1 Link-cut tree. Описание операций и способа хранения

Link-cut tree — это структура данных, хранящая лес деревьев и позволяющая делать следующие операции:

1) $min(v)$ - искать минимум на пути от корня до вершины v.

2) $add(v, c)$ - прибавлять константу c на пути от корня до вершины v.

3) $link(u, w)$ - подвешивать корень дерева u за вершину другого дерева w.

4) $cut(v)$ - отрезать дерево с корнем в вершине v.

Все эти операции будут выполняться за амортизированный $O(logn)$.

Как мы храним весь наш лес: мы разбиваем все ребра на два типа — сплошные и пунктирные. Неважно, как разбиваем, единственное условие — у каждой вершины не более одного ребенка по сплошному ребру. И так, мы по факту разбили наше дерево на сплошные пути и еще пунктирные ребра. Сплошные пути мы будем хранить как *splay*-деревья (в частности, одну вершину тоже) с ключом глубины вершины в дереве, а в корнях будут находиться ссылки, которые мы назовем *pathparent* — это будет ссылка на ту вершину, которая соединена сверху с этим путем по пунктирному ребру.

Теперь надо понять, как реализовываются все операции. Введем вспомогательную операцию $expose(u)$ — она перестраивает наше дерево так, что после нее u лежит на сплошном пути с корнем дерева, и u является корнем соответствующего *splay* —дерева. Реализуем её.

Алгоритм работы *expose*: для начала сделаем $splay(u)$, после неё u станет корнем своего *splay*—дерева, и в ней будет храниться *pathparent* — давайте перейдем в него, пусть это v. Сделаем $splay(v)$ и посмотрим на то, как теперь устроено это *splay*-дерево: т.к. у нас ключ — глубина, то в левом поддереве находится всё, что выше вершины v в изначальном дереве, а справа — то, что ниже. Ну и давайте просто сделаем *split* в этом *splay*-дереве так, чтобы правое поддерево отсоединилось от v, и положим в корень правого поддерева $pathparent = v$. Теперь мы можем безболезненно присоединить *splay*-дерево с корнем u к *splay*-дереву с корнем в v за сплошное ребро uv вместо правого поддерева — это просто операция $merge$ в *splay*-деревьях. Ну а теперь продолжаем это всё, пока мы в итоге не окажемся в корне (мы же вверх поднимаемся и прокладываем сплошные ребра просто, поэтому в итоге окажемся в корне и будет сплошной путь от u до корня).

Дальше все операции реализуются очевидным образом: $link(u, w)$ — просто проведем пунктирное ребро $u \to w$, нам это ничего не стоит. $cut(v)$ — если из v вверх не идет сплошное ребро, то просто удалим пунктирное ребро $a.k.a$ ссылку из v. Если там сплошное ребро - сначала запомним $t = pathparent(v)$, сделаем $expose(t)$, и отрежем правое поддерево в соответствующем *splay*-дереве.

Чуть сложнее с первыми двумя операциями — min и add. Тут самое важное - надо уметь следить за операциями на поддереве *splay*-дерева. Для этого мы будем держать в каждой вершине не только глубину и вес, но и *модификатор* — в начале это просто число, которое равно 0. Разберемся с $add(u, c)$: сначала делаем $expose(u)$ и получаем *splay*-дерево на соответствующем пути. Ну и просто добавляем к модификатору в $u+ = c$. Чтобы получить вес вершины теперь надо взять ее изначальный вес и просуммировать все модификаторы на пути от корня до u. В целом все просто, но надо также сделать согласованность с предыдущими операциями — $link$ и cut.

2 Оценка времени работы

Теорема 1. *expose выполняется за амортизированный* $O(logn)$

Из этого будет следовать, что все работает за амортизированный логарифм.

Доказательство. Для начала заметим очевидную вещь, что на любом пути в дереве у нас не более $logn$ легких ребер — таких, что если это ребро из u в родителя v, а $d(u), d(v)$ — кол-во их детей, то $d(u) \leqslant \frac{1}{2}d(v)$ (иначе у нас было бы просто больше чем $2^{logn} = n$ вершин). Теперь поймем, что сама операция *expose* — это преобразование сплошных ребер в пунктирные и наоборот, обозначим кол-во таких преобразований из пунктирных в сплошные за $M := |\{D \to S\}| = |\{L \cap D \to S\}| + |\{H \cap D \to S\}|$ (H — мн-во тяжелых ребер, L — легких, S — сплошных, D — пунктирных). Также введем потенциал $\Phi_a = n - |\{H \cap D\}|$, $\Delta\Phi_a$ — изменение потенциала за одну операцию *expose*.

Лемма 2. $M + \Delta\Phi_a \leqslant 2 \cdot logn.$

Доказательство.

$$
\begin{aligned}
M + \Delta\Phi_a = M + |\{H \cap S \to D\}| - |\{H \cap D \to S\}| \leqslant \\
\leqslant M + |\{L \cap D \to S\}| - |\{H \cap D \to S\}| = \\
= 2 \cdot |\{L \cap D \to S\}| \leqslant 2 \cdot logn.
\end{aligned}
$$

Первое неравенство появляется из-за того, что мы можем поменять сплошное ребро на пунктирное только в случае, когда мы меняем другое пунктирное ребро на сплошное, причем т.к. они являются смежными к одной вершине, то ровно одно из них тяжелое (у одной вершины не может быть > 1 тяжелого ребра). А второе неравенство просто следует из того, что кол-во таких ребер явно меньше чем кол-во легких ребер на пути, а их как мы знаем, не больше чем $logn$. □

Теперь заметим, что за хотя бы n операций *expose* у нас M в среднем будет не больше чем $1 + 2 \cdot logn$ (т.к. изначальное значение потенциала $= n - 1$). Осталось теперь сказать про сам *expose*.

Введем обозначения как для доказательства работы *splay*-дерева: $s(v)$ — кол-во вершин в поддереве v, $r(v) := log(s(v))$. Из док-ва для *splay*-дерева, мы знаем что стоимость i-ой операции *splay* не превосходит $1 + 3 \cdot (r(t) - r(v))$. Т.к. у нас таких операций было M в *expose*, то амортизационная стоимость *expose* не превосходит $M + 3 \cdot (logn - log(s(v))) = O(logn)$. □