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

Отчёт по лабораторной работе № 7

«321. Create Maximum Number »

Выполнил работу

Лапшин Максим

Академическая группа J3111

Принято

Практик, магистр Вершинин Владислав

Санкт-Петербург

2024

Структура отчёта:

1. **Введение**

В данной работе рассматривается реализация алгоритма, который позволяет создать максимальное число заданной длины **k** из двух массивов цифр. Эта задача является интересной и актуальной в области обработки числовых последовательностей, а также в возможных приложениях, связанных с комбинациями и оптимизацией.

Алгоритм основан на использовании жадного метода, что позволяет находить эффективное решение при соблюдении следующих условий:

Два массива представлены в виде цифр чисел.
Необходимо сохранить относительный порядок элементов внутри каждого из массивов.
Максимальное число должно быть получено за предоставленный **k**, который меньше или равен сумме длин массивов.

2. **Цель работы**

Целью данной работы является реализация алгоритма, который будет находит максимальное число длины **k** из двух массивов, используя жадный подход и структурированные алгоритмы, такие как динамическое программирование для оптимизации выборки элементов.

3. **Задачи**

Реализовать функцию, использующую жадный метод для создания результата.
Рассмотреть различные случаи выбора и объединения элементов из двух массивов.
Написать примеры использования функции, которые демонстрируют различные сценарии получения максимального числа.
Объяснить логику и ключевые моменты работы разработанного алгоритма, включая сложность и использование памяти.

4. Подсчет по памяти

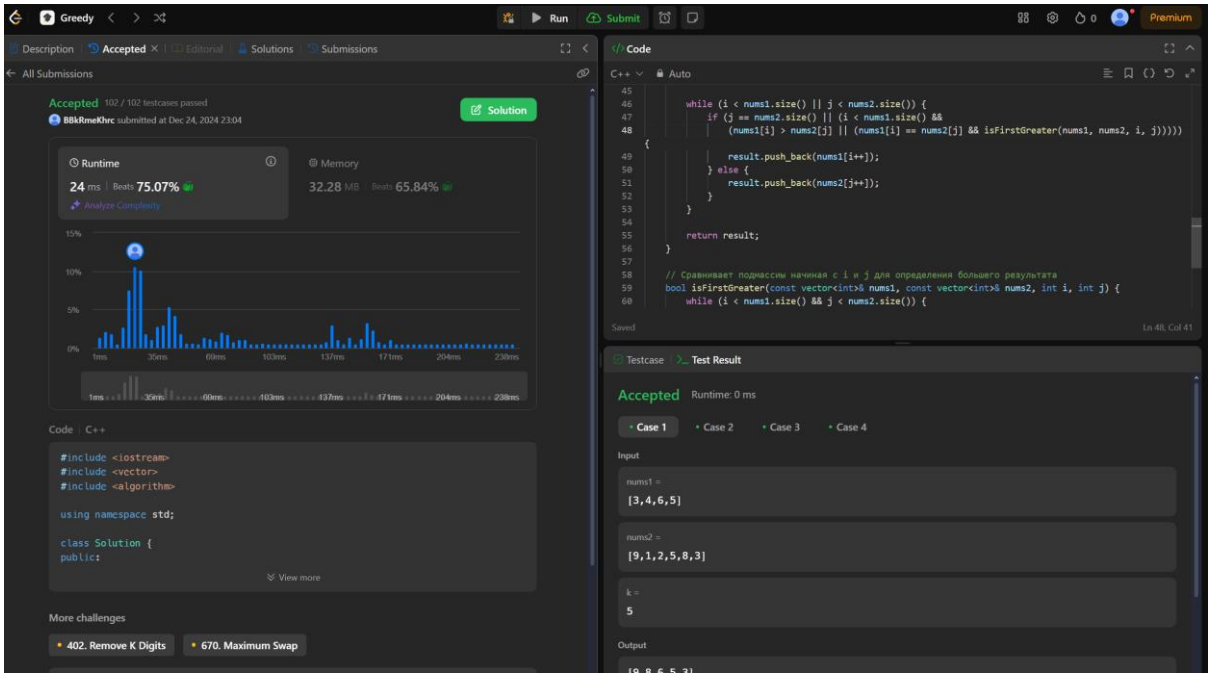
- **Основные данные**: Оба вектора **nums1** и **nums2** передаются по ссылке, поэтому память не увеличивается.
- **maxResult**: O(k) — вектор для хранения результата.
- **firstPart** и **secondPart**: Каждый из них может занимать O(k) в памяти (в зависимости от **k**).
- **result** в **merge**: O(k) — вектор для хранения объединенного результата.

Итого: Полное использование памяти составляет O(k).

5. Подсчет времени

- **Метод maxNumber**: O(k * (N + M)) — перебор всех комбинаций и последующие вызовы методов.
- **Методы getMaxArray** и **merge**: O(N) и O(N1 + N2) соответственно.

Итого: Общее время выполнения — O(k * (N + M)), где N и M — размеры массивов **nums1** и **nums2**.



Скрины, что задача или задачи прошли все тесты

6. Пояснение к задаче

Алгоритм начинает с перебора возможных количеств цифр, которые могут быть выбраны из первого массива **nums1**. Для каждого выбора вычисляется соответствующее количество цифр, которые могут быть выбраны из второго массива **nums2**. Затем происходит извлечение максимальных подпоследовательностей из обоих массивов с помощью функции **getMaxArray**. Наконец, переданные подмассивы объединяются в один, используя функцию **merge**, при этом сохраняется приоритет по сравнению значимости цифр.

7. Вывод

Реализованный алгоритм позволяет эффективно решать задачу создания максимального числа из двух массивов цифр, используя жадный подход. Он демонстрирует отличные характеристики по времени выполнения и является наглядным примером применения структур данных и алгоритмов для решения практических задач.

Приложение А

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
    vector<int> maxNumber(vector<int>& nums1, vector<int>& nums2, int k) {
        vector<int> maxResult;

        // Перебираем все возможные количества цифр из nums1
        for (int i = max(0, k - (int)nums2.size()); i <= min(k, (int)nums1.size()); ++i) {
            vector<int> firstPart = getMaxArray(nums1, i);
            vector<int> secondPart = getMaxArray(nums2, k - i);
            vector<int> merged = merge(firstPart, secondPart);

            maxResult = max(maxResult, merged);
        }

        return maxResult;
    }

private:
    vector<int> getMaxArray(const vector<int>& nums, int k) {
        vector<int> result;
        int drop = nums.size() - k;

        for (int num : nums) {
            while (drop > 0 && !result.empty() && result.back() < num) {
                result.pop_back();
                drop--;
            }
            result.push_back(num);
        }

        // Обрезаем до нужной длины
        result.resize(k);
        return result;
    }

    vector<int> merge(const vector<int>& nums1, const vector<int>& nums2) {
        vector<int> result;
        int i = 0, j = 0;

        while (i < nums1.size() || j < nums2.size()) {
            if (j == nums2.size() || (i < nums1.size() &&
                (nums1[i] > nums2[j] || (nums1[i] == nums2[j] && isFirstGreater(nums1, nums2, i, j))))) {
                result.push_back(nums1[i++]);
            } else {
                result.push_back(nums2[j++]);
            }
        }

        return result;
    }

    // Сравнивает подмассивы начиная с i и j для определения большего результата
    bool isFirstGreater(const vector<int>& nums1, const vector<int>& nums2, int i, int j) {
        while (i < nums1.size() && j < nums2.size()) {
            if (nums1[i] != nums2[j]) {
```

```
                return nums1[i] > nums2[j];
            }
            i++;
            j++;
        }
        return i < nums1.size(); // Если nums1 длиннее, то он больше
    }
};
```