Домашняя работа №3

Шишацкий Михаил, 932

08.04.2020

Задача 5. Чтобы найти простой путь с максимальной суммой, можно написать DFS, возвращающий максимальную возможную сумму на простом пути, заканчивающимся ребром в вершину, из которой вызван DFS, и заканчивающимся где-то в её поддереве, а затем вызвать эту функцию в корне. Каждая вершина представлена структурой Node, содержащей поле edge - длину ребра, ведущего в эту вершину "сверху"(для корня положим это значение равным нулю). Также структура будет содержать массив Node* указателей на своих сыновей, названный child. Также функция будет обновлять некую глобальную переменную с максимальной суммой max_sum (инициализированную -INF), содержащую ответ на вопрос.

Приведём псевдокод функции DFS:

```
int DFS(Node* v) {
    int fst_max = -INF;
    int snd_max = -INF;

    int c_way = 0;
    for (c : v->child) {
        c_way = DFS(c);
        if (c_way > fst_max) {
            snd_max = fst_max;
            fst_max = c_way;
        } else if (c_way > snd_max) {
            snd_max = c_way;
        }
    }

    int best_transit_way = fst_max + snd_max;   //;     *
                                                //;    / \

    int best_subtree_way_including_this = max({best_transit_way,
                                                fst_max, 0});
    \\;     *     or     *      or      *
    \\;                   \            / \
```

```
    int best_way_up = max({fst_max, 0}) + v->edge;

    \\;     |                  |
    \\;     *       or         *
    \\;                         \

    max_sum = max({max_sum, best_subtree_way_including_this});

    return best_way_up;
}
```

После вызова DFS в переменной max_sum будет лежать ответ на задачу, т.к. для каждой вершины были учтены все возможные варианты прохождения искомого пути через неё. DFS отработает за O(n), т.к. побывает в каждой вершине ровно один раз, при этом, каждая вершина обрабатывается за O(1) [вызов из верхнего цикла + арифметические операции и выбор максимума].

Чтобы восстановить искомый простой путь, можно проиндексировать вершины по порядку выхода из функции DFS и в момент выхода запоминать в массиве, было ли обновлено значение максимума, первую наилучшую ветку и вторую. Когда выполнение функции DFS закончится, пройдёмся справа (самая правая ячейка - корень) по массиву. Если в текущей ячейке было обновлено значение максимума, значит, наилучший путь проходит через неё и 0-2 максимальные ветки этой вершины (это также можно запоминать в массиве). Добавим эту вершину к пути и вызовем нашу процедуру от максимальных веток в случае, если максимальный путь достигается с ними (или выберем лишь первую максимальную, т.к. мы пришли сверху, несмотря на то, что в этой вершине лучший путь включал две ветки). Если же значение максимума обновлено не было, путь целиком содержится в каком-то из поддеревьев: пройдёмся по их корням (несколько ячеек левее текущей) справа налево и вызовем процедуру (где также сохранится возможность выбора двух веток а не одной, несмотря на то, что вызов произведен сверху) от первой справа, где было обновлено значение максимума. Вызов процедуры от других веток означает рекурсивный вызов от некоторых ячеек массива, которые обязательно расположены левее, и участки массива, задающие их поддеревья не пересекаются. Таким образом мы побываем не более чем в n вершинах: из каждой вершины на более высоком уровне мы перейдем не более чем к двум сыновьям.

Задача 6а. Идея с ДД так и не добилась :C
(а кажется, она масштабировалась до 6б)

В начале создадим массив сетов размера 1, каждый из которых содержит соответствующий элемент в начальном множестве.

Создадим произвольное дерево поиска по ключу - значение элемента. Дополнительной нагрузкой к ключу будет указатель на сет, в котором этот ключ лежит (как вариант - номер ячейки в массиве сетов). Но не будем строить дерево по массиву сетов сразу, т.к. это обойдётся дорого. До тех пор, пока мы не обратились к слиянию элемента с каким-то другим множеством, мы знаем, в каком сете он лежит.

Слияние множеств будем выполнять следующим образом: обратимся сначала к нашему дереву поиска по ключам обоих элементов, по которым произведён запрос. Если кого-то из них нет в дереве, возьмем множество, в котором он лежит по индексу в массиве. Сольём два сета путём извлечения элемента из меньшего сета, добавим этот элемент с указателем на сет, в который мы его добавим, в дерево поиска, а потом вставим этот элемент в больший сет. В конце можно удалить меньший сет. Заметим, что каждый раз элемент меньшего (или равного, если нельзя выбрать) множества попадает в множество, размер которого как минимум в 2 раза больше размера старого множества. Значит, каждый элемент будет переброшен не более $log(N)$ раз, а следовательно, для k слияний будет произведено не более $k \cdot log(N)$ перебрасываний. Поэтому на одно слияние придётся в среднем $log(N)$ перебрасываний, каждое из которых производится за $O(log(N))$. Поэтому амортизированная сложность одного слияния - $O(log^2(N))$. (поиск в начале - о-малое от сложности самого слияния)

Искать последователя будем следующим образом: по элементу найдем в дереве поиска сет, в котором он содержится. Затем вызовем метод lower_bound. Асимптотика обеих операций - $O(log(N))$.

Таким образом, мы можем выполнить q запросов за $O(q \cdot log^2(N))$ в среднем.