МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Построение и анализ алгоритмов»
Тема: Кнут-Моррис-Пратт

Студентка гр. 1304 _____ Чернякова В.А.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для нахождения подстроки в строке и определения циклического сдвига.

Задание.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найдите все вхождения P в T.

Вход:

Первая строка - P

Вторая строка - T

Выход:

Индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1.

Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$). Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и A состоит из суффикса B, склеенного с префиксом B). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - A

Вторая строка - B

Выход:

Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

Основные теоретические положения.

Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

Дана строка s[0…n-1]. Требуется вычислить для неё префикс-функцию, т.е. массив чисел pi[0…n-1], где pi[i] определяется следующим образом: это такая наибольшая длина наибольшего собственного суффикса подстроки s[0…i], совпадающего с её префиксом (собственный суффикс — значит не совпадающий со всей строкой). В частности, значение pi[0] полагается равным нулю.

Выполнение работы.

Для решения задач определения вхождения подстроки в строку был реализован алгоритм Кнута-Морриса-Пратта с использованием значения префиксной функции.

Основные функции:

1) *std::vector<int> computePrefixFunction(std::string pattern)*. Функция, вычисляющая вспомогательную префиксную функцию заданного образца для работы алгоритма Кнута-Морриса-Пратта. На вход подается строка, для которой необходимо вычислить функцию. Такая функция представлена в виде вектора длины равной длине строки, инициализированного изначально нулями. По нулевому индексу значение всегда остается равным нулю. С помощью цикла *for* составляем префиксную функцию. Алгоритм внутри *for* следующий: ищем, какой префикс-суффикс можно расширить. С помощью цикла *while* проверяем, можно ли расширить данный суффикс или же обращаемся к предыдущему. Если значение строки от значения текущего префикс-суффикса и текущей итерации совпали, то расширяем найденный префикс-суффикс. Префиксная функция от текущей итерации равна текущему вычисленному значению. Функция возвращает вектор, хранящий значение префиксной функции.

2) *std::vector<int> findOccurrenceInTextKMP(std::string text, std::string pattern)*. Функция, реализующая алгоритм Кнута-Морриса-Пратта для нахождения всех вхождений шаблона в текст. Принимает на вход текст и шаблон. Находим с помощью функции, описанной выше, префиксную функцию для шаблона и заводим переменную типа *int count_equal_symbols*, где будем

считать количество совпавших символов в тексте и шаблоне. С помощью цикла *for* просматриваем все символы текста. Если значение шаблона изначально не совпало с текстом, то выбираем из префиксной функции предыдущее значение для сравнения. Если значение шаблона по индексу, равному *count_equal_symbols*, равно значению текста от текущей итерации, то увеличиваем количество совпавших символов. Если количество совпавших символов равно длине шаблона, то мы нашли вхождение, в вектор, хранящий индексы вхождения добавляем его, а для поиска следующего совпадения значение *count_equal_symbols* теперь будет равно следующему значению *prefix_function_pattern[count_equal_symbols-1]*, то есть сдвигаем шаблон до такого префикса. Функция возвращает вектор с индексами вхождения.

3) *int checkCycleShiftKMP(std::string first_string, std::string second_string)*. Функция, реализующая алгоритм Кнута-Морриса-Пратта для определения, является ли первая строка циклическим сдвигом второй строки. Принимается на вход первая и вторая строка соответственно. Используется такой же алгоритм Кнута-Морриса-Пратта описанный выше, но с модернизацией. Вначале осуществляется проверка основного условия, когда сдвиг возможен – строки равные по длине. Чтобы алгоритм работал также, как и описанный раннее, необходимо первую строку представить в виде текста, для этого её удваиваем. Далее алгоритм работает также. Но только при первом же нахождении значения работа функции завершается. Функция возвращает первый индекс начала второй строки в первой.

4) *void printFirstTaskAnswer(std::vector<int> finding_index_inclusion)*. Функция, выводящая отформатированный ответ на первое задание. Принимает на вход вектор, хранящий индексы начала вхождений шаблона в текст. Если длина такого вектора не ноль, то выводятся индексы через запятую. Иначе выводится на экран *-1*.

5) *void startFirstTaskSolution()*. Функция, запускающая решение первого задания. С консоли считываются шаблон и текст, запускается работа функции

findOccurrenceInTextKMP(), полученный результат выводится с помощью *printFirstTaskAnswer()*.

6) *void printSecondTaskAnswer(int index_begin_second_in_first)*. Функция, выводящая отформатированный ответ на второе задание. Принимает на вход значение, хранящее индекс начала второй строки в первой. Данное значение выводится на экран.

7) *void startSecondTaskSolution()*. Функция, запускающая решение второго задания. С консоли считываются две строки, запускается работа функции *checkCycleShiftKMP()*, полученный результат выводится с помощью *printSecondTaskAnswer()*.

Разработанный программный код смотреть в приложении А.

Выводы.

В ходе лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта. Разработан программный код, позволяющий решить следующие задачи с помощью данного алгоритма: нахождение индексов вхождения шаблона в текст и определение циклического сдвига в строке. На языке программирования *C++* реализованы функции, представляющий собой решение поставленных задач.

Для работы алгоритма Кнута-Морриса-Пратта была изучена и реализована на языке программирования C++ префиксная функция, которая транслирует строковый образец в представление (массив индексов в строковом образце), удобное для быстрого определения, находится ли целый образец в данном месте строки.

Программа предусматривает возможность отсутствия строки в тексте и циклического сдвига, в соответствии с чем выводится значение, сигнализирующее о данном случае.

Разработанный программный код для решения поставленных задач успешно прошел тестирование на онлайн платформе *Stepik*.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

#define pattern_not_in_text -1
#define index_not_find -1

// Функция, вычисляющая вспомогательную префиксную функцию
// заданного образца для работы алгоритма Кнута-Морриса-Пратта
// Принимает на вход строку, для которой необходимо вычислить функцию
// Возвращает вектор - значение префиксной функции
std::vector<int> computePrefixFunction(std::string pattern) {
    int pattern_length = pattern.size();
    std::vector<int>  prefix_function(pattern_length,0);
    int prefix_value = 0;
    for (int i  = 1; i < pattern_length; i++) {
             while (prefix_value > 0 && pattern[prefix_value] != pattern[i]) {
            prefix_value = prefix_function[prefix_value-1];
        }
        if (pattern[prefix_value] == pattern[i]) {
            prefix_value += 1;
        }
        prefix_function[i] = prefix_value;
    }
    return prefix_function;
}

// Функция, реализующая алгоритм Кнута-Морриса-Пратта
// для нахождения всех вхождений шаблона в текст
// Принимает на вход текст и шаблон
// Возвращает вектор, хранящий индексы начала вхождений шаблона в текст
```

```
    std::vector<int>     findOccurrenceInTextKMP(std::string     text,
std::string pattern) {
        int text_length = text.size();
        int pattern_length = pattern.size();
        std::vector<int> index_inclusion;
                    std::vector<int>    prefix_function_pattern    =
computePrefixFunction(pattern);
        int count_equal_symbols = 0;
        for (int i = 0; i < text_length; i++) {
                            while   (count_equal_symbols   >   0   &&
pattern[count_equal_symbols] != text[i]) {
                                                count_equal_symbols    =
prefix_function_pattern[count_equal_symbols-1];
            }
            if (pattern[count_equal_symbols] == text[i]) {
                count_equal_symbols = count_equal_symbols + 1;
            }
            if (count_equal_symbols == pattern_length) {
                index_inclusion.push_back(i - pattern_length + 1);
                                                count_equal_symbols    =
prefix_function_pattern[count_equal_symbols-1];
            }

        }
        return index_inclusion;
    }

    // Функция, реализующая алгоритм Кнута-Морриса-Пратта для
определения,
    // является ли первая строка циклическим сдвигом второй строки
    // Принимает на вход две строки
    // Возвращает значение - индекс начала второй строки в первой
    int   checkCycleShiftKMP(std::string   first_string,   std::string
second_string) {
        int index_cycle_shift = index_not_find;
        int first_string_length = first_string.size();
        int second_string_length = second_string.size();
        if (first_string_length == second_string_length) {
```

```
                    std::string first_string_modify= first_string +
first_string;
                                int first_string_modify_length =
first_string_modify.size();
                    std::vector <int> prefix_function_second_string =
computePrefixFunction(second_string);
            int count_equal_symbols = 0;
            for (int i = 0; i < first_string_modify_length; i++) {
                                while (count_equal_symbols > 0 &&
second_string[count_equal_symbols] != first_string_modify[i]) {
                                                count_equal_symbols =
prefix_function_second_string[count_equal_symbols-1];
                }
                            if (second_string[count_equal_symbols] ==
first_string_modify[i]) {
                    count_equal_symbols = count_equal_symbols + 1;
                }
                if (count_equal_symbols == second_string_length) {
                    index_cycle_shift = i - second_string_length + 1;
                    return index_cycle_shift;
                }
            }
        }
        return index_cycle_shift;
    }

    // Функция, выводящая отформатированный ответ на 1 задание
    // Принимает на вход вектор, хранящий индексы начала вхождений
шаблона в текст
    void printFirstTaskAnswer(std::vector<int> finding_index_inclusion)
{
        if (finding_index_inclusion.size() != 0) {
             for (int i = 0; i < finding_index_inclusion.size() - 1; i+
+) {
                std::cout << finding_index_inclusion[i]<<",";
            }
            std::cout << finding_index_inclusion.back();
        }
        else {
```

```
            std::cout << pattern_not_in_text;
        }
    }

    // Функция, запускающая решение 1 задания
    void startFirstTaskSolution() {
        std::string pattern, text;
        std::cin >> pattern;
        std::cin >> text;
                    std::vector<int>    finding_index_inclusion    =
findOccurrenceInTextKMP(text, pattern);
        printFirstTaskAnswer(finding_index_inclusion);
    }

    // Функция, выводящая отформатированный ответ на 2 задание
    // Принимает на вход значение, хранящее индекс начала второй строки
в первой
    void printSecondTaskAnswer(int index_begin_second_in_first) {
        if (index_begin_second_in_first == index_not_find) {
            std::cout << index_not_find;
        }
        else {
            std::cout << index_begin_second_in_first;
        }
    }

    // Функция, запускающая решение 2 задания
    void startSecondTaskSolution() {
        std::string first_string, second_string;
        std::cin >> first_string;
        std::cin >> second_string;
                        int     index_begin_second_in_first     =
checkCycleShiftKMP(first_string, second_string);
        printSecondTaskAnswer(index_begin_second_in_first);
    }

    int main() {
        startFirstTaskSolution();
        startSecondTaskSolution();
```

```
    return 0;
}
```