

Лекция 11

Поиск кратчайшего пути в графе

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

Курс «Структуры и алгоритмы обработки данных»

Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)

Весенний семестр, 2016

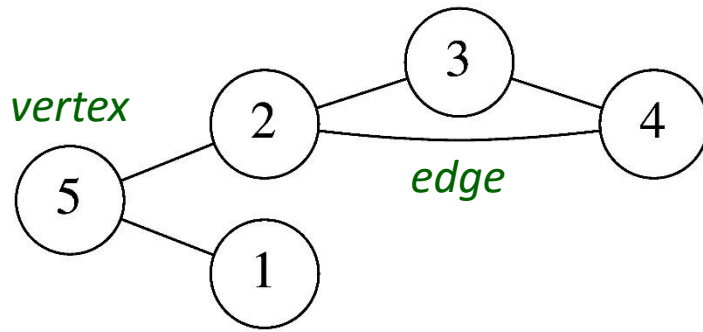
Понятие графа

- **Граф (graph)** – это совокупность непустого множества V вершин и множества E ребер

$$G = (V, E),$$

$$n = |V|, \quad m = |E|,$$

$$V = \{1, 2, \dots, n\}, \quad E = \{(u_1, v_1), (u_2, v_2), \dots, (u_m, v_m)\}$$



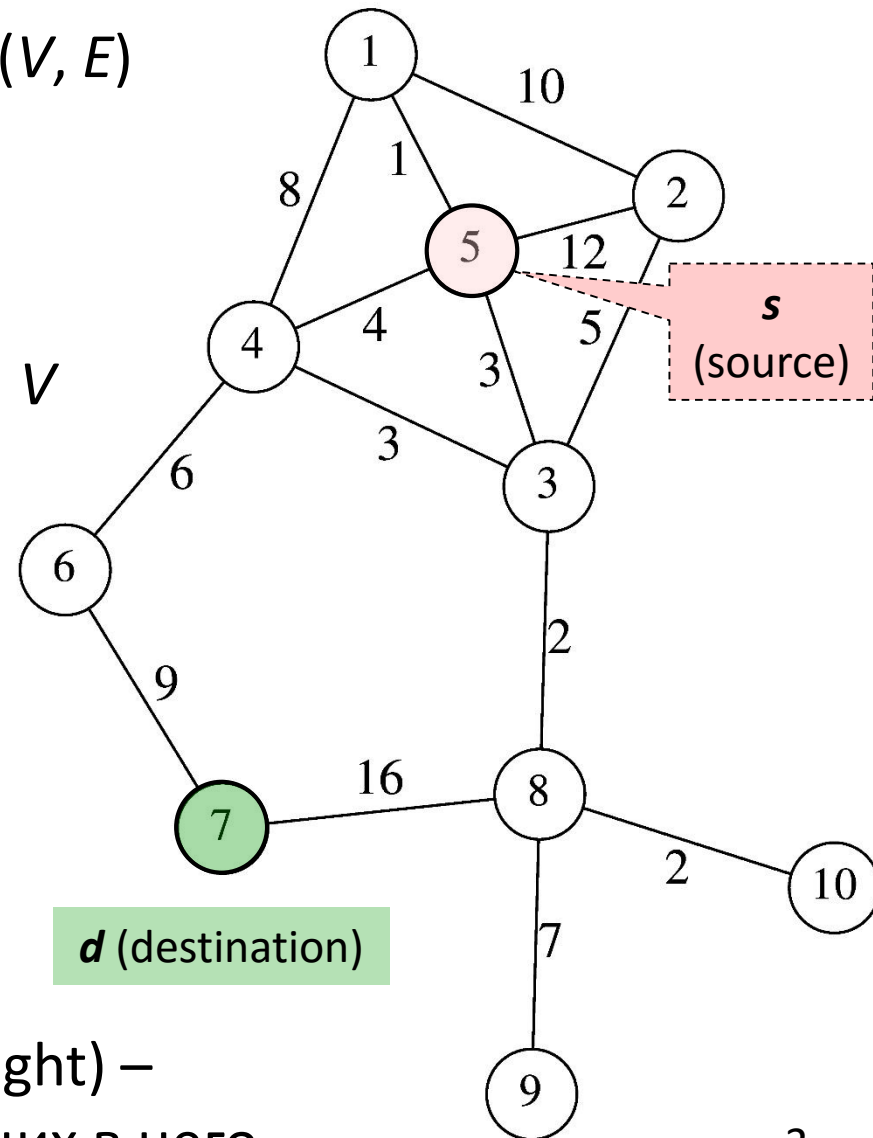
$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1, 5), (2, 5), (2, 3), (2, 4), (3, 4)\}$$

$(1, 5)$ и $(5, 1)$ – это одно и тоже ребро

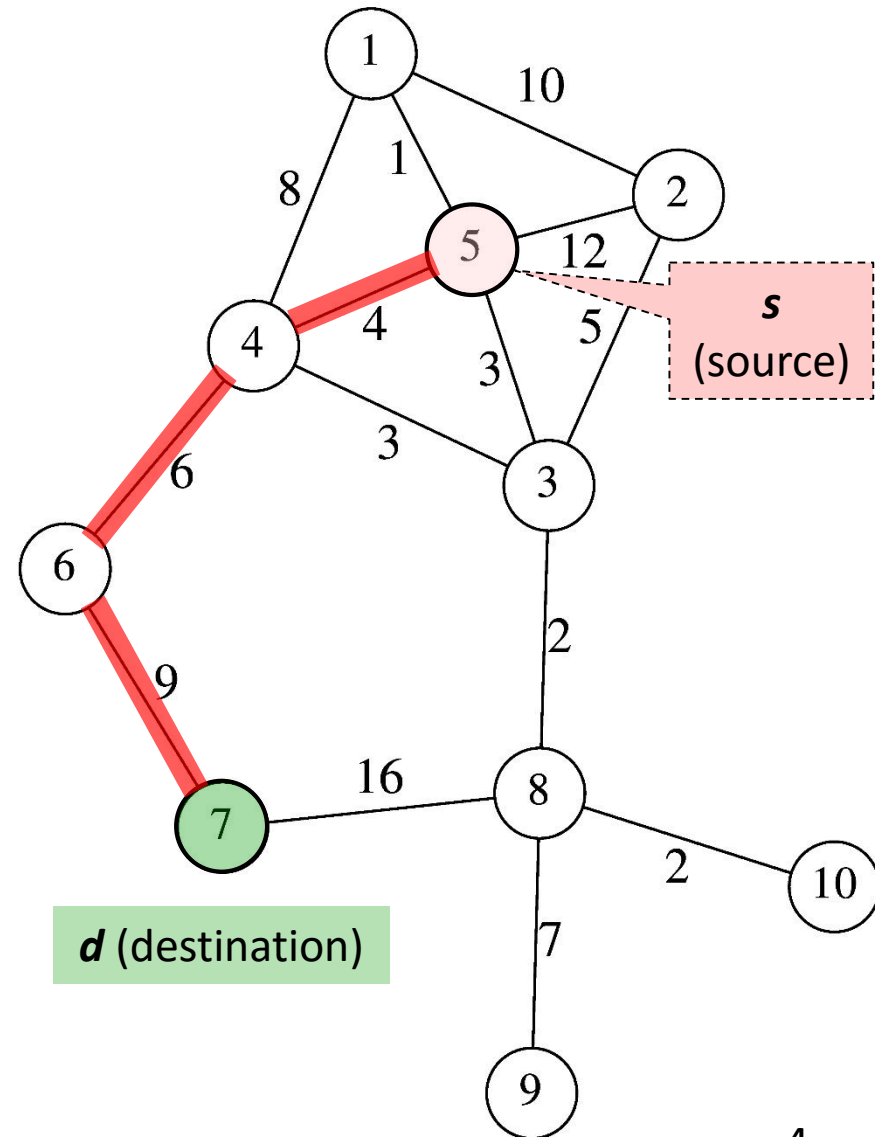
Поиск кратчайшего пути в графе

- Имеется взвешенный граф $G = (V, E)$
- Каждому ребру $(i, j) \in E$ назначен вес w_{ij}
- Заданы начальная вершина $s \in V$ и конечная $d \in V$
- Требуется найти кратчайший путь из вершины s в вершину d (shortest path problem)
- Длина пути (path length, path cost, path weight) – это сумма весов ребер, входящих в него



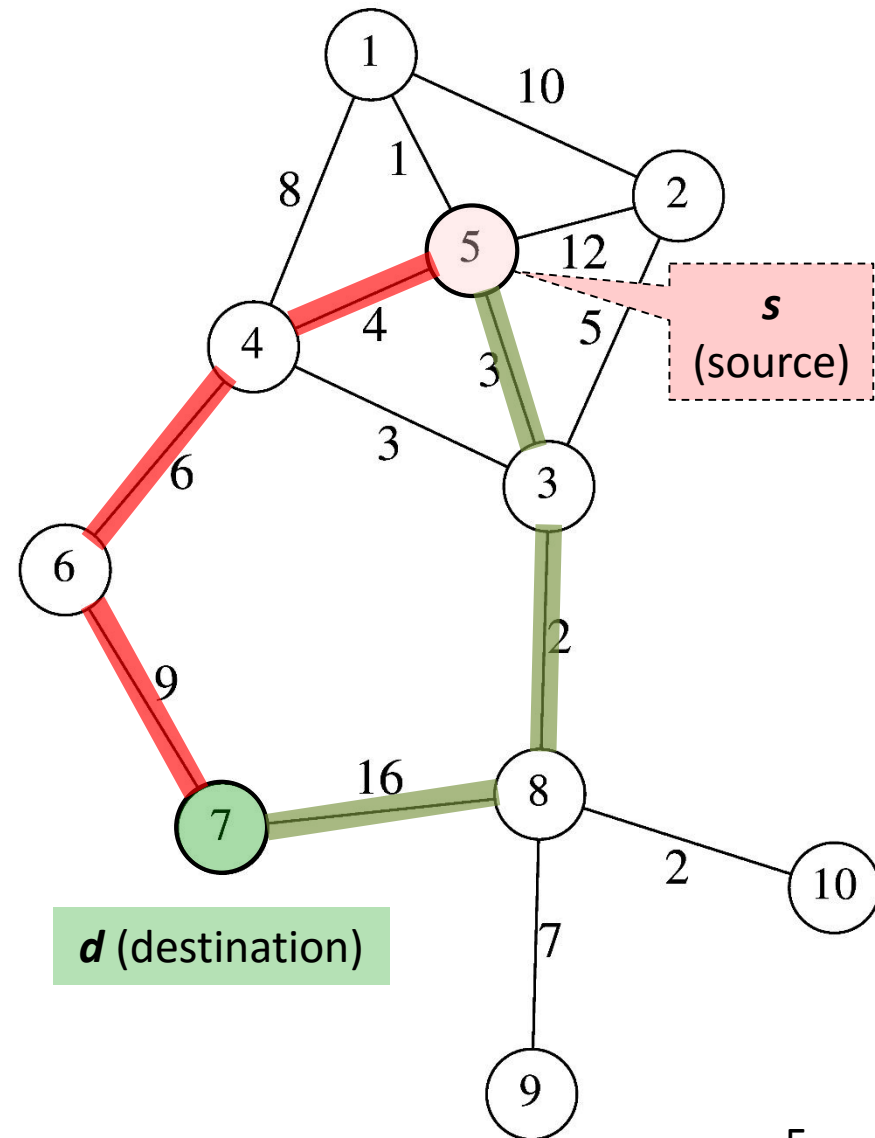
Длина пути в графе

- Длина пути (5, 4, 6, 7) =
 $w_{54} + w_{46} + w_{67} = 4 + 6 + 9 = 19$



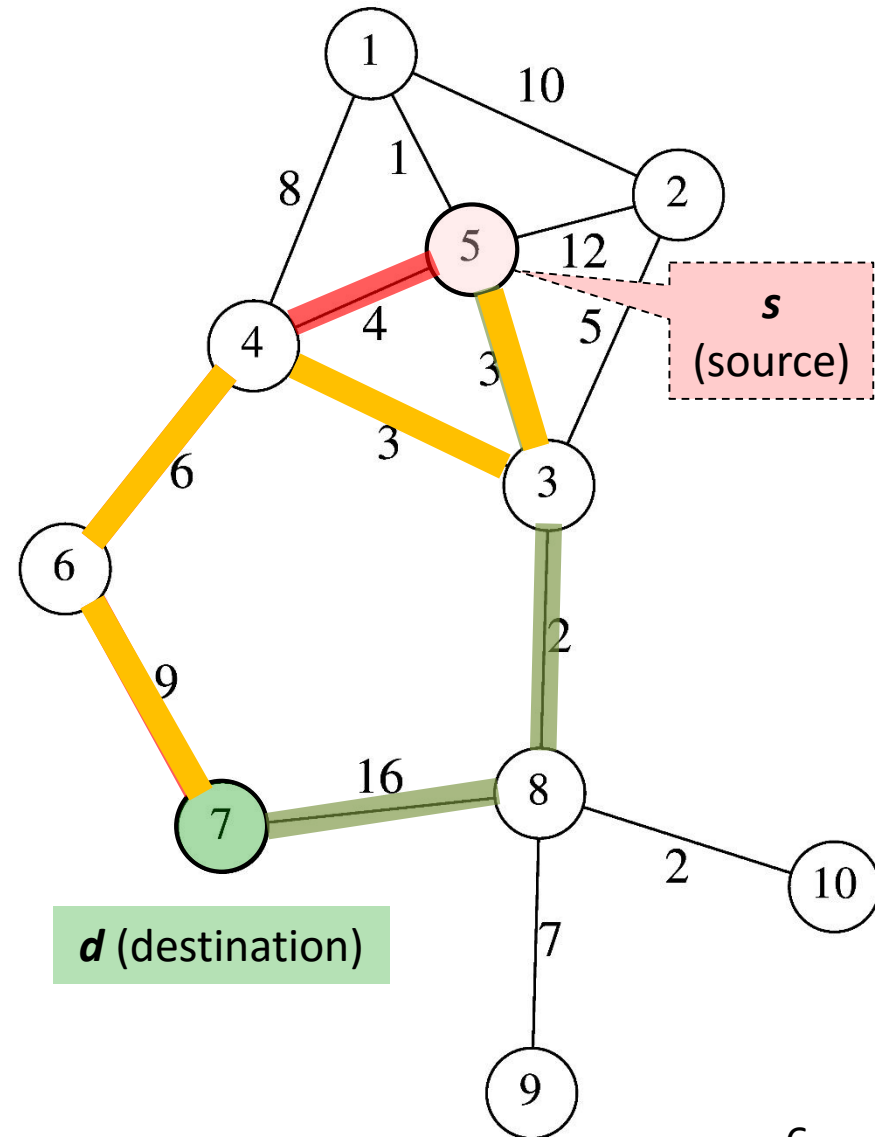
Длина пути в графе

- Длина пути (5, 4, 6, 7) =
 $w_{54} + w_{46} + w_{67} = 4 + 6 + 9 = 19$
- Длина пути (5, 3, 8, 7) =
 $3 + 2 + 16 = 21$



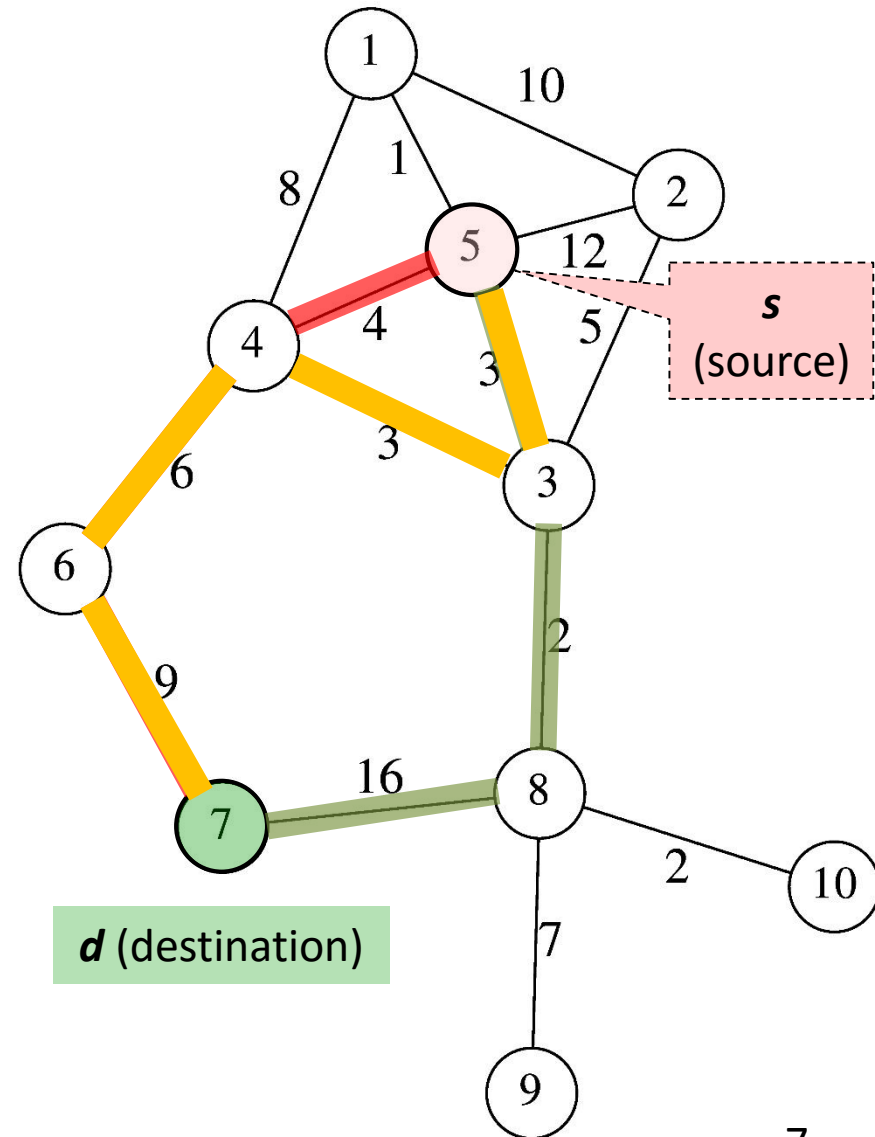
Длина пути в графе

- Длина пути (5, 4, 6, 7) =
 $w_{54} + w_{46} + w_{67} = 4 + 6 + 9 = 19$
- Длина пути (5, 3, 8, 7) =
 $3 + 2 + 16 = 21$
- Длина пути (5, 3, 4, 6, 7) =
 $3 + 3 + 6 + 9 = 21$



Длина пути в графе

- Длина пути (5, 4, 6, 7) =
 $w_{54} + w_{46} + w_{67} = 4 + 6 + 9 = 19$
- Длина пути (5, 3, 8, 7) =
 $3 + 2 + 16 = 21$
- Длина пути (5, 3, 4, 6, 7) =
 $3 + 3 + 6 + 9 = 21$
- **Альтернативные пути**
 - ❑ (5, 1, 4, 3, 8, 7)
 - ❑ (5, 2, 3, 8, 7)
 - ❑ ...



Постановки задачи о кратчайшем пути

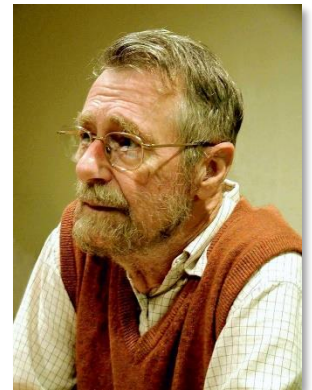
- **Задача о кратчайшем пути между парой вершин (single-pair shortest path problem)**
Требуется найти кратчайший путь из заданной вершины s в заданную вершину d
- **Задача о кратчайших путях из заданной вершины во все (single-source shortest path problem)**
Найти кратчайшие пути из заданной вершины s во все
- **Задача о кратчайшем пути в заданный пункт назначения (single-destination shortest path problem)**
Требуется найти кратчайшие пути в заданную вершину v из всех вершин графа
- **Задача о кратчайшем пути между всеми парами вершин (all-pairs shortest path problem)**
Требуется найти кратчайший путь из каждой вершины u в каждую вершину v

Алгоритмы поиска кратчайшего пути в графе

Алгоритм	Применение
Алгоритм Дейкстры	Находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса ($w_{ij} \geq 0$)
Алгоритм Беллмана-Форда	Находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным
Алгоритм поиска A* (A star)	Находит путь с наименьшей стоимостью от одной вершины к другой, используя алгоритм поиска по первому наилучшему совпадению на графе
Алгоритм Флойда-Уоршелла	Находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа
Алгоритм Джонсона	Находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа (должны отсутствовать циклы с отрицательным весом)
Алгоритм Ли (волновой алгоритм)	Находит путь между вершинами s и t графа, содержащий минимальное количество промежуточных вершин (трассировки электрических соединений на кристаллах микросхем и на печатных платах)
Алгоритмы Viterbi, Cherkassky, ...	

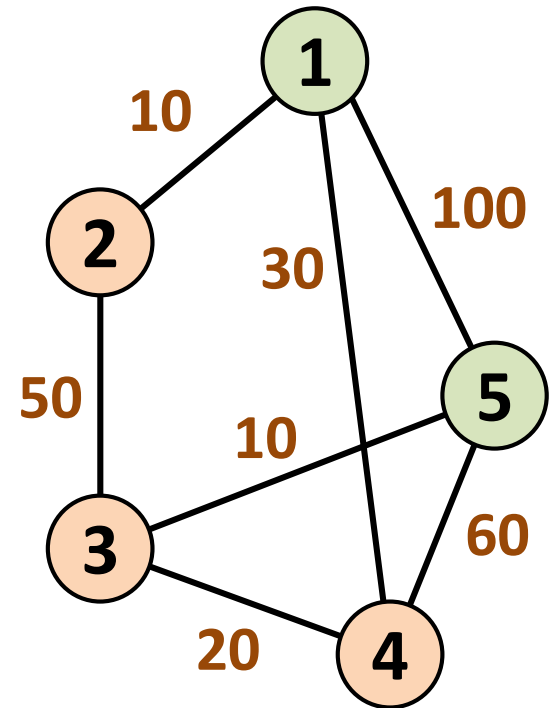
Алгоритм Дейкстры

- **Алгоритм Дейкстры** (Dijkstra's algorithm, 1959) – алгоритм поиска кратчайшего пути в графе из заданной вершины во все остальные (single-source shortest path problem)
- Находит кратчайшее *расстояние* от одной из вершин графа до всех остальных
- Применим только для графов без ребер отрицательного веса и петель ($w_{ij} \geq 0$)
- **Эдсгер Дейкстра** (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. — М.: Мир, 1978. — С. 275.
 2. Дал У., Дейкстра Э., Хоор К. Структурное программирование = Structured Programming. — М.: Мир, 1975. — С. 247.



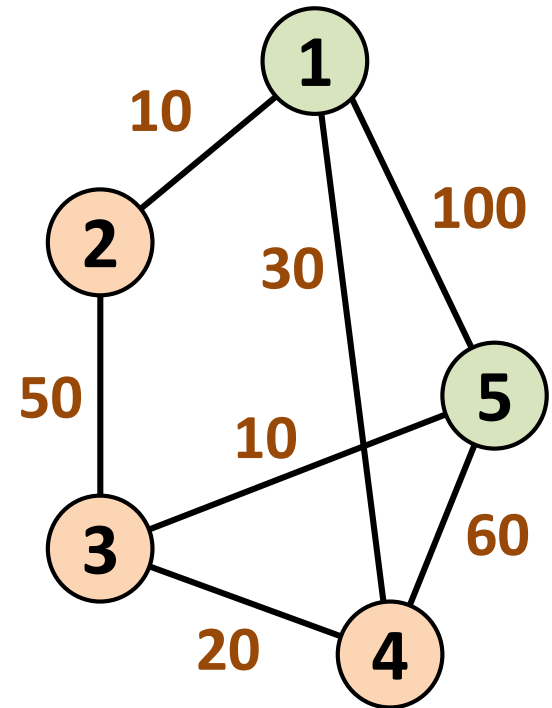
Алгоритм Дейкстры (Dijkstra)

- **Пример:** найти кратчайший путь из вершины 1 в вершину 5
- Введем обозначения:
 - H – множество посещенных вершин
 - $D[i]$ – текущее известное кратчайшее расстояние от вершины s до вершины i
 - $prev[i]$ – номер вершины, предшествующей i в пути



Алгоритм Дейкстры (Dijkstra)

1. Устанавливаем расстояние $D[i]$ от начальной вершины s до всех остальных в ∞
2. Полагаем $D[s] = 0$
3. Помещаем все вершины в очередь с приоритетом Q (min-heap): приоритет вершины i это значение $D[i]$



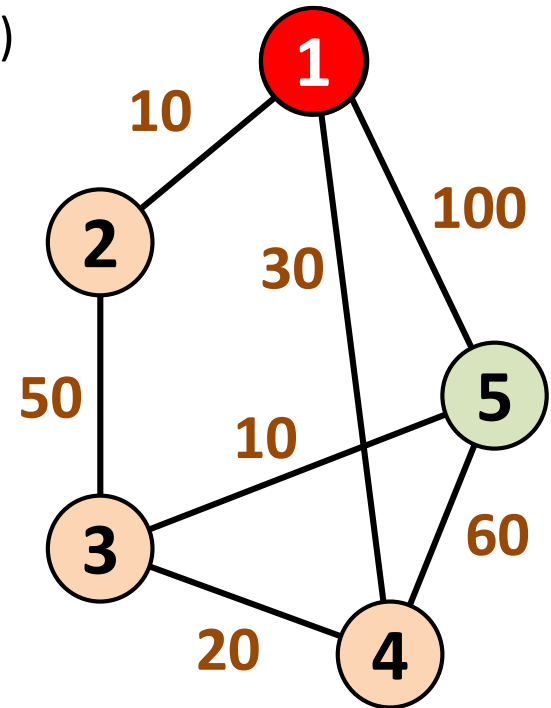
	1	2	3	4	5
$D[i]$	0	∞	∞	∞	∞

Алгоритм Дейкстры (Dijkstra)

4. Запускаем цикл из n итераций (по числу вершин)

1. Извлекаем из очереди Q вершину v с минимальным приоритетом – ближайшую к s вершину
2. Отмечаем вершину v как посещенную (помещаем v во множество H)
3. Возможно пути из s через вершину v стали короче, выполняем проверку: для каждой вершины u смежной с вершиной v и не включенной в H проверяем и корректируем расстояние $D[u]$

```
if  $D[v] + w(v, u) < D[u]$  then  
    // Путь из  $s$  до  $u$  через  $(v, u)$  короче  
     $D[u] = D[v] + w(v, u)$   
    PriorityQueueDecrease( $Q, u, D[u]$ )  
     $prev[u] = v$   
end if
```



$D[2] = 10$

$D[4] = 30$

$D[5] = 100$

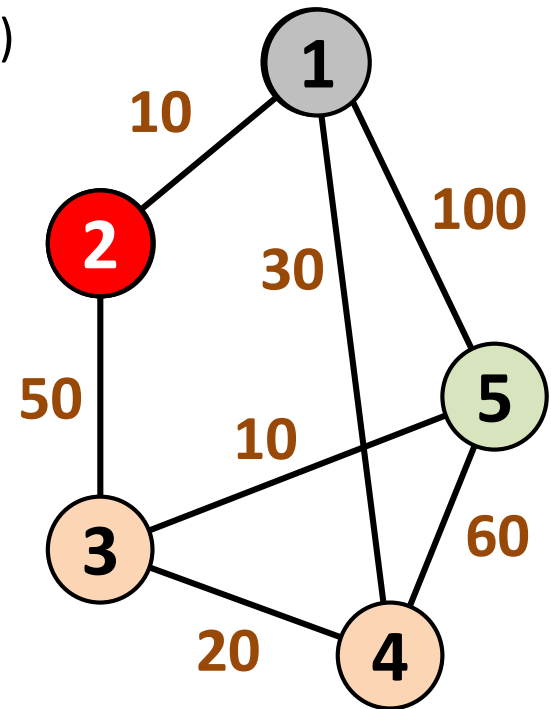
$D[i]$	0	10	∞	30	100	13
--------	---	----	----------	----	-----	----

Алгоритм Дейкстры (Dijkstra)

4. Запускаем цикл из n итераций (по числу вершин)

1. Извлекаем из очереди Q вершину v с минимальным приоритетом – ближайшую к s вершину
2. Отмечаем вершину v как посещенную (помещаем v во множество H)
3. Возможно пути из s через вершину v стали короче, выполняем проверку: для каждой вершины u смежной с вершиной v и не включенной в H проверяем и корректируем расстояние $D[u]$

```
if  $D[v] + w(v, u) < D[u]$  then  
    // Путь из  $s$  до  $u$  через  $(v, u)$  короче  
     $D[u] = D[v] + w(v, u)$   
    PriorityQueueDecrease( $Q, u, D[u]$ )  
     $prev[u] = v$   
end if
```



$D[3] = 60$

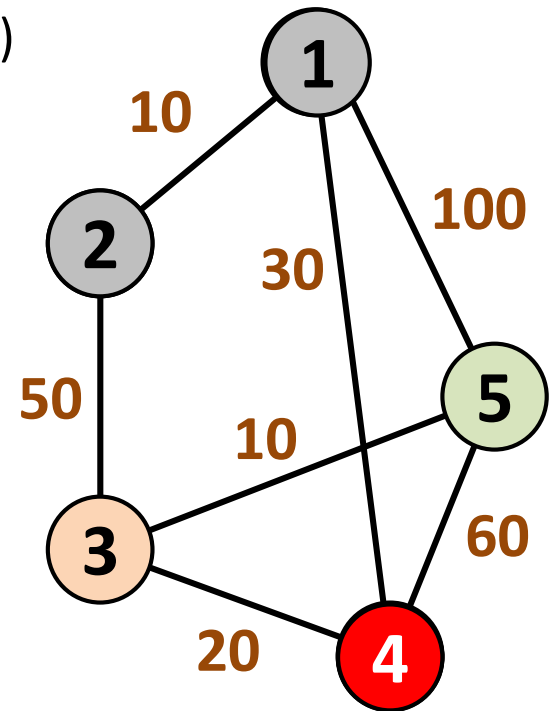
$D[i]$	0	10	60	30	100	14
--------	---	----	----	----	-----	----

Алгоритм Дейкстры (Dijkstra)

4. Запускаем цикл из n итераций (по числу вершин)

1. Извлекаем из очереди Q вершину v с минимальным приоритетом – ближайшую к s вершину
2. Отмечаем вершину v как посещенную (помещаем v во множество H)
3. Возможно пути из s через вершину v стали короче, выполняем проверку: для каждой вершины u смежной с вершиной v и не включенной в H проверяем и корректируем расстояние $D[u]$

```
if  $D[v] + w(v, u) < D[u]$  then  
    // Путь из  $s$  до  $u$  через  $(v, u)$  короче  
     $D[u] = D[v] + w(v, u)$   
    PriorityQueueDecrease( $Q, u, D[u]$ )  
     $prev[u] = v$   
end if
```



$D[3] = 50$

$D[5] = 90$

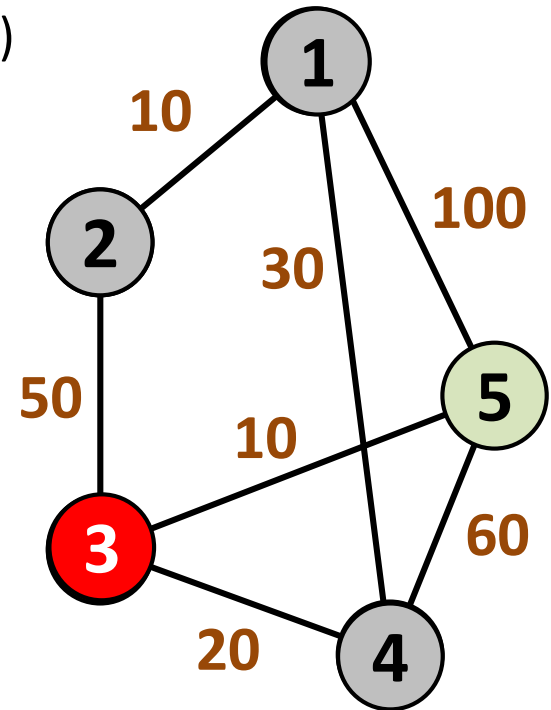
$D[i]$	0	10	50	30	90	15
--------	---	----	----	----	----	----

Алгоритм Дейкстры (Dijkstra)

4. Запускаем цикл из n итераций (по числу вершин)

1. Извлекаем из очереди Q вершину v с минимальным приоритетом – ближайшую к s вершину
2. Отмечаем вершину v как посещенную (помещаем v во множество H)
3. Возможно пути из s через вершину v стали короче, выполняем проверку: для каждой вершины u смежной с вершиной v и не включенной в H проверяем и корректируем расстояние $D[u]$

```
if  $D[v] + w(v, u) < D[u]$  then  
    // Путь из  $s$  до  $u$  через  $(v, u)$  короче  
     $D[u] = D[v] + w(v, u)$   
    PriorityQueueDecrease( $Q, u, D[u]$ )  
    prev[u] = v  
end if
```



$D[5] = 60$

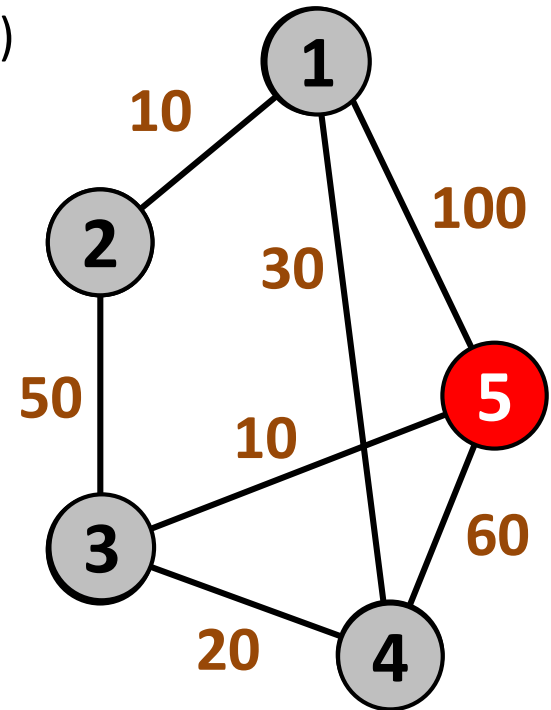
$D[i]$	0	10	50	30	60	16
--------	---	----	----	----	----	----

Алгоритм Дейкстры (Dijkstra)

4. Запускаем цикл из n итераций (по числу вершин)

1. Извлекаем из очереди Q вершину v с минимальным приоритетом – ближайшую к s вершину
2. Отмечаем вершину v как посещенную (помещаем v во множество H)
3. Возможно пути из s через вершину v стали короче, выполняем проверку: для каждой вершины u смежной с вершиной v и не включенной в H проверяем и корректируем расстояние $D[u]$

```
if  $D[v] + w(v, u) < D[u]$  then  
    // Путь из  $s$  до  $u$  через  $(v, u)$  короче  
     $D[u] = D[v] + w(v, u)$   
    PriorityQueueDecrease( $Q, u, D[u]$ )  
     $prev[u] = v$   
end if
```

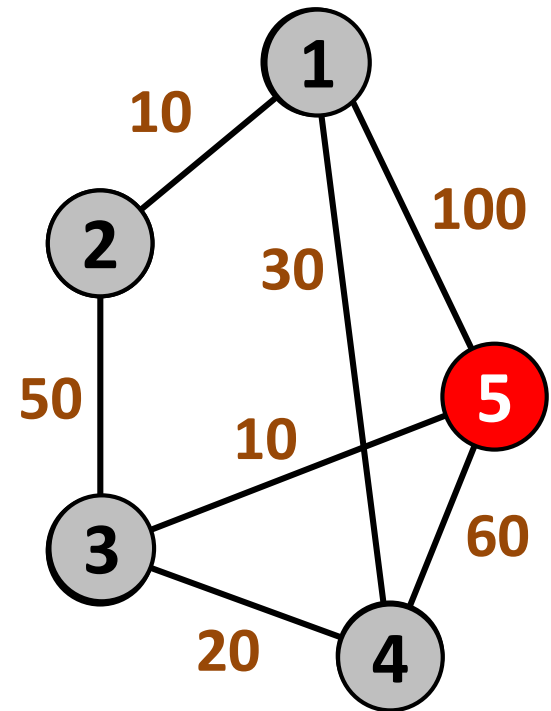


$D[i]$	0	10	50	30	60	17
--------	---	----	----	----	----	----

Алгоритм Дейкстры (Dijkstra)

- В массиве $D[1:n]$ содержатся длины кратчайших путей из начальной вершины $s = 1$
 - $D[1]$ – длина пути из 1 в 1
 - $D[2]$ – длина пути из 1 в 2
 - $D[3]$ – длина пути из 1 в 3
 - $D[4]$ – длина пути из 1 в 4
 - $D[5]$ – длина пути из 1 в 5

$D[i]$	0	10	50	30	60
--------	---	----	----	----	----



- Как определить какие *вершины* входят в кратчайший путь из $s = 1$ в $d = 5$?
- Как восстановить путь?

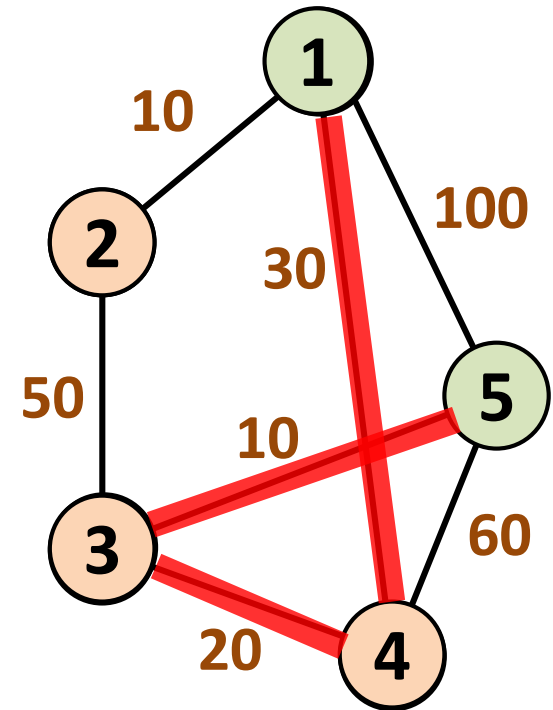
Алгоритм Дейкстры (Dijkstra)

- Восстановление кратчайшего пути
- Массив $prev[i]$ содержит номер вершины, предшествующей i в пути

	1	2	3	4	5
$prev[i]$	-1	1	4	1	3

- Восстанавливаем путь с конца
 - ☐ Вершина 5
 - ☐ Вершина $prev[5] = 3$
 - ☐ Вершина $prev[3] = 4$
 - ☐ Вершина $prev[4] = 1$

Кратчайший путь (1, 4, 3, 5)



$D[i]$	0	10	50	30	60
--------	---	----	----	----	----

Алгоритм Дейкстры (Dijkstra)

```
function ShortestPath_Dijkstra(G, src, d, prev)
    // Input: G = (V, E), src, dst
    // Output: d[1:n], prev[1:n]
    //     prev[i] - узел, предшествующий i в пути

    // Помещаем вершины в очередь с приоритетом
    for each i in V \ {src} do
        d[i] = Infinity
        prev[i] = -1
        PriorityQueueInsert(Q, i, d[i])
    end for

    d[src] = 0
    prev[src] = -1
    PriorityQueueInsert(Q, src, d[src])
```

Алгоритм Дейкстры (Dijkstra)

```
for i = 0 to n - 1 do
    // Извлекаем узел ближайший к начальному
    v = PriorityQueueRemoveMin(Q)
    // Отмечаем v как посещенный
    H = H + {v}

    // Цикл по смежным вершинам узла v
    for each u in Adj(v) \ H do
        // Путь через u короче текущего пути?
        if d[v] + w(v, u) < d[u] then
            d[u] = d[v] + w(v, u)
            PriorityQueueDecrease(Q, u, d[u])
            prev[u] = v
        end if
    end for
end for
end function
```

Восстановление кратчайшего пути

```
function SearchShortestPath(G, src, dst)
    ShortestPath_Dijkstra(G, src, d, prev)

    // Восстановление пути из src в dst
    i = dst
    pathlen = 1
    while i != src do
        pathlen = pathlen + 1
        i = prev[i]
    end while

    j = 0
    i = dst
    while i != s do
        path[pathlen - j] = i
        i = prev[i]
        j = j + 1
    end while
    return path[], pathlen
end function
```

$$T = T_{Dijkstra} + O(|V|)$$

Вычислительная сложность алгоритм Дейкстры

- Вычислительная сложность алгоритма Дейкстры определяется следующими факторами:
 - 1) выбором структуры данных для хранения графа (матрица смежности, списки смежных вершин)
 - 2) способом поиска вершины с минимальным расстоянием $D[i]$:
 - Очередь с приоритетом:
Binary heap – $O(\log n)$, Fibonacci heap, ...
 - Сбалансированное дерево поиска:
Red-black tree – $O(\log n)$, AVL-tree, ...
 - Линейный поиск – $O(n)$

Алгоритм Дейкстры (Dijkstra)

```
function ShortestPath_Dijkstra(G, src, d, prev)
```

```
    // Input: G = (V, E), src, dst
```

```
    // Output: d[1:n], prev[1:n]
```

```
    //   prev[i] - узел, предшествующий i в пути
```

```
    // Помещаем вершины в очередь с приоритетом
```

```
    for each i in V \ {src} do
```

```
        d[i] = Infinity
```

```
        prev[i] = -1
```

```
        PriorityQueueInsert(Q, i, d[i])
```

BinaryHeap

$O(n \log n)$

```
    end for
```

```
    d[src] = 0
```

```
    prev[src] = -1
```

```
    PriorityQueueInsert(Q, src, d[src])
```

$O(\log n)$

Алгоритм Дейкстры (Dijkstra)

```
for i = 0 to n - 1 do BinaryHeap  
    // Извлекаем узел ближайший к начальному  
    v = PriorityQueueRemoveMin(Q)  $O(\log n)$   
    // Отмечаем v как посещенный  
    H = H + {v}  
  
    // Цикл по смежным вершинам узла v  
    for each u in Adj(v) \ H do  
        // Путь через u короче текущего пути?  
        if d[v] + w(v, u) < d[u] then  $O(\log n)$   
            d[u] = d[v] + w(v, u)  
            PriorityQueueDecrease(Q, u, d[u])  
            prev[u] = v
```

- В худшем случае функция PriorityQueueRemoveMin() вызывается n раз, суммарная сложность $O(n \log n)$
- В худшем случае функция PriorityQueueDecrease() вызывается для каждого из m ребер графа, суммарная сложность $O(m \log n)$

Вычислительная сложность алгоритм Дейкстры

- **Вариант 1.** $D[i]$ – это массив или список (поиск за время $O(n)$)

$$T_{Dijkstra} = O(n^2 + m) = O(|V|^2 + |E|)$$

- **Вариант 2.** $D[i]$ хранятся в бинарной куче (Binary heap)

$$T_{Dijkstra} = O(n \log n + m \log n) = O(m \log n)$$

- **Вариант 3.** $D[i]$ хранятся в Фибоначчиевой куче (Fibonacci heap)

$$T_{Dijkstra} = O(m + n \log n)$$

- **Вариант 4. ...**

- В ориентированных ациклических графах (Directed acyclic graph) кратчайший путь можно найти за время $O(n)$

Разреженные и насыщенные графы

Вариант реализации алгоритма Дейкстры	Насыщенный граф $m = O(n^2)$	Разреженный граф $m = O(n)$
Вариант 1 $D[i]$ – это массив (поиск за время $O(n)$)	$T = O(n^2 + m) =$ $O(n^2)$	$T = O(n^2 + m) =$ $O(n^2)$
Вариант 2 $D[i]$ хранятся в бинарной куче	$T = O(n \log n + m \log n)$ $= O(n^2 \log n)$	$T = O(n \log n + m \log n)$ $= O(n \log n)$
Вариант 3 $D[i]$ хранятся в Фибоначчиевой куче	$T = O(m + n \log n)$ $= O(n^2)$	$T = O(n + n \log n)$ $= O(n \log n)$

Алгоритм Дейкстры + бинарная куча

- **Необходимые операции над очередью с приоритетом**

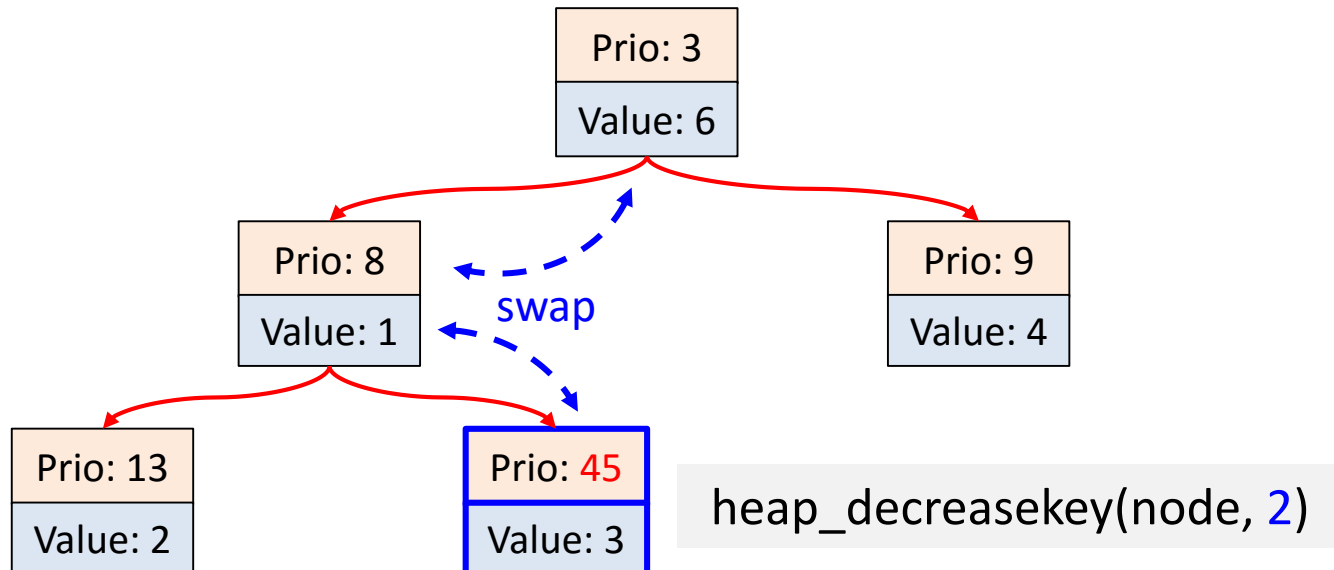
- ❑ `int heap_insert(struct heap *h,
int key, int value)`

- ❑ `int heap_remove_min(struct heap *h)`

- ❑ `void heap_decreasekey(struct heap *node,
int key)`

Алгоритм Дейкстры + бинарная куча

- `void heap_decreasekey(struct heap *node, int key)`
- Изменяем у узла `node` приоритет на `priority`
- Восстанавливаем свойства кучи – поднимаем элемент вверх по дереву (min-heap)



Домашнее задание + чтение

- Оценить трудоемкость по памяти алгоритма Дейкстры в следующих случаях:
 - при использовании матрицы смежности и двоичной кучи;
 - при использовании списков смежности и двоичной кучи
- Описание алгоритма Дейкстры в [CLRS, С. 696]