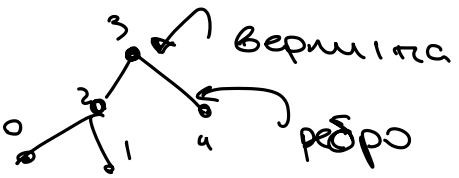
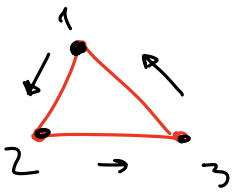
Бинарные деревья поиска (АТД)

① Дерево

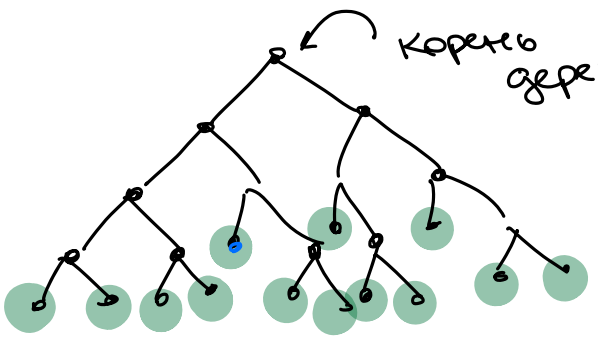


$e_1 = (0; 2)$ ↑ ребро

Дерево – связный граф без циклов.



– запрещён!

② 

"Графовые" родственники.

висячие вершины / листья дерева

в дереве между ∀ двумя вершинами ровно один маршрут (путь)

⇓

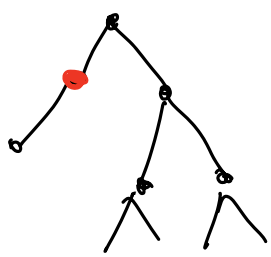
из корня в ∀ вершину ∃! путь.

③ Бинарное дерево – не более двух дочерних вершин.

предки:

1 предок

root ⇒ нет предка



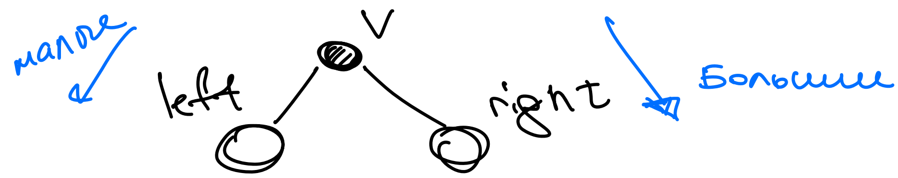
дети:

2 →

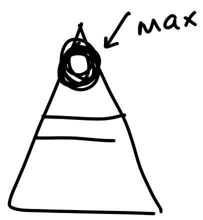
1 →

0 → лист

④ BST (Бинарное дерево поиска)

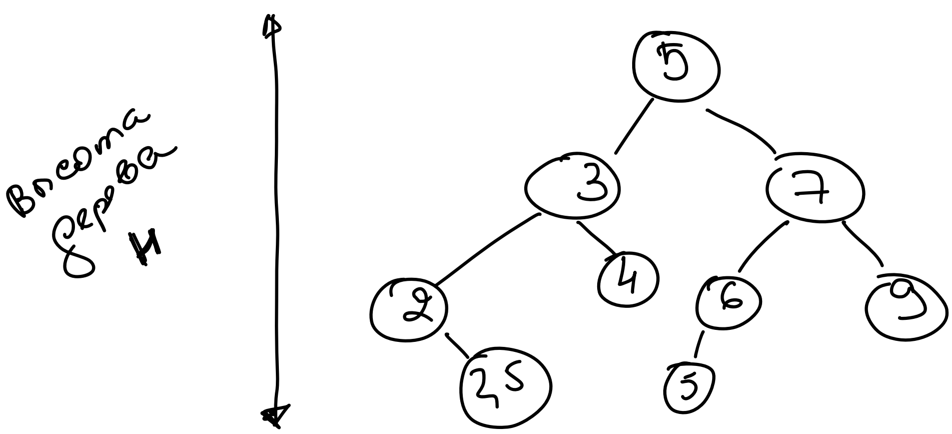


left.value < V.value ≤ right.value



Не путать с пирамидой!

⑤ Пример



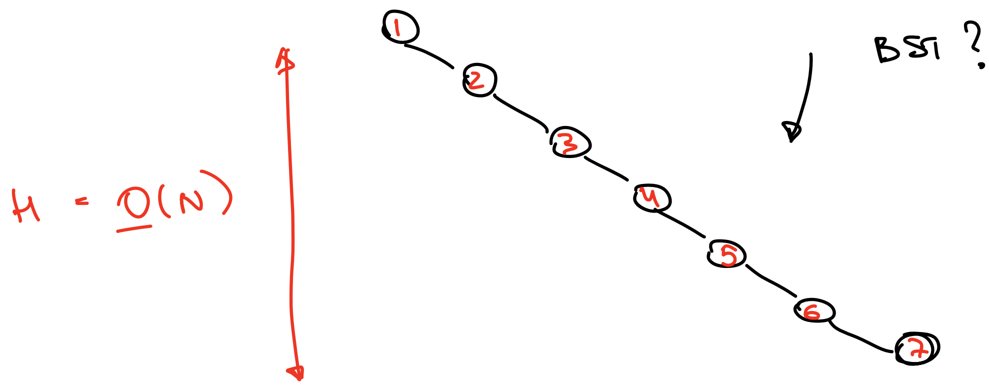
Бинарность (+)
Дерево (+)
Поиска (+)

⑥

N элементов в АТД

$H = \log_2 N$

Вырождение в бамбук



Несбалансированное дерево поиска

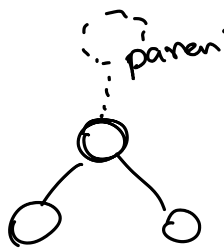
Методы:

- findMax()
- findMin()
- removeMax()
- removeMin()
- Search(elem)
- Add(elem)
- Remove(elem)

за $O(h)$ ← высота дерева.

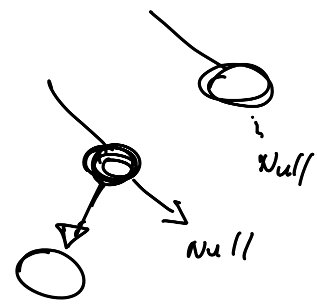
II Реализация:

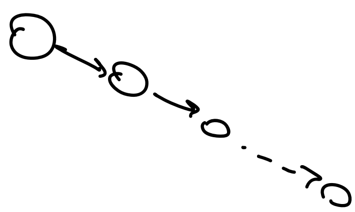
Хранение узла (вершина)



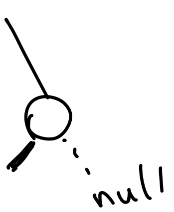

```
class Node {
    → value        // число
    → left = Null  } Node
    → right = Null }
}
```

```
class BST {
    root = Null   → корень дерева.
    ...  → методы
}
```

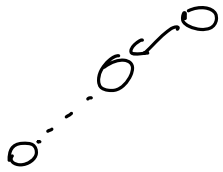




```
метод  findMax() {
    runner = root
    if runner = Null
        return False  // ошибка
    while runner.right != null
        ↳ runner = runner.right
    return runner.value
}
```

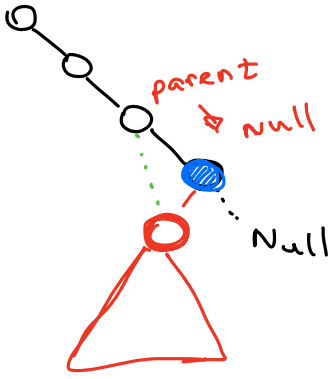


Find Min ? аналогично

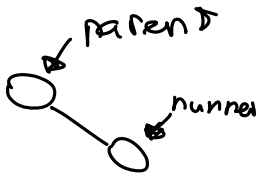


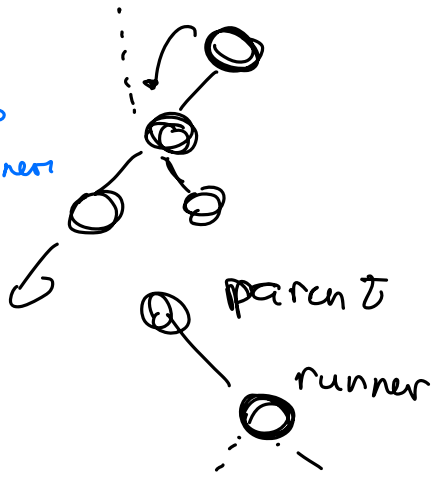
метод RemoveMax

- ищем → просто удаляем
- Есть левый ребенок



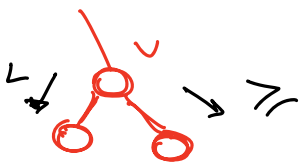

```
RemoveMax ( ) {
    runner = root
    if runner = null
        ↳ return        // не было удалять
    parent = null
    while runner.right != Null
        parent = runner
        runner = runner.right
    if parent = null    // корень максимален
        root = root.left
        delete runner
        return
    parent.right = runner.left
    delete runner
    return
}
```





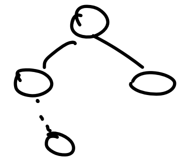
метод Add (elem)

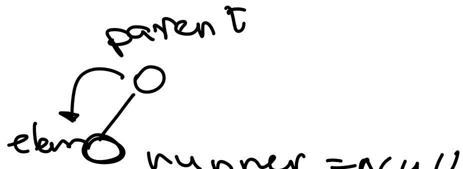
```
runner = root
if runner = null:
```



вставка всегда происходит выше

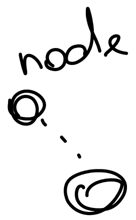
```
↳ root = Node (elem)
  return
parent = Null
while runner != null
{ если  runner.value > elem :
            parent = runner
            runner = runner.left
  если  runner.value <= elem
            parent = runner
            runner = runner.right
}

если    parent.value > elem
        parent.left = Node (elem)
иначе
        parent.right = Node (elem)
```

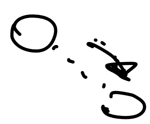




Рекурсивный:

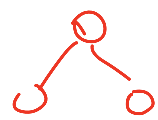
```
recrAdd (elem, node = root)      ← поддерево
    если  node = null
             ↳ ил   пустое   дерево
    если  elem >= node.value
    {
        если   node.right = Null
               node.right = Node (elem)
        иначе
               recrAdd (elem, node.right)
    }
    Аналогично   если   elem < node.value
```





метод Add (elem)
{

```
if root = null:
  ↳ root = Node (elem)
    return
 elese
  ↳ recrAdd ( elem, root )
}
```



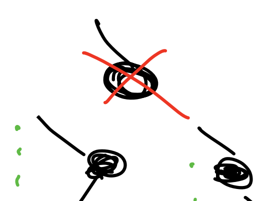
```
метод Search ( elem ) {
    runner = root
    while runner != null
    {
        если runner.value < elem
                runner = runner.right
        иначе если runner.value > elem
                runner = runner.left
        иначе
                return True // runner
    }
    return False // null
}
```

Удаление

Шаг 1. Найти elem (search)

Шаг 2. Удалить

- лист
- 1 ребенок



Remove (elem)

runner = root

parent = null

если root = null

↳ return

while runner != null

{

если runner.value > elem :

parent = runner

runner = runner.left

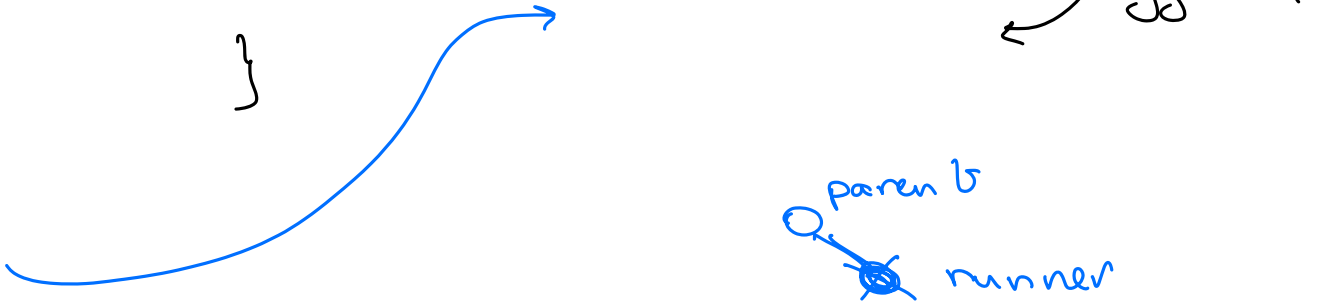
если runner.value < elem

parent = runner

runner = runner.right

если runner.value = elem: удалить

}



если runner.left = runner.right = null

если parent.left = runner

parent.left = null

иначе

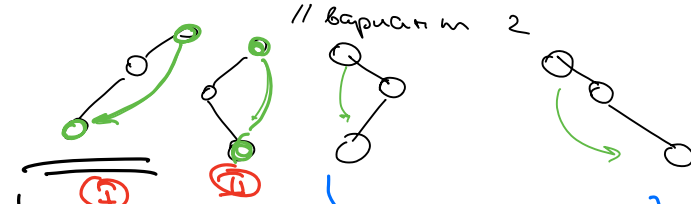
parent.right = null

delete runner

если runner.left != null and runner.right != null

3 варианта

иначе



если parent.left = runner:

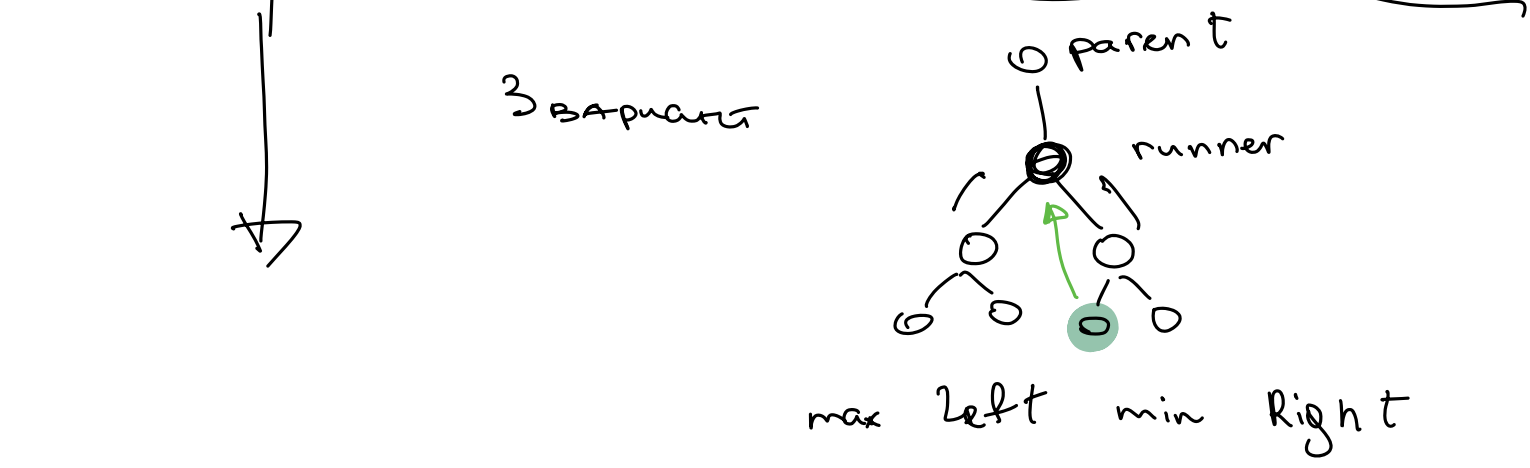
① если runner.left != null

parent.left = runner.left

② иначе

parent.left = runner.right

иначе



3 варианта

max Left min Right

```
min = findMin(runner.right)
RemoveMin(runner.right)
runner.value = min
```