Алгоритмы поиска кратчайшего пути

Преподаватель: Солодухин Андрей Геннадьевич

План:

1. Постановка задачи о кратчайших путях.
2. Алгоритм Дийкстры нахождения кратчайших путей до всех вершин

ВСПОМИНАЕМ:

Граф называется взвешенным или ***сетью***, если каждому его ребру поставлено в соответствие некоторое число (вес).

Взвешенными графами могут быть схемы в электронике, электрические схемы, схемы компьютерных сетей, карты автомобильных и железных дорог и др.

На картах автодорог вершины являются населенными пунктами, ребра — дорогами, а весом — числа, равные расстоянию между населенными пунктами.

Ребрам графа могут соответствовать числа, означающие длину, уклон, запланированное время и другие характеристики.

ЗАДАЧА О КРАТЧАЙШЕМ ПУТИ

Задача о кратчайшем пути— задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов рёбер, составляющих путь.

в GPS-навигаторах осуществляется поиск кратчайшего пути между двумя перекрестками. В качестве вершин выступают перекрестки, а дороги являются ребрами, которые лежат между ними. Если сумма длин дорог между перекрестками минимальна, тогда найденный путь самый короткий.

ВАРИАНТЫ ПОСТАНОВКИ

Задача о кратчайшем пути в заданный пункт назначения.
Требуется найти кратчайший путь в заданную вершину назначения t, который начинается в каждой из вершин графа (кроме t).

Задача о кратчайшем пути между заданной парой вершин.
Требуется найти кратчайший путь из заданной вершины u в заданную вершину v.

Задача о кратчайшем пути между всеми парами вершин.
Требуется найти кратчайший путь из каждой вершины u в каждую вершину v.

НАИБОЛЕЕ ПОПУЛЯРНЫЕ АЛГОРИТМЫ

Алгоритм Дийкстры находит кратчайший путь от одной из вершин графа до всех остальных

Алгоритм Беллмана — Форда находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным

Алгоритм поиска A* находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе

Алгоритм Флойда — Уоршелла находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа

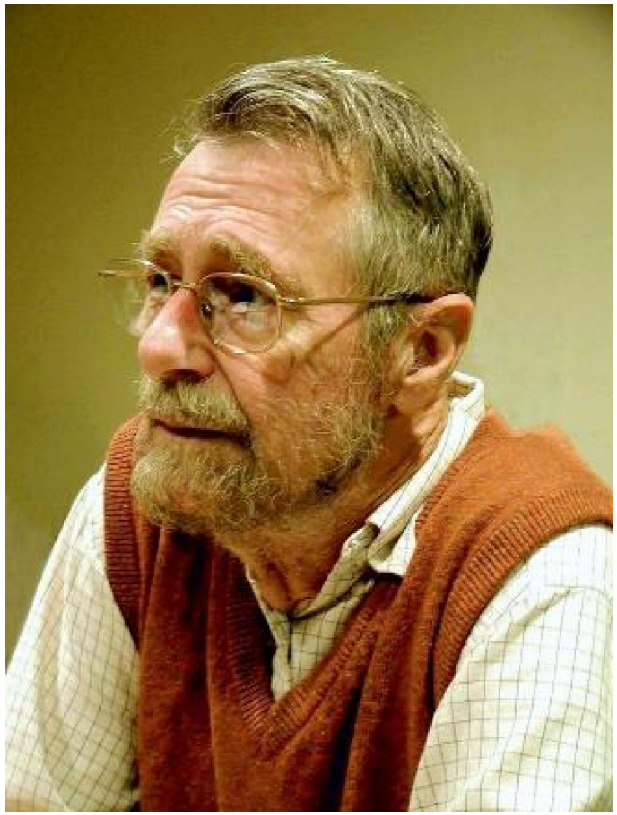
Алгоритм Ли (волновой алгоритм) основан на методе поиска в ширину. Находит путь между вершинами s и t графа, содержащий минимальное количество промежуточных вершин (ребер). Основное применение — трассировки электрических соединений на кристаллах микросхем и на печатных платах.

Задача о кратчайшем пути широко применяется в программировании и технологиях, например, его использует протокол OSPF для устранения кольцевых маршрутов.
OSPF разбивает процесс построения таблицы маршрутизации на 2 этапа. Второй этап состоит в нахождении оптимальных маршрутов с помощью полученного графа.
Задача нахождения оптимального пути на графе является достаточно сложной и ёмкой. Каждый маршрутизатор считает себя центром сети и ищет оптимальный маршрут до каждой известной ему сети.

АЛГОРИТМ ДИЙКСТРЫ

Задан орграф G(V,E), каждой дуге (u,v) ставится в соответствие число ***L(u,v)*** – длина дуги (расстояния, стоимости и т.п.)
В общем случае возможно $L > 0$, $L < 0$, $L = 0$.
Под длиной пути понимаем, как и прежде, сумму длин дуг, составляющих путь.
Найти длины кратчайших путей и сами пути от вершины ***s*** до всех остальных вершин графа v_i.

Известна схема дорог. Требуется перевезти груз из одного пункта в другой по маршруту минимальной длины.
Если в графе нет циклов с отрицательной длиной, то кратчайшие пути существуют и любой кратчайший путь – это простая цепь.



Эдсгер Дийкстра (Edsger Wybe Dijkstra) – нидерландский ученый (структурное программирование, язык Алгол, семафоры, распределенные вычисления)

Лауреат премии Тьюринга (ACM A.M. Turing Award)

1. Дейкстра Э. Дисциплина программирования = A discipline of programming.
2. Дал У., Дейкстра Э., Хоор К. Структурное программирование = Structured Programming.

Алгоритм основан на приписывании вершинам v_i временных меток $d(v_i)$. Метка вершины дает верхнюю границу длины пути от s к этой вершине

Шаг 1 (начальная установка).

Процесс построения дерева начинается с заданной вершины.

Положить d(s)=0, считать метку постоянной.

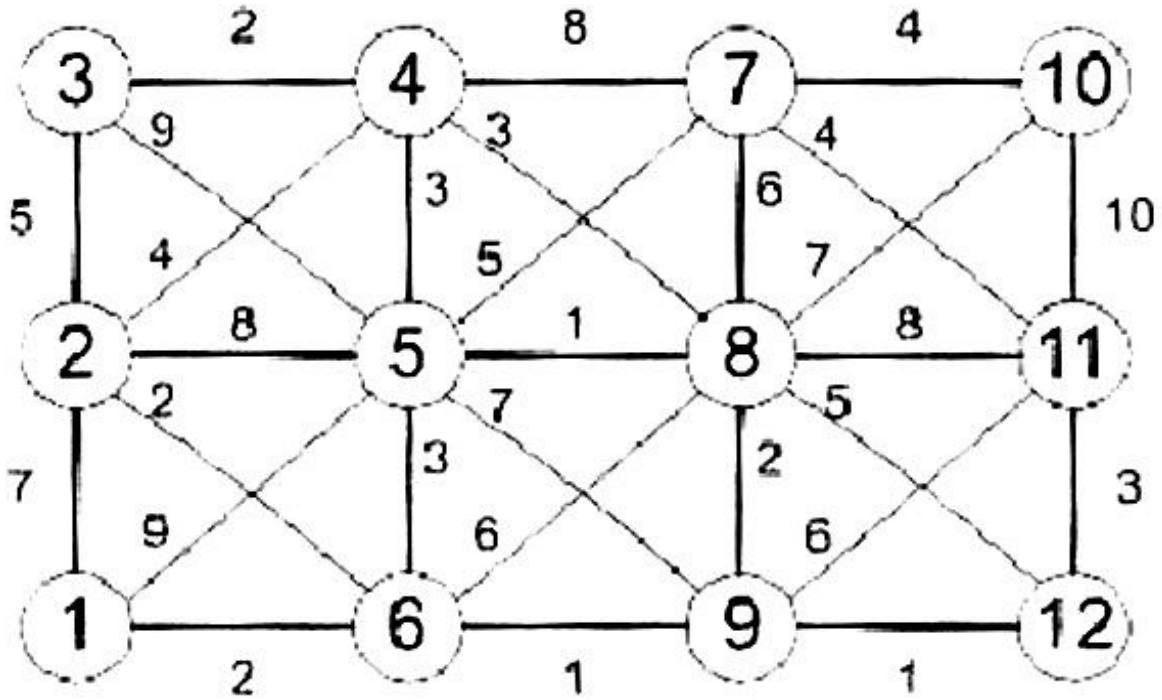
$d(v_i)=\infty$, i=1...n, считать метки временными. ***p=s***.

Шаг 2 (общий шаг).

В дальнейшем на каждом шаге к дереву присоединяется одно новое ребро (и одна вершина). Это ребро выбирается из подходящих ребер, причем подходящим считается ребро, соединяющее вершину дерева с вершиной, ему не принадлежащей. Среди подходящих ребер выбирается ребро наименьшего веса.

Повторяется *n* раз, пока не будут упорядочены все вершины.

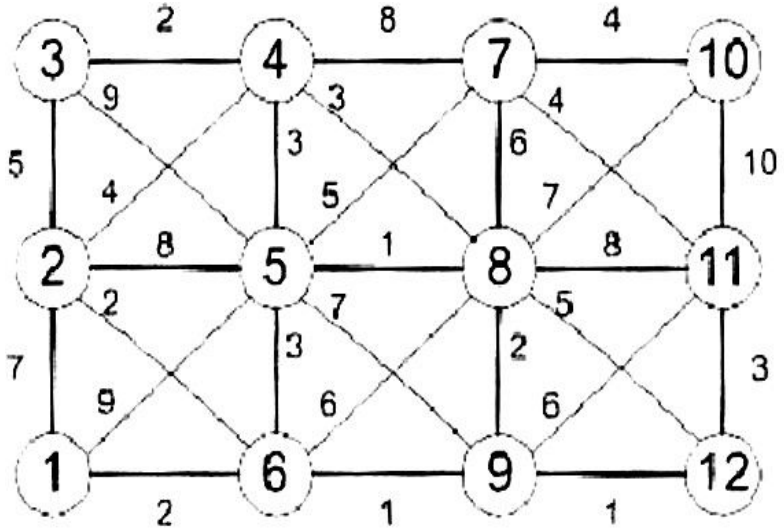
ПРИМЕР



Выполним шаг 1 и заполним первую строку таблицы.

вершины	1	2	3	4	5	6	7	8	9	10	11	12	
шаг 1	**0+**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	p = 1

ПРИМЕР



Из вершины 1 выходят дуги в вершины 2, 5, 6. Вычисляем метки этих вершин.

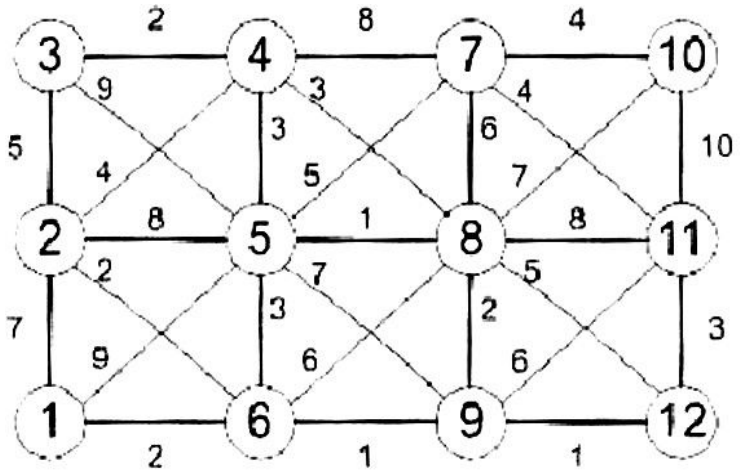
d(2) = min (∞; 0 + 7) = 7

d(5) = min (∞; 0 + 9) = 9

d(6) = min (∞; 0 + 2) = 2

вершины	1	2	3	4	5	6	7	8	9	10	11	12	
шаг 1	**0+**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	p = 1
шаг 2		7	∞	∞	9	**2+**	∞	∞	∞	∞	∞	∞	p = 6

ПРИМЕР - продолжение



Из найденных значений наименьшее – для вершины 6 (2). Помечаем эту вершину и снова пересчитываем метки вершин, в которые можно перейти из вершины 6:

d(2) = min (7; 2 + 2) = 4; d(5) = min (9; 2 + 3) = 5;

d(8) = min (∞; 2 + 6) = 8; d(9) = min (9; 2 + 1) = 3

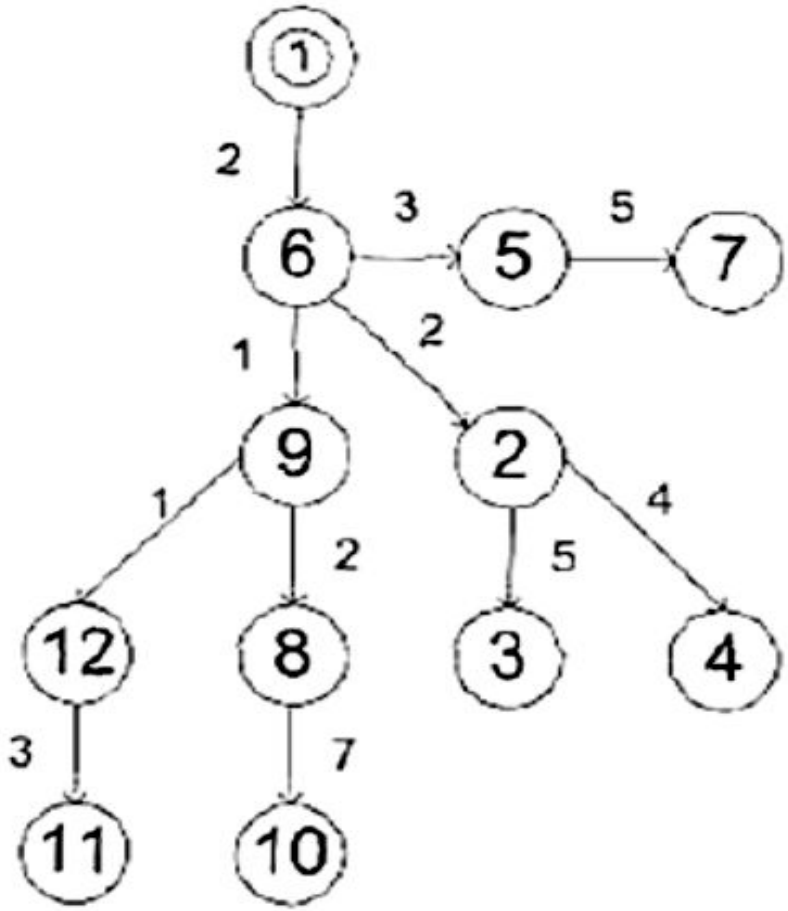
вершины	1	2	3	4	5	6	7	8	9	10	11	12	
шаг 1	**0+**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	p = 1
шаг 2		7	∞	∞	9	**2+**	∞	∞	∞	∞	∞	∞	p = 6
шаг 3		4	∞	∞	5		∞	8	**3+**	∞	∞	∞	p = 9

Метка вершины 9 становится постоянной. Пересчитываем метки вершин, в которые можно перейти из вершины 9. И так далее заполняем остальные строки таблицы.

вершины	1	2	3	4	5	6	7	8	9	10	11	12	
шаг 1	**0+**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	p = 1
шаг 2		7	∞	∞	9	**2+**	∞	∞	∞	∞	∞	∞	p = 6
шаг 3		4	∞	∞	5		∞	8	**3+**	∞	∞	∞	p = 9
шаг 4		**4+**	∞	∞	5		∞	5		∞	9	4	p = 2
шаг 5			9	8	5		∞	5		∞	9	**4+**	p = 12
шаг 6			9	8	**5+**		∞	5		∞	7		p = 5
шаг 7			9	8			10	**5+**		∞	7		p = 8
шаг 8			9	8			10			12	**7+**		p = 11
шаг 9			9	**8+**			10			12			p = 4
шаг 10			**9+**				10			12			p = 3
шаг 11							**10+**			12			p = 7
шаг 12										**12+**			p = 10

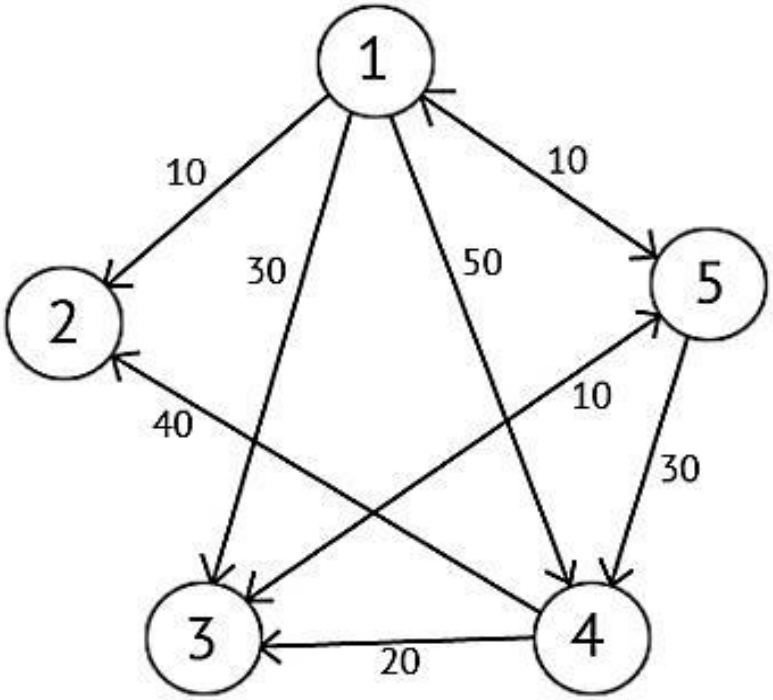
ДЕРЕВО ПУТЕЙ

Дерево кратчайших путей – это ориентированное дерево с корнем в вершине S. Все пути в этом дереве – кратчайшие для данного графа.



Строится по таблице, в него включаются вершины в том порядке, в котором они получали постоянные метки

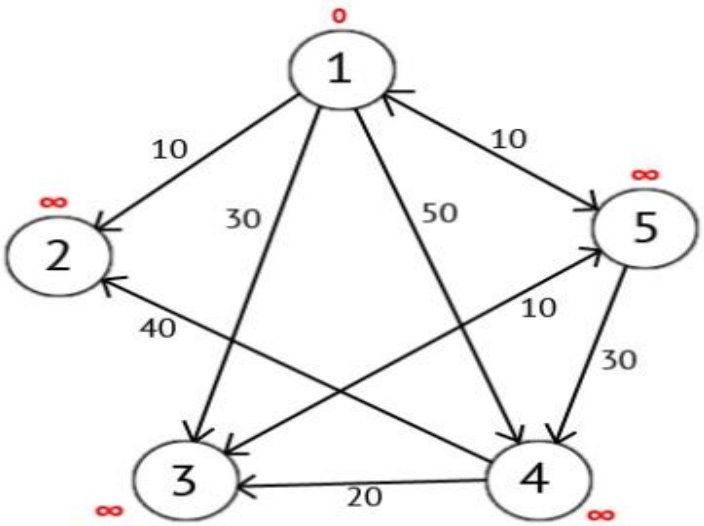
ПРИМЕР 2



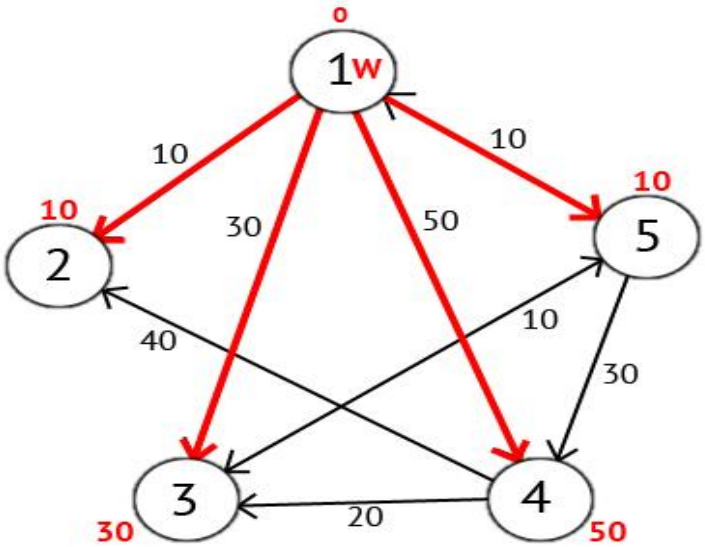
	1	2	3	4	5
1		10	30	50	10
2					
3					10
4		40	20		
5	10		10	30	

Возьмем в качестве источника вершину 1. Это значит что мы будем искать кратчайшие маршруты из вершины 1 в вершины 2, 3, 4 и 5.

ПРИМЕР

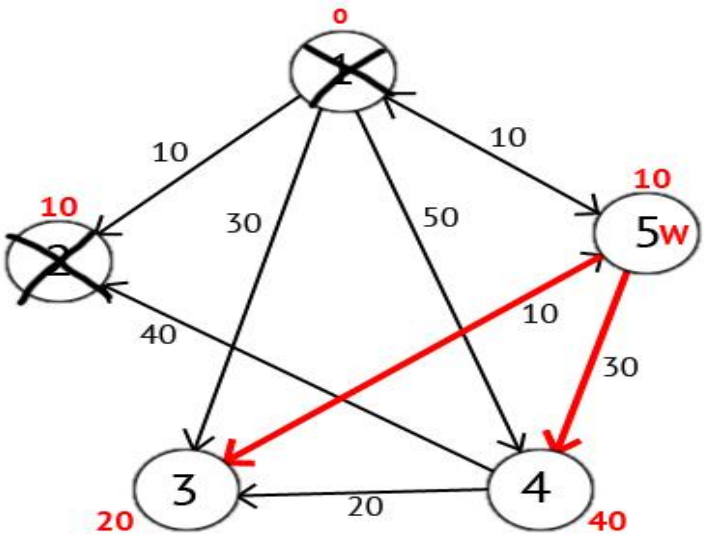


Присвоим 1-й вершине метку равную 0, потому как эта вершина — источник. Остальным вершинам присвоим метки равные бесконечности.

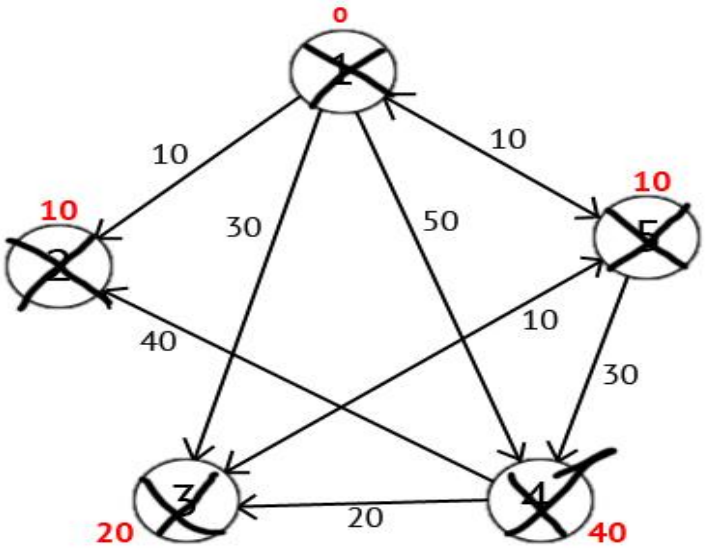


Рассмотрим все вершины в которые из вершины 1 есть путь, не содержащий вершин посредников

ПРИМЕР



выбираем из ещё не посещенных такую, которая имеет минимальное значение метки. В данном случае это вершина 2 или 5. Но из 2 нет ни одного исходящего пути, поэтому мы сразу отмечаем эту вершину как посещенную



Вершина 5 имеет минимальную метку. Рассмотрим все вершины в которые есть прямые пути из 5, но которые ещё не помечены как посещенные

ВЕКТОР МАРШРУТОВ

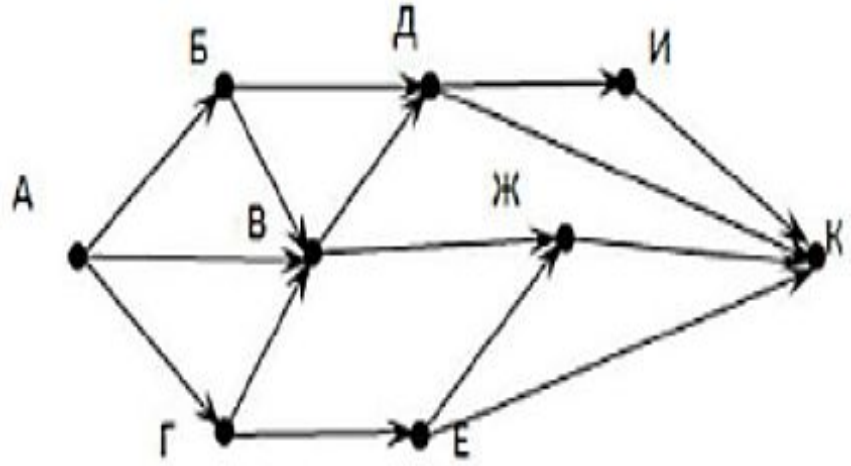
По количеству элементов этот вектор равен количеству вершин в графе, Каждый элемент содержит последнюю промежуточную вершину на кратчайшем пути между вершиной-источником и конечной вершиной.

В начале алгоритма все элементы вектора P равны вершине источнику (в нашем случае P = {1, 1, 1, 1, 1})

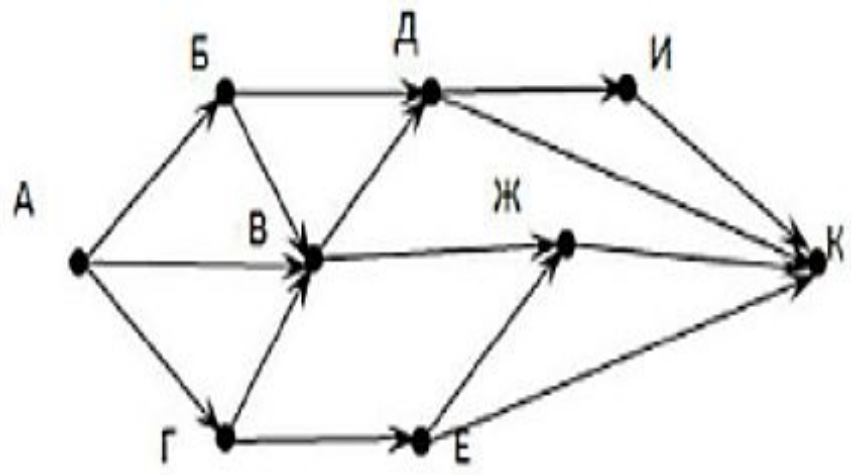
Далее на этапе пересчета значения метки для рассматриваемой вершины: если метка рассматриваемой вершины меняется на меньшую, в массив P мы записываем значение текущей вершины W. Например: у вершины 3 была метка со значением «30», при W=1. Далее при W=5, метка 3-ей вершины изменилась на «20», следовательно мы запишем значение в вектор P — P[3]=5

В результате получим вектор **P = {1, 1, 5, 5, 1}**

НАХОЖДЕНИЕ КОЛИЧЕСТВА ПУТЕЙ В ГРАФЕ



Шаг 1. Решение начинаем из конечного пункта. Для каждой вершины отмечаем, из каких вершин можно ее достичь.
Шаг 2 (обратный ход). Подставляем значения количества путей: $N_R = N_X + N_Y + N_Z$



вершина	откуда
К	ИДЖЕ
И	Д
Ж	ВЕ
Е	Г
Д	БВ
Г	А
В	АБГ
Б	А

вершина	откуда	
К	**ИДЖЕ**	**13**
И	Д	**4**
Ж	ВЕ	**4**
Е	Г	**1**
Д	БВ	**4**
Г	А	1
В	АБГ	3
Б	А	1

$N_Б=1$; $N_Г=1$;

$N_В=N_А+N_В+N_Г = 1+1+1 = 3$;

$N_Д=N_Б+N_В = 1+3 = 4$;

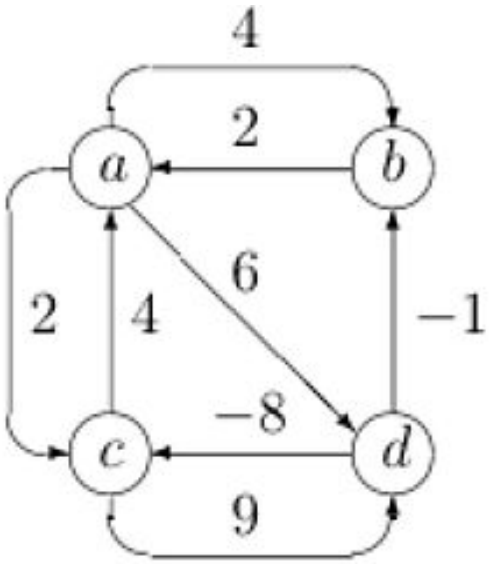
$N_Е=N_Г = 1$;

$N_Ж=N_В+N_Е = 3+1 = 4$;

$N_И=N_Д = 4$;

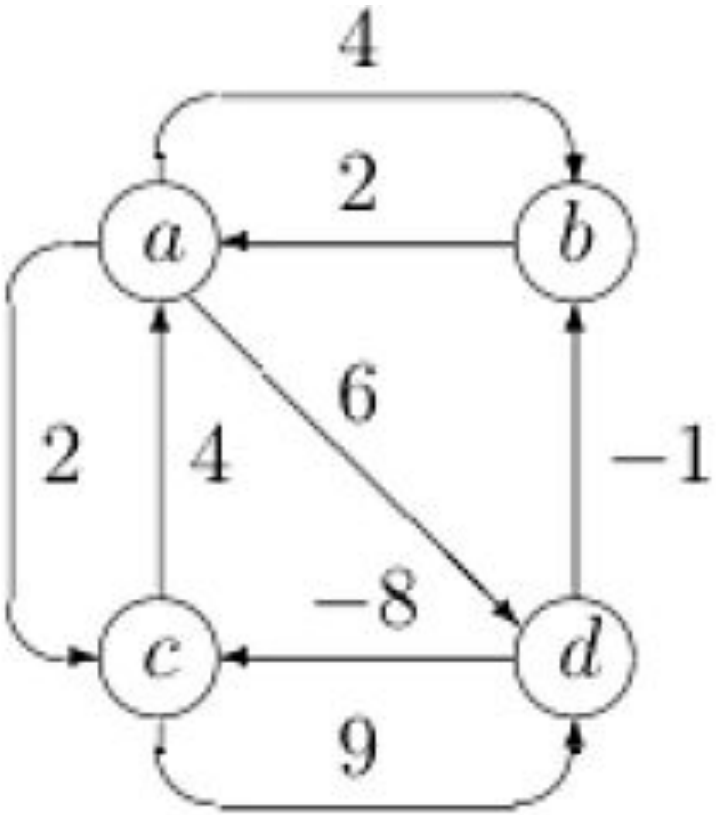
$N_К=N_И+N_Д+N_Ж+N_Е = 4+4+4+1 = 13$

АЛГОРИТМ БЕЛЛМАНА-ФОРДА



Алгоритм применяется при наличии дуг ОТРИЦАТЕЛЬНОЙ длины

1. Начало – аналогично алгоритму Дийкстры.
2. На шаге 2 пересчет величин d(x) производится для всех вершин, а не только для непомеченных
3. Если для некоторой помеченной вершины происходит уменьшение величины d(x), то с этой вершины и пометка снимается



вершины	a	b	c	d	
шаг 0	0+	∞	∞	∞	p = a
шаг 1	0+	4	2+	6	p = c
шаг 2	0+	4+	2+	6	p = b
шаг 3	0+	4+	2+	6+	p = d
шаг 4	0+	4+	-2+	6+	

Источники информации

- Программирование, компьютеры и сети https://progr-system.ru/

Благодарю за внимание!