Бинарные деревья поиска

25/02/2023

В предыдущих сериях

- Хотим структуру с $get, lower\_bound$ по значению, добавлением и удалением по значению
- Skip-list подходит

Пара слов про кучи

В куче нет запроса get, есть только get_min. Такое бывает полезно:

- Например, если вы закинули все элементы массива в кучу, а потом доставали по очереди минимумы и удаляли их, то вы отсортировали массив (это называется heap sort).

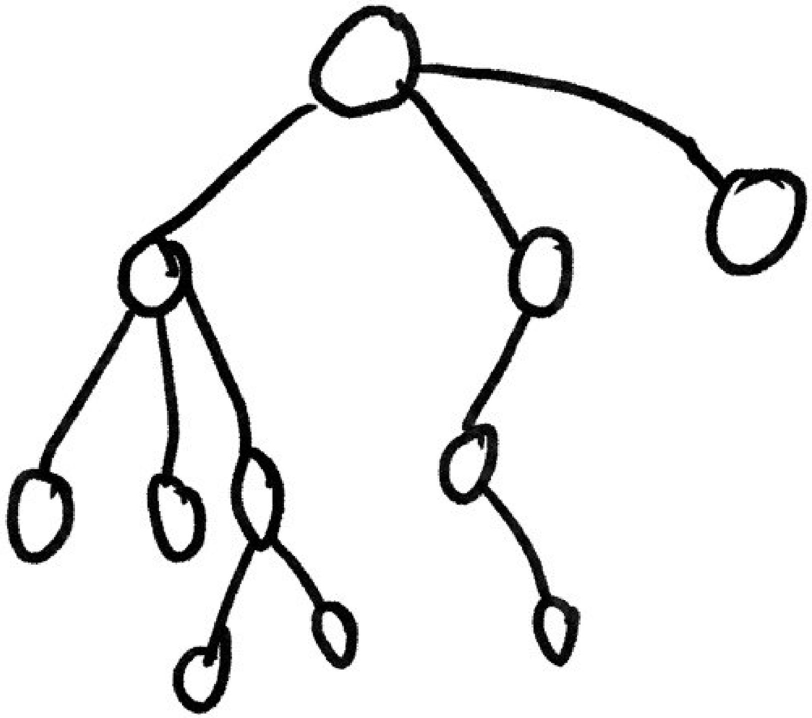
Но нам это не очень интересно, потому что деревья делают все то же, что куча, и даже больше.

Дерево поиска

Мы с вами уже обсуждали списки - это такая структура, которая хранит ссылку на следующий элемент.

В деревьях узлы хранят ссылку на несколько "следующих" узлов, которые называются "сыновьями". Каждый сын является корнем своего поддерева. Все поддеревья сыновей независимы и не пересекаются. Точка входа в дерево это корень.

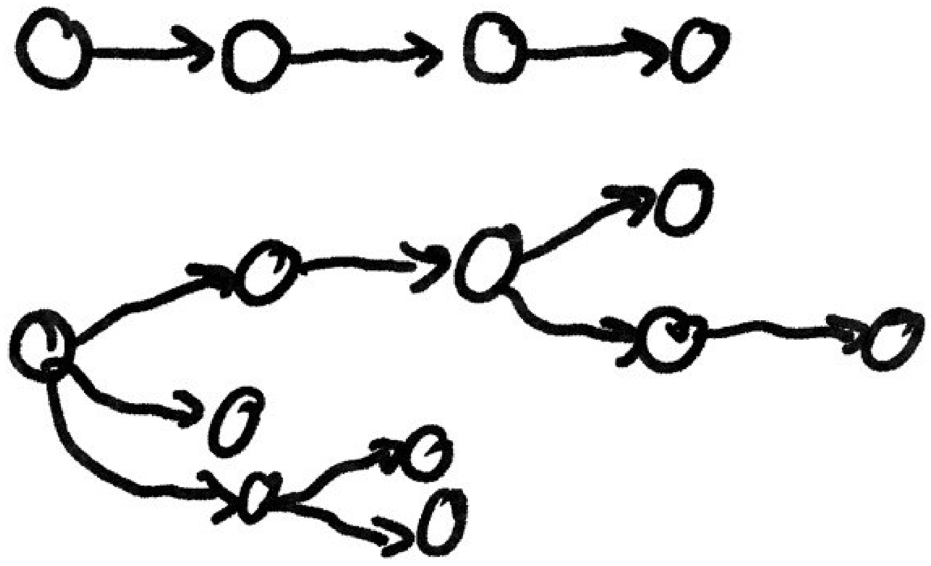
Дерево



Вопрос

Является ли список деревом?

Под другим углом



Поддеревья

Некоторые величины можно пересчитывать через поддерево с помощью рекурсивных определений

Например,

$$sum(tree) = root_{val} + \sum_{sons} sum(subtree(sons))$$

$$sz(tree) = 1 + \sum_{sons} sz(subtree(sons))$$

Бинарное дерево поиска

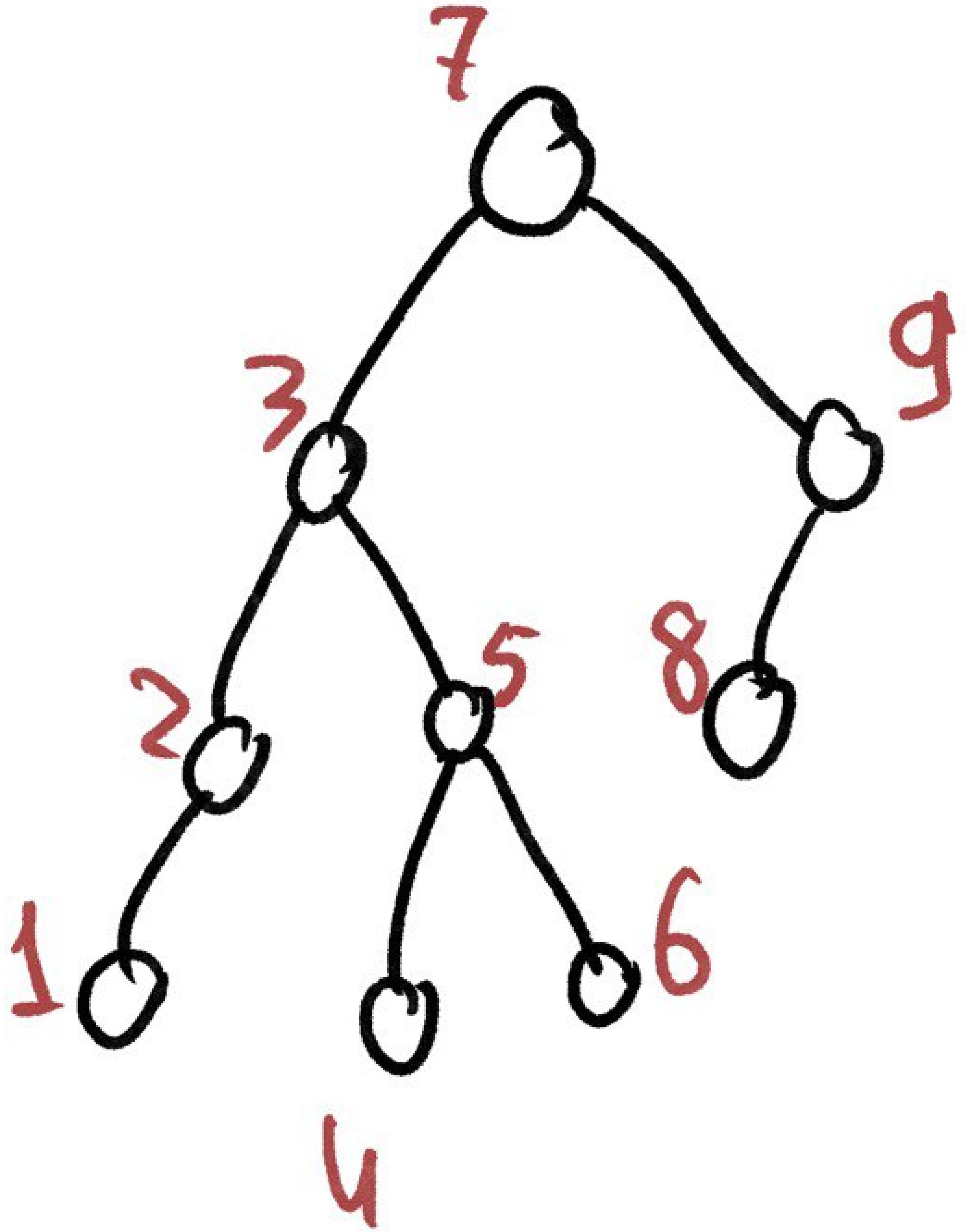
Дерево называется *бинарным*, если у каждой вершины не больше 2 сыновей. Тогда можно их назвать левым и правым

```
class Node:
    left: None
    right: None
    val: 0
```

Бинарное дерево становится деревом поиска, если для каждой вершины $node$:

$$\max(subtree(node_l)) \leq node_{val} \leq \min(subtree(node_r))$$

BST

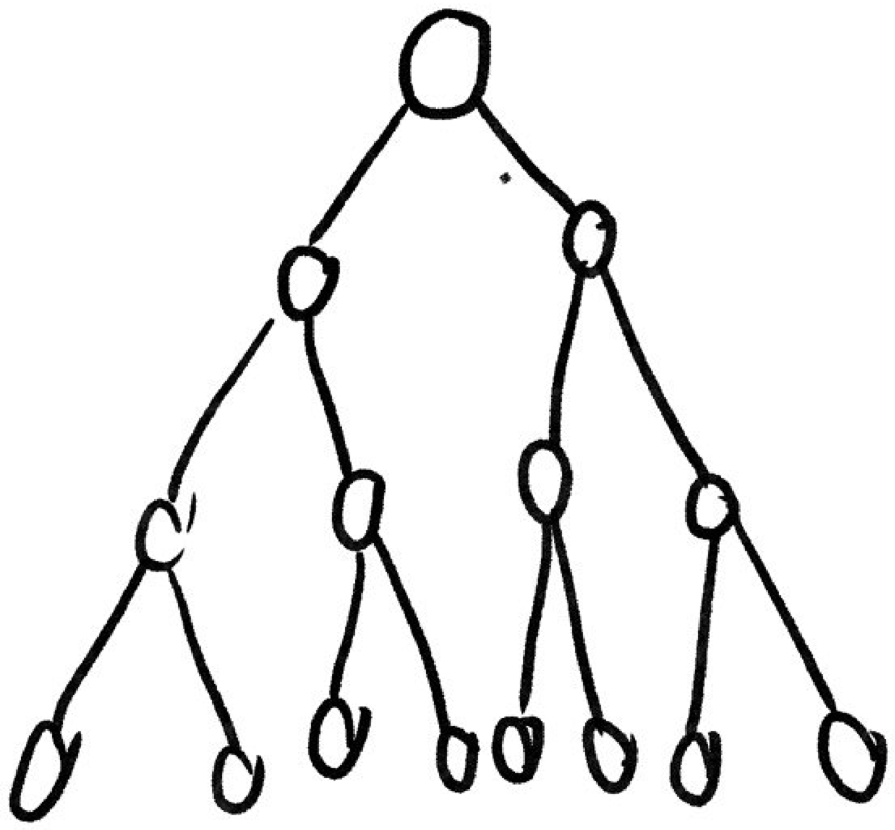


Алгоритм поиска

```python
def find(node, x):
    if node.val == x:
        return node
    if node.val < x:
        return find(node.r, x)
    return find(node.l, x)
```

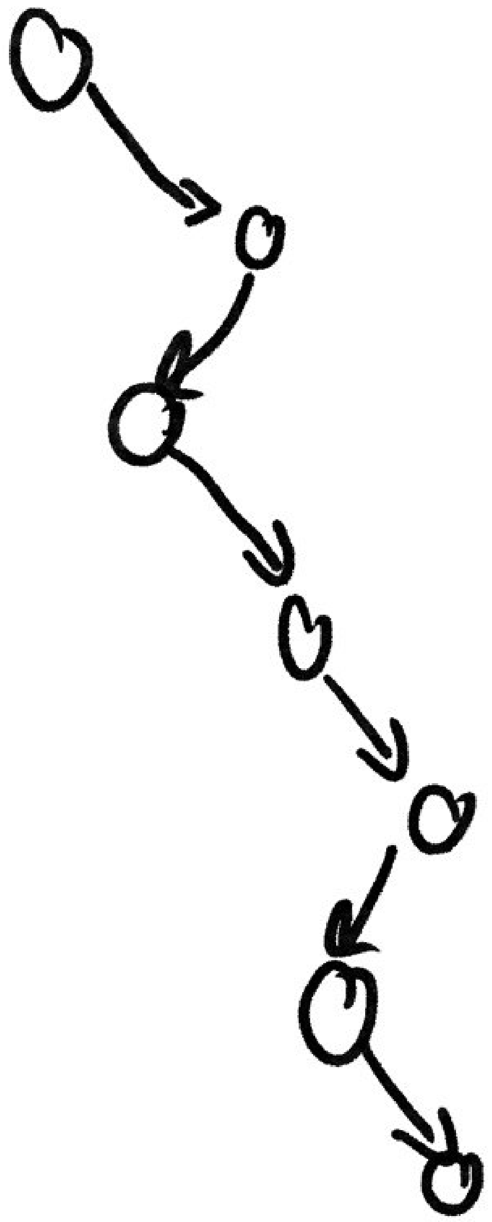
Вопрос

Какая асимптотика у `find` в таком дереве?



Вопрос

А в таком?



Проблема

Дерево должно иметь небольшую глубину. При наивном `insert` (`find` + добавление в конце, если `None`) дерево может иметь глубину $O(n)$.

Принято делать перебалансировку после `insert` вдоль соответствующего пути в дереве.

Детерминированно:

- AVL-tree
- RB-tree

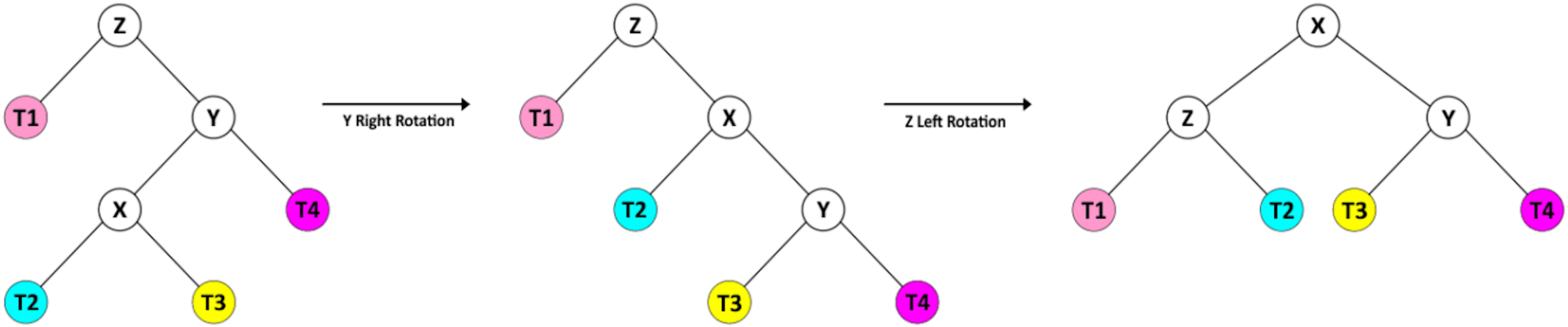
Недетерминированно:

- Декартово дерево (Treap)

AVL

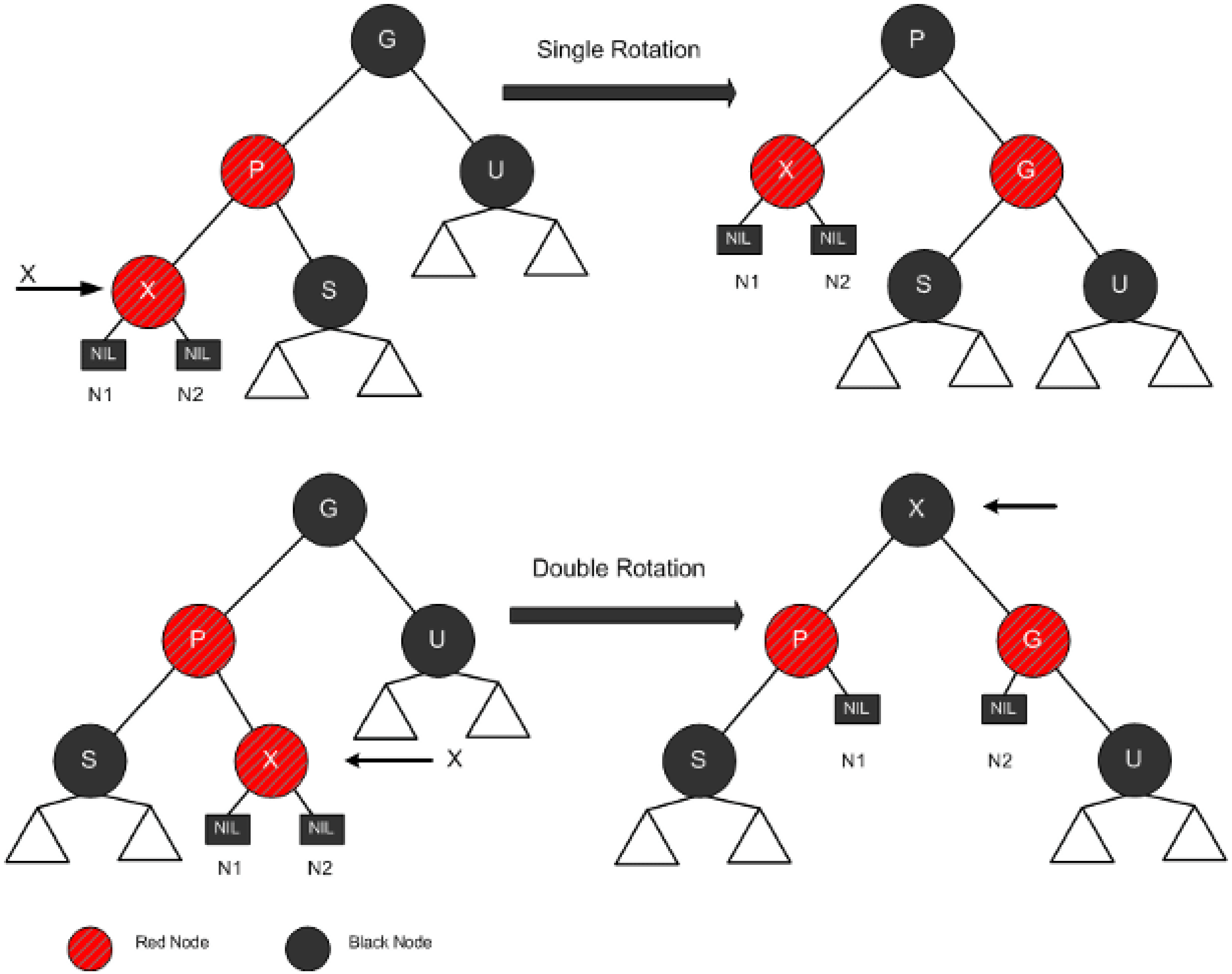
Основное свойство - у любой вершины глубина левого и правого поддерева отличается не более, чем на 1. Если это не так, то можно сделать некоротые переподвешивания и починить это свойство. Глубина $O(\log n)$.

Повороты в дереве



Red-black tree

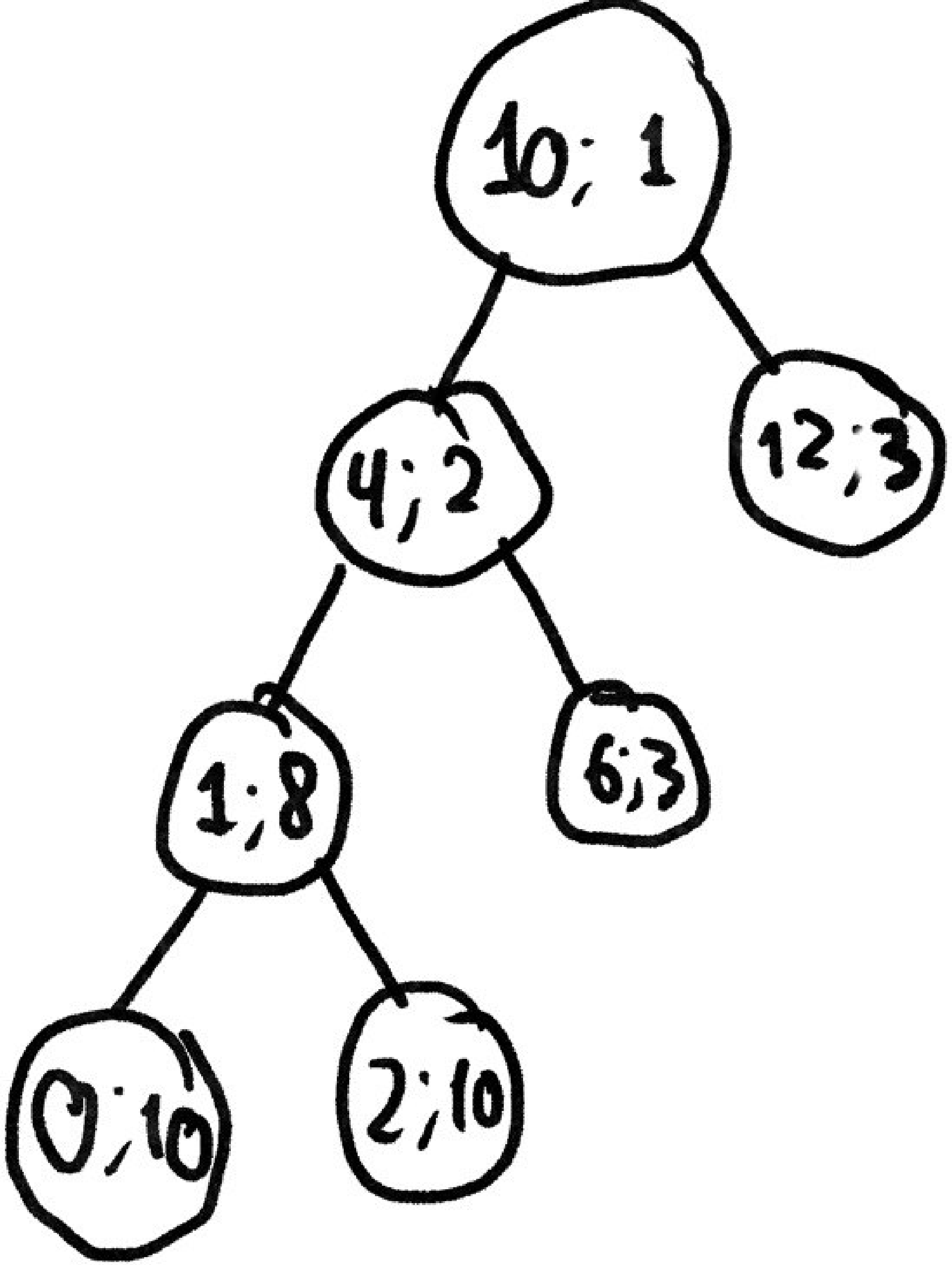
Вершина имеет цвет, и на цвета наложены некоторые правила. Если правило нарушено, можно починить его поворотом. Структура дерева менее четкая, но зато нужно меньше изменений структуры. Глубина $O(\log n)$ все равно гарантируется


G
Single Rotation
P
P
U
X
G
X
NIL
NIL
NIL
NIL
N1
N2
S
U
S
N1
N2
G
X
Double Rotation
P
U
P
G
NIL
NIL
S
X
X
N1
N2
S
U
NIL
NIL
N1
N2
Red Node
Black Node

Декартово дерево

Вершинам выдается *приоритет* y. Декартово дерево является бинарным деревом поиска по x и кучей по y. Таким образом, оно строится однозначно.

```
class Node:
    def __init__(self, x):
        self.x = x
        self.y = random()
```

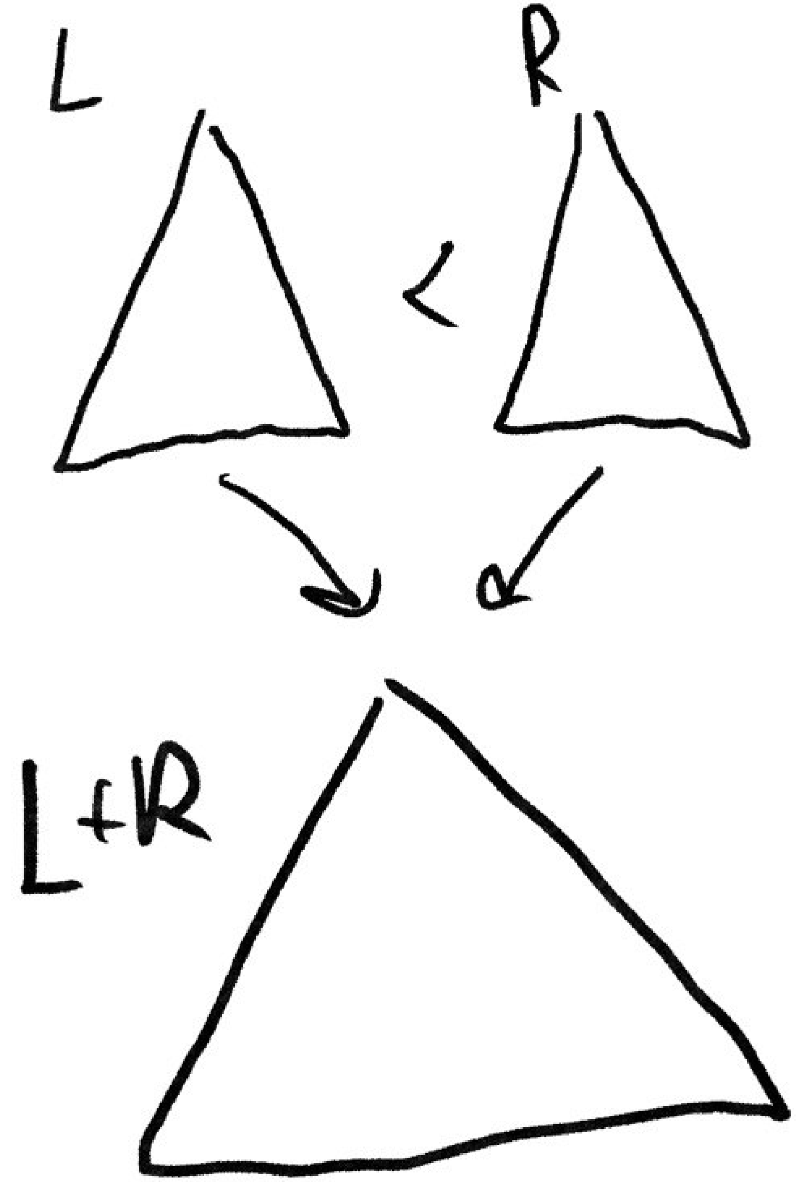


10; 1
4; 2
12; 3
1; 8
6; 3
0; 10
2; 10

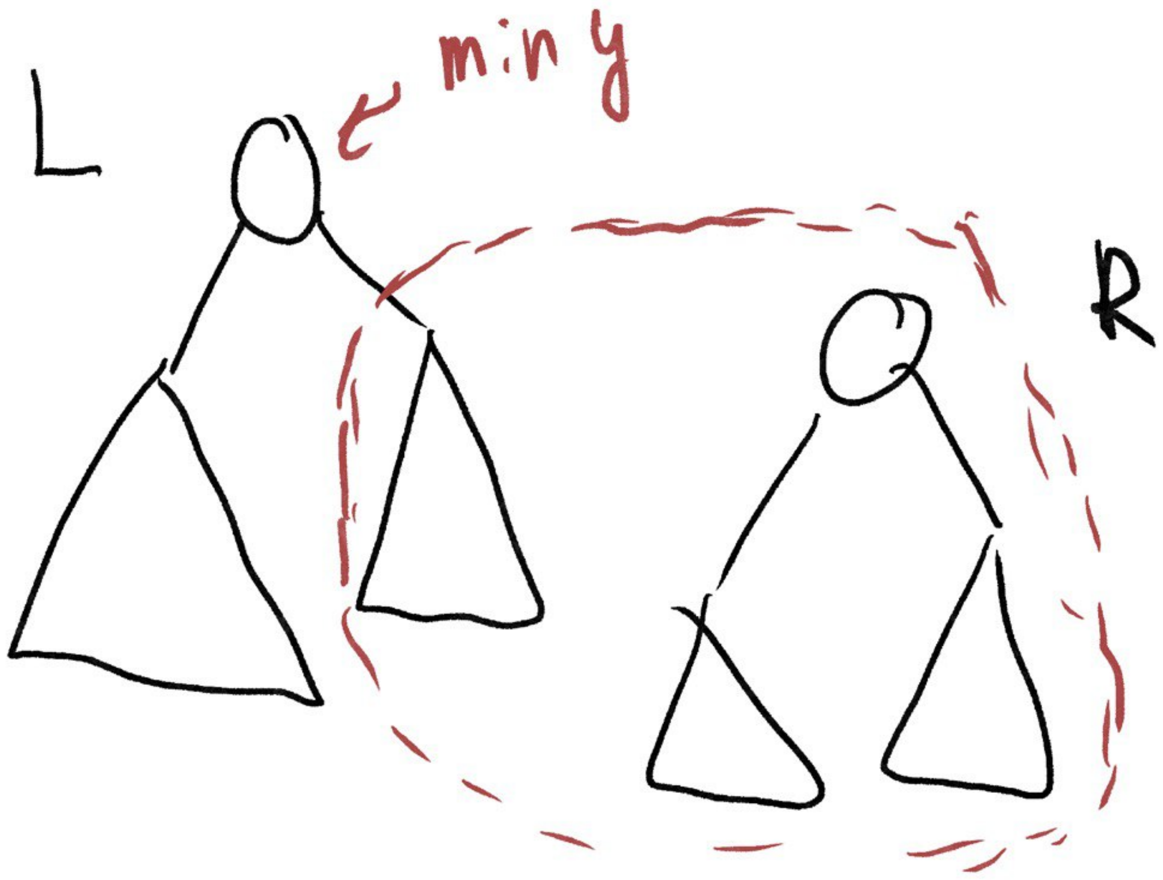
Операции

Декартово дерево строится на двух операциях: `merge` и `split`.

Merge

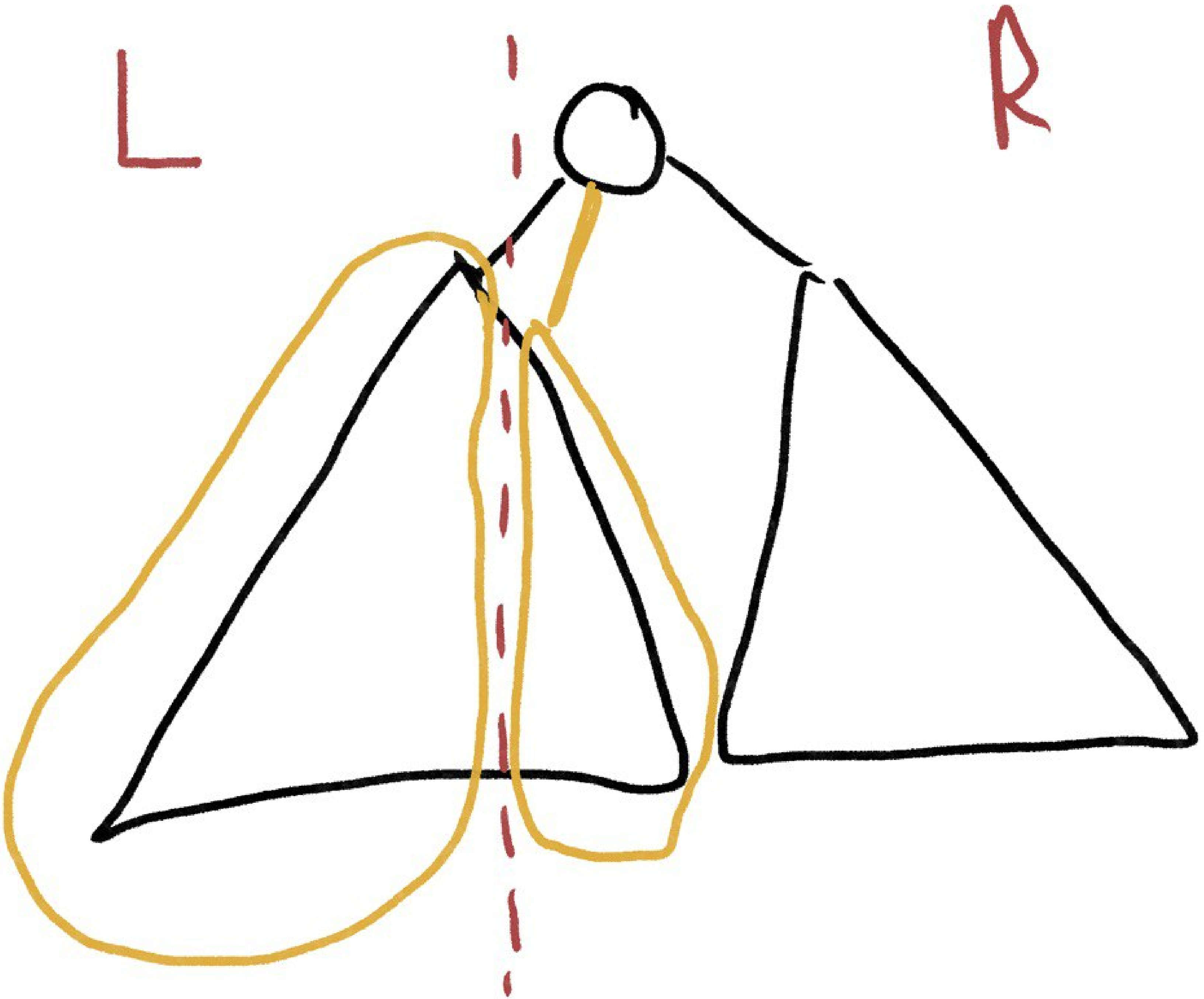


Merge



Merge

```python
def merge(l, r):
    if l is None:
        return r
    if r is None:
        return l
    if l.y < r.y:
        l.r = merge(l.r, r)
        return l
    else:
        r.l = merge(l, r.l)
        return r
```


L
R

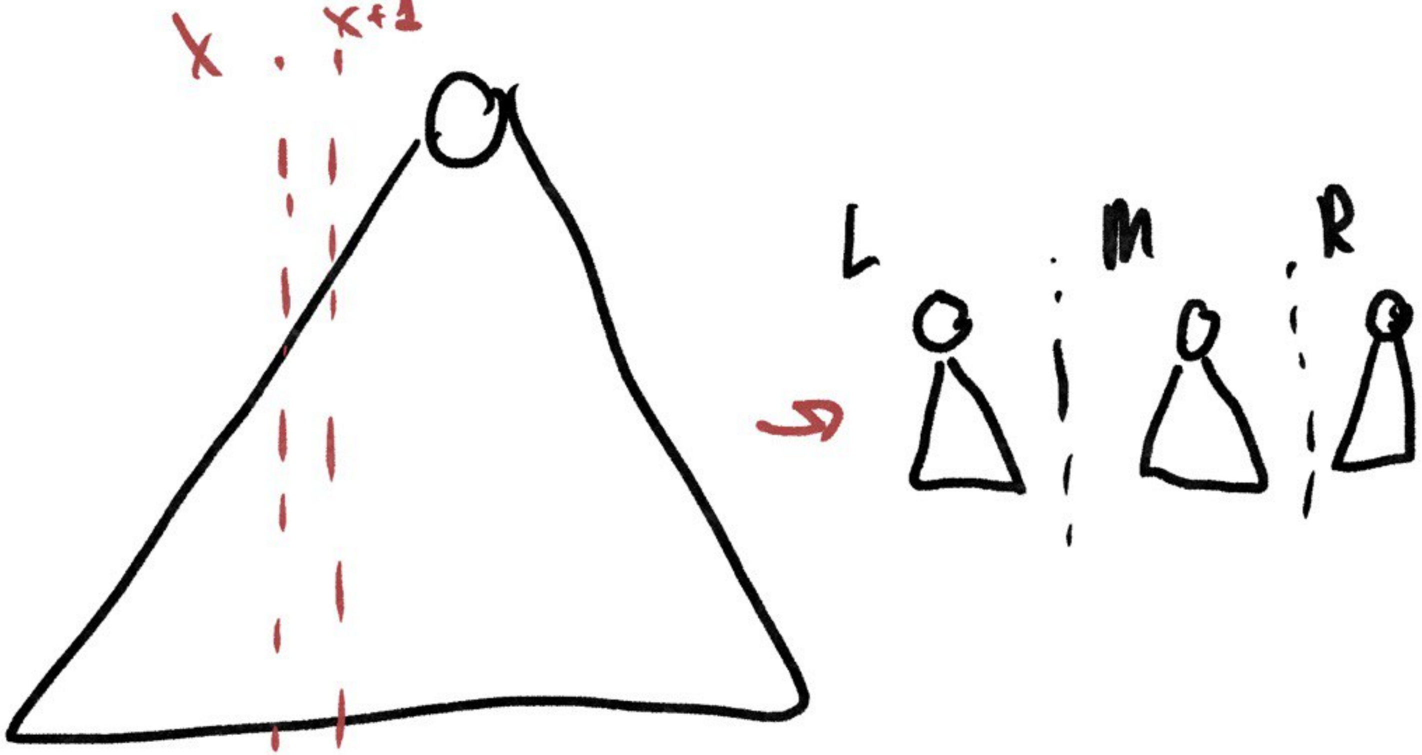
Split

```
def split(node, x):
    if node is None:
        return None, None
    if node.x < x:
        l, r = split(node.r, x)
        node.r = l
        return node, r
    else:
        l, r = split(node.l, x)
        node.l = r
        return l, node
```

Вопрос

Как реализовать поиск? Добавление? Удаление?

Добавление



Асимптотика

$O(h)$ на операцию. Надо только понять, почему $h \sim \log n$.

Надо понять матожидание количества предков у вершины.

$$P[\text{u is ancestor of v}] = P[u_y = \max_{w \in [u;v]} w_y] = \frac{1}{|u - v|}$$

$$\sum_u \frac{1}{|u - v|} = O(\log n)$$

B-tree

Давайте сделаем немного другой подход, где у вершины не 2 сына, а произвольное количество. Сын хранит максимума в своем поддереве, поэтому мы можем правильно делать поиск.

```
class Leaf:
    def __init__(self, x):
        self.x = x

class Node:
    def __init__(self):
        self.children = []
        self.max = None
```

B+ tree

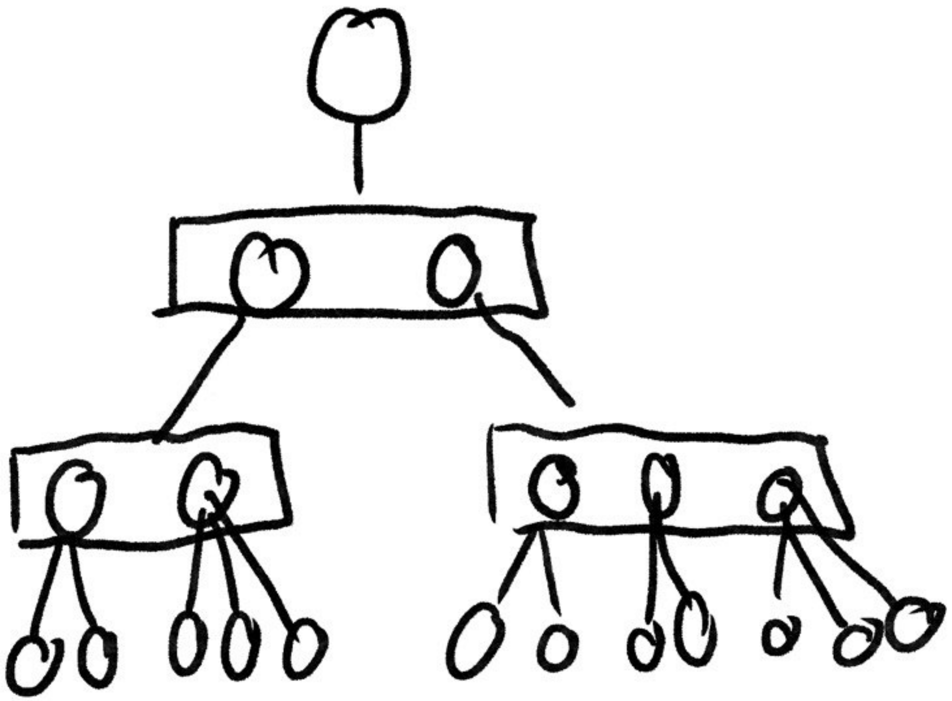
В реализациях B-деревьев есть разные подходы, мы сделаем тот, который называется B+ деревом.

В нашем дереве все значения будут записаны в листьях, а вершины выше будут *поисковыми* вершинами. У каждой из них от b до $b \cdot 2 - 1$ сыновей (отсюда и название.).

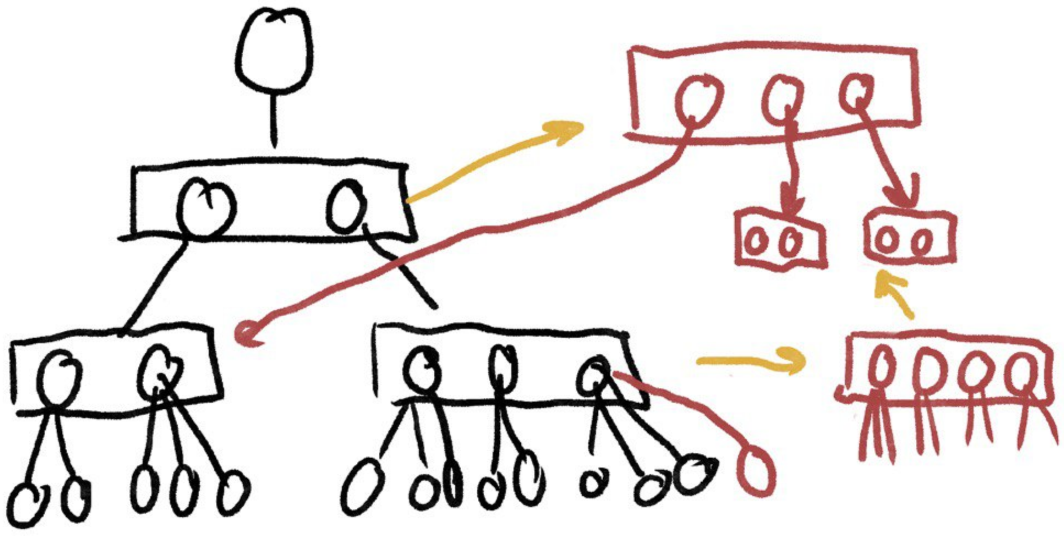
Когда добавляется значение, мы закидываем новый лист в нужное место дерева. Если это сломает инвариант предка, предка нужно разделить на две вершины. Если это сломает инвариант его предка, то рекурсивно решаем дальше.

2-3 Дерево

B+ дерево для $B = 2$.



Добавление



Мотивация для B-дерева

Базы данных обычно хранят данные на диске, куда лучше складывать данные последовательными блоками (условно, 16кб). Поэтому обработать 160 элементов в одном блоке быстрее, чем 10 элементов в разных блоках.

Получается ситуация, в которой выгодно сделать $B \sim 1000$. Тогда глубина дерева становится равна $O(\log_B n)$, а время ответа на запрос получается $O(B \log_B n) + k_{memory} \cdot O(\log_B n)$.

Спуск по дереву

Кроме того, чтобы делать стандартный get в структуре, иногда хочется найти первый элемент, который удовлетворяет заданному свойству.

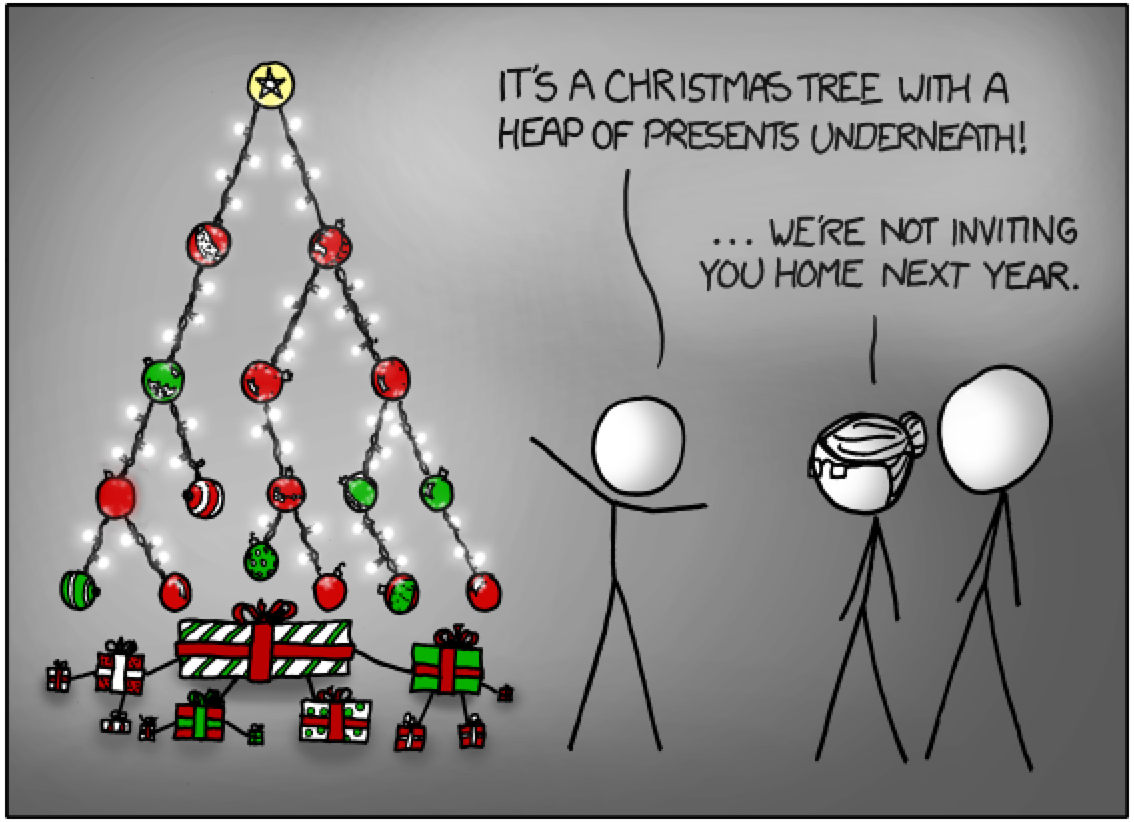
Тогда можно хранить в дереве некоторую дополнительную информацию, и идти в левое поддерево, если в его поддереве есть хотя бы одна подходящая вершина, и в правое поддререво, если такой вершины нет.

Применимо практически в любых деревьях

Спуск по дереву

```
def find_kth(node, k):
    if node.l.count > k:
        return find_kth(node.l, k)
    if node.l.count + 1 == k:
        return node
    return find_kth(node.r, k - node.l.count - 1)
```

Мем


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

Идея пет-проекта

Можно написать с помощью B-дерева свою небольшую файловую систему. Можно, например, выписать все файлы в алфавитном порядке и использовать их полные пути как ключи. Затем можно сделать разбиение файлов на блоки так, чтобы размер блока был B = 100. Тогда найти любой файл в вашей файловой системе можно за $\log_{100} n$ обращений к хранилищу.

Поверх вашей файловой системы можно написать свою функцию `open()` и запустить на ней какую-то программу, которая работает с файлами.

```
./0_course_intro
./0_course_intro/main.pdf
./1_binary_search
./1_binary_search/main.pdf
./2_sorting
./2_sorting/main.pdf
./3_containers
./3_containers/main.pdf
```

Что читать дальше

- AVL
- RB-tree
- Декартово дерево по неявному ключу
- https://en.wikipedia.org/wiki/B-tree#B-tree_usage_in_databases
- https://ru.algorithmica.org/cs/tree-structures/treap/