Алгоритмы и Структуры Данных. Лекция 5

13.03.2024

_scarleteagle **AberKadaber**

Декартово дерево

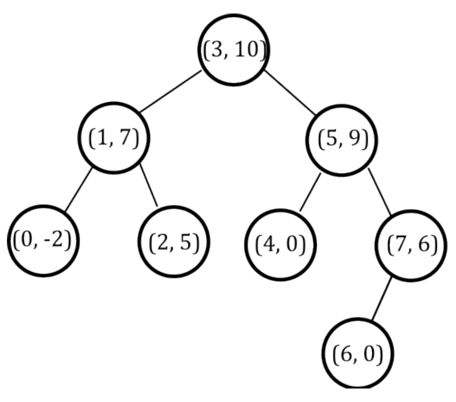
Мы уже рассмотрели два хороших варианта деревьев, а декартово дерево будет работать на рандом

— Михаил Первеев

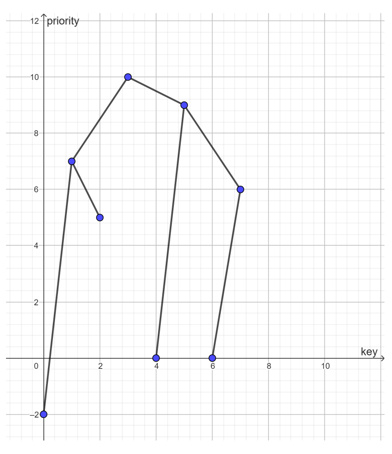
Будем хранить:
- `key` — ключ (BST)
- `priority` (По ним выполняются свойства кучи)

Вершина выглядит как `(key, priority)`
Команды:
- `insert(t, x)`
- `find(x)`
- `remove(t, x)`
- `split(t, x)`
- `merge(l, r)`

По английски это дерево называется Treap = Tree + Heap или Cartesian Tree
По русски иногда называется:
1. Пиво = Пирамида + Дерево
2. Курево = Куча + Дерево
3. Дуча = Дерево + Куча
4. Дерамида = Дерево + Пирамида



Декартово дерево



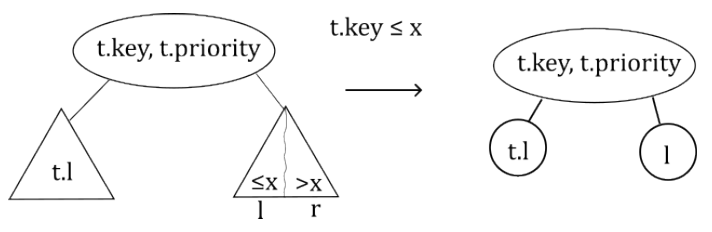
Декартово дерево в декартовой системе координат

Как балансировать?
$k_1, k_2, ..., k_n$ — ключи
$p_1, p_2, ...p_n$ — случайные целые числа (попарно различные)

Для начала т.к. по приоритетам выполняются свойства кучи элемент с максимальным приоритетом в корень
Теперь разделим оставшиеся вершины на те у которых ключ меньше и те у кого ключ больше чем у корня и запустимся рекурсивно от каждого из двух получившихся множеств
Заметим что такое построение практически такое же как сортировка QuickSort, поэтому асимптотика времени работы будет $O(n \log(n))$. Также мы знаем что матожидание глубины рекурсии QuickSort $O(\log(n))$, а в терминах дерева это означает что матожидание глубины дерева равна $O(\log(n))$

Теперь научимся делать операции `split` и `merge`

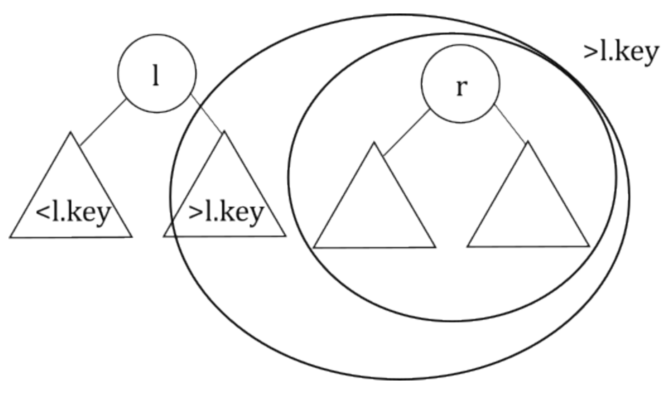
```python
def split(t: Node, x: int): # на L <= x, R > x
  if t is None:
    return (None, None)
  if t.key <= x:
    l, r = split(t.r, x)
    t.r = l
    return t, r
  else:
    l, r = split(t.l, x)
    t.l = r
    return (t, l)
```



Раньше мы проталковали в детей, теперь мы их режем на куски. Ютуб не блокируй пожалуйста

— Михаил Первеев

```python
def merge(l: Node, r: Node) -> Node:
  if l is None:
    return r
  if r is None:
    return l
  if l.priority > r.priority:
    l.r = merge(l.r, r)
    return l
  else:
    r.l = merge(l, r.l)
    return r
```



Тут в чате пишут что сейчас сделаем смешную нарезку детей...

— Михаил Первеев

Работает за $O(\log(n))$ в среднем (в средем здесь $\neq$ амортизировано, а = матожидание)

```python
def insert(t: Node, x: int):
  l, r = split(t, x)
  t = merge(merge(l, x), r)
```

Круто, работает за $O(\log(n))$, но при этом константа достаточно большая, как минимум 3, а на самом деле больше

```python
def remove(t: Node, x: int):
  m, r = split(t, x)
  l, m = split(m, x - 1)
  t = merge(l, r)
```

Круто, это тоже работает за $O(\log(n))$, но опять же с большой константой

Декартово дерево по неявному ключу

Пусть есть некоторый массив и мы хотим научиться делать на нем операции типа циклически сдвинуть
Для этого представим его в виде декартового дерева:
1. ~~`key` = индекс элемента в массиве(`i`)~~
   `size` = размер поддерева
2. `priority`
3. `value` = значение элемента (a_i)

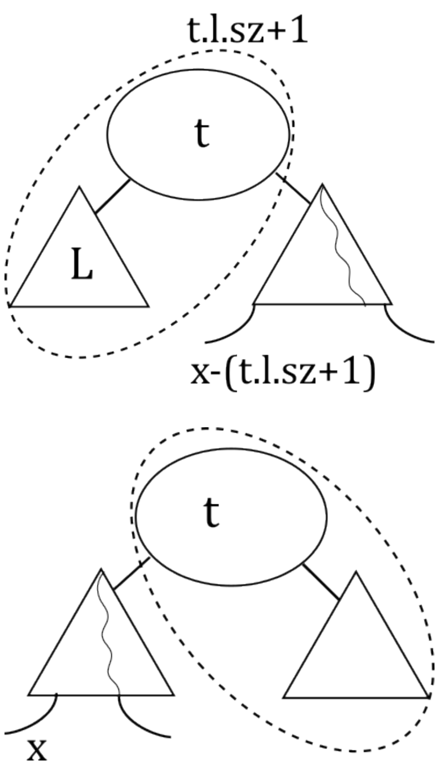
`split(x)` разделит нам дерево на два: первый с индексами от 0 ... x, второй с индексами $x + 1$ ... n
После этого делаем `merge()` и все ломается
Чтобы этого не происходило сделаем фукнцию update и вместо ключа будем хранить размер поддерева

```python
def update(t: Node)"
  t.sz = 1
  if t.l != None:
    t.sz += t.l.sz
  if t.r != None:
    t.sz += t.r.sz
```

при `merge()` произведем `update()` перед каждым `return`

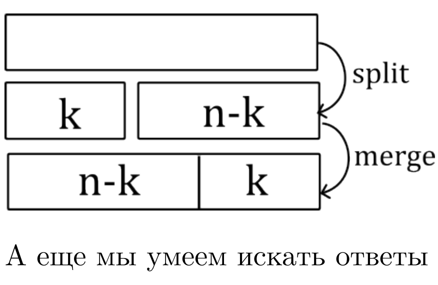
Теперь поменяем `split`

```python
def split(t: Node, x: int):
  if t is None:
    return (None, None)
  if t.l.sz + 1 <= x:
    l, r = split(t.r, x-(t.l.sz+1))
    t.r = l
    update(t)
    return (t, r)
  else:
    l, r = split(t.l, x)
    t.l = r
    update(t)
    return (l, t)
```



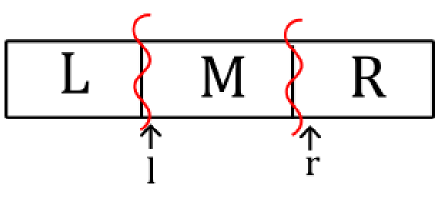
Победа, теперь мы можем сделать циклический сдвиг массива, причем за $O(\log(n))$

```python
t = None
# a_0, a_1, ... a_n
for i in range(n):
  t = merge(t, Node(a[i]))
l, r = split(t, k)
t = merge(r, l)
```



А еще мы умеем искать ответы на запросы на отрезке:

```python
L, M = split(t, l)
M, R = split(M, r - l)
print(M.min) # здесь мы добавили поле min - минимум в поддереве
```



Также можно сделать и отложеные операции на отрезке

Итог: мы умеем делать кучу интересных операций, даже за $O(\log(n))$, но к сожалению константа достаточно большая