Лекция 9. Дерево Фенвика.

#вшпи #аисд #теория #структуры_данных #двумерные_структуры

Автор конспекта: Гридчин Михаил

Хотим уметь решать задачу online dyncamic RSQ с изменением в точке.
Мы уже умеем решать эту задачу деревом отрезков. (Лекция 8. Разреженная таблица. Дерево отрезков.)

Дерево Фенвика

Преимущества дерева Фенвика

- оно быстрое (в несколько раз быстрее дерева отрезков)
- оно лаконичное
- оно легко обобщается на большие измерения

Интуиция 1

Каждый отрезок $[l, r]$ представлен объединением отрезков при реализации дерева отрезков

Интуиция 2

Давайте научимся разбивать отрезок $[1, r]$ на отрезков и считать ответ на отрезке $[l, r]$, как ответ на префиксе $[1, r]$ минус ответ на запрос на префиксе $[1, l - 1]$.
Тогда для операции точно нужны свойства

- коммутативность
- ассоциативность
- обратимость
- нейтральный элемент

Построение разбиения

Рассмотрим отрезок $[1, R]$. Как его можно разбить на $O(\log n)$ отрезков? Ответ: по разложению R на степени двойки. Для примера

$$14 = 1110_2 = 8 + 4 + 2$$

$$[1, 14] = [1, 8] \cup [9, 12] \cup [13, 14]$$

Теперь вопрос, сколько всего подотрезков нам нужно хранить?

Утверждение. Подотрезков ровно n.

□ Покажем, что n подотрезков хватит: пронумеруем отрезки для i-го подотрезка

$$i = 2^{k_1} + 2^{k_2} + \ldots + 2^{k_t}, k_1 > k_2 > \ldots > k_t$$

Будем хранить сумму на подотрезке $[i - 2^{k_t} + 1, i]$

Пусть в массиве c на i-й позиции хранится сумма $sum[i - 2^{k_t} + 1, i]$. Тогда

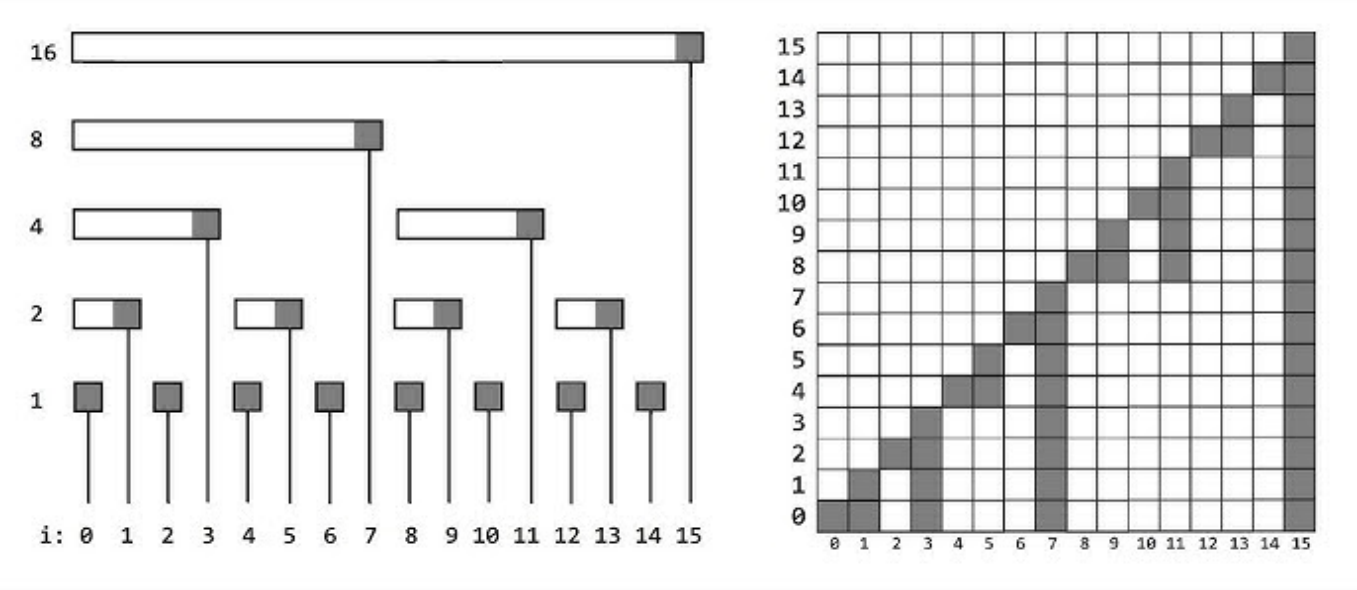
$$PrfixSum(i) = c[i] + c[i - 2^{k_t}] + c[i - 2^{k_t} - 2^{k_{t-1}}] + \ldots + c[2^{k_1}] =$$
$$= sum[i - 2^{k_t} + 1, i] + sum[i - 2^{k_t} - 2^{k_{t-1}} + 1, i - 2^{k_t}] + \ldots + sum[1, 2^{k_1}] = sum[1, i]$$

■

Например, если $c = 14$. Тогда

$$PrefixSum(14) = c[14] + c[14 - 2] + c[14 - 2 - 4] = c[14] + c[12] + c[8]$$

Теперь понятно, как сделать запрос суммы.



дерево Фенвика таблица.png

Обновление в точке

Посмотрим на слагаемое $c[i] = sum[i - 2^{k_t} + 1, i]$

$$j \in [i - 2^{k_t} + 1, i] \iff i - 2^{k_t} + 1 \le j \le i$$

Как это выглядит в двоичной записи?

$$i = \underbrace{1010110110}_{\text{какие-то цифры}} \underbrace{0000000000000}_{\text{блок нулей длины } k}$$

$$j = \underbrace{1010110100}_{\text{почти те же цифры}} \underbrace{0000010011101}_{\text{блок цифр длины } k}$$

Утверждение. Чтобы получить все i, для которых выполняется неравенство, нужно итеративно находить наименьший 0 в j после которого есть единицы, заменить его на 1, а всё после него заменить на 0.
Доказательство. В качестве упражнения.

Пример. Для $j = 42 = 101010_2$ будет $i_1 = 44 = 101100_2$ и $i_2 = 48 = 110000_2$, $i_3 = 64 = 1000000_2$ и так далее.

Резюме алгоритма

$GetSum[1, r]$ свели к 2 $GetPrefixSum$, $Update(j, delta)$ - в обоих алгоритмах $O(\log n)$ операций $\implies$ время работы структуры на все запросы $O(\log n)$, если умеем находить индексы за $O(1)$.

Как находить индексы?

Для 1-индексации нужно уметь находить первую справа единицу в битовом представлении и делать её равную 0. За счёт того, что мы делаем $i- = 2^{k_t}$, где k_t - это минимальный единичный бит. То есть итеративно зануляем младший бит. Для $Update$ находим самый первый ноль, после которого есть единицы, ноль меняем на единицу, а все остальные цифры после него меняем на нули. То есть, другими словами, мы нашли младший бит k_t в числе и сделали $i+ = 2^{k_t}$.

Теорема (без доказательства). В 0-индексации младший бит можно находить по формуле $2^{k_t} = i\& - i$, где k_t - младший бит i, $\&$ - побитовое И. В 0-индексации уменьшение индекса можно реализовать, как $Backward(i) = i\&(i + 1) - 1$, увеличение индекса можно реализовать, как $Forward(i) = i \mid (i + 1)$ (последний 0 заменили на 1).

Итого

```
def PrefixSum(r):
    s = 0
    while r >= 0:
        s += c[r]
        r = Backward(r)
    return s

def Update(id, delta):
    while id < n:
        c[id] += delta
        id = Forward(id)
```

Многомерный случай

Теперь, если у нас 2D случай, и мы хотим находить сумму на префиксном прямоугольнике.

Утверждение. Нам ничего не мешает посчитать суммы на непересекающихся $O(\log n \cdot \log m)$ подпрямоугольниках. В реализации добавится всего лишь один внутренний цикл.