1 Дерево фенвика (binary indexed tree)

1.1 Одномерный случай

Будем решать задачу вычисления на отрезке обратимой функции (например сумма), с поддержкой обновления в точке. Обратимость функции позволяет разбить задачу на две задачи на префиксах.

Пусть a — исходный массив. Будем в ячейке $bit[x]$ хранить $\sum_{i=f(x)}^{x} a[i]$, для некоторой функции f. Что нужно от функции f?

- Необходимо маленькое число итераций, чтобы дойти до начала массива
- Каждое число лежит в маленьком числе отрезков $[f(y), y]$, и эти y можно быстро найти по x.

Выберем в качества функции $f(x) = x\&(x+1)$. В терминах битовой записи это обнуление всех 1, на конце числа. В таком случае $sum(0, r) = sum(0, f(r) - 1) + bit[r]$. Вычисление этого значение потребует $O(\log n)$ времени.

В каком случае $x \in [f(y), y]$. Рассмотрим первое различие в битовой записи x и y. В этом бите, x должен быть 0, а y должен быть 1. С другой стороны $f(y)$ в этом бите должен быть 0. А значит, все следующие биты y равны 1, по определению функции f.

То есть множество интересующих нас y — это такие, в которых несколько последних нулей в битовой записи x заменены на 1. Они получаются итерировнием функции $g(x) = (x|(x+1))$. Проход по всем таким значениям потребует времени $O(\log n)$.

Основными преимуществами дерева фунвика являются простота в написании, скорость работы и легкость обощения на многомерный случай.

1.2 Обобщение на многомерный случай

Пусть та же самая задача решается на прямогуольнике, а не на отрезке. Тогда аналогичная струтура позволяет решать задачу за $O(\log^2 n)$ на запрос и обновление.

В $bit[x][y]$ будем хранить $\sum_{i=f(x)}^{x} \sum_{j=f(y)}^{y} a[i][j]$. Тогда вычисление суммы сводится к $\log n$ вычислений одномерных сумм. А обновление — к $\log n$ одномерных обновлений. Аналогичную конструкцию можно строить в большем числе измерений.

2 Декартово дерево

2.1 Бинарное дерево поиска

Бинарным деревом поиска, называется бинарное дерево, в каждой вершине которого написан ключ x_i, причем выполнено условие, что все ключи в левом поддерве меньше ключа в вершина, в все ключи в правом поддереве больше.

Бинарное дерево поиска позволяет выполнять операции с множеством ключей, такие как добавлние, удаление, поиск элемента, за время порядка высоты дерева. Для того, чтобы добиться логарифмической высоты, используются различные методы балансировки. Декартово дерево является одним из них.

2.2 Декартово дерево

Будем вместе с каждой вершиной хранить приоритет y_i. Добавим дополнительное требование: приоритет предка всегда должен быть меньше, чем приоритет потомка.

Теорема 1. *Если все ключи различны, и все приоритеты различны, то декартово дерево единственно.*

Доказательство. Корень единственный, разбиение на левое и правое поддерево единственно, индукция по размеру. □

Выберем в качестве y_i случайные числа.

Лемма 1. *Пусть $x_1 < x_2 < \cdots < x_{n-1} < x_n$. Пусть y_i случайные. В таком случае $\mathbb{P}\{x_i$ - предок $x_j\} = \frac{1}{|i-j|+1}$*

Доказательство. Для этого необходимо, чтобы у x_i был самый маленький приоритет, среди всех ключей от x_i до x_j, включительно. □

Теорема 2. *Математическое ожидание высоты вершины декартового дерева $O(\log n)$*

Доказательство. Высота = количество предков. Осталась линейность мат. ожидания и сумма гармонического ряда. □

2.3 Merge

Пусть L, R — два декартовых дерева, причем все ключи в L меньше всех ключей в R. В таком случае, их можно объединить за время $O(h)$.

Корнем будет либо корень L, либо корень R. После чего, одно поддерво надо оставить как есть, второе рекурсивно слить.

2.4 Split

Пусть T — декартово дерево. В таком случае его за время $O(h)$ можно разделить на два, так что в первом все ключи не больше некоторого x, а во втором все больше.

Для этого, надо посмотреть в какую из половин пойдет корень, одно поддерево разделить на сына корня и вторую половину ответа, одно оставить как есть.

2.5 Выражение операций через Merge и Split

Add(x) = Split(x, l, r) + Merge(Merge(l, x), r); Delete(x) = Split(x, l, r) + Split(x-1, l, m) + Merge(l, r);

2.6 Быстрая реализация добавления и удаления

Добавление: спускаемся до места, куда надо вставить. Заменяем вершину на новую, а сыновей новой на Split. Удаление: спускаемся до вершины. Заменяем вершину на Merge ее сыновей.

2.7 Неявный ключ

Заметим, что если хранить размер поддерева, то наша струтура поддерживает запрос "отрезать k наименьших эелементов".

Если заменить Split на такую операцию, то ключи не нужны для Split и Merge. Если убрать ключи, то получается струтура данных, которая хранит массив, и поддерживает следующие операции

- Доступ к элементу по номеру за $O(\log n)$
- Конкатенация двух массивов за $O(\log n)$
- Разрезание одного массива на два за $O(\log n)$

Фактически, ключем в данном случае является количество вершин, расположенных раньше данной в порядке обхода. При этом ключ не хранится, а пересчитывается по мере необходимости. Поэтому такая струтура называется декартово дерево по неявному ключу.

При соединении с идеей отложенных операций, такая струтура позволяет делать все операции, которые может делать дерево отрезков и многое другое.

2.8 Пример: переворот отрезка массива

В качестве отложенной операции будем хранить надо ли перевернуть отрезок, соответствующей поддереву этой вершины. При проталкивании, необходимо поменять местами сыновей, и в обоих сыновьях поменять на противоположную метку о перевороте.

3 Задачи

- Метод сканирующей прямой:
  - Количество прямоугольников покрывающих точку
  - Объединение прямоугольников
- Сложные штуки в дереве отрезков
  - Подотрезок с максимальной суммой на отрезке
  - Сумма сумм на подотрзке