Алгоритмы и Структуры Данных. Лекция 3

21.02.2024

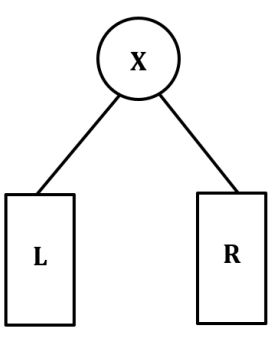
_scarleteagle **imkochelorov**

Двоичное дерево поиска (BST)

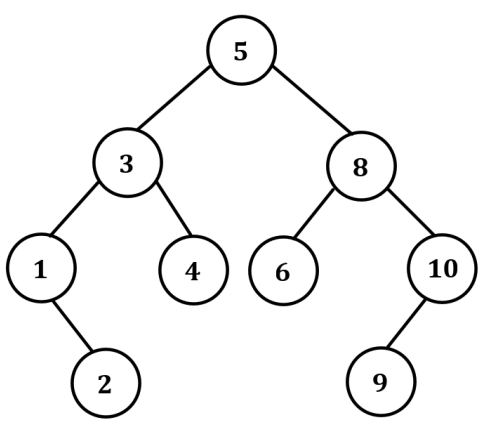
Введение:
Реализуем структуру данных set

- `insert(x)` — добавить в множество элемент, если ранее его в нём не было
- `find(x)` — проверить, есть ли число в множестве
- `remove(x)` — удалить элемент из множества, если он в нём присутствует

Инвариант дерева:

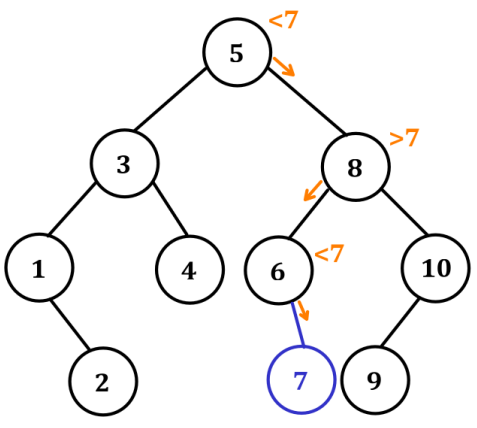


$$\forall y \in L \quad y < x$$
$$\forall y \in R \quad y > x$$

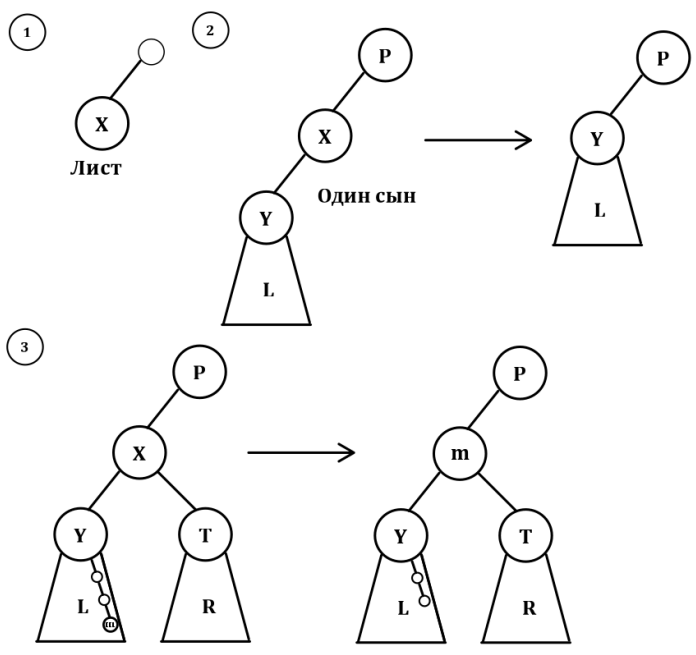


Пример двоичного дерева поиска

Реализация операций:



работа `insert(7)` на дереве из примера



3 случая работы `remove(x)`

Визуализация работы `find(x)` представляется очевидной

```
class Node:
  def __init__(self, key: int):
    self.key = key
    self.l = None
    self.r = None
```

`find(x)`: очевидно
`insert(x)`: достаточно визуального описания
`remove(x)`:

- Если `x` лист, то удаляем `x`
- Если у `x` один сын, то заменяем `x` на сына
- Если у `x` два сына, то идем до конца в правого сына (самая большая вершина в поддереве), перекинем его вместо `x` и удалим его на исходном месте одним из 2 предыдущих способов

Асимптотика операций: $O(h)$, где h — высота дерева.
Что грустно, так как в худшем случае мы получим дерево вида “бамбук” с высотой равной числу всех элементов

Сбалансированное двоичное дерево. AVL-дерево

by Адельсон-Вельский & Ландис (1962)

Сбалансированное двоичное дерево — $h = O(\log n)$

Инвариант: $\forall v \quad |h(v.l) - h(v.r)| \leq 1$

Пусть $f(h)$ — min возможное кол-во вершин с высотой h

$f(0) = 0$

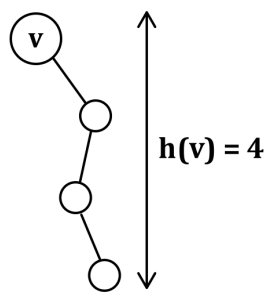
$f(1) = 1$

$f(h) = f(h-1) + f(h-2) + 1$

$f(h) \geq F_h \sim \varphi^h$

$f(h) = \Omega(\varphi^h)$

$h = O(\log n)$

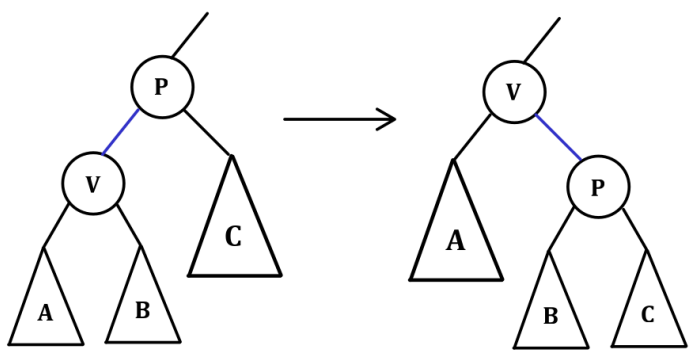


$h(v)$ считается с учётом вершины начала и вершины конца поддерева

Поворот ребра:

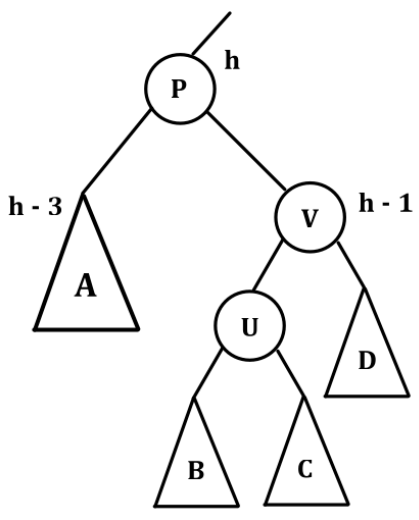
```
def rotateRight(v, p):
  A = v.l
  B = v.r
  C = p.r
  par = p.p
  p.l = B
  B.p = p
  v.r = p
  p.p = v
  v.p = par
  if par.l = p:
    par.l = v
  else:
    par.r = v
```

$h(v) = \max(h(v.l), h(v.r)) + 1$



Визуализация поворота вершины

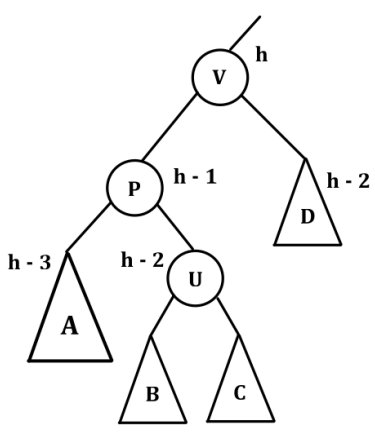
Поворотов вершин достаточно, чтобы сохранять инвариант AVL-дерева. Для реализации этого будем смотреть, у какой вершины сломался инвариант
После вставки или удаления вершины, пройдёмся снизу-вверх, проверяя инвариант вершин. Пусть p — первая вершина, в которой сломался инваривант:



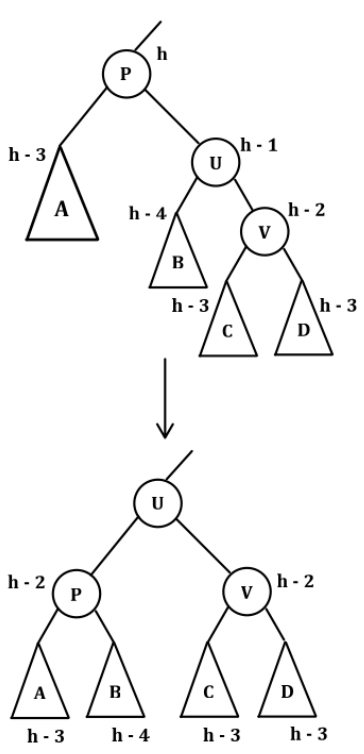
$$h(v) - h(A) = 2$$

Рассмотрим случаи добавления в дерево:

1) $h(u) = h(D) = h - 2$
Повернём (p, v)



2) $h(u) = h - 2,\ h(D) = h - 3$
Повернём (u, v)
Повернём (p, u)



3) $h(u) = h - 3,\ h(D) = h - 2$
Повернём (p, v)