* [Групповые операции в декартовом дереве](https://foxford.ru/lessons/32285/conspects/1)
* [Декартово дерево по неявному ключу](https://foxford.ru/lessons/32285/conspects/2)

**Групповые операции в декартовом дереве**

**Групповые запросы в декартовом дереве**

Как и в дереве отрезков, можно реализовать групповые запросы в декартовом дереве (например, запрос суммы, максимума, минимума на подотрезке, то есть на наборе подряд идущих элементов). Часто групповые операции используются в декартовом дереве по неявному ключу.

Например, для того, чтобы отвечать на запрос о нахождении наибольшего значения будем хранить в каждой вершине значение максимума в данном поддереве. Для этого нужно модифицировать описание структуры

**node**

:

**struct node**

**{**

**int x;**

**int y;**

**int max;**

**node \* left;**

**node \* right;**

**node(int val)**

**{**

**x = val;**

**max = val;**

**y = rand();**

**left = nullptr;**

**right = nullptr;**

**}**

**};**

Напишем функцию, которая возвращает значение максимума в данном поддереве с проверкой на то, что указатель может быть и нулевым (в этом случае возвращается минус бесконечность):

**int get\_max(node \* root)**

**{**

**if (root)**

**return root->max;**

**else**

**return -INF;**

**}**

Напишем также функцию, которая обновляет значение поля max в вершине.

**void update(node \* root)**

**{**

**if (root == nullptr)**

**return;**

**root->max = max(x, max(get\_max(root->left), get\_max(root->right)));**

**}**

Эту функцию необходимо вызывать при каждой операцией с вершиной декартова дерева, например, при каждом вызове операции

**split**

.

**pair <node \*, node \*> split(node \* root, int val)**

**{**

**if (root == nullptr)**

**return {nullptr, nullptr};**

**if (root->x <= val)**

**{**

**auto res = split(root->right, val);**

**root->right = res.first;**

**update\_max(root);**

**return {root, res.second};**

**}**

**else**

**{**

**auto res = split(root->left, val);**

**root->left = res.second;**

**update\_max(root);**

**return {res.first, root};**

**}**

**}**

И при операции

**merge**

L

**node \* merge(node \* root1, node \* root2)**

**{**

**if (root1 == nullptr)**

**return root2;**

**else if (root2 == nullptr)**

**return root1;**

**if (root1->y < root2->y)**

**{**

**root1->right = merge(root1->right, root2);**

**update\_max(root1);**

**return root1;**

**}**

**else**

**{**

**root2->left = merge(root1, root2->left);**

**update\_max(root2);**

**return root2;**

**}**

**}**

Запрос значения максимума на отрезке удобно также делать при помощи операций

**split**

 и

**merge**

: сначала разобьем дерево на три части так, чтобы интересующее нас дерево было отдельной частью, тогда для ответа на запрос достаточно вызвать функцию

**get\_max**

 от корня дерева, после чего вызвать операции

**merge**

.

**Групповое обновление в декартовом дереве**

Групповое обновление в декартовом дереве можно сделать, если в каждой вершине хранить дополнительное поле - значение групповой операции. Например, пусть необходимо реализовать групповую операцию - добавление ко всем элементам одной и той же величины. Добавим в структуру

**node**

 новое поле

**add**

 - значение групповой добавки ко всем вершинам поддерева с данным корнем. Теперь функция

**get\_max**

 будет вычисляться так:

**int get\_max(node \* root)**

**{**

**if (root)**

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

**else**

**return -INF;**

**}**

Однако, здесь возникают проблемы - при операциях split и merge происходит разделение разделение или объединение деревьев, поэтом необходимо изменять значение этой групповой добавки. Для этого необходимо реализовать функцию push, которая "снимает" значение групповой операции в вершине, "проталкивая" его в потомки — добавка спускается из этой вершины в потомки этой вершины:

**void push(node \* root)**

**{**

**if (root == nullptr)**

**return;**

**if (root->left)**

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

**if (root->right)**

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

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

**root->add = 0;**

**}**

Эту операцию нужно вызывать при каждом вызове операций

**split**

.

**pair <node \*, node \*> split(node \* root, int val)**

**{**

**if (root == nullptr)**

**return {nullptr, nullptr};**

**push(root);**

**if (root->x <= val)**

**{**

**auto res = split(root->right, val);**

**root->right = res.first;**

**update\_max(root);**

**return {root, res.second};**

**}**

**else**

**{**

**auto res = split(root->left, val);**

**root->left = res.second;**

**update\_max(root);**

**return {res.first, root};**

**}**

**}**

И при операции

**merge**

:

**node \* merge(node \* root1, node \* root2)**

**{**

**if (root1 == nullptr)**

**return root2;**

**else if (root2 == nullptr)**

**return root1;**

**if (root1->y < root2->y)**

**{**

**push(root1);**

**root1->right = merge(root1->right, root2);**

**update\_max(root1);**

**return root1;**

**}**

**else**

**{**

**push(root2);**

**root2->left = merge(root1, root2->left);**

**update\_max(root2);**

**return root2;**

**}**

**}**

Для группового обновления необходимо также разбить декартово дерево на части и в одной части установить значение поля

**add**

, после чего объединить части вместе.

#### Декартово дерево по неявному ключу

Распространенной практикой при реализации декартова дерева является хранение в каждой вершине дерева размера всего поддерева, заведем для этого поле size. Будем обновлять значение поля size в функции update:

**void update(node \* root)**

**{**

**if (root == nullptr)**

**return;**

**root->size = 1;**

**if (root->left)**

**root->size += root->left->size;**

**if (root->right)**

**root->size += root->right->size;**

**}**

Благодаря наличию этого поля можно определять в дереве k-й по счету элемент, или, например, разбивать дерево на две части. Реализуем подобную функцию split\_kth, которая разбивает дерево на две части, в первой части при этом будет k элементов:

**pair <node \*, node \*> split\_kth(node \* root, int k)**

**{**

**if (root == nullptr)**

**return {nullptr, nullptr};**

**if (get\_size(root) <= k)**

**return {root, nullptr};**

**if (k == 0)**

**return {nullptr, root};**

**push(root);**

**if (get\_size(root->left) >= k)**

**{**

**auto res = split\_kth(root->left, k);**

**root->left = res.second;**

**update(root);**

**return {res.first, root};**

**}**

**else**

**{**

**auto res = split\_kth(root->right, k - get\_size(root->left) - 1);**

**root->right = res.first;**

**update(root);**

**return {root, res.second};**

**}**

**}**

Теперь можно вообще игнорировать поле x в нашем дереве, вернее, будем считать, что в поле x хранятся нужные нам данные — элементы массива. Дерево более неупорядочено по значению поля x, а мы просто рассматриваем элементы дерева в порядке их хранения. Ключом теперь будет индекс элемента — каким он по счету является в дереве. Но при этом ключи явно не хранятся, они могут быть вычислены при необходимости, а хранятся только размеры поддеревьев. Такая структура данных называется декартовым деревом по неявному ключу.

Декартово дерево по неявному ключу следует рассматривать, как массив, который позволяет выполнять следующие операции:

1. Обращение к i-му элементу дерева (массива), его модификация (за O(nlog⁡n)).
2. Вставка и удаление элемента за \nO(log\n) в среднем.
3. Удаление фрагмента дерева за O(log⁡n+k), где k - количество удаляемых элементов.
4. Объединение двух декартовых деревьев вместе.
5. Вставка одного декартова дерева в середину другого декартова дерева и т.д.