Наименьший общий предок

Миронов Валерий

4 декабря 2019 г.

План

1. Постановка задачи
2. Наивное решение
3. Метод двоичного подъема

Постановка задачи

Пусть T – корневое дерево с корнем $v_0 \in V = V(T), n = |V|$.

Определение

Наименьшим общим предком (*least common ancestor*, LCA) вершин $u, v \in V$ называется вершина $w \in V$, которая имеет наибольшую глубину (расстояние до корня) среди всех предков u и v.

Предложение

Для любых вершин $u, v \in V$ их наименьший общий предок существует и единственный.

- Далее мы будем обозначать наименьшего общего предка вершин $u, v \in V$ как $LCA(u, v)$.

Постановка задачи

Пусть T – корневое дерево с корнем $v_0 \in V = V(T), n = |V|$.

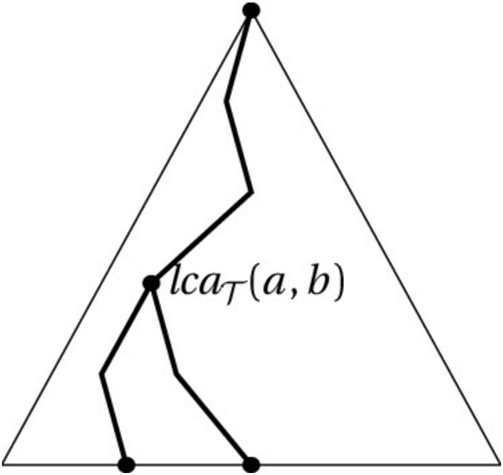
Определение

Наименьшим общим предком (least common ancestor, LCA) вершин $u, v \in V$ называется вершина $w \in V$, которая имеет наибольшую глубину (расстояние до корня) среди всех предков u и v.

Предложение

Для любых вершин $u, v \in V$ их наименьший общий предок существует и единственный.

- Далее мы будем обозначать наименьшего общего предка вершин $u, v \in V$ как $LCA(u, v)$.
- Задача поиска наименьшего общего предка: для каждого запроса вида $(u, v) \in V^2$ найти $LCA(u, v)$.


$lca_T(a, b)$

Наивное решение

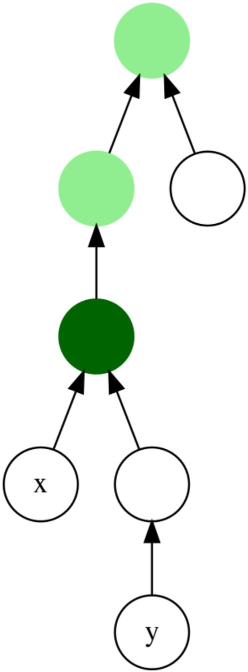
- Пусть $u, v \in V$, мы хотим найти $LCA(u, v)$.

Наивное решение

- Пусть $u, v \in V$, мы хотим найти $LCA(u, v)$.
- Будем идти от вершины u к корню, переходя по родителю и отмечая пройденные вершины.
- Запустим такой же процесс из вершины v. Первая отмеченная вершина на пути и будет являться $LCA(u, v)$.

Наивное решение

- Пусть $u, v \in V$, мы хотим найти $LCA(u, v)$.
- Будем идти от вершины u к корню, переходя по родителю и отмечая пройденные вершины.
- Запустим такой же процесс из вершины v. Первая отмеченная вершина на пути и будет являться $LCA(u, v)$.
- Сложность алгоритма – $\mathcal{O}(h)$ (в несбалансированных деревьях $h \sim n$).


x
y

Метод двоичного подъема

- Будем решать через динамику: $dp[v][i]$ – номер вершины, в которую мы придем, если будем идти из вершины v вверх по дереву 2^i шагов (если пришли в корень, то остаемся в нем).

Метод двоичного подъема

- Будем решать через динамику: $dp[v][i]$ – номер вершины, в которую мы придем, если будем идти из вершины v вверх по дереву 2^i шагов (если пришли в корень, то остаемся в нем).
- Пусть $depth[v]$ – глубина вершины v ($depth[v_0] = v_0$), $p[v]$ – номер родителя вершины v (для корня $p[v_0] = v_0$).
- $dp[v][i] = \begin{cases} p[v], & i = 0 \\ dp[dp[v][i-1]][i-1], & i > 0 \end{cases}$
- Нам нужно посчитать только $dp[v][i]$ для $i < \log_2 |V|$ ($2^i \geq n$ при $i \geq \log_2 |V|$, и $dp[v][i] = v_0$).

Метод двоичного подъема

- Будем решать через динамику: $dp[v][i]$ – номер вершины, в которую мы придем, если будем идти из вершины v вверх по дереву 2^i шагов (если пришли в корень, то остаемся в нем).
- Пусть $depth[v]$ – глубина вершины v ($depth[v_0] = v_0$), $p[v]$ – номер родителя вершины v (для корня $p[v_0] = v_0$).
- $dp[v][i] = \begin{cases} p[v], & i = 0 \\ dp[dp[v][i-1]][i-1], & i > 0 \end{cases}$
- Нам нужно посчитать только $dp[v][i]$ для $i < \log_2 |V|$ ($2^i \geq n$ при $i \geq \log_2 |V|$, и $dp[v][i] = v_0$).
- Всего значений dp – $\mathcal{O}(n \log n)$. Препроцессинг делается за $\mathcal{O}(n \log n)$.

```
function preprocess():
    int[] p = dfs(0)
    for i = 1 to n:
        dp[i][0] = p[i]
    for j = 1 to log(n):
        for i = 1 to n:
            dp[i][j] = dp[dp[i][j - 1]][j - 1]

int lca(int v, int u):
    if d[v] > d[u]:
        swap(v, u)
    for i = log(n) downto 0:
        if d[u] - d[v]:
            u = dp[u][i]
    if v == u:
        return v
    for i = log(n) downto 0:
        if dp[v][i] != dp[u][i]:
            v = dp[v][i]
            u = dp[u][i]
    return p[v]
```

Сведение к RMQ

- Задача минимума на отрезке (*RMQ, Range Minimum Query*): дан массив $a[1 \dots m]$; на запрос вида $(l, r) \in [1 \dots m]^2, l < r$ нужно научиться отвечать, чему равен $\min a[l \dots r]$.

Сведение к RMQ

- Задача минимума на отрезке (*RMQ, Range Minimum Query*): дан массив $a[1 \dots m]$; на запрос вида $(l, r) \in [1 \dots m]^2, l < r$ нужно научиться отвечать, чему равен $\min a[l \dots r]$.
- Запустим обход в глубинц из корня. Заведем список посещенных вершин a. Вершина добавляется в конец a при входе в нее и после возвращения из каждого из ее сыновей. Также будем сохранять индекс idx в a, по которому была записана очередная вершина в список при входе в нее.

Сведение к RMQ

- Задача минимума на отрезке (*RMQ, Range Minimum Query*): дан массив $a[1 \dots m]$; на запрос вида $(l, r) \in [1 \dots m]^2, l < r$ нужно научиться отвечать, чему равен $\min a[l \dots r]$.
- Запустим обход в глубинц из корня. Заведем список посещенных вершин a. Вершина добавляется в конец a при входе в нее и после возвращения из каждого из ее сыновей. Также будем сохранять индекс idx в a, по которому была записана очередная вершина в список при входе в нее.

Предложение

Для любых $u, v \in V$

$$LCA(u, v) = RMQ(idx[u], idx[v]),$$

где RMQ *возвращает элемент массива* a*, в котором достигается минимум* d *на отрезке* $pr_{depth}a[idx[u] \dots idx[v]]$ *(*$pr_{depth}a$ *– массив составленный из глубин соответствующих вершин массива* d*).*

Замечание

НУО, $idx[u] \leq idx[v]$*, так как* $LCA(u, v) = LCA(v, u)$*.*

Доказательство.

- Отрезок $a_1 = a[idx[u] \dots idx[v]]$ – путь в T из u в v $\implies a_1 \ni w = LCA(u, v)$ (в дереве между любыми двумя вершинами $\exists!$ простой путь) $\implies d \leq depth[w]$.
- Заметим, что в момент добавления $a[idx[u]]$, обход уже посещал поддерево с корнем W. В момент посещения $a[idx[v]]$ мы находимся в поддереве с корнем w. $\implies$ на отрезке a_1 мы находимся в поддереве с корнем w.
- $\implies$ на отрезке a_1 нет вершины $\neq w$ с глубиной $\leq depth[w]$ (так как такой вершины нет в поддереве с корнем w).

$\square$

Дерево отрезков

- Деревом отрезков (ДО, *segment tree*) называется бинарное дерево, которое хранит однородную информацию о подотрезках массива: если $a[1 \ldots m]$ – массив, то в корне ДО хранит информацию о $a[1 \ldots m]$, и если в вершине оно хранит информацию о $a[l \ldots r]$, то в левом ребенке оно хранит информацию о $a[l \ldots \frac{r-l}{2}]$, в правом – о $a[\frac{r-l}{2}+1, r]$ (в листах хранится информация об одноэлементных подотрезках $a[i]$).

```
function treeBuild(T a[], int i, int tl, int tr):  // [tl, tr)
    if tl == tr:
        return
    if tr - tl == 1:
        t[i] = a[tl]
    else:
        tm = (tl + tr) / 2
        treeBuild(a, 2 * i + 1, tl, tm)
        treeBuild(a, 2 * i + 2, tm, tr)
        t[i] = min(t[2 * i + 1], t[2 * i + 2])

int query(int node, int a, int b):
    l = tree[node].left
    r = tree[node].right
    if [l, r).intersect([a, b)): return infinity
    if [l, r) in [a, b): return tree[node].res
    return min(
        query(node * 2 + 1, a, b),
        query(node * 2 + 2, a, b))
```

- В списке а $n + const$ вершин, а значит в дереве отрезков не более чем $n + \frac{n}{2} + \ldots 1 + const \leq n + const = \mathcal{O}(n)$ вершин.
- Ответ на запрос – $\mathcal{O}(\log n)$