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

Отчёт по лабораторной работе № 6

«Даны две строки s и t. Найти количество всех подпоследовательностей s, эквивалентных t.»

Выполнил работу

Воробьев Егор

Академическая группа J3113

Принято

Ассистент, Дунаев Максим

Санкт-Петербург

2024

1. Введение

Цель: Найти все подпоследовательности s, эквивалентные t.

Задачи:

1) Написать оптмизированный код с применением динамического программировния.
2) Протестировать и сравнить алгоритм на различных размерах строк.

2. Теоретическая подготовка

Типы данных:

1) std::vector – для реализации двух массивов текущего шага и предыдущего;
2) std::string – для инициализации двух строк s и t.

Заголовочные файлы:

1) <vector> - для использования типа данных std::vector;
2) <string> - для использования типа данных std::string;
3) <chrono> - для измерения времени выполнения алгоритма.

3. Реализация

1) Сначала создаем два массива с длиной на единицу больше строки t, этими массивами мы будем вычислять количество способов получить определенную подпоследовательность s, включая пустую. Первый элемент двух массивов равен единице, так как получить пустую последовательность можно лишь одним способом: убрать все символы из строки, то есть получить пустую подпоследовательность.

```
int numDistinct(string s, string t) {
    int n = s.size(), m = t.size();
    std::vector<int> prev(m+1, 0), curr(m+1, 0);
    prev[0] = 1;
    curr[0] = 1;
```

Изображение №1 – Стартовые действия

2) Основная часть заключается в переборе символов s и t. А именно для каждого символа t подбираем символ из s. Если два символа равны, то к элементу в текущем массиве прибавляем количество способов получить прошлую подпоследовательность. Если не равны, то оставляем элемент из текущего массива таким же, как и в

предыдущем массива. После прохода по всем символам s массив предыдущего шага становится массивом текущего.

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++) {
        if (s[i-1] == t[j-1])
            curr[j] = prev[j-1] + prev[j];
        else
            curr[j] = prev[j];
    }
    prev = curr;
}
```

Изображение №2 – Сопоставление каждому символу t символа s и заполнение массивов двух шагов.

3) В конце выводим количество способов составить t, то есть последний элемент из массива шага.

```
return curr[m];
```

Изображение №3 – Вывод количества способов составления строки t.

4. Экспериментальная часть

Теоретически, сложность алгоритма O(n*m), так как всего два цикла, один проходит n элементов, а другой – m элементов.

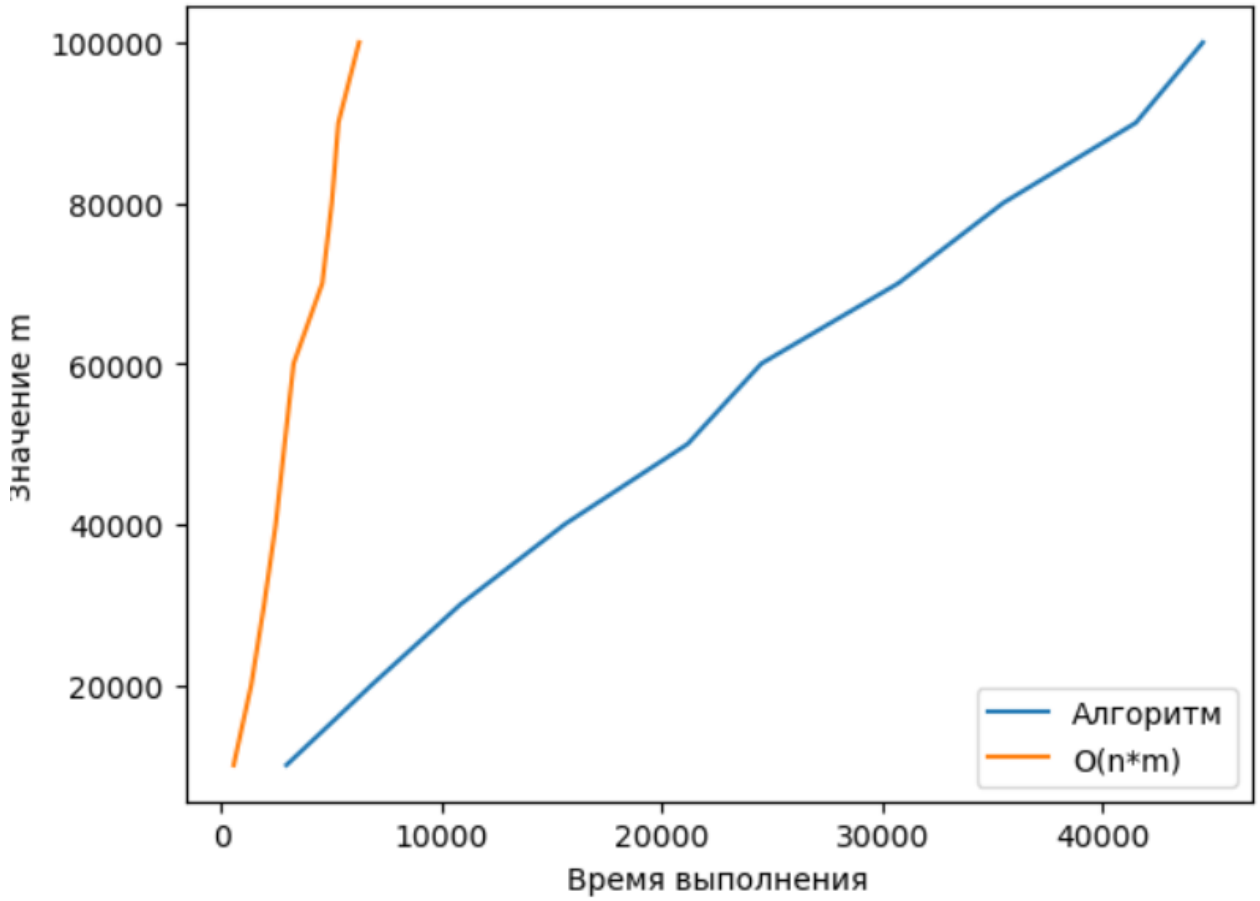
Для определения эффективности алгоритма, было измерено его время выполнения на разных размерах массива. Собранная статистика отображена в таблице №1.

Таблица №1 – Время выполнения алгоритма (в мс) при различных размерах массива

Значение n	Значение m	Наш алгоритм	O(n*m)
100000	10000	2941	548
100000	20000	6819	1348
100000	30000	10837	1932
100000	40000	15580	2460
100000	50000	21161	2861
100000	60000	24502	3269

100000	70000	30717	4571
100000	80000	35475	5003
100000	90000	41505	5302
100000	100000	44542	6240

Ниже предоставлен график, визуализирующий данные из таблицы №1.



Изображение №4 – График зависимости времени выполенения алгоритма от размера массива и нотации, близкой к нашей

Как видно из графика, сложность алгоритма точно больше O(n*m). Это явление можно объяснить тем, что помимо циклов есть и другие операции, такие как проверка условия, вставка элемента, обращение по индексу и т.д. Все они в сумме и повлияли на время выполнения.

5. Заключение

В ходе выполнения работы мною был реализован оптимизированный алгоритм решения задачи с платформы leetcode, который прошел все тесты на сайте. Цель работы была достигнута путём тестирования алгоритма на разных входных данных. Полученные результаты примерно равны теоретическим оценкам сложности алгоритма.

6. Приложения

ПРИЛОЖЕНИЕ 1

Основной код алгоритма.

```cpp
int numDistinct(string s, string t) {
    int n = s.size(), m = t.size();
    std::vector<int> prev(m+1, 0), curr(m+1, 0);
    prev[0] = 1;
    curr[0] = 1;
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=m; j++) {
            if (s[i-1] == t[j-1])
                curr[j] = prev[j-1] + prev[j];
            else
                curr[j] = prev[j];
        }
        prev = curr;
    }
    return curr[m];
}
```

ПРИЛОЖЕНИЕ 2

Ссылка на задачу на платформе leetcode.

https://leetcode.com/problems/distinct-subsequences/description/?envType=problem-list-v2&envId=dynamic-programming&difficulty=HARD

ПРИЛОЖЕНИЕ 3

Скриншот решенной задачи на платформе.