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

Отчёт по лабораторной работе № 6

«10. Regular Expression Matching»

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 сопоставления строк с использованием регулярных выражений. Регулярные выражения находят широкое применение в области обработки текста, поиска и валидации данных, а также в других задачах программирования. В частности, мы реализуем алгоритм, который позволяет определять, соответствует ли заданная строка определенному шаблону, содержащему символы, такие как . и *, где . соответствует любому символу, а * указывает на ноль или более экземпляров предыдущего символа.

2. **Цель работы**

Целью данной работы является реализация и демонстрация работы алгоритма сопоставления строк, который позволяет определить, соответствует ли строка заданному шаблону, используя динамическое программирование для эффективного решения проблемы.

3. **Задачи**

Реализовать функцию, использующую динамическое программирование для сопоставления строки с шаблоном.

Рассмотреть различные случаи, когда шаблон может включать специальные символы, такие как . и *.

Написать примеры использования функции, показывающие различные сценарии сопоставления строк.

Объяснить логику работы разработанного алгоритма и его ключевые моменты.

4. **Подсчет по памяти**

В данном решении используется двумерный вектор dp, который хранит информацию о том, соответствуют ли подстроки строки s и шаблона p. Размер этого вектора определяется размерами строки и шаблона:

s.size() + 1 и p.size() + 1.

Таким образом, память, используемая для хранения таблицы dp, составляет:O(m·n)где m — длина строки s, а n — длина шаблона p.

Кроме того, память для хранения входных данных (строки и шаблона) также занимаетO(m+n)

O(m+n) , но она не учитывается в основном подсчете, так как чаще всего интересует лишь дополнительная память.

5. Подсчет асимптотики

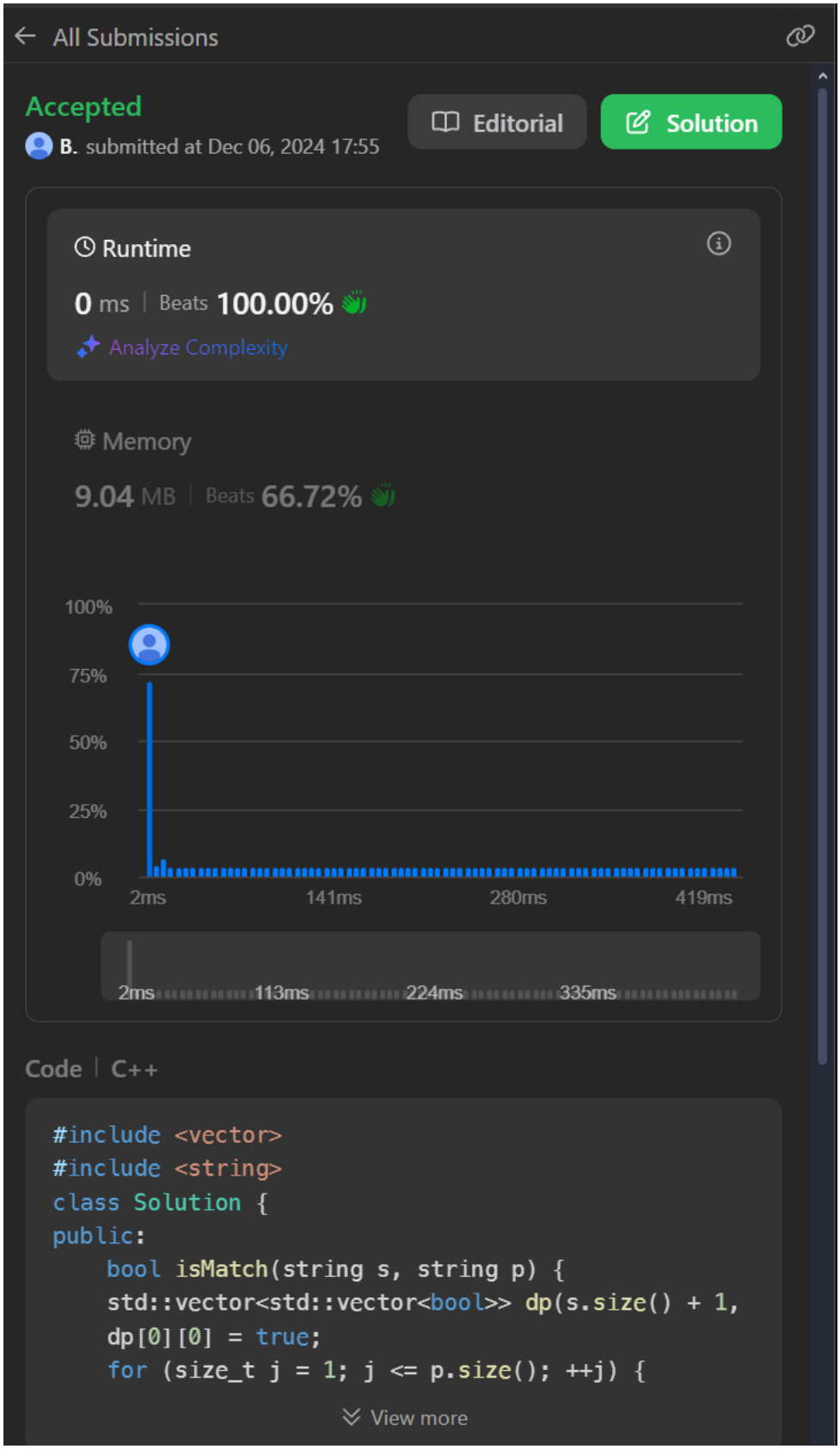
Асимптотика этого алгоритма выясняется из вложенных циклов, которые проходят все возможные подстроки строки и шаблона:

Внешний цикл проходит по всем символам строки s, что составляет O(m) .

Внутренний цикл проходит по всем символам шаблона p, что составляет O(n) .

Таким образом, общее время выполнения алгоритма составит:

O(m·n)



Скрины, что задача или задачи прошли все тесты

6. Пояснение к задаче

Задача состоит в проверке, соответствует ли строка заданному шаблону с учетом специальных символов:

. – соответствует любому одиночному символу,

* – соответствует нулю или более предыдущим символам.

Использование динамического программирования тут оправдано, так как существует множество перекрывающихся подзадач. Для каждой позиции в строке и шаблоне можно использовать ранее вычисленные значения, чтобы принять решение о текущем состоянии. Это значительно улучшает эффективность алгоритма по сравнению с наивным подходом, который мог бы требовать экспоненциального времени.

7. Вывод

Таким образом, применение ДП позволяет решать задачу эффективно, сохраняя состояние и избегая повторного вычисления одних и тех же значений.

Приложение А

```
#include <vector>
#include <string>
class Solution {
public:
    bool isMatch(string s, string p) {
    std::vector<std::vector<bool>> dp(s.size() + 1, std::vector<bool>(p.size() + 1, false));
        dp[0][0] = true;
        for (size_t j = 1; j <= p.size(); ++j) {
            if (p[j - 1] == '*') {
                dp[0][j] = dp[0][j - 2];
            }
        }
        for (size_t i = 1; i <= s.size(); ++i) {
            for (size_t j = 1; j <= p.size(); ++j) {
                if (p[j - 1] == s[i - 1] || p[j - 1] == '.') {
                    dp[i][j] = dp[i - 1][j - 1];
                }
                else if (p[j - 1] == '*') {
                    dp[i][j] = dp[i][j - 2] || (dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'));
                }
            }
        }
        return dp[s.size()][p.size()];
    }
};
```