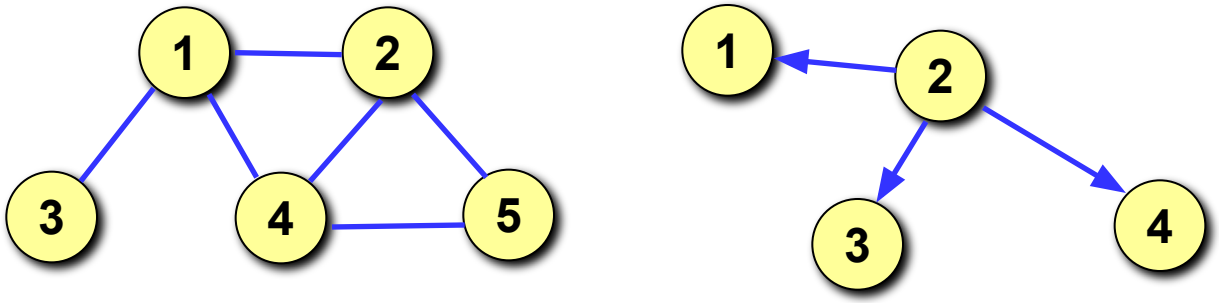
Определения

Граф – это набор вершин (узлов) и соединяющих их ребер (дуг).



Направленный граф (ориентированный, орграф) – это граф, в котором все дуги имеют направления.

Цепь – это последовательность ребер, соединяющих две вершины (в орграфе – **путь**).

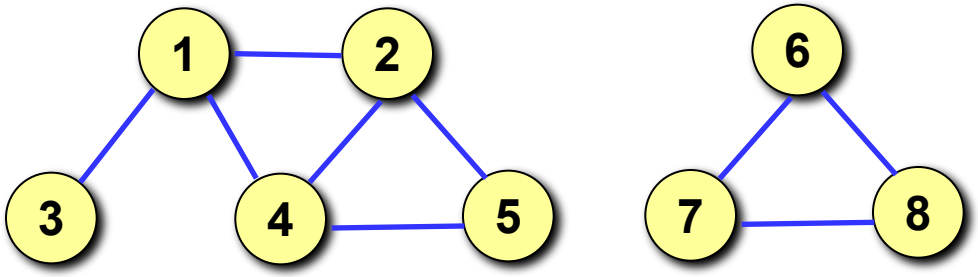
Цикл – это цепь из какой-то вершины в нее саму.

Взвешенный граф (сеть) – это граф, в котором каждому ребру приписывается вес (длина).

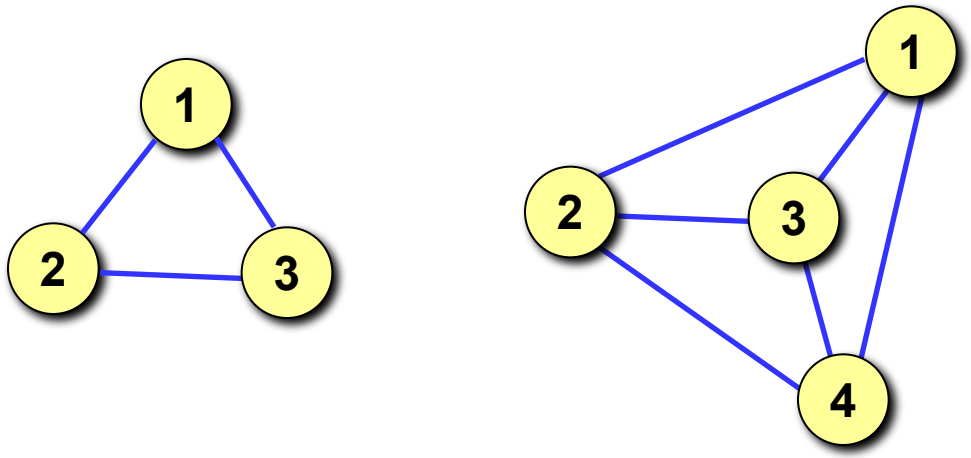
Определения

Связный граф – это граф, в котором существует цепь между каждой парой вершин.

k-связный граф – это граф, который можно разбить на **k** связных частей.

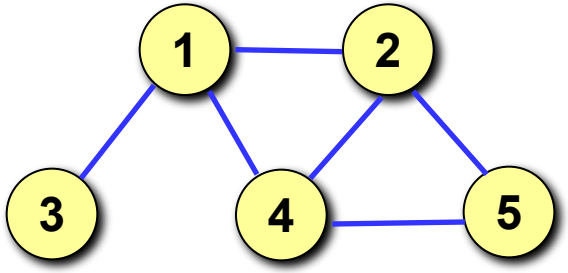


Полный граф – это граф, в котором проведены все возможные ребра (**n** вершин → **n(n-1)/2** ребер).



Описание графа

Матрица смежности – это матрица, элемент `M[i][j]` которой равен `1`, если существует ребро из вершины `i` в вершину `j`, и равен `0`, если такого ребра нет.

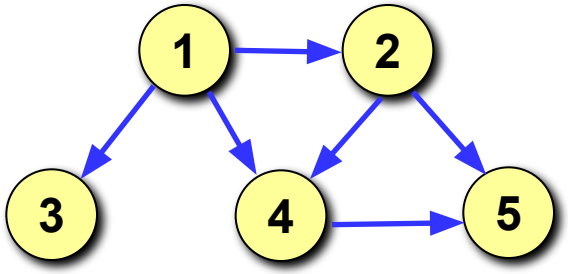


	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	1
3	1	0	0	0	0
4	1	1	0	0	1
5	0	1	0	1	0

Список смежности

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

Симметрия!

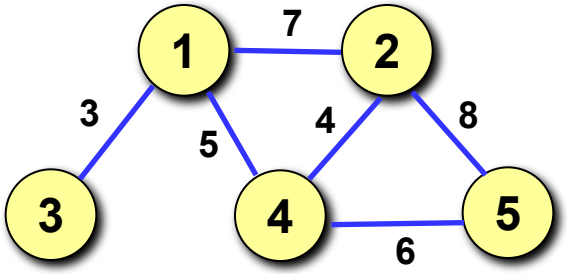


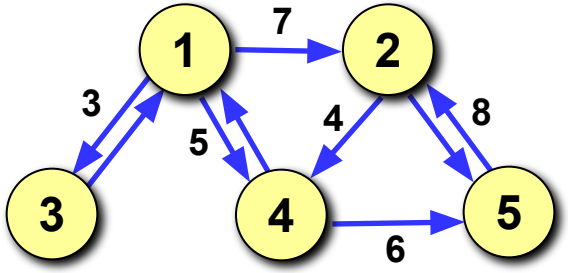
	1	2	3	4	5
1	0	1	1	1	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	1
5	0	0	0	0	0

1	2	3	4		
2	4	5			
3					
4	5				
5					

Весовая матрица

Весовая матрица – это матрица, элемент **`W[i,j]`** которой равен весу ребра из вершины **`i`** в вершину **`j`** (если оно есть), или равен ∞, если такого ребра нет.





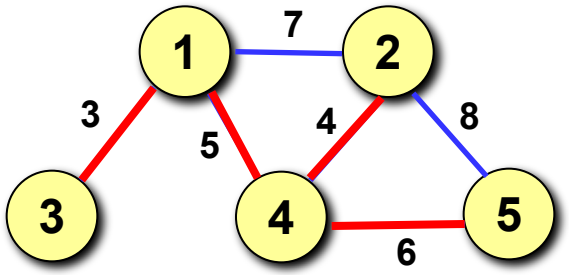
	1	2	3	4	5
1	0	7	3	5	∞
2	7	0	∞	4	8
3	3	∞	0	∞	∞
4	5	4	∞	0	6
5	∞	8	∞	6	0

	1	2	3	4	5
1	0	7	3	5	∞
2	∞	0	∞	4	8
3	3	∞	0	∞	∞
4	5	∞	∞	0	6
5	∞	8	∞	∞	0

Задача Прима-Краскала

Задача: соединить **N** городов телефонной сетью так, чтобы длина телефонных линий была минимальная.

Та же задача: дан связный граф с **N** вершинами, веса ребер заданы весовой матрицей **W**. Нужно найти набор ребер, соединяющий все вершины графа (***остовное дерево***) и имеющий наименьший вес.



	1	2	3	4	5
1	0	7	3	5	∞
2	7	0	∞	4	8
3	3	∞	0	∞	∞
4	5	4	∞	0	6
5	∞	8	∞	6	0

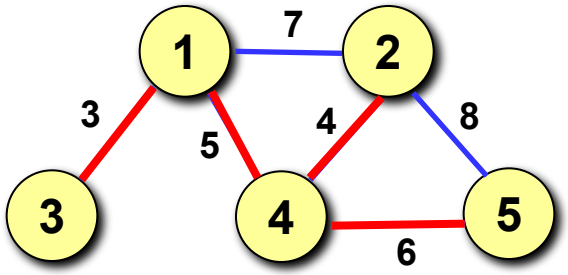
Жадный алгоритм

Жадный алгоритм – это многошаговый алгоритм, в котором на каждом шаге принимается решение, лучшее в данный момент.



В целом может получиться не оптимальное решение (последовательность шагов)!

Шаг в задаче Прима-Краскала – это выбор еще невыбранного ребра и добавление его к решению.

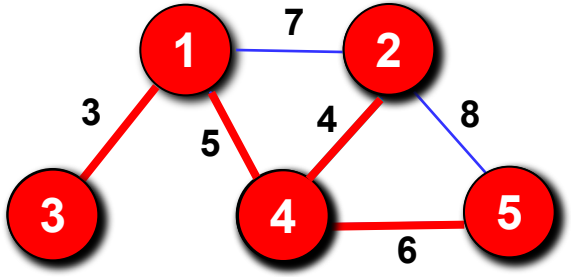


В задаче Прима-Краскала жадный алгоритм дает оптимальное решение!

Реализация алгоритма Прима-Краскала

Проблема: как проверить, что
1) ребро не выбрано, и
2) ребро не образует цикла с выбранными ребрами.

Решение: присвоить каждой вершине свой цвет и перекрашивать вершины при добавлении ребра.



Алгоритм:

1) покрасить все вершины в разные цвета;
2) сделать `N-1` раз в цикле:
   - выбрать ребро `(i,j)` минимальной длины из всех ребер, соединяющих вершины разного цвета;
   - перекрасить все вершины, имеющие цвет `j`, в цвет `i`.
3) вывести найденные ребра.

Реализация алгоритма Прима-Краскала

Структура «ребро»:

```
type rebro = record
       i, j: integer; { номера вершин }
     end;
```

Основная программа:

весовая матрица

цвета вершин

```
const N = 5;
var W: array[1..N,1..N] of integer;
    Color: array[1..N] of integer;
    i, j, k, min, col_i, col_j: integer;
    Reb: array[1..N-1] of rebro;
begin
...     { здесь надо ввести матрицу W }
for i:=1 to N do { раскрасить в разные цвета }
  Color[i] := i;
... { основной алгоритм – заполнение массива Reb }
... { вывести найденные ребра (массив Reb)}
end.
```

Реализация алгоритма Прима-Краскала

Основной алгоритм:

```
for k:=1 to N-1 do begin
  min := MaxInt;
  for i:=1 to N do
    for j:=i+1 to N do
     if (Color[i] <> Color[j]) and
        (W[i,j] < min) then begin
       min := W[i,j];
    Reb[k].i := i;
       Reb[k].j := j;
       col_i := Color[i];
       col_j := Color[j];
     end;
  for i:=1 to N do
    if Color[i] = col_j then
       Color[i] := col_i;
end;
```

нужно выбрать всего N-1 ребер

цикл по всем парам вершин

учитываем только пары с разным цветом вершин

запоминаем ребро и цвета вершин

перекрашиваем вершины цвета col_j

Сложность алгоритма

Основной цикл:

```
for k:=1 to N-1 do begin
  ...
  for i:=1 to N do
    for j:=i+1 to N do
    ...
end;
```

три вложенных цикла, в каждом число шагов <=N

Количество операций:

$O(N^3)$ растет не быстрее, чем N^3

Требуемая память:

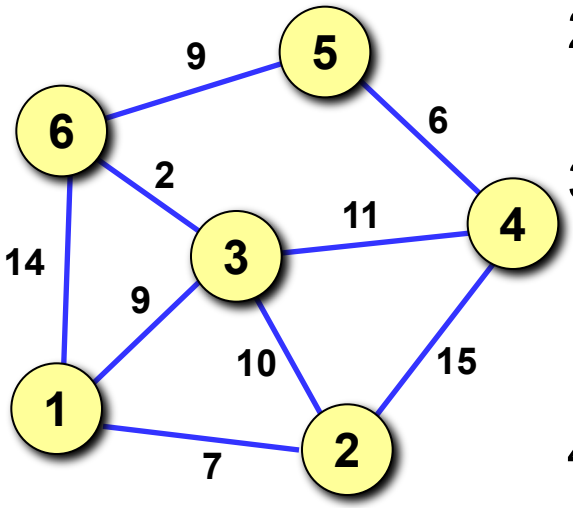
```
var W: array[1..N,1..N] of integer;
    Color: array[1..N] of integer;
    Reb: array[1..N-1] of rebro;
```

➡ $O(N^2)$

Кратчайшие пути (алгоритм Дейкстры)

Задача: задана сеть дорог между городами, часть которых могут иметь одностороннее движение. Найти кратчайшие расстояния от заданного города до всех остальных городов.

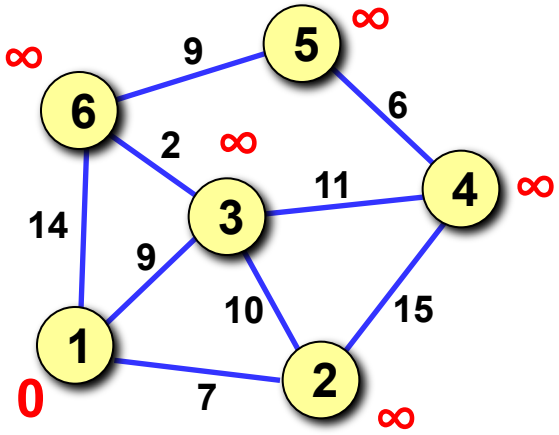
Та же задача: дан связный граф с **N** вершинами, веса ребер заданы матрицей **W**. Найти кратчайшие расстояния от заданной вершины до всех остальных.

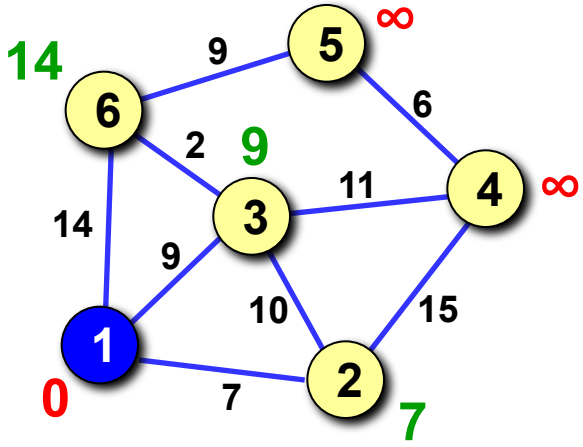


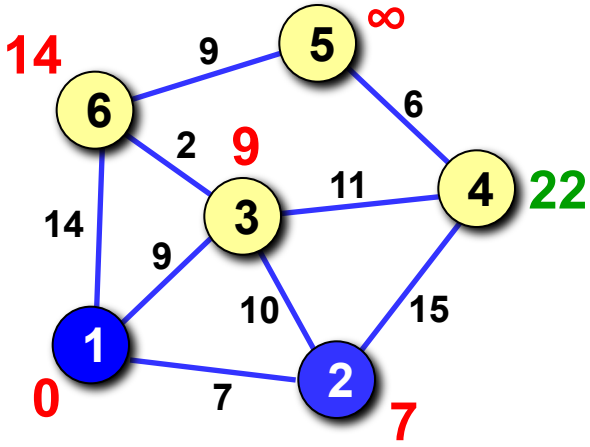
Алгоритм Дейкстры (*E.W. Dijkstra*, 1959)

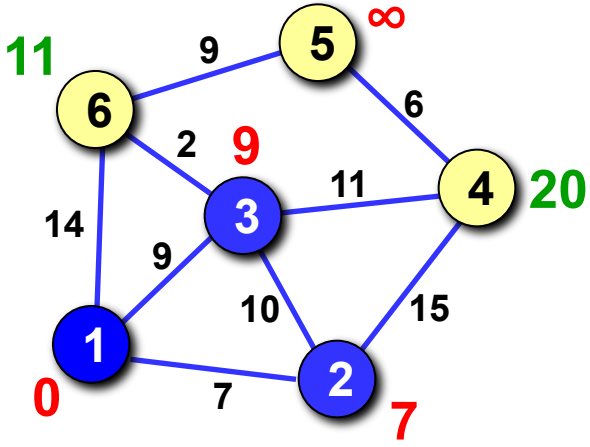
1) присвоить всем вершинам метку ∞;
2) среди нерассмотренных вершин найти вершину **j** с наименьшей меткой;
3) для каждой необработанной вершины **i**: если путь к вершине **i** через вершину **j** меньше существующей метки, заменить метку на новое расстояние;
4) если остались необработанны вершины, перейти к шагу 2;
5) метка = минимальное расстояние.

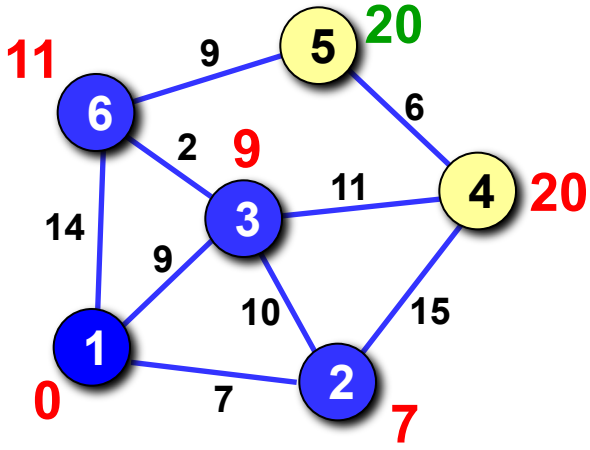
Алгоритм Дейкстры

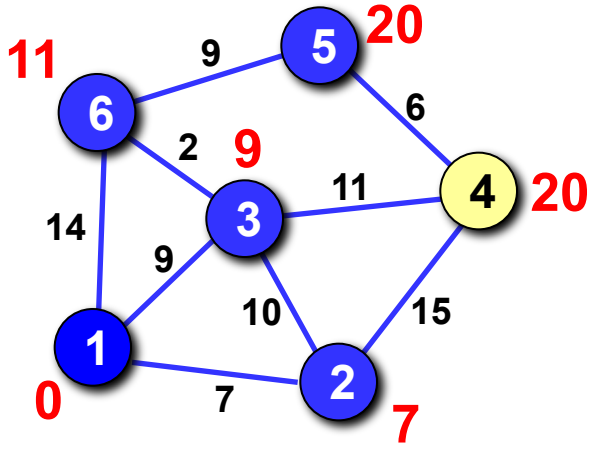












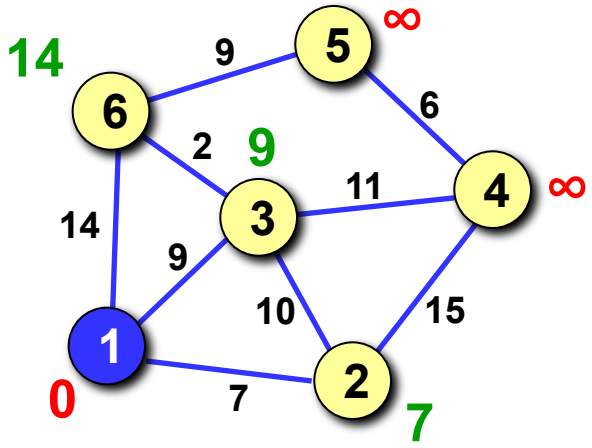
Реализация алгоритма Дейкстры

Массивы:

1) массив **a**, такой что **a[i]=1**, если вершина уже рассмотрена, и **a[i]=0**, если нет.
2) массив **b**, такой что **b[i]** – длина текущего кратчайшего пути из заданной вершины **x** в вершину **i**;
3) массив **c**, такой что **c[i]** – номер вершины, из которой нужно идти в вершину **i** в текущем кратчайшем пути.

Инициализация:

4) заполнить массив **a** нулями (вершины не обработаны);
5) записать в **b[i]** значение **W[x][i]**;
6) заполнить массив **c** значением **x**;
7) записать **a[x]=1**.



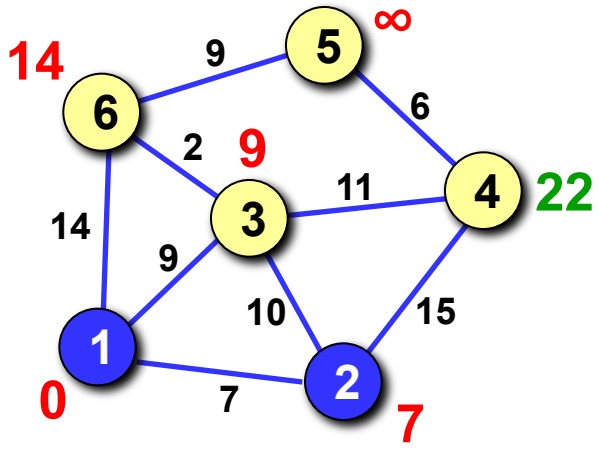
	1	2	3	4	5	6
a	1	0	0	0	0	0
b	0	7	9	∞	∞	14
c	0	0	0	0	0	0

Реализация алгоритма Дейкстры

Основной цикл:

1) если все вершины рассмотрены, то стоп.
2) среди всех нерассмотренных вершин (`a[i]=0`) найти вершину `j`, для которой `b[i]` – минимальное;
3) записать `a[j]:=1`;
4) для всех вершин `k`: если путь в вершину `k` через вершину `j` короче, чем найденный ранее кратчайший путь, запомнить его: записать `b[k]:=b[j]+W[j,k]` и `c[k]=j`.

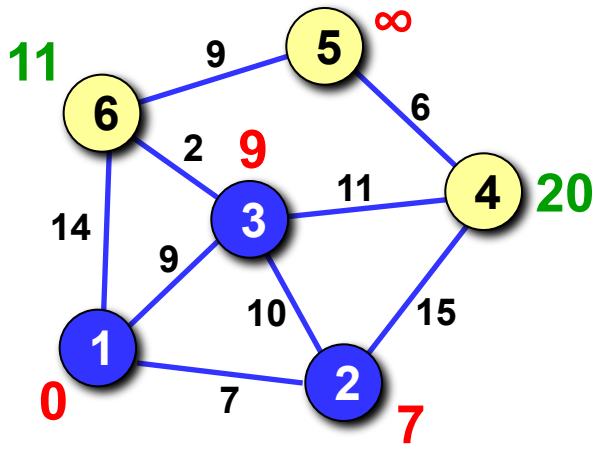
Шаг 1:



	1	2	3	4	5	6
a	1	1	0	0	0	0
b	0	7	9	22	∞	14
c	0	0	0	1	0	0

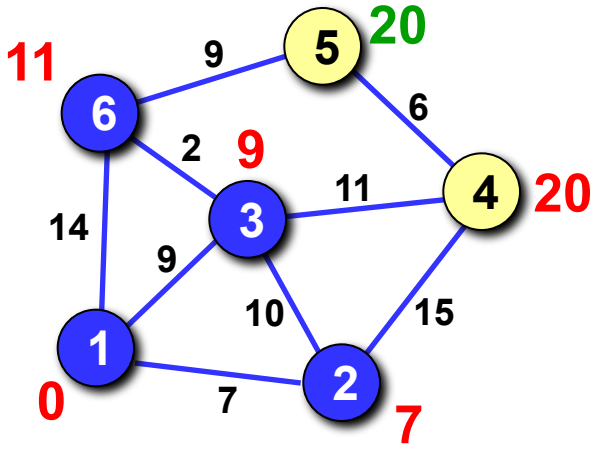
Реализация алгоритма Дейкстры

Шаг 2:



	1	2	3	4	5	6
a	1	1	1	0	0	0
b	0	7	9	20	∞	11
c	0	0	0	2	0	2

Шаг 3:



	1	4	3	4	5	6
a	1	1	1	0	0	1
b	0	7	9	20	20	11
c	0	0	0	2	5	2

Дальше массивы не изменяются!

Как вывести маршрут?

Результат работа алгоритма Дейкстры:

	1	2	3	4	5	6
a	1	1	1	1	1	1
b	0	7	9	20	20	11
c	0	0	0	2	5	2

длины путей

Маршрут из вершины 0 в вершину 4:

4 ➡ 5 ➡ 2 ➡ 0

Вывод маршрута в вершину `i` (использование массива `c`):

1) установить `z:=i`;
2) пока `c[i]<>x` присвоить `z:=c[z]` и вывести `z`.

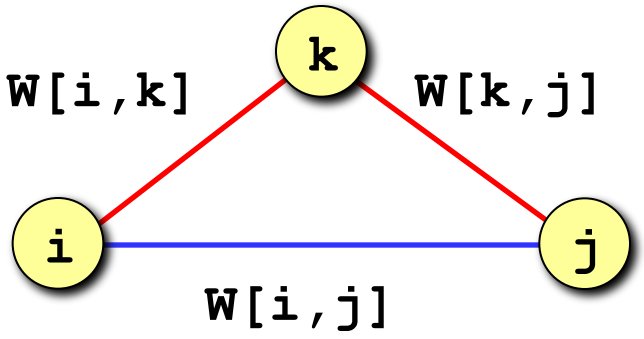
Сложность алгоритма Дейкстры:

два вложенных цикла по `N` шагов $O(N^2)$

Алгоритм Флойда-Уоршелла

Задача: задана сеть дорог между городами, часть которых могут иметь одностороннее движение. Найти **все кратчайшие расстояния**, от каждого города до всех остальных городов.

```
for k:=1 to N
  for i:=1 to N
    for j:=1 to N
      if W[i,j] > W[i,k]+W[k,j] then
          W[i,j]:=W[i,k]+W[k,j];
```



Если из вершины **i** в вершину **j** короче ехать через вершину **k**, мы едем через вершину **k**!

Нет информации о маршруте, только кратчайшие расстояния!

Алгоритм Флойда-Уоршелла

Версия с запоминанием маршрута:

```
for i:=1 to N
  for j :=1 to N
    c[i,j] :=i;
...
for k: =1 to N
  for i: =1 to N
    for j: =1 to N
      if W[i,j] > W[i,k] + W[k,j] then begin
        W[i,j] := W[i,k] + W[k,j];
        c[i,j] := c[k,j];
      end;
```

i–ая строка строится так же, как массив **c** в алгоритме Дейкстры

в конце цикла **c[i,j]** – предпоследняя вершина в кратчайшем маршруте из вершины **i** в вершину **j**

Какова сложность алгоритма?

$O(N^3)$

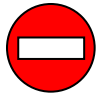
Задача коммивояжера

Задача коммивояжера. Коммивояжер (бродячий торговец) должен выйти из первого города и, посетив по разу в неизвестном порядке города **2,3,...N**, вернуться обратно в первый город. В каком порядке надо обходить города, чтобы замкнутый путь (тур) коммивояжера был кратчайшим?

Это NP-полная задача, которая строго решается только перебором вариантов (пока)!

Точные методы:

1) простой перебор;
2) метод ветвей и границ;
3) метод Литтла;
4) …



большое время счета для больших **N**

O(N!)

Приближенные методы:

5) метод случайных перестановок (*Matlab*);
6) генетические алгоритмы;
7) метод муравьиных колоний;
8) …

не гарантируется оптимальное решение

Другие классические задачи

Задача на минимум суммы. Имеется **N** населенных пунктов, в каждом из которых живет p_i школьников (**i=1,...,N**). Надо разместить школу в одном из них так, чтобы общее расстояние, проходимое всеми учениками по дороге в школу, было минимальным.

Задача о наибольшем потоке. Есть система труб, которые имеют соединения в **N** узлах. Один узел **S** является источником, еще один – стоком **T**. Известны пропускные способности каждой трубы. Надо найти наибольший поток от источника к стоку.

Задача о наибольшем паросочетании. Есть **M** мужчин и **N** женщин. Каждый мужчина указывает несколько (от **0** до **N**) женщин, на которых он согласен жениться. Каждая женщина указывает несколько мужчин (от **0** до **M**), за которых она согласна выйти замуж. Требуется заключить наибольшее количество моногамных браков.