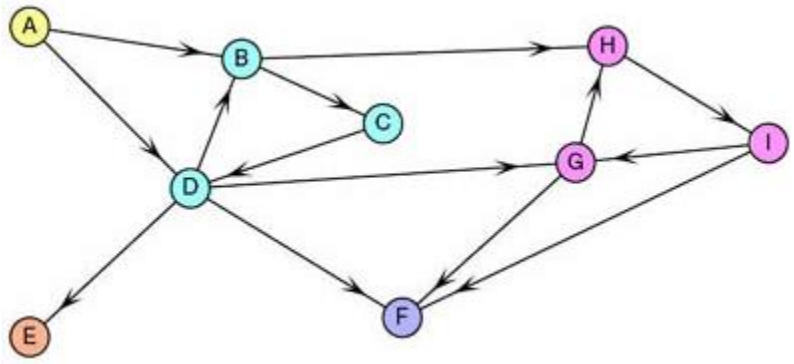
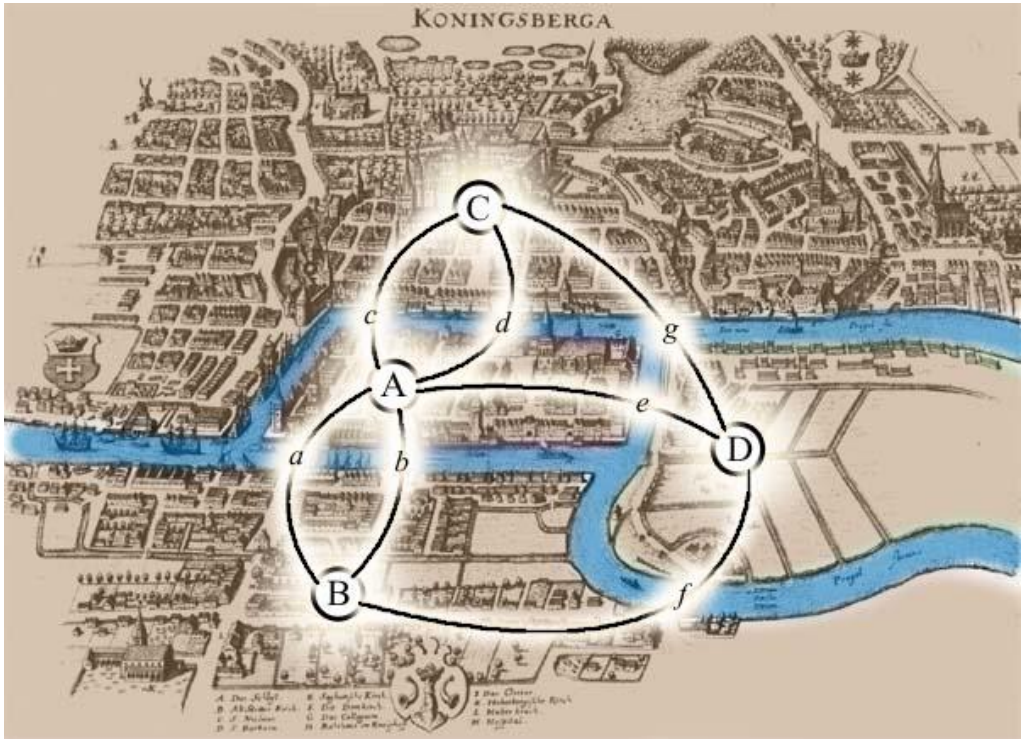
Основы теории графов

Теория графов изучает математические объекты, описывающие связи между элементами конечного множества





ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ГРАФОВ

Графом называется тройка $<M,N,T>$, где

M — непустое конечное множество ***вершин***,

N — конечное множество ***дуг***, соединяющих вершины,

T — отображение из N в $M \times M$, которое сопоставляет каждой дуге упорядоченную пару вершин. Первая из них называется началом этой дуги, а вторая — концом дуги.

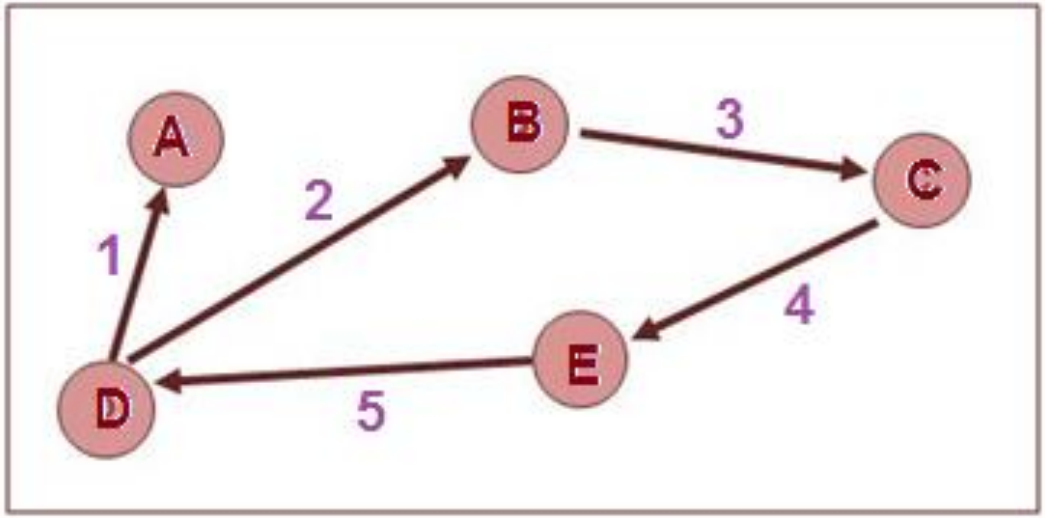
M ={A, B, C, D, E}

N = {1, 2, 3, 4, 5},

T(1) = (D,A); T(2) = (D,B);

T(3) = (B,C); T(4) = (C,E);

T(5) = (E,D);



ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ГРАФОВ

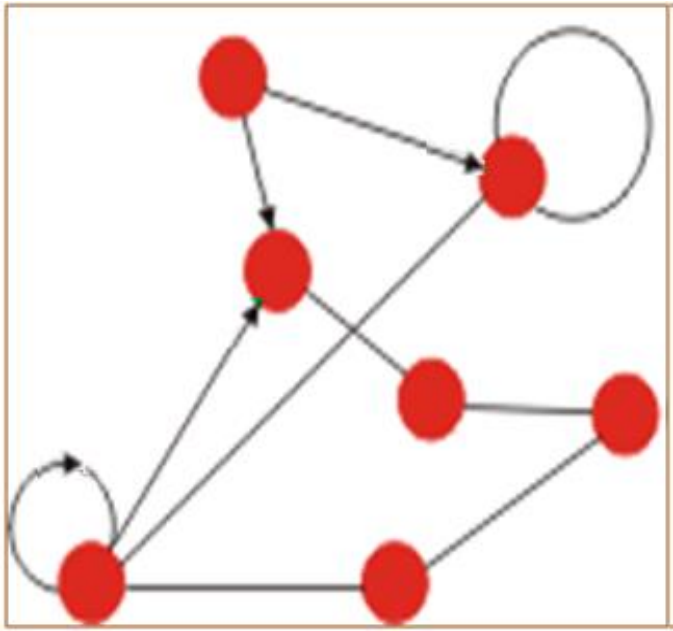
Ребро – неориентированная дуга.

Граф называется неориентированным**,** если каждая его дуга не ориентирована, ориентированным (орграфом), если ориентированы все его дуги, смешанным, если есть как ориентированные, так и неориентированные дуги.

Петля – это дуга, начальная и конечная вершина которой совпадают.

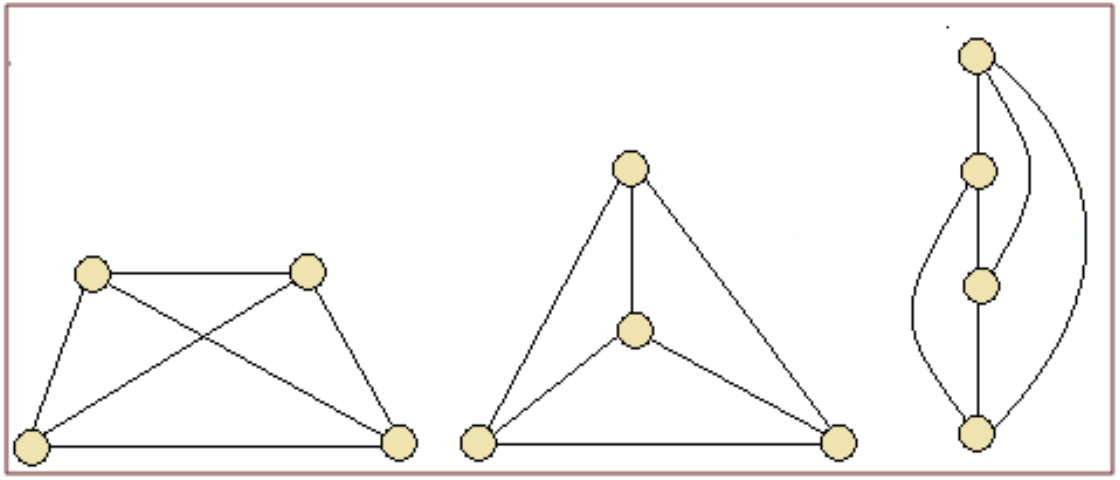
Вершины, прилегающие к одному и тому же ребру, называются смежными.

Полным называется граф, в котором каждые две вершины смежные.



Наглядное представление

Граф можно изобразить графически: сопоставить его вершинам точки, а дугам – линии, идущее от начальной вершины к конечной.



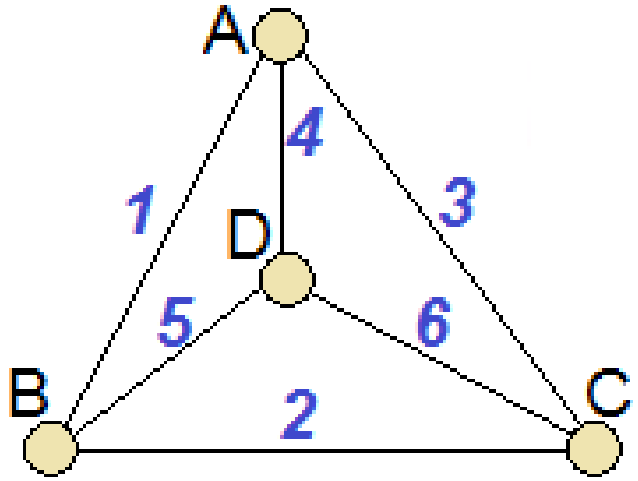
ВАЖНО: Форма линии и расположение вершин на рисунке произвольны.

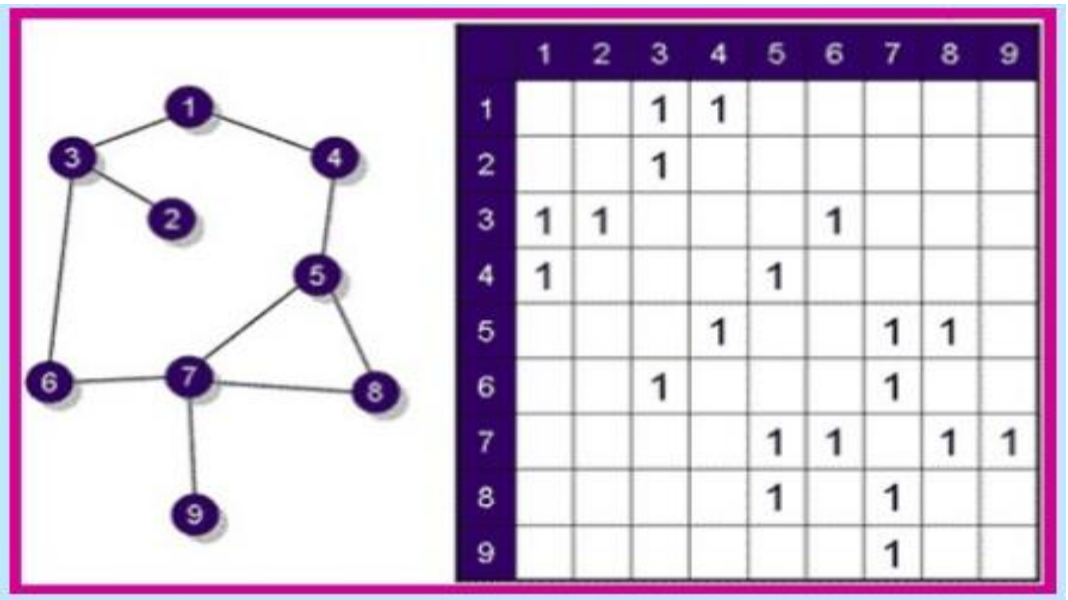
Одному графу можно сопоставить несколько графических представлений.

Изображение призвано лишь показать, какие пары вершин соединены рёбрами, а какие — нет.

Табличное представление

	A	B	C	D
1	+	+		
2		+	+	
3	+		+	
4	+			+
5		+		+
6			+	+





	1	2	3	4	5	6	7	8	9
1			1	1					
2			1						
3	1	1				1			
4	1				1				
5				1			1	1	
6			1				1		
7					1	1		1	1
8					1		1		
9							1		

	A	B	C	D
A		1	3	4
B	1		2	5
C	3	2		6
D	4	5	6	

СВЯЗНОСТЬ ГРАФА

Маршрутом называется такая конечная или бесконечная последовательность ребер, что каждые два соседних ребра имеют общую вершину.

Маршрут называется **циклическим**, если его конечная вершина совпадает с начальной.

Циклический маршрут называется **циклом**, если каждое его ребро встречается в нем не более одного раза; вершины могут повторяться и несколько раз.

Две вершины A и B называются **связанными**, если существует маршрут с концами A и B.

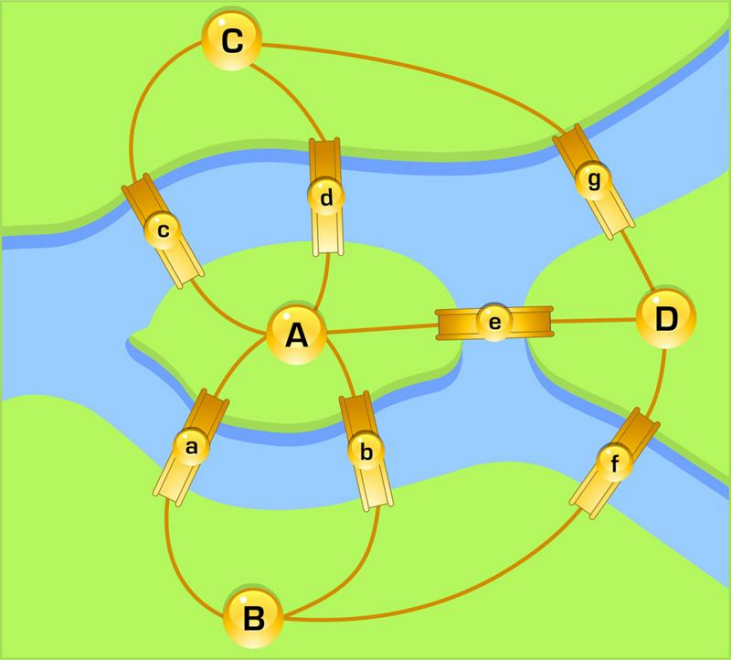
Граф называется **связным**, если любая пара его вершин связана.

Цикл Эйлера

Долгое время отдельные задачи теории графов появлялись как занимательные задачи, которые легко формулировались, но были почему-то очень трудны для решения.

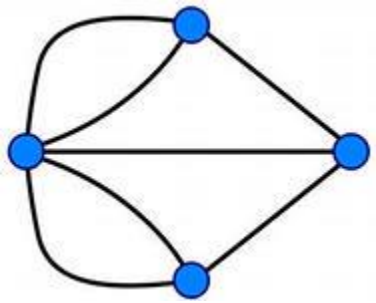
Среди таких задач наибольшей известностью пользуется задача о семи кёнигсбергских мостах: **в городе Кёнигсберге в начале XVIII века было семь мостов. Возник вопрос: возможна ли такая прогулка, в которой путь пройдет по всем мостам, и по каждому мосту ровно один раз.**

Этот вопрос был предложен знаменитому Леонарду Эйлеру, и в 1735 г. он эту задачу решил: нельзя.

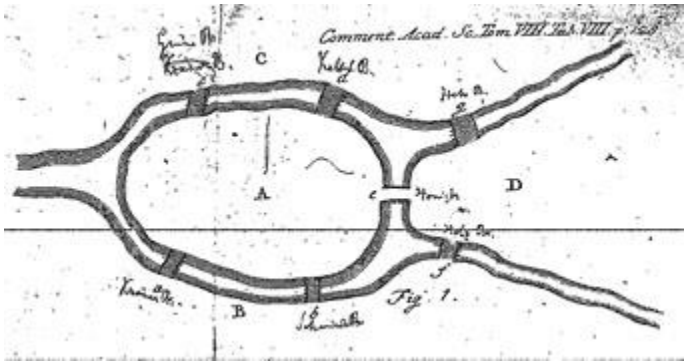


В ходе рассуждений Эйлер пришёл к следующим выводам:

- **Число нечётных вершин** (к которым ведёт нечётное число рёбер) **графа должно быть чётно. Не может существовать граф, который имел бы нечётное число нечётных вершин.**
- **Если все вершины графа чётные, то можно, не отрывая карандаша от бумаги, начертить граф, при этом можно начинать с любой вершины графа и завершить его в той же вершине.**
- **Граф с более чем двумя нечётными вершинами невозможно начертить одним росчерком.**



Граф кёнигсбергских мостов имел четыре нечётные вершины (то есть все), следовательно, невозможно пройти по всем мостам, не проходя ни по одному из них дважды.

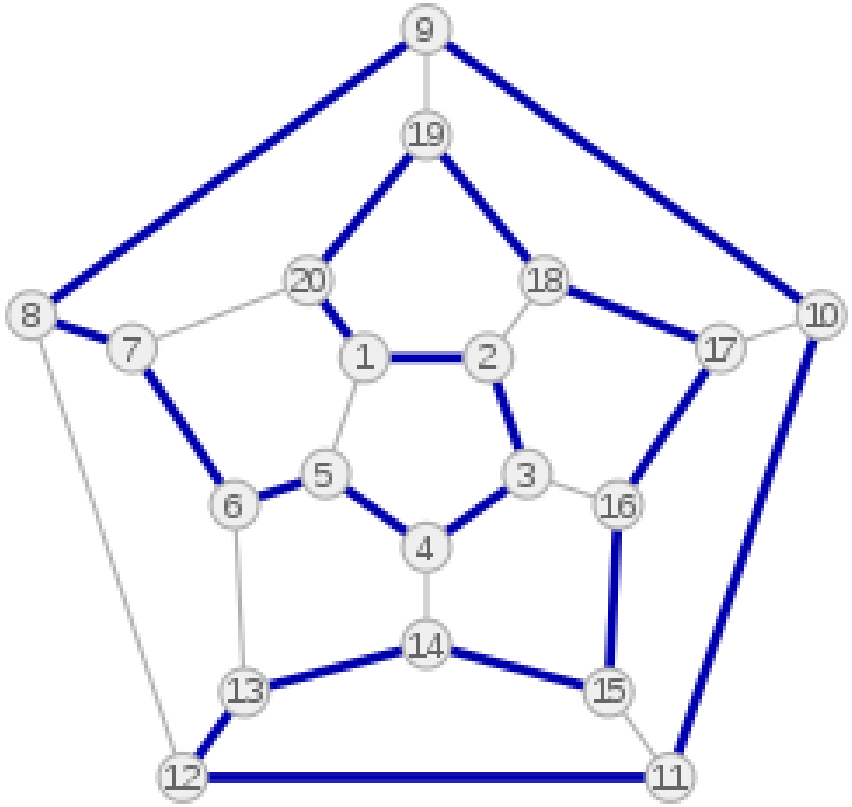


Карта из статьи 28-летнего Эйлера в трудах Санкт-Петербургской Академии наук

Созданная Эйлером теория графов нашла очень широкое применение: например, её используют при изучении транспортных и коммуникационных систем, в частности, для маршрутизации данных в Интернете.

Гамильтонов цикл

Уильям Роуэн Гамильтон



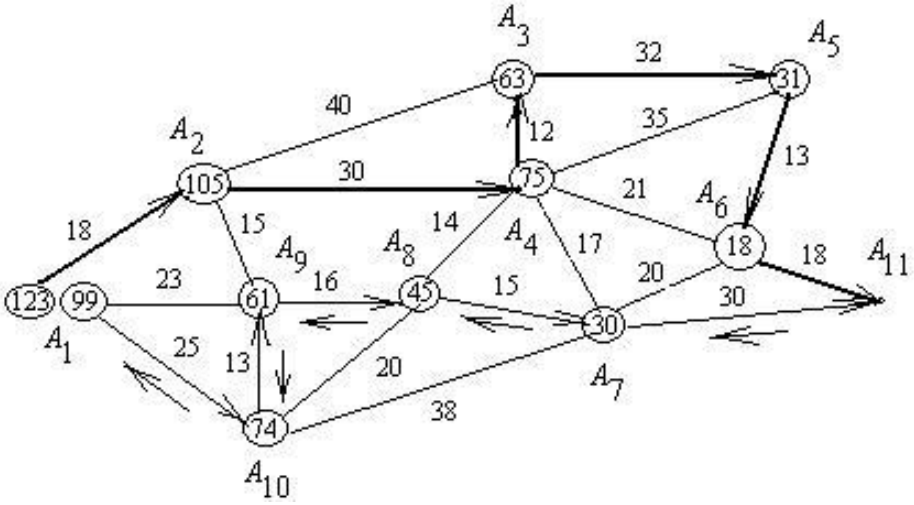
Гамильтонов путь — маршрут, содержащий каждую ***вершину*** графа ровно один раз.

Гамильтонов путь, начальная и конечная вершины которого совпадают, называется **гамильтоновым циклом**.

Гамильтоновы путь, цикл и граф названы в честь ирландского математика У. Гамильтона, который впервые определил эти классы, исследовав задачу «кругосветного путешествия» по додекаэдру, узловые вершины которого символизировали крупнейшие города Земли, а рёбра — соединяющие их дороги.

ВЗВЕШЕННЫЙ ГРАФ (СЕТЬ)

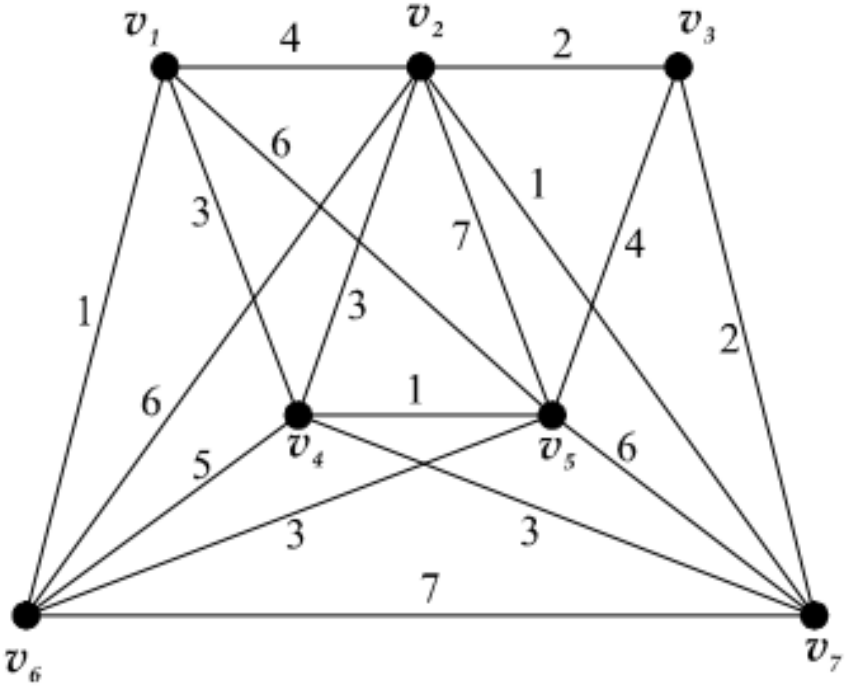
Сетью называется граф, элементам которого поставлены в соответствие некоторые параметры (стоимость, расстояние, пропускная способность и т.п.).



Сеть автомобильных дорог

Построение остовного дерева

В семь населенных пунктов нужно провести кабельное телевидение. Определить маршруты прокладки кабелей вдоль существующих дорог, схема которых представлена на графе, таким образом, чтобы общая длина была минимальной.



Математическая постановка

Предположим, что задан связный граф $<M,N>$, и каждой его дуге $j \in N$ сопоставлено некоторое число $w(j)$, называемое весом или длиной этой дуги.

Сумму весов дуг дерева называют весом дерева.

Требуется найти такое основное дерево, у которого вес был бы минимален.

Такое дерево называют минимальным остовным деревом.

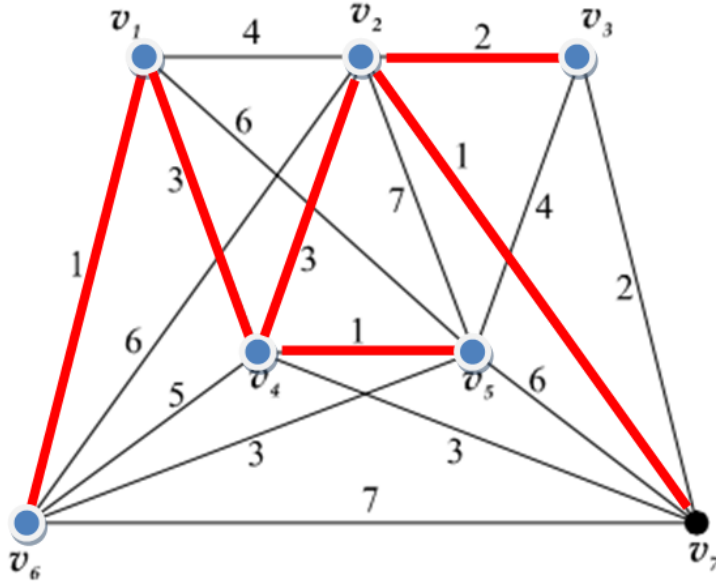
Рассмотрим два метода: алгоритм Прима и алгоритм Краскала

Алгоритм Прима

—алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э.Дейкстрой в 1959 году.

Суть алгоритма: находим ребро минимального веса и выделяем подмножество V* связывающих его вершин. Для всех вершин выделенного подмножества находим ребро с минимальным весом среди всех ребер, ведущих к вершинам, не входящим пока в подмножество V*. Включаем соответствующую вершину в подмножество V*. И так далее, пока все вершины графа не будут включены в подмножествоV*



Алгоритм Прима

Находим ребро минимального веса $(V_1, V_6) = \mathbf{1}$.
Вводим эти вершины в множество $V^*=\{V_1, V_6\}$.
Выбираем ребро минимального веса, исходящее из вершин V^*: $(V_1, V_4)=\mathbf{3}$.
Добавляем V_4 в V^*: $V^*=\{V_1, V_4, V_6\}$

Выбираем ребро минимального веса, смежное с вершинами V^*: $(V_4, V_5)=\mathbf{1}$;
добавляем вершину V_5 в V^*: $V^*=\{V_1, V_4, V_5, V_6\}$
Выбираем ребро минимального веса, смежное с вершинами V^*: $(V_4, V_2) = \mathbf{3}$;
добавляем вершину V_2 в V^*: $V^*=\{V_1, V_2, V_4, V_5, V_6\}$.
Выбираем ребро минимального веса, смежное с вершинами V^*: $(V_2, V_7) = \mathbf{1}$;
добавляем вершину V_7 в V^*: $V^*=\{V_1, V_2, V_4, V_5, V_6, V_7\}$.
Выбираем ребро минимального веса, смежное с вершинами V^*: $(V_2, V_3) = \mathbf{2}$;
добавляем вершину V_3 в V^*: $V^*=\{V_1, V_2, V_3, V_4, V_5, V_6, V_7\}$.
Неохваченных вершин не осталось.

Длина остовного дерева: $\boldsymbol{L}= 1+3+1+3+1+2 = \boldsymbol{11}$

Алгоритм Краскала

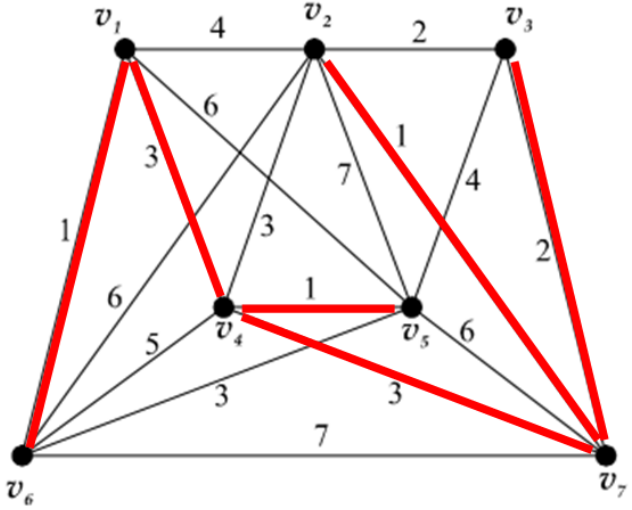
Алгоритм, более привлекательный в вычислительном отношении, был предложен Джозефом Краскалом в 1957 г.

Алгоритм состоит из двух фаз.

Джозеф Краскал

- На подготовительной фазе все дуги удаляются из дерева и упорядочиваются по возрастанию их весов. В графе остаются только вершины, каждая из которых образует отдельную компоненту связности.
- Во второй фазе дуги перебираются в порядке возрастания веса. Если начало и конец очередной дуги принадлежат одной и той же компоненте связности, дуга игнорируется. Если же они лежат в разных компонентах связности. дуга добавляется к графу, а эти две компоненты связности объединяются в одну. Если число компонент связности дойдет до 1, цикл завершается досрочно.

Алгоритм Краскала



Находим ребро минимального веса 1: $(V_1, V_6) = \mathbf{1}$

Вводим вершины в множество $V^*=\{ V_1, V_6\}$

Находим ребро веса 1: $(V_2, V_7) = \mathbf{1}$.

Вводим вершины в множество $V^*=\{ V_1, V_2, V_6, V_7\}$

Находим ребро веса 1: $(V_4, V_5) = \mathbf{1}$.

Вводим вершины в множество $V^*=\{ V_1, V_2, V_4, V_5, V_6, V_7\}$

Ребер веса 1 больше нет.

Теперь вводим ребра веса 2, так, *чтобы не образовать циклы, но повышать связность графа*.

Вводим в дерево ребро веса 2: $(V_3, V_7) = \mathbf{2}$.

Вводим вершины в множество $V^*=\{ V_1, V_2, V_3, V_4, V_5, V_6, V_7\}$

Ребер веса 2 (не образующих циклов с существующими) больше нет.

Теперь вводим ребра веса 3, так, *чтобы не образовать циклы, но повышать связность графа.*

Находим ребро веса 3: $(V_1, V_4) = \mathbf{3}$.

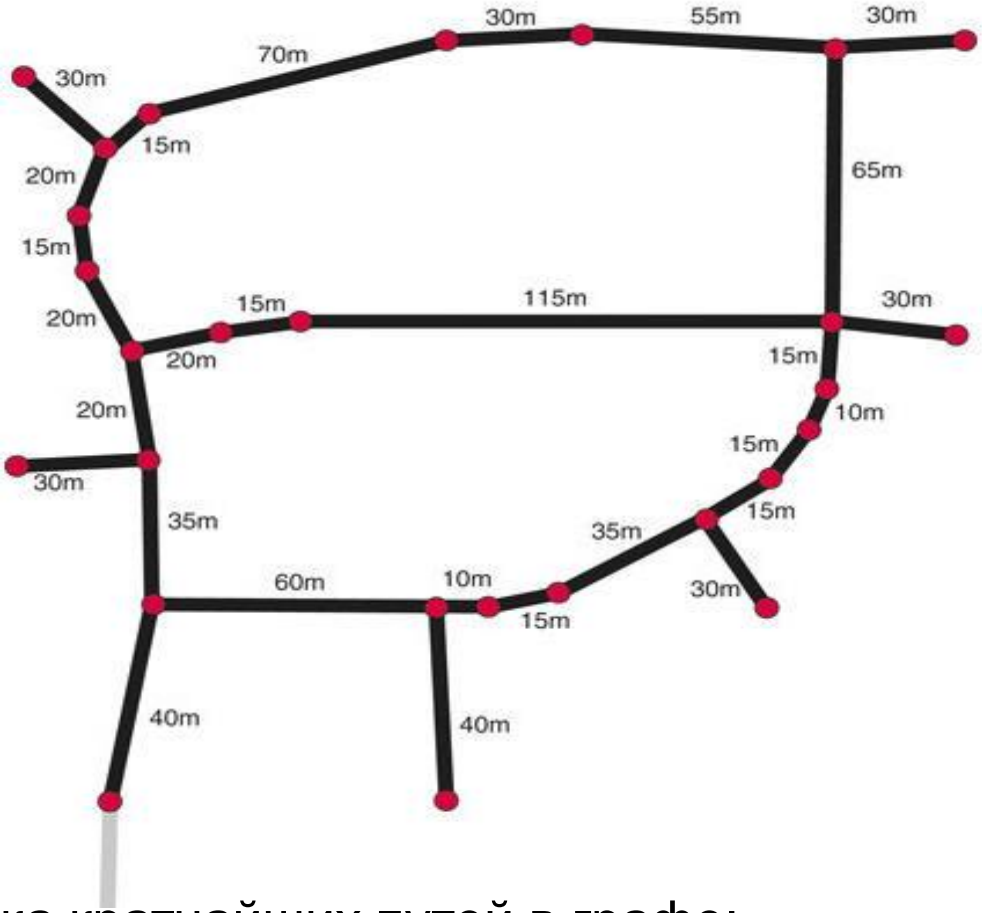
Находим ребро веса 3: $(V_4, V_7) = \mathbf{3}$.

Все вершины включены в дерево. Граф связный.

Вес дерева $\boldsymbol{L}$ = 1+1+1+2+3+3 = ***11***.

Поиск кратчайшего пути в графе

У нас есть две точки:
начало и конец маршрута.
Также у нас есть карта,
представляющая собой
большой массив векторов.
Мы должны использовать
его и рассчитать
оптимальный маршрут.



Существует несколько методов поиска кратчайших путей в графе:

- алгоритм Дейкстры (поиск кратчайших путей *от заданной* вершины);
- алгоритм Флойда (поиска длин *всех* кратчайших путей в графе);
- алгоритм Данцига.

Алгоритм Дейкстры

Метод считается одним из наиболее эффективных алгоритмов решения задачи. Предложен в 1959 г. голландским математиком Дейкстрой (1930-2002), известным, в частности, своей борьбой против безбрежного использования в программировании оператора ***go to***. Эта борьба привела к существенному улучшению стиля программирования.

Эдсгер Вибе **Дейкстра**

Главная идея, лежащая в основе алгоритма Дейкстры
Предположим, что нам известны m вершин, ближайших к вершине **S** (близость любой вершины **X** к вершине **S** определяется длиной кратчайшего пути, ведущего из **S** в **X**). Пусть также известны сами кратчайшие пути, соединяющие вершину **S** с выделенными m вершинами). Нужно ввести правило, как может быть определена (m+1)-я ближайшая к **S** вершина.

Идея алгоритма Дейкстры

Окрасим вершину **S** и **m** ближайших к ней вершин.

Построим для каждой неокрашенной вершины **Y** пути, непосредственно соединяющие с помощью дуг (x, y) каждую окрашенную вершину **X** с **Y**.

Выберем из этих путей кратчайший, и будем считать его условно кратчайшим путем из вершины **S** в вершину **Y**.

Какая же из неокрашенных вершин является (m+1)-й ближайшей к **S** вершиной?

Та, для которой условно кратчайший путь имеет наименьшую длину. Это обусловливается тем, что кратчайший путь из вершины **S** в (m+1)-ю ближайшую вершину при положительном значении длин всех дуг должен содержать в качестве промежуточных лишь окрашенные вершины, т. е. вершины, входящие в число **m** вершин, ближайших к вершине **S**.

Итак, если известны m ближайших к s вершин, то и (m+1)-я ближайшая к **S** вершина может быть найдена.

Начиная с m = 0, описанная процедура может повторяться до тех пор, пока не будет получен кратчайший путь, ведущий из вершины **S** к вершине **T**.

Алгоритм Дейкстры

1) Каждой вершине X в ходе алгоритма присваивается число d(x), равное длине кратчайшего пути из вершины S в вершину X и включающем только окрашенные вершины.
2) Положиv d(s)=0 и d(x)=∞ для всех остальных вершин графа.
3) Окрашиваем вершину S и полагаем Y=S, где Y – последняя окрашенная вершина.
4) Для каждой неокрашенной вершины X пересчитывается величина d(x) по следующей формуле:

$$d(x) = \min \{d(x); d(y) + a_{y,x}\}$$

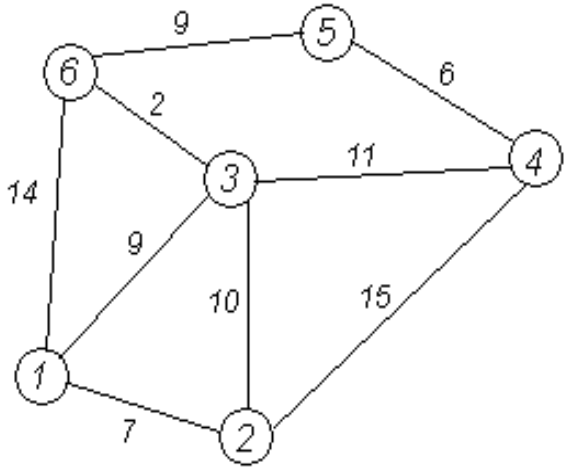
5) Если d(x)=∞ для всех неокрашенных вершин, то алгоритм заканчивается т. к. отсутствуют пути из вершины S в неокрашенные вершины. Иначе окрашивается та вершина, для которой величина d(x) является минимальной.

   Окрашивается и дуга, ведущая в эту вершину, и полагаем Y=X.

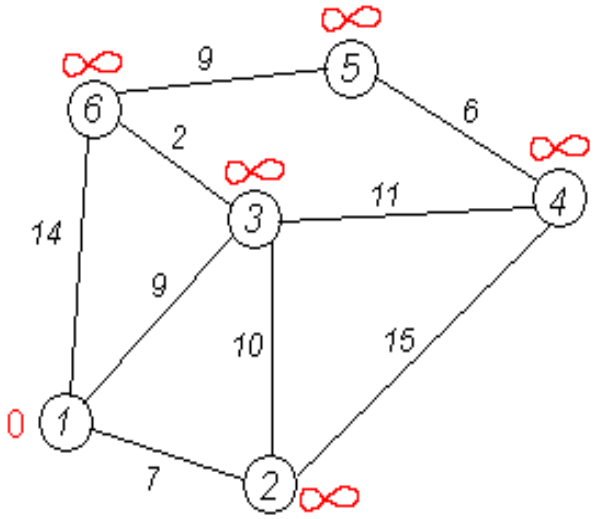
6) Если Y=T, кратчайший путь из S в T найден. Иначе переходим к шагу 2.

ПРИМЕР

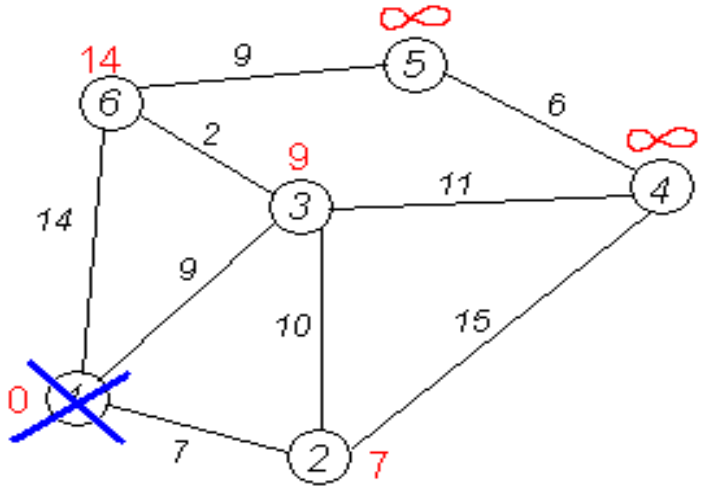
Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных.



Первый шаг.
Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6.

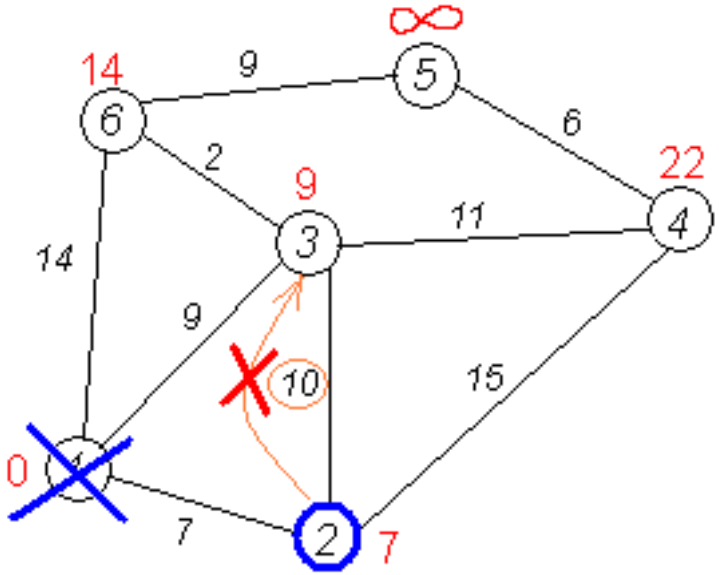


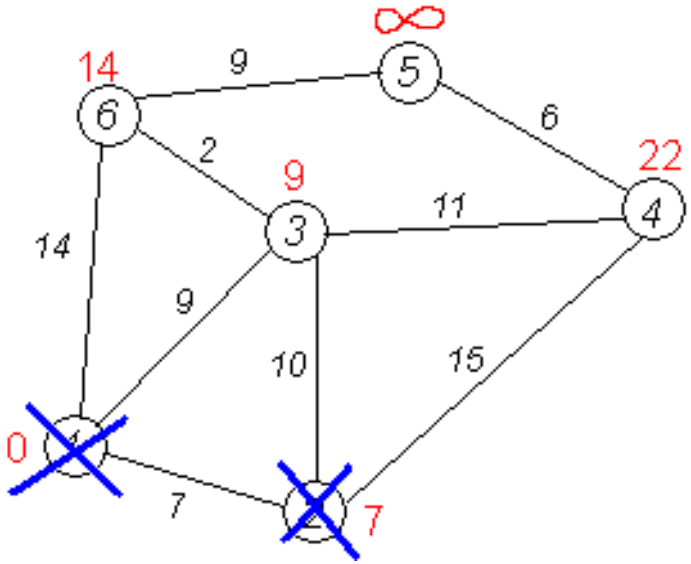
«Проверим» всех «соседей» вершины 1. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит. Вычеркнем её из графа, чтобы отметить, что эта вершина посещена.

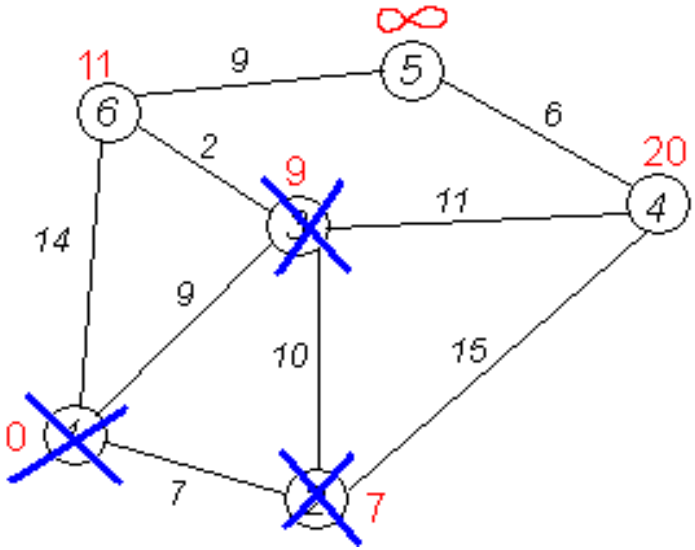


Второй шаг. Шаг алгоритма повторяется. Снова находим «ближайшую» из непосещенных вершин. Это вершина 2 с меткой 7.

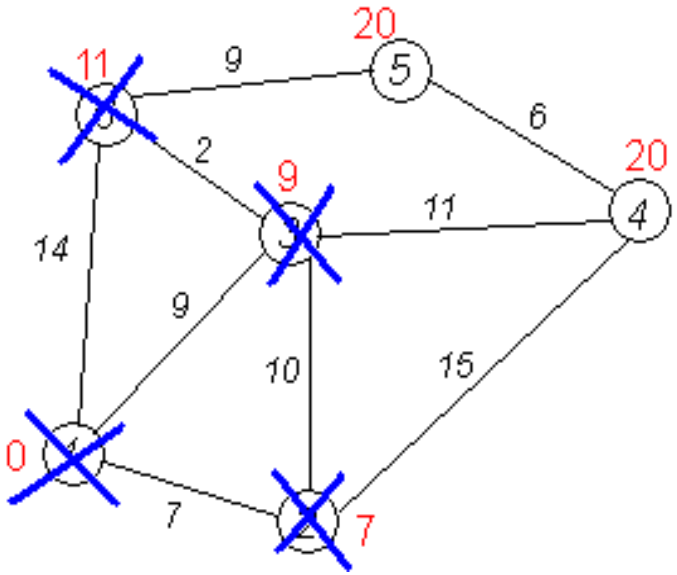
Пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.



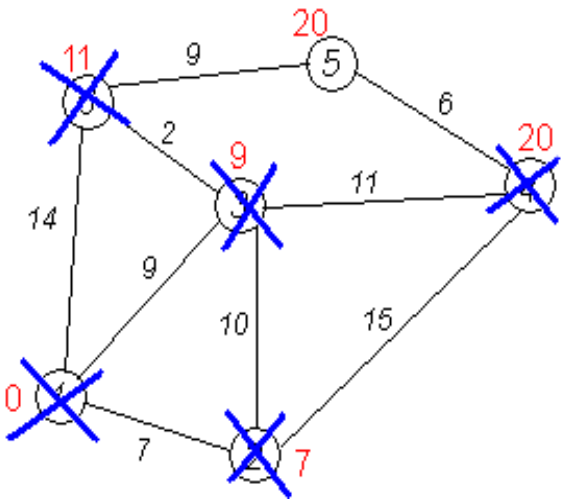




Третий шаг. Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим:



Дальнейшие шаги. Повторяем шаг алгоритма для оставшихся вершин. Это будут вершины 6, 4 и 5, соответственно порядку.



Алгоритм Флойда - Уоршелла

Роберт Флойд

Разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом

В отличии от алгоритма Дейкстры, который позволяет построить ориентированное дерево кратчайших путей от некоторой вершины, метод Флойда позволяет найти длины всех кратчайших путей в графе.

Конечно эта задача может быть решена и многократным применением алгоритма Дейкстры (каждый раз последовательно выбираем вершину от первой до N-ной, пока не получим кратчайшие пути от всех вершин графа), однако реализация подобной процедуры требует значительных вычислительных затрат.

Обозначения

Перенумеруем вершины графа целыми числами от 1 до N.

Обозначим через $d_{i,j}^{m}$ длину кратчайшего пути из вершины *i* в вершину *j*, который в качестве промежуточных может содержать только первые *m* вершин графа (промежуточной вершиной пути является любая принадлежащая ему вершина, не совпадающая с его начальной или конечной вершинами).

Если между вершинами *i* и j не существует ни одного пути указанного типа, то условно будем считать, что $d_{i,j}^{m}=\infty$.

Величина $d_{i,j}^{0}$, представляет длину кратчайшего пути из вершины *i* в вершину *j*, не имеющего промежуточных вершин, т. е. длину кратчайшей дуги, соединяющей *i* с *j* (если такие дуги присутствуют в графе).

Обозначения

Обозначим через D^m матрицу размера NxN, элемент (i,j) которой совпадает с $d_{i,j}^m$.

Если в исходном графе нам известна длина каждой дуги, то мы можем сформировать матрицу D^0, которая в алгоритме Флойда выступает в качестве исходной.

Вначале из этой матрицы вычисляется матрица D^1. Затем по матрице D^1 вычисляется матрица D^2 и т. д. по формуле:

$$d_{i,j}^m = \min\{ d_{i,m}^{m-1} + d_{m,j}^{m-1};\ d_{i,j}^{m-1}\}$$

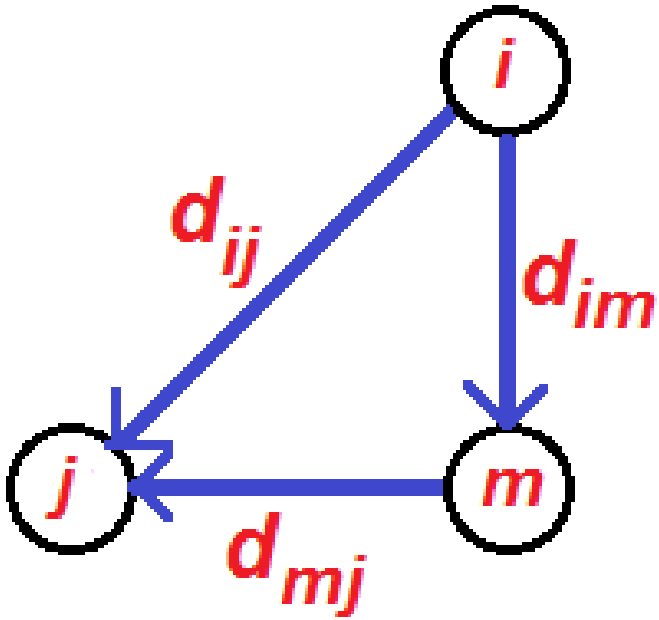
Процесс повторяется до тех пор, пока по матрице D^{N-1} не будет вычислена матрица D^N.

Суть алгоритма Флойда

заключается в проверке того, не окажется ли путь из вершины ***i*** в вершину ***j*** короче, если будет проходить через некоторую промежуточную вершину ***m***.

Пусть есть три вершины – i, j, m – и расстояния между ними: d_{ij}, d_{im}, d_{mj}.

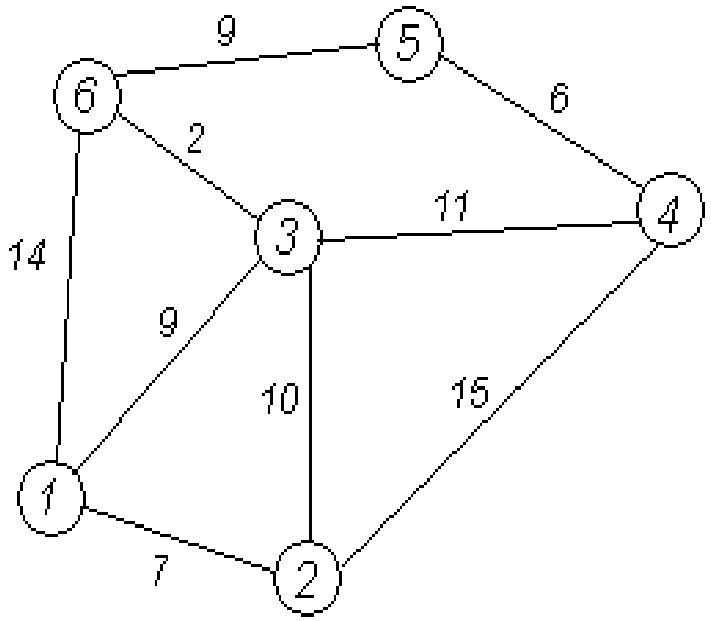
Если выполняется неравенство $d_{im} + d_{mj} < d_{ij}$, то целесообразно заменить путь $i \rightarrow j$ путём $i \rightarrow m \rightarrow j$



«Треугольный оператор»

Алгоритм Флойда на примере

Этап 1. Перенумеровать вершины графа от 1 до N, определить матрицу D^0, каждый элемент $d_{i,j}^0$ которой есть длина кратчайшей дуги между вершинами ***i*** и ***j***. Если такой дуги нет, положить значение элемента $d_{i,j}^0 = \infty$. Значения диагональных элементов $d_{i,i} = 0$.



D^0	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	∞
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	∞	2	∞	9	0

Алгоритм Флойда на примере

Матрицу S^0, в которой будем запоминать последовательность вершин в кратчайшем пути, заполним так: $s_{ij} = j$

D^0

	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	∞
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	∞	2	∞	9	0

S^0

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

Алгоритм Флойда на примере

Основной этап

Задаём строку m и столбец m как *ведущую строку* и *ведущий столбец*.

Для всех элементов d_{ij} матрицы D^{m-1} рассматриваем возможность применения *треугольного оператора*. Если выполняется неравенство:

$$d_{im} + d_{mj} < d_{ij} \quad (i \neq j,\ j \neq m,\ i \neq m),$$

то делаем следующее:

a) в матрице D^m заменяем элемент d_{ij} матрицы D^{m-1} суммой $d_{im} + d_{mj}$;

b) в матрице S^m меняем элемент s_{ij} матрицы S^{m-1} на m;

c) полагаем $m=m+1$, повторяем основной этап, пока $m < N$

Алгоритм Флойда на примере

D^0

	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	∞
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	∞	2	∞	9	0

S^0

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

D^1

	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	21
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	21	2	∞	9	0

S^1

	1	2	3	4	5	6
1	1	2	3	4	5	6
2	1	2	3	4	5	*1*
3	1	2	3	4	5	6
4	1	2	3	4	5	6
5	1	2	3	4	5	6
6	1	*1*	3	4	5	6

Алгоритм Флойда на примере

D^1

	1	2	3	4	5	6
1	0	7	9	∞	∞	14
2	7	0	10	15	∞	21
3	9	10	0	11	∞	2
4	∞	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	21	2	∞	9	0

S^1

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

D^2

	1	2	3	4	5	6
1	0	7	9	22	∞	14
2	7	0	10	15	∞	21
3	9	10	0	11	∞	2
4	22	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	21	2	36	9	0

S^2

	1	2	3	4	5	6
1	1	2	3	*2*	5	6
2	1	2	3	4	5	1
3	1	2	3	4	5	6
4	*2*	2	3	4	5	6
5	1	2	3	4	5	6
6	1	1	3	*2*	5	6

Алгоритм Флойда на примере

D^2

	1	2	3	4	5	6
1	0	7	9	22	∞	14
2	7	0	10	15	∞	21
3	9	10	0	11	∞	2
4	22	15	11	0	6	∞
5	∞	∞	∞	6	0	9
6	14	21	2	36	9	0

S^2

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

D^3

	1	2	3	4	5	6
1	0	7	9	20	∞	11
2	7	0	10	15	∞	12
3	9	10	0	11	∞	2
4	20	15	11	0	6	13
5	∞	∞	∞	6	0	9
6	11	12	2	13	9	0

S^3

	1	2	3	4	5	6
1	1	2	3	***3***	5	***3***
2	1	2	3	4	5	***3***
3	1	2	3	4	5	6
4	***3***	2	3	4	5	***3***
5	1	2	3	4	5	6
6	***3***	***3***	3	***3***	5	6

Алгоритм Флойда на примере

D^3

	1	2	3	4	5	6
1	0	7	9	20	∞	11
2	7	0	10	15	∞	12
3	9	10	0	11	∞	2
4	20	15	11	0	6	13
5	∞	∞	∞	6	0	9
6	11	12	2	13	9	0

S^3

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

D^4

	1	2	3	4	5	6
1	0	7	9	20	26	11
2	7	0	10	15	21	12
3	9	10	0	11	17	2
4	20	15	11	0	6	13
5	26	21	17	6	0	9
6	11	12	2	13	9	0

S^4

	1	2	3	4	5	6
1	1	2	3	3	*4*	3
2	1	2	3	4	*4*	3
3	1	2	3	4	*4*	6
4	3	2	3	4	5	3
5	*4*	*4*	*4*	4	5	6
6	3	3	3	3	5	6

Алгоритм Флойда на примере

D^4

	1	2	3	4	5	6
1	0	7	9	20	26	11
2	7	0	10	15	21	12
3	9	10	0	11	17	2
4	20	15	11	0	6	13
5	26	21	17	6	0	9
6	11	12	2	13	9	0

S^4

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

D^5

	1	2	3	4	5	6
1	0	7	9	20	26	11
2	7	0	10	15	21	12
3	9	10	0	11	17	2
4	20	15	11	0	6	13
5	26	21	17	6	0	9
6	11	12	2	13	9	0

S^5

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

Алгоритм Флойда на примере

D^5

	1	2	3	4	5	6
1	0	7	9	20	26	11
2	7	0	10	15	21	12
3	9	10	0	11	17	2
4	20	15	11	0	6	13
5	26	21	17	6	0	9
6	11	12	2	13	9	0

S^5

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

D^6

	1	2	3	4	5	6
1	0	7	9	20	20	11
2	7	0	10	15	21	12
3	9	10	0	11	11	2
4	20	15	11	0	6	13
5	20	21	11	6	0	9
6	11	12	2	13	9	0

S^6

	1	2	3	4	5	6
1	1	2	3	3	*6*	3
2	1	2	3	4	4	3
3	1	2	3	4	*6*	6
4	3	2	3	4	5	3
5	*6*	4	*6*	4	5	6
6	3	3	3	3	5	6

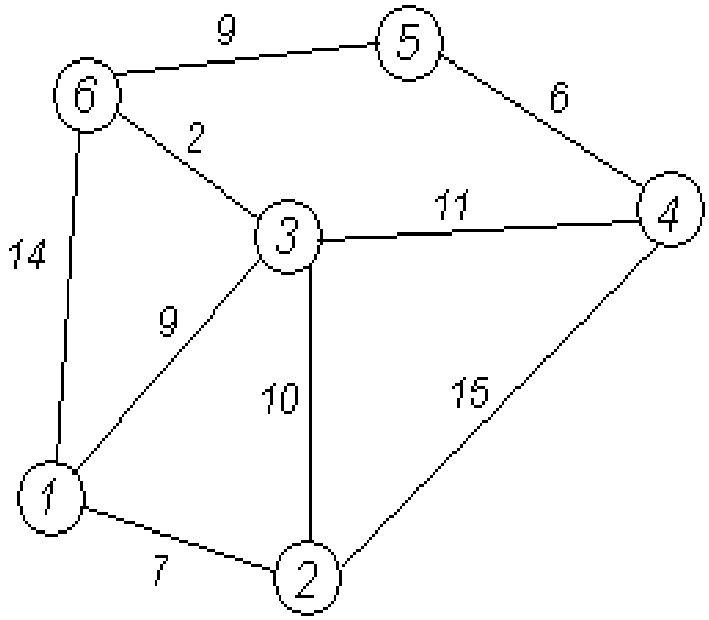
Алгоритм Флойда на примере

Последний этап.

После реализации ***N*** этапов алгоритма определение по матрицам ***Dn*** и ***Sn*** кратчайшего пути между узлами ***i*** и ***j*** выполняется по следующим правилам:

1. Кратчайшее расстояние между узлами ***i*** и ***j*** равно элементу d_{ij} в матрице D^n.
2. Промежуточные узлы пути от узла ***i*** к узлу ***j*** определяем по матрице S^n. Пусть $s_{ij} = k$, тогда имеем путь $i \rightarrow j \rightarrow k$.
   Если далее $s_{ik} = k$ и $s_{kj} = j$, то считаем, что весь путь определён. В противном случае повторяем процедуру для путей от узла ***i*** к узлу ***k*** и от узла ***k*** к узлу ***j***.

Алгоритм Флойда на примере



d_{25} = 21

Путь: 2→4 →5

d_{51} = 20

Путь: 5→6 →3 → 1

D^6

	1	2	3	4	5	6
1	0	7	9	20	20	11
2	7	0	10	15	21	12
3	9	10	0	11	11	2
4	20	15	11	0	6	13
5	20	21	11	6	0	9
6	11	12	2	13	9	0

S^6

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

Задача коммивояжёра

Формулировка (1934 г.)

Коммивояжер должен выйти из первого города, посетить по разу в неизвестном порядке города 2, 3, …, n и вернуться в первый город. Расстояния между городами известны. В каком порядке следует обходить города, чтобы замкнутый путь (тур) коммивояжера был кратчайшим?

В терминах теории графов задача формулируется так: имеется полный ориентированный граф $G = (M, N)$, каждой дуге (i, j) которого сопоставлен вес d_{ij}. Требуется найти в этом графе гамильтонов контур наименьшей стоимости.

Метод ветвей и границ ("поиск с возвратом", "backtracking")

Идея метода состоит в следующем:

чтобы избежать полного перебора нужно разделить огромное число перебираемых вариантов на классы и получить оценки (снизу – в задаче минимизации, сверху – в задаче максимизации) для этих классов, чтобы иметь *возможность отбрасывать варианты не по одному, а целыми классами*.

Трудность состоит в том, чтобы найти такое разделение на классы (ветви) и такие оценки (границы), чтобы процедура была эффективной.

Алгоритм Литтла для задачи коммивояжера (Литтла, Мурти, Суини и Кэрел)

Применительно к задаче о коммивояжере идея метода ветвей и границ такова:

Ветвление основано на следующем простом соображении: *переезд из любого данного города i в любой другой город j может либо принадлежать оптимальному циклу коммивояжера, либо не принадлежать ему*. При вычислении же границ используется тот факт, что ***изменение длин*** всех путей, приводящих в данный город, или всех путей, выводящих из данного города, ***на одну и ту же величину*** приводит к новой задаче, оптимальный план которой совпадает с оптимальным планом исходной задачи.

Алгоритм Литтла

1. В каждой строке матрицы стоимости найдем минимальный элемент и вычтем его из всех элементов строки. Сделаем это и для столбцов, не содержащих нуля. Получим матрицу стоимости, каждая строка и каждый столбец которой содержат *хотя бы один нулевой элемент*.
2. Для каждого нулевого элемента матрицы d_{ij} рассчитаем коэффициент Γ_{ij}, который равен сумме наименьшего элемента i строки (исключая элемент $d_{ij}=0$) и наименьшего элемента j столбца. Из всех коэффициентов Γ_{ij} выберем такой, который является максимальным $\Gamma_{kl} = max\{\Gamma_{ij}\}$. В гамильтонов контур вносится соответствующая дуга (k,l).

Алгоритм Литтла

3. Удаляем k-тую строку и столбец l, поменяем на бесконечность значение элемента d_{lk} (поскольку дуга (k,l) включена в контур, то обратный путь из l в k недопустим).
4. Повторяем алгоритм с шага 1, пока порядок матрицы не станет равным двум.
5. Затем в текущий ориентированный граф вносим две недостающие дуги, определяющиеся однозначно матрицей прядка 2. Получаем гамильтонов контур.

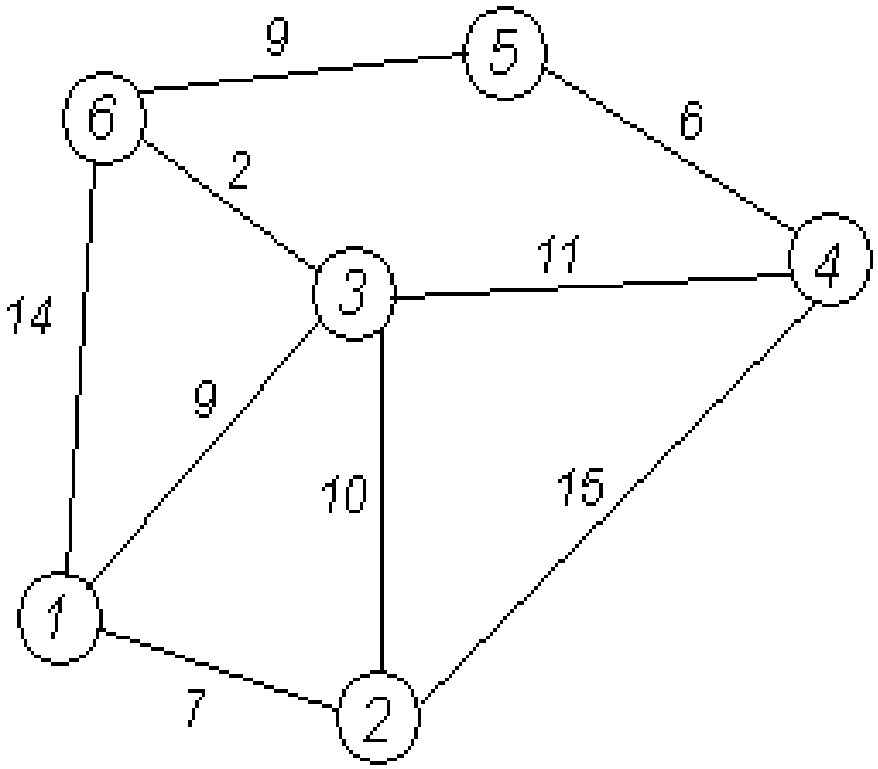
Алгоритм Литтла

В ходе решения ведется постоянный подсчет текущего значения **нижней границы**.

Нижняя граница равна сумме всех вычтенных элементов в строках и столбцах.

Итоговое значение нижней границы должно совпасть с длиной результирующего контура (пути).

Алгоритм Литтла (пример)



	1	2	3	4	5	6
1	∞	7	9	20	20	11
2	7	∞	10	15	21	12
3	9	10	∞	11	11	2
4	20	15	11	∞	6	13
5	20	21	11	6	∞	9
6	11	12	2	13	9	∞

Матрица кратчайших расстояний, где $d_{ii} = \infty$

Алгоритм Литтла (пример. Шаг 1)

	1	2	3	4	5	6
1	∞	7	9	20	20	11
2	7	∞	10	15	21	12
3	9	10	∞	11	11	2
4	20	15	11	∞	6	13
5	20	21	11	6	∞	9
6	11	12	2	13	9	∞

мин
7
7
2
6
6
2

Сумма констант приведения равна 7+7+2+6+6+2 = **30**

	1	2	3	4	5	6
1	∞	0	2	13	13	4
2	0	∞	3	8	14	5
3	7	8	∞	9	9	0
4	14	9	5	∞	0	7
5	14	16	5	0	∞	3
6	9	10	0	11	7	∞

Во всех столбцах есть нулевые элементы.
Приводить по столбцам не нужно

Алгоритм Литтла (пример. Шаг 1)

	1	2	3	4	5	6
1	∞	0	2	13	13	4
2	0	∞	3	8	14	5
3	7	8	∞	9	9	0
4	14	9	5	∞	0	7
5	14	16	5	0	∞	3
6	9	10	0	11	7	∞

Тур, проходящий только через ребра нулевой стоимости, будет, очевидно, минимальным.

Его стоимость 30.

Таким образом, мы получили нижнюю оценку стоимости класса всех возможных туров.

То есть минимальный тур в данной задаче не может стоить меньше, чем 30.

Алгоритм Литтла (пример. Шаг 2)

Назовем *оценкой нуля* в позиции (*i*, *j*) в матрице сумму минимальных элементов в *i*-й строке и *j*-м столбце (не считая сам этот ноль).

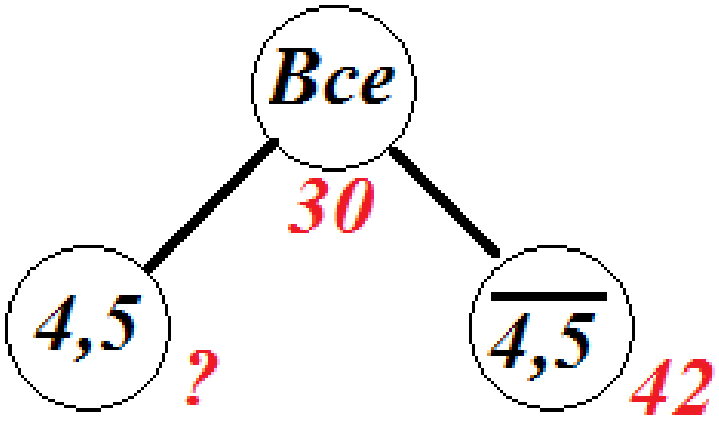
Оценим каждый ноль в приведенной матрице:

	1	2	3	4	5	6
1	∞	0 (2+8)	2	13	13	4
2	0 (3+7)	∞	3	8	14	5
3	7	8	∞	9	9	0 (3+7)
4	14	9	5	∞	0 (5+7)	7
5	14	16	5	0 (3+8)	∞	3
6	9	10	0 (2+7)	11	7	∞

max=12

Алгоритм Литтла (пример. Шаг 2)

Оценка *k* нуля, в позиции (*i, j*) означает буквально следующее: если в тур не будет включен путь из *i* в *j* (стоимостью 0), то придется доплатить как минимум k. Поэтому, можно разделить класс всех возможных туров на два: *туры, содержащие ребро (i, j) и туры, не содержащие его*. Для последних минимальная оценка увеличится на *k*.



Алгоритм Литтла (пример. Шаг 3)

	1	2	3	4	5	6
1	∞	0	2	13	13	4
2	0	∞	3	8	14	5
3	7	8	∞	9	9	0
4	14	9	5	∞	0	7
5	14	16	5	0	∞	3
6	9	10	0	11	7	∞

	1	2	3	4	6
1	∞	0	2	13	4
2	0	∞	3	8	5
3	7	8	∞	9	0
5	14	16	5	∞	3
6	9	10	0	11	∞

Рассмотрим ребро, соответствующее нулю с максимальной оценкой. В данном случае это ребро (4, 5). Нижняя оценка стоимости класса туров, не содержащих (4,5) увеличивается до 30+12=42.

Чтобы определить оценку для класса туров, содержащих дугу (4,5), *удалим из матрицы строку 4 и столбец 5*. Элемент d_{54} заменим ∞, чтобы не было замкнутого контура.

Алгоритм Литтла (пример. Шаг 1)

	1	2	3	4	6
1	∞	0	2	13	4
2	0	∞	3	8	5
3	7	8	∞	9	0
5	14	16	5	∞	3
6	9	10	0	11	∞

мин
0
0
0
3
0

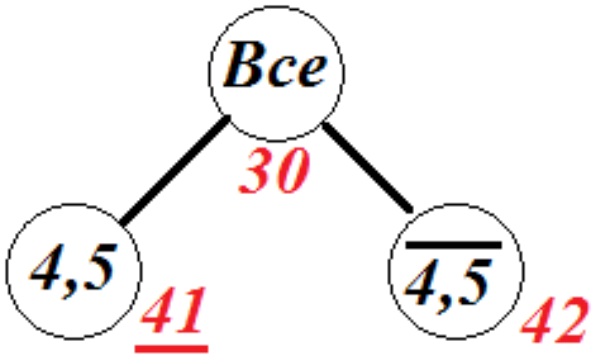
Сумма констант приведения равна 3 + 8 = 11

	1	2	3	4	6
1	∞	0	2	13	4
2	0	∞	3	8	5
3	7	8	∞	9	0
5	11	13	2	∞	0
6	9	10	0	11	∞

мин	0	0	0	8	0

	1	2	3	4	6
1	∞	0	2	5	4
2	0	∞	3	0	5
3	7	8	∞	1	0
5	11	13	2	∞	0
6	9	10	0	3	∞

Алгоритм Литтла (пример. Шаг 2)



Дугу (4, 5) включаем в путь

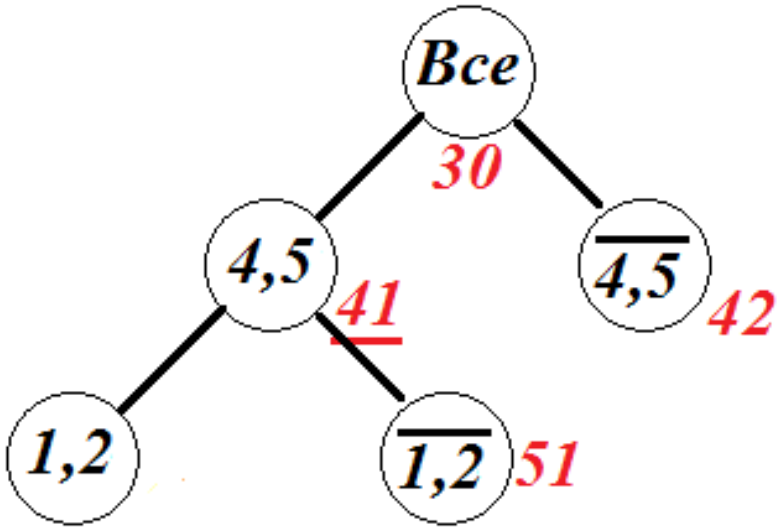
	1	2	3	4	6
1	∞	0 (2+8)	2	5	4
2	0 (0+7)	∞	3	0 (0+1)	5
3	7	8	∞	1	0 (0+1)
5	11	13	2	∞	0 (0+2)
6	9	10	0 (2+3)	3	∞

max=10

Рассчитываем оценку для каждого нулевого элемента как сумму минимальных по строке и столбцу

Алгоритм Литтла (пример. Шаг 3)

Нижняя оценка стоимости класса туров, не содержащих (1,2) увеличивается до 41+10=51.



Чтобы определить оценку для класса туров, содержащих дугу (1,2), *удалим из матрицы строку 1 и столбец 2*. Элемент d_{21} заменим ∞, чтобы не было замкнутого контура.

Алгоритм Литтла (пример. Шаг 3)

	1	2	3	4	6
1	∞	0	2	5	4
2	0	∞	3	0	5
3	7	8	∞	1	0
5	11	13	2	∞	0
6	9	10	0	3	∞

	1	3	4	6
2	∞	3	0	5
3	7	∞	1	0
5	11	2	∞	0
6	9	0	3	∞

Алгоритм Литтла (пример. Шаг 1,2)

Приведем матрицу, вычитая минимальные значения по строкам и столбцам.

	1	3	4	6
2	∞	3	0	5
3	7	∞	1	0
5	11	2	∞	0
6	9	0	3	∞

мин
0
0
0
0

мин	7	0	0	0

	1	3	4	6
2	∞	3	0	5
3	0	∞	1	0
5	4	2	∞	0
6	2	0	3	∞

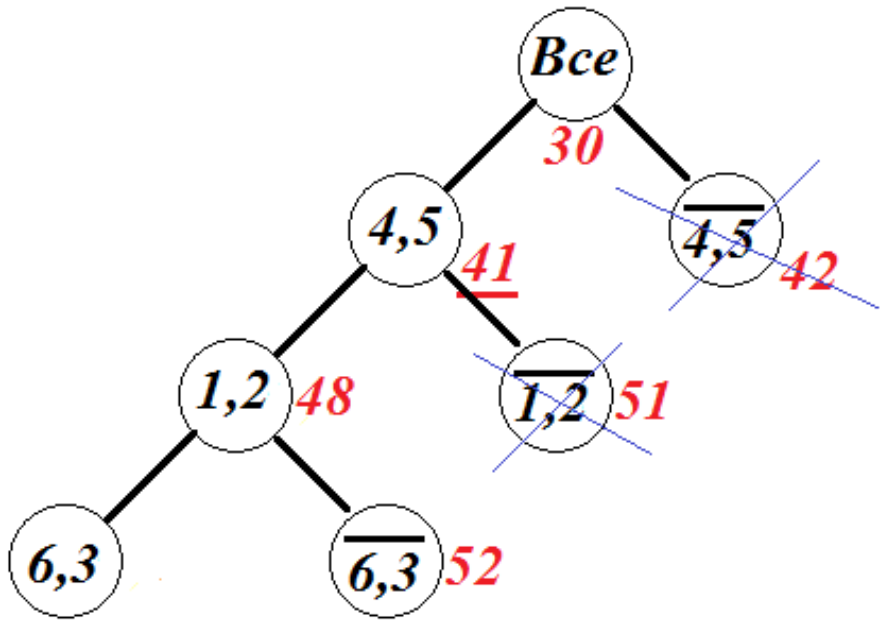
Сумма констант приведения равна 0 + 7 = 7

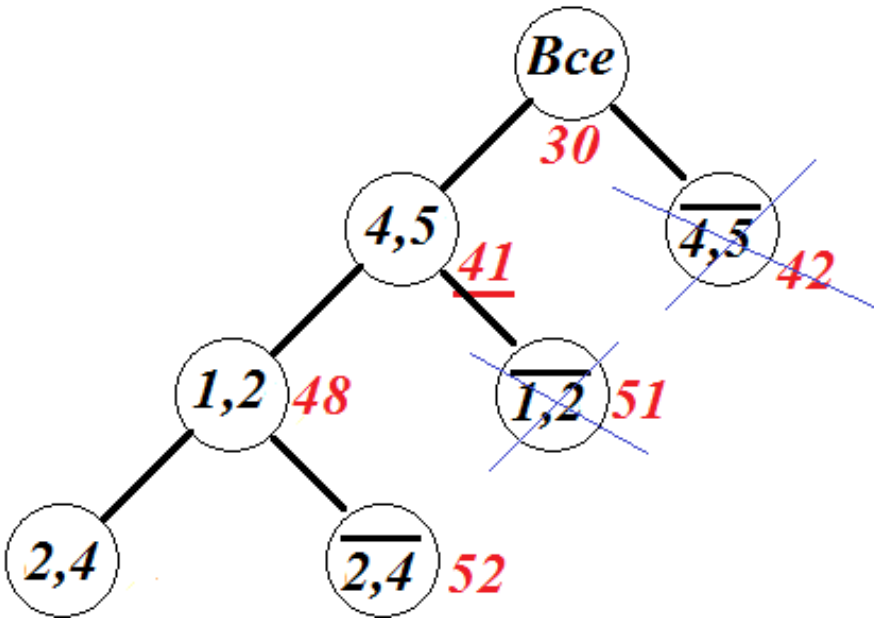
Оценим нулевые элементы

	1	3	4	6
2	∞	3	0 (3+1)	5
3	0 (0+2)	∞	1	0 (0+0)
5	4	2	∞	0
6	2	0 (2+2)	3	∞

max=4

Алгоритм Литтла (пример. Шаг 1,2)





Поскольку у двух нулевых элементов (6,3) и (2,4) одинаковые оценки, то нужно рассмотреть оба варианта.

Алгоритм Литтла (пример. Шаг 1,2)

	1	3	4	6
2	∞	3	0	5
3	0	∞	1	0
5	4	2	∞	0
6	2	0	3	∞

Удаляем 6 строку и

3 столбец; d_{36}= ∞

	1	4	6
2	∞	0	5
3	0	1	∞
5	4	∞	0

В каждой строке и столбце есть 0, поэтому сумма констант приведения равна 0

Получаем оценку нулевых элементов

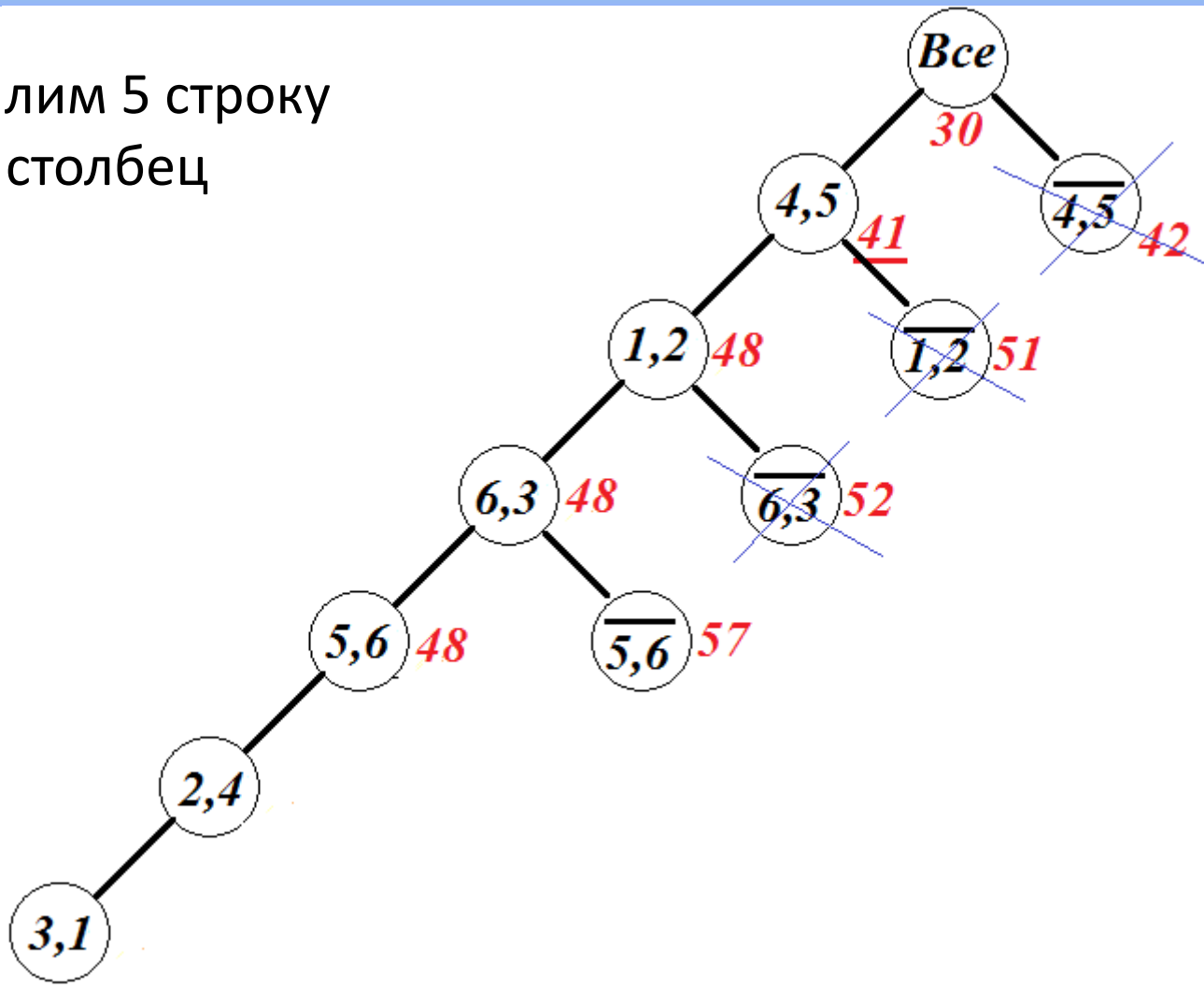
	1	4	6
2	∞	0 (5+1)	5
3	0 (4+1)	1	∞
5	4	∞	0 (4+5)

max=9

Алгоритм Литтла (пример. Шаг 3)

	1	4	6
2	∞	0	5
3	0	1	∞
5	4	∞	0

Удалим 5 строку
и 6 столбец



	1	4
2	∞	0
3	0	∞

Осталась
матрица 2x2.

Первый путь
найден.

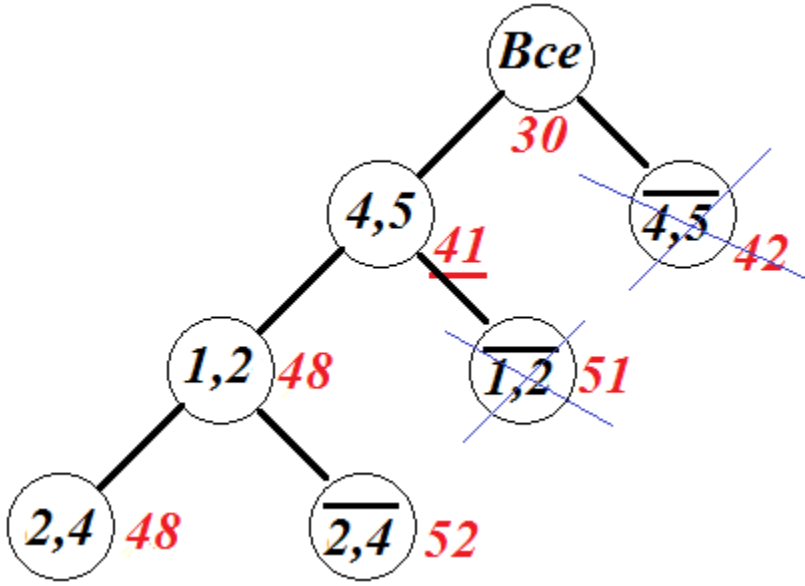
Путь: 1 → 2 → 4 → 5 → 6 → 3 → 1.

Длина пути: 48.

Алгоритм Литтла (пример. Возврат к слайду 56)

	1	3	4	6
2	∞	3	0	5
3	0	∞	1	0
5	4	2	∞	0
6	2	0	3	∞

Удалим 2 строку
и 4 столбец



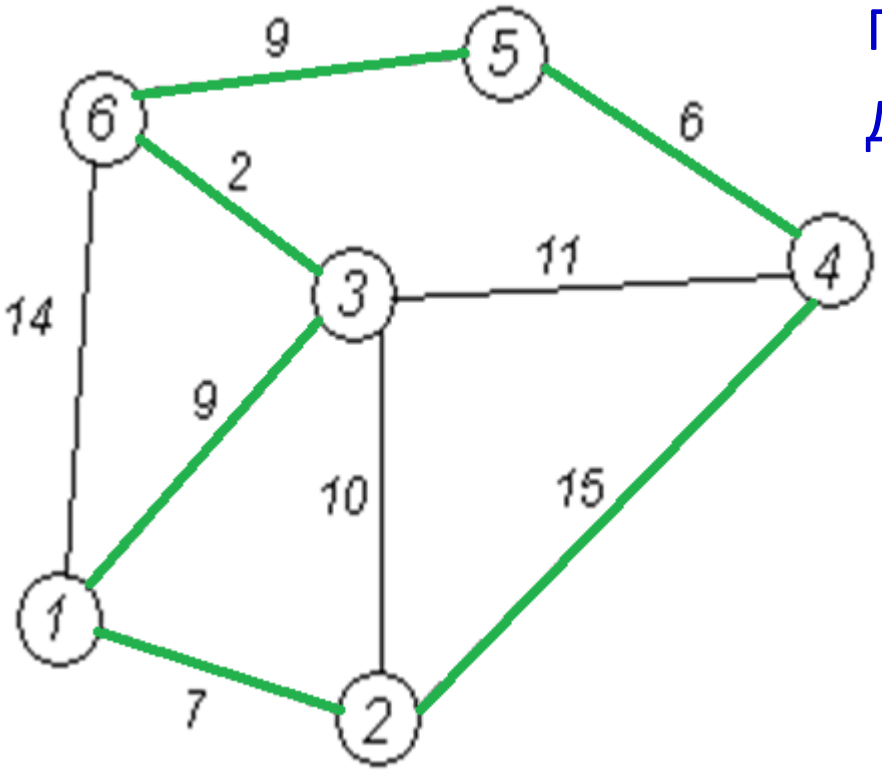
	1	3	6
3	0	∞	0
5	∞	2	0
6	2	0	∞

Сумма констант
приведения =0

	1	3	6
3	0 (0+2)	∞	0
5	∞	2	0 (0+2)
6	2	0 (0+2)	∞

Оценки совпадают у трёх нулевых элементов, но все дуги входят в ранее найденный путь длиной 48

Алгоритм Литтла (оптимальное решение)



Путь: 1 → 2 → 4 → 5 → 6 → 3 → 1.

Длина пути: 48.