ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Васькин Виктор

Академическая группа №J3110

Принято

Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель — разработать алгоритм, который позволит найти длину самой длинной корректной подстроки из круглых скобок в заданной строке. Корректная подстрока скобок — это последовательность, в которой каждая открывающая скобка имеет соответствующую закрывающую скобку и расположена в правильном порядке.

Для достижения указанной цели лабораторной работы необходимо решить несколько задач:

1. Изучить алгоритмы, применяемые для нахождения корректных подстрок, с акцентом на динамическое программирование.

2. Реализовать функцию на языке C++, которая будет определять длину самой длинной корректной подстроки из круглых скобок.

3. Провести тестирование разработанного алгоритма на различных примерах, чтобы гарантировать его корректность и эффективность.

4. Проанализировать временные и пространственные характеристики алгоритма, оценив их сложность.

1. Теоретическая подготовка

2.1. Основные понятия и определения

Корректные скобочные последовательности — это последовательности, состоящие из открывающих и закрывающих скобок, которые соблюдают следующие правила:

- Каждая открывающая скобка имеет соответствующую закрывающую.

- Закрывающая скобка не может стоять перед своей открывающей.

2.2. Динамическое программирование

Динамическое программирование — это метод решения задач, который позволяет упростить решение более сложных задач, разбивая их на меньшие подзадачи. Этот подход особенно полезен в рамках задач, где результаты одних подзадач могут использоваться для вычисления результатов других.

Преимущества:

- Повышение эффективности за счет избежания повторных вычислений одной и той же подзадачи.

- Стратегия позволяет относительно простым образом хранить уже вычисленные результаты, используя такие структуры данных, как массивы или векторы.

2.3. Структуры данных

Для реализации задачи о нахождении длины самой длинной корректной подстроки из круглых скобок используются следующие структуры данных:

- Строка (`std::string`): Позволяет хранить и манипулировать входной последовательностью скобок.

- Вектор (`std::vector<int>`): Используется для хранения длины самой длинной корректной подстроки, которая заканчивается на каждом индексе входной строки. Вектор динамически изменяет размер и позволяет удобно хранить данные, что делает его идеальным для данной задачи.

2.4. Временная и пространственная сложность

- Временная сложность: O(n), где n — длина входной строки. Алгоритм проходит по строке один раз, выполняя фиксированное количество операций на каждой итерации.

- Пространственная сложность: O(n) из-за использования вектора, который хранит длины корректных подстрок. Память, затраченная на этот вектор, растет линейно с увеличением длины строки.

2.5. Понимание корректных скобочных последовательностей

Для успешного анализа и обработки строк, состоящих из скобок, важно понимать основные принципы:

- Каждая открывающая скобка (`(`) должна иметь свою закрывающую пару (`)`).

- Неправильное расположение скобок вызывает ошибку в парности.

1. Реализация

3.1. Выбор инструментов

Для реализации задачи были использованы стандартные библиотеки:

- <string> для работы со строковыми данными.

- <vector> для хранения длин корректных подстрок.

- <algorithm> для использования функции std::max\_element, необходимой для нахождения максимального значения в векторе.

3.2. Разработка алгоритма

Процесс разработки алгоритма был разделён на несколько этапов:

1. Анализ задачи: важно было понять, как именно определяются корректные подстроки и как можно использовать динамическое программирование для их нахождения. Были сформулированы условия, в которых подстрока считается корректной.

2. Определение структуры данных: для хранения промежуточных результатов и длин корректных подстрок был выбран вектор, который позволяет динамически изменять размер и эффективно управлять памятью.

3. Написание основного кода:

- Реализация функции longestValidParentheses, которая принимает строку и возвращает длину самой длинной корректной подстроки.

- Основная логика алгоритма основывается на итерации по строке с использованием индексов, проверкой наличия корректных пар скобок, обновлением вектора длин корректных подстрок, а также нахождением максимального значения в нем.

#include <string>

#include <vector>

#include <algorithm>

class Solution {

public:

int longestValidParentheses(std::string s) {

int length = s.size();

std::vector<int> validLengths(length + 1, 0);

for (int i = 2; i <= length; i++) {

if (s[i - 1] == ')') {

if (s[i - 2] == '(') {

validLengths[i] = validLengths[i - 2] + 2;

} else {

int matchingOpenIndex = i - validLengths[i - 1] - 1;

if (matchingOpenIndex > 0 && s[matchingOpenIndex - 1] == '(') {

validLengths[i] = validLengths[i - 1] + 2 + validLengths[matchingOpenIndex - 1];

}

}

}

}

return \*std::max\_element(validLengths.begin(), validLengths.end());

}

};

* 1. Инициализация:
  + Определение длины строки.
  + Создание массива (вектора) для хранения длин корректных подстрок.

3.2 Итерация по строке:

* + Начиная с третьего символа, проверяем каждый символ, чтобы определить, является ли он закрывающей скобкой.
  + Если символ — закрывающая скобка, то:
    - Если перед ней открывающая, обновляем длину последовательности.
    - Если перед ней также закрывающая, ищем соответствующую открывающую скобку, используя ранее сохраненные значения.

3.3 Вывод результата:

* + Находим максимальное значение в векторе, которое будет являться длиной самой длинной корректной подстроки.

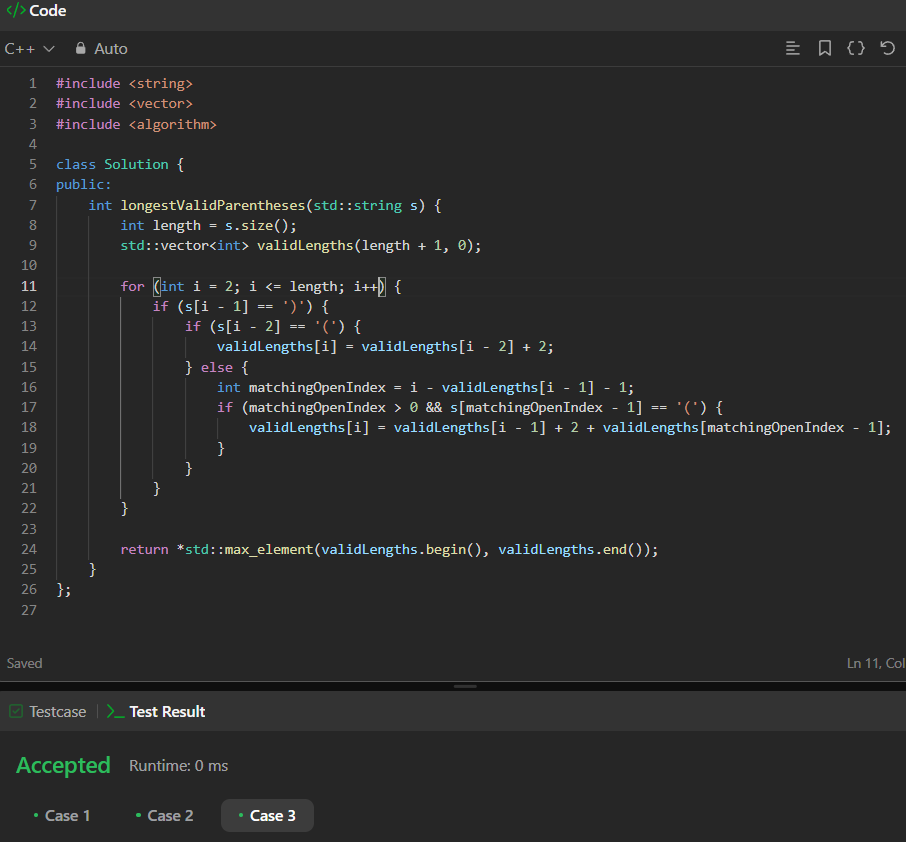
4. Тестирование: после написания кода было проведено тестирование на различных примерах, чтобы убедиться в корректности работы алгоритма. Были использованы тестовые случаи с разными комбинациями скобок, включая пустые строки, строки с одной открывающей или закрывающей скобкой и более сложные вложенные строки.

5. Анализ и обсуждение особенностей реализации

- Эффективность: благодаря использованию динамического программирования время выполнения алгоритма составляет O(n), что позволяет эффективно обрабатывать даже длинные строки. Пространственная сложность — O(n) — также остается в разумных пределах, так как мы используем вектор для хранения длин подстрок.

1. Экспериментальная часть

По завершении реализации алгоритма, он был успешно протестирован на платформе LeetCode. Все тесты, включая различные случаи с различными комбинациями скобок, были пройдены, что подтвердило его корректность и эффективность. Алгоритм продемонстрировал свою надежность и работоспособность, что делает его подходящим для решения поставленной задачи.



Скриншот результатов тестирования

5. Заключение

В ходе выполнения работы был разработан алгоритм для нахождения длины самой длинной корректной подстроки из круглых скобок. Цель работы была достигнута путем реализации алгоритма на языке C++ с использованием концепции динамического программирования, что позволило эффективно решать поставленную задачу.

Разработка алгоритма включала анализ корректных скобочных последовательностей, использование векторов для хранения промежуточных значений и выполнение тестирования на платформе LeetCode, где были пройдены все тесты. Это подтвердило корректность и эффективность предложенного решения.

Полученные результаты совпадают с теоретическими оценками временной и пространственной сложности алгоритма, что свидетельствует о его высоких показателях производительности.

В качестве дальнейших направлений для исследований можно рассмотреть возможность оптимизации алгоритма с точки зрения уменьшения потребления памяти. Также полезным будет исследование других типов скобок или расширение алгоритма на более общие структуры данных, такие как различные виды скобочных последовательностей.