ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Название лабораторной работы (Задача на динамическое программирование на leetcode)»

Выполнила работу

Смирнова Анна

Академическая группа №J3112

Принято

Преподаватель: Дунаев Максим Владимирович

Санкт-Петербург

2024

1. Введение

Задача на нахождение самых длинных действительных скобок.

Цель: Дана строка, содержащая только символы, необходимо найти длину самой длинной действительной (правильно сформированной) подстроки скобок.

Задачи:

- Задача должна быть решена с помощью динамического программирования

1. Теоретическая подготовка

- Знание языка программирования C++

- Понимание динамического программирования для решения задачи

3. Процесс этапа выполнения работы:

1. Постановка задачи:

Необходимо выяснить, как найти длину самой длинной действительной подстроки скобок.

1. Анализ поставленной задачи:

Для решения необходимо использовать динамическое программирование.

1. Проектирование алгоритма:

int longestValidParentheses(const string& s) {

    int n = s.size();

    if (n == 0) return 0;

    vector<int> dp(n, 0);

    int maxLength = 0;

    // Проходим по строке, начиная со второго символа

    for (int i = 1; i < n; ++i) {

        // Проверяем, является ли текущий символ закрывающей скобкой

        if (s[i] == ')') {

            // Если предыдущий символ - открывающая скобка

            if (s[i - 1] == '(') {

                dp[i] = (i >= 2 ? dp[i - 2] : 0) + 2; // Увеличиваем длину на 2

            }

            // Если предыдущая скобка закрывающая и есть открывающая перед ней

            else if (i - dp[i - 1] > 0 && s[i - dp[i - 1] - 1] == '(') {

                dp[i] = dp[i - 1] + (i >= 2 + dp[i - 1] ? dp[i - 2 - dp[i - 1]] 0) + 2; // Увеличиваем длину на длину предыдущей действительной подстроки плюс 2

            }

            // Обновляем максимальную длину, если текущая больше

            maxLength = max(maxLength, dp[i]);

        }

    }

    return maxLength;

}

4. Экспериментальная часть

Подсчёт по памяти:

Переменные: sizeof(n) + sizeof(maxLength) + sizeof(s) = 4 + 4 + 32 = 38 байт

Массив: sizeof(dp) = 4 \* n байт

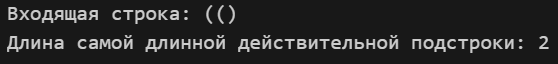
Общая память: 38 + 4 \* n байт

Подсчет асимптотики (только для циклов и сложных структур):

Цикл: O(n)

Общая асимптотика: O(n)

Тестирования:









5. Заключение

В ходе выполнения работы мною был реализован алгоритм, позволяющий найти длину самой длинной действительной подстроки скобок. Цель работы была достигнута путём тестирования алгоритма для разных сочетаний скобок. Для решения использовано ДП потому, что скобочные последовательности могут быть вложенными. ДП позволяет легко учитывать такие вложенности, так как при нахождении действительной подстроки мы можем ссылаться на предыдущие значения в массиве. Также ДП обеспечивает линейную сложность O(n), что также подчёркивает его эффективность. Обычным подходом задача не может быть решена, так как обычные подходы не учитывают вложенность скобок и могут привести к неправильным результатам или чрезмерным вычислениям.

6. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab6.cpp

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// Функция для нахождения длины самой длинной действительной подстроки скобок

int longestValidParentheses(const string& s) {

    int n = s.size();

    if (n == 0) return 0;

    vector<int> dp(n, 0);

    int maxLength = 0;

    // Проходим по строке, начиная со второго символа

    for (int i = 1; i < n; ++i) {

        // Проверяем, является ли текущий символ закрывающей скобкой

        if (s[i] == ')') {

            // Если предыдущий символ - открывающая скобка

            if (s[i - 1] == '(') {

                dp[i] = (i >= 2 ? dp[i - 2] : 0) + 2; // Увеличиваем длину на 2

            }

            // Если предыдущая скобка закрывающая и есть открывающая перед ней

            else if (i - dp[i - 1] > 0 && s[i - dp[i - 1] - 1] == '(') {

                dp[i] = dp[i - 1] + (i >= 2 + dp[i - 1] ? dp[i - 2 - dp[i - 1]] : 0) + 2; // Увеличиваем длину на длину предыдущей действительной подстроки плюс 2

            }

            // Обновляем максимальную длину, если текущая больше

            maxLength = max(maxLength, dp[i]);

        }

    }

    return maxLength;

}

int main() {

    string s = "))(()";

    int result = longestValidParentheses(s);

    cout << "Входящая строка: " << s << endl;

    cout << "Длина самой длинной действительной подстроки: " << result << endl;

    return 0;

}