Алгоритмы и структуры данных. Домашнее задание №4

Выполнил студент Наседкин Дмитрий Сергеевич (группа 242)

Письменная часть

№ 1

```
write_keys(node* root) {
    v = root
    returnFromLeftChild = False
    returnFromRightChild = False
    while True:
        # Если вернулись из правого сына
        if returnFromRightChild:
            # то поднимемся до тех пор пока не дойдем до другого поворота
            while True:
                if v == root:
                    return
                if v.parent.right == v:
                    v = v.parent
                else:
                    v = v.parent
                    returnFromRightChild = False
                    break;
            continue
        # Если вернулись из левого сына
        if returnFromLeftChild:
            print(key)
            # то нужно сходить в правого
            if v.right == null:
                returnFromRightChild = True
                continue
            else:
                v = v.right
                returnFromLeftChild = False
        while v.left != null:
            v = v.left
        returnFromLeftChild = True
}
```

№ 2

Подготовка и предподсчет

Первым делом, для каждой вершины ее список смежности отсортируем по номерам ребер.

Теперь посчитаем $dp[v]$ - количество путей из v в n. Для этого просто будем рассматривать вершины в порядке топ. сорта, и формула пересчета очевидна: $dp[v] = \sum_{(v,u)\in E} dp[u]$. Теперь на-

считаем $pref[v][i] = \sum_{i=1,\ (v,u)\in E}^{d_+(v)} dp[u]$, то есть префиксные суммы по дпшкам выходящих вершин отсортированных по возрастанию по номерам ребер(преф сумму будем насчитывать через предыдущие + значение очередной дпшки, за линию).

Насчитаем теперь время:

1. $O(n+m) = O(m)$ - на топ. сорт + насчитывание дпшки и преф. сумм
2. $O(m \log m) = O(m \log C)$, так как $m \le C + n$, значит $O(m-n) = O(m) = O(C)$

Итого $O(m \log m)$ на предподсчет.

Ответы на запросы

Пункт (а)

Пусть мы стоим в v вершине (если $v = n$ то мы нашли нужную длину) и текущая длина равна L ($v = 1, L = 0$ в начале)

Для этого для просто по массиву преф. сумм найдем первый $i \mid pref[v][i] \ge k$, то есть по всем прошлым вершинам суммарно путей было меньше k, и если пройти через данную, то k-ый путь найдется, следовательно нужно $v := u$ (она i-ая по счету), $k := k - pref[v][i-1]$ и $L := L + 1$. После чего начем шаг заново.

Понятно, что в худшем случае мы пройдемся по всем ребрам, то есть асимптотика равна $O(m)$.

Пункт (б)

Давайте искать первый $i \mid pref[v][i] \ge k$ не перебором i, а с помощью бин. поиска. Докажем, что теперь алгоритм работает за $O(\log C)$:

Очевидно, что мы запустим бин. поиск в худшем случае от всех вершин кроме n-ой, то есть алгоритм будет работать за $O(\log d_1 + \log d_2 + \cdots + \log d_{n-1}) = O(\log \prod_{i=1}^{n-1} d_i) = O(\log C)$, где d_i - степень i-ой вершины.

Устная и письменная часть

№ 3

Пункт а

Давайте для каждого из m отрезков найдем первый момент, когда он будет иметь хотя бы одну покрашенную клетку. Для этого построим массив a[i] - номер j | p[j] = i, то есть время когда ячейка i покрасилась в черный. Тогда раз мы знаем p заранее, то заполним наш массив a. Тогда, можно чтобы для отрезка i узнать номер момента когда он будет иметь хотя бы одну черную клетку, это минимум на отрезке от [l, r]. Это можно сделать с помощью ДО. ($O(n)$ на построение и $O(m \log n)$ на все запросы) В какой-то дополнительный массив extra добавим еденичку к этому времени, то есть extra[min_t]++.

Пойдем по массиву extra слева направо поддерживая текущую сумму на префиксе, тогда количество отрезков из m у которых к этому перекрашиванию хотя бы одна клетка черная, равна как раз этой сумме.

№ 4

Давайте для начала научимся находить LA с помощью HLD за $O(n)$ на предподсчет, и $O(\log n)$ на запрос:

Для этого построим HLD-декомпозицию на нашем дереве за $O(n)$, то есть для каждой вершины проведем тяжелое ребро в сына с наибольшим поддеревом. После чего для каждой вершины запомним в каком тяжелом пути она лежит, а также первую вершину в этом пути(предков и высоты насчитаем в начале с помощью dfs). И для каждого тяжелого пути все вершины в нем положим в вектор с номером этого пути по порядку спуску вершин. Очевидно, что это работает за $O(n)$ времени и памяти.

Мы знаем, что на пути от v от корня не более $O(\log n)$ тяжелых путей, тогда отсюда очевидно как получить LA(v, k): если наш k-ый предок в том же пути что и наша вершина, то просто возьем нужную вершину в векторе для этого пути(мы знаем высоты для первой вершины и нашей, тогда и k-ую тоже легко вычислить), иначе передем в предка от "головы"нашего тяжелого пути(отняв от k расстояния в высотах между v и головой) и повторим тот же шаг. Так как путей $O(\log n)$, то и ответ на запрос работает за $O(\log n)$.

Теперь ускорим наше решение:

1. Обозначим каждый тяжелый путь вершиной, тогда все эти пути представляют из себя дерево(у каждого тяжелого пути есть предок - прошлый для него тяжелый путь, кроме первого). Высота дерева из соображений выше - $O(\log n)$

2. Давайте на этом дереве насчитаем бинарные подъемы, так как высота нашего дерева $O(\log n)$ и вершин точно не больше n, то это будет работать за $O(n \log \log n)$.

3. Теперь научимся отвечать на запросы, сначала получим номер пути(вершину в новом дереве) и от нее с помощью бин. подъемов найдем первую вершину, что в тяжелом пути для него содержится нужная нам высота(мы знаем отрезок высот для каждого тяжелого пути, так как есть первая и последняя вершина в вектора для этого пути), и после того как нашли легко найти нужную нам LA(v, k) (уже описывали выше как). Бин. подъемы работают за $O(\log h)$ - где h - высота дерева, тогда в нашем случае получим $O(\log \log n)$.

№ 5

Предподсчет

Давайте для дерева отрезков с номером L считать, что она отвечает за префикс длины L. На нем построим ДО, где вершина хранит количество элементов со значением от [l, r]. (То есть для листовых вершин равно числу элементов со значением x). Так как ДО L и ДО $L+1$ отличаются только одним добавлением в точке(а именно нужно сделать tree[L + 1] = *tree[L], add(T[L + 1], p[L + 1], 1)), то реализуем персистентное ДО. Очевидно, что предподсчет работает за $O(n \log n)$ времени и памяти.

Запросы

1. запрос 1-ого типа (кол-во элементов на отрезке от i до j со значением от x до y):

Если бы у нас было ДО, но для нужного отрезка, а не префикса, то наш запрос - это просто сумма на отрезке [l, r]. Как же получить нужный отрезок для ДО?
Чтобы получить вершину в ДО, которая хранит количество элементов со значением от [l, r] для отрезка [i, j], то просто получим разность значений в вершинах j и $(i-1)$ (получить отрезок можно из двух префиксов). Значит мы умеем отвечать на данный запрос за время $O(\log n)$ - время спуска по ДО.

2. Запрос 2-ого типа (реверс двух соседних в перестановке, меняются i и i + 1)

Заметим, что после реверса, для ДОшек до i позиции префикс не менялся, а после $i+1$ тоже не поменялся(так как мы строим на значениях), то есть изменилось только ДО для i-ой позиции, а именно вместо $p[i]$ нужно записать $p[i+1]$, для этого сделаем 2 запроса add в эту ДОшку(один с -1 для индекса p[i], другой с +1 для индекса p[i + 1]). (Сделаем точно также, сначала перекопируем

старые вершины, чтобы те ДОшки, что ссылались на них не изменились), после чего уже можно менять значения. Значит мы умеем отвечать на данный запрос за время $O(\log n)$ - время спуска по ДО.