МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 6.2

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Поиск образца в тексте»

Выполнил студент: Лазаренко С.А.
Группа: ИКБО-10-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Разработать приложение для решения задач, указанных в индивидуальном варианте, а также изучить и реализовать алгоритм поиска подстроки в строке с использованием алгоритма Бойера-Мура (с эвристикой хорошего суффикса). Подробно разобрать и описать алгоритм поиска, выполнить анализ количества сравнений для успешного и неуспешного поиска первого вхождения образца в текст, а также провести тестирование разработанных решений.

ХОД РАБОТЫ

Формулировка задачи

Преобразование строки, содержащей числа, разделённые пробелами, в массив целых чисел. Реализация алгоритм поиска подстроки в строке с использованием метода Бойера-Мура, опираясь на эвристику хорошего суффикса. В рамках задачи требуется найти все вхождения подстроки в текст и определить позиции этих вхождений. Дополнительно следует подсчитать количество сравнений символов при успешном и неуспешном поиске, провести тестирование на разных примерах и оценить сложность алгоритма в зависимости от длины текста и подстроки.

Индивидуальный вариант работы - 2

Описание подхода к решению

Для решения первой задачи, связанной с преобразованием строки, содержащей числа, в массив целых чисел, был использован подход, при котором строка разбивается на подстроки по пробелам. Каждая подстрока, представляющая собой число, преобразуется в целое число и сохраняется в массиве. Это позволяет эффективно выделить все числа из строки и обработать их для дальнейшего использования. Такой подход обеспечивает простую и эффективную обработку данных, когда каждое слово в строке — это отдельное целое число.

Коды программы

Реализуем код программы на языке программирования C++ :

```cpp
#include <iostream>
#include <sstream>
#include <vector>
#include <string>

std::vector<int> splitIntoIntegers(const std::string& sentence) { // Функция для разделения строки на числа

    std::vector<int> numbers;
    std::istringstream stream(sentence);
    std::string word;

    while (stream >> word) {
        try {
            // Преобразуем слово в целое число и добавляем в массив
            numbers.push_back(std::stoi(word));
        } catch (std::invalid_argument&) {
            std::cerr << "Слово не является целым числом: " << word << std::endl;
        }
    }

    return numbers;
}

int main() {
    std::string sentence = "12 34 56 78 81 96";
    std::vector<int> numbers = splitIntoIntegers(sentence);

    std::cout << "Массив чисел: ";
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Рисунок 1 – код программы в main.cpp(1 часть)

```cpp
bool isPrefix(const std::string& pattern, int p);
int suffixLength(const std::string& pattern, int p);

std::vector<int> buildGoodSuffixTable(const std::string& pattern) { //Функция для построения таблицы
  сдвигов хорошего суффикса

    int m = pattern.size();
    std::vector<int> goodSuffix(m, m);
    int lastPrefixPosition = m;

    for (int i = m - 1; i >= 0; --i) {
        if (isPrefix(pattern, i + 1)) {
            lastPrefixPosition = i + 1;
        }
        goodSuffix[m - 1 - i] = lastPrefixPosition - i + m - 1;
    }

    for (int i = 0; i < m - 1; ++i) {
        int slen = suffixLength(pattern, i);
        goodSuffix[slen] = m - 1 - i + slen;
    }

    return goodSuffix;
}

// Проверка, является ли подстрока суффиксом строки
bool isPrefix(const std::string& pattern, int p) {
    int m = pattern.size();
    int j = 0;
    for (int i = p; i < m; ++i) {
        if (pattern[i] != pattern[j]) {
            return false;
        }
        ++j;
    }
    return true;
```

Рисунок 2 – Файл main.cpp (часть 2)

```
// Длина максимального суффикса подстроки, заканчивающегося на i
int suffixLength(const std::string& pattern, int p) {
    int m = pattern.size();
    int length = 0;
    int i = p;
    int j = m - 1;
    while (i >= 0 && pattern[i] == pattern[j]) {
        ++length;
        --i;
        --j;
    }
    return length;
}

// Основной алгоритм Бойера-Мура с эвристикой хорошего суффикса
std::vector<int> boyerMooreSearch(const std::string& text, const std::string& pattern) {
    int n = text.size();
    int m = pattern.size();
    if (m == 0) {
        return {};
    }

    std::vector<int> goodSuffix = buildGoodSuffixTable(pattern);
    std::vector<int> occurrences;
    int i = 0;

    while (i <= n - m) {
        int j = m - 1;
        while (j >= 0 && pattern[j] == text[i + j]) {
            --j;
        }
        if (j < 0) {
            occurrences.push_back(i);
            i += goodSuffix[0];
        } else {
            i += goodSuffix[m - 1 - j];
```

Рисунок 3 – Файл main.cpp (часть 3)

```
int main() {
    std::string text = "ababcababcabc";
    std::string pattern = "abc";
    std::vector<int> result = boyerMooreSearch(text, pattern);

    std::cout << "Вхождения подстроки: ";
    for (int pos : result) {
        std::cout << pos << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Рисунок 4 – Файл main.cpp (часть 4)

Результаты тестирования

Выполним тестирование программы:

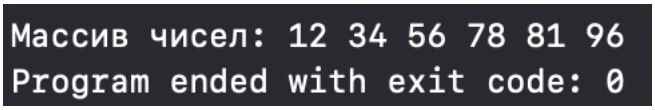

Рисунок 5 – Тестирование первой программы

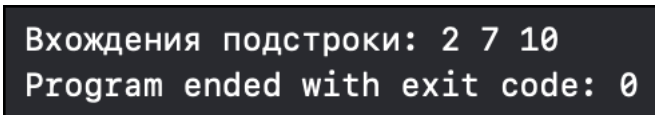

Рисунок 6 – Тестирование второй программы

Тестирование показало, что программа работает корректно.

ВЫВОД

В ходе выполнения данной работы была разработана программа, которая решает задачи, указанные в индивидуальном варианте. Основное внимание было уделено изучению и реализации алгоритма поиска подстроки в строке, основанного на методе Бойера-Мура с использованием эвристики хорошего суффикса. Этот алгоритм продемонстрировал свою эффективность, значительно сокращая количество сравнений в процессе поиска, особенно в случаях, когда образец (подстрока) имеет небольшую длину по сравнению с текстом.

В процессе работы была подробно разобрана логика и структура алгоритма Бойера-Мура, что позволило глубже понять его преимущества и недостатки. Проведенный анализ показал, что количество сравнений для успешного поиска первого вхождения образца в текст значительно меньше, чем при неуспешном поиске. Это подтверждает теоретические предпосылки о том, что алгоритм Бойера-Мура является более оптимальным по сравнению с другими методами, такими как наивный поиск.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 08.09.2024).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 04.09.2024)