ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу Мельников Олег Академическая группа №J3112

Принято Дунаев Максим Владимирович

Санкт-Петербург 2024

# ВВЕДЕНИЕ

**Цель работы:** разработать и реализовать алгоритм на языке программирования C++ для решения задачи “Матрешка”.

Задача заключается в том, чтобы определить какое максимальное количество конвертов возможно поместить друг в друга.

**Задачи:**

* Изучить условия задачу;
* Разработать алгоритм на основе динамического алгоритма;
* Проанализировать результаты, оценить эффективность.

# ТЕОРЕТИЧЕСКАЯ ПОДГОТОВКА

Задача заключается в следующем: имеется двухмерный массив целых чисел. 1 параметр представляет ширину конверта, 2 параметр представляет высоту конверта. Один конверт может поместиться в другой тогда и только тогда, когда ширина и высота одного конверта больше ширины и высоты другого конверта.

Вернуть максимально возможное количество конвертов русской кукле (т.е. положить один в другой).

**Примечание.** Конверт нельзя повернуть.

**Ограничения:**

* 1 <= envelopes.length <= 105
* envelopes[i].length == 2
* 1 <= wi, hi <= 105

# РЕАЛИЗАЦИЯ

Чтобы решить данную задачу, использовалось динамическое программирование для эффективного нахождения длины максимальной увеличивающейся подпоследовательности высот конвертов после их предварительной сортировки.

* **Сортировка:**
  + Внутри метода maxEnvelopes, конверты сортируются с помощью функции sort, которая требует функции сравнения, чтобы знать, каким образом упорядочивать элементы. В данной реализации, лямбда-функция передаётся как третий аргумент в sort.
  + Лямбда-функция определяет, как сравнивать конверты. Она делает следующее: сначала сравнивает ширину двух конвертов (a[0] и b[0]). Если ширина одинаковая, то выполняется сравнение по высоте (a[1] и b[1]).При таком сравнении, если ширины равны, высоты сортируются в порядке убывания (a[1] > b[1]), что предохраняет от ситуации, когда два конверта с одинаковой шириной могут быть вложены друг в друга. Если ширина не равна, конверты сортируются в порядке возрастания ширины (a[0] < b[0]).
* **Использование массива dp**:
  + Мы используем одномерный массив dp, чтобы хранить текущие максимальные высоты последовательностей конвертов. Каждый элемент массива представляет собой последнюю высоту увеличивающейся подпоследовательности, где длина массива dp в конце вычислений будет равна максимальному количеству вложенных конвертов.
* **Бинарный поиск**:
  + Для вставки высоты конверта в массив dp мы используем метод бинарного поиска через функцию lower\_bound. Это позволяет нам находить место для вставки нового элемента, поддерживая упорядоченность массива, и делать это за O(log n) времени.
* **Обновление значений**:
  + Если текущая высота больше всех в массиве dp, мы добавляем её. Если нет, мы обновляем существующий элемент, что позволяет изолировать индекс, по которому можно продолжать строить увеличивающуюся последовательность.

# ЭКСПЕРЕМИНТАЛЬНАЯ ЧАСТЬ

Анализ асимптотической сложности алгоритма

1. Сортировка конвертов

**Операция**: Сортировка массива конвертов.

**Сложность**: Используемая сортировка имеет сложность O(n log n).

2. Поиск длины максимальной увеличивающейся последовательности

**Операция**: Проход по каждому конверту и вставка высоты в массив dp с использованием бинарного поиска.

**Сложность**:

Проход по массиву конвертов — O(n).

Вставка каждого элемента в массив dp с использованием бинарного поиска — O(log m), где m — текущая длина массива dp. В худшем случае длина dp также будет n, что приводит к сложности O(n log n) для этой части.

Общая асимптотическая сложность складывается из двух основных компонентов:

Сложность сортировки: O(n log n)

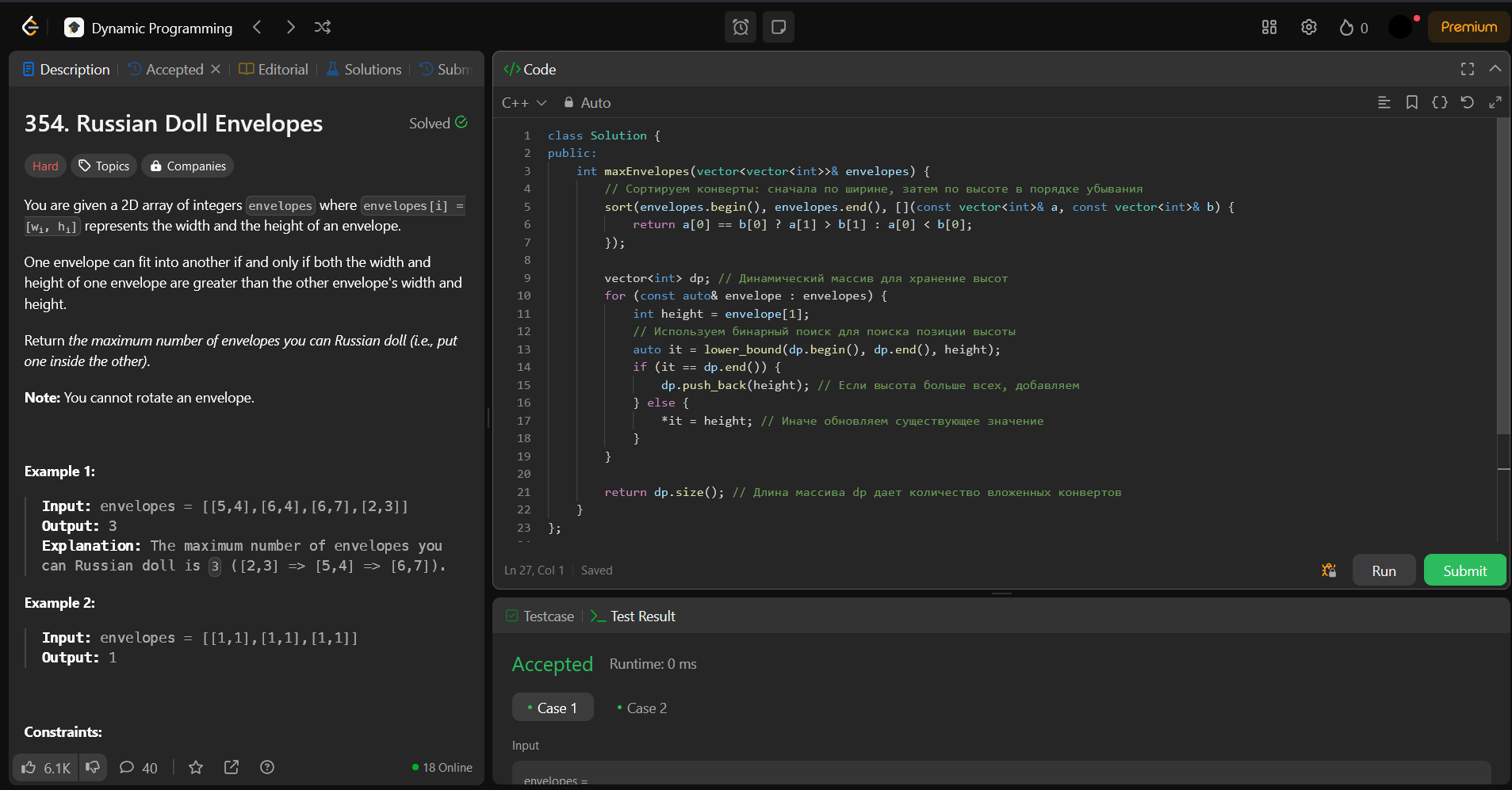
Сложность поиска длины увеличивающейся последовательности: O(n log n)

Таким образом, общая сложность алгоритма составляет:

O(n log n) + O(n log n) = O(n log n)

3. Пространственная сложность

Используемый дополнительный массив dp может занимать до O(n) памяти в худшем случае.



# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был разработан и реализован алгоритм “Матрёшка”. Цель работы была достигнута: алгоритм успешно решает задачу на тестовых данных. Также алгоритм показывает высокую эффективность со сложностью O(n log n)

# ПРИЛОЖЕНИЕ

Листинг кода решения

class Solution {

public:

    int maxEnvelopes(vector<vector<int>>& envelopes) {

        // Сортируем конверты: сначала по ширине, затем по высоте в порядке убывания

        sort(envelopes.begin(), envelopes.end(), [](const vector<int>& a, const vector<int>& b) {

            return a[0] == b[0] ? a[1] > b[1] : a[0] < b[0];

        });

        vector<int> dp; // Динамический массив для хранение высот

        for (const auto& envelope : envelopes) {

            int height = envelope[1];

            // Используем бинарный поиск для поиска позиции высоты

            auto it = lower\_bound(dp.begin(), dp.end(), height);

            if (it == dp.end()) {

                dp.push\_back(height); // Если высота больше всех, добавляем

            } else {

                \*it = height; // Иначе обновляем существующее значение

            }

        }

        return dp.size(); // Длина массива dp дает количество вложенных конвертов

    }

};