Алгоритмы поиска пути в графе

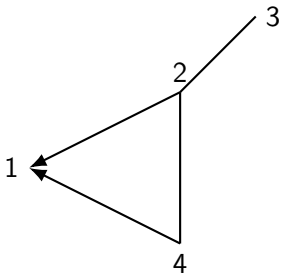
Чубий Савва, 9ФМ

Лицей НИУ ВШЭ

Очень краткое введение в оценку ассимптитики

$$f(x) = O(g(x)) \Leftrightarrow \exists C = const : \ \forall x : \frac{|f(x)|}{|g(x)|} \leq C$$

Способы хранения графа



- Матрица смежности

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

- Список смежности

  1: —

  2: 1 3 4

  3: 2

  4: 1 2

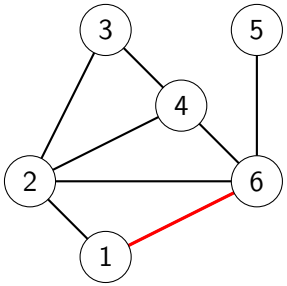
- Скисок ребер

  2–1; 2–3; 2–4; 3–2; 4–1; 4–2

Обозначения

- G — матрица смежности
- g — список смежности
- E — список ребер
- $n = |V|$ — количество вершин
- $m = |E|$ — количество ребер
- w — массив весов ребер

Depth-first search



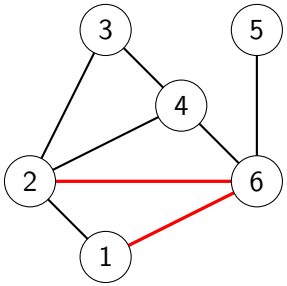
$O(n + m)$

```
doDfs(n: int, g: int[][], s: int)
    used = [0] * n

    dfs(u: int)
        used[u] = 1
        for v ∈ g[u]
            dfs(v)

    dfs(s)
```

Depth-first search



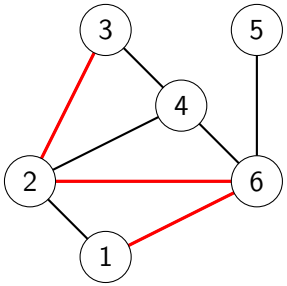
$O(n + m)$

```
doDfs(n: int, g: int[][], s: int)
    used = [0] * n

    dfs(u: int)
        used[u] = 1
        for v ∈ g[u]
            dfs(v)

    dfs(s)
```

Depth-first search



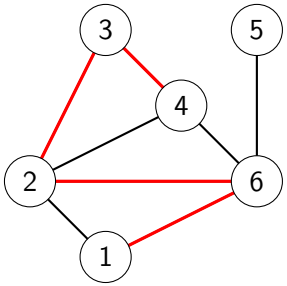
$O(n + m)$

```
doDfs(n: int, g: int[][], s: int)
    used = [0] * n

    dfs(u: int)
        used[u] = 1
        for v ∈ g[u]
            dfs(v)

    dfs(s)
```

Depth-first search



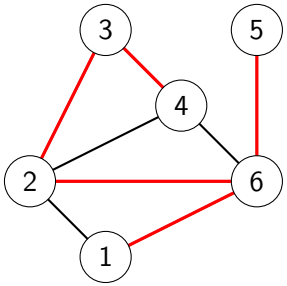
$O(n + m)$

```
doDfs(n: int, g: int[][], s: int)
    used = [0] * n

    dfs(u: int)
        used[u] = 1
        for v ∈ g[u]
            dfs(v)

    dfs(s)
```

Depth-first search



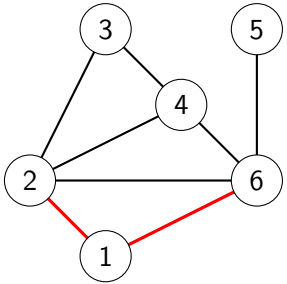
$O(n + m)$

```
doDfs(n: int, g: int[][], s: int)
    used = [0] * n

    dfs(u: int)
        used[u] = 1
        for v ∈ g[u]
            dfs(v)

    dfs(s)
```

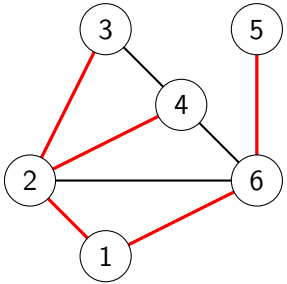
Breadth-first search



$O(n+m)$

```
doBfs(n: int, g: int[][], s: int)
    dist = [−1] * n
    queue q
    q.push(s)
    dist[u] = 0
    while q ≠ ∅
        u = q.pop()
        for v ∈ g[u]
            if dist[v] == −1
                dist[v] = dist[u] + 1
                q.push(v)
```

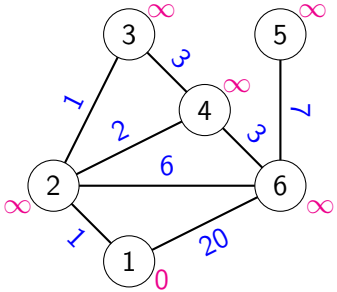
Breadth-first search



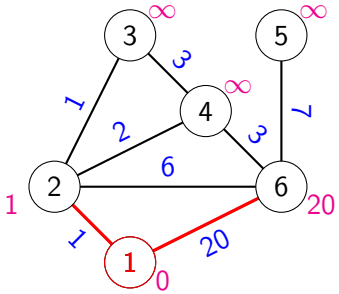
$O(n+m)$

```
doBfs(n: int, g: int[][], s: int)
    dist = [−1] * n
    queue q
    q.push(s)
    dist[u] = 0
    while q ≠ ∅
        u = q.pop()
        for v ∈ g[u]
            if dist[v] == −1
                dist[v] = dist[u] + 1
                q.push(v)
```

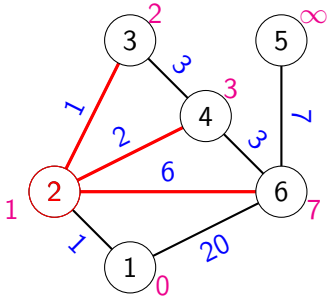
Dijkstra's algorithm



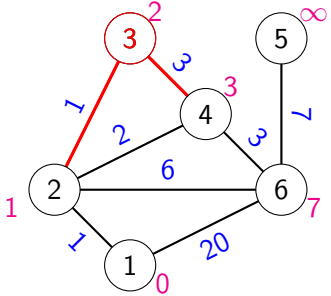
Dijkstra's algorithm



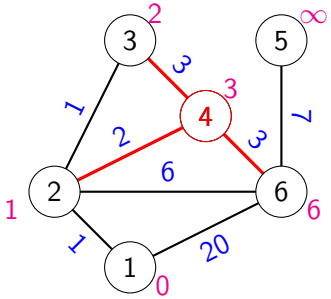
Dijkstra's algorithm



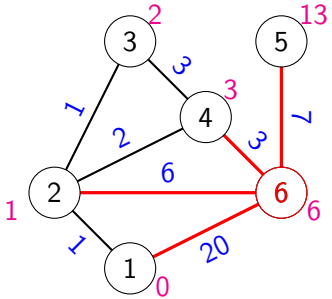
Dijkstra's algorithm



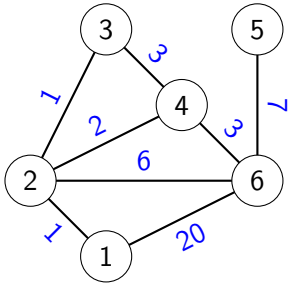
Dijkstra's algorithm



Dijkstra's algorithm



Dijkstra's algorithm

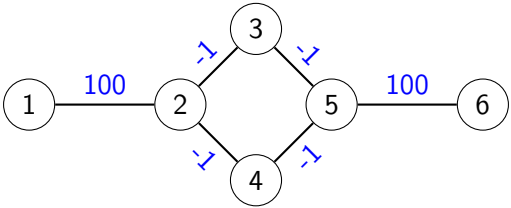


Dijkstra's algorithm

$O(m + n \log n)$

```
doDijkstra(n: int, g: int[][], w: int[][], s: int)
    set<int, int> not_visited
    dist = [inf] * n
    dist[s] = 0
    for u = 0...n-1
        not_visited.insert(dist[u], u)
    while s ≠ ∅
        d, u = not_visited.pop()
        for v ∈ g[u]
            if dist[v] > dist[u] + w[u][v]
                not_visited.erase(dist[v], v)
                dist[v] = dist[u] + w[u][v]
                not_visited.insert(dist[v], v)
```

Отрицательный цикл



Ford-Bellman algorithm

$O(mn)$

```
doFord-Bellman(n: int, E: (int, int)[], w: int[][], s: int)
    d = [inf] * n
    d[s] = 0
    for i=0...n-1
        for (u, v) ∈ E
            if d[v] > d[u] + w[u][v]
                d[v] = d[u] + w[u][v]
```

Floyd algorithm

$O(n^3)$

```
doFloyd(n: int, G: int[][], w: int[][])
    d = w
    for k = 0...n-1
        for u = 0...n-1
            for v = 0...n-1
                if d[u][v] > d[u][i] + d[i][v]
                    d[u][v] = d[u][i] + d[i][v]
```