Дерево доминаторов

Лазарев Никита

30 июня 2022 г.

1 Введение

1.1 Определение

Давайте рассмотрим ориентированный граф (V, E). Пусть в этом графе отмечена какая-то вершина S, назовем ее истоком.

Вершина u считается доминатором вершины v, если все пути из S в v проходят через u. Определим непосредственным доминатором вершины v, такую вершину u, что количество вершин, для которых u является доминатором минимально. Для удобства давайте дальше будем называть непосредственный доминатор для вершины v, как $idom_v$ (immediate dominator - непосредственный доминатор).

Предположим, что мы знаем все $idom_v$. Давайте построим граф, в котором вершины будут идентичны изначально графу, а ребра будут вида $idom_v \to v$. Именно такой граф и называется деревом доминаторов.

Лемма 1 *Давайте докажем, что дерево доминаторов является деревом.*

Предположим, что это не так, тогда у нас есть три вершины a, b, c, такие что мы имеем пути $a \to b$, $b \to c$, $a \to c$, которые не пересекаются по вершинам, кроме вершин a, b, c, при чем путь от b до c состоит из одного ребра. Но тогда у нас есть два пути до вершины c, один из которых не проходит через b, а значит b не является доминатором c. Получили противоречие.

1.2 Построение дерева доминаторов за $O(nm)$

Сначала очевидно обработаем все вершины до которых мы не можем добраться из S. Далее будем удалять каждую по одной вершине и запускать DFS из S. Тогда мы получим для каждой вершины список вершин для которых она является доминатором. Построить дерево доминаторов по этому списку достаточно легко. Давайте сделаем на полученном списке граф, очевидно что он получится DAG'ом. Теперь для каждой вершины нам нужно найти самый длинный путь до этой вершины из S. Именно такой путь и должен быть в нашем дереве доминаторов. Все эти этапы можно делать за $O(nm)$

2 Полудоминаторы

2.1 Определение

Для построения дерева доминаторов за $O((n+m)logn)$ нам понадобится такая вещь как полудоминаторы.

Перед тем как определять полудоминаторы. Давайте перенумеруем вершины в порядке входа в дерево обхода DFS.

Определим полудоминатор вершины v как такую минимальную вершину $sdom_v$, что существует путь через вершины $sdom_v, v_1, v_2, v_3, \ldots, v$, такой что $v_i > v$ (здесь номер вершины это уже измененная версия, то есть tin_v).

Лемма 2 *Давайте дакажем, что если $v \neq S$, то $sdom_v$ предок вершины v в дереве DFS.*

Заметим, что самый первый претендент на то чтобы быть полудоминатором это непосредственный предок v. Тогда $sdom_v \leq parent_v$. Если $sdom_v$ не является предком v, то тогда $sdom_v$ лежит в каком-то левом поддереве относительно v. Но тогда у нас есть путь из левого поддерева в v. А это значит что мы должны были посетить вершину v в этом поддереве и наше дерево обхода DFS неверное.

Лемма 3 *Давайте дакажем, что если $v \neq S$, то $idom_v$ предок вершины $sdom_v$ в дереве DFS.*

Во-первых, очевидно, что $idom_v$ является предком v(поймите сами почему так).

Во-вторых, предположим, что $idom_v$ лежит ниже чем $sdom_v$, но тогда мы имееем два различных путя из S в v(банально по определению полудоминатора). Имеем противоречение.

2.2 Поиск полудоминаторов

Посмотрим на наш $sdom_v$. Каждому полудоминатору, как мы сказали ранее, соответствует путь $sdom_v, v_1, v_2, v_3, \ldots, v$, при чем $v_i > v$. Давайте посмотрим на предпоследнюю вершину, то есть v_m. По условию у нас есть ребро $v_m \to v$. Будем пересчитывать полудомнатор именно через такие вершины v_m.

Посмотрим на какую-то вершину v, мы можем дойти до нее от полудоминатора через вершины B, такие что $B \to v$. Если $B < v$, то B - претендент на полудоминатор, но не более, потому что мы не можем проходить через эту вершину на пути в v.

Для остальных $B > v$ мы хотим пересчитать $sdom_v$ как-то поумнее.

Лемма 4 *Возьмем две наши вершины v и B. Они лежат в разных поддеревьях дерева обхода DFS. Давайте посчитаем их lca в этом дереве, пусть это будет вершина S. Утверждается, что $sdom_v = minsdom_u$, для всех таких u, которые лежат на пути от S до B, кроме конечно вершины S.*

Докажем это. Пусть мы уже знаем полудоминатор C нашей вершины v. Если у нас есть ребро $C \to v$, то мы победили. Иначе рассмотрим путь из в v(этот путь будет вида $C, v_1, v_2, v_3, \ldots, B, v$).

Давайте рассмотрим минимальную вершину D среди вершин $v_1, v_2, v_3, \ldots, B$. Легко осознать, что C подходит под определение полудоминатора для D, поэтому $sdom_D \leq C$. Однако заметим, что полудоминатор $sdom_D$ является полудоминатором A. Это происходит из-за того что все вершины на пути из $sdom_D$ в D больше чем v, а также у нас есть путь из D в v по вершинам больше v. Из этих рассуждений можно сделать вывод, что $C = sdom_D$.

Из этого рассуждения мы можем понять, что либо у нас полудоминатор вершины это какой-то ее сосед, либо мы ее будем пересчитывать через вершины с бОльшим номером чем v.

Следующее существенное замечание состоит в том, что D лежит на пути из C в B. Давайте докажем это. Так как $D < B$ и B достижима из D, то D лежит выше в дереве дфса B.

2.3 Построение полудоминаторов за $O(nm)$

Давайте будем рассматривать вершины в порядке уменьшения номера.

Пусть сейчас мы рассматриваем вершину v. Посмотрим на все вершины B, такие что есть ребро $B \to v$. Если $B < v$, то B кандидат на $sdom_v$. Иначе давайте будем подниматься из B выше в дереве DFS'а пока не наткнемся на вершину с номером меньше v. Если мы прошли через вершины $D_1, D_2, D_3, D_4 \ldots, D_k$, то у нас есть $sdom_{D_1}, sdom_{D_2}, \ldots, sdom_{D_k}$ - претенденты на $sdom_v$. Мы можем просто банально пройтись по всем этим вершинам и прорелаксировать $sdom_v$. Заметим что для каждого ребра мы будем подниматься в дереве DFS'а не более n раз, поэтому ассимптотика этого решения будет $O(nm)$.

2.4 Построение полудоминаторов за $O((n+m)logn)$

В решении за $O(nm)$ у нас дольше всего времени занимало поднятие по вершинам до LCA. Это можно оптимизировать либо с помощью HLD, либо можно сделать умнее и написать структуру под название link-eval.

Она поддерживает следующие операции:

- link(a, b) - подвесить b за a.
- eval(a) - посчитать какую-то функцию на пути от до корня поддерева в котором находится a.

Тогда алгоритм описанный выше можно реализовать за $O((n+m)logn)$.

3 Поиск доминаторов

3.1 Важные факты

Перед тем как рассказать про алгоритм давайте опишем еще пару важных свойств.

Лемма 5 *Пусть существует вершина v, и мы знаем ее $idom_v$. Тогда если есть какая то вершина u, такая что она предок v, а $idom_v$ предок v, то $idom_v$ является предком $idom_u$.*

Преположим, что это не так. Тогда есть два путя из $idom_u$ в u. И так как u это предок v, поэтому есть два пути из $idom_u$ в v. Из-за этого есть путь из $idom_u$ в v, который не проходит через $idom_v$. Получили противоречие.

Лемма 6 *Предположим есть вершина v и мы знаем ее $sdom_v$. Утверждается, что если для всех вершин u, таких что и предок v, а $sdom_v$ предок u, верно что $sdom_v \leq sdom_u$, то $idom_v = sdom_v$.*

Давайте банально предположим обратное. Тогда у нас есть путь из $idom_v$ в v в обход $sdom_v$. Как может идти этот путь? Рассмотрим два варианта.

Первый вариант это когда у нас есть ребро на пути $idom_v \to sdom_v$, который ведет либо в поддерево вершины v и потом поднимается в v, либо идет в правое поддерево и потом ведет каким-то ребром в v. Но в таком случае $sdom_v$ должен быть выше(а именно той вершиной из которой ведет это ребро). Получили противоречие.

Второй вариант. Это когда у нас есть ребро которое выходит из вершины на пути $idom_v \to sdom_v$ и входит в вершину на пути $sdom_v \to v$, но тогда у нас $sdom_v \leq sdom_u$, что неверно по условию.

Лемма 7 *Предположим есть вершина v и мы знаем ее $sdom_v$. Утверждается, что если для всех вершин u, таких что и предок v, а $sdom_v$ предок u, верно что $sdom_u \leq sdom_v$, то $idom_v = idom_u$.*

По теореме 5 имеем что $idom_u \leq idom_v$. Предположим, что $idom_u < idom_v$. Тогда должен быть путь $idom_u \to u$ в обход $idom_v$. Рассмотрим два варианта схожих теореме 6.

В первом варианте у нас путь не содержит ребра на пути $sdom_u \to u$. Рассуждения в этом варианте аналогичны первому варианту из 6.

Второй вариант интереснее. Давайте посмотрим на наш путь в обход $idom_v$. Возьмем первую вершину на этом пути для которой выполняется условие что она предок v, а также что $idom_u$ - ее предок. Пусть эта вершина будет с номером y. Заметьте, что мы не накладываем никаких условий положения относительно u (y можем быть как ниже так и выше u).

Если у нас $u < y$ (то есть y выше u в дереве обхода DFS), то у нас есть путь из $idom_v$ в u в обход $idom_u$. Получили противоречие.

Если $u > y$ (то есть y ниже u), то $sdom_y < sdom_u$. Это противоречит тому, что $sdom_u \to min$. Получили противоречие.

Тогда $idom_v = idom_u$. Ч.т.д.

Лемма 8 *Предположим есть вершина v и мы знаем ее $sdom_v$. Найдем вершину u с минимальным $sdom_u$, такую что $sdom_v$ - предок u, а u - предок v. Тогда утверждается, что:*

$$idom_v = \begin{cases} sdom_v & \textit{если } sdom_v = sdom_u \\ idom_u & \textit{в противном случае} \end{cases}$$

Это утверждение легко выводится из теорем 6 и 7.

3.2 Нахождение $idom$ за $O((n+m)logn)$

Перед тем как строить доминаторы, давайте построим полудоминаторы за $O((n+m)logn)$.

Теперь из утверждения в теореме 8 легко понять как искать доминаторы. Во-первых $idom$ надо считать в порядке обхода DFS. Во вторых нам надо уметь находить минимум на пути. Это легко сделать просто поддерживая двоичные подъемы и минимум на них.

Мы смогли построить дерево доминатров за $O((n+m)logn)$.

4 Реализация

Листинг 1: структура link-eval

```
struct linkeval {
    vector<pair<int, int>> a;
    vector<int> link;

    linkeval(int n) {
        a.assign(n, make_pair(1e9, 1e9));
        link.resize(n);
        for (int i = 0; i < n; i++)
            link[i] = i;
    }

    linkeval() {}

    void unite(int v, int u) {
        get(v);
        get(v);
        if (link[v] != link[u])
            link[u] = v;
        get(u);
    }

    pair<int, int> get(int b) {
        if (link[b] == b) {
            return a[b];
        } else {
            pair<int, int> answer = get(link[b]);
            a[b] = min(answer, a[b]);
            link[b] = link[link[b]];
            return a[b];
        }
    }
};
```

Листинг 2: построение полудоминаторов

```
linkeval lnk;

void buildsdom(int v) {
    for (int i: gr[v]) {
        if (tin[i] > tin[v]) {
            sdom[v] = min(sdom[v], lnk.get(i));
        }
        sdom[v] = min(sdom[v], make_pair(tin[i], i));
    }

    lnk.a[v] = sdom[v];
    for (int i: g[v]) {
        if (tin[v] < tin[i])
            lnk.unite(v, i);
    }
}
```

Листинг 3: построение дерева доминаторов

```
bool upper(int a, int b) {
    return tin[a] < tin[b] && tout[b] < tout[a];
}

void buildidom(int v, int p = 0) {
    if (v) {
        up[v][0] = p, d[v][0] = v;
        for (int i = 1; i < 18; i++) {
            up[v][i] = up[up[v][i - 1]][i - 1];
            if (sdom[d[v][i - 1]].first < sdom[d[up[v][i - 1]][i -
   1]].first)
                d[v][i] = d[v][i - 1];
            else
                d[v][i] = d[up[v][i - 1]][i - 1];
        }
        int ans = v, cur = v;
        for (int i = 17; i >= 0; i--) {
            if (!upper(up[cur][i], sdom[v].second)) {
                if (sdom[ans].first > sdom[d[cur][i]].first)
                    ans = d[cur][i];
                cur = up[cur][i];
            }
        }
        if (sdom[ans].second == sdom[v].second)
            idom[v] = sdom[v].second;
        else
            idom[v] = idom[ans];
    }
    used[v] = true;
    for (int i: g[v])
        if (!used[i])
            buildidom(i, v);
}
```