Лекция 11.
Поиск кратчайшего пути в графе

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.

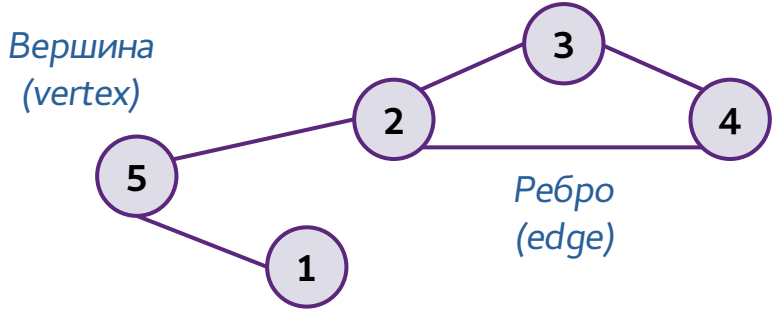
Понятие графа

- **Граф** (graph) — это совокупность непустого множества *V* вершин и множества *E* рёбер

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

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

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



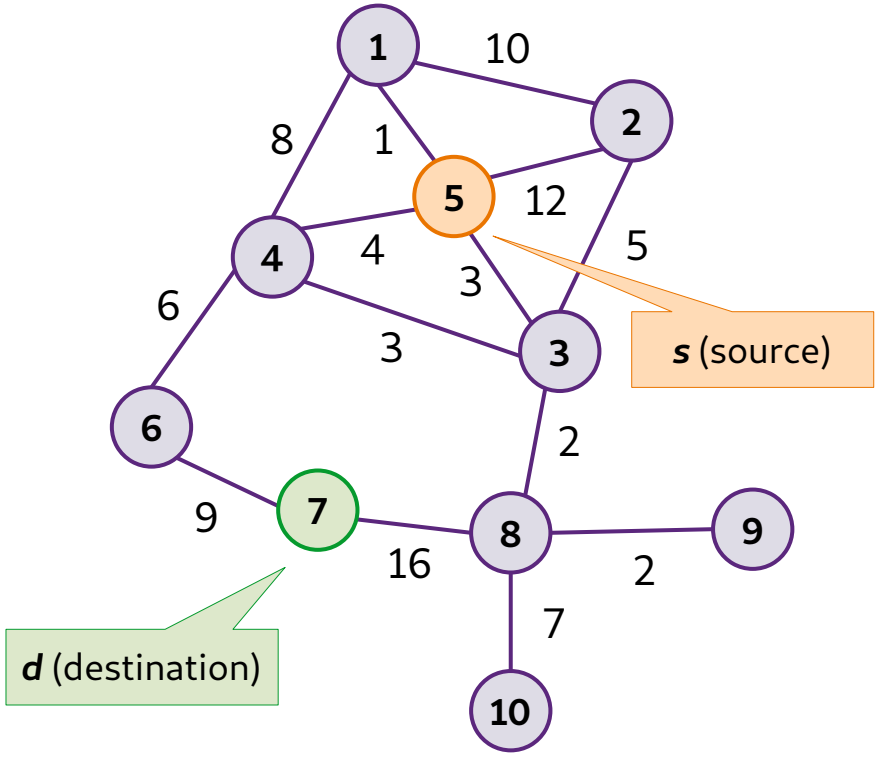
V = {1, 2, 3, 4, 5}
E = {(1, 5), (2, 5), (2, 3), (2, 4), (3, 4)}

(1, 5) и (5, 1) — это одно и то же ребро

Поиск кратчайшего пути в графе

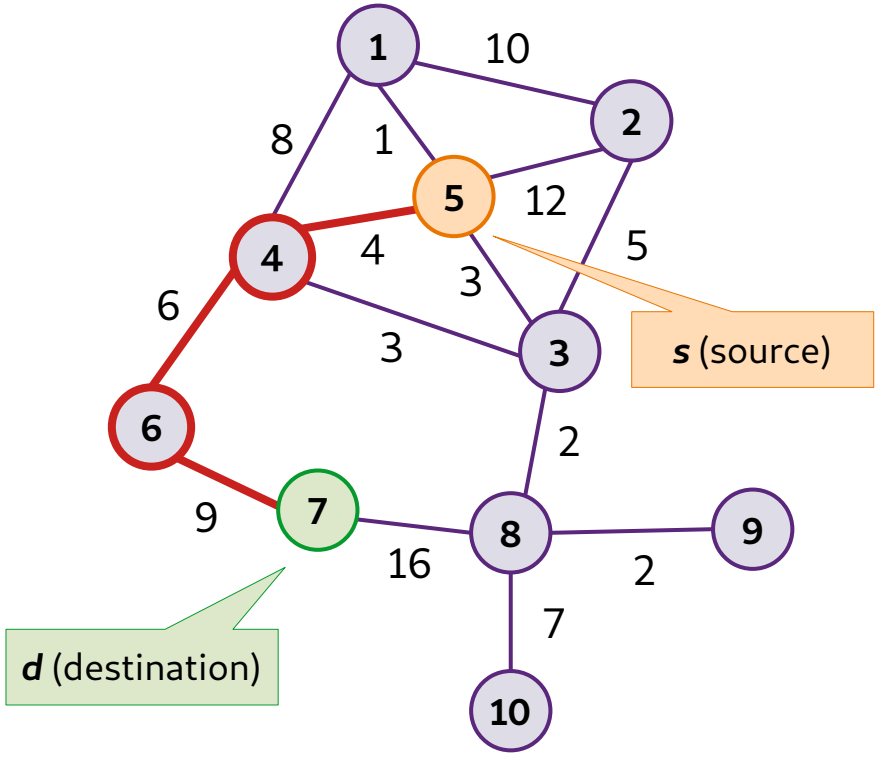
- **Имеется** взвешенный граф $G = (V, E)$
- Каждому ребру $(i, j) \in E$ назначен вес w_{ij}
- Заданы начальная вершина $s \in V$ и конечная $d \in V$

- **Требуется** найти *кратчайший путь* из вершины s в вершину d (shortest path problem)
- Длина пути (path length, path cost, path weight) — это сумма весов рёбер, входящих в него



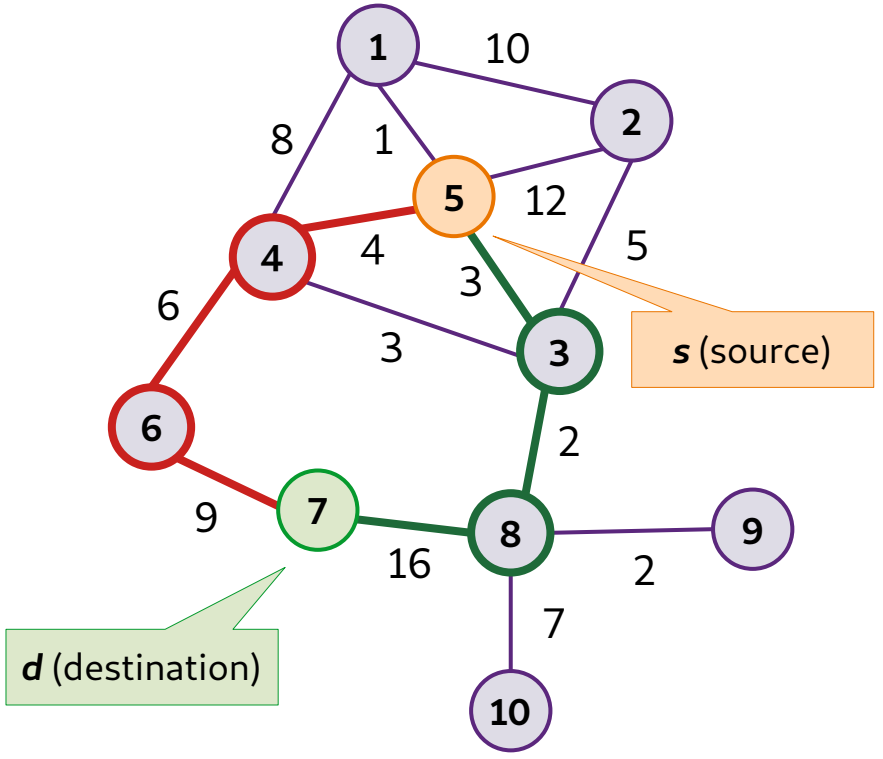
Длина пути в графе

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



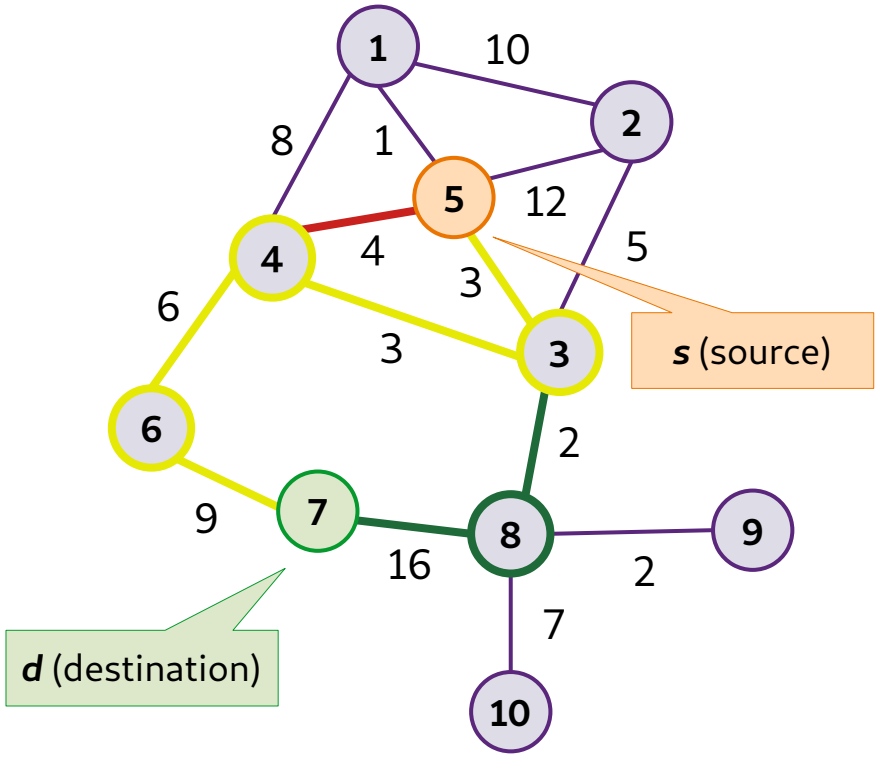
Длина пути в графе

- 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

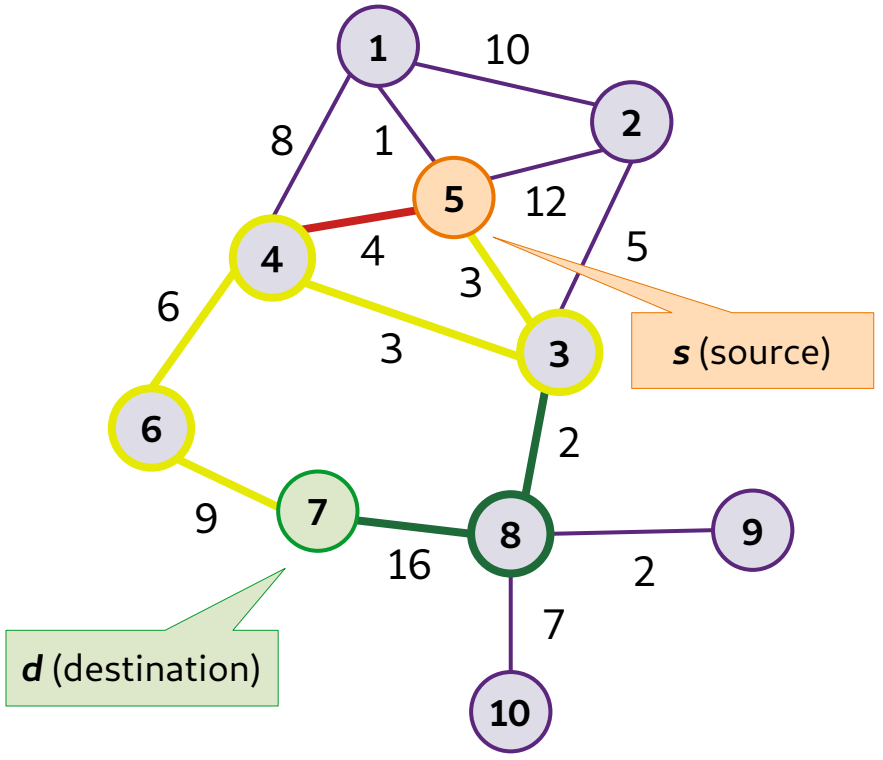
- Длина пути (5, 4, 6, 7) = $w_{54} + w_{46} + w_{67} = 4 + 6 + 9 = 19$
- Длина пути (5, 3, 8, 7) = 3 + 2 + 16 = 21
- Длина пути (5, 3, 4, 6, 7) = 3 + 3 + 6 + 9 = 21



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)**
  Требуется 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 *d* из всех вершин графа

- **Задача о кратчайшем пути между всеми парами вершин (all-pairs shortest path problem)**
  Требуется найти кратчайший путь из каждой вершины *v* в каждую вершину *u*

Алгоритмы поиска кратчайшего пути в графе

Алгоритм	Применение
Алгоритм Дейкстры	Находит кратчайший путь от одной вершины графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса ($w_{ij} \geq 0$)
Алгоритм Беллмана-Форда	Находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес рёбер 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 *d* графа, содержащий минимальное число промежуточных вершин (трассировка электрических соединений на кристаллах микросхем и печатных платах)
Алгоритмы Витерби, Черкасского, ...	

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)
- Позволяет найти кратчайшее расстояние от одной из вершин графа до всех остальных
- Применим только для графов без рёбер отрицательного веса и петель ($w_{ij} \geq 0$)

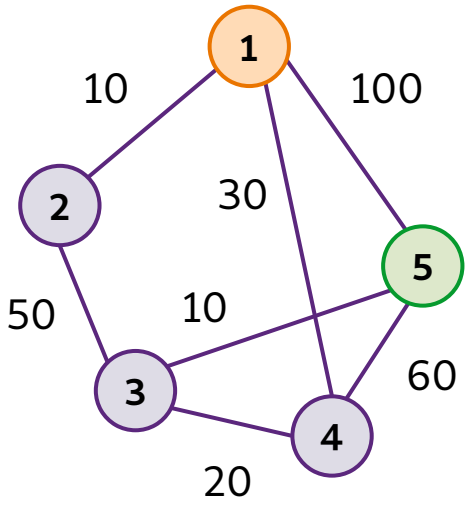
- **Эдсгер Вибе Дейкстра** (Edsger Wybe Dijkstra) — нидерландский учёный (структурное программирование, язык Алгол, семафоры, распределённые вычисления)
- Лауреат премии Тьюринга (ACM A. M. Turing Award)

- Дейкстра Э. Дисциплина программирования = A disciple of programming. — 1-е изд. — М.: Мир, 1978. — 275 с.
- Дал У., Дейкстра Э., Хоор К. Структурное программирование = Structured programming. — 1-е изд. — М.: Мир, 1975. — 247 с.

Алгоритм Дейкстры

- **Пример:** найти кратчайший путь из вершины **1** в вершину **5**

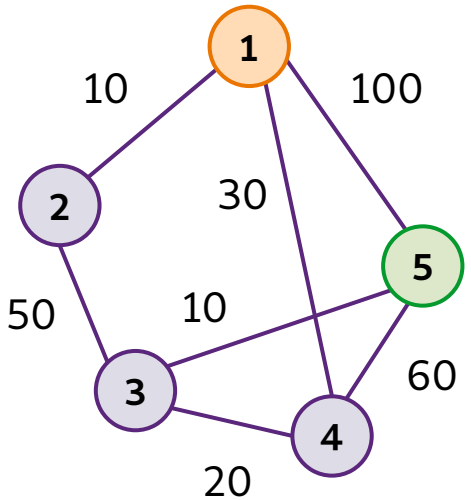
- Введём обозначения:
  - H — множество посещённых вершин
  - $D[i]$ — текущее известное расстояние от вершины s до вершины i
  - $prev[i]$ — номер вершины, предшествующей i в кратчайшем пути



Алгоритм Дейкстры

- **Пример:** найти кратчайший путь из вершины **1** в вершину **5**

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 $D[i]$

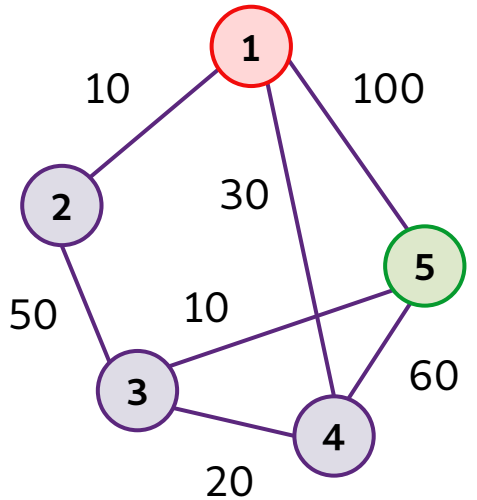


	1	2	3	4	5
D:	0	∞	∞	∞	∞

Алгоритм Дейкстры

4. Запускаем цикл из *n* итераций (по числу вершин):
   а. Извлекаем из очереди *Q* вершину *v* с минимальным приоритетом — ближайшую к *s* вершину
   б. Отмечаем вершину *v* как посещенную (помещаем во множество *H*)
   в. Возможно, пути из *s* через вершину *v* стали короче, выполняем проверку: для каждой вершины *u*, смежной с *v* и не включённой 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)
    PriorityQueueDecreaseKey(Q, u, D[u])
    prev[u] = v
end if
```



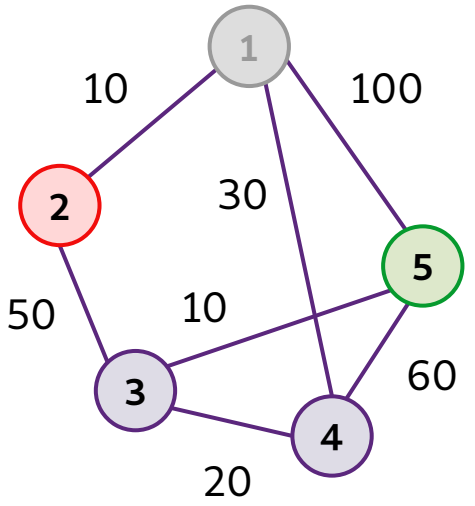
D[2] = 10
D[4] = 30
D[5] = 100

	1	2	3	4	5
D:	0	10	∞	30	100

Алгоритм Дейкстры

4. Запускаем цикл из *n* итераций (по числу вершин):
   а. Извлекаем из очереди *Q* вершину *v* с минимальным приоритетом — ближайшую к *s* вершину
   б. Отмечаем вершину *v* как посещенную (помещаем во множество *H*)
   в. Возможно, пути из *s* через вершину *v* стали короче, выполняем проверку: для каждой вершины *u*, смежной с *v* и не включённой 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)
    PriorityQueueDecreaseKey(Q, u, D[u])
    prev[u] = v
end if
```



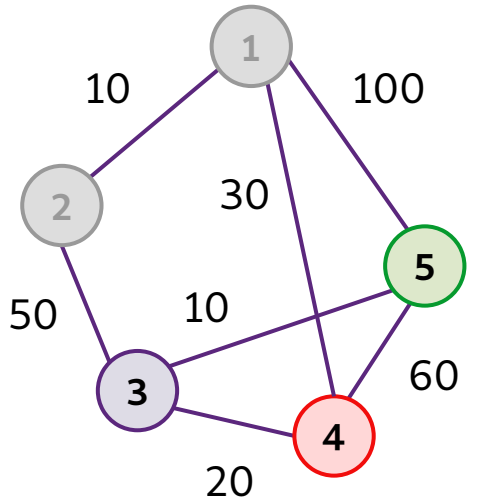
D[3] = 60

	1	2	3	4	5
D:	0	10	60	30	100

Алгоритм Дейкстры

4. Запускаем цикл из *n* итераций (по числу вершин):
   а. Извлекаем из очереди *Q* вершину *v* с минимальным приоритетом — ближайшую к *s* вершину
   б. Отмечаем вершину *v* как посещенную (помещаем во множество *H*)
   в. Возможно, пути из *s* через вершину *v* стали короче, выполняем проверку: для каждой вершины *u*, смежной с *v* и не включённой 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)
    PriorityQueueDecreaseKey(Q, u, D[u])
    prev[u] = v
end if
```



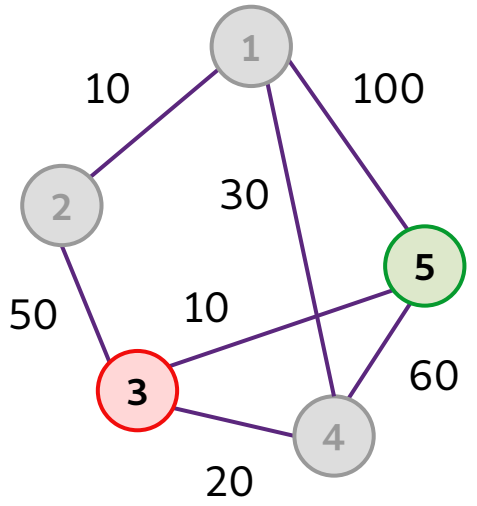
D[3] = 50
D[5] = 90

	1	2	3	4	5
D:	0	10	50	30	90

Алгоритм Дейкстры

4. Запускаем цикл из *n* итераций (по числу вершин):
   а. Извлекаем из очереди *Q* вершину *v* с минимальным приоритетом — ближайшую к *s* вершину
   б. Отмечаем вершину *v* как посещенную (помещаем во множество *H*)
   в. Возможно, пути из *s* через вершину *v* стали короче, выполняем проверку: для каждой вершины *u*, смежной с *v* и не включённой 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)
    PriorityQueueDecreaseKey(Q, u, D[u])
    prev[u] = v
end if
```

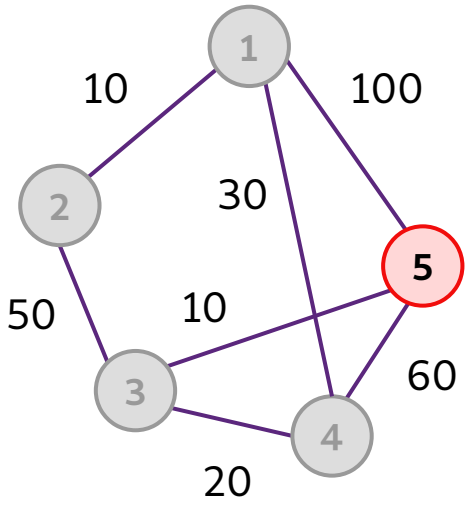


D[5] = 60

	1	2	3	4	5
D:	0	10	50	30	60

Алгоритм Дейкстры

4. Запускаем цикл из *n* итераций (по числу вершин):
   а. Извлекаем из очереди *Q* вершину *v* с минимальным приоритетом — ближайшую к *s* вершину
   б. Отмечаем вершину *v* как посещенную (помещаем во множество *H*)
   в. Возможно, пути из *s* через вершину *v* стали короче, выполняем проверку: для каждой вершины *u*, смежной с *v* и не включённой в *H*, проверяем и корректируем расстояние *D*[*u*]



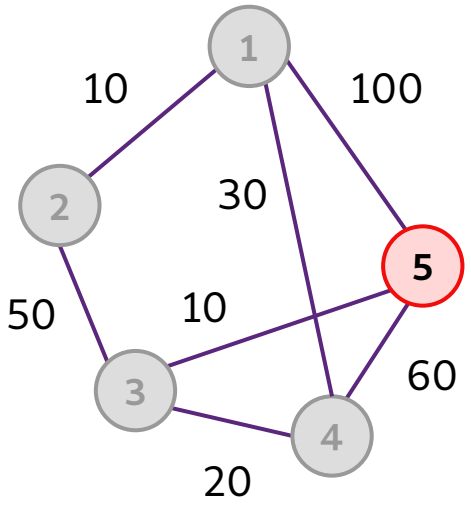

```
if D[v] + w(v, u) < D[u] then
    // Путь из s до u через (v, u) короче
    D[u] = D[v] + w(v, u)
    PriorityQueueDecreaseKey(Q, u, D[u])
    prev[u] = v
end if
```

	1	2	3	4	5
D:	0	10	50	30	60

Алгоритм Дейкстры

- 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

	1	2	3	4	5
D:	0	10	50	30	60



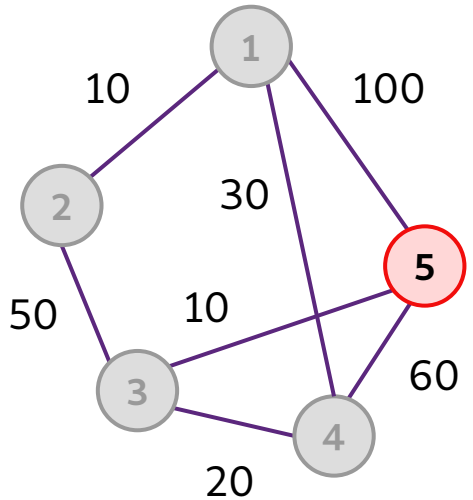
- **Какие вершины входят в кратчайший путь из $s = 1$ в $d = 5$?**
- **Как восстановить путь?**

Алгоритм Дейкстры

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

	1	2	3	4	5
prev:	-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)**



	1	2	3	4	5
D:	0	10	50	30	60

Алгоритм Дейкстры

```
function ShortestPathDijkstra(G, src, D, prev)
    for each i in V \ src do    // Помещаем вершины в очередь с приоритетом
        D[i] = ∞
        prev[i] = -1
        PriorityQueueInsert(Q, i, D[i])
    end for
    D[src] = 0
    prev[src] = -1
    PriorityQueueInsert(Q, src, D[src])
```

Алгоритм Дейкстры (продолжение)

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

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

Восстановление кратчайшего пути

```
function SearchShortestPath(G, src, dst)
    ShortestPathDijkstra(G, src, D, prev)
    i = dst
    pathlen = 1
    while i != src do
        pathlen = pathlen + 1
        i = prev[i]
    end while
    j = 0
    i = dst
    while i != src do
        path[pathlen - j] = i
        j = j + 1
    end while
    return path[], pathlen
end function
```

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

Вычислительная сложность алгоритма Дейкстры

- Вычислительная сложность алгоритма Дейкстры определяется следующими факторами:
  1. Выбор структуры данных для хранения графа (матрица смежности, список смежности)
  2. Способ поиска вершины с минимальным расстоянием $D[i]$:
     - Очередь с приоритетом: бинарная куча — $O(\log n)$, фибоначчиева куча, ...
     - Сбалансированное дерево поиска: красно-чёрное дерево — $O(\log n)$, АВЛ-дерево, ...
     - Линейный поиск — $O(n)$

Вычислительная сложность алгоритма Дейкстры

```
function ShortestPathDijkstra(G, src, D, prev)
    for each i in V \ src do    // Помещаем вершины в очередь с приоритетом
        D[i] = ∞
        prev[i] = -1
        PriorityQueueInsert(Q, i, D[i])
    end for
    D[src] = 0
    prev[src] = -1
    PriorityQueueInsert(Q, src, D[src])
```

Binary heap: $O(n \log n)$

Вычислительная сложность алгоритма Дейкстры

```
    for i = 0 to n - 1 do                                                    Binary heap
        PriorityQueueExtractMin(Q)     // Извлекаем узел, ближайший к начальному    O(logn)
        H = H + {v}                    // Отмечаем v как посещённый

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

- В худшем случае функция *PriorityQueueExtractMin*() вызывается *n* раз, суммарная сложность — $O(n\log n)$
- В худшем случае функция *PriorityQueueDecreaseKey*() вызывается для каждого из *m* рёбер графа, суммарная сложность — $O(m\log n)$

Вычислительная сложность алгоритма Дейкстры

- **Вариант 1.** D — это массив или список: поиск за время $O(n)$

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

- **Вариант 2.** D — это бинарная куча

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

- **Вариант 3.** D — это фибоначчиева куча (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** $m = O(n)$
D — массив	$T = O(n^2 + m) = \mathbf{O(n^2)}$	$T = O(n^2 + m) = \mathbf{O(n^2)}$
D — бинарная куча	$T = O(n\log n + m\log n) = \mathbf{O(n^2 \log n)}$	$T = O(n\log n + m\log n) = \mathbf{O(n \log n)}$
D — фибоначчиева куча	$T = O(m + n\log n) = \mathbf{O(n^2)}$	$T = O(m + n\log n) = \mathbf{O(n \log n)}$

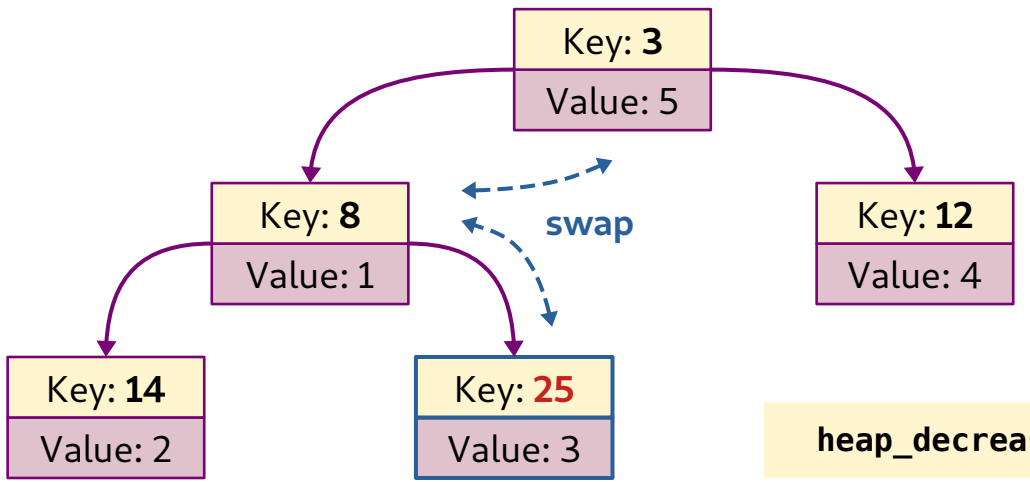
Алгоритм Дейкстры на основе бинарной кучи

- **Необходимые операции очереди с приоритетом:**
  - `int heap_insert(struct heap *h, int key, int value)`
  - `int heap_extract_min(struct heap *h)`
  - `void heap_decrease_key(struct heap *node, int newkey)`

Алгоритм Дейкстры на основе бинарной кучи

- `void heap_decrease_key(struct heap *node, int newkey)`

- *Изменяем приоритет узла node на newkey (newkey < node.key)*
- Восстанавливаем свойства кучи — поднимаем элемент вверх по дереву (min-heap)




```
heap_decrease_key(node, 2)
```

Домашнее чтение

- Оцените трудоёмкость по памяти алгоритма Дейкстры в следующих случаях:
  - при использовании матрицы смежности и двоичной кучи
  - при использовании списков смежности и двоичной кучи

- Ознакомьтесь с описанием алгоритма Дейкстры в [**CLRS**, С. 696]

ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.