Heavy-light декомпозиция

Heavy-light декомпозиция — техника разбиения подвешенного дерева на множество путей для решения задач о запросах на пути в дереве (в том числе с модификациями).

Содержание

Описание задачи

Задача:

Имеется подвешенное дерево T с n вершинами и необходимо проводить операции на нем на пути от вершины v до вершины u.

Примеры запросов:

- сумма на пути,
- максимум на пути,
- количество рёбер на пути, вес которых больше заданного c.

Примеры модификаций:

- модификация одного ребра,
- прибавление к весу всех рёбер на пути,
- установка веса всех рёбер на пути в заданное c.

Множество подобных запросов делаются за время за полином от логарифма (обычно $O(\log^2 n)$) с помощью heavy-light декомпозиции.

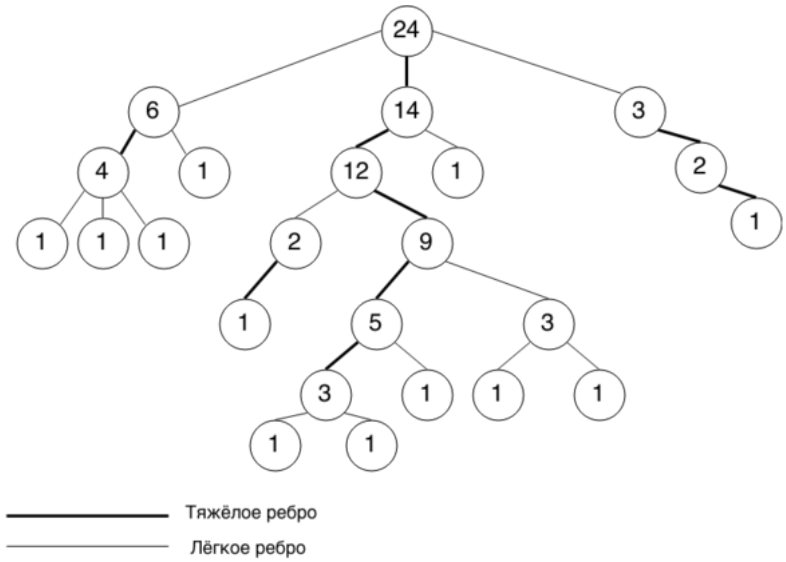
Описание декомпозиции

Необходимо составить такую декомпозицию дерева на множество рёберно-непересекающихся путей, что при прохождении от одной вершины до другой произойдет смена не более $O(\log n)$ путей из декомпозиции.

Декомпозиция заключается в классификации всех рёбер дерева T в 2 вида: легкие и тяжёлые. Введём функцию $s(v)$, которая будет обозначать размер поддерева вершины v.

Тяжёлые ребра (англ. *heavy edge*) — ребра (u, v) такие, что $s(v) \geqslant \dfrac{s(u)}{2}$.

Лёгкие ребра (англ. *light edge*) — соответственно все остальные.



Пример разбиения. В вершинах указан размер поддерева.

Очевидно, что из вершины может выходить как максимум одно тяжёлое ребро, т.к. иначе у нас есть два поддерева по как минимум $\dfrac{s(u)}{2}$ вершин, а также сама вершина u. Итого $s(u) + 1$ вершин, тогда как у нас всего $s(u)$ вершин в поддереве.

Теперь рассмотрим вершины, из которых не ведет вниз ни одно тяжёлое ребро. Будем идти от них вверх до корня или пока не пройдем легкое ребро. Получится какое-то множество путей. Утверждается, что полученная таким образом декомпозиция будет являться искомой и корректной.

Утверждение:

Полученная декомпозиция является искомой.

▷

Докажем по отдельности корректность декомпозиции.

1. Все рёбра покрыты путями.
   Есть два типа вершин: те, из которых ведёт ровно одно тяжёлое ребро и те, из которых не ведёт ни одного тяжёлого ребра. Для первого типа вершин мы дойдем до них некоторым путём через тяжёлое ребро снизу по определению выбора путей, а лёгкие рёбра ведущие из неё возьмем как последние рёбра в соответствующих путях. Для второго типа вершин мы по определению выбора путей возьмем их как начальные и пойдем вверх.
   Таким образом все рёбра будут покрыты.
2. Все пути не пересекаются.
   Докажем от противного. Пусть мы взяли какое-то ребро дважды. Это значит, что из какой-то вершины v ведет более 1 тяжелого ребра в детей. Эти ребра относятся к разным путями, однако пути имеют хотя бы общее ребро — ребро из v в отца v. Более 1 тяжелого ребра из вершины идти не может, следовательно, получили противоречие.
3. При прохождении пути от вершины v до вершины u произойдет смена не более, чем $O(\log n)$ путей.
   Докажем эквивалентный факт, что при пути от любой вершины до корня мы сменим не более, чем $O(\log n)$ путей. Рассмотрим лёгкое ребро. Заметим, что проход вниз по такому ребру уменьшает размер поддерева как минимум в 2 раза. Но смена пути может произойти только при переходе по лёгкому ребру. Таким образом мы сменим не более $O(\log n)$ путей.

◁

Существует вариант heavy-light декомпозиции на вершинно-непересекающихся путях. Чтобы получить такой путь нужно всего-лишь выкинуть последнее ребро из всех путей в рёберно-непересекающейся декомпозиции. Это может быть удобно при решении задач, где веса находятся не на рёбрах, а на вершинах и соответствующие запросы также делаются на вершинах.

Применение

Сумма на пути

Классическая задача о сумме на пути в дереве с n вершинами может быть решена с помощью heavy-light декомпозиции за время $O(\log^2 n)$. Возможны модификации веса.

Построим дерево отрезков над каждым путём. Рассмотрим запрос $sum(u, v)$. Найдем вершину c, которая является $\mathrm{LCA}(u, v)$ (например с помощью двоичного подъема. Мы разбили запрос на два: (u, c) и (c, v), на каждый из которых можно легко ответить разбив его на множество путей из декомпозиции и ответив на каждый путь из этого множества по отдельности за $O(\log n)$ с помощью дерева отрезков на этом пути. Всего таких путей нужно будет рассмотреть $O(\log n)$. Итого мы способны решить эту задачу за время $O(\log^2 n)$.

Хоть это и не самый эффективный способ для решения этой задачи, но можно заметить, что навесив дерево отрезков на каждый путь мы способны отвечать на любые операции, определяемые на множестве, на котором данная операция ассоциативна, и существует нейтральный элемент относительно этой операции, то есть на моноиде (операции, поддерживаемые деревом отрезков), такие как: сумма на пути, максимум на пути, количество рёбер на пути, удовлетворяющих какому-то свойству.

LCA

Задача о наименьшем общем предке для двух вершин в дереве с n вершинами также может быть решена с помощью heavy-light декомпозиции. Воспользуемся основной идеей: декомпозиция разбивает все вершины дерева на реберно-непересекающиеся пути так, что поднимаясь от любой вершины до корня дерева придется сменить не более $\log n$ различных путей.

Лемма:
Пусть есть вершины u и v, лежащие на разных путях. При этом U, V — корни путей, на которых они лежат. Если U более удален от корня дерева, чем V, то $\mathrm{LCA}(u, v) = \mathrm{LCA}(U, v)$.
Доказательство:
▷

Допустим, пути не пересекаются. Предположим, что $\mathrm{LCA}(u, v)$ и $\mathrm{LCA}(U, v)$ это разные вершины. Тогда существует вершина, на пути от u к U, являющаяся LCA. Значит LCA должен принадлежать двум путям, но по предположению пути не пересекаются. Тем самым пришли к противоречию.

Теперь рассмотрим случай, когда пути пересекаются. Пути не могут совпадать более, чем в одной вершине, так как построенная декомпозиция является реберно-непересекающейся. При этом корень одного из путей является вершиной другого (либо корни совпадают, что равносильно), поскольку в противном случае пути пересекаются в более чем 1 вершине, что противоречит предыдущему условию. LCA должен принадлежать двум путям, значит именно этот корень и будет LCA.
◁

Препроцессинг

Построим heavy-light декомпозицию данного нам дерева. Для каждой вершины, помимо её предка, будем хранить дополнительно следующие значения:

1. Расстояние до корня дерева.
   Вычисляется за $O(1)$ с помощью времен входа\выхода в каждую вершину.
2. Корень пути, на котором лежит вершина.
   Поскольку вершина может принадлежать нескольким путям, выберем тот, чья начальная вершина наиболее удалена от корня дерева. Имея разбиение на пути, найти корень можно за $O(1)$.
3. Вторая вершина этого пути.
   Аналогично, находится за $O(1)$ при построении.

Вычисление LCA

Найдем LCA для двух вершин. Для этого будем рекурсивно подниматься от этих вершин в направлении корня. Пусть на данной итерации рассматриваем вершины u и v. Заметим, что если эти вершины лежат на одном пути, то ответ — это такая вершина (u или v), которая находится ближе к корню. Очевидно, что если расстояние от корня до u меньше, чем расстояние до v, то u является предком v. Иначе, наоборот.

Для проверки этого условия недостаточно знать только корни путей, потому что несколько путей могу иметь общий корень. Но любые два пути пересекаются не более чем в одной вершине. Воспользуемся этим фактом.

Пусть a и b — вторые вершины путей, содержащих вершины u и v соответственно. Важно заметить, что любая вершина, помимо корня дерева является некорневой вершиной какого-либо другого пути, поэтому такие a и b всегда существуют.

- Заметим, что если $a = b$, то u и v лежат на одном пути. Этот случай мы уже рассмотрели ранее.
- Если это не так, то вершины лежат на разных путях. По лемме, так как пути реберно не пересекаются, то ответ не изменится, если вместо одной из вершин взять корень того пути, на котором она лежит. Эту операцию будем производить с той вершиной, чей предок наиболее удален от корня. Рекурсивно запустимся от выбранной и оставшейся вершин.

Очевидно, что в результате придем или в одну и ту же вершину, или одна из вершин окажется на пути от корня к другой. Тем самым мы найдем LCA.

Псевдокод

Объявим несколько массивов для хранения дополнительной информации:

- `dist` — расстояние от корня до вершины.
- `last` — начало пути, на котором лежит вершина. Из всех путей выбирается путь с самой удаленной от корня дерева начальной вершины.
- `turn` — вторая вершина этого пути.

Ниже представлен псевдокод функции получения наименьшего общего предка:

```
// Находит наименьшего общего предка вершин u и v
int lca(int u, int v)
  // Проверяем вторые вершины путей, содержащих u и v.
  if (turn[u] == turn[v])
    // Ответ найден, выберем ближайшую к корню.
    if (dist[u] < dist[v])
      return u
```

```
    else
      return v

  // Рекурсивно запустимся от вершины, чей предок наиболее удален от корня дерева.
  if (dist[last[u]] > dist[last[v]])
    return lca(last[u], v)
  return lca(last[v], u)
```

Асимптотика

- **Память**: для реализации алгоритма требуется $O(n)$ памяти.
- **Препроцессинг**: heavy-light декомпозиция строится за $O(n)$, вся дополнительная информация считается за $O(1)$ для каждой из вершин.
- **Запросы**: по свойству heavy-light декомпозиции, на пути от вершины к корню мы сменим не более $\log n$ путей. Значит время выполнения запроса также $O(\log n)$.

Реализация

Ниже будет приведена реализация запроса сумма на пути между любыми двумя вершинами в дереве без запросов модификации. Все запросы, сводящиеся к навешиванию дерева отрезков на пути из декомпозиции делаются похожим образом.

Опущены некоторые детали реализации: построение и дерево отрезков.

- $\mathrm{pathPos}$ — функция, позволяющая найти смещение вершины в пути относительно корня пути,
- $\mathrm{getValue}(\mathtt{i}, \mathtt{j})$ — функция, позволяющая найти вес $\mathtt{j}$-ого ребра в $\mathtt{i}$-ом пути.

Пример реализации запроса суммы на пути:

```
int query(int u, int v)
    int res = 0
    int root = корень пути, в котором находится u
    while root не является предком v      // поднимаемся до тех пор, пока наш путь не содержит общего предка u и v
        segmentTree = дерево отрезков, соответствующее пути, в котором лежит u
        res += segmentTree.sum(0, pathPos(u))
        u = предок root                   // вырезали нижний путь и подняли нижнюю вершину до нижней вершины следующего пу
        root = корень пути, в котором находится u

    root = корень пути, в котором находится v
    while root не является предком u      // аналогично прошлому while, но с другой стороны
        segmentTree = дерево отрезков, соответствующее пути, в котором лежит v
        res += segmentTree.sum(0, pathPos(v))
        v = предок root
        root = корень пути, в котором находится v

    // последний путь (тот, что содержит общего предка) обрезан с двух сторон полученными вершинами
    segmentTree = дерево отрезков, соответствующее пути в котором лежит u
    res += segmentTree.sum(min(pathPos(u), pathPos(v)), max(pathPos(u), pathPos(v)))
    return res
```

См.также

- Метод двоичного подъема
- Дерево отрезков. Построение
- Link-Cut Tree

Источники информации

- Wikipedia — Heavy path decomposition (http://en.wikipedia.org/wiki/Heavy_path_decomposition)
- MAXimal :: algo :: Heavy-light декомпозиция (http://e-maxx.ru/algo/heavy_light)
- Habrahabr — Алгоритм поиска наименьшего общего предка в дереве (https://habrahabr.ru/post/198464)

Источник — «http://neerc.ifmo.ru/wiki/index.php?title=Heavy-light_декомпозиция&oldid=65279»

- Эта страница последний раз была отредактирована 9 мая 2018 в 22:16.