

31.03.2025

Графы (продолжение).

Филиппов Михаил Витальевич

m.filippov@g.nsu.ru

89232283872

Императивное программирование, 2024-2025

N * Новосибирский
государственный
университет
***НАСТОЯЩАЯ НАУКА**

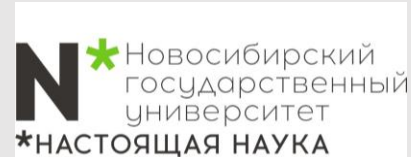


Давайте познакомимся



Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++




Вступительная часть

Адженда



Графы

90 минут

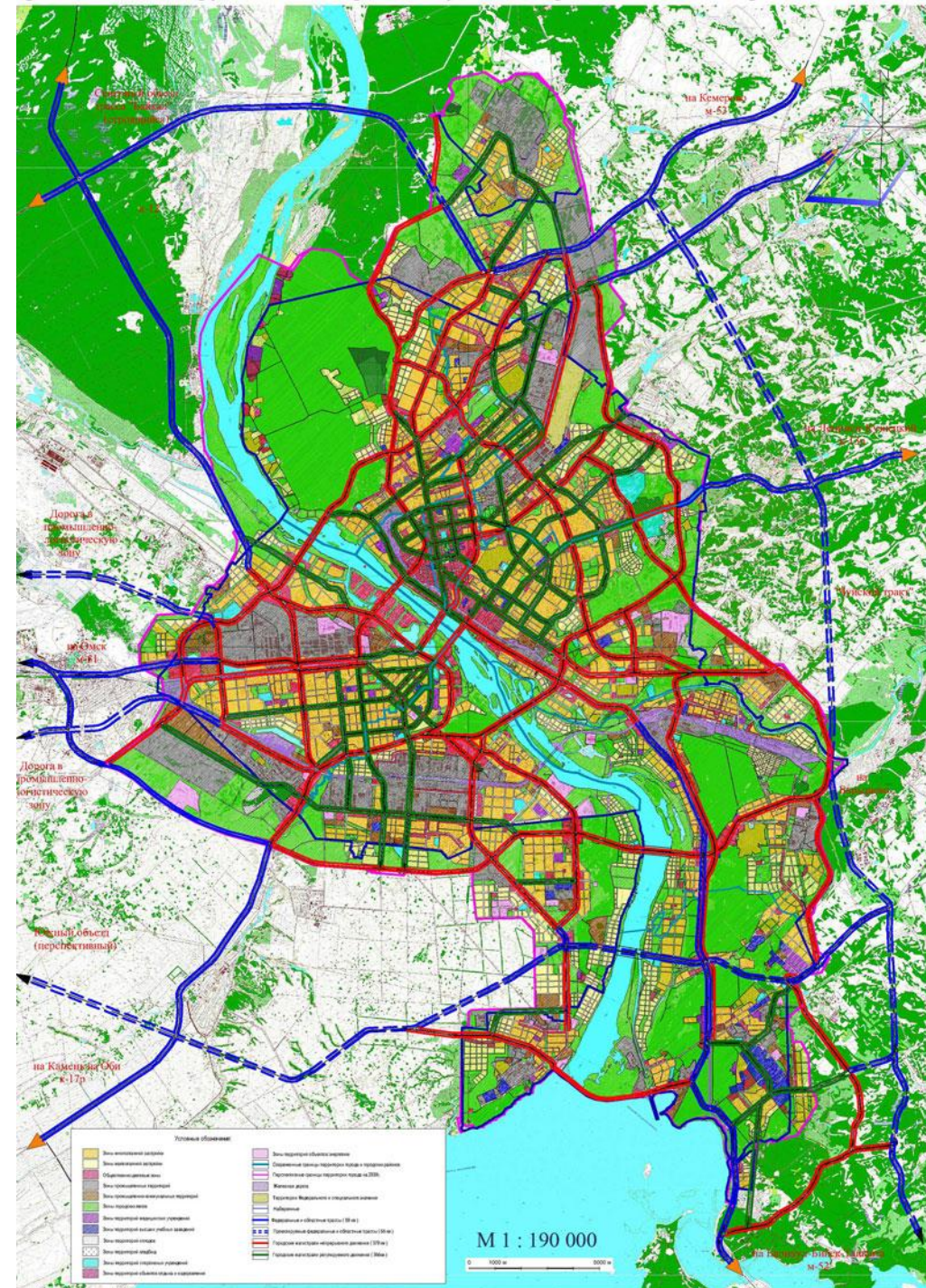


Напоминание с прошлого семестра (лекция 11)

Введение

Граф — это абстрактная структура данных, которая используется для реализации математической концепции графов. По сути, это набор вершин (также называемых узлами) и ребер, которые соединяют эти вершины. Граф часто рассматривается как обобщение древовидной структуры, где вместо чисто родительско-дочерних отношений между узлами дерева может существовать любой вид сложных отношений. Почему графы полезны? Графы широко используются для моделирования любой ситуации, когда сущности или вещи связаны друг с другом парами. Например, следующая информация может быть представлена графами:

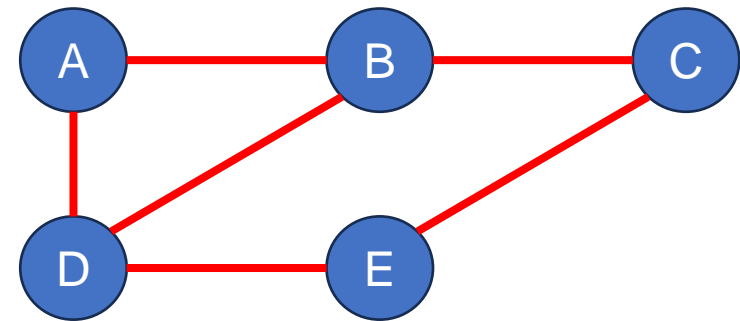
- Семейные деревья, в которых узлы-члены имеют ребро от родителя к каждому из своих дочерних узлов.
- Транспортные сети, в которых узлами являются аэропорты, перекрестки, порты и т. д. Ребра могут быть авиарейсами, односторонними дорогами, судоходными маршрутами и т. д.



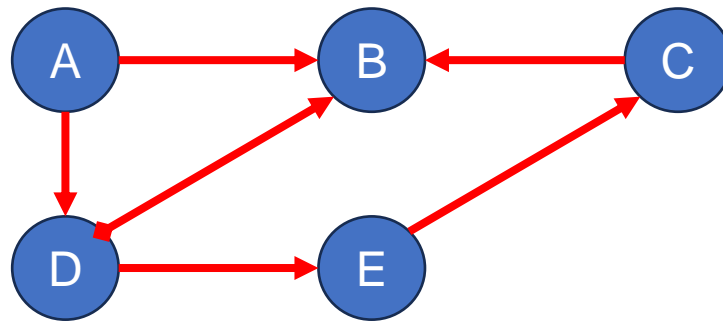
Как определяется граф

Граф G определяется как упорядоченное множество (V, E) , где $V(G)$ представляет множество вершин, а $E(G)$ представляет ребра, соединяющие эти вершины. На рисунке показан граф с $V(G) = \{A, B, C, D \text{ и } E\}$ и $E(G) = \{(A, B), (B, C), (A, D), (B, D), (D, E), (C, E)\}$. Обратите внимание, что в графе пять вершин или узлов и шесть ребер.

Граф бывает неориентированным и ориентированным



Неориентированный
граф



Направленный
граф

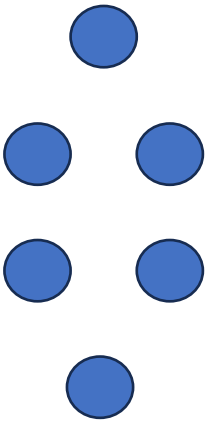


Терминология графа

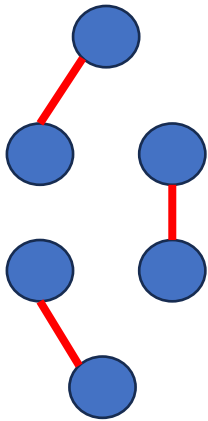
Смежные узлы или соседи Для каждого ребра $e = (u, v)$, соединяющего узлы u и v , узлы u и v являются конечными точками и называются смежными узлами или соседями.

Степень узла Степень узла u , $\deg(u)$, — это общее количество ребер, содержащих узел u . Если $\deg(u) = 0$, это означает, что u не принадлежит ни одному ребру, и такой узел называется изолированным узлом.

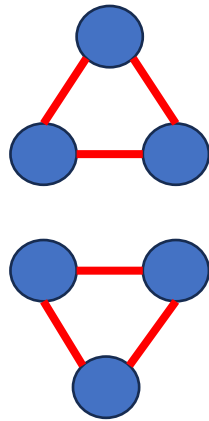
Регулярный граф Это граф, в котором каждая вершина имеет одинаковое количество соседей. То есть каждый узел имеет одинаковую степень. Регулярный граф с вершинами степени k называется k -регулярным графом или регулярным графом степени k .



(0-регулярный граф)



(1-регулярный граф)



(2-регулярный граф)



Терминология графа

Путь Путь P , записанный как $P = \{v_0, v_1, v_2, \dots, v_n\}$, длины n от узла u до v , определяется как последовательность из $(n+1)$ узлов. Здесь $u = v_0$, $v = v_n$ и v_{i-1} смежно с v_i для $i = 1, 2, 3, \dots, n$.

Замкнутый путь Путь P называется замкнутым путем, если ребро имеет одинаковые конечные точки. То есть, если $v_0 = v_n$.

Простой путь Путь P называется простым путем, если все узлы в пути различны, за исключением того, что v_0 может быть равен v_n . Если $v_0 = v_n$, то путь называется замкнутым простым путем.

Цикл Путь, в котором первая и последняя вершины одинаковы. Простой цикл не имеет повторяющихся ребер или вершин (кроме первой и последней вершин).

Кликовый граф В неориентированном графе $G = (V, E)$ клики — это подмножество множества вершин $C \subseteq V$, такое, что для каждой двух вершин в C существует ребро, соединяющее две вершины.

Петля Ребро, которое имеет идентичные конечные точки, называется петлей. То есть $e = (u, u)$.

Размер графа Размер графа — это общее количество ребер в нем.



Терминология графа

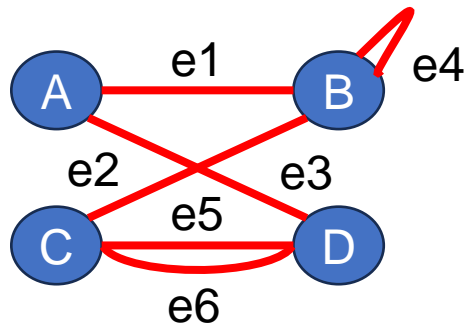
Связный граф Граф называется связным, если для любых двух вершин (u, v) в V существует путь от u до v . То есть в связном графе нет изолированных узлов. Связный граф, не имеющий цикла, называется деревом. Поэтому дерево рассматривается как особый граф.

Полный граф Граф G называется полным, если все его узлы полностью связаны. То есть существует путь от одного узла до каждого другого узла в графе. Полный граф имеет $n(n-1)/2$ ребер, где n — число узлов в G .

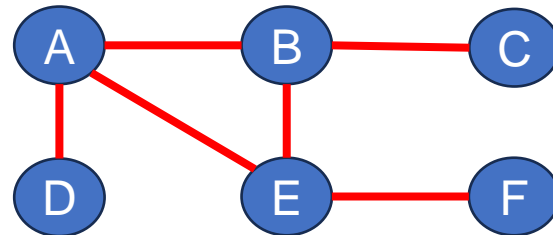
Помеченный граф или взвешенный граф Граф называется помеченным, если каждому ребру в графе назначены некоторые данные. В взвешенном графе ребрам графа назначен некоторый вес или длина. Вес ребра, обозначенный как $w(e)$, является положительным значением, которое указывает стоимость прохождения ребра.

Множественные ребра Различные ребра, которые соединяют одни и те же конечные точки, называются множественными ребрами. То есть $e = (u, v)$ и $e' = (u, v)$ известны как множественные ребра G .

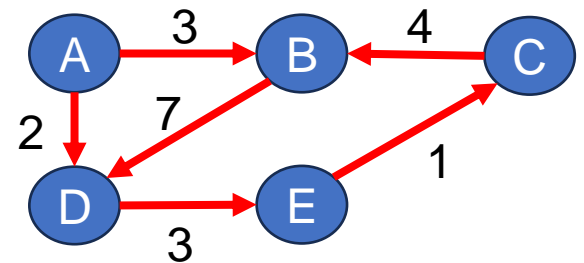
Мультиграф Граф с множественными ребрами и/или петлями называется мультиграфом.



(a) Мультиграф



(b) Дерево



(c) Взвешенный граф

Терминология направленного графа

Исходная степень узла Исходная степень узла u , записанная как $\text{outdeg}(u)$, — это количество ребер, которые начинаются в u .

Входящая степень узла Входящая степень узла u , записанная как $\text{indeg}(u)$, — это количество ребер, которые заканчиваются в u .

Степень узла Степень узла, записанная как $\text{deg}(u)$, равна сумме входящей и исходящей степени этого узла. Следовательно, $\text{deg}(u) = \text{indeg}(u) + \text{outdeg}(u)$.

Изолированная вершина Вершина с нулевой степенью. Такая вершина не является конечной точкой ни одного ребра.

Висячая вершина (также известная как вершина листа) Вершина с единичной степенью.

Срезанная вершина Вершина, удаление которой разорвет оставшийся граф.

Источник Узел u называется источником, если он имеет положительную исходящую степень, но нулевую входящую степень.

Сток Узел u называется стоком, если он имеет положительную входящую степень, но нулевую исходящую степень.

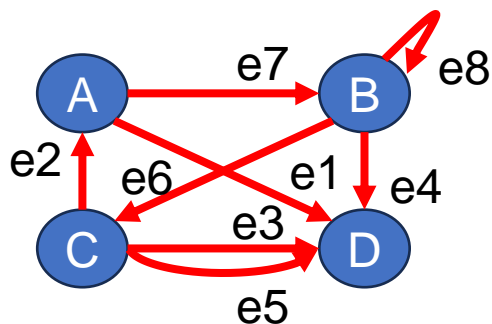
Сильно связанный ориентированный граф Ортаф называется сильно связанным, если и только если существует путь между каждой парой узлов в G . То есть, если существует путь от узла u до v , то должен быть путь от узла v до u .

Односторонне связанный граф Ортаф называется односторонне связанным, если существует путь между любой парой узлов u, v в G такой, что существует путь от u до v или путь от v до u , но не оба.

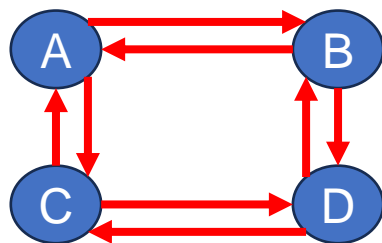
Терминология направленного графа

Параллельные/кратные ребра Различные ребра, которые соединяют одни и те же конечные точки, называются кратными ребрами. То есть $e = (u, v)$ и $e' = (u, v)$ известны как кратные ребра G .

Простой направленный граф Направленный граф G называется простым направленным графом тогда и только тогда, когда он не имеет параллельных ребер. Однако простой направленный граф может содержать циклы, за исключением того, что он не может иметь более одной петли в данном узле.



(a) Направленный
ациклический граф



(b) сильно связанный
направленный
ациклический граф

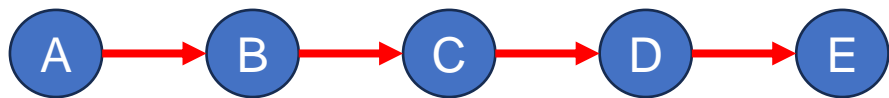


Транзитивное замыкание направленного графа

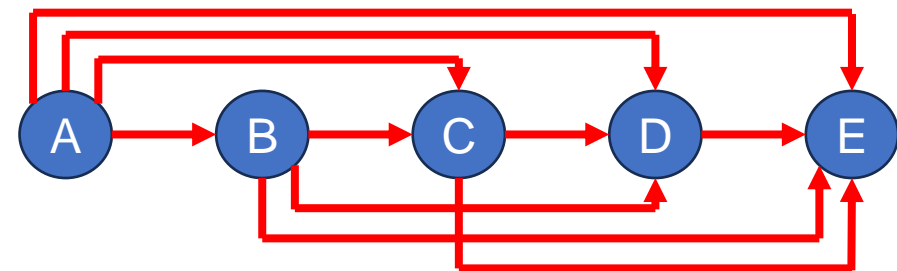
Определение Для направленного графа $G = (V, E)$, где V — множество вершин, а E — множество ребер, транзитивное замыкание G — это граф $G^* = (V, E^*)$. В G^* для каждой пары вершин v, w в V существует ребро (v, w) в E^* тогда и только тогда, когда существует допустимый путь от v до w в G .

Где и зачем это нужно? Нахождение транзитивного замыкания направленного графа является важной проблемой в следующих вычислительных задачах:

- Транзитивное замыкание используется для поиска анализа достижимости сетей переходов, представляющих распределенные и параллельные системы.
- Оно используется при построении автоматов синтаксического анализа при построении компилятора.
- В последнее время вычисление транзитивного замыкания используется для оценки рекурсивных запросов к базе данных (потому что почти все практические рекурсивные запросы являются транзитивными по своей природе).



(a) Граф G



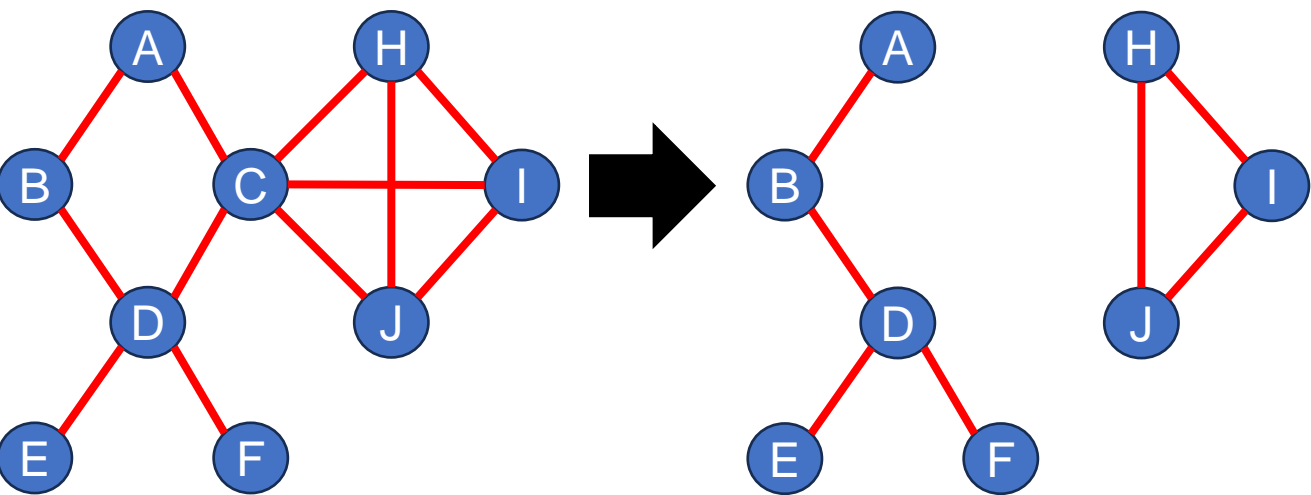
(b) транзитивное замыкание G^*

Двусвязные компоненты

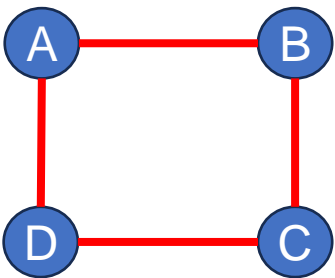
Вершина v графа G называется **точкой сочленения**, если удаление v вместе с ребрами, инцидентными v , приводит к графу, который имеет по крайней мере два связных компонента. **Двусвязный граф** определяется как связный граф, который не имеет вершин сочленения.

Двусвязный неориентированный граф — это связный граф, который нельзя разбить на несвязные части путем удаления любой одной вершины.

В двусвязном ориентированном графе для любых двух вершин v и w существуют два направленных пути из v в w , которые не имеют общих вершин, кроме v и w .



Не двусвязный граф

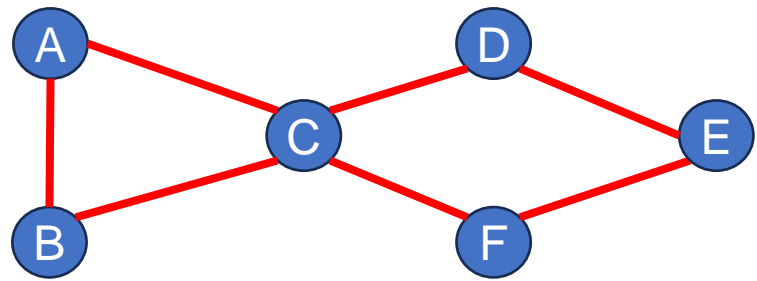


Двусвязный граф

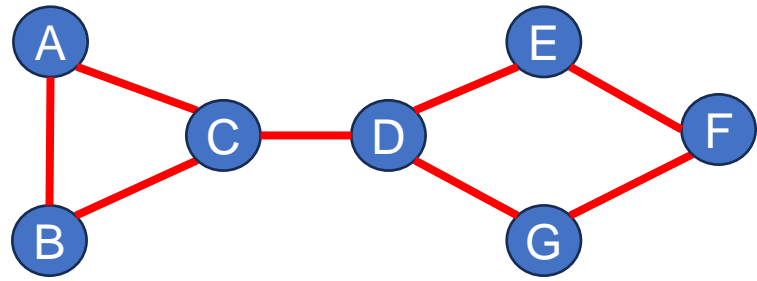


Двусвязные компоненты

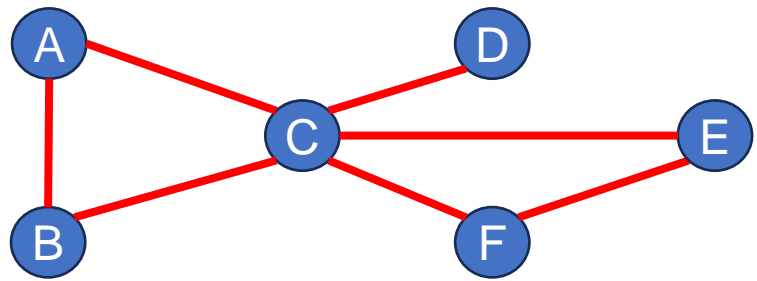
Что касается вершин, то для ребер существует связанное понятие. Ребро в графе называется мостом, если удаление этого ребра приводит к несвязному графу. Кроме того, ребро в графе, которое не лежит на цикле, является мостом. Это означает, что мост имеет по крайней мере одну точку сочленения на своем конце, хотя не обязательно, чтобы точка сочленения была связана с мостом.



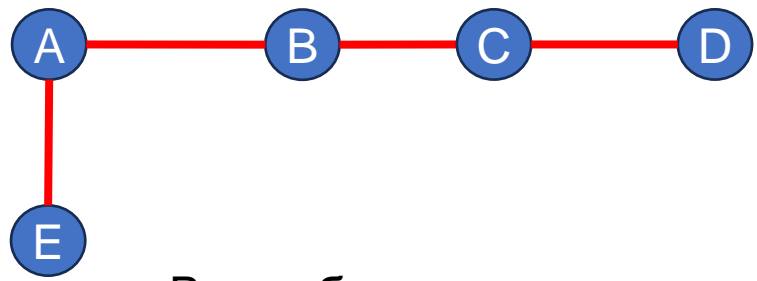
Нет мостов



CD — мост



CD — мост



Все ребра — мосты



Представление графов

Существует три распространенных способа хранения графов в памяти компьютера. Это:

- Последовательное представление с использованием матрицы смежности
- Связанное представление с использованием списка смежности, который хранит соседей узла с помощью связанного списка.
- Мультисписок смежности, который является расширением связанного представления.

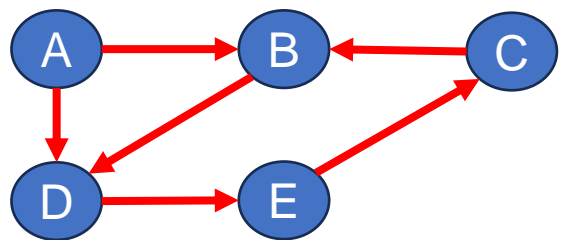


Матрица смежности

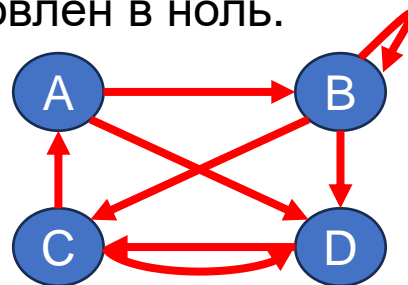
Матрица смежности используется для представления того, какие узлы смежны друг с другом. По определению, два узла считаются смежными, если их соединяет ребро.

В направленном графе G , если узел v смежный с узлом u , то определенно есть ребро от u к v . То есть, если v смежный с u , мы можем добраться от u к v , пройдя по одному ребру. Для любого графа G , имеющего n узлов, матрица смежности будет иметь размерность $n \times n$.

В матрице смежности строки и столбцы помечены вершинами графа. Элемент a_{ij} в матрице смежности будет содержать 1, если вершины v_i и v_j смежны друг с другом. Однако, если узлы не смежны, a_{ij} будет установлен в ноль.



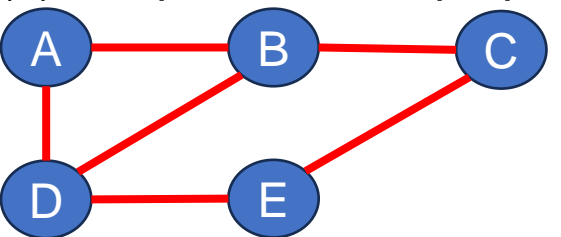
	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	0
C	0	1	0	0	0
D	0	0	0	0	1
E	0	0	1	0	0



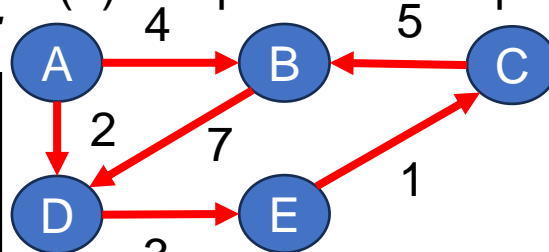
	A	B	C	D
A	0	1	0	1
B	0	1	1	1
C	1	0	0	1
D	0	0	1	0

(a) Направленный граф

(b) Направленный граф с петлей



	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0



	A	B	C	D	E
A	0	4	0	2	0
B	0	0	0	7	0
C	0	5	0	0	0
D	0	0	0	0	3
E	0	0	1	0	0

(c) Ненаправленный граф

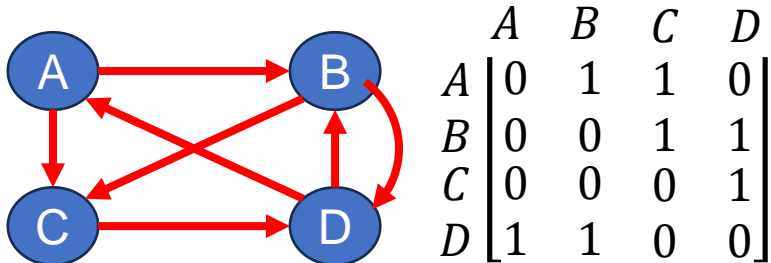
(d) Взвешенный граф

Матрица смежности

Матрица смежности используется для представления того, какие узлы смежны друг с другом. По определению, два узла считаются смежными, если их соединяет ребро.

В направленном графе G , если узел v смежный с узлом u , то определенно есть ребро от u к v . То есть, если v смежный с u , мы можем добраться от u к v , пройдя по одному ребру. Для любого графа G , имеющего n узлов, матрица смежности будет иметь размерность $n \times n$.

В матрице смежности строки и столбцы помечены вершинами графа. Элемент a_{ij} в матрице смежности будет содержать 1, если вершины v_i и v_j смежны друг с другом. Однако, если узлы не смежны, a_{ij} будет установлен в ноль.



Направленный граф с его матрицей смежности

Матрица смежности

Мощности матрицы смежности. Из матрицы смежности A^1 можно сделать вывод, что запись 1 в i -й строке и j -м столбце означает, что существует путь длины 1 от v_i до v_j . Теперь рассмотрим A^2 , A^3 и A^4 .

$$a_{ij}^2 = \sum a_{ik} a_{kj}$$

$$A^2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & 3 \end{bmatrix}$$

$$A^4 = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 3 \\ 3 & 4 & 3 & 4 \end{bmatrix}$$

Теперь, определяем матрицу B как: $B^r = A^1 + A^2 + A^3 + \dots + A^r$

$$B = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 3 & 4 & 2 \\ 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 3 \\ 3 & 4 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 6 & 5 \\ 3 & 5 & 6 & 7 \\ 2 & 3 & 3 & 5 \\ 6 & 8 & 7 & 8 \end{bmatrix}$$

P_{ij} → если есть путь от v_i до v_j
 → иначе

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

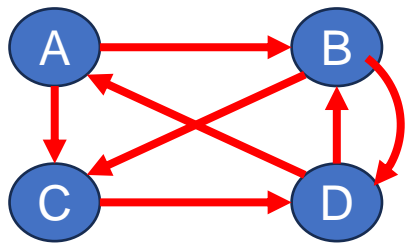
Список смежности

Список смежности — это еще один способ представления графов в памяти компьютера. Эта структура состоит из списка всех узлов в G . Кроме того, каждый узел, в свою очередь, связан со своим собственным списком, который содержит имена всех других узлов, смежных с ним. Основные преимущества использования списка смежности:

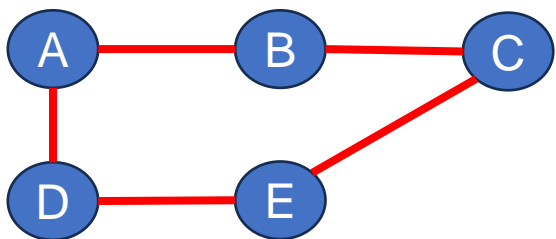
- Он прост для понимания и четко показывает смежные узлы конкретного узла.
- Он часто используется для хранения графов с небольшим или средним количеством ребер. То есть список смежности предпочтителен для представления разреженных графов в памяти компьютера; в противном случае хорошим выбором будет матрица смежности.
- Добавление новых узлов в G легко и просто, когда G представлен с помощью списка смежности. Добавление новых узлов в матрицу смежности — сложная задача, так как необходимо изменить размер матрицы и, возможно, придется переупорядочить существующие узлы.



Список смежности

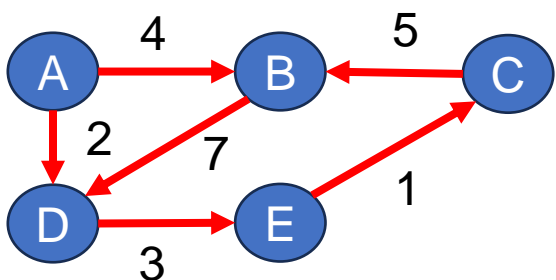


A	→	B		→	C	X
B	→	C		→	D	X
C	→	D	X			
D	→	A		→	B	X



A	→	B	→	D	X	
B	→	A	→	C		D X
C	→	B	→	E	X	
D	→	A	→	B		E X
E	→	C	→	D	X	

Неориентированный граф



A	→	B	4		→	D	2	X
B	→	D	7	X				
C	→	B	5	X				
D	→	E	3	X				
E	→	C	1	X				

Взвешенный граф



Мультисписок смежности

Графы также могут быть представлены с помощью мультисписков, которые можно назвать модифицированной версией списков смежности.

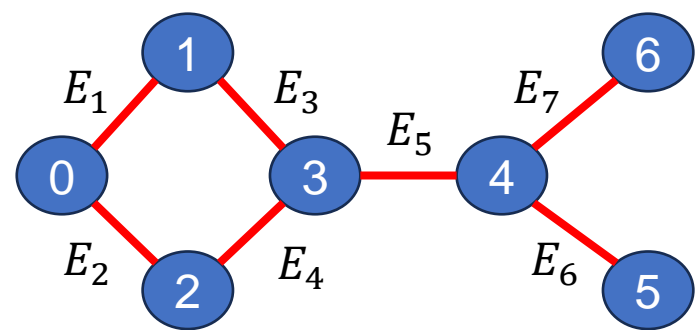
Мультисписок смежности — это представление графов на основе ребер, а не вершин. Мультисписочное представление в основном состоит из двух частей — каталога информации об узлах и набора связанных списков, хранящих информацию о ребрах. Хотя в каталоге узлов есть одна запись для каждого узла, с другой стороны, каждый узел появляется в двух списках смежности (по одному для узла на каждом конце ребра).

Например, запись каталога для узла i указывает на список смежности для узла i . Это означает, что узлы совместно используются несколькими списками.

В мультисписочном представлении информация о ребре (v_i, v_j) неориентированного графа может храниться с использованием следующих атрибутов: M : однобитовое поле для указания того, было ли ребро проверено или нет.



Мультисписок смежности



v_i : Вершина в графе, которая соединена с вершиной v_j ребром.
 v_j : Вершина в графе, которая соединена с вершиной v_i ребром.
Ссылка i для v_i : Ссылка, которая указывает на другой узел, имеющий ребро, инцидентное v_i .
Ссылка j для v_i : Ссылка, которая указывает на другой узел, имеющий ребро, инцидентное v_j .

Ребро 1		0	1	Ребро 2	Ребро 3
Ребро 2		0	2	NULL	Ребро 4
Ребро 3		1	3	NULL	Ребро 4
Ребро 4		2	3	NULL	Ребро 5
Ребро 5		3	4	NULL	Ребро 6
Ребро 6		4	5	Ребро 7	NULL
Ребро 7		4	6	NULL	NULL

Вершины	Список ребер
0	Ребро 1, Ребро 2
1	Ребро 1, Ребро 3
2	Ребро 2, Ребро 4
3	Ребро 3, Ребро 4, Ребро 5
4	Ребро 5, Ребро 6, Ребро 7
5	Ребро 6
6	Ребро 7

Алгоритмы обхода графа

Как обходить графы?

Под обходом графа мы подразумеваем метод изучения узлов и ребер графа. Существует два стандартных метода обхода графа, которые мы обсудим в этом разделе. Эти два метода:

1. Поиск в ширину
2. Поиск в глубину

В то время как поиск в ширину использует очередь в качестве вспомогательной структуры данных для хранения узлов для дальнейшей обработки, схема поиска в глубину использует стек. Но оба эти алгоритма используют переменную STATUS. Во время выполнения алгоритма каждый узел в графе будет иметь переменную STATUS, установленную на 1 или 2, в зависимости от его текущего состояния.

Статус	Состояние узла	Описание
1	Готов	Начальное состояние узла N
2	Ожидание	Узел N помещается в очередь или стек и ожидает обработки
3	Обработан	Узел N полностью обработан



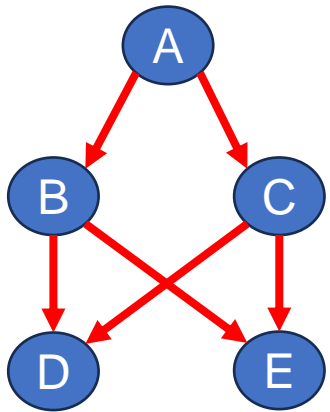
Топологическая сортировка

Топологическая сортировка направленного ациклического графа (DAG) G определяется как линейное упорядочение его узлов, в котором каждый узел предшествует всем узлам, к которым у него есть исходящие ребра. Каждый DAG имеет одно или несколько топологических сортировок.

Топологическая сортировка DAG G — это упорядочение вершин G , такое что если G содержит ребро (u, v) , то u появляется перед v в упорядочении. Обратите внимание, что топологическая сортировка возможна только для направленных ациклических графов, которые не имеют циклов. Для DAG, содержащего циклы, линейное упорядочение его вершин невозможно. Проще говоря, топологическое упорядочение DAG G — это упорядочение его вершин, такое что любой направленный путь в G проходит вершины в порядке возрастания. Топологическая сортировка широко используется при планировании приложений, заданий или задач. Задания, которые должны быть выполнены, представлены узлами, и есть ребро от узла u к v , если задание u должно быть выполнено до того, как задание v может быть запущено. Топологическая сортировка такого графа дает порядок, в котором данные задания должны быть выполнены.

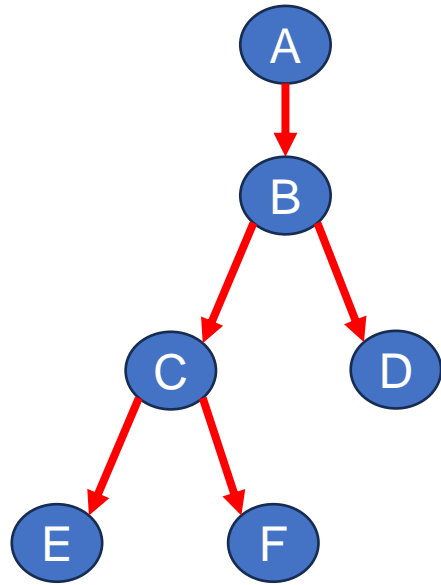


Топологическая сортировка



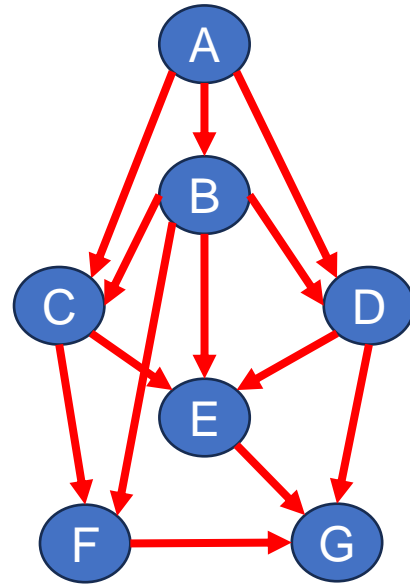
Топологическая сортировка может быть задана как:

- A, B, C, D, E
- A, B, C, E, D
- A, C, B, D, E
- A, C, B, E, D



Топологическая сортировка может быть задана как:

- A, B, D, C, E, F
- A, B, D, C, F, E
- A, B, C, D, E, F
- A, B, C, D, F, E



Топологическая сортировка может быть задана как:

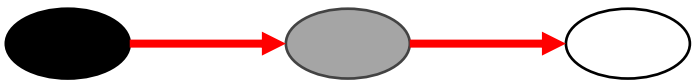
- A, B, C, F, D, E, G
- A, B, C, D, E, F, G
- A, B, C, D, F, E, G
- A, B, D, C, E, F, G





Новая информация

Сильно связанные компоненты



Теорема (Теорема о белом пути)

В лесу поиска в глубину (ориентированного или неориентированного) графа $G = (V, E)$ вершина v является потомком вершины u тогда, и только тогда, когда в момент времени $u.d$ (открытие вершины u) вершина v достижима из u по пути, состоящему только из белых вершин. (без доказательства)

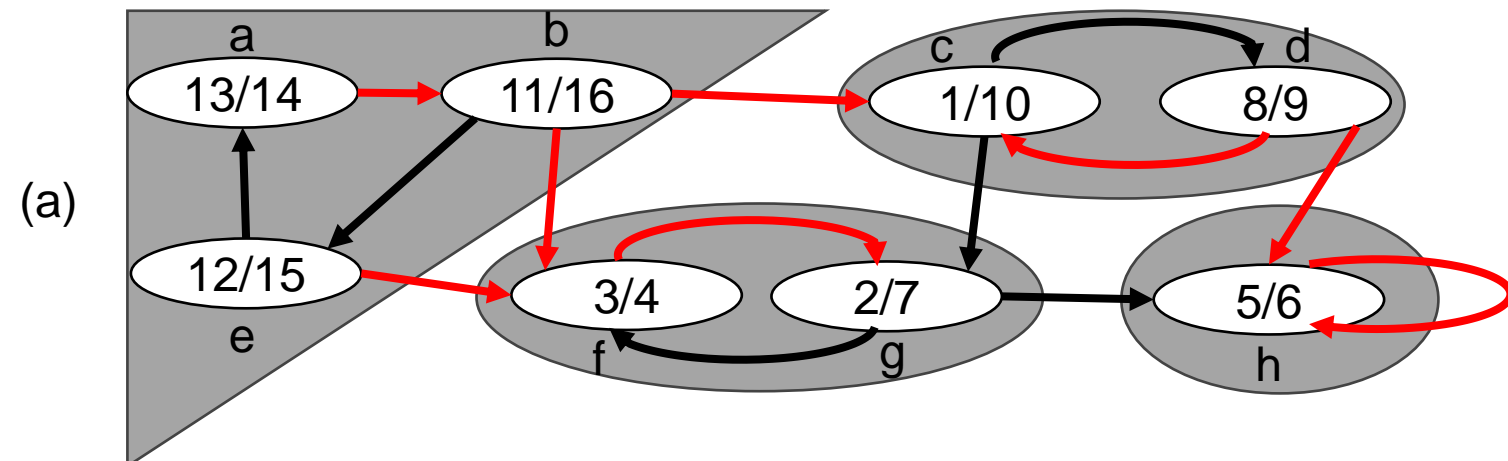
Теперь рассмотрим классическое применение поиска в глубину: разложение ориентированного графа на сильно связанные компоненты. Как выполнить такое разложение с помощью двух поисков в глубину?

Ряд алгоритмов для работы с графами начинается с выполнения такого разложения графа, и после разложения алгоритм работает с каждым сильно связным компонентом отдельно. Полученные решения затем комбинируются в соответствии со структурой связей между сильно связными компонентами.

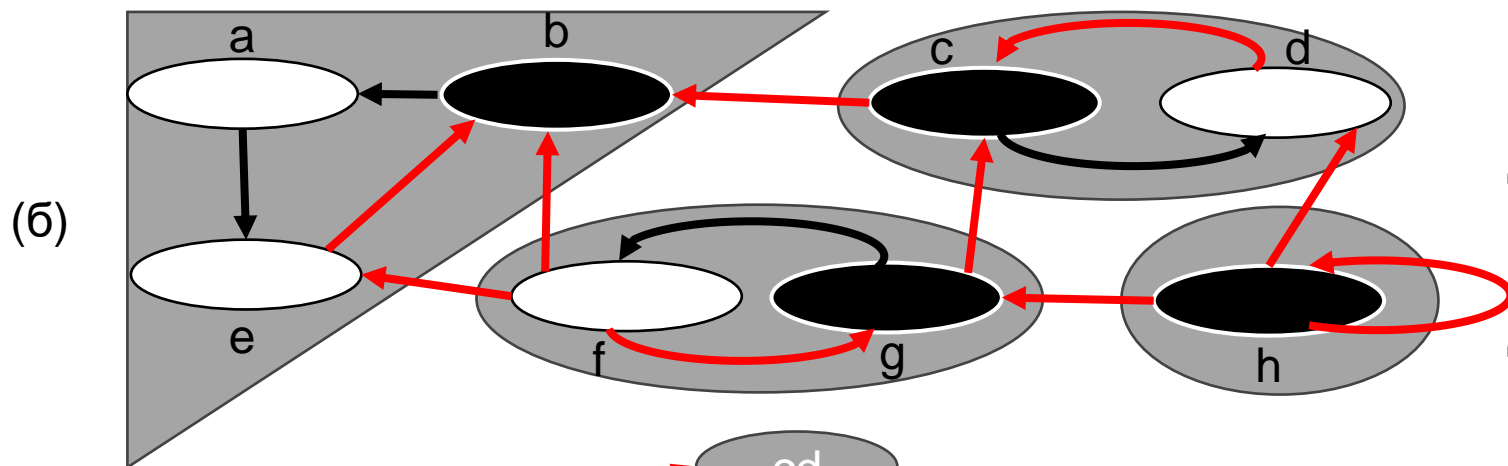
Сильно связный компонент ориентированного графа $G = (V, E)$ представляет собой максимальное множество вершин $S \subseteq V$, такое, что для каждой пары вершин u и v из S справедливо как $u \rightsquigarrow v$, так и $v \rightsquigarrow u$, т.е. вершины u и v достижимы одна из другой.



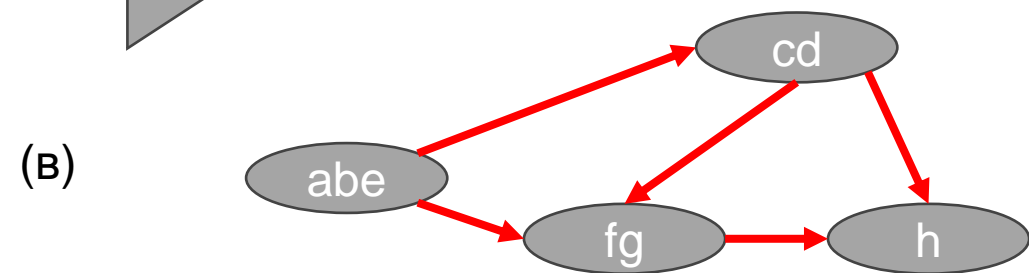
Сильно связанные компоненты



(а) Ориентированный граф G . Каждая заштрихованная область представляет собой сильно связный компонент G . Каждая вершина помечена ее временем открытия и завершения при поиске в глубину.



(б) Граф G^T , транспонированный G . Каждый сильно связный компонент соответствует одному дереву поиска в глубину. Вершины b, c, g и h, выделенные темным цветом, представляют собой корни деревьев поиска в глубину, сгенерированных поиском в глубину в графе G^T .



(в) Ациклический граф компонентов G^{SCC} , полученный путем сжатия всех ребер в пределах сильно связанных компонентов G , так что в каждом компоненте остается только по одной вершине.

СИЛЬНО СВЯЗНЫЕ КОМПОНЕНТЫ

```

void StronglyConnectedComponents(Graph* g) {
    for (int i = 0; i < g->V; i++)
        visited[i] = 0;
    time = 0;
    for (int u = 0; u < g->V; u++)
        if (!visited[u])
            DFS1(g, u); // Шаг 1: Первый DFS
    Graph* gt = transpose(g); // Шаг 2: Построение транспонированного графа
    for (int i = 0; i < g->V; i++)
        visited[i] = 0;
    printf("Сильно связанные компоненты:\n");
    for (int i = g->V - 1; i >= 0; i--) {
        int u = -1;
        for (int j = 0; j < g->V; j++) {
            if (finishing_times[j] == i && !visited[j]) {
                u = j;
                break;
            }
        }
        if (u != -1) {
            printf("SCC: ");
            DFS2(gt, u); // Шаг 3: Второй DFS на GT
        }
    }
    free(gt);
}

```

Сильно связанные компоненты



Теорема (Теорема о белом пути)

В лесу поиска в глубину (ориентированного или неориентированного) графа $G = (V, E)$ вершина v является потомком вершины u тогда, и только тогда, когда в момент времени $u.d$ (открытие вершины u) вершина v достижима из u по пути, состоящему только из белых вершин. (без доказательства)

Теперь рассмотрим классическое применение поиска в глубину: разложение ориентированного графа на сильно связанные компоненты. Как выполнить такое разложение с помощью двух поисков в глубину?

Ряд алгоритмов для работы с графами начинается с выполнения такого разложения графа, и после разложения алгоритм работает с каждым сильно связным компонентом отдельно. Полученные решения затем комбинируются в соответствии со структурой связей между сильно связными компонентами.

Сильно связный компонент ориентированного графа $G = (V, E)$ представляет собой максимальное множество вершин $S \subseteq V$, такое, что для каждой пары вершин u и v из S справедливо как $u \rightsquigarrow v$, так и $v \rightsquigarrow u$, т.е. вершины u и v достижимы одна из другой.



Алгоритмы кратчайшего пути

Мы обсудим три различных алгоритма для вычисления кратчайшего пути между вершинами графа G . Эти алгоритмы включают в себя:

- Минимальное остовное дерево
- Алгоритм Дейкстры
- Алгоритм Уоршалла

В то время как первые два используют список смежности для поиска кратчайшего пути, алгоритм Уоршалла использует матрицу смежности для того же самого.



Минимальные остовные деревья

Остовное дерево связного неориентированного графа G — это подграф G , который является деревом, соединяющим все вершины вместе. Граф G может иметь много различных остовных деревьев. Мы можем назначить веса каждому ребру (это число, которое представляет, насколько неблагоприятно ребро) и использовать его для назначения веса остовному дереву путем вычисления суммы весов ребер в этом остовном дереве. Минимальное остовное дерево (MST) определяется как остовное дерево с весом, меньшим или равным весу любого другого остовного дерева. Другими словами, **минимальное остовное дерево** — это остовное дерево, у которого есть веса, связанные с его ребрами, и общий вес дерева (сумма весов его ребер) минимален.

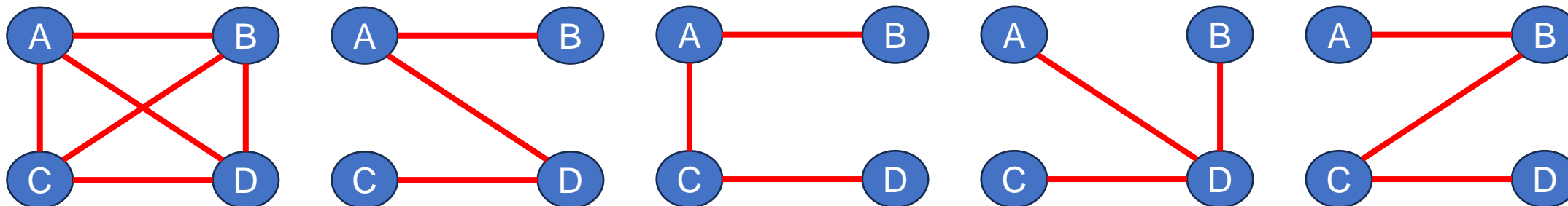
Аналогия

Возьмем аналогию с кабельной телевизионной компанией, прокладывающей кабель в новом районе. Если она ограничена прокладкой кабеля только вдоль определенных путей, то мы можем создать граф, представляющий точки, которые соединены этими путями. Некоторые пути могут быть более дорогими (из-за их длины или глубины, на которой должен быть зарыт кабель), чем другие. Мы можем представить эти пути ребрами с большими весами. Следовательно, остовное дерево для такого графа будет подмножеством тех путей, которое не имеет циклов, но все еще соединяется с каждым домом.

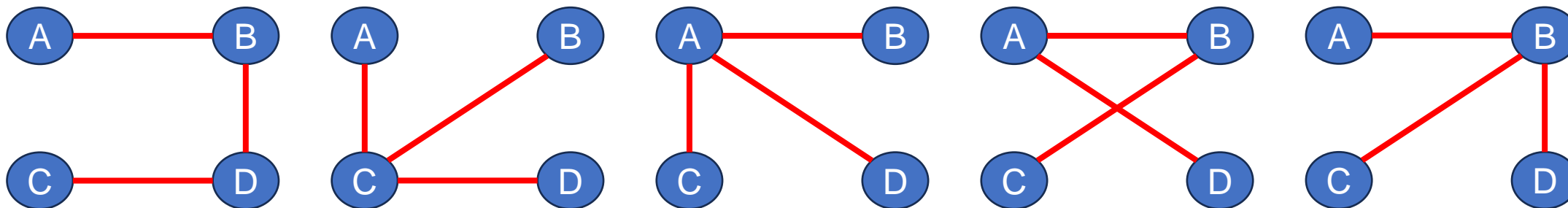
Минимальные остовные деревья - Свойства

- **Возможная множественность** Может быть несколько минимальных остовных деревьев одинакового веса. В частности, если все веса одинаковы, то каждое остовное дерево будет минимальным.
- **Уникальность** Когда каждому ребру в графе назначен разный вес, то будет только одно уникальное минимальное остовное дерево.
- **Подграф минимальной стоимости** Если ребрам графа назначены неотрицательные веса, то минимальное остовное дерево фактически является подграфом минимальной стоимости или деревом, которое соединяет все вершины.
- **Свойство цикла** Если в графе G существует цикл C , вес которого больше, чем у других ребер C , то это ребро не может принадлежать MST.
- **Полезность** Минимальные остовные деревья можно вычислить быстро и легко, чтобы предоставить оптимальные решения. Эти деревья создают разреженный подграф, который многое отражает в исходном графе.
- **Простота** Минимальное остовное дерево взвешенного графа — это не что иное, как остовное дерево графа, состоящее из $n-1$ ребер минимального общего веса.

Минимальные остовные деревья - Пример

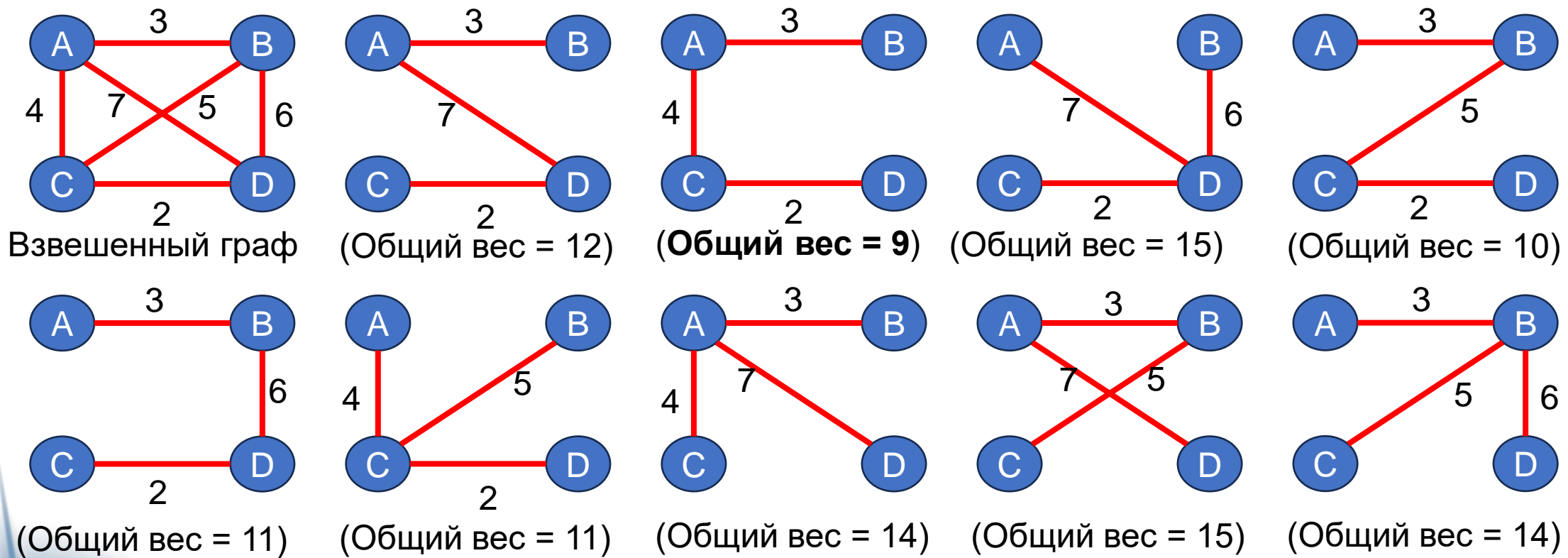


Невзвешенный граф



Невзвешенный граф и его остовные деревья

Минимальные остовные деревья - Пример



Невзвешенный граф и его остовные деревья

Применение минимальных остовных деревьев

1. MST широко используются для проектирования сетей. Например, люди, разделенные разным расстоянием, хотят быть соединены друг с другом через телефонную сеть. Минимальное остовное дерево используется для определения наименее затратных путей без циклов в этой сети, тем самым обеспечивая соединение с минимальной стоимостью.
2. MST используются для поиска маршрутов авиалиний. В то время как вершины в графе обозначают города, ребра представляют маршруты между этими городами. Несомненно, чем больше расстояние между городами, тем выше будет взимаемая сумма. Поэтому MST используются для оптимизации маршрутов авиалиний путем поиска наименее затратного пути без циклов.
3. MST также используются для поиска самого дешевого способа соединения терминалов, таких как города, электронные компоненты или компьютеры через дороги, авиалинии, железные дороги, провода или телефонные линии.
4. MST применяются в алгоритмах маршрутизации для поиска наиболее эффективного пути.

Алгоритм Прима

Алгоритм Прима — это жадный алгоритм, который используется для формирования минимального остовного дерева для связного взвешенного неориентированного графа. Другими словами, алгоритм строит дерево, которое включает каждую вершину и подмножество ребер таким образом, чтобы общий вес всех ребер в дереве был минимизирован. Для этого алгоритм поддерживает три набора вершин, которые могут быть заданы следующим образом:

- **Вершины дерева** Вершины, которые являются частью минимального остовного дерева T .
- **Вершины края** Вершины, которые в данный момент не являются частью T , но смежны с некоторой вершиной дерева.
- **Невидимые вершины** Вершины, которые не являются ни вершинами дерева, ни вершинами края, попадают в эту категорию.

Шаги, включенные в алгоритм Прима.

- Выберите начальную вершину.
- Выйдите из начальной вершины и во время каждой итерации выберите новую вершину и ребро. По сути, во время каждой итерации алгоритма мы должны выбрать вершину из вершин края таким образом, чтобы ребро, соединяющее вершину дерева и новую вершину, имело минимальный назначенный ему вес.

Время выполнения алгоритма Прима можно задать как $O(E \log V)$, где E — количество ребер, а V — количество вершин в графе.



Алгоритм Прима

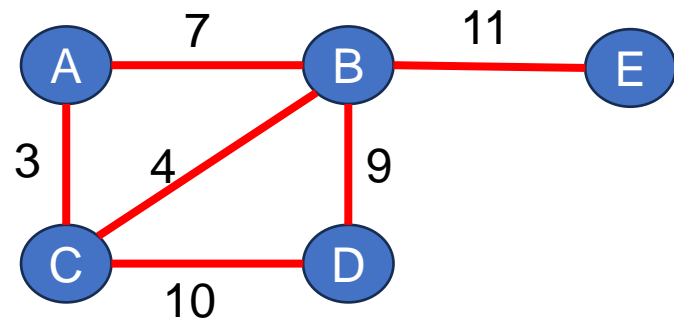
Шаг 1: Выберите начальную вершину

Шаг 2: Повторите шаги 3 и 4, пока не появятся вершины края

Шаг 3: Выберите ребро, соединяющее вершину дерева и вершину края, имеющее минимальный вес

Шаг 4: Добавьте выбранное ребро и вершину в минимальное остовное дерево T
[КОНЕЦ ЦИКЛА]

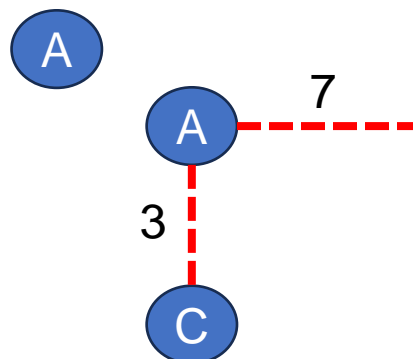
Шаг 5: ВЫХОД



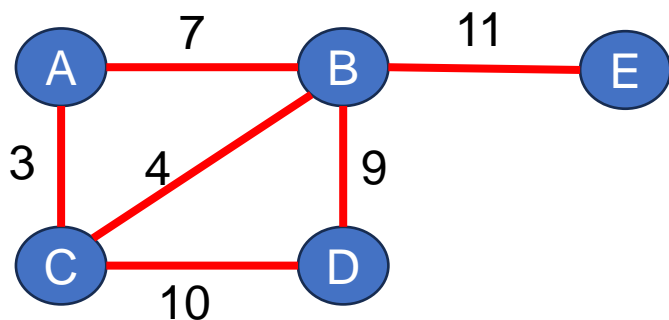
Построим минимальное
остовное дерево графа

Шаг 1: Выберите начальную вершину A.

Шаг 2: Добавьте вершины края (которые смежны с A). Ребра, соединяющие вершину и вершины края, показаны пунктирными линиями.



Алгоритм Прима

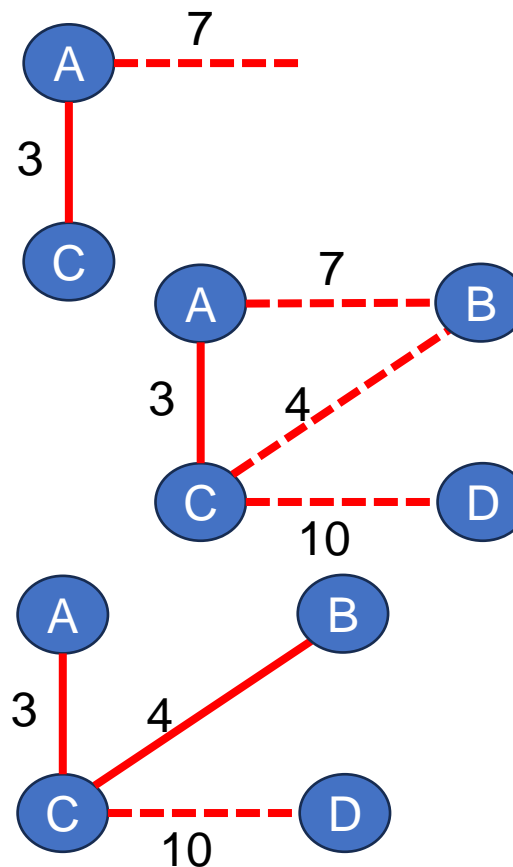


Построим минимальное остовное дерево графа

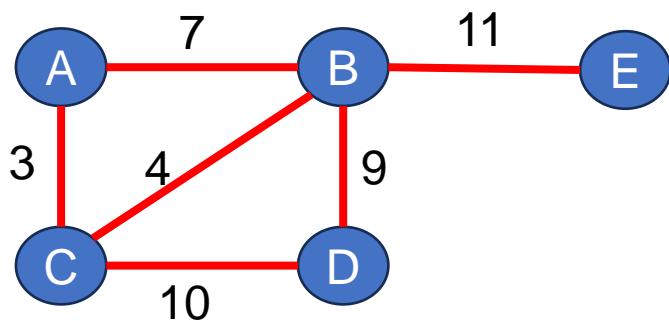
Шаг 3: Выберите ребро, соединяющее вершину дерева и вершину края, имеющее минимальный вес, и добавьте выбранное ребро и вершину в минимальное остовное дерево T . Поскольку ребро, соединяющее A и C , имеет меньший вес, добавьте C в дерево. Теперь C не вершина края, а вершина дерева.

Шаг 4: Добавьте вершины края (которые смежны с C).

Шаг 5: Выберите ребро, соединяющее вершину дерева и вершину края, имеющее минимальный вес, и добавьте выбранное ребро и вершину в минимальное остовное дерево T . Поскольку ребро, соединяющее C и B , имеет меньший вес, добавьте B в дерево. Теперь B не является вершиной края, а вершиной дерева.



Алгоритм Прима

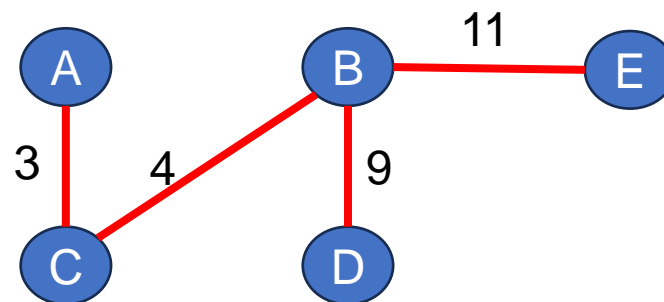
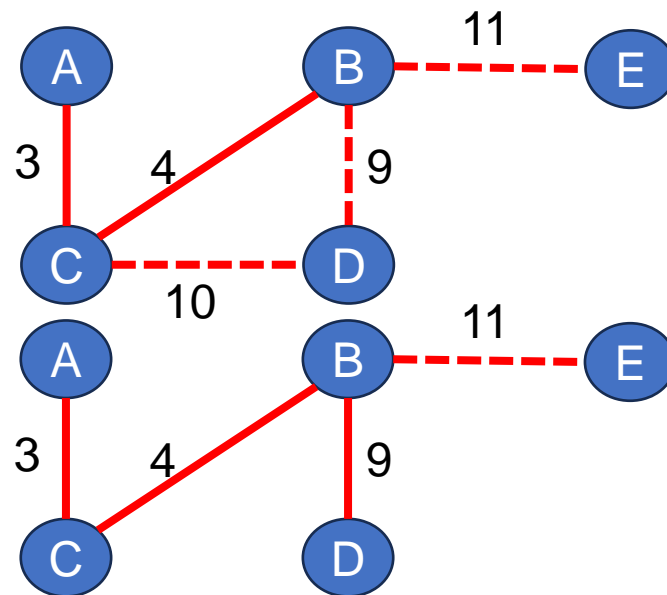


Построим минимальное остовное дерево графа

Шаг 6: Добавьте вершины края (которые являются смежными с B).

Шаг 7: Выберите ребро, соединяющее вершину дерева и вершину края, имеющую минимальный вес, и добавьте выбранное ребро и вершину в минимальное остовное дерево T. Поскольку ребро, соединяющее B и D, имеет меньший вес, добавьте D в дерево. Теперь D не является вершиной края, а вершиной дерева.

Шаг 8: Обратите внимание, теперь узел E не подключен, поэтому мы добавим его в дерево, поскольку минимальное остовное дерево — это то, в котором все n узлов соединены с $n-1$ ребрами, имеющими минимальный вес. Таким образом, минимальное остовное дерево теперь можно задать как,





Алгоритм Краскала (Крускала)

Алгоритм Краскала используется для поиска минимального остовного дерева для связного взвешенного графа. Целью алгоритма является поиск подмножества ребер, которое образует дерево, включающее каждую вершину. Общий вес всех ребер в дереве минимизируется. Однако, если граф не связан, то он находит минимальный остовный лес. Обратите внимание, что лес — это набор деревьев. Аналогично, минимальный остовный лес — это набор минимальных остовных деревьев.

Шаг 1: Создайте лес таким образом, чтобы каждый граф был отдельным деревом.

Шаг 2: Создайте очередь приоритетов Q , которая содержит все ребра графа.

Шаг 3: Повторите шаги 4 и 5, пока Q НЕ ПУСТО

Шаг 4: Удалите ребро из Q

Шаг 5: ЕСЛИ ребро, полученное на шаге 4, соединяет два разных дерева, то добавьте его в лес (для объединения двух деревьев в одно).

ИНАЧЕ Удалите ребро

Шаг 6: КОНЕЦ

В алгоритме мы используем приоритетную очередь Q , в которой ребра с минимальным весом имеют приоритет над любым другим ребром в графе. Когда алгоритм Крускала завершается, лес имеет только один компонент и образует минимальное остовное дерево графа. Время выполнения алгоритма Крускала можно задать как $O(E \log V)$, где E — количество ребер, а V — количество вершин в графе.



Алгоритм Краскала

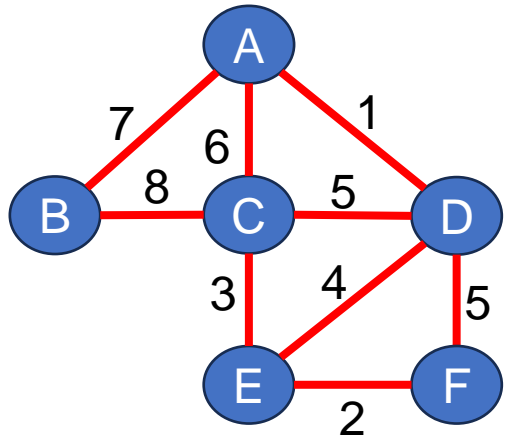
Применим алгоритм Краскала к графу.

Изначально у нас есть

$F = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}\}$

$MST = \{\}$

$Q = \{(A, D), (E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$



Шаг 1: Удалим ребро **(A, D)** из Q и сделаем следующие изменения:

$F = \{\{A, D\}, \{B\}, \{C\}, \{E\}, \{F\}\}$

$MST = \{A, D\}$

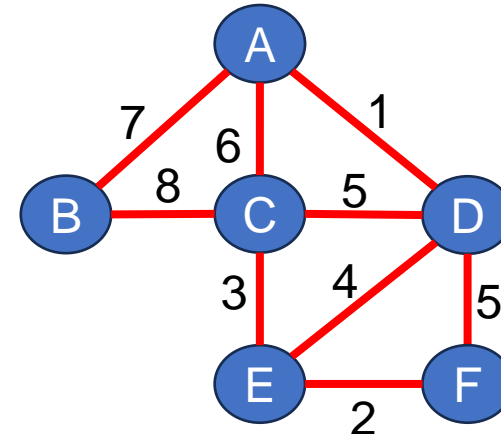
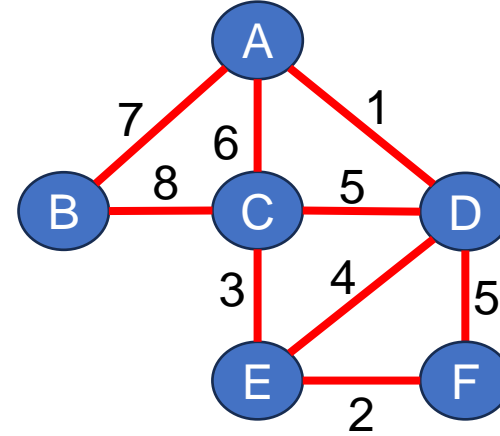
$Q = \{(E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$

Шаг 2: Удалим ребро **(E, F)** из Q и сделаем следующие изменения:

$F = \{\{A, D\}, \{B\}, \{C\}, \{E, F\}\}$

$MST = \{(A, D), (E, F)\}$

$Q = \{(C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$



Алгоритм Краскала

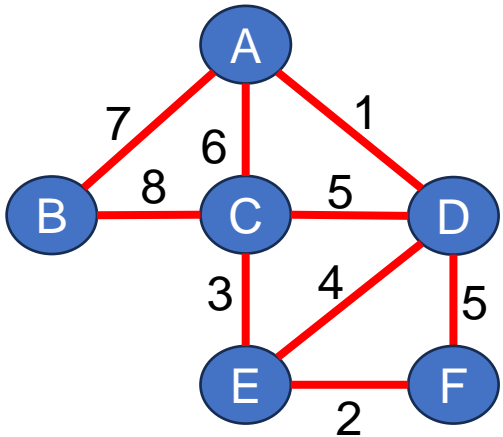
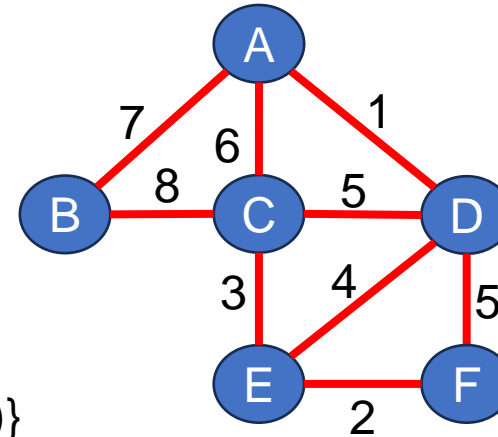
Применим алгоритм Краскала к графу.

Изначально у нас есть

$F = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}\}$

$MST = \{\}$

$Q = \{(A, D), (E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$



Шаг 3: Удалите ребро **(C, E)** из Q и внесите следующие изменения:

$F = \{\{A, D\}, \{B\}, \{C, E, F\}\}$

$MST = \{(A, D), (C, E), (E, F)\}$

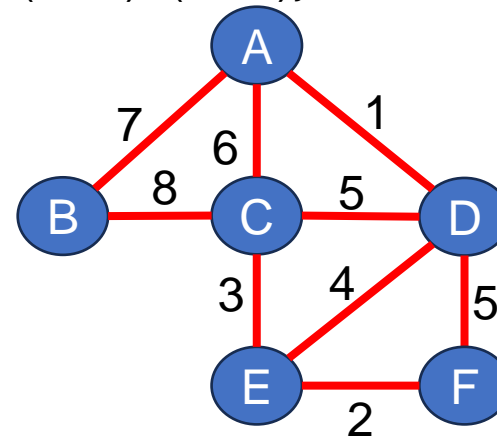
$Q = \{(E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$

Шаг 4: Удалите ребро **(E, D)** из Q и внесите следующие изменения:

$F = \{\{A, C, D, E, F\}, \{B\}\}$

$MST = \{(A, D), (C, E), (E, F), (E, D)\}$

$Q = \{(C, D), (D, F), (A, C), (A, B), (B, C)\}$



Алгоритм Краскала

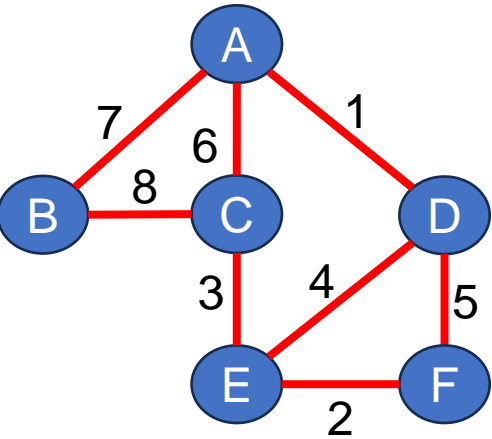
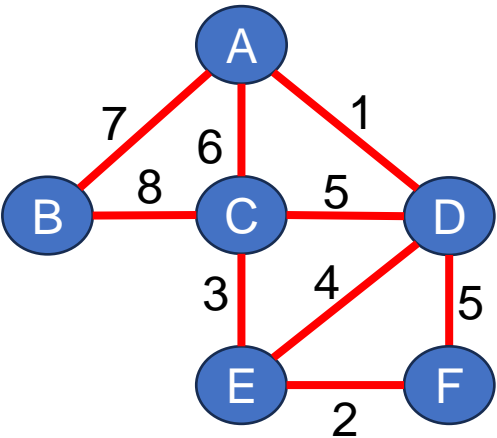
Применим алгоритм Краскала к графу.

Изначально у нас есть

$F = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}\}$

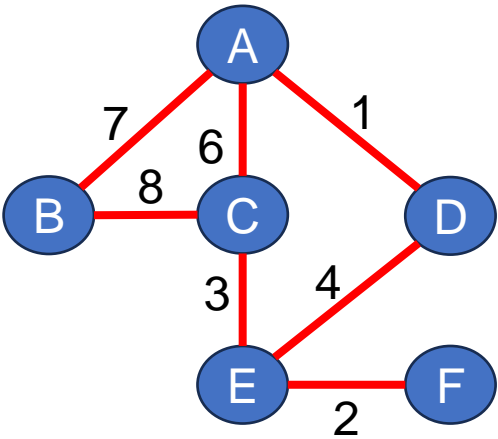
$MST = \{\}$

$Q = \{(A, D), (E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$



Шаг 5: Удалите ребро **(C, D)** из Q. Обратите внимание, что это ребро не соединяет разные деревья, поэтому просто отбросьте это ребро. В MST будет добавлено только ребро, соединяющее (A, D, C, E, F) с B. Следовательно,
 $F = \{\{A, C, D, E, F\}, \{B\}\}$
 $MST = \{(A, D), (C, E), (E, F), (E, D)\}$ $Q = \{(D, F), (A, C), (A, B), (B, C)\}$

Шаг 6: Удалите ребро (D, F) из Q. Обратите внимание, что это ребро не соединяет разные деревья, поэтому просто отбросьте это ребро. В MST будет добавлено только ребро, соединяющее (A, D, C, E, F) с B.
 $F = \{\{A, C, D, E, F\}, \{B\}\}$
 $MST = \{(A, D), (C, E), (E, F), (E, D)\}$
 $Q = \{(A, C), (A, B), (B, C)\}$



Алгоритм Краскала

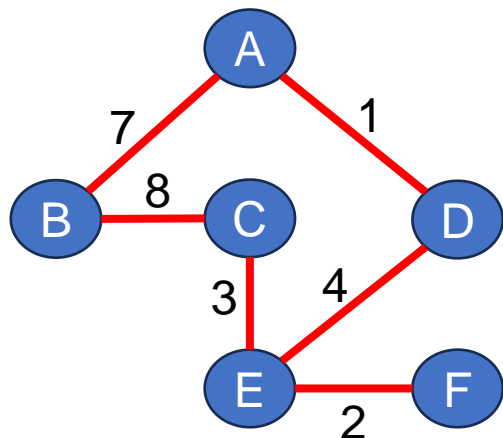
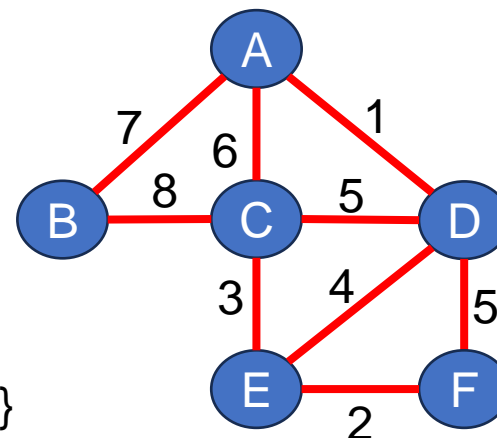
Применим алгоритм Краскала к графу.

Изначально у нас есть

$F = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}\}$

$MST = \{\}$

$Q = \{(A, D), (E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$



Шаг 7: Удалите ребро **(A, C)** из Q . Обратите внимание, что это ребро не соединяет разные деревья, поэтому просто отбросьте это ребро. Только ребро, соединяющее (A, D, C, E, F) с B , будет добавлено в MST .

$F = \{\{A, C, D, E, F\}, \{B\}\}$

$MST = \{(A, D), (C, E), (E, F), (E, D)\}$

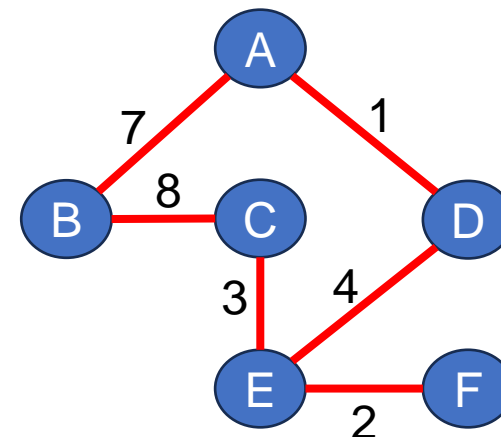
$Q = \{(A, B), (B, C)\}$

Шаг 8: Удалить ребро **(A, B)** из Q и внести следующие изменения:

$F = \{A, B, C, D, E, F\}$

$MST = \{(A, D), (C, E), (E, F), (E, D), (A, B)\}$

$Q = \{(B, C)\}$



Алгоритм Краскала

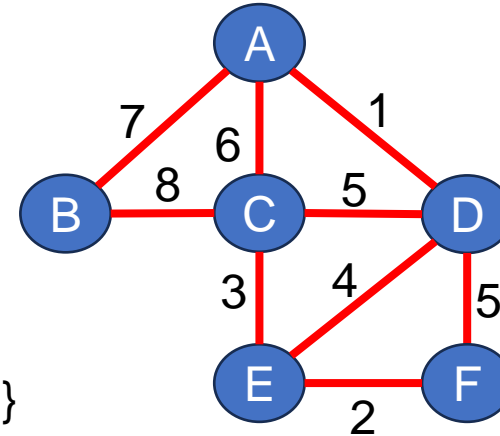
Применим алгоритм Краскала к графу.

Изначально у нас есть

$F = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}\}$

$MST = \{\}$

$Q = \{(A, D), (E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)\}$

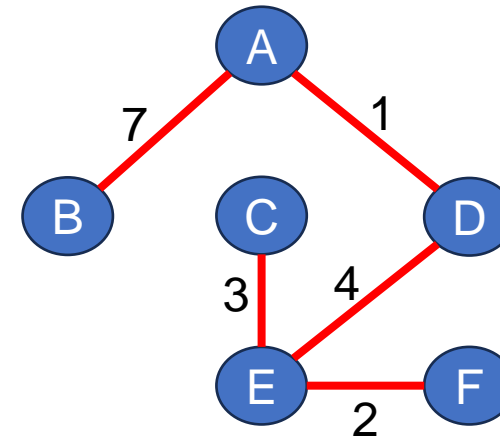


Шаг 9: Алгоритм продолжается до тех пор, пока Q не станет пустым. Поскольку весь лес стал одним деревом, все оставшиеся ребра будут просто отброшены. Результирующее MS может быть представлено, как показано ниже.

$F = \{A, B, C, D, E, F\}$

$MST = \{(A, D), (C, E), (E, F), (E, D), (A, B)\}$

$Q = \{\}$



Кратчайшие пути из одной вершины

Водителю автомобиля нужно найти самый короткий путь из Москвы в Новосибирск. Допустим, у него есть карта России, на которой указаны расстояния между каждой парой пересечений дорог. Как найти кратчайший маршрут?

Один из возможных способов - пронумеровать все маршруты из Москвы в Новосибирск, просуммировать длины участков на каждом маршруте и выбрать кратчайший из них. Однако легко понять, что даже если исключить маршруты, содержащие циклы, получится очень много вариантов, большинство которых просто не имеет смысла рассматривать. Например, очевидно, что маршрут из Москвы в Новосибирск через Краснодар - далеко не лучший выбор. Точнее говоря, такой маршрут никуда не годится, потому что Краснодар находится относительно Москвы совсем в другой стороне.

Как эффективно решаются такие задачи? В задаче о кратчайшем пути (shortest-paths problem) задается взвешенный ориентированный граф $G = (V, E)$ с весовой функцией $w : E \rightarrow \mathbb{R}$, отображающей ребра на их веса, значения которых выражаются действительными числами. Вес (weight) пути $p = \langle v_0, v_1, \dots, v_k \rangle$ равен суммарному весу входящих в него ребер:

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i).$$

Кратчайшие пути из одной вершины

Вес кратчайшего пути (shortest-path weight) $\delta(u, v)$ из вершины u в вершину v определяется соотношением

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \rightsquigarrow v\}, & \text{если существует путь из } u \text{ в } v, \\ \infty & \text{в противном случае.} \end{cases}$$

Тогда по определению **кратчайший путь** (shortest path) из вершины u в вершину v - это любой путь, вес которого удовлетворяет соотношению $w(p) = \delta(u, v)$.

В примере, в котором рассматривается маршрут из Москвы в Новосибирск, карту дорог можно смоделировать в виде графа, вершины которого представляют перекрестки дорог, а ребра - отрезки дорог между перекрестками, причем вес каждого ребра равен расстоянию между соответствующими перекрестками. Цель - найти кратчайший путь от заданного перекрестка в Москве к заданному перекрестку в Новосибирске (скажем, между улицами Пирогова и университетским проспектом).

Вес каждого из ребер можно интерпретировать не как расстояние, а как другую метрику. Часто они используются для представления временных интервалов, стоимости, штрафов, убытков или любой другой величины, которая линейно накапливается по мере продвижения вдоль ребер графа и которую нужно свести к минимуму.

Алгоритм поиска в ширину, представляет собой алгоритм поиска кратчайшего пути в невзвешенном графе, т.е. в графе, каждому ребру которого приписывается единичный вес. Поскольку многие концепции, применяемые в алгоритме поиска в ширину, возникают при исследовании задачи о кратчайшем пути по взвешенным графам.

Кратчайшие пути из одной вершины

Варианты

Есть задача о кратчайших путях из одной вершины (single-source shortest-paths problem), в которой для заданного графа $G = (V, E)$ требуется найти кратчайшие пути, которые начинаются в определенной исходной вершине (source vertex) $s \in V$ (для краткости будем именовать ее истоком) и заканчиваются в каждой из вершин $v \in V$. Предназначенный для решения этой задачи алгоритм позволяет решать многие другие задачи, в том числе перечисленные ниже.

Задача о кратчайших путях в одну вершину. Требуется найти кратчайшие пути в заданную целевую вершину (destination vertex) t , которые начинаются в каждой из вершин v . Поменяв направление каждого принадлежащего графу ребра, эту задачу можно свести к задаче о единой исходной вершине.

Задача о кратчайшем пути между заданной парой вершин. Требуется найти кратчайший путь из заданной вершины u в заданную вершину v . Если решена задача поиска кратчайших путей из заданной исходной вершины u , то эта задача также решается. Более того, все известные для решения данной задачи алгоритмы имеют то же время работы в наихудшем случае, что и наилучшие алгоритмы поиска кратчайших путей из одной вершины.

Задача о кратчайшем пути между всеми вершинами. Требуется найти кратчайший путь из каждой вершины u в каждую вершину v . Эту задачу также можно решить с помощью алгоритма, предназначенного для решения задачи об одной исходной вершине, однако обычно она решается быстрее. Кроме того, структура этой задачи представляет интерес сама по себе.

Кратчайшие пути из одной вершины

Оптимальная подструктура кратчайших путей

Алгоритмы поиска кратчайших путей обычно основаны на том свойстве, что кратчайший путь между двумя вершинами содержит в себе другие кратчайшие пути. В сформулированной ниже лемме данное свойство оптимальной структуры определяется более точно.

Лемма (Подпути кратчайших путей есть кратчайшие пути)

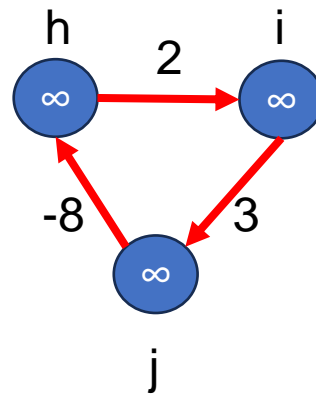
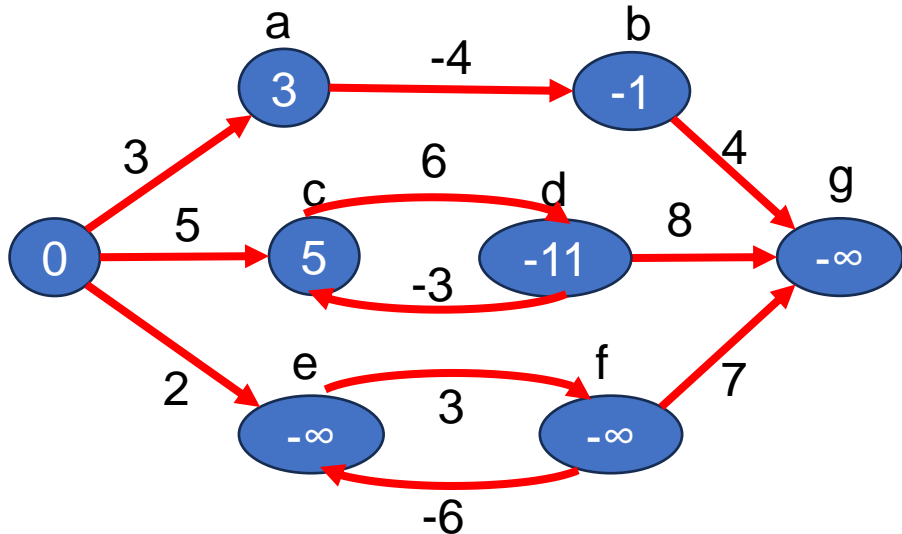
Пусть $p = \langle v_0, v_1, \dots, v_k \rangle$ - кратчайший путь из вершины v_0 в вершину v_k в заданном взвешенном ориентированном графе $G = (V, E)$ с весовой функцией $w : E \rightarrow \mathbb{R}$ и пусть для любых i и j , таких, что $0 \leq i \leq j \leq k$, путь $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ является подпутем p из вершины v_i в вершину v_j . Тогда p_{ij} является кратчайшим путем из v_i в v_j .

Доказательство. Если разложить путь p на составные части $v_0 \rightsquigarrow v_i \rightsquigarrow v_j \rightsquigarrow v_k$, то будет выполняться соотношение $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Теперь предположим, что существует путь p'_{ij} из вершины v_i в вершину v_j с весом $w(p'_{ij}) < w(p_{ij})$. Тогда $v_0 \rightsquigarrow v_i \rightsquigarrow v_j \rightsquigarrow v_k$ представляет собой путь из v_0 в v_k , вес которого $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ меньше, чем $w(p)$, что противоречит предположению о том, что p является кратчайшим путем из вершины v_0 в вершину v_k .

Кратчайшие пути из одной вершины

Ребра с отрицательным весом

В некоторых экземплярах задачи о кратчайшем пути из фиксированного истока веса ребер могут принимать отрицательные значения. Если граф $G = (V, E)$ не содержит циклов с отрицательным весом, достижимых из истока s , то вес кратчайшего пути $\delta(s, v)$ остается вполне определенной величиной для каждой вершины $v \in V$, даже если он принимает отрицательное значение. Если же такой цикл достижим из истока s , веса кратчайших путей перестают быть вполне определенными величинами. В этой ситуации ни один путь из истока s в любую из вершин цикла не может быть кратчайшим, потому что всегда можно найти путь с меньшим весом, который проходит по предложенному "кратчайшему" пути, а затем обходит цикл с отрицательным весом. Если на некотором пути из вершины s к вершине v встречается цикл с отрицательным весом, мы определяем $\delta(s, v) = -\infty$.



Ребра с отрицательными весами в ориентированном графе. Для каждой вершины указан вес кратчайшего пути до нее из вершины s . Поскольку вершины e и f образуют цикл с отрицательным весом, достижимый из s , веса соответствующих кратчайших путей равны $-\infty$. Вершина g достижима из вершины, вес кратчайшего пути к которой равен $-\infty$, поэтому она также имеет вес кратчайшего пути, равный $-\infty$.

Кратчайшие пути из одной вершины

Циклы

Может ли кратчайший путь содержать цикл? Только что мы убедились в том, что он не может содержать цикл с отрицательным весом. В него также не может входить цикл с положительным весом, поскольку в результате удаления этого цикла из пути получится путь, который исходит из того же истока и заканчивается в той же вершине, но обладает меньшим весом. То есть, если $p = \langle v_0, v_1, \dots, v_k \rangle$ является путем, а $c = \langle v_i, v_{i+1}, \dots, v_j \rangle$ - цикл с положительным весом на этом пути (так что $v_i = v_j$ и $w(c) > 0$), то путь $p' = \langle v_0, v_1, \dots, v_i, v_j, v_{j+1}, \dots, v_k \rangle$ имеет вес $w(p') = w(p) - w(c) < w(p)$, а значит, p не может быть кратчайшим путем из v_0 в v_k .

Остаются только циклы с нулевым весом. Однако из пути можно удалить цикл с нулевым весом, в результате чего получится другой путь с тем же весом. Таким образом, если существует кратчайший путь из истока s в целевую вершину v , содержащий цикл с нулевым весом, то существует и другой кратчайший путь из истока s в целевую вершину v , в котором этот цикл не содержится. Если кратчайший путь содержит циклы с нулевым весом, эти циклы можно поочередно удалять до тех пор, пока не получится кратчайший путь, в котором циклы отсутствуют. Поэтому без потери общности можно предположить, что когда мы находим кратчайшие пути, они не содержат циклов. Поскольку в любой ациклический путь в графе $G = (V, E)$ входит не более $|V|$ различных вершин, в нем содержится не более $|V| - 1$ ребер. Таким образом, можно ограничиться рассмотрением кратчайших путей, состоящих не более чем из $|V| - 1$ ребер.

Кратчайшие пути из одной вершины

Представление кратчайших путей

Часто требуется вычислить не только вес каждого из кратчайших путей, но и входящие в их состав вершины. В заданном графе $G = (V, E)$ для каждой вершины $v \in V$ поддерживается атрибут предшественник (predecessor) $v.\pi$, в роли которого выступает либо другая вершина, либо значение NIL. В рассмотренных алгоритмах поиска кратчайших путей атрибуты π присваиваются таким образом, что цепочка предшественников, которая начинается в вершине v , позволяет проследить путь, обратный кратчайшему пути из вершины s в вершину v . Таким образом, для заданной вершины v , у которой $v.\pi \neq \text{NIL}$, с помощью процедуры $\text{Print-Path}(G, s, v)$ можно вывести кратчайший путь из вершины s в вершину v .

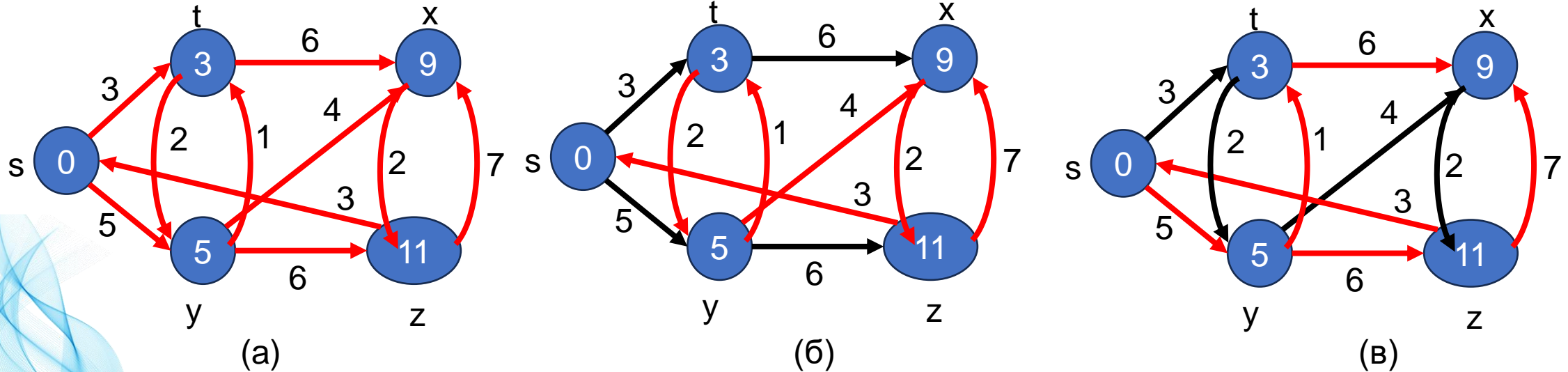
Однако до тех пор, пока алгоритм поиска кратчайших путей не закончил свою работу, значения π не обязательно указывают кратчайшие пути. Как и при поиске в ширину, нас будет интересовать подграф предшествования (predecessor subgraph) $G_\pi = (V_\pi, E_\pi)$, порожденный значениями π . Как и ранее, определим множество вершин V_π как множество, состоящее из тех вершин графа G , предшественниками которых не являются значения NIL, а также включающее исток s :

$$V_\pi = \{v \in V : v.\pi \neq \text{nil}\} \cup \{s\}.$$

Множество ориентированных ребер E_π - это множество ребер, порожденных значениями π у вершин из множества V_π :

$$E_\pi = \{(v, \pi, v) \in E : v \in V_\pi - \{s\}\}.$$

Кратчайшие пути из одной вершины



(а) Взвешенный ориентированный граф с весами кратчайших путей из истока s. (б) Заштрихованные ребра образуют дерево кратчайших путей с корнем в истоке s. (в) Еще одно дерево кратчайших путей с тем же корнем.

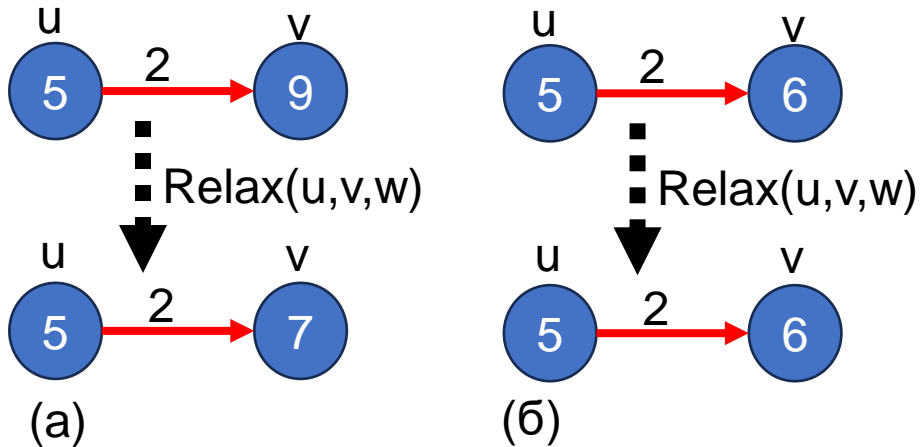
Дерево кратчайших путей (shortest-paths tree) с корнем в вершине s представляет собой ориентированный подграф $G' = (V', E')$, в котором множества $V' \subseteq V$ и $E' \subseteq E$ определяются следующими условиями.

1. Множество V' представляет собой множество вершин, достижимых из истока s графа G.
2. Граф G' образует корневое дерево с корнем s.
3. Для всех $v \in V'$ однозначно определяемый простой путь из вершины s вершину v в графе G' является кратчайшим путем из s в v в G

Кратчайшие пути из одной вершины

Ослабление

В алгоритмах используется метод **релаксации**, или ослабления (relaxation). Для каждой вершины $v \in V$ поддерживается атрибут $v.d$, представляющий собой верхнюю границу веса, которым обладает кратчайший путь из истока s в вершину v . Мы называем атрибут $v.d$ **оценкой кратчайшего пути** (shortest-path estimate). Инициализация оценок кратчайших путей и предшественников проводится в приведенной ниже процедуре, время работы которой равно $\Theta(V)$.



Ослабление ребра (u, v) с весом $w(u, v) = 2$. Для каждой вершины приведена оценка ее кратчайшего пути. (a) Поскольку перед ослаблением $v.d > u.d + w(u, v)$, значение $v.d$ уменьшается. (б) Здесь перед ослаблением ребра $v.d \leq u.d + w(u, v)$, так что ослабление оставляет значение $v.d$ неизменным.

```

Initialize-Single-Source( $G, s$ )
1 for каждой вершины  $v \in G.V$ 
2      $v.d = \infty$ 
3      $V.\pi = \text{NIL}$ 
4  $s.d = 0$ 
  
```

```

Relax( $u, v, w$ )
1 if  $v.d > u.d + w(u, v)$ 
2      $v.d = u.d + w(u, v)$ 
3      $v.\pi = u$ 
  
```

Кратчайшие пути из одной вершины

Свойства кратчайших путей и ослабление

Неравенство треугольника

Для каждого ребра $(u,v) \in E$ выполняется неравенство $\delta(s,v) \leq \delta(s,u) + w(u, v)$.

Свойство верхней границы

Для всех вершин $v \in V$ всегда выполняется неравенство $v.d > \delta(s, v)$, а после того как величина $v.d$ достигает значения $\delta(s, v)$, она больше не изменяется.

Свойство отсутствия пути

Если из вершины s в вершину v нет пути, то всегда выполняется соотношение $v.d = \delta(s, v) = \infty$.

Свойство сходимости

Если $s \rightsquigarrow u \rightarrow v$ является кратчайшим путем в G для некоторых $u,v \in V$ и если $u.d = \delta(s,u)$ в любой момент до ослабления ребра (u,v) , то $v.d = \delta(s, v)$ в любой момент после этого.

Свойство ослабления пути

Если $p = \langle v_0, v_1, \dots, v_k \rangle$ является кратчайшим путем из $s = v_0$ в v_k и если мы ослабляем ребра p в порядке $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$, то $v_k.d = \delta(s, v_k)$.

Это свойство выполняется независимо от других этапов ослабления, даже если они чередуются с ослаблением ребер, принадлежащих пути p .

Свойство подграфа предшествования

Если для всех вершин $v \in V$ выполняется $v.d = \delta(s, v)$, то подграф предшествования представляет собой дерево кратчайших путей с корнем в истоке s .

Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда (Bellman-Ford algorithm) решает задачу о кратчайшем пути из одной вершины в общем случае, когда вес каждого из ребер может быть отрицательным. Для заданного взвешенного ориентированного графа $G = (V, E)$ с истоком s и весовой функцией $w : E \rightarrow \mathbb{R}$ алгоритм Беллмана Форда возвращает логическое значение, указывающее, содержится ли в графе цикл с отрицательным весом, достижимый из истока. Если такой цикл существует, алгоритм указывает, что решения не существует. Если же таких циклов нет, алгоритм выдает кратчайшие пути и их веса.

В этом алгоритме используется ослабление, в результате которого величина $v.d$, представляющая собой оценку веса кратчайшего пути из истока s к каждой из вершин $v \in V$, постепенно уменьшается до тех пор, пока не станет равной фактическому весу кратчайшего пути $\delta(s, v)$. Значение TRUE возвращается алгоритмом тогда и только тогда, когда граф не содержит циклов с отрицательным весом, достижимых из истока.

```
Bellman-Ford( $G, w, s$ )
```

```
1 Initialize-Single-Source( $G, s$ )
```

```
2 for  $i = 1$  to  $|G.V| - 1$ 
```

```
3     for каждого ребра  $(u, v) \in G.E$ 
```

```
4         Relax( $u, v, w$ )
```

```
5 for каждого ребра  $(u, v) \in G.E$ 
```

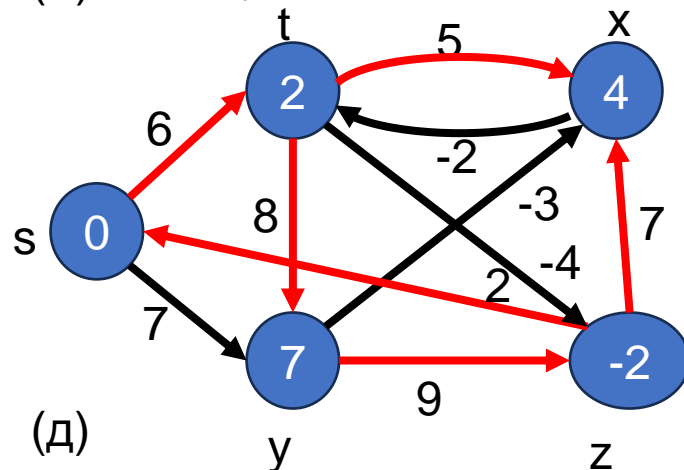
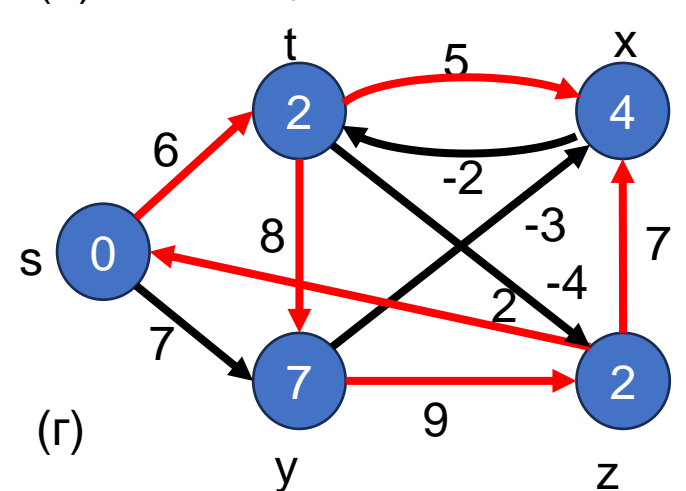
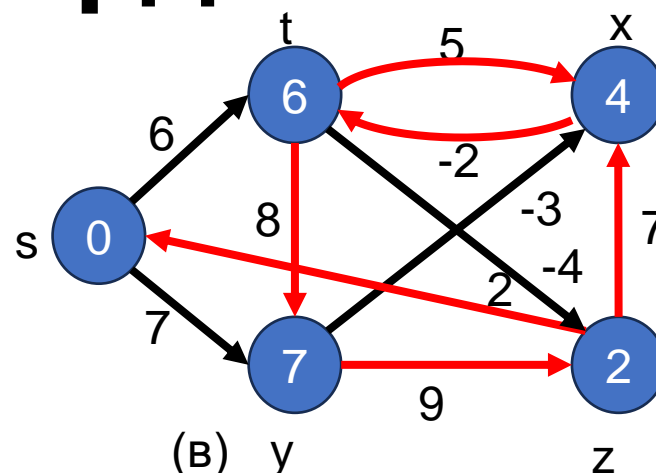
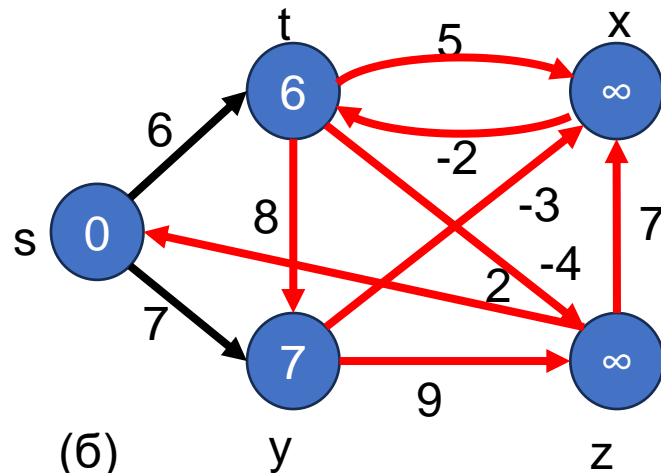
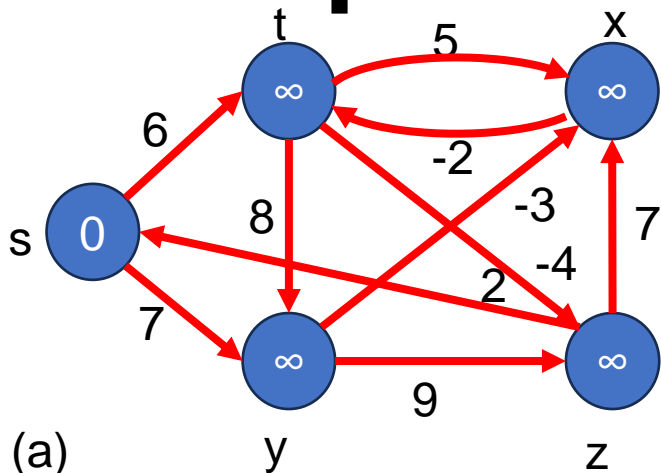
```
6     if  $v.d > u.d + w(u, v)$ 
```

```
7         return FALSE
```

```
8 return TRUE
```

Время работы алгоритма Беллмана Форда составляет $O(VE)$, поскольку инициализация в строке 1 занимает время $\Theta(V)$, на каждый из $|V| - 1$ проходов по ребрам в строках 2-4 требуется время $\Theta(E)$, а на выполнение цикла for в строках 5-7 затрачивается время $O(E)$.

Алгоритм Беллмана-Форда



Выполнение алгоритма Беллмана Форда. Истоком является вершина s . В вершинах показаны значения d , а заштрихованные ребра указывают предшественников: если ребро (u, v) заштриховано, то $v.\pi = u$. В этом конкретном примере каждый проход ослабляет ребра в порядке (t, x) , (t, y) , (t, z) , (x, t) , (y, x) , (y, z) , (z, x) , (z, s) , (s, t) , (s, y) . Алгоритм Беллмана Форда в данном примере возвращает значение TRUE.

Кратчайшие пути из одной вершины в ориентированных ациклических графах

Ослабляя ребра взвешенного ориентированного ациклического графа $G = (V, E)$ в порядке, определенном топологической сортировкой его вершин, кратчайшие пути из одной вершины можно найти за время $\Theta(V + E)$. В ориентированном ациклическом графе кратчайшие пути всегда вполне определены, поскольку, даже если вес некоторых ребер отрицателен, циклов с отрицательными весами не существует.

Работа алгоритма начинается с топологической сортировки ориентированного ациклического графа, которая должна установить линейное упорядочение вершин. Если путь из вершины u к вершине v существует, то в топологической сортировке вершина u предшествует вершине v . По вершинам, расположенным в топологическом порядке, проход выполняется только один раз. При обработке каждой вершины производится ослабление всех ребер, исходящих из этой вершины.

Dag-Shortest-Paths(G, w, s)

1 Топологическая сортировка вершин графа G

2 Initialize-Single-Source(G, s)

3 for каждой вершины u в порядке топологической сортировки

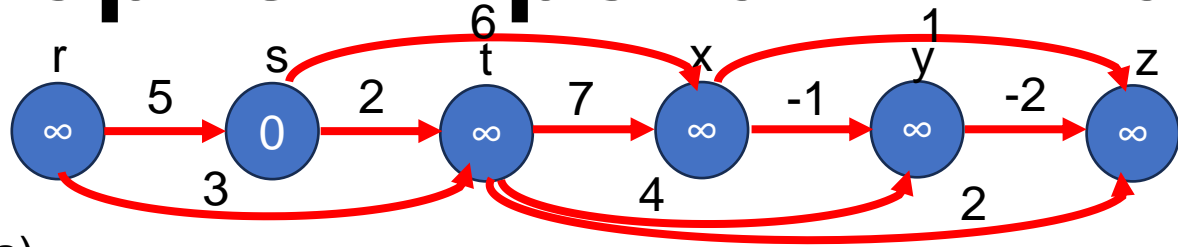
4 for каждой вершины $v \in G$. Adj[u]

5 Relax(u, v, w)

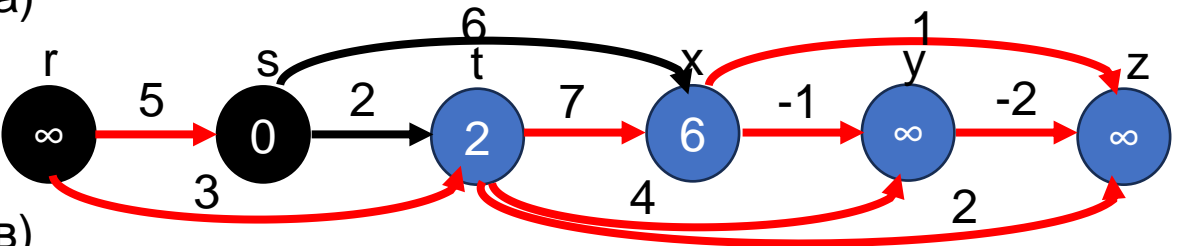
Топологическая сортировка в строке 1 выполняется за время $\Theta(V + E)$. Вызов процедуры Initialize-Single-Source в строке 2 занимает время $\Theta(V)$. На каждую вершину приходится по одной итерации цикла for в строках 3-5. Цикл for в строках 4 и 5 ослабляет каждое ребро по одному разу.

полное время работы алгоритма равно $\Theta(V + E)$.

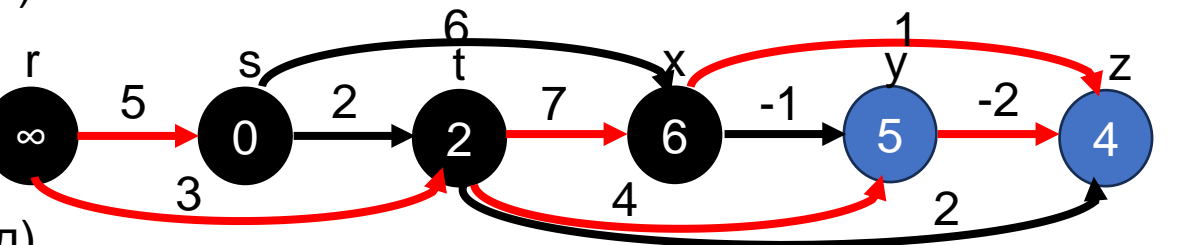
Кратчайшие пути из одной вершины в ориентированных ациклических графах



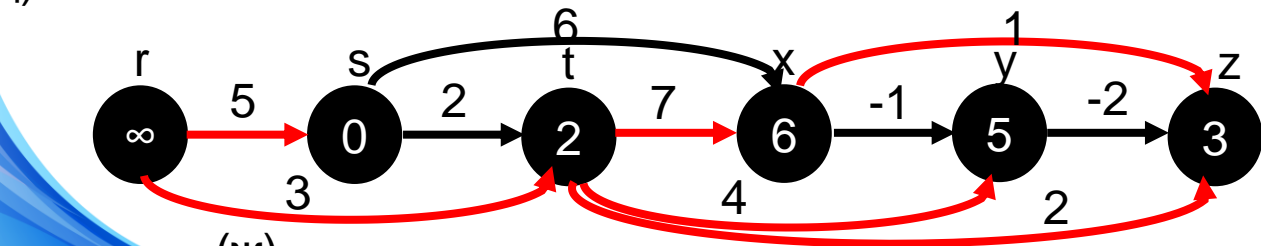
(a)



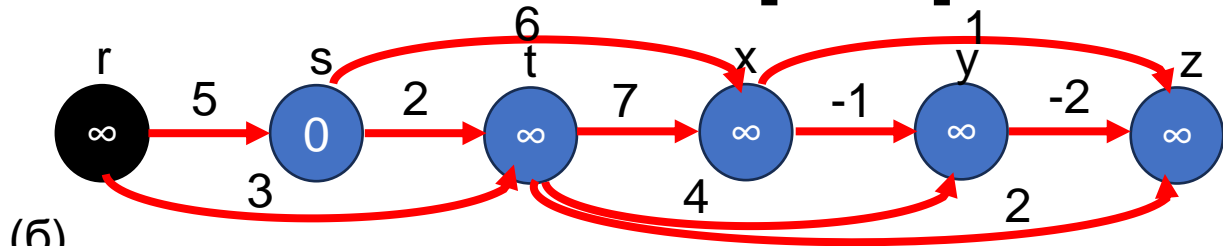
(b)



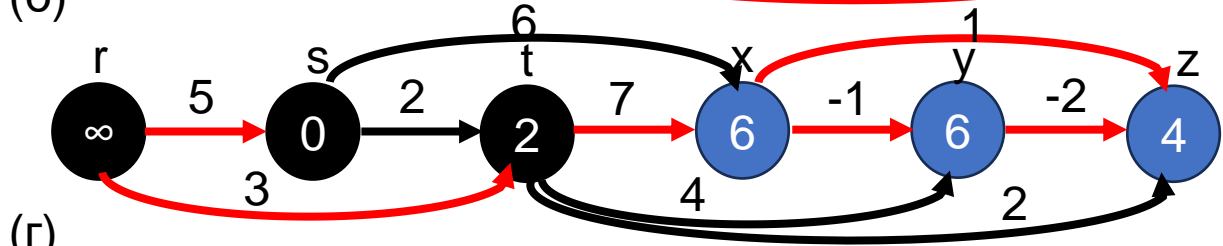
(d)



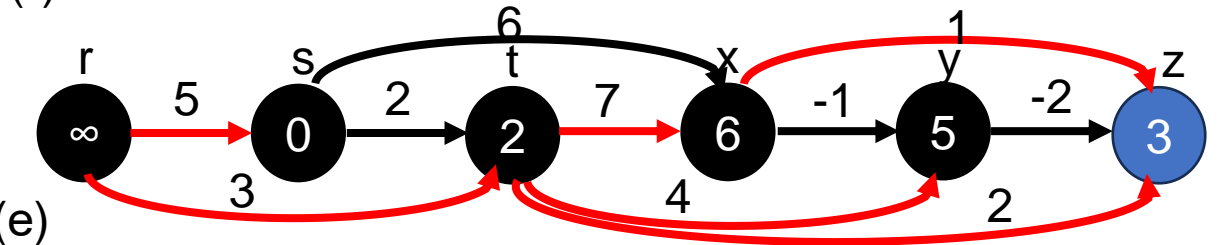
(ж)



(б)



(г)



(e)

Работа алгоритма поиска кратчайших путей в ориентированном ациклическом графе.

Алгоритм Дейкстры

Алгоритм Дейкстры, предложенный голландским ученым Эдсгером Дейкстрой в 1959 году, используется для поиска дерева кратчайших путей. Этот алгоритм широко используется в сетевых протоколах маршрутизации, в частности IS-IS и OSPF (Open Shortest Path First).

Для заданного графа G и исходного узла A алгоритм используется для поиска кратчайшего пути (имеющего наименьшую стоимость) между A (исходным узлом) и каждым другим узлом. Более того, алгоритм Дейкстры также используется для поиска стоимости кратчайших путей от исходного узла до конечного узла. Например, если мы нарисуем граф, в котором узлы представляют города, а взвешенные ребра представляют расстояния между парами городов, соединенных прямой дорогой, то алгоритм Дейкстры при применении дает кратчайший маршрут между одним городом и всеми другими городами.



Алгоритм Дейкстры

Алгоритм Дейкстры используется для поиска длины оптимального пути между двумя узлами в графе. Термин «оптимальный» может означать что угодно: кратчайший, самый дешевый или самый быстрый. Если мы начнем алгоритм с начального узла, то расстояние до узла Y можно задать как расстояние от начального узла до этого узла.

1. Выберите исходный узел, также называемый начальным узлом
2. Определите пустое множество N , которое будет использоваться для хранения узлов, к которым был найден кратчайший путь.
3. Пометьте начальный узел как и вставьте его в N .
4. Повторите шаги 5–7, пока целевой узел не окажется в N или в N больше не останется помеченных узлов.
5. Рассмотрим каждый узел, который не находится в N и соединен ребром с вновь вставленным узлом.
6.
 - (a) Если узел, который не находится в N , не имеет метки, то **ЗАДАЙТЕ** метку узла = метке вновь вставленного узла + длине ребра.
 - (b) Иначе, если узел, который не находится в N , уже был помечен, то **SET** его новую метку = минимум (метка новой вставленной вершины + длина ребра, старая метка)
7. Выберите аном не в N , которому назначена наименьшая метка, и добавьте ее к N .



Алгоритм Дейкстры

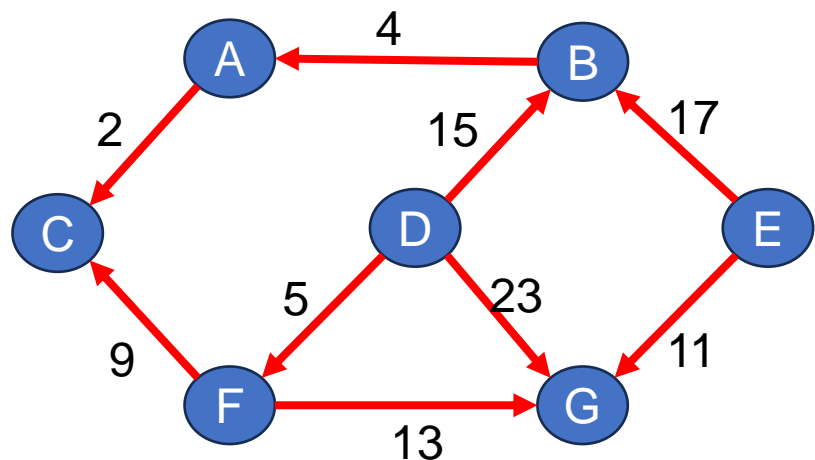
Алгоритм Дейкстры помечает каждый узел в графе, где метки представляют расстояние (стоимость) от исходного узла до этого узла. Существует два вида меток: временные и постоянные. Временные метки назначаются узлам, которые не были достигнуты, в то время как постоянные метки даются узлам, которые были достигнуты, и их расстояние (стоимость) до исходного узла известно. Узел должен быть постоянной меткой или временной меткой, но не обеими одновременно.

Выполнение этого алгоритма даст один из следующих двух результатов:

1. Если целевой узел помечен, то метка, в свою очередь, будет представлять расстояние от исходного узла до конечного узла.
2. Если узел назначения не помечен, то нет пути от источника до узла назначения.



Алгоритм Дейкстры - пример



Взяв D в качестве начального узла, выполните на нем алгоритм Дейкстры.

Шаг 1: Установите метку $D = 0$ и $N = \{D\}$.

Шаг 2: Метка $D = 0$, $B = 15$, $G = 23$ и $F = 5$. Следовательно, $N = \{D, F\}$.

Шаг 3: Метка $D = 0$, $B = 15$, G была перемаркирована на 18, поскольку минимум $(5 + 13, 23) = 18$, C была перемаркирована на 14 $(5 + 9)$. Следовательно, $N = \{D, F, C\}$.

Шаг 4: Метка $D = 0$, $B = 15$, $G = 18$. Следовательно, $N = \{D, F, C, B\}$.

Шаг 5: Метка $D = 0$, $B = 15$, $G = 18$ и $A = 19$ $(15 + 4)$. Следовательно, $N = \{D, F, C, B, G\}$.

Шаг 6: Метка $D = 0$ и $A = 19$. Следовательно, $N = \{D, F, C, B, G, A\}$

Обратите внимание, что у нас нет меток для узла E; это означает, что E недостижим из D. Только узлы, находящиеся в N, достижимы из B. Время выполнения алгоритма Дейкстры можно задать как $O(|V|^2 + |E|) = O(|V|^2)$, где V — множество вершин, а E — в графе.

Кратчайшие пути между всеми парами

Задача о поиске кратчайших путей между всеми парами вершин графа. Эта задача может возникнуть, например, при составлении таблицы расстояний между всеми парами городов, нанесенных на атлас дорог. В этой задаче задается взвешенный ориентированный граф $G = (V, E)$ с весовой функцией $w : E \rightarrow R$, отображающей ребра на их веса, выраженные действительными числами. Для каждой пары вершин $u, v \in V$ требуется найти кратчайший (обладающий наименьшим весом) путь из вершины u в вершину v , вес которого определяется как сумма весов входящих в него ребер. Обычно выходные данные представляются в табличной форме: на пересечении строки с индексом u и столбца с индексом v расположен вес кратчайшего пути из вершины u в вершину v .

Задачу о поиске кратчайших путей между всеми парами вершин можно решить, выполнив $|V|$ раз алгоритм поиска кратчайших путей из одной вершины, каждый раз выбирая в качестве истока новую вершину графа. Если веса всех ребер неотрицательные, можно воспользоваться алгоритмом Дейкстры. Если используется реализация неубывающей очереди с приоритетами в виде линейного массива, то время работы такого алгоритма равно $O(V^3 + VE) = O(V^3)$. Если же неубывающая очередь с приоритетами реализована в виде бинарной неубывающей пирамиды, то время работы будет равно $O(VE \lg V)$, что предпочтительнее для разреженных графов. Можно также реализовать неубывающую очередь с приоритетами с помощью пирамиды Фибоначчи; в этом случае время работы алгоритма равно $O(V^2 \lg V + VE)$.

Если в графе могут быть ребра с отрицательным весом, алгоритм Дейкстры неприменим. Вместо него для каждой вершины следует выполнить более медленный алгоритм Беллмана-Форда. Полученное в результате время работы равно $O(V^2E)$, что для плотных графов можно записать как $O(V^4)$.

Алгоритм Уоршалла

Если граф G задан как $G = (V, E)$, где V — множество вершин, а E — множество ребер, матрицу пути G можно найти как $P = A + A^2 + A^3 + \dots + A^n$. Это длительный процесс, поэтому Уоршалл дал очень эффективный алгоритм для вычисления матрицы пути. Алгоритм Уоршалла определяет матрицы $P_0, P_1, P_2, \dots, P_n$.

Запись матрицы пути

$P_k[i][j]$

1

[если есть путь от v_i до v_j Путь не должен использовать никаких других узлов, кроме v_1, v_2, \dots, v_k]

0

[иначе]

Это означает, что если $P_0[i][j] = 1$, то существует ребро от узла v_i до v_j .

Если $P_1[i][j] = 1$, то существует ребро из v_i в v_j , которое не использует никакую другую вершину, кроме v_1 .

Если $P_2[i][j] = 1$, то существует ребро из v_i в v_j , которое не использует никакую другую вершину, кроме v_1 и v_2 .

Обратите внимание, что P_0 равно матрице смежности G . Если G содержит n узлов, то $P_n = P$, что является матрицей пути графа G .

Из вышеизложенного обсуждения можно сделать вывод, что $P_k[i][j]$ равно 1 только тогда, когда происходит один из двух следующих случаев:

- Существует путь из v_i в v_j , который не использует никакой другой узел, кроме v_1, v_2, \dots, v_{k-1} . Следовательно, $P_{k-1}[i][j] = 1$.
- Существует путь из v_i в v_k и путь из v_k в v_j , где все узлы используют v_1, v_2, \dots, v_{k-1} .

Следовательно, $P_{k-1}[i][k] = 1$ и $P_{k-1}[k][j] = 1$

Следовательно, матрицу пути P_n можно рассчитать по формуле, заданной как:

$P_k[i][j] = P_{k-1}[i][j] \vee (P_{k-1}[i][k] \wedge P_{k-1}[k][j])$, где \vee обозначает логическую операцию ИЛИ, а \wedge обозначает логическую операцию И.

Алгоритм Уоршалла

Алгоритм Уоршалла для поиска матрицы пути P с использованием матрицы смежности A .

Шаг 1: [ИНИЦИАЛИЗИРУЙТЕ матрицу пути]

Повторите шаг 2 для $I = 0$ до $n-1$, где n — количество узлов в графе

Шаг 2: Повторите шаг 3 для $J = 0$ до $n-1$

Шаг 3: ЕСЛИ $A[I][J] = 0$, то УСТАНОВИТЕ $P[I][J] = 0$

ИНАЧЕ $P[I][J] = 1$

[КОНЕЦ ЦИКЛА]

[КОНЕЦ ЦИКЛА]

Шаг 4: [Вычислите матрицу пути P]

Повторите шаг 5 для $K = 0$ до $n-1$

Шаг 5: Повторите шаг 6 для $I = 0$ до $n-1$

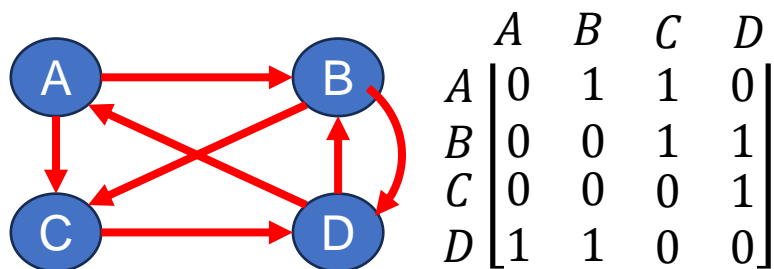
Шаг 6: Повторите шаг 7 для $J = 0$ до $n-1$

Шаг 7: Установите $P_k[I][J] = P_{k-1}[I][J] \vee (P_{k-1}[I][K] \wedge P_{k-1}[K][J])$

Шаг 8: ВЫХОД



Алгоритм Уоршалла - пример



Граф G и его матрица пути P

Мы можем сразу вычислить матрицу пути P, используя алгоритм Уоршелла.

Матрица пути P может быть задана за один шаг как:

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Таким образом, мы видим, что вычисление $A, A^2, A^3, A^4, \dots, A^5$ для вычисления P является очень медленным и неэффективным методом по сравнению с методом Уоршелла.

Напишите программу для реализации алгоритма Уоршелла для нахождения матрицы пути.



Модифицированный алгоритм Уоршелла

Алгоритм Уоршелла можно модифицировать для получения матрицы, которая дает кратчайшие пути между узлами в графе G . В качестве входных данных для алгоритма мы берем матрицу смежности A графа G и заменяем все значения A , которые равны нулю, на бесконечность (∞). Бесконечность (∞) обозначает очень большое число и указывает на то, что между вершинами нет пути. В модифицированном алгоритме Уоршелла мы получаем набор матриц $Q_0, Q_1, Q_2, \dots, Q_m$, используя формулу, приведенную ниже.

$$Q_k[i][j] = \text{Minimum}(M_{k-1}[i][j], M_{k-1}[i][k] + M_{k-1}[k][j])$$

Q_0 точно такой же, как A , с небольшой разницей, что каждый элемент, имеющий нулевое значение в A , заменяется на (∞) в Q_0 . Используя данную формулу, матрица Q_n даст матрицу пути, которая имеет кратчайший путь между вершинами графа.



Модифицированный алгоритм Уоршелла

Шаг 1: [Инициализация матрицы кратчайшего пути, Q]

Повторите шаг 2 для I = от 0 до n-1, где n — количество узлов в графе

Шаг 2: Повторите шаг 3 для J = от 0 до n-1

Шаг 3: ЕСЛИ $A[I][J] = 0$, то SET $Q[I][J] = \text{Бесконечность}$ (или 9999)

ИНАЧЕ $Q[I][J] = A[I][J]$

[КОНЕЦ ЦИКЛА]

[КОНЕЦ ЦИКЛА]

Шаг 4: [Вычислите матрицу кратчайшего пути Q]

Повторите шаг 5 для K = от 0 до n-1

Шаг 5: Повторите шаг 6 для I = от 0 до n-1

Шаг 6: Повторите шаг 7 для J = от 0 до n-1

Шаг 7: ЕСЛИ $Q[I][J] \leq Q[I][K] + Q[K][J]$ SET $Q[I][J] = Q[I][J]$

ИНАЧЕ SET $Q[I][J] = Q[I][K] + Q[K][J]$ [КОНЕЦ IF]

[КОНЕЦ ЦИКЛА]

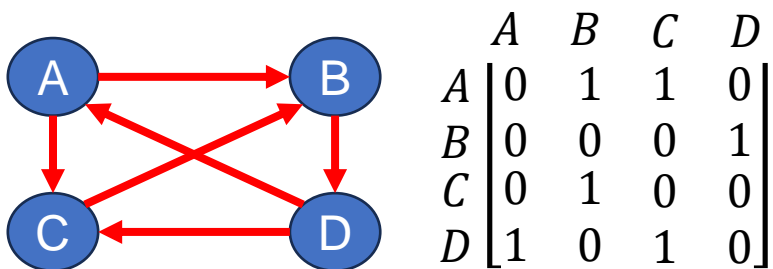
[КОНЕЦ ЦИКЛА]

[КОНЕЦ ЦИКЛА]

Шаг 8: ВЫХОД



Модифицированный алгоритм Уоршелла



$$Q_0 = \begin{bmatrix} 9999 & 1 & 1 & 9999 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 1 & 9999 & 9999 \\ 1 & 9999 & 1 & 9999 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 9999 & 1 & 1 & 9999 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 1 & 9999 & 9999 \\ 1 & 2 & 1 & 9999 \end{bmatrix}$$

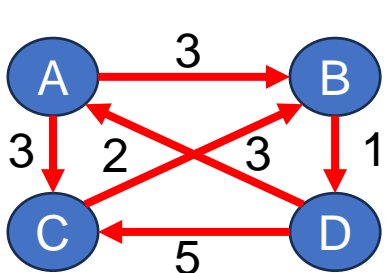
$$Q_2 = \begin{bmatrix} 9999 & 1 & 1 & 2 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 1 & 9999 & 2 \\ 1 & 2 & 1 & 3 \end{bmatrix}$$

$$Q_3 = \begin{bmatrix} 9999 & 1 & 1 & 2 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 1 & 9999 & 2 \\ 1 & 2 & 1 & 3 \end{bmatrix}$$

$$Q_4 = Q \begin{bmatrix} 3 & 1 & 1 & 2 \\ 2 & 3 & 3 & 1 \\ 3 & 1 & 1 & 2 \\ 1 & 2 & 1 & 3 \end{bmatrix}$$



Модифицированный алгоритм Уоршелла



	A	B	C	D
A	0	3	3	0
B	0	0	0	1
C	0	3	0	0
D	2	0	5	0

$$Q_0 = \begin{bmatrix} 9999 & 3 & 3 & 9999 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 3 & 9999 & 9999 \\ 2 & 9999 & 5 & 9999 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 9999 & 3 & 3 & 9999 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 3 & 9999 & 9999 \\ 2 & 5 & 5 & 9999 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 9999 & 3 & 3 & 4 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 1 & 9999 & 4 \\ 2 & 5 & 5 & 6 \end{bmatrix}$$

$$Q_3 = \begin{bmatrix} 9999 & 3 & 3 & 4 \\ 9999 & 9999 & 9999 & 1 \\ 9999 & 3 & 9999 & 6 \\ 2 & 5 & 5 & 6 \end{bmatrix}$$

$$Q_4 = Q = \begin{bmatrix} 6 & 3 & 3 & 4 \\ 3 & 6 & 6 & 1 \\ 6 & 3 & 9 & 4 \\ 2 & 5 & 5 & 6 \end{bmatrix}$$



Алгоритм Джонсона для разреженных графов – домашнее задание

Алгоритм Джонсона позволяет найти кратчайшие пути между всеми парами вершин за время $O(V^2 \lg V + VE)$. Для разреженных графов в асимптотическом пределе он ведет себя лучше, чем алгоритм многократного возведения матриц квадрат и алгоритм Флойда-Уоршелла. Этот алгоритм либо возвращает матрицу, содержащую веса кратчайших путей для всех пар вершин, либо выводит: сообщение о том, что входной граф содержит цикл с отрицательным весом.



Применение графов

Графы строятся для различных типов приложений, таких как:

- В сетях схем, где точки соединения рисуются как вершины, а проводники компонентов становятся ребрами графа.
- В транспортных сетях, где станции рисуются как вершины, а маршруты становятся ребрами графа.
- На картах, где города/штаты/регионы рисуются как вершины, а отношения смежности — как ребра.
- В анализе потока программы, где процедуры или модули рассматриваются как вершины, а вызовы этих процедур рисуются как ребра графа.
- Как только у нас есть граф определенной концепции, его можно легко использовать для поиска кратчайших путей, планирования проекта и т. д.
- В блок-схемах или графах потока управления операторы и условия в программе представлены как узлы, а поток управления представлен ребрами.
- В диаграммах перехода состояний узлы используются для представления состояний, а ребра представляют разрешенные переходы из одного состояния в другое.
- Графы также используются для построения диаграмм сетей активности. Эти диаграммы широко используются в качестве инструмента управления проектами для представления взаимозависимых отношений между группами, шагами и задачами, которые оказывают значительное влияние на проект.

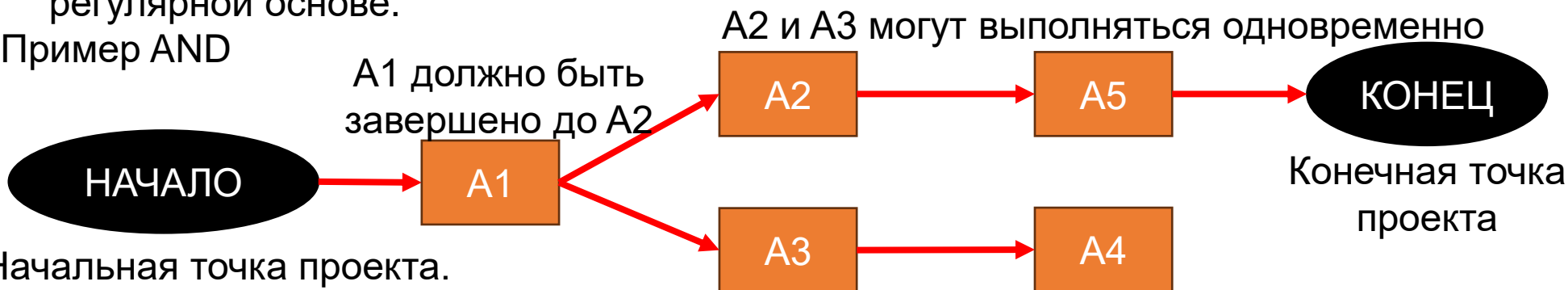


Применение графов

Диаграмма сети активности (AND), также известная как диаграмма стрелок или PERT (метод обзора оценки программы), используется для определения временных последовательностей событий, которые имеют решающее значение для целей. Она также полезна, когда проект имеет несколько видов деятельности, требующих одновременного управления. AND помогают команде разработчиков проекта создать реалистичный график проекта, рисуя графики, которые показывают:

- общее количество времени, необходимое для завершения проекта
- последовательность, в которой должны выполняться виды деятельности
- действия, которые могут выполняться одновременно
- критические виды деятельности, которые должны контролироваться на регулярной основе.

Пример AND



Начальная точка проекта.



Квадраты обозначают активность



Стрелки обозначают зависимость между активностями

А4 необходимо выполнить до А5, но после А3