

Метод поиска в глубину

Лекция 5

Поиск в глубину (*Depth-first search, DFS*)

Пусть задан граф $G = (V, E)$.

Алгоритм поиска описывается следующим образом:

для каждой непройденной вершины необходимо найти все непройденные смежные вершины и повторить поиск для них.

Пусть в начальный момент времени все вершины окрашены в *белый* цвет.

1. Из множества всех *белых* вершин выберем любую вершину: v_1 .
2. Выполним для нее процедуру Поиск(v_1).
3. Перекрасим ее в *черный* цвет.

Повторяем шаги 1-3 до тех пор, пока множество *белых* вершин не пусто.

Процедура Поиск(u)

Поиск (u)

```
{
    цвет [ $u$ ] ← серый;
     $d[u] = time++$ ; // время входа в вершину,
                    // порядковый глубинный номер вершины
    для  $\forall v \in \text{смежные}(u)$  выполнить
    {
        если (цвет[ $v$ ] = белый) то
        {
            отец [ $v$ ] ←  $u$ ;
            Поиск( $v$ );
        }
    }
    цвет [ $u$ ] ← чёрный;
     $f[u] \leftarrow time++$ ; // время выхода из вершины
}
```

Процедура Поиск_в_графе

```
Поиск_в_графе ()
{
  для  $\forall u \in V$  выполнить
  {
    цвет [u]  $\leftarrow$  белый;
    отец [u]  $\leftarrow$  NULL;
  }
  time  $\leftarrow$  0;
  для  $\forall u \in V$  выполнить
    если (цвет [u] = белый) то
      Поиск (u) ;
}
```

Анализ

Общее число операций при выполнении

Поиск_в_графе:

$$O(|V|)$$

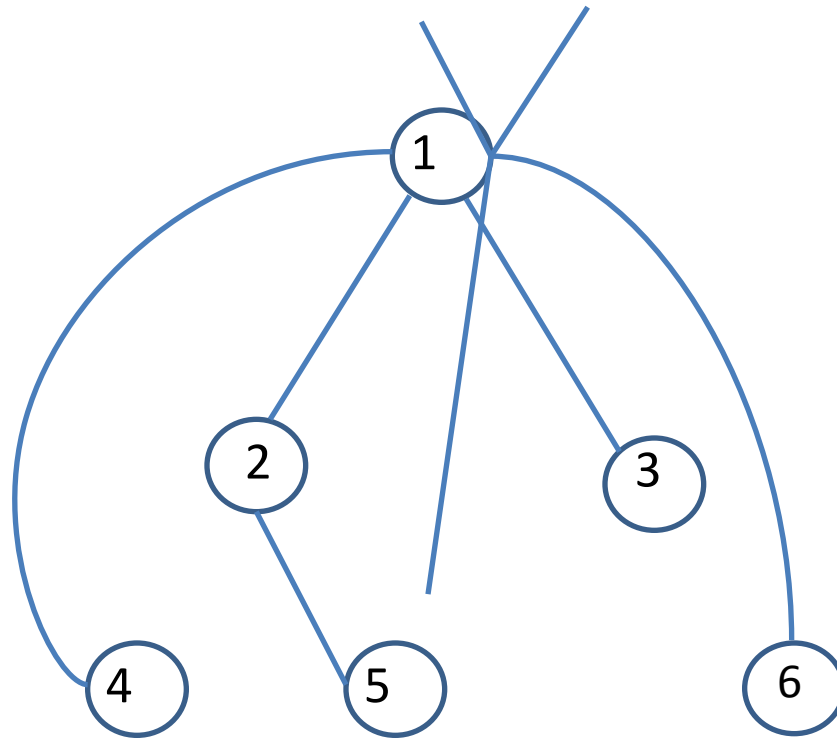
Общее число операций при выполнении *Поиск(u)*:

Цикл выполняется $| \text{смежные}[v] |$ раз.

$$\sum | \text{смежные}[v] | = O(|E|)$$

Общее число операций: **$O(|V| + |E|)$**

Поиск в глубину в неориентированном графе



Глубинный остовный лес

Поиск в глубину на неориентированном графе $G = (V, E)$ разбивает ребра, составляющие E , на два множества T и B .

Ребро (v, w) помещается в множество T , если узел w не посещался до того момента, когда мы, рассматривая ребро (u, w) , оказались в узле v . В противном случае ребро (v, w) помещается в множество B .

Ребра из T будем называть *древесными*, а из B — *обратными*.

Подграф (V, T) представляет собой неориентированный лес, называемый *остовным лесом для G , построенным поиском в глубину*, или, короче, *глубинным остовным лесом* для G .

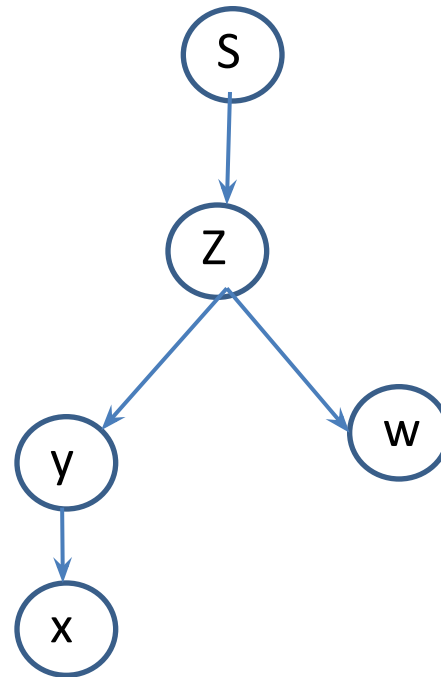
Если этот лес состоит из единственного дерева, (V, T) будем называть по аналогии *глубинным остовным деревом*.

Заметим, что если граф связан, то глубинный остовный лес будет деревом.

Узел, с которого начинался поиск, считается корнем соответствующего дерева.

Свойства поиска в глубину

Времена обнаружения и окончания обработки вершин образуют правильную скобочную структуру.



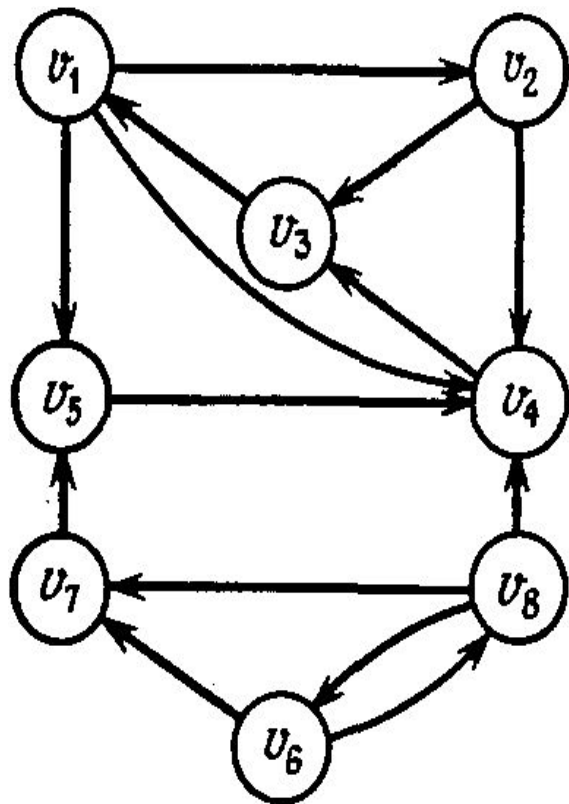
1 2 3 4 5 6 7 8 9 10
(s (z (y (x x) y) (w w) z) s)

Теорема

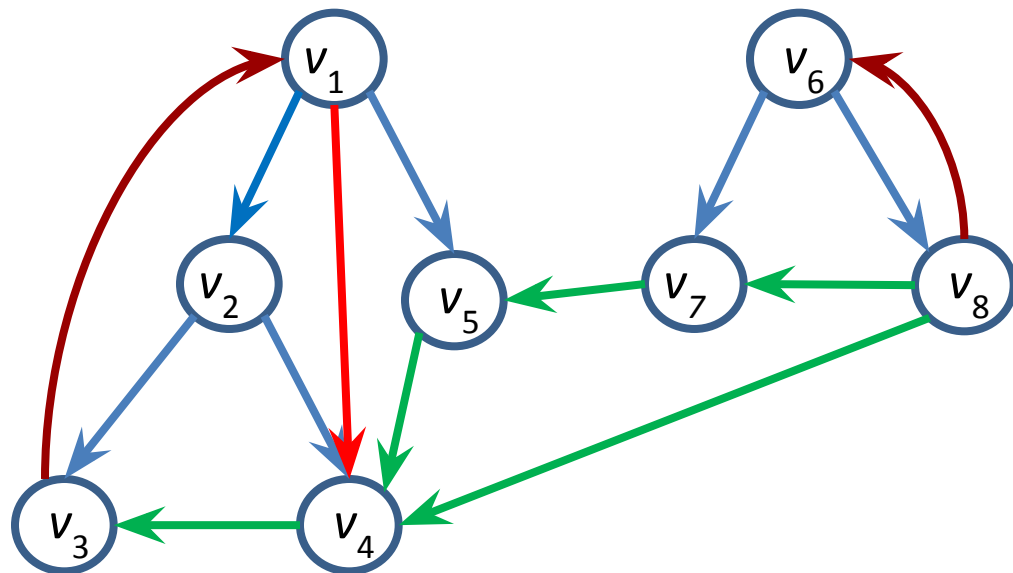
При поиске в глубину в графе $G = (V, E)$ для любых двух вершин u и v выполняется одно из следующих утверждений:

- 1) Отрезки $[d[u], f[u]]$ и $[d[v], f[v]]$ не пересекаются.
- 1) Отрезок $[d[u], f[u]]$ целиком содержится внутри отрезка $[d[v], f[v]]$ и u есть потомок v в дереве поиска в глубину.
- 1) Отрезок $[d[v], f[v]]$ целиком содержится внутри отрезка $[d[u], f[u]]$ и v есть потомок u в дереве поиска в глубину.

Поиск в глубину в ориентированном графе



a



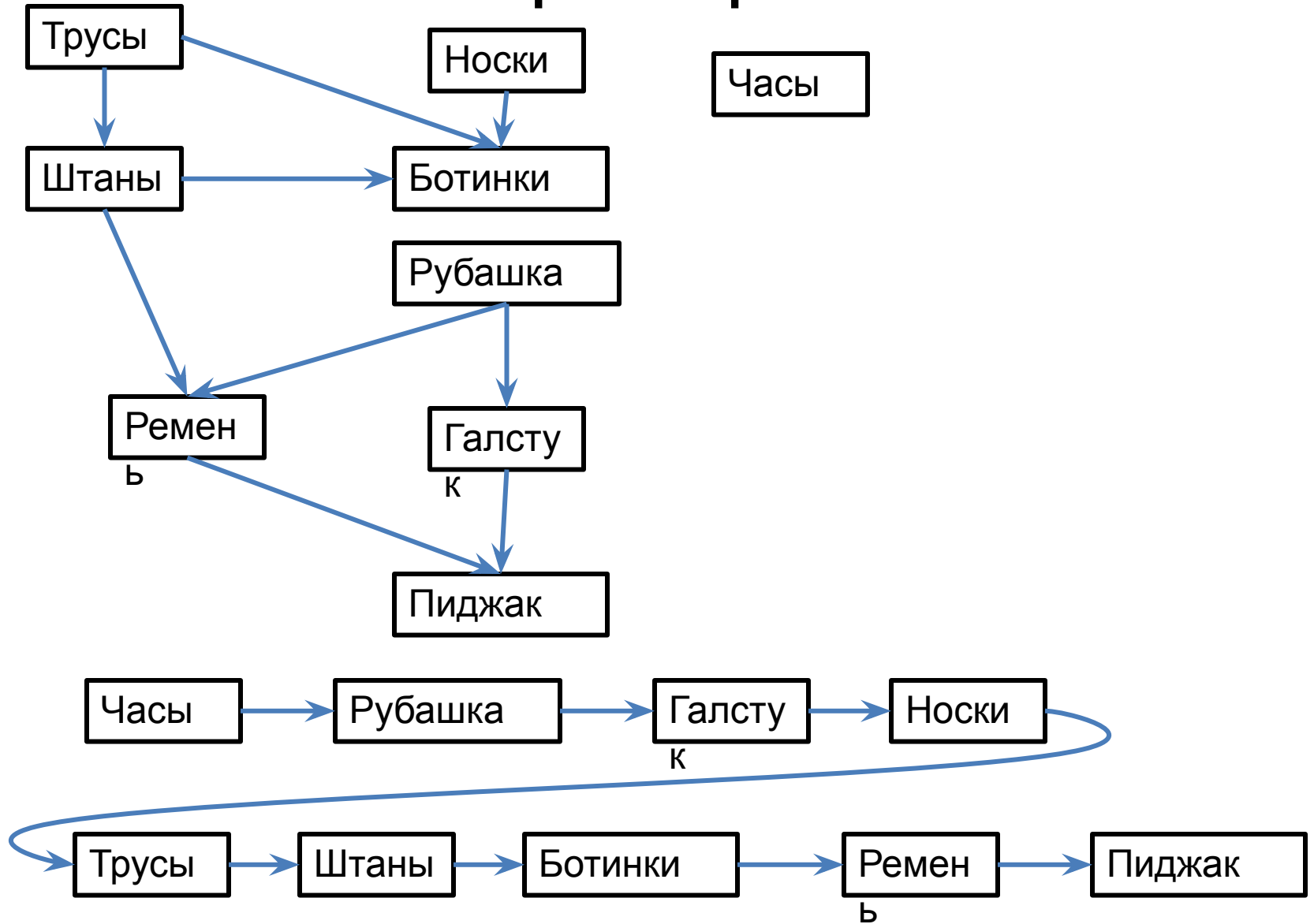
Поиск в глубину в ориентированном графе G разбивает множество его ребер на четыре класса.

- 1) *Древесные ребра*, идущие к новым узлам в процессе поиска.
- 2) *Прямые ребра*, идущие от предков к подлинным потомкам, но не являющиеся древесными ребрами.
- 3) *Обратные ребра*, идущие от потомков к предкам (возможно, из узла в себя).
- 4) *Поперечные ребра*, соединяющие узлы, которые не являются ни предками, ни потомками друг друга.

Решение задачи топологической сортировки методом поиска в глубину

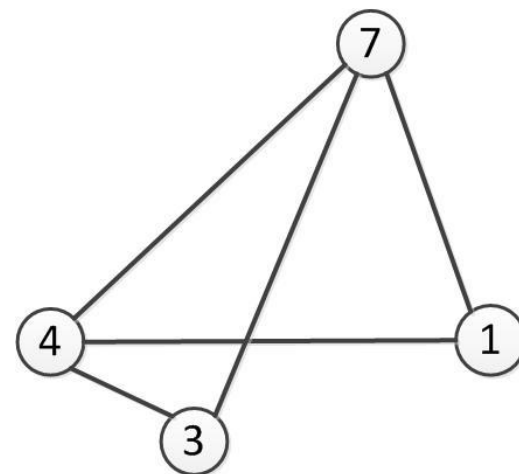
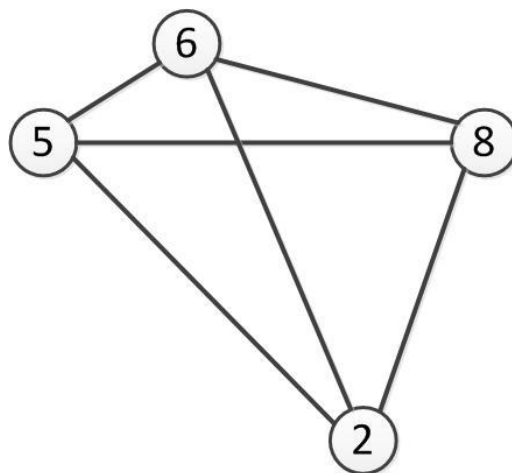
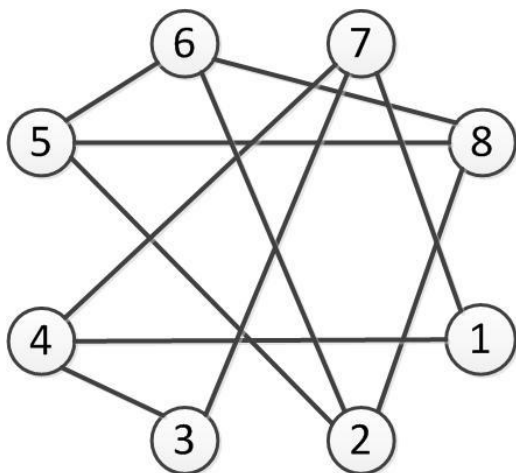
```
Топологическая_сортировка (u)
{
    цвет [u] ← серый;
    для  $\forall v \in \text{смежные}(u)$     выполнить
    {
        если (цвет[v] = белый) то
        {
            Топологическая_сортировка (v) ;
        }
    }
    цвет[u] ← чёрный;
    Поместить u в начало списка;
}
```

Пример



Поиск компонент связности в графе

Компонента связности графа – это такое множество вершин графа, что для любых двух вершин из этого множества существует путь из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества.



Реализация поиска компонент связности в графе

```
Поиск (u, n)
{
    цвет [u] ← серый;
    C[u] ← n;           // номер компоненты связности
    для  $\forall v \in \text{смежные}(u)$  выполнить
    {
        если (цвет[v] = белый) то
            Поиск(v, n);
    }
    цвет[u] ← чёрный;
}
Поиск_в_графе()
{
    для  $\forall u \in V$  выполнить
        цвет [u] ← белый;
    nk ← 0;
    для  $\forall u \in V$  выполнить
        если (цвет [u] = белый) то
        {
            nk ++;
            Поиск(u, nk);
        }
}
```

Метод поиска в ширину (BFS, Breadth-first search)

Пусть задан граф $G = (V, E)$ и некоторая начальная вершина s .

Алгоритм поиска в ширину перечисляет все достижимые из s вершины в порядке возрастания расстояния от s . Расстоянием считается число ребер кратчайшего пути.

Время работы алгоритма - $O(|V| + |E|)$.

Пусть в начальный момент времени все вершины окрашены в *белый* цвет.

1. Вершину s окрасим в серый цвет и припишем расстояние 0. Смежные с ней вершины окрасим в серый цвет, припишем им расстояние 1, их предок - s . Окрасим вершину s в черный цвет.
2. На шаге n поочередно рассматриваем белые вершины, смежные с вершинами с пометками $n - 1$, и каждую из них раскрашиваем в серый цвет, приписываем им предка и расстояние n . После этого вершины с расстоянием $n - 1$ окрашиваются в черный цвет.

Алгоритм

Инициализация

для ($\forall u \in (V \setminus \{s\})$) выполнить

{

цвет[u] \leftarrow белый;

предок[u] \leftarrow NULL;

$d[u] \leftarrow \infty$;

}

$d[s] \leftarrow 0$;

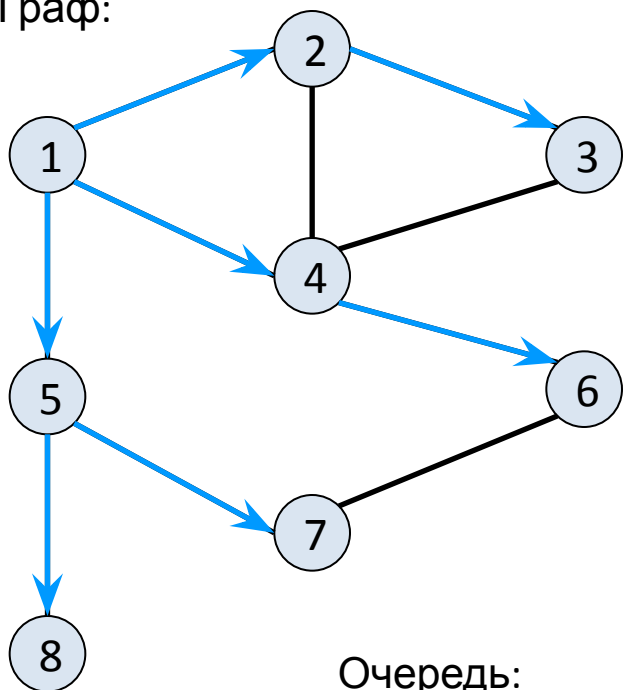
предок[s] \leftarrow NULL;

put (s , Q);

```
пока (Q ≠ ∅) выполнить
{
    u ← first (Q);
    для ( ∀ v ∈ смежные [u] ) выполнить
    {
        если (цвет[v] = белый) то
        {
            цвет [v] ← серый;
            предок[v] ← u;
            d[v] ← d[u] + 1;
            put (v, Q);
        }
    }
    get (Q);
    цвет [u] ← черный;
}
```

Использование очереди

Граф:



Очередь:



В качестве промежуточной структуры хранения при обходе в ширину будем использовать очередь.



Можно также получить дерево обхода в ширину, если отмечать каждую прямую дугу.

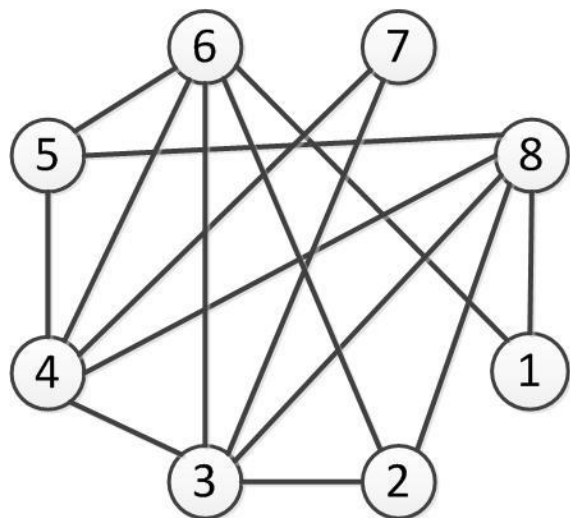
1	2	3	4	5	6	7	8
	1	2	1	1	4	5	5

Нахождение кратчайшего пути в лабиринте

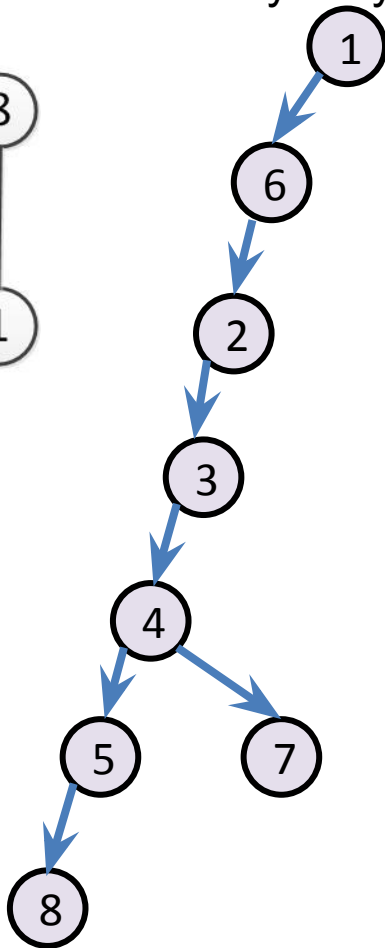
	1	2	3	4	5	6	7	8	9	10
1	15	14		4	3	2	1	2	3	4
2		13		5		3	2	3		5
3		12		6						6
4		11		7						7
5		10	9	8				20		8
6		11						19		9
7	13	12	13	14	15	16	17	18		
8	14					17		19		
9	15	16	17	18	19	18		20		
10				19	20	19				

1. Пометить числом 1 и поместить входную клетку в очередь.
2. Взять из очереди клетку. Если это выходная клетка, то перейти на шаг 4, иначе пометить все непомеченные соседние клетки числом , на 1 большим, чем данная, и поместить их в очередь.
3. Если очередь пуста, то выдать «Выхода нет» и выйти, иначе перейти на шаг 2.
4. **Обратный ход:** начиная с выходной клетки, каждый раз смещаться на клетку, помеченную на 1 меньше, чем текущая, пока не

Пример построения остовных деревьев



В
глубину:



В
ширину:

