Конспект по алгоритмам и структурам данных
II семестр

Коченюк Анатолий

3 марта 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) $\quad \sum\limits_{i=l}^{r-1} a[i]$

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

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.

$max, +, \cdot, \&, |, \text{НОД}, \text{НОК}$

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. $add(l, r, v) \quad a[i] += v \quad i = l \ldots r-1$
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 $modify(l, r, v) \quad a[i] = a[i] \bigstar v$, где ★ 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

1.4 Дерево Фенвика

Задача 1. Есть массив

Нужно:

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

Оба за логарифм

Казалось бы всё это умеет дерево отрезков.. Но можно лучше! (всё ещё за логарифм, но)

$f(i) = \sum\limits_{j=p(i)}^{i} a[j]$

$sum(l, r) = sum(l) - sum(r)$

Как считается $sum(i)$: В последнем элементе лежит какая-то сумма. Добавим её к ответу. В элементе до этой суммы (префикса) лежит какая-то сумма. Добавим её к ответу... Так, пока не дойдём до начала массива.

inc(i, v): ищем все суммы, содержащие i и увеличиваем их на v

По итогу нам нужна хорошая $p(x)$, чтобы и отрезков в $sum(i)$ было поменьше (логарифм), и отрезков, содержащих i было тоже логариф.

$x: \quad \max k : (x+1) \vdots 2^k \quad p(x) = x + 1 - 2^k$ (В терминах двоичной записи все единички идущие подряд в конце числа становятся нулями.

$p(x)$ довольно просто найти, это $x \& (x+1)$

```
sum(l,r):
    return sum(r) - sum(l)

sum(r):
    x = r - 1
    res = 0
    while x >= 0:
        res += f[x]
        x = (x&(x+1))-1
    return res
```

Для инкремента нужно найти все такие $j \quad p(j) \leqslant i \leqslant j$

j – префикс, 0 и последовательность единиц. $p(j)$ – префикс, 0 и последовательность нулей.

Тогда i, между ними, – префикс, ноль и что-угодно. Соответственно такие j из i можно получать заменяя последние n бит числа i на единицы.

```
inc(i, v):
    j = i
    while (j < n):
        f[j] += v
        j = j | (j+1)
```

1.5 Разреженная таблица

Хотим. Структуру. Данных.

Задан массив и он не меняется. Хотим посчитать минимум на любом отрезке за 1.

Можем предпосчитать за квадрат на всех отрезках и просто выдавать, но это долго.

$m[i,j] = \min\left(a\left[i \dots i + 2^j - 1\right]\right) \qquad i = 0 \dots n-1 \quad j = 0 \dots \log n$

```
for i = 0 .. n m[i,0] = a[i]
for j = 1 .. log n
    for i = 0 .. n-2^j
       m[i,j] = min(m[i,j-1], m[i+2^(j-1), j-1])
```

Теперь собственно как искать минимум:

$d = r - l \quad \max k : 2^k \leqslant d$

$res = \min\left(m[l,k], m[r-2^k,k]\right)$

Нужно искать к-шки за O(1). Вообще с помощью *битовых извращений* можно найти за $\log\log$, а с помощью *страшных битовых извращений* вообще за $O(1)$, но мы пока не будем выпендриваться и просто предпосчитаем к-шки для всех d

Ура! всё работает. Но вся эта схема сильно использует тот факт, что мы считаем минимум (использует свойство идемпотентности $min(x,x) = x$). Таких функций не прям чтоб много, а хотелось бы и с другими аналогичные вещи делать.

Давайте поделим отрезок пополам и посчитаем префиксные суммы справа и слева от середины. Тогда большой отрезок в запросе можно разбить на два: до и после середины, сложить (применить функцию) и получить ответ.

Проблема: если отрезок в одной половине, то его в общем случае не посчитать. Решение: поделим половины отрезков так же, как делили целый. В итоге мы найдём $2n$ сумм. n на всё отрезке и по $\frac{n}{2}$ на половинах. С каждым таким делением добавляется n. Всего делить можем $\log n$ раз, значит сумм всего будет $n \log n$

Отвечаем на запрос: находим границу, которую отрезок от l до r пересекает (первый бит числа $l\&r$)

1.6 Чё можно делать в двумерном случае

Задача 2. Есть прямоугольнички, они пересекаются. Ещё есть набор точек. Для каждой точки надо найти сколько прямоугольников покрывает эту точку

Решение. Метод заметающей прямой. Поставим прямую и будем двигать её направо. Когда встретим точку, ответим на запрос про неё.

Для одной линии считаем сколько покрывает каждую точку.

Выделим все у кординаты, в которых есть вершина прямоугольника.

Встретили границу, делаем +=1 или -=1

∎

Задача 3. Есть прямоугольники. Надо найти площадь их объединения

Структура:

1. $a[i] \pm 1$ на отрезка
2. $\sum len(i) : a[i] > 0$

Дерево отрезков:

1. sum(i,v)
2. sum(l,r)

Прямой аналог:

1. set(i,j,v)   a[i,j] = v
2. sum(l,r,t,b)

Сделаем дерево отрзков для строчек. Каждая вершина отвечает за полосу из строчек. В каждой вершине хранится дерево отрезков по столбцам.

1.7 Дерево Фенвика 2D

$$f(i) = \sum_{j=p(i)}^{i} a[j]$$

$$f(i,j) = \sum_{x=p(i)}^{i} \sum_{y=p(j)}^{j} a[x,y]$$

План: сделать всё точно также

```
x = i-1
while x >= 0:
    rest += f(x)
    x = x&(x+1)-1

x = i
while x >= 0:
    y = j
    while y >= 0:
        rest += f(x,y)
        y = (y+1)&y-1
    x = (x+1)&x-1
```

1.8 Разреженные таблицы 2D

$m[i,j] = min\left(a[i, i + a^j - 1]\right)$

$m[i_1, j_1, i_2, j_2] = min\left(a[i_1, i_1 + 2^{j_1} - 1, i_2, i_2 + 2^{j_2} - 1\right)$

precalc $O(n^2 \lg^2)$

Запрос: $O(1)$

Задача 4. Есть двумерные точки. Хотим вывести все точки в прямоугольнике.

Спроектируем точки на одну координату и построим на них дерево отрезков. В вершинах сортированные по другой координате точке.

1.9 Бинпоиск.каскадирование

Делаем один и тот же бинпоиск на многих массивах. – $O\left(n \log m\right)$

Если знать в нашей последней задаче где бинпоиск в корне, можно легко понять как будет в рекурсии, элементы только пропадают и позиции однозначно соответствуют

Частичное каскадирование, Fractional Cascading

Чтобы не появлялось много лишних элементов, запихаем массива наверх. Чтобы не было слишком много, запихаем половину (каждый второй).

При переходе смотрим на ближайшие запихнутые элементы. X между ними, проверем две позиции.

А теперь у нас много элементов.