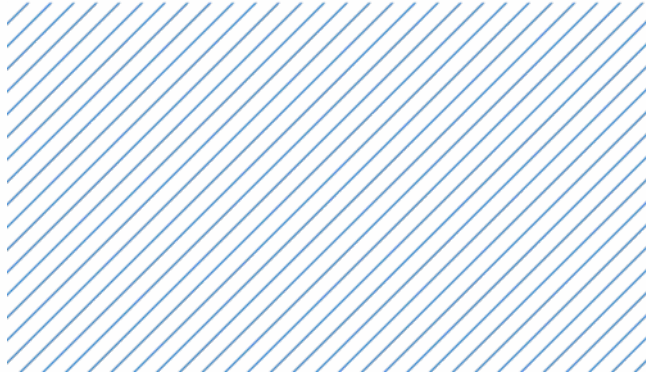
HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Лекция

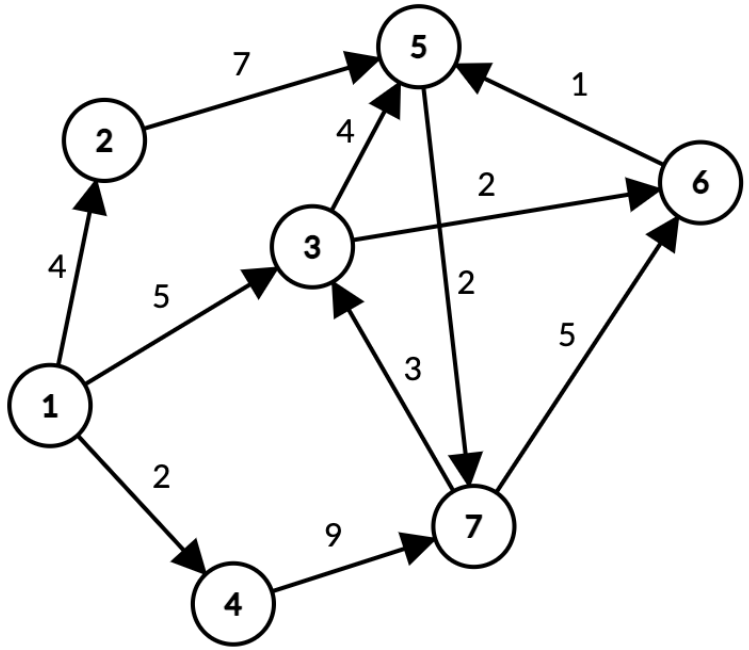
Имплементация и анализ
Алгоритм IDA*

Константин Леладзе
ФКН ВШЭ



Задача первая

Дан **взвешенный ориентированный** граф *G(V, E)* **без ребер отрицательного веса**.
Найти кратчайшие пути от некоторой вершины *start* графа *G* до всех остальных вершин этого графа.



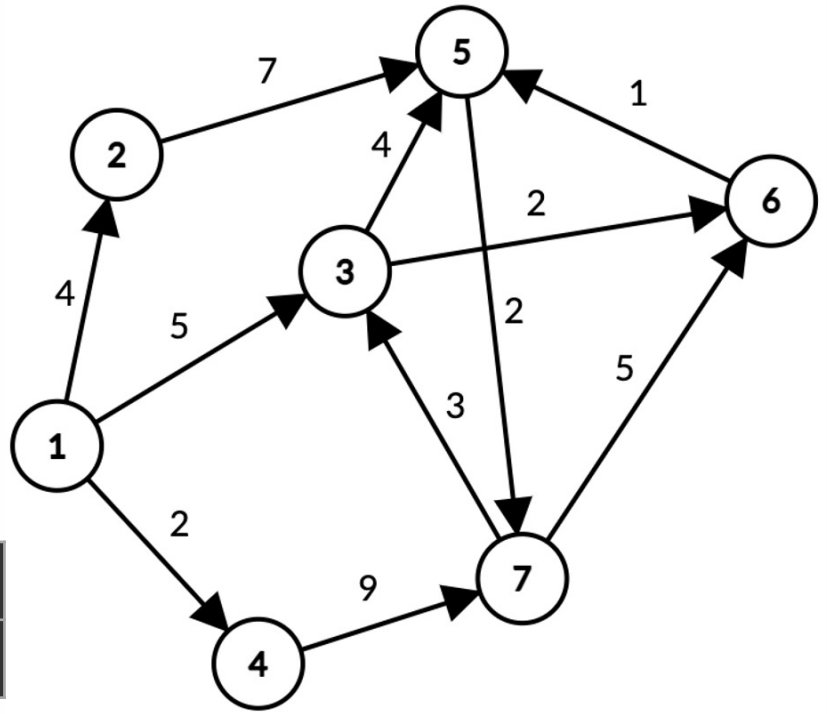
Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \rightarrow w$ с весом e, обновим метку: $d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{1, 2, 3, 4, 5, 6, 7\}$

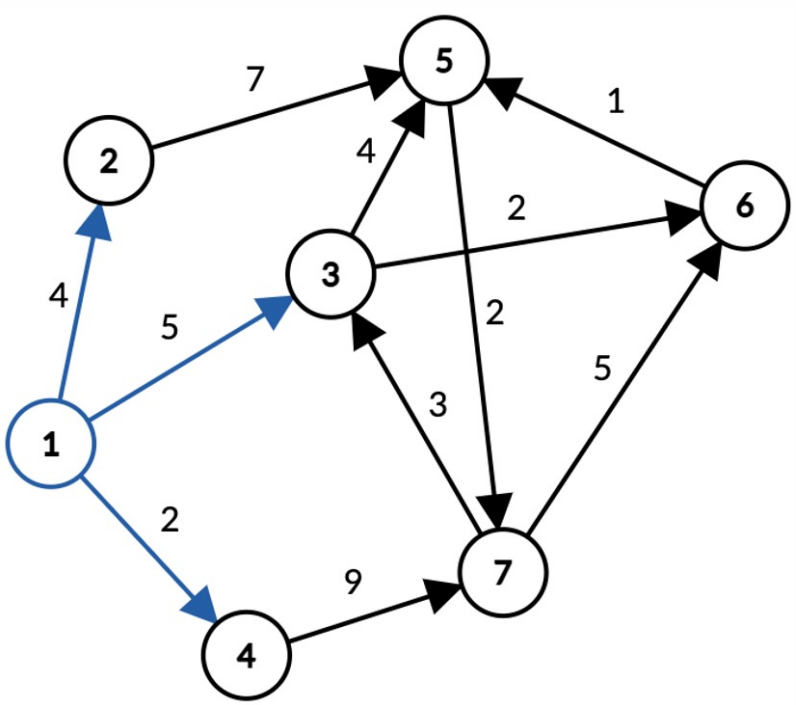
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	∞	∞	∞	∞	∞	∞

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \rightarrow w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{1, 2, 3, 4, 5, 6, 7\}$

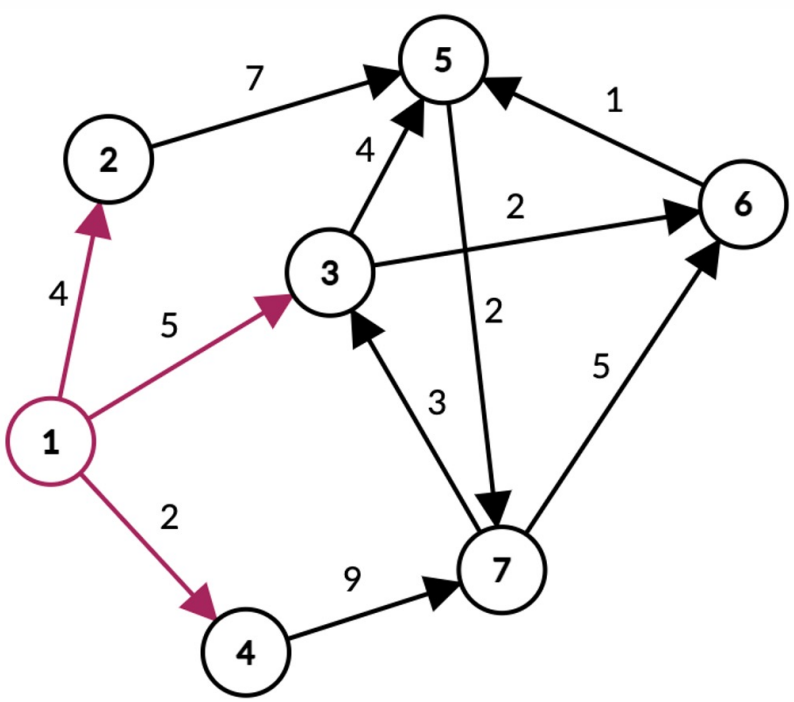
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	∞	∞	∞

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \rightarrow w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{\sout{1}, 2, 3, 4, 5, 6, 7\}$

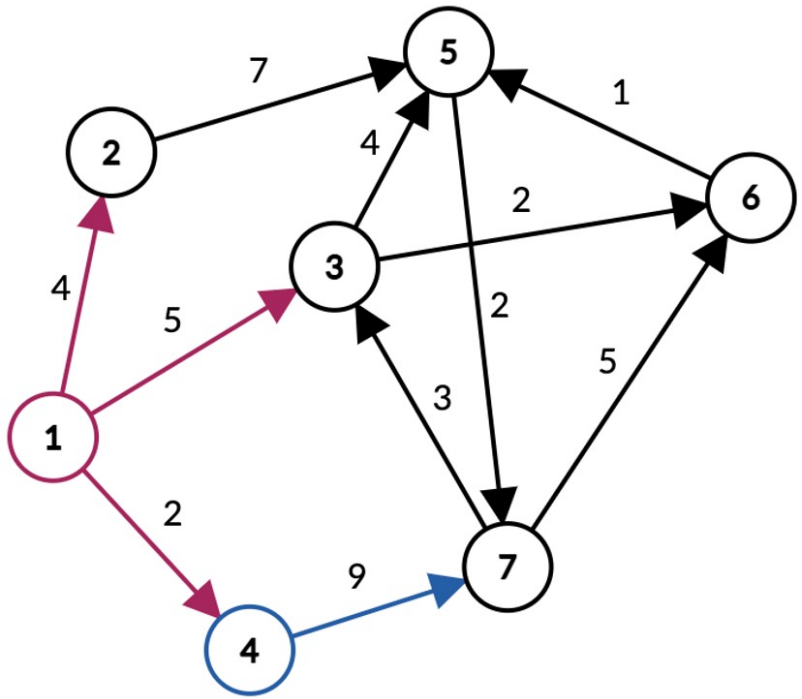
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	∞	∞	∞

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \rightarrow w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{2, 3, 4, 5, 6, 7\}$

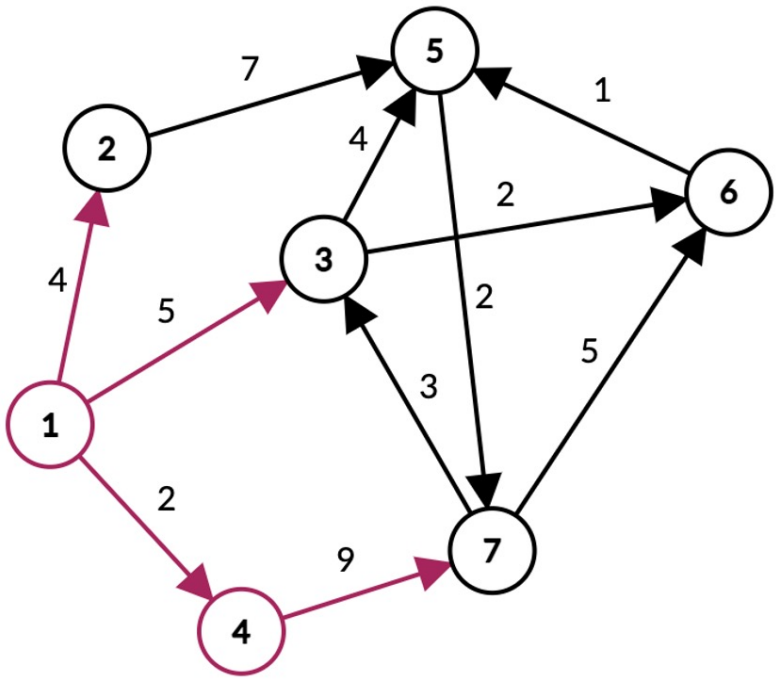
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	∞	∞	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{2, 3, 4, 5, 6, 7\}$

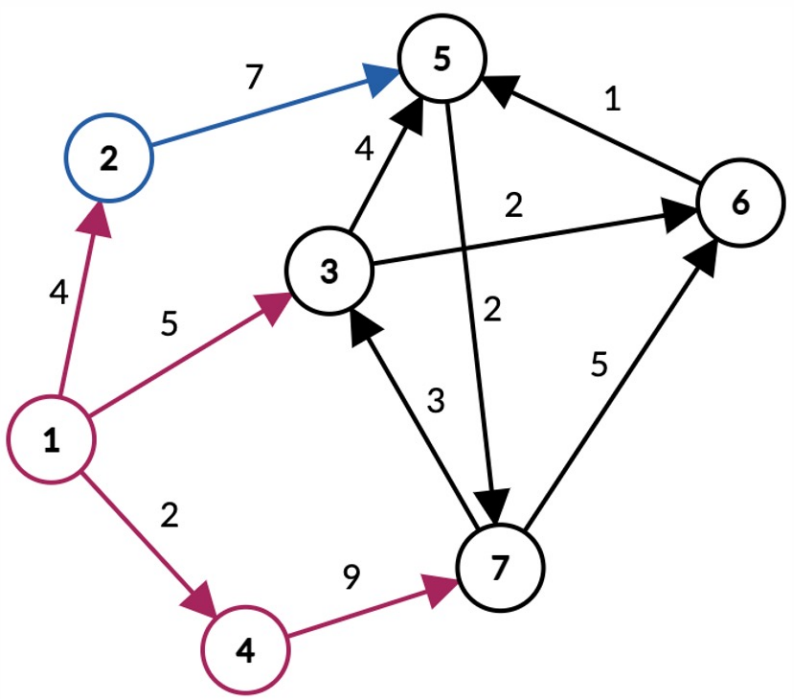
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	∞	∞	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{2, 3, 5, 6, 7\}$

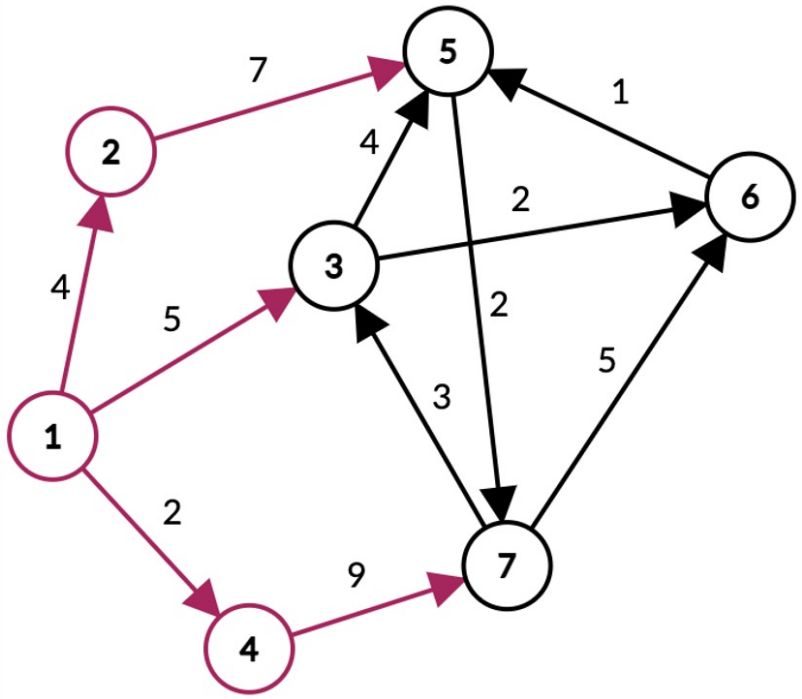
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	11	∞	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \rightarrow w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{\cancel{2}, 3, 5, 6, 7\}$

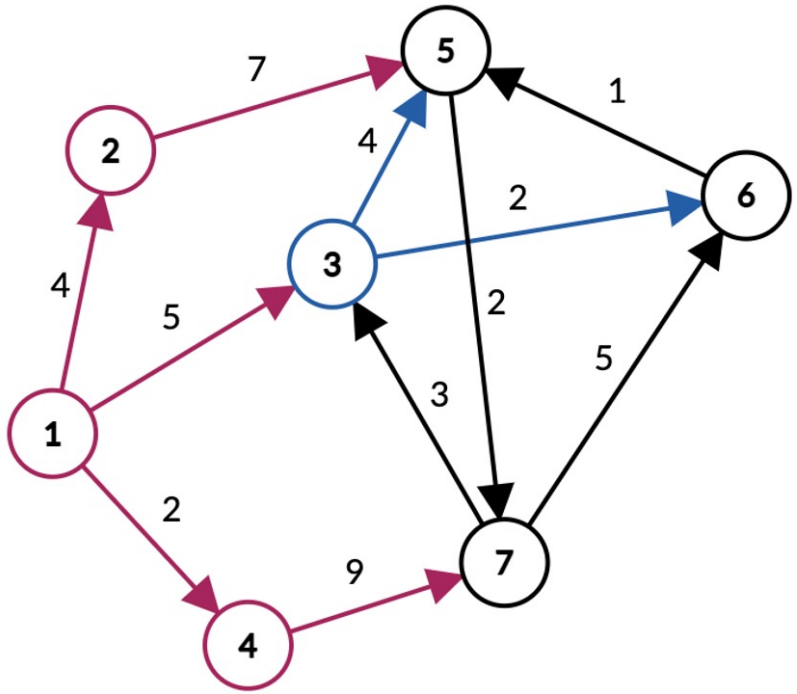
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	11	∞	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{3, 5, 6, 7\}$

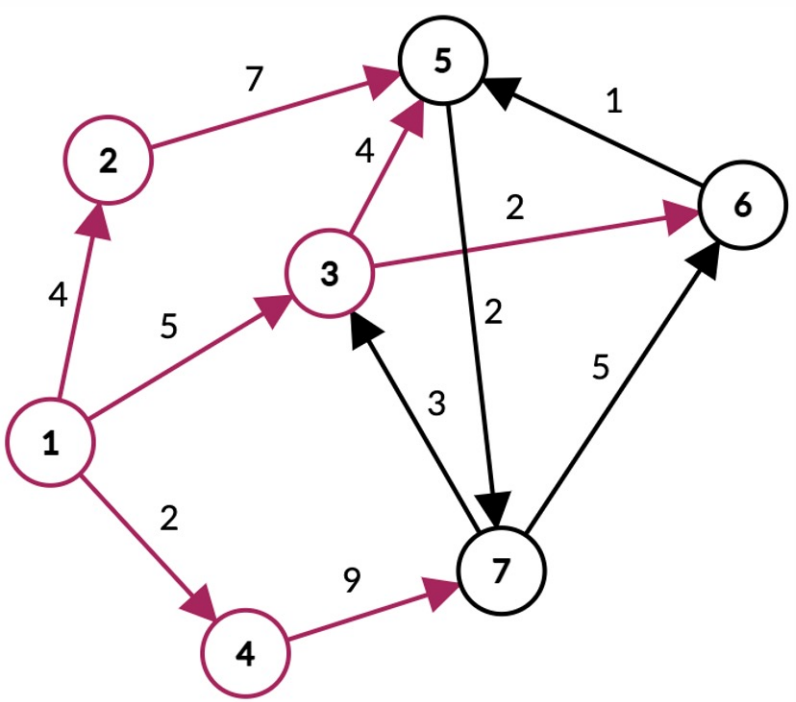
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	9	7	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{\cancel{3}, 5, 6, 7\}$

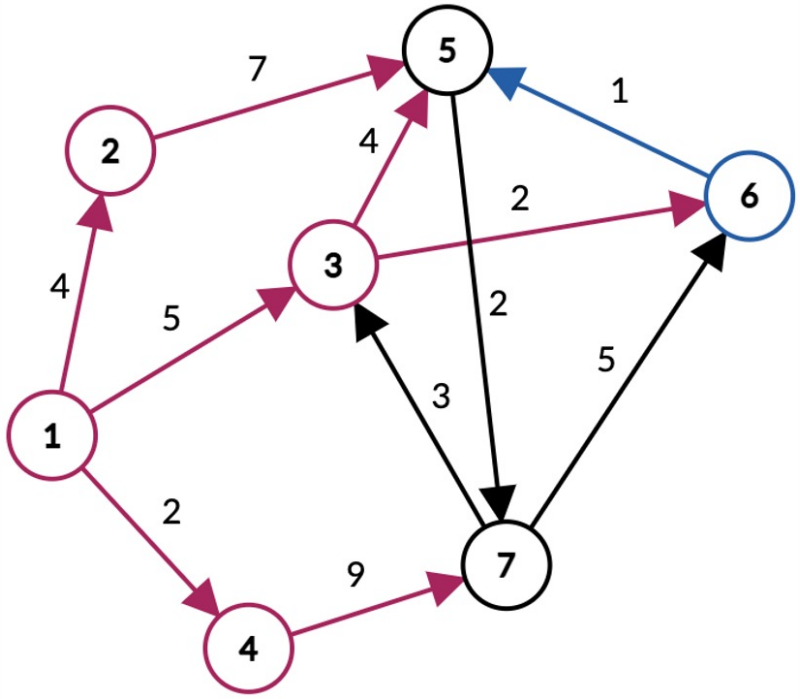
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	9	7	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{5, 6, 7\}$

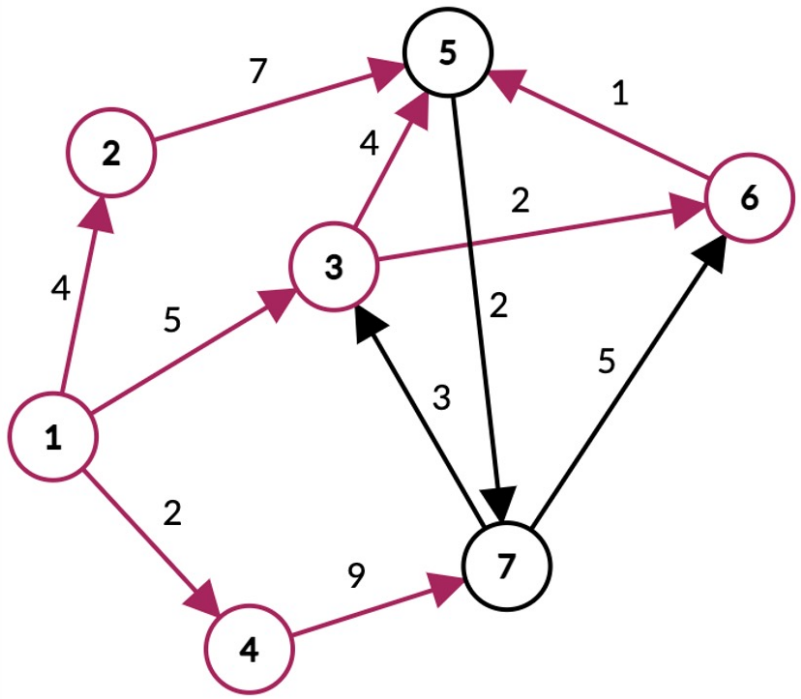
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	8	7	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \rightarrow w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{5, \cancel{6}, 7\}$

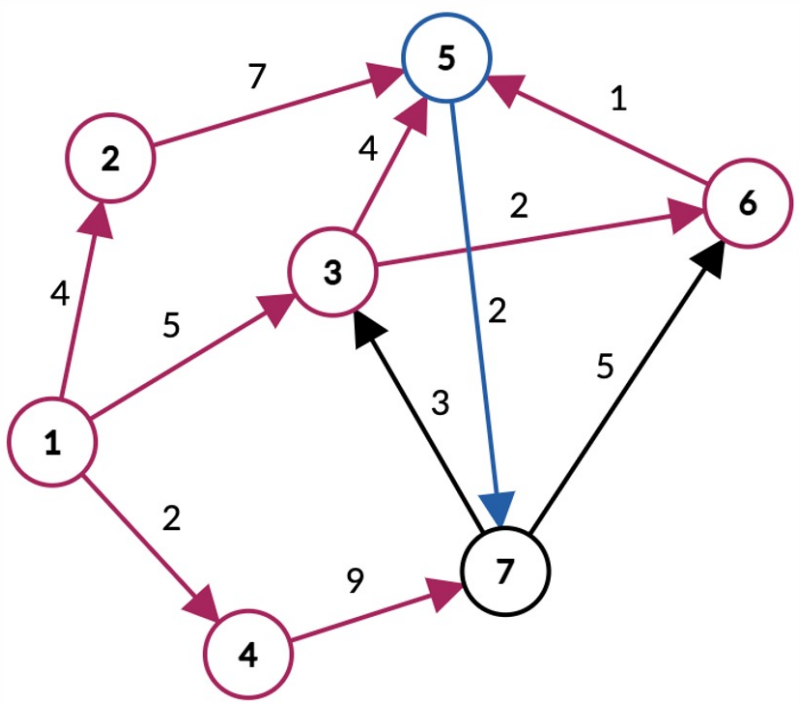
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	8	7	11

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{5, 7\}$

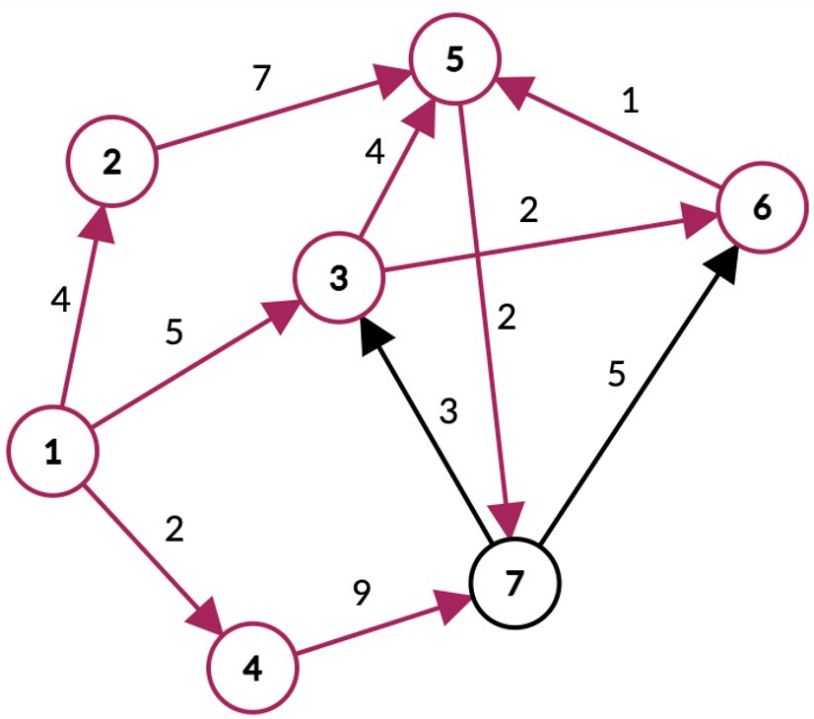
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	8	7	10

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{\cancel{5}, 7\}$

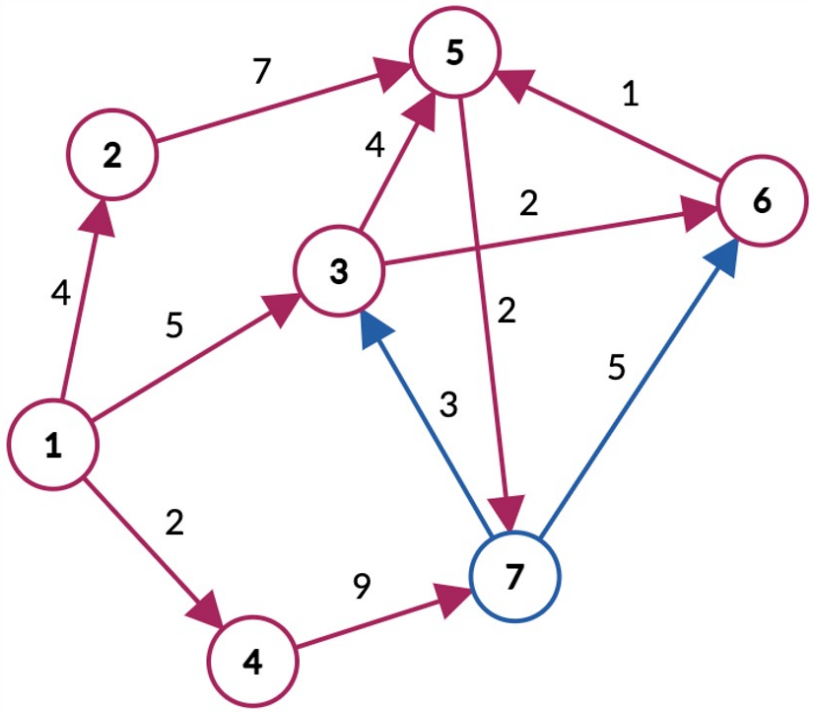
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	8	7	10

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{7\}$

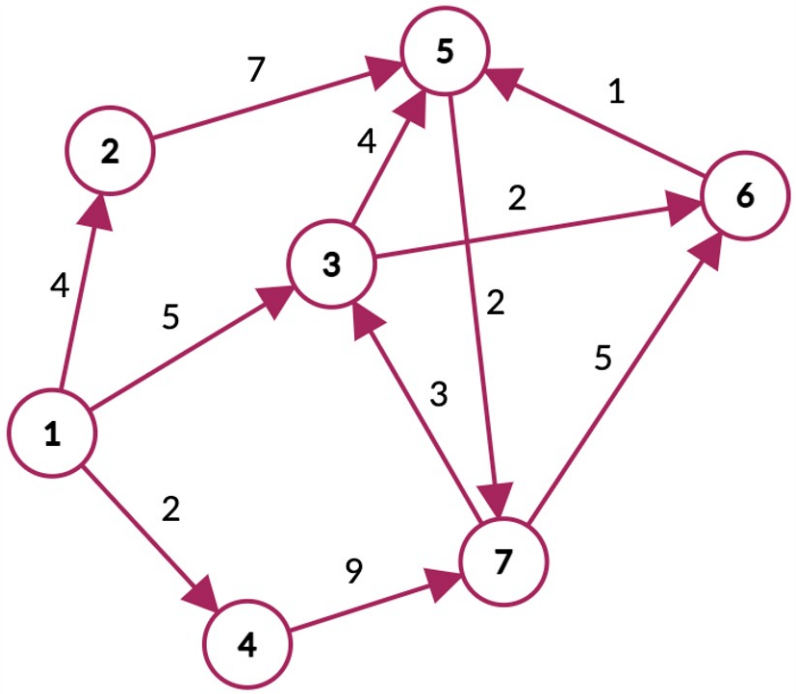
Массив меток вершин:

v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	8	7	10

Алгоритм Дейкстры

1. Каждой вершине v поставим в соответствие метку $d[v]$, хранить которую будем в массиве. установим $d[start] = 0$ и $d[v] = \infty \; \forall \; v \; \neq start$
2. Разделим вершины на два множества: **посещенные** ($V \backslash U$) и **непосещенные** (U) вершины. Изначально $U = V$
3. Выполним $|V|$ итераций. На каждой из них:
    1. Выберем вершину u из множества U с минимальным значением d
    2. Для каждого соседа w вершины u, соединенного ней ребром $u \to w$ с весом e, обновим метку:$d[w] \; =_{min} \; d[u] + e$
    3. Удалим вершину u из множества U

Пример



Непосещенные вершины: $U = \{\cancel{7}\}$

Массив меток вершин:

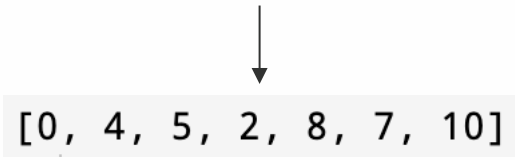
v	1	2	3	4	5	6	7
$d[v]$	0	4	5	2	8	7	10

Алгоритм Дейкстры: реализация

Изучим же наконец-то код реализации данного алгоритма:

```python
def dijkstra(graph):
    v_cnt = len(graph)
    d = [0] + [float('inf')] * (v_cnt - 1)
    u = {*range(v_cnt)}

    while len(u) > 0:
        # Find the vertex with the minimal d-label
        v = -1
        for candidate in u:
            if v == -1 or d[candidate] < d[v]:
                v = candidate

        # Iterate over its adjacent vertices and optimise their labels:
        for w, e in graph[v]:
            d[w] = min(d[w], d[v] + e)

        # v is now visited
        u.remove(v)

    return d
```

```python
test = [
    [(1, 4), (2, 5), (3, 2)],
    [(4, 7)],
    [(4, 4), (5, 2)],
    [(6, 9)],
    [(6, 2)],
    [(4, 1)],
    [(2, 3), (5, 5)]
]

print(dijkstra(test))
```



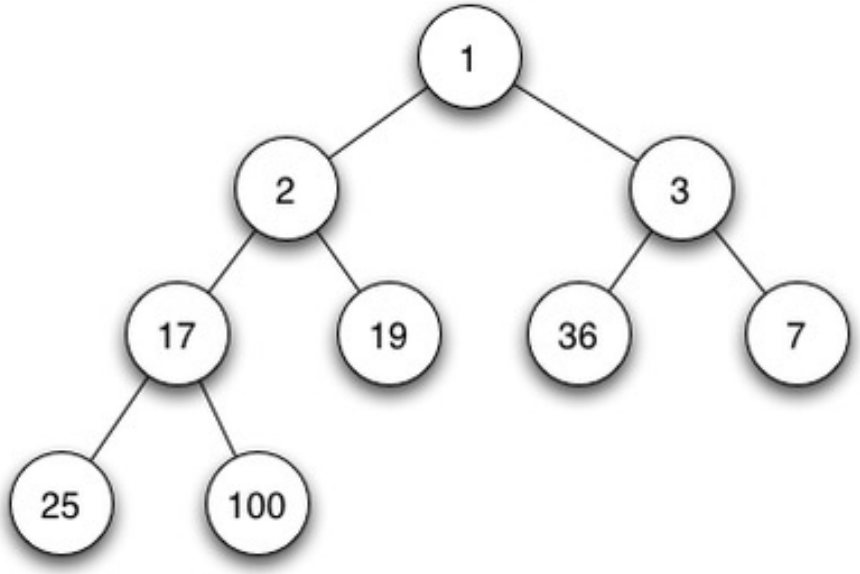
```
[0, 4, 5, 2, 8, 7, 10]
```

Асимптотика данной реализации в среднем случае: $O(|V|^2 + |E|)$

Приоритетная очередь

Для ускорения решения можно использовать специальную структуру данных – приоритетную очередь. Она хранит множество объектов, каждому из которых поставлено в соответствии число – его приоритет и поддерживает операции:

- Получение элемента, имеющего минимальный приоритет
- Добавление нового элемента
- Удаление элемента
- Обновление приоритета элемента



Будучи реализованной с использованием двоичный кучи, асимптотические сложности всех вышеперечисленных операций – $O(logN)$, где N – количество элементов в куче.

Ускорим же наш с использованием этой структуры данных!

Алгоритм Дейкстры: реализация

К сожалению, в Python3 нет встроенной приоритетной кучи, поэтому класс priority_heap придется писать самостоятельно…

```
def dijkstra(graph):
    v_cnt = len(graph)
    d = [0] + [float('inf')] * (v_cnt - 1)
    u = priority_heap({ v: d[v] for v in range(v_cnt) })

    while u.size() > 0:
        # Find the vertex with the minimal d-label
        v = u.min()

        # Iterate over its adjacent vertices and optimise their labels:
        for w, e in graph[v]:
            d[w] = min(d[w], d[v] + e)
            u.update_priority(w, d[w])

        # v is now visited
        u.remove(v)

    return d
```

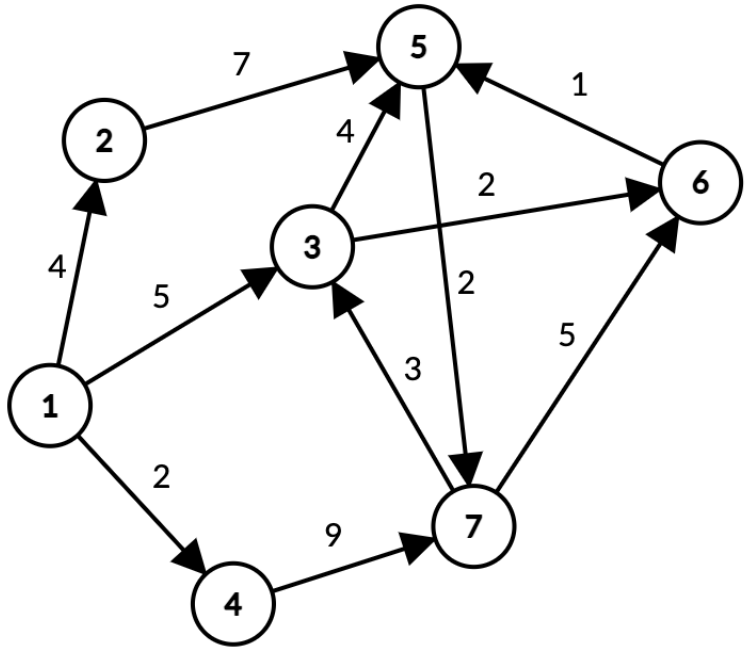
Асимптотика такого решения – $O((|E| + |V|)\log|V|)$

Заметка: Асимптотику просто посчитать, заметив сии факты:

- Операции min и remove вызываются $O(|V|)$ раз
- Операция update_priority вызывается $O(|E|)$ раз в худшем случае
- Для двоичной кучи, все вышеупомянутые операции выполняются за $O(\log N)$, где N в свою очередь пропорционально $|V|$

Задача вторая

Дан **взвешенный ориентированный** граф *G(V, E)* **без ребер отрицательного веса**.
Найти **оптимальный путь** от некоторой вершины *start* графа *G* до вершины *end*.



Алгоритм A*

Применять алгоритм Дейкстры в такой ситуации – неоптимально, ведь вершина *end* может быть совсем недалеко от вершины *start*, а алгоритм Дейкстры никак не учитывает этот фактор при итерировании по графу: действительно, мы никак не оцениваем то, сколько нам **примерно осталось идти** до *end* при выборе оптимальной вершины, поэтому может получиться так, что истинная оптимальная вершина была проигнорирована в угоду другим, располагающимся, однако, близко к *start*.

Применим эту идею: введем функцию $h(v)$, оценивающую то, сколько осталось идти до вершины *end*. Назовем ее **эвристикой**.

Теперь же, в качестве оптимальной вершины будем выбирать не ту, которая ближе к началу, а ту, для которой сумма расстояния от start и оценки расстояния до end минимальна.
Формально: $v = argmin\ f(v)$, где $f(v) = d[v] + h(v)$.

Заметим также, что останавливать алгоритм достаточно при достижении финальной вершины *end*, не обязательно обходить весь граф.

Алгоритм A*: типы эвристик

К сожалению, выбор произвольной эвристической функции не гарантирует нам сохранение корректности модифицированного алгоритма Дейкстры (алгоритма A*). Рано или поздно, искомая вершина будет найдена, однако в общем случае, расстояние может оказаться не кратчайшим.

Утверждение: Чтобы алгоритм A* был оптимален, необходимо, чтобы эвристика была **допустимой**.

Определение: эвристика – допустима, если для любой выбранной вершины ее значение не превышает веса оптимального пути до конечной точки.

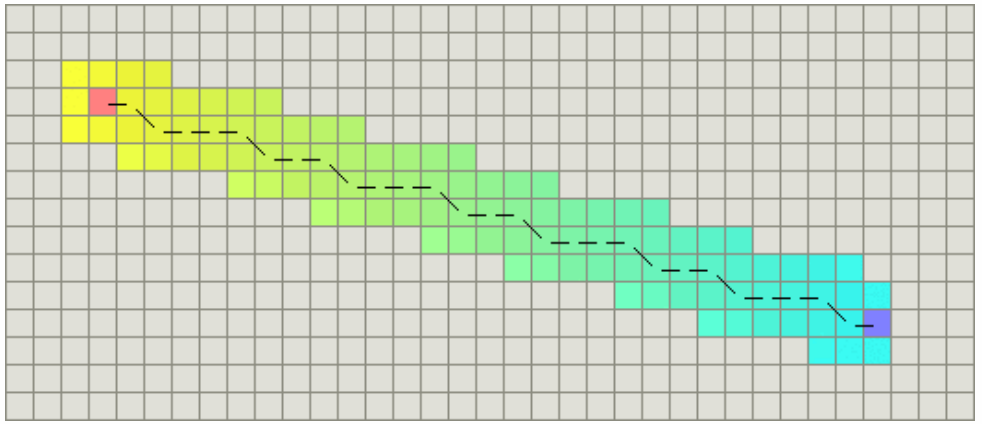
Однако это условие не является достаточным.

Утверждение: алгоритм A* – оптимален, если эвристика **монотонна**.

Определение: эвристика – монотонна, если для любого выбранного ребра $v1 \rightarrow v2$ веса e выполняется условие $h(v_2) - h(v_1) \leq e$, причем $h(end) = 0$.

Алгоритм A*: примеры эвристик

Поведение алгоритма сильно зависит от того, какая эвристика используется. В свою очередь, выбор эвристики зависит от постановки задачи. Часто A* используется для моделирования перемещения по поверхности, покрытой координатной сеткой.



- Если перемещаться можно только в 4-х направлениях, следует применить норму L_1 (Манхэттенское расстояние):

$$h(v) = L_1(v, end) = |v.x - end.x| + |v.y - end.y|$$

- Если перемещаться можно в 8-ми направлениях (добавляются диагонали), следует применить норму L_2 (расстояние Чебышева):

$$h(v) = L\infty(v, end) = \max(|v.x - end.x|, |v.y - end.y|)$$

- Если же перемещение не ограничено сеткой, следует использовать норму того пространства, в котором решается задача (в нашем случае – двумерном):

$$h(v) = L_2(v, end) = \sqrt{|v.x - end.x|^2 + |v.y - end.y|^2}$$

Алгоритм A*: имплементация

Имплементация алгоритма слабо отличается от алгоритма Дейкстры. Рассмотрим ее:

```
class priority_heap:
    # To be implemented...
    def __init__(self, name):
        self.name = name


def a_star(graph, end, h):
    v_cnt = len(graph)
    u = priority_heap({ v: d[v] for v in range(v_cnt) })

    while u.size() > 0:
        v = u.min()

        for w, e in graph[v]:
            d[w] = min(d[w], d[v] + e)
            f = d[w] + h(w)

            if w == end:
                return f

            u.update_priority(w, f)

        u.remove(v)

    return None
```

```
def l1(a, b):
    return abs(a.x - b.x) + abs(a.y - b.y)

def l2(a, b):
    return ((a.x - b.x) ** 2 + (a.y - b.y) ** 2) ** 0.5

def l_inf(a, b):
    return max(abs(a.x - b.x), abs(a.y - b.y))

two_d_grid = [] # define 2-d grid here...
vertex_coordinates = [] # map each vertex to its coordinates...
end = 0 # input goal vertex idx here...
h_inf = lambda v_idx: l_inf(vertex_coordinates[v_idx], end)

print(a_star(test, goal, h_inf))
```

Алгоритм IDA*

Получив, наконец, реализацию алгоритма A*, можно улучшить ее еще сильнее.

Алгоритм A* достаточно быстрый, однако во многих случаях он находит не самый оптимальный путь.

Идея: ограничим дальность (глубину) поиска вершины *end*, произведем поиск, и если не найдем вершину, увеличим глубину и повторим поиск еще раз (и так до тех пор, пока не найдем вершину).

```
path                current search path (acts like a stack)
node                current node (last node in current path)
g                   the cost to reach current node
f                   estimated cost of the cheapest path (root..node..goal)
h(node)             estimated cost of the cheapest path (node..goal)
cost(node, succ)    step cost function
is_goal(node)       goal test
successors(node)    node expanding function, expand nodes ordered by g + h(node)
ida_star(root)      return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
    bound := h(root)
    path := [root]
    loop
        t := search(path, 0, bound)
        if t = FOUND then return (path, bound)
        if t = ∞ then return NOT_FOUND
        bound := t
    end loop
end procedure

function search(path, g, bound)
    node := path.last
    f := g + h(node)
    if f > bound then return f
    if is_goal(node) then return FOUND
    min := ∞
    for succ in successors(node) do
        if succ not in path then
            path.push(succ)
            t := search(path, g + cost(node, succ), bound)
            if t = FOUND then return FOUND
            if t < min then min := t
            path.pop()
        end if
    end for
    return min
end function
```

Вывод

Алгоритмы, рассмотренные сегодня – классические в решении задач поиска оптимального пути в графах.

Они применяются во многих задах: от поиска оптимального маршрута на карте до нахождения лучшего хода в шахматной партии.

HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Лекція

Имплементація и анализъ
Алгориѳм IDA*

Благодарю васъ за вниманіе,

Желаю всѣмъ хорошаго дня!

Леладзе Константинъ Григорьевичъ