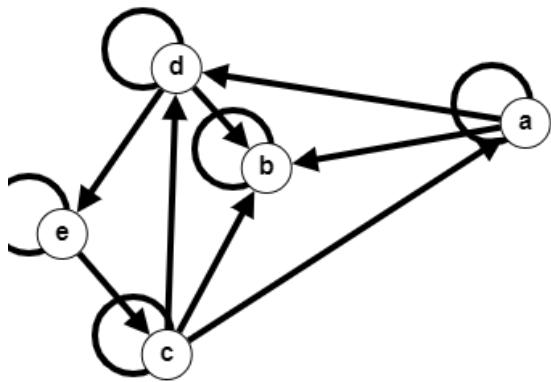
Свойства графов проиллюстрированы на следующем рисунке.



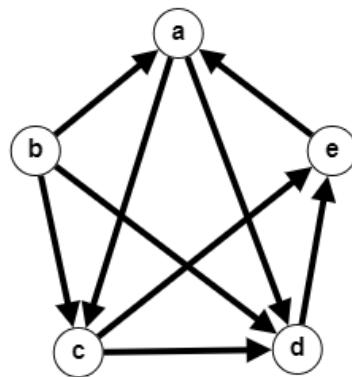
а) Антисимметричный граф –  
простой ориентированный граф



б) Симметричный граф –  
неориентированный граф

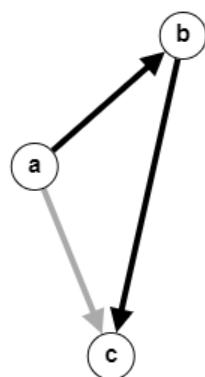


с) Рефлексивный граф

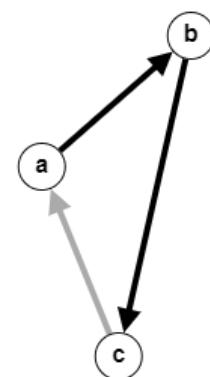


д) Транзитивный граф

Рисунок 3 – Свойства графов



а) Транзитивная тройка



б) Циклическая тройка

Рисунок 4 – Понятие транзитивно замыкающей дуги

Путем в графе  $G$  называется такая последовательность дуг, в которой конец каждой предыдущей дуги совпадает с началом следующей (рис. 5). Для неографа такая последовательность ребер называется цепью. Если путь (цепь)  $P$  проходит через вершины  $x_{i_1}, x_{i_2}, \dots, x_{i_m}$ , то будем обозначать его (ее) следующим образом:  $P = [x_{i_1}, x_{i_2}, \dots, x_{i_m}]$ .

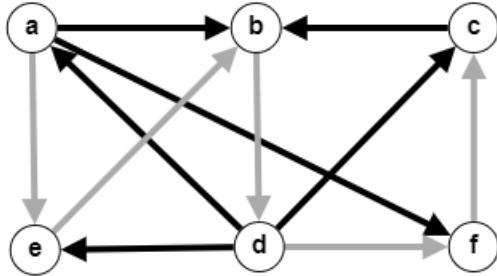


Рисунок 5 – Путь из вершины  $a$  в вершину  $c$  (выделен светлым цветом)

Под *длиной пути (цепи)* подразумевается количество дуг (ребер), которые составляют этот путь (цепь). Если каждой дуге (ребру) приписано некоторое число, называемой *весом*, то граф называется *взвешенным*. Тогда длина пути (цепи) – это сумма весов дуг (ребер), входящих в этот путь.

Путь (цепь) называется *простым(-ой)*, если в нем никакая дуга (ребро) не встречается дважды, и *составным(-ой)* – в противном случае. Простая цепь с  $n$  вершинами обозначается через  $P_n$ . Путь (цепь) называется *элементарным(-ой)*, если в нем ни одна вершина не встречается дважды. На рис. 5 выделенный путь является и простым, и элементарным. Путь (цепь), у которого(-ой) начальная и конечная вершина совпадают, называется *контуром (циклом)*. На рис. 6 выделен контур на вершинах  $b, c, f, d$ .

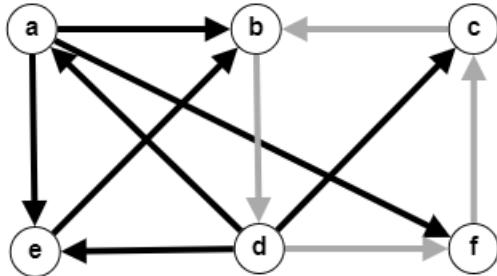


Рисунок 6 – Контур на вершинах  $b, c, f, d$

Элементарный путь, проходящий через все вершины графа, называется *гамильтоновым*. На рис. 1 имеется гамильтонов путь  $P = [f, b, e, c, d, a]$  из вершины  $f$  в вершину  $a$ . Так как существует дуга  $(a, f)$ , то имеем контур, который также называется гамильтоновым.

Рассмотрим графы, обладающие определенными свойствами.

Граф, в котором любые две вершины смежны, называется *полным*. Полный граф на  $n$  вершинах обозначается  $K_n$  (рис. 3).

Полный орграф называется *турниром*. Этот термин получил свое название от соревнований по круговой системе, графическое представление которых имеет структуру полного ориентированного графа. В турнирах по круговой системе играют несколько команд, каждая со всеми остальными по одному разу. Игра по правилам не может закончитьсяничью. В представлении

графом командам соответствуют вершины, а дуга  $(x, y)$  присутствует тогда и только тогда, когда команда  $x$  победила команду  $y$ . Количество очков команды соответствует числу побежденных ею противников.

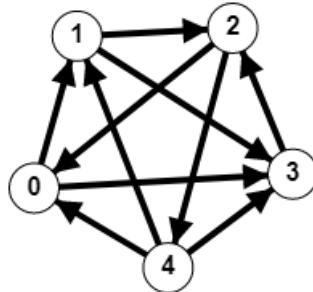


Рисунок 7 – Граф  $K_5$ , являющийся турниром

Граф  $G = (X, U)$ , множество вершин которого можно разбить на два непересекающихся множества  $X_1$  и  $X_2$  так, что каждое ребро в  $G$  соединяет какую-нибудь вершину из  $X_1$  с какой-либо вершиной из  $X_2$ , называется *двуодольным*, при этом множества  $X_1$  и  $X_2$  называются *долями*. Полный двуодольный граф, в котором  $|X_1| = n_1$ ,  $|X_2| = n_2$  обозначается через  $K_{n_1, n_2}$ .

Граф  $G$ , который можно изобразить на плоскости так, чтобы никакие два ребра не пересекались в точках, отличных от вершин, называется *планарным* графом. Такой рисунок называют *плоским графом* или *картой*  $G$ . Не все графы являются планарными и их можно представить картой. Например, графы  $K_5$  и  $K_{3,3}$  не являются планарными.



Рисунок 8 – Графы  $K_5$  и  $K_{3,3}$

Графы, образованные вершинами и ребрами пяти правильных многоугранников – платоновых тел: тетраэдра, куба, октаэдра, додекаэдра и икосаэдра, называются *платоновыми* графами. Заметим, что все платоновы графы являются полными.

Если задан граф  $G$ , то от него можно перейти к реберному и дополнительному графикам.

*Реберным* графиком  $L(G)$  простого графа  $G$  называется граф, вершины которого взаимно однозначно сопоставлены ребрам графа  $G$ , причем две

вершины в  $L(G)$  смежны тогда и только тогда, когда соответствующие им ребра смежны в  $G$ .

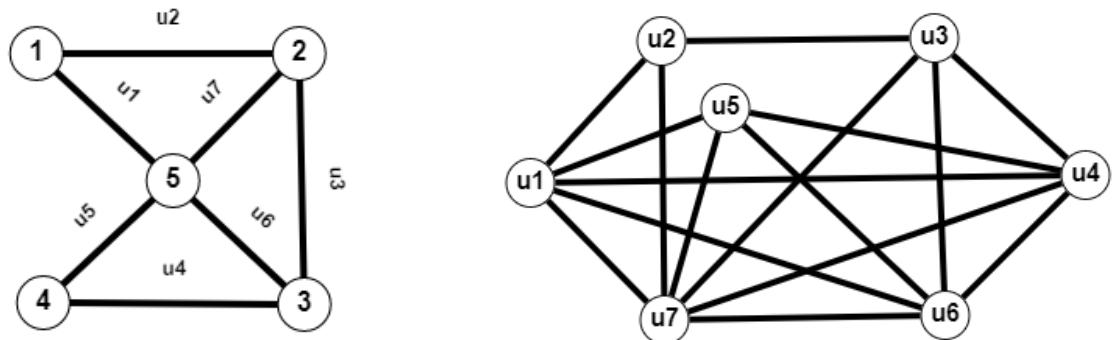


Рисунок 9 – Неорграф и его реберный граф

Орграф  $G^c$  называется *дополнительным* или *обратным* к данному орграфу  $G$ , если он имеет те же вершины, что и  $G$ , а дуга  $(x, y)$  принадлежит  $G^c$  тогда и только тогда, когда дуга  $(y, x)$  принадлежит  $G$ .

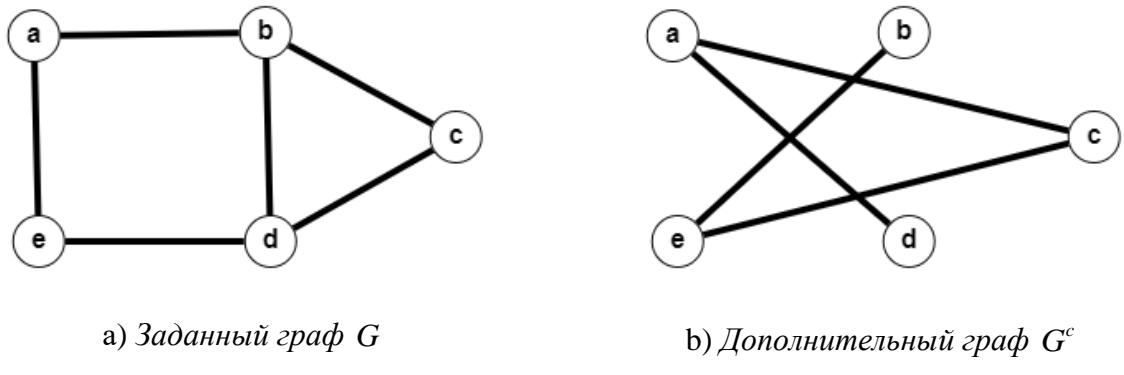


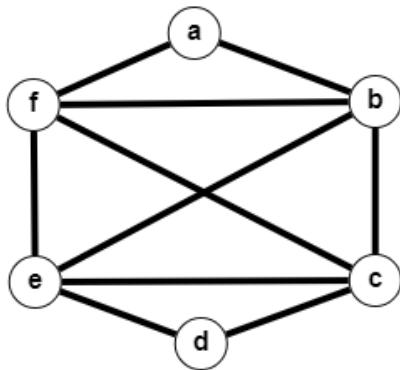
Рисунок 10 – Неорграф и его дополнительный граф

В графе можно выделить части – подграфы, обладающие определенными свойствами. Рассмотрим граф  $G = (X, U)$ .

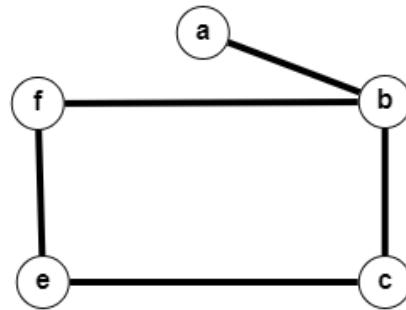
Граф  $G' = (X', U')$  называется *собственным подграфом* графа  $G$ , если  $X' \subset X$  и  $U' \subset U$  являются соответственно такими подмножествами  $X$  и  $U$ , что ребро  $(x, y)$  содержится в  $U'$  только в том случае, если  $x$  и  $y$  содержатся в  $X'$ .

Если  $X' = X$ , то такой собственный подграф называется *остовным*.

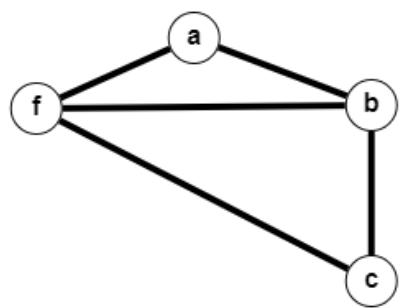
*Порожденным* подграфом графа  $G$  на множестве вершин  $X'$  называется собственный подграф  $G' = (X', U')$ , такой, что  $U'$  содержит все те ребра из  $U$ , которые соединяют вершины из  $X'$ .



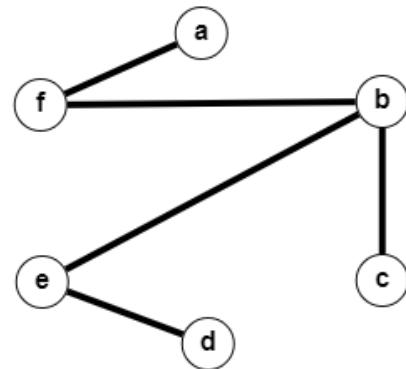
a) Заданный граф  $G$



b) Собственный подграф  $G$



c) Порожденный подграф на множестве вершин  $\{a, b, c, f\}$



d) Остовный подграф графа  $G$

Рисунок 10 – Граф и его подграфы

Подграф  $G'$  графа  $G$  называется *максимальным подграфом по отношению к некоторому свойству  $\alpha$* , если  $G'$  обладает свойством  $\alpha$  и  $G'$  не является собственным подграфом никакого другого подграфа графа  $G$ , обладающего свойством  $\alpha$ . Подграф  $G'$  графа  $G$  называется *минимальным подграфом по отношению к некоторому свойству  $\alpha$* , если  $G'$  обладает свойством  $\alpha$  и никакой подграф графа  $G$ , обладающий свойством  $\alpha$ , не является собственным подграфом  $G'$ .

Свойство  $\alpha$  графа  $G$  называется *наследственным*, если каждый подграф графа  $\alpha$  обладает этим свойством. Примером наследственного свойства является, например, бесконтурность.

### Степени вершин графа

Определим в графе  $G = (X, U)$  для каждой вершины множество  $\Gamma(x_i) = \{x_j : (x_i, x_j) \in U\}$ , которое называется *окрестностью* вершины  $x_i$ . Используя понятие окрестности, граф можно определить парой  $G = (X, \Gamma)$ , где  $\Gamma : X \rightarrow X$  – отображение, которое каждой вершине  $x_i \in X$  ставит в соответствие ее окрестность  $\Gamma(x_i)$ . Для  $\Gamma$  можно определить обратное отоб-

ражение  $\Gamma^{-1}$ , при этом  $\Gamma^{-1}(x_i) = \{x_k : (x_k, x_i) \in U\}$ . Пусть  $A \subset X$ , тогда  $\Gamma(A) = \bigcup_{x_i \in A} \Gamma(x_i)$ .

**Пример.** Рассмотрим граф на рис. 11.

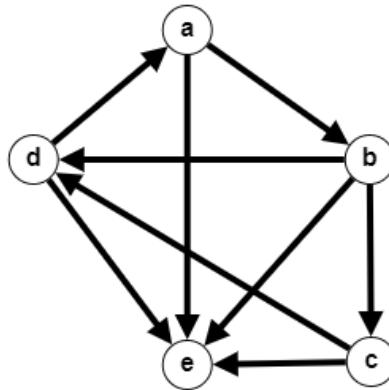


Рисунок 11 – Ориентированный граф  $G$

Для каждой вершины найдем множества, порождаемые отображениями  $\Gamma$  и  $\Gamma^{-1}$ .

$$\Gamma(a) = \{b, e\}, \Gamma(b) = \{d, e, c\}, \Gamma(c) = \{e, d\}, \Gamma(d) = \{a, e\}, \Gamma(e) = \emptyset;$$

$$\Gamma^{-1}(a) = \{d\}, \Gamma^{-1}(b) = \{a\}, \Gamma^{-1}(c) = \{b\}, \Gamma^{-1}(d) = \{b, c\}, \Gamma^{-1}(e) = \{a, b, c, d\}.$$

В орграфе для каждой вершины  $x_i$  можно определить *полустепень исхода*  $d_i^- = d^-(x_i) = |\Gamma(x_i)|$  – количество выходящих дуг и *полустепень захода*  $d_i^+ = d^+(x_i) = |\Gamma^{-1}(x_i)|$  – количество входящих дуг.

В неорграфе для каждой вершины  $x_i$  можно определить ее *степень*  $d_i = d(x_i) = |\Gamma(x_i)|$  – количество дуг, инцидентных данной вершине. Для орграфа  $d_i = d_i^+ + d_i^-$ .

**Пример.** Для графа на рис. 11 найдем полустепени исхода и захода, а также степени для всех вершин.

Таблица 1 – Степени, полустепени исхода и захода вершин

$d_a^- = 2$	$d_b^- = 3$	$d_c^- = 2$	$d_d^- = 2$	$d_e^- = 0$
$d_a^+ = 1$	$d_b^+ = 1$	$d_c^+ = 1$	$d_d^+ = 2$	$d_e^+ = 4$
$d_a = 3$	$d_b = 4$	$d_c = 3$	$d_d = 4$	$d_e = 4$

Имеют место следующие утверждения:

$$1) \sum_{\{i: x_i \in X\}} d_i^- = \sum_{\{i: x_i \in X\}} d_i^+ = m, \text{ где } m = |U|;$$

- 2) сумма степеней вершин простого графа равна удвоенному числу ребер, то есть  $\sum_{\{i: x_i \in X\}} d_i = 2m$ , где  $m = |U|$  (теорема Эйлера).

Граф, у которого все вершины имеют одну и ту же степень, например,  $r$ , называется *регулярным (однородным) степени  $r$* . Например, граф-треугольник является регулярным графом степени 2. Примерами регулярных графов являются платоновы графы.

Для каждого графа можно определить упорядоченную последовательность  $d_1 \geq d_2 \geq \dots \geq d_n$  степеней, которая называется *степенной последовательностью*.

Для графа на рис. 11 степенная последовательность имеет вид  $(4, 4, 4, 3, 3)$ .

С другой стороны, известна следующая задача: пусть задана упорядоченная последовательность  $d_1 \geq d_2 \geq \dots \geq d_n$ ; требуется определить, существует ли граф, для которого заданная последовательность является степенной. Если существует граф  $G$  на  $n$  вершинах  $x_1, x_2, \dots, x_n$  такой, что для вершины  $x_i$  ее степень равна  $d_i$  для каждого индекса  $i = \overline{1, n}$ , то такая последовательность называется *графической*. В общем случае графическая последовательность имеет много реализаций и их число определить сложно.

**Замечание 1.** В бесконтурном графе существует, по крайней мере, одна вершина с нулевой полустепенью захода и одна вершина с нулевой полустепенью исхода. Вершина с нулевой полустепенью захода называется *источником*, а с нулевой полустепенью исхода – *стоком*. *Топологической сортировкой* бесконтурного графа называется процедура получения правильной (монотонной) нумерации вершин – такой, что для каждой дуги  $(i, j)$  номер ее начала меньше номера конца, т.е. выполняется неравенство  $i < j$ . В основе топологической сортировки лежит разложение графа на уровни. Если оно получено, то вершины нумеруются, начиная с верхнего уровня, при этом, если вершины расположены на одном уровне, то номера им приписываются в произвольном порядке.

**Пример.** Пусть задан граф  $G$ , изображенный на рис. 11.

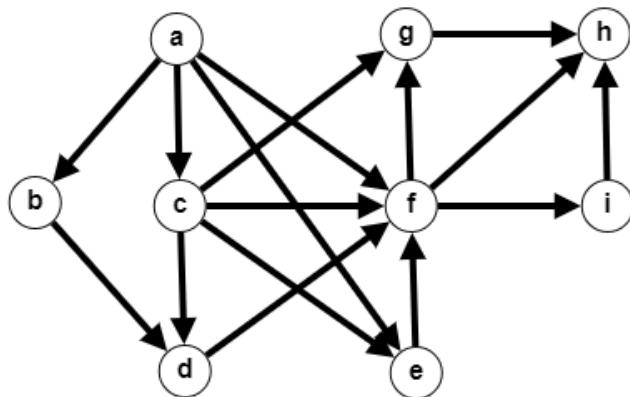


Рисунок 11 – Граф  $G$

Построим его разложение на уровни, например, с помощью метода вычеркивания дуг: на первом уровне располагаем вершины без входящих дуг – эти вершины считаем просмотренными. Затем удаляем из графа просмотренные вершины. В полученном подграфе ищем вершины без входящих дуг и располагаем их на следующем уровне и так действует до тех пор, пока все вершины не будут распределены по уровням. Результаты работы метода представлены в следующей таблице.

Таблица 2 – Пошаговая реализация метода вычеркивания дуг

<pre> graph LR     b((b)) --&gt; d((d))     c((c)) --&gt; d     c((c)) --&gt; f((f))     d((d)) --&gt; e((e))     e((e)) --&gt; f     f((f)) --&gt; g((g))     f((f)) --&gt; h((h))     f((f)) --&gt; i((i))     g((g)) --&gt; h((h))     h((h)) --&gt; i((i))   </pre>	<p><u>Итерация 1:</u> в графе <math>G</math> вершина <math>a</math> не имеет входящих дуг, поэтому ей приписывается уровень 1, и она исключается из графа.</p>
<pre> graph LR     b((b)) --&gt; d((d))     c((c)) --&gt; d     c((c)) --&gt; f((f))     d((d)) --&gt; e((e))     e((e)) --&gt; f     f((f)) --&gt; g((g))     f((f)) --&gt; h((h))     f((f)) --&gt; i((i))     g((g)) --&gt; h((h))   </pre>	<p><u>Итерация 2:</u> в полученном подграфе вершины <math>b</math> и <math>c</math> не имеют входящих дуг – им приписывается номер уровня 2, а затем исключаются из графа вместе с инцидентными дугами.</p>
<pre> graph LR     d((d)) --&gt; e((e))     f((f)) --&gt; g((g))     f((f)) --&gt; h((h))     f((f)) --&gt; i((i))     g((g)) --&gt; h((h))   </pre>	<p><u>Итерация 3:</u> на данной итерации вершины <math>d</math> и <math>e</math> не имеют входящих дуг, поэтому им приписывается следующий по порядку уровень 3, и они исключаются из графа.</p>
<pre> graph LR     f((f)) --&gt; g((g))     f((f)) --&gt; h((h))     f((f)) --&gt; i((i))     g((g)) --&gt; h((h))   </pre>	<p><u>Итерация 4:</u> вершина <math>f</math> получает номер уровня 4 и исключается из графа.</p>
<pre> graph LR     g((g)) --&gt; h((h))   </pre>	<p><u>Итерация 5:</u> вершинам <math>g</math> и <math>i</math> приписывается номер уровня 5, они исключаются из графа; оставшейся вершине <math>h</math> приписывается номер уровня 6.</p>

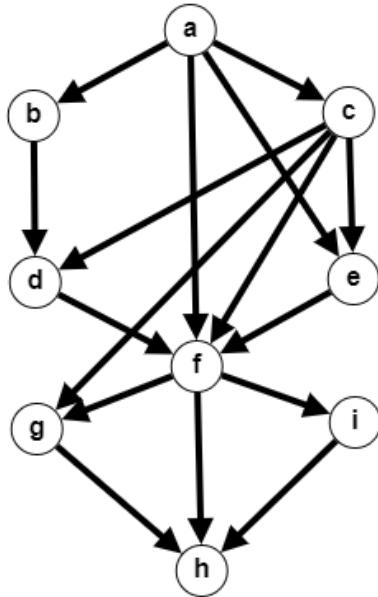


Рисунок 12 – Упорядоченный граф (разложен по уровням)

Важной особенностью разложения является то, что из вершин фиксированного уровня, исключая последний, дуги ведут только в вершины следующих уровней. Кроме того, вершины, находящиеся на одном уровне, не смежны.

**Замечание 2.** *Разбиением* неотрицательного числа  $n$  называется конечный набор неотрицательных чисел, сумма которых равна  $n$ . *Разбиение графа* – это представление числа  $2m$  в виде суммы степеней вершин графа, что, вообще говоря, не всегда возможно. Так, только два из пяти разбиений числа 4 ( $4+0$ ,  $3+1$ ,  $2+2$ ,  $2+1+1$ ,  $1+1+1+1$ ) на положительные слагаемые реализуются графиками (рис. 13).



Рисунок 13 – Графическое изображение разбиения числа 4

### Основные матрицы графов и их свойства

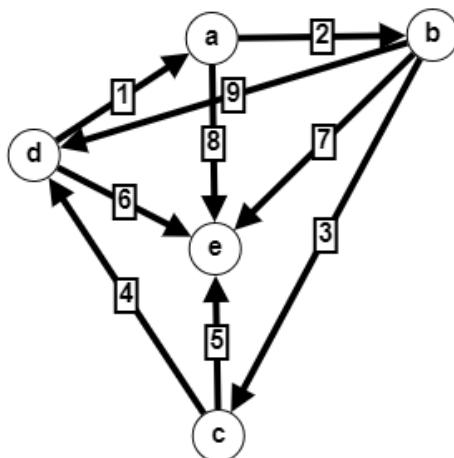
Основными матрицами графа являются матрица смежности и матрица инцидентности.

Матрица смежности графа  $G = (X, U)$ , в котором  $|X| = n$ , – это матрица  $A = (a_{ij})_{n \times n}$ , где

$$a_{ij} = \begin{cases} 1, & \text{если } (x_i, x_j) \in U, \\ 0, & \text{иначе.} \end{cases}$$

Матрица инцидентности ориентированного графа  $G = (X, U)$ , в котором  $|X| = n$ ,  $|U| = m$ , – это матрица  $B = (b_{ij})_{n \times n}$ , где

$$b_{ij} = \begin{cases} 1, & \text{если вершина } x_i \in X \text{ является началом дуги } u_j \in U, \\ -1, & \text{если вершина } x_i \in X \text{ является концом дуги } u_j \in U, \\ 0, & \text{если вершина } x_i \in X \text{ не инцидентна дуге } u_j \in U. \end{cases}$$



$A$	$a$	$b$	$c$	$d$	$e$
$a$	0	1	0	0	1
$b$	0	0	1	1	1
$c$	0	0	0	1	1
$d$	1	0	0	0	1
$e$					

$B$	1	2	3	4	5	6	7	8	9
$a$	-1	1						1	
$b$		-1	1				1		1
$c$			-1	1	1				
$d$	1			-1		1			-1
$e$					-1	-1	-1	-1	

Рисунок 14 – Граф  $G$ ,  
его матрицы смежности ( $A$ )  
и инцидентности ( $B$ )

Если граф неориентированный, то  $b_{ij}$  принимает только два значения 0 и 1 в том случае, если вершина  $x_i$  является концевой вершиной ребра  $u_j$ .

Для орграфа  $\sum_{i=1}^n b_{ij} = 0$  для всех  $j = \overline{1, m}$ . В неорграфе  $\sum_{i=1}^n b_{ij} = 2$  для всех  $j = \overline{1, m}$ .

Ниже перечислены некоторые важные свойства матрица смежности  $A$  простого графа.

1. Для неорграфа матрица смежности является симметричной относительно главной диагонали, для орграфа – антисимметричной, т.е.  $a_{ij} \cdot a_{ji} = 0$  для всех пар индексов  $(i, j)$ .

2. Для орграфа  $(i, j)$ -элемент матрицы  $A^N$  определяет количество путей из вершины  $x_i$  в вершину  $x_j$  длины  $N$ , если под длиной пути подразумевается количество дуг, которые этот путь составляют.

3. Граф является двудольным тогда и только тогда, когда для любого нечетного числа  $N$  все диагональные элементы матрицы  $A^N$  равны 0.

4. Два графа являются изоморфными, тогда и только тогда, когда матрицу смежности одного из них можно получить из матрицы смежности другого графа путем одновременной перестановки строк и столбцов.

5. Чтобы  $n$ -вершинный граф с матрицей смежности  $A$  имел хотя бы один контур, необходимо и достаточно, чтобы матрица  $(A^2 + A^3 + \dots + A^n)$  имела ненулевые диагональные элементы.

6. Вершины бесконтурного орграфа можно упорядочить таким образом, что его матрица смежности будет иметь верхний треугольный вид (для приведения матрицы к треугольному виду используется порядковая функция).

Теперь перечислим некоторые свойства матрицы инцидентности  $B$ .

1. В матрице инцидентности значение любого минора равно 0, 1 или -1.

2. Для связного графа с  $n$  вершинами ранг матрицы инцидентности  $B$  равен  $(n - 1)$ . Ранг матрицы инцидентности произвольного графа равен разности числа вершин и числа компонент связности.

3. Для любого неорграфа матрица смежности  $A$  выражается через матрицу инцидентности  $B$  следующим образом:

$$A = BB^T - \text{diag}(d_1, \dots, d_n),$$

где  $d_i$  – степень вершины  $x_i$ ;  $\text{diag}(d_1, \dots, d_n)$  – матрица размерности  $n \times n$  с элементами  $d_1, \dots, d_n$  на главной диагонали (все остальные элементы равны 0).

### **Связность и достижимость**

Если для любой пары вершин существует цепь их соединяющая, то такой граф называется *связным*, иначе *несвязным*.

Любой простой граф либо сам связан, либо связан его дополнительный граф.

Для орграфов понятие связности является более содержательным. Можно выделить три типа связности орграфа.

Орграф называется *слабо связным*, если его основание есть связный граф; *односторонне связным*, если для любых двух различных вершин  $x_i$  и  $x_j$  существует путь из  $x_i$  в  $x_j$  или из  $x_j$  в  $x_i$ ; *сильно связным*, если для любых двух различных вершин  $x_i$  и  $x_j$  существует путь из  $x_i$  в  $x_j$  и обратно.

Если в графе  $G$  существует путь из вершины  $x_i$  в вершину  $x_j$ , то говорят, что вершина  $x_j$  *достижима из вершины  $x_i$* . Таким образом, орграф является односторонне связным, если для каждой пары вершин выполняется

условие:  $x_i$  достижима из  $x_j$  или наоборот; орграф является сильно связным, если  $x_i$  достижима из  $x_j$  и наоборот, т.е. вершины  $x_i$  и  $x_j$  являются взаимно достижимыми, т.е. лежат на одно контуре. Будем считать, что вершина  $x_i$  достижима из себя. Тогда отношение взаимной достижимости для заданной пары вершин можно рассматривать как отношение эквивалентности (рефлексивное, симметричное, транзитивное).

Свойство связности является наследственным, т.е. в данном графе можно рассматривать подграфы, обладающие теми же типами связности, что и исходный граф.

Пусть  $G$  – заданный граф,  $\alpha$  – тип связности. Максимальный подграф данного графа  $G$ , обладающий типом связности  $\alpha$ ,  $\alpha$ -компонентой, а именно *связной компонентой* или *компонентой связности*, *односторонней компонентой*, *сильной компонентой*.

Рассмотрим задачу определения сильных компонент в связном орграфе  $G = (X, U)$ . Введем некоторые дополнительные определения.

Пусть  $x \in X$  – некоторая вершина графа.

*Множеством достижимости* вершины  $x$  называется множество вершин, которые достижимы из  $x$ , т.е.

$$R(x) = \{y \in X : \text{из } x \text{ в } y \text{ существует путь}\} = \{y \in X : x \rightarrow\rightarrow y\}.$$

*Множество контраст достижимости* определяется двойственным образом – это множество вершин, из которых достижима вершина  $x$ , т.е.

$$Q(x) = \{z \in X : \text{из } z \text{ в } x \text{ существует путь}\} = \{z \in X : z \rightarrow\rightarrow x\}.$$

Будем считать, что  $x \in R(x)$  и  $x \in Q(x)$ . Заметим, что множество  $R(x) \cap Q(x)$  содержит вершину  $x$  и все такие вершины, которые принадлежат различным контурам, включающим вершину  $x$ , т.е. это множество взаимно достижимых вершин, а, следовательно, это сильная компонента (содержащая вершину  $x$ ). В частном случае множество  $R(x) \cap Q(x)$  состоит из единственной вершины  $x$ . Так как свойство взаимной достижимости является эквивалентностью, то вершина  $x$  не может принадлежать другим компонентам, поэтому, если удалить найденную сильную компоненту из графа, то можно продолжить поиск в полученном подграфе.

Сформулируем следующий

#### Алгоритм нахождения сильных компонент

Шаг 1. Пусть  $G = (X, U)$  – данный орграф. Положить  $i = 0$  (счетчик сильных компонент).

Шаг 2. Выбрать произвольную вершину  $x_0 \in X$  и сформировать для нее множества  $R(x_0)$  и  $Q(x_0)$ .

Шаг 3. Положить  $i = i + 1$  и найти сильную компоненту  $CK_i = R(x_0) \cap Q(x_0)$ . Удалить из графа вершины, принадлежащие  $CK_i$  вместе с инцидентными дугами.

Шаг 4. Если полученный подграф пуст, то останов (все сильные компоненты определены), иначе не все вершины распределены по сильный компонентам и следует перейти к шагу 2.

**Пример.** Рассмотрим граф, изображенный на рисунке.

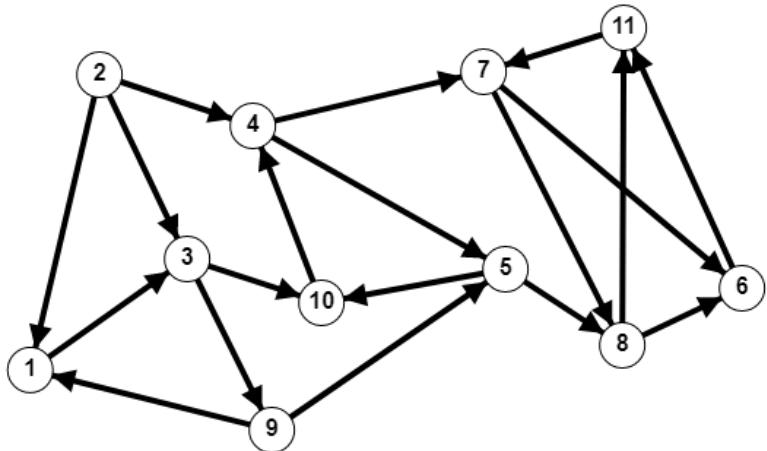


Рисунок 15 – Заданный граф

На первой итерации выберем вершину 6 и определим для нее множество достижимости  $R(6) = \{6, 11, 7, 8\}$  и множество контрдостижимости  $Q(6) = \{6, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11\}$ . Найдем  $CK_1 = \{6, 7, 8, 11\}$  и удалим ее из графа. Получим граф, изображенный на рис. 16.

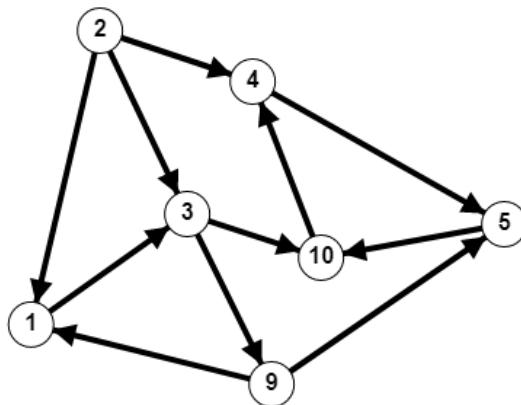


Рисунок 16 – Граф после первой итерации

На второй итерации выберем любую вершину, например, 3. Найдем  $R(3) = \{3, 1, 4, 5, 9, 10\}$  и  $Q(3) = \{3, 1, 2, 9\}$ . Соответствующая сильная компонента будет иметь вид  $CK_2 = \{3, 1, 9\}$ . Вершины  $CK_2$  удалим из графа и получим граф, изображенный на рис. 17.

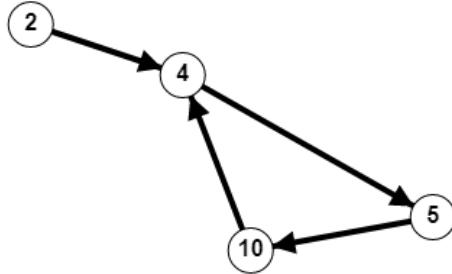


Рисунок 17 – Граф после второй итерации

На третьей итерации выберем вершину 2, для которой  $R(2) = \{2, 4, 5, 10\}$  и  $Q(2) = \{2\}$ , следовательно  $CK_3 = \{2\}$ . Удалив вершину 2 из графа, получим граф, изображенный на рисунке 18.

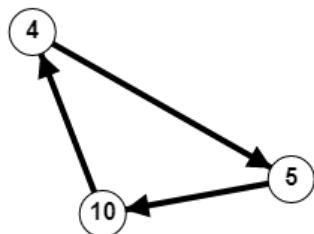


Рисунок 18 – Граф после третьей итерации

Видно, что все вершины принадлежат контуру и образуют сильную компоненту. С другой стороны, можно действовать согласно алгоритму.

На четвертой итерации выберем вершину 4. Для нее  $R(4) = \{4, 5, 10\}$  и  $Q(4) = \{4, 5, 10\}$ , откуда получим  $CK_4 = \{4, 5, 10\}$ . Удаляя эти вершины, получим пустой граф, и алгоритм завершает работу.

Таким образом, заданный граф содержит следующие сильные компоненты:  $CK_1 = \{6, 7, 8, 11\}$ ,  $CK_2 = \{3, 1, 9\}$ ,  $CK_3 = \{2\}$ ,  $CK_4 = \{4, 5, 10\}$ . Заметим, что они образуют разбиение множества вершин графа, а вершины, входящие в одну компоненту, принадлежат одному классу эквивалентности.

## Лекция 2

### Выявление структуры ориентированного графа

Существуют два вида связности – вершинная и реберная. Число вершинной связности – это наименьшее число вершин, удаление которых вместе с инцидентными ребрами приводит к несвязному графу. Число реберной связности – это наименьшее число ребер, удаление которых приводит к несвязному графу. Рассмотрим более подробно эти понятия.

Если для любой пары вершин существует цепь их соединяющая, то такой граф называется *связным*, иначе *несвязным*.

Любой простой граф либо сам связан, либо связан его дополнительный граф.

Для орграфов понятие связности является более содержательным. Можно выделить три типа связности орграфа.

Орграф называется *слабо связным*, если его основание есть связный граф; *односторонне связным*, если для любых двух различных вершин  $x_i$  и  $x_j$  существует путь из  $x_i$  в  $x_j$  или из  $x_j$  в  $x_i$ ; *сильно связным*, если для любых двух различных вершин  $x_i$  и  $x_j$  существует путь из  $x_i$  в  $x_j$  и обратно.

Если в графе  $G$  существует путь из вершины  $x_i$  в вершину  $x_j$ , то говорят, что вершина  $x_j$  *достижима из вершины  $x_i$* . Таким образом, орграф является односторонне связным, если для каждой пары вершин выполняется условие:  $x_i$  достижима из  $x_j$  или наоборот; орграф является сильно связным, если  $x_i$  достижима из  $x_j$  и наоборот, т.е. вершины  $x_i$  и  $x_j$  являются взаимно достижимыми, т.е. лежат на одно контуре. Будем считать, что вершина  $x_i$  достижима из себя. Тогда отношение взаимной достижимости для заданной пары вершин можно рассматривать как отношение эквивалентности (рефлексивное, симметричное, транзитивное).

Пусть  $G$  – заданный граф,  $\alpha$  – тип связности. Максимальный подграф данного графа  $G$ , обладающий типом связности  $\alpha$ ,  $\alpha$ -компонентой, а именно *связной компонентой* или *компонентой связности*, *односторонней компонентой*, *сильной компонентой*.

Каждый граф представляется в виде объединения своих связных компонент. Разложение графа на связные компоненты определено однозначно. Односторонняя компонента представляет собой односторонне связный максимальный подграф, а *сильная компонента* (СК) максимальный сильно связный подграф графа  $G$ . Иначе сильная компонента графа называется *бикомпонентой*, поскольку она состоит из вершин, которые являются взаимно достижимыми. Из определений следует, что односторонние компоненты графа могут иметь общие вершины, сильная компонента должна содержаться, по крайней мере, в одной односторонней компоненте, а односторонняя компонента содержится в некоторой связной компоненте данного графа  $G$ . Очевидно, что всякий сильно связный граф одновременно является односторонне связным и слабо связным. Всякий односторонне связный граф в то же время

является и слабо связным. Обратные утверждения не верны. В орграфе каждая вершина входит в одну и только в одну сильную компоненту.

Рассмотрим задачу определения сильных компонент в связном орграфе  $G = (X, U)$ . Введем некоторые дополнительные определения.

Пусть  $x \in X$  – некоторая вершина графа.

*Множеством достижимости* вершины  $x$  называется множество вершин, которые достижимы из  $x$ , т.е.

$$R(x) = \{y \in X : \text{из } x \text{ в } y \text{ существует путь}\} = \{y \in X : x \rightarrow\rightarrow y\}.$$

*Множество контрдостижимости* определяется двойственным образом – это множество вершин, из которых достижима вершина  $x$ , т.е.

$$Q(x) = \{z \in X : \text{из } z \text{ в } x \text{ существует путь}\} = \{z \in X : z \rightarrow\rightarrow x\}.$$

Будем считать, что  $x \in R(x)$  и  $x \in Q(x)$ . Заметим, что множество  $R(x) \cap Q(x)$  содержит вершину  $x$  и все такие вершины, которые принадлежат различным контурам, включающим вершину  $x$ , т.е. это множество взаимно достижимых вершин, а, следовательно, это сильная компонента (содержащая вершину  $x$ ). В частном случае множество  $R(x) \cap Q(x)$  состоит из единственной вершины  $x$ . Так как свойство взаимной достижимости является эквивалентностью, то вершина  $x$  не может принадлежать другим компонентам, поэтому, если удалить найденную сильную компоненту из графа, то можно продолжить поиск в полученном подграфе.

### Алгоритм нахождения сильных компонент на основе множеств достижимости и контрдостижимости

Шаг 1. Пусть  $G = (X, U)$  – данный орграф. Положить  $i = 0$  (счетчик сильных компонент).

Шаг 2. Выбрать произвольную вершину  $x_0 \in X$  и сформировать для нее множества  $R(x_0)$  и  $Q(x_0)$ .

Шаг 3. Положить  $i = i + 1$  и найти сильную компоненту  $CK_i = R(x_0) \cap Q(x_0)$ . Удалить из графа вершины, принадлежащие  $CK_i$  вместе с инцидентными дугами.

Шаг 4. Если полученный подграф пуст, то останов (все сильные компоненты определены), иначе не все вершины распределены по сильный компонентам и следует перейти к шагу 2.

**Пример.** Рассмотрим граф, изображенный на рисунке.

На первой итерации выберем вершину 6 и определим для нее множество достижимости  $R(6) = \{6, 11, 7, 8\}$  и множество контрдостижимости  $Q(6) = \{6, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11\}$ . Найдем  $CK_1 = \{6, 7, 8, 11\}$  и удалим ее из графа. Получим граф, изображенный на рис. 1.

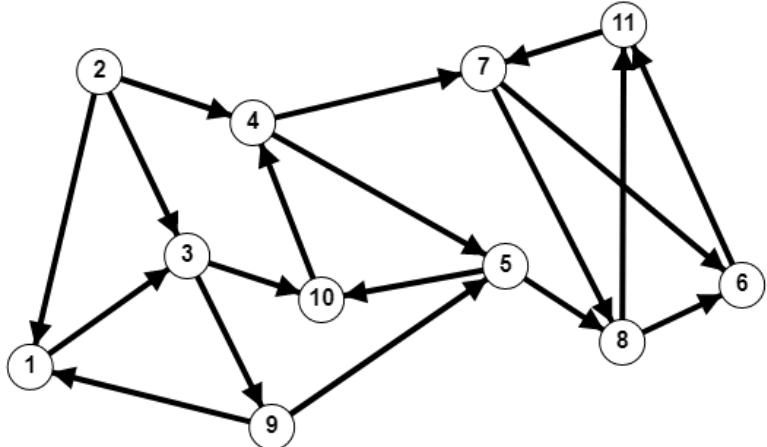


Рисунок 1 – Заданный граф

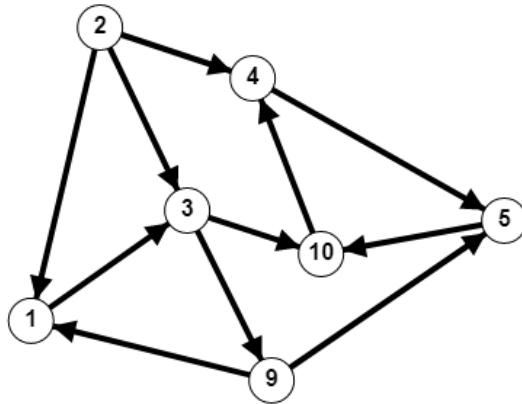


Рисунок 2 – Граф после первой итерации

На второй итерации выберем любую вершину, например, 3. Найдем  $R(3) = \{3, 1, 4, 5, 9, 10\}$  и  $Q(3) = \{3, 1, 2, 9\}$ . Соответствующая сильная компонента будет иметь вид  $CK_2 = \{3, 1, 9\}$ . Вершины  $CK_2$  удалим из графа и получим граф, изображенный на рис. 3.

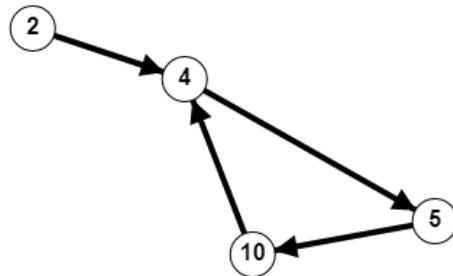


Рисунок 3 – Граф после второй итерации

На третьей итерации выберем вершину 2, для которой  $R(2) = \{2, 4, 5, 10\}$  и  $Q(2) = \{2\}$ , следовательно  $CK_3 = \{2\}$ . Удалив вершину 2 из графа, получим граф, изображенный на рисунке 4.

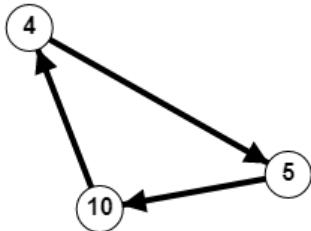


Рисунок 4 – Граф после третьей итерации

Видно, что все вершины принадлежат контуру и образуют сильную компоненту. С другой стороны, можно действовать согласно алгоритму.

На четвертой итерации выберем вершину 4. Для нее  $R(4) = \{4, 5, 10\}$  и  $Q(4) = \{4, 5, 10\}$ , откуда получим  $CK_4 = \{4, 5, 10\}$ . Удаляя эти вершины, получим пустой граф, и алгоритм завершает работу.

Таким образом, заданный граф содержит следующие сильные компоненты:  $CK_1 = \{6, 7, 8, 11\}$ ,  $CK_2 = \{3, 1, 9\}$ ,  $CK_3 = \{2\}$ ,  $CK_4 = \{4, 5, 10\}$ . Заметим, что они образуют разбиение множества вершин графа, а вершины, входящие в одну компоненту, принадлежат одному классу эквивалентности.

*Матрицей достижимости* орграфа  $G$  называется матрица  $R = (r_{ij})_{n \times n}$ , где

$$r_{ij} = \begin{cases} 1, & \text{если существует путь из } x_i \text{ в } x_j, \\ 0, & \text{иначе.} \end{cases}$$

В матрице контродостижимости  $Q = (q_{ij})_{n \times n}$

$$q_{ij} = \begin{cases} 1, & \text{если существует путь из } x_j \text{ в } x_i, \\ 0, & \text{иначе,} \end{cases}$$

поэтому  $Q = R^T$ .

Матрица достижимости несет важную информацию об орграфе. Ее анализ позволяет найти сильные компоненты графа, в которые входят взаимно достижимые вершины. Для двух таких вершин с номерами  $i$  и  $j$  должно выполняться равенство  $r_{ij} = r_{ji} = 1$ , поэтому, чтобы найти сильную компоненту, в которую входит  $i$ -я вершина орграфа, нужно просмотреть  $i$ -ю строку и  $i$ -й столбец матрицы достижимости  $R$  и сформировать множество  $P_i = \{j : r_{ij} = r_{ji} = 1\}$  номеров вершин, порождающих искомую сильную компоненту компоненту. Поскольку две различные сильные компоненты не пересекаются, вершины с номерами из множества  $P_i$  при поиске других сильных компонент можно исключить из рассмотрения. Процесс поиска начинается с произвольной вершины и заканчивается, когда для каждой вершины будет найдена содержащая ее сильная компонента. Может оказаться, что некоторые (а может быть и все) сильные компоненты содержат только по одной

вершине, поскольку каждая вершина, по определению, достижима сама из себя.

*Алгоритм нахождения сильных компонент на основе матриц достижимости и контрдостигимости:*

Шаг 1. Пусть  $G = (X, U)$  – данный орграф. Для него определить матрицу достижимости  $R$  и матрицу контрдостигимости  $Q = R^T$ .

Шаг 2. Построить матрицу  $C = (c_{ij})_{n \times n} = R \times Q$ , где  $c_{ij} = r_{ij} \cdot q_{ij}$ .

Шаг 3. Преобразовать матрицу  $C$  к блочно-диагональному виду путем одновременной перестановки строк и столбцов. Каждая из диагональных подматриц соответствует сильной компоненте графа  $G$ . Останов.

Рассмотрим данный алгоритм на примере.

Пример. На рис. 5 представлен заданный граф. Здесь ребра соответствуют двум противоположно направленным дугам

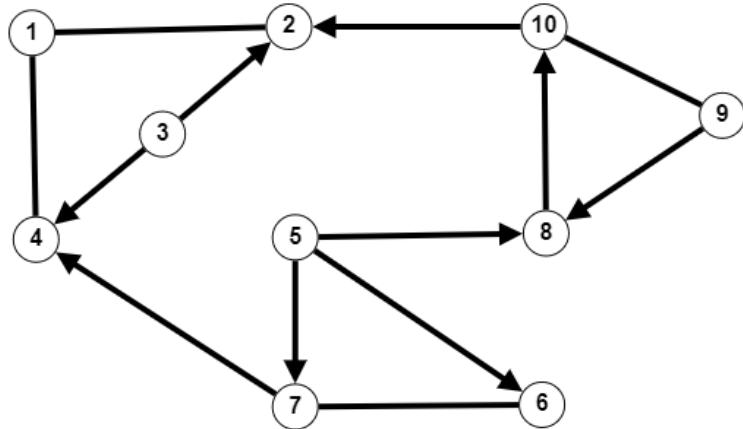


Рисунок 5 – Ориентированный граф, содержащий 10 вершин

Для данного графа матрица достижимости  $R$  имеет вид.

$R$	1	2	3	4	5	6	7	8	9	10
1	1	1	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0	0	0
3	1	1	1	1	0	0	0	0	0	0
4	1	1	0	1	0	0	0	0	0	0
5	1	1	0	1	1	1	1	1	1	1
6	1	1	0	1	0	1	1	0	0	0
7	1	1	0	1	0	1	1	0	0	0
8	1	1	0	1	0	0	0	1	1	1
9	1	1	0	1	0	0	0	1	1	1
10	1	1	0	1	0	0	0	1	1	1

По матрице  $R$  определяется матрица контрдостижимости  $Q = R^T$ . Матрица  $C = R \times Q$  имеет следующий вид.

$C$	1	2	3	4	5	6	7	8	9	10
1	1	1	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	1	1	0	1	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	1	1	0	0	0
7	0	0	0	0	0	1	1	0	0	0
8	0	0	0	0	0	0	0	1	1	1
9	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	1	1	1

Группируя одинаковые строки, а затем, взяв в том же порядке столбцы получим блочно-диагональный вид матрицы  $C$ .

$C$	1	2	4	3	5	6	7	8	9	10
1	1	1	1	0	0	0	0	0	0	0
2	1	1	1	0	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	1	1	0	0	0
7	0	0	0	0	0	1	1	0	0	0
8	0	0	0	0	0	0	0	1	1	1
9	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	1	1	1

Каждый блок из единиц соответствует сильной компоненте графа  $G$ :

$$CK_1 = \{1, 2, 4\}, CK_2 = \{3\}, CK_3 = \{5\}, CK_4 = \{6, 7\}, CK_5 = \{8, 9, 10\}.$$

Таким образом, для сильно связного графа существует разбиение множества вершин на непересекающиеся подмножества, каждое из которых образует сильную компоненту исходного графа. Тогда очевидно, что на множестве сильных компонент можно установить некоторое отношение порядка, причем в качестве модели такого отношения выступают бесконтурные графы.

В сильно связном графе можно выделить следующие экстремальные подграфы. *Базой* графа  $G$  называется минимальный по включению подграф  $B$ , из которого достижима любая вершина графа  $G$ . Здесь минимальность означает, что ни из какого собственного подмножества  $B$  нельзя достичь всех оставшихся вершин. *Антибазой* графа  $G$  называется минимальный подграф  $\bar{B}$ , такой что, какова бы ни была вершина графа  $G$ , из нее достижима неко-

торая вершина в  $\bar{B}$ . Множество вершин, которое одновременно является базой и антибазой, называется *дивазой*.

Определив в графе сильные компоненты, можно перейти к бесконурному графу, который называется конденсацией.

Пусть  $\{CK_1, \dots, CK_N\}$  – множество сильных компонент графа  $G = (X, U)$ . Граф  $G^*$  называется *конденсацией* графа  $G$ , если каждая его вершина соответствует сильной компоненте графа  $G$ , а дуга  $(CK_i, CK_j)$  существует в  $G^*$  тогда и только тогда, когда в графе  $G$  существует дуга  $(x, y) \in U$ , причем  $x \in CK_i$  и  $y \in CK_j$ .

Поскольку конденсация представляет собой бесконтурный граф, то ей присущи следующие замечательные свойства: существует хотя бы одна вершина без входящих дуг и хотя бы одна вершина без выходящих дуг; бесконтурный граф можно разложить на уровни и представить в виде иерархии, а на основе данного разложения можно осуществить топологическую сортировку, что позволит построить некоторое ранжирование, упорядочивающее сильные компоненты.

Имеют место следующие утверждения:

- конденсация содержит единственную базу, состоящую из вершин без входящих дуг;
- конденсация содержит единственную антибазу, состоящую из вершин без выходящих дуг;
- пусть  $B^*$  – единственная база конденсации  $G^*$  орграфа  $G$ , тогда вершинными базами в  $G$  являются такие множества  $B$ , которые содержат по одной вершине из каждой сильной компоненты, принадлежащей  $B^*$ .
- пусть  $\bar{B}^*$  – единственная антибаза конденсации  $G^*$  орграфа  $G$ , тогда вершинными антибазами в  $G$  являются такие множества  $\bar{B}$ , которые содержат по одной вершине из каждой сильной компоненты, принадлежащей  $\bar{B}^*$ .
- любые две базы (антибазы) содержат одинаковое количество вершин.

Приведенные утверждения позволяют сформулировать следующие алгоритмы.

#### Алгоритм для нахождения баз и антибаз орграфа

Шаг 1. Для заданного графа  $G$  определить все сильные компоненты.

Шаг 2. Построить конденсацию  $G^*$  графа  $G$ .

Шаг 3. Определить базу  $B^*$  конденсации  $G^*$ , включив в нее те вершины  $G^*$ , которые не имеют входящих дуг (их полустепени захода равны 0).

Шаг 4. Перечислить базы  $B$  графа  $G$ , включив в каждую по одной вершине из сильных компонент, входящих в  $B^*$ .

Шаг 5. Определить антибазу  $\bar{B}^*$  конденсации  $G^*$ , включив в нее те вершины  $G^*$ , которые не имеют выходящих дуг (их полустепени исхода равны 0).

Шаг 6. Перечислить базы  $\bar{B}$  графа  $G$ , включив в каждую по одной вершине из сильных компонент, входящих в  $\bar{B}^*$ .

Пример. Построим базы и антибазы для графа из предыдущего примера. Здесь найдены следующие сильные компоненты:

$$CK_1 = \{1, 2, 4\}, CK_2 = \{3\}, CK_3 = \{5\}, CK_4 = \{6, 7\}, CK_5 = \{8, 9, 10\}.$$

Граф конденсации, разложенный на уровни, изображен на рис. 6.

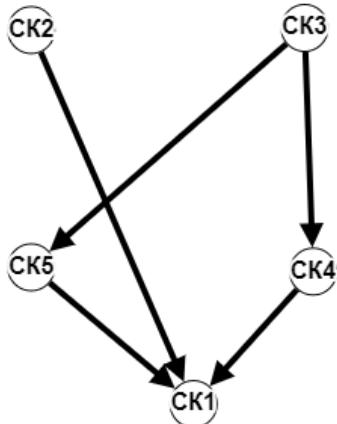


Рисунок 6 – Граф конденсации  $G^*$  для примера 1.

Вершины, соответствующие сильным компонентам  $CK_2$  и  $CK_3$ , не имеют входящих дуг. Следовательно, они образуют базу  $B^* = \{CK_2, CK_3\}$  конденсации, тогда исходный граф  $G$  имеет базу  $B = \{x_3, x_5\}$  – это минимальное множество вершин, из которых достижимы все остальные вершины графа. Вершина  $CK_1$  не имеет выходящих дуг, поэтому  $\bar{B}^* = \{CK_1\}$ . Тогда для исходного графа имеем три антибазы  $\bar{B}_1 = \{x_1\}$ ,  $\bar{B}_2 = \{x_2\}$ ,  $\bar{B}_3 = \{x_4\}$ .

### Зачетная задача 1.

Имеет место утверждение: если  $x_i$  такая вершина графа  $G = (X, U)$ , на которой достигается  $\max_{x_i \in X} |R(x_i)|$ , то  $x_i \in B$ . Сконструируйте алгоритм нахождения баз, который использует данный факт. Можно ли сформулировать аналогичное утверждение для антибаз? Свои рассуждения подкрепите примерами.

### Зачетная задача 2.

Предложите алгоритм для определения компонент реберной связности, числа реберной связности. Как в этом случае интерпретируются результаты?

**Зачетная задача 3.** Построить граф (не менее 10 вершин, 20 дуг), являющийся моделью некоторой прикладной задачи (например, это может быть граф предпочтений; граф, отражающий взаимоотношения в некотором коллективе, например, в социальной сети и т.п.). Провести анализ данного графа (определить сильные компоненты, базы, антибазы; построить иерархию) и интерпретировать полученные результаты.

## ТЕМА: Степенные последовательности

Рассмотрим орграф  $G = (X, U)$  на  $n = |X|$  вершинах и отображение  $\Gamma: X \rightarrow X$ , такое, что для любой вершины  $x_i \in X$  можно определить множество  $\Gamma(x_i) = \{x_j : (x_i, x_j) \in U\}$ , называемое *окрестностью* вершины  $x_i$ .

Для отображения  $\Gamma$  можно определить обратное отображение  $\Gamma^{-1}: X \rightarrow X$ , такое что для любой вершины  $x_i \in X$   $\Gamma^{-1}(x_i) = \{x_k : (x_k, x_i) \in U\}$  – множество вершин, из которых дуги ведут в вершину  $x_i$ .

*Полустепенью захода* вершины  $x_i$  называется число  $d^+(x_i) = |\Gamma(x_i)|$  – количество дуг, входящих в вершину  $x_i$ .

*Полустепенью исхода* вершины  $x_i$  называется число  $d^-(x_i) = |\Gamma^{-1}(x_i)|$  – количество дуг, выходящих из вершины  $x_i$ .

*Степенью*  $d_i$  вершины  $x_i \in X$  орграфа  $G$  называется величина  $d(x_i) = d^+(x_i) + d^-(x_i)$ .

Пусть  $G = (X, U)$  – неорграф на  $n = |X|$  вершинах.

*Степенью*  $d_i$  вершины  $x_i \in X$  называется количество ребер, для которых эта вершина является концевой.

Последовательность  $(d_1, \dots, d_n)$  называется *степенной последовательностью графа*  $G$ ,

Имеют место следующие утверждения:

1. В бесконтурном графе существует хотя бы одна вершина  $x_i$  с нулевой полустепенью захода  $d^+(x_i)$  и хотя бы одна вершина с нулевой полустепенью исхода  $d^-(x_i)$ , причем это свойство является наследственным, т.е. после удаления любой вершины вместе с инцидентными дугами, оно сохраняется.
2. Пусть  $G = (X, U)$  – орграф с  $n = |X|$  ребрами и  $m = |U|$  дугами, тогда

$$\sum_{i=1}^n d^+(x_i) = \sum_{i=1}^n d^-(x_i) = m.$$

3. Если  $G = (X, U)$  – простой неорграф с  $n$  вершинами и  $m$  ребрами, то

$$\sum_{i=1}^n d(x_i) = 2m.$$

(Данное утверждение является исторически первой теоремой теории графов и доказано Эйлером).

4. В любом простом неорграфе число вершин с нечетной степенью четно (следствие из 3).

Замечание: Из 3) следует, что для связного неорграфа  $2m \in \mathbb{N}$ . Разбиение графа – это представление числа  $2m$  в виде суммы степеней вершин графа. Разбиение графа существует не всегда. Например,

$$4 = (4 + 0, 3 + 1, 2 + 2, 2 + 1 + 1, 1 + 1 + 1 + 1),$$

однако, только для двух последних сумм существуют графы (какие?).

Пусть задана последовательность неотрицательных чисел  $d_1, \dots, d_n$ . Данная последовательность называется *правильной*, если

a)  $d_1 \geq d_2 \geq \dots \geq d_n$ ,

б)  $\sum_{i=1}^n d_i$  – четное число.

Последовательность  $d = (d_1, \dots, d_n)$  называется *графической*, если существует граф, степенная последовательность которого совпадает с  $d$ . Заметим, что всякую графическую последовательность можно представить в виде правильной. Кроме того, в ней должны быть равные члены (если  $n > 1$ ), поскольку не существует графа, степени всех вершин которого попарно различны. Указанные условия не являются достаточными для графичности последовательности (например, последовательность  $(3, 3, 1, 1)$  не является графической, хотя и удовлетворяет перечисленным выше условиям).

В общем случае графическая последовательность имеет много реализаций и их число определить сложно.

Пусть  $d = (d_1, \dots, d_n)$  – правильная последовательность. Зафиксируем индекс  $i$ . Предположим, что  $d_i = k$ . Сформируем новую последовательность по правилам:

а) удалим из последовательности  $d$  член  $d_i$ ;

б) вычтем из оставшихся первых  $k$  членов по 1.

Полученную последовательность обозначим через  $d'(i) = (d'_1, \dots, d'_{n-1})$  и назовем *остаточной*.

Имеют место следующие утверждения:

1. Если последовательность  $d = (d_1, \dots, d_n)$  графическая, то каждая из остаточных последовательностей  $d'(i)$  также является графической.
2. Пусть  $d = (d_1, \dots, d_n)$  – правильная последовательность,  $n > 1$ . Если для какого-либо индекса  $i (i = \overline{1, n})$  остаточная последовательность  $d'(i)$  является графической, то и  $d$  графическая последовательность.

Пусть  $d = (d_1, \dots, d_n)$  – правильная последовательность,  $n > 1$ . Рассмотрим вполне несвязный граф, содержащий  $n$  вершин  $x_1, \dots, x_n$ . С каждой вершиной свяжем член последовательности  $d$ .

Введем понятие *d-процедуры*:

Шаг 1. Выбрать ведущую вершину  $j$  с положительной меткой  $d_j = k$  и сформировать множество  $S(j)$  – вершин, которые соответствуют первым  $k$  членам остаточной последовательности, не считая вершину  $j$ .

Шаг 2. Вершину  $j$  соединить ребром с каждой вершиной из множества  $S(j)$ .

Шаг 3. Изменить метки вершин следующим образом:

$$d_j \rightarrow 0,$$

$$\text{для каждой вершины } u \in S(j) (d_u \rightarrow d_u - 1).$$

Замечание: если в результате применения данной процедуры какая-либо из меток становится отрицательной, то последовательность не является графической.

Рассмотрим несколько критериев графичности последовательности.

Критерий для построения регулярного графа степени  $r$ : пусть  $r$  – степень вершины регулярного графа  $G = (X, U)$ ,  $|X| = n$ ,  $|U| = m$ . При выполнении условия  $nr = 2m$  всегда можно построить такой граф с помощью d-процедуры.

Критерий для построения связного графа: правильная последовательность  $(d_1, \dots, d_n)$  может быть реализована связным графом тогда и только то-

гда, когда  $d_i \geq 1$  для всех  $i = \overline{1, n}$  и  $\sum_{i=1}^n d_i \geq 2(n-1)$ , при этом на каждом шаге в

качестве ведущей вершины следует выбирать вершину с минимальной положительной степенью.

Критерий для построения дерева: последовательность степеней  $(d_1, \dots, d_n)$  является последовательностью степеней дерева тогда и только то-

гда, когда  $d_i \geq 1$  для всех  $i = \overline{1, n}$  и  $\sum_{i=1}^n d_i = 2(n-1)$ , при этом на каждом шаге в

качестве ведущей вершины следует выбирать вершину с минимальной положительной степенью.

Критерий для построения гамильтонова графа: если существует графическая реализация правильной последовательности  $(d_1, \dots, d_n)$  с гамильтоновым путем, проходящим через вершину степени  $d_i$ , то к такой реализации приведет d-процедура, на первом шаге которой выбирается вершина степени  $d_i > 1$ , а на каждом из последующих – вершина с минимальной положительной остаточной степенью из множества  $\Gamma(v)$ , где  $v$  – ведущая вершина на предыдущем шаге.

Пример 1. Пусть задана последовательность  $(5, 3, 3, 2, 2, 1)$ . Так как  $16 > 10$ , то последовательность может быть реализована связным графом. Расчеты приведены в следующей таблице (серым выделен ведущий элемент).

Номер итерации	Вершины упорядочиваются на каждой итерации по убыванию степеней; столбцы, соответствующие вершинам с нулевыми степенями, располагаются в произвольном порядке.					
1	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
	5	3	3	2	2	1
	4	3	3	2	2	0
2	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
	4	3	3	2	2	0
	3	2	3	0	2	0
3	$x_1$	$x_3$	$x_2$	$x_5$	$x_6$	$x_4$
	3	3	2	2	0	0
	2	2	2	0	0	0
4	$x_1$	$x_3$	$x_2$	$x_6$	$x_4$	$x_5$
	2	2	2	0	0	0
	1	1	0	0	0	0
5	$x_1$	$x_3$	$x_6$	$x_4$	$x_5$	$x_2$
	1	1	0	0	0	0
	0	0	0	0	0	0
Признак останова – все 0!						

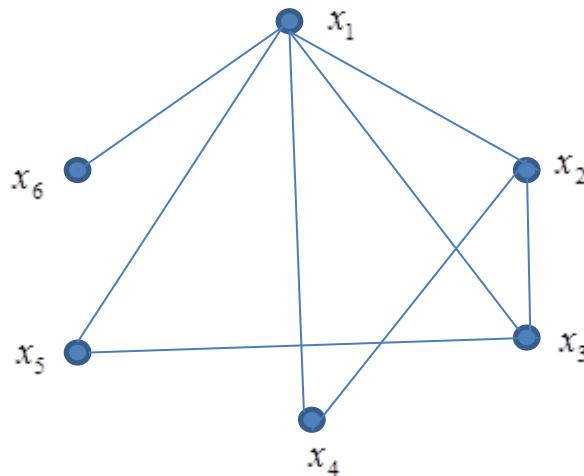


Рис. 1 – Связный граф для заданной графической последовательности

Пример 2. Пусть задана последовательность  $(3, 3, 2, 2, 2, 2)$ . Так как  $14 > 10$ , то последовательность может быть реализована связным графом. Попробуем построить гамильтонов граф. Расчеты приведены в следующей таблице (серым выделен ведущий элемент  $v$ , темно-серым – элементы множества  $\Gamma(v)$ ).

Номер итерации	Вершины упорядочиваются на каждой итерации по убыванию степеней; столбцы, соответствующие вершинам с нулевыми степенями располагаются в произвольном порядке.					
<b>1</b>	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
	3	3	2	2	2	2
	2	2	2	2	2	0
<b>2</b>	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
	2	2	2	2	2	0
	0	1	1	2	2	0
<b>3</b>	$x_4$	$x_5$	$x_2$	$x_3$	$x_1$	$x_6$
	2	2	1	1	0	0
	1	2	0	1	0	0
<b>4</b>	$x_5$	$x_4$	$x_3$	$x_2$	$x_1$	$x_6$
	2	1	1	0	0	0
	1	0	1	0	0	0
<b>5</b>	$x_5$	$x_3$	$x_4$	$x_2$	$x_1$	$x_6$
	1	1	0	0	0	0
	0	0	0	0	0	0
Признак останова – все 0!						

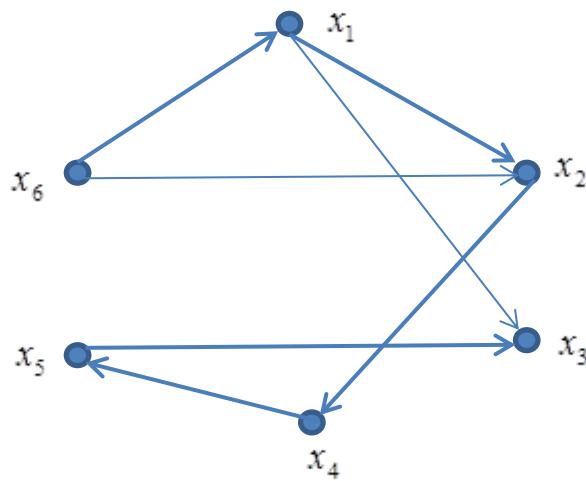


Рисунок 2 – Гамильтонов граф для заданной графической последовательности

На рисунке жирной линией выделен гамильтонов путь (каждая вершина проходится только один раз) с началом в вершине  $x_6$ , которая выбиралась в качестве ведущей на первом шаге. Если на каком-то шаге получили ли бы отрицательные  $d_i$ , то гамильтонова реализация не существует.

## ЛЕКЦИЯ

### УСТОЙЧИВЫЕ МНОЖЕСТВА В ГРАФЕ

К устойчивым множествам в графе относятся внешне устойчивые (доминирующие) и внутренне устойчивые (независимые) множества. Ядро – множество вершин, которое является одновременно минимальным доминирующим и максимальным независимым. Устойчивые множества и связанные с ними числовые характеристики описывают важные структурные свойства графа и имеют разнообразные приложения при ведении проектного планирования исследовательских работ, в кластерном анализе, параллельных вычислениях на ЭВМ, при размещении предприятий обслуживания, а также источников и потребителей в энергосистемах.

Пусть  $G = (X, U)$  – простой орграф (без петель и кратных дуг).

Подмножество вершин  $S \subseteq X$  графа  $G$  называется *независимым* (*внутренне устойчивым*), если никакие две вершины в нем не являются смежными, т. е.

$$\forall x \in S \left( \Gamma(x) \cap S = \emptyset \right).$$

Напомним, что для каждой вершины  $\Gamma(x) = \{y : (x, y) \in U\}$  – множество вершин, в которые ведут дуги из заданной вершины  $x \in X$ .

Независимое множество называется *максимальным*, если оно не является собственным подмножеством никакого другого независимого множества.

Пусть  $\theta_s$  – семейство максимальных независимых множеств, тогда число

$$\alpha(G) = \max_{S \in \theta_s} |S|$$

называется *числом независимости*.

Таким образом,  $\alpha(G)$  – наибольшее число вершин, которые являются попарно не смежными.

Независимое множество вершин графа имеет следующую матричную интерпретацию: пусть  $S$  – независимое множество вершин графа  $G$ ,  $A$  – матрица смежности этого графа, тогда множеству  $S$  в  $A$  соответствует подматрица их нулевых элементов.

Понятием, противоположным внутренне устойчивому множеству, является понятие клики.

Максимальный полный подграф графа  $G$  называется *кликой*.

Связь между наибольшими независимыми множествами вершин и наибольшими кликами устанавливается с помощью дополнительных графов: максимальное независимое множество графа  $G$  соответствует клике графа  $\bar{G}$  и наоборот.

Пусть  $\theta_k$  – множество клик данного графа  $G$ . Число

$$\rho(G) = \max_{K \in \theta_k} |K|$$

называется *кликовым числом*.

Таким образом, кликовое число – максимальное число вершин в кликах данного графа.

*Графом клик* данного графа  $G$  называется граф, в котором вершинам соответствуют клики графа  $G$ , а ребро существует, если в соответствующих кликах найдутся смежные вершины.

Подмножество вершин  $D \subseteq X$  называется *доминирующим* (внешне устойчивым), если для каждой вершины, не входящей в него, существует дуга с начальной вершиной в этом множестве, т.е.

$$\forall x \in X \setminus D (\Gamma^{-1}(x) \cap D \neq \emptyset).$$

Доминирующее множество называется *минимальным*, если оно не содержит никакого другого доминирующего множества.

Пусть  $\theta_D$  – семейство доминирующих множеств заданного графа  $G$ , число

$$\beta(G) = \min_{D \in \theta_D} |D|$$

называется *числом доминирования* графа  $G$ .

Таким образом, число доминирования – это мощность наименьшего доминирующего множества.

Для нахождения устойчивых множеств используется алгоритм Magu, основанный на построении специальной логической функции и анализе ее дизъюнктивной нормальной формы (ДНФ).

### Алгоритм Magu для определения независимых и доминирующих множеств

Шаг 1. Пусть  $G = (X, U)$  – заданный орграф с множеством вершин  $X = \{1, \dots, n\}$ ,  $A = \{a_{ij}\}_{n \times n}$  – матрица смежности графа  $G$ . Вершине с номером  $j$  поставить в соответствие булеву переменную  $x_j$ .

Шаг 2. На основе матрицы смежности построить логические (булевы) функции следующего вида

➤ для определения максимальных независимых множеств (МНМ)

$$\Phi_{MNM}(x_1, \dots, x_n) = \bigwedge_{\{(i,j): a_{ij}=1\}} (\bar{x}_i \vee \bar{x}_j),$$

➤ для определения минимальных доминирующих множеств (МДМ)

$$\Phi_{MDM}(x_1, \dots, x_n) = \bigwedge_{i=1}^n \left( x_i \vee \left( \bigvee_{\{j: a_{ji}=1\}} x_j \right) \right).$$

Шаг 3. Привести полученное на предыдущем шаге выражение к ДНФ и сократить ее, применяя законы поглощения ( $u \vee uv = u$ ) и идемпотентности ( $u \wedge u = u, u \vee u = u$ ).

Шаг 4. Пусть  $\Phi = \bigvee_{r=1}^p K_r$ , где  $K_r$  – элементарная конъюнкция. Для каждого  $r = \overline{1, p}$  найти множества

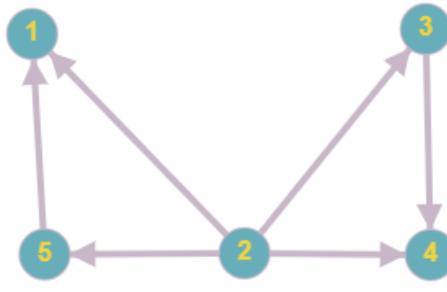
- при определении максимальных независимых множеств

$$S_r = \{j : \bar{x}_j \text{ не входит в } K_r\};$$

- при определении минимальных доминирующих множеств

$$D_r = \{j : x_j \text{ входит в } K_r\}.$$

**Пример.** Найдем МДМ и МНМ для графа с матрицей смежности  $A$ .



$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Для нахождения МНМ составим логическую функцию

$$\begin{aligned} \Phi_{MNM} = & (\bar{x}_2 \vee \bar{x}_1) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee \bar{x}_5) \wedge \\ & \wedge (\bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_3 \vee \bar{x}_1). \end{aligned}$$

Приведем полученную функцию к ДНФ:

$$\begin{aligned} \Phi_{MNM} = & (\bar{x}_2 \vee \bar{x}_1 \bar{x}_3 \bar{x}_4 \bar{x}_5) \wedge (\bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_5 \vee \bar{x}_1) = (\bar{x}_2 \bar{x}_3 \vee \bar{x}_2 \bar{x}_4 \vee \bar{x}_1 \bar{x}_3 \bar{x}_4 \bar{x}_5) \wedge \\ & \wedge (\bar{x}_5 \vee \bar{x}_1) = \bar{x}_2 \bar{x}_3 \bar{x}_5 \vee \bar{x}_2 \bar{x}_3 \bar{x}_1 \vee \bar{x}_2 \bar{x}_4 \bar{x}_5 \vee \bar{x}_2 \bar{x}_4 \bar{x}_1 \vee \bar{x}_1 \bar{x}_3 \bar{x}_4 \bar{x}_5. \end{aligned}$$

Заметим, что полученную ДНФ сократить не возможно, поэтому на ее основе можно выписать следующие МНМ:

$$S_1 = \{1, 4\}, S_2 = \{4, 5\}, S_3 = \{1, 3\}, S_4 = \{3, 5\}, S_5 = \{2\}.$$

Теперь найдем МДМ. Составим логическую функцию

$$\Phi_{MDM} = (x_1 \vee x_2 \vee x_5) \wedge x_2 \wedge (x_3 \vee x_2) \wedge (x_4 \vee x_2 \vee x_3) \wedge (x_5 \vee x_2).$$

Применяя закон поглощения, получим

$$\Phi_{MDM} = \{x_2\}.$$

Таким образом, в графе существует единственное доминирующее множество  $D_1 = \{2\}$ .

Замечание. Метод Магу без изменения переносится на неорграфы и псевдографы с петлями и кратными дугами, при этом можно легко подкорректировать формулы.

Ядром графа  $G$  называется подмножество вершин  $Y$ , которое одновременно является доминирующим и независимым, т.е. выполняются условия:

$$\forall x \in Y (\Gamma(x) \cap Y = \emptyset),$$

$$\forall x \in X \setminus Y \left( \Gamma^{-1}(x) \cap Y \neq \emptyset \right).$$

◆ **Упражнение [3].** Доказать, что множество  $Y \subseteq X$  является ядром орграфа  $G = (X, U)$  тогда и только тогда, когда оно одновременно максимальное независимое и минимальное доминирующее.

Из данного утверждения следует, что для выделения ядер орграфа достаточно, например, найти все минимальные доминирующие множества вершин орграфа  $G$ , а затем выбрать из них максимальные независимые множества.

В рассмотренном выше примере множество  $\{x_2\}$  является доминирующем и независимым, а, следовательно, ядром, т.е.  $Y = \{x_2\}$ . Кроме того,  $\alpha(G) = 2, \beta(G) = 1$ .

В неорграфе каждое максимальное независимое множество является доминирующим, а, следовательно, ядром. Кроме того, для нахождения ядер применим изложенный выше алгоритм со следующими модификациями:

1) логическая функция имеет вид

$$\Phi_{ядро} (x_1, \dots, x_n) = \bigwedge_{i=1}^{n-1} \left( x_i \vee \left( \bigwedge_{\{(i,j): a_{ij}=1, j>i\}} x_j \right) \right);$$

2) пусть  $\Phi = \bigvee_{r=1}^p K_r$ , где  $K_r$  – элементарная конъюнкция. Для каждого  $r = \overline{1, p}$  найти множества

$$Y_r = \{j : x_j \text{ не входит в } K_r\}.$$

◆ **Упражнение [6].** Доказать, что если  $Y$  – ядро графа, то его числа устойчивости удовлетворяют соотношению  $\alpha(G) \geq |Y| \geq \beta(G)$ .

Заметим, что орграф может не иметь ядра, иметь одно или несколько ядер. Рассмотрим некоторые признаки существования ядра в орграфе.

Утверждение 1: Каждому конечному бесконтурному орграфу соответствует единственное ядро.

Для нахождения ядра в бесконтурном орграфе  $G = (X, U)$  предлагается следующая процедура 1:

Шаг 1. Выбрать произвольную вершину  $x_0$  с нулевой полустепенью захода и поместить ее в множество  $Y$ . Найти  $\Gamma(x_0) = \{w : (x_0, w) \in U\}$  – множество вершин, в которые ведет дуга из вершины  $x_0$ .

Шаг 2. Удалить из графа  $G$  вершину  $x_0$ , а также вершины множества  $\Gamma(x_0)$  вместе с инцидентными дугами.

Шаг 3. Шаги 1-2 повторять до тех пор, пока множество вершин  $X$  не окажется пустым. Построенное множество  $Y$  содержит единственное ядро графа  $G$ .

**Пример.** Ниже на рисунках представлены результаты работы алгоритма на бесконтурном графе, включающем 9 вершин.

	<p><u>Шаг 1.</u> На данном шаге вершины 1 и 2 не имеют входящих дуг, можно выбрать любую из них. Выберем вершину 2 и положим <math>Y = \{2\}</math>. Найдем <math>\Gamma(2) = \{3, 9\}</math>, поэтому из графа необходимо удалить вершины 2, 3, 9.</p>
	<p><u>Шаг 2.</u> В полученном подграфе вершины 1, 4 и 8 не имеют входящих дуг, поэтому можно выбрать любую из них. Выберем вершину 1 и включим ее в строящееся ядро <math>Y = \{2, 1\}</math>. Найдем <math>\Gamma(1) = \{7\}</math>, удалим из графа вершины 1 и 7 вместе с инцидентными дугами.</p>
	<p><u>Шаг 3.</u> В полученном подграфе вершины 4, 6, 8 не имеют входящих дуг. Выбирая, например, вершину 8, получим <math>Y = \{2, 1, 8\}</math>, <math>\Gamma(8) = 5</math>. Удалив вершины 8 и 5, получим подграф, который состоит из двух изолированных вершин 4 и 6. Последовательно выбирая их и включая в строящееся ядро, на последнем шаге получим <math>Y = \{2, 1, 8, 4, 6\}</math>.</p>

♦ **Упражнение.** Предложить модификацию алгоритма для сокращения количества шагов.

Утверждение 2: Каждый конечный сильно связный орграф, в котором содержится более одной вершины и нет нечетных контуров, имеет ровно два ядра (контур с нечетным количеством дуг – нечетный).

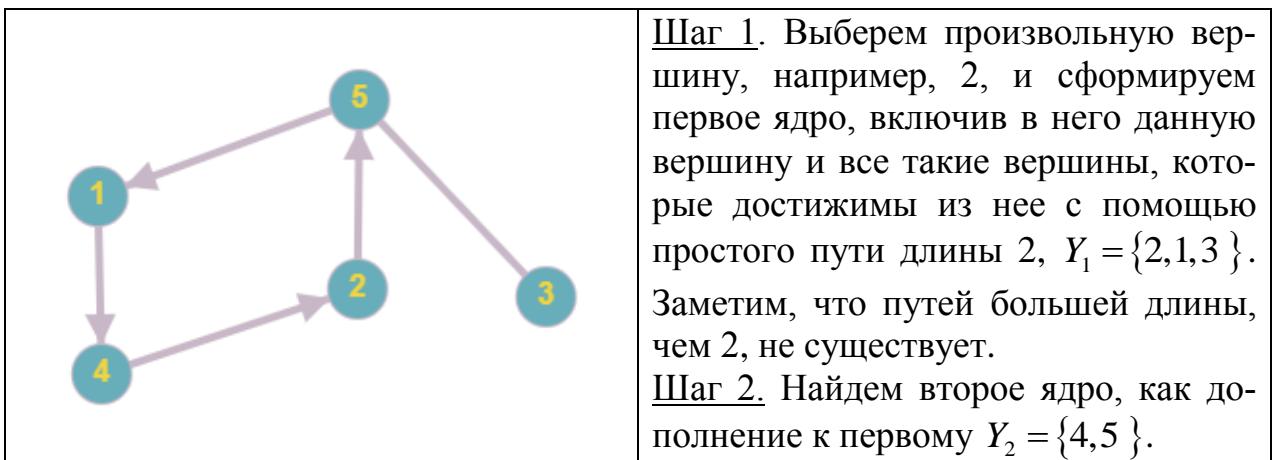
Для нахождения устойчивого множества в сильно связном орграфе  $G = (X, U)$  без нечетных контуров предлагается следующая процедура 2:

Шаг 1. Выбрать произвольную вершину  $x_0$  и поместить ее в  $Y_1$ .

Шаг 2. Дополнить множество  $Y_1$ , включив в него все вершины, которые можно достичь из  $x_0$  с помощью простого пути четной длины (в простом пути никакая дуга не встречается дважды).

Шаг 3. Положить  $Y_2 = X \setminus Y_1$ . Так как  $G$  сильно связан и имеет более одной вершины, то  $Y_1$  и  $Y_2$  не пусты и являются ядрами.

**Пример.** На рисунке изображен орграф, причем вершины 3 и 5 соединены ребром. Это означается наличие двух противоположно направленных дуг  $(3,5)$  и  $(5,3)$ .



Утверждение 3: Каждый конечный орграф  $G = (X, U)$  без нечетных контуров имеет ядро.

♦ **Упражнение.** Предложите процедуру для поиска ядер в орграфе без нечетных контуров.

## ЛЕКЦИЯ

### Функция Гранди

Пусть  $G = (X, U)$  – орграф. На вершинах графа определим функцию  $g : X \rightarrow N \cup \{0\}$ , которая каждой вершине  $x \in X$  ставит в соответствие целое неотрицательное число.

Будем говорить, что  $g(x)$  есть *функция Гранди* для данного графа  $G$ , если в каждой вершине  $x$  значение  $g(x)$  представляет собой наименьшее из тех неотрицательных чисел, которые не принадлежат множеству  $\{g(y) : y \in \Gamma(x)\}$ .

Если для орграфа существует функция Гранди, то говорят, что орграф *допускает* (в противном случае – не допускает) функцию Гранди. Не всякий орграф допускает функцию Гранди, а если и допускает, то она не обязательно единственная.

♦ **Упражнение** Пусть  $G$  – орграф без контуров. Доказать, что он допускает функцию Гранди и притом единственную.

♦ **Упражнение.** Доказать, что симметрический орграф допускает функцию Гранди тогда и только тогда, когда он не имеет петель.

Заметим, что неорграф можно рассматривать как симметричный орграф без петель, поэтому в нем всегда существует функция Гранди.

Рассмотрим некоторые алгоритмы построения функции Гранди.

#### Алгоритм построения функции Гранди для неорграфа

Шаг 1. Пусть  $G = (X, U)$  – неорграф на множестве вершин  $X$ . Положить  $r = 0$  ( $r$  – текущее значение функции Гранди).

Шаг 2. Выбрать в  $G$  произвольное максимальное независимое множество  $S$ . Для всех вершин  $x \in S$  положить  $g(x) = r$ .

Шаг 3. Удалить из  $G$  вершины множества  $S$  вместе с инцидентными ребрами. Если  $X = \emptyset$ , то останов – функция Гранди построена, иначе положить  $r = r + 1$  и перейти к шагу 2.

Заметим, что данный алгоритм позволяет построить в неорграфе одну из функций Гранди.

**Пример.** Рассмотрим граф  $G$ , представленный на рис. 1. Пусть  $r = 0$ . Согласно алгоритму, выделим в нем произвольное максимальное независимое множество, например,  $S_1 = \{1, 3\}$ . Положим  $g(1) = 0, g(3) = 0$ . Удалим вершины 1 и 3 из графа вместе с инцидентными дугами. В полученном подграфе (рис. 2) вновь выделим максимальное независимое множество, например,  $S_2 = \{4\}$ . Вершине 4 приписывается следующее значение функции Гранди, т.е.  $g(4) = 1$ . Удалим вершину 4 из графа, и в полученном подграфе (рис. 3) выберем максимальное независимое множество, например,

$S_3 = \{2, 6\}$ . Припишем соответствующим вершинам следующее значение функции Гранди, т.е.  $g(2) = 2, g(6) = 2$ . Удалив вершины 2 и 6, получим несмежные вершины 5 и 7, которые образуют максимальное независимое множество в подграфе из двух изолированных вершин. Поэтому положим  $g(5) = 3, g(7) = 3$ .

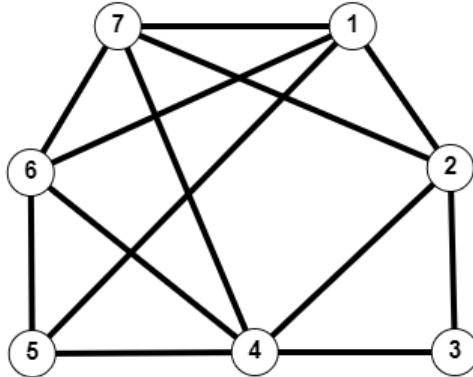


Рисунок 1 – Заданный график

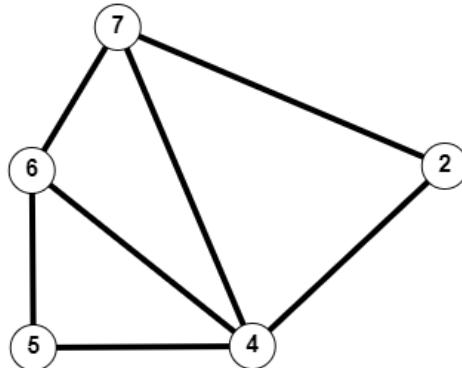


Рисунок 2 – Подграф после первой итерации

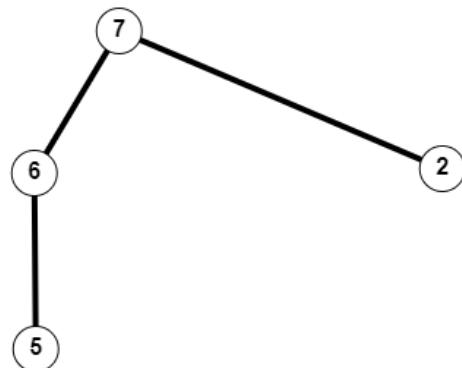


Рисунок 3 – Подграф после второй итерации

Так как на каждой итерации можно выбирать любое максимальное независимое подмножество, то алгоритм позволяет найти одну из возможных функций Гранди.

Алгоритм построения функции Гранди для бесконтурного графа

Шаг 1. Пусть  $G = (X, U)$  – данный бесконтурный орграф. Разложить его на уровни, применив, например, *метод вычеркивания дуг*.

Шаг 2. Вершинам  $x_0$ , не имеющим входящих дуг, присвоить значение  $g(x_0) = 0$ .

Шаг 3. Двигаясь снизу вверх от уровня к уровню, присыпывать каждой вершине  $x$  значение функции Гранди на основе определения, т.е.

$$g(x) = \min\{(N \cup \{0\}) \setminus \{g(y) : y \in \Gamma(x)\}\}.$$

**Пример.** Пусть задан граф  $G$ , изображенный на рис. 4.

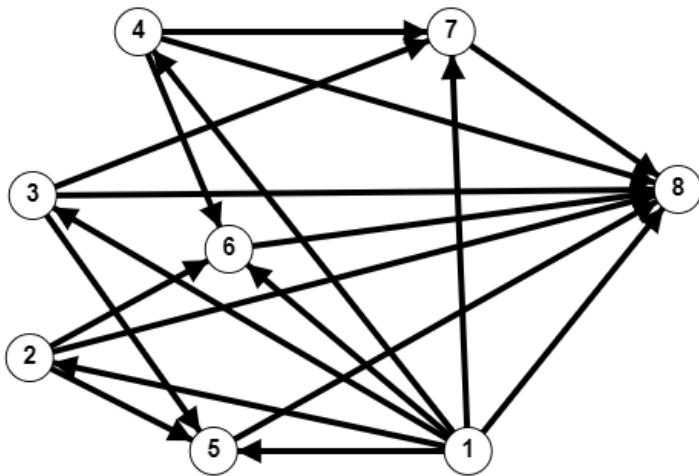


Рисунок 4 – Граф  $G$

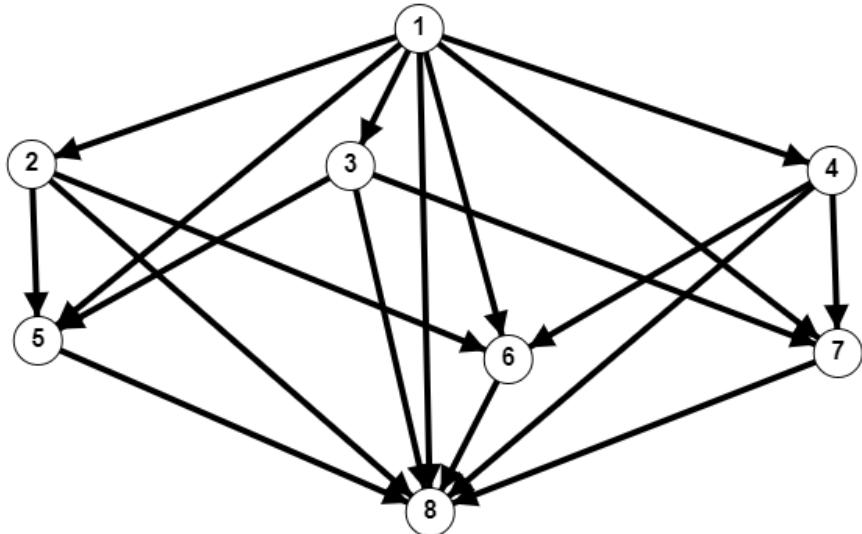


Рисунок 5 – Упорядоченный граф (разложен по уровням)

Построим его разложение на уровни, например, с помощью метода вычеркивания дуг (рис. 5): на первом уровне располагаем вершины без входящих дуг – эти вершины считаем просмотренными. Затем удаляем из графа просмотренные вершины. В полученном подграфе ищем вершины без вхо-

дящих дуг и располагаем их на следующем уровне и так действует до тех пор, пока все вершины не будут распределены по уровням. Теперь перейдем к шагу 2. Вершина 8 находится на нижнем уровне и не имеет выходящих дуг, ей приписываем  $g(8)=0$ . Далее перемещается к вершинам предыдущего уровня, их можно рассматривать в произвольном порядке. Каждая из вершин имеет одну выходящую дугу, поэтому положим  $g(5)=1, g(6)=1, g(7)=1$ . Теперь будем рассматривать вершины 2,3,4. Рассмотрим вершину 2,  $\Gamma(2)=\{5,6,8\}$ , при этом  $g(8)=0, g(5)=g(6)=1$ . Тогда в соответствии с определением функции Гранди для вершины 2 ее значение есть наименьшее целое неотрицательное число, не совпадающее с 0 и 1, поэтому  $g(2)=2$ . Аналогично рассуждая, получим  $g(3)=2, g(4)=2$ . Перемещается к следующему уровню, на котором находится вершина 1. Для нее  $\Gamma(1)=\{2,3,4,5,6,7,8\}$ . Значения функции Гранди для всех этих вершин известны  $\{0,1,2\}$ , поэтому по определению  $g(1)=3$ .

## ЛЕКЦИЯ 29-30.10.2020

### Паросочетания

Пусть  $G = (X, U)$  – граф без петель. Два ребра  $u_i$  и  $u_j$  из  $U$  называются смежными, если они имеют общую вершину.

Подмножество ребер  $P \subset U$  называется *паросочетанием* графа  $G$ , если в  $P$  нет двух смежных ребер.

Связь между паросочетаниями и независимыми множествами устанавливается с помощью реберных графов.

Говорят, что вершина *покрыта относительно*  $P$  или является *насыщенной* в паросочетании  $P$ , если она является концевой вершиной какого-нибудь ребра  $P$ . Вершина  $x$ , не являющаяся концевой вершиной никакого ребра из  $P$ , называется *экспонированной вершиной*.

Паросочетание, насыщающее все вершины графа  $G$ , называется *совершенным*. Заметим, что совершенное паросочетание, в частности, не существует для графов с нечетным множеством вершин.

Паросочетание  $P$  графа  $G$  будем называть *максимальным*, если не существует паросочетания большей мощности, включающего  $P$ . Мощность максимального паросочетания называется *числом паросочетания*.

Если существует совершенное паросочетание для графа  $G$ , то оно является и максимальным.

Пусть  $P$  – паросочетание графа  $G = (X, U)$ .

*Чередующаяся (альтернирующая) относительно  $P$  цепь* – это простая цепь, ребра которой попеременно принадлежат и не принадлежат паросочетанию  $P$ .

*Аугментальная относительно  $P$  цепь* – это такая чередующаяся цепь, у которой начальная и конечная вершины являются экспонированными.

Имеет место следующая

**Теорема** (Берж). Паросочетание  $P$  в графе  $G$  является максимальным тогда и только тогда, когда  $G$  не содержит аугментальной цепи относительно  $P$ .

Чередующееся дерево – это дерево, вершины которого разбиты на два класса – внутренние и внешние таким образом, что каждая внутренняя вершина имеет степень 2, и каждое ребро соединяет внутреннюю вершину с внешней.

Граф, изображенный на рис. 1, является чередующимся деревом, при этом вершины 3,5,7,9,10 – внутренние вершины, вершины 1,2,4,6,8,11 – внешние вершины.

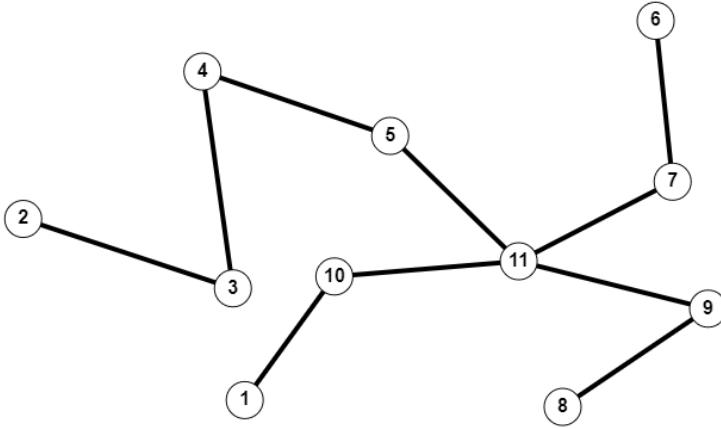


Рисунок 1 – Чередующееся дерево

Заметим, что висячие вершины, имеющие степень 1, всегда внешние. Количество внешних вершин чередующегося дерева всегда точно на 1 больше числа внутренних вершин. В самом деле, каждое ребро дерева инцидентно в точности одной внутренней вершине. Следовательно, если в графе  $m$  ребер, то существует  $\frac{m}{2}$  внутренних вершин и  $(m+1) - \frac{m}{2} = \frac{m}{2} + 1$  внешних вершин.

Для нахождения максимального паросочетания чередующегося дерева используется следующая процедура:

Шаг 1. Выбрать любую внешнюю вершину в качестве  $x_0$  и приписать ей метку  $d_0 = 0$ . Каждой из остальных вершин  $x_k$  приписать метку  $d_k$ , равную числу ребер в каждой простой цепи, соединяющей вершину  $x_0$  с данной вершиной  $x_k$ .

Шаг 2. Перечислить множество ребер  $(x_i^{\text{внут}}, x_j^{\text{внеш}})$ , у которых одна концевая вершина – внутренняя вершина  $x_i^{\text{внут}}$ , другая концевая вершина –  $x_j^{\text{внеш}}$ , причем  $d_j = d_i + 1$ . Данное множество ребер образует максимальное паросочетание.

Пример. Найдем максимальное паросочетание для чередующегося дерева, изображенного на рис. 1. На рис. 2 каждой вершине соответствует метка – длина цепи, найденной из вершины  $x_0 = 1$  (для нее  $d_0 = 0$ ). Будем последовательно рассматривать внутренние вершины 3, 5, 7, 9, 10 (рис. 1). Для вершины 3 метка равна 5, следовательно, нужно выбрать внешнюю вершину 2, у которой метка на 1 больше, тогда ребро  $(3, 2)$  попадает в паросочетание. Следующая внутренняя вершина 5 (рис. 1), ее метка равна 3 (рис. 2). Данная вершина смежна с вершинами 4 и 11, имеющими метки 4 и 2 соответственно. Выбираем вершину 4, и включаем ребро  $(5, 4)$  в паросочетание. Внутренняя вершина 7 (рис. 1) смежна с вершинами 7 и 11. Ее метка 3 на 1 меньше метки 4 у вершины 6, поэтому ребро  $(7, 6)$  дополняет строящееся паросочетание. Внутренняя вершина 9 (рис. 1) смежна с вершинами 8 и 11. Включаем в па-

росочетание ребро  $(9,8)$ , так как метки этих вершин 3 и 4 соответственно отличаются на 1. Наконец, внутренняя вершина 10 (рис. 1), смежная с вершинами 1 и 11, позволяет включить в паросочетание ребро  $(10,11)$  в соответствии с теми же рассуждениями, которые приведены для остальных внутренних вершин. Таким образом, получим максимальное паросочетание

$$P = \{(3,2),(5,4),(7,6),(9,8),(10,11)\}.$$

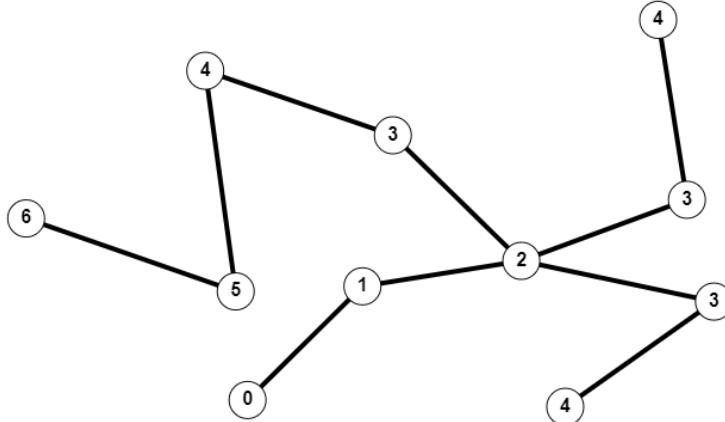


Рисунок 2 – Вершинам чередующегося дерева поставлены  
в соответствие метки

На рис. 3 представлено паросочетание  $P$  и его матрица смежности.

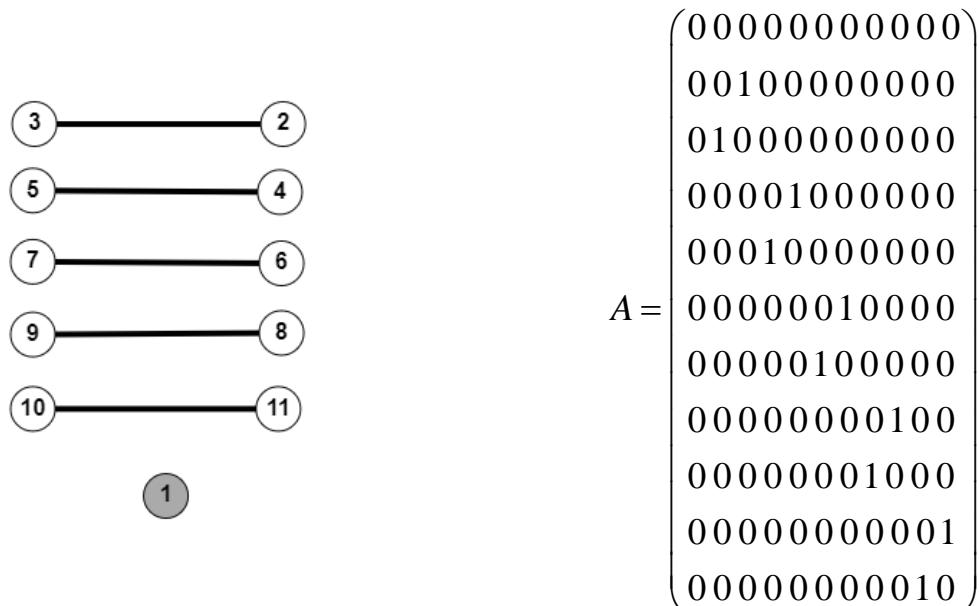


Рисунок 3 – Максимальное паросочетание  $P$  и его матрица смежности

Заметим, что найденному паросочетанию соответствует разбиение всех вершин графа, за исключением начальной вершины, при этом одно подмножество состоит из внутренних вершин, другое подмножество из внешних вершин. Если оставить только те ребра, которые образуют паросочетание, то в матрице смежности изолированной вершине соответствуют строка и столбец из 0; каждая единица является единственной в строке и в столбце.

Граф  $G = (X, U)$  называется двудольным, если существует такое разбиение его вершин  $\langle X_1, X_2 \rangle$ , когда каждое ребро  $u \in U$  имеет одну концевую вершину во множестве  $X_1$ , а другую концевую вершину – во множестве  $X_2$ . Множества  $X_1$  и  $X_2$  называются долями. Так как  $\langle X_1, X_2 \rangle$  – разбиение, то  $X_1 \cup X_2 = X$ ,  $X_1 \cap X_2 = \emptyset$ , и каждая вершина  $x \in X$  принадлежит только одному из множеств.

Рассмотрим алгоритм поиска максимального паросочетания в двудольном графе.

Шаг 1. Построить двудольный граф и ориентировать его ребра из вершин множества  $X_1$  в вершины множества  $X_2$ . К доле  $X_1$  добавить вершину  $s$  (источник), которую следует соединить дугой с каждой вершиной из множества  $X_1$ . К доле  $X_2$  добавить вершину  $t$  (сток), и каждую вершину из  $X_2$  соединить дугой с вершиной  $t$ . В полученном графе существует множество путей из  $s$  в  $t$ . Каждый такой путь будем называть допустимым, если он учитывает ориентацию дуг, которые его образуют.

Шаг 2. Выбрать любой допустимый путь из  $s$  в  $t$  и поменять ориентацию дуг на противоположную, а затем удалить начальную дугу пути, выходящую из  $s$ , и конечную дугу, входящую в  $t$ . Если дуга ориентирована из вершины  $X_1$  в вершину  $X_2$ , то будем называть ее прямой, иначе – обратной. Прямые дуги добавляются в решение, а обратные – удаляются.

Шаг 3. Шаг 2 повторять до тех пор, пока в графе найдется допустимый путь, иначе выписать решение.

Пример. Рассмотрим неориентированный двудольный граф, изображенный на рис. 4. Его доли  $X_1 = \{1, 2, 3\}$  и  $\{4, 5, 6, 7\}$ . Ориентируем ребра из вершин множества  $X_1$  в вершины множества  $X_2$ . Также добавим источник  $s$  и сток  $t$ , как показано на рис. 5.

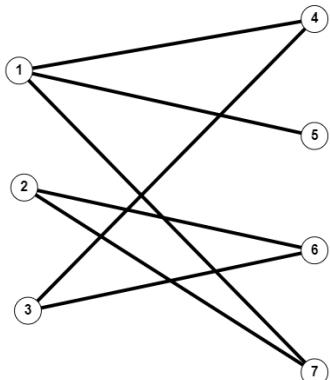


Рисунок 4

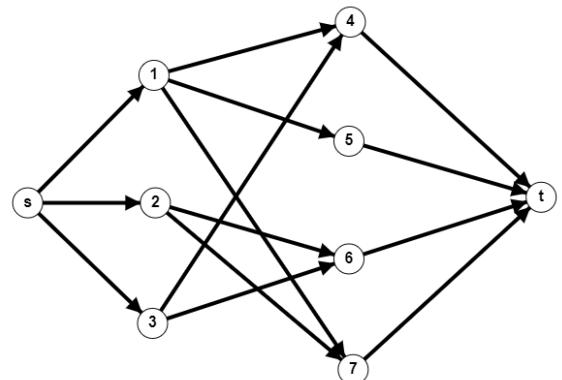
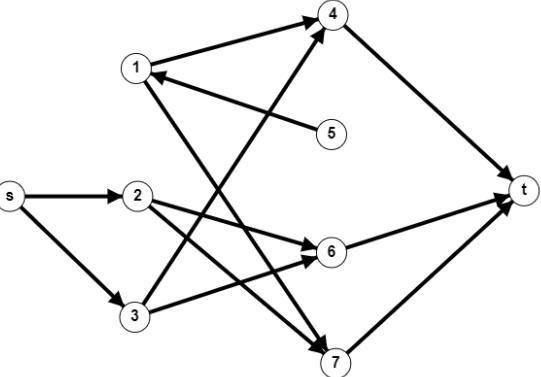
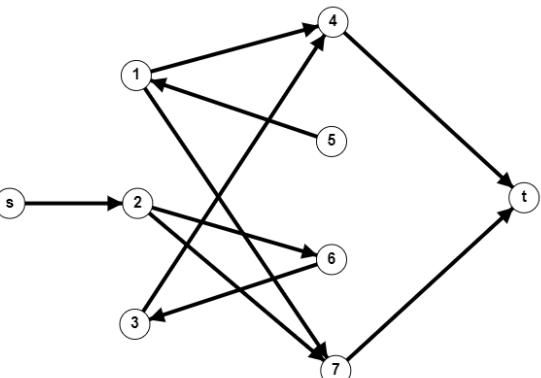


Рисунок 5

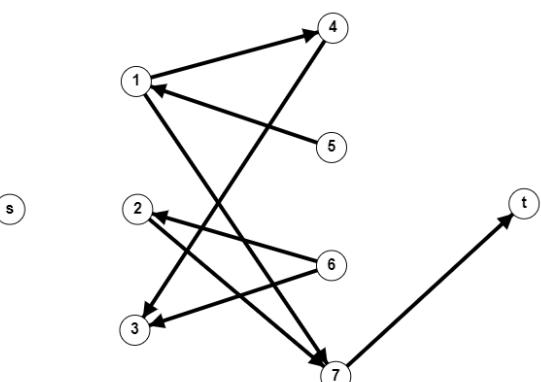
Дальнейшие действия представлены в следующей таблице.



Выберем любой путь из  $s$  в  $t$ , например,  $s \rightarrow 1 \rightarrow 5 \rightarrow t$ . Переориентируя дугу, получим дугу  $(5,1)$ , удалим начальную и конечную дуги пути. Включаем дугу  $(1,5)$  в строящееся паросочетание  $P = \{(1,5)\}$ .



Выберем другой допустимый путь, например,  $s \rightarrow 3 \rightarrow 6 \rightarrow t$ . Переориентируем прямую дугу  $(6,3)$ , удалим начальную и конечную дуги выбранного пути. Положим  $P = \{(1,5), (3,6)\}$ .



Выберем путь  $s \rightarrow 2 \xrightarrow{\text{прямая дуга}} 6 \xrightarrow{\text{обратная дуга}} 3 \xrightarrow{\text{прямая дуга}} 4 \rightarrow t$ . Переориентируем прямые дуги, удалим начальную и конечную дуги выбранного пути. Прямые дуги включаем во множество  $P = \{(1,5), (2,6), (3,4)\}$ , при этом исключается дуга  $(3,6)$ .

Таким образом, найдено подмножество ребер  $P = \{(1,5), (2,6), (3,4)\}$ , образующих максимальное паросочетание в двудольном графе на рис. 4.

Рассмотрим следующую задачу, известную как *задача о назначениях*.

Пусть имеется  $n$  работ и  $n$  исполнителей, каждый из которых умеет выполнять все работы. Назначение исполнителя  $i$  на работу  $j$  связано с затратами  $c_{ij}$ . Требуется так распределить работы между исполнителями, чтобы суммарные затраты были минимальны, при этом каждого исполнителя можно назначить только на одну работу, и каждая работа может выполняться только одним исполнителем.

Существует несколько подходов к построению модели задачи о назначениях. С одной стороны, решение этой задачи представляет собой перестановку  $(i_1, i_2, \dots, i_n)$  чисел  $1, 2, \dots, n$ . Соответствия вида  $k \rightarrow i_k$  ( $k = \overline{1, n}$ ) описывают возможные назначения (исполнитель  $k$  назначается на работу  $i_k$ ). Под стоимостью перестановки будем понимать сумму стоимостей соответствую-

ющих назначений. Задача сводится к нахождению такой перестановки, у которой стоимость минимальная.

С другой стороны, задача о назначениях относится к задачам дискретной оптимизации. Введем переменную

$$x_{ij} = \begin{cases} 1, & \text{если исполнитель } i \text{ назначен на работу } j, \\ 0, & \text{иначе.} \end{cases}$$

Тогда оптимизационная модель будет иметь следующий вид:

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} &\rightarrow \min \\ \sum_{j=1}^n x_{ij} = 1 &\left( i = \overline{1, n} \right), \quad \sum_{i=1}^n x_{ij} = 1 \left( j = \overline{1, n} \right), \\ x_{ij} &\geq 0, \quad i, j = \overline{1, n}. \end{aligned}$$

Заметим, что в данной постановке допустимое решение задачи представляется матрицей  $X = (x_{ij})_{n \times n}$ , в которой в каждой строке и в каждом столбце имеется только один 0.

Наиболее известным методом для решения задачи о назначениях является *венгерский метод*, который включает следующие шаги:

Шаг 1 (Приведение матрицы  $C$ ). Из всех элементов каждой строки матрицы  $C$  вычитается минимальный элемент соответствующей строки; из всех элементов каждого столбца вычитается минимальный элемент соответствующего столбца. В результате выполнения данного шага будет построена приведенная матрица затрат  $C'$ , в которой каждая строка и каждый столбец содержит хотя бы один 0.

Шаг 2 (Определение максимального числа независимых 0). Вычеркнуть все нулевые элементы приведенной матрицы минимальным числом горизонтальных и вертикальных линий. Если число линий, вычеркивающих все нули, меньше  $n$ , то решение не оптимально и переходим к следующему шагу. Иначе выписать решение в виде матрицы  $X$ , где 1 отмечены позиции независимых 0.

Шаг 3 (Получение новых 0). Выбираем наименьший невычеркнутый элемент, вычитаем его из каждого невычеркнутого элемента и прибавляем к каждому элементу, вычеркнутому дважды (стоит на пересечении проведенных прямых). Перейти к шагу 2.

В данном алгоритме наиболее проблематичным шагом является шаг 2. Отыскание независимых нулей можно реализовать различными способами. Один из способов основан на нахождении максимального паросочетания двудольного графа. Заметим, что так как мощности долей (исполнители и работы) равны, то количество вершин в соответствующем графе четно, поэтому максимальное паросочетание будет совпадать с совершенным.

## ТЕМА: ПАРОСОЧЕТАНИЯ

### ЗАЧЕТНАЯ ЗАДАЧА 1

Рассмотрим следующую задачу, известную как *задача о назначениях*.

Пусть имеется  $n$  работ и  $n$  исполнителей, каждый из которых умеет выполнять все работы. Назначение исполнителя  $i$  на работу  $j$  связано с затратами  $c_{ij}$ . Требуется так распределить работы между исполнителями, чтобы суммарные затраты были минимальны, при этом каждого исполнителя можно назначить только на одну работу, и каждая работа может выполняться только одним исполнителем.

Существует несколько подходов к построению модели задачи о назначениях. С одной стороны, решение этой задачи представляет собой перестановку  $(i_1, i_2, \dots, i_n)$  чисел  $1, 2, \dots, n$ . Соответствия вида  $k \rightarrow i_k$  ( $k = \overline{1, n}$ ) описывают возможные назначения (исполнитель  $k$  назначается на работу  $i_k$ ). Под *стоимостью перестановки* будем понимать сумму стоимостей соответствующих назначений. Задача сводится к нахождению такой перестановки, у которой стоимость минимальная.

С другой стороны, задача о назначениях относится к задачам дискретной оптимизации. Введем переменную

$$x_{ij} = \begin{cases} 1, & \text{если исполнитель } i \text{ назначен на работу } j, \\ 0, & \text{иначе.} \end{cases}$$

Тогда оптимизационная модель будет иметь следующий вид:

$$\begin{aligned} & \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \rightarrow \min \\ & \sum_{j=1}^n x_{ij} = 1 \left( i = \overline{1, n} \right), \quad \sum_{i=1}^n x_{ij} = 1 \left( j = \overline{1, n} \right), \\ & x_{ij} \geq 0, \quad i, j = \overline{1, n}. \end{aligned}$$

Заметим, что в данной постановке допустимое решение задачи представляется матрицей  $X = (x_{ij})_{n \times n}$ , в которой в каждой строке и в каждом столбце имеется только один 0.

Наиболее известным методом для решения задачи о назначениях является *венгерский метод*, который включает следующие шаги:

Шаг 1 (Приведение матрицы  $C$ ). Из всех элементов каждой строки матрицы  $C$  вычитается минимальный элемент соответствующей строки; из всех элементов каждого столбца вычитается минимальный элемент соответствующего столбца. В результате выполнения данного шага будет построена приведенная матрица затрат  $C'$ , в которой каждая строка и каждый столбец содержит хотя бы один 0.

Шаг 2 (Определение максимального числа независимых 0). Вычеркнуть все нулевые элементы приведенной матрицы минимальным числом горизонтальных и вертикальных линий. Если число линий, вычеркивающих все ну-

ли, меньше  $n$ , то решение не оптимально и переходим к следующему шагу. Иначе выписать решение в виде матрицы  $X$ , где 1 отмечены позиции независимых 0.

Шаг 3 (Получение новых 0). Выбираем наименьший невычеркнутый элемент, вычитаем его из каждого невычеркнутого элемента и прибавляем к каждому элементу, вычеркнутому дважды (стоит на пересечении проведенных прямых). Перейти к шагу 2.

В данном алгоритме наиболее проблематичным шагом является шаг 2. Отыскание независимых нулей можно реализовать различными способами. Один из способов основан на нахождении максимального паросочетания двудольного графа. Заметим, что так как мощности долей (исполнители и работы) равны, то количество вершин в соответствующем графе четно, поэтому максимальное паросочетание будет совершенным.

Пример. Пусть работы A,B,C,D нужно распределить между 4 исполнителями с номерами 1, 2, 3, 4, при этом матрица затрат имеет следующий вид:

$$C = \begin{pmatrix} 2 & 10 & 9 & 7 \\ 15 & 4 & 14 & 8 \\ 13 & 14 & 16 & 11 \\ 4 & 15 & 13 & 19 \end{pmatrix}.$$

Приведем матрицу сначала по строкам, а затем по столбцам. Вычитая из элементов каждой строки минимальный в этой строке элемент, получим матрицу  $C'$ . Производя ту же операцию, но для столбцов, получим матрицу  $C''$ .

$$C' = \begin{pmatrix} 0 & 8 & 7 & 5 \\ 11 & 0 & 10 & 4 \\ 2 & 3 & 5 & 0 \\ 0 & 11 & 9 & 15 \end{pmatrix}, \quad C'' = \begin{pmatrix} 0 & 8 & 2 & 5 \\ 11 & 0 & 5 & 4 \\ 2 & 3 & 0 & 0 \\ 0 & 11 & 4 & 15 \end{pmatrix}.$$

В матрице  $C''$  в каждой строке и в каждом столбце имеется хотя бы один 0. Необходимо выделить так называемые *независимые* нули (в каждом столбце и в каждой строке имеется только один такой ноль). Для этого воспользуемся алгоритмом, который рассматривался выше. На рис 1а) представлен граф, в котором вершины одной доли – это работы  $a,b,c,d$ , вершины другой доли – это исполнители 1,2,3,4, а ребра соответствуют нулям матрицы  $C''$ . На рис. 1б) представлен орграф с добавленными источником  $s$  и стоком  $t$ . На рис. 1с)-1е) представлена работа алгоритма.

Выберем произвольный путь из  $s$  в  $t$ , например,  $s \rightarrow a \rightarrow 1 \rightarrow t$ , переориентируем прямую дугу  $(a,1)$ , удалим дуги  $(s,a)$  и  $(1,t)$ . Включим прямую дугу в строящееся паросочетание  $P = \{(a,1)\}$ . Далее последовательно будем рассматривать пути  $s \rightarrow b \rightarrow 2 \rightarrow t$ ,  $s \rightarrow c \rightarrow 3 \rightarrow t$ . Удаляя начальные и конечные дуги путей и переориентируя прямые дуги в соответствии с алго-

ритмом, получим паросочетание  $P = \{(a,1), (b,2), (c,3)\}$ , при этом граф будет иметь вид, изображенный на рис. 1e). В нем не существует пути из  $s$  в  $t$ . Заметим, что работе  $d$  исполнитель не назначен, а исполнитель 4 не получил работу. Паросочетание  $P$  является максимальным но не совершенным, так как вершины  $d$  и 4 являются экспонированными (не покрытыми).

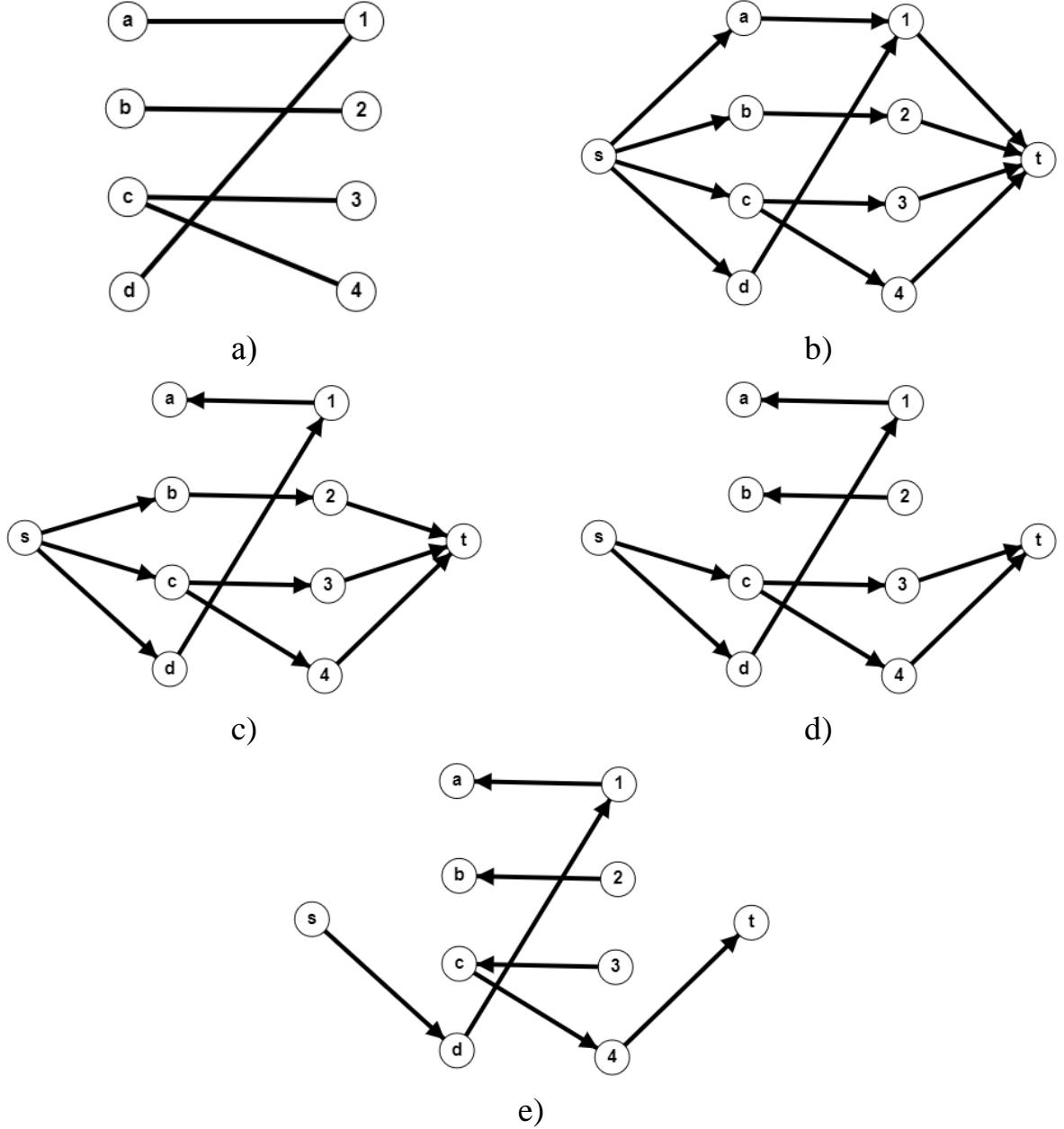


Рисунок 1 – Реализация шагов алгоритма на первом этапе

Паросочетание  $P$  позволяет определить независимые нули матрицы  $C''$ . Теперь необходимо минимальным числом вертикальных и/или горизонтальных линий вычеркнуть все нули матрицы. Для матриц малой размерности это делается на основе полного перебора, однако, существуют и специальные методы. В нашем примере можно вычеркнуть первый столбец и вторую и третью строки. Количество необходимых линий  $n = 3$  не равно размерности матрицы, поэтому решение не оптимально и, венгерский алгоритм должен продолжить свою работу. Среди невычеркнутых элементов найдем мини-

мальный  $\min = 2$ . Вычитаем его из всех невычеркнутых элементов и прибавляем к элементам, вычеркнутым дважды. В результате матрица  $C''$  обновится и будет иметь следующий вид:

$$C'' = \begin{pmatrix} 0 & 6 & 0 & 3 \\ 13 & 0 & 5 & 4 \\ 4 & 3 & 0 & 0 \\ 0 & 9 & 2 & 13 \end{pmatrix}.$$

Снова будем искать независимые нули матрицы  $C''$  с помощью алгоритма поиска максимальных паросочетаний.

Ребра граф на рис. 2а) соответствуют нулям матрицы  $C''$ . Ориентируя ребра и вводя источник  $s$  и сток  $t$ , получим граф, изображенный на рис. 2б). На рис. 2с)-2е) представлены шаги алгоритма. При выборе пути  $s \rightarrow a \rightarrow 1 \rightarrow t$  получим  $P = \{(a,1)\}$  и обратную дугу  $(1,a)$ , дуги  $(s,a)$  и  $(1,t)$  удаляются (рис. 2с)). При выборе пути  $s \rightarrow b \rightarrow 2 \rightarrow t$  получим  $P = \{(a,1), (b,2)\}$  и обратную дугу  $(2,b)$ , дуги  $(s,b)$  и  $(2,t)$  удаляются (рис. 2д)). В полученном графе существует несколько вариантов выбора путей из  $s$  в  $t$ , которые появились благодаря переориентации дуг. Выберем путь  $s \rightarrow d \rightarrow 1 \rightarrow a \rightarrow 3 \rightarrow t$ , в котором  $(d,1)$  и  $(a,3)$  – прямые дуги. Включаем

прямая дуга                           прямая дуга

их в строящееся паросочетание, при этом дугу  $(a,1)$ , которая в рассматриваемом пути встречается как обратная, исключаем. В результате получим множество  $P = \{(b,2), (d,1), (a,3)\}$ . Соответствующий преобразованный граф представлен на рис. 2е). Данный граф содержит единственный путь  $s \rightarrow c \rightarrow 4 \rightarrow t$ , содержащий прямую дугу  $(c,4)$ , которая включается в множество  $P$ . После удаления начальной и конечной дуг пути и переориентации дуги граф примет вид, представленный на рис. 2ф).  $P = \{(b,2), (d,1), (a,3), (c,4)\}$  можно получить по графу, учитывая обратные дуги. Паросочетание  $P$  является совершенным, поэтому все нули матрицы  $C''$  можно вычеркнуть  $n=4$  линиями. Решение задачи представляет собой квадратную матрицу вида

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

В этой матрице строки соответствуют работам  $a, b, c, d$ , столбцы – исполнителям под номерами 1,2,3,4. Заметим, что единицы матрицы  $X_{onm}$  соответствуют элементам паросочетания  $P$ . Соотнеся единицы матрицы  $X_{onm}$  с соответствующими коэффициентами затрат из матрицы  $C$ , получим суммарные затраты  $9 + 4 + 11 + 4 = 28$ .

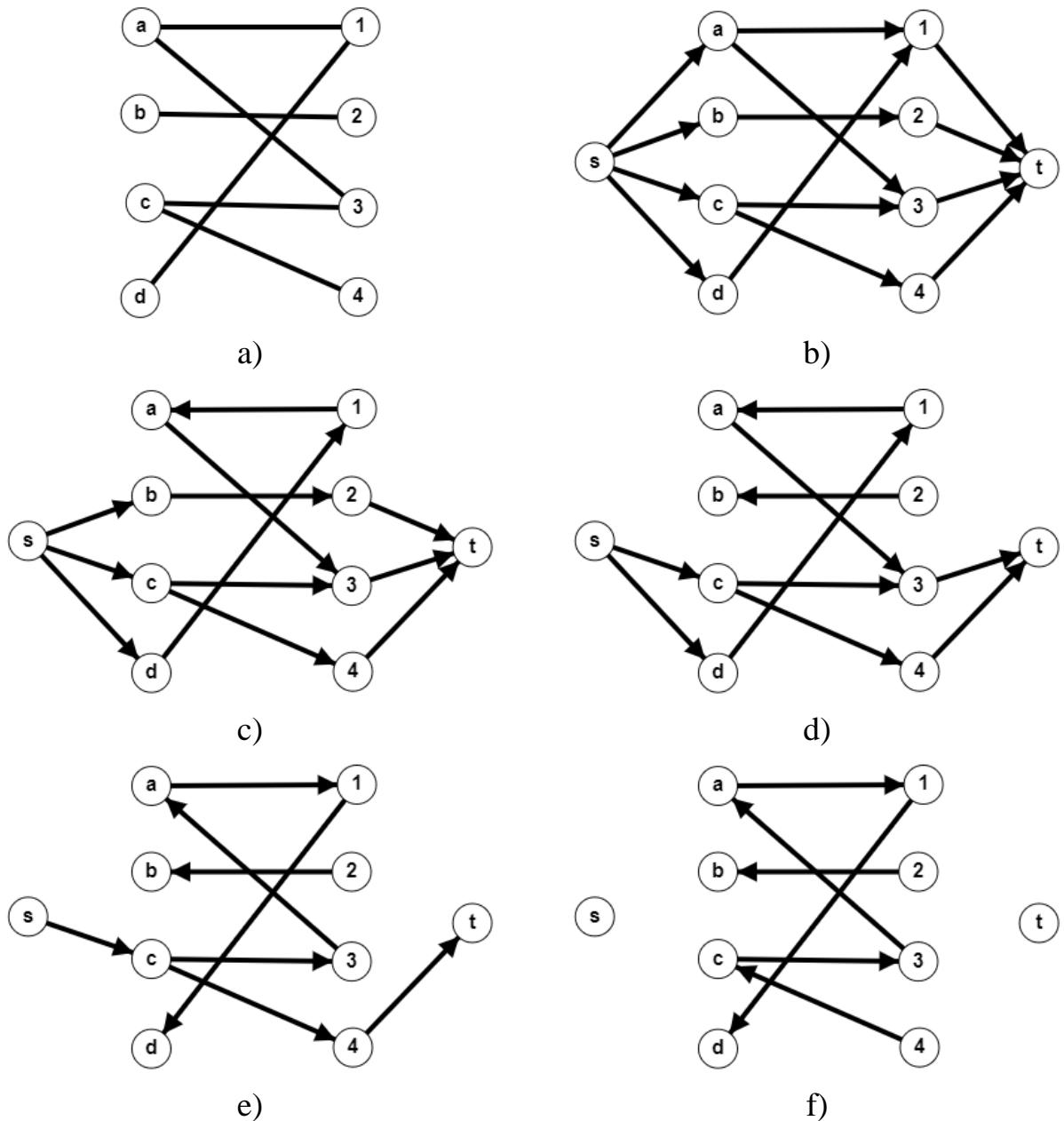


Рисунок 2 – Реализация шагов алгоритма на втором этапе

**Задание:** Предположим, что вычислительная система включает  $n$  вычислительных устройств разной производительности для решения  $n$  типов задач. Временные затраты на решение задачи  $i$ -го типа  $j$ -ым устройством заданы в форме матрицы  $C$ . Определить такое распределение задач между вычислительными устройствами, чтобы затраты времени на обработку пакета из  $n$  различных задач были минимальными.

Для  $n = 6$  задать самостоятельно матрицу  $C$  и решить задачу.

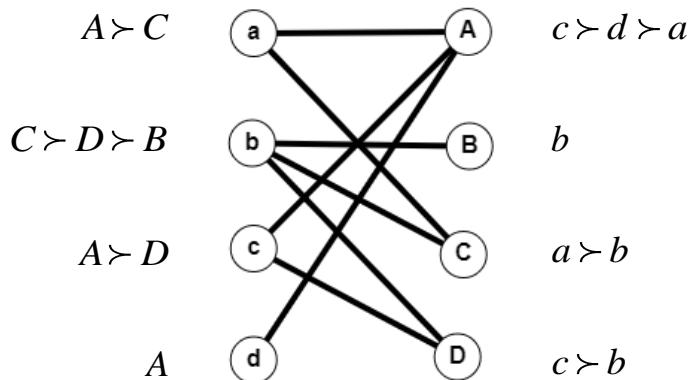
## ЗАЧЕТНАЯ ЗАДАЧА 2 (задача о свадьбах)

Пусть  $X$  – множество юношей,  $Y$  – множество девушек, а наличие ребра  $(x, y)$  означает, что  $x$  и  $y$  могут пожениться, тогда паросочетание есть коллективная свадьба, в которой каждый заключает не более одного брака. Ясно, что в реальной жизни лицо имеет предпочтения, и эту особенность необходимо учитывать в нашей ситуации. Предположим, что для каждой вершины  $v \in X \cup Y$  задано упорядочение вершин  $R(v) = \{z_1 \succ z_2 \succ \dots \succ z_{d(v)}\}$ , которые с ней смежны, при этом на первом месте стоит вершина  $z_1$ , которой соответствует наиболее предпочтительное лицо, а на последнем месте –  $z_{d(v)}$  – наименее предпочтительное лицо.

Паросочетание  $P$  в двудольном графе  $G = (X \cup Y, E)$  называется устойчивым, если для любого ребра  $(u, v) \in E \setminus P$ , где  $u \in X$ ,  $v \in Y$ , либо  $(u, v) \in P$  и  $u \succ v$  в  $R(u)$ , либо  $(v, u) \in P$  и  $v \succ u$  в  $R(v)$ , либо выполняется и то, и другое.

В нашей интерпретации множество бракосочетаний устойчиво, если в нем отсутствуют ситуации, когда брак между  $u \in R(v)$  и  $v \in R(u)$  не заключен, но  $u$  предпочитает  $v$  своей жене (или не женат), а  $v$  предпочитает  $u$  своему мужу (или не замужем).

Пример. На рисунке изображен двудольный граф, в котором одна доля содержит вершины  $a, b, c, d$ , в вторая доля содержит вершины  $A, B, C, D$ . Для каждой вершины задан список предпочтений вершин из другой доли.



В данном графе имеется единственное максимальное паросочетание  $P = \{(a, C), (b, B), (c, D), (d, A)\}$  с четырьмя ребрами, но оно не устойчиво из-за ребра  $(c, A)$ .

Известно следующее

Утверждение 1. Устойчивое паросочетание всегда существует.

Доказательство данного утверждения основано на следующем алгоритме. На первом шаге все юноши  $u \in X$  делают предложения своим самым желанным девушкам. Если девушка получает хотя бы одно предложение, она

выбирает наиболее желанного юношу и считает его кандидатом в женихи. Оставшиеся юноши отвергнуты и образуют резерв  $T$ . На втором шаге все юноши из  $T$  делают предложения своим вторым по предпочтениям девушкам. Каждая девушка сравнивает предложения (учитывая кандидата в женихи, если таковой имеется), выбирает наиболее предпочтительного юношу и делает его кандидатом в женихи. Отвергнутые юноши составляют новый резерв  $T$ .

После этого юноши из  $T$  делают предложения своим следующим по предпочтениям девушкам, и т.д. Юноша, который сделал предложение последней по его предпочтениям девушке и был отвергнут, исключается из дальнейшего рассмотрения, а также из резерва. Ясно, что через некоторое время резерв  $T$  окажется пустым, и в этот момент алгоритм заканчивает работу.

Утверждение 2. *Если алгоритм завершил работу, то кандидаты в женихи вместе с выбранными ими девушками образуют устойчивое паросочетание.*

Заметим вначале, что для каждой девушки выбираемые ею кандидаты в женихи становятся (для нее) все более желанными, так как на каждом шаге она сравнивает новые предложения с кандидатом в женихи и выбирает наиболее предпочтительного. Поэтому, если  $(u, v) \in E$ , но  $(u, v) \notin P$ , то либо  $u$  никогда не делал предложение  $v$  (и в этом случае  $u$  нашел лучшую супругу прежде, чем очередь дошла до  $v$ , следовательно,  $(u, y) \in P$ , причем  $y \succ v$  в  $R(u)$ ), либо  $u$  делал предложение  $v$  и был отвергнут, откуда следует, что  $(x, v) \in P$ , где  $x \succ u$  в множестве  $R(v)$ . Но это есть как раз условие, определяющее устойчивое паросочетание.

**Задание.** Придумайте иллюстративный пример для двудольного графа  $K_{5,5}$ , подробно распишите шаги алгоритма.

## ЛЕКЦИЯ

### ДЕРЕВЬЯ

Понятие дерева как математического объекта было впервые предложено Кирхгофом в связи с определением фундаментальных циклов, применяемых при анализе электрических цепей. Деревья имеют особое положение в теории графов из-за предельной простоты строения, и часто при решении какой-либо задачи на графах ее сначала исследуют на деревьях. Понятие дерева применяется при конструировании различных алгоритмов. Задача о кратчайшем остове находит применение при нахождении таких сетей, которые связывает  $n$  точек таким образом, чтобы общая длина «линий связи» была минимальной (прокладка дорог, газопроводов, линий электропередач и т.п.).

*Деревом* называется конечный связный граф без циклов, имеющий не менее двух вершин.

Пусть  $G$  –  $n$ -вершинное дерево, тогда имеют место следующие его свойства:

- a)  $G$  имеет  $(n - 1)$  ребер;
- b)  $G$  не содержит циклов, но добавление ребра между любыми двумя несмежными вершинами приводит к появлению цикла;
- c)  $G$  связан, но утрачивает это свойство после удаления любого ребра;
- d) всякая пара вершин соединена простой цепью и притом только одной;
- e) в  $G$  имеется, по крайней мере, две висячие вершины;
- f)  $G$  является двудольным графом.

Последовательность  $\{d_1, \dots, d_n\}$  неотрицательных целых чисел является *последовательностью степеней дерева* тогда и только тогда, когда  $d_i \geq 1$  для

$$\text{всех } i = \overline{1, n} \text{ и } \sum_{i=1}^n d_i = 2(n - 1).$$

Замечание. Если перечисленные условия выполняются, то d-процедура построит графическую реализацию последовательности в виде дерева, если на каждом шаге выбирать в качестве ведущей вершину с минимальной положительной остаточной степенью.

*Древовидностью* называется орграф с ациклическим основанием, в котором полустепень захода вершины, называемой корнем, равна 0, а полустепени захода остальных вершин равны 1.

*Двоичным деревом* называется орграф с ациклическим основанием, полустепень исхода каждой вершины которого, не превышает 2.

*k*-деревом называется ациклический неорграф, состоящий из  $k$  компонент связности.

*Остовным деревом* или *остовом* связного графа  $G$  называется всякий остовный подграф  $T$ , являющийся деревом.

Ребра, принадлежащие оству  $T$  в графе  $G$ , называют *ветвями*, а остальные ребра  $G$  называют *хордами*.

Пусть  $G$  состоит из нескольких компонент связности  $G_1, \dots, G_m$ , тогда, определяя для каждой компоненты  $G_i$  остав  $T_i$ , получим оставный подграф  $G$ , который называется *лесом*.

Рассмотрим некоторые алгоритмы, касающиеся нахождения произвольного оства или оства с определенной характеристикой для заданного графа.

Пусть  $G$  – связный неорграф. Для построения произвольного оства можно использовать следующий алгоритм, который основан на просмотре в произвольном порядке ребер графа и их раскрашивании. Пусть голубой цвет используется для окраски ребер, включаемых в остав, а желтый – для окраски ребер, не включаемых в остав. При выборе некоторого ребра осуществляется проверка того, не образует ли данное ребро в совокупности с ребрами, уже включенными в остав, цикл. Заметим, что ребра, включенные в остав, образуют граф, состоящий из одной или нескольких компонент связности. Вершины, принадлежащие отдельной взятой компоненте, объединяются в «букет». Некоторое ребро образует цикл с ребрами, уже включенными в остав, если обе его концевые вершины принадлежат одному и тому же букету.

Алгоритм определения произвольного оства в связном неорграфе включает следующие шаги:

Шаг 1. Выбрать любое неокрашенное ребро и рассмотреть следующие ситуации:

- если обе концевые вершины выбранного ребра принадлежат одному букету, то окрасить выбранное ребро в желтый цвет;
- если одна из концевых вершин выбранного ребра принадлежит некоторому букету, а другая концевая вершина не принадлежит ни одному букету, то окрасить выбранное ребро в голубой цвет и включить его концевую вершину, не принадлежавшую ранее ни одному букету, в тот же букет, которому принадлежит другая концевая вершина рассматриваемого ребра;
- если ни одна из концевых вершин не принадлежит ни одному букету, то окрасить рассматриваемое ребро в голубой цвет и сформировать новый букет из его концевых вершин;
- если концевые вершины выбранного ребра принадлежат различным букетам, то окрасить ребро в голубой цвет, а оба букета, которым принадлежат его концевые вершины, слить в один букет.

Шаг 2. Если все вершины графа вошли в один букет, то остав – голубые ребра образуют остав. Иначе перейти к шагу 1.

**Пример.** Рассмотрим граф  $G$ , изображенный на рис. 1.

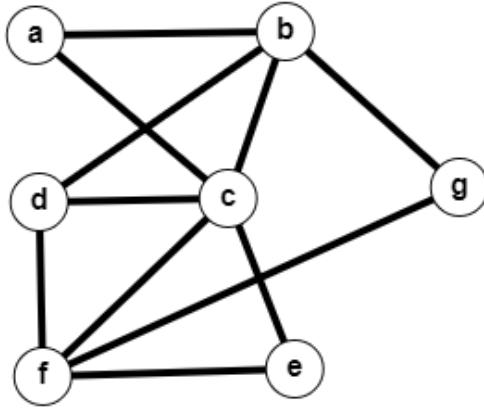


Рисунок 1 – Граф  $G$

Процесс образования букетов проиллюстрируем таблицей.

Итерация	Выбираем ребро	Цвет	Букет
1	$(a, b)$	голубой	$\{a, b\}$
2	$(d, c)$	голубой	$\{d, c\}$
3	$(d, f)$	голубой	$\{c, d, f\}$
4	$(b, g)$	голубой	$\{a, b, g\}$
5	$(c, e)$	голубой	$\{c, d, f, e\}$
6	$(f, g)$	голубой	$\{a, b, c, d, e, f, g\}$
7	$(f, c)$	желтый	-
8	$(a, c)$	желтый	-

После того, как получили букет, включающий все вершины, оставшиеся ребра, которые не выбирались, будут окрашены в желтый цвет.

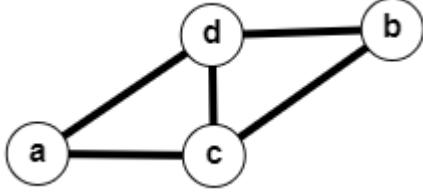
Заметим, что рассмотренный алгоритм позволяет определить произвольный остов связного графа. Здесь возникает проблема перечисления всех остовов заданного графа. Сначала ответим на вопрос: сколько существует остовов в  $n$ -вершинном графе?

Имеет место следующее

Утверждение 1. Пусть  $G$  –  $n$ -вершинный орграф без петель, и  $B_0$  – его матрица инцидентности с одной удаленной строкой,  $B_0^T$  – транспонированная матрица  $B_0$ , тогда количество различных остовых деревьев графа  $G$  есть  $\det(B_0 B_0^T)$ .

Замечание. Данное утверждение можно применить к нахождению остовов неорграфа, если в нем ввести произвольную ориентацию.

**Пример.** Рассмотрим граф, изображенный на рис. 2а). Произвольным образом введем ориентацию и построим соответствующую ему матрицу инцидентности  $B$ . Затем найдем усеченную матрицу  $B_0$  и матрицу  $B_0B_0^T$ .



a)

$B$	$u_1$	$u_2$	$u_3$	$u_4$	$u_5$
$a$	1	0	0	-1	0
$b$	0	0	1	0	-1
$c$	-1	-1	0	0	1
$d$	0	1	-1	1	0

b)

$B_0$	$u_1$	$u_2$	$u_3$	$u_4$	$u_5$
$a$	1	0	0	-1	0
$b$	0	0	1	0	-1
$c$	-1	-1	0	0	1

c)

$B_0B_0^T$	$a$	$b$	$c$
$a$	2	0	-1
$b$	0	2	-1
$c$	-1	-1	3

d)

Рисунок 1: а) заданный граф  $G$ ; б) матрица инцидентности  $B$ ;  
в)  $B_0$  — усеченная матрица инцидентности (удалена строка  $d$ );  
г) матрица  $B_0B_0^T$ .

Вычислим  $\det(B_0B_0^T) = 12 + 0 - 0 - 2 - 2 - 0 = 8$ . Таким образом, граф  $G$  имеет 8 остовов. На рис. 3 эти остовы перечислены.

Рассмотрим дерево  $T = (X, U)$ , где  $U \subset X^2$ . Преобразуем дерево  $T$  следующим образом: удалим из  $T$  некоторое ребро  $u \in U$  и добавим другое ребро  $v \in X^2 \setminus U$ , такое, что полученный граф  $T'$  снова является деревом. Преобразование  $T$  в  $T'$  называется *элементарным преобразованием*. Заметим, что деревья  $T$  и  $T'$  будут различаться лишь одним ребром.

*Расстоянием*  $d(T_i, T_j)$  между деревьями  $T_i$  и  $T_j$ , которые определены на одном и том же множестве вершин  $X$ , называется количество элементарных преобразований, позволяющих получить дерево  $T_j$  из дерева  $T_i$ .

Таким образом, если дерево  $T_j$  можно получить из дерева  $T_i$  с помощью  $k$  элементарных преобразований, то  $d(T_i, T_j) = k$ .

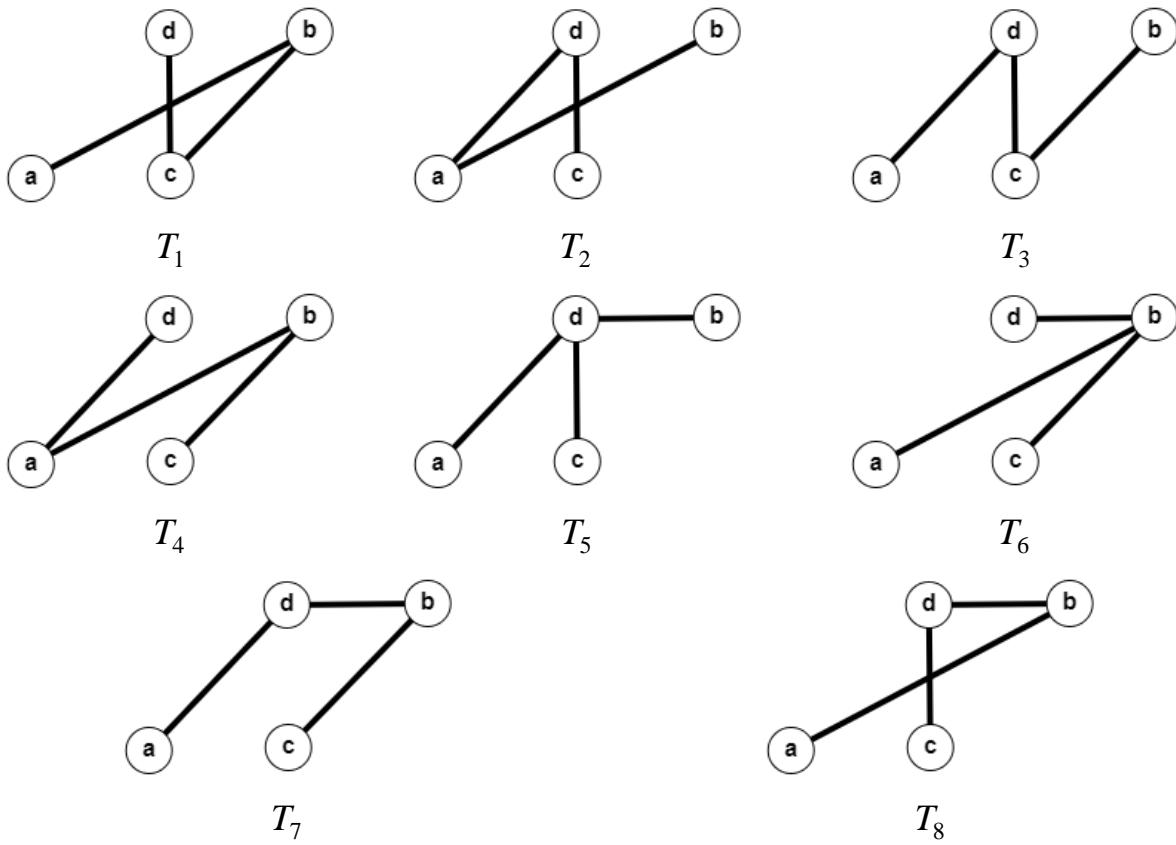


Рисунок 3 – Остовные деревья графа  $G$

Имеет место следующее

Утверждение 2. Если  $T_i$  и  $T_j$  – остовы графа  $G$  и  $d(T_i, T_j) = k$ , то остов  $T_j$  может быть получен из остова  $T_i$  с помощью серии из  $k$  элементарных преобразований.

Таким образом, если график  $G$  имеет  $m$  остовов, то их перечисление всех остовов можно осуществить с помощью элементарных преобразований.

*Графом остовов* графа  $G$  называется график, полученный следующим образом: каждому остову  $T_i$  поставим в соответствие вершину, и две вершины в графике остовов соединяются ребром тогда и только тогда, когда расстояние между соответствующими остовами равно 1.

**Задание.** Постройте подграф график остовов для примера, рассмотренного выше, выбрав произвольные 4 остовных дерева.

Ориентированное дерево в графике  $G$  может быть получено как результат работы алгоритмов поиска. Различают  *поиск в глубину* и  *поиск в ширину*. При использовании поисковых алгоритмов каждой вершине присваивается метка (глубина или ширина), которая соответствует порядковому номеру вершины при обходе графа данным видом поиска. Вершина, которая рассматривается в данный момент, называется *текущей*. При поиске в глубину выбирается *ко-*

*рень*  $r$  (вершина, в которой начинается обход графа), ему присваивается глубина, равная 1. В данный момент корень является текущей вершиной. Затем выбирается любая вершина из окрестности текущей. Заметим, что для всякой вершины  $x$ , за исключением корня, можно установить *отца* – вершину, из которой мы пришли в  $x$ , и *сына* – вершину, которая назначается текущей из вершины  $x$ . Если в какой-то момент оказалось, что все вершины, находящиеся в окрестности текущей, имеют глубину, то осуществляется шаг возврата из текущей вершины к ее отцу и далее осуществляется поиск из отца. Алгоритм заканчивает работу, если мы вернулись в корень.

Таким образом, при поиске в глубину строится ориентированное дерево поиска с корнем в заданной вершине. Введем обозначения:  $r$  – корень дерева поиска;  $tree$  – массив, содержащий дуги дерева поиска в глубину;  $back$  – массив ребер, не вошедших в дерево поиска;  $v$  – текущая вершина;  $depth(\cdot)$  – глубина вершины.

#### Алгоритм поиска в глубину в связном неорграфе без петель

Шаг 1.  $G$  – данный граф. Выбрать корень  $r$ , для которого положить  $depth(r) = 1$ . Положить  $v = r$ ,  $tree = \emptyset$ ,  $back = \emptyset$ ,  $k = 1$  ( $k$  – счетчик итераций).

Шаг 2. Если в окрестности  $v$  существуют непросмотренные ребра, то выбрать любое такое с концевой вершиной  $w$  и перейти к шагу 3, иначе перейти к шагу 4.

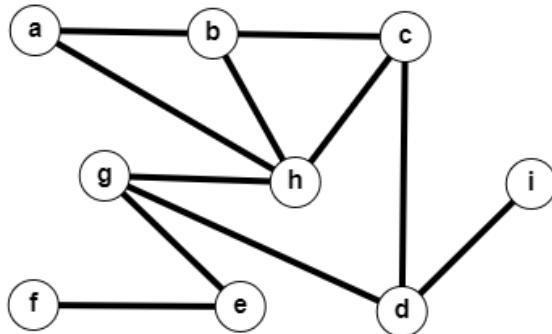
Шаг 3. Если вершина  $w$  не имеет глубины, то ориентировать ребро  $(v, w)$  от  $v$  к  $w$ , положить  $tree = tree \cup \{(v, w)\}$ ,  $father(w) = v$ ,  $k = k + 1$ ,  $dfn(w) = k$ ,  $v = w$  и перейти к шагу 2. Иначе положить  $back = back \cup \{(v, w)\}$  и перейти к шагу 2.

Шаг 4. Положить  $v = father(v)$ . Если  $v \neq r$ , то перейти к шагу 2, иначе (при  $v = r$ ) останов – дерево поиска в глубину содержит ветви в массиве  $tree$ , при этом массив  $back$  содержит хорды, а массив  $depth$  – глубины вершин.

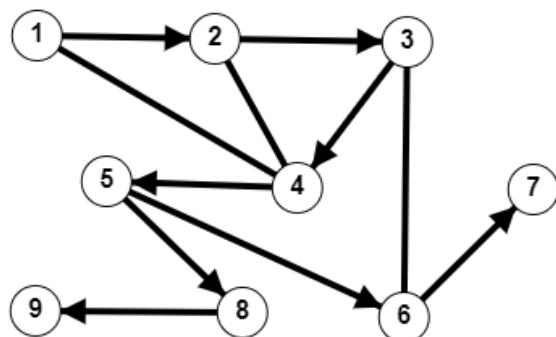
Таким образом, обход графа в соответствии с данным алгоритмом приводит к тому, что очередная текущая вершина получает следующий по порядку номер, который называется глубиной вершины. Дерево поиска в глубину – это ориентированный остов исходного графа, в котором каждая вершина имеет глубину.

Другим методом обхода графа является *поиск в ширину*. Если при поиске в глубину в качестве текущей вершины выбирается произвольная из окрестности вершины, которой на данном шаге приписана глубина, то при поиске в ширину выбор текущей вершины осуществляется только после того, как будет приписана ширина всем вершинам из окрестности предыдущей текущей вершины.

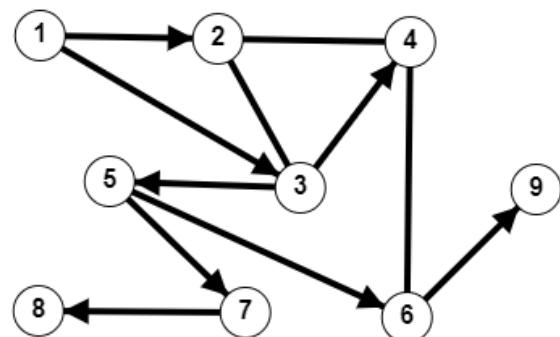
Поисковые процедуры являются базовой компонентой некоторых алгоритмов. Так, с помощью поиска в глубину можно проверить является ли граф связным, двудольным; осуществить топологическую сортировку вершин графа; найти базисные циклы и базисные разрезающие множества. Поиск в ширину используется в алгоритмах определения кратчайших путей в графе, если под длиной пути подразумевается количество дуг, которые этот путь составляют.



а) Заданный граф



б) Дерево поиска в глубину



в) Дерево поиска в ширину

Рисунок 4 – Поисковые алгоритмы для неорграфа

**Зачетное задание.** Задайте граф с 9 вершинами и 12 ребрами. С помощью алгоритма поиска в глубину в связном неорграфе постройте дерево поиска в глубину, выбрав в качестве корня любую вершину.

## ЛЕКЦИЯ

### Задача о кратчайшем остове

Пусть задан взвешенный неорграф  $G$ . Каждому его остову можно поставить в соответствие вес, равный сумме весов ребер, которые образуют данный остов. Требуется определить такой остов, который имеет минимальный суммарный вес. Такой остов называется *минимальным* или *кратчайшим*.

Рассмотрим некоторые алгоритмы нахождения кратчайшего остова в неорграфе.

#### Алгоритм ПРИМА построения кратчайшего остова взвешенного графа

Замечание. В алгоритме используются следующие множества:  $A$  – дополняемое на каждой итерации множество вершин графа, при этом на первой итерации выбирается некоторая произвольная вершина;  $T$  – дополняемое на каждой итерации множество ребер строящегося кратчайшего остова. В процессе работы алгоритма каждой вершине  $x_j$  ставится в соответствие метка вида  $[\alpha_j, \beta_j]$ , где  $\alpha_j$  – вершина из множества  $A$ , связанная с вершиной  $x_j$  ребром минимального веса по сравнению с другими вершинами из  $A$ ;  $\beta_j$  – вес этого ребра. Критерий останова может быть сформулирован в одном из следующих вариантов:  $|A|=n$  или  $|T|=n-1$ .

Шаг 1. Пусть  $G=(X, \Gamma)$  – взвешенный неорграф с матрицей весов  $C = (c_{ij} = c(x_i, x_j))_{n \times n}$ , где  $c(x_i, x_j)$  – вес ребра  $(x_i, x_j)$ . Выбрать произвольную вершину  $x_0$  и положить  $A=\{x_0\}$ ,  $T=\emptyset$ .

Шаг 2. Для каждой вершины  $x_j \notin A$  найти вершину  $\alpha_j \in A$  такую, что  $\beta_j = c(\alpha_j, x_j) = \min_{x_i \in A} \{c(x_i, x_j)\}$  и приписать вершине  $x_j$  метку  $[\alpha_j, \beta_j]$ . Если такой вершины  $\alpha_j$  не существует, то приписать вершине  $x_j$  метку  $[0, \infty]$ . Перейти к шагу 3.

Шаг 3. Выбрать вершину  $x^* \in A$ , которая имеет минимальную метку  $\beta_j$ . Пусть  $[\alpha^*, \beta^*]$  – метка вершины  $x^*$ , тогда положить  $A = A \cup \{x^*\}$ ,  $T = T \cup \{(\alpha^*, x^*)\}$ .  $|A|=n$ , то останов – ребра из  $T$  образуют кратчайший остов, иначе перейти к шагу 4.

Шаг 4. Для всех  $x_k \notin A$ , но  $x_k \in \Gamma(x^*)$ , проверить, нужно ли обновить метки и обновить в соответствии с правилом: если  $\beta_k > c(x_k, x^*)$ , то положить  $\alpha_k = x^*$ ,  $\beta_k = c(x_k, x^*)$ . Перейти к шагу 3.

**Пример.** Рассмотрим взвешенный граф, изображенный на рис. X.

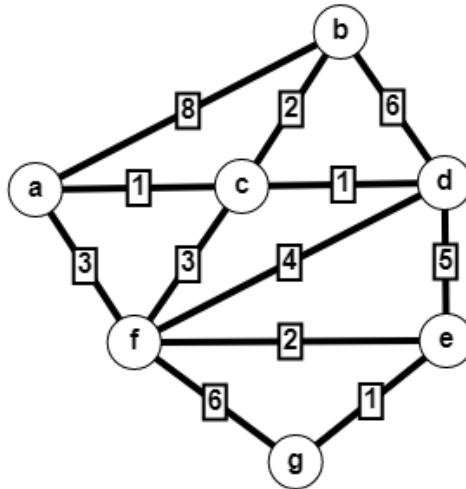


Рисунок 1 – Взвешенный неорграф

Результаты работы алгоритма представлены в следующей таблице.

Таблица 1 – Результаты работы алгоритма Прима

Итерации	$a$	$b$	$c$	$d$	$e$	$f$	$g$	$A$	$T$
1								$c$	$\emptyset$
2	$[c,1]$	$[c,2]$	*	$[c,1]$	$[0,\infty]$	$[c,3]$	$[0,\infty]$	$a$	$(c,a)$
3	*	$[c,2]$	*	$[c,1]$	$[0,\infty]$	$[c,3]$	$[0,\infty]$	$d$	$(d,c)$
4	*	$[c,2]$	*	*	$[d,5]$	$[c,3]$	$[0,\infty]$	$b$	$(b,c)$
5	*	*	*	*	$[d,5]$	$[c,3]$	$[0,\infty]$	$f$	$(f,c)$
6	*	*	*	*	$[f,2]$	*	$[f,6]$	$e$	$(e,f)$
7	*	*	*	*	*	*	$[e,1]$	$g$	$(g,e)$

На первой итерации выберем вершину  $c$  и поместим ее в множество  $A$ .  $T = \emptyset$ . Всем вершинам, не принадлежащим  $A$ , нужно приписать метки в соответствии с правилом (шаг 2), при этом первая часть метки ( $\alpha$ ) определяет вершину, принадлежащую множеству  $A$ , которая связана ребром минимального веса с данной вершиной, вторая часть метки ( $\beta$ ) равна длине этого ребра. На второй итерации рассматриваются вершины, которые смежны с вершиной  $c$ , но не принадлежат множеству  $A$ , т.е. вершины  $a,b,d,f$ . Поскольку в  $A$  единственная вершина, то  $\min$  не используется (выбирать нечего) и вершины получают следующие метки: вершина  $a$  –  $[c,1]$ ,  $b$  –  $[c,2]$ ,  $d$  –  $[c,1]$ ,  $f$  –  $[c,3]$ . Остальным вершинам приписывается метка  $[0,\infty]$ , поскольку они не смежны с вершиной  $c$ . Заметим, что вершины  $a$  и  $d$  имеют минимальные

значения (1) метки  $\beta$ , поэтому следующей текущей вершиной можно выбрать любую из них. Выберем вершину  $a$ . Рассмотрим вершины, не принадлежащие  $A$ , но входящие в окрестность вершины  $a$ , – это вершины  $b, f$ . Проверим, нужно ли обновлять метки. У вершины  $b$   $\beta = 2$ , но вес ребра  $(a,b) = 8 > 2$ , поэтому метку вершины  $b$  изменять не нужно. У вершины  $f$   $\beta = 3$  и вес ребра  $(a,f) = 3$ , поэтому метку вершины  $f$  тоже можно не изменять. Таким образом, множество  $A$  пополняется вершиной  $a$ , множество  $T$  – ребром  $(c,a)$ . На третьей итерации выберем вершину  $d$ , так как она имеет минимальное значение  $\beta$ . Вершинами из окрестности  $d$ , не принадлежащими  $A$ , являются  $b, e, f$ . Выясним, нужно ли менять их метки. У вершины  $b$   $\beta = 2$ , а вес ребра  $(d,b)$  равен 6, что больше 2, поэтому метку вершины  $b$  не меняем. Следующая вершина  $e$  имеет метку  $[0, \infty]$ . В соответствии с правилом изменим ее метку на метку  $[d, 5]$ . У вершины  $f$   $\beta = 3$ , а вес ребра  $(d,f)$  равен 4, что больше 3, поэтому метку вершины  $f$  не меняем. Итерация завершается включением вершины  $d$  в множество и  $(d,c)$  в множество  $T$ . В данной строке минимальное значение  $\beta$  имеет вершина  $b$ . В ее окрестности нет вершин, не принадлежащих множеству  $A$ , следовательно, метки вершин не изменятся. На четвертой итерации включаем вершину  $b$  в множество  $A$ , а ребро  $(b,c)$  – в множество  $T$ . На пятой итерации выбираем вершину  $f$ , поскольку она имеет минимальное значение  $\beta$ . Она смежна с вершинами  $e$  и  $g$ . Метка  $[0, \infty]$  вершины  $g$  заменяется на  $[f, 6]$ . У вершины  $e$   $\beta = 5$ , но к ней ближе находится вершина  $f$ , поэтому метка  $[d, 5]$  заменяется на метку  $[f, 2]$ . На данной итерации включаем вершину  $f$  в  $A$ , а ребро  $(f,c)$  – в множество  $T$ . На шестой итерации минимальное значение  $\beta$  имеет вершина  $e$ . В ее окрестность входит вершина  $g$ , которая не принадлежит  $A$  и имеет метку  $[f, 6]$ . Но вес ребра  $(e,g)$ , равный 1, меньше 6, поэтому изменим метку на  $[e, 1]$ . Включим на данной итерации вершину  $e$  в множество  $A$ , а ребро  $(e,f)$  – в множество  $T$ . На седьмой итерации осталась только одна метка  $[e, 1]$  у вершины  $g$ . На ее основе включаем вершину  $g$  в множество  $A$ , а ребро  $(g,e)$  – в множество  $T$ . Заметим, что все вершины включены в множество  $A$ , поэтому алгоритм завершает свою работу. Ребра в множестве  $T$  образуют кратчайший остов (рис. 2). Вес данного остова – 10.

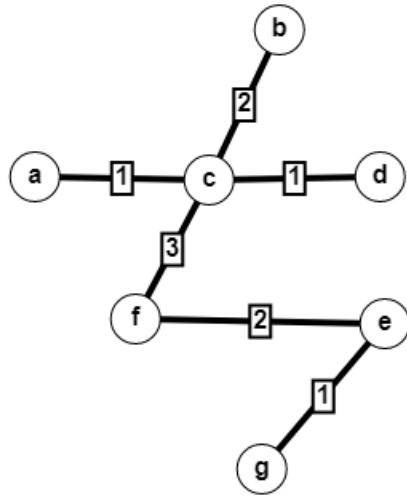


Рисунок 2 – Кратчайший остов

Для нахождения кратчайшего остова также используется *алгоритм Краскала*, включающий следующие шаги:

Шаг 1. Упорядочить ребра заданного взвешенного  $n$ -вершинного неорграфа по неубыванию весов, при этом ребра, имеющие одинаковый вес, располагаются в этом списке в произвольном порядке.

Шаг 2. Начав с графа, состоящего только из вершин, дополнить его  $(n - 1)$  ребром из списка, при этом не использовать ребра, которые с уже включенными ребрами образуют цикл.

**Пример.** Рассмотрим тот же граф, что и в предыдущем примере. Упорядочим его ребра по неубыванию (табл. 2).

Таблица 2 – Упорядочение ребер

$(a,c)$	$(c,d)$	$(g,e)$	$(c,b)$	$(f,e)$	$(a,f)$	$(c,f)$	$(f,d)$	$(b,d)$	$(f,g)$	$(a,b)$
1	1	1	2	2	3	3	4	6	6	8

Последовательно рассматривая ребра из списка, добавляем в граф, содержащий одни вершины (рис. 3а), каждый раз проверяя, не образуется ли цикл. Легко заметить, что ребра  $(a,c)$ ,  $(c,d)$ ,  $(g,e)$ ,  $(c,b)$  и  $(f,e)$  не образуют циклов, поэтому добавляем их в граф (рис. 3б). Следующие два ребра  $(a,f)$  и  $(c,f)$  имеют вес 3. Добавим, например, ребро  $(a,f)$  и получим кратчайший остов с весом 10 (рис. 3с)). Если добавить ребро  $(c,f)$ , то получим остов, изображенный на рис. 2.

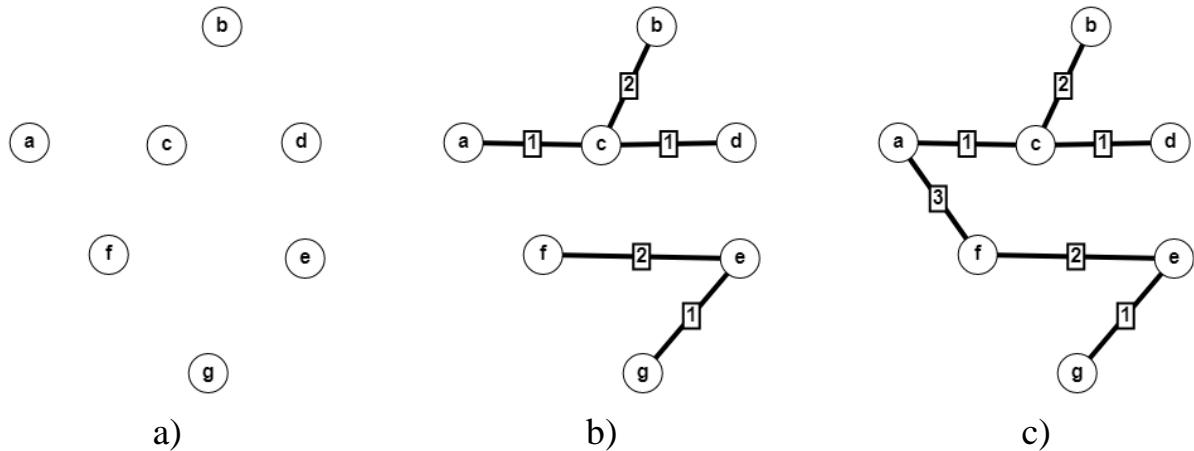


Рис. 3 – Построение кратчайшего остова с помощью алгоритма Краскала

Замечание 1. Может оказаться, что в заданном взвешенном неорграфе существует несколько кратчайших остовов с одинаковым суммарным весом. Результатом обоих алгоритмов является один из кратчайших остовов.

Замечание 2. Если необходимо найти остов максимального суммарного веса, то в алгоритме Краскала нужно рассматривать список, в котором ребра упорядочены по невозрастанию.

**Задание:** Модифицировать алгоритм Прима для нахождения остова максимального суммарного веса.

## ЛЕКЦИЯ

### Задача раскраски

Некоторые задачи, возникающие при планировании производства, составлении графиков осмотра и транспортировки товаров, могут быть представлены как известная в теории графов задача раскраски, которая формулируется следующим образом: для данного графа определить минимальное количество цветов, необходимых для раскраски вершин графа таким образом, чтобы никакие две смежные вершины не были окрашены в один цвет.

*Вершинной раскраской* неорграфа  $G = (X, U)$   $r$  цветами называется функция  $CV : X \rightarrow \{1, 2, \dots, r\}$ , такая, что для любых двух смежных вершин  $x_1$  и  $x_2$  имеем  $CV(x_1) \neq CV(x_2)$ . Функция  $CV(x)$  называется *функцией раскраски*.

Говорят, что граф  $G$  *допускает раскраску  $r$  цветами*, если существует функция *color* с областью значений  $\{1, 2, \dots, r\}$ .

*Хроматическим числом*  $\gamma(G)$  графа  $G$  называется наименьшее такое  $r$ , при котором граф  $G$  допускает вершинную раскраску  $r$  цветами. В этом случае граф называется  *$r$ -хроматическим*. Множество вершин одного цвета называется *одноцветным классом*.

Для хроматического числа различными исследователями предложены оценки, перечислим некоторые из них:

нижние оценки  $\gamma(G)$ :

$$\gamma(G) \geq \frac{n}{\gamma(\bar{G})};$$

$$\gamma(G) \geq \frac{n^2}{n^2 - 2m};$$

если граф содержит полный подграф на  $m$  вершинах, то  $\gamma(G) \geq m$ ;

верхние оценки для  $\gamma(G)$ :

$$\gamma(G) \leq n + 1 - \gamma(\bar{G});$$

$$\gamma(G) \leq 1 + \max_{\{i: x_i \in X\}} \{d_i + 1\}.$$

Кроме того, для хроматических чисел графа  $G$  и его дополнения  $\bar{G}$  известны следующие неравенства:

$$2\sqrt{n} \leq \gamma(G) + \gamma(\bar{G}) \leq n + 1;$$

$$n \leq \gamma(G) \cdot \gamma(\bar{G}) \leq \left( \frac{n+1}{2} \right)^2.$$

*Реберной раскраской* неорграфа  $G = (X, U)$  в цветами называется функция  $CE : U \rightarrow \{1, 2, \dots, v\}$ , такая, что для любых двух ребер  $u_1$  и  $u_2$  имеем  $CE(u_1) \neq CE(u_2)$ .

*Хроматическим индексом* графа  $G = (X, U)$  называется наименьшее такое  $v$ , что граф  $G$  допускает реберную раскраску  $v$  цветами.

Заметим, что задача реберной раскраски решается как задача вершинной раскраски, если от заданного графа  $G$  перейти к его реберному графу  $L(G)$ , поэтому задача реберной раскраски является избыточной по постановке.

Для графов  $G_1 = (X, U_1)$  и  $G_2 = (X, U_2)$  с одним и тем же множеством вершин  $X$  имеем  $\gamma(G_1 \cup G_2) = \gamma(G_1) + \gamma(G_2)$ .

Граф называется *критическим*, если удаление любой из его вершин вместе с инцидентными ребрами приводит к графу с меньшим хроматическим числом.

Известно, что всякий критический граф  $G$ , являющийся  $r$ -хроматическим, обладает следующими свойствами:

- a)  $G$  связен;
- b) степень каждой вершины не меньше  $(r - 1)$ ;
- c) не существует вершины, удаление которой приводит к несвязному графу.

Рассмотрим некоторые процедуры раскрашивания графа.

#### Эвристическая процедура раскрашивания графа

Замечание: Данный алгоритм определяет оценку  $y(G)$  хроматического числа  $\gamma(G)$  графа  $G$ , которая удовлетворяет соотношению

$$\gamma(G) \leq y(G) \leq \max \min_{\{i\}} \{i, d(i) + 1\},$$

где  $d(i)$  – степень  $i$ -ой вершины, причем индексация вершин  $x_i$  осуществляется в соответствии с убыванием их степеней.

Шаг 1. Пусть  $G = (X, U)$  – данный график. Для каждой вершины  $x_i$  определить ее степень  $d_i$ , и упорядочить вершины в порядке невозрастания их степеней, в результате будет сформирован список вершин для просмотра.

Шаг 2. Пусть  $r$  – номер текущего цвета. Просматривая список от начала к концу будем раскрашивать вершины по правилу: первая неокрашенная вершина в списке окрашивается в цвет  $r$ , затем в этот же цвет окрашивается любая другая вершина, которая не смежна с уже окрашенными в данный цвет  $r$ .

Шаг 3. Повторять Шаг 2 для  $r \geq 1$  до тех пор, пока все вершины не будут окрашены, при этом количество использованных цветов определяет приближенное значение хроматического числа; вершины, окрашенные в один цвет образуют одноцветные классы.

**Пример.** Рассмотрим граф, изображенный на рис. 2. Для каждой вершины определим ее степень и составим список вершин.

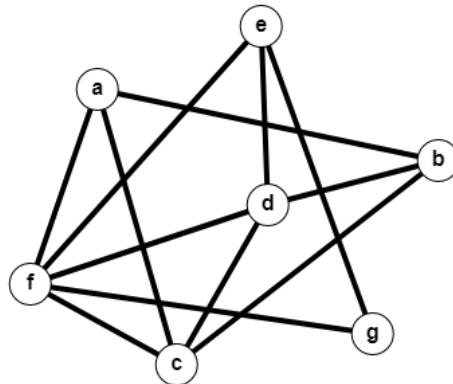


Рисунок 1 – Неорграф  $G$

Вершины	$f$	$c$	$d$	$a$	$b$	$e$	$g$
Степени	5	4	4	3	3	3	2
Цвет	1	2	3	3	1	2	3

Вершину  $f$  окрасим в цвет №1. Следующие вершины  $c, d, a$  смежны с вершиной  $f$ , поэтому их пропускаем. Следующая вершина  $b$  не смежна с  $f$ , поэтому ее окрашиваем в цвет №1. Оставшиеся вершины  $e$  и  $g$  смежны с  $f$  – они не могут быть окрашены в цвет №1. Не все вершины окрашены, поэтому выбираем следующий цвет №2 и возвращаемся к началу списка. Первая неокрашенная вершина –  $c$ , окрашиваем ее в цвет №2. Следующие в списке неокрашенные вершины  $d$  и  $a$  смежны с  $c$ , поэтому их пропускаем. Следующая неокрашенная вершина  $e$  не смежна с  $c$ , поэтому ее также окрашиваем в цвет №2. Вершина  $g$  смежна с  $e$ , поэтому не может быть окрашена в цвет №2. Снова не все вершины окрашены, поэтому выбираем краску №3 и возвращаемся к первой неокрашенной вершине. Это вершина  $d$ , окрашиваем ее в цвет №3. Следующую неокрашенную вершину также окрашиваем в цвет №3, поскольку она не смежна с  $d$ . Следующая неокрашенная в списке вершина  $g$  не смежна ни с  $d$ , ни с  $a$ . Таким образом, все вершины оказались окрашенными, поэтому процесс раскрашивания закончен. Количество использованных красок равно 3, поэтому хроматическое число заданного графа равно 3.

Замечание. Процедура позволяет определить лишь приближенное значение хроматического числа, причем примененная к одному и тому же графу она может дать различные варианты раскраски. На рис.1 представлены различные раскраски графа, при этом в первом случае хроматическое число равно 4, а во втором – 3.

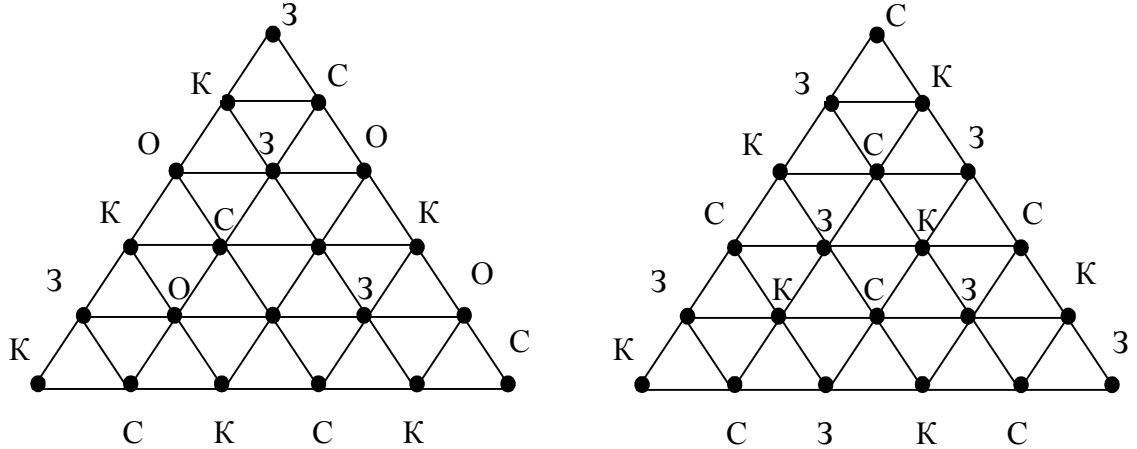


Рисунок 2 – Различная раскраска графа  
(цвета: З – зеленый, С – синий, К – красный, О – оранжевый)

#### Модификация процедуры раскрашивания графа

Шаг 1. Положить  $r=1$  (здесь  $r$  – приближенное значение хроматического числа).

Шаг 2. Пусть  $G(X)$  – граф на множестве вершин  $X$ . Для каждой вершины  $x_i$  определить степень  $d_i$  и расположить вершины в порядке невозрастания степеней.

Шаг 3. Первая вершина окрашивается в цвет  $r$ ; затем, двигаясь по списку, в цвет  $r$  окрашивается всякая вершина, которая не смежна с другой, уже окрашенной в этот цвет. Пусть  $S_r$  – множество вершин, окрашенных в цвет  $r$ .

Шаг 4. Положить  $X = X \setminus S_r$ . Если  $X \neq \emptyset$ , то на множестве вершин  $X$  получить порожденный подграф  $G(X)$ . Положить  $r=r+1$  и перейти к шагу 2. Иначе останов – все вершины графа окрашены,  $r$  приближенное значение хроматического числа.

Данный алгоритм целесообразно использовать, если имеется множество вершин с одинаковыми степенями. Заметим, что после удаления окрашенных вершин к полученному подграфу вновь применяется основная процедура, т.е. определяются степени вершин и формируется новый список.

Заметим, что одноцветный класс представляет собой множество попарно не смежных вершин. Такие множества называются *независимыми*.

Независимое множество называется *максимальным*, если оно не является собственным подмножеством никакого другого независимого множества. Заданный граф может иметь целое семейство максимальных независимых множеств. Мощность независимого множества максимальной мощности (наибольшего независимого множества) называется *числом независимости*.

Если известно, что граф  $G = (X, U)$  является  $r$ -хроматическим, то он может быть окрашен с использованием  $r$  красок на основе следующей процедуры: сначала в один цвет окрашивается некоторое максимальное независимое множество  $S(X)$ , затем окрашенные вершины вместе с инцидентными вершинами удаляются из графа и в следующий цвет окрашивается множество  $S(X \setminus S(X))$  и т.д. до тех пор пока не будут раскрашены все вершины.

**Пример.** Рассмотрим граф на рис. 3.

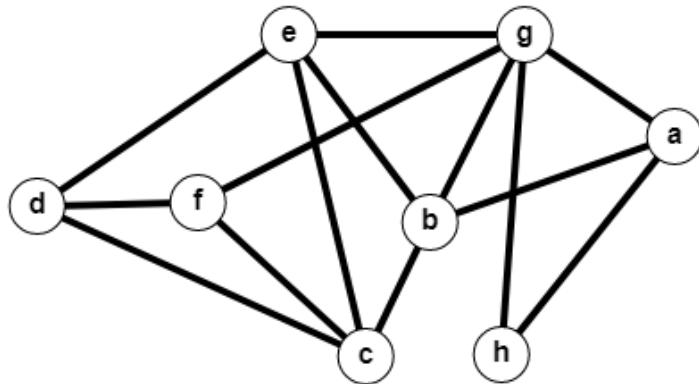


Рисунок 3 – Неорграф для раскраски

Найдем в нем произвольное максимальное независимое множество, например,  $S_1 = \{b, f, h\}$  и окрасим его вершины в цвет 1. Удалим вершины  $b, f$  и  $h$  из графа вместе с инцидентными ребрами. Получим его подграф на рис. 4.

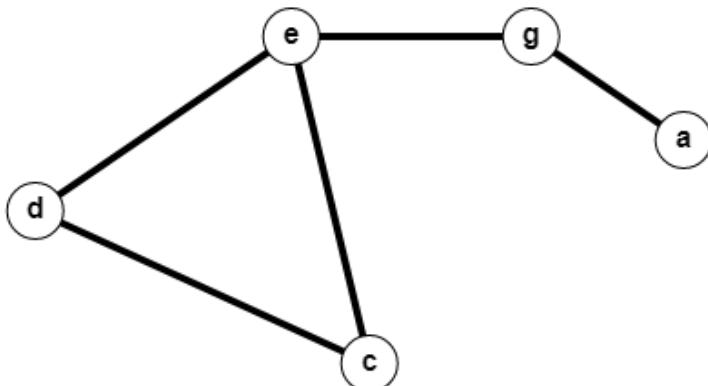


Рисунок 4 – Граф после удаления вершин  $S_1 = \{b, f, h\}$

В полученном подграфе вновь найдем максимальное независимое множество, например,  $S_2 = \{c, a\}$ , и окрасим его вершины в цвет 2. На рис. 5 представлен подграф, полученный после удаления окрашенных вершин.

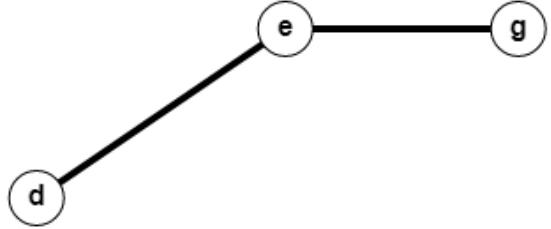


Рисунок 5 – Граф после удаления вершин  $S_2 = \{c, a\}$

На рис. 5 видно, что потребуется еще два цвета, чтобы окрасить вершины. Здесь  $S_3 = \{d, g\}$  – независимое множество окрашивается в цвет 3. После удаления этих вершин остается единственная изолированная вершина,  $S_4 = \{e\}$  и вершина  $e$  окрашивается в цвет 4. Таким образом, хроматическое число равно 4; одноцветные классы:  $\{b, f, h\}$ ,  $\{c, a\}$ ,  $\{d, g\}$ ,  $\{e\}$ .

Заметим, что выбор максимальных независимых множеств на каждом шаге произволен, поэтому алгоритм находит некоторую раскраску. При другом выборе в общем случае можно получить другую раскраску.

## Лекция

### Задача о максимальном потоке

Пусть  $G = (X, U)$  – орграф без петель с источником  $s$  и стоком  $t$ . Через  $(i, j)$  будем обозначать дугу с началом в вершине  $i$  и концом в вершине  $j$ .

*Источником* называется вершина  $s$ , из которой достижимы все вершины графа; *сток* определяется двойственным образом, т.е. сток – это вершина  $t$ , которая достижима из любой вершины графа. Каждой дуге поставим в соответствие число  $r_{ij}$ , называемое *пропускной способностью дуги*  $(i, j)$ .

Множество чисел  $\{x_{ij}\}_{(i,j) \in U}$ , определенных на дугах  $(i, j) \in U$ , называют *потоками* на дугах графа, если выполняются следующие условия для всех  $(i, j) \in U$ :

a) условие допустимости потока  $0 \leq x_{ij} \leq r_{ij}$ ;

b) условие баланса  $\sum_{\{j: j \in \Gamma(i)\}} x_{ij} - \sum_{\{k: k \in \Gamma^{-1}(i)\}} x_{ij} = \begin{cases} -V, & \text{если } i = s, \\ V, & \text{если } i = t, \\ 0, & \text{иначе.} \end{cases}$

Заметим, что согласно условию b) допустимый поток на дуге  $(i, j)$  не может превышать ее пропускной способности. Условие a) означает, что для любой вершины, не совпадающей с источником и стоком, входящий поток равен выходящему, а для графа в целом – поток, выходящий из вершины  $s$ , равен потоку, выходящему в вершину  $t$ . Задача о максимальном потоке заключается в нахождении такого множества потоков  $x_{ij}$  на дугах, чтобы величина  $V$  была максимальной.

Известными модификациями задачи о максимальном потоке являются:

a) Если вместо одного источника и одного стока рассмотреть несколько заданных источников и стоков, причем между различными источниками и стоками протекают потоки различных продуктов, то задача максимизации суммы всех потоков между источниками и стоками называется задачей о *многопродуктовом потоке*. В этой задаче пропускная способность дуги является ограничением для суммы всех потоков всех видов продуктов через эту дугу.

b) Если рассмотреть граф, в котором выходной поток дуги равен ее входному потоку, умноженному на некоторое неотрицательное число, то задачу о максимальном потоке от  $s$  к  $t$  называют задачей о *потоках с выигрышами*. В такой задаче потоки могут и «поглощаться», и «порождаться» самим графом, так что поток, входящий в источник  $s$ , и поток, выходящий из  $t$ , в общем случае изменяются независимо.

c) Если каждой дуге приписаны верхняя и нижняя границы потока через дугу, а также стоимость единицы потока, протекающего по дуге, то возникает задача нахождения допустимого потока минимальной стоимости от источника  $s$  к стоку  $t$ .

Решение задачи о максимальном потоке основано на *теореме Форда-Фалкерсона*: величина максимального потока из  $s$  в  $t$  равна величине минимального разреза  $\langle R^*, \bar{R}^* \rangle$ , отделяющего  $s$  от  $t$ .

*Разрезом*  $\langle R, \bar{R} \rangle$  в орграфе  $G$  называется минимальное множество дуг при удалении которых граф разбивается на две компоненты связности. Если  $R \subset X$  и  $\bar{R} \subset X$  – множества вершин соответствующих компонент, образующих разбиение множества  $X$ , то разрез  $\langle R, \bar{R} \rangle$  включается такие дуги  $(i', j')$ , у которых начальная вершина  $i'$  находится во множестве  $R$ , а конечная вершина  $j'$  – во множестве  $\bar{R}$ , т.е.  $\langle R, \bar{R} \rangle = \{(i, j) : i \in R, j \in \bar{R}\}$ .

*Разрез*  $\langle R, \bar{R} \rangle$  *отделяет*  $s$  от  $t$ , если  $s \in R$  и  $t \in \bar{R}$ .

*Величиной разреза* называется сумма пропускных способностей всех дуг графа, образующих разрез, т.е.

$$V(\langle R, \bar{R} \rangle) = \sum_{\{(i, j) : i \in R \text{ и } j \in \bar{R}\}} r_{ij}.$$

Минимальный разрез – это разрез  $\langle R^*, \bar{R}^* \rangle$  с наименьшим значением

$$V^* = \min_{\{\langle R, \bar{R} \rangle\}} \{V(\langle R, \bar{R} \rangle)\}.$$

Эта задача и ее варианты могут возникать во многих практических приложениях, например при определении максимальной интенсивности транспортного потока между двумя пунктами на карте дорог, представляемой графом. В этом примере решение задачи о максимальном потоке укажет ту часть сети дорог, которая образует «узкое место» транспортного потока между двумя указанными концевыми пунктами.

Пусть  $G$  – сеть с источником  $s$  и стоком  $t$ . Каждой дуге  $(i, j)$  соответствует пропускная способность  $r_{ij} > 0$  и величина потока  $x_{ij} = 0$ .

Для нахождения решения задачи о максимальном потоке используется следующий алгоритм.

#### *Алгоритм Форда-Фалкерсона*

Замечание: Данный алгоритм основан на приписывании вершинам меток, при этом данная вершина может находиться в одном из трех состояний:

- вершина помечена и просмотрена, т.е. все смежные вершины также помечены;
- вершина помечена, но не просмотрена, т.е. не все смежные с ней вершины получили метки;
- вершина не имеет метки.

Метка произвольной вершины  $j$  состоит из двух частей  $[\pm i, \delta]$  и означает, что поток вдоль дуги  $(i, j)$  может быть увеличен на величину  $\delta$ , если перед  $i$  стоит знак плюс, или уменьшен на ту же самую величину, если перед  $i$  стоит знак минус. В обоих случаях  $\delta$  задает максимальную величину до-

полнительного потока, который может протекать по некоторому пути от источника  $s$  к вершине  $i$ .

Пусть  $i$  – непросмотренная вершина с меткой  $[\pm l, \delta_i]$ , тогда метки смежных вершин формируются по следующим правилам:

a) каждой непомеченной вершине  $j \in \Gamma(i)$ , для которой  $x_{ij} < r_{ij}$ , присвоить метку  $[+i, \delta_j]$ , где  $\delta_j = \min\{\delta_i, r_{ij} - x_{ij}\}$ ;

b) каждой непомеченной вершине  $k \in \Gamma^{-1}(i)$ , для которой  $x_{ki} > 0$ , присвоить метку  $[-i, \delta_k]$ , где  $\delta_k = \min\{\delta_i, x_{ki}\}$ .

Каждая итерация алгоритма включает прямой и обратный шаги. Прямой шаг заключается в приписывании меток всем вершинам, начиная с источника  $s$  и заканчивая стоком  $t$ . В результате будет определена величина  $\delta_t$  дополнительного потока, который можно «пропустить» от  $s$  к  $t$ . На обратном шаге с учетом имеющихся меток вершин поток увеличивается или уменьшается на одну и ту же величину  $\delta_t$ . Реализация каждой итерации приводит к появлению в сети хотя бы одной *насыщенной дуги* – такой дуги  $(i, j)$ , у которой поток равен пропускной способности, т.е.  $x_{ij} = r_{ij}$ . Итерации повторяются до тех пор, пока можно пометить вершину  $t$ . Вершина  $s$  на каждой итерации имеет метку  $[+s, \infty]$ , поэтому она всегда помечаема. Если возникает ситуация, когда некоторые вершины, в том числе и вершину  $t$ , пометить не удается, то формируется разрез и находится его величина. Согласно теореме Форда-Фалкерсона, разрез определяет то узкое место в сети, которое пропускает поток максимальной величины.

Рассмотрим более подробно шаги алгоритма.

Шаг 1. Присвоить источнику  $s$  метку  $[+s, \infty]$ .

Шаг 2. Выбрав непросмотренную вершину  $i$  с меткой  $[\pm l, \delta_i]$ , пометить непомеченные вершины из множеств  $\Gamma(i)$  и  $\Gamma^{-1}(i)$  по правилам а) и б) соответственно.

Шаг 3. Повторять шаг 2 до возникновения одной из следующих ситуаций: 1) вершина  $t$  окажется помеченной и тогда перейти к шагу 4; 2) вершина  $t$  не помечена, но никаких меток по правилам а) и б) расставить нельзя. Если при этом  $R^*$  – множество помеченных вершин, а  $\bar{R}^*$  – множество непомеченных вершин, то  $\langle R^*, \bar{R}^* \rangle$  – искомый минимальный разрез, а сумма пропускных способностей на дугах этого разреза есть величина максимального потока  $V^*$ .

Шаг 4. Пусть метка вершины  $t$  имеет вид  $[+z, \delta_t]$ , то положить  $x = t$  и перейти к шагу 5.

Шаг 5. Если метка в вершине  $x$  имеет вид  $[+y, \delta_x]$ , то увеличить поток вдоль дуги  $(y, x)$  на величину  $\delta_t$ . Если метка в вершине  $x$  имеет вид  $[-y, \delta_x]$ , то уменьшить поток вдоль дуги  $(x, y)$  на величину  $\delta_t$ . Перейти к шагу 6.

Шаг 6. Если  $y = s$ , то стереть все метки и перейти к шагу 1, иначе положить  $x = y$  и перейти к шагу 5.

Рассмотренный выше алгоритм можно модифицировать, помечая на каждой итерации не все вершины графа, а только те, которые принадлежат некоторому пути. Путь из  $s$  в  $t$ , который не содержит насыщенных дуг, назовем *допустимым*. Путь из  $s$  в  $t$ , содержащий хотя бы одну насыщенную дугу, называется *запрещенным*. Как только все пути становятся запрещенными, стратегия меняется: необходимо расставить все возможные метки на основе правил а) и б), тогда множество помеченных вершин – это  $R^*$ , а множество непомеченных –  $\bar{R}^*$ .

#### *Модификация алгоритма Форда-Фалкерсона*

Шаг 1. Присвоить источнику  $s$  метку  $[+s, \infty]$ . Положить  $m=0$  ( $m$  – номер итерации).

Шаг 2. Выбрать любой допустимый путь  $\mu_m$  из  $s$  в  $t$  и пометить все его вершины в соответствии с правилом а). Если допустимых путей нет, то перейти к шагу 4.

Шаг 3. Пусть метка вершины  $t$  имеет вид  $[+z, \delta_t]$ . Увеличить поток вдоль дуг данного пути на величину  $\delta_t$ , при этом появится хотя бы одна насыщенная дуга, и данный путь  $\mu_m$  станет запрещенным. Кроме того, запрещенными станут все пути, которые включают появившиеся на данной итерации насыщенные дуги. Положить  $m = m + 1$  и перейти к шагу 2.

Шаг 4. Начиная с вершины  $s$ , метка которой на каждой итерации есть  $[+s, \infty]$ , расставить все возможные метки с помощью правил а) и б). Включить в множество  $R^*$  все помеченные вершины, а в  $\bar{R}^*$  – все непомеченные. Сформировать разрез  $\langle R^*, \bar{R}^* \rangle$  и найти его величину  $V^*$ .

**Пример.** Рассмотрим сеть, изображенную на рис. 1.

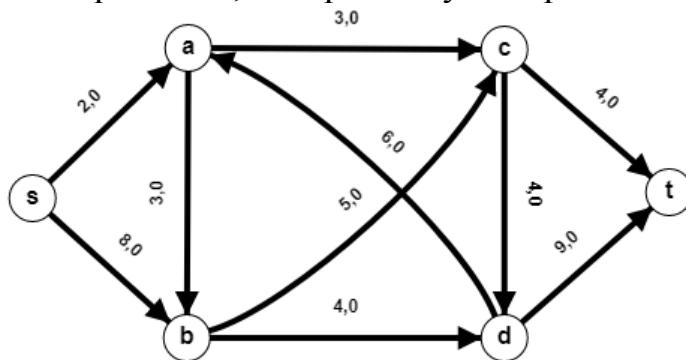


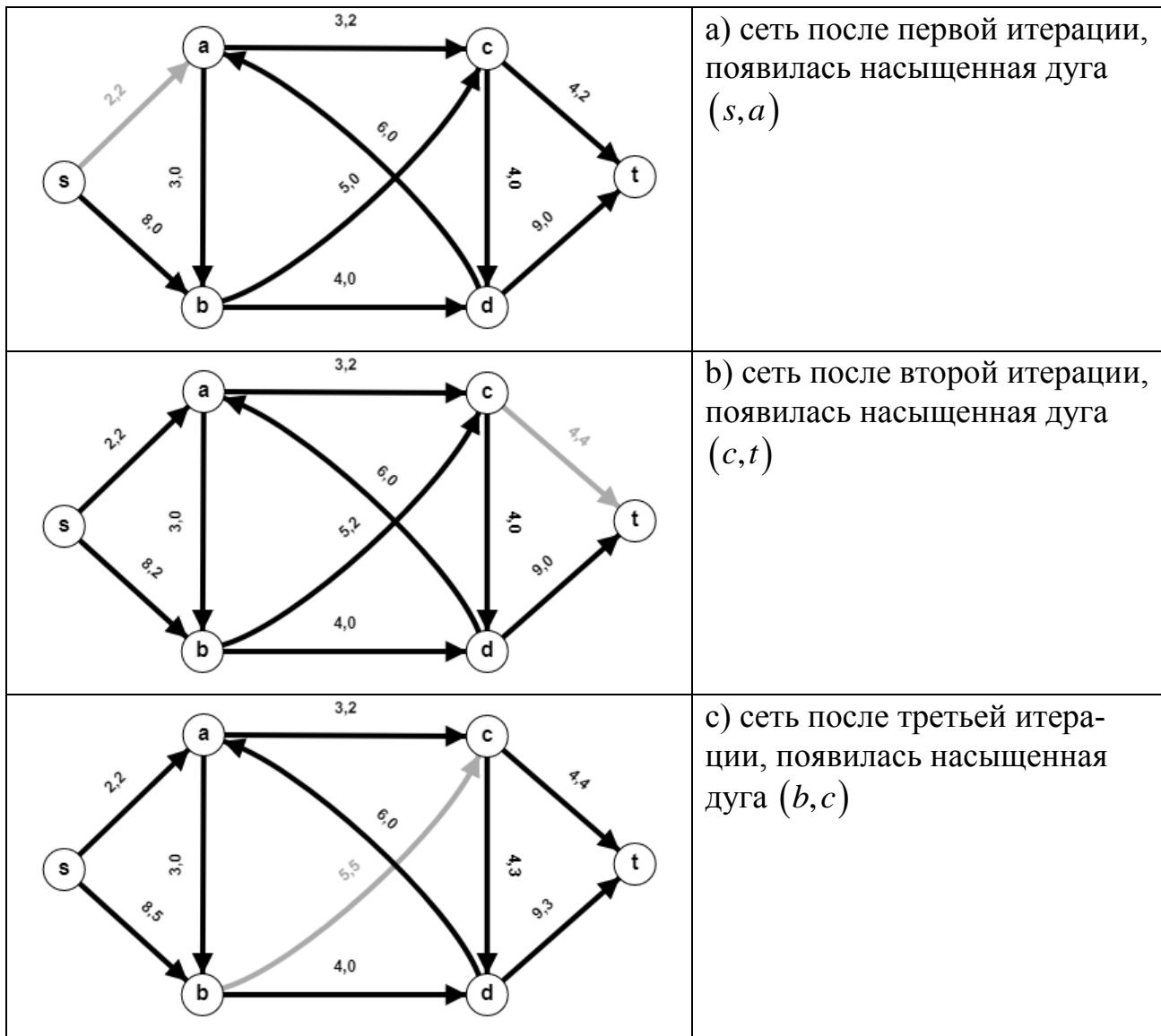
Рисунок 1 – Взвешенный орграф, дугам соответствуют пропускные способности

Работа алгоритма представлена в расчетной таблице (табл. 1). На каждой итерации выбирается некоторый путь из  $s$  в  $t$ , который не содержит насыщенные дуги, запрещающие некоторые пути. Метки, приписываемые

вершинам, позволяют определить величину дополнительного потока вдоль выбранного пути.

Таблица 1 – Метки вершин по итерациям

Итерация	1	2	3	4
$s$	$[0, \infty]$	$[0, \infty]$	$[0, \infty]$	$[0, \infty]$
$a$	$[+s, 2]$			
$b$		$[+s, 8]$	$[+s, 6]$	$[+s, 3]$
$c$	$[+a, 2]$	$[+b, 5]$	$[+b, 3]$	
$d$			$[+c, 3]$	$[+b, 3]$
$t$	$[+c, 2]$	$[+c, 2]$	$[+d, 3]$	$[+d, 3]$
$\delta_t$	2	2	3	3
<b>Путь</b>	$s \rightarrow a \rightarrow c \rightarrow t$	$s \rightarrow b \rightarrow c \rightarrow t$	$s \rightarrow b \rightarrow c \rightarrow d \rightarrow t$	$s \rightarrow b \rightarrow d \rightarrow t$



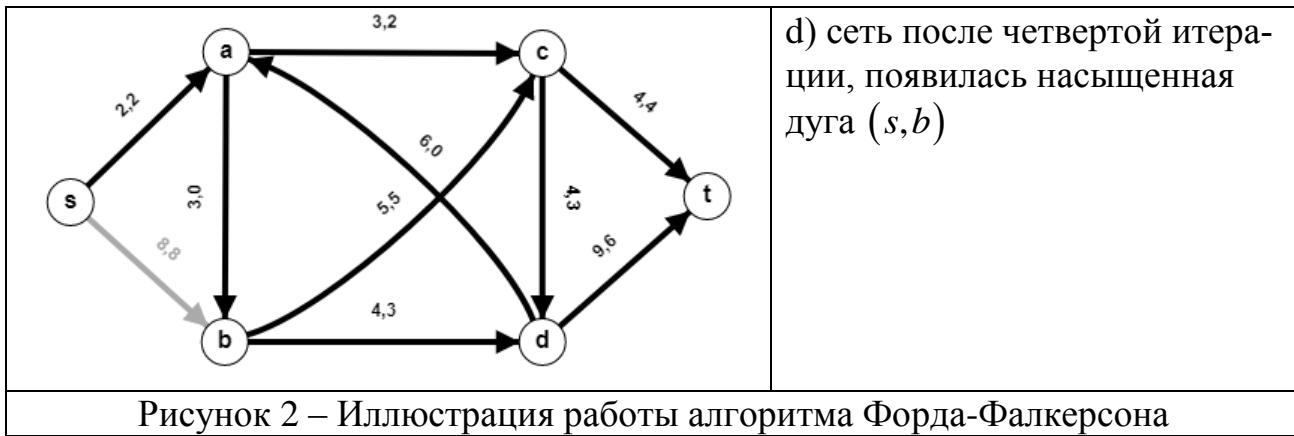


Рисунок 2 – Иллюстрация работы алгоритма Форда-Фалкерсона

В графе, который представлен на рис. 3, представлены все насыщенные дуги, которые получены в результате работы алгоритма.

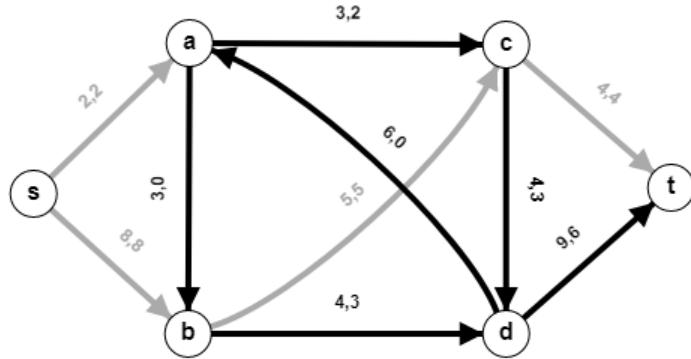


Рисунок 3 – В сети все пути являются запрещенными

На пятой итерации припишем вершине  $s$  метку  $[0, \infty]$ . Заметим, что из вершины  $s$  выходят две насыщенные дуги (рис. 3), следовательно, другие вершины пометить нельзя. Тогда сформируем множества  $R^* = \{s\}$ ,  $\bar{R}^* = \{a, b, c, d, t\}$ . Разрез содержит следующие дуги  $\langle R^*, \bar{R}^* \rangle = \{(s, a), (s, b)\}$ , его величина определяется как сумма потоков на его дугах, т.е.  $V^* = 2 + 8 = 10$ .

Рассмотрим задачу о максимальном потоке при условии, что сеть является двунаправленной, то есть вершины соединяются двумя противоположно направленными дугами и каждой дуге соответствует своя пропускная способность. Требуется определить максимальный поток из источника  $s$  в сток  $t$ .

Для решения данной задача используется следующий алгоритм.

Шаг 1. Пусть  $G = (X, U)$  – двунаправленный взвешенный граф,  $|X| = n$ ,  $s$  – источник,  $t$  – сток. Матрица пропускных способностей  $P = (p_{ij})_{n \times n}$  определяет пропускную способность  $p_{ij}$  дуги  $(x_i, x_j)$ . Введем счетчик итераций  $k$  и положим  $k = 1$ ,  $P^{(k)} = P$ .

Шаг 2. Выберем произвольный путь  $\mu_k$  из  $s$  в  $t$ . Элементы матрицы  $P^{(k)}$ , соответствующие дугам выбранного пути, пометить знаком «–»; элементы, соответствующие обратным дугам, пометить знаком «+».

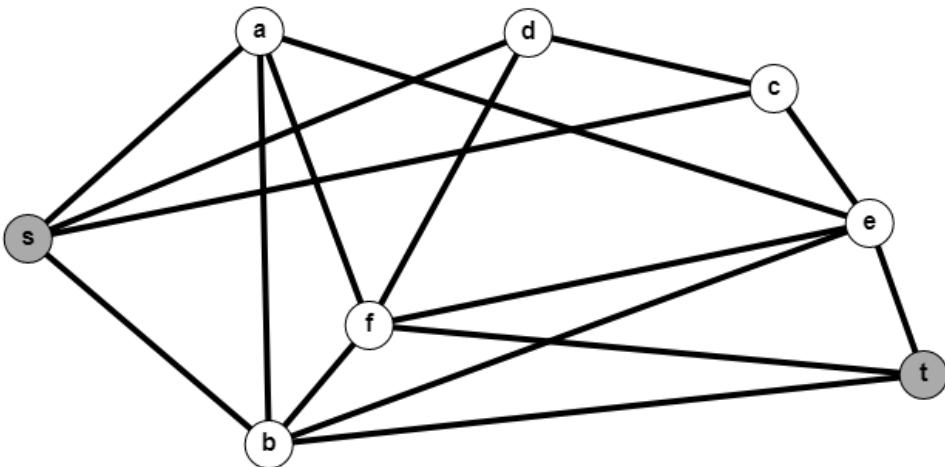
Шаг 3. Определить пропускную способность  $\theta_k$  выбранного пути  $\mu_k$  как минимум из элементов матрицы  $P^{(k)}$ , помеченных знаком « $-$ ». Положить  $k \rightarrow k + 1$  и сформировать новую матрицу  $P^{(k)}$ , прибавляя величину  $\theta_k$  к элементам, помеченным знаком « $+$ », и вычитая из элементов, помеченных знаком « $-$ ».

Шаг 4. Если столбец, соответствующий  $t$ , содержит только 0, то перейти к шагу 5, иначе перейти к шагу 2.

Шаг 5. Построить матрицу  $P^*$ , включая в нее только положительные элементы  $p_{ij}^{(1)} - p_{ij}^{(k)}$ , а остальные элементы положить равными 0. Построить сеть с весами из  $P^*$ .

Замечание. Для проверки правильности вычислений используются следующие условия: а) поток, выходящий из  $s$ , равен потоку, входящему в  $t$ ; б) для любой другой вершины входящий поток равен выходящему.

**Задание.** Решить задачу о максимальном потоке для сети, представленной на рис. 6.



Матрица весов  $P$  задана в следующей таблице.

$P$	$s$	$a$	$b$	$c$	$d$	$e$	$f$	$t$
$s$	0	3	4	3	3	0	0	0
$a$	2	0	3	0	0	2	2	0
$b$	3	2	0	0	0	3	4	3
$c$	3	0	0	0	1	4	0	0
$d$	2	0	0	1	0	0	2	0
$e$	0	3	3	2	0	0	0	4
$f$	0	2	4	0	1	2	0	4
$t$	0	0	3	0	0	5	4	0

## ЛЕКЦИЯ

### ЭКСТРЕМАЛЬНЫЕ ПУТИ НА ГРАФАХ

Рассмотрим некоторые экстремальные задачи на графах.

**Задача о кратчайшем пути** заключается в нахождении пути минимальной длины от заданной вершины  $s$  к заданной вершине  $t$  в ориентированном графе. Если граф является взвешенным, то под *длиной пути* подразумевается сумма длин дуг, которые этот путь составляют; иначе длина пути – это количество дуг, образующих этот путь. Обобщениями задачи о кратчайшем пути являются:

а) задача нахождения кратчайших путей между вершиной  $s$  и всеми другими вершинами графа;

б) задача нахождения кратчайших путей между всеми парами вершин.

Если определить кратчайшие пути от заданной вершины  $s$  ко всем остальным вершинам графа, то можно построить дерево кратчайших путей. При решении задачи о кратчайшем пути во взвешенном графе следует рассматривать три случая:

i) все дуги имеют неотрицательный вес;

ii) некоторые дуги имеют отрицательный вес, но в графе не существует контуров с суммарным отрицательным весом;

iii) в сети присутствует один или несколько контуров с отрицательным суммарным весом.

В последнем случае задачу о кратчайшем пути решить нельзя, но можно обнаружить контуры с отрицательным суммарным весом. В случае а) для поиска кратчайшего пути между заданными вершинами используется алгоритм Дейкстры; для определения кратчайшего пути с заданным числом дуг – алгоритм Форда-Беллмана; для поиска кратчайших путей между заданной вершиной  $s$  и всеми остальными может быть использован как алгоритм Дейкстры, так и его модификации. В случаях б) и в) используется алгоритм Форда-Мура. Для определения кратчайшего пути между всеми парами вершин применим алгоритм Флойда. Рассмотрим некоторые из этих алгоритмов.

В алгоритмах используются следующие обозначения:  $\Gamma(u) = \{v : u \rightarrow v\}$  – множество вершин  $v$ , в которые ведут дуги из  $u$ ;  $\Gamma^{-1}(u) = \{w : w \rightarrow u\}$  – множество вершин  $w$ , из которых дуги ведут в вершину  $u$ .

#### *Алгоритм Дейкстры*

**Комментарий.** В процессе работы алгоритма каждой вершине приписывается метка. Метки делятся на временные и постоянные (к таким приписывается символ  $+$ ). Считается, что метка  $l(s) = 0^+$  начальной вершины пути  $s$  всегда является постоянной. На каждой итерации некоторая вершина с постоянной меткой назначается текущей (обозначается  $p$ ). Изменить временные метки можно только из текущей вершины. Если вершина получила постоян-

ную метку, то дальше она уже не рассматривается. Критерием останова является достижение конечной вершины пути  $t$ , т.е. возникновение ситуации  $p = t$ .

Шаг 1 (*присвоение начальных меток*). Пусть  $G = (X, \Gamma)$  – ориентированный граф с матрицей весов  $C = (c_{ij} = c(x_i, x_j))_{n \times n}$ , где  $n = |X|$ ,  $c(x_i, x_j) \geq 0$  – вес дуги  $(x_i, x_j)$ ;  $s$  – начальная вершина пути;  $t$  – конечная вершина пути. Вершине  $s$  приписать метку  $l(s) = 0$  и считать ее постоянной (пометив, например, символом +). Всем остальным вершинам  $x \neq s$  приписать метку  $l(x) = \infty$  и считать эти метки временными. Положить  $p = s$ .

Шаг 2 (*обновление меток*). Для всех вершин  $x \in \Gamma(p)$  из окрестности текущей вершины  $p$  с временными метками изменить метки в соответствии с правилом

$$l(x) = \min\{l(x), l(p) + c(p, x)\}.$$

Шаг 3 (*превращение временных меток в постоянные*). Пусть выделено множество вершин с временными метками на данной итерации. Найти такую вершину  $x^*$ , для которой  $l(x^*) = \min_x\{l(x)\}$ . Считать метку вершины  $x^*$  постоянной, и положить  $p = x^*$ .

Шаг 4 (*проверка критерия останова*). Если  $p \neq t$ , то перейти к шагу 2, иначе  $l(p)$  – длина кратчайшего пути. Для нахождения самого пути применяется следующая процедура: положить  $y = t$ ; из всех вершин  $x \in \Gamma^{-1}(y)$  выделить такую вершину  $x$ , для которой выполняется соотношение  $l(y) = l(x) + c(x, y)$ ; затем в качестве  $y$  рассматривается найденная вершина  $x$ ; процесс продолжается до тех пор, пока не будет достигнута вершина  $s$ , тогда путь из дуг, соединяющих выделенные вершины – искомый кратчайший путь из  $s$  в  $t$ .

**Пример.** Рассмотрим график, представленный на рис. 1.

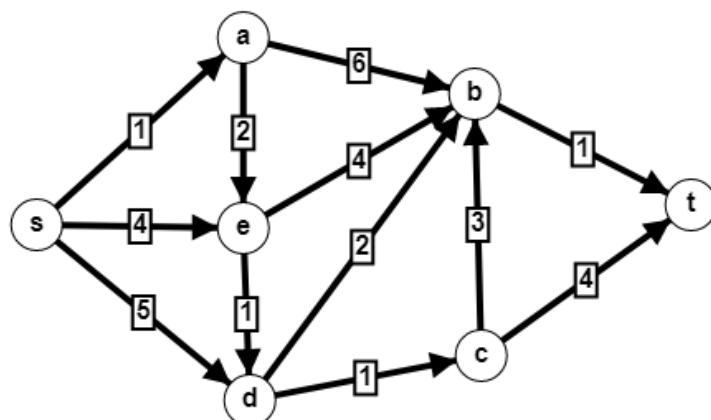


Рисунок 1 – Ориентированный взвешенный граф  $G$

В таблице представлена работа алгоритма Дейкстры.

Таблица 1 – Расчетная таблица для алгоритма Дейкстры

Вершины	Итерации						
	1	2	3	4	5	6	7
$s$	$0^+$						
$a$	$\infty$	$1^+$					
$b$	$\infty$	$\infty$	7	7	6	$6^+$	
$c$	$\infty$	$\infty$	$\infty$	$\infty$	$5^+$		
$d$	$\infty$	5	5	$4^+$			
$e$	$\infty$	4	$3^+$				
$t$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	9	$7^+$
<b>Текущая вершина</b>	$s$	$a$	$e$	$d$	$c$	$b$	$t$

На первой итерации вершине  $s$  приписывается метка  $l(s) = 0^+$ , которая считается постоянной, всем остальным вершинам – метка  $l(x) = \infty$ ,  $x \in \{a, b, c, d, e\}$ , причем эти метки являются временными. На первой итерации вершина  $s$  является текущей. Найдем множество  $\Gamma(s) = \{a, d, e\}$ , этим вершинам приписываются метки

$$l(a) = \min\{\infty, 0 + 1\} = 1,$$

$$l(d) = \min\{\infty, 0 + 5\} = 5,$$

$$l(e) = \min\{\infty, 0 + 4\} = 4.$$

На второй итерации минимальной меткой обладает вершина  $a$ , поэтому ее метка теперь считается постоянной ( $1^+$ ), и она выбирается в качестве текущей. Найдем  $\Gamma(a) = \{b, e\}$ . Обновим метки этих вершин следующим образом:  $l(b) = \min\{\infty, 1 + 6\} = 7$ ,  $l(e) = \min\{4, 1 + 2\} = 3$ .

На третьей итерации минимальную метку имеет вершина  $e$  ( $3^+$ ). Эту метку будем считать постоянной, а вершина  $e$  становится текущей. Найдем множество  $\Gamma(e) = \{b, d\}$ . Заметим, что метку вершины  $b$  обновлять не нужно. У вершины  $d$  метка обновится  $l(d) = \min\{5, 3 + 1\} = 4$ .

На следующей итерации минимальную метку имеет вершина  $d$ , сделаем ее постоянной ( $4^+$ ) и найдем множество  $\Gamma(d) = \{b, c\}$ . Имеющиеся метки вершин нужно обновить следующим образом:  $l(b) = \min\{7, 4 + 2\} = 6$ ,  $l(c) = \min\{\infty, 4 + 1\} = 5$ .

На пятой итерации минимальную метку имеет вершина  $c$ , ее метка объявляется постоянной  $5^+$ . Для текущей вершины  $c$  найдем множество

$\Gamma(c) = \{b, t\}$ . Метку вершины  $b$  изменять не нужно, а метка вершины  $t$  обновиться  $l(t) = \min\{\infty, 5 + 4\} = 9$ .

На шестой итерации минимальной меткой обладает вершина  $b$ , она становится текущей, а ее метка постоянной  $(6^+)$ . Найдем  $\Gamma(b) = \{t\}$ . Метку вершины  $t$  можно обновить  $l(t) = \min\{9, 6 + 1\} = 7$ .

В силу единственности эта метка становится постоянной  $(7^+)$ .

Таким образом, все вершины получили постоянные метки. Чтобы найти сам путь, нужно двигаться от  $t$  к  $s$ , при этом каждый раз нужно выбирать такую вершину  $y$ , чтобы выполнялось соотношение

$$l^+(x) + c(x, y) = l^+(y).$$

Итак, находимся в вершине  $t$ . Из входящих дуг выбираем дугу  $(b, t)$ , так как  $6^+ + 1 = 7^+$ . Теперь выбираем дуги из вершины  $b$ . Эта вершина имеет четыре входящих дуги, но только для дуги  $(d, b)$  выполняется условие  $4^+ + 2 = 6^+$ . Теперь находимся в вершине  $d$ , которая имеет две входящих дуги. Выбираем дугу  $(e, d)$ , так как для нее  $3^+ + 1 = 4^+$ . Из вершины  $e$  выберем дугу  $(a, e)$ , так как  $1^+ + 2 = 3^+$ . Наконец, в вершину  $a$  входит единственная дуга из вершины  $s$ , причем выполняется соотношение  $0^+ + 1 = 1^+$ , а, следовательно, дуга  $(s, a)$  принадлежит критическому пути. Таким образом, найден кратчайший путь  $s \rightarrow a \rightarrow e \rightarrow d \rightarrow b \rightarrow t$  (рис. 2), его длина равна 7.

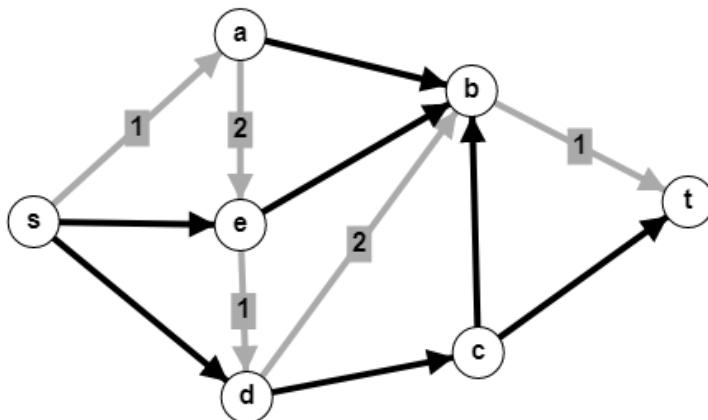


Рисунок 2 – Кратчайший путь  $s \rightarrow a \rightarrow e \rightarrow d \rightarrow b \rightarrow t$

Замечание. Различная интерпретация весов графа приводит к различным задачам, которые можно решить с помощью алгоритма Дейкстры. Пусть вес  $\rho_{ij} \in (0, 1]$  дуги  $(x_i, x_j)$  означает ее надежность, тогда каждому пути  $P$  от  $s$  к  $t$  можно поставить в соответствие его надежность по формуле  $\rho(P) = \prod_{(x_i, x_j) \in P} \rho_{ij}$ . Наиболее надежный путь – это путь, которому соответствует

максимальное значение величины  $\rho(P)$ . Заметим, что  $\ln \rho(P) = \sum_{(x_i, x_j) \in P} \ln \rho_{ij}$ ,

где  $\ln \rho_{ij} \leq 0$ , тогда, умножая левую и правую часть выражения (X) на  $-1$  и рассматривая в качестве весов дуг  $(x_i, x_j)$  величины  $c_{ij} = -\ln \rho_{ij} \geq 0$ , получим, что задача нахождения наиболее надежного пути от  $s$  к  $t$  сводится к задаче нахождения кратчайшего пути из  $s$  в  $t$ . ♦

Если требуется определить кратчайший путь между заданными вершинами орграфа, причем под длиной пути подразумевается количество дуг, которые этот путь составляют, то используется специальный алгоритм, причем его можно применить и к неориентированным графам.

#### *Алгоритм «фронта волны»*

Комментарий. Идея алгоритма заключается в следующем: начиная с начальной вершиной  $s$ , которой приписывается метка  $l(s) = 0$ , на каждой итерации строится и помечается множество вершин, которые достижимы из помеченных вершин на предыдущей итерации. Алгоритм завершает работу, если вершина  $t$  получает метку. Данная метка определяет длину кратчайшего пути. В данном алгоритме для множества  $A$  формируется множество смежных вершин  $\Gamma(A) = \bigcup_{x \in A} \Gamma(x)$ .

Шаг 1. Пусть  $G = (X, \Gamma)$  – данный орграф,  $s, t$  – начальная и конечная вершины искомого кратчайшего пути. Вершине  $s$  приписать метку 0. Положить  $A = \{s\}$ ,  $k = 0$ .

Шаг 2. Найти множество  $\Gamma(A)$ . Положить  $k = k + 1$ , и всем вершинам, которые не помечены, но принадлежат множеству  $\Gamma(A)$ , приписать метки, равные  $k$ .

Шаг 3. Если вершина  $t$  получила метку, то перейти к шагу 4, иначе сформировать новое множество  $A$ , включив в него вершины, которым назначены метки на шаге 2, а затем перейти на шаг 2.

Шаг 4. Пусть вершина  $t$  получила метку  $l(t)$ , тогда  $l(t)$  – длина пути с минимальным числом дуг. Для его нахождения используется следующая процедура: среди вершин, смежных с  $t$ , выделяется вершина с меткой  $l-1$  – пусть это будет  $x$ , затем из  $x$  осуществляется поиск вершины с меткой  $l-2$ , и так далее, пока не будет достигнута вершина  $s$ . Путь, включающий выделенные вершины – искомый.

**Пример.** Рассмотрим граф на рис. 3. Найдем кратчайшие пути из вершины 1 в вершину 12.

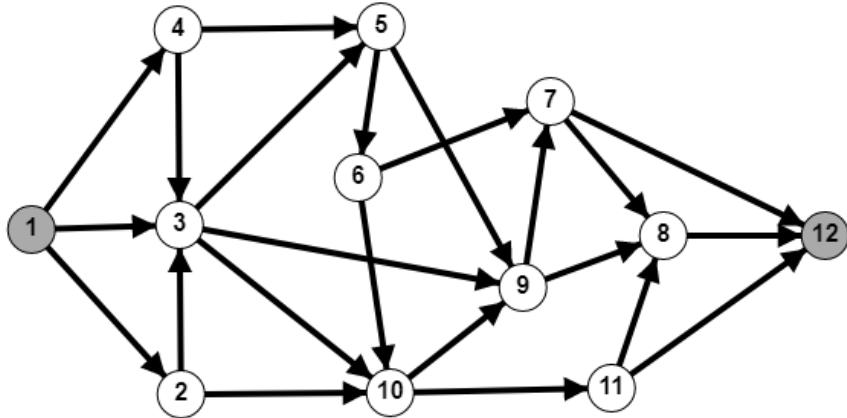


Рисунок 3 – Ориентированный граф  $G$

На первом шаге  $A = \{1\}$  и вершине 1 приписывается метка 0. Найдем  $\Gamma(1) = \{2, 3, 4\}$ , тогда вершинам 2, 3, 4 припишем метку 1 и положим  $A = \{2, 3, 4\}$ . Теперь найдем непомеченные вершины из множества  $\Gamma(A)$ . Это будет множество  $\{5, 9, 10\}$ . Припишем им метку 2 и положим  $A = \{5, 9, 10\}$ . Непомеченные вершины из множества  $\Gamma(A)$  на данном шаге – это вершины 6, 8, 9, 11. Им всем приписывается метка 3. Теперь  $A = \{6, 8, 9, 11\}$ , причем единственной непомеченной вершиной из множества  $\Gamma(A)$  является вершина 12, ей приписывается метка 4. На рис. 4 представлены матки вершин после выполнения алгоритма.

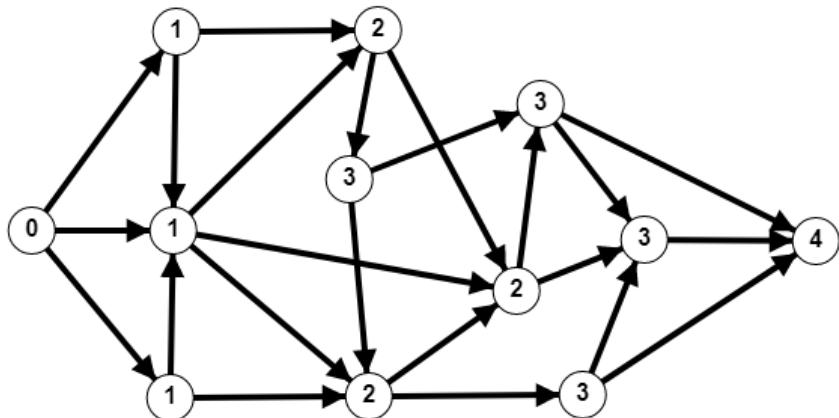
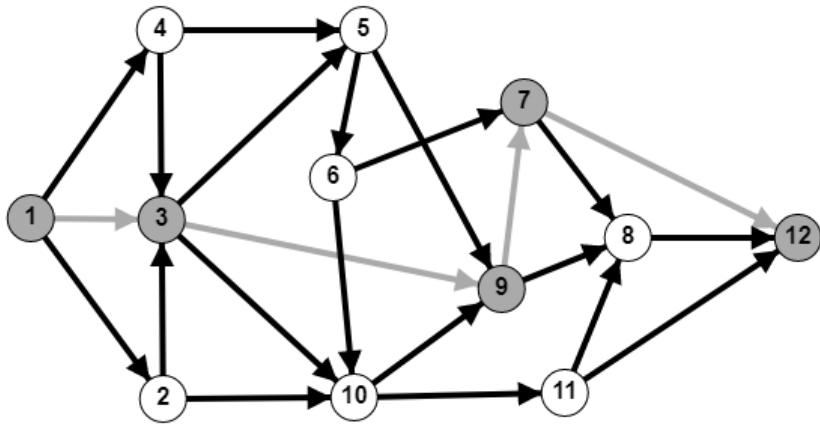


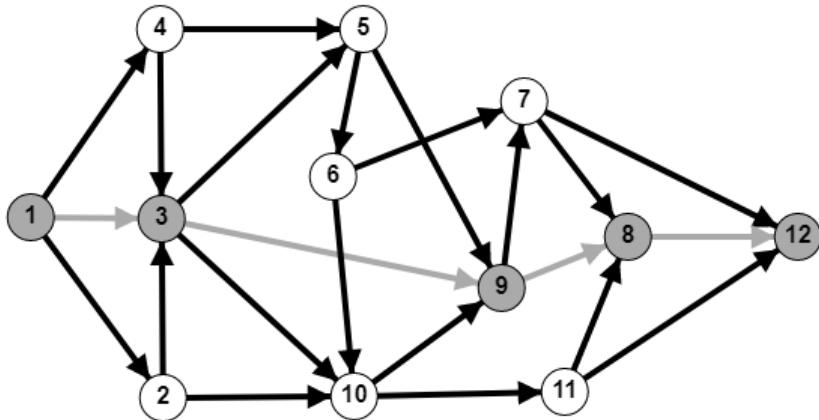
Рисунок 4 – Метки вершин после выполнения шагов алгоритма

На основе найденных меток легко определяются кратчайшие пути (рис. 5):

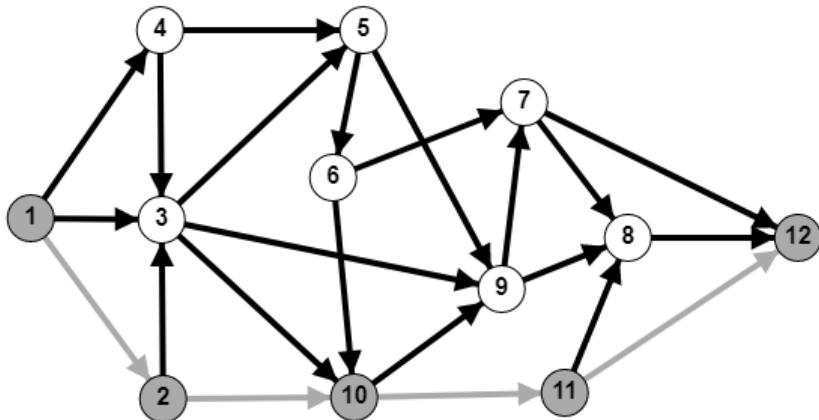
$$\begin{aligned} 1 &\rightarrow 3 \rightarrow 9 \rightarrow 7 \rightarrow 12, \\ 1 &\rightarrow 3 \rightarrow 9 \rightarrow 8 \rightarrow 12, \\ 1 &\rightarrow 2 \rightarrow 10 \rightarrow 11 \rightarrow 12. \end{aligned}$$



a)



b)



c)

Рисунок 5 – Кратчайшие пути графа

**Задача о критическом пути** отличается от задачи о кратчайшем пути тем, что ищется путь максимальной длины. Для неорграфов соответствующая задача заключается в нахождении максимальной цепи. Для взвешенных орграфов задачу можно решить модифицируя алгоритм Дейкстры, однако существуют и специальные алгоритмы.

Если на графике накладываются ограничения, то необходимо использовать специальные алгоритмы. Так в задачах сетевого планирования орграф не дол-

жен содержать контуров. В этом случае для поиска максимального пути, который называется **критическим**, используется следующий алгоритм.

*Алгоритм построения максимального (критического) пути  
в бесконтурном взвешенном орграфе*

Шаг 1. Пусть  $G = (X, U)$  – заданный орграф с матрицей весов  $C = \left( c_{ij} = c(x_i, x_j) \right)_{n \times n}$ , где  $n = |X|$ ,  $c(x_i, x_j) \geq 0$  – вес ребра  $(x_i, x_j)$ ;  $s, t$  – начальная и конечная вершина искомого пути (важно: вершина  $s$  не имеет входящих дуг, а вершина  $t$  – выходящих). Выполнить топологическую сортировку вершин, при этом вершина  $s$  получает правильный номер 1, а вершина  $t$  – правильный номер  $n$ . Вершине  $s$  присвоить метку  $l(s) = 0$ .

Шаг 2. Рассматривая вершины в порядке возрастания правильных номеров вершин, присвоить каждой вершине  $j$  метку  $l(j)$  по правилу

$$l(j) = \max_{i \in \Gamma^{-1}(j)} \{l(i) + c(i, j)\}.$$

Шаг 3. Пусть вершина  $t$  с номером  $n$  имеет метку  $l(n)$ , тогда  $l(n)$  – длина критического пути. Сам путь определяется на основе меток с помощью следующей процедуры: пусть  $p = n$  – текущая вершина с меткой  $l(p)$ , выбрать такую вершину  $k \in \Gamma^{-1}(p)$ , для которой выполняется соотношение  $l(k) + c(k, p) = l(p)$ , затем процесс продолжается, начиная с новой текущей вершины  $p = k$  до тех, пока не будет достигнута вершина  $s$ , которой соответствует номер 1.

Замечание: Задачи сетевого планирования и управления – это задачи, которые заключаются в определении последовательности операций, реализующих некоторый проект, и в распределении ресурсов между ними. В качестве критерии оптимальности рассматриваются минимизация времени выполнения проекта, затрат, риска и т.п. Длина критического пути в сети – это время, необходимое для реализации всего проекта, а работы (дуги), лежащие на критическом пути, обладают тем свойством, что не имеют резервов, поэтому именно на них следует обратить внимание при планировании работ. Изменение длины критической дуги ведет к изменению критического времени.

**Пример.** Граф на рис. 3 является бесконтурным. В результате топологической сортировки вершин получим монотонную (правильную) нумерацию (для каждой дуги номер начала меньше номера конца) (рис. 6). Припишем дугам веса и получим граф на рис. 7.

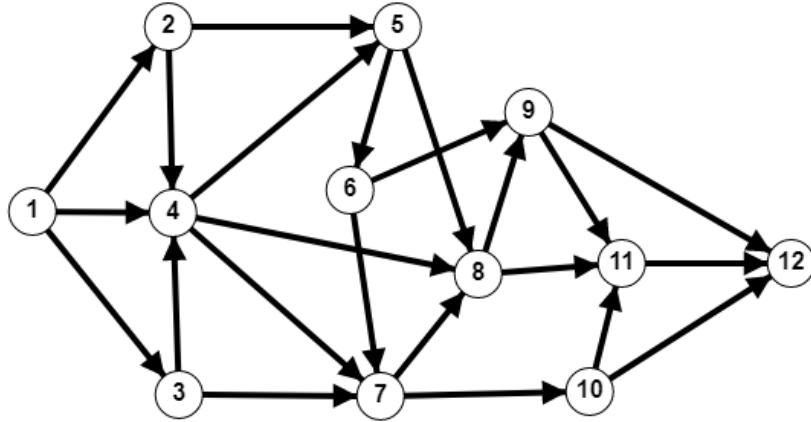


Рисунок 6 – Правильная нумерация вершин

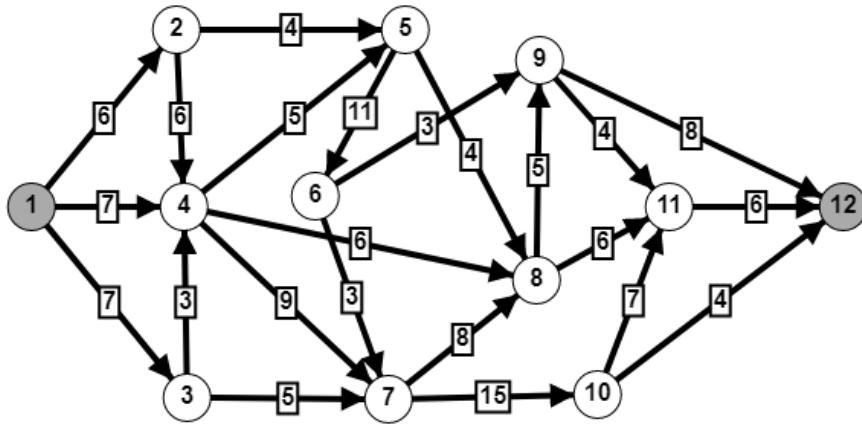


Рисунок 7 – Ориентированный взвешенный граф

Топологическая сортировка позволяет сформировать порядок вершин для расстановки меток. Итак, первой вершине приписывается метка  $l(1)=0$ . Вершины 2 и 3 имеют по одной входящей дуге, поэтому  $l(2)=6$ ,  $l(3)=7$ . Вершина 4 имеет три входящих дуги, поэтому

$$l(4) = \max \{l(1)+7, l(2)+6, l(3)+3\} = \max \{0+7, 6+6, 7+3\} = 12.$$

Аналогично, посчитаем остальные метки:

$$l(5) = \max \{l(2)+4, l(4)+5\} = \max \{6+4, 12+5\} = 17;$$

$$l(6) = l(5)+11 = 17+11 = 28;$$

$$l(7) = \max \{l(3)+5, l(4)+9, l(6)+3\} = \max \{7+5, 12+9, 28+3\} = 31;$$

$$l(8) = \max \{l(4)+6, l(5)+4, l(7)+8\} = \max \{12+6, 17+4, 31+8\} = 39;$$

$$l(9) = \max \{l(6)+3, l(8)+5\} = \max \{28+3, 39+5\} = 44;$$

$$l(10) = l(7)+15 = 31+15 = 46;$$

$$l(11) = \max \{l(8)+6, l(9)+4, l(10)+7\} = \max \{39+6, 44+4, 46+7\} = 53;$$

$$l(12) = \max \{l(9)+8, l(10)+4, l(11)+6\} = \max \{44+8, 46+4, 53+6\} = 59.$$

Таким образом, длина максимального пути равна 59. Сам путь определяется по меткам, так же, как в алгоритме Дейкстры. На графе этот путь выделен более светлым цветом (рис. 7).

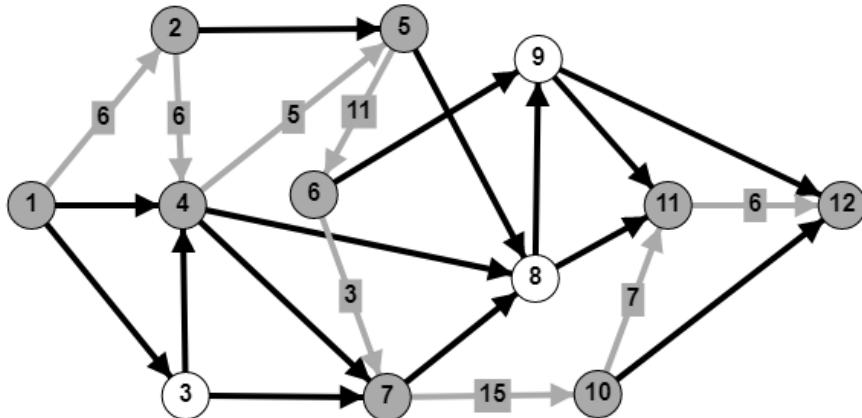


Рисунок 8 – Максимальный путь  $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 10 \rightarrow 11 \rightarrow 12$

Замечание. В рассмотренных выше алгоритмах метка любой вершины есть длина максимального (или минимального (кратчайшего)) пути из вершины  $s$  в данную вершину.

## Алгоритмы раскраски графа

Необходимо раскрасить вершины графа таким образом, чтобы смежные вершины были окрашены в разные цвета. Минимальное число красок, в которые можно раскрасить граф называется *хроматическим числом графа*.

Задача раскраски вершин графа относится к NP-полным задачам.

Различают точные и приближенные алгоритмы раскраски.

Примером точных алгоритмов служит алгоритм Вейссмана.

Алгоритм состоит из двух частей:

1. Построение семейства максимальных внутренне устойчивых множеств (МВУМ) (метод Магу);
2. Выбор минимального числа МВУМ, покрывающих все вершины графа (метод Петрика).

Множество вершин  $X_s$  графа  $G(X, U)$  называется *внутренне устойчивым (независимым)*, если никакие две вершины из этого множества не смежны,  $X_s \subset X$  [ $\Gamma X_s \cap X_s = \emptyset$ ]. Внутренне устойчивое множество называется *максимальным*, если оно не является собственным подмножеством некоторого другого независимого множества.

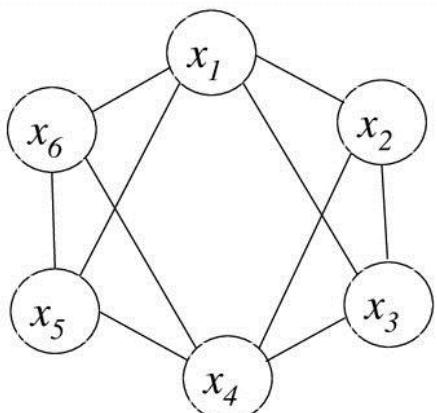
1. В матрице соединений  $R$  для каждой вершины подсчитывается число ненулевых элементов  $r_i$ ;
2. Находится вершина  $x_i$  с  $\max r_i$ , если таких вершин несколько, то выбирается любая;
3. Для выбранной вершины  $x_i$  записывается выражение  $C_i = (x_i \vee x_a x_b \dots x_q)$ , где  $\Gamma x_i = \{x_a, x_b, \dots, x_q\}$ ;
4. Из матрицы  $R$  удаляются строка и столбец, соответствующие вершине  $x_i$ ;
5. Если  $R \neq \emptyset$ , то переход к п. 2, иначе к п. 6;
6. Составляется конъюнкция  $\Pi = \wedge C_i$ . Раскрываются скобки. В полученной дизъюнкции на основе законов булевой алгебры выполняется минимизация.
7. Результат минимизации записывается в виде  $\Pi = \vee K_j$ ;
8. Для каждого  $K_j$  ищутся вершины графа, не вошедшие в него. Получено  $\varphi_j$  и семейство МВУМ  $\Psi = \{\varphi_1, \varphi_2, \dots, \varphi_l\}$ ;
9. Для каждой вершины  $x_i \in X$  определяются подмножества  $\varphi_j$ , в которые входит вершина  $x_i \in \varphi_j$ . Составляется дизъюнкция  $t_i = \vee \varphi_j$ ;

10. Составляется конъюнкция  $\Pi' = \wedge t_i$ . Раскрываются скобки. В полученной дизъюнкции на основе законов булевой алгебры выполняется минимизация;

11. Получена дизъюнкция конъюнктивных термов  $\Pi' = \vee(\wedge\varphi_j)$ . Выбирается конъюнктивный терм  $\wedge\varphi_j$  с минимальным числом сомножителей.

Количество сомножителей в этом терме и есть хроматическое число графа. Число минимальных термов – число вариантов раскраски графа. А каждое  $\varphi_j$  – множество вершин, которые можно окрасить в один цвет.

Заметим, что п.п. 1-8 составляют метод Магу, а п.п. 9-11 – метод Петрика.



	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$r_i$
$x_1$	0	1	1	0	1	1	4
$x_2$		0	1	1	0	0	3
$x_3$			0	1	0	0	3
$x_4$				0	1	1	4
$x_5$					0	1	3
$x_6$						0	3

1. В матрице  $R$  подсчитываем число ненулевых элементов  $r_i$ ;
2.  $\max r_i = r_1 = r_4 = 4$ , выбираем  $x_1$ ;
3.  $\Gamma_{x_1} = \{x_2, x_3, x_5, x_6\}$ , записываем выражение  
 $C_1 = (x_1 \vee x_2 x_3 x_5 x_6);$
4. Из матрицы  $R$  удаляем строку и столбец, соответствующие вершине  $x_1$ ;

$R =$	$\begin{array}{ c ccccc c } \hline & x_2 & x_3 & x_4 & x_5 & x_6 & r_i \\ \hline x_2 & 0 & 1 & 1 & 0 & 0 & 2 \\ x_3 & & 0 & 1 & 0 & 0 & 2 \\ x_4 & & & 0 & 1 & 1 & 4 \\ x_5 & & & & 0 & 1 & 2 \\ x_6 & & & & & 0 & 2 \\ \hline \end{array}$	5. $R \neq \emptyset, \max r_i = r_4 = 4;$ $\Gamma_{x_4} = \{x_2, x_3, x_5, x_6\},$ $C_4 = (x_4 \vee x_2 x_3 x_5 x_6);$
$R =$	$\begin{array}{ c cccc c } \hline & x_2 & x_3 & x_5 & x_6 & r_i \\ \hline x_2 & 0 & 1 & 0 & 0 & 1 \\ x_3 & & 0 & 0 & 0 & 1 \\ x_5 & & & 0 & 1 & 1 \\ x_6 & & & & 0 & 1 \\ \hline \end{array}$	6. Из матрицы $R$ удаляем строку и столбец, соответствующие вершине $x_4$ ;
$R =$	$\begin{array}{ c ccc c } \hline & x_3 & x_5 & x_6 & r_i \\ \hline x_3 & 0 & 0 & 0 & 0 \\ x_5 & & 0 & 1 & 1 \\ x_6 & & & 0 & 1 \\ \hline \end{array}$	7. $R \neq \emptyset, \max r_i = r_2 = r_3 = r_5 = r_6 = 1,$ выбираем $x_2;$ $\Gamma_{x_2} = \{x_3\}, C_2 = (x_2 \vee x_3);$
$R =$	$\begin{array}{ c cc c } \hline & x_3 & x_6 & r_i \\ \hline x_3 & 0 & 0 & 0 \\ x_6 & & 0 & 0 \\ \hline \end{array}$	8. Из матрицы $R$ удаляем строку и столбец, соответствующие вершине $x_2;$
		9. $R \neq \emptyset, \max r_i = r_5 = r_6 = 1,$ выбираем $x_5;$ $\Gamma_{x_5} = \{x_6\}, C_5 = (x_5 \vee x_6);$
		10. Из матрицы $R$ удаляем строку и столбец, соответствующие вершине $x_5;$
		11. $R = \emptyset;$

12. Составляем конъюнкцию  $C_i$  и выполняем минимизацию

$$\begin{aligned} \Pi = \wedge C_i = C_1 C_2 C_4 C_5 &= (x_1 \vee x_2 x_3 x_5 x_6)(x_2 \vee x_3)(x_4 \vee x_2 x_3 x_5 x_6)(x_5 \vee x_6) = \\ &= x_1 x_2 x_4 x_5 \vee x_1 x_2 x_4 x_6 \vee x_1 x_3 x_4 x_5 \vee x_1 x_3 x_4 x_6 \vee x_2 x_3 x_5 x_6 = \vee K_j = \\ &= K_1 \quad \vee \quad K_2 \quad \vee \quad K_3 \quad \vee \quad K_4 \quad \vee \quad K_5; \end{aligned}$$

13. Для каждого  $K_j$  ищем  $\varphi_j$ :

$\varphi_1 = \{x_3, x_6\}, \varphi_2 = \{x_3, x_5\}, \varphi_3 = \{x_2, x_6\}, \varphi_4 = \{x_2, x_5\}, \varphi_5 = \{x_1, x_4\}$ . Получено семейство МВУМ  $\Psi$ ;

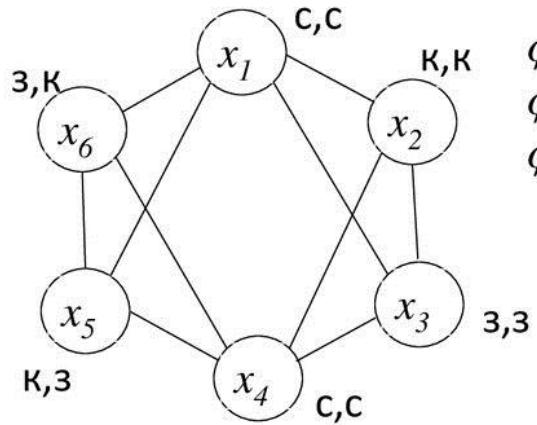
14. Для каждой вершины определим подмножества  $\varphi_j$ , в которые она входит. Строим дизъюнкцию  $t_i = \vee \varphi_j$ ;

$$t_1 = \varphi_5; t_2 = \varphi_3 \vee \varphi_4; t_3 = \varphi_1 \vee \varphi_2; t_4 = \varphi_5; t_5 = \varphi_2 \vee \varphi_4; t_6 = \varphi_1 \vee \varphi_3;$$

15. Составляем конъюнкцию и выполняем минимизацию булевой функции

$$\begin{aligned} \Pi' = \wedge t_i = t_1 t_2 t_3 t_4 t_5 t_6 &= \varphi_5(\varphi_3 \vee \varphi_4)(\varphi_1 \vee \varphi_2) \varphi_5(\varphi_2 \vee \varphi_4)(\varphi_1 \vee \varphi_3) = \\ &= \varphi_1 \varphi_4 \varphi_5 \vee \varphi_2 \varphi_3 \varphi_5 \end{aligned}$$

Хроматическое число графа  $\chi(G) = 3$ . Существует два варианта раскраски графа.

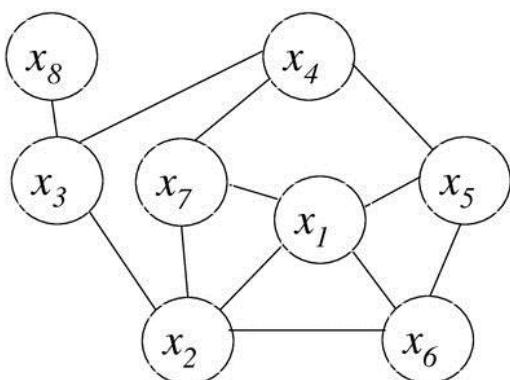


$\varphi_1 = \{x_3, x_6\}$ ,  $\varphi_2 = \{x_3, x_5\}$ ,  $\varphi_3 = \{x_2, x_6\}$ ,  
 $\varphi_4 = \{x_2, x_5\}$ ,  $\varphi_5 = \{x_1, x_4\}$ .  
 $\varphi_1\varphi_4\varphi_5 \vee \varphi_2\varphi_3\varphi_5$

Недостатком точных алгоритмов является низкое быстродействие. Поэтому на практике используют приближенные алгоритмы, примером которых может служить

### Алгоритм, использующий упорядочивание вершин

1. Положить  $j = 1$ ;
2. В матрице  $R$  подсчитываем число ненулевых элементов  $r_i$ ;
3. Упорядочим вершины графа в порядке не возрастания  $r_i$ ;
4. Просматривая последовательность слева направо, красить в цвет  $j$  каждую неокрашенную вершину, не смежную с уже окрашенными в этот цвет;
5. Если остались неокрашенные вершины, то удалить из матрицы  $R$  строки и столбцы, соответствующие окрашенным вершинам. Положить  $j = j + 1$  и перейти к п. 2, иначе, задача решена.



	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$r_i$
$x_1$	0	1	0	0	1	1	1	0	4
$x_2$	0	1	0	0	1	1	0	0	4
$x_3$	0	1	0	0	0	0	1	0	3
$x_4$	0	1	0	1	0	1	0	0	3
$x_5$	0	1	0	0	1	0	0	0	3
$x_6$	0	0	0	0	0	0	0	0	3
$x_7$	0	0	0	0	0	0	0	0	3
$x_8$	0	0	0	0	0	0	0	1	1

1. Положим  $j = 1$ ;
2. Упорядочим вершины графа в порядке не возрастания  $r_i$ .  
 $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$ ;
3. Красим в первый цвет вершины  $x_1$  и  $x_3$ . Вершины  $x_4$  и  $x_8$  смежны вершине  $x_3$ , остальные – смежны вершине  $x_1$ ;
4. Остались неокрашенные вершины, поэтому удалим из матрицы  $R$  строки и столбцы, соответствующие вершинам  $x_1$  и  $x_3$ . Положим  $j = j + 1 = 2$ .

	$x_2$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$r_i$
$x_2$	0	0	0	1	1	0	2
$x_4$		0	1	0	1	0	2
$x_5$			0	1	0	0	2
$x_6$				0	0	0	2
$x_7$					0	0	2
$x_8$						0	0

5. Упорядочим вершины графа в порядке не возрастания  $r_i$ :

$x_2, x_4, x_5, x_6, x_7, x_8$ ;

6. Красим во второй цвет вершины  $x_2, x_4$  и  $x_8$ . Вершины  $x_5$  и  $x_7$ , смежны вершине  $x_4$ , вершина  $x_6$  смежна вершине  $x_2$ ;

	$x_5$	$x_6$	$x_7$	$r_i$
$x_5$	0	1	0	1
$x_6$		0	0	1
$x_7$			0	0

7. Остались неокрашенные вершины, удалим из матрицы  $R$  строки и столбцы, соответствующие вершинам  $x_2, x_4$  и  $x_8$ . Положим  $j = j + 1 = 3$ .

8. Упорядочим вершины графа в  $r_i$ :  $x_5, x_6, x_7$ .

9. Красим в третий цвет вершины  $x_5$  и  $x_7$ . Вершины  $x_6$  и  $x_5$  смежны;

10. Осталась неокрашенная вершина, удалим из матрицы  $R$  строки и столбцы, соответствующие вершинам  $x_5$  и  $x_7$ . Положим  $j = j + 1 = 4$ .

11. В четвертый цвет окрашиваем вершину  $x_6$ .

Все вершины окрашены.

Достоинство алгоритма – быстродействие. Недостаток – не оптимальность.

Для раскраски вершин графа приближенным алгоритмом потребовалось четыре цвета. А хроматическое число графа  $\chi(G) = 3$ . Действительно, если в первый цвет окрасить вершины  $x_1, x_4$  и  $x_8$ , во второй –  $x_2$  и  $x_5$ , то в третий можно окрасить оставшиеся вершины  $x_3, x_6$  и  $x_7$ .

## Лекция

### ЦЕНТРЫ И МЕДИАНЫ ГРАФА

Рассмотрим некоторую транспортную сеть и построим модель в виде графа, в котором вершины соответствуют жилым районам или населенным пунктам, а дуги определяют возможность перемещения из одного населенного пункта в другой, т.е. соответствуют дорогам, их связывающим. Учитывая длину дорог, можно построить матрицу кратчайших расстояний или времен проезда между этими районами, причем эта матрица необязательно симметрическая (например, имеются улицы с односторонним движением и т.д.). Требуется определить место расположения центра обслуживания данного населенного пункта или определенной территории. При решении данной задачи необходимо учитывать специфику обслуживания. Обслуживание может быть срочным, например, при размещении пунктов экстренных служб, когда необходимо затратить минимальное время, чтобы добраться до любого пункта обслуживания. Соответствующая задача называется *минимаксной задачей размещения*. В этом случае критерием оптимальности является минимизация расстояния до самого удаленного населенного пункта или жилого района. Другим критерием оптимальности является минимизация общего расстояния при обслуживании всех населенных пунктом или жилых районов. Такой критерий является наиболее подходящим, например, в задаче о размещении телефонных станций в телефонной сети, где вершины представляют абонентов. Задачи такого типа относятся к *минисуммным задачам размещения*, хотя целевая функция является часто не просто суммой расстояний, а суммой различных функций от расстояний. Задача становится более реальной, если каждой вершине графа приписывается вес, отражающий потребность данного района в соответствующем обслуживании (эти веса, например, могут быть пропорциональны численности населения каждого района).

Рассмотрим ориентированный взвешенный граф  $G = (X, U)$ , в котором  $X$  – множество вершин,  $U$  – множество дуг, которые соединяют некоторые пары вершин из  $X$ . Каждой дуге  $(x_i, x_j)$  ставится в соответствие вес  $c_{ij}$  – некоторое число, которое может быть интерпретировано как расстояние или время, необходимое для преодоления этого расстояния. Однако, в зависимости от задачи вес  $c_{ij}$  может иметь другую интерпретацию. Совокупность весов на дугах графа образует матрицу  $C = (c_{ij})_{n \times n}$ , где  $n = |X|$  – количество вершин графа. Если график неориентированный, то матрица  $C$  является симметричной.

Путь будем обозначать последовательностью вершин, которые в него входят  $\mu = [x = x_{i_1}, x_{i_2}, \dots, x_{i_N} = y]$ , при этом  $x$  – начальная вершина пути, а  $y$  – конечная. Если график является взвешенным, то под длиной пути подразумевается сумма длин дуг, которые этот путь составляют, т.е.

$$L(\mu) = \sum_{k=1}^{N-1} c_{i_k i_{k+1}}.$$

Если граф не является взвешенным, то под длиной пути подразумевается количество дуг, которые этот путь составляют. Таким образом, для пути  $\mu = [x = x_{i_1}, x_{i_2}, \dots, x_{i_N} = y]$  получим  $L(\mu) = N - 1$ .

*Расстоянием*  $d(x, y)$  между вершинами  $x$  и  $y$  называется длина кратчайшего пути из  $x$  в  $y$ . При  $x = y$  полагают  $d(x, y) = 0$ ; если не существует пути из  $x$  в  $y$ , то полагают  $d(x, y) = \infty$ .

Определив расстояние между каждой парой вершин, можно построить матрицу расстояний  $D = (d_{ij})_{n \times n}$ , где  $d_{ij}$  – расстояние между вершинами  $x_i, x_j \in X$  и  $|X| = n$ .

### 1. Центр графа как решение минисуммной задачи размещения

Пусть вершине  $x_i$  приписан вес  $v_i$ , тогда ей в соответствие поставим число

$$S_0(x_i) = \max_{x_j \in X} \{v_j \cdot d(x_i, x_j)\} = \max_j \{v_j \cdot d_{ij}\},$$

называемое *числом внешнего разделения*,

и число

$$S_t(x_i) = \max_{x_j \in X} \{v_j \cdot d(x_j, x_i)\} = \max_j \{v_j \cdot d_{ji}\},$$

называемое *числом внутреннего разделения*.

Если для всех  $x_j$  веса  $v_j = 1$ , то получим числа

$$S_0(x_i) = \max_{x_j \in X} \{d(x_i, x_j)\} = \max_j \{d_{ij}\}$$

и

$$S_t(x_i) = \max_{x_j \in X} \{d(x_j, x_i)\} = \max_j \{d_{ji}\}.$$

В общем случае матрица расстояний не является симметричной относительно главной диагонали, т.е.  $d_{ij} \neq d_{ji}$ , поэтому можно рассматривать две величины

$$S_0(x_i) = \max_j \{d_{ij}\} = l_0(x_i)$$

и

$$S_t(x_i) = \max_j \{d_{ji}\} = l_t(x_i)$$

которые называются соответственно *внешним* и *внутренним эксцентризитетом* вершины  $x_i$ .

Внешний эксцентризитет показывает удаленность или отклонение вершины  $x_i$  от всех остальных вершин графа, а внутренний – удаленность всех остальных вершин графа от данной вершины  $x_i$ .

С учетом приведенных рассуждений получим, что число внешнего разделения показывает удаленность вершины  $x_i$  от всех остальных вершин с учетом их весов. Число внутреннего разделения показывает удаленность вершин графа от вершины  $x_i$  опять же с учетом их весов.

Рассмотрим **пример**. Пусть задан граф  $G$ , матрица расстояний для которого имеет вид (табл. 1)

Таблица 1. Матрица расстояний

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$S_0(x_i)$
$x_1$	0	2	4	7	2	1	7
$x_2$	2	0	4	3	3	3	4
$x_3$	2	1	0	2	1	1	2
$x_4$	3	3	2	0	4	5	5
$x_5$	1	1	1	1	0	1	1
$x_6$	3	2	7	2	1	0	7
$S_t(x_i)$	3	3	7	7	4	5	

В табл. 1 для каждой вершины можно легко найти ее внешний и внутренний эксцентризитет.

Число  $S_0(x_i)$  является наибольшим числом в строке  $x_i$  матрицы  $D'$ , полученной в результате умножения каждого столбца  $x_j$  матрицы расстояний  $D$  на вес  $v_j$ .

Число  $S_t(x_i)$  является наибольшим числом в столбце  $x_i$  матрицы  $D''$ , полученной в результате умножения каждой строки  $x_j$  матрицы расстояний  $D$  на вес  $v_j$ .

Вершина  $x_0^*$ , для которой

$$S_0(x_0^*) = \min_{x_i \in X} \{S_0(x_i)\},$$

называется *внешним центром* графа.

Вершина  $x_t^*$ , для которой

$$S_t(x_t^*) = \min_{x_i \in X} \{S_t(x_i)\},$$

называется *внутренним центром* графа.

Может оказаться, что у графа имеется несколько внешних и внутренних центров, которые образуют множества внешних и внутренних центров соответственно.

Число внешнего разделения вершины  $x_0^*$ , которая является внешним центром, называется *внешним радиусом* графа

$$\rho_0 = S_0(x_0^*).$$

Число внешнего разделения вершины  $x_t^*$ , которая является внутренним центром, называется *внутренним радиусом* графа

$$\rho_t = S_t(x_t^*).$$

В рассмотренном примере внешним центром является вершина  $x_5$ , внутренним центром являются две вершины  $x_1$  или  $x_2$ , при этом внешний радиус равен  $\rho_0 = 1$ , а внутренний радиус равен  $\rho_t = 3$ .

Вершина  $x^*$ , для которой эксцентризитет определяется формулой

$$l(x^*) = \min_{x_i \in X} \{l(x_i)\},$$

называется *центром* графа.

Известны следующие признаки существования центра:

1) если каждая пара вершин ориентированного графа соединена путем, по крайней мере, в одном направлении, то граф имеет центр;

2) каждое дерево имеет центр, состоящий из одной вершины, или из двух смежных вершин (в дереве самой дальней от любой его вершины будет висячая вершина. Если из дерева удалить все висячие вершины вместе с инцидентными ребрами, то эксцентризитет каждой вершины уменьшится, но центральные вершины останутся теми же. Продолжая процесс удаления висячих вершин, придем к ситуации, когда останутся одна или две вершины. В силу того, что центральные вершины на каждой итерации не меняются, то на последней они и будут выявлены).

Если граф  $G$  имеет центр  $x^*$ , то его эксцентризитет  $\rho = l(x^*)$  называется *радиусом* графа.

Если конечный граф  $G$  является полным, то его радиус  $\rho \leq 2$ , а всякая вершина  $x^*$ , для которой

$$|\Gamma(x^*) \setminus \{x^*\}| = \max_{x \in X} |\Gamma(x) \setminus \{x\}|$$

служит центром.

Заметим, что если связный граф является неориентированным, то в нем каждые две вершины соединены цепью, а кратчайшая цепь является единственной, поэтому  $d_{ij} = d_{ji}$  для всех пар индексов  $(i, j)$ . Тогда введенные парные определения будут совпадать, а соответствующие понятия называются числом разделения, центром и радиусом.

Введенные понятия можно обобщить на случай искусственных вершин графа. Под *искусственной вершиной* будем понимать любую точку, которую можно разместить на некоторой дуге графа. Пусть  $(x_i, x_j)$  – дуга графа  $G$  с весом  $c_{ij}$ , и  $y$  – искусственная вершина на дуге  $(x_i, x_j)$ , тогда

$$c_{ij} = l(x_i, y) + l(y, x_j),$$

где  $l(x_i, y)$  и  $l(y, x_j)$  – веса, например, длины соответствующих участков.

Вводя на графе искусственные вершины, можно обобщить введенные выше понятия, при этом подразумевается, что  $y$  – это вершина исходная графа или искусственная вершина. Обобщенные понятия приведены в табл. 2.

Таблица 2. Обобщенные понятия

Число внешнего разделения	$S_0(y) = \max_{x_i \in X} \{v_i \cdot d(y, x_i)\}$
Число внутреннего разделения	$S_t(y) = \max_{y \in G} \{v_i \cdot d(x_i, y)\}$
Абсолютный внешний центр графа $y_0^*$	$S_0(y_0^*) = \min_{y \in G} \{S_0(y)\}$
Абсолютный внутренний центр графа $y_t^*$	$S_t(y_t^*) = \min_{y \in G} \{S_t(y)\}$
Абсолютный внешний радиус	$S_0(y_0^*)$
Абсолютный внутренний радиус	$S_t(y_t^*)$

Формулы для центров графа можно записать иначе:

$$S_0(y_0^*) = \min_{y \in G} \max_{x_i \in X} \{v_i \cdot d(y, x_i)\},$$

$$S_t(y_t^*) = \min_{y \in G} \max_{y \in G} \{v_i \cdot d(x_i, y)\}.$$

Если исходный граф  $G$  является взвешенным, то центры и радиусы можно найти на основе определений с помощью следующей процедуры:

Шаг 1. Для графа  $G$  задать веса вершин и дуг с учетом решаемой задачи.

Шаг 2. С помощью алгоритма Дейкстры построить матрицу кратчайших расстояний.

Шаг 3. Для каждой вершины  $x_i$  определить числа внешнего  $S_0(x_i)$  и внутреннего  $S_t(x_i)$  разделений.

Шаг 4. Найти внешние  $x_0^*$  и внутренние  $x_t^*$  центры графа.

Шаг 5. Найти внешний  $\rho_0$  и внутренний  $\rho_t$  радиусы графа.

## 2. Медиана графа как решение минисуммной задачи размещения

Пусть вершине  $x_i$  приписан вес  $v_i$ , тогда ей в соответствие поставим число

$$\sigma_0(x_i) = \sum_{x_j \in X} v_j \cdot d(x_i, x_j) = \sum_{x_i \in X} v_j \cdot d_{ij},$$

называемое *внешним передаточным числом*,

и число

$$\sigma_t(x_i) = \sum_{x_j \in X} v_j \cdot d(x_j, x_i) = \sum_{x_j \in X} v_j \cdot d_{ji},$$

называемое *внутренним передаточным числом*.

Таким образом, число  $\sigma_0(x_i)$  есть сумма элементов строки  $x_i$  матрицы, полученной после умножения каждого столбца матрицы расстояний  $D$  на вес, соответствующий этому столбцу вершины (столбец  $x_j$  умножается на  $v_j$ ). Число  $\sigma_t(x_i)$  есть сумма элементов столбца матрицы, полученной после умножения каждой строки  $x_i$  на вес  $v_i$ , который ей соответствует.

Вершина  $x_0^{**}$ , для которой

$$\sigma_0(x_0^{**}) = \min_{x_i \in X} \{\sigma_0(x_i)\},$$

называется *внешней медианой* графа, а вершина  $x_t^{**}$ , для которой

$$\sigma_t(x_t^{**}) = \min_{x_i \in X} \{\sigma_t(x_i)\},$$

называется *внутренней медианой* графа.

Если существует такая вершина  $x^{***}$ , что для нее  $\sigma_0(x^{***}) = \sigma_t(x^{***})$ , то она называется *внешне-внутренней медианой*.

Рассмотрим пример. Пусть задан граф  $G$ , матрица расстояний для которого имеет вид (табл. 3)

Таблица 3. Матрица расстояний

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$\sigma_0(x_i)$
$x_1$	0	2	4	7	2	1	16
$x_2$	2	0	4	3	3	3	15
$x_3$	2	1	0	2	1	1	7
$x_4$	3	3	2	0	4	5	17
$x_5$	1	1	1	1	0	1	5
$x_6$	3	2	7	2	1	0	15
$\sigma_t(x_i)$	11	9	16	15	11	11	

В табл. 4 для каждой вершины найдены внешние и внутренние передаточные числа. Выбирая минимальные значения в приписанных строке и столбце. Получаем, что внешней медианой является вершина  $x_3$ , а внутренней медианой – вершина  $x_2$ .

### 3. Метод Хакими для нахождения абсолютного центра

Рассмотрим пример. Пусть задан граф, изображенный на рис.1.

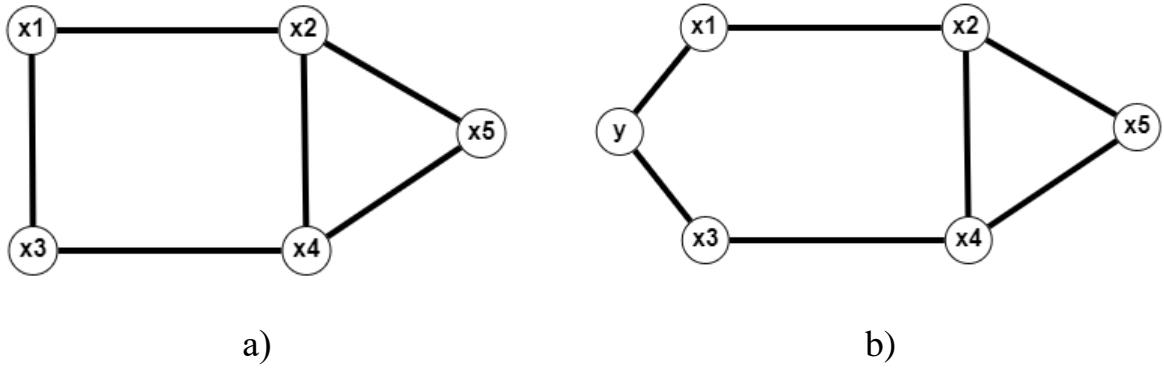


Рисунок 1 – Исходный граф  $G$  (а) и тот же граф (б)  
с добавленной искусственной вершиной.

Для графа  $G$  найдем матрицу расстояний.

$D$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
$x_1$	0	1	1	2	2
$x_2$	1	0	2	1	1
$x_3$	1	2	0	1	2
$x_4$	2	1	1	0	1
$x_5$	2	1	2	1	0

Заметим, что центром является каждая вершина, и радиус равен 2. Введем искусственную вершину  $y$  на ребре  $(x_1, x_3)$ , для которой

$$l(x_1, y) = l(y, x_3) = 0.5.$$

В этом случае к матрице  $D$  добавится строка следующего вида

$D$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$y$
$y$	0.5	1.5	0.5	1.5	2.5	0

В этом случае наибольшее число разделения будет равно 2.5, а, следовательно, вершина  $y$  более центральна, чем другие вершины графа. Вершина  $y$  является абсолютным центром графа. Аналогично, всякая вершина, расположенная подобным образом на любом ребре графа, будет абсолютным центром. Таким образом, абсолютные центры могут располагаться как в вершинах графа, так в искусственных вершинах на ребрах.

Теперь рассмотрим один из самых известных алгоритмов для нахождения абсолютных центров неориентированного графа – метод Хакими, идея которого заключается в следующем:

- для каждого ребра графа  $u_k$  найти точку  $y_k^*$  с наименьшим числом разделения;
- из всех точек  $y_k^*$  выбрать точку  $y^*$  с наименьшим числом разделения в качестве абсолютного центра графа.

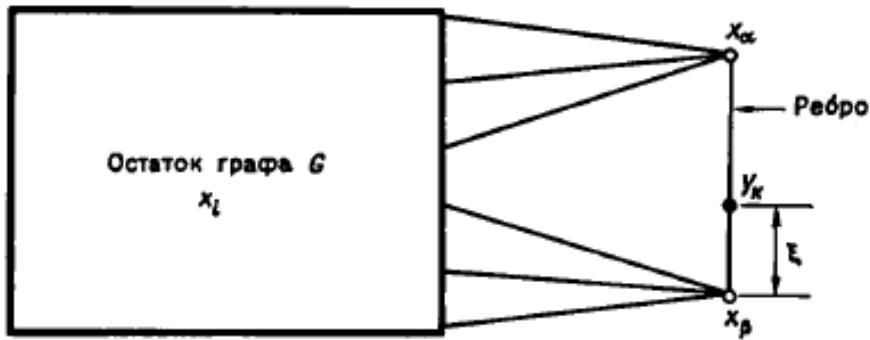


Рисунок 2 – Положение искусственной вершины.

Выведем формулу для числа разделения искусственной вершины  $y_k$ , расположенной на ребре  $u_k = (x_\alpha, x_\beta)$ .

Для вершины  $y_k$  число разделения определяется по формуле

$$\begin{aligned} S(y_k) &= \max_{x_i \in X} \{v_i \cdot d(y_k, x_i)\} = \\ &= \max_{x_i \in X} \left\{ v_i \cdot \min \left\{ l(y_k, x_\alpha) + d(x_\alpha, x_i), l(y_k, x_\beta) + d(x_\beta, x_i) \right\} \right\}, \end{aligned}$$

где  $d(x_\alpha, x_i)$ ,  $d(x_\beta, x_i)$  – это кратчайшие расстояния от концевых вершин ребра  $u_k = (x_\alpha, x_\beta)$  до некоторой вершины  $x_i$ ;  $l(y_k, x_\alpha)$ ,  $l(y_k, x_\beta)$  – длины соответствующих частей ребра  $u_k$ .

Обозначим, например,  $l(y_k, x_\beta) = \xi$ . Так как  $c_{\alpha\beta} = l(y_k, x_\alpha) + l(y_k, x_\beta)$ , то  $l(y_k, x_\alpha) = c_{\alpha\beta} - \xi$ , и формула для числа разделения примет вид

$$S(y_k) = \max_{x_i \in X} \min \left\{ v_i \cdot (d(x_\alpha, x_i) + c_{\alpha\beta} - \xi), v_i \cdot (d(x_\beta, x_i) + \xi) \right\}.$$

Введем следующие обозначения:

$$\begin{aligned} T_i &= v_i \cdot (d(x_\beta, x_i) + \xi), \\ T'_i &= v_i \cdot (d(x_\alpha, x_i) + c_{\alpha\beta} - \xi), \end{aligned}$$

тогда

$$S(y_k) = \max_{x_i \in X} \min \{T'_i, T_i\}.$$

Рассмотрим метод Хакими более подробно.

Шаг 1. Пусть  $G$  – заданный взвешенный неориентированный граф. Перебирая в произвольном порядке ребра, зафиксируем некоторое ребро  $u_k = (x_\alpha, x_\beta)$  и выберем на нем точку  $y_k$  в качестве искусственной вершины. Положим  $l(y_k, x_\beta) = \xi$ ,  $l(y_k, x_\alpha) = c_{\alpha\beta} - \xi$ .

Шаг 2. Для фиксированной вершины  $x_i$  при каждом значении  $0 \leq \xi \leq c_{\alpha\beta}$  найти наименьшее значение выражений  $T_i$  и  $T'_i$ , которые рассматриваются как

линейные функции относительно  $\zeta$ . Для этого строятся соответствующие прямые и находится их нижняя «огибающая», которая представляет собой ломаную линию, состоящую из двух нижних лучей (от точки пересечения) рассматриваемых прямых линий.

Шаг 3. Повторить шаг 2 для всех вершин  $x_i \in X$ , строя для каждого случая нижнюю «огибающую».

Шаг 4. Построить верхнюю «огибающую» для полученных нижних «огибающих», которая дает числа разделения  $S(y_k)$  для всех значений параметра  $\zeta$ , а, следовательно, для всех точек  $y_k$ , расположенных на ребре  $u_k$ .

Шаг 5. На верхней «огибающей» найти точку минимума, которая соответствует абсолютному центру  $y_k^*$  и лежит на ребре  $u_k$ .

Шаг 6. Найти абсолютный центр графа, как точку на одном из ребер  $u_k$ , на котором достигается наименьшее значение минимума для абсолютных центров  $y_k^*$ .

Известен модифицированный метод Хакими для определения границ значения абсолютного центра. Заметим, что любому локальному центру, расположенному на ребре  $(x_i, x_j)$ , соответствует его абсолютный локальный радиус (обозначим его через  $r_{ij}$ ), который не меньше, чем величина

$$\rho_{ij} = \max \left\{ v_s \cdot \min \left\{ d(x_i, x_s), d(x_s, x_j) \right\} \right\}.$$

Можно сделать заключение, что, если абсолютный центр лежит на ребре  $(x_i, x_j)$ , то  $\rho_{ij}$  есть нижняя оценка абсолютного радиуса графа. Тогда величину

$$P = \max_{(x_i, x_j)} \{\rho_{ij}\}$$

можно рассматривать как нижнюю оценку абсолютного радиуса графа.

Предположим, что абсолютный центр расположен в середине ребра  $(x_i, x_j)$ , абсолютный радиус равен  $\rho_{ij} + \frac{1}{2} \cdot v_s \cdot c_{ij}$ , где  $v_s$  – вес той вершины  $x$ , в которой достигается наибольшее значение величины  $\rho_{ij}$ , следовательно,

$$H = \min \left\{ \rho_{ij} + \frac{1}{2} \cdot v_s \cdot c_{ij} \right\}$$

является верхней оценкой абсолютного радиуса графа. Таким образом, можно сделать следующий вывод: любое ребро  $(x_i, x_j)$ , для которого  $H < \rho_{ij}$  может не рассматриваться при поиске абсолютного центра. Такой подход сокращает количество итераций, и оптимизирует работу алгоритма.

**Пример.** Найдем абсолютный центр графа, представленного на рис. 3.

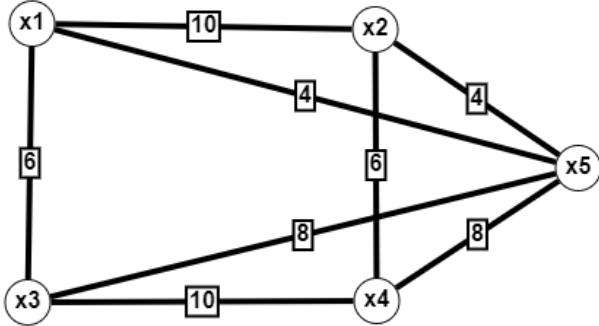


Рисунок 3 – Заданный граф

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$			$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
$x_1$	0	10	6	0	4			$x_1$	0	10	6	12	4
$x_2$	10	0	0	6	4			$x_2$	10	0	12	6	4
$x_3$	6	0	0	10	8			$x_3$	6	12	0	10	8
$x_4$	0	6	10	0	8			$x_4$	12	6	10	0	8
$x_5$	4	4	8	8	0			$x_5$	4	4	8	8	0

Матрица весов графа $C$	Матрица $D$ кратчайших расстояний
-------------------------	--------------------------------------

Найдем абсолютные центры заданного графа методом Хакими. Для каждого ребра графа  $G = (X, U)$  найдем функции

$$T_i = v_i \cdot (\xi + d(x_\beta, x_i)), \quad T'_i = v_i \cdot (c_{\alpha\beta} + d(x_\alpha, x_i) - \xi).$$

Так как граф симметричный, некоторые ребра учитываться не будут. Рассматривая в произвольной последовательности ребра графа  $G$ , построим функции  $T_i$  и  $T'_i$ . Полученные результаты представлены на следующих рисунках.

$$\begin{aligned}
 T_1 &= \xi + 10, T'_1 = 10 - \xi \\
 T_2 &= \xi, T'_2 = 20 - \xi \\
 T_3 &= \xi + 12, T'_3 = 16 - \xi \\
 T_4 &= \xi + 6, T'_4 = 22 - \xi \\
 T_5 &= \xi + 4, T'_5 = 14 - \xi
 \end{aligned}$$

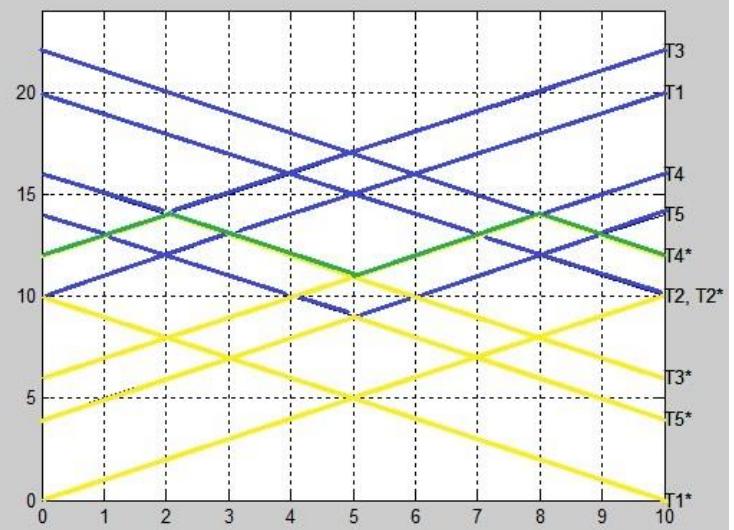


Рисунок 4 – Ребро  $(x_1, x_2)$  с весом  $c_{12} = 10$ . Желтые линии – нижняя огибающая пересечения  $T_i$  и  $T'_i$ , зеленая линия – верхняя огибающая всех желтых линий.

$$\begin{aligned}
 T_1 &= \xi + 6, T'_1 = 6 - \xi \\
 T_2 &= \xi + 12, T'_2 = 16 - \xi \\
 T_3 &= \xi, T'_3 = 12 - \xi \\
 T_4 &= \xi + 10, T'_4 = 18 - \xi \\
 T_5 &= \xi + 8, T'_5 = 10 - \xi
 \end{aligned}$$

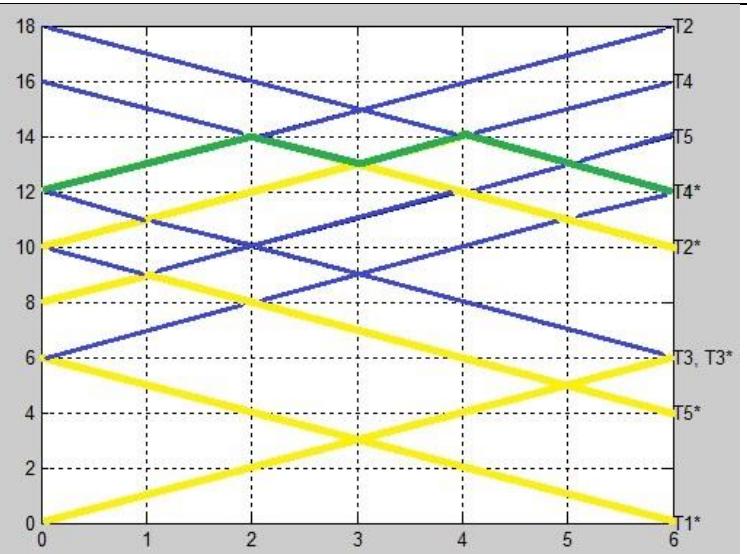


Рисунок 5 – Ребро  $(x_1, x_3)$  с весом  $c_{13} = 6$ . Желтые линии – нижняя огибающая пересечения  $T_i$  и  $T'_i$ , зеленая линия – верхняя огибающая всех желтых линий.

$$\begin{aligned}
 T_1 &= \xi + 4, T'_1 = 4 - \xi \\
 T_2 &= \xi + 4, T'_2 = 14 - \xi \\
 T_3 &= \xi + 8, T'_3 = 10 - \xi \\
 T_4 &= \xi + 8, T'_4 = 16 - \xi \\
 T_5 &= \xi, T'_5 = 8 - \xi
 \end{aligned}$$

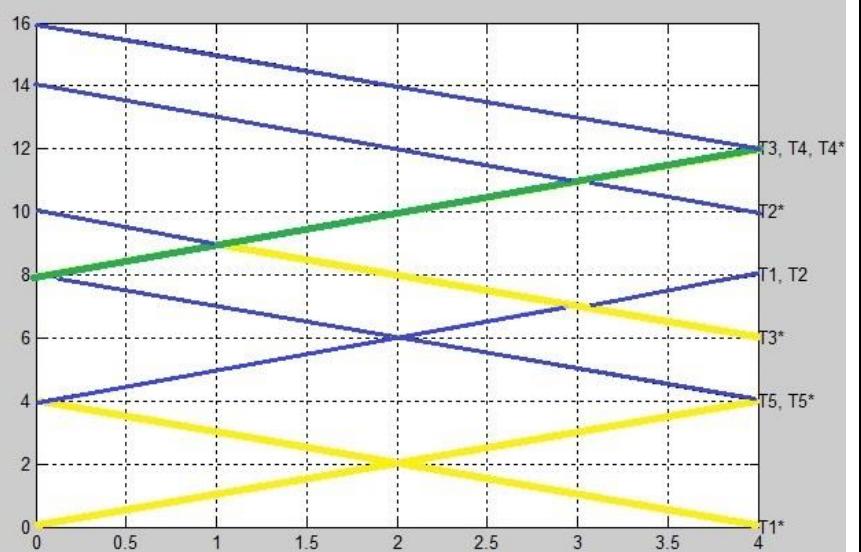


Рисунок 6 – Ребро  $(x_1, x_5)$  с весом  $c_{15} = 4$ . Желтые линии – нижняя огибающая пересечения  $T_i$  и  $T'_i$ , зеленая линия – верхняя огибающая всех желтых линий.

$$\begin{aligned}
 T_1 &= \xi + 6, T'_1 = 14 - \xi \\
 T_2 &= \xi + 4, T'_2 = 16 - \xi \\
 T_3 &= \xi + 8, T'_3 = 4 - \xi \\
 T_4 &= \xi + 8, T'_4 = 14 - \xi \\
 T_5 &= \xi, T'_5 = 12 - \xi
 \end{aligned}$$

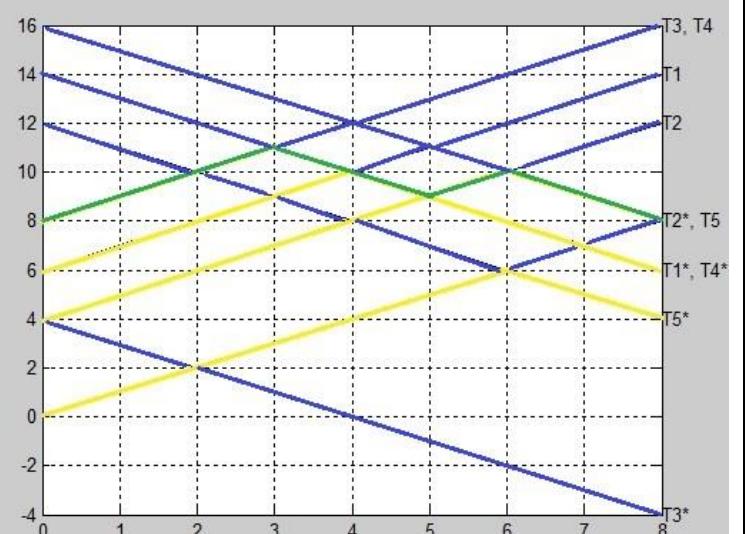


Рисунок 7 – Ребро  $(x_3, x_5)$  с весом  $c_{35} = 8$ . Желтые линии – нижняя огибающая пересечения  $T_i$  и  $T'_i$ , зеленая линия – верхняя огибающая всех желтых линий.

$$\begin{aligned}
 T_1 &= \xi + 4, T'_1 = 14 - \xi \\
 T_2 &= \xi + 4, T'_2 = 4 - \xi \\
 T_3 &= \xi + 8, T'_3 = 16 - \xi \\
 T_4 &= \xi + 8, T'_4 = 10 - \xi \\
 T_5 &= \xi, T'_5 = 8 - \xi
 \end{aligned}$$

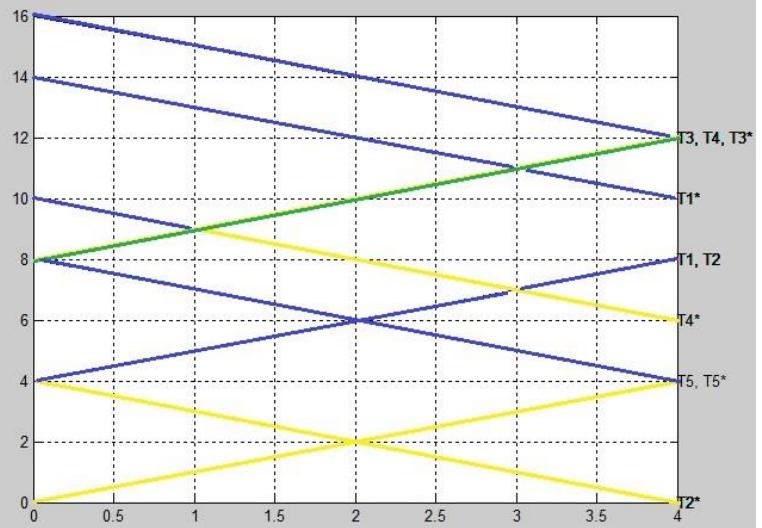


Рисунок 8 – Ребро  $(x_2, x_5)$  с весом  $c_{25} = 4$ . Желтые линии – нижняя огибающая пересечения  $T_i$  и  $T'_i$ , зеленая линия – верхняя огибающая всех желтых линий.

$$\begin{aligned}
 T_1 &= \xi + 12, T'_1 = 16 - \xi \\
 T_2 &= \xi + 6, T'_2 = 22 - \xi \\
 T_3 &= \xi + 10, T'_3 = 10 - \xi \\
 T_4 &= \xi, T'_4 = 20 - \xi \\
 T_5 &= \xi + 8, T'_5 = 18 - \xi
 \end{aligned}$$

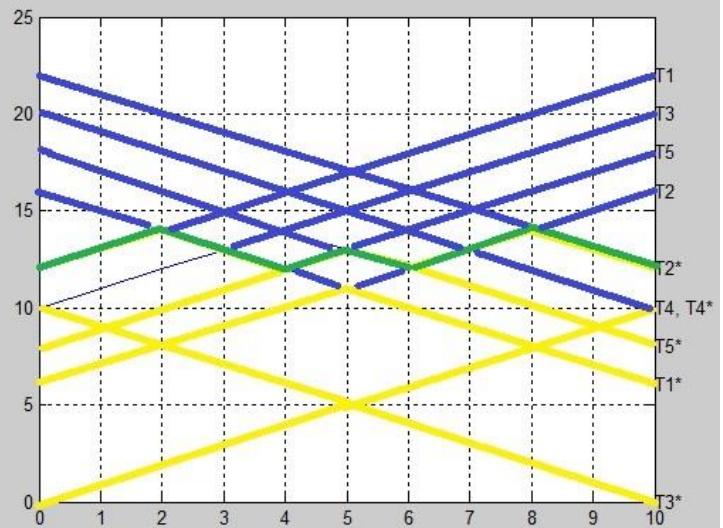


Рисунок 9 – Ребро  $(x_3, x_4)$  с весом  $c_{34} = 10$ . Желтые линии – нижняя огибающая пересечения  $T_i$  и  $T'_i$ , зеленая линия – верхняя огибающая всех желтых линий.

Теперь найдем границы значений абсолютного центра в предположении, что веса вершин равны единице.

$$\begin{aligned}
 p_{12} &= \max \{ \min \{6, 12\}, \min \{12, 6\}, \min \{4, 4\} \} = \max \{6, 6, 4\} = 6, \\
 p_{35} &= \max \{ \min \{4, 6\}, \min \{10, 8\}, \min \{12, 4\} \} = \max \{4, 4, 8\} = 8, \\
 p_{13} &= \max \{ \min \{10, 12\}, \min \{12, 10\}, \min \{4, 8\} \} = \max \{10, 10, 4\} = 10, \\
 p_{15} &= \max \{ \min \{10, 4\}, \min \{6, 8\}, \min \{12, 8\} \} = \max \{4, 6, 8\} = 8, \\
 p_{34} &= \max \{ \min \{6, 12\}, \min \{12, 6\}, \min \{8, 8\} \} = \max \{6, 6, 8\} = 8, \\
 p_{24} &= \max \{ \min \{10, 12\}, \min \{12, 10\}, \min \{4, 8\} \} = \max \{10, 10, 4\} = 10, \\
 p_{45} &= \max \{ \min \{4, 6\}, \min \{10, 8\}, \min \{12, 4\} \} = \max \{4, 4, 8\} = 8, \\
 p_{25} &= \max \{ \min \{10, 4\}, \min \{6, 8\}, \min \{12, 8\} \} = \max \{4, 6, 8\} = 8.
 \end{aligned}$$

Найдем

$$H_{\min} = \min \{p_{ij}\} = 8,$$

$$H_{\max} = \min \left\{ p_{ij} + \frac{1}{2} c_{ij} v_i \right\} = \min \{6 + 5, 8 + 4, 10 + 3, 8 + 2, 10 + 3, 8 + 4, 8 + 2\} = 10.$$

Таким образом, значение абсолютного центра исходного графа удовлетворяет неравенству

$$8 \leq y' \leq 10.$$

Учитывая полученное значение, из рассмотрения удаляются те ребра, для которых значения центра не удовлетворяют неравенству. Анализируя результаты, получим, что значение абсолютного центра равно 8, и центром является одна из вершин, принадлежащая множеству  $\{x_1, x_2\}$ .

## Лекция

### УПРАВЛЕНИЕ ПРОЕКТАМИ: задача календарного планирования

На практике часто встречаются задачи планирования разнообразных по своему содержанию работ, например, строительство промышленного предприятия, создание сети вычислительных устройств, разработка сложной научно-исследовательской темы и другие проекты, включающие множество работ, которые должны выполняться в определенной последовательности. При разработке планов реализации таких проектов возникают следующие вопросы:

- в какие моменты начинать и заканчивать каждую из работ?
- как распределить между ними ресурсы?
- как изменятся параметры плана выполнения работ в случае непредвиденных обстоятельств?

Возможно, что при небольшом количестве работ ответы можно получить из практических соображений, однако, в общем случае возникает необходимость в привлечении специальных методов. Одним из таких методов является метод **сетевого планирования**. Он базируется на представлении совокупности работ, выполняемых в определенной последовательности, в виде сети – взвешенного бесконтурного графа, а соответствующие алгоритмы обработки сети позволяют составить календарный план проекта и провести его анализ.

#### Временные параметры сетевого графика

Временные параметры сетевого графика позволяют составить календарный план реализации проекта и ответить на основной вопрос: сколько времени необходимо для реализации проекта.

Введем основные определения.

Пусть длина дуги соответствует продолжительности выполнения работы, тогда *временем наступления события* будем считать время окончания все работ, предшествующих данному событию, при этом предполагается, что работа над проектом начинается в момент времени, равный нулю.

Каждому событию  $i$  можно поставить в соответствие следующие параметры:

$t_e(i)$  – *раннее время наступления  $i$ -го события* – время, раньше которого событие  $i$  наступить не может, иначе не будут завершены работы, которые ему предшествуют;

$t_l(i)$  – *позднее время наступления  $i$ -го события* – время, позже которого событие  $i$  наступить не должно, иначе увеличится время, необходимое для реализации проекта.

Заметим, что  $t_e(i) \leq t_l(i)$ , при этом величина  $R(i) = t_l(i) - t_e(i)$  называется *резервом времени  $i$ -го события*.

Зафиксируем событие  $i$ . Очевидно, что для его наступления необходимо, чтобы все работы, которые ему предшествуют, были завершены. Тогда раннее время  $t_e(i)$  можно интерпретировать как длину максимального пути из исходного события в данное событие  $i$ .

Длина максимального пути из исходного события в завершающее – это время, необходимое для реализации всего проекта, так как время завершения всего комплекса работ не может быть меньше, чем суммарная продолжительность всех работ вдоль самого длинного пути. Это время называется *критическим* и обозначается  $T_{max}$ . Сетевой график может иметь несколько критических путей. Все они имеют одинаковую длину  $T_{max}$ .

Позднее время  $t_l(i)$  наступления  $i$ -го события можно интерпретировать как разность между длиной критического пути и длиной критического пути из  $i$ -го события в завершающее событие сети.

Работы и события, принадлежащие критическому пути, также называются *критическими*. Особенность критических работ заключается в том, что они не имеют резервов времени, поэтому невыполнение срока окончания любой из критических работ приводит к увеличению критического времени  $T_{max}$ . В связи с этим именно критические работы требуют бесперебойного обеспечения ресурсами и контроля за выполнением. Все остальные работы располагают определенными резервами времени.

Имеет место следующее утверждение: для того, чтобы событие  $i$  принадлежало критическому пути необходимо и достаточно, чтобы раннее и позднее времена наступления этого события совпадали, то есть  $t_e(i) = t_l(i)$ .

Для определения, является ли событие  $i$  критическим, можно использовать его резерв времени  $R(i)$ , который показывает, насколько можно задержать наступление этого события, не вызывая при этом увеличения

критического времени. Для критических событий данный резерв равен нулю, для некритических – положителен.

Пусть  $i$  и  $j$  – события, соответствующие началу и окончанию некоторой работы, тогда будем обозначать ее  $(i, j)$ . В сетевом графике данной работе соответствует дуга с тем же обозначением  $(i, j)$  и весом  $t_{ij}$ , равным продолжительности работы. К временным параметрам сетевого графика также относятся резервы времени работы.

*Полный резерв времени работы  $(i, j)$*

$$R_{full}(i, j) = t_l(j) - t_e(i) - t_{ij}$$

есть разность между критическим временем выполнения проекта  $T_{max}$  и максимальной длиной пути, проходящего через эту работу, поэтому это максимальное время, которым можно располагать для увеличения продолжительности работы  $(i, j)$  без увеличения  $T_{max}$ .

*Свободный резерв времени работы  $(i, j)$*

$$R_{free}(i, j) = t_e(j) - t_e(i) - t_{ij}$$

– это максимально допустимое время увеличения продолжительности работы  $(i, j)$ , при котором все работы  $(j, k)$ , предшествующие событию  $j$ , начинаются в ранее время  $t_e(k)$ .

*Независимый резерв времени работы  $(i, j)$*

$$R_{indep}(i, j) = \max\{0, t_e(j) - t_l(i) - t_{ij}\}$$

есть максимально допустимое количество времени для увеличения продолжительности работы  $(i, j)$  при условии, что все работы  $(k, i)$ , входящие в  $i$ -е событие, заканчиваются в позднее время  $t_l(i)$ , а все работы  $(j, m)$ , выходящие из события  $j$ , начинаются в раннее время  $t_e(j)$ .

Отрицательное значение независимого резерва означает, что любая задержка в выполнении работы приведет к дополнительным ограничениям на выполнение других работ.

Определим взаимосвязи между введенными резервами.

Заметим, что

$$\begin{aligned} R_{full}(i, j) - R_{free}(i, j) &= t_l(j) - t_e(i) - t_{ij} - t_e(j) + t_e(i) + t_{ij} = \\ &= t_l(j) - t_e(j) = R(j). \end{aligned}$$

Увеличение продолжительности работы  $(i, j)$  на часть ее свободного резерва  $R_{free}(i, j)$  может уменьшить полный резерв  $R_{full}(k, i)$  у работ  $(k, i)$ , предшествующих  $i$ -му событию, и не влияет на резервы работ, выходящих из  $j$ -го события.

Пусть для работы  $(i, j)$  имеем  $t_e(j) - t_l(i) - t_{ij} > 0$ , тогда

$$\begin{aligned} R_{full}(i, j) - R_{indep}(i, j) &= t_l(j) - t_e(i) - t_{ij} - t_e(j) + t_l(i) + t_{ij} = \\ &= (t_l(i) - t_e(i)) + (t_l(j) - t_e(j)) = R(i) + R(j). \end{aligned}$$

Если  $t_e(j) - t_l(i) - t_{ij} < 0$ , то  $R_{indep}(i, j) = 0$  и

$$R_{full}(i, j) - R_{indep}(i, j) = R_{full}(i, j).$$

Тогда, если работа  $(i, j)$  является критической,

$$R_{full}(i, j) = R_{free}(i, j) = R_{indep}(i, j) = 0,$$

для некритических работ

$$R_{full}(i, j) > R_{free}(i, j),$$

$$R_{full}(i, j) > R_{indep}(i, j).$$

Найдем  $R_{free}(i, j) - R_{indep}(i, j)$ .

Если для работы  $(i, j)$   $t_e(j) - t_l(i) - t_{ij} > 0$ , то

$$\begin{aligned} R_{free}(i, j) - R_{indep}(i, j) &= t_e(j) - t_e(i) - t_{ij} - t_e(j) + t_l(i) + t_{ij} = \\ &= t_l(i) - t_e(i) = R(i). \end{aligned}$$

Если для работы  $(i, j)$   $t_e(j) - t_l(i) - t_{ij} < 0$ , то  $R_{indep}(i, j) = 0$  и

$$R_{free}(i, j) - R_{indep}(i, j) = R_{free}(i, j).$$

Увеличение продолжительности работы на величину ее независимого резерва не изменяет резервы других работ. Для каждой работы, выходящей из события, лежащего на критическом пути, независимый резерв времени совпадает со свободным.

Таким образом, имеем следующие соотношения между резервами:

$$R_{full}(i, j) = \begin{cases} R_{free}(i, j) + R(j), \\ R_{indep}(i, j) + R(i) + R(j). \end{cases}$$

Имеет место следующее утверждение: для того, чтобы работа  $(i, j)$  была критической, необходимо и достаточно, чтобы  $R_{full}(i, j) = 0$ .

У критических работ полный резерв равен 0, поэтому полный резерв является как бы мерой «критичности» работы: чем меньше полный резерв, тем ближе к критическому пути максимальный по длине путь, проходящий через данную работу. Некритические работы обладают положительным полным резервом. Увеличение продолжительности некритической работы за счет использование всего полного резерва обязательно влечет появление нового критического пути, в состав которого войдет эта работа.

Анализ сетевого графика предполагает определение критических работ и резервов времени некритических работ, разумная задержка которых позволяет улучшить ситуацию с критическими работами. В табл. 1 приведены рекомендации для использования резервов времени некритических работ.

Таблица 1. Рекомендации по использованию резервов времени работ

Резерв	Рекомендация
Какое максимальное количество времени можно выделить для выполнения работы $(i, j)$ без увеличения времени выполнения всего проекта? $R_{full}(i, j)$	При увеличении продолжительности работы $(i, j)$ на величину полного резерва ее начальное событие наступит в самый ранний момент, а конечное событие – в самый поздний.
Какое максимальное количество времени можно выделить для выполнения работы $(i, j)$ без введения дополнительных временных ограничений на последующие работы? $R_{free}(i, j)$	Резерв используется для предотвращения случайностей, – если планировать выполнение работ по ранним срокам их начала и окончания, то всегда будет возможность при необходимости перейти на поздние сроки начала и окончания работ.
Какое максимальное количество времени можно выделить для выполнения работы $(i, j)$ без введения дополнительных временных ограничений на все работы проекта? $R_{indep}(i, j)$	Резерв используется тогда, когда окончание хотя бы одной из предыдущих работ произошло в поздний срок, а последующие работы хотят выполнить в ранние сроки.

### Топологическая сортировка

Под *правильной* (или *монотонной*) *нумерацией* вершин бесконтурного графа подразумевается такая нумерация, согласно которой для каждой дуги  $(i, j)$  номер начала  $i$  меньше номера конца  $j$ . В сетевом графике дуге соответствует работа, поэтому вполне естественно, что номер события, которое соответствует началу работу, будет меньше номера события, которое

соответствует ее окончанию. В сетевом графике правильная нумерация событий позволяет ввести бинарные отношения предшествования на множестве работ. Известно, что топологическая сортировка всегда реализуема в бесконтурном графе. Поскольку сетевой график является бесконтурным графом, то правильная нумерация в нем всегда существует.

Под *рангом вершины*  $i$  понимается длина максимального пути из исходной вершины в данную вершину  $i$ . Заметим, что топологическая сортировка позволяет единственным образом распределить все вершины бесконтурного графа по рангам.

#### Алгоритм нумерации вершин, основанный на рангах

1. Положить  $r = 0$ .
2. Определить в графе вершины без входящих дуг, присвоить им ранг  $r$ , а затем удалить данные вершины из графа вместе с инцидентными дугами.
3. Если граф не пуст, то положить  $r \rightarrow r + 1$  и перейти к шагу 2, иначе получено распределение вершин по рангам.
4. Пронумеровать вершины по возрастанию рангов. Если несколько вершин имеют один и тот же ранг, то они нумеруются в произвольном порядке.

### **Определение временных параметров сетевого графика**

Рассмотрим алгоритм определения временных параметров сетевого графика и пример расчетов.

#### *Шаг 1. Топологическая сортировка сети.*

Получить правильную нумерацию вершин сетевого графика, при этом исходное событие сети получает номер 0, а завершающее – номер  $n$ .

#### *Шаг 2. Определение ранних времен наступления событий.*

Положить  $t_e(0) = 0$ . Двигаясь по пронумерованной сети в порядке возрастания вершин, для каждой вершины  $j$  определить

$$t_e(j) = \max_{i \in \Gamma^{-1}(j)} \{t_e(i) + t_{ij}\},$$

где  $\Gamma^{-1}(j) = \{i : i \rightarrow j\}$  – множество вершин, из которых дуги ведут в вершину  $j$ .

#### *Шаг 3. Определение критического пути.*

Положить  $T_{max} = t_e(n)$ , где  $n$  – завершающее событие сети. Для определения критического пути выполнить следующие действия: из всех дуг,

входящих в завершающее событие  $n$ , выделить те дуги  $(i, j)$ , которые удовлетворяют условию

$$t_e(j) - t_e(i) = t_{ij}.$$

Затем рассматриваются те вершины, из которых выходят выделенные дуги, и снова из входящих в них дуг выделяются те, которые удовлетворяют тому же условию. Процесс продолжается до тех пор, пока не будет достигнуто исходное событие сети. Путь из исходного события в завершающее событие, составленный из выделенных дуг, является критическим.

*Шаг 4. Определение поздних времен наступления событий.*

Положить

$$t_l(n) = t_e(n) = T_{max}.$$

Двигаясь по сети от завершающего события в порядке убывания номеров вершин, определить для каждого  $i$ -го события

$$t_l(i) = \min_{j \in \Gamma(i)} \{t_l(j) - t_{ij}\},$$

где  $\Gamma(i) = \{j : i \rightarrow j\}$  – множество вершин, в которые ведут дуги из  $i$ .

*Шаг 5. Определение резервов времени.*

Для каждой работы  $(i, j)$  определить резервы времени  $R_{full}(i, j), R_{free}(i, j), R_{indep}(i, j)$ .

**Замечание 1.** Алгоритм расчета ранних времен требует одной операции сложения для каждой дуги и одной операции сравнения на максимум для каждого события, кроме исходного. алгоритм расчета поздних времен требует одной операции вычитания для каждой дуги и одной операции сравнения на минимум для каждого события, кроме завершающего.

**Замечание 2.** Определение  $t_e(j)$  можно осуществлять и в том случае, если сеть не пронумерована, при этом производится столько итераций, сколько необходимо для того, чтобы очередная итерация не изменила предыдущего множества  $\{t_e(j)\}$ . Можно показать, что число итераций при этом не превысит максимального ранга вершин сети.

**Замечание 3.** При выполнении проекта критический путь может меняться, это связано с возможным изменением длительности некоторых работ, которые могут оказаться критическими.

**Пример.** Рассмотрим сетевой график, представленный на рис. 1, и рассчитаем его временные параметры.

По формулам найдем ранние времена наступления событий:

$$t_e(1) = 0, t_e(2) = 0 + 2 = 2, t_e(3) = \max\{2 + 3, 0 + 4\} = 5,$$

$$t_e(4) = \max\{0 + 6, 5 + 5\} = 10, t_e(5) = \max\{2 + 7, 10 + 5\} = 15,$$

$$t_e(6) = \max\{2 + 4, 5 + 4, 15 + 2\} = 17, t_e(7) = \max\{15 + 6, 17 + 5\} = 22.$$

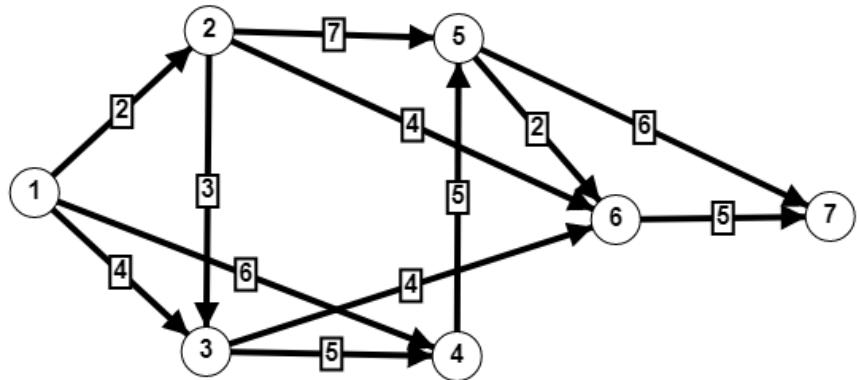


Рисунок 1 – Сетевой график

Таким образом,  $T_{\max} = 20$  и поэтому для реализации проекта, сетевой график которого представлен на рисунке, необходимо 20 единиц времени.

Найдем поздние времена событий:

$$t_l(7) = 22, t_l(6) = 22 - 5 = 17,$$

$$t_l(5) = \min\{17 - 2, 22 - 6\} = 15,$$

$$t_l(4) = 15 - 5 = 10, t_l(3) = \min\{10 - 5, 17 - 4\} = 5,$$

$$t_l(2) = \min\{5 - 3, 15 - 7, 17 - 4\} = 2,$$

$$t_l(1) = \min\{2 - 2, 5 - 4, 10 - 6\} = 0.$$

### Пример: проектирование торгово-развлекательного центра

Торгово-развлекательные центры (ТРЦ) (рис. 2) дают возможность приобрести товары различных категорий в одном месте, чередуя шопинг с развлечениями и посещениями кафе. ТРЦ, как правило, имеют несколько этажей и включают не только торговые площади, но и центры досуга – кинозалы, боулинги, квест-комнаты, островки для развлечения детей на время шопинга родителей, фуд-зону. Основная задача владельца ТРЦ – сдать объект в эксплуатацию и привлечь арендаторов, после чего центр начнет приносить доход без существенных затрат.



Рисунок 2 – Торгово-развлекательный центр

Для строительства торгово-развлекательного центра необходимо особенно тщательно провести расчеты, составить проект и сметную документацию, поскольку стартовые вложения очень велики.

В табл. 2 представлены этапы работы по строительству и вводу в эксплуатацию ТРЦ «Рябина».

Таблица 2 – Перечень этапов, работ и их продолжительностей

Работа	Содержание	Продолжительность (нед.)	Предшествующие работы
A	Анализ рынка недвижимости города	2	-
B	Выбор земельного участка	6	A
C	Разработка концепции торгового центра	4	B
D	Заключение инвестиционного договора	4	C
E	Получение кредита	8	D
F	Внесение в госреестр, постановка на учет в налоговых и административных органах	4	E
G	Предброкеридж и внесение изменений в концепцию	5	C
H	Разработка рабочей проектной документации	4	F
I	Заказ и покупка оборудования	6	G
J	Подготовка стройплощадки и закладка фундамента	6	H
K	Монтаж вертикальных стен, перекрытий и крыши	8	J

L	Монтаж наружных сетей	7	J
M	Монтаж систем отопления, вентиляции и кондиционирования	8	L
N	Электромонтажные работы	4	L
O	Отделочные работы и дизайн интерьера	9	M, N
P	Установка оборудования	3	O
Q	Наем и обучение персонала	8	O
R	Брокеридж торгового центра	2	P
S	Проведение маркетинговой компании	2	O, P
T	Подготовка к церемонии открытия	1	Q, R

На рис. 3 представлен сетевой график проекта. Соответствие между работами из табл. 2 и дугами сетевого графика представлено в табл. 3.

Таблица 3 – Соответствие между дугами и работами в сетевом графике

A	(1,2)	E	(6,7)	I	(5,14)	M	(11,12)	Q	(14,16)
B	(2,3)	F	(7,8)	J	(9,10)	N	(11,13)	R	(15,17)
C	(3,4)	G	(4,5)	K	(10,12)	O	(12,14)	S	(17,18)
D	(4,6)	H	(8,9)	L	(10,11)	P	(14,15)	T	(16,18)

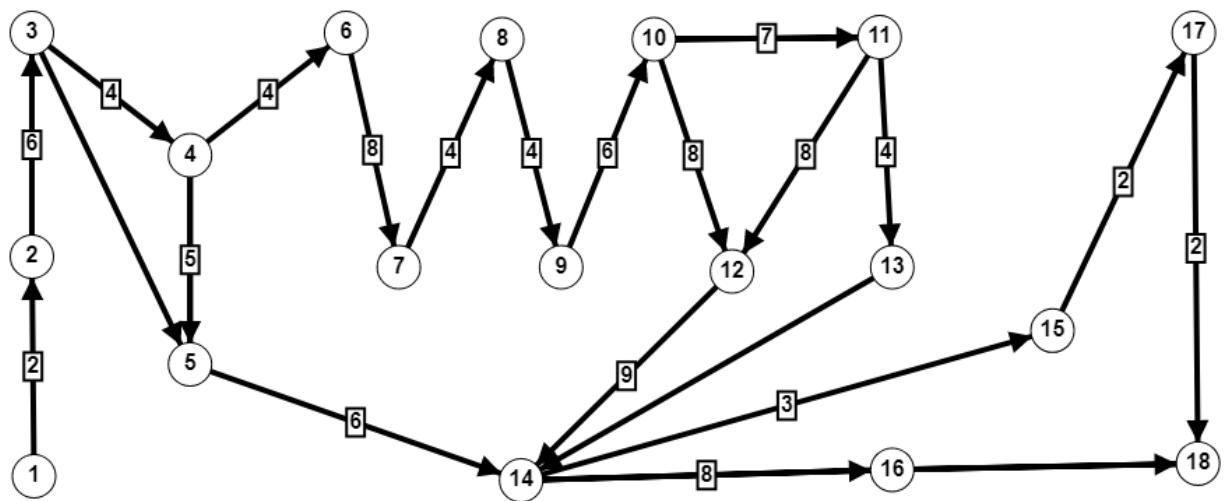


Рис. 3. Сетевой график проекта

Результаты расчетов представлены на рис. 4.

## Результаты вычислений

## Ранние и поздние времена

Номер события	Ранние времена	Поздние времена
1	0	0
2	2	2
3	8	8
4	12	12
5	17	56
6	16	16
7	24	24
8	28	28
9	32	32
10	38	38
11	45	45
12	53	53

## Резервы времени

(i, j)	Свободные резервы	Полные резервы
(1, 2)	0	0
(2, 3)	0	0
(3, 4)	0	0
(4, 5)	0	39
(4, 6)	0	0
(5, 14)	39	39
(6, 7)	0	0
(7, 8)	0	0
(8, 9)	0	0
(9, 10)	0	0
(10, 11)	0	0
(10, 12)	7	7

Критический путь: (1, 2)(2, 3)(3, 4)(4, 6)(6, 7)(7, 8)(8, 9)(9, 10)(10, 11)(11, 12)(12, 14)(14, 16)(16, 18)

Рисунок 4 – Результаты расчетов

## Лекция

### Графовые алгоритмы планирования ресурсов

#### 1. Основные определения

Графом задания называется размеченный взвешенный ориентированный бесконтурный граф  $G = (V, E, \text{init}, \text{fin}, \delta, \gamma)$ , где  $V$  – множество вершин, соответствующих задачам;  $E$  – множество дуг, соответствующих потокам данных;  $\delta(e)$  – вес дуги  $e = (v_i, v_j)$ , определяющий объем данных, который необходимо передать по дуге  $e$  от задачи, ассоциированной с вершиной  $v_i$ , к задаче, ассоциированной с вершиной  $v_j$ . Метка  $\gamma(v) = (m_v, t_v)$  вершины  $v$  определяет максимальное количество процессорных ядер  $m_v$ , на которых задача  $v$  имеет ускорение, близкое к линейному, и время  $t_v$  выполнения задачи на одном ядре. Данная модель предполагает, что вычислительная стоимость  $\varphi(v, j_v)$  задачи  $v$  на  $j_v$  процессорных ядрах определяется формулой

$$\varphi(v, j_v) = \begin{cases} \frac{t_v}{j_v}, & \text{если } 1 \leq j_v \leq m_v, \\ \frac{t_v}{m_v}, & \text{если } j_v > m_v. \end{cases} \quad (1)$$

Из определения (1) следует, что при увеличении количества процессорных ядер в диапазоне от 1 до  $m_v$ , время счета будет прямо пропорционально уменьшаться, при дальнейшем увеличении количества ядер ускорение будет отсутствовать.

На рис. 1 приведен пример задания, содержащего 8 задач. Для каждой вершины  $v$  указана метка в виде пары  $(m_v, t_v)$ ; для каждой дуги  $e$  указан объем данных  $\delta(e)$ , передаваемых по данной дуге.

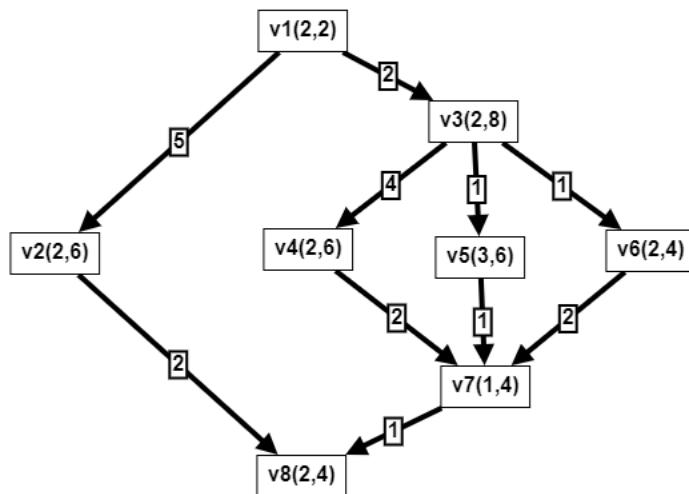


Рисунок 1 – Граф задания

Таким образом, граф задания  $G$  содержит информацию о вычислительной стоимости каждой задачи, о взаимосвязях между задачами, что предопределяет некоторый порядок при их выполнении.

Пусть  $G = (V, E)$  – бесконтурный граф. *Ярусно-параллельной формой* (ЯПФ) называется его представление в виде разложения по уровням (ярусам)  $L_i (i = \overline{1, r})$ , при этом существует монотонная нумерация вершин такая, что для каждой дуги  $e = (v_i, v_j)$  номер вершины  $v_i$  меньше номера вершины  $v_j$ .

Количество ярусов в ЯПФ называется ее *высотой*, а максимальное количество вершин в ярусах – *шириной*.

ЯПФ называется *канонической*, если все входные вершины (не имеющие входных дуг) принадлежат ярусу с номером 1, а для всех вершин, расположенных на ярусе с номером  $l$ , длина максимального пути из какой-либо входной вершины в данную вершину равна  $l - 1$ .

Поскольку граф задания является бесконтурным, то он представим в ЯПФ. Вершины без входящих дуг – начальные вершины *init*; вершины без выходящих дуг – *fin*.

*Вычислительным узлом* называется упорядоченное множество процессорных ядер  $\{C_0, C_1, \dots, C_{d-1}\}$ .

*Вычислительной системой* называется упорядоченное множество вычислительных узлов  $P = \{P_0, P_1, \dots, P_{k-1}\}$ .

*Кластеризацией* называется отображение  $w: V \rightarrow P$  множества вершин задач  $V$  графа задания  $G$  на множество вычислительных узлов  $P$ .

Пусть задана вычислительная система  $S = \{P_0, P_1, \dots, P_{k-1}\}$ , состоящая из  $k$  вычислительных узлов. Кластер  $W_i = \{v \in V : w(v) = P_i \in P\}$  – это подмножество задач, распределяемых на вычислительный узел  $P_i$ .

Заметим, что кластеры образуют разбиение множества задач, так что

$$\bigcup_{i=0}^{k-1} W_i = V, \quad \forall i \neq j (W_i \cap W_j = \emptyset).$$

В реальной ситуации вычислительная система может быть распределенной вычислительной системой, объединяющей несколько вычислительных кластеров, каждый из которых является отдельным узлом этой системы.

Граф  $G$ , для которого определена функция кластеризации, называется *кластеризованным*.

В рамках модели мы предполагаем, что время передачи любого объема данных между узлами, принадлежащими одному кластеру, близко к 0, а время передачи данных между узлами из разных кластеров, пропорционально объему передаваемых данных с коэффициентом 1. В соответствии с этим мы можем определить функцию  $\sigma: U \rightarrow \mathbb{Z}_{\geq 0}$ , вычисляющую *коммуникационную стоимость* (время) передачи данных по дуге  $e \in E$ , следующим образом:

$$\sigma(e) = \begin{cases} 0, & \text{если задачи, соответствующие концевым вершинам,} \\ & \text{из одного кластера;} \\ \delta(e), & \text{если вершины из разных кластеров.} \end{cases} \quad (2)$$

Таким образом, в кластеризованном графе коммуникационные стоимости некоторых дуг (а именно дуг, соединяющих вершины из одного кластера) становятся нулевыми, а, следовательно, время, необходимое для выполнения задания сокращается.

*Расписанием* называется отображение  $\xi : V \rightarrow \mathbb{Z}_{\geq 0} \times \mathbb{N}$ , которое каждой вершине  $v \in V$  ставит в соответствие пару  $\xi(v) = (\tau_v, j_v)$ , где  $\tau_v$  определяет время запуска задачи  $v$ ,  $j_v$  – количество ядер, выделяемых задаче  $v$ .

Обозначим через  $s_v$  – время останова задачи  $v$ , тогда

$$s_v = \tau_v + \varphi(v, j_v). \quad (3)$$

Расписание называется *корректным*, если оно удовлетворяет следующим условиям:

- 1) для любой дуги  $e = (v_1, v_2)$  время запуска задачи  $v_2$  не может быть меньше суммы следующих величин: время запуска задачи  $v_1$ , время выполнения задачи  $v_1$ , коммуникационная стоимость дуги  $e$ ;
- 2) количество ядер, выделяемое задаче  $v_1$ , не превышает границы линейной масштабируемости, заданной меткой  $\gamma(v) = (m_v, t_v)$ ;
- 3) в любой момент времени количество процессорных ядер, выделяемых задачам на узле с номером  $i$ , не может превосходить количества ядер на этом узле.

Кластеризованный граф с заданным расписанием называется *распланированным* и обозначается  $G = (V, E, init, fin, \delta, \gamma, w, \xi)$ .

Пусть в распланированном графе  $G = (V, E, init, fin, \delta, \gamma, w, \xi)$  задан путь  $\mu = [e_{i_1}, \dots, e_{i_N}]$ . Для дуги  $e_{i_k}$  начальную вершину обозначим через  $init(e_{i_k})$ , а конечную – через  $fin(e_{i_k})$ . Стоимостью пути  $\mu$  называется величина

$$u(\mu) = \varphi\left(fin\left(e_{i_N}\right), j_{fin\left(e_{i_N}\right)}\right) + \sum_{k=1}^N \left( \varphi\left(init\left(e_{i_k}\right), j_{init\left(e_{i_k}\right)}\right) + \max\left\{\sigma\left(e_{i_k}\right), \tau_{fin\left(e_{i_k}\right)} - s_{init\left(e_{i_k}\right)}\right\} \right), \quad (4)$$

где  $\varphi$  – функция, определяемая формулой (1), значением которой является вычислительная стоимость вершины;  $\sigma$  – функция, определяемая формулой (2), значением которой является коммуникационная стоимость дуги;  $\tau$  – время запуска задачи, которая соответствует вершине  $fin(e_{i_k})$ ;  $s$  – время останова задачи, которая соответствует вершине  $init(e_{i_k})$ .

Помимо формулы (4) для вычисления стоимости пути используются и другие подходы. Например, в качестве стоимостной функции пути можно рассматривать следующую функцию:

$$u(\mu) = \alpha \cdot \sum_i \varphi_i + (1 - \alpha) \left( \beta \cdot \sum_{(i,j)} \delta_{ij} + (1 - \beta) \cdot \sum_{(i^*,j)} \delta_{i^*j} \right),$$

где  $\alpha, \beta$  – весовые коэффициенты (нормализующие множители);  $\varphi_i$  – вычислительная стоимость  $i$ -й вершины;  $\delta_{ij}$  – коммуникационная стоимость дуги  $(i,j)$ ;  $\delta_{i^*j}$  – коммуникационная стоимость дуг между вершиной  $i^*$  критического пути и всеми его смежными вершинами  $j$ , не входящими в критический путь. Положим  $\alpha = \beta = 0.5$ , тогда

$$u(\mu) = 0.5 \left( \sum_i \varphi_i + 0.5 \left( \sum_{(i,j)} \delta_{ij} + \sum_{(i^*,j)} \delta_{i^*j} \right) \right).$$

Для упрощения вычислений стоимость пути будем определять по формуле

$$u(\mu) = \sum_i \varphi_i + \sum_{(i,j)} \delta_{ij}, \quad (5)$$

суммируя вычислительные стоимости вершин-задач и коммуникационные стоимости дуг.

Пусть  $\{\mu\}$  – множество путей в графе  $G$ , тогда путь  $\mu^*$  с максимальной стоимостью называется *критическим* (обозначение  $P_{\max}$ ).

Важным показателем при составлении расписаний является параллельное время, которое определяется по распланированному графу. Пусть  $p_1(i)$  – длина максимального пути из вершины *init* в вершину  $i$  без учета веса вершины  $i$ ;  $p_2(i)$  – длина максимального пути из вершины  $i$  в вершину *fin*, тогда *параллельное время*  $PT$  для распланированного графа вычисляется по формуле

$$PT = \max_{i \in V} \{ p_1(i) + p_2(i) \}. \quad (6)$$

Независимые задачи в кластере упорядочиваются на основе величин  $p_2(i)$ , полученных на предыдущем шаге.

## 2. Алгоритмы кластеризации в параллельных и распределенных системах

Основная идея данных алгоритмов заключается в кластеризации взаимосвязанных задач в маркированные группы для дальнейшего их присвоения некоторой группе ресурсов. Применение алгоритмов кластеризации – это эффективный способ уменьшить коммуникационную задержку.

Если кластер содержит только одну задачу, то он называется *единичным*. Кластер называется *нелинейным*, если он содержит независимые задачи, в противном случае он называется *линейным*. Если порядок выполнения работ в линейном кластере понятен, то для нелинейного кластера необходимо составить расписание.

Если вершины-задачи, являющиеся концевыми вершинами некоторой дуги, попали в один кластер, то коммуникационная стоимость дуги полагается равной 0, соответствующие задачи выполняются на одном процессоре. Порядок выполнения задач в кластере определяется наибольшим значением суммы всех коммуникационных стоимостей дуг и вычислительных стоимостей вершин, принадлежащих пути между данной вершиной-задачей и нижней вершиной в ЯПФ.

Пусть имеется вычислительная система в виде упорядоченного множества вычислительных узлов  $P = \{P_0, \dots, P_{k-1}\}$  и граф задания  $G = (V, E, init, fin, \delta, \gamma)$ , причем  $|V| \leq |P|$ .

При использовании алгоритмов кластеризации граф  $G$  представляется в канонической ярусной форме с уровнями  $L_i (i = \overline{1, r})$ . Правильная нумерация вершин позволяет определить последовательность выполнения задач.

#### *Алгоритм линейной кластеризации Кима и Брауна (КВ/L)*

Данный алгоритм предназначен для кластеризации графа задания и предполагает, что количество вычислительных узлов неограничено. Целевой функцией является минимизация параллельного времени графа.

Шаг 1. Построить ЯПФ графа задания  $G = (V, E, init, fin, \phi, \delta)$ . Все дуги графа задания маркировать как «нерассмотренные».

Шаг 2. С помощью стоимостной функции (5) определить критический путь  $\mu^*$  из вершины верхнего уровня  $init$  в вершину нижнего уровня  $fin$ , который включает только «нерассмотренные» дуги.

Шаг 3. Все вершины найденного критического пути объединяются в один кластер, коммуникационные стоимости дуг обнуляются. Дуги, инцидентные вершинам критического пути, маркируются как «рассмотренные».

Шаг 4. Шаги 2-3 повторять до тех пор, пока все дуги не будут рассмотрены.

**Пример.** Пусть граф задания имеет вид, изображенный на рис. 2. Каждой вершине-задаче соответствует вычислительная стоимость, каждой дуге – коммуникационная стоимость. Здесь  $init = v_1$ ,  $fin = v_7$ . Рассматривая все возможные пути из вершины  $v_1$  в вершину  $v_7$  (все дуги «нерассмотренные»), вычислим стоимость каждого пути. Здесь имеем следующие пути:

$$\mu_1 = [v_1, v_2, v_7], u(\mu_1) = (3 + 7 + 5) + (5 + 5) = 25,$$

$$\mu_2 = [v_1, v_3, v_4, v_6, v_7], u(\mu_2) = (3 + 3 + 1 + 2 + 5) + (1 + 2 + 1 + 1) = 19,$$

$$\mu_3 = [v_1, v_3, v_5, v_6, v_7], u(\mu_3) = (3 + 3 + 2 + 2 + 5) + (1 + 3 + 2 + 1) = 22.$$

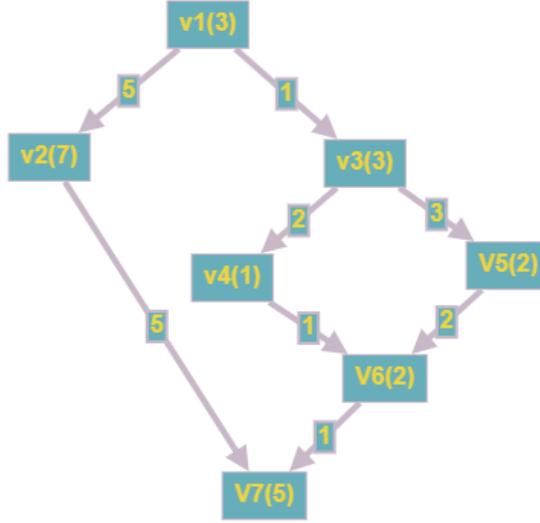


Рис. 2. – Граф задания, представленный в ЯПФ

Таким образом, путь  $\mu_1$  имеет максимальное значение стоимости и, следовательно, является критическим. Вершины-задачи данного пути объединяются в один кластер  $W_1 = \{v_1, v_2, v_7\}$ ; коммуникационные стоимости дуг пути обнуляются; дуги критического пути помечаются как «рассмотренные» (на рис. 3 «рассмотренные» дуги желтого цвета), вершины из  $W_1$  исключаются из рассмотрения. Так как не все дуги рассмотрены, то алгоритм продолжает работу. Теперь рассмотрим подграф исходного графа на вершинах  $\{v_3, v_4, v_5, v_6\}$  (рис. 3). Найдем длины путей из  $v_3$  в  $v_6$ .

$$\begin{aligned}\mu_1 &= [v_3, v_4, v_6], u(\mu_1) = (3 + 1 + 2) + (2 + 1) = 9, \\ \mu_2 &= [v_3, v_5, v_6], u(\mu_2) = (3 + 2 + 2) + (3 + 2) = 12.\end{aligned}$$

Критическим является путь  $\mu_2$ . Вершины, которые принадлежат этому пути, объединяются в кластер  $W_2 = \{v_3, v_5, v_6\}$ , коммуникационные стоимости дуг обнуляются, и дуги считаются просмотренными. После двух шагов остается одна вершина  $v_4$ , которая образует кластер  $W_3 = \{v_4\}$ . Найдем параллельное время по графу на рис. 4:

$$v_2 \rightarrow 3 + 7 + 5 = 15;$$

$$v_3 \rightarrow 3 + 1 + \underbrace{3 + 2 + 1 + 1 + 2 + 1 + 5}_{p_2} = 19;$$

$$v_4 \rightarrow \underbrace{3 + 1 + 3 + 2}_{p_1} + \underbrace{1 + 1 + 2 + 1 + 5}_{p_2} = 19;$$

$$v_5 \rightarrow \underbrace{3 + 1 + 3 + 0}_{p_1} + \underbrace{2 + 0 + 2 + 1 + 5}_{p_2} = 17;$$

$$v_6 \rightarrow \underbrace{3 + 1 + 3 + 2 + 1 + 1}_{p_1} + \underbrace{2 + 1 + 5}_{p_2} = 19.$$

<p>Рисунок 3 – Граф задания после первого шага</p>	<p>Рисунок 4 – Граф задания после второго шага</p>

Таким образом, если задачи распределить по трем кластерам и выполнять их параллельно, то общее время обработки составит 19 ед. Полученная кластеризация является линейной, так как в каждом кластере последовательность выполнения работ четко определена.

### *Алгоритм Саркара*

Шаг 1. Составить список дуг графа задания в порядке убывания их коммуникационной стоимости.

Шаг 2. Выбрать дугу из списка с максимальной коммуникационной стоимостью. Обнулить коммуникационную стоимость и вычислить параллельное время. Если параллельное время не увеличилось, то сформировать кластер, включая в него вершины, принадлежащие выбранному ребру. Иначе перейти к следующей дуге в списке.

Шаг 3. Шаг 2 повторять до тех пор, пока не будут рассмотрены все дуги.

Таким образом, начиная с дуги с наибольшей коммуникационной стоимостью, производится процесс обнуления: коммуникационная стоимость дуги обнуляется только в том случае, если параллельное время не увеличится на следующем шаге

**Пример 1.** Рассмотрим график задания, представленный на следующем рисунке. Применим к нему алгоритм Саркара. Составим список дуг по убыванию их коммуникационных стоимостей

$$\{(v_1, v_2), (v_3, v_4), (v_3, v_5), (v_2, v_7), (v_4, v_6), (v_5, v_6), (v_1, v_3), (v_6, v_7)\}.$$

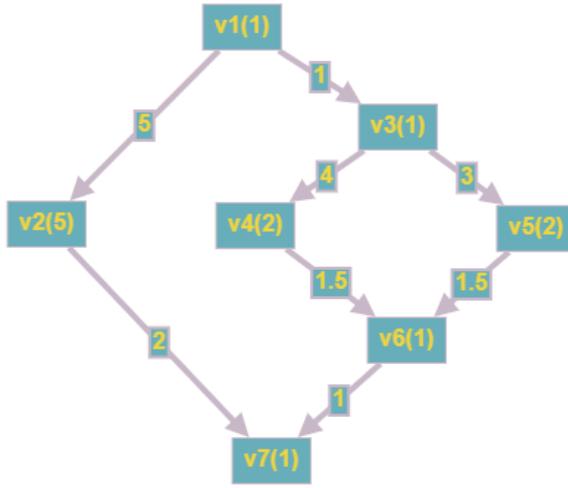
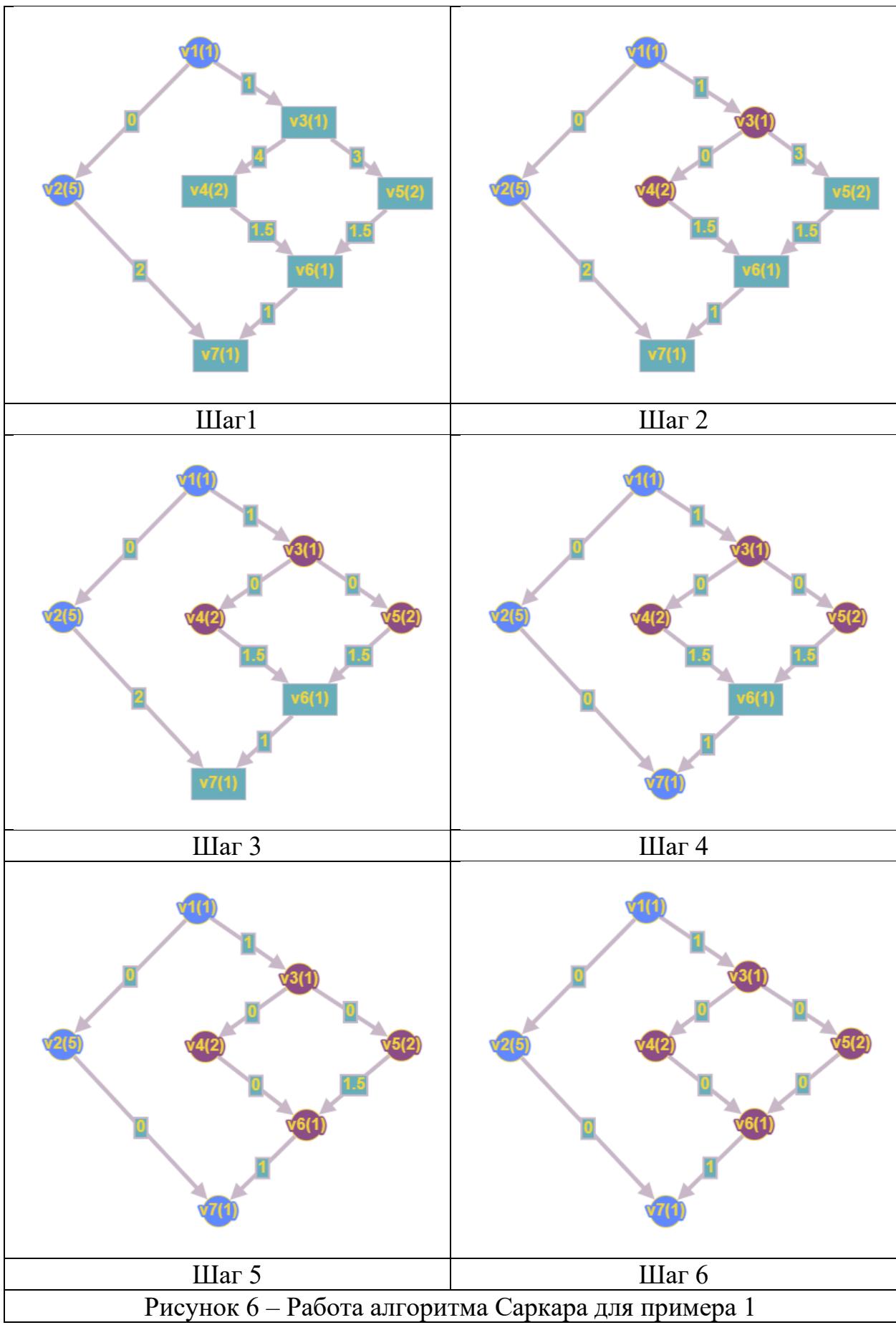


Рисунок 5 – Граф задания

Сначала каждая вершина графа задания находится в отдельном кластере, параллельное время  $PT = 14$ . Рассмотрим первую в списке дугу  $(v_1, v_2)$ . Если ее коммуникационную стоимость обнулить, то параллельное время уменьшится до 13.5, поэтому принимается решение об обнулении, вершины  $v_1$  и  $v_2$  образуют кластер  $W_1 = \{v_1, v_2\}$ . Граф задания после первого шага изображен на рис. 6. Следующая в списке дуга  $(v_3, v_4)$ . Если ее обнулить, то параллельное время уменьшится до 12.5, поэтому она обнуляется, а вершины образуют кластер  $W_2 = \{v_3, v_4\}$ . Обнуление коммуникационной стоимости дуги  $(v_3, v_5)$  приведет к тому, что параллельное время уменьшится до 9.5, поэтому дуга обнуляется. Так как вершина  $v_3$  уже принадлежит кластеру, то к нему добавляется и вершина  $v_5$ , тогда на данном шаге получим кластер  $W_2 = \{v_3, v_4, v_5\}$ . Следующая в списке дуга –  $(v_2, v_7)$ . Если ее обнулить, то параллельное время не изменится, поэтому коммуникационная стоимость дуги  $(v_2, v_7)$  полагается равной 0, вершина  $v_7$  добавляется в кластер  $W_1 = \{v_1, v_2, v_7\}$ . Теперь рассмотрим дугу  $(v_4, v_6)$ , ее обнуление не приведет к увеличению параллельного времени 9.5, поэтому, коммуникационная стоимость обнуляется, вершина  $v_6$  помещается в кластер  $W_2$ , тогда получим  $W_2 = \{v_3, v_4, v_5, v_6\}$ . Заметим, что вершины распределились по кластерам, но коммуникационные стоимости дуг, которые соединяют вершины одного кластера должны быть нулевые. Выбираем следующую в списке дугу  $(v_5, v_6)$ . Ее обнуление приводит к тому, что параллельное время уменьшится до 8, поэтому кластер  $W_2 = \{v_3, v_4, v_5, v_6\}$  остается без изменений. Полученный на последнем шаге график является кластеризованным с параллельным временем 8.

Заметим, что алгоритм Саркара позволяет получить нелинейную кластеризацию.



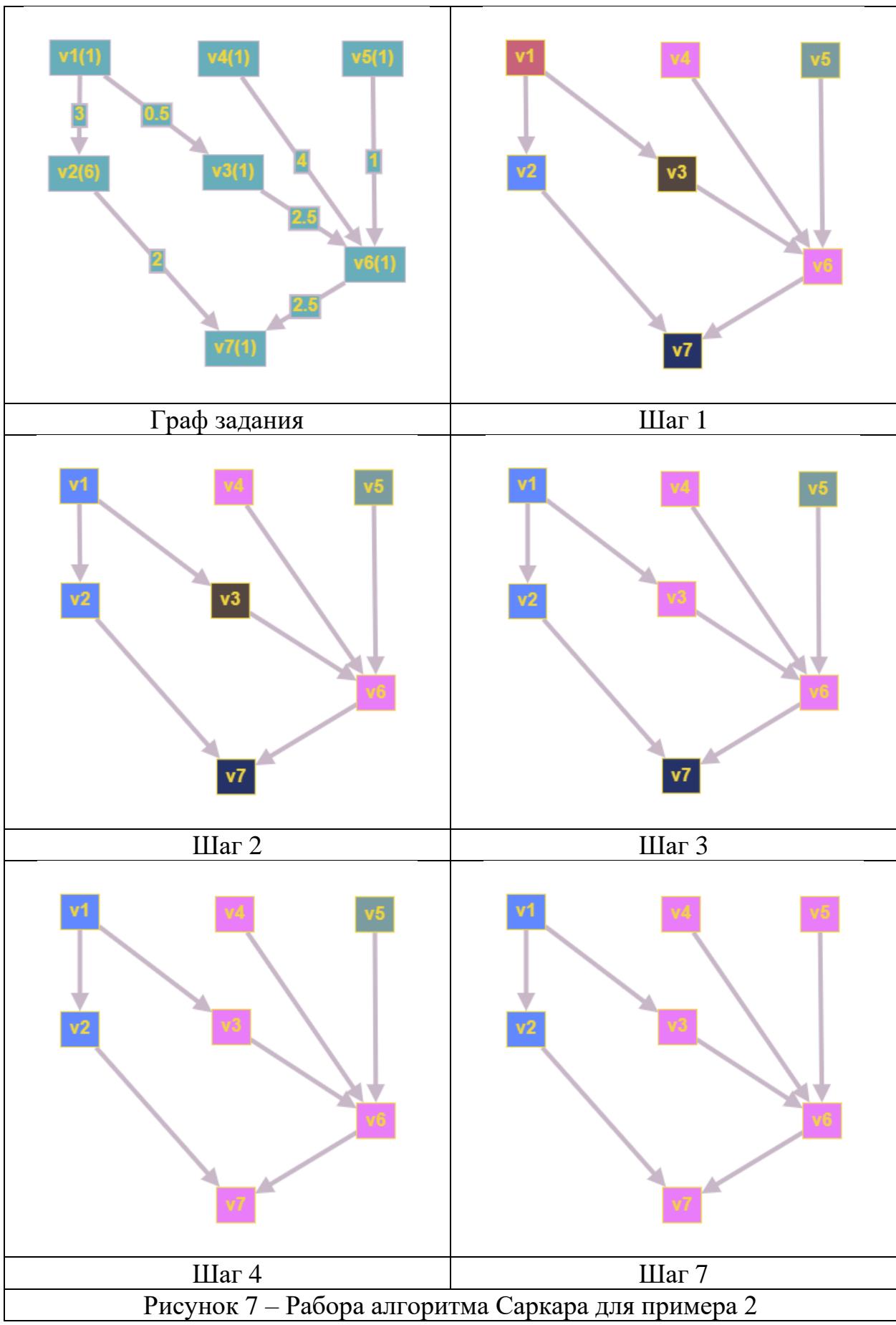
**Пример 2.** Упорядоченный список ребер имеет вид

$$\{(v_4, v_6), (v_1, v_2), (v_3, v_6), (v_6, v_7), (v_2, v_7), (v_1, v_3), (v_5, v_6)\}.$$

В табл. 1 представлена работа алгоритма. Здесь  $PT$  обозначает параллельное время, а  $PT_i$  – параллельное время для кластеризованного графа после  $i$ -й итерации. Изначально каждая задача находится в отдельном кластере,  $\mu = [v_1, v_2, v_7]$ ,  $PT_0 = 13$ . На шаге 1 исследуется ребро  $(v_4, v_6)$ : если его коммуникационная стоимость обнуляется, то  $PT$  остается равным 13 – это обнуление принимается. На шагах 2,3,4 стоимости всех рассматриваемых дуг обнуляются, так как каждое обнуление не увеличивает  $PT$ . На шаге 5 проверяется дуга  $(v_2, v_7)$ , и при его обнулении параллельное время увеличивается с 10 до 11. Таким образом, это обнуление отвергается. Аналогично, на шаге 6 не обнуляется дуга  $(v_1, v_3)$ . На шаге 7 обнуляется дуга  $(v_5, v_6)$ , и получается два кластера  $W_1 = \{v_1, v_2\}$  и  $W_2 = \{v_3, v_4, v_5, v_6\}$ . Параллельное время равно 10. На рис. 7 показаны этапы кластеризации алгоритма Саркара. Вершины, принадлежащие одному кластеру, окрашиваются в один цвет.

Таблица 1

Шаг	Дуга	Критический путь, если стоимость дуги обнуляется	Обнуление?	Параллельное время на данном шаге
0				13
1	$(v_4, v_6)$	13	да	13
2	$(v_1, v_2)$	10	да	10
3	$(v_3, v_6)$	10	да	10
4	$(v_6, v_7)$	10	да	10
5	$(v_2, v_7)$	11	нет	10
6	$(v_1, v_3)$	11	нет	10
7	$(v_5, v_6)$	10	да	10



## *Алгоритм кластеризации доминирующей последовательностью* (Dominant Sequence Clustering – DSC)

Доминирующая последовательность – это критический путь распланированного графа. Идея алгоритма DSC состоит в выполнении последовательности шагов по обнулению коммуникационных стоимостей дуг графа задания с целью сокращения длины доминирующей последовательности.

В начальный момент времени каждая вершина образует кластер, все дуги графа помечаются как «не просмотренные». После рассмотрения некоторой дуги на предмет возможности ее обнуления, дуга помечается как «просмотренная», а вершина, из которой исходит дуга, помечается как «распланированная». «Распланированные» вершины образуют множество  $S$ , а «не распланированные» – множество  $\bar{S}$ . Вершина называется «свободной», если все ее предшествующие вершины «распланированные».

На первом шаге алгоритма из дуг, принадлежащих доминирующей последовательности графа задания, выбирается первая не просмотренная дуга. Поиск дуги в доминирующей последовательности осуществляется сверху вниз. На втором шаге дуга обнуляется, а ее вершины объединяются в один кластер, если параллельное время не увеличивается. Алгоритм DSC завершает работу, когда все пути рассмотрены.

**Пример.** На рис. 5 изображен граф задания, включающий 7 вершин задач

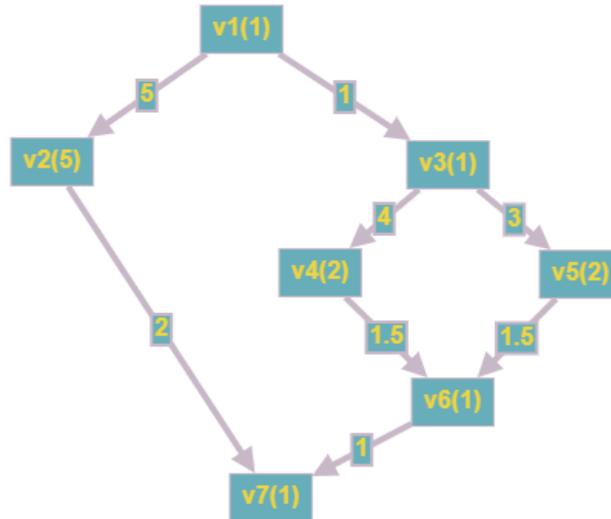


Рис. 5 – Граф задания из 7 задач