В. М. КОТОВ
Е. П. СОБОЛЕВСКАЯ
А. А. ТОЛСТИКОВ

«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

ГЛАВА 4. ОСНОВНЫЕ АЛГОРИТМЫ НА ГРАФАХ

Допущено
Министерством образования Республики Беларусь
в качестве учебного пособия для студентов
учреждения высшего образования
по специальностям «Прикладная математика»,
«Информатика», «Актуарная математика»

Минск
БГУ
2011

УДК 519.712(075.8)
ББК 22.18я73
К73

Рецензенты:
Кафедра математического и информационного обеспечения
Экономических систем ГГУ имени Янки Купалы
(заведующий кафедрой кандидат физико-математических наук, доцент
В.И. Ляликова);
доктор физико-математических наук, профессор С.В. Колосов

Котов, В.М.
К73 Алгоритмы и структуры данных: учеб. пособие / В.М. Котов, Е.П. Соболевская, А.А. Толстиков. – Минск : БГУ, 2011. – 267 с. – (Классическое университетское издание).
ISBN 978-985-518-530-8.

В учебном пособии изложены фундаментальные понятия, используемые при разработке алгоритмов и оценке их трудоемкости. Теоретический материал дополнен примерами и рисунками, облегчающими самостоятельное изучение материала, а также перечнем задач для самостоятельного решения. В приложении разбираются алгоритмы решения творческих задач повышенной сложности.

Предназначено для студентов высших учебных заведений, обучающихся по специальтностям «Прикладная математика», «Информатика», «Актуарная математика».

СОДЕРЖАНИЕ

Глава 4. ОСНОВНЫЕ АЛГОРИТМЫ НА ГРАФАХ

4.1. Графы. Основные определения…… 4

4.2. Поиск в глубину в графе…….. 9

4.3. Поиск в ширину в графе…….. 17

4.4. Пути в орграфах. Маршруты в графах ……... 21

4.4.1. Кратчайший элементарный путь…… 21

4.4.2. Кратчайшие пути (маршруты)…… 33

4.4.3. Максимальный путь в бесконтурном орграфе…….. 36

4.4.4. Эйлеров цикл в графе…….. 39

4.4.5.Наибольшее число маршрутов между заданной парой вершин, не пересекающихся по ребрам…….…… 43

4.4.6. *k*-кратчайших (v, w) маршрутов, не пересекающихся по ребрам…… 46

4.5. Максимальный поток в сети и его приложения……. 50

4.6. Циклы отрицательной стоимости…… 67

4.6.1. Наибольшее паросочетание максимального веса в двудольном графе…… 67

4.6.2. Максимальный поток минимальной стоимости……. 71

4.6.3. Минимальный средний контур в орграфе с положительными стоимостями дуг…… 75

4.7. Минимальное остовное дерево графа……. 79

4.8. Кратчайший маршрут с нечетным числом ребер…….. 85

4.9. Задача китайского почтальона…… 88

4.10. Задачи для самостоятельного решения……. 92

Г л а в а 4

ОСНОВНЫЕ АЛГОРИТМЫ НА ГРАФАХ

Существует много алгоритмов на графах, в основе которых лежит такой перебор вершин графа, при котором каждая вершина просматривается в точности один раз. Поэтому важной задачей является нахождение хороших методов поиска в графе.

Будем говорить, что:

- вершина *помеченная*, если она занесена в структуру (т. е. ей дается некоторая метка);
- вершина *просмотренная*, если она удалена из структуры первый раз (иногда в структуре некоторая вершина может храниться несколько раз, но с разными метками, и первое удаление данной вершины из структуры делает ее просмотренной).

4.1. ГРАФЫ. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Ориентированный граф (или короче – *орграф*) $G=(V,U)$ состоит из непустого множества узлов V и множества упорядоченных пар узлов (вершин) U. Упорядоченная пара узлов (v, w) называется дугой (v – начало дуги; w – конец дуги). *Неориентированный граф* (или короче – *граф*) $G=(V,E)$ состоит из непустого множества узлов V и множества пар узлов (вершин) E. Неупорядоченная пара узлов (v, w) называется *ребром*. В дальнейшем будем предполагать $|V|=n$, $|U|=m$, $|E|=m$.

Если (v, w) – ребро (дуга), то говорят, что это ребро (дуга) *инцидентно* вершинам v и w, а эти вершины *смежные* между собой. *Степень вершины* для графа определяется как число ребер, инцидентных ей. Для орграфа определяют *полустепень захода* (число заходящих в вершину дуг) и *полустепень исхода* (число выходящих из вершины дуг).

Путем в орграфе $G=(V,U)$ называется последовательность вершин вида $v_1,v_2,\ldots,v_k$, где $(v_i,v_{i+1})\in U, i=1,\ldots k-1$. *Альтернирующим путем* называется последовательность вершин $v_1,v_2,\ldots,v_k$, где $(v_i,v_{i+1})\in U$ или $(v_{i+1},v_i)\in U, i=1,\ldots k-1$. *Маршрутом* (*цепью*) в графе $G=(V,E)$ называется последовательность вершин вида $v_1,v_2,\ldots,v_k$ где $(v_i,v_{i+1})\in E, i=1,\ldots k-1$.

Элементарный путь в орграфе – это такой путь, в котором каждая вершина встречается не более одного раза. *Элементарный маршрут* в графе – это такая цепь, в которой каждая вершина встречается не более одного раза.

Элементарный контур в орграфе – это замкнутый путь, в котором каждая вершина встречается не более одного раза (за исключением начальной и конечной вершин контура). *Элементарный цикл* в графе – это замкнутый маршрут, в котором каждая вершина встречается не более одного раза (за исключением начальной и конечной вершин цикла).

Гамильтонов контур в орграфе – это такой замкнутый путь, который проходит через каждую вершину орграфа один и только один раз (за исключением начальной и конечной вершин контура). *Гамильтонов цикл* в графе – это такой замкнутый маршрут, который проходит через каждую вершину графа один и только один раз (за исключением начальной и конечной вершин цикла).

Простой путь в орграфе – это такой путь, в котором каждая дуга встречается не более одного раза. *Простой маршрут* в графе – это такой маршрут, в котором каждое ребро встречается не более одного раза.

Эйлеров контур в орграфе – это такой замкнутый путь, который проходит ровно один раз по каждой дуге. *Эйлеров цикл* в графе – это такой замкнутый маршрут, который проходит ровно один раз по каждому ребру.

Предположим, что задан орграф $G=(V,U)$ (граф $G=(V,E)$), каждой дуге (ребру) (v, w) которого приписана некоторая стоимость $c(v, w)$. Такой орграф (граф) будем называть *взвешенным*. *Кратчайший путь* во взвешенном орграфе – это такой путь, суммарная стоимость дуг которого минимальна. *Кратчайший маршрут* во взвешенном графе – это такой маршрут, суммарная стоимость ребер которого минимальна.

Частичным графом графа $G=(V,E)$ называется такой произвольный граф $G'=(V',E')$, что $V'\subseteq V$ и $E'\subseteq E$. *Подграф* графа $G=(V,E)$ – это частичный подграф $G'=(V',E')$, который удовлетворяет дополнительному условию $\forall v,w\in V'$, если $(v,w)\in E$, то следует, что $(v,w)\in E'$.

Граф называется *связным*, если для любой пары вершин существует цепь, их соединяющая. В противном случае граф называется *несвязным*.

Орграф называется *сильносвязным*, если для любой пары вершин существует путь как в одну, так и в другую сторону; орграф называется *слабосвязным*, если между любой парой вершин существует альтернирующий путь, в противном случае орграф называется *несвязным*.

Максимальный подграф, в котором между любой парой вершин существует путь как в одну, так и в другую сторону, называется *сильносвязной* компонентой орграфа.

Граф называется *двудольным*, если множество его вершин можно разбить на два подмножества (доли) таким образом, что каждое ребро соединяет только вершины различных подмножеств. Граф называется *полным двудольным*, если каждая вершина одной доли соединена ребром с каждой вершиной другой доли.

Приведем отличия в основных понятиях для графа и орграфа (табл. 4.1)

Таблица 4.1

Орграф $G=(V,U)$	**Граф** $G=(V,E)$
дуга	ребро
полустепень захода (исхода)	степень вершины
путь	маршрут (цепь)
элементарный путь	элементарный маршрут
элементарный контур	элементарный цикл
гамильтонов контур	гамильтонов цикл
простой путь	простой маршрут
эйлеров контур	эйлеров цикл
кратчайший путь	кратчайший маршрут
сильносвязный орграф слабосвязный орграф	связный граф

На практике граф часто изображают следующим образом: каждому узлу ставится в соответствие точка плоскости, а для изображения ребер (дуг) используют линии (направленные линии), соединяющие соответствующие точки. На рис. 4.1 изображен орграф $G=(V,U)$, где V = {1, 2, 3, 4, 5, 6}; U={(1, 2), (1, 3), (3, 2), (3, 4), (5, 4), (5, 6),(6, 5)}.

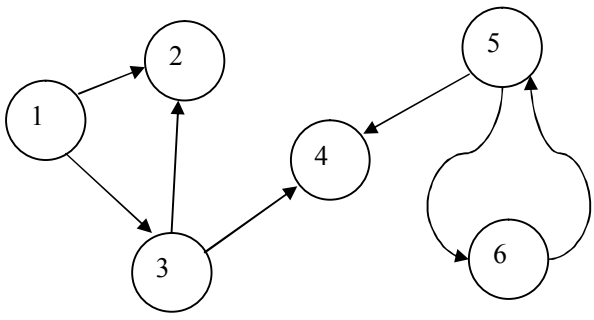


Рис. 4.1

Существуют различные способы задания графа в памяти. Рассмотрим некоторые из них.

Одним из классических способов задания графа (орграфа) является *матрица смежности* A размером $n \times n$, где $a[v, w] = 1$, если существует ребро (дуга), соединяющее вершины v и w (идущее из вершины v в вершину w), и $a[v, w] = 0$ в противном случае. Так, для орграфа, изображенного на рис. 4.1, матрица смежности будет иметь вид

i/*j*	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Преимущество такого представления заключается в том, что за время $O(1)$ можно дать ответ на вопрос, существует ли ребро (дуга) (v, w). Недостаток заключается в том, что объем требуемой памяти вне зависимости от количества ребер (дуг) составляет n^2.

Другим традиционным способом представления графа (орграфа) служит *матрица инцидентности*. Это матрица A с n строками, соответствующими вершинам, и m столбцами, соответствующими ребрам (дугам). Для графа столбец, соответствующий ребру (v, w), содержит единицы в строках, соответствующих вершинам v и w, и нули во всех остальных строках. Для орграфа столбец, соответствующий дуге (v, w), содержит: единицу в строке, соответствующей вершине v, минус единицу в строке, соответствующей вершине w, нули во всех остальных строках (петлю удобно представлять другим значением, например 2). Так, для орграфа, изображенного на рис. 4.1, матрица инцидентности будет иметь вид

v/(*v*, *w*)	(1, 2)	(1, 3)	(3, 2)	(3, 4)	(5, 4)	(5, 6)	(6, 5)
1	1	1	0	0	0	0	0
2	–1	0	–1	0	0	0	0
3	0	–1	1	1	0	0	0
4	0	0	0	–1	–1	0	0
5	0	0	0	0	1	1	–1
6	0	0	0	0	0	–1	1

Этот способ задания графа (орграфа) имеет ряд недостатков. Во-первых, требует $n \cdot m$ ячеек памяти, которые в своем большинстве заполнены нулями. Во-вторых, ответы на вопросы «Существует ли ребро (дуга) (v, w)?», «Какие вершины смежные с вершиной v?», «К каким

вершинам ведут дуги из вершины *v*?» требуют в худшем случае перебора всех столбцов матрицы инцидентности.

Более экономным в отношении памяти, особенно когда $m << n^2$, (читается: «*m* намного меньше n^2»), является *метод представления графа с помощью списка пар*, соответствующим ребрам (дугам): пара (*v*, *w*) соответствует ребру (дуге) (*v*, *w*). Так, для орграфа, изображенного на рис. 4.1, список пар дуг будет иметь вид

v	1	1	3	3	5	5	6
w	2	3	2	4	4	6	5

Объем памяти в этом случае составляет $O(m)$, но неудобство заключается в большом количестве шагов (в худшем случае порядка *m*), необходимом для получения множества вершин, к которым ведут дуги из данной вершины *v* (которые смежные с вершиной *v*).

Во многих ситуациях наиболее приемлемым способом задания графа (орграфа) является структура данных, которую часто называют *списками смежности*. Данная структура может быть описана следующим образом:

```
type
   uk = ^ el;
    el= record
        key : byte;
        next : uk;
   end;
var  graf : array [1..n] of uk;
```

Для графа: массив *graf* – статический массив размера *n*, *i*-й элемент массива служит указателем на начало линейного списка вершин, смежных с *i*-й вершиной; если степень вершины равна нулю, то $graf[i] = nil$. Для орграфа: массив *graf* – статический массив размера *n*, *i*-й элемент массива – указатель на начало линейного списка вершин, в которые ведут дуги из вершины *i*, если из вершины *i* нет выходящих дуг, то $graf[i] = nil$. Так, для орграфа, изображенного на рис. 4.1, списки смежности будут иметь вид как на рис. 4.2.

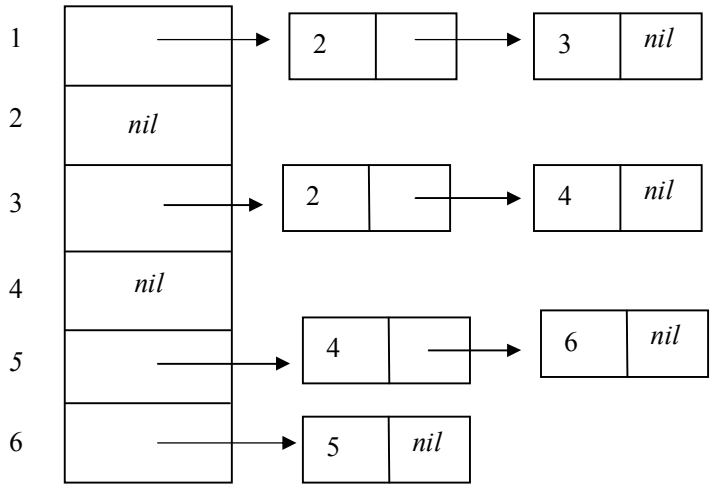


Рис. 4.2.

Таким образом, граф (орграф) представлен в виде n списков смежностей, по одному для каждой вершины. Объем памяти в этом случае $\mathrm{O}(n+m)$.

4.2. ПОИСК В ГЛУБИНУ

Поиск в глубину – один из методов обхода всех вершин и ребер графа (орграфа), который послужил основой для разработки ряда эффективных алгоритмов на графах. В процессе поиска в глубину в графе вершинам присваиваются метки, которые соответствуют порядку обхода вершин графа, а его ребра помечаются как «древесные» или «обратные». В процессе поиска в глубину в орграфе дуги помечаются как «древесные», «обратные», «прямые» или «поперечные». В начале работы алгоритма все вершины не имеют меток, а все ребра (дуги) не помечены.

Для программной реализации алгоритма поиска в глубину будем использовать структуру данных *стек*. В процессе работы алгоритма каждая вершина включается и исключается из стека только один раз. Она включается в стек после того, как ей присваивается метка, и исключается из стека, когда происходит возвращение из этой вершины.

Алгоритм поиска в глубину
в орграфе

1. Некоторой стартовой вершине *s* присваивается метка $k = 1$, и вершина *s* заносится в стек.

2. Пока стек не станет пуст (произойдет возвращение в вершину *s*, а из нее все дуги уже помечены), повторять следующие действия: пусть *v* – последняя занесенная в стек вершина и *k* – последняя присвоенная метка, тогда возможна одна из ситуаций:

- имеется непомеченная дуга (*v*, *w*), а у вершины *w* уже имеется метка, тогда продолжаем поиск непомеченной дуги, выходящей из вершины *v*, предварительно помечая дугу (*v*, *w*) по правилам:

– если вершина *w* находится в стеке, то дугу (*v*, *w*) помечаем как «обратную» (back edges);

– если вершина *w* уже была удалена из стека и ее метка меньше, чем метка вершины *v*, то дугу (*v*, *w*) помечаем как «поперечную» (cross edges);

– если вершина *w* уже была удалена из стека и ее метка больше, чем метка вершины *v*, то дугу (*v*, *w*) помечаем как «прямую» (forward edges);

- имеется непомеченная дуга (*v*, *w*), а вершина *w* не имеет метки (является новой), тогда:

– присваиваем вершине *w* метку, равную $k = k + 1$;

– помечаем дугу (*v*, *w*) как «древесную» (tree edges);

– заносим вершину *w* в стек;

– возвращаемся к шагу 2 алгоритма;

- все дуги, выходящие из вершины *v*, уже помечены, тогда:

– удаляем вершину *v* из стека (вершина *v* становится просмотренной);

– возвращаемся к шагу 2 алгоритма.

3. Если существуют непомеченные вершины, то необходимо выбрать любую из них в качестве стартовой, присвоить ей метку $k = k + 1$, занести вершину в стек и вернуться к шагу 2 алгоритма.

Пример 4.1. Выполнить поиск в глубину для орграфа, приведенного на рис. 4.3.

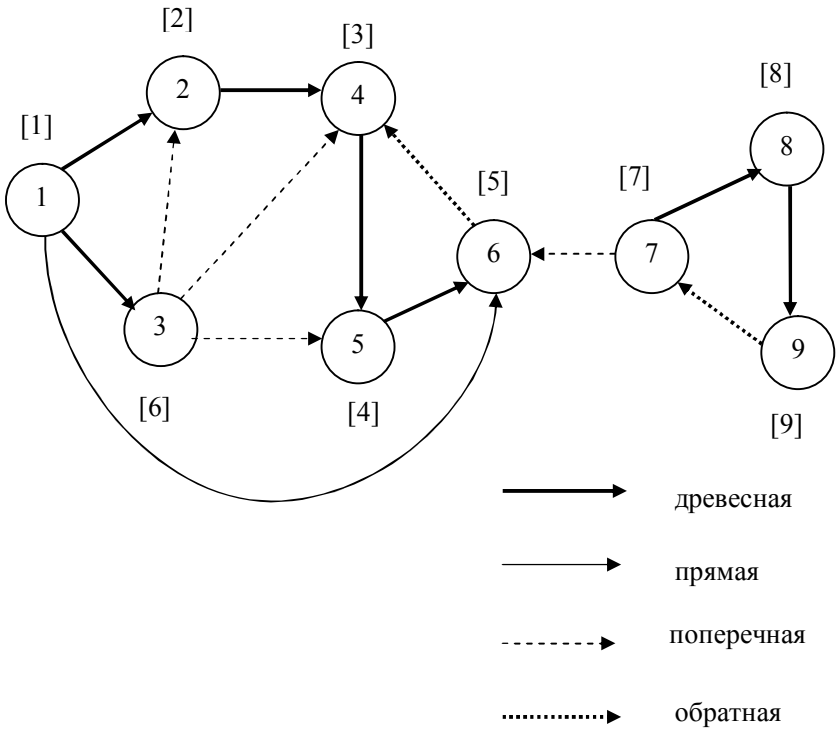


Рис. 4.3 Поиск в глубину в орграфе

Решение. В результате поиска в глубину каждой вершине присвоена метка, которая указана возле вершины в квадратных скобках, а все дуги помечены как «древесные», «обратные», «прямые» или «поперечные» (рис. 4.3). Поиск в глубину в орграфе заключался сначала в поиске в глубину из вершины 1, а затем в поиске в глубину из вершины 7.

Алгоритм поиска в глубину в графе

1. Некоторой стартовой вершине s присваивается метка $k = 1$, и вершина s заносится в стек.
2. Пока стек не станет пуст (произойдет возвращение в вершину s, а из нее все ребра уже помечены), повторять следующие действия: пусть v – последняя занесенная в стек вершина и k – последняя присвоенная метка, тогда возможна одна из ситуаций:

а) имеется непомеченное ребро (v, w), а у вершины w уже имеется метка, тогда продолжаем поиск непомеченного ребра, инцидентного вершине v, помечая ребро (v, w) как «обратное»;

б) имеется непомеченное ребро (*v*, *w*), а вершина *w* не имеет метки (является новой), тогда:

- присваиваем вершине *w* метку, равную $k = k + 1$;
- помечаем ребро (*v*, *w*) как «древесное»;
- заносим вершину *w* в стек;
- возвращаемся к шагу 2 алгоритма;

в) все ребра, инцидентные вершине *v*, уже помечены, тогда:

- удаляем вершину *v* из стека (вершина *v* становится просмотренной);
- возвращаемся к шагу 2 алгоритма.

4. Если существуют непомеченные вершины, то необходимо выбрать любую из них в качестве стартовой, присвоить ей метку $k = k + 1$, занести в стек. Вернуться к шагу 2 алгоритма.

В дальнейшем будем понимать под *поиском в глубину в графе* такой поиск, когда все вершины графа получают метки, а под *поиском в глубину из вершины s* такой поиск, который прекращается, как только стек становится пустым, т. е. становятся помеченными только все достижимые из *s* вершины.

Оценим трудоемкость алгоритма поиска в глубину. Каждое включение и исключение вершины из стека выполняется за время $O(1)$. Поскольку каждая вершина помечается и просматривается только один раз, то для всей работы, связанной со стеком, требуется время $O(n)$. Каждое ребро (дуга) графа (орграфа) помечается только один раз, и это требует времени порядка $O(1)$, поэтому трудоемкость пометки ребер (дуг) есть $O(m)$. Следовательно, трудоемкость поиска в глубину есть $O(n + m)$.

Замечание 4.1. Заметим, что множество всех ребер (дуг) графа (орграфа) *G*, которые были помечены как «древесные», в результате поиска в глубину из некоторой стартовой вершины *s* образуют ориентированное корневое дерево *T* графа (орграфа) *G* с корнем в вершине *s* (это дерево часто называют *деревом поиска в глубину* (depth-first-tree)). Если вершина *w* была помечена (занесена в стек) из вершины *v*, то в дереве поиска в глубину этому соответствует дуга (*v*, *w*).

Для графа, изображенного на рис. 4.3, был выполнен поиск в глубину, и на рис. 4.4 приведен *лес поиска в глубину* (два дерева поиска в глубину с корнями в вершинах 1 и 7).

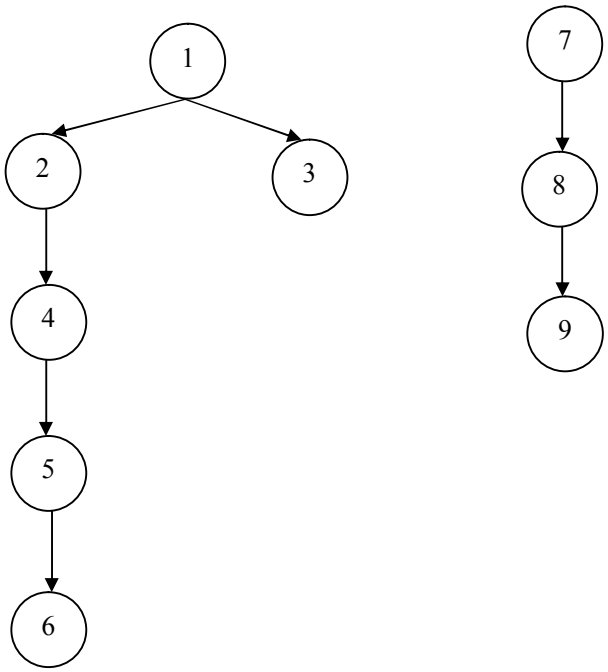


Рис. 4.4

Замечание 4.2. Поиск в глубину может быть использован для выделения связных компонент неориентированного графа. Для этого достаточно выполнить поиск в глубину, начиная с произвольной непомеченной вершины *s*. Множество помеченных во время поиска в глубину из вершины *s* вершин и множество «древесных» и «обратных» дуг, соединяющих данные вершины, образуют компоненту связности графа. Если после выполненного поиска в глубину имеются еще непомеченные вершины, то необходимо найти одну из них и, полагая ее в качестве стартовой вершины *s*, повторить для нее описанный выше процесс поиска в глубину (при этом будет получена новая компонента связности графа). Процесс выделения связных компонент графа заканчивается, когда все вершины будут помечены. Трудоемкость алгоритма выделения связных компонент неориентированного графа есть $O(|V|+|E|)$. Связные компоненты орграфа находятся так же, как в неориентированном графе, если рассматривать исходный орграф как неориентированный.

Замечание 4.3. Поиск в глубину может использоваться для выделения сильносвязных компонент орграфа. Для этого:

а) выполняем поиск в глубину в орграфе; во время поиска в глубину вершинам присваиваем две метки: первая метка присваивается вершине,

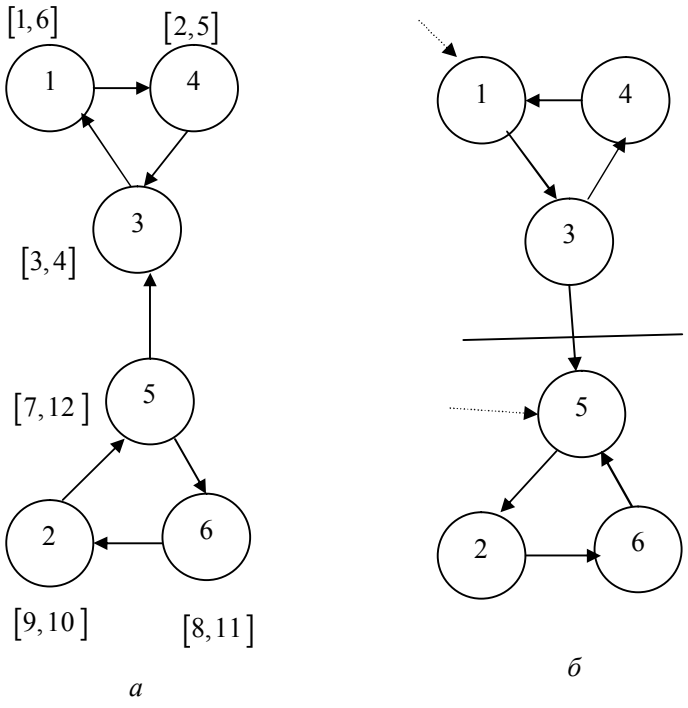
когда она заносится в стек, а вторая – когда она удаляется из стека (нумерация обеих меток общая);

б) транспонируем орграф (каждая дуга заменяется на противоположно направленную дугу), запоминаем все вторые метки вершин и считаем все вершины непомеченными;

в) пока существуют непомеченные вершины, поступаем следующим образом: выбираем непомеченную вершину *s* с максимальной второй меткой и запускаем поиск в глубину из нее (все вершины, которые будут помечены во время поиска в глубину из вершины *s* вместе с соединяющими их дугами, будут принадлежать той же сильносвязной компоненте, что и вершина *s*).

Пример 4.2. Для орграфа, изображенного на рис. 4.5, *а*, выделить сильносвязные компоненты.

Решение. Для решения поставленной задачи выполним сначала поиск в глубину в орграфе (на рис. 4.5, *а* возле каждой вершины в квадратных скобках указаны первая и вторая метки). Затем выполним переориентацию орграфа (рис. 4.5, *б*).



а *б*

Рис. 4.5

Наибольшую вторую метку (= 12) имеет вершина с номером 5. Запускаем поиск в глубину из вершины 5, при этом удается пометить верши-

ны 2 и 6. Таким образом, вершины 2, 5 и 6 вместе с соединяющими их дугами принадлежат одной сильносвязной компоненте орграфа.

Среди непомеченных вершин наибольшую вторую метку (= 6) имеет вершина 1. Запускаем поиск в глубину из вершины 1, при этом удается пометить вершины 3 и 4. Таким образом, вершины 1, 3 и 4 вместе с соединяющими их дугами принадлежат другой сильносвязной компоненте.

В орграфе все вершины помечены, следовательно, все сильносвязные компоненты орграфа выделены.

Замечание 4.4. Заметим, что когда поиск в глубину выполняется из вершины *s* и в стек заносится некоторая вершина *t*, то стек содержит последовательность вершин, определяющих путь (маршрут) из стартовой вершины *s* в вершину *t*.

Таким образом, если необходимо найти путь (маршрут) из вершины *s* в вершину *t*, то выполняем поиск в глубину из вершины *s*, который завершаем в одной из двух ситуаций:

г) на некотором шаге в стек занесена вершина *t*, тогда содержимое стека задает последовательность вершин пути (маршрута) из *s* в *t*;

- стек пуст, а вершина *t* осталась непомеченной, тогда не существует пути из *s* в *t*.

Замечание 4.5. Поиск в глубину может быть использован для определения двудольности графа. Пусть *v* – некоторая вершина, которая последней была занесена в стек во время поиска в глубину, и предположим, что она была отнесена к первой доле. Тогда все вершины, которые будут помечены из вершины *v*, будут отнесены ко второй (т. е. противоположной) доле. Граф будет двудольным, если все обратные ребра будут соединять вершины разных долей. Для определения двудольности орграфа нужно дуги заменить ребрами и применить описанный выше алгоритм.

Пример 4.3. Определить, является ли граф, изображенный на рис. 4.6, двудольным.

Решение. Выполним поиск в глубину в графе. Возле каждой вершины на рис. 4.6 указаны: в квадратных скобках – метка, задающая порядок посещения вершины, и через двоеточие – номер присвоенной доли. Каждое ребро помечено как «древесное» или «обратное».

Приведенный на рис. 4.6 граф двудольный, так как обратное ребро (1, 3) соединяет вершины разных долей.

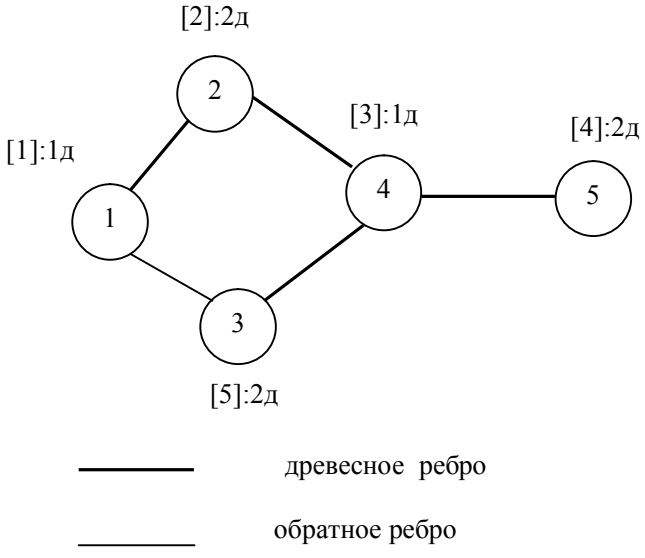


Рис. 4.6

Топологическая сортировка – такое линейное упорядочивание вершин орграфа, что для любой дуги $(v, w) \in U$ вершина v в этом порядке располагается до вершины w. Другими словами, топологическая сортировка – такое размещение вершин орграфа на одной горизонтальной линии, что все дуги $(v, w) \in U$ идут слева направо.

Замечание 4.6. Поиск в глубину может использоваться для топологической сортировки вершин орграфа $G = (V, U)$, который не содержит контуров. Во время поиска в глубину вершинам присваиваем две метки: первая метка присваивается вершине, когда она заносится в стек, а вторая – когда она удаляется из стека (нумерация обоих меток общая). Для получения топологически упорядоченного множества (ТУМ) располагаем вершины в порядке убывания вторых меток. Трудоемкость описанного алгоритма $O(|V| + |U|)$.

Пример 4.4. Для орграфа, изображенного на рис. 4.7, выполнить топологическую сортировку вершин (орграф не содержит контуров).

Решение. Выполним поиск в глубину в орграфе. Во время поиска в глубину каждой вершине поставим в соответствие две метки по правилам, описанным выше. В ТУМ вершины располагаются в порядке убывания вторых меток (рис. 4.7.).

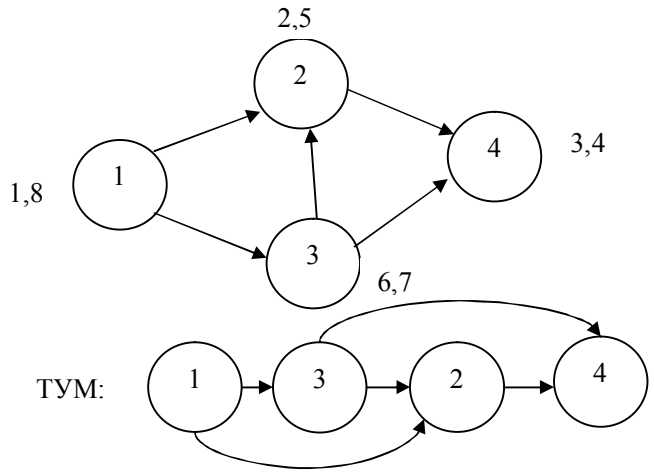


Рис. 4.7

Существует еще один достаточно простой алгоритм топологической сортировки вершин орграфа. В данном алгоритме на очередной итерации находят вершину без входящих дуг (если таких вершин несколько, то выбирают любую из них) и переносят ее в топологически упорядоченное множество. Затем удаляют эту вершину из орграфа вместе с выходящими из нее дугами и повторяют опять процесс поиска новой вершины без входящих в нее дуг. Для орграфа, который не содержит контуров, алгоритм завершает свою работу, когда все вершины перенесены в ТУМ. Если граф содержит контур, то на некотором шаге алгоритма окажется, что нет вершин со степенью полузахода, равной нулю, а при этом не все вершины орграфа перенесены в ТУМ (топологическая сортировка не может быть выполнена). Описанный алгоритм может быть реализован с трудоемкостью $O(|V|+|U|)$.

4.3. ПОИСК В ШИРИНУ

Еще одним методом обхода графа (орграфа) является поиск в ширину. В процессе поиска в ширину, выполненного из вершины *s*, всем вершинам *v*, которые достижимы из вершины *s*, присваиваются метки (минимальное расстояние от вершины *s* к вершине *v*). В начале работы алгоритма все вершины не имеют меток. По аналогии с поиском в глубину строится *дерево поиска в ширину* с корнем в вершине *s*, которое содержит все вершины графа (орграфа), достижимые из вершины *s*. Для программной реализации алгоритма поиска в ширину используется структура данных *очередь*. В процессе работы алгоритма каждая вершина включается и исключается из оче-

реди только один раз. Вершина включается в очередь после того, как ей присваивается метка, и исключается из очереди, когда в очередь заносятся все смежные с ней непомеченные вершины.

Алгоритм поиска в ширину
в графе (*орграфе*)

1. Присваиваем стартовой вершине s метку $key(s)=0$, заносим ее в очередь и считаем помеченной.
2. Пока все вершины не помечены или очередь не станет пустой, повторяем следующую последовательность действий: пусть первым элементом очереди является вершина v и $S(v)$ – множество непомеченных вершин, смежных с вершиной v, тогда:

а) каждую вершину $w \in S(v)$:

– помечаем меткой $key(w)=key(v)+1$;

– заносим w в очередь;

– запоминаем, что вершина w была помечена из вершины v (формирование дерева поиска в ширину);

б) вершину v считаем просмотренной, удаляем ее из очереди и возвращаемся к шагу 2 алгоритма.

Как следует из описания алгоритма, чем раньше помечается вершина, тем раньше она будет просмотрена. Трудоемкость поиска в ширину в графе есть $O(|V|+|E|)$.

Пример 4.5. Выполнить поиск в ширину для графа, который приведен на рис. 4.8.

Решение. В качестве стартовой вершины выбираем вершину с номером 1. В результате поиска в ширину каждая вершина характеризуется

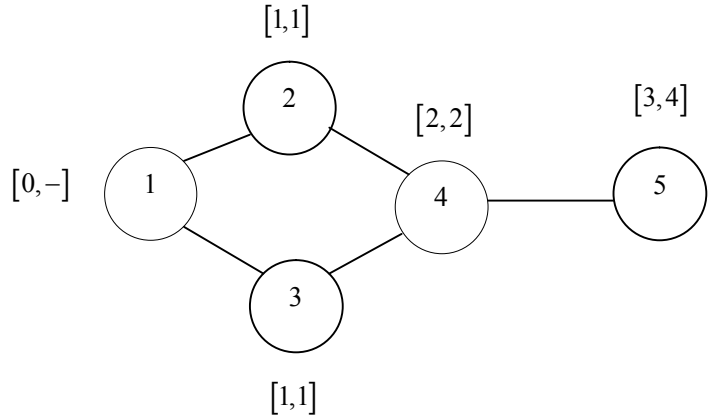


Рис. 4.8

следующим образом: [метка; номер вершины, из которой данная вершина получила метку].

Дерево поиска в ширину для графа, приведенного на рис. 4.8, изображено на рис.4.9. Данное дерево позволяет определить наименьший по количеству ребер маршрут из стартовой вершины поиска (вершина 1) в любую достижимую из нее вершину.

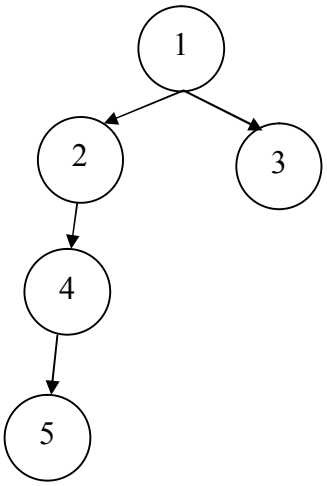


Рис. 4.9

Замечание 4.7. Поиск в ширину может быть использован для выделения компонент связности графа. Если граф $G=(V,E)$ связен, то в результате работы алгоритма поиска в ширину будут помечены все вершины графа. Если граф не связен, то будут помечены вершины только той компоненты связности, которой принадлежит стартовая вершина поиска в ширину. В этом случае для построения следующей компоненты связности достаточно взять любую непомеченную вершину и выполнить поиск в ширину из нее.

Замечание 4.8. Поиск в ширину можно использовать для определения двудольности связной компоненты графа. Если во время поиска в ширину отнести произвольную вершину связной компоненты графа к первому подмножеству (метка 1), то все смежные с ней вершины должны быть отнесены ко второму подмножеству (метка 2); смежные с ними – к первому подмножеству и т. д. Если граф не двудольный, то в какой-то момент поиска мы обнаружим, что вершина, помеченная 1 или 2, окажется смежной с вершиной той же метки.

Пример 4.6. Определить, является ли двудольным граф, изображенный на рис. 4.10.

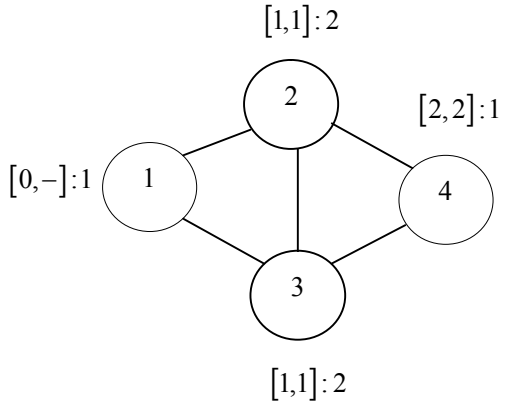


Рис. 4.10

Решение. Выполним поиск в ширину, начиная с вершины с номером 1. Во время поиска в ширину каждая вершина характеризуется следующим образом: [метка; номер вершины, из которой данная вершина получила метку], номер доли. Стартовую вершину 1 относим к 1-й доле. Вершины 2 и 3, которые были помечены из вершины 1, относим ко 2-й доле. Теперь вершины, смежные с вершиной 2, должны быть отнесены к 1-й доле. Одна из таких вершин – вершина 3, которая ранее была отнесена ко 2-й доле. Следовательно, возникает конфликт, так как две смежные вершины с номерами 2 и 3 помечены одной меткой доли – 2. Граф, изображенный на рис. 4.10, не является двудольным.

Замечание 4.9. Поиск в ширину может быть использован для поиска наименьшего (по числу дуг (ребер)) пути (маршрута) между заданной вершиной s и любой достижимой из нее вершиной. Метка, которой помечается на некотором шаге алгоритма поиска в ширину вершина v, и есть длина наименьшего пути (маршрута). Для восстановления самого пути (маршрута) используется дерево поиска в ширину. Если в результате поиска в ширину некоторая вершина w осталась непомеченной, то это свидетельствует о том, что пути (маршрута) из вершины s в вершину w не существует.

4.4. ПУТИ В ОРГРАФАХ. МАРШРУТЫ В ГРАФАХ

4.4.1. Кратчайший элементарный путь

Задан орграф $G=(V,U)$ ($|V|=n$ и $|U|=m$), каждой дуге $(v,w)\in U$ приписана некоторая стоимость $c(v, w)$. Требуется найти кратчайший элементарный путь из вершины s в вершину t.

Оказывается, что задача поиска кратчайшего элементарного пути в орграфе из вершины s в вершину t эквивалентна поиску кратчайших элементарных путей из вершины s во все достижимые из нее вершины.

Алгоритм Форда – Беллмана (Ford – Bellman)

Алгоритм Форда (1956) – Беллмана (1958) находит кратчайшие элементарные пути из некоторой вершины s во все достижимые из нее вершины. Трудоемкость алгоритма – $O(mn)$. Алгоритм находит кратчайшие пути при следующих ограничениях:

- веса дуг орграфа произвольны;
- в орграфе отсутствуют элементарные контуры отрицательной стоимости, достижимые из вершины s.

Если в графе отсутствуют контуры отрицательной стоимости, то после завершения работы алгоритма для каждой вершины v мы получаем две величины $D(v)$ и $P(v)$. Величина $D(v)$ – длина кратчайшего пути из вершины s в вершину v, который состоит из кратчайшего пути из вершины s в вершину $P(v)$ и дуги $(P(v),v)$. Если вершина v не достижима из вершины s, то $D(v)=+\infty$, а значение $P(v)$ неопределенно.

Алгоритм Форда – Беллмана

1. Первоначально для всех вершин v таких, что $(s,v)\in U$, полагаем $D(v)=c(s,v)$, для вершины s полагаем $D(s)=0$, а для всех оставшихся вершин – $D(v)=+\infty$.

2. Выполнить $(n-1)$ итерацию. На каждой итерации алгоритма для всех дуг $(u,v)\in U$ выполнить пересчет компонентов вектора D по следующему правилу (рис. 4.11):

if $D(v)>D(u)+c(u,v)$ then

$D(v):=D(u)+c(u,v)$ and $P(v):=u$.

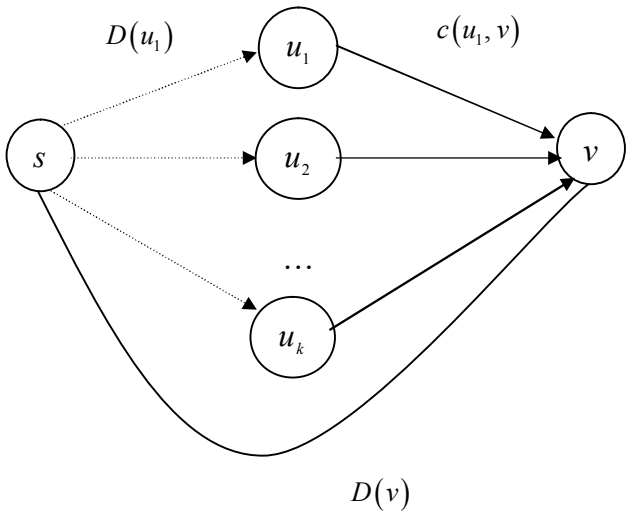


Рис. 4.11

3. Если для каждой дуги $(u, v) \in U$ выполняется неравенство $D(v) \le D(u) + c(u, v)$, то в орграфе отсутствуют контуры отрицательной стоимости и алгоритм нашел длины кратчайших путей из вершины *s* во все достижимые из нее вершины.

Следует заметить, что алгоритм может закончить свою работу раньше в том случае, если при выполнении очередной итерации ни один из компонентов вектора *D* не изменил своего значения.

К достоинствам алгоритма Форда – Беллмана следует отнести то, что он определяет, есть ли в орграфе контуры отрицательной стоимости, достижимые из вершины *s*. Если после завершения работы алгоритма существует дуга (u, v), для которой

$$D(v) > D(u) + c(u, v),$$

то тогда в последовательности

$$v, u, P(u), P(P(u)), \ldots$$

некоторая вершина встретится дважды (выделен контур отрицательной стоимости).

Пример 4.7. Для графа, изображенного на рис. 4.12, найти кратчайший элементарный путь из вершины $s = 1$ во все достижимые из нее вершины, используя алгоритм Форда – Беллмана.

Решение. Поскольку в орграфе, изображенном на рис. 4.12, количество вершин $|V| = 6$, то выполним пять итераций алгоритма. Результаты первых трех итераций приведены в табл 4.2. Нижний индекс у $D(v)$ – номер

вершины, из которой компонент v изменил свое значение (для восстановления самих кратчайших путей).

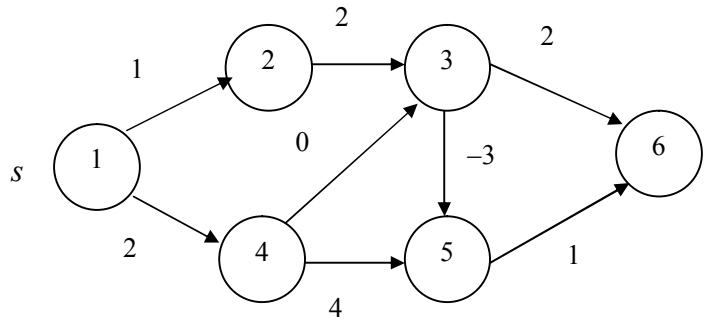


Рис. 4.12

Таблица 4.2

	$D(1)$	$D(2)$	$D(3)$	$D(4)$	$D(5)$	$D(6)$
$k=0$	0_-	1_1	$+\infty_1$	2_1	$+\infty_1$	$+\infty_1$
$k=1$	0_-	1_1	2_4	2_1	-1_3	0_5
$k=2$	0_-	1_1	2_4	2_1	-1_3	0_5

На итерации $k=2$ ни один из компонентов вектора D не изменил своего значения, следовательно, итерации алгоритма можно завершить досрочно. Поскольку для каждой дуги $(u,v)\in U$ выполняется неравенство $D(v)\le D(u)+c(u,v)$, то в орграфе отсутствуют контуры отрицательной стоимости, достижимые из вершины 1, и алгоритм Форда – Беллмана нашел длины кратчайших путей из вершины s во все достижимые из нее вершины.

Пример 4.8. Для графа, изображенного на рис. 4.13, найти кратчайший элементарный путь из вершины $s=1$ во все достижимые из нее вершины, используя алгоритм Форда – Беллмана.

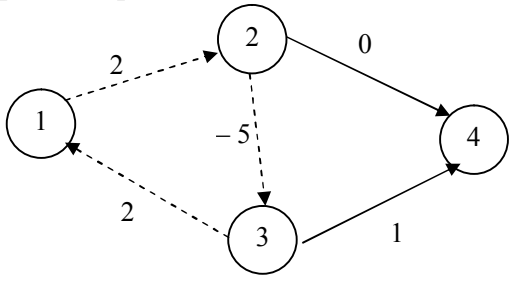


Рис. 4.13

Решение. Поскольку в орграфе, изображенном на рис. 4.13, количество вершин $|V| = 4$, то выполним три итерации алгоритма. Результаты итераций приведены в табл. 4.3 (нижний индекс у $D(v)$ – номер вершины, из которой компонент v изменил свое значение (для восстановления самих кратчайших путей)).

Таблица 4.3

	$D(1)$	$D(2)$	$D(3)$	$D(4)$
$k = 0$	0_-	2_1	$+\infty_1$	$+\infty_1$
Итер. $k = 1$	-1_3	2_1	-3_2	-2_3
Итер. $k = 2$	-2_3	1_1	-4_2	-3_3
Итер. $k = 3$	-3_3	0_1	-5_2	-4_3

Для каждой дуги проверим неравенство: $D(v) \le D(u) + c(u, v)$. Для дуги $(1, 2) \in U$ неравенство не выполняется:

$$D(2) > D(1) + c(1, 2) \quad (0 > -1).$$

Следовательно, орграф, приведенный на рис. 4.13, содержит контур отрицательной стоимости, достижимый из вершины 1 (на рис. 4.13 дуги контура отрицательной стоимости выделены), и задача построения кратчайших путей из вершины 1 не имеет решения.

Если нам необходимо найти элементарные кратчайшие пути из всех вершин графа во все оставшиеся, то необходимо запустить алгоритм Форда – Беллмана для каждой вершины (на каждой итерации происходит пересчет матрицы D размером $n \times n$: каждая строка матрицы – решение задачи для отдельной вершины v исходного графа). Трудоемкость этого алгоритма $O(mn^2)$.

Следует заметить, что если орграф бесконтурный, то для вычисления кратчайшего элементарного пути из некоторой вершины во все оставшиеся достижимые существует алгоритм трудоемкости $O(n + m)$, в котором каждая компонента вектора D вычисляется только один раз. Алгоритм использует тот факт, что для вершин бесконтурного орграфа можно выполнить топологическую сортировку. После того как выполнена топологическая сортировка вершин орграфа, вычисление компонентов вектора D происходит в порядке их следования в линейном порядке. Путь u – некоторая вершина в линейном порядке, тогда для всех дуг $(u, v) \in U$ вычисляем значение компонента v вектора D по правилу: $D(v) = \min(D(v),\ D(u) + c(u, v))$.

Алгоритм Дейкстры (Shortest-Path Algorithm due to Dijkstra(SPD))

Многие практические задачи сводятся к задаче нахождения кратчайшего элементарного пути в орграфе с неотрицательными стоимостями дуг. Для этого используется алгоритм Дейкстры (1959), который находит кратчайшие пути из вершины s во все достижимые из нее вершины в предположении, что веса дуг орграфа неотрицательны.

Предположим, что все множество вершин исходного графа разбито на два подмножества S и $V \setminus S$. Множество S – это множество вершин, до которых кратчайший элементарный путь из источника s уже известен. Первоначально $S = \{s\}$. На очередном шаге алгоритм Дейкстры находит вершину v, которая принадлежит множеству $V \setminus S$ и расстояние до которой от источника s наименьшее среди всех вершин множества $V \setminus S$. После чего найденная вершина добавляется во множество S. Алгоритм Дейкстры заканчивает свою работу, когда $|S| = n$.

Общая схема алгоритма Дейкстры

1. $S = \{s\}$;
   $\forall v : (s,v) \in U : \; D(v) := c(s,v)$;
   $\forall v : (s,v) \notin U : \; D(v) := +\infty$.
2. Пока $|S| \neq n$, выполнять:

а) $w := \arg\min_{v \in V \setminus S} D(v)$;

б) $S := S \cup \{w\}$;

в) $d(s,w) = D(w)$;

г) $\forall v \in V \setminus S, (w,v) \in U : D(v) := \min\{D(v), D(w) + c(w,v)\}$.

Заметим, что на каждой итерации для каждой вершины $v \in V$ значение $D(v)$ соответствует длине наименьшего на данный момент элементарного пути из стартовой вершины s в данную вершину.

Пример 4.9. Задан орграф, каждой дуге (v,w) которого поставлена в соответствие некоторая неотрицательная стоимость $c(v,w)$(рис. 4.14). Требуется найти кратчайшие элементарные пути из вершины $s = 1$ во все достижимые из нее вершины, используя алгоритм Дейкстры (реализованный на массиве).

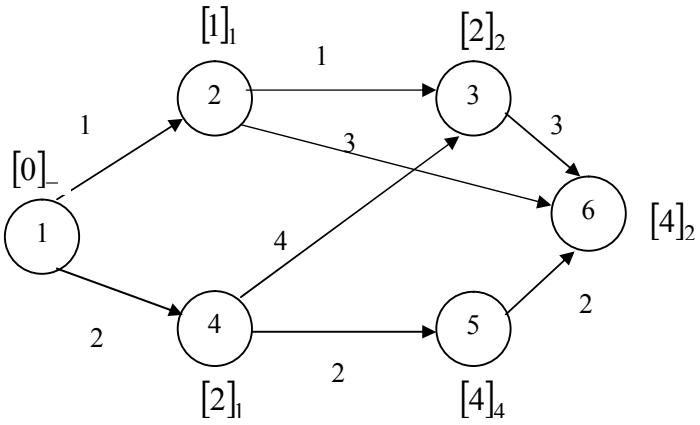


Рис. 4.14

Решение. Приведем последовательность итераций алгоритма Дейкстры. Для каждой итерации укажем:

- множество S, которое включает в себя те вершины орграфа, для которых кратчайшие элементарные пути из s уже построены;
- для $w \in S$: $d(s,w) = D(w)$;
- для $v \in V \setminus S$ находим значение $D(v)$, которое соответствует длине наименьшего элементарного пути из стартовой вершины s в данную вершину;
- значение $path(v)$ – номер вершины, из которой вершина $v \in V$ получила свое значение $D(v)$.

0-я итерация: $S = \{1\}$, $d(1,1) = 0$.

v	2	3	4	5	6
$D(v)$	**1**	+∞	2	+∞	+∞
Path(v)	1	–	1	–	–

1-я итерация: $S = \{1,2\}$, $d(1,2) = 1$.

v	3	4	5	6
$D(v)$	**2**	2	+∞	4
Path(v)	2	1	–	2

2-я итерация: $S = \{1,2,3\}$, $d(1,3) = 2$.

v	4	5	6
$D(v)$	**2**	+∞	4
Path(v)	1	–	2

3-я итерация: $S = \{1, 2, 3, 4\}$, $d(1, 4) = 2$.

v	5	6
$D(v)$	**4**	4
Path(v)	4	2

4-я итерация: $S = \{1, 2, 3, 4, 5\}$, $d(1, 5) = 4$.

v	6
$D(v)$	**4**
Path(v)	2

5-я итерация: $S = \{1, 2, 3, 4, 5, 6\}$, $d(1, 6) = 4$.

На рис. 4.14, возле каждой вершины v в квадратных скобках указана длина кратчайшего элементарного пути из стартовой вершины $s = 1$ в данную вершину (эту длину мы будем в дальнейшем называть потенциалом), а в качестве нижнего индекса у потенциала указан номер вершины, из которой вершина v получила свой потенциал.

Для оценки трудоемкости алгоритма заметим, что количество итераций ограничено количеством вершин орграфа ($|V| = n$) (на каждой итерации новая вершина v добавляется к множеству S). Учитывая, что поиск минимума в массиве D из n элементов есть $O(n)$, а общее количество операций, необходимых для пересчета компонентов вектора D, ограничено числом дуг ($|U| = m$), получаем, что трудоемкость приведенной реализации алгоритма Дейкстры есть $O(n^2 + m) = O(n^2)$.

Применяя алгоритм Дейкстры для каждой вершины орграфа, можно найти кратчайшие элементарные пути из всех вершин орграфа во все достижимые (трудоемкость – $O(n^3)$).

Заметим, что наиболее трудоемкой операцией в алгоритме Дейкстры являлся поиск минимального элемента в n-элементном массиве. Для выполнения операций поиска и удаления минимального элемента наиболее эффективная структура данных – это *приоритетная очередь* (куча). Напомним, что если приоритетную очередь реализовать в виде полного бинарного дерева (бинарная куча), то трудоемкость операций поиска минимального элемента – $O(1)$, добавление элемента и удаление минимального элемента – $O(\log n)$, где n – количество элементов в куче.

Реализация алгоритма Дейкстры
с использованием структуры данных
приоритетная очередь

1. Добавляем в кучу стартовую вершину s с приоритетом $p(s)=0$. Все вершины орграфа считаем непросмотренными.

2. Пока куча не станет пустой, выполняем следующую последовательность шагов:

а) из кучи удаляется вершина v (с минимальным приоритетом $p(v)$);

б) если вершина v ранее удалялась из кучи, то она игнорируется, и мы возвращаемся к шагу 2 алгоритма;

в) вершина v считается просмотренной;

г) для вершины v находятся непросмотренные вершины w: $(v,w)\in U$ (эти вершины уже могут быть в куче) и добавляются в кучу с приоритетом $p(w)=p(v)+c(v,w)$.

Замечание 4.10. Элементом кучи может быть, например, следующая тройка чисел:

- номер вершины;
- приоритет (это поле является ключевым);
- предшествующий элемент (т. е. та вершина, из которой данная вершина получила свой приоритет).

Замечание 4.11. Одна вершина может несколько раз добавляться в кучу. Приоритет, с которым вершина v заносится в кучу – длина некоторого элементарного пути из стартовой вершины s в вершину v. Вершина становится просмотренной при ее первом удалении из кучи. Каждой вершине v ставим в соответствие потенциал, равный тому приоритету, который был у вершины v при ее первом удалении из кучи (потенциал вершины v – длина кратчайшего элементарного пути из стартовой вершины s в данную вершину). В случае когда вершина v удаляется из кучи повторно, она игнорируется (так как вершина v уже ранее удалялась из кучи, то для нее известен кратчайший элементарный путь из s, а тот факт, что вершина v находилась в куче с приоритетом $p(v)$, говорит о том, что в вершину v из стартовой вершины s существует еще один элементарный путь, но его длина $p(v)$ не меньше, чем длина кратчайшего пути). Количество элементов в куче ограничено числом ребер.

Замечание 4.12. Восстановить кратчайший путь из вершины s в вершину v можно одним из следующих двух способов:

- двигаться из вершины *v* в вершину *s* по обратным дугам, для которых стоимость дуги (u, w) равна разности потенциалов вершины *u* и вершины *w*;
- двигаться из вершины *v* в вершину *s*, используя метку «предшествующий элемент» (*pred*).

Трудоемкость реализации алгоритма Дейкстры с использованием структуры данных *приоритетная очередь* (*бинарная куча*) – $O(m \log n)$.

Если реализовать алгоритм Дейкстры, используя кучу Фибоначчи (сначала в кучу заносятся все вершины, а на итерациях алгоритма выполняются операции удаления минимального элемента и уменьшения ключа), то получим трудоемкость $O(m + n \log n)$ (все операции уменьшения ключа – $O(m)$; операция создания кучи Фибоначчи из *n* элементов и все операции удаления минимального элемента – $O(n \log n)$).

Пример 4.10. Задан орграф, каждой дуге (v, w) которого поставлена в соответствие некоторая неотрицательная стоимость $c(v, w)$ (рис. 4.15). Необходимо найти кратчайшие элементарные пути из вершины $s = 1$ во все достижимые из нее вершины, используя алгоритм Дейкстры (реализованный с помощью приоритетной очереди).

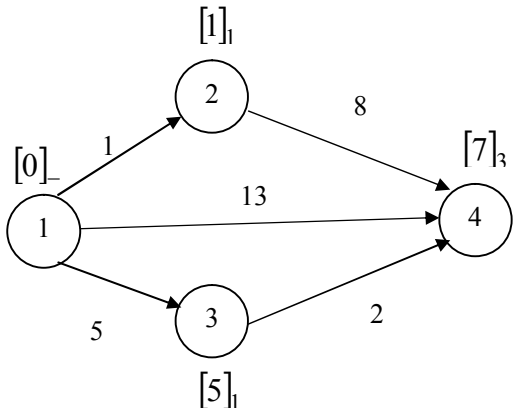


Рис. 4.15

Решение. Приведем содержимое структуры данных *куча* на каждой итерации алгоритма:

1) начальное состояние кучи:
(*n_ver* = 1, *prior* = 0, *pred* = 0);

2) вершина 1 (*prior* = 0) удаляется из кучи и полагается просмотренной; $ptn[1] = prior[1] = 0$;

состояние кучи:

(*n_ver* = 2, *prior* = 1, *pred* = 1);
(*n_ver* = 3, *prior* = 5, *pred* = 1);
(*n_ver* = 4, *prior* = 13, *pred* = 1);

3) вершина 2 (*prior* = 1) удаляется из кучи и полагается просмотренной; $ptn[2] = prior[2] = 1$;

состояние кучи:

(*n_ver* = 3, *prior* = 5, *pred* = 1);
(*n_ver* = 4, *prior* = 13, *pred* = 1);
(*n_ver* = 4, *prior* = 9, *pred* = 2);

4) вершина 3 (*prior* = 5) удаляется из кучи и полагается просмотренной; $ptn[3] = prior[3] = 5$;

состояние кучи:

(*n_ver* = 4, *prior* = 13, *pred* = 1);
(*n_ver* = 4, *prior* = 9, *pred* = 2);
(*n_ver* = 4, *prior* = 7, *pred* = 3);

5) вершина 4 (*prior* = 7) удаляется из кучи и полагается просмотренной; $ptn[4] = prior[4] = 7$;

состояние кучи:

(*n_ver* = 4, *prior* = 13, *pred* = 1);
(*n_ver* = 4, *prior* = 9, *pred* = 2);

6) вершина 4 (*prior* = 9) удаляется из кучи, но так как она уже просмотрена, то игнорируем ее;

состояние кучи:

(*n_ver* = 4, *prior* = 13, *pred* = 1);

7) вершина 4 (*prior* = 13) удаляется из кучи, но так как она уже просмотрена, то игнорируем ее;

состояние кучи: куча пуста.

После того как куча становится пустой, все вершины, достижимые из стартовой вершины $s = 1$, имеют потенциал – длина кратчайшего элементарного пути из стартовой вершины s в данную вершину (на рис. 4.15 потенциал указан возле вершины в квадратных скобках). Для восстановления самого пути можно, например, одновременно с потенциалом хранить номер той вершины, из которой данная вершина получила свой потенциал (на рис. 4.15 – нижний индекс у метки потенциала).

Алгоритм Флойда – Варшалла (Floyd – Warshall)

В случае когда стоимости дуг орграфа произвольны (предполагается, что контуры отрицательной стоимости отсутствуют), для поиска кратчайших элементарных путей между всеми парами вершин можно использовать алгоритм Форда – Беллмана, который имел бы в этом случае трудоемкость $O\left(mn^2\right)$. Существует алгоритм Флойда– Варшалла, который решает эту задачу с трудоемкостью $O\left(n^3\right)$.

Если стоимости дуг орграфа неотрицательны, то такую же трудоемкость $O\left(n^3\right)$, как и у алгоритма Флойда – Варшалла, имеет алгоритм Дейкстры. Реализация алгоритма Дейкстры с использованием структуры данных *приоритетная очередь* (*бинарная куча*) имеет трудоемкость $O(nm\log n)$, и если предположить, что количество дуг m орграфа намного меньше чем n^2, то этот алгоритм предпочтительнее, чем алгоритм Флойда – Варшалла.

В алгоритме Флойда – Варшалла происходит пересчет матрицы элементарных путей $A_k\,(0 \le k \le n)$ между всеми парами вершин. После k-й итерации элемент матрицы $A_k[i, j]$ содержит длину кратчайшего элементарного пути из вершины i в вершину j, который проходит только по вершинам с номерами меньше либо равными k (сами значения i и j могут быть как больше, так и меньше k). Количество итераций алгоритма равно количеству вершин орграфа (n). Матрица $A_n[i, j]$ дает конечный ответ: $A_n[i, j]$ – длина кратчайшего пути между вершинами i и j в орграфе, а $\left(P[i, j], j\right)$ – финальная дуга такого пути.

Первоначально значения элементов матрицы A определяются следующим образом:

- $A_0[i, j] = c(i, j)$, если $i \ne j$ и существует дуга $(i, j) \in U$;
- $A_0[i, j] = +\infty$, если $i \ne j$ и не существует дуги $(i, j) \in U$;
- $A_0[i, i] = 0,\ \forall i \in V$;
- $P[i, j] = i, \forall i, j \in V$.

Пересчет элементов матрицы A ($k \ge 1$) осуществляется по следующей формуле:

$$A_k[i, j] = \min\left\{A_{k-1}[i, j], A_{k-1}[i, k] + A_{k-1}[k, j]\right\}.$$

Программная реализация пересчета элементов матрицы A может быть выполнена следующим образом:

```
for k := 1 to n do
  for i := 1 to n do
    for j := 1 to n do
      if A[i,k] + A[k,j] < A[i,j] then
          A[i,j] := A[i,k] + A[k,j] and P[i,j] := P[k,j];
```

Пример 4.11. Задан взвешенный орграф (рис. 4.16). Необходимо найти кратчайшие элементарные пути между всеми парами вершин, используя алгоритм Флойда – Варшалла.

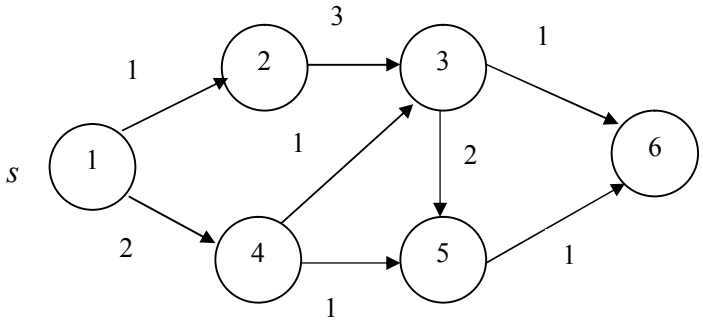


Рис. 4.16

Решение. Приведем состояние матрицы A после каждой итерации алгоритма (нижний индекс у элементов $A[i,j]$ – значения $P[i,j]$).

0-я итерация, 1- я итерация:

	1	2	3	4	5	6
1	0_1	1_1	$+\infty$	2_1	$+\infty$	$+\infty$
2	$+\infty$	0_2	3_2	$+\infty$	$+\infty$	$+\infty$
3	$+\infty$	$+\infty$	0_3	$+\infty$	2_3	1_3
4	$+\infty$	$+\infty$	1_4	0_4	1_4	$+\infty$
5	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_5	1_5
6	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_6

2-я итерация:

	1	2	3	4	5	6
1	0_1	1_1	4_2	2_1	$+\infty$	$+\infty$
2	$+\infty$	0_2	3_2	$+\infty$	$+\infty$	$+\infty$
3	$+\infty$	$+\infty$	0_3	$+\infty$	2_3	1_3
4	$+\infty$	$+\infty$	1_4	0_4	1_4	$+\infty$
5	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_5	1_5
6	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_6

3-я итерация:

	1	2	3	4	5	6
1	0_1	1_1	4_2	2_1	6_3	5_3
2	$+\infty$	0_2	3_2	$+\infty$	5_3	4_3
3	$+\infty$	$+\infty$	0_3	$+\infty$	2_3	1_3
4	$+\infty$	$+\infty$	1_4	0_4	1_4	2_3
5	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_5	1_5
6	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_6

4-я итерация:

	1	2	3	4	5	6
1	0_1	1_1	3_4	2_1	3_4	4_3
2	$+\infty$	0_2	3_2	$+\infty$	5_3	4_3
3	$+\infty$	$+\infty$	0_3	$+\infty$	2_3	1_3
4	$+\infty$	$+\infty$	1_4	0_4	1_4	2_3
5	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_5	1_5
6	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0_6

После 4-й итерации матрица кратчайших путей A не изменяется. Постоенная матрица A задает кратчайшие элементарные пути между всеми парами вершин для орграфа, изображенного на рис. 4.16 (используя масив P, можно восстановить последовательности вершин кратчайших путей).

4.4.2. Кратчайшие пути (маршруты)

Заметим, что не всегда кратчайший путь (маршрут) является элементарным, в чем можно убедиться, рассмотрев далее примеры.

Задача 1. Имеется N городов, которые пронумерованы числами от 1 до N, где N – натуральное число. Некоторые из них соединены двусторонними дорогами, пересекающимися только в городах. Имеется два типа дорог – шоссейные и железные. Для каждой дороги известна базовая стоимость проезда по ней. Стоимость проезда зависит от набора дорог, по которым мы проезжаем, и от способа проезда. Так, если вы подъехали к городу C по шоссейной (железной) дороге $X \to C$ и хотите ехать дальше по дороге $C \to Y$ того же типа, то вы должны уплатить только базовую стоимость проезда по дороге $C \to Y$. Если тип дороги $C \to Y$ отличен от типа дороги $X \to C$, то вы должны уплатить базовую стоимость проезда по дороге $C \to Y$ плюс 10 % от базовой стоимости проезда по этой дороге (страховой взнос). При выезде из города A страховой взнос платится всегда. Необходимо определить самый дешевый маршрут проезда

из города A в город B в виде последовательности городов, а также вычислить стоимость проезда по этому маршруту.

Задача 2. Даны декартовы координаты N перекрестков города, которые пронумерованы числами от 1 до N. На каждом перекрестке имеется светофор. Некоторые из перекрестков соединены двусторонними дорогами с правосторонним движением, пересекающимися только на перекрестках. Для каждой дороги известно время для проезда по ней от одного перекрестка до другого. Время проезда зависит от набора дорог, по которым вы едете, и от времени ожидания на перекрестках. Так, если вы подъехали от перекрестка X к перекрестку C по дороге $X \to C$ и хотите ехать дальше по дороге $C \to Y$, то время ожидания на перекрестке C зависит от того, поворачиваете вы налево или нет. Если вы поворачиваете налево, то время ожидания будет равно $D \cdot K$, где D – количество дорог, пересекающихся на перекрестке C, а K – некоторая константа. Если вы не поворачиваете налево, то время ожидания равно нулю. Необходимо проехать от перекрестка с номером A до перекрестка с номером B за минимальное время.

Задача 3. Имеется N городов, соединенных двусторонними дорогами с правосторонним движением. Для каждой дороги задана ее протяженность. Машина может поворачивать (изменять направление движения) только в городах. Машина имеет бак вместимостью z литров бензина, и для нее задан расход бензина x литров на один километр. В некоторых городах имеются заправочные станции. Для каждой заправочной станции определена своя цена за один литр бензина. Машина сможет заправиться только в том случае, если ее бак заполнен менее чем наполовину. Необходимо определить самый дешевый маршрут из города A в город B.

Задача 4. Задан взвешенный орграф. Необходимо найти второй по стоимости маршрут из вершины s в вершину t.

Рассмотрим далее каждую задачу отдельно. В первой задаче будут существовать случаи, когда мы проезжаем через некоторый город два раза (т. е. маршрут не является элементарным). Действительно, если подъехать к городу A и выехать из него на поезде по железной дороге, где-то дальше в городе B можно пересесть на автобус, двигающийся на шоссейной дороге и по ней приехать к городу A. Теперь выезжаем из него, но уже по шоссейной дороге. Легко понять, что существуют такие стоимости проезда по дорогам, когда такой маршрут будет иметь наименьшую стоимость (просто стоимость смены дорог в городе может быть очень велика: больше, чем проезд от города B и обратно).

Заметим, что в первой задаче в оптимальном маршруте через каждый город можно проехать не более двух раз. Для данной задачи это условие можно заменить следующим: по оптимальному маршруту по каждой дороге можно проехать не более одного раза, т. е. оптимальный маршрут является простым.

Подобная ситуация возникает и в задаче 2, когда через некоторый перекресток можно проехать более одного раза. Другой особенностью задачи 2 является тот факт, что ребро графа (дорогу) лучше рассматривать как две дуги. В этом случае по каждой дуге орграфа в оптимальном маршруте не надо ехать более одного раза. Это связано с тем, что если к перекрестку подъехать с каждой из возможных сторон по разу наилучшим образом, то этим можно и ограничиться (ничего нового и лучшего мы не найдем). Таким образом, в первой и второй задачах может быть реализован поиск оптимального маршрута с использованием структуры данных *приоритетная очередь* (*куча*). Поскольку заранее известно, что по каждой дуге (ребру) орграфа мы будем проходить не более одного раза, то в качестве элемента, добавляемого в кучу, можно брать дугу (ребро) орграфа. Тогда дуга (ребро) будет считаться просмотренной при ее первом удалении из кучи, и мы никогда не пройдем дважды по одной и той же дуге (ребру).

Оптимальные маршруты первых двух задач были простыми. Однако существует много задач, в которых требуемый маршрут (путь) проходит по некоторым ребрам (дугам) графа более одного раза. В третьей задаче может оказаться, что по некоторой дороге придется ездить более одного раза, потому что некоторые маршруты не могут быть реализованы в силу недостаточности бензина. Поэтому для каждого пункта необходимо иметь информацию не только о дороге, по которой мы в него приехали, но и о количестве бензина, имеющегося на данный момент в баке машины. Значит, имеет смысл рассматривать только такие маршруты, в которых при повторном проезде по дороге мы приезжаем в конечный пункт дороги с большим количеством бензина. Таким образом, количество проездов по одной дороге ограничено объемом бака машины z. Поэтому если был реализован некоторый маршрут с количеством бензина x литров в баке для данной дороги, то на следующей итерации при использовании кучи имеет смысл рассматривать только те варианты, когда заполнение бака будет больше, чем величина x. Это не касается городов, где можно заправляться. Для таких городов иногда более важно иметь меньше бензина, чтобы осуществить заправку.

В четвертой задаче второй по длине путь также может не являться простым путем.

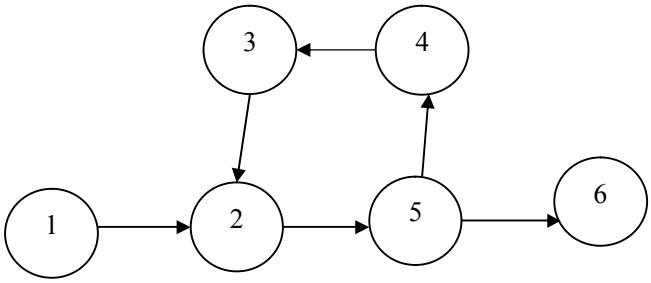


Рис. 4.17

Так для орграфа, приведенного на рис. 4.17, второй по длине путь из вершины 1 в вершину 6 задается следующей последовательностью вершин:

$$1 \to 2 \to 5 \to 4 \to 3 \to 2 \to 5 \to 6.$$

Указанный путь не является простым, так как проходит по дуге (2, 5) более одного раза.

Для решения четвертой задачи можно модифицировать алгоритм Дейкстры с использование бинарной кучи. В алгоритме в кучу заносятся вершины. Вершина v считается просмотренной, когда она первый раз удаляется из кучи (приоритет, с которым она удаляется из кучи, – длина кратчайшего элементарного пути из стартовой вершины в вершину v). Повторное удаление вершины v из кучи требовало игнорировать ее. На самом деле, тот приоритет, с которым вершина v второй раз удаляется из кучи, есть не что иное, как значение второго по длине пути из стартовой вершины в вершину v. Поэтому если нам надо найти второй по длине путь, то будем считать вершину v просмотренной только после ее второго удаления из кучи. Обобщая, для нахождения k-го по длине кратчайшего маршрута (пути) в графе (орграфе) будем считать вершину просмотренной только после ее k-го удаления из кучи.

4.4.3. Максимальный путь в бесконтурном орграфе

Рассмотрим следующую задачу сетевого планирования. Задана последовательность работ и порядок их исполнения: некоторой работе может предшествовать выполнение других работ. Для каждой работы задана длительность ее выполнения. Проект начинается в момент времени 0 и

заканчивается, когда все работы завершены. Требуется найти минимальное время завершения проекта.

Математической моделью приведенной выше задачи может являться связный орграф $G=(V,U)$ без контуров. Вершинам орграфа соответствуют события, дугам (v,w) – работы с длительностью $c(v,w)>0$. Событие, соответствующее вершине *i*, – возможность начала выполнения работ, соответствующих дугам, выходящим из вершины *i.* Минимальное время завершения проекта равно длине максимального пути в орграфе (длиной пути в орграфе называется суммарная длина дуг этого пути).

Для проекта, заданного последовательностью работ с порядком предшествования и длительностью работ, указанными в табл. 4.4, соответствующий орграф приведен на рис. 4.18.

Таблица 4.4

№ работы	Непосредственно предшествующие работы	Длительность работы	Соответствующая работе дуга орграфа
а	-	2	(1, 2)
б	-	10	(1, 3)
в	*а*	3	(2, 4)
г	*б*	4	(3, 4)
д	*в, г*	8	(4, 5)
е	*д*	8	(5, 7)
ж	*в, г*	10	(4, 6)
з	*ж*	60	(6, 7)

На рис. 4.16 возле каждой дуги орграфа указаны соответствующий дуге номер работы и длительность работы (в скобках).

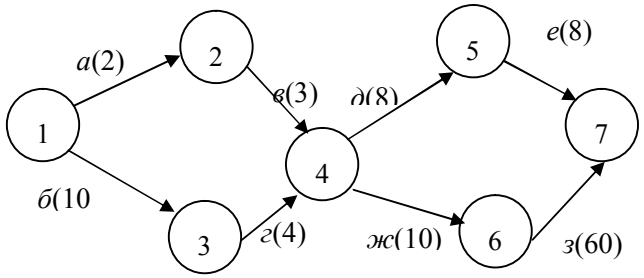


Рис. 4.18

Длина максимального пути для орграфа, изображенного на рис. 4.18, равна 84.

Для решения задачи нахождения максимального пути в бесконтурном орграфе будем использовать следующий алгоритм.

Алгоритм
нахождения максимального пути
в бесконтурном орграфе

1. Поскольку орграф не содержит контуров, то выполним топологическую сортировку его вершин.

2. К каждой вершине *i* орграфа $G=(V,U)$ поставим метку $M[i]$, которая равна самому раннему сроку, когда можно начать событие, соответствующее вершине *i*, потому что все предшествующие работы уже завершены. Первоначально метки всех вершин равны нулю.

3. Вычисление меток вершин происходит в порядке их нумерации (в топологическом порядке). Пусть *j* – некоторая вершина, тогда полагаем

$$M[j]=\max_{(i,j)\in U}\{M[j],\quad M[i]+c(i,j)\}.$$

4. Длина максимального взвешенного пути равна наибольшему значению $M[i]$.

Пример 4.12. Для орграфа, изображенного на рис. 4.19, найти длину максимального взвешенного пути.

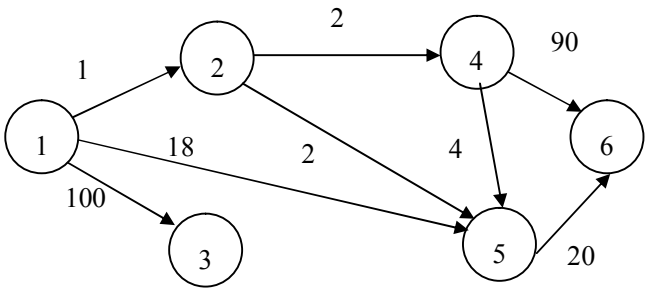


Рис. 4.19

Решение. Выполним топологическую сортировку вершин орграфа, приведенного на рис. 4.19. Топологический порядок вершин: 1, 3, 2, 4, 5, 6. Расставим метки *M* вершинам орграфа.

i	1	3	2	4	5	6
$M[i]$	0	100	1	3	18	93

На рис. 4.20 возле каждой вершины в квадратных скобках указана соответствующая вершине метка.

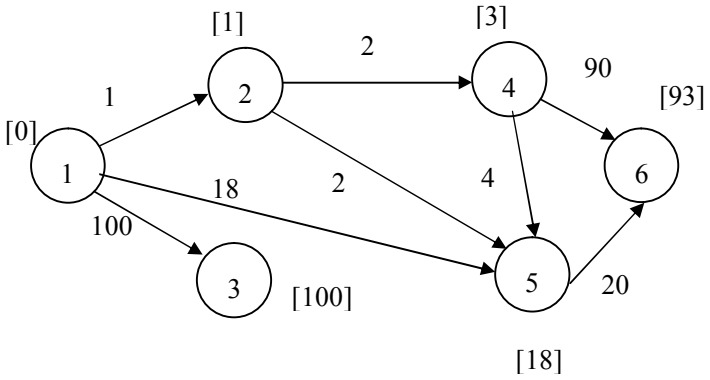


Рис. 4.20

Максимальная метка равна 100 (метка вершины 3). Следовательно, длина максимального взвешенного пути орграфа, изображенного на рис. 4.20, равна 100.

4.4.4. Эйлеров цикл

Ранее были рассмотрены маршруты, проходящие не более одного раза по каждому ребру, и маршруты, которые могли проходить по некоторым ребрам графа более одного раза. Сейчас рассмотрим задачу нахождения эйлерова цикла в графе, т. е. такого замкнутого маршрута, который проходит по каждому ребру графа ровно один раз.

Задача существования цикла, проходящего по каждому ребру графа ровно один раз, возникла при решении задачи о кенигсбергских мостах, которая формулировалась следующим образом. В Калининграде (ранее г. Кенигсберге) на реке Преголя было два острова, которые были соединены между собой и с берегами реки семью мостами (рис. 4.21).

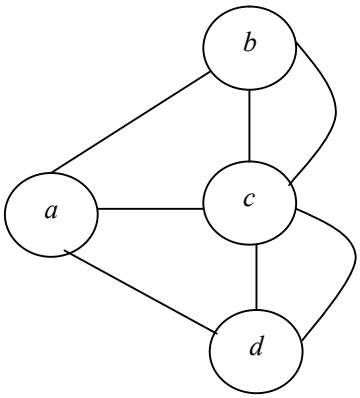


Рис. 4.21

Можно ли, двигаясь с одного из четырех участков суши (на рис. 4.21 этим участкам соответствуют вершины графа *a*, *b*, *c*, *d*), только один раз пройти по каждому мосту

(мостам в графе соответствуют ребра) и снова вернуться в тот участок суши, с которого начиналось движение?

Шведский математик Эйлер сформулировал эту задачу с точки зрения теории графов и доказал теорему о существовании в графе замкнутого цикла, проходящего по каждому ребру графа ровно один раз. С тех пор такие циклы получили название эйлеровых, а графы, содержащие эйлеровы циклы, стали называть *эйлеровыми графами*.

Заметим, что если граф является эйлеровым, то, двигаясь вдоль эйлерова цикла, можно, не отрывая руки, нарисовать весь граф.

ТЕОРЕМА 4.1. (***Эйлер, 1736***) Связный граф является эйлеровым тогда и только тогда, когда степени всех его вершин четны.

Напомним, что степень вершины v равна количеству инцидентных ей ребер. В дальнейшем степень вершины v будем обозначать через $d(v)$.

Для орграфа теорема Эйлера может быть сформулирована следующим образом: орграф содержит эйлеров контур тогда и только тогда, когда для каждой вершины орграфа количество входящих дуг равно количеству выходящих дуг и орграф сильносвязный.

В случае когда нам необходимо построить *эйлеров маршрут* для связного графа из вершины v в вершину w, необходимо, чтобы:

- степени вершин v и w были нечетны;
- степени всех оставшихся вершин – четны.

Аналогично для слабосвязного орграфа существует *эйлеров путь* из вершины v в вершину w, если:

- количество дуг, входящих в вершину v на единицу меньше, чем количество выходящих из нее дуг;
- количество дуг, входящих в вершину w на единицу больше, чем количество выходящих из нее дуг;
- для всех оставшихся вершин количество входящих дуг равно количеству выходящих дуг.

Алгоритм построения эйлерова цикла для графа

Алгоритм работает в предположении, что граф $G = (V, E)$ связный, а степени всех его вершин четны.

1. Эйлеров цикл $C = \{Ø\}$.
2. Пусть v_0 – некоторая произвольная вершина графа. Будем двигаться от вершины v_0 по неиспользованным ребрам графа до тех пор, пока не вернемся опять в эту вершину (построили цикл). Предположим, что построен цикл C'.

3. Если все ребра графа использованы, то полагаем $C = C'$, и алгоритм заканчивает свою работу.

4. Если существуют неиспользованные ребра, то в силу связности графа должна существовать такая вершина графа v_i, которая принадлежит циклу C и является концом какого-то еще неиспользованного ребра.

5. Удалим из исходного графа G ребра цикла C. В оставшемся графе все вершины по-прежнему будут иметь четную степень (при удалении ребер степени каждой вершины цикла уменьшаются на четное число). В полученном графе построим цикл C', начиная движение от вершины v_i.

6. Если все ребра использованы, то эйлеров цикл построен. Требуемый эйлеров цикл – это часть цикла C от вершины v_0 до вершины v_i, затем цикл C', а затем оставшаяся часть цикла C от вершины v_i до вершины v_0.

7. Если все ребра не использованы, то полагаем $C = C \bigcup C'$ и возвращаемся к шагу 4 алгоритма.

Для программной реализации алгоритма построения эйлерова цикла в неориентированном графе можно использовать такие структуры данных, как *стек* и *очередь*. Тогда последовательность шагов алгоритма будет выглядеть следующим образом.

Реализация алгоритма
построения эйлерова цикла

1. Пусть v_0 – произвольная вершина графа G. Занесем ее в стек. Очередь полагаем пустой.

2. Пусть v – последняя занесенная в стек вершина. Тогда, пока стек не опустеет, выполнять следующую последовательность шагов:

а) если существует ребро (v, w), то удаляем это ребро из графа, а вершину w заносим в стек;

б) если не существует ребра, выходящего из вершины v, то удаляем вершину v из стека и добавляем ее в очередь.

3. Последовательность вершин очереди и задает порядок обхода вершин графа в эйлеровом цикле.

Пример 4.13. Для графа $G = (V, E)$, приведенного на рис. 4.22, построить эйлеров цикл.

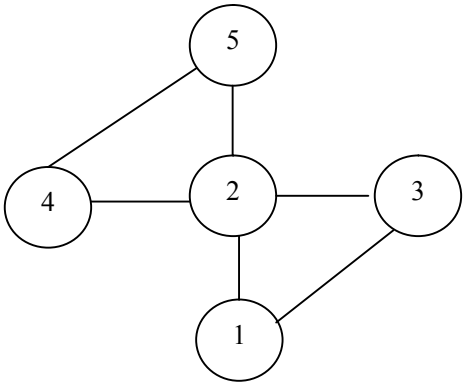


Рис. 4.22

Решение. Сначала делаем очередь пустой, а в стек заносим вершину 1:

$$S = \{1\},$$
$$O = \{\varnothing\}.$$

Затем последовательно двигаемся по ребрам

$$(1, 2), (2, 3), (3, 1),$$

удаляем их из исходного графа и заносим в стек вершины 2, 3 и 1. После выполнения данной последовательности действий содержимое стека будет равно

$$S = \{1, 2, 3, 1\}.$$

После чего мы видим, что не существует ребер, выходящих из вершины 1, поэтому удаляем ее из стека и переносим в очередь. Аналогично поступаем и с вершиной 3.

$$S = \{1, 2\},$$
$$O = \{1, 3\}.$$

Затем последовательно двигаемся по ребрам

$$(2, 4), (4, 5), (5, 2),$$

удаляем их, а содержимое стека будет иметь вид

$$S = \{1, 2, 4, 5, 2\}.$$

После чего в графе вообще нет ребер, поэтому удаляем последовательно все элементы из стека и переносим их в очередь:

$$O = \{1, 3, 2, 5, 4, 2, 1\}.$$

Последовательность вершин очереди и задает порядок обхода вершин в эйлеровом цикле.

Упражнение 4.1. Модифицировать приведенный алгоритм для построения эйлерова контура ориентированного графа.

Упражнение 4.2. Задана последовательность слов. Игра заключается в том, что игроки по очереди называют слова из заданной последовательности. Правило, по которому называется слово, заключается в следующем: если названо какое-то слово, то следующий игрок может назвать слово, начинающееся с последней буквы предыдущего слова и которое еще не было названо. Необходимо определить, можно ли выстроить цепочку из всех слов, причем последнее слово должно заканчиваться на ту букву, с которой начиналось первое слово. Так, например, для последовательности слов: Караганда, Воронеж, Киев, Жданов, Витебск, Архангельск требуемая цепочка слов имеет вид: Киев → Воронеж → Жданов → Витебск → Караганда → Архангельск.

Упражнение 4.3. На столе выложены пластинки домино. Каждая состоит из двух частей, и на них изображено некоторое число точек. Пластинки домино могут выкладываться в ряд одна за другой по следующему правилу: количества точек на примыкающих частях соседних в ряду пластинок должны совпадать. Необходимо определить, можно ли выстроить цепочку, в которой каждая пластинка домино встречается ровно один раз, причем количества точек на свободных концах пластинок, которые являются крайними в цепочке, должны совпадать.

4.4.5. Наибольшее число маршрутов между заданной парой вершин, не пересекающихся по ребрам

Задан граф $G = (V, E)$, в котором выделены две вершины *v* и *w*. Требуется найти наибольшее число маршрутов, соединяющих вершины *v* и *w*, которые не пересекаются по ребрам. Для решения данной задачи будем использовать следующий алгоритм.

Алгоритм
построения наибольшего числа маршрутов между
заданной парой вершин, не пересекающихся по ребрам

1. Выполнить поиск в ширину или глубину для поиска маршрута из вершины *v* в вершину *w*. Ребра найденного маршрута ориентировать дугами по направлению пути из *v* в *w*: если маршрут задается последовательностью вершин $v, v_1, v_2, \ldots, v_k, w$, то мы получим дуги $(v, v_1), (v_1, v_2), \ldots, (v_k, w)$. Полагаем очередь пустой.
2. Заносим в очередь вершину *v*. Пока очередь не будет пуста или в нее не будет занесена конечная вершина *w*, выполняем расширенный поиск в ширину. Расширенный поиск в ширину ведется по ребрам и входящим дугам. Пусть *i* – вершина, которая удалена из очереди на текущей итерации расширенного поиска в ширину, тогда
   - если (i, j) – ребро, то вершину *j* заносим в очередь;
   - если (i, j) – дуга, то ничего не делаем;
   - если (j, i) – дуга, то вершину *j* заносим в очередь.
3. Если конечная вершина *w* занесена в очередь, то восстанавливаем путь из вершины *v* в вершину *w*. Дуги этого пути заменяем ребрами, а ребра – дугами. Убираем пометки вершин и возвращаемся к шагу 2 алгоритма.
4. Если очередь пуста, а конечная вершина *w* не была занесена в очередь, то алгоритм завершает свою работу. Количество дуг, выходящих из

вершины v, соответствует наибольшему числу не пересекающихся по ребрам маршрутов из вершины v в вершину w. Для восстановления не пересекающихся по ребрам маршрутов из вершины v в вершину w будем выполнять поиск в глубину по дугам. Для этого заносим в стек вершину v и выполняем поиск в глубину по дугам до тех пор, пока в стек не будет занесена конечная вершина w: один из маршрутов найден. Запоминаем последовательность вершин найденного маршрута, заменяем дуги маршрута ребрами и повторяем поиск в глубину из вершины v. Если на некотором этапе поиска в глубину не удается пометить конечную вершину w, то процесс восстановления наибольшего количества реберно-непересекающихся маршрутов между заданной парой вершин в орграфе завершен.

Пример 4.14. Для графа $G = (V, E)$, приведенного на рис 4.23, найти наибольшее число реберно-непересекающихся маршрутов между вершинами 1 и 9.

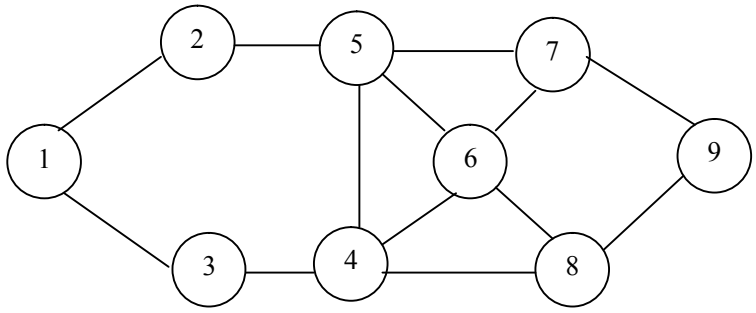


Рис. 4.23

Решение. Используя поиск в глубину или ширину, построим произвольный маршрут между вершинами 1 и 9. Последовательность вершин этого маршрута: 1, 3, 4, 8, 6, 5, 7, 9. На рис. 4.24 ребра найденного пути ориентированы дугами.

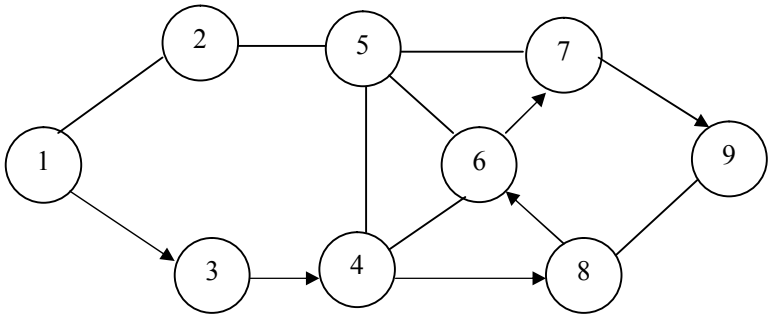


Рис. 4.24

Выполним расширенный поиск из вершины 1 в ширину по ребрам и входящим дугам. На рис. 4.25 возле каждой вершины указаны пометки вершин: в квадратных скобках – метка вершины, нижний индекс – номер вершины, из которой данная вершина получила метку.

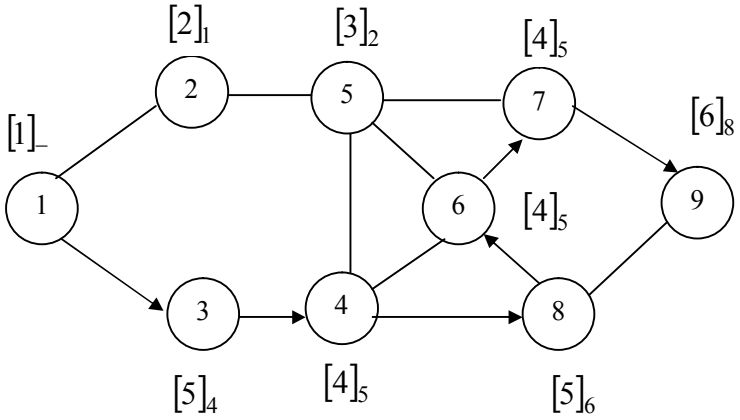


Рис. 4.25

Увеличивающий путь: 1, 2, 5, 6, 8, 9. Заменяем ребра этого пути дугами, а дуги – ребрами (рис. 4.26).

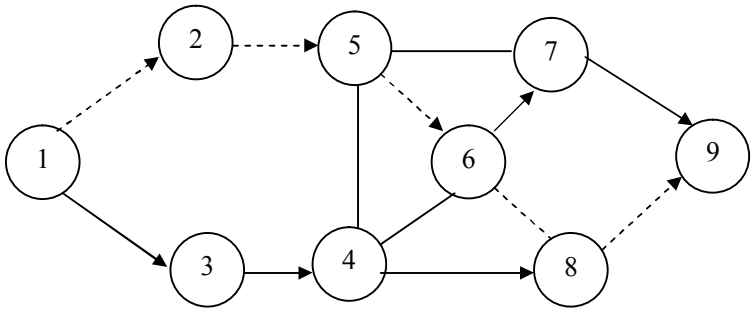


Рис. 4.26

Запускаем снова из вершины 1 расширенный поиск в ширину. Однако нам не удается пометить вершину 9, следовательно, наибольшее число реберно-непересекающихся маршрутов между вершинами 1 и 9 найдено (количество маршрутов равно 2). Для восстановления самих маршрутов запускаем из вершины 1 два раза поиск в глубину по дугам, имеем следующие два маршрута: 1, 2, 5, 6, 7, 9 и 1, 3, 4, 8, 9 (рис. 4.26).

4.4.6. *k* кратчайших (v, w) маршрутов, не пересекающихся по ребрам

Задан граф $G=(V, E)$, каждому ребру $(v, w)\in E$ которого приписана некоторая неотрицательная стоимость $c(v, w)\geq 0$. Стоимость *k* маршрутов определим как сумму стоимостей всех ребер, появляющихся во всех *k* маршрутах. Задача построения *k* кратчайших (v, w) маршрутов (*k* Shortest-Paths Problem) заключается в нахождении в графе *k* маршрутов минимальной стоимости между заданной парой вершин *v* и *w*. В данном разделе будет рассмотрена задача построения *k*-кратчайших (v, w) маршрутов, взаимно не пересекающихся по ребрам (shortest set of *k* mutually edge-disjoint paths).

Алгоритм построения *k* кратчайших (v, w) маршрутов, которые не пересекаются по ребрам, использует модифицированный алгоритм Дейкстры (Modified Dijkstra's Shortest-Path Algorithm SPDM). Данный алгоритм находит кратчайший путь в орграфе *G* из вершины *s* во все достижимые из нее вершины и работает в предположении, что:

- стоимости дуг орграфа *G* произвольны;
- в орграфе *G* отсутствуют контуры отрицательной стоимости.

В данном алгоритме на каждой итерации для вершины $v\in V$ значение *D*(*v*) соответствует длине наименьшего на данный момент элементарного пути из стартовой вершины *s* в данную вершину. После завершения работы алгоритма значение $d(s, v)$ – длина кратчайшего пути из вершины *s* в вершину *v*.

Алгоритм

(Modified Dijkstra's Shortest-Path Algorithm SPDM)

1. $S=\{s\}$;

   $\forall v:(s,v)\in U:\ D(v):=c(s, v)$;

   $\forall v:(s,v)\notin U:\ D(v):=+\infty$.

2. Пока $|S|\neq n$ выполнять:

$w:=\arg\min_{v\in V\setminus S} D(v)$;

$S:=S\cup\{w\}$;

$d(s, w)=D(w)$;

$\forall v \in V, (w, v) \in U$ если $D(v) > D(w) + c(w, v)$,

то $D(v) := D(w) + c(w, v)$ и $S := S \setminus \{v\}$.

Заметим, что если на итерации алгоритма для некоторой вершины v происходит корректировка метки $D(v)$, то данная вершина удаляется из множества S (возвращается во множество вершин, для которых кратчайшие пути из стартовой вершины s еще не найдены).

Алгоритм
построения k-кратчайших (v, w) маршрутов,
не пересекающихся по ребрам

1. Полагаем $p = 1$. Находим кратчайший маршрут из вершины v в вершину w любым известным алгоритмом, например, алгоритмом Дейкстры: $v, v_1, v_2, \ldots, v_k, w$, где $(v_i, v_{i+1}) \in E, i = 1, \ldots, k-1$.

Обозначим через L_p множество ребер кратчайшего маршрута:

$$L_p = \{(v, v_1), (v_1, v_2), \ldots, (v_k, w)\},$$

а через $c(L_p)$ – его стоимость:

$$c(L_p) = \sum_{(x, y) \in L_p} c(x, y).$$

2. Строим модифицированный орграф G' из исходного графа G, заменяя в графе G ребра из множества L_p дугами: $(v_1, v), (v_2, v_1), \ldots, (w, v_k)$. Стоимости ребер орграфа G' полагаем такими же, как и в графе G, а стоимости дуг – отрицательными:

$$c(v_1, v) = -c(v, v_1), c(v_2, v_1) = -c(v_1, v_2), \ldots, c(w, v_k) = -c(v_k, w).$$

3. Поскольку орграф G' не содержит контуров отрицательной стоимости, то, используя, например, модифицированный алгоритм Дейкстры (SPDM), находим в орграфе G' кратчайший путь из вершины v в вершину w. Множество ребер (дуг) найденного пути обозначим L. Заменим дуги пути L ребрами.

4. Два множества L_p и L могут содержать общие элементы (ребра). Удалим общие элементы из множеств L_p и L.

5. Полагаем $p = p + 1$, $L_p = L_{p-1} \cup L$, $c(L_p) = c(L_{p-1}) + c(L)$.

6. Если $p = k$, то алгоритм завершает свою работу. Элементы множества L_p соответствуют ребрам p-кратчайших (v, w) маршрутов, которые

не пересекаются по ребрам. В противном случае возвращаемся к шагу 2 алгоритма.

Для восстановления самих k-кратчайших (v, w) маршрутов можно выполнить k раз поиск в глубину из вершины v в вершину w по ребрам из L_p (после того как некоторый маршрут из вершины v в вершину w найден, блокируем ребра этого маршрута).

Пример 4.15. Для взвешенного графа G, изображенного на рис. 4.27, найти 2 кратчайших маршрута из вершины A в вершину Z, которые не пересекаются по ребрам.

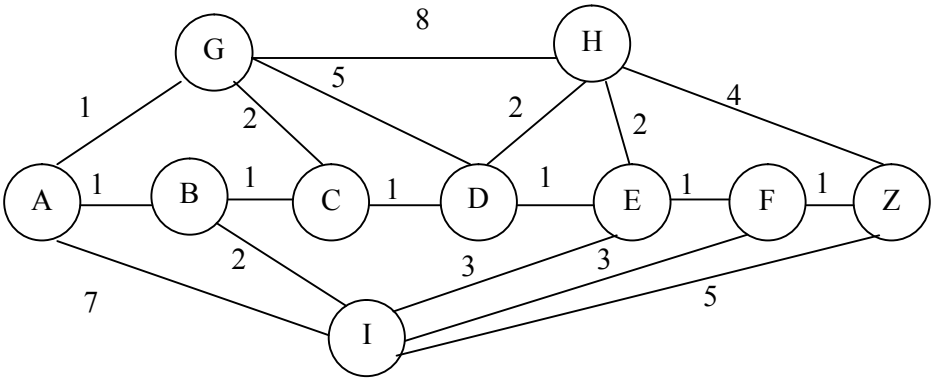


Рис. 4.27

Решение. Построим кратчайший маршрут из вершины A в вершину Z, используя, например, алгоритм Дейкстры. Последовательность ребер маршрута L_1 равна (A, B), (B, C), (C, D), (D, E), (E, F), (F, Z) (на рис. 4.28 ребра множества L_1 выделены). Стоимость $c(L_1) = 6$.

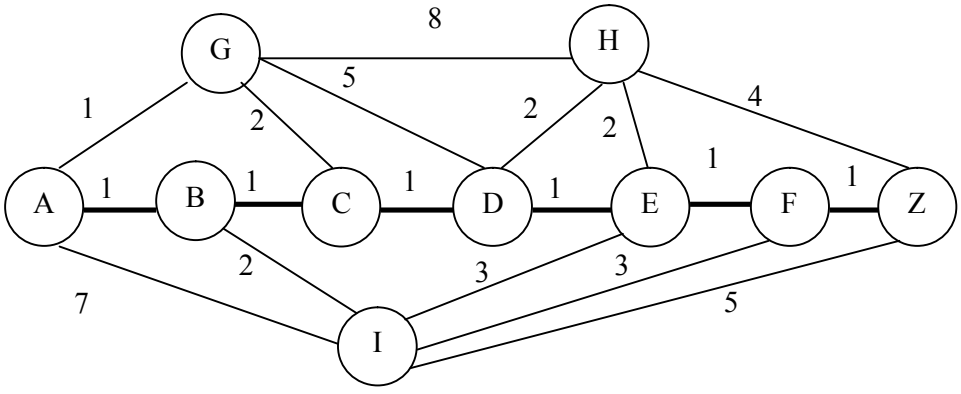


Рис. 4.28

Построим из графа G модифицированный орграф G', заменяя в графе G ребра из множества L_1 дугами с отрицательной стоимостью. Получим модифицированный орграф G', который приведен на рис. 4.29.

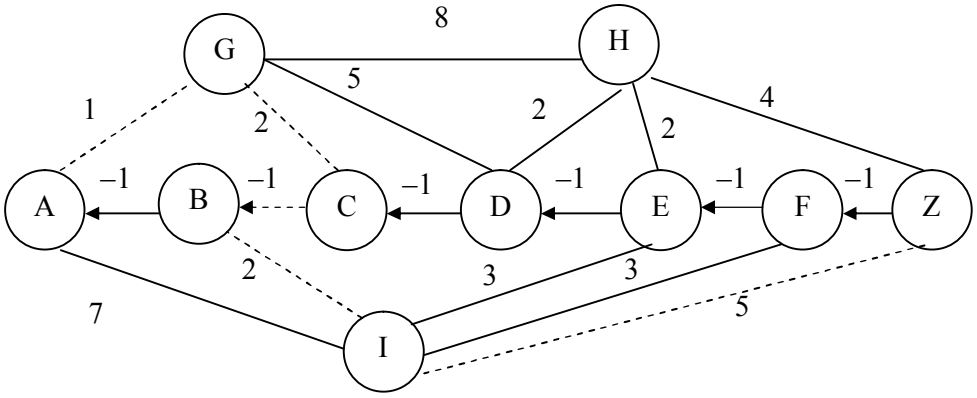


Рис. 4.29

Находим кратчайший путь из вершины A в вершину Z в орграфе G', используя модифицированный алгоритм Дейкстры (SPDM). Последовательность ребер (дуг) пути L: (A, G), (G, C), (C, B), (B, I), (I, Z); $c(L)=9$ (на рис. 4.29 ребра и дуги пути выделены). Заменим дугу (C, B) пути L ребром.

На рис. 4.30 ребра из множества L выделены пунктирными линиями, а из множества L_1 – жирными линиями.

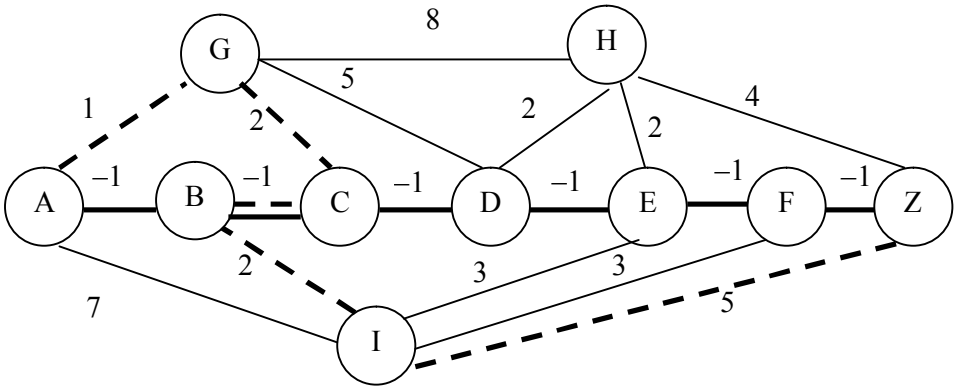


Рис. 4.30

И множество L_1, и множество L имеют общее ребро (B, C), которое удалим из этих множеств, а оставшиеся элементы множеств L_1 и L объ-

единим во множество L_2: (*A*, *B*), (*C*, *D*), (*D*, *E*), (*E*, *F*), (*F*, *Z*), (*A*, *G*), (*G*, *C*), (*B*, *I*), (*I*, *Z*) (на рис. 4.31 элементы из множества L_2 выделены).

Ребра из множества L_2 соответствуют двум кратчайшим маршрутам между вершинами *A* и *Z*, которые не пересекаются по ребрам ($c(L_2) = c(L_1) + c(L) = 15$).

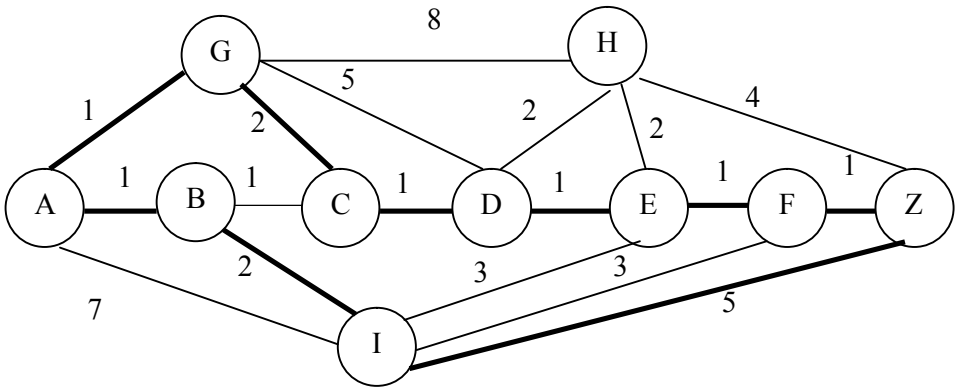


Рис. 4.31

Запуская два раза поиск в глубину по ребрам из множества L_2, построим два кратчайших маршрута из *A* в *Z*, которые не пересекаются по ребрам: (*A*, *G*, *C*, *D*, *E*, *F*, *Z*) и (*A*, *B*, *I*, *Z*).

4.5. МАКСИМАЛЬНЫЙ ПОТОК В СЕТИ И ЕГО ПРИЛОЖЕНИЯ

Предположим, что задан орграф $G = (V, U)$. Каждой дуге $(v, w) \in U$ орграфа *G* приписана некоторая пропускная способность $c(v, w)$. Выделены две вершины: *s* (в которую нет входящих дуг) и *t* (из которой нет выходящих дуг). Таким образом, мы имеем некоторую (s, t)-сеть.

Определение 4.1. *Потоком в (s, t)-сети* называется такая функция $f : U \to R$, которая удовлетворяет следующим свойствам:

1. Поток по дуге не превосходит пропускной способности дуги :

$$0 \le f(v, w) \le c(v, w).$$

2. Количество продукта, выходящего из вершины *s*, равно количеству продукта, входящего в вершину *t*:

$$f(v_1, t) + f(v_2, t) + \ldots + f(v_q, t) = f(s, u_1) + f(s, u_2) + \ldots + f(s, u_p).$$

3. Для каждой промежуточной вершины w (отличной от s и t) количество входящего продукта равно количеству выходящего продукта:

$$f(v_1, w) + f(v_2, w) + \ldots + f(v_q, w) = f(w, u_1) + f(w, u_2) + \ldots + f(w, u_p).$$

Величиной потока называется количество продукта, выходящего из вершины s. Поток, величина которого максимальна, *называется максимальным потоком*.

Последовательность вершин

$$s = v_1, v_2, \ldots, v_k = t$$

такая, что (v_i, v_{i+1}) – дуга либо (v_{i+1}, v_i) – дуга, называется альтернирующим путем. Дуги, направление которых совпадает с направлением пути из s в t, будем называть *прямыми дугами*, а оставшиеся – *обратными*.

На рис. 4.32 все дуги пути: v_1, v_2, v_3, v_4, v_5, кроме дуги (v_4, v_3), прямые.

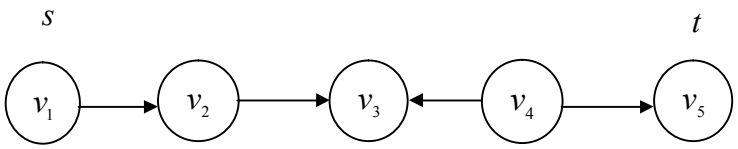


Рис. 4.32

Альтернирующий путь из s в t будем называть *увеличивающим*, если прямые дуги недогружены ($f(v, w) < c(v, w)$), а поток по обратным дугам больше нуля. Поток будет максимальным, если в сети отсутствует увеличивающий путь.

Простейший метод Форда–Фалкерсона построения максимального потока является итеративным и состоит в увеличении текущего потока (первоначально поток по всем дугам полагается равным нулю) вдоль увеличивающего пути. Для построения увеличивающего пути может использоваться, например, расширенный вариант поиска в ширину или глубину (по прямым недогруженным дугам и обратным дугам, поток по которым больше нуля). В процессе поиска увеличивающего пути вершинам присваиваются метки по определенному правилу; когда невозможно пометить конечную вершину t, максимальный поток найден.

Пометки вершин

Вершине *s* ставится в соответствие метка $met[s]$, равная максимальному количеству продукта среди дуг, которые выходят из вершины *s* (либо метка, равная $+\infty$).

Предположим, что вершине *v* поставлена в соответствие метка $met[v]$. Рассмотрим соседние с *v* непомеченные вершины. Возможна одна из следующих ситуаций.

- Если (v, w_1) – дуга и $f(v, w_1) < c(v, w_1)$, то дуга (v, w_1) не догружена на величину $c(v, w_1) - f(v, w_1)$ (данная величина называется остаточной пропускной способностью дуги). Тогда вершине w_1 ставится метка $met[w_1] := \min(|met[v]|, c(v, w_1) - f(v, w_1))$, соответствующая количеству продукта, которое может быть доставлено в вершину w_1 (с учетом того, что в вершину *v* доставлено $|met[v]|$ продукта).
- Если (v, w_1) – дуга и $f(v, w_1) = c(v, w_1)$ (дуга полностью загружена), то вершина w_1 из вершины *v* не помечается.
- Если (w_2, v) – обратная дуга и $f(w_2, v) > 0$, то вершине w_2 ставится метка $met[w_2] := -\min(|met[v]|, f(w_2, v))$. Величина $|met[w_2]|$ соответствует количеству продукта, которое может быть доставлено в вершину w_2 (выталкиванием продукта по обратной дуге (w_2, v)).

Если метка вершины *v* положительная, то эта вершина была помечена по прямой дуге, а если отрицательная – по обратной дуге.

Если в результате пометок удается пометить вершину *t*, то существует увеличивающий путь, и текущий в сети поток можно увеличить на величину $met[t]$. Увеличение потока осуществляется следующим образом. По каждой прямой дуге увеличивающего пути текущий поток увеличивается на величину $met[t]$; по каждой обратной дуге увеличивающего пути текущий поток уменьшается на величину $met[t]$.

Алгоритм
построения максимального потока

1. В качестве начального потока взять нулевой.
2. Используя расширенный поиск в ширину, расставить метки вершинам, начиная с вершины *s*, пока не будет помечена конечная вершина *t*. Если вершину *t* пометить не удается, то не существует увеличивающе-

го пути, текущий поток является максимальным, и алгоритм заканчивает работу.

3. Восстановить увеличивающий путь из s в t, используя дерево поиска в ширину.

4. Увеличить поток из s в t на величину $met[t]$ вдоль дуг увеличивающего пути.

5. Убрать все метки, приписанные вершинам, и вернуться к шагу 1 алгоритма.

Трудоемкость алгоритма построения максимального потока равна $O(f^* \cdot m)$, где f^* – значение величины максимального потока (трудоемкость расширенного поиска в ширину равна $O(n+m)=O(m)$, а за одну итерацию алгоритма величина потока увеличивается как минимум на 1).

Пример 4.16. Задана (s,t)-сеть (рис. 4.31). Требуется найти максимальный поток в (s,t)-сети ($s=1$; $t=6$).

Решение. На рис. 4.33 для каждой дуги (v,w) (s,t)-сети указаны пропускная способность дуги $c(v,w)$ и текущий поток $f(v,w)$ по этой дуге. Первоначально текущий поток по каждой дуге нулевой.

На рис. 4.33 выполнена первая итерация алгоритма. На каждой итерации вершине v ставятся в соответствие две величины (на рисунке эти величины пишутся над вершиной в квадратных скобках):

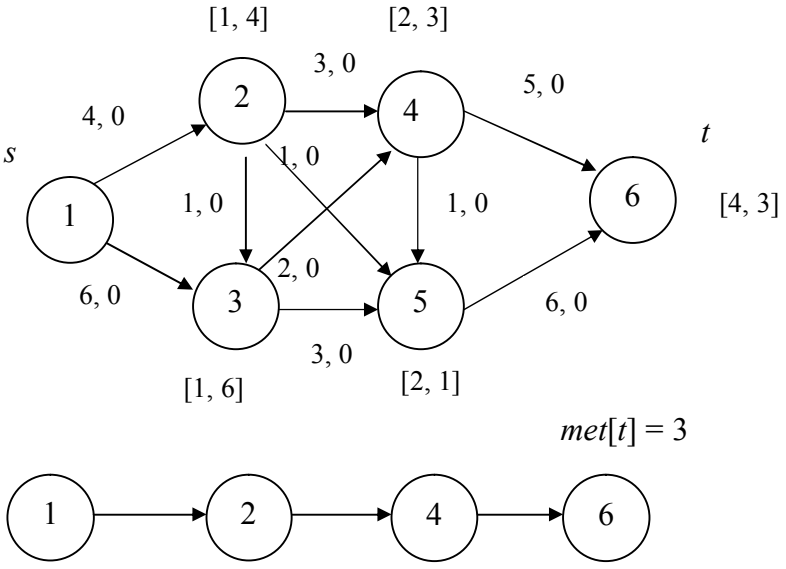


Рис. 4.33

- номер вершины, из которой данная вершина была помечена;
- метка *met*[*v*], определяемая по правилу, описанному выше.

Поскольку текущий поток еще не является максимальным, то продолжим выполнение итераций алгоритма нахождения максимального потока в сети. На рис. 4.34 проиллюстрирована 2-я и на рис. 4.35 – 3-я итерации алгоритма нахождения максимального потока.

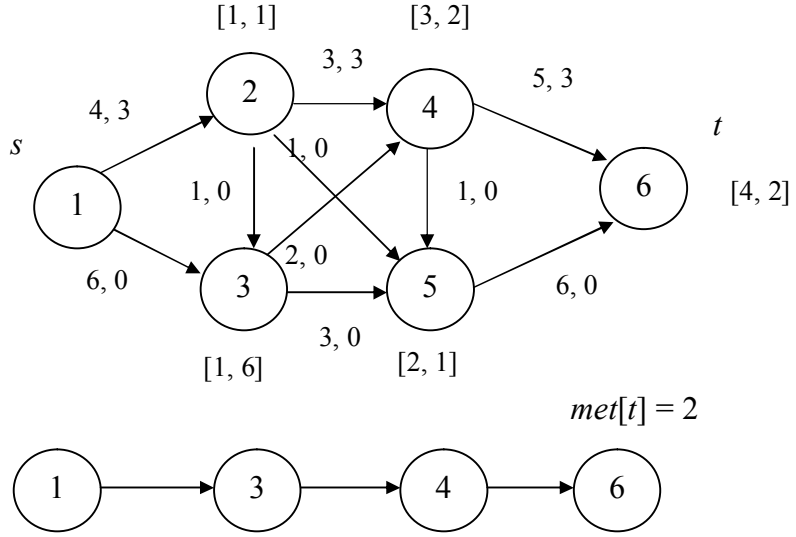
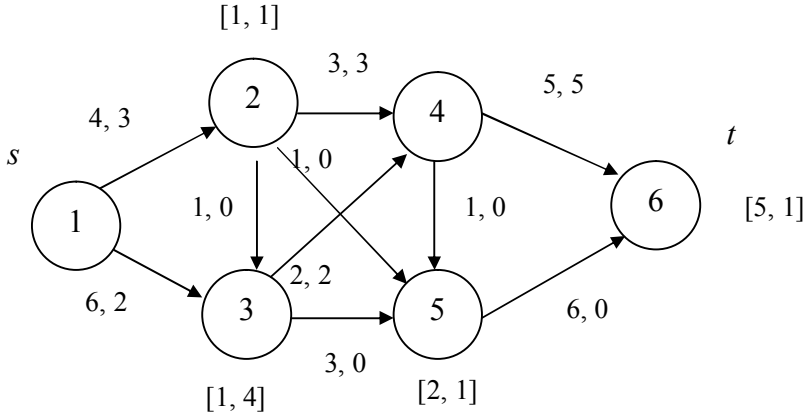


Рис. 4.34



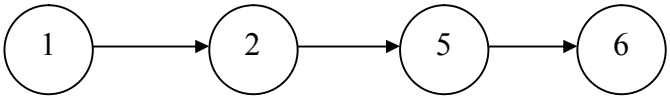


Рис. 4.35

Поскольку после выполнения 3-й итерации текущий поток еще не является максимальным, то продолжим выполнение итераций алгоритма нахождения максимального потока в сети. На рис. 4.36 проиллюстрирована 4-я итерация алгоритма нахождения максимального потока.

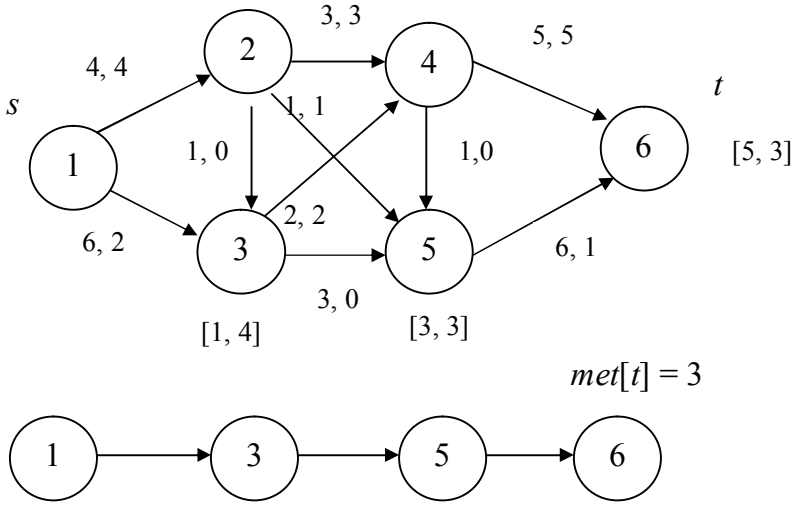


Рис. 4.36

На 5-й итерации можно пометить только третью вершину; таким образом, вершина $t = 6$ останется непомеченной. Следовательно, текущий поток величины 9 является максимальным (рис. 4.37).

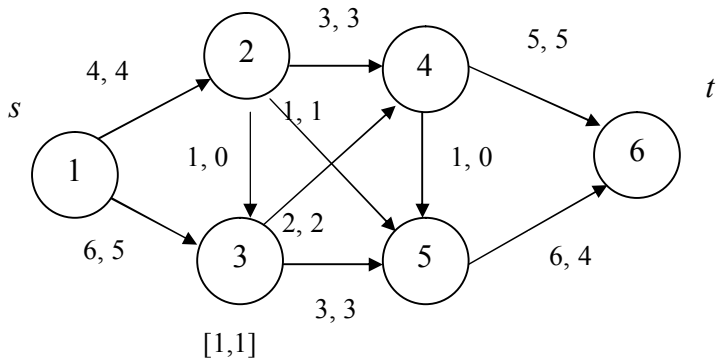


Рис. 4.37

Максимальное количество путей,

не пересекающихся по дугам (вершинам)

Многие задачи на орграфах могут быть сведены к задаче нахождения максимального потока в сети с пропускной способностью дуг, равной 1. Например, к данной задаче сводится задача нахождения максимального количества путей в орграфе, которые не пересекаются по дугам (вершинам). Для решения данной задачи пропускные способности всех дуг исходного орграфа полагают равными единице, и находится максимальный поток между заданной парой вершин. Величина максимального потока равна количеству путей, не пересекающихся по дугам.

Пример 4.17. Для орграфа, который изображен на рис. 4.38, найти максимальное количество путей из вершины $s=1$ в вершину $t=10$, которые не пересекаются по дугам.

Решение. Полагаем пропускные способности всех дуг равными единице. Найдем максимальный поток из вершины $s=1$ в вершину $t=10$. Первоначально текущий поток по всем дугам полагаем равным нулю (текущий поток по дуге на рисунках будем указывать возле каждой дуги, но если величина потока по дуге равна нулю, то ничего писать не будем). На каждой итерации вершине v ставятся в соответствие две величины (на рисунке эти величины пишутся возле вершины в квадратных скобках):

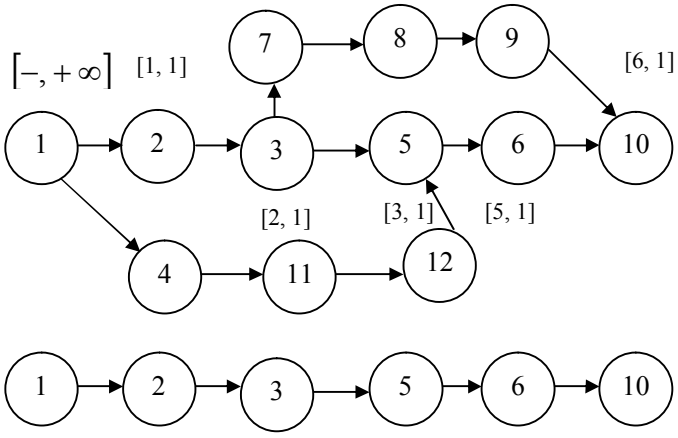


Рис. 4.38

- номер вершины, из которой данная вершина была помечена;
- метка *met*[*v*], определяемая по правилу, описанному ранее в алгоритме построения максимального потока в сети.

После первой итерации (рис. 4.38) получим увеличивающий путь, состоящий из последовательности вершин: 1, 2, 3, 5, 6, 10. Текущий поток увеличится на единицу. На рис. 4.39 показан поток, построенный в результате изменения текущего потока, полученного на предыдущей итерации.

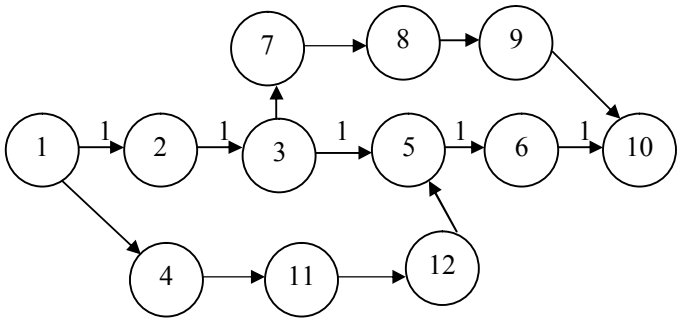


Рис. 4.39

На рис. 4.40 выполнена вторая итерация алгоритма. Дуги, по которым текущий поток, построенный на предыдущей итерации, равнялся единице, выделены жирными линиями.

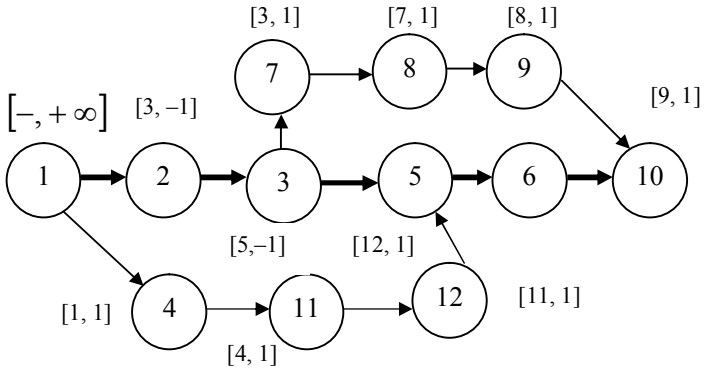


Рис. 4.40

После второй итерации получим увеличивающий путь: 1, 4, 11, 12, 5, 3, 7, 8, 9, 10 (рис. 4.41).

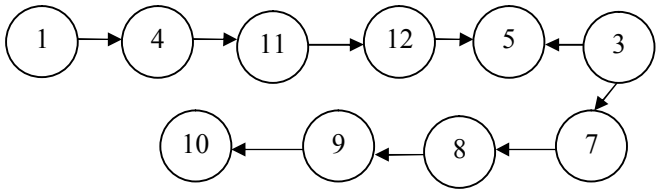


Рис. 4.41

Увеличиваем текущий поток (величина потока увеличится на единицу). Поскольку дуга (3, 5) увеличивающего пути является обратной, то после увеличения потока текущий поток по ней будет равен нулю. Последовательность дуг, по которым поток равен единице, образует два пути из вершины $s=1$ в вершину $t=10$, которые не пересекаются по дугам (дуги путей выделены на рис. 4.42).

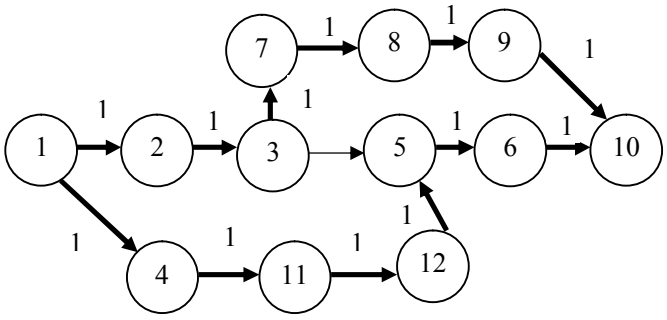


Рис. 4.42

Полскольку величина максимального потока равна двум, то в орграфе существуют два пути, не пересекающихся по дугам из вершины $s=1$ в вершину $t=10$. Для восстановления дуг, по которым проходят пути, можно выполнить поиск в глубину. Поиск в глубину выполняется по дугам, максимальный поток по которым равен единице (поиск в глубину выполняется из точки старта *s* столько раз, сколько различных максимальных путей существует; во время поиска блокируются дуги, по которым осуществляется движение).

Таким образом, в задачах нахождения максимального количества путей, не пересекающихся по дугам, множество дуг делится на два подмножества:

- *дуги*, *по которым ходили* (поток по ним равен единице);
- *дуги*, *по которым не ходили* (поток по ним равен нулю).

Рассмотрим сейчас алгоритм нахождения максимального количества путей, не пересекающихся по дугам, который основан на алгоритме нахождения максимального потока.

Алгоритм
нахождения максимального количества путей,
не пересекающихся по дугам

1. Занести в очередь стартовую вершину *s*. Все множество дуг относим к *подмножеству дуг*, *по которым не ходили*.

2. Пока очередь не станет пустой или в очередь не будет занесена конечная вершина t, будем выполнять шаги в следующей очередности.

а) Предположим, что первой в очереди находится вершина v. Для каждой вершины w, в которую ведут дуги из вершины v:

- если (v, w) – дуга, которая относится к *подмножеству дуг*, *по которым не ходили*, то вершину w заносим в очередь;
- если (v, w) – дуга, которая относится к *подмножеству дуг*, *по которым ходили*, то ничего не делаем;
- если (w, v) – дуга, которая относится к *подмножеству дуг*, *по которым ходили*, то вершину w заносим в очередь;
- если (w, v) – дуга, которая относится к *подмножеству дуг*, *по которым не ходили*, то ничего не делаем.

б) Удаляем вершину v из очереди.

3. Если конечная вершина t была занесена в очередь, то делаем очередь пустой и возвращаемся к шагу 1 алгоритма. При этом дуги, реализующие путь из s в t, переформировываем по следующему правилу:

- прямые дуги пути (эти дуги ранее относились к *подмножеству дуг*, *по которым не ходили*) относим к *подмножеству дуг*, *по которым ходили*;
- обратные дуги пути (эти дуги ранее относились к *подмножеству дуг*, *по которым ходили*) относим к *подмножеству дуг*, *по которым не ходили*.

4. Если очередь пуста, а конечная вершина t не была занесена в очередь, то дуги, принадлежащие *подмножеству дуг*, *по которым ходили*, входят в максимальное количество путей, не пересекающихся по дугам. Алгоритм заканчивает свою работу.

Следует отметить, что не всегда по всем дугам, принадлежащим *подмножеству дуг*, *по которым ходили*, пройдут искомые пути. Проиллюстрируем это на следующем примере.

На рис. 4.41 изображен орграф, для которого требуется найти максимальное число путей, не пересекающихся по дугам из вершины s в вершину t.

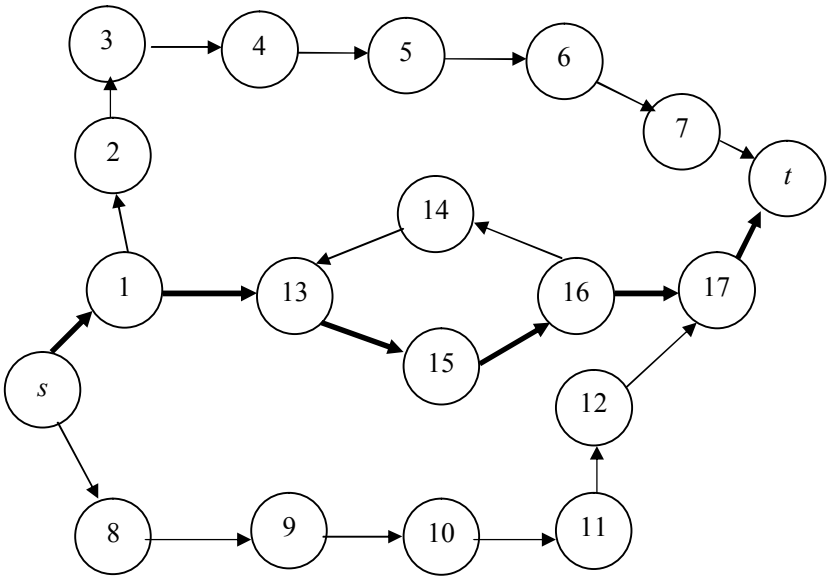


Рис. 4.43

После первой итерации дуги, принадлежащие *подмножеству дуг*, *по которым ходили*, на рис. 4.43 выделены:

(*s*, 1), (1, 13), (13, 15), (15,16), (16, 17), (17, *t*).

После второй итерации дуги увеличивающего пути выделены на рис. 4.44 пунктиром. Увеличивающий путь задается следующими дугами:

(*s*, 8), (8, 9), (9, 10), (10,11), (11, 12), (12, 17), (16, 17), (16, 14),
(14, 13), (1, 13), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, *t*).

Дуги приведенного выше увеличивающего пути: (1, 13) и (16, 17) ранее относились к *подмножеству дуг*, *по которым ходили*, поэтому теперь эти две дуги будут отнесены к *подмножеству дуг*, *по которым не ходили*.

Все остальные дуги увеличивающего пути добавим к *подмножеству дуг*, *по которым ходили*, имеющему теперь вид:

(*s*, 1), (1, 13), (13, 15), (15,16),
(*s*, 8), (8, 9), (9, 10), (10,11),
(11, 12), (12, 17), (16, 14),
(14, 13), (1, 2), (2, 3), (3, 4),
(4, 5), (5, 6), (6, 7), (7, *t*).

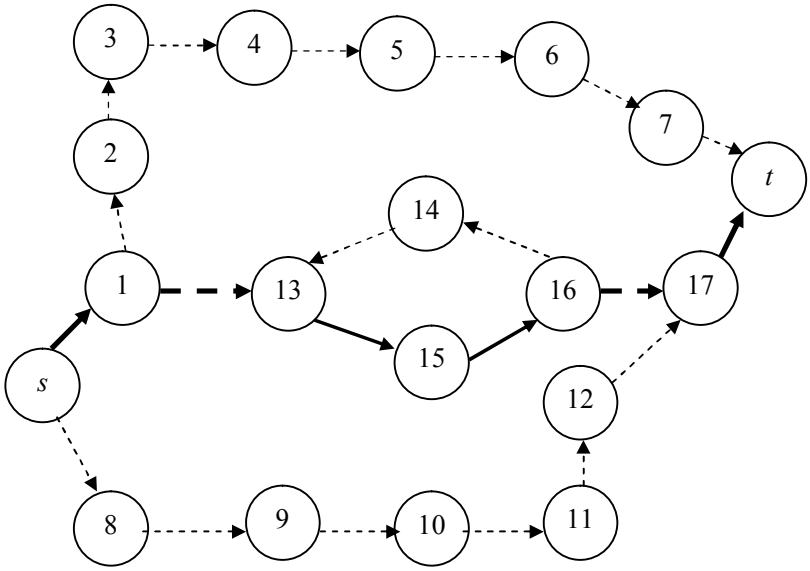


Рис. 4.44

Итоговое множество дуг, принадлежащее *подмножеству дуг*, *по которым ходили*, на рис. 4.45 выделено жирными линиями.

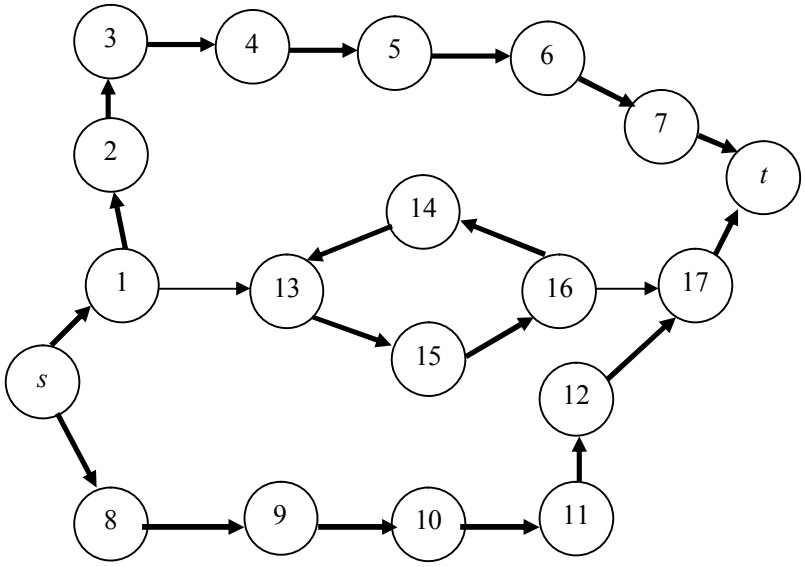


Рис. 4.45

Поскольку величина максимального потока равна двум, то в орграфе существуют два пути из s в t, которые не пересекаются по дугам. Оба пути проходят по дугам, принадлежащим *подмножеству дуг*, *по которым ходили*. Первый путь проходит по дугам

$$(s, 8),\ (8, 9),\ (9, 10),\ (10,11),\ (11, 12),\ (12, 17),\ (17, t).$$

Второй путь проходит по дугам

$$(s, 1),\ 1, 2),\ (2, 3),\ (3, 4),\ (4, 5),\ (5, 6),\ (6, 7),\ (7, t).$$

Однако существуют четыре дуги принадлежащие *подмножеству дуг*, *по которым ходили*, но пути по ним не проходят:

$$(13, 15),\ (15,16),\ (16, 14),\ (14, 13).$$

Следовательно, для восстановления дуг, по которым проходят пути, можно выполнить поиск в глубину. Поиск в глубину выполняется по дугам, принадлежащим *подмножеству дуг*, *по которым ходили*. Поиск выполняется из точки старта s столько раз, сколько различных максимальных путей существует; во время поиска дуги, по которым осуществляется движение, относим к *подмножеству дуг*, *по которым не ходили*.

Упражнение 4.4. Обобщить алгоритм нахождения максимального количества путей, не пересекающихся по дугам, для нескольких источников и стоков.

Задача нахождения *максимального числа путей* между вершинами s и t, *которые не пересекаются по вершинам*, может быть сведена к предыдущей задаче. Для этого:

- каждую вершину v (отличную от s и t) расщепляем на две вершины: v и v';
- соединяем вершины v и v' дугой;
- все дуги, которые входили в вершину v, оставляем без изменения, а дуги, которые выходили из вершины v, теперь будут выходить из вершины v';
- пропускные способности всех дуг полагаем равными единице и находим максимальный поток между вершинами s и t; величина максимального потока – максимальное число путей между вершинами s и t, которые не пересекаются по вершинам;
- для восстановления дуг, по которым пройдут пути, сжимаем расщепленные вершины; дуги, по которым максимальный поток равен единице, могут быть выбраны в качестве дуг путей (для восстановления самих путей поступаем аналогично, как и в предыдущей задаче).

Упражнение 4.5. Разработать алгоритм нахождения максимального количества путей, не пересекающихся по вершинам для графа.

Наибольшее паросочетание в двудольном графе

Напомним, что *паросочетание* – это такое подмножество ребер графа, что никакие два ребра из этого подмножества не имеют общих вершин. Паросочетание наибольшей мощности называется *наибольшим паросочетанием*. Паросочетание, которое покрывает все вершины графа, называется *совершенным паросочетанием*.

Задача построения наибольшего паросочетания в двудольном графе $G=(V, E)$, $V=X \bigcup Y$, где X, Y – множества вершин первой и второй доли графа G соответственно, сводится к решению задачи о максимальном потоке. Для этого двудольный граф сначала достраиваем до сети:

- вводим фиктивную вершину s, которую соединяем дугами со всеми вершинами первой доли;
- вводим фиктивную вершину t, с которой соединяем дугами все вершины второй доли t;
- каждому ребру двудольного графа $(v, w) \in E, v \in X, w \in Y$, в сети ставим в соответствие дугу (v, w) (дуги идут от вершин первой доли в вершины второй доли);
- пропускные способности всех дуг в сети полагаем равными единице.

В сгенерированной сети находим максимальный поток. Величина максимального потока – мощность наибольшего паросочетания. Дуги сети, соответствующие ребрам двудольного графа, и поток по которым равен 1, дают множество ребер наибольшего паросочетания двудольного графа.

Трудоемкость алгоритма построения наибольшего паросочетания в двудольном графе равна $O(n \cdot m)$ (так как любое паросочетание в двудольном графе имеет мощность не более чем $n/2$).

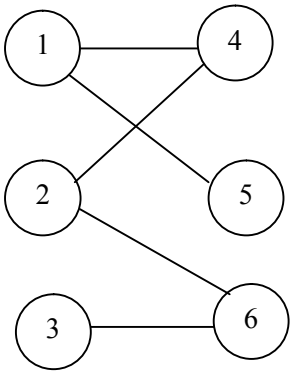


Рис. 4.46

Пример 4.18. Для двудольного графа, изображенного на рис. 4.46, построить наибольшее паросочетание.

Решение. Достраиваем исходный граф до сети. Вводим фиктивные вершины s и t, соединяем вершину s дугами со всеми вершинами первой доли:

$$(s, 1), (s, 2), (s, 3).$$

Соединяем дугами все вершины второй доли с вершиной *t*:

(4, *t*), (5, *t*), (6, *t*).

Ориентируем ребра двудольного графа дугами (от вершин первой доли к вершинам второй доли). Пропускные способности всех дуг полагаем равными единице и находим максимальный поток в сети.

Величина максимального потока равна 3 (дуги, по которым проходит максимальный поток, на рис. 4.47 выделены).

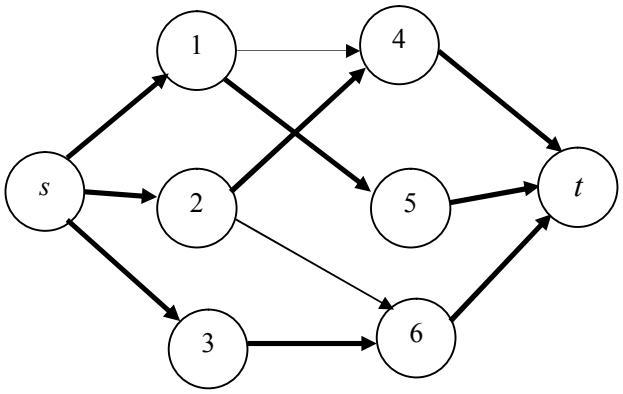


Рис. 4.47

Величина максимального потока – мощность наибольшего паросочетания. Ребра двудольного графа, поток по которым равен единице, соответствуют множеству ребер наибольшего паросочетания (паросочетание является совершенным): (1,5), (2, 4), (3,6) (рис. 4.48).

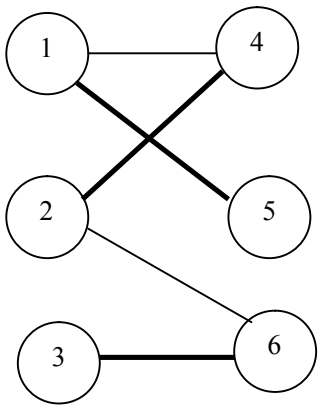


Рис. 4.48

Замечание 4.13. Для построения наибольшего паросочетания в орграфе заменяем каждую дугу ребром и применяем описанный выше алгоритм.

Допустимая циркуляция

Определение 4.2. *Циркуляцией* в орграфе $G=(V,U)$ называется такая функция $g:U\to R$, при которой любой вершины v орграфа G выполняется условие баланса:

$$\sum_{(w,v)\in U(V,v)} g(w,v)=\sum_{(v,w)\in U(v,V)} g(v,w),$$

где $U(V,v)$ – множество дуг, входящих в вершину v, а $U(v,V)$– множество дуг, выходящих из вершины v.

Предположим, что для каждой дуги орграфа задана верхняя и нижняя границы:

$$l(v,w)\le g(v,w)\le c(v,w),\quad 0\le l(v,w)\le c(v,w).$$

Циркуляция, которая удовлетворяет приведенному выше неравенству, называется *допустимой циркуляцией*. Мы будем рассматривать задачу построения допустимой циркуляции в орграфе.

Расширим исходный орграф $G=(V,U)$ до сети G', присоединяя вершины s и t, а также множество дуг (s,v) и (v,t) для каждой вершины $v\in V$.

Для построенной сети введем следующие пропускные способности дуг:

1. $c'(v,w)=c(v,w)-l(v,w),\quad \forall(v,w)\in U.$
2. $c'(s,v)=\sum_{(w,v)\in U(V,v)} l(w,v)$– это минимальное количество продукта, которое обязательно должно быть ввезено в вершину v.
3. $c'(v,t)=\sum_{(v,w)\in U(v,V)} l(v,w)$– это минимальное количество продукта, которое обязательно должно быть вывезено из вершины v.

Теперь решим для сети G' задачу о максимальном потоке.

Если величина максимального потока f равна сумме пропускных способностей дуг, выходящих из s (входящих в t), то допустимая циркуляция существует и она может быть вычислена по следующему правилу:

$$\forall(v,w)\in U: g(v,w)=f(v,w)+l(v,w).$$

В противном случае допустимой циркуляции не существует.

Пример 4.19. Для орграфа G, изображенного на рис. 4.49, найти допустимую циркуляцию.

Возле каждой дуги (v, w) указаны в виде дроби нижняя и верхняя границы для циркуляции: $l(v, w) \big/ c(v, w)$.

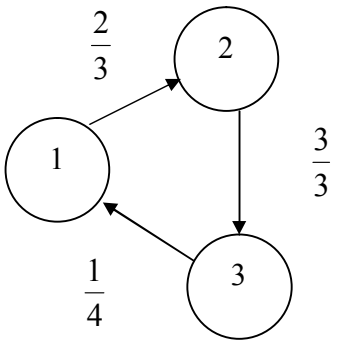


Рис. 4.49

Решение. Расширим исходный орграф до сети G' и решим задачу о максимальном потоке.

На рис. 4.50 возле каждой дуги (v, w) указаны два числа: пропускная способность $c'(v, w)$ и величина потока по этой дуге $f(v, w)$.

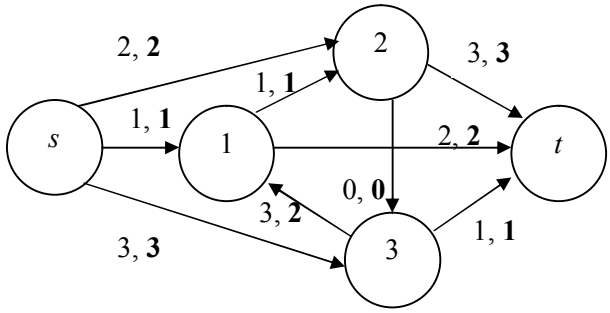


Рис. 4.50

Величина максимального потока $f = 6$ равна сумме пропускных способностей дуг, выходящих из s (рис. 4.48), поэтому допустимая циркуляция существует и может быть вычислена по формуле

$$\forall (v, w) \in U : g(v, w) = f(v, w) + l(v, w).$$

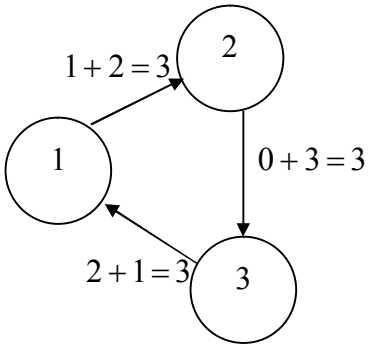


Рис. 4.51

Допустимая циркуляция изображена на рис. 4.51.

4.6. ЦИКЛЫ ОТРИЦАТЕЛЬНОЙ СТОИМОСТИ

Большинство алгоритмов построения кратчайших путей в графах работали в предположении, что в графе отсутствуют циклы отрицательной стоимости. Однако существует целый ряд задач с решением, сводящимся к решению подзадач, в которых необходимо определить цикл отрицательной стоимости. В п. 4.6 будут рассмотрены такие задачи, как:

- наибольшее паросочетание максимального веса в двудольном графе;
- максимальный поток минимальной стоимости в графе;
- минимальный средний цикл в графе.

4.6.1. Наибольшее паросочетание максимального веса в двудольном графе

Задан двудольный граф $G=(V, E)$, $V=X \bigcup Y$, где X, Y – множества вершин первой и второй доли графа G соответственно. Каждому ребру (v, w) графа G поставлен в соответствие неотрицательный вес $c(v, w) \geq 0$.

Определение 4.3. Задача нахождения в двудольном графе G наибольшего паросочетания с максимальным суммарным весом ребер называется *задачей о наибольшем паросочетании максимального веса в двудольном графе.*

Пусть $P=\{(x_1, y_1), (x_2, y_2),\ldots,(x_l, y_l)\}, x_i \in X, y_i \in Y$, есть множество ребер наибольшего паросочетания в двудольном графе $G=(V, E)$. Для построения наибольшего паросочетания в двудольном графе можно использовать, например, алгоритм, описанный в предыдущем разделе, который основан на методе Форда–Фалкерсона.

Алгоритм
построения наибольшего паросочетания
максимального веса в двудольном графе

1. Строим орграф G' по следующему правилу:

- каждому ребру (x_k, y_k) исходного двудольного графа G, которое принадлежит паросочетанию P, в орграфе G' будет соответствовать дуга (x_k, y_k) веса $c'(x_k, y_k) = c(x_k, y_k)$;
- каждому ребру (x_k, y_k) исходного двудольного графа G, которое не принадлежит паросочетанию P, в орграфе G' будет соответствовать дуга (y_k, x_k) веса $c'(y_k, x_k) = -c(x_k, y_k)$.

2. Находим в орграфе G' контур C отрицательной стоимости (для этого можно использовать алгоритм Форда–Беллмана).

Наличие контура отрицательной стоимости означает, что имеющееся паросочетание P не является наибольшим паросочетанием максимального веса.

Действительно, если получен контур отрицательной стоимости, то в таком случае половина дуг контура будет соответствовать ребрам паросочетания P, а половина – нет.

Нетрудно заметить, что одно множество может быть заменено на другое, и при этом получится новое наибольшее паросочетание, вес которого станет больше (часть ребер паросочетания P заменилась другими ребрами, суммарный вес которых больше).

Преобразуем паросочетание P по следующему правилу:

- если дуга (y_k, x_k) контура C имеет отрицательный вес, то добавляем ребро (x_k, y_k) к паросочетанию P;
- если дуга (x_k, y_k) контура C имеет положительный вес, то удаляем ребро (x_k, y_k) из паросочетания P.

Если вес отрицательного контура есть число z, то вес нового паросочетания P будет больше веса предыдущего паросочетания на величину $|z|$. Возвращаемся к шагу 1 алгоритма.

4. Если в орграфе G' не существует контура отрицательной стоимости, то текущее паросочетание P является наибольшим паросочетанием максимального веса в двудольном графе $G = (V, E)$.

Алгоритм завершает свою работу.

Пример 4.20. В двудольном графе $G=(V,E)$, $V=X\bigcup Y$, $X=\{1,2,3\}, Y=\{4,5,6\}$, изображенном на рис. 4.52, найти наибольшее паросочетание максимального веса.

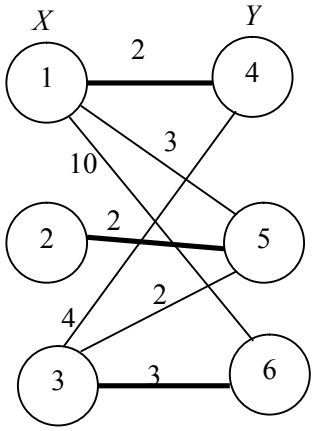


Рис. 4.52

Решение. В качестве начального максимального паросочетания P берем следующий набор ребер исходного графа G: $P=\{(1,4),(2,5),(3,6)\}$ (на рис. 4.52 ребра паросочетания P выделены). Вес паросочетания P равен семи.

Строим по паросочетанию P орграф G' (рис. 4.53).

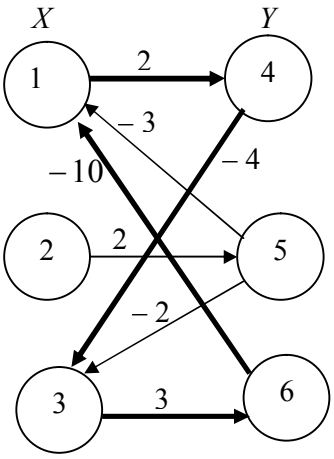


Рис. 4.53

В орграфе G' существует контур $C = \{(3,6),(6,1),(1,4),(4,3)\}$ отрицательного веса (на рис. 4.53 дуги контура отрицательного веса выделены). Вес контура C: $z = -9$.

Дуги контура C: $(1,4)$ и $(3,6)$ имеют положительный вес, поэтому исключаем их из паросочетания P. Дуги контура C: $(6,1)$ и $(4,3)$ имеют отрицательный вес, поэтому добавляем ребра $(1,6)$ и $(3,4)$ в паросочетание P. Паросочетание после преобразования имеет вид

$$P = \{(1,6),(2,5),(3,4)\},$$

а его вес равен 16 (на рис. 4.54 ребра паросочетания выделены).

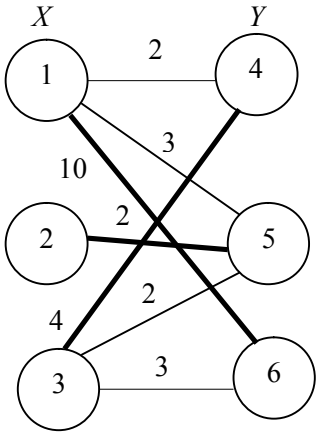


Рис. 4.54

Строим по паросочетанию P орграф G' (рис. 4.55).

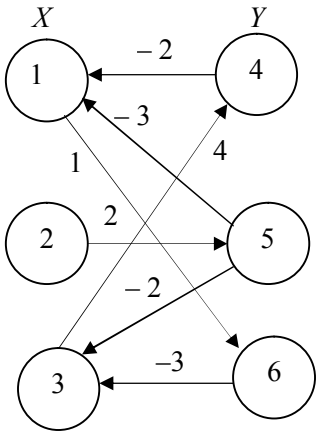


Рис. 4.55

В орграфе G' отсутствуют контуры отрицательного веса. Следовательно, паросочетание $P=\{(1,6),(2,5),(3,4)\}$ стоимости 16 является максимальным взвешенным паросочетанием в двудольном графе G (на рис. 4.54 ребра паросочетания выделены).

Если ставится задача определения максимального взвешенного паросочетания для орграфа, то заменяем каждую дугу ребром и применяем описанный выше алгоритм.

4.6.2. Максимальный поток минимальной стоимости

Задан орграф $G=(V,U)$, каждой дуге (v,w) орграфа G поставлены в соответствие две величины:

- стоимость дуги $p(v,w)$, равная той стоимости, которая должна быть уплачена за провоз одной единицы продукта по дуге;
- пропускная способность дуги $c(v,w)$.

Выделены две вершины: s (в которую нет входящих дуг) и t (из которой нет выходящих дуг). Таким образом, мы имеем некоторую сеть S.

Определение 4.4. Максимальным потоком минимальной стоимости называется максимальный поток, стоимость которого минимальна.

Алгоритм
построения максимального потока
минимальной стоимости

1. В сети S находим максимальный поток f любым известным алгоритмом.
2. Формируем новую сеть: $S'=(V,E')$. Множество дуг сети E' и их стоимости p' зависят от максимального потока f. После построения максимального потока f возможно появление трех типов дуг:

- насыщенные дуги (v,w): $f(v,w)=c(v,w)$, им соответствует в сети S' одна дуга:

$$(v,w)\colon p'(v,w)=-p(v,w);$$

- ненасыщенные дуги (v,w), поток по которым больше нуля ($0<f(v,w)<c(v,w)$), им соответствуют в сети S' две дуги:

$$(v,w)\colon p'(v,w)=-p(v,w);$$
$$(w,v)\colon p'(w,v)=p(v,w);$$

- дуги (v, w), поток по которым равен 0 ($f(v, w) = 0$), им соответствует в сети S' одна дуга:

$$(w, v)\colon p'(w, v) = p(v, w).$$

3. Строим в сети S' контур отрицательной стоимости, например, используя алгоритм Форда–Беллмана.

4. Если контур отрицательной стоимости есть, то существует максимальный поток меньшей стоимости, чем текущий поток *f*.

Действительно, если величина контура отрицательной стоимости есть число $z < 0$, то это говорит о том, что, перераспределяя вдоль контура отрицательной стоимости одну единицу потока, можно получить поток, стоимость которого будет меньше на $|z|$ единиц, чем текущий поток.

Обозначим через *C* контур отрицательной стоимости:

$$C = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_1)\}.$$

Для дуг сети, не принадлежащих контуру, величину потока по ним не изменяем.

Для дуг сети, принадлежащих контуру, модифицируем поток *f* по следующему правилу:

- если дуга $(v_l, v_{l+1}) \in C$ имеет отрицательную стоимость в сети S', то величина потока по ней в орграфе *G* будет уменьшаться на величину *d*;
- если дуга $(v_l, v_{l+1}) \in C$ имеет положительную стоимость в сети S', то величина потока по дуге (v_{l+1}, v_l) в орграфе *G* будет увеличиваться на величину *d*.

Величину *d* изменения потока определяем следующим образом:

$$d = \min_{(v, w) \in C} \begin{cases} f(v, w), & \text{если} \quad p'(v, w) < 0; \\ c(w, v) - f(w, v), & \text{если} \quad p'(v, w) > 0. \end{cases}$$

После модификации потока f возвращаемся к шагу 2 алгоритма.

5. Если контура отрицательной стоимости в сети S' не существует, то текущий поток f является максимальным потоком минимальной стоимости в сети *S* и алгоритм завершает свою работу.

Пример 4.21. Определить максимальный поток минимальной стоимости в сети *S*, изображенной на рис. 4.56.

Для каждой дуги (v, w) сети *S* указаны: сначала пропускная способность $c(v, w)$ (выделена жирным шрифтом) и затем – стоимость $p(v, w)$.

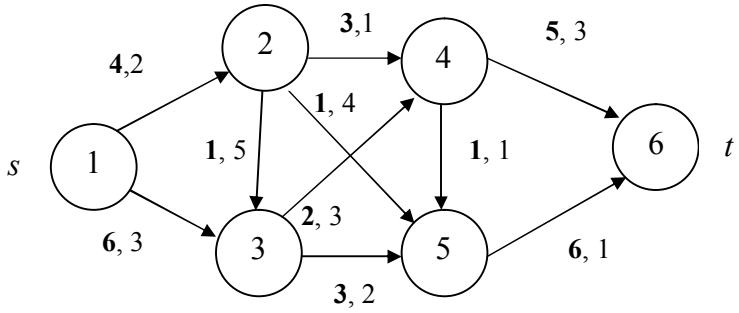


Рис. 4.56

Решение. Находим в сети S, изображенной на рис. 4.56, максимальный поток f. Величина максимального потока равна 9, а его стоимость – 61 (на рис. 4.57 возле каждой дуги сети S указаны две величины: пропускная способность дуги (выделена жирным шрифтом) и величина текущего потока по дуге).

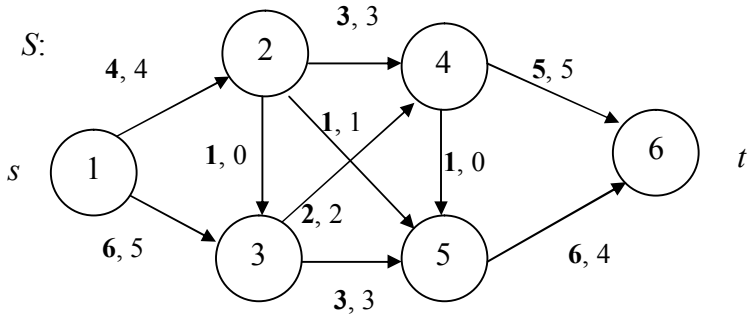


Рис. 4.57

По максимальному потоку f строим сеть S' (на рис. 4.58 возле каждой дуги сети указана ее стоимость p').

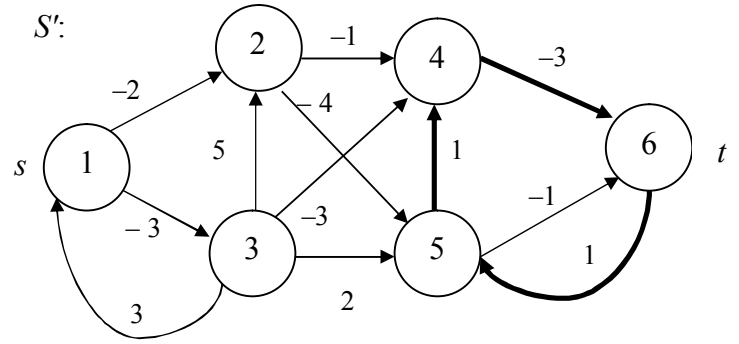


Рис. 4.58

В сети S' существует контур отрицательной стоимости: $C = \{(4,6),(6,5),(5,4)\}$ (на рис. 4.58 дуги контура выделены). Стоимость контура равна $z = -1$. Следовательно, перераспределяя поток вдоль этого контура, на каждой единице потока можно получить выигрыш в стоимости, равный 1. Определим величину *d*, равную той максимальной величине потока, которую можно перераспределить вдоль найденного контура *C*:

$$d = \min\{f(4,6), c(5,6) - f(5,6), c(4,5) - f(4,5)\} = 1.$$

Перераспределяя поток вдоль контура *C* по описанному ранее правилу, получим новый максимальный поток *f* (на рис. 4.59 возле каждой дуги сети *S* указаны ее пропускная способность (выделена жирным шрифтом) и величина модифицированного максимального потока *f*).

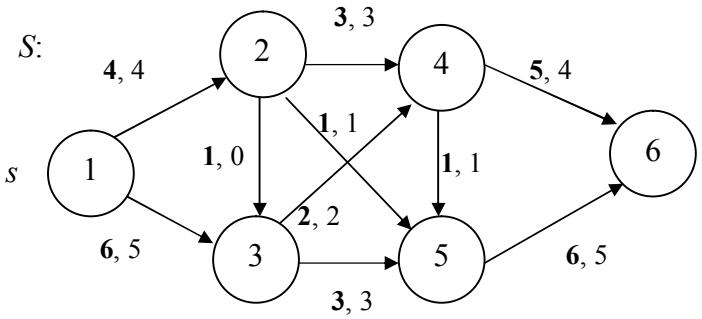


Рис. 4.59

Стоимость нового максимального потока должна уменьшиться на величину $d \cdot |z| = 1$. Модифицированный текущий поток имеет стоимость, равную 60.

По новому максимальному потоку *f* строим сеть S' (рис. 4.60).

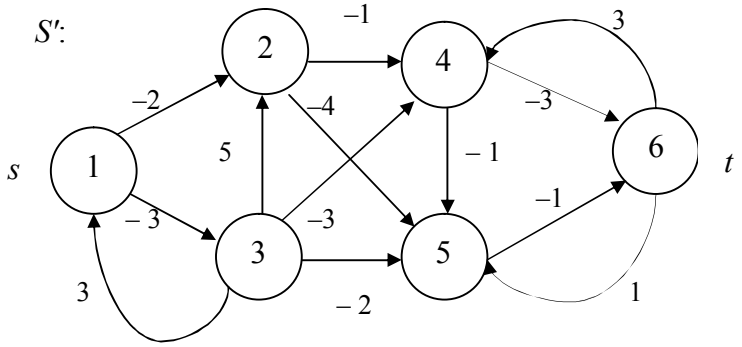


Рис. 4.60

В сети S' отсутствуют контуры отрицательной стоимости. Следовательно, текущий максимальный поток f является максимальным потоком минимальной стоимости 60.

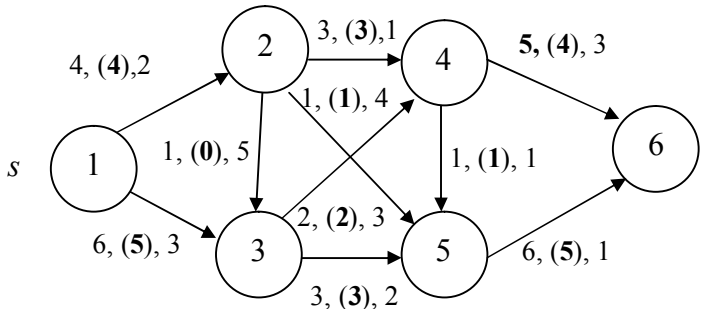


Рис. 4.61

На рис 4.61 возле каждой дуги (v, w) указаны: сначала пропускная способность дуги $c(v, w)$, затем (в скобках) – величина максимального потока минимальной стоимости $f(v, w)$ по данной дуге и, наконец, – стоимость одной единицы потока $p(v, w)$ по дуге.

4.6.3. Минимальный средний контур в орграфе с положительными стоимостями дуг

В дискретной оптимизации весьма интересный раздел составляют задачи поиска подмножества заданного вида, имеющего минимальный усредненный вес элемента.

Пусть задан взвешенный орграф $G = (V, U)$, где $|V| = n$. Каждой дуге (v, w) орграфа поставлена в соответствие неотрицательная стоимость $p(v, w)$ (имеем матрицу стоимостей $P[n \times n]$).

Определение 4.5. Для ориентированного графа G с матрицей стоимостей P *стоимость контура* $C = \left\{ \left(v_{i_1}, v_{i_2}\right), \left(v_{i_2}, v_{i_3}\right), \ldots, \left(v_{i_{k-1}}, v_{i_k}\right), \left(v_{i_k}, v_{i_1}\right) \right\}$ определяется по формуле

$$P(C) = \sum_{(v, w) \in C} p(v, w),$$

а *средняя стоимость контура* определяется как

$$P_a(C) = \frac{P(C)}{|C|}.$$

Задача о минимальном среднем контуре в орграфе G заключается в нахождении такого контура, средняя стоимость которого минимальна.

Предположим, что предприниматель хочет организовать некоторую циклическую последовательность действий по уменьшению своих расходов. В этом случае множеству возможных действий поставим в соответствие дуги орграфа, затратам на определенное действие – вес дуги и предположим, что каждое действие выполняется за единицу времени. В этом случае средняя стоимость некоторого контура орграфа – расходы предпринимателя в единицу времени в соответствии с выбранным маршрутом. Таким образом, предпринимателю для минимизации своих расходов выгодно придерживаться последовательности, соответствующей контуру с минимальной средней стоимостью.

Для орграфа G, наряду с матрицей стоимостей P, будем рассматривать матрицу стоимостей $P^x[n \times n]$, элементы которой вычисляются по правилу

$$p^x[v, w] = p[v, w] - x.$$

Заметим, что для любого контура C орграфа G справедливо следующее равенство:

$$P_a(C) = P_a^x(C) + x,$$

откуда следует, что задачи нахождения минимального среднего контура в орграфе G с матрицами стоимостей P и P^x эквивалентны.

Пусть m – минимальная средняя стоимость контура в орграфе G с матрицей стоимостей P, а C^m – множество дуг этого контура. Очевидно, что $L \le m \le R$, где

$$L = \min_{(v, w) \in U} p(v, w), \quad R = \max_{(v, w) \in U} p(v, w).$$

Предположим, что x – некоторое число, тогда справедливы следующие утверждения:

- для любого значения $x < m$ стоимость произвольного контура Y орграфа G больше нуля, т. е. $P^x(Y) > 0$;
- для любого значения $x > m$ существует контур Y отрицательной стоимости, т. е. $P^x(Y) < 0$.

Тогда, полагаем

$$x = \left\lceil \frac{L + R}{2} \right\rceil,$$

где значения L и R определены по правилу, описанному ранее. Используя дихотомию по значению x на отрезке $[L, R]$ (если в орграфе G существует контур отрицательной стоимости, то полагаем $R = x$; если в орграфе G не существует контура отрицательной стоимости, то полагаем $L = x$), определяем такое значение x^*, для которого в орграфе G существует контур отрицательной стоимости, а для величины $x^* - \varepsilon$ (при $\varepsilon \to 0$) контура отрицательной стоимости не существует. Значение контура минимальной средней стоимости принадлежит отрезку $\left[x^* - \varepsilon, x^*\right]$. Отметим, что если значение контура минимальной средней стоимости есть целое число, то алгоритм дихотомического поиска будет завершен на отрезке $\left[x^* - 1, x^*\right]$, причем для значения $\left(x^* - 1\right)$ в орграфе G не существует контура отрицательной стоимости, но существует контур нулевой стоимости. Именно этот контур и является минимальным средним контуром в орграфе, а значение $\left(x^* - 1\right)$ – минимальное среднее значение контура.

Пример 4.22. Для орграфа G, каждой дуге (v, w) которого поставлена в соответствие некоторая стоимость $p(v, w)$, определить минимальный средний контур (рис. 4.62).

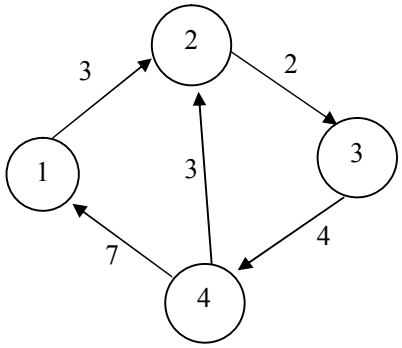


Рис. 4.62

Решение. Полагаем первоначально $L = 2, R = 7, x = 5$. Тогда получаем для орграфа G матрицу стоимостей P^5. В орграфе G с матрицей стоимостей P^5 (рис. 4.63) существует контур отрицательной стоимости:

$$C = \{(1, 2), (2, 3), (3, 4), (4, 1)\}, \quad P^5(C) = -4.$$

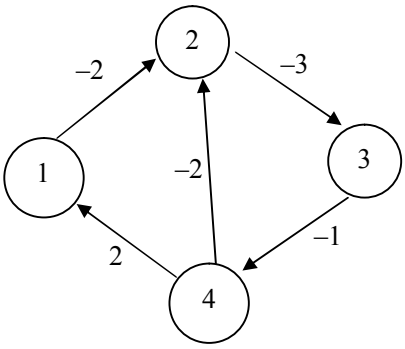


Рис. 4.63

Продолжаем дихотомический поиск, полагая $L = 2, R = 5, x = 4$. На рис. 4.64 изображен орграф G с матрицей стоимостей P^4.

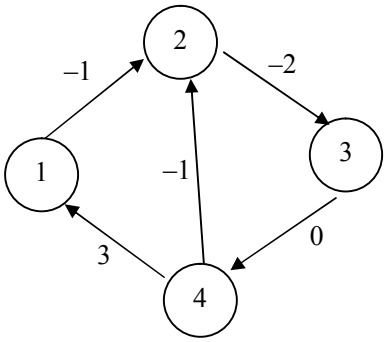


Рис. 4.64

В орграфе G с матрицей стоимостей P^4 существует контур отрицательной стоимости:

$$C = \{(2,3),(3,4),(4,2)\}, \quad P^3(C) = -3.$$

Продолжаем дихотомический поиск, полагая $L = 2, R = 4, x = 3$.

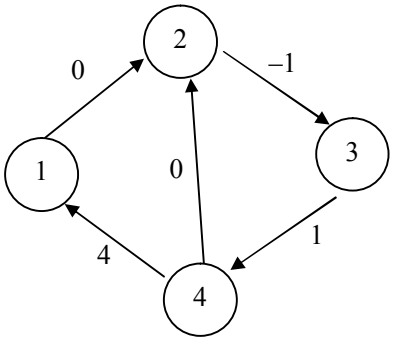


Рис. 4.65

На рис. 4.65 изображен орграф G с матрицей стоимостей P^3, в котором не существует контура отрицательной стоимости, но существует контур нулевой стоимости. Поэтому контур нулевой стоимости:

$$C = \{(2,3),(3,4),(4,2)\}, \quad P^3(C) = 0$$

является минимальным средним контуром в орграфе, и его средняя стоимость равна 3 (рис. 4.66).

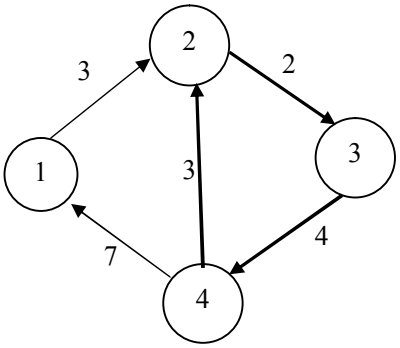


Рис. 4.66

Упражнение 4.6. Как обобщить задачу о минимальном среднем контуре в случае, когда каждой дуге соответствуют затраты и длительность действий?

4.7. МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО ГРАФА

Пусть задан взвешенный граф $G = (V, E)$ ($|V| = n, |E| = m$), каждому ребру (v, w) которого поставлена в соответствие стоимость $c(v, w)$.

Определение 4.6. Остовным деревом T графа G называется неориентированное дерево, которое содержит все вершины графа.

Определение 4.7. Задача о наименьшем остовном дереве заключается в построении остовного дерева T графа G, суммарная стоимость ребер которого минимальна.

Известно несколько эффективных алгоритмов решения задачи о наименьшем остовном дереве, но все они используют следующий факт.

ТЕОРЕМА 4.2. Пусть $\{(U_1,T_1),(U_2,T_2),\ldots,(U_k,T_k)\}$ – остовный лес на множестве V, и пусть (v,u) – кратчайшее из всех ребер, у которых ровно один конец содержится в U_1. Тогда среди всех остовных деревьев, со-

держащих все ребра из множества $T = T_1 \bigcup T_2 \bigcup \ldots \bigcup T_k$, существует оптимальное остовное дерево, содержащее также ребро (v, u).

Общая стратегия алгоритма построения остовного дерева состоит в том, что на очередном шаге выбирается ребро, которое не образует цикла с уже выбранными на предыдущих шагах ребрами.

Для построения минимального остовного дерева достаточно на каждом очередном шаге выбирать ребро минимальной стоимости.

Рассмотрим два алгоритма построения минимального остовного дерева. Каждый из рассматриваемых алгоритмов при своей реализации использует множества.

Напомним, что основными операциями над множествами являются:

1) определение принадлежность элемента множеству

имя : = *НАЙТИ*(*i*);

2) объединение непересекающихся множеств

ОБЪЕДИНИТЬ(*имя1*, *имя2*, *имя3*).

Алгоритм 1 (*Крускала*)
построения минимального остовного дерева

a. Все ребра $(v, w) \in E$ графа G помещаем в структуру данных *куча*, где приоритетом является стоимость ребра.

Формат: Список

b. Каждой вершине $v \in V$ графа G соответствует одноэлементное множество. Пусть k – количество множеств на некотором шаге алгоритма (первоначально $k = n$).

c. Полагаем минимальное остовное дерево T графа G пустым.

d. Пока количество множеств k больше единицы, повторять следующую последовательность шагов:

i. Пусть $(v, w) \in E$ – некоторое ребро, полученное в результате выполнения операции удаления минимального элемента из кучи.

ii. Полагаем

имя1 : = *НАЙТИ* (*v*);
имя2 : = *НАЙТИ* (*w*);

(т. е. *имя1* и *имя2* – имена множеств, которым принадлежат вершины *v* и *w*).

- Если *имя*1 ≠ *имя*2 (т. е. вершины *v* и *w* принадлежат разным множествам), то выполнить следующие действия:

ОБЪЕДИНИТЬ(*имя1*, *имя2*, *имя1*);
$k := k - 1$;
$T := T \bigcup (v, w)$.

Оценим трудоемкость алгоритма Крускала.

Поскольку трудоемкость последовательности из m операций *НАЙТИ* и *ОБЪЕДИНИТЬ* – $O(m\log n)$, трудоемкость создания кучи из m элементов – $O(m)$, а трудоемкость удаления m элементов из кучи – $O(m\log m)$, то трудоемкость алгоритма Крускала – $O(m\log n)$.

Пример 4.23. Для графа, приведенного на рис. 4.67, построить минимальное остовное дерево, используя алгоритм Крускала.

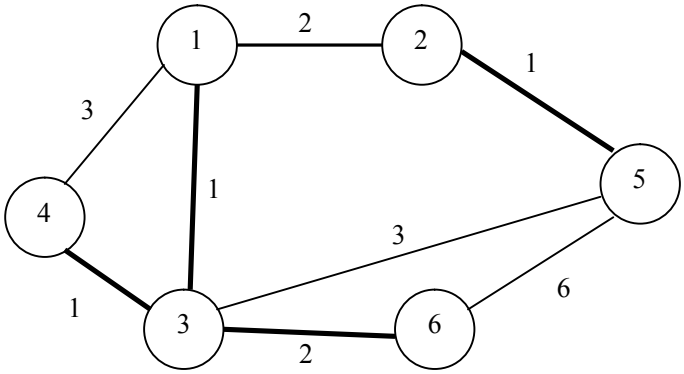


Рис. 4.67

Решение. Приведем последовательность шагов алгоритма Крускала построения минимального остовного дерева.

Итерация	Ребро	Множества
0		{1}, {2},{3},{4},{5},{6}
1	(1, 3)	{1, 3}, {2},{4},{5},{6}
2	(2, 5)	{1, 3}, {2, 5},{4},{6}
3	(3, 4)	{1, 3, 4}, {2, 5},{6}
4	(1, 2)	{1, 2, 3, 4, 5},{6}
5	(3, 6)	{1, 2, 3, 4, 5,6}

Дерево $T = \{(1,3),(2,5),(3,4),(1,2),(3,6)\}$ является минимальным остовным деревом графа G. Вес минимального остовного дерева равен 7.

Рассмотрим еще один алгоритм построения минимального остовного дерева, который в отличие от алгоритма Крускала не использует структуру данных *куча*.

Алгоритм 2
построения минимального остовного дерева

1. Каждой вершине $v \in V$ графа G поставим в соответствие одноэлементное множество VS_v (в качестве имен множеств VS_i будем рассматривать имена от 1 до n). Пусть k – количество непустых множеств (первоначально $k = n$).

2. Полагаем, что минимальное остовное дерево T графа G пустое.

3. Пока $k > 1$ (количество непустых множеств больше единицы), шаги выполняются в следующей последовательности:

- Для каждого из множеств $VS_v, v = 1,\ldots,k$, определить величину *БЛИЖАЙШЕЕ* (VS_v) (первоначально для каждого множества данная величина может быть взята как число, большее максимальной из стоимостей ребер графа G).
- Определить для каждого множества $VS_v, v = 1,\ldots,k$, величину *РЕБРО* (VS_v) (первоначально все элементы *РЕБРО* (VS_v) есть пустые множества).
- Для каждого ребра $(v, w) \in E$ выполняются следующие действия:
  - *имя1* : = *НАЙТИ*(v);
  - *имя2* : = *НАЙТИ*(w);
  - если $c(v, w) < БЛИЖАЙШЕЕ(имя1)$, то

```
begin
  БЛИЖАЙШЕЕ(имя1) : = c(v, w);
  РЕБРО(имя1) : = (v, w);
end;
```

  - если $c(v, w) \le БЛИЖАЙШЕЕ(имя2)$, то

```
begin
  БЛИЖАЙШЕЕ(имя2) : = c(v, w);
  РЕБРО(имя2) : = (v, w);
end;
```

- После того как просмотрены все ребра $(v, w) \in E$, получен набор ребер $E' = \{РЕБРО(1), РЕБРО(2), \ldots, РЕБРО(n)\}$. Производим слияние множеств VS_v в соответствии с ребрами из E': для каждого ребра $(v, w) \in E'$, если $НАЙТИ(v) \neq НАЙТИ(w)$, то выполнить операцию *ОБЪЕДИНИТЬ*(*НАЙТИ*(v), *НАЙТИ*(w), *НАЙТИ*(v)) и уменьшить количество непустых множеств k на единицу.
- Положить минимальное остовное дерево $T = T \cup E'$.

Пример 4.24. Для графа, приведенного на рис. 4.68, построить минимальное остовное дерево, используя алгоритм 2.

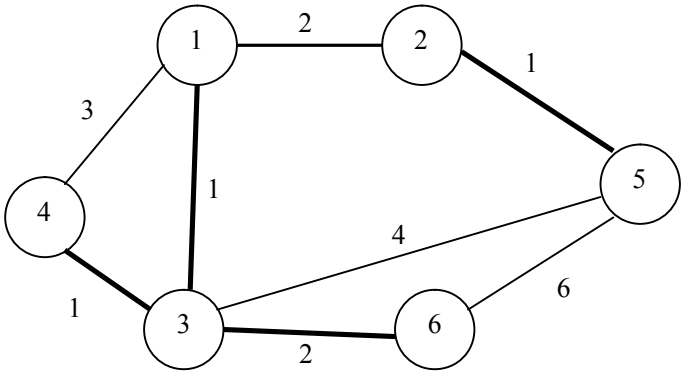


Рис. 4.68

Решение. Приведем последовательность итераций алгоритма 2 построения минимального остовного дерева. Отметим, что на третьем шаге алгоритма ребра берутся в произвольном порядке.

1-*я итерация*

Множества: $VS = \left(1 = \{1\}, 2 = \{2\}, 3 = \{3\}, 4 = \{4\}, 5 = \{5\}, 6 = \{6\}\right)$. Минимальное остовное дерево T – пустое.

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	–	–	–	–	–	–
БЛИЖАЙШЕЕ	7	7	7	7	7	7

$c(1,2) = 2;$

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	(1, 2)	(1, 2)	–	–	–	–
БЛИЖАЙШЕЕ	2	2	7	7	7	7

$c(1,4) = 3;$

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	(1, 2)	(1, 2)	–	(1, 4)	–	–
БЛИЖАЙШЕЕ	2	2	7	3	7	7

$c(1,3) = 1;$

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	(1, 3)	(1, 2)	(1, 3)	(1, 4)	–	–
БЛИЖАЙШЕЕ	1	2	1	3	7	7

$c(2,5) = 1;$

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	(1, 3)	(2, 5)	(1, 3)	(1, 4)	(2, 5)	–
БЛИЖАЙШЕЕ	1	1	1	3	1	7

$c(3,5)=4;\ \ c(3,4)=1;$

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	(1, 3)	(2, 5)	(1, 3)	(3, 4)	(2, 5)	–
БЛИЖАЙШЕЕ	1	1	1	1	1	7

$c(3,6)=2;$

Имена множеств	**1**	**2**	**3**	**4**	**5**	**6**
РЕБРО	(1, 3)	(2, 5)	(1, 3)	(3, 4)	(2, 5)	(3, 6)
БЛИЖАЙШЕЕ	1	1	1	1	1	2

$c(5,6)=6.$

Набор ребер $E'=\{(1,3),(2,5),(1,3),(3,4),(2,5),(3,6)\}$.

Остовное дерево $T=T\cup E'=\{(1,3),(2,5),\ (3,4),\ (3,6)\}$.

Множества $VS=(1=\{1,3,4,6\},2=\{2,5\})$.

Количество непустых множеств $k=2$.

2-*я итерация*

После первой итерации получены два множества: $1=\{1,3,4,6\}$ и $2=\{2,5\}$.

Для простоты реализации наряду с полученными в результате слияний множествами (в таблице их имена выделены жирным шрифтом) будем рассматривать «фиктивные» пустые множества (имена которых исчезли в результате слияния).

Имена множеств	**1**	**2**	3	4	5	6
РЕБРО	–	–	–	–	–	–
БЛИЖАЙШЕЕ	7	7	7	7	7	7

Будем рассматривать только те ребра графа G, которые соединяют вершины, принадлежащие разным множествам.

$c(1,2)=2;$

Имена множеств	**1**	**2**	3	4	5	6
РЕБРО	(1, 2)	(1, 2)	–	–	–	–
БЛИЖАЙШЕЕ	2	2	7	7	7	7

$c(3,5)=4;\ \ c(6,5)=6.$

Набор ребер $E' = \{(1,2),(1,2),-,-,-,-\}$ (множества, обозначенные знаком минус, соответствуют «фиктивным» множествам).

Остовное дерево $T = T \cup E' = \{(1,3),(2,5),(3,4),(3,6),(1,2)\}$.

Множества $VS = (1 = \{1,2,3,4,5,6\})$. Количество непустых множеств $k = 1$, и алгоритм заканчивает работу. Дерево T является минимальным остовным деревом графа G (на рис. 4.68 ребра минимального остовного дерева выделены).

Оценим трудоемкость алгоритма 2 построения минимального остовного дерева. После выполнения шага 3 алгоритма каждому из множеств будет найдено ближайшее, после чего произойдет их слияние. Следовательно, на каждой итерации алгоритма количество множеств уменьшается, по крайней мере, вдвое. Таким образом, количество итераций алгоритма 2 есть $O(\log n)$. Поскольку трудоемкость одной итерации алгоритма есть $O(m)$, то трудоемкость алгоритма 2 построения минимального остовного дерева есть $O(m \log n)$.

4.8. КРАТЧАЙШИЙ МАРШРУТ С НЕЧЕТНЫМ ЧИСЛОМ РЕБЕР

Задан граф $G = (V, E)$. Каждому ребру графа $(v, w) \in E$ поставлена в соответствие неотрицательная стоимость $c(v, w) \geq 0$. Необходимо найти в графе G кратчайший элементарный маршрут из вершины s в вершину t с нечетным числом ребер. Наряду с исходным графом G будем рассматривать модифицированный граф G', который получается из исходного графа по следующему правилу:

- вершины s и t оставляем без изменения;
- все вершины $v \in V$, отличные от s и t, расщепляем на две вершины v и v';
- если в исходном графе существует ребро (v, w), соединяющее расщепляемые вершины v и w, то в модифицированном графе G' будут существовать ребра

$$(v, w), (v', w'), (v, w'), (v', w),$$

стоимость каждого из вышеперечисленных ребер равна стоимости ребра (v, w) в исходном графе G и ребра

$$(v, v'), (w, w')$$

стоимости 0 (рис. 4.69).

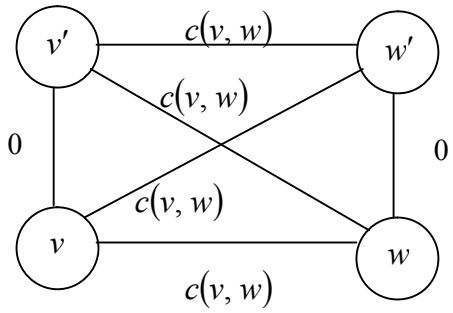


Рис. 4.69

Утверждение 1. Если в исходном графе G существует элементарный маршрут с нечетным числом ребер стоимости L, то в модифицированном графе G' существует совершенное паросочетание стоимости L. И обратно, если в модифицированном графе G' существует совершенное паросочетание стоимости L, то в исходном графе G существует элементарный маршрут с нечетным числом ребер стоимости L.

Утверждение 2. Если в исходном графе G не существует элементарного пути с нечетным числом ребер, то в модифицированном графе G' не существует совершенного паросочетания. И обратно, если в модифицированном графе G' не существует совершенного паросочетания, то в исходном графе G не существует элементарного маршрута с нечетным числом ребер.

Заметим, что в графе G' количество вершин всегда четно: вершины s и t плюс $2 \cdot (n-2)$, где n – количество вершин исходного графа. Поэтому для графа G' есть вероятность того, что в нем может существовать совершенное паросочетание (очевидно, что в графе с нечетным количеством вершин его не существует).

Находим в графе G' совершенное паросочетание минимальной стоимости. Если совершенного паросочетания не существует, то в соответствии с утверждением 2 в исходном графе не существует маршрутов нечетной длины. Если в графе G' найдено совершенное паросочетание минимальной стоимости L, то в силу утверждения 1 в исходном графе существует элементарный маршрут с нечетным числом ребер стоимости L. Данный маршрут кратчайший среди всех элементарных маршрутов с нечетным числом ребер. Для восстановления требуемого маршрута необходимо сжать в графе G' все расщепленные вершины (исчезнут ребра нулевой стоимости), и мы получим из совершенного паросочетания минимальной стоимости кратчайший элементарный маршрут с нечетным числом ребер.

Пример 4.25. Для графа G, приведенного на рис. 4.70, необходимо найти кратчайший элементарный маршрут с нечетным числом ребер.

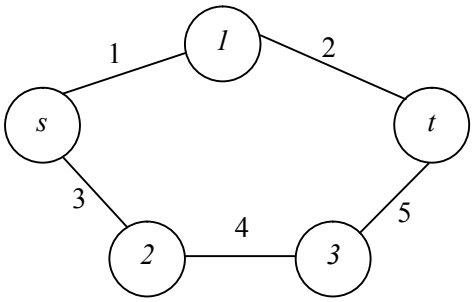


Рис. 4.70

Построим модифицированный граф G' и найдем в нем совершенное паросочетание минимального веса. На рис. 4.71 ребра паросочетания выделены (стоимость паросочетания равна 12).

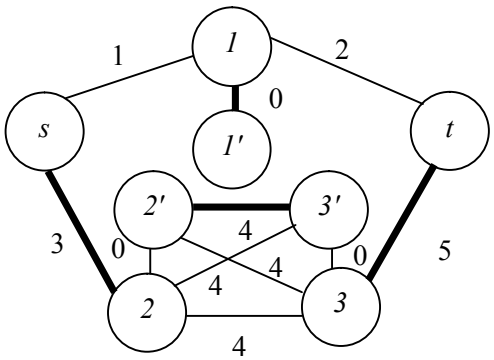


Рис. 4.71

Совершенное паросочетание минимального веса состоит из ребер $(s,2),(2',3'),(1,1'),(3,t)$. Сжимаем расщепленные вершины и получаем требуемый кратчайший элементарный маршрут с нечетным числом ребер (рис. 4.72): $(s,2),(2,3),(3,t)$ (стоимость маршрута равна 12).

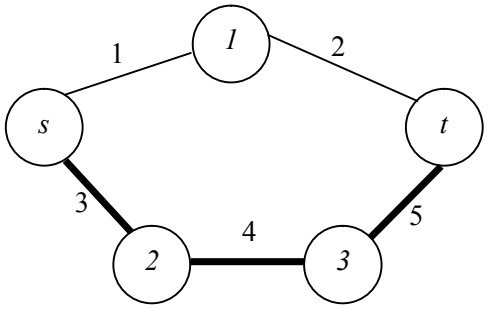


Рис. 4.72

4.9. ЗАДАЧА КИТАЙСКОГО ПОЧТАЛЬОНА

С задачей построения эйлерова цикла в неориентированном графе тесно связана следующая задача. Задан неориентированный связный граф, каждому ребру которого поставлена в соответствие некоторая неотрицательная стоимость. В графе требуется найти цикл, проходящий не менее одного раза по каждому ребру и стоимость которого минимальна. Данную задачу часто называют *задачей китайского почтальона*, так как задача доставки почты с минимизацией общего километража может быть сформулирована описанным выше образом. Несомненно, что подобным образом может быть сформулировано большое число задач, встречающихся на практике.

Рассмотрим алгоритм решения задачи китайского почтальона. Если степени всех вершин исходного графа четны, то по теореме Эйлера в связном графе существует эйлеров цикл. Данный цикл проходит по каждому ребру графа ровно один раз, следовательно, требование, предъявляемое к нужному пути, выполнено, и данный цикл является циклом минимальной стоимости.

Предположим теперь, что в графе существуют вершины, степени которых нечетны. Покажем, что количество таких вершин четно. Действительно, сумма степеней всех вершин графа (будем обозначать эту величину: $d(V)$) четная, так как каждое ребро добавляет две единицы четности к общей сумме. Если $|E| = m$, то $d(V) = 2 \cdot m$. Обозначим через X – вершины графа, имеющие четную степень, а через Y – вершины графа, имеющие нечетную степень. Тогда

$$\sum_{v_i \in X} d(v_i) + \sum_{v_i \in Y} d(v_i) = 2 \cdot m.$$

Поскольку величина $\sum_{v_i \in X} d(v_i)$ всегда четна, то для четности всей суммы необходимо, чтобы величина $\sum_{v_i \in Y} d(v_i)$ также была четна. Учитывая, что под знаком последней суммы стоят нечетные числа, следует, что их количество должно быть четным. Таким образом, мы показали, что в графе G количество вершин, степени которых нечетны, четное число.

Алгоритм решения задачи китайского почтальона

1. На множестве вершин исходного графа G, которые имели нечетную степень, построим полный граф $G' = (Y, E')$. Каждому ребру $(v, w) \in E'$ поставим в соответствие стоимость $c'(v, w) = p(v, w)$, где

$p(v,w)$ – стоимость кратчайшего элементарного пути из вершины v в вершину w в графе G. Обозначим через $Path\_min(v,w)$ ребра этого пути.

2. Поскольку в полном графе G' количество вершин четно, то в нем существует совершенное паросочетание. Найдем в графе G' совершенное паросочетание P минимальной стоимости.

3. Заменим каждое ребро $(v,w)\in P$ последовательностью ребер из множеств $Path\_min(v,w)$. Обозначим через P' объединение множеств ребер этих путей:

$$P' = \bigcup_{(v,w)\in P} Path\_min(v,w).$$

4. Построим мультиграф $G' = (V, E \cup P')$. В таком графе могут быть кратные ребра. Заметим, что степени всех вершин графа G' являются четным (степени всех вершин, которые имели в графе G нечетную степень, увеличились на 1, а степени вершин, которые имели четную степень в графе G, по-прежнему четны).

5. Таким образом, мы имеем Эйлеров граф G', в котором маршрут может быть найден по описанному в предыдущем параграфе алгоритму. Данный маршрут является решением задачи китайского почтальона.

Пример 4.26. Для графа $G = (V, E)$, приведенного на рис. 4.73, найти такой цикл, который проходит по каждому ребру графа не менее одного раза и суммарная стоимость ребер которого минимальна.

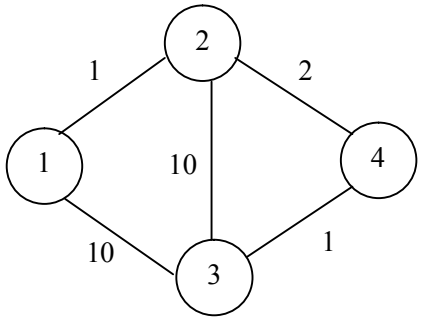


Рис. 4.73

В графе две вершины: 2 и 3 имеют нечетную степень. На рис. 4.74 приведен полный граф G'

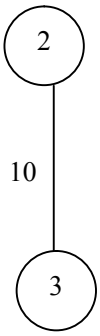


Рис. 4.74

Совершенное паросочетание минимальной стоимости для графа G' есть $P = \{(2, 3)\}$. Длина кратчайшего пути в графе G между вершинами 2 и 3 равна 3. На рис. 4.75 приведена последовательность ребер этого пути: $Path\_min(2, 3) = \{(3, 4), (4, 2)\}$.

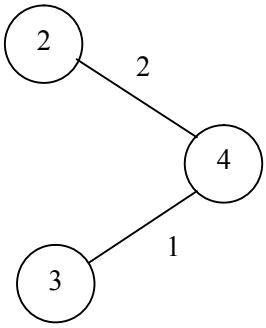


Рис. 4.75

Построим мультиграф $G' = (V, E \cup P')$ (рис. 4.76)

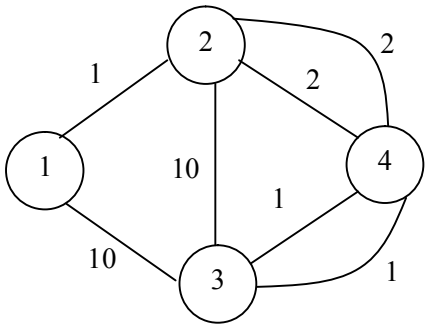


Рис. 4.76

В построенном мультиграфе G' находим эйлеров цикл. Найдем эйлеров цикл описанным в предыдущих разделах алгоритмом. Для этого сначала делаем очередь пустой, а в стек заносим вершину 1.

$$S = \{1\},$$
$$O = \{Ø\}.$$

Затем последовательно двигаемся по ребрам (1, 2), (2, 3), (3, 1), удаляем их из исходного графа и заносим в стек вершины 2, 3 и 1. После выполнения данной последовательности действий содержимое стека будет равно

$$S = \{1, 2, 3, 1\}.$$

После чего мы видим, что не существует ребер, выходящих из вершины 1, поэтому удаляем ее из стека и переносим в очередь.

$$O = \{1\},$$
$$S = \{1, 2, 3\}.$$

Затем последовательно двигаемся по ребрам (3, 4), (4, 3), удаляем их из исходного графа и заносим в стек вершины 4 и 3. После выполнения данной последовательности действий содержимое стека будет равно

$$S = \{1, 2, 3, 4, 3\}.$$

После чего мы видим, что не существует ребер, выходящих из вершины 3, поэтому удаляем ее из стека и переносим в очередь.

$$O = \{1, 3\},$$
$$S = \{1, 2, 3, 4\}.$$

Затем последовательно двигаемся по ребрам (4, 2), (2 4), удаляем их из исходного графа и заносим в стек вершины 2 и 4. После выполнения данной последовательности действий содержимое стека будет равно

$$S = \{1, 2, 3, 4, 2, 4\}.$$

После чего в графе вообще нет ребер, поэтому удаляем последовательно все элементы из стека и переносим их в очередь:

$$O = \{1, 3, 4, 2, 4, 3, 2, 1\}.$$

Последовательность вершин очереди и задает порядок обхода вершин в эйлеровом цикле в графе G'.

Таким образом, цикл, проходящий по каждому ребру графа G не менее одного раза и суммарная стоимость ребер которого минимальна, задается последовательностью вершин {1, 3, 4, 2, 4, 3, 2, 1}. Стоимость данного цикла равна 27.

4.10. ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1. *Шестеренки*. Заданы *n* шестеренок, которым присвоены номера – числа от 1 до *n*. Задано *m* соединений пар шестеренок в виде пар (i, j) (шестеренка с номером *i* находится в зацеплении с шестеренкой с номером *j*). *Необходимо* определить, можно ли повернуть шестеренку с номером 1. Если «да», то найти количество шестеренок, которые пришли при этом в движение.

2. *Открытки*. Имеется *n* прямоугольных конвертов и *n* прямоугольных открыток различных размеров. *Необходимо* определить, можно ли разложить все открытки по конвертам так, чтобы в каждом конверте было по одной открытке. Открытки нельзя складывать, но можно помещать в конверт под углом. Например, открытка с размерами сторон $5:1$ помещается в конверты с размерами $5:1$, $6:3$, но не входит в конверты с размерами $4:1$, $10:0{,}5$, $4{,}2:4{,}2$.

3. *Разбиение на команды по численности*. Имеется группа из *n* студентов. Некоторые из студентов знают друг друга. Круг знакомств задается матрицей *A* размером $n \times n$. Элемент матрицы $A[i, j] = 1$, если *i*-й студент знаком с *j*-м, и $A[i, j] = 0$ в противном случае. *Необходимо* определить, можно ли произвольно разбить *n* студентов на 2 команды, численность которых отличается не более чем в 2 раза, если известно, что в любой команде должны быть студенты, которые обязательно знакомы друг с другом.

4. *Олимпиада.* На олимпиаду прибыло *n* человек. Некоторые из них знакомы между собой. Круг знакомств задается матрицей *A* размером $n \times n$. Элемент матрицы $A[i, j] = 1$, если *i*-й человек знает *j*-го, и $A[i, j] = 0$, в противном случае (если *i*-ый человек знает *j*-го, то считаем, что и *j*-ый человек знает *i*-го). *Необходимо* определить, можно ли опосредованно перезнакомить всех людей между собой (незнакомые люди могут познакомиться только через общего знакомого). Если нет, то какое максимальное количество людей будет знать друг друга?

5. *Станки.* Задано *n* различных станков, которые один за другим связаны в конвейер. Имеется *n* рабочих. Задана матрица *C* размером $n \times n$, где элемент матрицы $C[i, j]$ задает производительность *i*-го рабочего на *j*-м станке (если рабочий *i* не может работать на *j*-м станке, то значение элемента матрицы $C[i, j]$ равно 0). *Необходимо* определить, каким должно быть распределение рабочих по станкам (каждый рабочий может быть назначен только на один станок, и на каждом станке может рабо-

тать только один рабочий), чтобы производительность конвейера была максимальной.

6. *Встреча*. Вводятся два числа: *n* – количество домиков и *k* – количество дорог. Домики пронумерованы целыми числами от 1 до *n.* Каждая дорога определяется тройкой чисел: двумя номерами домиков, которые являются концами этой дороги, и длиной дороги (длины дорог – положительные целые числа). В каждом домике живет по одному человеку. *Необходимо* найти точку (место встречи всех людей), от которой суммарное расстояние до всех домиков будет минимальным. Если точка лежит на дороге, то указать номера домиков, которые являются концами этой дороги, и расстояние от первого из этих домиков. Если точка совпадает с домиком, то указать его номер.

7. *Сеть дорог*. Сеть дорог определяется следующим образом. Имеется *n* перекрестков, занумерованных целыми числами от 1 до *n*, и *k* дорог, связывающих перекрестки. Каждая дорога определяется тройкой чисел: двумя номерами перекрестков и временем, требующимся на проезд по этой дороге. Движение по дороге возможно в обоих направлениях. *Необходимо* найти кратчайший по времени маршрут машины от перекрестка с номером *i* до перекрестка с номером *j*, если на перекрестке машина должна ждать время, равное числу пересекающихся в этом перекрестке дорог (для преодоления начального (*i*) и конечного (*j*) перекрестков машина также должна ждать).

8. *Хакеры*. Компьютерная сеть состоит из связанных между собой больших ЭВМ, к каждой из которых подключается несколько терминалов (в сети ни у каких двух ЭВМ количество терминалов не совпадает). Подключение к одной из больших ЭВМ позволяет получить информацию, содержащуюся в памяти этой ЭВМ, а также всю информацию, доступную для ЭВМ, к которым данная ЭВМ могла направлять запросы. Необходимо защитить компьютерную сеть от хакеров, которые выкачивают из компьютеров секретную информацию. Хакеры и раньше нападали на подобные компьютерные сети, и их тактика была известна. Тактика хакеров: при нападении они всегда получают доступ к информации всех ЭВМ сети. Добиваются они этого, захватывая некоторые ЭВМ сети так,

чтобы от них можно было запросить информацию, которая имеется у оставшихся ЭВМ. Существует много способов захвата, например захватить все ЭВМ. Однако хакеры всегда выбирают такой вариант, при котором суммарное количество терминалов у захваченных ЭВМ минимально. *Необходимо* определить список номеров ЭВМ, которые могут быть выбраны хакерами для захвата сети согласно их тактике.

9. *Пирамида Хеопса*. Внутри пирамиды Хеопса есть n комнат, в которых установлено $2m$ модулей, составляющих m устройств. Каждое устройство состоит из двух модулей, которые располагаются в разных комнатах и предназначены для перемещения между парой комнат, в которых они установлены. Перемещение происходит за 0,5 единиц времени. В начальный момент времени модули всех устройств переходят в подготовительный режим. Каждый из модулей имеет некоторый свой целочисленный период времени, в течение которого он находится в подготовительном режиме. По истечении этого времени модуль мгновенно включается, после чего опять переходит в подготовительный режим. Устройством можно воспользоваться только в тот момент, когда одновременно включаются оба его модуля. Преступник сумел проникнуть в гробницу фараона. Обследовав ее, он включил устройства и собрался уходить, но в этот момент проснулся хранитель гробницы. Теперь преступнику необходимо как можно быстрее попасть из комнаты номер 1 в комнату с номером n, в которой находится выход из пирамиды. При этом из комнаты в комнату он может попадать только при помощи устройств, так как проснувшийся хранитель закрыл все двери в комнатах пирамиды. *Необходимо* написать программу, которая получает на входе описание расположения устройств и их характеристик, а выдает значение оптимального времени и последовательность устройств, предназначенных для того, чтобы попасть из комнаты номер 1 в комнату номер n за это время.

10. *Без левых поворотов*. План города представляет множество перекрестков, соединенных дорогами. Перекрестки обозначаются точками на плоскости с координатами $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, а дорогам соответствуют отрезки (i, j), где i и j обозначают номера перекрестков, со-

единенные этой дорогой. *Необходимо* определить, можно ли проехать от перекрестка с номером *s* до перекрестка с номером *f*, не делая левых поворотов, т. е. двигаясь только прямо или вправо. Если да, то указать последовательность перекрестков. Первоначально мы будем предполагать, что если у перекрестка *s* координаты (x_s, y_s), то машина подъехала к этому перекрестку из точки с координатами $(x_s, y_s - 1)$.

11. *Прогулка.* Хозяин вышел на прогулку с собакой. Известно, что путь хозяина представляет собой ломаную линию, координаты отрезков ломаной заданы: (x_i, y_i), $i = 1,\ldots,n$. Точка (x_1, y_1) – начальная точка ломаной линии, а точка (x_n, y_n) – конечная точка ломаной. Хозяин двигается во время прогулки от стартовой точки, далее – по отрезкам ломаной линии и заканчивает путь в конечной точке ломаной. У собаки есть свои любимые места с координатами $(xdog_j, ydog_j)$, $j = 1,\ldots,m$, которые собака хотела бы посетить. В то время, пока хозяин проходит один отрезок ломаной линии, собака может посетить только одно из своих любимых мест. В начальной и конечной координате каждого отрезка ломаной собака обязана подбежать к хозяину. Известно, что скорость собаки в два раза выше скорости хозяина. *Необходимо* определить, какое наибольшее количество своих любимых мест и в какой последовательности сможет посетить собака за время прогулки.

12. *Трубопроводы.* Имеется *n* пунктов, которые занумерованы целыми числами от 1 до *n,* и сеть трубопроводов по перекачке нефти из пункта *A* в пункт *B*. Для каждого участка трубопровода указаны пункты, которые он соединяет, пропускная способность (тонна/час) и стоимость транспортировки (транзита) тонны нефти по участку. *Необходимо* организовать перекачку максимального количества нефти при минимальных суммарных затратах на транзит из пункта *A* в пункт *B*.

ЛИТЕРАТУРА

Ахо, *А*. *В*. Структуры данных и алгоритмы: Учеб. пособие //А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман : пер. с англ. М.: Вильямс, 2000. 384 с.

Волчкова, *Г. П.* Сборник задач по теории алгоритмов для студентов физико-математических спец. БГУ // Г. П. Волчкова, В. М. Котов, Е. П. Соболевская – Минск: БГУ, 2005. 59 с.

Волчкова, *Г. П.* Сборник задач по теории алгоритмов. Организация перебора вариантов и приближенные алгоритмы: для студентов спец. 1-31 03 04 «Информатика» // Г. П. Волчкова, В. М. Котов, Е. П. Соболевская – Минск: БГУ, 2008. 59 с.

Кормен, *Т.* Алгоритмы: построение и анализ// Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн – Москва : Вильямс, 2005. 1296 с.

Котов, *В. М.* Структуры данных и алгоритмы: теория и практика: Учеб. пособие // В. М. Котов, Е. П. Соболевская – Минск: БГУ, 2004. 255 с.

Котов, *В. М.* Разработка и анализ алгоритом: теория и практика: пособие для студентов мат. и физ. специальностей // В. М. Котов, Е. П. Соболевская – Минск: БГУ, 2009. 251 с.