# Структуры данных на деревьях

pakhomovee

Compiled: 28 декабря 2023 г.

# 1 Heavy-light decomposition

## 1.1 Базовая структура

### 1.1.1 Чего мы хотим?

Мы хотим разбить наше дерево на вертикальные пути так, чтобы какой-бы путь в дереве мы не взяли, он пересекался с $O(\log n)$ наших путей.

### 1.1.2 Разбиение на пути

Для каждой вершины зафиксируем сына, у которого размер поддерева максимален. Скажем, что этот сын будет лежать в том же пути, что и наша вершина. Почему же любой путь пересекает все $O(\log n)$ путей такого разбиения? Наш путь разбивается на два вертикальных, а вертикальный путь пересекает только $O(\log n)$ путей разбиения, так как, если при переходе из вершины в предка происходит смена пути, размер поддерева, в котором она находится увеличивается хотя бы вдвое. Ребра внутри одного пути назовем тяжелыми, а между путями — легкими.

### 1.1.3 А зачем это надо?

С помощью такой структуры мы можем считать почти любую функцию на пути в дереве. Для этого на каждом пути из разбиения построим дерево отрезков. Тогда при смене пути нам нужно узнать значени функции на подотрезке с помощью дерева отрезков и обновить ответ. Например, так можно считать минимум из значений в вершинах на пути и обновлять значение в вершине. Также стоит отметить, что при отсутствии запросов изменения можно отвечать на запрос за $O(\log n)$, так как все запросы к путям разбиения, кроме одного, будут вычислять функцию на суффиксе пути.

### 1.1.4 Запросы на поддеревьях

А еще, если изначально перенумеровать вершины дерева в порядке эйлерова обхода, можно воспользоваться стандартным замечанием о том, что поддерево любой вершины соответствует какому-то подотрезку эйлерова обхода. Из-за этого можно в одном дереве отрезков хранить функцию для всех путей, да еще и обрабатывать запросы на поддеревьях!

## 1.2 HLD за $O(\log n)$

Итак, мы научились делать крутую декомпозицию дерева на пути. Давайте теперь улучшим структуру, которую мы строим на каждом пути. Вместо дерева отрезков воспользуемся бинарным деревом поиска, построенным по следующему принципу:

Рассмотрим путь $(v_1, \dots, v_k)$ из декомпозиции. Выкинем из дерева все ребра между вершинами пути. Тогда $w(v)$ равно количеству вершин дерева, которые после этого находятся в поддереве $v$. При этом они, конечно, должны быть с $v$ в одной компоненте связности.

Пусть $W$ — сумма $w$ по вершинам пути. Найдем первую вершину пути, что префиксная сумма $w$ до нее включительно $\geq W$. Сделаем ее корнем нашего двоичного дерева и запустимся от кусков, на которые она разбила путь рекурсивно.

Очевидно, что глубина такого дерева $O(\log W)$. Но на самом деле всё куда лучше: у нас есть набор деревьев для каждого из путей. А еще есть легкие ребра. Из корня дерева на пути проведем ребро в «верхний» конец легкого

ребра. Получим какое-то дерево, уже не обязательно двоичное. Однако несложно понять, что наши запросы на префиксе это просто хождение от вершины $x$ к корню её дерева, а переход по легкому ребру — переход наверх 1 раз, то есть все операции, которые мы делаем, уменьшают глубину рассматриваемой вершинки на 1, поэтому достаточно показать, что глубина нового дерева $O(\log N)$. Тогда и все запросы будут работать за такую асимптотику. Но это очевидно, так как каждый переход в предка увеличивает размер поддерева хотя бы в 2 раза.

Значит, запросы к структуре будут работать за $O(\log n)$.

# 2 Минимум на пути

Пусть дано дерево с весами на ребрах и мы хотим эффективно искать минимум на пути. При этом у нас нет запросов изменения и мы хотим использовать $O(n)$ памяти.

Отсортируем ребра по неубыванию веса. Будем добавлять их в граф по одному. При этом в каждый момент времени будем хранить набор бамбуков. При добавлении ребра возьмем бамбуки, в которых лежат вершины, соединяемые этим ребром и соединим их крайние вершины этим ребром. Получим новый набор бамбуков.

Утверждается, что после добавления всех ребер мы получим бамбук, в котором минимум на пути между любой парой вершин равен минимум на пути между этими вершинами в исходном дереве. И правда, если какое-то ребро обеспечивает достижимость этого минимума, до его добавления вершины находились бы в разных компонентах связности, а потому были бы в различных кусочках бамбука.

Значит, можно построить дерево отрезков или Sparse table и отвечать на запросы минимума на пути очень легко!

# 3 Двоичные подъемы с $O(n)$ памяти

Здесь я просто приведу код из книжки Егора Горбачева. Доказательство времени работы можно прочитать в ней же.

```
void add_leaf(int v, int par) {
    parent[v] = par;
    depth[v] = depth[par] + 1;
    if (depth[par] - depth[jump[par]] == depth[jump[par]] - depth[jump[jump[par]]]) {
        jump[v] = jump[jump[par]];
    } else {
        jump[v] = par;
    }
}
```

# 4 Эйлеров обход против HLD

Рассмотрим следующую задачу на HLD:

Дано дерево. Обработайте $q$ запросов:

- добавить $x$ к весу всех вершин в поддереве $v$
- найти сумму на пути из $u$ в $v$

Мы хотим каждый запрос обрабатывать за $O(\log n)$. Для этого, конечно, можно воспользоваться HLD, но более рационально сделать следующий трюк:

Выпишем два эйлеровых обхода нашего дерева. В первом мы добавляем вершину в обход при входе в нее, а потом обходим детей. Во втором мы сначала обходим детей, а потом добавляем вершину.

Рассмотрим вершину $v$. Поддереву вершины $v$ в обоих обходах соответствуют подотрезки. Найдем префикс наименьшей длины, содержащий поддерево $v$ в каждом из обходов (пусть это $r_1$ и $r_2$). Тогда если рассмотреть множество $\{a_1, \dots, a_{r_1}\} \backslash \{b_1, b_2, \dots, b_{r_2}\}$, где $a$ и $b$ — первый и второй обходы, оно будет содержать в точности вершины пути от $v$ до корня дерева (без $v$). Значит, используя дерево фенвика на таких обходах мы можем решить исходную задачу без HLD за $O(\log n)$ времени и линию памяти!

# 5 Euler Tour Tree

## 5.1 Структура

Мы хотим научиться решать следующую задачу за $O(n \log n)$:

Задача 1.

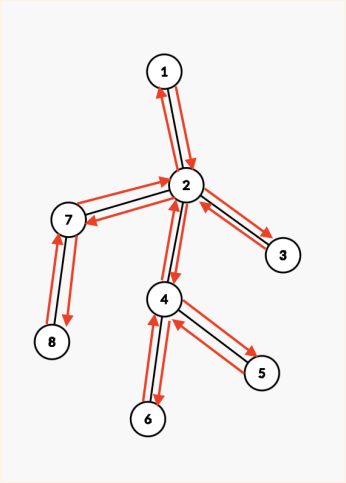
Дан лес на $n$ вершинах. Нужно научиться обрабатывать запросы:

- Добавить ребро $(u, v)$, $u$ и $v$ находятся в разных компонентах связности
- Удалить ребро $(u, v)$
- Сказать, лежат ли $u$ и $v$ в одной компоненте связности

**Proposition 5.1**

Рассмотрим любой эйлеров обход дерева, выпишем рёбра в порядке перехода по ним.
Сохраним ребра в декартовом дереве. Для каждой вершины сохраним $out(v)$ — указатель на ребро, которое выходит из неё, $in(v)$ — указатель на ребро, которое входит в неё.

- Проверка на связность: пусть $r(e)$ — корень ДД, в котором лежит ребро $e$. Проверим, что $r(in(u)) == r(in(v))$.
- Добавление ребра: введём функцию $make\_root(v)$, которая сделает вершину $v$ коренем её дерева. Для этого циклически сдвинем ДД, чтобы $in(v)$ было первым ребром в массиве. Теперь выполним $make\_root(u), make\_root(v)$, после чего добавим в конец ДД вершины $u$ ребро $(u, v)$, ДД вершины $v$, ребро $(v, u)$.
- Удаление ребра: для каждого ребра сохраним обратное ему ребро $back(e)$. Сделаем ребро $e$ первым в ДД. Найдем индекс ребра $back(e)$ в ДД (для этого нужно для каждой вершине в ДД хранить её предка). Отрежем всё левее $back(e)$, удалим рёбра $e$ и $back(e)$. Структура перестроилась корректно.




## 5.2 Другие задачи

Задача 2.

Дан лес на $n$ вершинах. Нужно научиться обрабатывать запросы:

- Добавить ребро $(u, v)$, $u$ и $v$ находятся в разных компонентах связности
- Удалить ребро $(u, v)$
- Найти XOR чисел на ребрах на пути $(u, v)$

Для этого найдем префиксный XOR до $in(u)$ и до $in(v)$. Тогда ответ равен XOR этих чисел.

Задача 3.

Дано дерево на $n$ вершинах, корень постоянный. Нужно научиться обрабатывать запросы:

- Подвесить вершину $u$ к вершине $v$, вершина $u$ до этого не встречалась
- Удалить ребро $(u, v)$, вершина $v$ является листом
- Найти $lca(u, v)$

Если мы подвесили дерево, то у каждого ребра есть направление — «вниз» или «вверх». На ребрах «вниз» поставим число 1, а на ребрах вверх число -1. Достаточно просто вырезать отрезок от $in(u)$ до $in(v)$ и на нём найти вершину ДД с наименьшей префиксной суммой.