Студент: Тимур Хабибуллин
Группа: 1
Дата: 5 мая 2022 г.

---

1. <empty>

2. <empty>

3. Пусть корневое дерево T подвешено за вершину r, в массиве $\mathtt{T}[i]$ указаны предки для каждой вершины $i \in V$, $\mathtt{T}[r] = -1$. Чтобы поместить конень в другую вершину q, надо обратить порядок следования вершин в ветке $T(r,q)$, идущей вниз от r к q (на $T(r,q)$ у нас есть отношение порядка, определяемое глубиной вершины в ветке), при этом все вершины вне $T(r,q)$ сохранят своих предков, потому что на соответствующих ветках (поддеревьях) отношение порядка (глубина) не нарушается (т.е. если вершина v была ниже, чем u (она может быть и r или q), а так же $(u,v) \notin T(r,q)$, то после изменения корня v останется все также ниже u).
   Поэтому алгоритм следующий: для каждой вершины `cur` смотрим ее родителя `parent`, запоминаем его родителя `next_parent`, переходим к `parent`, ставим его родителем `cur`; в конце передвигаем `cur` в `parent`, а `parent` - в `new_parent`.

```
    cur = q
    parent = T[q]
    T[q] = -1
    while cur != r:
        next_parent = T[parent]
        T[parent] = cur
        cur = parent
        parent = next_parent
```

   Время работы $O(V)$, поскольку каждую вершину дерева пройдем не более 1 раза. Худший случай достигается, если наш граф - "бамбук подвещенный за один конец, а нам надо переподвесить его за противоположный.

4. При топологической сортировке получаем некоторую перестановку исходный вершин графа

$$(1, 2, ..., n) \to (p_1, p_2, .., p_n).$$

   Чтобы получить лексикографически минимальный топологический порядок, необходимо сделать так, чтобы в начале шли вершины с наименьшими номерами, а в конце - с наибольшими. Для этого сделаем топологическую сортировку следующим образом (алгоритм с лекции) - сначала посчитаем для всех вершин входящие степени $\mathtt{deg}_{in}(v)$ (пройдемся по списку смежности и если $v \in \mathtt{edges}(u)$, то $\mathtt{deg}_{in}(v) += 1$), найдем все истоки (их не менее одного, иначе в графе цикл и топологическая сортировка невозможна).
   Истоки должны идти в начале перестановки, поэтому поместим найденные истоки мин-кучу и возьмем исток с наименьшим номером, положим его в ответ, это p_1. Далее удалим p_1 из графа, т.е. из всех смежных с p_1 ребер вычтем единичку. И если у нас при вычитании получился ноль, то добавляем в кучу эту вершину. Далее вытаскиваем из кучи вершину с наименьшим номером и кладем в ответ, это p_2. Повторяем, пока куча не станет пустой. Получаем такой жадный алгоритм.
   Он работает корректно, потому что если рассмотреть оптимальный ответ и если первое отличие будет в каком-то p_i, то на i шаге мы в жадном алгоритме положили вершину, номер которой больше, чем в оптимальном. Но тогда это значит, что удалив из графа все предыдущие вершины

$p_1, ..., p_{i-1}$ мы или вытащили из минкучи не минимум, или мы не учли какой-то исток. Однако это невозможно по построению алгоритма, значит, наш жадный алгоритм должен не отличаться от оптимального.

5. Пусть дано дерево $T =< V, E >$. Так как в T нет циклов, то между двумя любыми вершинами $v, u \in T$ существует единственный путь. Тогда из этого получается, что любой путь в графе определяется начальной и конечной вершинами в пути.
Кроме этого, поскольку мы работаем с деревом, то любое ребро - есть мост, поэтому если возьмем какое-то ребро (u, v) и разобьем T на 2 подграфа (непересекающихся) - содержащий u (пусть A)и содержащий v (пусть B), то любой путь из вершины $x \in A$ в вершину $y \in b$ будет проходить через ребро (u, v). Но так как путь определяется своим началом и концом, то чтобы пересчитать все пути через ребро (u, v), надо взять любую вершину из A и любую вершину из B, таких вариантов ровно $|A| \cdot |B| = |A| \cdot (|T| - |A)$.
Тогда для того, чтобы пересчитать число путей через данное ребро (v, x), надо посчитать число вершин в поддереве одной из вершин ребра и по формуле выше вычислить результат. Чтобы это сделать можно модифицировать dfs: будем сохранять в массив `dp[v]` число вершин в поддереве v. Проинициализируем `dp[u]` $= 1$, $\forall u \in T$. Запустимся из v, дойдем до z, листа поддерева v. Поддерево образованное z, - сама вершина z. Вернувшись в родительскую вершину z, сложим число дочерних вершин `dp[z]` (как бы вершин в поддеревьях этих листов) и добавим в ячейку родительской вершины в `dp`, и тд. Дойдя до v, складываем число вершин `dp[u]` в поддерерьях дочерних вершин u (среди них не должно быть x) и добавляем в `dp[v]`. Тогда результат для ребра (v, x) есть `dp[v]` $* (|V| -$ `dp[v]`$)$. Замечу, что на каждом шаге рекурсии мы обновляем результат для соответсвующих ребер, как и для (v, x), поэтому мы за один запуск посчитаем все.
Время работы есть время работы обычного dfs, т.е. $O(V + E)$ (хотя у нас $E = V - 1$, так что, наверно, правильнее написать $O(V)$.