Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Алгоритмам и структурам данных»

Выполнил:

Студент группы P3200

Шишкин Н.Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2019

Задача №1494 «Монобильярд»

Пояснение к примененному алгоритму:

Будем по порядку симулировать процесс доставания шаров, предполагая, что жульничества не было. В таком случае шар с номером i мы могли достать либо сразу, после того как его положили, либо взяв перед этим все шары над ним.

Начинаем по порядку класть шары, начиная с первого, также начинаем отслеживать на каком шаре в порядке их вытаскивания мы находимся. На каждом шаге после того, как положили шар, проверяем сколько мы можем достать, т.е. если положили шар x и в массиве извлечения шаров мы тоже находимся на элементе x, мы его вынимаем и продвигаемся в массиве извлечения. Повторяем эту процедуру, пока можем, затем повторяем с начала.

В конце, если не было жульничества и мы могли достать все шары, то стек шаров будет пустым, иначе - жульничество было.

Очевидно, что такой процесс вынимания шаров является единственно возможным, поскольку если у нас была возможность достать шар и мы этого не сделали - мы уже никогда не достанем этот шар, т.к. над ним уже лежат другие, а брали мы строго в каком то порядке. Поскольку при не жульничестве порядок, в котором Чичиков кладет шары строго определен, также как и порядок их изъятия, то и вариант развития событий был только один, как раз тот, который мы и симулируем.

Сложность алгоритма - O(n)

Задача №1067 «Disk Tree»

Пояснение к примененному алгоритму:

Строим дерево каталогов, в дереве, в качестве путей между узлами выступают названия каталогов, для каждой вершины храним массив каталогов, по которым можем из неё перейти. Считываем путь, разбиваем его последовательно на части по “\” и начинаем вставлять в дерево. Изначально вставляем в корень (пути абсолютные) первую часть, получаем некий узел (существующий или только что созданный), к нему вставляем следующую часть пути и т.д. пока путь не закончится. Повторяем эту процедуру для каждого пути.

Чтобы поддерживать лексиграфический порядок, будем хранить массив путей в отсортированном виде, тогда поиск в нем будет занимать log(m) (m - число путей) и вставка столько же. Для ускорения проверки на совпадение, для каждого пути будем также хранить его хэш.

Чтобы напечатать дерево просто обойдем его в глубину по потомкам в отсортированном порядке.

Сложность алгоритма - O(n*log(n)).

Задача №1521 «Военные учения 2»

Пояснение к примененному алгоритму:

Строим дерево отрезков для массива солдат, для каждого отрезка будем хранить число ещё не вышедших солдат в нем. Высота этого дерева - log(n)

В цикле начинаем по одному выбирать солдат, на каждом этапе мы удаляем солдата из круга, соответственно модифицируя дерево отрезков. Операция удаления будет занимать

log(n). После удаления вычисляем какой солдат будет следующим. Допустим, что всего солдат n, и в строе мы удалили i-го солдата, тогда наш строй разбит на две части - от 1-го до i-го, и от i-го до n. Нам нужно переместиться на k солдат вперед, т.к. массив цикличный (солдаты стоят в кругу) и мы знаем количество оставшихся солдат от 1-го до i-го, мы сложим это число с k и возьмем по модулю числа оставшихся солдат (на случай, если эта сумма больше количества оставшихся солдат). Данная сумма (назовем её u) говорит о том, сколько оставшихся солдат начиная с 1-го нам надо отсчитать, чтобы получить нового выбывшего.

В дереве отрезков находим наименьший по размеру префикс массива, в котором содержится u не выбивших солдат, его правый индекс и будет искомым солдатом, это также произойдет за log(n).

Заметим, что во время процесса нахождения префикса, мы можем одновременно производить операцию удаления искомого солдата (чтобы не пробегать одни и те же узлы дерева, только с разной целью).

Проделав процедуру n раз получим ответ.

Сложность алгоритма - O(n*log(n)).

//Дополненная часть.

Задача №1628 «Белые полосы»

Пояснение к примененному алгоритму:

Будем считать белой полосой промежуток между черными днями, изначально окружим каждую неделю фиктивным черным днем до начала и после её конца. В таком случае наш подсчёт корректен с самого начала. Номера черных дней будем хранить в структуре set (он поддерживается отсортированным). Считываем черные дни, последовательно вставляя их номера в наш set. После этого подсчитаем горизонтальные полосы, для этого проходимся по set и считаем количество белых отрезков (расстояние между двумя соседними черными днями больше 1). Для подсчёта вертикальных полос повторяем те же самые действия, только если попались два черных дня с расстоянием 1, то необходимо проверить условие максимальности по включению. Для этого смотрим на соседей этого дня в горизонтальном представлении, если они оба черные - включаем отрезок в ответ, иначе - нет.

Сложность алгоритма - O(s*log(s)), где s = n + m + k.

Задача №1650 «Миллиардеры»

Пояснение к примененному алгоритму:

Для хранения суммы сбережений в городе будем использовать двоичную кучу. Тогда город с максимальной суммой сбережений всегда будет в вершине дерева, и мы можем за O(1) получать его. В куче будем хранить пару - суммарные сбережения (ключ) и имя города.

Также будем хранить две хэш-мапы.

В одной по ключу имени миллионера мы храним его сбережения и текущий город, в котором он находится.

В другой храним по ключу имени города номер узла в куче, который в данный момент соответствует этому городу, и количество дней, когда этот город лидировал.

При считывании информации о миллиардерах заполняем информацию о них в хэш-мапе, а также информацию о городах в хэш-мапе и куче.

Для каждого дня мы делаем следующее - берем из кучи максимум (предварительно сравнив его с детьми) и если нет ничьей увеличиваем ответ для соответствующего города. Дальше проверяем, есть ли перемещения в этот день, если нет - итерируемся дальше. Но если такие дни есть, для каждого перемещения проделываем следующую процедуру - уменьшаем значение ключа в куче для текущего города миллиардера (информацию о нем храним в хэш-мапе), и увеличиваем для того, в который он приехал. При этом поддерживая актуальной информацию в обеих хэш-мапах.

В конце выводим отсортированное содержимое хэш-мапы с городами, там и хранится ответ для каждого города.

Время работы - O((n+k)*log(n+k)).