ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

« Maximum Spending After Buying Items»

Выполнил работу

Макаров Эльдар

Академическая группа J3114

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

1. Введение

Цель работы: изучение и реализация алгоритма максимизации трат за фиксированное количество дней, используя жадный подход для выбора наилучшего доступного значения.

Задачи:

Изучить теоретические аспекты алгоритма.

Реализовать алгоритм на языке C++.

Использовать жадный подход для эффективного выбора максимального значения.

Проанализировать производительность программы.

Оценить временную и пространственную сложность.

Сравнить экспериментальные результаты с теоретическими оценками.

1. Теоретическая подготовка

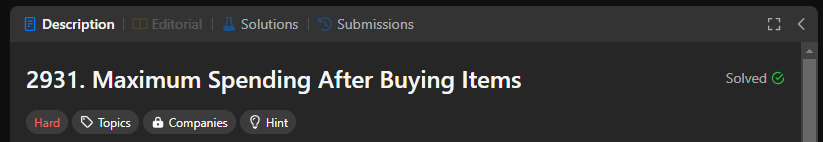
Жадный алгоритм — это класс алгоритмов, которые на каждом шаге делают локально оптимальный выбор, полагая, что это приведёт к глобально оптимальному решению в конце.

Сама задача состоит в нахождении максимальной суммы затрат.

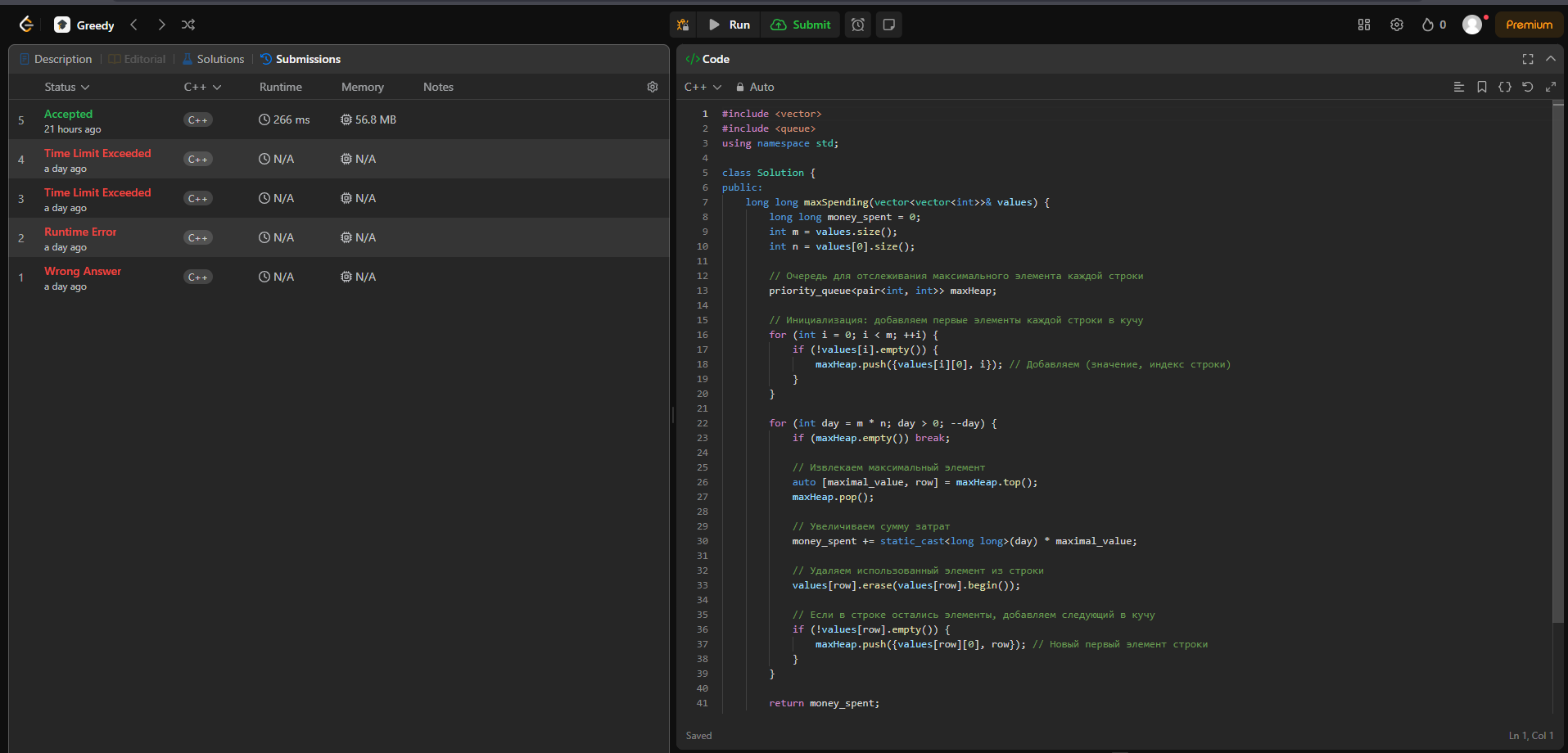
Используемые типы данных:

Вектор (vector): хранит строки с элементами. Элементы упорядочены по убыванию.

Приоритетная очередь (priority\_queue): позволяет эффективно выбирать максимальный элемент.  
Задача:



Доказательство выполнения:



1. Реализация

Этап 1: Анализ задачи

Задача заключается в максимизации общей стоимости товаров. Каждая строка содержит элементы, упорядоченные по убыванию. На каждом шаге необходимо выбрать самый большой доступный элемент, умножить его на номер текущего дня и добавить к общей сумме.

Первоначально, мой алгоритм начинал свою работу от самых маленьких чисел, из-за того, что счетчик дней идет от меньшего к большему. Однако в будущем я изменил подход, из-за того, что считать наибольшее мне показалось более удобным. Далее я подумал о том, чтобы на каждом шаге перебирать все строки и находить максимальный элемент среди них. Однако такой код оказался слишком медленным.

После долгого обдумывания задачи я понял, что эффективным решением будет использование жадного подхода.

Этап 2: Разработка алгоритма

Изначально я думал о том, чтобы на каждом шаге перебирать все строки и находить максимальный элемент среди них. Однако такой код оказался слишком медленным.

Для решения проблемы оптимизации я изучил соответствующую информацию и решил использовать приоритетную очередь. Она весьма эффективна и позволяет извлекать текущий максимум и добавлять новые элементы за логарифмическое время.

Мой алгоритм выглядит следующим образом:

Инициализировать приоритетную очередь, добавив в нее первые элементы всех строк.

На каждом шаге извлекать максимум из очереди, обновлять общую сумму затрат и удалять использованный элемент из строки.

Если в строке остаются элементы, добавлять следующий максимальный элемент в очередь.

Повторять до того, как закончатся все значения.

Этап 3: Написание кода

#include <vector>

#include <queue>

using namespace std;

class Solution {

public:

    long long maxSpending(vector<vector<int>>& values) {

        long long money\_spent = 0;

        int m = values.size();

        int n = values[0].size();

        // Очередь для отслеживания максимального элемента каждой строки

        priority\_queue<pair<int, int>> maxHeap;

        // Инициализация: добавляем первые элементы каждой строки в кучу

        for (int i = 0; i < m; ++i) {

            if (!values[i].empty()) {

                maxHeap.push({values[i][0], i}); // Добавляем (значение, индекс строки)

            }

        }

        for (int day = m \* n; day > 0; --day) {

            if (maxHeap.empty()) break;

            // Извлекаем максимальный элемент

            auto [maximal\_value, row] = maxHeap.top();

            maxHeap.pop();

            // Увеличиваем сумму затрат

            money\_spent += static\_cast<long long>(day) \* maximal\_value;

            // Удаляем использованный элемент из строки

            values[row].erase(values[row].begin());

            // Если в строке остались элементы, добавляем следующий в кучу

            if (!values[row].empty()) {

                maxHeap.push({values[row][0], row}); // Новый первый элемент строки

            }

        }

        return money\_spent;

    }

};

1. Экспериментальная часть

Подсчет асимптотики:

Инициализация очереди: O(m log m), где m — число строк.

Основной цикл: O((m \* n) log m), так как на каждом шаге происходит извлечение и добавление в очередь.

Результаты работы кода на случайных строках:

|  |  |  |  |
| --- | --- | --- | --- |
| Количество строк | Количество значений в строке | Время (мс) | Память (KB) |
| 5 | 5 | 0.0349 | 0.253906 |
| 5 | 50 | 0.2646 | 1.13281 |
| 10 | 5 | 0.0638 | 0.507812 |
| 10 | 50 | 0.5618 | 2.26562 |
| 10 | 100 | 1.0635 | 4.21875 |
| 50 | 100 | 5.7539 | 21.0938 |
| 50 | 200 | 11.8676 | 40.625 |
| 50 | 400 | 24.3314 | 79.6875 |
| 100 | 400 | 47.8703 | 159.375 |
| 100 | 600 | 72.8883 | 237.5 |
| 100 | 1000 | 143.351 | 393.75 |

1. Заключение

В ходе выполнения работы мною был реализован жадный алгоритм для задачи максимизации трат. Цель работы была достигнута благодаря жадному алгоритму, он был необходим из-за того, что на каждом шаге код выбирает локально лучший вариант, а такой подход гарантирует правильный результат.

Алгоритм был протестирован на различных входных данных, включая случайно сгенерированные матрицы разных размеров. Результаты экспериментов показали соответствие теоретическим оценкам сложности алгоритма и подтвердили его корректность.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти и ускорение работы алгоритма.