

# Лекция 7. Деревья поиска II. Декартово дерево. Splay-дерево

#вшпи #аисд #теория #декартово\_дерево #структуры\_данных

Автор конспекта: Гридчин Михаил

## Сливающие деревья поиска

### Split/Merge и остальные операции

**Def.** *Split* принимает дерево поиска  $T$  и ключ  $x$  и разделяет на два дерева поиска, в первом все ключи  $< x$ , во втором  $\geq x$ .

**Def.** Операция *Merge* принимает  $T_1$  и  $T_2$ . При этом все ключи в  $T_1 \leq$  всех ключей в  $T_2$ . Она объединяет два дерева в одно дерево  $T$ .

Как сделать вставку элемента  $x$  в дерево  $T$ ?

- если  $x$  уже есть в  $T$ , то алгоритм окончен
- иначе делаем  $Split(T, x) \Rightarrow \{T_1, T_2\}$
- Возвращаем  $Merge(T_1, Merge(Tree(x), T_2))$

Как сделать *Erase* элемента  $x$  из дерева  $T$ ?

- $Split(T, x) \Rightarrow \{T_1, T_2\}$
- удаляем наивно  $x$  из  $T_1$  (у него только один ребёнок)
- $return \quad Merge(T'_1, T_2)$

Как сделать удаление  $x$  быстрее?

- Найдём  $x$  в дереве и вместо него запишем результат *Merge* его детей, если их два, иначе сделаем наивное удаление

## Декартово дерево поиска

**Def.** Декартовым деревом поиска называется бинарное дерево, содержащее пары  $\{x_i, y_i\}$ , при этом это двоичное дерево поиска по ключам и бинарная куча по приоритетам.

**Свойство:** декартово дерево поиска всегда можно построить единственным образом по данным парам  $\{x_i, y_i\}$ .

*Merge* в декартовом дереве деревьев  $T_1$  и  $T_2$ , где все ключи  $T_1$  меньше всех ключей  $T_2$ .

- Если  $T_1.root.y > T_2.root.y$ , то  $T.root = T_1.root$
- Значит известно левое поддерево  $T$ :  $T.root.left = T_1.root.left$
- $T.root.right = \text{Merge}(T_1.root.right, T_2)$
- Иначе действуем симметрично

### *Split*

- Если  $x > T.root.x$ , то известно левое поддерево  $T_1$ :  $T_1.left = T.left$
- Тогда  $\{T_1.right, T_2\} = \text{Split}(T.right, x)$
- Иначе действуем симметрично

**Утверждение.** Время работы *Split* и *Merge*  $\sim O(h)$ . Время работы *Erase*, *Insert* пропорционально максимуму из времени работы *Merge* и *Split*.

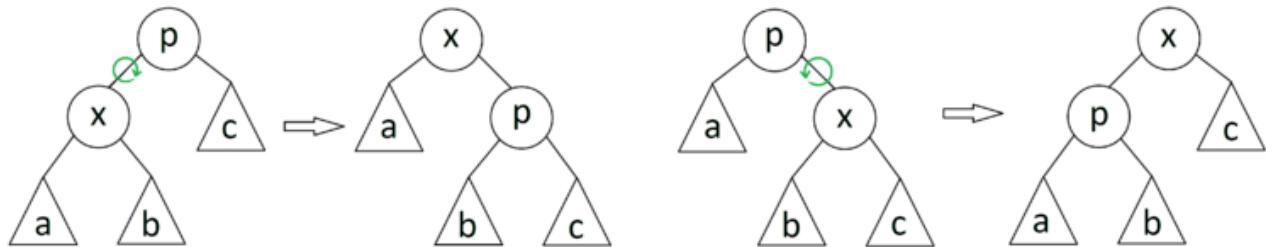
**Теорема** (б/д). В декартовом дереве из  $N$  узлов, приоритеты у которого являются случайными величинами с равновероятным равномерным распределением, средняя глубина вершины  $O(\log N)$

**Следствие.** Время работы *Erase*, *Insert* составляет  $O(\log N)$  в среднем.

## Splay дерево

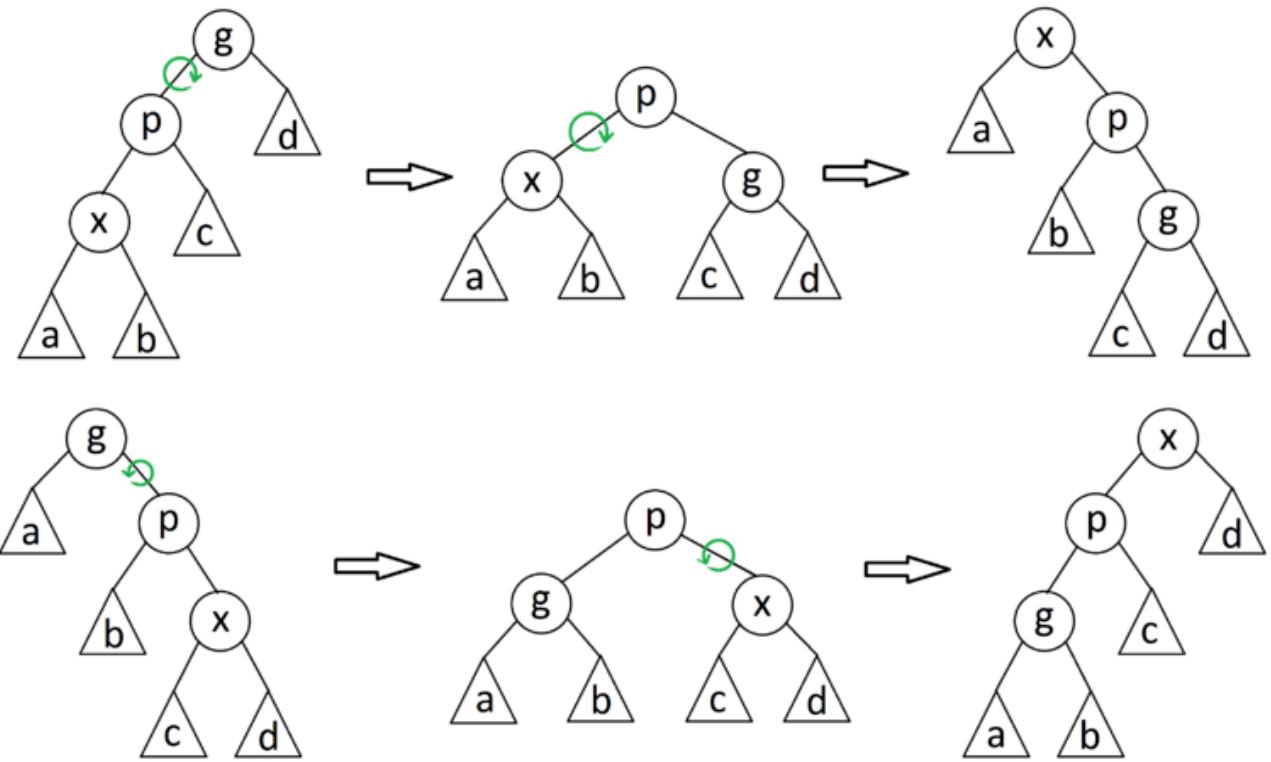
Пусть есть элементы, к которым обращаются с разной частотой. Есть "горячие элементы", к которым чаще обращаются и "холодные элементы", к которым реже обращаются. Хочется хранить "горячие элементы" выше, чтобы доступ к ним был быстрее, чем к "холодным". Можно доказать, что Splay-дерево - это теоретически лучшее дерево поиска. То есть утверждается, что оно теоретически оптимально с точки зрения теории информации.

**Def.** Операция **zig** применяется только для узлов на глубине 1 и делает поворот вокруг ребра  $\{x, p(x)\}$



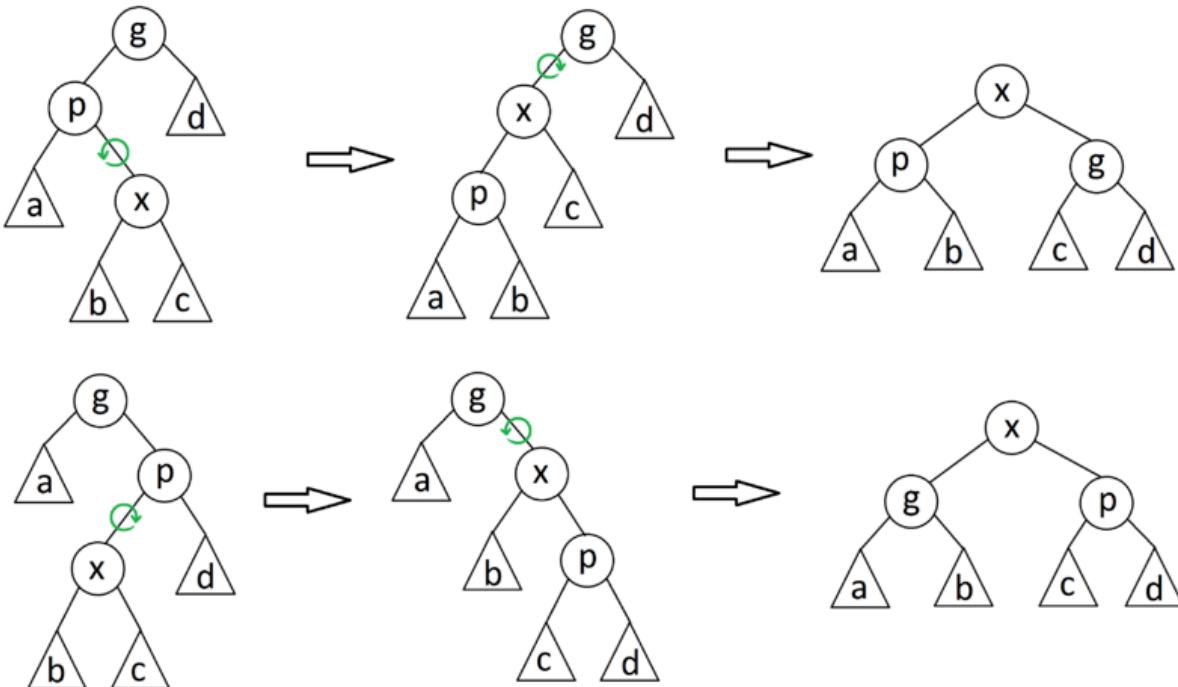
*splay-tree zig.png*

**Def.** Операция **zig-zig** применяется, если  $x$  - односторонний внук. Выполняется сначала поворот  $\{p, g\}$ , затем поворот вокруг ребра  $\{x, p\}$ .



splay-tree zig-zig.png

**Def.** Операция **zig-zag** применяется, если  $x$  - разносторонний внук. Сначала выполняется поворот  $\{x, p\}$ , затем поворот  $\{x, g\}$ .



splay-tree zig-zag.png

**Def.** Операция **Splay(x)** является комбинацией поворотов различного типа, чтобы сделать  $x$  корнем дерева. То есть Splay( $x$ ) поднимает вершину  $x$  в корень дерева, производя при этом повороты **zig**, **zig-zig**, **zig-zag**.

Другие операции в Splay-дереве:

$Merge(T_l, T_r)$

- выполняем  $Splay(T_l.\max)$
- Подвешиваем  $T_r$  как правый ребёнок  $T_l.\max$
- $Split(x)$
- выполним  $Splay(lower\_bound(x))$ ,  $lower\_bound$  как в наивной реализации
- Возвращаем  $x.left$  и  $x$
- $Find(x)$
- находим узел в этом дереве
- выполняем  $Splay(x)$
- $Insert(x)$
- наивно вставляем  $x$
- выполняем  $Splay(x)$
- $Erase(x)$
- делаем  $Find(x)$
- выполняем  $Merge$  от его детей.
- удаляем  $x$

Доказательство времени работы  $Splay$

**Def.** Рангом вершины  $x$  назовём величину  $r(x) = \log_2 C(x)$ , где  $C(x)$  - размер поддерева вершины  $x$  включая её. Заметим, что  $r(x) \geq 1$ .

**Def.** Потенциалом Splay-дерева  $T$  назовём величину

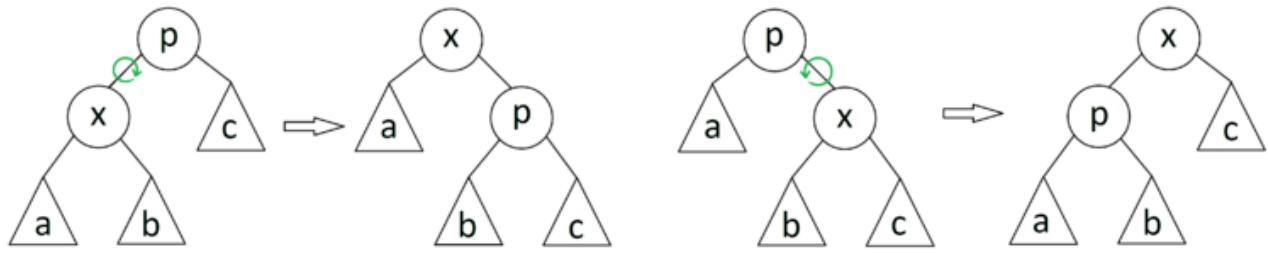
$$\phi(T) := \sum_{v \in T} r(v)$$

**Note.** далее  $r$  и  $r'$  - ранги до и после преобразований

**Теорема (время работы Splay):** Амортизированное время поворотов в  $Splay(x)$  с корнем в  $t$  не превосходит  $3r(t) - 3r(x) + 1$ . Амортизированное время работы  $Splay(x)$  равно  $O(\log N)$

Доказательство:

**Операция zig**



splay-tree zig.png

Поскольку выполнен один поворот, то амортизированное время выполнения шага равно

$$T = 1 + r'(x) + r'(p) - r(x) - r(p)$$

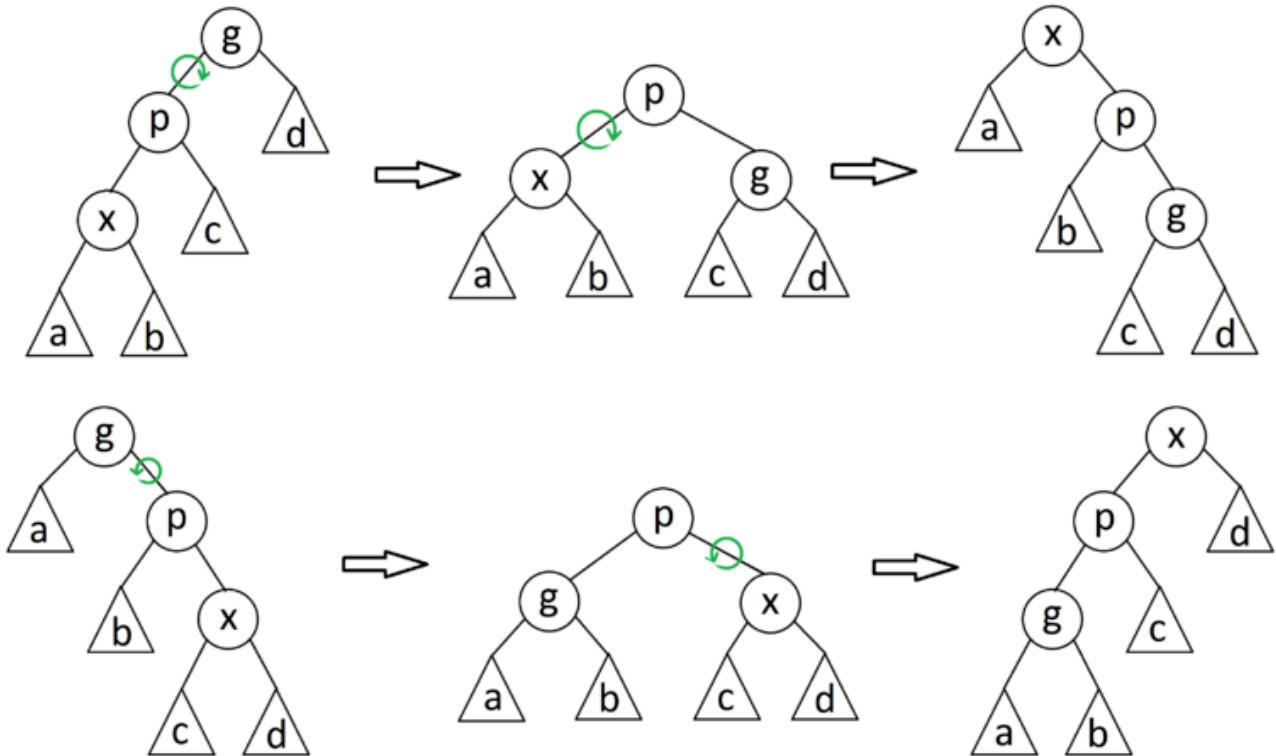
$r(p)$  уменьшился, поэтому  $T \leq 1 + r'(x) - r(x)$  (т.к.  $r'(p) - r(p) \leq 0$ )

$r(x)$  увеличился, поэтому  $r'(x) - r(x) \geq 0$

Поэтому умножим на 3, не меняя знака

$$T \leq 1 + 3r'(x) - 3r(x) \implies Q.E.D$$

### Операция zig-zig



splay-tree zig-zig.png

Выполнено два поворота. Поэтому амортизированное время выполнения шага равно

$$T = 2 + r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g)$$

Заметим, что  $r'(x) = r(g)$ . Тогда

$$T = 2 + r'(p) + r'(g) - r(x) - r(p)$$

Далее, так как  $r(x) \leq r(p)$ , получаем, что  $T \leq 2 + r'(p) + r'(g) - 2r(x)$ .

$r'(p) \leq r'(x)$ , получим, что  $T \leq 2 + r'(x) + r'(g) - 2r(x)$ .

Хотим доказать, что  $T \leq 2 + r'(x) + r'(g) - 2r(x) \leq 3(r'(x) - r(x))$ , то есть

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

Это равносильно

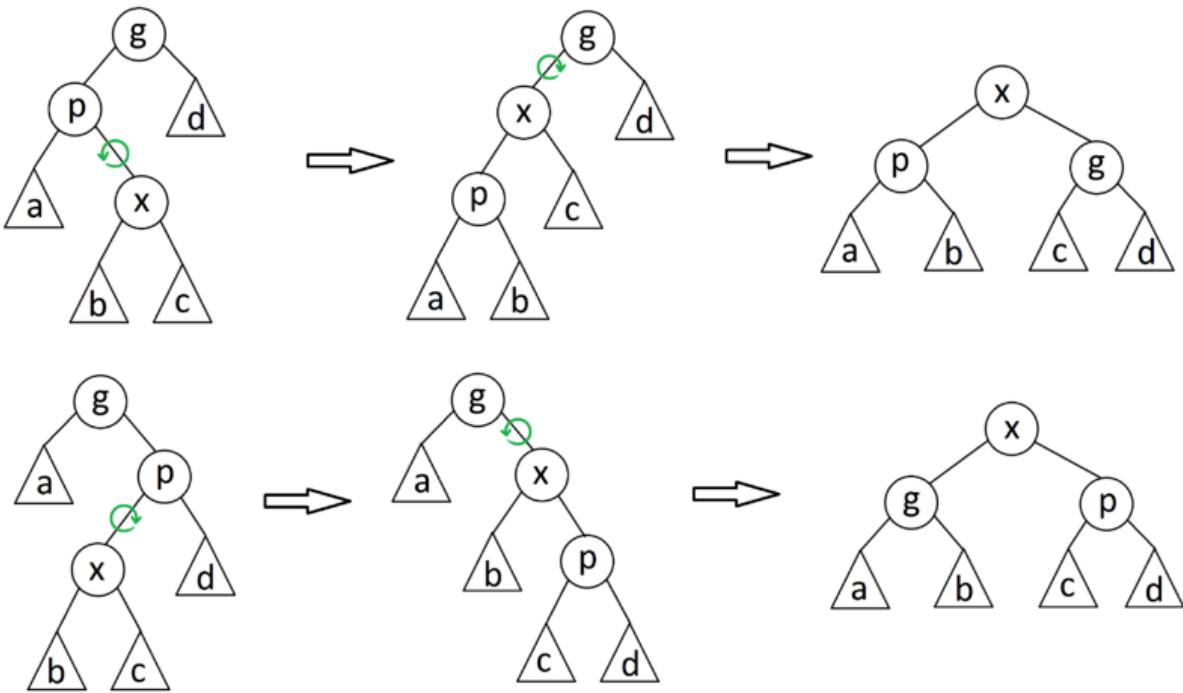
$$(r(x) - r'(x)) + (r'(g) - r'(x)) = \log_2 \frac{C(x)}{C'(x)} + \log_2 \frac{C'(g)}{C'(x)} = \log_2 \frac{C(x)C'(g)}{C'(x)^2} \leq -2$$

Заметим, что

$$\begin{cases} C'(g) + C(x) \leq C'(x) \iff (\frac{C'(g)+C(x)}{2})^2 \leq \frac{C'(x)^2}{4} \\ \sqrt{C'(g)C(x)} \leq \frac{C'(g)+C(x)}{2} \iff C'(g)C(x) \leq (\frac{C'(g)+C(x)}{2})^2 \end{cases} \implies C'(g)C(x) \leq \frac{C'(x)^2}{4}$$

Откуда  $\frac{C'(g)C(x)}{C'(x)^2} \leq \frac{1}{4} \iff \log_2 \frac{C(x)C'(g)}{C'(x)^2} \leq -2 \implies Q.E.D.$

### Операция zig-zag



splay-tree zig-zag.png

Выполнено два поворота, амортизированное время равно

$$T = 2 + r'(x) + r'(p) + r'(g) - r(p) - r(x) - r(g)$$

Заметим, что  $r'(x) = r(g)$ . Тогда

$$T = 2 + r'(p) + r'(g) - r(x) - r(p)$$

Далее, так как  $r(x) \leq r(p)$ , получаем, что  $T \leq 2 + r'(p) + r'(g) - 2r(x)$ .

Хотим доказать, что  $T \leq 2 + r'(p) + r'(g) - 2r(x) \leq 2(r'(x) - r(x))$ , то есть, что  $r'(p) + r'(g) - 2r'(x) \leq -2$

Это равносильно

$$(r'(p) - r'(x)) + (r'(g) - r'(x)) = \log_2 \frac{C'(p)}{C'(x)} + \log_2 \frac{C'(g)}{C'(x)} \leq -2$$

Заметим, что  $C'(p) + C'(g) \leq C'(x)$ , применение неравенства о средних аналогично операции **zig-zig**  $\implies Q.E.D.$

### Соединяем

Из рассуждений выше операции **zig-zig** и **zig-zag** удовлетворяет соотношению

$$T \leq 3r'(x) - 3r(x).$$

Пусть ключ  $x$  стартует с узла  $x_0$  в дереве и проходит положения  $x_1, \dots, x_k$  до операции **zig**.

Тогда получается, что  $x_{k+1} = t$ . Откуда время работы  $Splay(x)$  равно

$$\begin{aligned} T(Splay) &\leq \sum_{i=0}^{k-1} (3r(x_{i+1}) - 3r(x_i)) + T(zig) = 3r(x_k) - 3r(x_0) + T(zig) \leq \\ &\leq 3r(t) - 3r(x_0) + 1 = O(\log N) \implies Q.E.D \end{aligned}$$