Алгоритмы и Структуры Данных. Лекция 9

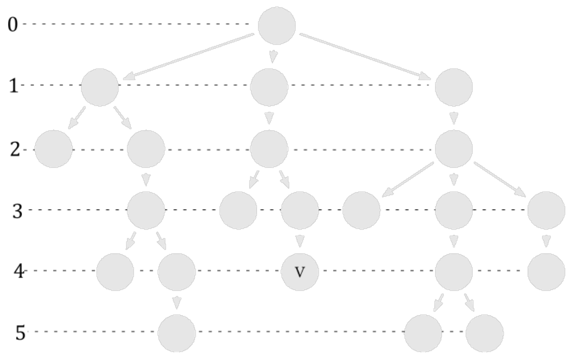
08.05.2024

_scarleteagle **imkochelorov**

LA

Level Ancestor

- `v k` — нахождение предка вершины `v` на уровне `k`



Двоичные подъёмы

Препроцессинг: $O(n \cdot \log n)$ *Запрос:* $O(\log n)$

`d[v]` — глубина вершины

`up[i][v]` — двоичные подъёмы

`dist = d[v] - k`

Поднимемся на `dist` вверх с помощью двоичных подъёмов (см. по ссылке: LCA)

Long Path Decomposition

Препроцессинг: $O(n)$ *Запрос:* $O(\sqrt{n})$

Для каждой вершины выберем самый длинный путь вниз из неё

`h[v]` — самый длинный путь вниз из `v`

`h[v] =` $\max\limits_{u \,\in\, \text{children}[v]}$ `h[u] + 1`

`pathId[v]` — номер пути, в котором находится `v`

`pathOrder[v]` — номер вершины `v` внутри её пути

`paths[i][j]` — `j`-ая вершина в `i`-ом пути

`u = paths[pathId[v]][0]` — первая вершина на пути из v

1) `d[u] > k` $\Rightarrow$ `v = p[u]` — `u` слишком глубокая, перейдём в предка `u`

2) `d[u] <= k` $\Rightarrow$ `ans = paths[pathId[v]][k - d[u]]` — `k` лежит на пути от `v` до `u`

В худшем случае нам придётся совершить следующую последовательность подъёмов:

$$1 + 2 + 3 + 4 + 5 + ... + t \leq n$$

$$t = O(\sqrt{n})$$

Ladder Decomposition

Препроцессинг: $O(n \cdot \log n)$ *Запрос:* $O(1)$

Увеличим пути вверх в 2 раза

$$l' - \text{изначальный путь}$$

$$l' = \frac{l}{2}, \quad l - \text{новый путь}$$

$$O(l_1 + l_2 + ... + l_t) = O(2 \cdot (l'_1 + l'_2 + ... + l'_t)) = O(n)$$

1) Делаем самый большой возможный прыжок из `v`

`up[i][v] = u =` $\lfloor \log_2($`dist`$) \rfloor$

Теперь вершина `u` и ответ лежат в одном удлинённом пути

2) В пути найдём искомую вершину по предподсчитанным путям

`x` — наивысшая вершина удлинённого пути

`k - d[x]` — номер искомой вершины в пути

Произведём не асимптотическую оптимизацию:

Алгоритм четырёх русских

Препроцессинг: $O(n)$ *Запрос:* $O(1)$

А давайте будем считать двоичные подъёмы только для листьев

Заметим, что все предки `v` также и предки всех потомков `v`

`leaf[v]` — *какой-то* лист в поддереве `v`

Считается тривиально, не ухудшая асимптотику

$$\texttt{c} \sim \log_2 n$$

Поддерево `v` — *маленькое*, если его размер $\leq$ `c`

Вырежем все маленькие поддеревья. На их месте оставим *особенную* вершину

В новом дереве останется $O(\frac{n}{c})$ листьев, где каждый лист — *особенная вершина*

Но теперь мы разучились отвечать на запросы из удалённых вершин :(

В каждом удалённом поддереве $\leq$ `c` вершин

Возможных запросов в удалённое поддерево $O(\texttt{c}^2)$

`ans[treeId][v][k]` — look-up таблица для вершин удалённых поддеревьев

$\sqrt{n}$ c c

Также нужно решить проблему того, что после удаления поддеревьев съезжают номера вершин в дереве

$O(\frac{n}{c} \cdot c) = O(n)$ — память на matching id вершин

Рассмотренные алгоритмы:

Алгоритм	Препроцессинг	Запрос
Двоичные подъёмы	$O(n \cdot \log n)$	$O(\log n)$
Long Path Decomposition	$O(n)$	$O(\sqrt{n})$
Ladder Decomposition	$O(n \cdot \log n)$	$O(1)$
Алгоритм четырёх русских	$O(n)$	$O(1)$