Treap

Билет № ?

11 декабря 2025 г.

1 Определение

Декартово дерево. Пусть дан набор пар (x_1, y_1), ... $(x_i, y_i) \in \mathrm{R}^2$, где будем называть x_k ключом, а y_k - приоритетом. Тогда **декартовым деревом** назовем струтуру данных, которая представляет собой бинарное дерево поиска по ключам и кучу по приоритетам.

2 Основные операции

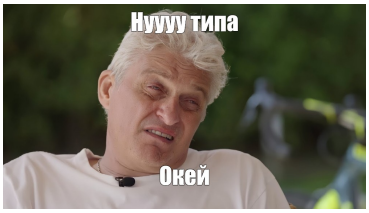
Для такой структуры данных вводится две операции. SPLIT и MERGE.

2.1 Split

Пусть у нас есть некоторое число x и декартово дерево T. Мы хотим расцепить его на T_l, T_r так, что все ключи левого дерева не больше x, а в правом - строго больше x.

2.1.1 Split Algorithm

Утверждение. Split работает за $O(h(T))$, так как мы обходим дерева от корня до листа по одному пути: на каждом шаге идём либо влево, либо вправо.



Algorithm 1 Split(T, k) — разделение декартова дерева

```
1: function SPLIT(x, T)
2:     if T = null then
3:         return (null, null)
4:     end if
5:     if T.key > x then
6:         (T'_l, T'_r) ← SPLIT(x, T_l)
7:         T_l = T'_r        ▷ Переподвязываем часть поддерева, которая > x
8:         return (T'_l, T)
9:     else
10:        (T'_l, T'_r) ← SPLIT(x, T_r)
11:        T_r = T'_l
12:        return (T, T'_r)
13:    end if
14: end function
```

2.2 Merge

Надо бы уметь не только разрушать для баланса вселенной. Имеем два декартовых дерва T_l и T_r, хотим их слить.

2.2.1 Merge Algorithm

Примечание. Merge работает при условии, что все ключи левого дерева $<$ ключей правого дерева. Предусловие: $\forall x \in \mathrm{T}^L$, $\forall y \in \mathrm{T}^R$ выполнено, что x.key $<$ y.key. ИНАЧЕ MERGE НЕ РАБОТАЕТ!

Инвариант: На каждом шаге мы продвигаемся хотя бы на один узел вглубь одного из деревьев. T(h_1, h_2) <= O(h_1) + O(h_2).

2.3 Операции, которые порождают Split и Merge

2.3.1 Search(x)

Поиск ключа выполняется как в обычном дереве поиска, потому что ДД и есть дерево поиска.

2.3.2 Insert((x, y))

1. Split(x) -> T^L, T^R
2. Merge(T^L, (x, y)) -> $T^{L\prime}$
3. Merge($T^{L\prime}$, T^R)

Algorithm 2 Merge(T^L, T^R)

```
1: function MERGE(T^L, T^R)
2:     if T^R = null then
3:         return T^L
4:     end if
5:     if T^L = null then
6:         return T^R
7:     end if
8:     if T^L.p > T^R.pr then
9:         T^L_r ← Merge(T^L_r, T^R)
10:        return T^L
11:    else
12:        T^R_l ← Merge(T^L, T^R_l)
13:        return T^R
14:    end if
15: end function
```

2.3.3 Delete((x, y))

1. Split(x) -> (T^L, T^R)
2. В T^L находится (x, y). Более того, это самая правая вершина. У самый правой вершины, очевидно, нет правого ребенка. Просто удаляем x, а на его место подвешиваем его левого сына. Далее сливаем два получившихся дерева.

3 Построение

Пусть у нас даны пары (x_1, y_1), ... (x_n, y_n), причем $\forall i, j \in$N: i $<$ j $\Rightarrow$ $x_i < x_j$. Будем поочередно вставлять пары. В силу возрастания $\{x_k\}$ последний вставленный элемент будет самым правым. Будем пытаться вставлять постоянно вправо. Если у нас несогласованны приоритеты, то идем вверх, пока не встретим место, где они будут удовлетворять свойству кучи. Тогда делаем нашу новую ноду правым сыном, а старое правое поддерево делаем левым сыном поддеревом новой ноды. Если доходим до корня, то старый корень делаем левым поддеревом новой ноды. (см. пример в Семинар_Декартово_Дерево_-7.pdf)

- **Утверждение.** Работает за $O(n)$.

- **Утверждение.** Если приоритеты выбраны из равномерного распределения, то в среднем глубина декартового дерева равна $O(log(n))$.

4 Implicit Treap

В обычном декартовом дереве у каждого узла есть явный ключ (число, строка), по которому строится бинарное дерево поиска.

1. Здесь нет явных ключей
2. Порядок определяется позицией в дереве (как в массиве)
3. Вместо ключа - размер левого поддерева
4. Все операции аналогичны ДД по явному ключу

5 Задача о развороте подотрезка

Task. Как разверуть подозрезок за O(log(n)) в среднем?

1. Будем хранить в ноде булевский flag, означающий, что нода нуждается в развороте ее детей
2. Вырежем дерево [l, r]:
   (a) Split(всё дерево, l). Получим два дерева: одно с индексами [0, l-1] (L), второе [l, n]. Берём второе (M + R).
   (b) Split(M + R, r - l + 1). Получаем дерево на [l, r]. Назовем его M, а оставшееся - R.
   (c) Ставим в корень M flag = true
   (d) Будем разворачивать детей по мере обращения к их родителю (при этом, если развернули детей -> flag = false для родителя и flag = true для детей)
3. merge(L, M) = L+M
4. merge(L+M, R) = T

Примечание. Разворот подотрезка работает за 2 split + 2 merge + поставить flag = O(log n)