ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнила работу

Шакина Анна

Академическая группа №J3110

Принято

Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель работы: разработать алгоритм, определяющий максимальное число монет, которое можно получить, если лопнуть все воздушные шары.

Задачи:

1. Написать код.
2. Посчитать используемую память и асимптотику.
3. Убедиться, что в задаче действительно требуется использование динамического программирования.
4. Реализация
5. Используемые библиотеки

<iostream> - для использования ввода и вывода

<vector> - для использования std::vector

<algorithm> - для использования таких функций, как std::max

* 1. Условие задачи

Даны n воздушных шаров с индексами от 0 до (n – 1). На каждом шаре нарисован номер, представленный массивом nums. Вам нужно лопнуть все шары.

Если лопнуть i-й шарик , то получите nums[i - 1] \* nums[i] \* nums[i + 1] монет. Если (i – 1) или (i + 1) выходит за пределы массива, то считайте, что это шарик с 1 на нём.

Определите максимальное количество монет, которое вы можете собрать, разумно ломая шарики.

* 1. Алгоритм решения задачи

Если не использовать динамическое программирование и для каждого шарика считать, сколько мы получим монет, если его лопнем, то у алгоритма будет сложность N^N, где N – количество воздушных шаров.

Чтобы оптимизировать решение задачи, можно разбивать её на подзадачи и хранить результаты выполнения этих подзадач. То есть, надо использовать динамическое программирование.

Пусть подзадачи – это вычисление количества монет, которое можно получить, если лопнуть шарики справа и слева от одного из шаров. Хранить результаты подзадач можно, используя вектор векторов.

Тогда программа будет выполнять следующие шаги:

1. Преобразование массива, поступившего на вход: добавит 1 в начало и 1 в конец для корректного подсчёта монет.
2. Обработка шариков с помощью цикла for: для шарика k вычисляется количество монет, которое мы получим, если лопнем шарик k, максимальное количество монет, которое мы получим, если лопнем все шары слева и справа от k.
3. В таблицу maximum\_of\_coins записывается количество монет, вычисленное в шаге 2). Проход по таблице осуществляется с помощью двух вложенных циклов.
   1. Основные фрагменты кода

Преобразование массива:

    int num\_balloons = (int)balloons.**size**();

    std::**vector**<int> **extended\_balloons**(num\_balloons + 2);

    extended\_balloons**[**0**]** = extended\_balloons**[**num\_balloons + 1**]** = 1;

    for (int i = 0; i < num\_balloons; i++)

    {

        extended\_balloons**[**i + 1**]** = balloons**[**i**]**;

    }

Заполнение «таблицы»:

    std::**vector**<std::**vector**<int>> **maximum\_of\_coins**(num\_balloons + 2, std::**vector**<int>(num\_balloons + 2));

    for (int length = 1; length <= num\_balloons; length++)

    {

        for (int left = 1; left <= num\_balloons - length + 1; left++)

        {

            int right = left + length - 1;

            for (int k = left; k <= right; k++)

            {

                int coins = extended\_balloons**[**left - 1**]** \* extended\_balloons**[**k**]** \* extended\_balloons**[**right + 1**]**;

                coins += maximum\_of\_coins**[**left**][**k - 1**]** + maximum\_of\_coins**[**k + 1**][**right**]**;

                maximum\_of\_coins**[**left**][**right**]** = std::**max**(maximum\_of\_coins**[**left**][**right**]**, coins);

            }

        }

    }

1. Подсчёт по памяти

std::vector<int> extended\_balloons: (num\_balloons + 2) \* sizeof(int) = (num\_balloons + 2) \* 4 байт

std::vector<std::vector<int>> maximum\_of\_coins: ((num\_balloons + 2) \* (num\_balloons + 2)) \* sizeof(int) = ((num\_balloons + 2) \* (num\_balloons + 2)) \* 4 байт

Общая память, используемая функцией, будет следующая:

(num\_balloons + 2) \* 4 + ((num\_balloons + 2) \* (num\_balloons + 2)) \* 4 = 4 \* (num\_balloons + 2) \* (1 + (num\_balloons + 2))

1. Подсчёт асимптотики

При расширении вектора шаров extended\_balloons размером num\_balloons + 2 требуется O(N) времени, где N — количество шаров.

Внешний цикл по length проходит от 1 до num\_balloons – O(N).

Внутренний цикл по left проходит от 1 до num\_balloons - length + 1, который в худшем случае также может дать O(n) – O(N).

В самом внутреннем цикле (по k) мы пробуем все возможные значения от left до right – O(N).

Таким образом, у нас есть три вложенных цикла, каждый из которых может выполняться до n раз:

Внешний цикл: O(N), средний цикл: O(N), внутренний цикл: O(N).

Таким образом, общая асимптотика составляет O(N^3).

1. Заключение

В ходе выполнения работы мною был реализован алгоритм подсчёта максимального количества монет, которое можно получить, если лопнуть все воздушные шары. Цель работы была достигнута с помощью динамического программирования. В качестве дальнейших исследований можно предложить оптимизацию алгоритма для уменьшения используемой памяти.

1. Приложения

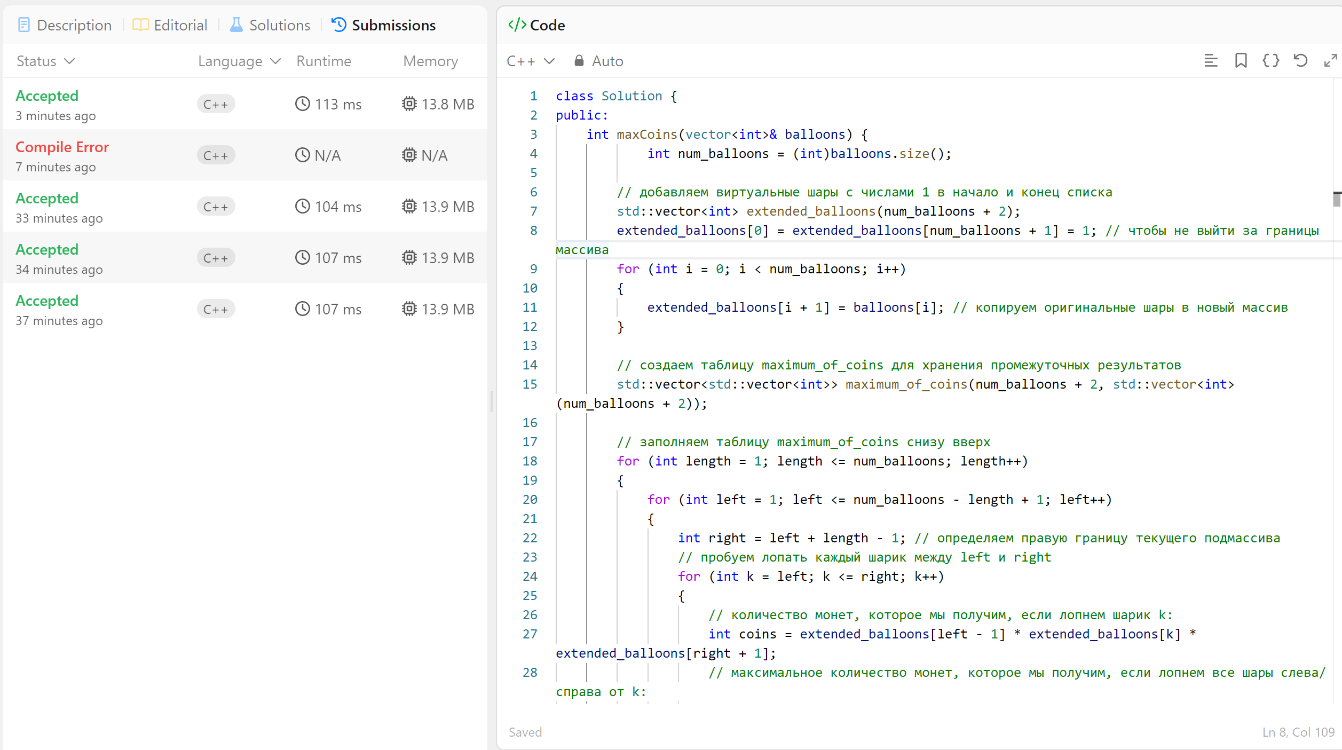


Рис. 1. Успешное прохождение тестов на leetcode

Листинг кода burst\_balloons.cpp

#include <iostream>

#include <vector>

#include <algorithm>

*// функция для вычисления максимального количества монет, которые можно получить*

int **max\_coins**(std::**vector**<int> &balloons)

{

    int num\_balloons = (int)balloons.**size**();

*// добавляем шары с числами 1 в начало и конец списка*

    std::**vector**<int> **extended\_balloons**(num\_balloons + 2);

    extended\_balloons**[**0**]** = extended\_balloons**[**num\_balloons + 1**]** = 1; *// чтобы не выйти за границы массива*

    for (int i = 0; i < num\_balloons; i++)

    {

        extended\_balloons**[**i + 1**]** = balloons**[**i**]**; *// копируем оригинальные шары в новый массив*

    }

*// создаем таблицу maximum\_of\_coins для хранения промежуточных результатов*

    std::**vector**<std::**vector**<int>> **maximum\_of\_coins**(num\_balloons + 2, std::**vector**<int>(num\_balloons + 2));

*// заполняем таблицу maximum\_of\_coins*

    for (int length = 1; length <= num\_balloons; length++)

    {

        for (int left = 1; left <= num\_balloons - length + 1; left++)

        {

            int right = left + length - 1; *// определяем правую границу текущего подмассива*

*// пробуем лопать каждый шарик между left и right*

            for (int k = left; k <= right; k++)

            {

*// количество монет, которое мы получим, если лопнем шарик k:*

                int coins = extended\_balloons**[**left - 1**]** \* extended\_balloons**[**k**]** \* extended\_balloons**[**right + 1**]**;

*// максимальное количество монет, которое мы получим, если лопнем все шары слева и справа от k:*

                coins += maximum\_of\_coins**[**left**][**k - 1**]** + maximum\_of\_coins**[**k + 1**][**right**]**;

*// обновляем значение maximum\_of\_coins[left][right]*

                maximum\_of\_coins**[**left**][**right**]** = std::**max**(maximum\_of\_coins**[**left**][**right**]**, coins);

            }

        }

    }

*// возвращаем максимальное количество монет при лопании всех шаров*

    return maximum\_of\_coins**[**1**][**num\_balloons**]**;

}

int **main**()

{

    std::**vector**<int> balloons = {1, 5};

    std::cout **<<** **max\_coins**(balloons) **<<** std::**endl**;

    return 0;

}