Алгоритмы и Структуры Данных. Лекция 4

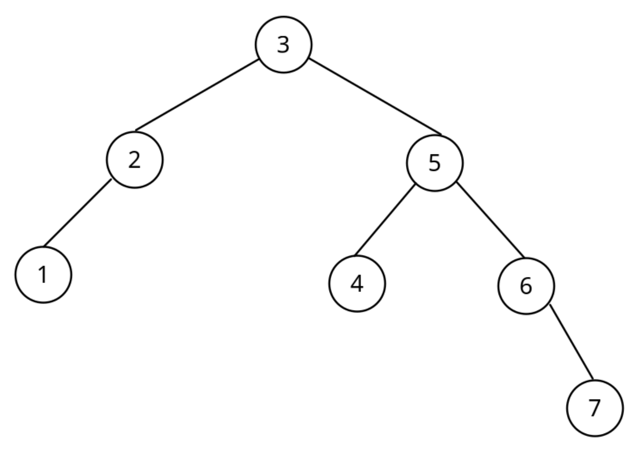
6.03.2024

_scarleteagle **imkochelorov**

Splay-дерево

Splay-дерево — обычное дерево поиска, которое является сбалансированным, благодаря операции `splay`

Операции Splay-дерева:
- `find(x)`
- `insert(x)`
- `remove(x)`
- `splay(x) # перестроить дерево так, чтобы x стал корнем`



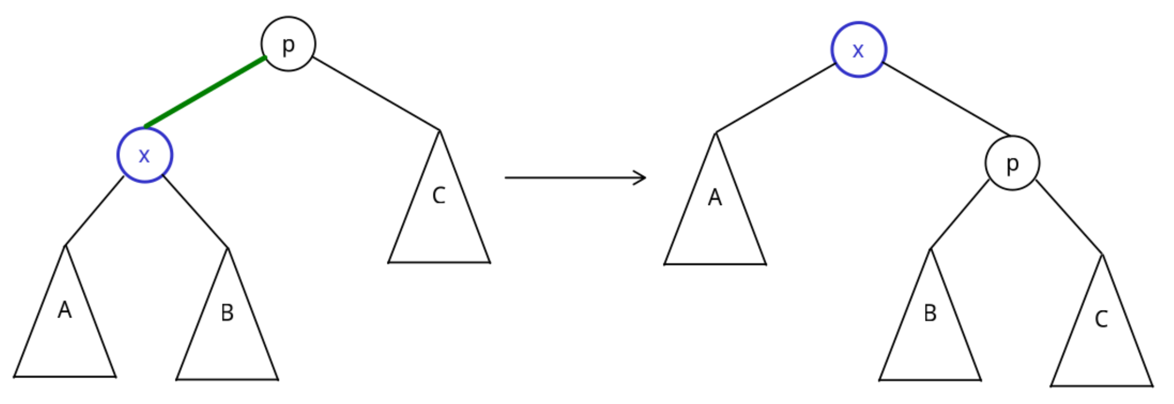
дерево поиска

Зададим 3 преобразования дерева, поднимающие необходимый нам элемент наверх

Преобразование дерева *зиг*

p — корень, нужно поднять x — сына p

Повернём (x, p)

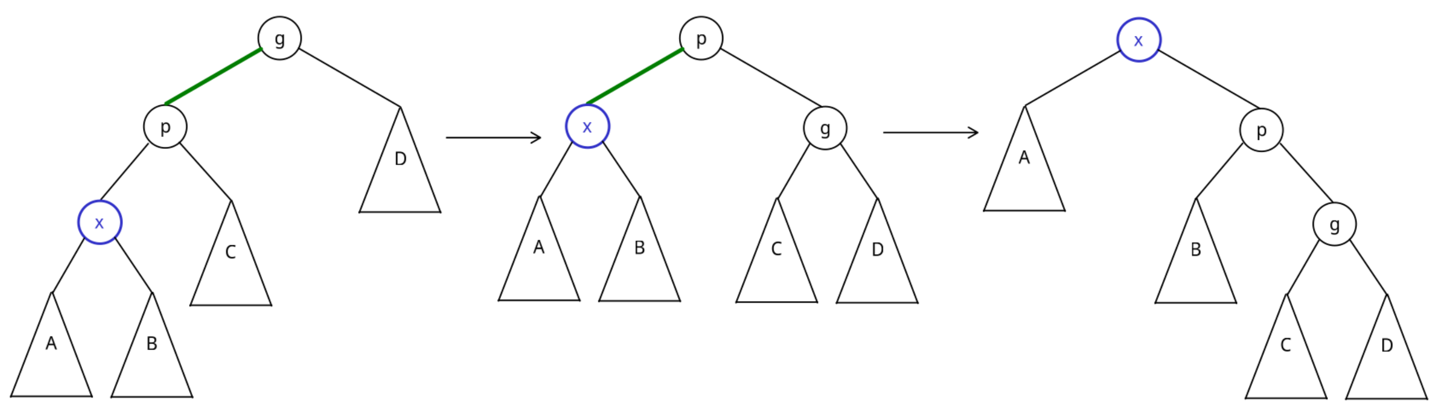


операция *зиг*

Преобразование дерева *зиг-зиг*

x — сын p, сына g, ориентация $(p,\ g)$ и $(x,\ p)$ совпадает.
Поднимем x на место g.

Повернём $(p,\ g)$, повернём $(x,\ p)$

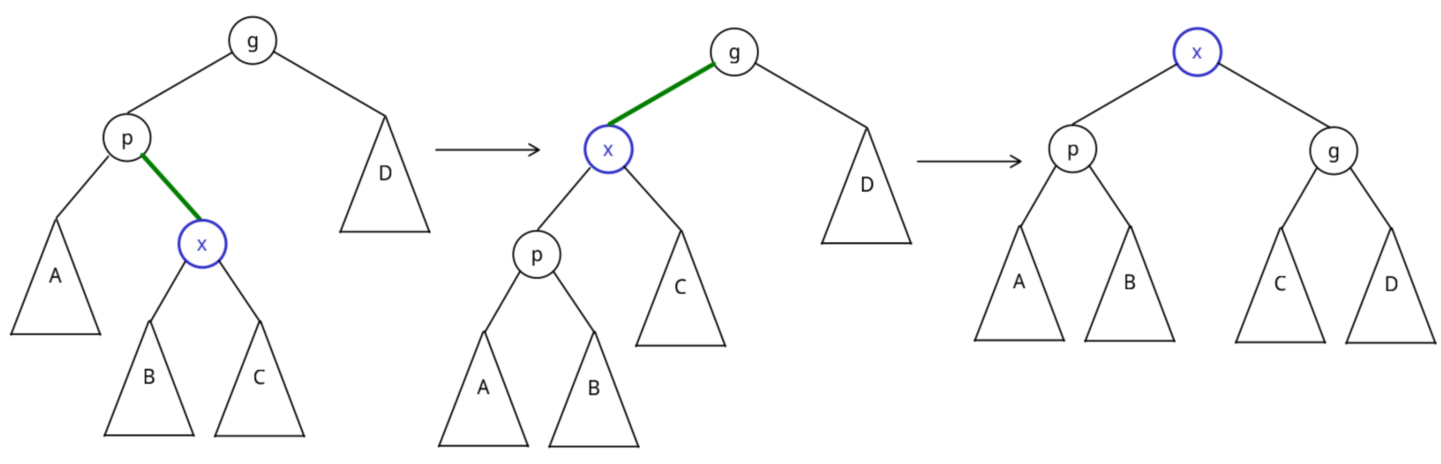


операция *зиг-зиг*

Преобразование дерева *зиг-заг*

x — сын p, сына g, ориентация $(p,\ g)$ и $(x,\ p)$ не совпадает.
Поднимем x на место g.

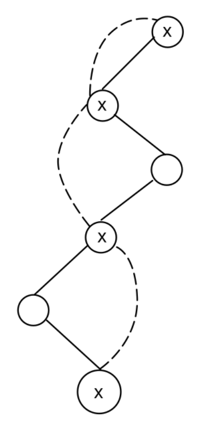
Повернём $(x,\ p)$, повернём $(x,\ g)$



операция *зиг-заг*

`splay(x)`

Находим x в дереве, идём снизу вверх, поднимаем x наверх, делая *зиг*, *зиг-зиг* или *зиг-заг*



$$T(\texttt{splay}) = O(H)$$

`find(x)`

В конце `find(x)` сделаем `splay(x)`.
$T(\texttt{find}) = T(\texttt{splay})$
Если мы не нашли x, сделаем `splay` от последней вершины, до которой дошли.

`insert(x)`

В конце `insert(x)` делаем `splay(x)`.
$T(\texttt{insert}) = T(\texttt{splay})$

Введём ещё две операции
- `merge(T1, T2)` — $\forall x \in T_1, y \in T_2 : x < y$
- `split(x)` $\longrightarrow (T_1, T_2) : \forall y \in T_1 : y \leq x, \forall y \in T_2 : y > x$

`merge(T1, T2)`

```
def merge(T1, T2):
  x = findMax(T1)
  splay(x)
  root1.right = T2
```

`split(x)`

```
def split(x):
  find(x)
  T2 = root.right
  T1 = root
  root.r = None
  return (T1, T2)
```

`remove(x)`

```
def remove(x):
  find(x)
  merge(root.left, root.right)
```

Доказательство асимптотики `splay(x)`

Утверждение: $\tilde{T}(\texttt{splay}) = O(\log n)$

Доказательство: воспользуемся методом потенциалов

$\tilde{T}(\text{op}) = T(\text{op}) + \Delta\varphi$

$s(v)$ — размер поддерева v

$r(v)$ — ранг вершины v

$r(v) \overset{\text{def}}{=} \log_2(s(v))$

$\varphi = \sum\limits_v r(v)$

Лемма:

$$\boxed{\tilde{T}(\texttt{splay(x)}) \leq 3(r(\text{root}) - r(x)) + 1}$$

1. $\tilde{T}(\textit{зиг}) = 1 + (r'(x) + r'(p) - r(x) - r(p))$, где r' — новый ранг, r — старый ранг

   $r'(p) < r(p)$

   $\tilde{T}(\textit{зиг}) = 1 + (r'(x) + r'(p) - r(x) - r(p)) \leq 1 + r'(x) - r(x) \leq 1 + 3(r'(x) - r(x))$

   Итог: $\tilde{T}(\textit{зиг}) \leq 1 + 3 \cdot (r'(x) - r(x))$

2. $\tilde{T}(\textit{зиг-зиг}) = 2 + (r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g))$

   $r'(x) = r(g)$

   $r(x) \leq r(p)$

   $r'(p) \leq r'(x)$

   $\tilde{T}(\textit{зиг-зиг}) = 2 + (r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g)) = 2 + (r'(p) + r'(g) - r(x) - r(p)) \leq$

   $\leq 2 + r'(p) + r'(g) - 2r(x) \leq 2 + r'(x) + r'(g) - 2r(x)$

   Утверждение: $2 + r'(x) + r'(g) - 2r(x) \leq 3(r'(x) - r(x))$

   Доказательство:

   $r'(x) + r'(g) - 2r(x) - 3r'(x) + 3r(x) \leq -2$

   $-2r'(x) + r'(g) + r(x) \leq -2$

   Рассмотрим: $(r(x) - r'(x)) + (r'(g) - r'(x))$.

   $\log_2 \frac{s(x)}{s'(x)} + \log_2 \frac{s'(g)}{s'(x)}$

   Заметим: $s'(g) + s(x) \leq s'(x)$

   $\underbrace{\frac{s'(g)}{s'(x)}}_{a} + \underbrace{\frac{s(x)}{s'(x)}}_{b} \leq 1$

   $a + b \leq 1$

   $\log_2 a + \log_2 b = \log_2(ab)$

   Итог: $\tilde{T}(\textit{зиг-зиг}) \leq 3(r'(x) - r(x))$

3. $\tilde{T}(\textit{зиг-заг}) \leq 3 \cdot (r'(x) - r(x))$ — домашняя работа 😊

Итого:

$\tilde{T}(\textit{зиг}) \leq 1 + 3 \cdot (r'(x) - r(x))$

$\tilde{T}(\textit{зиг-зиг}) \leq 3 \cdot (r'(x) - r(x))$

$\tilde{T}(\textit{зиг-заг}) \leq 3 \cdot (r'(x) - r(x))$

`splay(x)`: $r(x) \longrightarrow r'(x) \longrightarrow r''(x) \longrightarrow \ldots$

$3 \cdot (r'(x) - r(x)) + 3 \cdot (r''(x) - r'(x)) + 3 \cdot (r'''(x) - r''(x)) + \ldots$ — *телескопическая сумма*

$3 \cdot (\cancel{r'(x)} - r(x)) + 3 \cdot (\cancel{r''(x)} - \cancel{r'(x)}) + 3 \cdot (r'''(x) - \cancel{r''(x)}) + \ldots$

$3 \cdot r(\texttt{root}) - 3 \cdot r(x) + 1$

Заключение

Splay-дерево не является каким-то *странным* деревом. Мы не накладывали никакого дополнительного инварианта или ограничения на двоичное дерево поиска, как делали для AVL-дерева. Мы даже не рассматривали как выглядит Splay-дерево, потому что оно никак не отличается от обыкновенного дерева поиска, кроме операции `splay(x)` и может быть хоть бамбуком