Лекции по дискретной математике

Лекция 14. Элементы теории графов

Крохин А.Л.

УрФУ—РИ-РтФ

01.09.2010

Определение:

Обыкновенным графом или просто *графом* G называется пара (V, E), где V — непустое конечное множество, элементы которого называют *вершинами*,

Определение:

Обыкновенным графом или просто *графом* G называется пара (V, E), где V — непустое конечное множество, элементы которого называют *вершинами*, а E — конечное семейство неупорядоченных пар вершин, называемых *ребрами*.

Определение:

Обыкновенным графом или просто *графом* G называется пара (V, E), где V — непустое конечное множество, элементы которого называют *вершинами*, а E — конечное семейство неупорядоченных пар вершин, называемых *ребрами*.

Если ребро $e = \{u, v\}$ для некоторых вершин u, v графа G, то говорят, что ребро e *инцидентно* вершинам u, v и каждая из вершин u, v *инцидентна* ребру e.

Определение:

Обыкновенным графом или просто *графом* G называется пара (V, E), где V — непустое конечное множество, элементы которого называют *вершинами*, а E — конечное семейство неупорядоченных пар вершин, называемых *ребрами*.

Если ребро $e = \{u, v\}$ для некоторых вершин u, v графа G, то говорят, что ребро e *инцидентно* вершинам u, v и каждая из вершин u, v *инцидентна* ребру e.

Паре вершин из V $e = \{v, v\}$ соответствует *петля*, если граф содержит петли, то его называют – *псевдограф*.

Определение:

Обыкновенным графом или просто *графом* G называется пара (V, E), где V — непустое конечное множество, элементы которого называют *вершинами*, а E — конечное семейство неупорядоченных пар вершин, называемых *ребрами*.

Если ребро $e = \{u, v\}$ для некоторых вершин u, v графа G, то говорят, что ребро e *инцидентно* вершинам u, v и каждая из вершин u, v *инцидентна* ребру e.

Паре вершин из V $e = \{v, v\}$ соответствует *петля*, если граф содержит петли, то его называют – *псевдограф*.

В некоторых прикладных задачах естественным образом возникают *кратные* ребра, т.е. одинаковые пары вершин встречаются многократно, такой граф в литературе называют *мультиграфом*.

Для графа G через $V(G)$ мы будем обозначать множество всех его вершин, а через $E(G)$ — множество всех его ребер.

Для графа G через $V(G)$ мы будем обозначать множество всех его вершин, а через $E(G)$ — множество всех его ребер.

Количество вершин, $|V|$ – **порядок** графа, количество ребер, $|E|$ – **размер** графа.

Для графа G через $V(G)$ мы будем обозначать множество всех его вершин, а через $E(G)$ — множество всех его ребер.

Количество вершин, $|V|$ – **порядок** графа, количество ребер, $|E|$ – **размер** графа.

Определение:

Граф $G' = (V', E')$ называется *подграфом* графа G, если $V' \subseteq V$, $E' \subseteq E$.

Для графа G через $V(G)$ мы будем обозначать множество всех его вершин, а через $E(G)$ — множество всех его ребер.

Количество вершин, $|V|$ – **порядок** графа, количество ребер, $|E|$ – **размер** графа.

Определение:

Граф $G' = (V', E')$ называется *подграфом* графа G, если $V' \subseteq V$, $E' \subseteq E$.

Заметим, что это определение неявно подразумевает, что V' и E' связаны между собой, множество ребер подграфа E' состоит из двухэлементных подмножеств множества V'.

Определение:

Маршрут в графе G — это чередующаяся последовательность вершин и ребер, начинающаяся и кончающаяся вершиной, в которой каждое ребро инцидентно двум вершинам: непосредственно предшествующей ему и непосредственно следующей за ним.

Определение:

Маршрут в графе G — это чередующаяся последовательность вершин и ребер, начинающаяся и кончающаяся вершиной, в которой каждое ребро инцидентно двум вершинам: непосредственно предшествующей ему и непосредственно следующей за ним.

Определение:

Граф G называется ориентированным или *орграфом* (V, E), где V — множество вершин, а E — конечное множество упорядоченных пар вершин, называемых *дугами*.

Определение:

Маршрут в графе G — это чередующаяся последовательность вершин и ребер, начинающаяся и кончающаяся вершиной, в которой каждое ребро инцидентно двум вершинам: непосредственно предшествующей ему и непосредственно следующей за ним.

Определение:

Граф G называется ориентированным или *орграфом* (V, E), где V — множество вершин, а E — конечное множество упорядоченных пар вершин, называемых *дугами*. Дуга (v_1, v_2) *исходит* из вершины v_1 и *заходит* в вершину v_2.

Определение:

Маршрут в графе G — это чередующаяся последовательность вершин и ребер, начинающаяся и кончающаяся вершиной, в которой каждое ребро инцидентно двум вершинам: непосредственно предшествующей ему и непосредственно следующей за ним.

Определение:

Граф G называется ориентированным или *орграфом* (V, E), где V — множество вершин, а E — конечное множество упорядоченных пар вершин, называемых *дугами*. Дуга (v_1, v_2) *исходит* из вершины v_1 и *заходит* в вершину v_2.

Маршрут часто называют путем. Он может быть замкнутым, если начальная и конечная вершины совпадают, и открытым.

Определение:

Маршрут в графе G — это чередующаяся последовательность вершин и ребер, начинающаяся и кончающаяся вершиной, в которой каждое ребро инцидентно двум вершинам: непосредственно предшествующей ему и непосредственно следующей за ним.

Определение:

Граф G называется ориентированным или *орграфом* (V, E), где V — множество вершин, а E — конечное множество упорядоченных пар вершин, называемых *дугами*. Дуга (v_1, v_2) *исходит* из вершины v_1 и *заходит* в вершину v_2.

Маршрут часто называют путем. Он может быть замкнутым, если начальная и конечная вершины совпадают, и открытым.

Цепь - это путь без повторяющихся ребер (дуг), простая цепь - путь без повторяющихся вершин.

Определение:

Количество ребер, инцидентных вершине v, называют *степенью* вершины, $d(v)$.

Определение:

Количество ребер, инцидентных вершине v, называют *степенью* вершины, $d(v)$.

Две вершины графа, инцидентных одному ребру, называют смежными.

Определение:

Количество ребер, инцидентных вершине v, называют *степенью* вершины, $d(v)$.

Две вершины графа, инцидентных одному ребру, называют смежными.

Граф называют *пустым*, если множество ребер пустое, и *полным*, если каждая вершина смежна с любой другой вершиной.

Определение:

Количество ребер, инцидентных вершине v, называют *степенью* вершины, $d(v)$.

Две вершины графа, инцидентных одному ребру, называют смежными.

Граф называют *пустым*, если множество ребер пустое, и *полным*, если каждая вершина смежна с любой другой вершиной.

Полный граф обозначают K_n.

Определение:

Количество ребер, инцидентных вершине v, называют *степенью* вершины, $d(v)$.

Две вершины графа, инцидентных одному ребру, называют смежными.

Граф называют *пустым*, если множество ребер пустое, и *полным*, если каждая вершина смежна с любой другой вершиной.

Полный граф обозначают K_n.

Определение:

Для орграфа количество дуг, исходящих из вершины v — полустепень исхода, а количество заходящих дуг — полустепень захода.

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Вершины графа изображаются в виде точек, а ребра в виде отрезков или дуг, их соединяющих.

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Вершины графа изображаются в виде точек, а ребра в виде отрезков или дуг, их соединяющих.

Например, на рис. 1 изображен граф $G = (V, E)$, где

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Вершины графа изображаются в виде точек, а ребра в виде отрезков или дуг, их соединяющих.

Например, на рис. 1 изображен граф $G = (V, E)$, где

$$V = \{1, 2, \ldots, 7\},$$

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Вершины графа изображаются в виде точек, а ребра в виде отрезков или дуг, их соединяющих.

Например, на рис. 1 изображен граф $G = (V, E)$, где

$$V = \{1, 2, \ldots, 7\},$$
$$E = \{\{1,2\}, \{1,6\}, \{1,7\}, \{2,3\}, \{2,7\}, \{3,4\}, \{3,7\}, \{4,5\},$$

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Вершины графа изображаются в виде точек, а ребра в виде отрезков или дуг, их соединяющих.

Например, на рис. 1 изображен граф $G = (V, E)$, где

$$V = \{1, 2, \ldots, 7\},$$
$$E = \{\{1,2\}, \{1,6\}, \{1,7\}, \{2,3\}, \{2,7\}, \{3,4\}, \{3,7\}, \{4,5\}, \{4,6\}, \{4,7\}, \{5,6\}, \{5,7\}, \{6,7\}\}.$$

Граф $G = (V, E)$ представляют в виде диаграммы, которую часто также называют графом.

Вершины графа изображаются в виде точек, а ребра в виде отрезков или дуг, их соединяющих.

Например, на рис. 1 изображен граф $G = (V, E)$, где

$$V = \{1, 2, \ldots, 7\},$$
$$E = \{\{1,2\},\{1,6\},\{1,7\},\{2,3\},\{2,7\},\{3,4\},\{3,7\},\{4,5\},$$
$$\{4,6\},\{4,7\},\{5,6\},\{5,7\},\{6,7\}\}.$$

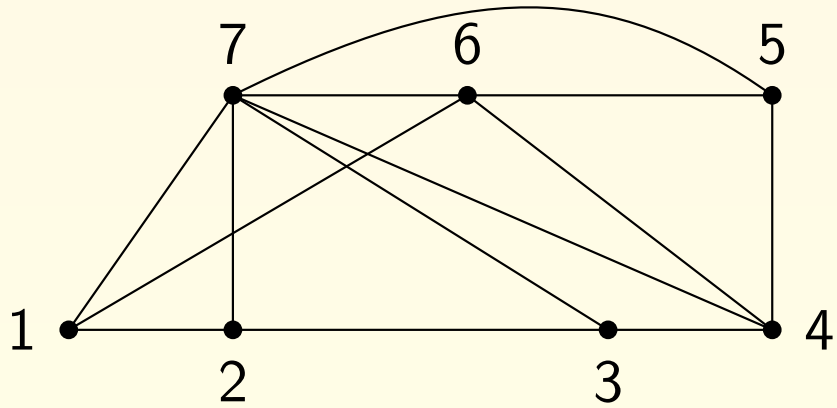


рис. 1. Простой связный граф

Имеет место следующее утверждение (лемма о рукопожатиях).

Имеет место следующее утверждение (лемма о рукопожатиях).

Теорема

В графе $G = (V, E)$, сумма степеней всех вершин в два раза больше числа ребер. Это можно записать так:

Имеет место следующее утверждение (лемма о рукопожатиях).

Теорема

В графе $G = (V, E)$, сумма степеней всех вершин в два раза больше числа ребер. Это можно записать так:

$$\sum_{v \in V} d(v) = 2|E|$$

Имеет место следующее утверждение (лемма о рукопожатиях).

Теорема

В графе $G = (V, E)$, сумма степеней всех вершин в два раза больше числа ребер. Это можно записать так:

$$\sum_{v \in V} d(v) = 2|E|$$

Действительно, рассмотрим пустой граф $G^0 \in G$, степени всех вершин равны нулю.

Имеет место следующее утверждение (лемма о рукопожатиях).

Теорема

В графе $G = (V, E)$, сумма степеней всех вершин в два раза больше числа ребер. Это можно записать так:

$$\sum_{v \in V} d(v) = 2|E|$$

Действительно, рассмотрим пустой граф $G^0 \in G$, степени всех вершин равны нулю.

Добавление к пустому графу ребра увеличивает степени двух инцидентных ему вершин на единицу каждой.

Имеет место следующее утверждение (лемма о рукопожатиях).

Теорема

В графе $G = (V, E)$, сумма степеней всех вершин в два раза больше числа ребер. Это можно записать так:

$$\sum_{v \in V} d(v) = 2|E|$$

Действительно, рассмотрим пустой граф $G^0 \in G$, степени всех вершин равны нулю.

Добавление к пустому графу ребра увеличивает степени двух инцидентных ему вершин на единицу каждой.

Так будет происходить при добавлении и остальных ребер из $E(G)$.

Имеет место следующее утверждение (лемма о рукопожатиях).

Теорема

В графе $G = (V, E)$, сумма степеней всех вершин в два раза больше числа ребер. Это можно записать так:

$$\sum_{v \in V} d(v) = 2|E|$$

Действительно, рассмотрим пустой граф $G^0 \in G$, степени всех вершин равны нулю.

Добавление к пустому графу ребра увеличивает степени двух инцидентных ему вершин на единицу каждой.

Так будет происходить при добавлении и остальных ребер из $E(G)$.

Получается искомое соотношение.

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.
Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные,

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.
Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные, и 0 — в противном случае.

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.

Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные, и 0 — в противном случае.

$a_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$; $a_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$.

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.

Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные, и 0 — в противном случае.

$a_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$; $a_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$.

В памяти содержимое матрицы смежности представляется битовыми переменными и занимает мало места.

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.

Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные, и 0 — в противном случае.

$a_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$; $a_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$.

В памяти содержимое матрицы смежности представляется битовыми переменными и занимает мало места.

Множество ребер в этом случае определяется косвенным образом.

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.

Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные, и 0 — в противном случае.

$a_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$; $a_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$.

В памяти содержимое матрицы смежности представляется битовыми переменными и занимает мало места.

Множество ребер в этом случае определяется косвенным образом.

Для графа, изображенного на рис.(2), матрица смежности имеет вид

Матрица смежности — это квадратная матрица, строки и столбцы которой нумеруются вершинами графа.

Матричный элемент с номером (v_i, v_j) равен 1, если вершины v_i, v_j смежные, и 0 — в противном случае.

$a_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$; $a_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$.

В памяти содержимое матрицы смежности представляется битовыми переменными и занимает мало места.

Множество ребер в этом случае определяется косвенным образом.

Для графа, изображенного на рис.(2), матрица смежности имеет вид

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

Для некоторых задач удобнее матрица инцидентности.

Для некоторых задач удобнее матрица инцидентности.

Это прямоугольная матрица $n \times m$, строки которой соответствуют ребрам, $n = |E|$, а столбцы — вершинам, $m = |V|$.

Для некоторых задач удобнее матрица инцидентности.

Это прямоугольная матрица $n \times m$, строки которой соответствуют ребрам, $n = |E|$, а столбцы — вершинам, $m = |V|$.

Ребра пронумерованы, в соответствующей строке стоит битовый (или логический) символ, отмечающий инцидентные вершины.

Для некоторых задач удобнее матрица инцидентности.

Это прямоугольная матрица $n \times m$, строки которой соответствуют ребрам, $n = |E|$, а столбцы — вершинам, $m = |V|$.

Ребра пронумерованы, в соответствующей строке стоит битовый (или логический) символ, отмечающий инцидентные вершины.

Для орграфа (-1) — исходящая вершина, (+1) — заходящая.

Используется еще список(таблица, структура) смежности(adjacency lists).

Используется еще список(таблица, структура) смежности(adjacency lists).

В этой таблице против каждой вершины(в колонку или в строку) записывается список смежных с ней вершин.

Используется еще список(таблица, структура) смежности(adjacency lists).

В этой таблице против каждой вершины(в колонку или в строку) записывается список смежных с ней вершин.

Списки смежности занимают в памяти значительно меньше места, чем, например, матрица смежности, если в графе высокого порядка сравнительно мало ребер (небольшой размер).

Используется еще список(таблица, структура) смежности(adjacency lists).

В этой таблице против каждой вершины(в колонку или в строку) записывается список смежных с ней вершин.

Списки смежности занимают в памяти значительно меньше места, чем, например, матрица смежности, если в графе высокого порядка сравнительно мало ребер (небольшой размер).

Для графа, изображенного на рис.(3), списки смежности имеют вид:

Используется еще список(таблица, структура) смежности(adjacency lists).

В этой таблице против каждой вершины(в колонку или в строку) записывается список смежных с ней вершин.

Списки смежности занимают в памяти значительно меньше места, чем, например, матрица смежности, если в графе высокого порядка сравнительно мало ребер (небольшой размер).

Для графа, изображенного на рис.(3), списки смежности имеют вид:

1	2	3	4	5	6	7
2	1	2	3	4	1	1
6	3	4	5	6	4	2
7	7	7	6	7	5	3
			7		7	4
						5
						6

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Связность может рассматриваться как бинарное отношение на множестве вершин графа.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Связность может рассматриваться как бинарное отношение на множестве вершин графа.

Это отношение, очевидно, рефлексивно и симметрично.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Связность может рассматриваться как бинарное отношение на множестве вершин графа.

Это отношение, очевидно, рефлексивно и симметрично.

Кроме того, можно легко убедиться, что это отношение транзитивно.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Связность может рассматриваться как бинарное отношение на множестве вершин графа.

Это отношение, очевидно, рефлексивно и симметрично.

Кроме того, можно легко убедиться, что это отношение транзитивно.

Для этого достаточно "склеить" цепь, соединяющую вершины a и b, с цепью, соединяющую вершины b и c.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Связность может рассматриваться как бинарное отношение на множестве вершин графа.

Это отношение, очевидно, рефлексивно и симметрично.

Кроме того, можно легко убедиться, что это отношение транзитивно.

Для этого достаточно "склеить" цепь, соединяющую вершины a и b, с цепью, соединяющую вершины b и c.

Таким образом, отношение связности как эквивалентность разбивает множество вершин на классы.

Определение:

Граф называется *связным*, если любые две его вершины соединяются простой цепью.

Компонента связности графа — это максимальный по включению его связный подграф.

Связность может рассматриваться как бинарное отношение на множестве вершин графа.

Это отношение, очевидно, рефлексивно и симметрично.

Кроме того, можно легко убедиться, что это отношение транзитивно.

Для этого достаточно "склеить" цепь, соединяющую вершины a и b, с цепью, соединяющую вершины b и c.

Таким образом, отношение связности как эквивалентность разбивает множество вершин на классы.

Компоненты связности можно определить как подграфы, состоящие из эквивалентных вершин и связывающих их ребер основного графа.

Определение:

Циклом называется замкнутая цепь, а простым циклом — замкнутый маршрут, все вершины которого (числом не менее трех) различны.

Определение:

Циклом называется замкнутая цепь, а простым циклом — замкнутый маршрут, все вершины которого (числом не менее трех) различны.

Определение:

Дерево — это связный граф, в котором отсутствуют циклы (*ациклический*).

Определение:

Циклом называется замкнутая цепь, а простым циклом — замкнутый маршрут, все вершины которого (числом не менее трех) различны.

Определение:

Дерево — это связный граф, в котором отсутствуют циклы (*ациклический*).

Определим на множестве ребер графа числовую функцию.

Определение:

Циклом называется замкнутая цепь, а простым циклом — замкнутый маршрут, все вершины которого (числом не менее трех) различны.

Определение:

Дерево — это связный граф, в котором отсутствуют циклы (*ациклический*).

Определим на множестве ребер графа числовую функцию.

Такой граф называют *взвешенным*.

Определение:

Циклом называется замкнутая цепь, а простым циклом — замкнутый маршрут, все вершины которого (числом не менее трех) различны.

Определение:

Дерево — это связный граф, в котором отсутствуют циклы (*ациклический*).

Определим на множестве ребер графа числовую функцию.

Такой граф называют *взвешенным*.

Значения функции — веса ребер, суммарный вес всех ребер графа — вес графа.

Пусть $G = (V, E)$ — связный взвешенный граф.

Пусть $G = (V, E)$ — связный взвешенный граф.

Определение:

Остовом графа G называется его подграф $\Gamma = (V, E')$, где $E' \subseteq E$, являющийся деревом.

Пусть $G = (V, E)$ — связный взвешенный граф.

Определение:

Остовом графа G называется его подграф $\Gamma = (V, E')$, где $E' \subseteq E$, являющийся деревом.

Определение:

Минимальным остовом графа G называется его остов наименьшего веса.

Теорема Кэли.

Количество различных деревьев с n именованными вершинами равно n^{n-2}.

Теорема Кэли.

Количество различных деревьев с n именованными вершинами равно n^{n-2}.

Простой перебор при построении минимального остова при 10 вершинах потребует 10^8 вариантов.

Теорема Кэли.

Количество различных деревьев с n именованными вершинами равно n^{n-2}.

Простой перебор при построении минимального остова при 10 вершинах потребует 10^8 вариантов.

Оценка на основе этой теоремы показывает, насколько неэффективен простой перебор.

Алгоритм Крускала построения минимального остова

Построение минимального остова начинаем с пустого подграфа, содержащего все вершины, количество компонент связности равно порядку графа.

Алгоритм Крускала построения минимального остова

Построение минимального остова начинаем с пустого подграфа, содержащего все вершины, количество компонент связности равно порядку графа.

Все ребра исходного графа $G = (V, E)$ не окрашены.

Алгоритм Крускала построения минимального остова

Построение минимального остова начинаем с пустого подграфа, содержащего все вершины, количество компонент связности равно порядку графа.

Все ребра исходного графа $G = (V, E)$ не окрашены.

Пусть $w(e)$ – вес ребра e.

Алгоритм Крускала построения минимального остова

Построение минимального остова начинаем с пустого подграфа, содержащего все вершины, количество компонент связности равно порядку графа.

Все ребра исходного графа $G = (V, E)$ не окрашены.

Пусть $w(e)$ – вес ребра e.

Упорядочим ребра $e_1, \ldots, e_m$ графа по возрастанию их весов,

Алгоритм Крускала построения минимального остова

Построение минимального остова начинаем с пустого подграфа, содержащего все вершины, количество компонент связности равно порядку графа.

Все ребра исходного графа $G = (V, E)$ не окрашены.

Пусть $w(e)$ – вес ребра e.

Упорядочим ребра $e_1, \ldots, e_m$ графа по возрастанию их весов,

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$$

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots; E_0 = \emptyset$.

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots; E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Остановка работы произойдет тогда, когда $|E_i| = n - 1$.

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Остановка работы произойдет тогда, когда $|E_i| = n - 1$. На каждом шаге получаем дерево.

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Остановка работы произойдет тогда, когда $|E_i| = n - 1$. На каждом шаге получаем дерево. Первое условие означает, что число компонент связности уменьшается на единицу.

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Остановка работы произойдет тогда, когда $|E_i| = n - 1$. На каждом шаге получаем дерево. Первое условие означает, что число компонент связности уменьшается на единицу. Какое из эквивалентных условий проверять, зависит от "исполнителя" алгоритма.

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Остановка работы произойдет тогда, когда $|E_i| = n - 1$. На каждом шаге получаем дерево. Первое условие означает, что число компонент связности уменьшается на единицу. Какое из эквивалентных условий проверять, зависит от "исполнителя" алгоритма.

Решение вручную учебных задач легче выполнять, ориентируясь на изменение числа компонент связности.

Алгоритм построения минимального остова

Алгоритм последовательно строит множества ребер $E_0, E_1, E_2, \ldots;\ E_0 = \emptyset$.

Очередной шаг перехода от E_{i-1} к E_i выполняется так:

$$E_i = \left\{ \begin{array}{cc} E_{i-1} \cup \{e_i\} & \text{если } E_{i-1} \cup \{e_i\} \text{не содержит циклов,} \\ E_{i-1} & \text{в противном случае.} \end{array} \right\}$$

Остановка работы произойдет тогда, когда $|E_i| = n - 1$. На каждом шаге получаем дерево. Первое условие означает, что число компонент связности уменьшается на единицу. Какое из эквивалентных условий проверять, зависит от "исполнителя" алгоритма.

Решение вручную учебных задач легче выполнять, ориентируясь на изменение числа компонент связности.

Алгоритм заканчивает свою работу, когда число компонент связности будет равно единице, т. е. дерево будет остовом.

Протокол работы алгоритма

Шаг 1.

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

$\Gamma_i = (\{u_1, v_1\}, \{e_1\})$.

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

$\Gamma_i = (\{u_1, v_1\}, \{e_1\})$.

Присвоить $i := 1$.

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

$\Gamma_i = (\{u_1, v_1\}, \{e_1\})$.

Присвоить $i := 1$.

Пока $i \leq m - 1$, выполнять шаг $i + 1$.

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

$\Gamma_i = (\{u_1, v_1\}, \{e_1\})$.

Присвоить $i := 1$.

Пока $i \leq m - 1$, выполнять шаг $i + 1$.

Шаг i+1.

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

$\Gamma_i = (\{u_1, v_1\}, \{e_1\})$.

Присвоить $i := 1$.

Пока $i \leq m - 1$, выполнять шаг $i + 1$.

Шаг i+1.

Пусть $e_{i+1} = \{u_{i+1}, v_{i+1}\}$.

Протокол работы алгоритма

Шаг 1.

Окрасить ребро $e_1 = \{u_1, v_1\}$ и образовать подграф

$\Gamma_i = (\{u_1, v_1\}, \{e_1\})$.

Присвоить $i := 1$.

Пока $i \leq m - 1$, выполнять шаг $i + 1$.

Шаг i+1.

Пусть $e_{i+1} = \{u_{i+1}, v_{i+1}\}$.

Возможны четыре варианта.

Протокол работы алгоритма

1). Пусть $u_{i+1}, v_{i+1} \notin V(\Gamma_i)$.

Протокол работы алгоритма

1). Пусть $u_{i+1}, v_{i+1} \notin V(\Gamma_i)$.

Тогда окрасить ребро e_i и сформировать подграф Γ_{i+1}, добавив к компонентам связности подграфа Γ_i новую компоненту связности — рассматриваемое ребро.

Протокол работы алгоритма

1). Пусть $u_{i+1}, v_{i+1} \notin V(\Gamma_i)$.

Тогда окрасить ребро e_i и сформировать подграф Γ_{i+1}, добавив к компонентам связности подграфа Γ_i новую компоненту связности — рассматриваемое ребро.

2). Пусть одна из вершин, инцидентных ребру e_{i+1}, принадлежит множеству, $V(\Gamma_i)$, а другая ему не принадлежит.

Протокол работы алгоритма

1). Пусть $u_{i+1}, v_{i+1} \notin V(\Gamma_i)$.

Тогда окрасить ребро e_i и сформировать подграф Γ_{i+1}, добавив к компонентам связности подграфа Γ_i новую компоненту связности — рассматриваемое ребро.

2). Пусть одна из вершин, инцидентных ребру e_{i+1}, принадлежит множеству, $V(\Gamma_i)$, а другая ему не принадлежит.

Без ограничения общности $u_{i+1} \in V(\Gamma_i)$, $v_{i+1} \notin V(\Gamma_i)$.

Протокол работы алгоритма

1). Пусть $u_{i+1}, v_{i+1} \notin V(\Gamma_i)$.

Тогда окрасить ребро e_i и сформировать подграф Γ_{i+1}, добавив к компонентам связности подграфа Γ_i новую компоненту связности — рассматриваемое ребро.

2). Пусть одна из вершин, инцидентных ребру e_{i+1}, принадлежит множеству, $V(\Gamma_i)$, а другая ему не принадлежит.

Без ограничения общности $u_{i+1} \in V(\Gamma_i)$, $v_{i+1} \notin V(\Gamma_i)$.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, добавив его к той компоненте связности подграфа Γ_i, которая содержит вершину u_{i+1}.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$.

Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

4). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1}, v_{i+1} \in V(\Gamma_i^j)$. Здесь Γ_i^j — одна из компонент связности графа Γ_i.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

4). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1}, v_{i+1} \in V(\Gamma_i^j)$. Здесь Γ_i^j — одна из компонент связности графа Γ_i.

Тогда ребро e_{i+1} не окрашивать и не включать в подграф Γ_i и положить $\Gamma_{i+1} := \Gamma_i$.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

4). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1}, v_{i+1} \in V(\Gamma_i^j)$. Здесь Γ_i^j — одна из компонент связности графа Γ_i.

Тогда ребро e_{i+1} не окрашивать и не включать в подграф Γ_i и положить $\Gamma_{i+1} := \Gamma_i$.

Получившиеся в результате преобразований компоненты связности подграфа Γ_{i+1} обозначить через $\Gamma_{i+1}^1, \Gamma_{i+1}^2, \ldots, \Gamma_{i+1}^{k_{i+1}}$.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

4). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1}, v_{i+1} \in V(\Gamma_i^j)$. Здесь Γ_i^j — одна из компонент связности графа Γ_i.

Тогда ребро e_{i+1} не окрашивать и не включать в подграф Γ_i и положить $\Gamma_{i+1} := \Gamma_i$.

Получившиеся в результате преобразований компоненты связности подграфа Γ_{i+1} обозначить через $\Gamma_{i+1}^1, \Gamma_{i+1}^2, \ldots, \Gamma_{i+1}^{k_{i+1}}$.

Присвоить $i := i + 1$.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

4). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1}, v_{i+1} \in V(\Gamma_i^j)$. Здесь Γ_i^j — одна из компонент связности графа Γ_i.

Тогда ребро e_{i+1} не окрашивать и не включать в подграф Γ_i и положить $\Gamma_{i+1} := \Gamma_i$.

Получившиеся в результате преобразований компоненты связности подграфа Γ_{i+1} обозначить через $\Gamma_{i+1}^1, \Gamma_{i+1}^2, \ldots, \Gamma_{i+1}^{k_{i+1}}$.

Присвоить $i := i + 1$.

Если $i = m$, то *КОНЕЦ АЛГОРИТМА*.

Протокол работы алгоритма

3). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1} \in V(\Gamma_i^k)$, $v_{i+1} \in V(\Gamma_i^l)$ и $k \neq l$. Здесь Γ_i^k, Γ_i^l — некоторые компоненты связности графа Γ_i.

Тогда окрасить ребро e_{i+1} и сформировать подграф Γ_{i+1}, объединив при помощи ребра e_{i+1} обе компоненты связности Γ_i^k, Γ_i^l подграфа Γ_i в одну компоненту связности подграфа Γ_{i+1}.

4). Пусть $u_{i+1}, v_{i+1} \in V(\Gamma_i)$, причем $u_{i+1}, v_{i+1} \in V(\Gamma_i^j)$. Здесь Γ_i^j — одна из компонент связности графа Γ_i.

Тогда ребро e_{i+1} не окрашивать и не включать в подграф Γ_i и положить $\Gamma_{i+1} := \Gamma_i$.

Получившиеся в результате преобразований компоненты связности подграфа Γ_{i+1} обозначить через $\Gamma_{i+1}^1, \Gamma_{i+1}^2, \ldots, \Gamma_{i+1}^{k_{i+1}}$.

Присвоить $i := i + 1$.

Если $i = m$, то *КОНЕЦ АЛГОРИТМА*.

Подграф Γ_m является искомым.

Пример 1. Найти минимальный остов взвешенного графа, заданного на рис. 1, если веса его ребер распределены следующим образом:
$c(1,2) = 1.1$, $c(1,3) = 0.8$, $c(1,4) = 3$, $c(1,6) = 7.3$, $c(2,4) = 0.6$,
$c(2,6) = 9$, $c(3,4) = 5$, $c(3,5) = 7.9$, $c(3,6) = 6.6$, $c(4,5) = 2.2$,
$c(4,7) = 10$, $c(5,6) = 8.2$, $c(5,7) = 7.4$, $c(6,7) = 5.1$, $c(6,8) = 4.1$,
$c(6,9) = 0.5$, $c(7,8) = 5.4$, $c(7,9) = 6.2$,
$c(7,10) = 7.8$, $c(8,9) = 6.8$, $c(8,10) = 3.3$, $c(9,10) = 6.7$, $c(9,11) = 10$,
$c(9,12) = 9.9$, $c(10,12) = 4$, $c(11,12) = 5.6$.

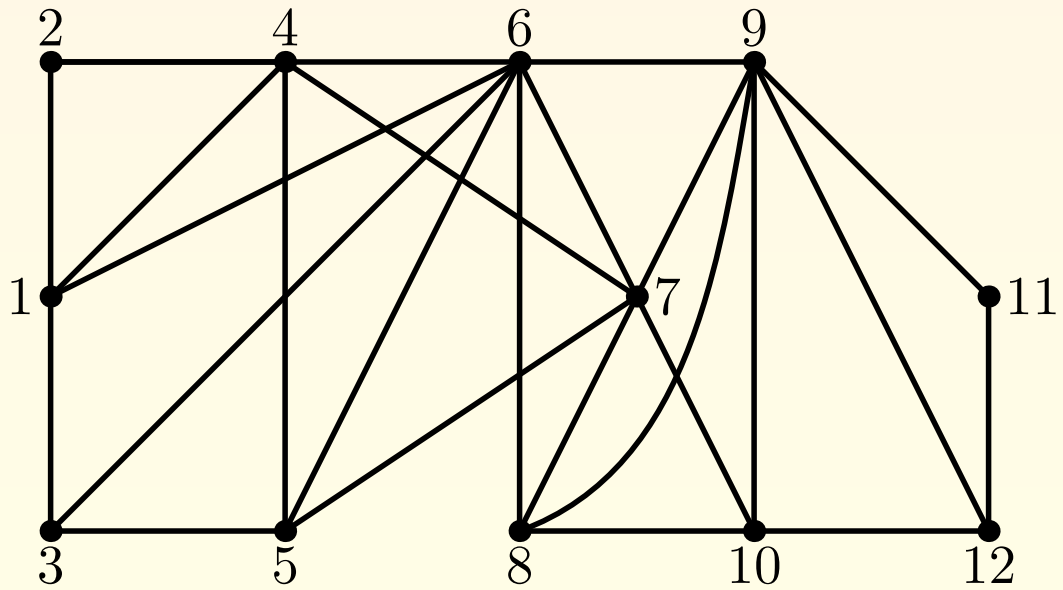
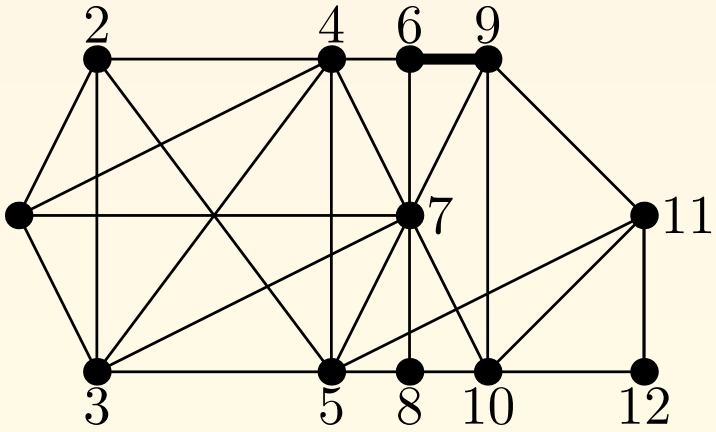


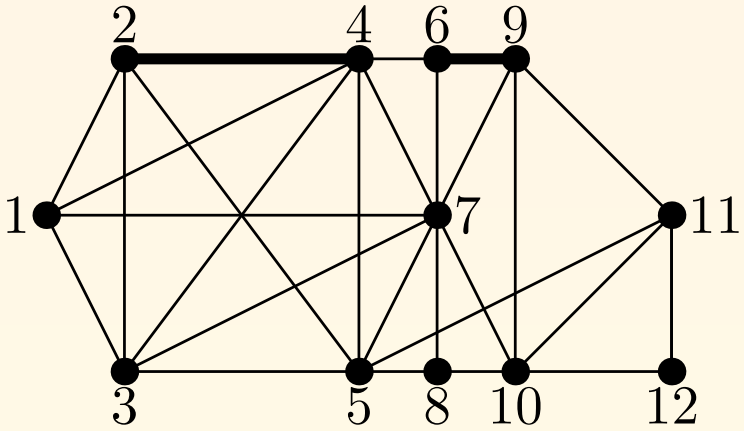
Рис.: Граф для примера 1

Упорядочим ребра графа по возрастанию их весов:
$e_1 = \{6, 9\}$, $e_2 = \{2, 4\}$, $e_3 = \{1, 3\}$, $e_4 = \{1, 2\}$, $e_5 = \{4, 5\}$, $e_6 = \{1, 4\}$,
$e_7 = \{8, 10\}$, $e_8 = \{10, 12\}$, $e_9 = \{6, 8\}$, $e_{10} = \{3, 4\}$, $e_{11} = \{6, 7\}$,
$e_{12} = \{7, 8\}$, $e_{13} = \{11, 12\}$, $e_{14} = \{7, 9\}$, $e_{15} = \{3, 6\}$, $e_{16} = \{9, 10\}$,
$e_{17} = \{8, 9\}$,
$e_{18} = \{1, 6\}$, $e_{19} = \{5, 7\}$, $e_{20} = \{7, 10\}$, $e_{21} = \{3, 5\}$, $e_{22} = \{5, 6\}$,
$e_{23} = \{2, 6\}$, $e_{24} = \{9, 12\}$, $e_{25} = \{4, 7\}$, $e_{26} = \{9, 11\}$.
Шаг 0. Начинаем с пустого графа, состоящего из вершин исходного.
Число компонент связности равно порядку.

(Шаг 1.) Выбираем ребро наименьшего ($\{6, 9\}$) веса и присоединяем. Число компонент связности уменьшилось.

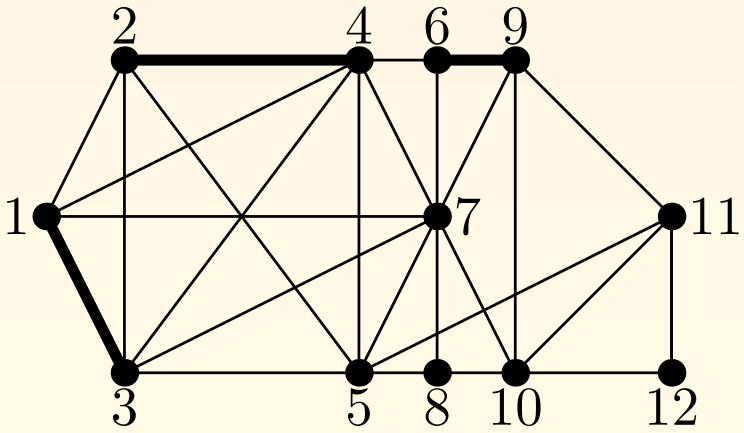


Шаг 2. Следующее ребро $e_2 = \{2, 4\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.

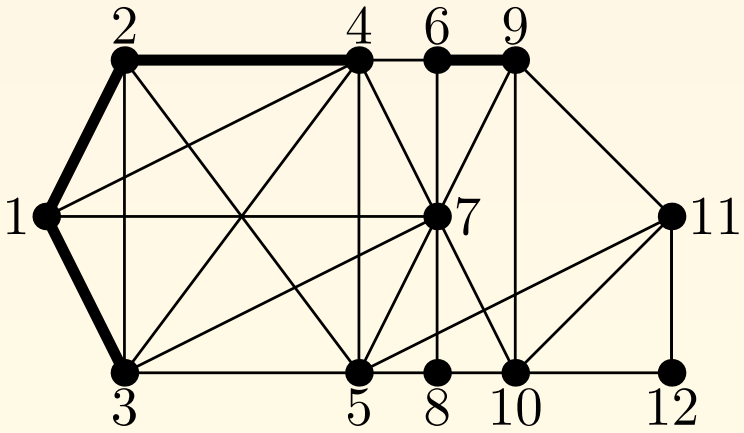


Последовательно по одному выбираем оставшиеся ребра наименьшего веса так, чтобы не появлялось циклов. Это соответствует уменьшению числа компонент связности несвязного подграфа при добавлении очередного ребра.

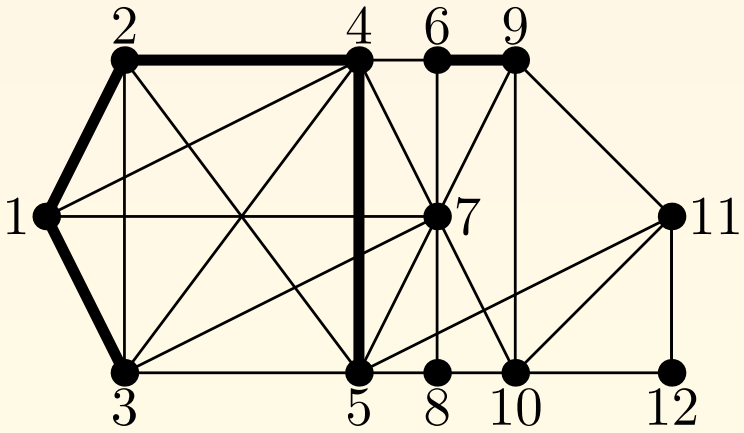
Шаг 3. Следующее ребро $e_2 = \{1, 3\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



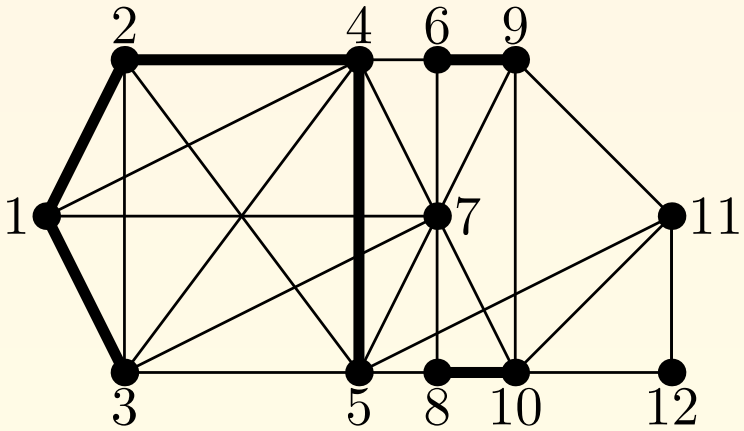
Шаг 4. Следующее ребро $e_2 = \{1, 2\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



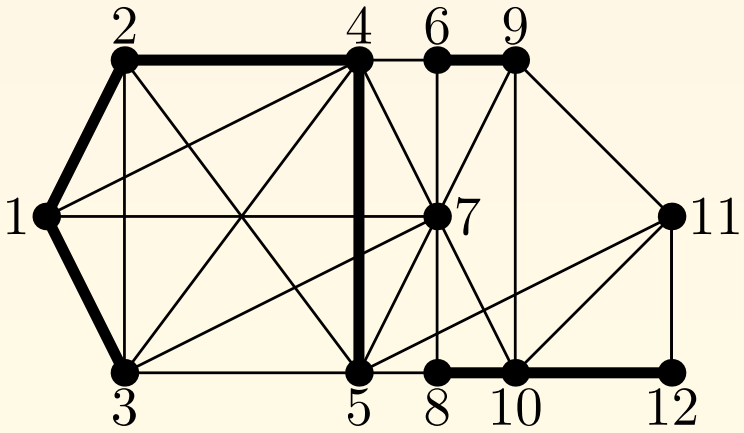
Шаг 5. Следующее ребро $e_2 = \{4, 5\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



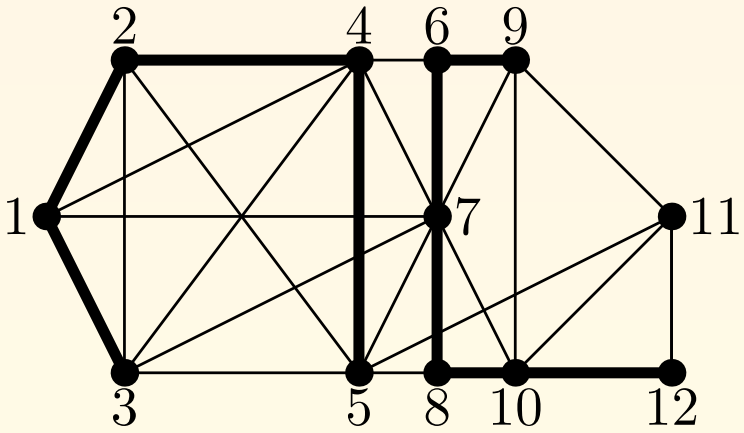
Шаг 6. Следующее ребро $e_2 = \{1, 4\}$. Число компонент связности не уменьшилось — отбрасываем, Следующее ребро $e_2 = \{8, 10\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



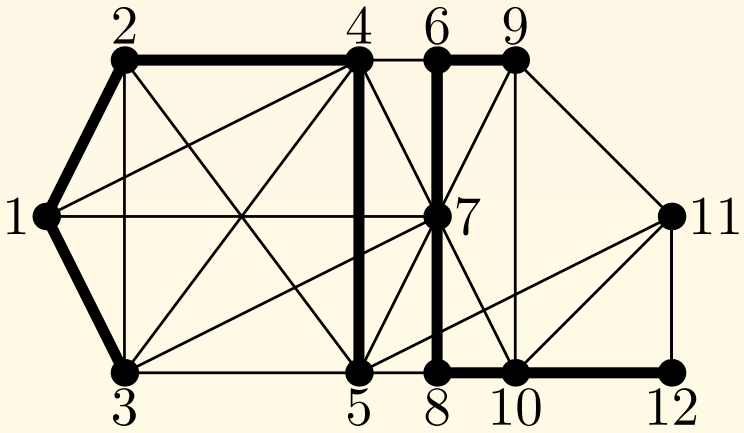
Шаг 7. Следующее ребро $e_2 = \{10, 12\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



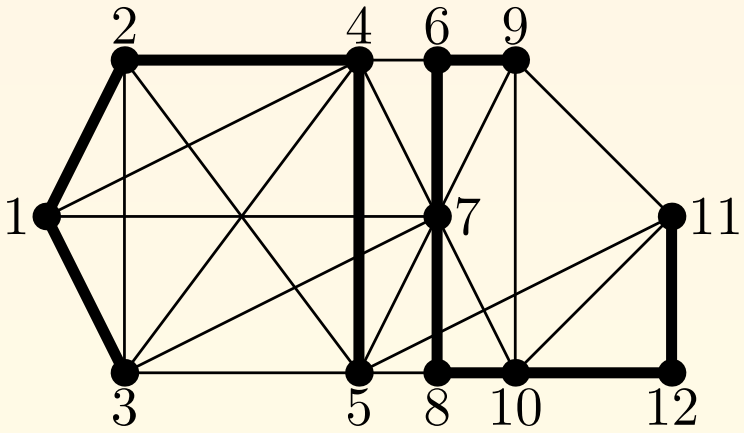
Шаг 8. Следующее ребро $e_2 = \{6, 8\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



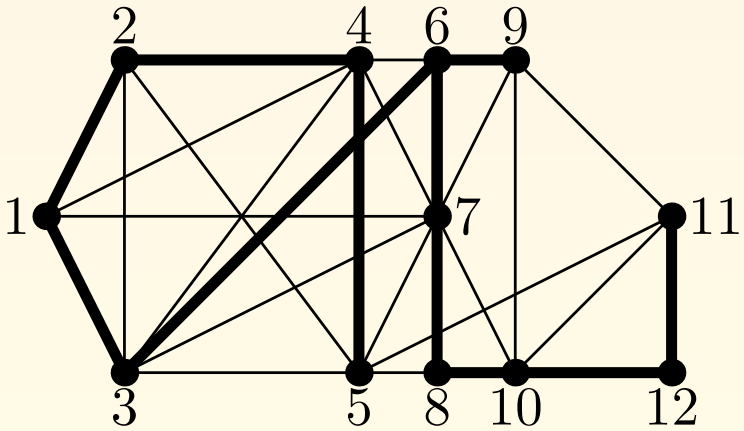
Шаг 9. Следующее ребро $e_2 = \{6, 7\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



Шаг 10. Следующее ребро $e_2 = \{11, 12\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу.



Шаг 11. Следующее ребро $e_2 = \{3, 6\}$. Число компонент связности уменьшилось, присоединяем ребро к несвязному подграфу. Конец алгоритма.



Еще одна важная задача для взвешенного графа — отыскание кратчайшего пути из одной вершины в другую.

Еще одна важная задача для взвешенного графа — отыскание кратчайшего пути из одной вершины в другую.

Пусть $G = (V, E)$ — связный взвешанный граф,

Еще одна важная задача для взвешенного графа — отыскание кратчайшего пути из одной вершины в другую.

Пусть $G = (V, E)$ — связный взвешанный граф,

$$V = \{v_1, v_2, \dots v_k\},\ E = \{e_1, e_2, \dots e_m\}$$

Еще одна важная задача для взвешенного графа — отыскание кратчайшего пути из одной вершины в другую.

Пусть $G = (V, E)$ — связный взвешанный граф,

$$V = \{v_1, v_2, \dots v_k\},\ E = \{e_1, e_2, \dots e_m\}$$

Определение:

Путь из вершины u в вершину v графа G — это простая цепь этого графа, соединяющая вершину u и вершину v.

Еще одна важная задача для взвешенного графа — отыскание кратчайшего пути из одной вершины в другую.

Пусть $G = (V, E)$ — связный взвешанный граф,

$$V = \{v_1, v_2, \dots v_k\},\ E = \{e_1, e_2, \dots e_m\}$$

Определение:

Путь из вершины u в вершину v графа G — это простая цепь этого графа, соединяющая вершину u и вершину v.

Определение:

Кратчайший путь из вершины u в вершину v — это путь из u в v наименьшего веса.

алгоритм Дейкстра

Определение:

Деревом кратчайших путей графа G из вершины u называется множество кратчайших путей из вершины u до каждой вершины этого графа.

алгоритм Дейкстра

Определение:

Деревом кратчайших путей графа G из вершины u называется множество кратчайших путей из вершины u до каждой вершины этого графа.

В ходе работы алгоритма каждой вершине присваивается так называемая *временная метка*, $\alpha(v)$.

алгоритм Дейкстра

Определение:

Деревом кратчайших путей графа G из вершины u называется множество кратчайших путей из вершины u до каждой вершины этого графа.

В ходе работы алгоритма каждой вершине присваивается так называемая *временная метка*, $\alpha(v)$.

Ее значение можно трактовать, как кратчайшее расстояние до данной вершины от начальной по уже построенному множеству ребер искомого дерева.

алгоритм Дейкстра

Определение:

Деревом кратчайших путей графа G из вершины u называется множество кратчайших путей из вершины u до каждой вершины этого графа.

В ходе работы алгоритма каждой вершине присваивается так называемая *временная метка*, $\alpha(v)$.

Ее значение можно трактовать, как кратчайшее расстояние до данной вершины от начальной по уже построенному множеству ребер искомого дерева.

В ходе работы на каждом шаге выбирается наименьшее значение временной метки, эта метка заменяется на постоянную.

алгоритм Дейкстра

Определение:

Деревом кратчайших путей графа G из вершины u называется множество кратчайших путей из вершины u до каждой вершины этого графа.

В ходе работы алгоритма каждой вершине присваивается так называемая *временная метка*, $\alpha(v)$.

Ее значение можно трактовать, как кратчайшее расстояние до данной вершины от начальной по уже построенному множеству ребер искомого дерева.

В ходе работы на каждом шаге выбирается наименьшее значение временной метки, эта метка заменяется на постоянную.

Вершина графа, получившая постоянную метку, становится *текущей*.

алгоритм Дейкстра

Определение:

Деревом кратчайших путей графа G из вершины u называется множество кратчайших путей из вершины u до каждой вершины этого графа.

В ходе работы алгоритма каждой вершине присваивается так называемая *временная метка*, $\alpha(v)$.

Ее значение можно трактовать, как кратчайшее расстояние до данной вершины от начальной по уже построенному множеству ребер искомого дерева.

В ходе работы на каждом шаге выбирается наименьшее значение временной метки, эта метка заменяется на постоянную.

Вершина графа, получившая постоянную метку, становится *текущей*.

Когда постоянную метку получит конечная вершина — дерево кратчайших путей построено.

Алгоритм построения кратчайшего пути

Шаг 1.

Алгоритм построения кратчайшего пути

Шаг 1.

Присвоить вершинам графа временные метки

Алгоритм построения кратчайшего пути

Шаг 1.

Присвоить вершинам графа временные метки

$\alpha(v_1) = 0$,

Алгоритм построения кратчайшего пути

Шаг 1.

Присвоить вершинам графа временные метки

$\alpha(v_1) = 0$,

$\alpha(v_2) = \ldots = \alpha(v_n) = +\infty$.

Алгоритм построения кратчайшего пути

Шаг 1.

Присвоить вершинам графа временные метки

$\alpha(v_1) = 0$,

$\alpha(v_2) = \ldots = \alpha(v_n) = +\infty$.

Наименьшая временная метка — у начальной вершины, она становится текущей, а ее метка — постоянной и не пересчитывается более: $v := v_1$.

Алгоритм построения кратчайшего пути

Шаг 2.

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

$$\alpha(v_i) = \min\{\alpha(v_i), \alpha(v) + wt(v, v_i)\} \quad (1)$$

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

$$\alpha(v_i) = \min\{\alpha(v_i), \alpha(v) + wt(v, v_i)\} \quad (1)$$

$wt(v, v_i)$ – вес соответствующего ребра.

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

$$\alpha(v_i) = \min\{\alpha(v_i), \alpha(v) + wt(v, v_i)\} \quad (1)$$

$wt(v, v_i)$ – вес соответствующего ребра.

Ту из вершин v_j, для которой число $\alpha(v_j)$ окажется наименьшим, выбираем за текущую, а ее метку делаем постоянной.

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

$$\alpha(v_i) = \min\{\alpha(v_i), \alpha(v) + wt(v, v_i)\} \quad (1)$$

$wt(v, v_i)$ – вес соответствующего ребра.

Ту из вершин v_j, для которой число $\alpha(v_j)$ окажется наименьшим, выбираем за текущую, а ее метку делаем постоянной.

Присоединить к дереву кратчайших путей также то ребро, по которому было получено значение постоянной метки.

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

$$\alpha(v_i) = \min\{\alpha(v_i), \alpha(v) + wt(v, v_i)\} \quad (1)$$

$wt(v, v_i)$ – вес соответствующего ребра.

Ту из вершин v_j, для которой число $\alpha(v_j)$ окажется наименьшим, выбираем за текущую, а ее метку делаем постоянной.

Присоединить к дереву кратчайших путей также то ребро, по которому было получено значение постоянной метки.

На рисунке графа окрашиваем это ребро.

Алгоритм построения кратчайшего пути

Шаг 2.

Для каждой смежной с текущей вершины со временной меткой v_i пересчитать метку по следующему правилу:

$$\alpha(v_i) = \min\{\alpha(v_i), \alpha(v) + wt(v, v_i)\} \quad (1)$$

$wt(v, v_i)$ – вес соответствующего ребра.

Ту из вершин v_j, для которой число $\alpha(v_j)$ окажется наименьшим, выбираем за текущую, а ее метку делаем постоянной.

Присоединить к дереву кратчайших путей также то ребро, по которому было получено значение постоянной метки.

На рисунке графа окрашиваем это ребро.

Положить $v = v_j$.

Алгоритм построения кратчайшего пути

Шаг 3.

Алгоритм построения кратчайшего пути

Шаг 3.

Если $v = v_n$, то *КОНЕЦ АЛГОРИТМА.*

Алгоритм построения кратчайшего пути

Шаг 3.

Если $v = v_n$, то *КОНЕЦ АЛГОРИТМА.*

Если $v \neq v_n$, то перейти к шагу 2.

Алгоритм построения кратчайшего пути

Шаг 3.

Если $v = v_n$, то *КОНЕЦ АЛГОРИТМА.*

Если $v \neq v_n$, то перейти к шагу 2.

Поднимаясь по дереву кратчайших путей от вершины v_n к вершине v_1, т.е. двигаясь по окрашенным ребрам, получаем простую цепь

Алгоритм построения кратчайшего пути

Шаг 3.

Если $v = v_n$, то *КОНЕЦ АЛГОРИТМА.*

Если $v \neq v_n$, то перейти к шагу 2.

Поднимаясь по дереву кратчайших путей от вершины v_n к вершине v_1, т.е. двигаясь по окрашенным ребрам, получаем простую цепь

$$[v_n, u_1, u_2, \ldots, u_{n-2}, v_1]$$

Алгоритм построения кратчайшего пути

Шаг 3.

Если $v = v_n$, то *КОНЕЦ АЛГОРИТМА.*

Если $v \neq v_n$, то перейти к шагу 2.

Поднимаясь по дереву кратчайших путей от вершины v_n к вершине v_1, т.е. двигаясь по окрашенным ребрам, получаем простую цепь

$$[v_n, u_1, u_2, \ldots, u_{n-2}, v_1]$$

Искомый кратчайший путь — это путь

$$[v_1, u_{n-2}, \ldots, u_2, u_1, v_n]$$

Алгоритм построения кратчайшего пути

Шаг 3.

Если $v = v_n$, то *КОНЕЦ АЛГОРИТМА.*

Если $v \neq v_n$, то перейти к шагу 2.

Поднимаясь по дереву кратчайших путей от вершины v_n к вершине v_1, т.е. двигаясь по окрашенным ребрам, получаем простую цепь

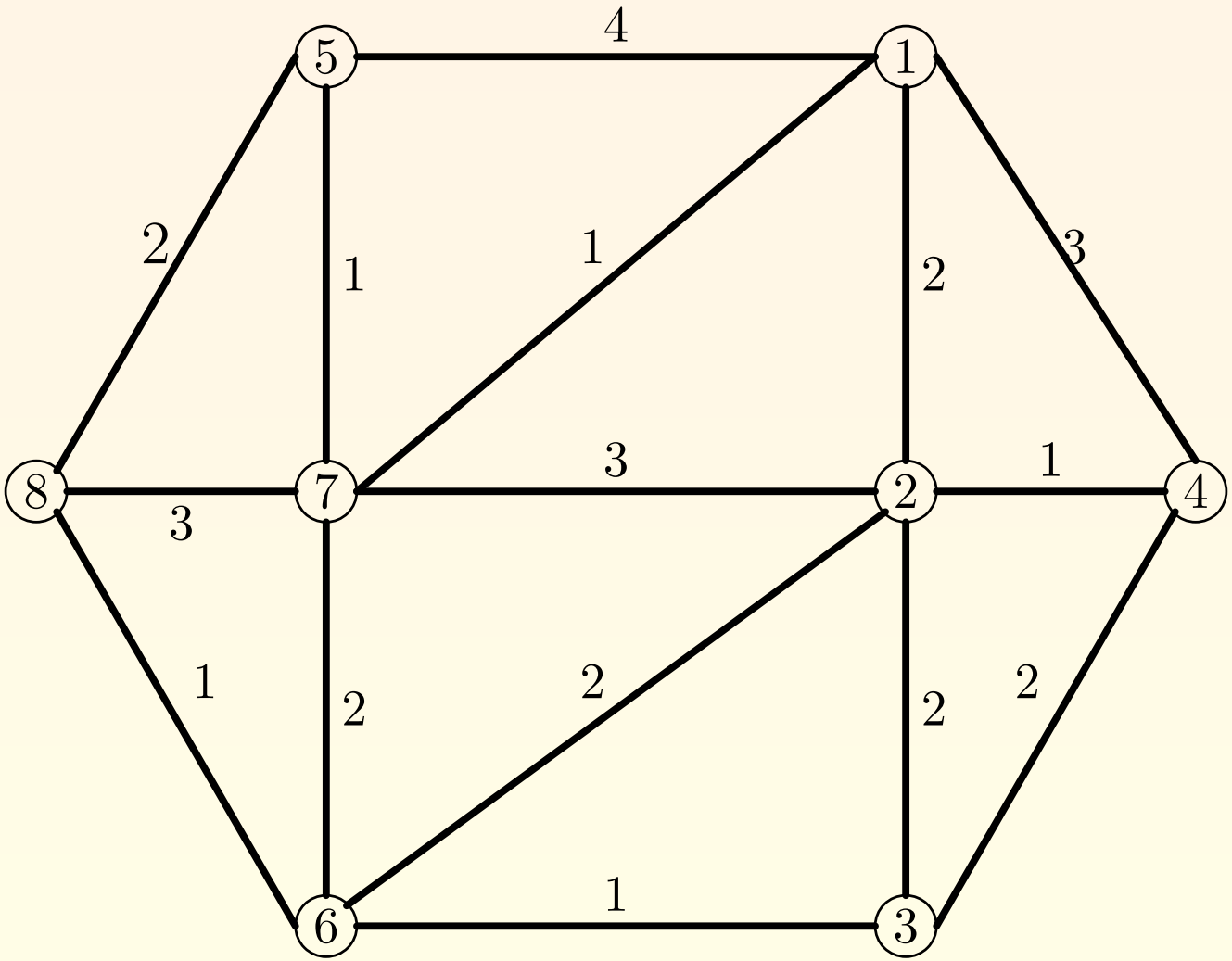
$$[v_n, u_1, u_2, \ldots, u_{n-2}, v_1]$$

Искомый кратчайший путь — это путь

$$[v_1, u_{n-2}, \ldots, u_2, u_1, v_n]$$

Его вес равен $c(v_1) + c(u_{n-1}) + \ldots + c(u_2) + c(v_n)$.

Пример 2. Применить алгоритм Дейкстры нахождения кратчайшего пути в графе.



Протокол работы удобно оформлять на следующем бланке:

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
v	(0)	∞	∞	∞	∞	∞	∞	∞		

Исходные временные метки вершин — текущая вершина v, ее метка становится постоянной.

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$

Текущая вершина v, пересчитываем временные метки смежных вершин.

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра выделены более жирными линиями.

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$
c	|	(2)	3	|	∞	3	2	∞	a	$\{v, a\}$

Текущая вершина c, пересчитываем временные метки смежных вершин.

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра выделены более жирными линиями.

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$
c	|	(2)	3	|	∞	3	2	∞	a	$\{v, a\}$
a	|	|	3	|	6	3	(2)	∞	f	$\{c, f\}$

Текущая вершина a, пересчитываем временные метки смежных вершин.

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра выделены более жирными линиями.

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$
c	|	(2)	3	|	∞	3	2	∞	a	$\{v, a\}$
a	|	|	3	|	6	3	(2)	∞	f	$\{c, f\}$
f	|	|	(3)	|	6	3	|	4	b	$\{v, b\}$

Текущая вершина f, пересчитываем временные метки смежных вершин.

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$
c	|	(2)	3	|	∞	3	2	∞	a	$\{v, a\}$
a	|	|	3	|	6	3	(2)	∞	f	$\{c, f\}$
f	|	|	(3)	|	6	3	|	4	b	$\{v, b\}$
b	|	|	|	|	4	(3)	|	4	e	$\{c, e\}$

Текущая вершина b, пересчитываем временные метки смежных вершин.

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра

Протокол работы удобно оформлять на следующем бланке:

	8	5	7	6	1	2	3	4		
	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$
c	|	(2)	3	|	∞	3	2	∞	a	$\{v, a\}$
a	|	|	3	|	6	3	(2)	∞	f	$\{c, f\}$
f	|	|	(3)	|	6	3	|	4	b	$\{v, b\}$
b	|	|	|	|	4	(3)	|	4	e	$\{c, e\}$
e	|	|	|	|	(4)	|	|	4	d	$\{b, d\}$

Текущая вершина e, пересчитываем временные метки смежных вершин.

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра

Протокол работы удобно оформлять на следующем бланке:

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра выделены более жирными линиями.

Протокол работы удобно оформлять на следующем бланке:

	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		Ребро
	(0)	∞	∞	∞	∞	∞	∞	∞		
v	|	2	3	(1)	∞	∞	∞	∞	c	$\{v, c\}$
c	|	(2)	3	|	∞	3	2	∞	a	$\{v, a\}$
a	|	|	3	|	6	3	(2)	∞	f	$\{c, f\}$
f	|	|	(3)	|	6	3	|	4	b	$\{v, b\}$
b	|	|	|	|	4	(3)	|	4	e	$\{c, e\}$
e	|	|	|	|	(4)	|	|	4	d	$\{b, d\}$
d	|	|	|	|	|	|	|	(4)	g	$\{f, g\}$

В круглые скобки взяты значения постоянных меток вершин, эти величины имеют смысл кратчайших расстояний от начальной вершины до данной. Последняя колонка содержит список ребер, составляющих дерево кратчайших расстояний. На рис. эти ребра выделены более жирными линиями.

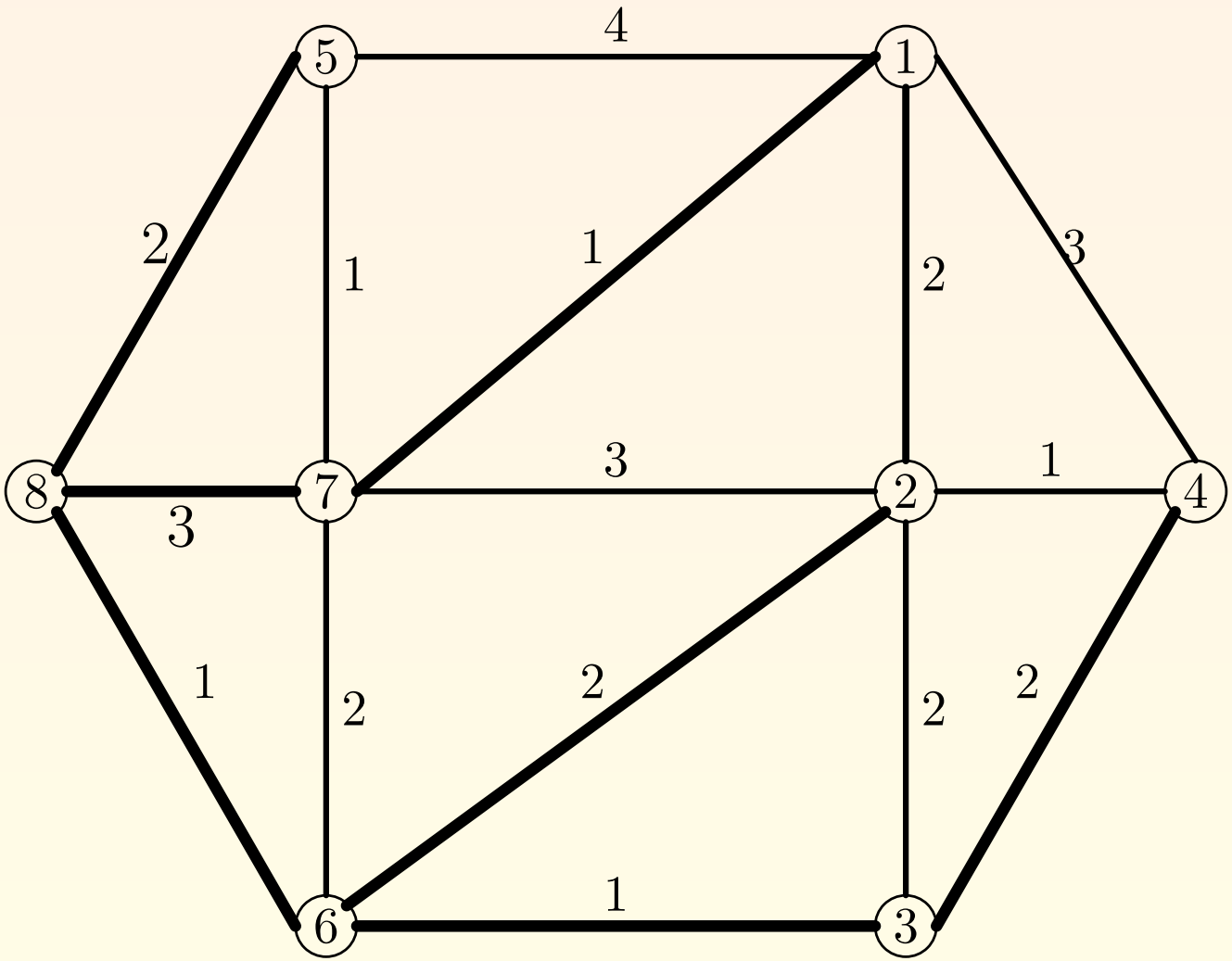


Рис.: Результат решения примера 2

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Ребра соединяют вершины только из различных подмножеств.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Ребра соединяют вершины только из различных подмножеств.

Компания или фирма заполняет штатные вакансии.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Ребра соединяют вершины только из различных подмножеств.

Компания или фирма заполняет штатные вакансии.

Имеется множество вакансий, каждой из которых соответствует список служебных обязанностей.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Ребра соединяют вершины только из различных подмножеств.

Компания или фирма заполняет штатные вакансии.

Имеется множество вакансий, каждой из которых соответствует список служебных обязанностей.

Также имеется множество претендентов, в резюме которых указаны способности к исполнению некоторых работ.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Ребра соединяют вершины только из различных подмножеств.

Компания или фирма заполняет штатные вакансии.

Имеется множество вакансий, каждой из которых соответствует список служебных обязанностей.

Также имеется множество претендентов, в резюме которых указаны способности к исполнению некоторых работ.

Ребро соединяет претендента с теми вакансиями, служебные обязанности которых он может выполнять.

Рассматриваемая задача возникает при рассмотрении так называемых *двудольных* графов.

Двудольными называют графы, множество вершин которых разбито на два непересекающихся подмножества.

Ребра соединяют вершины только из различных подмножеств.

Компания или фирма заполняет штатные вакансии.

Имеется множество вакансий, каждой из которых соответствует список служебных обязанностей.

Также имеется множество претендентов, в резюме которых указаны способности к исполнению некоторых работ.

Ребро соединяет претендента с теми вакансиями, служебные обязанности которых он может выполнять.

Требуется так провести отбор, чтобы заполнить максимальное количество вакансий.

Определение:

Паросочетанием M в графе G называется подмножество ребер этого графа, такое, что каждая вершина графа инцидентна не более, чем одному ребру из M.

Определение:

Паросочетанием M в графе G называется подмножество ребер этого графа, такое, что каждая вершина графа инцидентна не более, чем одному ребру из M.

Паросочетание с наибольшим числом ребер называется *максимальным*.

Определение:

Паросочетанием M в графе G называется подмножество ребер этого графа, такое, что каждая вершина графа инцидентна не более, чем одному ребру из M.

Паросочетание с наибольшим числом ребер называется *максимальным*.

В терминологии теории графов заполнение вакансий — задача о нахождении максимального паросочетания в графе.

Определение:

Паросочетанием M в графе G называется подмножество ребер этого графа, такое, что каждая вершина графа инцидентна не более, чем одному ребру из M.

Паросочетание с наибольшим числом ребер называется *максимальным*.

В терминологии теории графов заполнение вакансий — задача о нахождении максимального паросочетания в графе.

Можно, кстати, усложнить условие задачи, нагрузив граф и сформулировав некоторый критерий оптимальности паросочетания.

Определение:

Паросочетанием M в графе G называется подмножество ребер этого графа, такое, что каждая вершина графа инцидентна не более, чем одному ребру из M.

Паросочетание с наибольшим числом ребер называется *максимальным*.

В терминологии теории графов заполнение вакансий — задача о нахождении максимального паросочетания в графе.

Можно, кстати, усложнить условие задачи, нагрузив граф и сформулировав некоторый критерий оптимальности паросочетания.

Скажем, каждый претендент указывает желаемую зарплату, а наша цель — экономия фонда заработной платы.

Ограничимся задачей о максимальном паросочетании.

Ограничимся задачей о максимальном паросочетании.

Известный алгоритм решения этой задачи состоит в построении расширяющейся цепи.

Ограничимся задачей о максимальном паросочетании.

Известный алгоритм решения этой задачи состоит в построении расширяющейся цепи.

Определение:

Вершина графа G называется *открытой*, если она не инцидентна ни одному из ребер паросочетания M.

Ребра из M будем изображать темными линиями, а вершины им, инцидентные, — темными кругами.

Определение:

Ребра из M будем изображать темными линиями, а вершины им, инцидентные, — темными кругами.

Ребра, не попавшие в M, будем изображать светлыми линиями, а открытые вершины — окружностями.

Определение:

Ребра из M будем изображать темными линиями, а вершины им, инцидентные, — темными кругами.

Ребра, не попавшие в M, будем изображать светлыми линиями, а открытые вершины — окружностями.

Определение:

Увеличивающейся цепью называется простая цепь

Ребра из M будем изображать темными линиями, а вершины им, инцидентные, — темными кругами.

Ребра, не попавшие в M, будем изображать светлыми линиями, а открытые вершины — окружностями.

Определение:

Увеличивающейся цепью называется простая цепь

$$[v_0, u_1, v_1, u_2, v_2, u_3, \ldots, v_{n-1}, u_n, v_n, u_{n+1}]$$

Ребра из M будем изображать темными линиями, а вершины им, инцидентные, — темными кругами.

Ребра, не попавшие в M, будем изображать светлыми линиями, а открытые вершины — окружностями.

Определение:

Увеличивающейся цепью называется простая цепь

$$[v_0, u_1, v_1, u_2, v_2, u_3, \ldots, v_{n-1}, u_n, v_n, u_{n+1}]$$

графа G, такая, что $n \in \mathbb{N} \cup \{0\}$, $\{u_i, v_i\} \in M$, $\{v_j, u_{j+1}\} \notin M$ для любых $i \in \{1, \ldots, n\}$, $j \in \{0, \ldots, n\}$ и вершины v_0, u_{n+1} являются открытыми (см. рис. (4).)

Цепью длины k назовём некоторый простой путь (т. е. не содержащий повторяющихся вершин или рёбер), содержащий k рёбер.

Цепью длины k назовём некоторый простой путь (т. е. не содержащий повторяющихся вершин или рёбер), содержащий k рёбер.

Чередующейся цепью (в двудольном графе относительно некоторого паросочетания) назовём цепь, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

Цепью длины k назовём некоторый простой путь (т. е. не содержащий повторяющихся вершин или рёбер), содержащий k рёбер.

Чередующейся цепью (в двудольном графе относительно некоторого паросочетания) назовём цепь, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

Увеличивающей цепью (в двудольном графе относительно некоторого паросочетания) назовём цепь, у которой начальная и конечная вершины не принадлежат паросочетанию.

Теорема Бержа.

Паросочетание является максимальным тогда и только тогда, когда не существует увеличивающих относительно него цепей.

Теорема Бержа.

Паросочетание является максимальным тогда и только тогда, когда не существует увеличивающих относительно него цепей.

Далее, заметим, что если найдена некоторая увеличивающая цепь, то с её помощью легко увеличить мощность паросочетания на 1.

Теорема Бержа.

Паросочетание является максимальным тогда и только тогда, когда не существует увеличивающих относительно него цепей.

Далее, заметим, что если найдена некоторая увеличивающая цепь, то с её помощью легко увеличить мощность паросочетания на 1.

Пройдём вдоль этой цепи, и каждое ребро поочерёдно будем добавлять/удалять из паросочетания (т. е. первое ребро (которое по определению не принадлежит паросочетанию) добавим в паросочетание, второе удалим, третье добавим, и т.д.).

Действительно, в результате этой операции мы увеличим мощность паросочетания на 1 (для этого достаточно заметить, что длина увеличивающей цепи всегда нечётна, а корректность вышеописанного преобразования очевидна).

Действительно, в результате этой операции мы увеличим мощность паросочетания на 1 (для этого достаточно заметить, что длина увеличивающей цепи всегда нечётна, а корректность вышеописанного преобразования очевидна).

Таким образом, мы получили каркас алгоритма построения максимального паросочетания: искать в графе увеличивающие цепи, пока они существуют, и увеличивать паросочетание вдоль них.

Действительно, в результате этой операции мы увеличим мощность паросочетания на 1 (для этого достаточно заметить, что длина увеличивающей цепи всегда нечётна, а корректность вышеописанного преобразования очевидна).

Таким образом, мы получили каркас алгоритма построения максимального паросочетания: искать в графе увеличивающие цепи, пока они существуют, и увеличивать паросочетание вдоль них.

Осталось детализировать способ нахождения увеличивающих цепей.

Алгоритм Куна основан на поиске в глубину или в ширину, и выбирает каждый раз любую из найденных увеличивающих цепей.

Алгоритм Куна основан на поиске в глубину или в ширину, и выбирает каждый раз любую из найденных увеличивающих цепей.

Стоит заметить, что есть и другие способы, например, в более эффективном алгоритме Хопкрофта-Карпа.

Алгоритм Куна основан на поиске в глубину или в ширину, и выбирает каждый раз любую из найденных увеличивающих цепей.

Стоит заметить, что есть и другие способы, например, в более эффективном алгоритме Хопкрофта-Карпа.

Рассмотрим подробнее алгоритм поиска увеличивающей цепи (пусть, для определённости, это поиск в глубину).

Алгоритм Куна основан на поиске в глубину или в ширину, и выбирает каждый раз любую из найденных увеличивающих цепей.

Стоит заметить, что есть и другие способы, например, в более эффективном алгоритме Хопкрофта-Карпа.

Рассмотрим подробнее алгоритм поиска увеличивающей цепи (пусть, для определённости, это поиск в глубину).

Поиск начинает идти из вершины первой доли.

Алгоритм Куна основан на поиске в глубину или в ширину, и выбирает каждый раз любую из найденных увеличивающих цепей.

Стоит заметить, что есть и другие способы, например, в более эффективном алгоритме Хопкрофта-Карпа.

Рассмотрим подробнее алгоритм поиска увеличивающей цепи (пусть, для определённости, это поиск в глубину).

Поиск начинает идти из вершины первой доли.

Из первой доли во вторую он ходит только по рёбрам, не принадлежащим паросочетанию, а из второй доли в первую - наоборот, только по принадлежащим.

С точки зрения реализации, поиск в глубину всегда находится в вершине первой доли, и он возвращает булево значение - найдена цепь или не найдена.

С точки зрения реализации, поиск в глубину всегда находится в вершине первой доли, и он возвращает булево значение - найдена цепь или не найдена.

Из текущей вершины V поиск в глубину пытается пойти по всем смежным рёбрам (кроме принадлежащего паросочетанию), и если он может пойти в вершину T_0 второй доли, не принадлежащей паросочетанию, то возвращает True и добавляет ребро $(V,\ T_0)$ в паросочетание.

Если же он пытается пойти в вершину T_0, уже принадлежащую паросочетанию, то он вызывает себя из вершины $Mt[T_0]$ (соседа T_0 в паросочетании), и если цепь была найдена, то добавляет ребро $(V,\ T_0)$ в паросочетание.

Алгоритм построения максимального паросочетания

Шаг 1.

Алгоритм построения максимального паросочетания

Шаг 1.

Полагаем $M := \emptyset$.

Алгоритм построения максимального паросочетания

Шаг 1.

Полагаем $M := \emptyset$.

Шаг 2.

Алгоритм построения максимального паросочетания

Шаг 1.

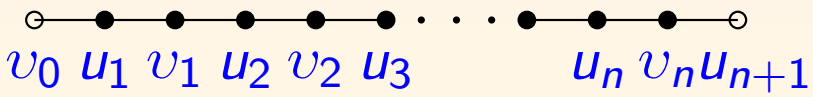
Полагаем $M := \emptyset$.

Шаг 2.

Выбираем произвольно открытую вершину v_0. Если существует увеличивающаяся цепь

Алгоритм построения максимального паросочетания

Шаг 1.



Полагаем $M := \emptyset$.

Шаг 2.

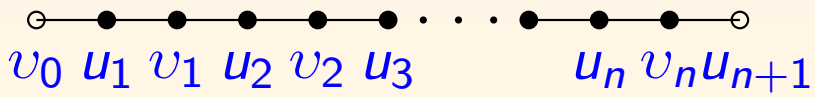
Выбираем произвольно открытую вершину v_0. Если существует увеличивающаяся цепь

$$p := [v_0, u_1, v_1, u_2, v_2, u_3, \ldots, v_{n-1}, u_n, v_n, u_{n+1}]$$

Алгоритм построения максимального паросочетания

Шаг 1.

Полагаем $M := \emptyset$.



Шаг 2.

Выбираем произвольно открытую вершину v_0. Если существует увеличивающаяся цепь

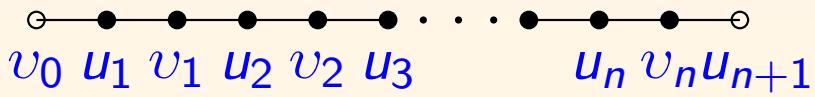
$$p := [v_0, u_1, v_1, u_2, v_2, u_3, \ldots, v_{n-1}, u_n, v_n, u_{n+1}]$$

начинающаяся с этой вершины (см. рис.), то полагаем

Алгоритм построения максимального паросочетания

Шаг 1.

Полагаем $M := \emptyset$.



Шаг 2.

Выбираем произвольно открытую вершину v_0. Если существует увеличивающаяся цепь

$$p := [v_0, u_1, v_1, u_2, v_2, u_3, \ldots, v_{n-1}, u_n, v_n, u_{n+1}]$$

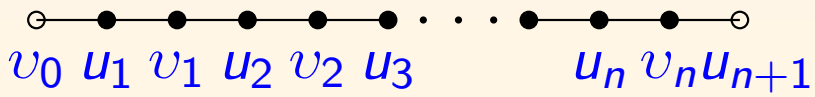
начинающаяся с этой вершины (см. рис.), то полагаем

$$M := M \cup \{\{v_0, u_1\}, \{v_1, u_2\}, \{v_2, u_3\}, \ldots, \{v_{n-1}, u_n\}, \{v_n, u_{n+1}\}\} \setminus$$

Алгоритм построения максимального паросочетания

Шаг 1.

Полагаем $M := \emptyset$.



Шаг 2.

Выбираем произвольно открытую вершину v_0. Если существует увеличивающаяся цепь

$$p := [v_0, u_1, v_1, u_2, v_2, u_3, \ldots, v_{n-1}, u_n, v_n, u_{n+1}]$$

начинающаяся с этой вершины (см. рис.), то полагаем

$$M := M \cup \{\{v_0, u_1\}, \{v_1, u_2\}, \{v_2, u_3\}, \ldots, \{v_{n-1}, u_n\}, \{v_n, u_{n+1}\}\} \setminus \setminus \{\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_n, v_n\}\}$$

Алгоритм построения максимального паросочетания

Таким образом, для любой увеличивающейся цепи "светлые" ребра включаем в паросочетание M, а "темные" ребра из него исключаем.

Алгоритм построения максимального паросочетания

Таким образом, для любой увеличивающейся цепи "светлые" ребра включаем в паросочетание M, а "темные" ребра из него исключаем.

А если такой цепи не существует, то *КОНЕЦ АЛГОРИТМА. Шаг 3.* Если еще есть открытые вершины, то переходим к шагу 2.

Алгоритм построения максимального паросочетания

Таким образом, для любой увеличивающейся цепи "светлые" ребра включаем в паросочетание M, а "темные" ребра из него исключаем.

А если такой цепи не существует, то *КОНЕЦ АЛГОРИТМА. Шаг 3.* Если еще есть открытые вершины, то переходим к шагу 2.

Если их нет, то *КОНЕЦ АЛГОРИТМА.*

Алгоритм построения максимального паросочетания

Таким образом, для любой увеличивающейся цепи "светлые" ребра включаем в паросочетание M, а "темные" ребра из него исключаем.

А если такой цепи не существует, то *КОНЕЦ АЛГОРИТМА. Шаг 3.* Если еще есть открытые вершины, то переходим к шагу 2.

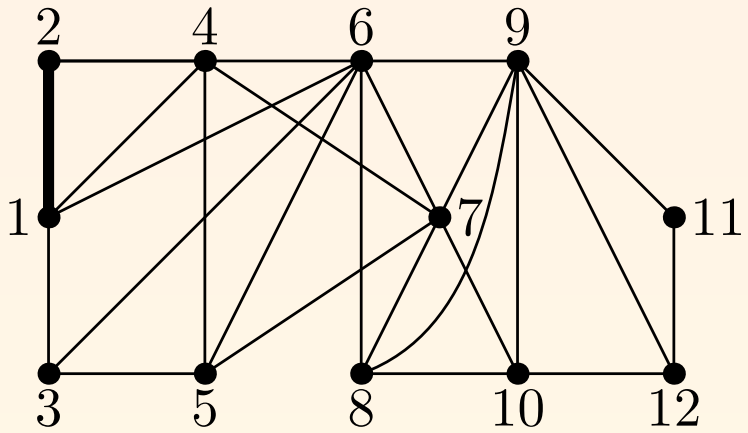
Если их нет, то *КОНЕЦ АЛГОРИТМА.*

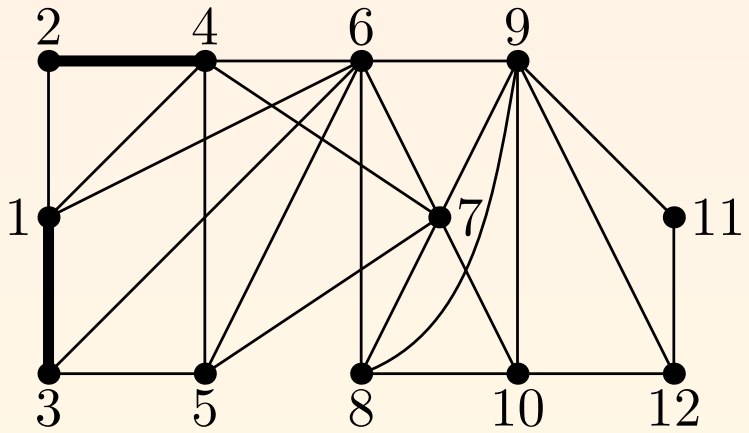
Заметим, что решение задачи о максимальном паросочетании, вообще говоря, неоднозначно и зависит от выбора открытых вершин и соответствующих увеличивающихся цепей на шаге 2.

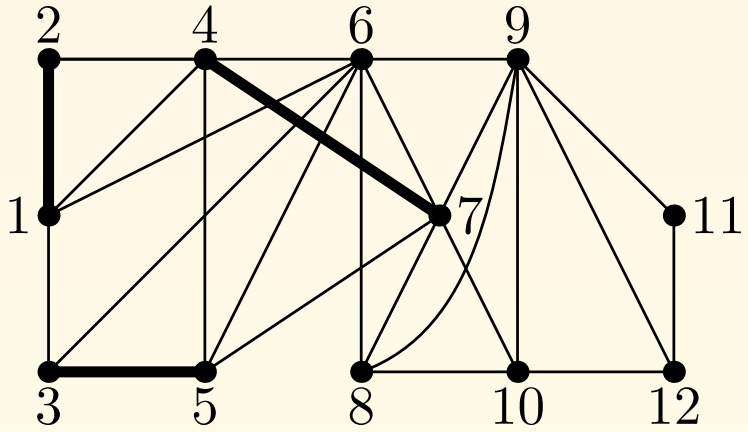
Таким образом, мы получили каркас алгоритма построения максимального паросочетания: искать в графе увеличивающие цепи, пока они существуют, и увеличивать паросочетание вдоль них.
Пример 3. Найти максимальное паросочетание в графе, изображенном на рис. 1.

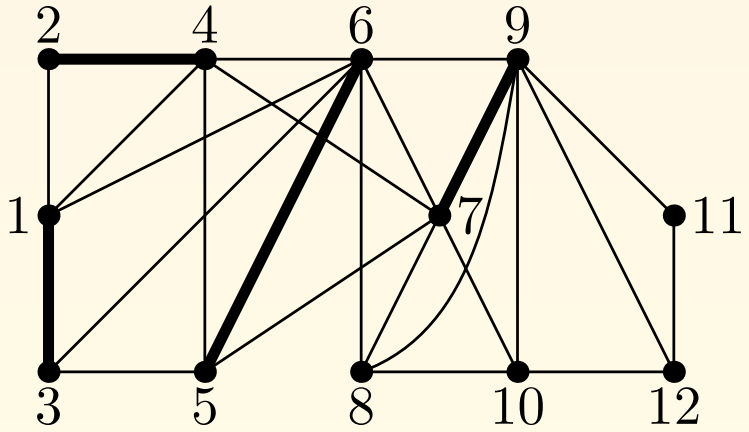
Приведем только протокол последовательного построения искомого паросочетания в виде серии картинок (рис. 3). Результатом будет множество

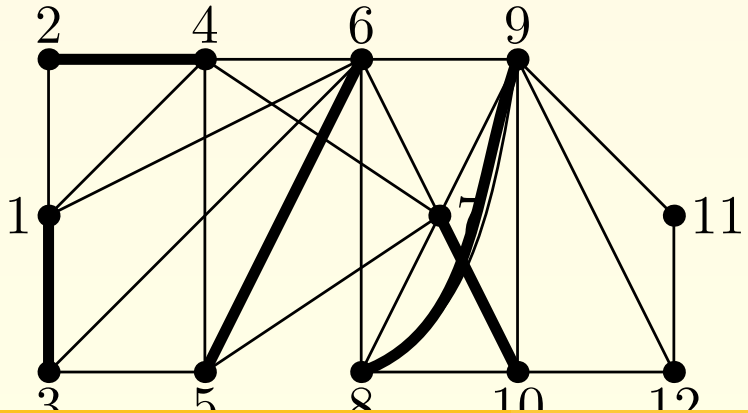
$$M = \{\{1,3\},\{2,4\},\{5,6\},\{7,8\},\{9,11\},\{10,12\}\}.$$

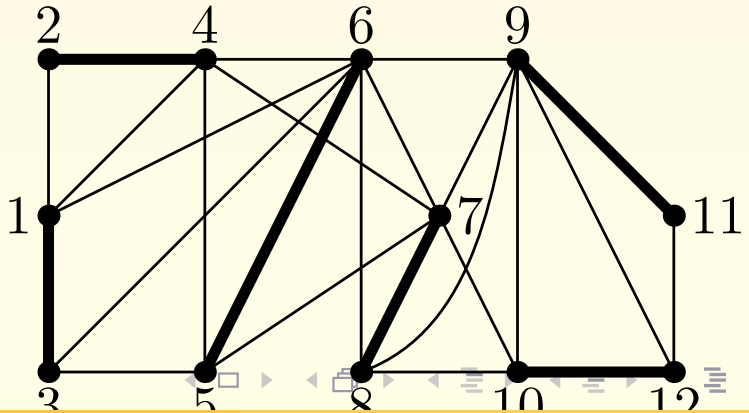

2
4
6
9
1
7
11
3
5
8
10
12


2
4
6
9
1
7
11
3
5
8
10
12


2
4
6
9
1
7
11
3
5
8
10
12


2
4
6
9
1
7
11
3
5
8
10
12


2
4
6
9
1
11
3
5
8
10
12


2
4
6
9
1
7
11
3
5
8
10
12

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

1. существует единственная вершина — *источник* — в которую не заходит ни одна дуга (только исходят);

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

1. существует единственная вершина — *источник* — в которую не заходит ни одна дуга (только исходят);
2. существует единственная вершина — *сток* — из которой не исходит ни одна дуга (только заходят);

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

1. существует единственная вершина — *источник* — в которую не заходит ни одна дуга (только исходят);
2. существует единственная вершина — *сток* — из которой не исходит ни одна дуга (только заходят);
3. на множестве дуг задана числовая функция — *пропускная способность*, значения которой $c(e_i) \geq 0$.

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

1. существует единственная вершина — *источник* — в которую не заходит ни одна дуга (только исходят);
2. существует единственная вершина — *сток* — из которой не исходит ни одна дуга (только заходят);
3. на множестве дуг задана числовая функция — *пропускная способность*, значения которой $c(e_i) \geq 0$.

Определим на множестве дуг еще одну функцию — *поток в транспортной сети* $f(v_i, v_j)$.
Поток принимает целочисленные значения, причем:

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

Определим на множестве дуг еще одну функцию — *поток в транспортной сети* $f(v_i, v_j)$.

Поток принимает целочисленные значения, причем:

1. $0 \leq f(v_i, v_j) \leq c(v_i, v_j)$;

Основные определения

Транспортная сеть — орграф D со множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$, удовлетворяющим следующим условиям:

Определим на множестве дуг еще одну функцию — *поток в транспортной сети* $f(v_i, v_j)$.
Поток принимает целочисленные значения, причем:

1. $0 \leq f(v_i, v_j) \leq c(v_i, v_j)$;
2. для любой вершины, кроме истока и стока, сумма значений потока по заходящим дугам равна сумме по исходящим.

Идея алгоритма Форда-Фалкерсона следующая. Ищем путь из истока в сток с ненулевыми (допустимыми) пропускными способностями всех дуг. Увеличиваем поток по этому пути до тех пор, пока хотя бы одна из дуг не становится насыщенной, т. е. поток по этой дуге не сравняется с пропускной способностью (насыщенный путь). Затем ищем еще один (ненасыщенный) путь из истока в сток и т. д.

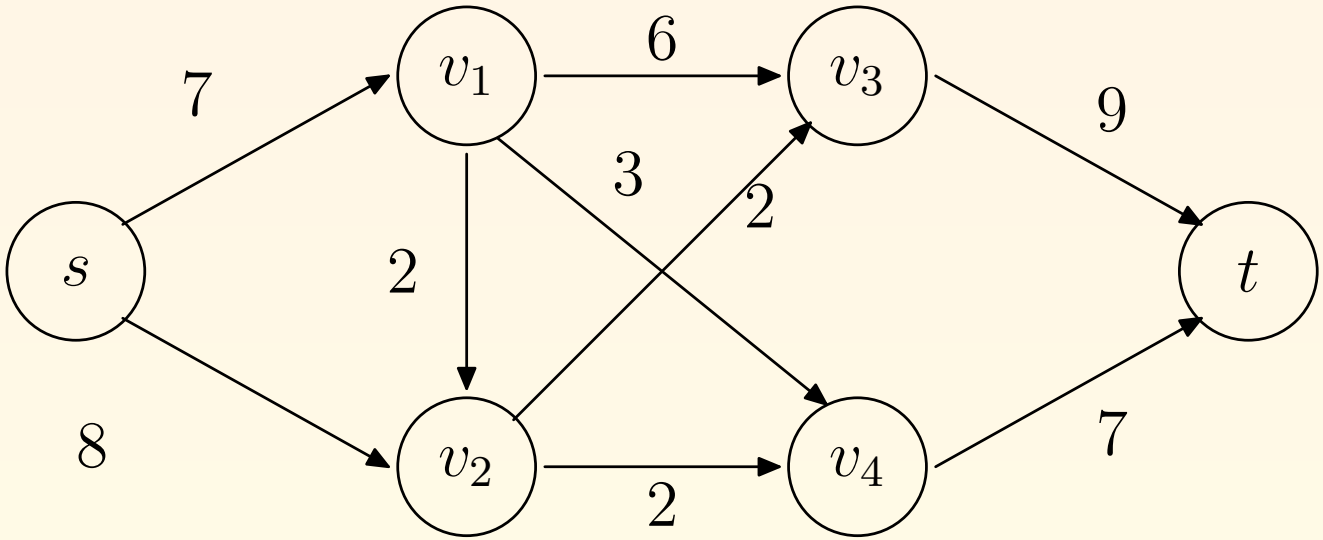
Дадим описание псевдокода алгоритма Форда-Фалкерсона, следуя [?].

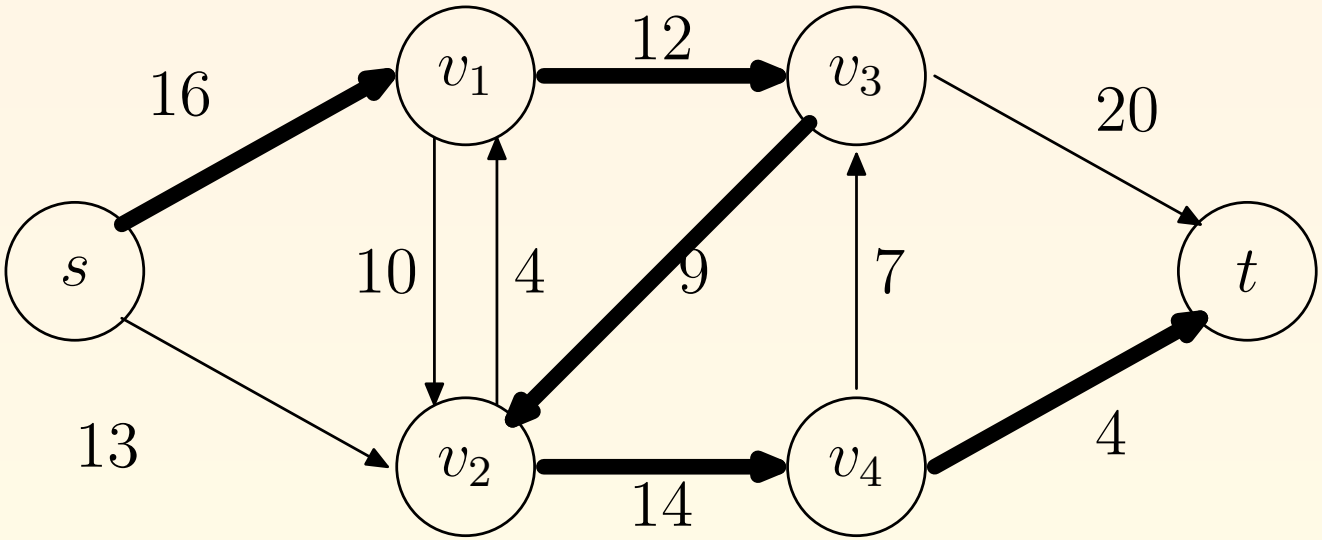
Алгоритм Ford-Fulkerson(D, s, t)

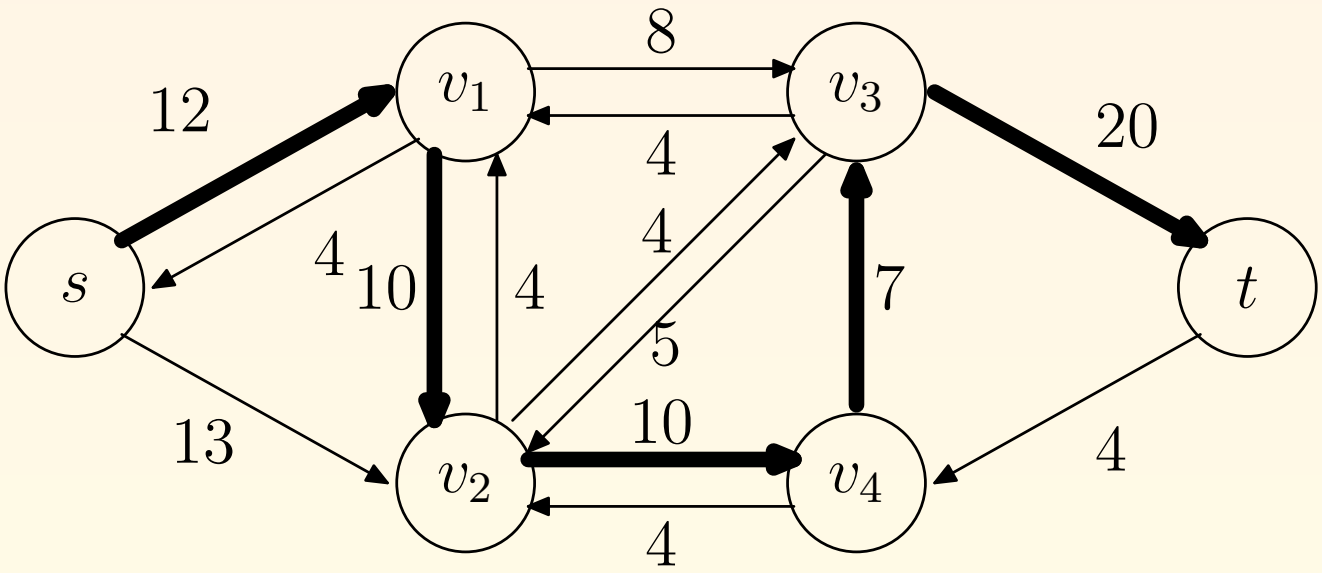
1 **for** для каждой дуги $(u, v) \in E[D]$
2 **do** $f[u, v] \leftarrow 0$
3 $f[v, u] \leftarrow 0$ ▷ Начальный поток.

4 **while** существует путь **p** из s в t в остаточной сети D_f
▷ Способ отыскания расширяющего пути не конкретизируется.
5 **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in \mathbf{p}\}$
6 **for** каждой дуги $(u, v) \in \mathbf{p}$
7 **do** $f[u, v] \leftarrow f[u, v] + c_f(\mathbf{p})$
8 $f[v, u] \leftarrow -f[v, u]$

Здесь (строка 5) использовано обозначение
$c_f(u, v) = c(u, v) - f(u, v)$ — остаточная пропускная способность.


v_1
v_3
s
t
v_2
v_4
7
6
9
3
2
2
8
2
7


v_1
v_3
s
t
v_2
v_4
16
12
20
10
4
9
7
13
14
4


v_1
v_3
s
t
v_2
v_4
8
12
4
20
4
4
10
4
7
5
10
13
4
4