ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Задача 312. Burst Balloons - LeetCode»

Выполнил работу

Смирнов Александр

Академическая группа №J3111

Принято

Ментор Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель работы: попрактиковаться в решении задач на метод ДП, решить одну из hard задач на LeetCode.

Задачи:

1. Выбрать задачу для решения;
2. Разобраться в условии, придумать применение метода ДП;
3. Реализовать алгоритм;
4. Провести анализ полученных результатов и оформить отчёт.

2. Теоретическая подготовка

В решении я использовал векторы и int-ы.

Использовал метод ДП, на который не многозначительно намекает тематика лабы и тэг задачи.

3. Реализация

Условие задачи следующее:

Вам дается n шариков с индексами от 0 до n - 1. На каждом шарике нанесен номер, представленный массивом nums. Вам предлагается лопнуть все шарики.

Если вы лопнете i-й шарик, то получите nums[i - 1] * nums[i] * nums[i + 1] монет. Если значение i - 1 или i + 1 выходит за пределы массива, то считайте, что это воздушный шар с нарисованной на нем цифрой 1.

Верните максимальное количество монет, которые вы сможете собрать, разумно лопнув воздушные шары.

Для её решения я использовал библиотеки iostream, time, algorithm и vector.

Решение задачи представлено на рис. 1:

C++ Auto

```cpp
class Solution {
public:
    int maxCoins(vector<int>& input_array) {
        // Задаем массив с номерами шариков, вместе с nums[-1] и nums[n]
        int num_ballons = input_array.size() + 2; // 4 байта
        // Матрица, в которой будем хранить промежуточные суммы выигранных монет (ДП)
        vector<vector<int>> dp(num_ballons, vector<int>(num_ballons)); // 24 + (n + 2)^2 байта

        // Добавляем в начало и конец массива единицы (крайние шарики nums[-1] и nums[n])
        input_array.insert(input_array.begin(), 1);
        input_array.insert(input_array.end(), 1);

        for(int len = 2; len <= num_ballons; len++) { // O(n) - проходимся по всем шарикам со второго правой границей (длина выбранного подмассива)
            for(int i = 0; i <= num_ballons - len; i++) { // O(n) - проходимся по всем шарикам в подмассиве левой границей
                int j = i + len - 1; // 4 байта, фиксируем правую границу
                // Выбираем, какой шарик выгоднее всего лопнуть в выбранном подмассиве
                for(int k = i + 1; k < j; k++) { // O(n) - проходимся по подмассиву
                    dp[i][j] = max(dp[i][j], dp[i][k] + dp[k][j] + input_array[i] * input_array[k] * input_array[j]);
                    // Запиываем в матрицу ДП максимальную награду, которую можно получить, лопнув шарик в этом подмассиве, считаем по формуле из условия задачи
                }
            }
        }
        return dp[0][num_ballons - 1]; // Выводим ответ
    }
};
```

Saved Ln 12, Col 9

Рисунок 1. Код решения задачи

Концептуально решение следующее: сперва я дополнил массив единицами в начале и в конце (чтобы было возможно лопнуть граничные шарики), затем составил матрицу dp размера (n + 2)^2, в которой будут хранится максимально возможное количества монет, которые можно получить, лопнув шарик в подмассиве (i, j). Далее в циклах перемещаю

левую и правую границы рассматриваемого подмассива и по формуле вычисляю возможное количество монет, которое можно получить.

4. Экспериментальная часть

Оценочная сложность решения – n^3, так как используется три вложенных цикла, каждый из которых совершает в худшем случае n итераций. Такую же сложность дает решению и ЛитКод, при этом скорость довольно высокая относительно других решений (представлено на рис.2)

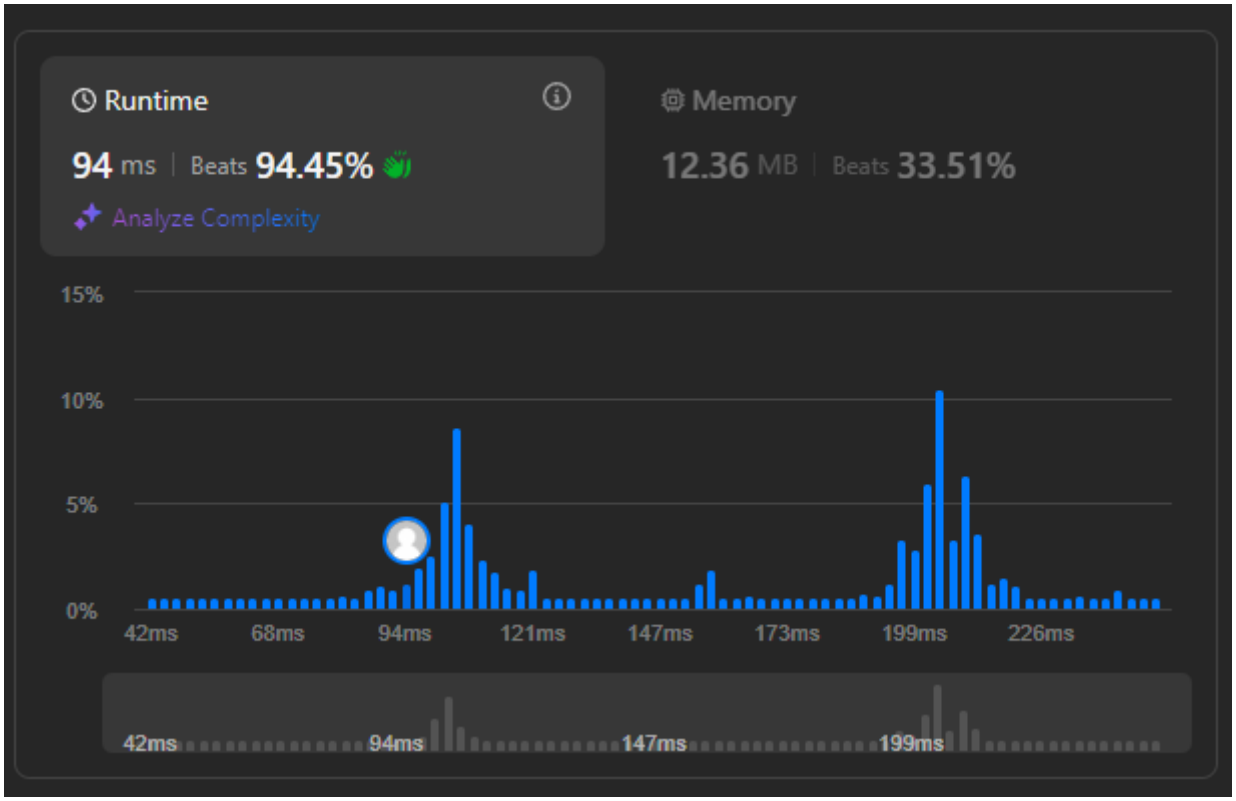


Рисунок 2. Скорость решения в сравнении с другими решениями этой задачи

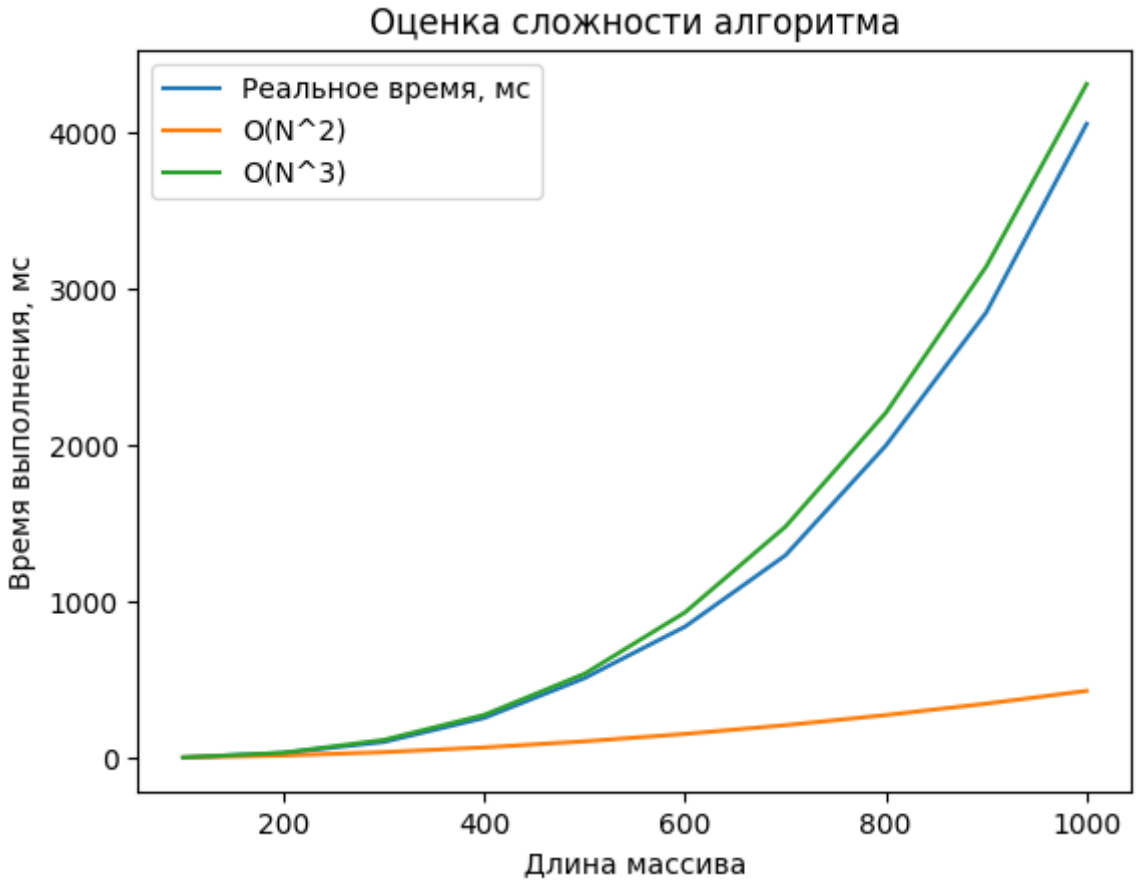
Почти вся память уходит на хранение матрицы dp, она занимает 24 + (n + 2)^2 байт. На вектор входных значений передается ссылка, так что функция не тратит на его хранение дополнительную память.

Проверим теоретическую сложность алгоритма, сравнив с O(N^2) и O(N^3) (таблица 1)

N	100.000000	200.000000	300.000000	400.000000	500.000000	600.000000	700.000000	800.000000	900.000000	1000.000000
Реальное время, мс	4.300000	33.000000	104.000000	258.000000	513.000000	842.000000	1298.000000	2000.000000	2852.000000	4058.000000
O(N^2)	4.386430	17.372430	38.958430	69.144430	107.930430	155.316430	211.302430	275.888430	349.074430	430.860430
O(N^3)	4.430294	34.918584	117.264874	277.269164	540.731454	933.451744	1481.230034	2209.866324	3145.160614	4312.912904

Таблица 1. Результаты замеров реального времени исполнения, O(N^2) и O(N^3)

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №3.



Изображение №3 - График работы алгоритма

Таким образом, видим, что реальная сложность алгоритма действительно очень близка к N^3, значит мы верно оценили сложность.

5. Заключение

В ходе выполнения работы мною был реализован алгоритм для решения задачи 312. Burst Balloons на LeetCode. Цель работы была достигнута путём тестирования на массивах с различным количеством элементов. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно рассмотреть варианты других решений от пользователей данного сайта, перенять от них идеи по оптимизации алгоритма с точки зрения используемой памяти.

6. Приложения

Рисунок 4. Это я грустный перечитываю свой отчет в ожидании, когда Владислав Константинович его проверит