ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«динамическое программирование»

Выполнил работу

Лорс Алина

Академическая группа №J3112

Санкт-Петербург

2024

Структура отчёта:

Введение

Цель данной работы заключается в разработке эффективного алгоритма для подсчета количества различных подпоследовательностей строки s, которые равны заданной строке t, с использованием динамического программирования.

Задачи работы

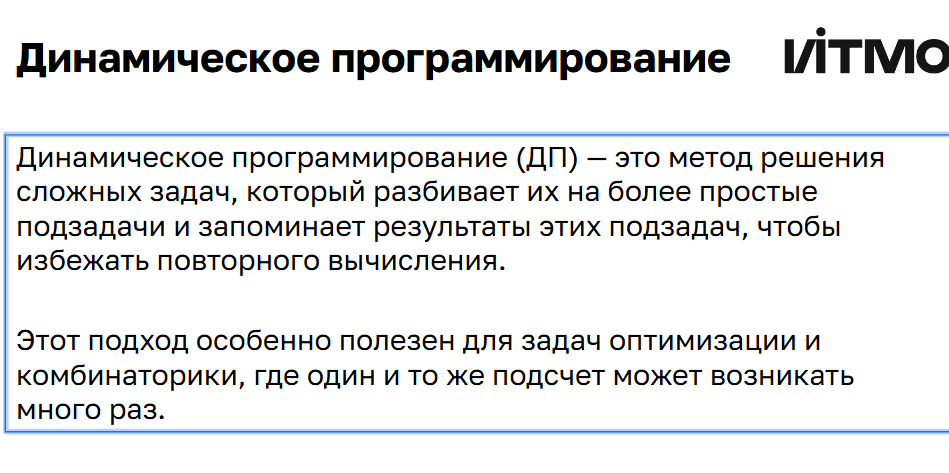
-Понять условия задачи, входные и выходные данные.

-Обосновать, почему простые методы (например, линейный проход) не могут быть применены.

-Оценить временную и пространственную сложность предложенного алгоритма

- Проверить корректность работы алгоритма

Теоретическая подготовка



Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

int:

Для индексов строк s и t.

Размер: 4 байта.

long long:

Для хранения значений массива dp, чтобы избежать переполнения при больших строках.

Размер: 8 байт.

vector<long long>:

Используется для одномерного массива dp.

Размер зависит от длины строки t.

Реализация

стандартные библиотеки языка C++:

<vector>: для хранения массива dp.

<string>: для работы со строками.

<iostream>: для ввода и вывода данных.

Инициализация массива dp:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описаниеЗаполнение массива справа налево: Чтобы не перезаписать значения, используем цикл по строке t в обратном порядке:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Возврат результата: return dp[targetLength];

Этап 1: Анализ задачи и постановка проблемы

Этап 2: основное соотношение для заполнения таблицы dp

если символы совпадают: dp[i][j] = dp[i-1][j-1] + dp[i-1][j]

else: dp[i][j] = dp[i-1][j]

Этап 3: Если t пустая, то существует ровно один способ получить её из s: удалить все символы. Если s пустая и t не пустая, способов нет.

Экспериментальная часть

Подсчет памяти

int sourceLength и targetLength = 4 \* 2

константа long long MODULO = 8 байт

Массив waysToFormSubstring:

Размер: targetLength + 1 элементов long long (8 байт)=  
(n + 1) × 8 байт

Итого: 16 + 8 \* (n+1) =   
24 + 8n

Подсчет асимптотики,где m=sourceLength,n=targetLength.

O(m); for (int I =1; i <= sourceLength; i++)

for (int j = targetLength; j >= 1; j--) { ... }

тот цикл выполняется от n до 1 = n раз в i (так как это вложенный цикл) => O(n)

Вывод = O(m×n) время

Дано две строки:

s (исходная строка) длиной m.

t (целевая подпоследовательность) длиной n.

Нужно подсчитать количество различных подпоследовательностей строки s, которые равны строке t.== Сколько способов существует для создания подстроки t[0:j] из s[0:i]?"

Решение этой задачи для индекса i зависит от решения для индекса i−1Например, если символы совпадают (s[i-1] == t[j-1]), можно как использовать текущий символ s[i-1], так и не использовать его. использовать результаты уже решенных подзадач и делая маленькие подзачи мы сможем решить эту задачу, ведь иначе не применяя динамичекое программирование При длине строк m и n, простое перечисление всех комбинаций приведет к экспоненциальному времени. А если использовать дп, то нам найти надо количество способов сформировать строку t из строки s, нужно рассмотреть все возможные комбинации удалений символов из s, сохраняя порядок символов в строке. Это как раз выполняется благодаря заполнению таблицу dp слева направо и сверху вниз, начиная с самых маленьких подзадач (например, пустая строка t и хранения результатов подзадач в массиве dp и используя их повторно вместо того, чтобы пересчитывать их каждый раз.

Минусы обычного прохода : Обычный проход по строке не способен перебрать все возможные комбинации символов без использования перебора, так как у нас o(n\*m):

Мы должны учитывать множество вариантов, где символы совпадают на разных позициях.

Перебор всех комбинаций вручную линейным проходом экспоненциально сложен

Для индекса i мы должны учитывать результаты из предыдущих индексов i-1.

Если символы совпадают (s[i] == t[j]), результат зависит от двух вариантов:

Использовать текущий символ.

Не использовать текущий символ.

Это требует сохранения состояний, что невозможно в простом линейном проходе

Обычный проход посчитает только одно совпадение и не сможет учесть другие комбинации.

2\*\* m не нужно считать => не масштабируется для больших строк из-за экспоненциального перебора.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Заключение

Главная цель работы была достигнута: программа корректно находит количество таких подпоследовательностей, и она работает эффективно даже на больших строках. Лабораторная работа оказалась для меня очень сложной, потому что придумать решение задачи с использованием динамического программирования казалось непосильной и страшной задачей. Но в процессе работы я поняла, что если разобраться в самой идее задачи и начать с небольших примеров, то можно постепенно всё оптимизировать и решить проблему шаг за шагом.

Я провела тестирование на разных примерах, включая граничные случаи и очень длинные строки, и убедилась, что программа дает правильные и стабильные результаты.

В результате работы получилось:

Программа работает за O(m×n) по времени, где m — длина строки s, а n — длина строки t.

Память была оптимизирована до O(n) благодаря использованию одномерного массива вместо двумерной таблицы.

Код устойчив к переполнению благодаря использованию арифметики по модулю.

На будущее можно попробовать улучшить программу еще больше. Например, можно сделать так, чтобы она работала еще быстрее