Алгоритмы и Структуры Данных. Лекция 1

07.02.2024

_scarleteagle **imkochelorov**

На следующей лекции будем пропихивать в детей

— Первеев Михаил Валерьевич

Дерево отрезков (Segment tree)

Вступление

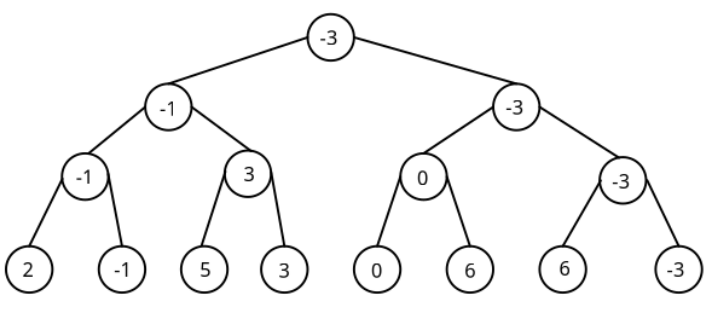
Пусть имеется массив `a = [1 -2 5 8 7 3 6]`
Имеются запросы вида

- l, r → `return a[l] + a[l + 1] + ... + a[r] # преф. суммы за O(1)` — часто называют RSQ (Range Summ Query)
- p, x → `a[p] = x # можем изменять наши преф. суммы за O(p), но потеряем всю скорость`
- l, r → `return min(a[e], a[e+1], ..., a[r])` — часто называют RMQ (Range Minimum Query)

Пример:
Неоходимые операции
1. `min(l, r)` — RMQ
2. `change(p, x)` — RSQ

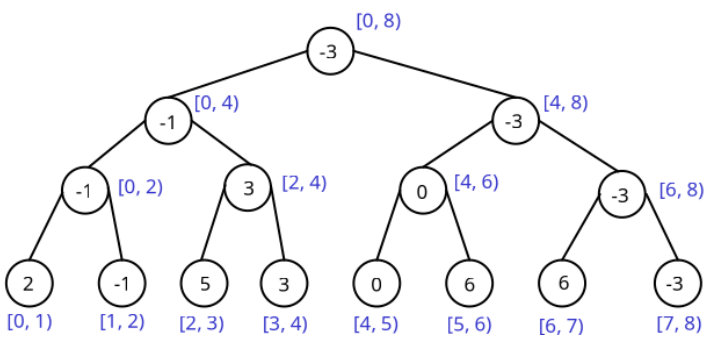
Массив имет вид:
`a = [2, -1, 5, 3, 0, 6, 6, -3]`

Построим дерево, хранящее минимумы на отрезках:



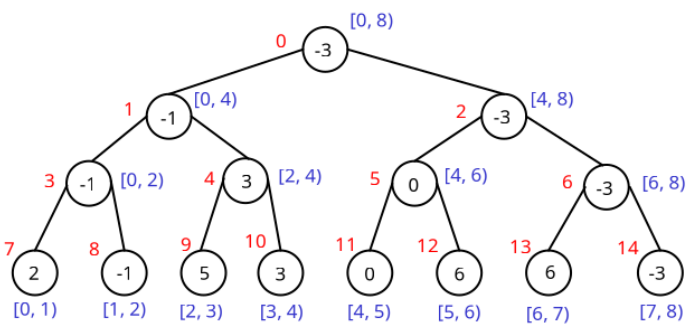
визуальное представление дерева отрезков

Обозначим, какой уровень дерева минимумы какой части массива хранит (полуинтервалами):



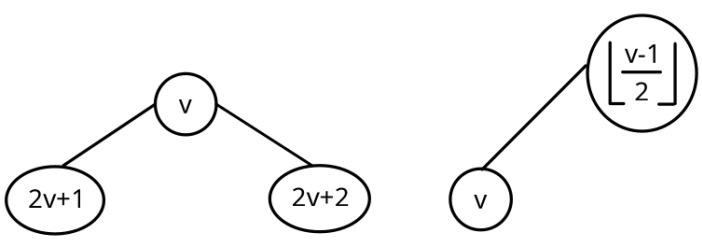
синим цветом выделены обозначениями

Пронумеруем вершины:



красным цветом выделены нумерация

Такая нумерация удобна, так как элемент с номером v имеет сыновей с номерами $v \cdot 2 + 1$ и $v \cdot 2 + 2$, а также родителя с номером $\lfloor \frac{v-1}{2} \rfloor$



$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... + 1 < 2n$, весь массив занимает $O(n)$ памяти

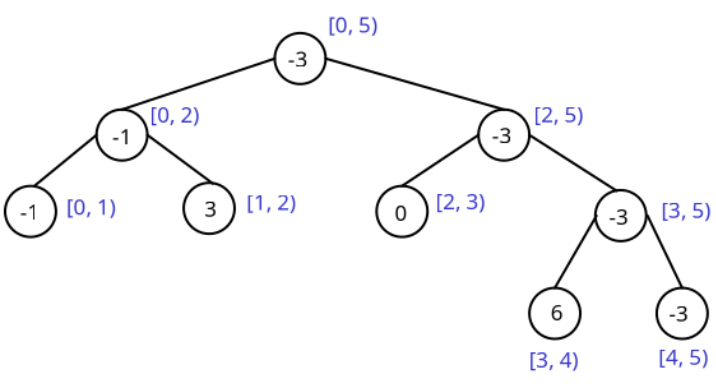
Реализация

Рекурсивное построение дерева из массива:

```
a # исходный массив
t # дерево отрезков

def build(v, l, r): # первый запуск от корня до конца массива, строит дерево отрезков за O(n)
  if r - l == 1: # находимся в листе
    t[v] = a[l]
    return
  m = (l + r) // 2
  # запускаемся по детям
  build(v * 2 + 1, l, m)
  build(v * 2 + 1, m, r)
  t[v] = min(t[v * 2 + 1], t[v * 2 + 2])
```

Построение дерева отрезков корректно и в случае, когда размер массива не равен степени

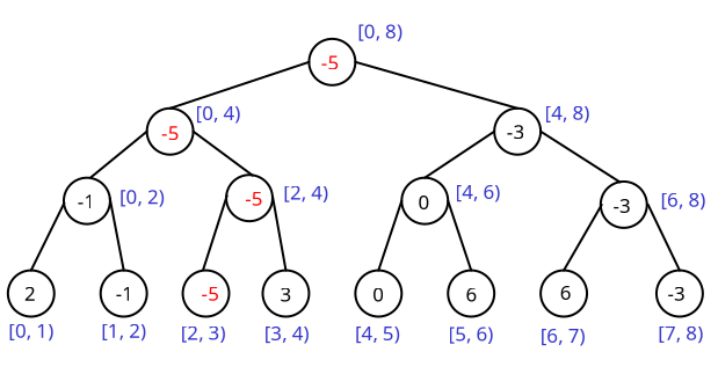


Но необходимо быть аккуратным с нумерацией вершин, так как в худшем случае последний элемент дерева будет иметь номер $2^{k+2} - 2$, что всё ещё меньше $4n$

Операции изменения массива:

```
def change(v, l, r, p, x): # первый запуск при l = 0 и r = n, работает за O(logn)
  if r - l == 1:
    t[v] = x
    return
  m = (l + r) // 2
  if p < m:
    change(v * 2 + 1, l, m, p, x)
  else:
    change(v * 2 + 2, m, r, p, x)
  t[v] = min(t[2 * v + 1], t[2 * v + 2])
```

Пример запроса:
`change(2, -5)`



красным выделены изменённые запросом вершины

Операция нахождения минимума на отрезке (RMQ):

```
MAX_INT = 1e12
def get(v, l, r, ql, qr):
  if ql >= r or qr <= l:
    return MAX_INT
  if ql <= l and r <= qr:
    return t[x]
  m = (l + r) // 2
  return min(get(v * 2 + 1, l, m, ql, qr), get(v * 2 + 1, m, r, ql, qr))
```

Что такое плюс бесконечность в коде сами разбирайтесь, у меня доска, мне пофиг

— Первеев Михаил Валерьевич

Для оценки времени работы докажем несколько фактов:
1. $\forall [ql,\ qr)$ можно разбить на $\leq 2 \cdot \log_2 n$ вершин в дереве отрезков

Доказательство:
~~очевидно~~

2. Функция get делает $\leq 4 \cdot \log_2 n$ рекурсивных вызовов

Доказательство:
1) База: корень — 1, вызов $1 < 4$
2) Переход: $n \to n + 1$: т.к. у отрезка 2 конца, “худших случаев” может быть не более 2, поэтому ≤ 4 рек. вызовов $\Rightarrow$ ≤ 4 рек. вызовов

Созданная нами структура легко изменяема для выполнения запросов с любыми другими функциями, имеющими нейтральный элемент, свойства ассоциативности и “*аддитивности*”

Напоследок

Задача:
Дан массив. Необходимо уметь менять элемент и находить самый левый элемент, не превосходящий x
Решение:
Построим ДО на минимум. Смотрим минимум слева от половины. Если он больше x, то ответ в левой половине, иначе пойдем направо.