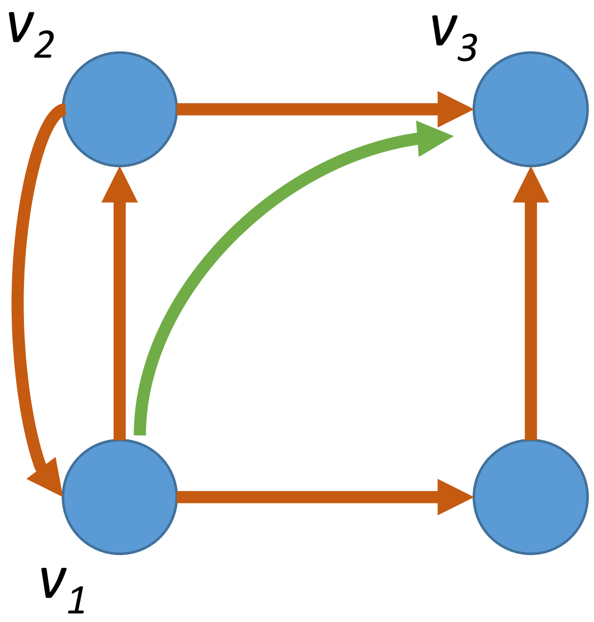
Алгоритмы
и структуры данных

Деев Богдан Юльевич
Почта: deevbogdanyi@yandex.ru
Телеграм: @BogdanDeev

Алгоритмы на графах. Пути

Путь – последовательность вершин:

$p=(v_1, v_2, \ldots, v_k)$, где $(v_i, v_{i+1}) \in E$ для всех $1 \leq i < k$.

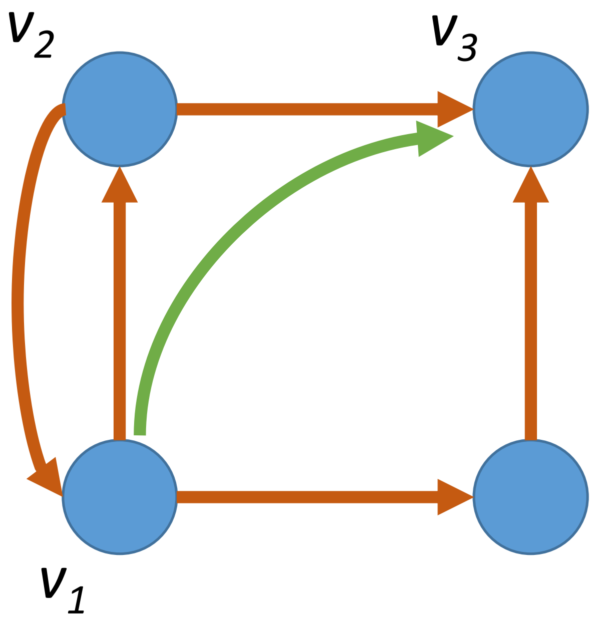


$p = (v_1, v_2, v_3)$

$l(p) = length$
$\delta(u, v) = length\ of\ shortest\ path$

Алгоритмы на графах. Пути

Простой путь – путь без повторений вершин



$p = (v_1, v_2, v_3)$

~~$p = (v_1, v_2, v_1, v_2, v_3)$~~

$l(p) = length$
$\delta(u, v) = length\ of\ shortest\ path$

Алгоритмы на графах. Задачи по поиску пути

Single_Pair_Reachability (G, s, t);

Есть ли пусть в G из s до t?

Single_Pair_Shortest_Path(G, s, t);

Возвращает кратчайшее расстояние в G из s до t.

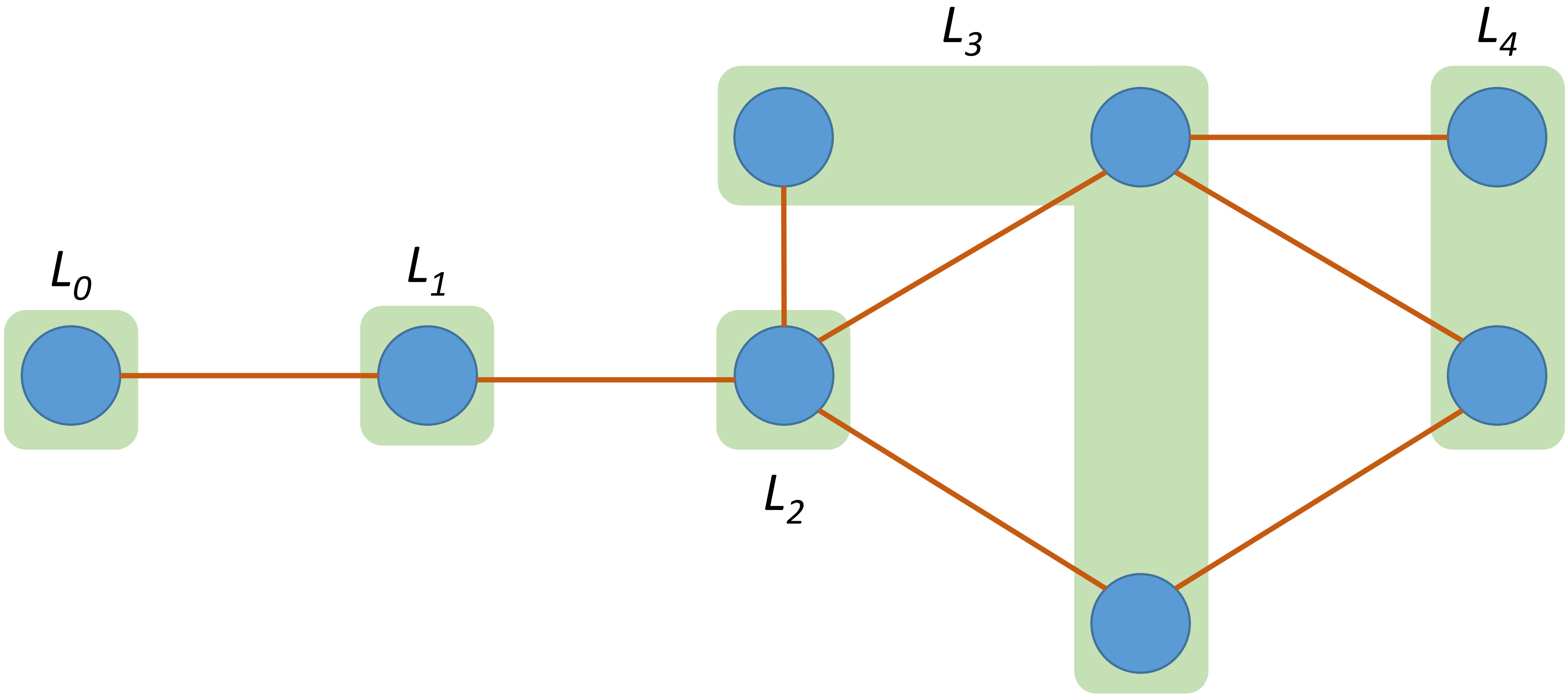
Single_Source_Shortest_Paths(G, s);

Возвращает кратчайшие расстояния из s до всех вершин и дерево кратчайших путей.

В порядке увеличения сложности

Алгоритмы на графах. Level Set

$$L_k = \{v \in V : \delta(s, v) = k\}$$



Алгоритмы на графах. Задачи по поиску пути

BFS

DFS

Breadth-First Search (Поиск в ширину)

- Базовый случай: $(i = 1)$: $L_0 = \{s\}$, $\delta(s, s)=0$, $P(s)= None$
- Шаг индукции для расчета L_i:

  для каждой вершины u в L_{i-1}:

    для каждой вершины $v \in Adj(u)$, которая не встречается в любом L_j для $j<i$:

      добавить v в L_i, установить $\delta(s, v)= i$ и $P(v)= u$

- Повторять вычисление L_i из L_j для $j<i$ для увеличивающихся i, до тех пор пока L_i не пустое множество
- Установить $\delta(s, v)= \infty$ для всех $v \in V$ у которых $\delta(s, v)$ не определен

Breadth-First Search (Поиск в ширину)

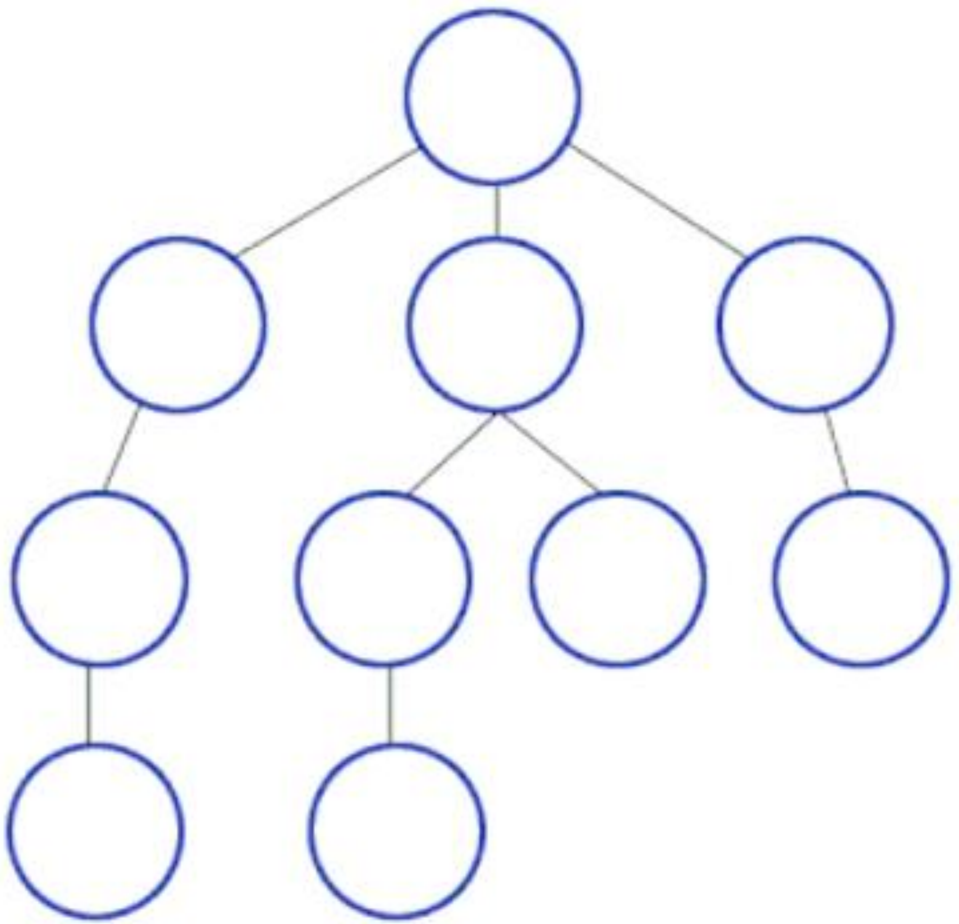
```
def bfs(Adj, s):                              # Adj: adjacency list, s: starting vertex
    parent = [None for v in Adj]              # O(V) (use hash if unlabeled)
    parent[s] = s                             # O(1) root
    level = [[s]]                             # O(1) initialize levels
    while 0 < len(level[-1]):                 # O(?) last level contains vertices
        level.append([])                      # O(1) amortized, make new level
        for u in level[-2]:                   # O(?) loop over last full level
            for v in Adj[u]:                  # O(Adj[u]) loop over neighbors
                if parent[v] is None:         # O(1) parent not yet assigned
                    parent[v] = u             # O(1) assign parent from level[-2]
                    level[-1].append(v)       # O(1) amortized, add to border
    return parent
```

“Линейное время”

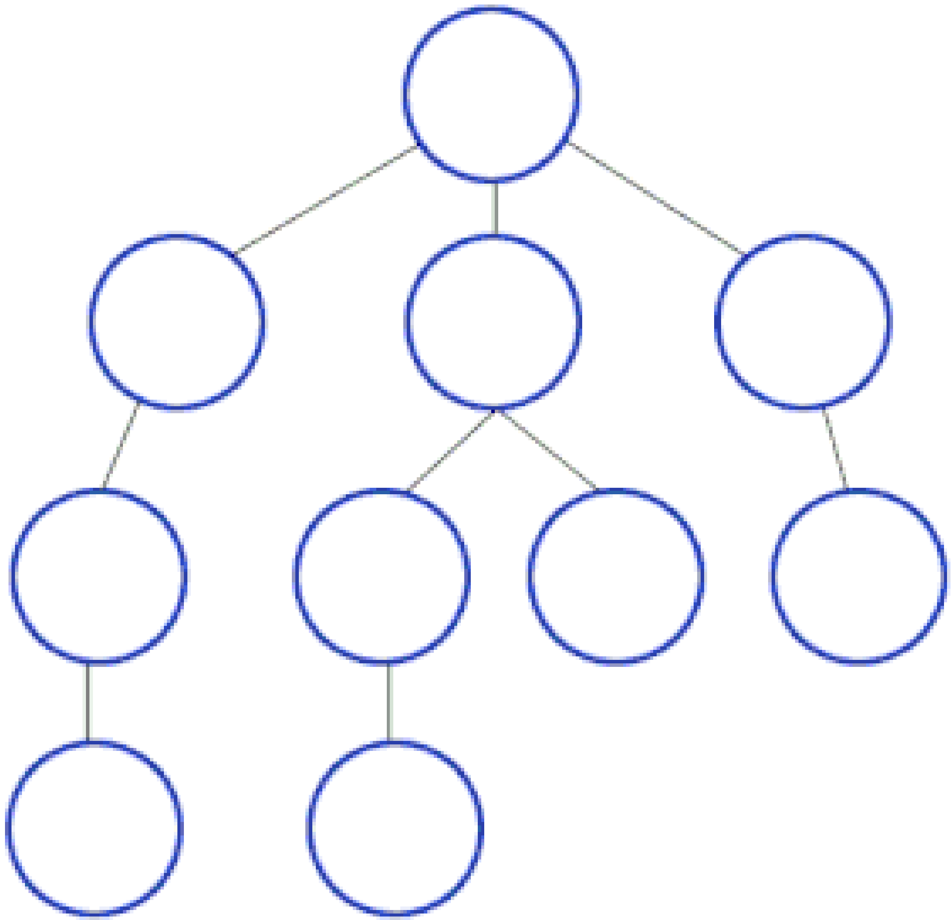
$$O(|V| + |E|)$$

Относительно входных данных

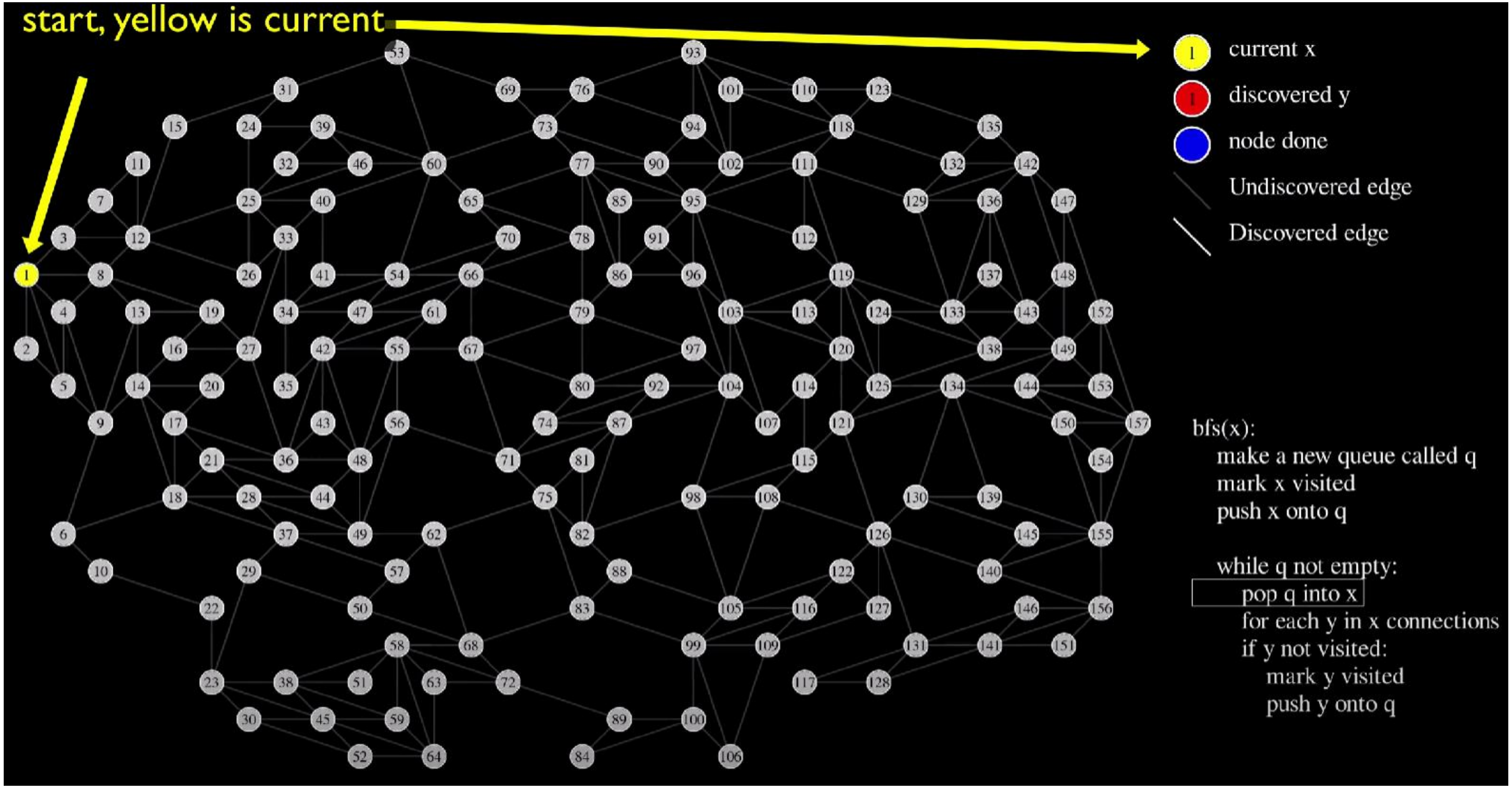
Breadth-First Search (Поиск в ширину)



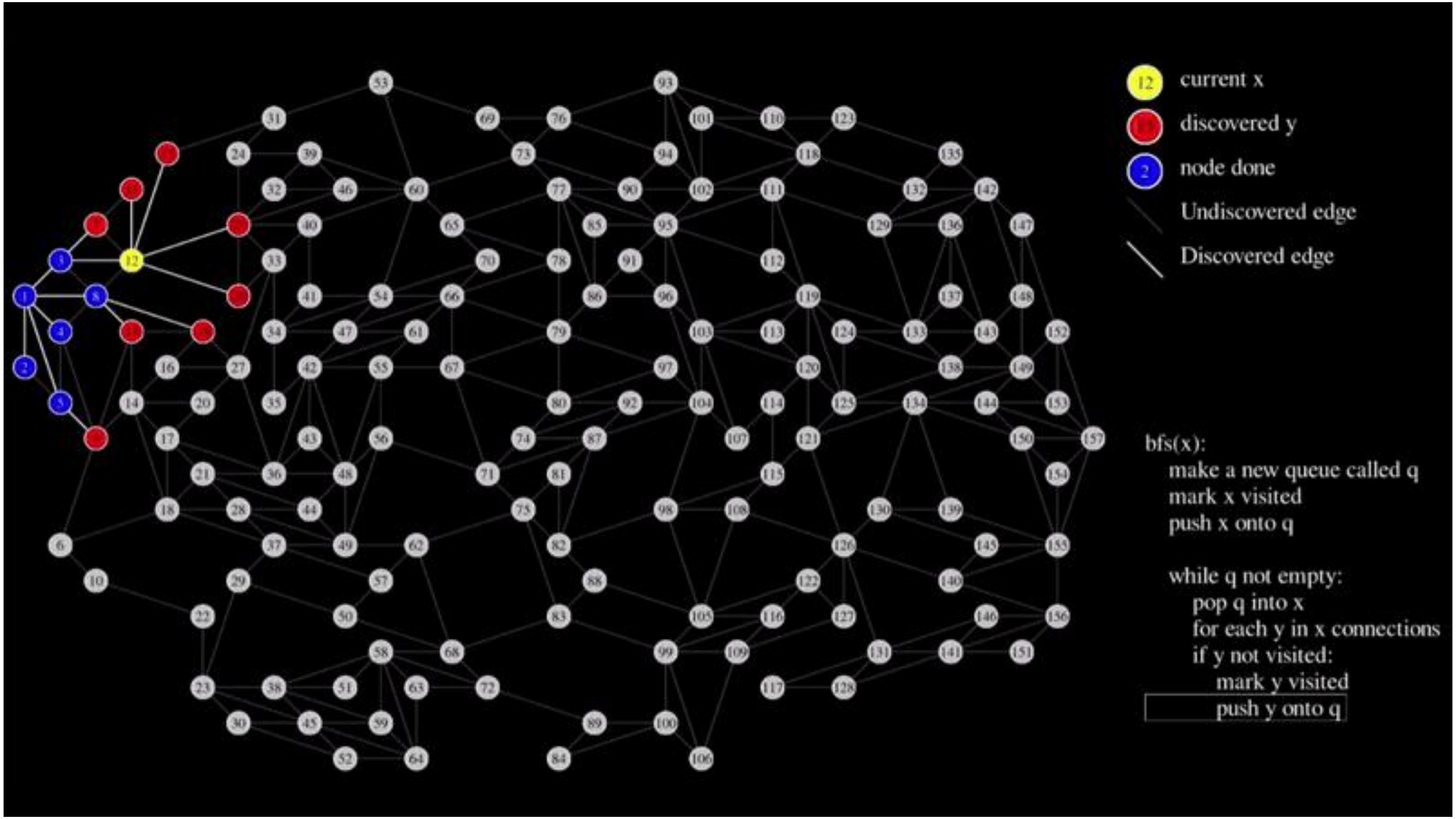
Breadth-First Search (Поиск в ширину)



Breadth-First Search (Поиск в ширину)



Breadth-First Search (Поиск в ширину)



Алгоритмы на графах. Поиск в ширину

Single_Pair_Reachability (G, s, t);

Есть ли пусть в G из s до t?

Single_Pair_Shortest_Path(G, s, t);

Возвращает кратчайшее расстояние в G из s до t.

Single_Source_Shortest_Paths(G, s);

Возвращает кратчайшие расстояния из s до всех вершин и дерево кратчайших путей.

Алгоритмы на графах. Поиск в глубину

Single_Pair_Reachability (G, s, t);

Есть ли пусть в G из s до t?

Single_Pair_Shortest_Path(G, s, t);

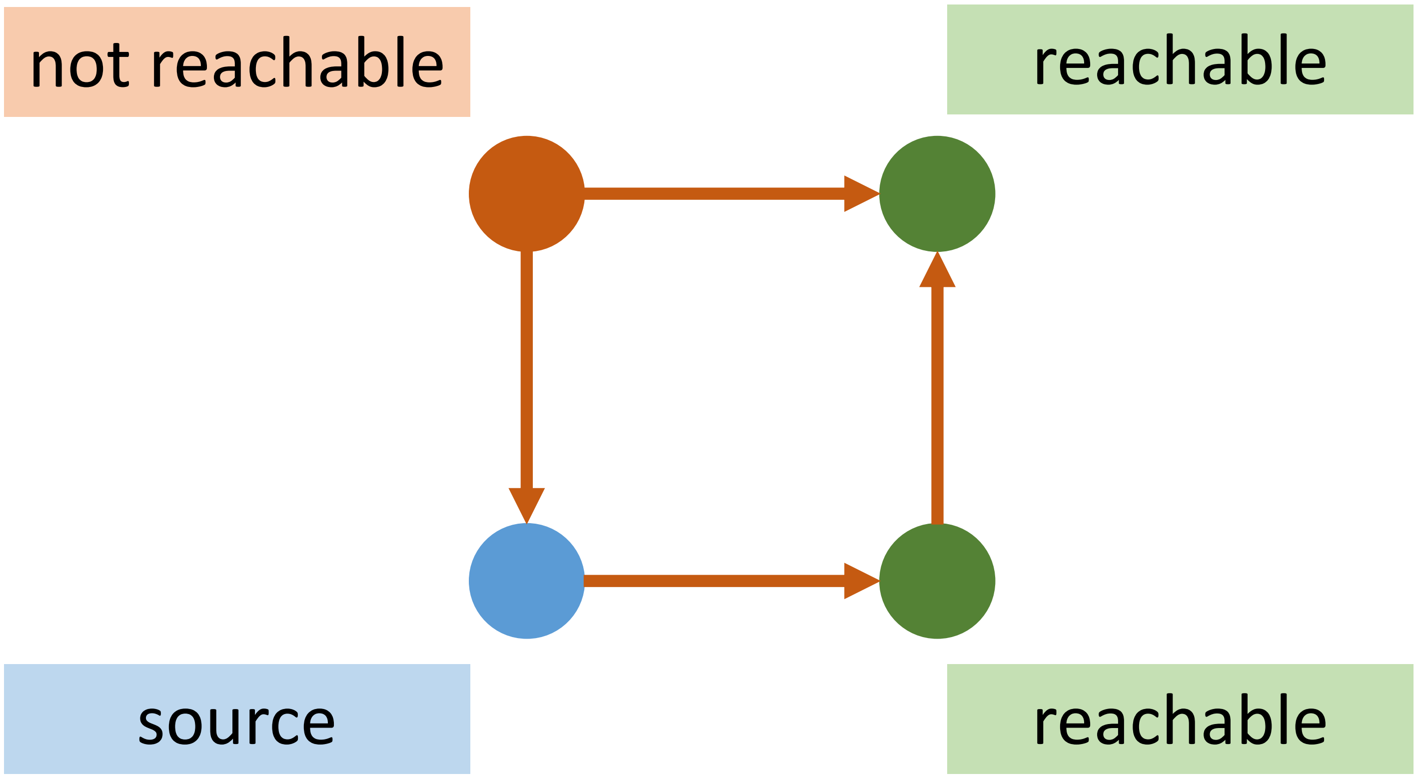
Возвращает кратчайшее расстояние в G из s до t.

Single_Source_Shortest_Paths(G, s);

Возвращает кратчайшие расстояния из s до всех вершин и дерево кратчайших путей.

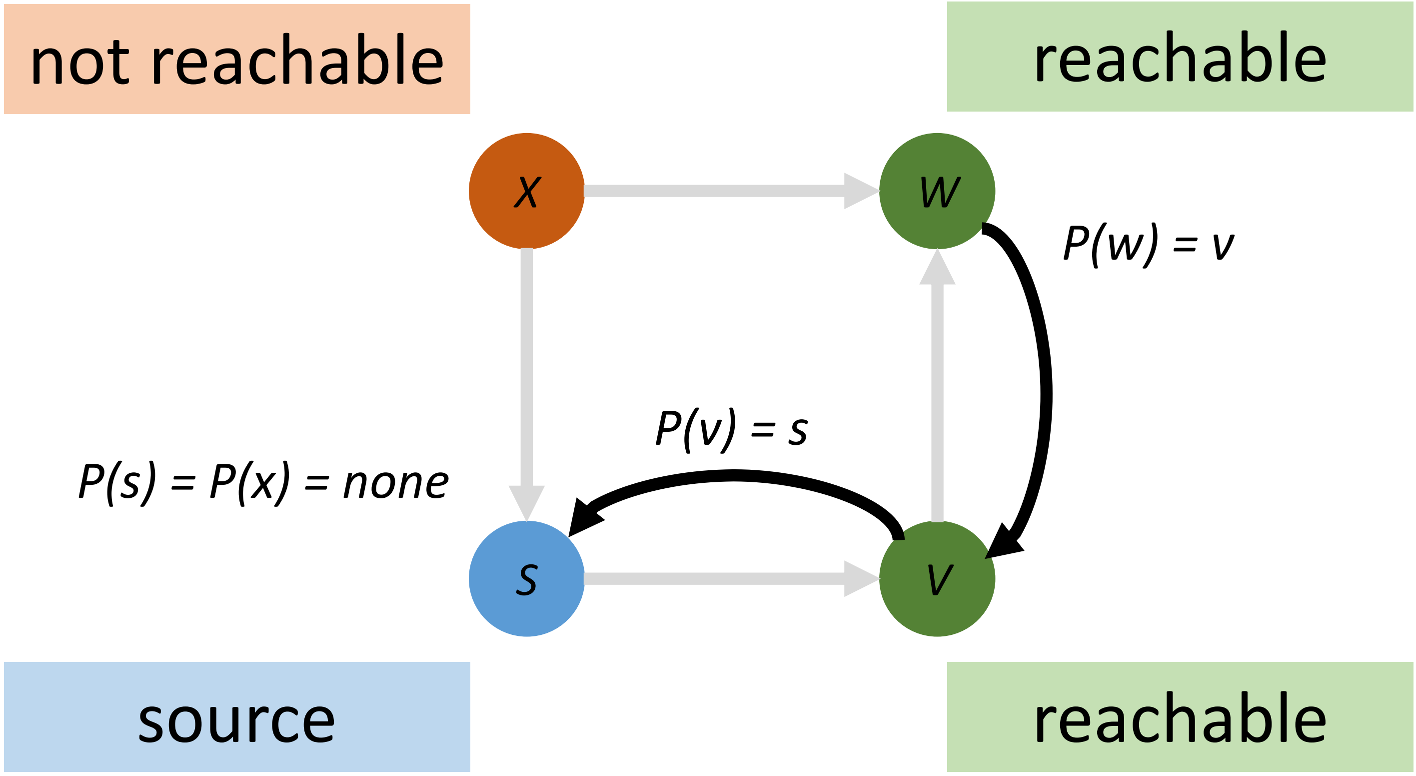
Алгоритмы на графах. *Reachability*

Single source reachability



Алгоритмы на графах. *Parent tree*

Single source reachability



Как дерево кратчайших путей!

Можно ли лучше “Линейного времени”?

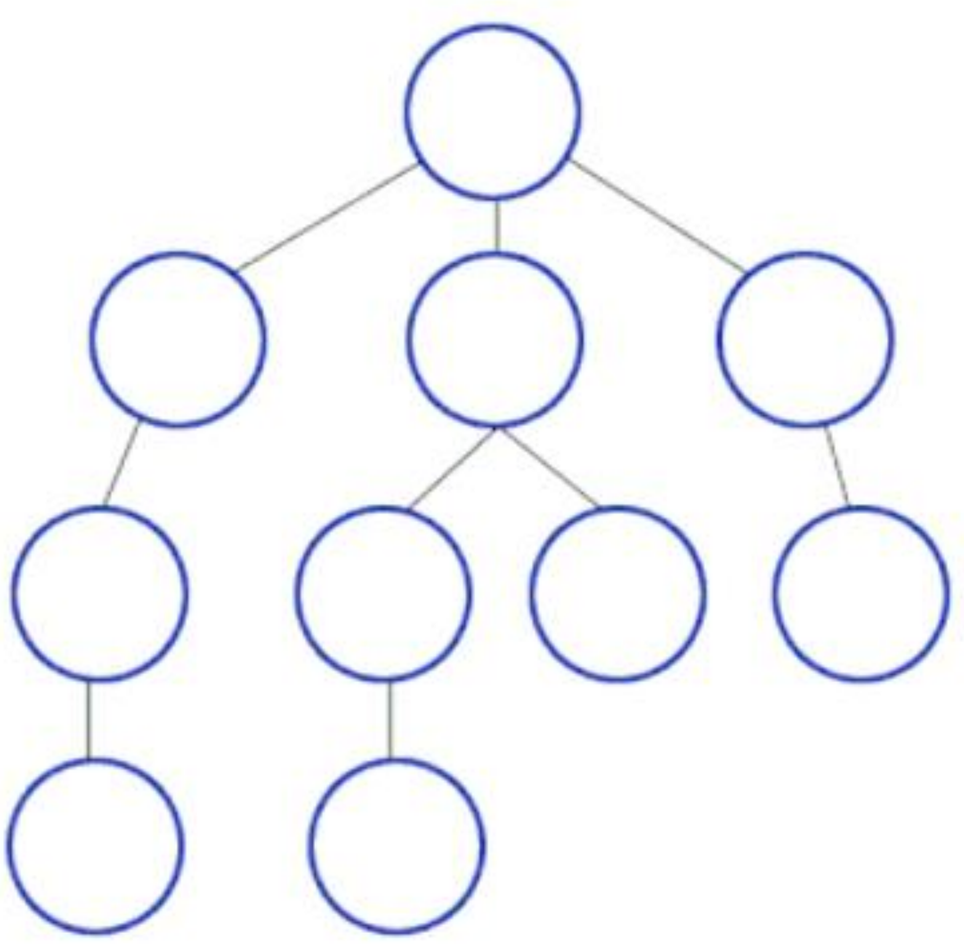
$$O(|V| + |E|)$$

Алгоритмы на графах. Поиск в глубину

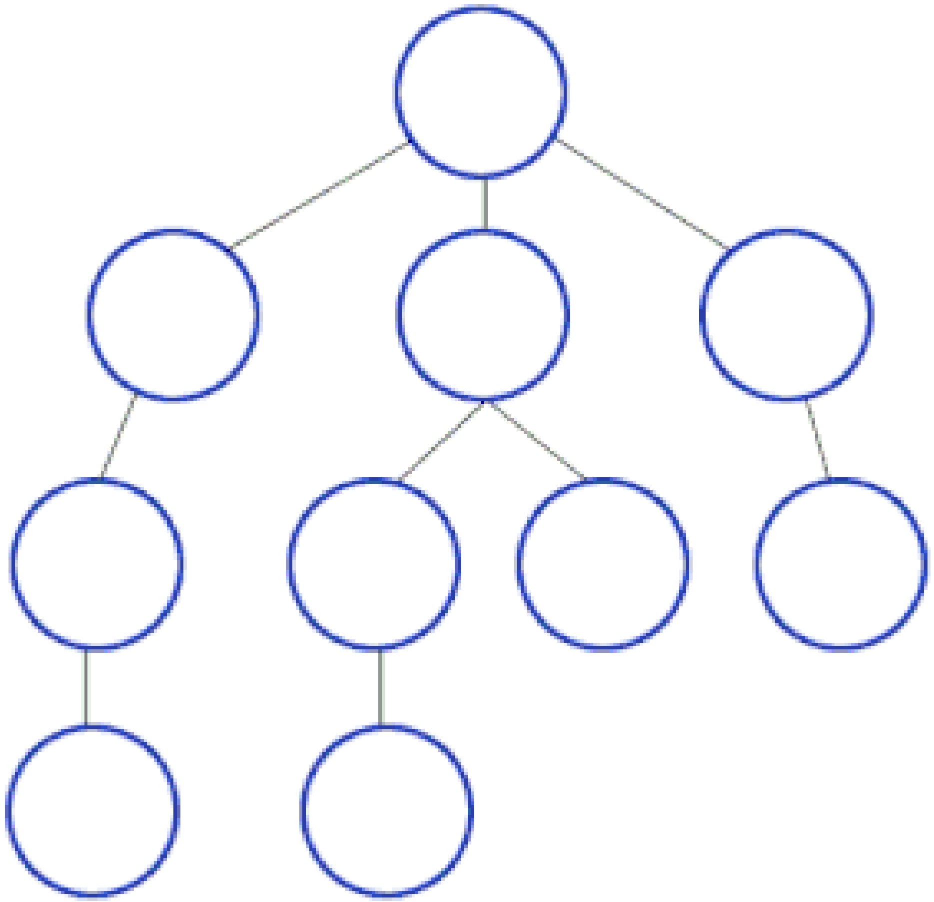
Идея: рекурсивно пройтись по исходящим соседям, не посещая вершину дважды.

Иначе: идти по любому пути до тех пор, пока не «застряли», возвращаться назад до тех пор, пока не найден новый путь для прохода.

Depth-first search (Поиск в глубину)



Depth-first search (Поиск в глубину)



Алгоритмы на графах. Поиск в глубину

$P(s) = None$, затем выполнить $visit(s)$

visit(u):

    для каждой $v \in Adj(u)$ которая не встречается в P:

        установить $P(v) = u$ и рекурсивно вызвать $visit(v)$

    выходим, посещая вершину u

Алгоритмы на графах. Поиск в глубину

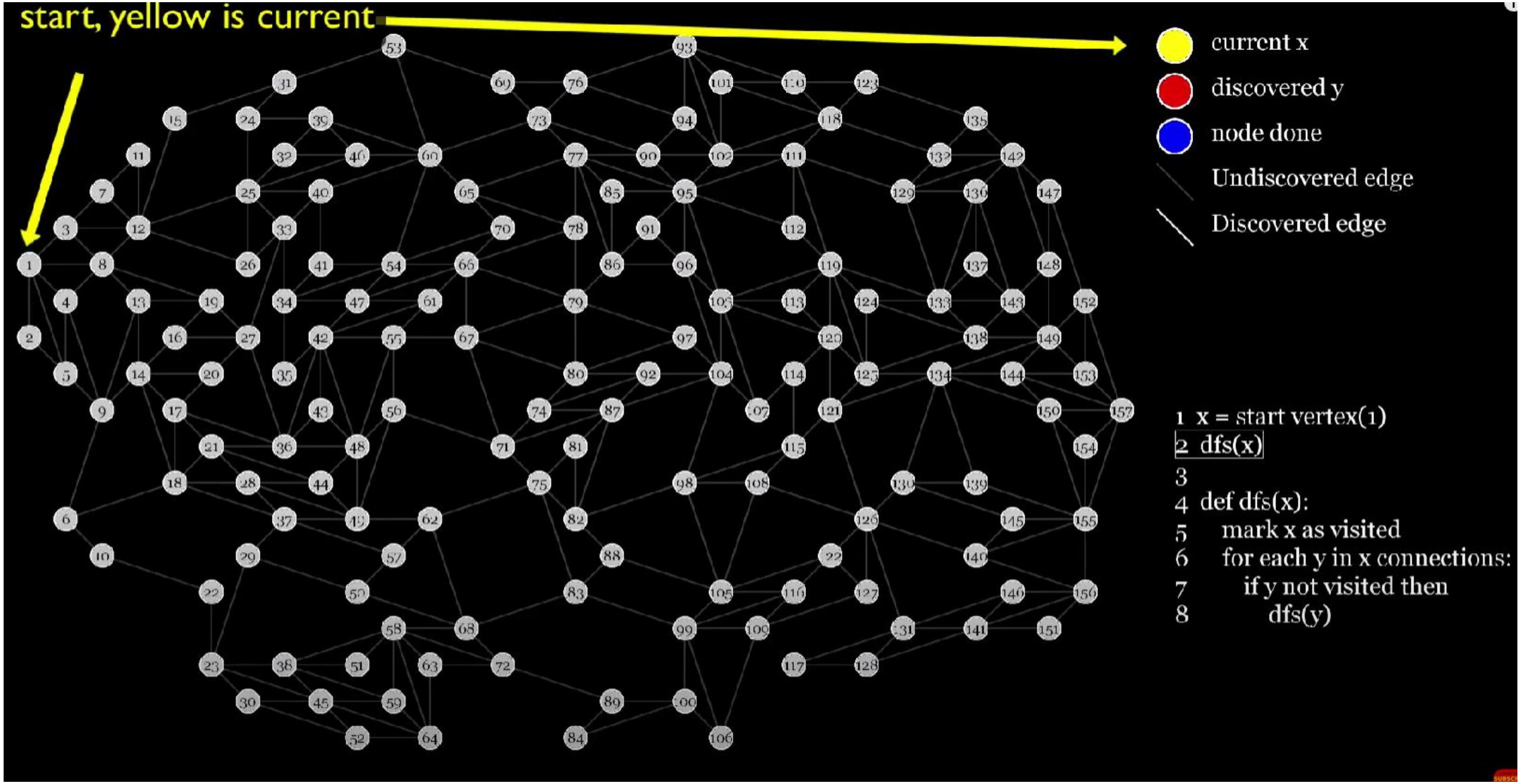
```
P(s) = None, visit(s)

visit(u):
    for every v ∈ Adj(u):
        if P(v) = None:
            Set P(v) = u
            Call visit(v)
```

“Линейное время”

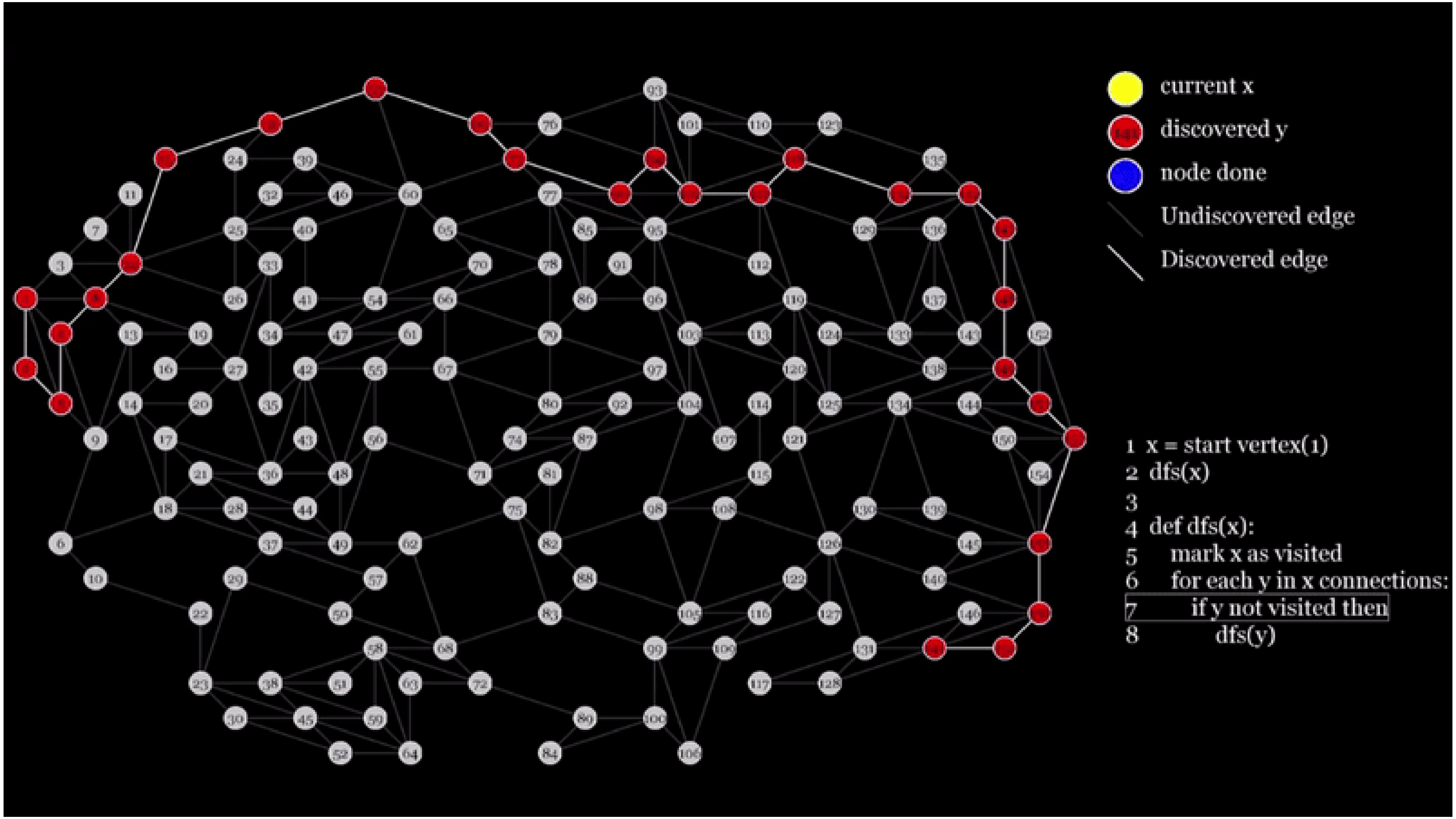
$$O(|E|)$$

Относительно количества ребер

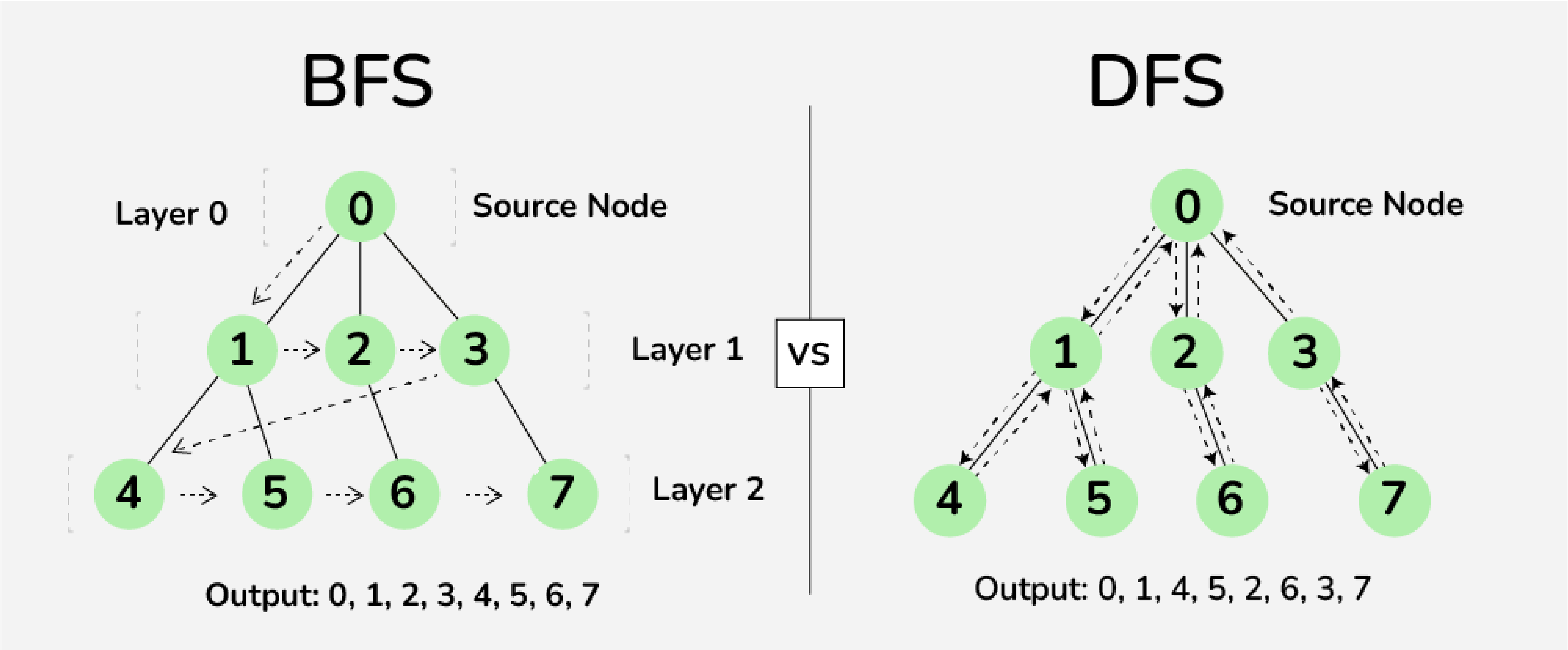
Depth-first search (Поиск в глубину)



Depth-first search (Поиск в глубину)



Алгоритмы на графах. BFS/DFS



Алгоритмы на графах. Что дальше?

- Связность графа (Full DFS)
- Топологическая сортировка
- Directed Acyclic Graph (DAG)
- Обнаружение циклов
- Разбор задач
- Контест