Конспект по алгоритмам и структурам данных
II семестр

Коченюк Анатолий

20 февраля 2021 г.

Глава 1

Новые структуры данных

1.1 Дерево отрезков

Segment Tree, Range Tree, Interval Tree – можно наткнутся на другие структуры

Есть массив a на n элементов

1. set(i,v) $\quad a[i] = v$
2. sum(l,r) $\quad \sum_{i=l}^{r-1} a[i]$

$[3, 5, 2, 1, 6, 4, 8, 3]$

Построим дерево, где в каждом узле сумма двух под ним. Каждое число это сумма чисел на каком–то отрезке.

Заменим 4 на 7. set(5, 7). Нужно пересчитать все узлы, которые выше него. Их логарифм.

Теперь к сумме. Сумму от l до r мы будем раскладывать на суммы, которые мы уже знаем.

1. Как найти эти суммы: обойдём наше дерево рекурсивно, не заходя в плохие поддереве. Идём вниз: если в поддереве нет нужных элементов, выходим, если полностью содержится в нужном, берём эту сумму, иначе идём дальше вниз.
2. Мы обошли не так много (порядка логарифма) узлов. Посмотрим на узлы, в которых не сработало отсечение. Тогда наш отрезок содержит одну из границ отрезка. Отрезков, которые содержат границу (правую или левую) не более $2 \log n$.

Запуск рекурсии: узлов, в которых не сработало отсечение – $2 \log n$, всего узлов $\approx 4 \log n$.

Будем хранить полное двоичное дерево. У узла i дети $2i + 1$ и $2i + 2$

```
set(i, v, x, lx, rx):
    if rx - lx = 1:
        tree[x] = v
    else:
        m = (lx+rx)/2
        if i < m:
            set(i,v,2x+1,lx,m)
        else:
            set(i,v,2x+2,m, rx)
        tree[x] = tree[2x+1] + tree[2x+2]

sum(l, r, x, lx, rx):
    if l >= rx || lx >= r:
        return 0
    if lx >= l && rx <= r:
        return tree[x]
    m = (lx+rx)/2
    s1 = sum(l, r, 2x+1, lx, m)
    s2 = sum(l, r, 2x+2, m, rx)
    return s1 + s2
```

Пусть мы теперь хотим посчитать минимум. Делаем то же самое, только в каждом узле храним не сумму на отрезке, а минимум. Только в случае пустого дерева вместо 0 в сумме надо вернуть $+\infty$ (нейтральный элемент по операции)

Если нужно сделать операцию $a \bigotimes b$. Если у неё есть ассоциативность $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, то её можно встроить в дерево отрезков.

$max, +, \cdot, \&, |,$ НОД, НОК

1.1.1 Снизу вверх

Занумеруем также, но будем идти снизу вверх и без рекурсии.

```
set(i, v): // n = 2^k
    x = i + n - 1
    delta = v - tree[x]
    while x >= 0:
        tree[x]+=delta
        x = (x+1)/2-1

sum(l, r):
```

```
l = n-1+r
r = n-2+r
res = 0
while r >= l:
    if l % 2 = 0:
        res += tree[l]
    l = l / 2
    if r % 2 = 1:
        res += tree[r]
    r = r/2-1
return res
```

Если нужно посчитать другую функцию, поменяется нейтральный элемент. Если не коммутативная функцию, можно сначала считать левую, потом правую, а потом их сложить.

Такая реализация чуть лучше работает как $o(\log(l-r))$. Каждый раз разница между l и r уменьшается в 2 раза.

1.2 Персистентное дерево отрезков

```
   15
  8   7
 3 5 6 1
```

set(2,8) Сделаем новый узел рядом с 6. Рядом с узлом 7 на верхнем слое приделаем ещё один узел 9. А к корню приделаем узел 17. Так у нас появилось две параллельные версия дерева отрезков.

1.3 Дерево отрезков v.2

Теперь мы хотим:

1. $add(l, r, v) \quad a[i] += v \quad i = l \dots r-1$
2. $get(i) \quad return\ a[i]$

Построим дерево отрезков над массивом как в прошлый раз. В начале везде запишем нолики. Хотим добавить 3 к отрезку. Разобъём его на кусочки и в верхние узлы над нужным отрезком запишем тройки.

```
add(l, r, v, x, lx, rx):
    if (lx >= r) || (l >= rx):
        return
    if lx >= l && rx <= r:
        tree[x] += v
        return
    m = (lx+rx)/2
```

```
    add(l, r, v, 2x+1, lx, m)
    add(l, r, v, 2x+2, m, rx)
```

Сделаем $modify(l, r, v)$ $a[i] = a[i]\bigstar v$, где $\bigstar$ ассоциативно и коммутативно.

Как жить с некоммутативными операциями? Записывать порядок. Точнее при получении значения пропихиваем верхние значения вниз справа.

```
propogate(x):
    tree[2x+1]+=tree[x]
    tree[2x+2]+=tree[x]
    tree[x] = 0
```

Пример (Присваивание)**.** set – задаёт значение на отрезке. код такой же

```
propogate(x):
    if tree[x] != NO_OPERATION
```

Теперь сделаем две операции: прибавление и минимум, оба на отрезке.

Сначала сделаем дерево на минимум. Дальше при добавлении находим нужные отрезки и запоминаем, что нужно добавить 3 в узлах. минимумы снизу мы пересчитывать не будем

Когда доходим до узла добавляем v в два массива: добавочный и минимумов + обвнояляем минимум, при спуске: $tm[x] = min(tm[2x+1], tm[2x+2]) + ta[x]$

```
propogate

min(l, r, x, lx, rx):
...
...
...
s1 = min(l, r, 2x+1, lx, m)
s2 = min(l, r, 2x+2, m, rx)
return min(s1,s2) + ta[x]
```

Здесь мы пользовались дистрибутивностью при пересчёте. На самом деле, можно для недистрибутивных (например двух плюсов) сделать её такой, или помнить как именно нужно изменить