Heavy-light декомпозиция

Филипп Грибов

29.09.2018

1 Heavy-light декомпозиция

1.1 Основная идея

Возьмём дерево и подвесим его за какую-то вершину. На каждой вершине будут висеть её дети, дети её детей, дети детей её детей и т.д. На некоторых вершинах будет висеть много вершин (в частности на корне висят все вершины дерева), а на некоторых — мало. Количество вершин, висящих на данной называется размер поддерева этой вершины.

Возьмём любую вершину. У неё есть сколько-то детей, причём на каждом висит какое-то количество вершин. Среди всех детей выбраной вершины возьмём того, размер поддерева которого максимальный. Замечаем, что этому ребёнку тяжелее, чем всем остальным, ведь на нём висит больше всего вершин. Тогда назовём ребро, соединяющее выбраную вершину с этим ребёнком «**тяжёлым**», а ребра ко всем остальным детям — «**лёгкими**». Тогда каждое ребро в дереве или тяжёлое, или лёгкое.

Теперь возьмём любое легкое ребро, соединяющее какую-то вершину A с ребёнком B. Тогда размер поддерева у A хотя бы в 2 раза больше размера поддерева у B. Это верно, так как существует вершина C, такая что C — ребёнок A и ребро между A и C — тяжелое. А значит размер поддерева у C больше размера поддерева у B. А так как размер поддерева у A больше, чем размеры поддеревьев у B и у C, то размер поддерева у B меньше, чем размер поддерева у A делённый на 2.

Тогда возьмём наше дерево, и разделим его на пути следующего вида. Пути могут начинаться в корне и в вершинах, из которых к отцу идёт лёгкое ребро. И путь идёт исключительно по тяжелым ребрам. Замечаем, что так как из каждой вершины (кроме листов) выходит ровно по одному тяжелому ребру, то каждое тяжелое ребро входит ровно в один путь и каждая вершина входит ровно в один путь. (При этом в некоторых путях может быть лишь одна вершина) Теперь возьмём путь от любой вершины до корня. Так как размер поддерева у легкого ребёнка хотя бы в 2 раза меньше размера поддерева его отца, то на таком пути будет не более $\log n$ легких ребер. А значит этот путь разбивается на $\log n$ частей выбранных нами путей по тяжелым ребрам.

Тогда если замутить какую-нибдуь структура данных типа ДО на этих выбранных тяжелых путях, то мы сможем отвечать на много запросов про пути в дереве и делать модификации.

1.2 Реализация

1.2.1 Задача

Есть дерево. На вершинах записаны числа. Бывают модификации в точке и запрос минимум на пути.

1.2.2 Массивы

$s[a]$ — список детей вершины a
$sz[a]$ — размер поддерева вершины a
$tin[a]$ — время входа в вершину a
$first[a]$ — первая вершина на тяжелом пути, содержащем a
$pre[a]$ — предок вершины a
$segtree$ — дерево отрезков, которое отвечает на запросы

1.2.3 Инициализация

По идее мы могли бы сделать по ДО для кажого путии и работать с разными ДО. Но мы нормальные люди и поэтому будем иметь только одно ДО. Как мы это будем делать?

Для начала посчитаем у каждой вершины размер её поддерева. Далее поменяем порядок детей у каждой вершины так, чтобы первым ребёнком был ребёнок с максимальным размером поддерева, т.е. чтобы это был ребёнок, к которому ведёт тяжелое ребро. Замечаем, что после такого изменения все вершины, лежащие на пути из тяжелых ребер будут лежать подряд в эйлеровом обходе.

Мы будем считать, что в эйлеровом обходе каждая вершина лежит рано 1 раз и она туда добавляется когда мы впервые входим в вершину

Посчитаем $tin[i]$ для каждой вершины. Замечаем, что $tin[i]$ как раз равен позиции вершины в эйлеровом обходе. Так же в dfs посичтаем $first[a]$. $first[0] = 0$, а дальше у первого ребёнка вершины a, $first$ от него равен $first[a]$ (так как ребро между a и ним тяжелое), а у всех остальных детей a их $first$ равен им самим (так как ребра между a и ними — лёгкие).

Далее заинициализируем ДО. На i-м месте будет стоять начальный вес в вершине, стоящей на i-м месте в эйлеровом обходе, т.е. вес вершины, tin которой равен i.

1.2.4 Запрос изменения

Если нам надо поменять вес вершины i, то в ДО меняем число на позиции $tin[i]$.

1.2.5 Минимум на пути

Пусть есть вершины A и B. Пусть C — их LCA. Тогда чтобы найти минимум на пути от A до B нам достаточно найти минимум на вертикальных путях от A до C и от B до C.

Будем это делать так: есть переменная $ans = \infty$ Далее пока $first[A] \neq first[C]$ обновляем ans с миннимумом на пути от $first[A]$ до A. Это делается одним запросом к ДО, так как $first[A]$ и A лежат на одном тяжёлом пути, который в свою очередь лежит подряд в эйлеровом обходе, и нам надо сделать запрос min на отрезке от $tin[first[A]]$ до $tin[A]$. Далее делаем $A = pre[first[A]]$. т.е. переходим по легкому ребру, выходящему вверх из $first[A]$.

В какой-то момент мы получим, что $first[A] = first[C]$, это значит что текающая A и C находятся на одном тяжелом пути. Тогда все вершины на пути от A до C лежат подряд в эйлеровом обходе, и тогда за одно обращение к ДО мы можем найти минимум на пути от A до C.

Так как на пути от A до корня не больше $\log n$ лёгких ребер, то на пути от A до C так же не более $\log n$ легких ребер. Значит число заросов к ДО которые мы сделали это $O(\log n)$.

Аналогично найдём минимум на пути от B до C, и того мы найдём минимум на пути от A до B за $O(\log^2 n)$.

1.3 Задачи и применения

1.3.1 Появляется ещё запрос минимум в поддереве

Замечаем, что все вершины в поддереве идут подряд в эйлеровом обходе. Тогда если мы у каждой вершины ещё посчитаем время выхода, то подотрезок эйлерова обхода между временем входа и временем выхода содержит по разу все вешины из поддерева данной. Так мы дополнительно можем отвечать на запрос минимум в поддереве.

1.3.2 Массовые операции

Операции можно делать массовыми на пути, их применение будет аналогично ответу на запрос на пути.

1.3.3 Кэширование запросов

Пусть нет никаких запросов изменения, а есть только обычные запосы минимума. Тогда замечаем, что когда мы отвечаем на запросы, мы $O(\log n)$ раз искали минимум на отрезке $[first[A], A]$ и только один раз был запрос на минимум не с начала тяжелого пути. Тогда для кажого пути для каждой вершины на нём посчитаем минимум на от начала пути до этой вершины. Тогда на запрос минимум на пути от $first[A]$ до A мы можем отвечать за $O(1)$. Тогда минимум на пути от A до C мы теперь будем искать за $O(\log n)$.