Олимпиадное программирование

Занятие 13. Алгоритм Мо. Дерево отрезков

Труфанов Павел Николаевич

Онлайн-школа  Фоксфорд

Foxford.ru 2019-2020

Алгоритм Мо

Задача: требуется отвечать на вопрос - сумма таких чисел на отрезке, которые встречаются хотя бы 3 раза.

Пусть мы умеем писать структуру данных, которая умеет отвечать на такой вопрос, если ровно все числа отрезка лежат в ней. Плюс она должна уметь добавлять элементы слева и справа, и удалять. Теперь отсортируем запросы таким образом $<< l/K, r >>$, где l, r - границы запроса, а K выбранная константа, примерно равная $\sqrt{N}$. Тогда покажем, что время работы будет маленькое.

Алгоритм 3D Мо

Продолжение идем предыдущего алгоритма. Теперь у нас появляются точечные изменения. Мы просто перенесем наше действие в трехмерное пространство. Отсортируем запросы о количестве чисел, которые встречаются хотя бы 3 раза, так $<< t/K, l/K, r >>$, где l, r по прежнему границы запроса, а t - количество изменений, которое произошло до нашего запроса. Выберем $K = n^{2/3}$ и мы молодцы)

Дерево отрезков

Мы хотим сделать ультимативную структуру данных, которая будет лишена таких недостатков префиксных сумм и разреженных таблиц, как ограничение на вид функции, и которая будет работать быстро.
Единственное ограничение в нашей структуре будет следующим: функция должна удовлетворять следующему свойству – если у нас есть функция на двух непересекающихся отрезках, которые лежат рядом, то мы должны уметь быстро считать функцию на их объединении.

Структура

Давайте нарисуем двоичное дерево. Пусть у каждой вершины будет свой полуинтервал массива, за который она отвечает. Тогда, скажем что корень отвечает за полуинтервал $[0; n)$, где n – длина массива. Если вершина отвечает за полуинтервал длины хотя бы 2, то у нее будут два ребенка. Пусть вершина отвечает за полуинтервал $[l; r)$. Тогда выберем $m = \frac{l+r}{2}$, и скажем, что наш левый сын отвечает за полуинтервал $[l; m)$, а правый отвечает за полуинтервал $[m; r)$.
Таким образом, глубина дерева получится $\log n$

Размер дерева

Допустим, что n – степень двойки. Тогда заметим, что количество вершин в дереве в точности $2n - 1$. Теперь пусть n – любое число. Тогда мы можем увеличить его в не более чем 2 раза и получить степень двойки. А далее увеличиваем еще в 2 раза и получаем размер дерева отрезков. Таким образом, можно сказать, что дерево отрезков занимает не более $4n$ памяти, то есть $O(n)$.

Нумерация

Давайте не будем хранить дерево как граф. Как удобно его хранить, чтобы все работало быстро. Давайте скажем, что у корня номер 0. Тогда, давайте нумеровать вершины так: у вершины с номером v левый сын имеет номер $2v + 1$, а правый $2v + 2$. Некоторые номера придется пропустить, если длина массива не степень двойки, но ничего страшного.

Построение

Заметим, что мы можем построить вершину, если построены ее дети. Давайте запускаться рекурсивно от детей, пока не придем в листики. Когда рекурсия раскручивается обратно - будем строить сами вершины

Код

```
void build(int v, int l, int r) {
    if (l == r - 1) {
        t[v] = a[l];
        return;
    }
    int m = (l + r) / 2;
    build(2 * v + 1, l, m);
    build(2 * v + 2, m, r);
    t[v]=f(t[2*v+1],t[2*v+2]);
}
```

Запрос

Пусть у нас есть некий отрезок запроса. У каждой вершины есть отрезок, за который она отвечает. Давайте мы каждую вершину покрасим в один из трех цветов.

- Красный, если эти два отрезка не пересекаются, то есть текущая вершина нам совсем неинтересна.
- Зеленый, если отрезок вершины вложен в отрезок запроса, то есть текущая вершина интересна нам полностью
- Желтый, если часть отрезка вершины входит в запрос, и часть не входит.

Запрос

Все зеленые вершины нам нужно взять, и дальше не идти. На красных вершинах нужно остановиться. В желтых вершинах надо пойти в детей и объединить ответы от них. Утверждается, что время работы такого запроса $O(\log n)$.

Код

```
int ask(int v, int l, int r,
        int askl, int askr) {
    if (l >= askr || r <= askl) {
        return NEUTRAL;
    }
    if (l >= askl && r <= askr) {
        return t[v];
    }
    int m = (l + r) / 2;
    return f(ask(2*v+1,l,m,askl,askr),
             ask(2*v+2,m,r,askl,askr));
}
```

Изменение

Пусть нужно сделать точечное изменение. Значит, точно должен поменяться какой-то лист. Ну тогда нужно поменять всех его предков, которых всего $O(\log n)$. Код простой.

Код

```
void change(int v, int l, int r,
            int pos, int val) {
    if (l == r - 1) {
        t[v] = val;
        return; }
    int m = (l + r) / 2;
    if (pos < m) {
        change(2*v+1,l,m,pos,val);
    } else {
        change(2*v+2,m,r,pos,val);
    }
    t[v]=f(t[2*v+1],t[2*v+2]);
}
```

Позиция максимума

Задача – получать позицию максимума на отрезке. Храним либо пару элементов (элемент, индекс), либо просто (индекс) и сравниваем элементы по двум индексам.

Поиск k-ого нуля в массиве с изменениями

Надо уметь получать позицию k-ого нуля в массиве.
Требуется решить за $O(\log^2 n)$ на запрос, а потом за $O(\log n)$ на запрос.

Спуск. Код

```
int getkth(int v,int l,int r,int k) {
    if (l == r - 1) {
        return l;
    }
    int m = (l + r) / 2;
    if (t[2 * v + 1] > k) {
        return getkth(2*v+1,l,m,k);
    } else {
        return getkth(2*v+2,m,r,
                      k-t[2*v+1]);
    }
}
```

Несколько задач

- Количество максимумов на отрезке
- Максимальная сумма двух чисел на отрезке
- Максимальный возрастающий подотрезок на отрезке.
- Максимальная сумма подотрезка на отрезке

До встречи!

FOXFORD.RU

Онлайн-школа Фоксфорд

Фоксфорд