Пономаренко Алексей
AI Masters, 2023

Алгоритмы
Контрольная работа №2

№2

• Алгоритм: Обходим дерево рекурсивно. Введем вспомог. рекурс. ф-ю func, кот-я будет возвращать флаг – все ли элементы поддерева равны и k – число вершин в этом поддереве. Также заведем 2 глоб. перем.: kmax = 0, node = null

Если мы оказались в листе, то возвращаем (True, 1), а в глоб. перем. kmax = 1 (если было 0) и node = указатель на текущ. узел (лист)

Если мы оказались не в листе, то смотрим на флаги: если хотя бы у 1 из них false, то возвращаем (false, 0). Если они оба true, тогда сравниваем число в данном узле со знач-ями в левом и правом ребенке:

– Если они все равны, то проверяем: если k1+k2+1 > > kmax, то kmax = k1+k2+1 (k1 и k2 – вернувшиеся знач-я у детей), node = указатель на тек. узел. Возвращаем (true, kmax)

– Иначе возвращаем (false, 0).

• Сложность: итого, мы ровно 1 раз посетим каждый узел дерева, а значит наш алгоритм линеен.

• Корректность: так как нам необх., чтобы всё поддерево целиком равнялось одному числу, мы возвращаем false сразу при любом несовпадении. Также на каждом шаге рекурсии при полном исходе мы обновляем наши глоб. перем., благодаря чему мы имеем ответ на каждом этапе алгоритма.

P.S. Итоговым шагом алгоритма будет запуск func от левого и правого поддерева, а затем проверка обоих флагов на true. Вернем node.

№3.

• Алгоритм: заметим, что нам на вход подаётся корневое дерево, а значит $|E| = O(|V|)$.

Запустим обход в глубину из корня дерева, параллельно формируя массивы s и f открытия и закрытия каждой вершины.

Далее для каждого из m запросов проверяем: если $s[u] < s[v]$ и $f[u] > f[v]$, то вершина u является предком v, иначе – нет.

• Сложность: как мы заметили, $|E| = O(|V|) \Rightarrow$ сложность обхода дерева в глубину $= O(n)$. Далее просто m раз делаем проверки за $O(1)$. Итого, $O(m+n)$ по времени и $O(n)$ по памяти

• Корректность: корректность следует из леммы о скобках

№4.

• Алгоритм: построим на данном графе максимальное остовное дерево. Все (отрицательные) рёбра, не вошедшие в это остовное дерево и будут формировать мн-во F.

• Сложность: воспользуемся, например, алгоритмом Крускала построения остовного дерева, использующего кучу. Тогда, сложность построения равна $O((|V|+|E|)\cdot \log|V|)$

• Корректность: по сути следует из опр-я максимального остовного дерева – это дерево на (всех) вершинах V, чья рёберная сумма максимальна. Это значит, что рёберная сумма оставшихся рёбер минимальна. Также мы не берём в мн-во полож. рёбра, т.к. они только увеличивают рёберную сумму.

№1.

• Алгоритм: запускаем поиск в глубину из вершины s с небольшим дополнением: на пути выполняем проверку на a-b-c: если мы пришли в вершину по ребру, то можем выйти только по b, если по b - то по c, если по c - то по a.

• Сложность: сложность обхода в глубину равна $O(|V|+|E|)$

• Корректность: обход в глубину позволит нам найти циклы. путь из s в t, а модификация позволит делать проверку на a-b-c путь на каждом шаге dfs.

№6

1) Мы всегда можем запустить такую посл-ть операций: 2, 3, 4, ..., n. Тогда, он сначала поместит на своё место 1, потом 2 и т.д.

№1.

• Алгоритм: запускаем поиск в ширину из вершины s (алгоритм Дейкстры), немного модифицировав её. Для каждой вершины будем сохранять букву, по кот-й мы в неё пришли: a → b, b → c, c → a. То есть нам необх. двигаться только по тем ребрам, кот-е удовл. a-b-c пути.

• Сложность: сложность Дейкстры на куче = $O((|V|+|E|)\cdot\log|V|)$

• Корректность: алгоритм Дейкстры находит кратчайшие пути до всех вершин, в т.ч. и до t. Доп. ус-е на хранение буквы, по кот-й мы вошли в верш., гарантирует a-b-c критерий.

№6.

2) Найти число операций, необх. для этого – это число инверсий в массиве a

3) Для получения напур. перестановки, нам нужно убрать все инверсии у a. Это можно сделать за не менее, чем кол-во инверсий ру. Это факт из лекции.

СПАСИБО ЗА КУРС! ♡