ЛКШ, ЛКШ.2018.Август B'
B', конспект лекции

Собрано 12 августа 2018 г. в 10:09

Содержание

Тема #1: Дерево отрезков

10 августа

WARNING! В этой версиии конспекта может быть количество опечаток, стремящееся к бесконечности. Не стоит относиться к этому конспекту как к последней инстанции.

1.1. Описание структуры

Дерево отрезков — мощная структура данных, позволяющая решать многие задачи, связанные с запросами на отрезке.
Дерево отрезков строится на массиве и хранится как куча.
root = 1, parent[v] = v / 2, leftSon[v] = 2 * v, rightSon[v] = 2 * v + 1
Если вершине соответствует полуинтервал $[vl, vr)$, $vm = (vl + vr)/2$, то сыновьям соответствую полуинтервалы $[vl, vm)$, $[vm, vr)$
TODO: картинка.
Корню соответствует весь массив.
Листья ДО — вершины, у которых $vl + 1 == vr$.
В вершинах ДО обычно (но не всегда) хранится какая-то функция от отрезка массива (сумма, минимум...)
TODO: картинка ДО на сумму или максимум
Если $n = 2^k$, то все листы ДО лежат на одном уровне.
Для хранения ДО при $n = 2^k$ достаточно массива из $2n$ элементов, при произвольном n — массива из $4n$ элементов.
При $n = 2^k$ листья (элементы массива) хранятся в элементах массива, начиная с n-ого.

1.2. Построение ДО

Будем рассматривать построение ДО для $n = 2^k$. Есть два способа построения: сверху и снизу.
Построение ДО сверху:
рекурсивно построим ДО для детей, посчитаем функцию в текущей вершине.
Рассмотрим построение ДО на сумму:

```
1  int a[n]; // изначальный массив
2  int tree[2 * n];
3  void build(int v = 1, int vl = 0, int vr = n) { // пишем на полуинтервалах
4      if (vr - vl == 1) {
5          tree[v] = a[vl];
6      }
7      int vm = (vl + vr) / 2;
8      build(2 * v, vl, vm);
9      build(2 * v + 1, vm, vr);
10     tree[v] = tree[2 * v] + tree[2 * v + 1]; // записали сумму на полуинтервале [L, R]
11 }
12
13 build();
```

Построение ДО снизу. Здесь важно, что $n = 2^k$ (мы пользуемся тем, что массив хранится в элементах ДО, начиная с позиции n) Бежим от $n-1$ до 1. Значения в потомках уже посчитаны $\Rightarrow$ можем посчитать значение для текущей вершины.

```
1  int a[n]; // изначальный массив
```

```
int tree[2 * n];
void build() {
    for (int i = n; i < 2 * n; ++i) {
        tree[i] = a[i - n];
    }
    for (int i = n - 1; i >= 1; --i) {
        tree[i] = tree[2 * i] + tree[2 * i + 1];
    }
}

build();
```

Первый способ работает для произвольного n, второй быстрее.

1.3. Изменение в точке

Мы сумели построить дерево отрезков. Давайте научимся его модифицировать, если к нам приходят запросы «измени значение массива на позиции i» (это называется изменение в точке). Какие вершины ДО нужно изменить? Только те, которые зависят от i-го элемента массива. Это все вершины, которые являются предками листа, отвечающего за элемент массива с индексом i.

Реализация сверху: рекурсивно спускаемся по дереву в нужные отрезки, пересчитываем значения в вершинах.

Код:

```
void update_point(int i, int new_val, int v = 1, int vl = 0, int vr = n) { // присвоить элементу с индексом i значение new_val
    if (vr - vl == 1) { // дошли до листа
        tree[v] = new_val;
    }
    int vm = (vl + vr) / 2;
    if (i < vm) { // i лежит в полуинтервале [vl, vm) => в левом сыне
        update_point(i, new_val, 2 * v, vl, vm);
    }
    else {
         update_point(i, new_val, 2 * v + 1, vm, vr);
    }
    tree[v] = tree[2 * v] + tree[2 * v + 1];
}
```

Реализация снизу: какие вершину нужно обновить? $i + n$ и всех ее предков.

```
void update_point(int i, int new_val) {
    i += n;
    tree[i] = new_val;
    while (i > 1) {
        i /= 2;
        tree[i] = tree[2 * i] + tree[2 * i + 1];
    }
}
```

Первое снова работает для произвольных n, а второе быстрее.

И то и то работает за $\mathcal{O}(\log n)$, поскольку такова высота ДО.

1.4. Сумма на отрезке

Нам приходит запрос «скажи сумму элементов на отрезке $[L, R]$».
Давайте будем спускаться по ДО сверху. Пусть мы стоим в очередной вершине. Она отвечает за полуинтервал $[vl, vr)$ в нашем массиве. Если этот полуинтервал лежит полностью внутри запроса, то просто прибавим к ответу сумму на этом полуинтервале (она записана в нашей вершине). Если полуинтервал и отрезок пересекаются — рекурсивно запустимся от детей.

```
1 int sum_on_seg(int L, int R, int v = 1, int vl = 0, int vr = n) { // запрос [L, R)
      тоже полуинтервал
2     if (L <= vl && vr <= R) return tree[v]; // полуинтервал, за который отвечает v лежит
          полностью внутри запроса
3     if (vr <= L || R <= vl) return 0; // полуинтервал и запрос не пересекаются
4     int vm = (vl + vr) / 2;
5     return sum_on_seg(L, R, 2 * v, vl, vm) + sum_on_seg(L, R, 2 * v + 1, vm, vr);
6 }
```

Утверждение. Этот алгоритм работает за $\mathcal{O}(\log n)$
Доказательство. Докажем, что на каждом уровне мы рассмотрим не более 4 вершин. Так как глубина дерева $\mathcal{O}(\log n)$, утверждение будет доказано.
Рассмотрим вершины на очередном уровне ДО. Пусть их не более четырех. На следующем уровне будут рассмотрены потомки только тех вершин, которые попали на границы отрезка. Таких вершин только две $\Rightarrow$ на следующем уровне не более 4.
Более того, верно, что на каждом уровне будет выбрано не более 2 вершин $\Rightarrow$ ДО умеет разделять отрезок запроса на $\leqslant 2 \log n$ непересекающихся отрезков.
О ДО можно думать как о структуре данных, которая умеет разбивать запрос на $\mathcal{O}(\log n)$ непересекающихся отрезков, на которых уже посчитан ответ на запрос.

Тема #2: Задачки на ДО

10 августа

2.1. Делаем свой set

Хотим для чисел $\leqslant 10^5$ реализовать структуру данных, которая умеет отвечать на следующие запросы:

1. Добавить элемент в множество
2. Удалить элемент из множества по значению
3. Сказать наименьший элемент $\geqslant x$ (lower_bound(x))

Сделаем ДО на 10^5 элементов на сумму. В листе будем хранить единицу, если элемент есть во множестве и ноль иначе. Таким образом, сумма в вершине равна количеству элементов в множестве, которые лежат на соответствующем вершине полуинтервале.
Операции удаления и прибавления — модификации в точке.
Как сделать операцию lower_bound?
Будем идти вниз по дереву. Ответ — лист, в который пришли.
По сути, это запрос «найди первую единицу на отрезке массива $[x, +\infty)$
Пусть мы стоим в вершине v. Если она отвечает за полуинтервал, полностью лежащий внутри отрезка запроса, то нужно идти в левое поддерево, если там не ноль, а иначе в правое.
Если полуинтервал v пересекается левым сыном с x, то нужно попробовать пойти в левое поддерево, а если ответа там не оказалось, то в правое.

```
1  int lower_bound(int x, int v = 1, int vl = 0, int vr = n) {
2      if (vr - vl == 1) {
3          return tree[v] ? vl : -1;
4      }
5      int vm = (vl + vr) / 2;
6      if (x <= vl || x >= vm) // либо наш отрезок внутри запроса, либо все элементы в левом
           отрезке по индексу меньше x
7          return tree[2 * v] ? lower_bound(x, 2 * v, vl, vm) : lower_bound(x, 2 * v
               + 1, vm, vr);
8      int res = lower_bound(x, 2 * v, vl, vm); // Левый сын пересекает границу запроса.
           пытаемся пойти в левого сына
9      return res != -1 ? res : lower_bound(x, 2 * v + 1, vm, vr); // Если не
           получилось, идем в правого
10 }
```

2.2. Дейкстра с ДО

В алгоритме Дейкстры нам нужен был черный ящик, который умеет изменять значение в массиве и говорить минимум на массиве. С этим замечательно справится дерево отрезков на минимум.
Изменение в массиве == изменение в точке, минимум на массиве == минимум на отрезке.
Чтобы не учитывать в минимуме элементы, которые мы уже обработали, можно присвоить им в точке значение бесконечность.

2.3. Задача про кубик

Есть кубик и массив, каждая ячейка — один из 6 поворотов кубика, запросы = какая будет грань сверху, если сделать все повороты из отрезка $[L, R]$?
Давайте зафиксируем начальное положение кубика. Для каждой вершины ДО насчитаем, в какое положение перейдет кубик из начального положения, если выполнить все эти операции. ДО разбило запрос на $\mathcal{O}(\log n)$ непересекающихся отрезков, для каждого из которых мы знаем результат из начального положения. Чтобы ответить на запрос надо по очереди применить к кубику эти преобразования.

2.4. Число инверсий

Дана перестановка, хотим посчитать количество инверсий в ней. Сделаем ДО на сумму на n элементов. В листе будет единица, если элемент уже есть в массиве и 0 иначе. Сначала поставим все нули. будем добавлять элементы массива по очереди. Очередной элемент образует инверсию с теми, что пришли раньше его и больше его. Это сумма на суффиксе в ДО.

2.5. Перестановка по инверсиям

Дан массив $inv[i]$ — сколько инверсий образует i-ый элемент. Хотим по массиву восстановить перестановку.
i-ый элемент образует инверсию с j-ым, если $i < j$ и $a_i > a_j$.
Будем ставить элементы по очереди. С какими элементами образует инверсию элемент $a[0]$? С теми, что меньше его и еще не в нашем массиве. Таким образом, $a[0] = inv[0] + 1$.
Аналогично i-ый элемент должен принять значение $inv[i] + 1$-го элемента из множества еще не поставленных в массив элементов.
Свели задачу к «скажи k-ю порядковую статистику на множестве, удали этот элемент из множества».
Заведем ДО на сумму по количеству элементов в перестановке. Изначально в листьях единицы. Когда приходит запрос, мы спускаемся вниз до k-ой единицы.
Как это сделать? Пусть мы стоим в вершине v и выбираем, в какого из сыновей пойти. Мы знаем, сколько единиц в левой половине и сколько в правой. Если в левой больше, чем k, то пойдем туда. Иначе пойдем в правую, не забыв вычесть из k количество единиц в левой (все как в поиске k-ой статистики за линию).

2.6. ПСП ли а отрезке?

Дана скобочная последовательноть. Приходят запросы вида «измени скобку на позиции i» и «правда ли, что на отрезке $[L, R]$ записана ПСП?».
Чтобы понять, ПСП ли там написана, нам нужно уметь считать минимальный баланс на этом отрезке.
TODO