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: Алгоритм Кнута-Морриса-Пратта

Студент гр. 7383 _____ Левкович Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Содержание

Цель работы

Цель работы: познакомиться с алгоритмом Кнута-Морриса-Пратта, написать алгоритм для поиска всех вхождений подстроки в строке и определить, является ли строка циклическим сдвигом другой строки.

Формулировка задачи: Реализуйте алгоритм КМП и с его помощью для заданных шаблона PP (|P|≤15000|P|≤15000) и текста TT (|T|≤5000000) найдите все вхождения P в T.

Заданы две строки A (|A|≤5000000) и B (|B|≤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 - AA

Вторая строка - BB

Выход:

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

Реализация задачи

Программа была написана на языке программирования C++.

Для реализации поставленной задачи был создан класс.

Для первой задачи для начала необходимо склеить шаблон и текст символом разделителем таким, которого нет в алфавите. Далее с помощью префикс функции вычисляется значение префикс функции для каждого символа склеенной строки. Затем с помощью метода K_M_P обходим наш вектор, который содержит значения префикс функции и смотрим на совпадения значения префикс функции с длиной шаблона. Если совпадение есть, значит шаблон находится в тексте, сохраняем индекс в вектор решений.

Для циклического сдвига так же склеиваем две строки с помощью символа разделителя, вычисляем значение префик функции для полученной склеенной строки и начинаем проверять символы для исходных строк, начиная с индекса, который равен значению префикс функции для последнего символа. Если символы не совпадут, значит циклического сдвига нет.

Исходный код программы представлен в приложении Б.

Исследование алгоритма

Было принято исследовать сложность алгоритма

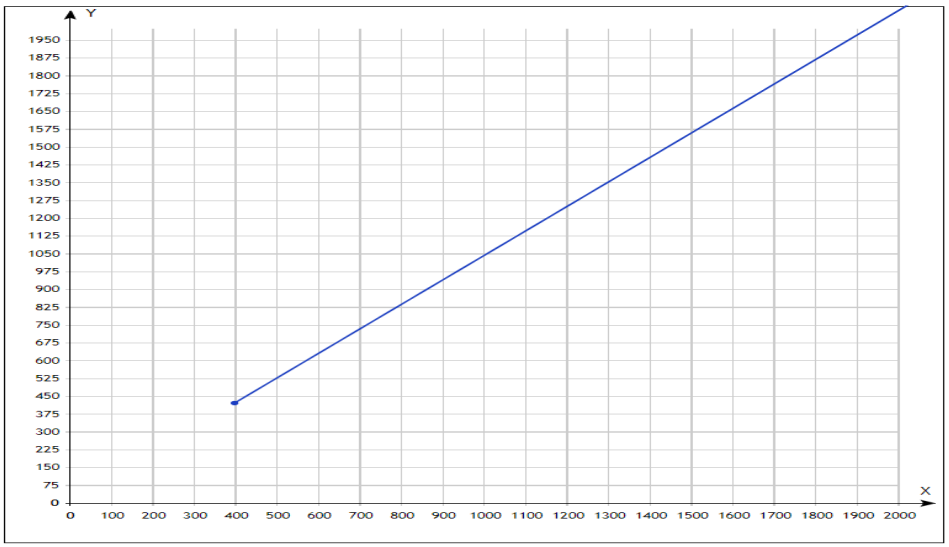

Рисунок 1 – График зависимости количества итераций длины текста и длины шаблона

Длина шаблона	Длина текста	Количество итераций
7	1520	1568
28	1520	1628
7	390	422
4	3754	3844
26	3754	3866

Алгоритм занимает дополнительно память O(m), где m – длина образца, необходимая для хранения значений префикс-функции самого образца.

Тестирование

1. Процесс тестирования

Программа собрана в операционной системе Windows компилятором g++14. В других ОС и компиляторах тестирование не проводилось.

2. Результаты тестирования

В результате тестирования были обнаружены и исправлены ошибки, приводящие к некорректным результатам на некоторых исходных данных. Тестовые случаи представлены в приложении А.

Вывод

В ходе лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта, который позволяет находить все вхождения подстроки в строку используя префикс-функцию и позволяет вычислять циклический сдвиг одной строки относительно другой.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Результат
ab abab	0,2
rt abcrtvsabrthtm uqtyfvu	3,9

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

```
#include <iostream>
#include <vector>
using namespace std;
class A{

public:
    vector<size_t> result;
    /*
int isCicleShift(string A, string B) {
    std::vector<size_t> p = prefix(A, B);
    size_t index = p[p.size() - 1];
    for (int i = index; i < A.size(); i++)
        if (A[i] != B[i - index])
            return -1;

    return index;
}
*/
    vector<size_t>  K_M_P(string A, string B){
        vector<size_t> prefix(A.length());

            size_t last_prefix = prefix[0] = 0;
            for (size_t i=1; i<A.length(); i++) {
                while (last_prefix > 0 && A[last_prefix] != A[i])
                    last_prefix = prefix[last_prefix-1];

                if (A[last_prefix] == A[i])
                    last_prefix++;

                prefix[i] = last_prefix;
            }

            last_prefix = 0;
            for (size_t i=0; i<B.length(); i++) {
                while (last_prefix > 0 && A[last_prefix] != B[i])
                    last_prefix = prefix[last_prefix-1];

                if (A[last_prefix] == B[i])
                    last_prefix++;

                if (last_prefix == A.length()) {
                   result.push_back(i+1-A.length());
                }
            }
            return result;
    }
```

```
};

int main() {
    A a;
    string A, B;
    cin >> A >> B;
    vector<size_t> t = a.K_M_P(A, B);
    if(t.size()>0){
    for(int i = 0;i<t.size()-1;i++){
        cout<<t[i]<<",";
    }
    cout<<t[t.size()-1];
    }else {
        cout<<-1;
}
    return 0;
}
```