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

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Бабич Александр

Академическая группа №J3111

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Основная цель — научиться находить и применять методы динамического программированиия на задачах с алгоритмическим умыслом. Задача — найти, и решить задачи по теме «ДП»

2. Теоретическая подготовка

Необходимо знать как пишутся и из чего состоит ДП: Конкретно здесь, разбиение на подзадачи

3.Реализация (Первая задача)

1)В первую очередь проверяется наличие символов 'w' или 'm' в строке. Если хотя бы один из них присутствует, сразу выводится результат 0, так как эти символы недопустимы. Инициализация массива состояний dp:

2)Создается массив dp размером n + 1, где n — длина строки. Инициализируются начальные значения:

dp[0] = 1 — пустая строка считается одним возможным разбиением.

dp[1] = 1 — первая позиция также имеет одно возможное разбиение, так как одна буква без других символов всегда является допустимым разбиением.

3)Начинаем итерацию от позиции 2 до n. Для каждой позиции i:

Берем значение dp[i-1], которое представляет количество способов разбить строку до предыдущей позиции, и присваиваем его dp[i].

Проверяем, совпадают ли символы на позиции i−1 и

i−2 и являются ли они 'u' или 'n'. Если да, значит можно объединить текущий символ с предыдущим, добавив к dp[i] значение dp[i-2].

Каждый раз значение dp[i] берется по модулю 10^9+7, чтобы избежать переполнения чисел.

Задача 2

Эта задача решается с помощью предварительных вычислений максимальных сумм подмассивов. Алгоритм использует три массива: один для максимальных сумм подмассивов от начала до каждой позиции (maxPrefixSums), второй — от конца к началу (maxSuffixSums), и третий — для

накопленных сумм, что помогает быстрее находить суммы подмассивов любой длины.

1) Предварительное накопление суммы подмассива длины k:Вычисляется начальная сумма первых k элементов, которая используется для последующего обновления значений в maxPrefixSums и maxSuffixSums.

2) Заполнение массива максимальных префиксов maxPrefixSums:

Для каждой позиции i от k до n, текущая сумма обновляется, добавляя следующий элемент и вычитая самый левый из предыдущего подмассива длины k. Если новая сумма больше значения для предыдущего индекса, значение обновляется. Это позволяет хранить максимальные суммы подмассивов длиной k от начала до каждой позиции.

3) Заполнение массива максимальных суффиксов maxSuffixSums:

Подобно предыдущему шагу, массив суффиксов заполняется с конца, сохраняя максимальные суммы подмассивов длиной k от каждой позиции до конца.

4)Расчёт накопленных сумм: cumulativeSums заполняется таким образом, что каждая позиция содержит сумму всех элементов от начала до текущего индекса. Это позволяет быстро находить сумму подмассива между любыми двумя позициями.

5)Основной цикл для выбора трёх подмассивов: Цикл проходит по всем возможным начальным индексам среднего подмассива длиной k, от k до $n-2\times k$.

Для каждого индекса mid вычисляются суммы:

leftSum — максимальная сумма подмассива длиной k в префиксе перед mid,

midSum — сумма подмассива длиной k начиная с mid,

rightSum — максимальная сумма подмассива длиной k в суффиксе после mid + k.

Если сумма этих трёх подмассивов больше текущего максимума, обновляются maxSum и индексы подмассивов.

4.Экспериментальная часть.(Задача 1)

Подсчет по памяти: Строка s: Строка хранится в памяти в виде массива символов. Она занимает O(n) памяти.

Вектор dp: Вектор используется для хранения промежуточных результатов динамического программирования. Он имеет размер n+1 то есть O(n) памяти.

То есть всего o(n) память

Посчитает время тут по сути 1 главный цикл, второй для определения символа не так важен.

```
for (int i = 2; i <= n; ++i) { // o(n)
    dp[i] = dp[i - 1]; // o(1)
    if (s[i - 1] == s[i - 2] && (s[i - 1] == 'u' || s[i - 1] == 'n')) // o(1)
        dp[i] = (dp[i] + dp[i - 2]) % MOD; // o(1)
}
```

Этот цикл выполняется O(n) раз, где

n — длина строки. Каждый шаг цикла включает несколько операций: присваивание и условную проверку, что выполняется за O(1) времени. Таким образом, сложность этого цикла — O(n).

Задача 2

Время выполнения:

Заполнение массивов maxPrefixSums и maxSuffixSums занимает O(n).

Заполнение массива накопленных сумм cumulativeSums также занимает O(n).

Основной цикл, проходящий по индексам mid для выбора подмассивов, также работает за O(n).

Итоговая асимптотика по времени: O(n).

Использование памяти:

Дополнительные массивы maxPrefixSums, maxSuffixSums и cumulativeSums занимают O(n) памяти каждый.

Итоговое использование памяти: O(n).

5. Заключение

В ходе выполнения работы я решил две задачи на динамическое программирование. Динамическое программирование в первой задачи используется для эффективного подсчета количества способов разбить строку, используя промежуточные результаты. ДП позволяет избежать повторного пересчета количества способов для каждой подстроки.
Во второй задаче используются предвычисленные значения для оптимизации поиска. Хранение максимальных префиксных и суффиксных сумм позволяет мгновенно выбирать подмассивы с наибольшей суммой, избегая многократных суммирований и уменьшая сложность задачи до O(n).

6.Приложения

ПРИЛОЖЕНИЕ А

Решение задачи Аппарат Константции

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


const int MOD = 1000000007;

int main() {
    std::string s;
    std::cin >> s;

    for (char x : s) {
        if (x == 'w' || x == 'm') {
            std::cout << 0 << '\n';
            return 0;
        }
```

```
    }

    int n = s.size();
    std::vector<int> dp(n + 1);
    dp[0] = 1;
    dp[1] = 1;
    for (int i = 2; i <= n; ++i) {
        dp[i] = dp[i - 1];
        if (s[i - 1] == s[i - 2] && (s[i - 1] == 'u' || s[i - 1] == 'n'))
            dp[i] = (dp[i] + dp[i - 2]) % MOD;
    }

    std::cout << dp[n] << '\n';

    return 0;
}
```

Приложение Б

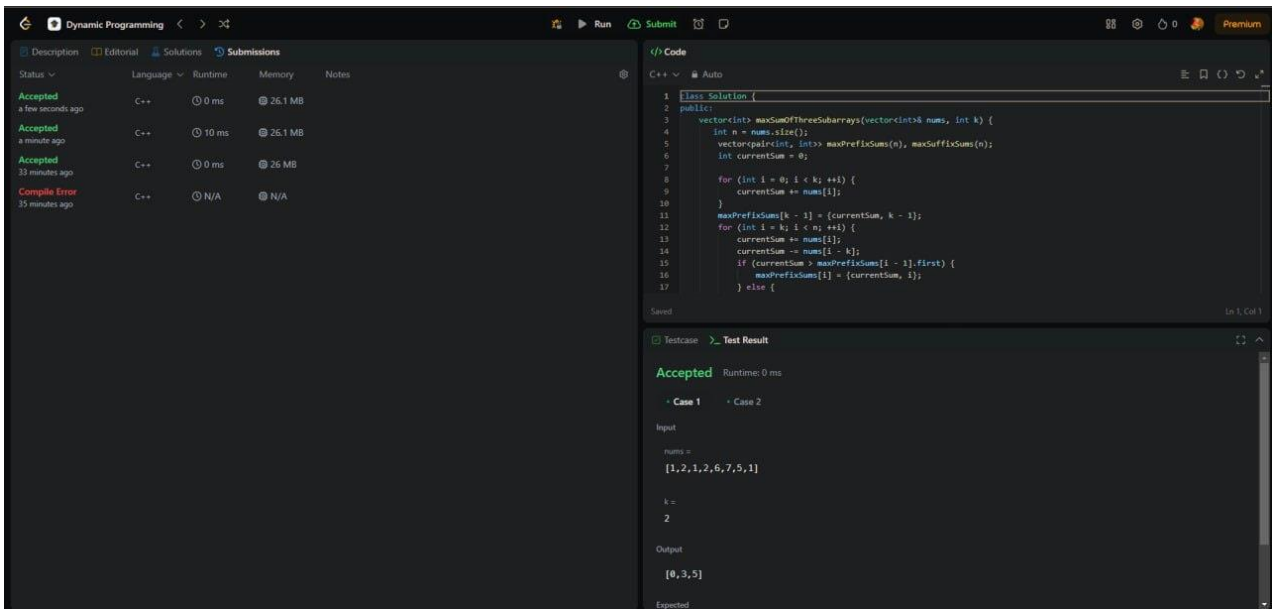
Задача Аппарат Константции прошла тесты

Мои посылки

№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
290976971	11.11.2024 00:11	Vardor	C - Аппарат Константции	C++20 (GCC 13-64)	Полное решение	77 мс	100 КБ

Приложение В

Задача Maximum Sum of 3 Non-Overlapping Subarrays с пройденными тестами



Приложение Г

Код решения Maximum Sum of 3 Non-Overlapping Subarrays

```
class Solution {
public:
    vector<int> maxSumOfThreeSubarrays(vector<int>& nums, int k) {
        int n = nums.size();
        vector<pair<int, int>> maxPrefixSums(n), maxSuffixSums(n);
        int currentSum = 0;

        for (int i = 0; i < k; ++i) {
            currentSum += nums[i];
        }
        maxPrefixSums[k - 1] = {currentSum, k - 1};
        for (int i = k; i < n; ++i) {
            currentSum += nums[i];
            currentSum -= nums[i - k];
            if (currentSum > maxPrefixSums[i - 1].first) {
                maxPrefixSums[i] = {currentSum, i};
            } else {
                maxPrefixSums[i] = maxPrefixSums[i - 1];
            }
        }

        currentSum = 0;
        for (int i = n - 1; i >= n - k; --i) {
            currentSum += nums[i];
        }
        maxSuffixSums[n - k] = {currentSum, n - k};
```

```
        for (int i = n - k - 1; i >= 0; --i) {
            currentSum += nums[i];
            currentSum -= nums[i + k];
            if (currentSum >= maxSuffixSums[i + 1].first) {
                maxSuffixSums[i] = {currentSum, i};
            } else {
                maxSuffixSums[i] = maxSuffixSums[i + 1];
            }
        }

        vector<int> cumulativeSums(n, 0);
        cumulativeSums[0] = nums[0];
        for (int i = 1; i < n; ++i) {
            cumulativeSums[i] = cumulativeSums[i - 1] + nums[i];
        }

        int maxSum = 0;
        vector<int> maxSumIndex;

        for (int mid = k; mid <= n - 2 * k; ++mid) {
            int leftSum = maxPrefixSums[mid - 1].first;
            int midSum = cumulativeSums[mid + k - 1] - cumulativeSums[mid - 1];
            int rightSum = maxSuffixSums[mid + k].first;

            int totalSum = leftSum + midSum + rightSum;

            if (totalSum > maxSum) {
                maxSum = totalSum;
                int leftIndex = maxPrefixSums[mid - 1].second - k + 1;
                int midIndex = mid;
                int rightIndex = maxSuffixSums[mid + k].second;

                maxSumIndex = {leftIndex, midIndex, rightIndex};
            }
        }

        return maxSumIndex;
    }
};
```