**Дерево отрезков**

**Дерево отрезков**

Дерево отрезков строится на массиве. Элемент дерева соответствует какому-то подотрезку массива [a;b]. При этом корень дерева соответствует всему отрезку [1;N].

Каждый отрезок разбивается на две равные (или отличающиеся по длине на 1) части. У вершины, соответствующей этому отрезку два потомка - левый и правый, соответствующий левому и правому отрезку разбиения. Отрезки длины 1 соответствуют отдельному элементу и являются листьями (у них нет потомков).

В каждой вершине хранится функция, для которой строится дерево отрезков (минимум, максимум, сумма).

Будем реализовывать дерево отрезков при помощи ссылочной реализации. Для представления одной вершины дерева будем использовать следующую структуру (пример для нахождения максимального значения на отрезке):

**struct node**

**{**

**int left, right, max;**

**node \* child\_left, \* child\_right;**

**};**

Поля структуры:

**left**

 - значение индекса левого конца отрезка массива, которому соответствует данная вершина.

**right**

 - значение индекса правого конца отрезка массива, которому соответствует данная вершина.

**max**

 - значение наибольшего элемента массива на отрезке [left;right].

**child\_left**

 - ссылка на левого потомка.

**child\_right**

 - ссылка на правого потомка.

**Построение дерева отрезков**

Дерево отрезков строится при помощи следующей рекурсивной функции:

**node\* build(int left, int right, vector<int> & values)**

**{**

**if (left > right)**

**return 0;**

**node \* res = new node;**

**res->left = left;**

**res->right = right;**

**if (right == left)**

**{**

**res->child\_left = res->child\_right = 0;**

**res->max = values[left];**

**}**

**else**

**{**

**int mid = (left + right) / 2;**

**res->child\_left = build(left, mid, values);**

**res->child\_right = build(mid + 1, right, values);**

**if (res->child\_left->max >= res->child\_right->max)**

**{**

**res->max = res->child\_left->max;**

**}**

**else**

**{**

**res->max = res->child\_right->max;**

**}**

**}**

**return res;**

**}**

Эта функция строит дерево отрезков для подотрезка массива

**values**

 с концами

**left**

 и

**right**

. Корень построенного дерева будет соответствовать всему подотрезку [left;right]. Функция возвращает указатель на корень построенного дерева.

Функция устроена так:

1. Если

**left > right**

, то это вырожденный случай (отрезок пустой) и функция возвращает нулевой указатель.

2. Если

**left == right**

, то отрезок состоит из одного элемента, это - лист дерева, у него нет потомков. Поэтому указатели на левого и правого потомка инициализируются нулями, а в поле

**max**

 записывается значение элемента массива.

3. Иначе отрезок разбивается на две части и функция вызывается рекурсивно для построения дерева отрезков для левой и правой части. Поля

**child\_left**

 и

**child\_right**

 инициализируются значениями, которая вернули рекурсивные вызовы, а поле

**max**

 инициализируется максимумом из значений

**max**

 для левого и правого подотрезков.

**Запрос значения на подотрезке**

Рассмотрим функцию нахождения максимума на отрезке [left;right]. Точнее, данная функция находит максимум на пересечении отрезка [left;right], который передается в качестве параметров функции, и того отрезка, которому соответствует корень дерева. Это необходимо для того, что в процессе рекурсивных вызовов отрезок, за который отвечает корень дерева, может уменьшатся, и может стать, например, подотрезком того отрезка, который задан в запросе.

**int query(node \* root, int left, int right)**

**{**

**if (right < root->left || left > root->right)**

**return -INF;**

**if (left <= root->left && root->right <= right)**

**return root->max;**

**int ans1 = query(root->child\_left, left, right);**

**int ans2 = query(root->child\_right, left, right);**

**return max(ans1, ans2);**

**}**

Прежде всего функция проверяет, пересекается ли отрезок запроса с отрезком корня дерева. Если не пересекаются, то функция возвращает специальное значение (-INF для функции-максимума, INF для функции-минимума, 0 для суммы).

Затем проверяется условие вложенности отрезка корня дерева в отрезок запроса. Если условие верно, то запрос полностью покрывает ту вершину дерева, о которой идет речь, поэтому в качестве ответа на запрос можно вернуть поле

**max**

 данной вершины.

Иначе сделаем два рекурсивных вызова для левого и правого поддерева и выберем наибольшее из тех значений, которые они вернули.

**Обновление элемента дерева**

Обновление элемента дерева также производится рекурсивной функцией.

Параметры функции - индекс обновляемого элемента

**idx**

 и новое значение

**val**

.

**void update(node \* root, int idx, int val)**

**{**

**if (idx < root->left || idx > root->right)**

**return;**

**if (root->left == root->right)**

**{**

**root->min = root->max = val;**

**return;**

**}**

**update(root->child\_left, idx, val);**

**update(root->child\_right, idx, val);**

**root->max = max(root->child\_left->max, root->child\_right->max);**

**}**

Функция устроена аналогично - если

**idx**

 не попадает внутрь отрезка дерева, функция ничего не делает. Если отрезок является листом, то функция обновляет поле

**max**

, присваивая ему значение

**val**

. Иначе функция вызывается для левого и правого поддерева (один из этих вызовов на самом деле сразу же завершится), после чего обновляет значение

**max**

 наибольшим из значением

**max**

 для поддеревьев.

**Групповое обновление с групповым запросом**

Рассмотрим дерево отрезков с групповой операций обновления: ко всем элементам подотрезка [left;right] прибавить одинаковое значение

**delta**

.

Для эффективной реализации группового обновления необходимо в каждой вершине дерева отрезков хранить дополнительное поле

**add**

 - это величина, которая добавляется ко всем элементам на данном отрезке. Если запрос на обновление целиком покрывает какую-то вершину дерева, то просто изменяется величина

**add**

 в данной вершине, никакие другие значения не меняются.

Максимальным значением в данном поддереве тогда будет сумма полей

**max**

 и

**add**

.

**Функция обновления значения на отрезке**

**void update(node \* root, int left, int right, int delta)**

**{**

**if (right < root->left || left > root->right)**

**return;**

**if (left <= root->left && right >= root->right)**

**root->add += delta;**

**else**

**{**

**update(root->child\_left, left, right, delta);**

**update(root->child\_right, left, right, delta);**

**root->max = max(root->child\_left->max + root->child\_left->add,**

**root->child\_right->max + root->child\_right->add);**

**}**

**}**

**Функция запроса максимума на отрезке**

**int query(node \* root, int left, int right)**

**{**

**if (right < root->left || left > root->right)**

**return -INF;**

**if (left <= root->left && right >= root->right)**

**return root->add + root->max;**

**int ans1 = query(root->child\_left, left, right);**

**int ans2 = query(root->child\_right, left, right);**

**return root->add + max(ans1, ans2);**

**}**

Функция работает следующим образом. Если отрезок запроса и отрезок вершины дерева не пересекаются, функция возвращает специальное значение -INF. Если отрезок вершины дерева целиком содержится в отрезке запроса, функция возвращает сумму полей

**add**

 и

**max**

.

Иначе запрос разбивается на две части и спускается в левое и правое поддерево. Из значений, которое вернули рекурсивные вызовы из левого и правого поддерева берётся наибольшее. Функция возвращает это наибольшее, добавив значение

**add**

 из этой вершины.