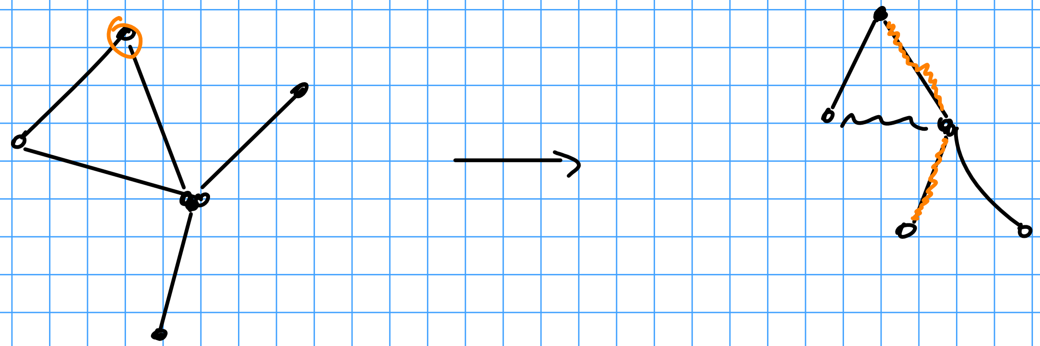
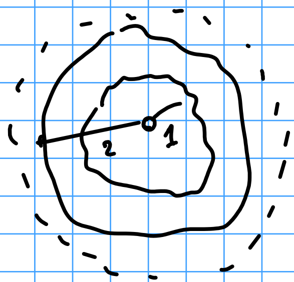
Кратчайшие пути





Идея алгоритма

Поиск в ширину (breadwidth search)

```
BFS((V, E), v)
    for u in V:
        dist[u] = +∞
        prev[u] = 0
    dist[v] = 0
    Q = Queue({v})
    while Q.size() ≥ 1:
        u = Q.dequeue()
        for (u, w) ∈ E:
            if dist[w] == +∞:
                dist[w] = dist[u] + 1
                prev[w] = u
                Q.enqueue(w)
```

$O(V + E)$

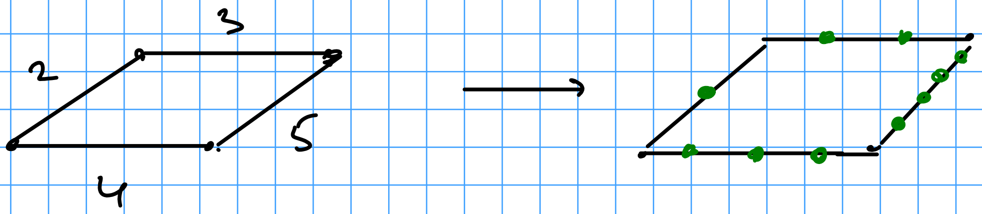
Утв (корректность). $\forall u$ $dist[u] = dist(v, u)$.

Д. База: $dist[u] = 0$

Предположение: где $dist[u] \le k$ – верно

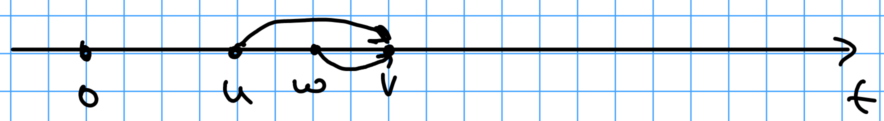
Переход: где $dist[u] = k+1$ – верно ◁

Кратчайшие пути во взвешенных графах



Можно добавить фиктивные вершины

- нецелые веса
- большие веса
- отрицательные веса



$t[v] = dist[v]$

Алгоритм Дейкстры (нет отрицательных рёбер)

АТД Очередь с приоритетом

- make_priority_queue() $O(n)$
- delete_min()
- insert(v, d)
- decrease_key(v, d)

Реализации:

На массиве (delete_min за $O(n)$, остальные $O(1)$)

На куче ($\log(n)$)

```
Dijkstra((V, E), v)
    for u ∈ V:
        dist[u] = +∞
        prev[u] = 0
    dist[v] = 0
```

```
P = make_priority_queue({(v, 0)})
while P.size() ≥ 1:
    (u, d) = P.delete_min()                  R ⇝ S
    for (u, w) ∈ E:
        if dist[w] == +∞:
            dist[w] = dist[u] + w(u, w)
            prev[w] = u
            P.insert((w, dist[w]))           T ⇝ R
        else dist[w] > dist[u] + w(u, w):
            dist[w] = dist[u] + w(u, w)
            prev[w] = u
            P.decrease_key((w, dist[w]))
```

__Утв__: Алгоритм Дейкстры корректен.

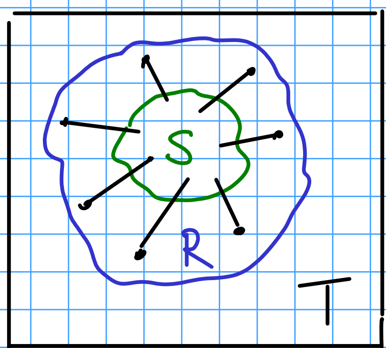
Алгоритм Дейкстры разбивает V на три множества:

S — уже обработанные вершины

R — вершины в очереди

T — вершины $dist = +\infty$

$T \rightsquigarrow R \rightsquigarrow S$



Если вершина u переходит из R в S $\Rightarrow$ $dist[u] = dist(v, u)$

▷ 1. База: $S = \{v\}$

2. Предположение:] для всех вершин в S расстояние вычислено корректно

3. Пусть на этом шаге мы ∠ вершину u: $S = S \cup \{u\}$

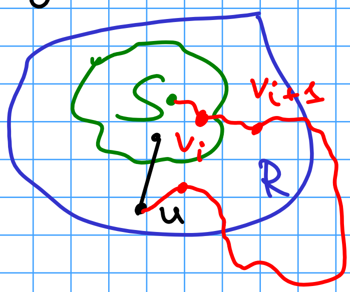
Покажем, что $dist[u] = dist(v, u)$.

От обратного:

$dist[u] > dist(v, u)$

$\exists$ путь: $v, v_1, v_2, \ldots, v_k, u$ — кратчайший

$\exists v_i$ — последняя вершина на это пути из S



$dist[v_i] \le dist[u]$

т.е. $v_{i+1} \in R$

$\left[\begin{array}{l} dist[v_{i+1}] < dist[u] \\ \text{противоречие с priority-queue} \\ dist[v_{i+1}] = dist[u] \\ \text{противоречие с тем, что } \exists \text{ более короткий путь} \end{array} \right.$

$dist[v_{i+1}] \le dist[v_i] + w(v_i, v_{i+1}) \le$

$\le$ длина пути $(v, v_1 \ldots v_k, u) < dist[u]$ $\triangleleft$

Сложность:

1 × make_priority_queue()

V × insert

V × delete_min

E × decrease_key

I на массиве:

Сложность: $O(V) + V \cdot O(1) + \underbrace{V \cdot O(V)}_{}^{\text{delete\_min}} + E \cdot O(1) =$

$= O(V^2 + E)$

Хорошо для плотных графов

II на куче:

Сложность: $O(V) + V \cdot O(\log V) + V \cdot O(\log V) + E \cdot O(\log V) =$

$= O((V + E) \cdot \log V)$

Хорошо для разреженных графов

III d-ичная куча

Сложность операций:

insert	$O(\log_d V)$
decrease_key	$O(\log_d V)$
delete_min	$O(d \cdot \log_d V)$

Сложность: $O(V) + V \cdot O(\log_d V) + V \cdot O(d \cdot \log_d V) +$
$+ E \cdot O(\log_d V) = O((\underline{Vd} + \underline{E}) \cdot \log_d V)$

$d \uparrow \Rightarrow \log_d V \downarrow$

$$d \approx \frac{E}{V}$$

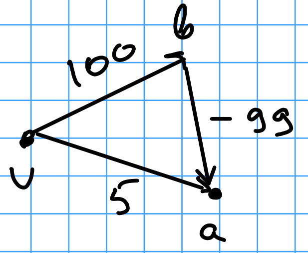
$\measuredangle$ 2 случая:

$E \sim V^2 \Rightarrow d \sim V \Rightarrow$ сложность $O(V^2)$

$E \sim V \Rightarrow d = O(1) \Rightarrow$ сложность $O((V+E)\log V)$

$E \sim V^{1.5} \Rightarrow d \sim \sqrt{V} \Rightarrow$ сложность $O(E \cdot \log_{\sqrt{V}} V) =$
$= O(E)$

Отрицательные ребра



Замечание: мы ищем пути от одной вершины до всех.