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

Студент гр. 0382 _____ Бочаров Г.С.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Написать программу, реализовывающую алгоритм Кнута-Морриса-Пратта, и применить ее для нахождения вхождений подстрок в строку и для определения, является ли строка циклическим сдвигом другой.

Задание.

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

2. 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.

Выполнение работы.

Лабораторная работа выполнена на языке c++.

Задача 1:

Была реализована префикс-функция prefix_funk(). На вход функции подается строка и массив для заполнения.

После выполнения на i-й позиции массива будет записана длина наибольшего префикса в подстроке.

Была реализована функция find_template(), которая реализуюет алгоритм Кнута-Морриса-Пратта для нахождения в строке переданной подстроки.

На вход функция принимает строку, шаблон для поиска, массив префиксов, построенный с помощью функции prefix_funk() и ссылку на результирующий массив вхождений.

Для этого сначала получаем массив префиксов для строки шаблона. Проводится поиск шаблона по строке.

Сначала происходит посимвольное сравнение символов строки и шаблона. В случае несовпадения символов шаблон сдвигается относительно строки вправо на число указанное в массиве префиксов с интедексом на 1 меньше индекса несовпадающего символа.

Задача 2:

В задаче определения является ли строка циклическим сдвигом другой используются та же префиекс-функция.

Для реализации определения циклического сдвига была написана функция find_c(), принимающая на вход две строки.

Сначала проверяется равенство длин строк, в случае совпадения длин, формируются две строки:

Первая строка является склейкой исходных строк (str_1 | str_2) разделенных символом не пренадлежащим к латинскому алфавиту.

Вторая строка тоже является склейкой исходных строк (str_2 | str_1) только теперь строки взяты в обратном порядке.

Для обеих построенных строк строится массив префиксов. Берутся концы массивов префиксов и их сумма сранивается с длинной одной из исходных строк.

Если сумма больше длинны исходных строк, значит строки являются циклическим сдвигом друг друга.

Позицией вхождения будет последний элемент второго массива префиксов.

Разработанный программный код см. в приложении А.

Тестирование.

Таблица 1 – Результаты тестирования для задачи 1

№ п/п	Входные данные	Выходные данные	Комментарии
1	ab abab	0,2	
2	abc abab	-1	Нет вхождения
3	hiha hihihaha	2	
4	fqf fqfqfqfqfqf	0,2,4,6,8	Перекрытие паттерна

Таблица 2 – Результаты тестирования для задачи 2

№ п/п	Входные данные	Выходные данные	Комментарии
1	defabc abcdef	3	
2	ab abc	-1	Разная длина строк
3	abcdef defabc	3	
4	abc abc	0	Одинаковые строки

Выводы.

В ходе работы была написана программа, реализовывающая алгоритм Кнута-Морриса-Пратта, и применена для нахождения вхождений подстрок в строку и для определения, является ли строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4_1.cpp

```
#include <iostream>
#include <set>
#include <vector>
#include <map>
#include <list>
#include <queue>
#include <sstream>
#include <algorithm>


void print_v(std::vector<int>& p)
{
     for (int i = 0; i < p.size(); i++)
          if (i < p.size() - 1)
               std::cout << p[i] << ",";
          else
               std::cout << p[i];
     std::cout << std::endl;
}


void prefix_funk(std::string &t, std::vector<int>& p)
{
     int i = 1;
     int j = 0;

     while (i < t.size())
     {
          if (t[i] == t[j])
          {
               p[i] = j + 1;
               i += 1;
               j += 1;
          }
          else
          {
               if (j == 0)
               {
                    p[i] = 0;
                    i += 1;
               }
               else
                    j = p[j - 1];
          }
     }
}
```

```
     void    find_template(std::string&    s,    std::string&    t,
std::vector<int>& p, std::vector<int>& res)
     {

          int m = t.size();
          int n = s.size();
          int i = 0;
          int j = 0;
          bool f = false;

          while (i < n)
          {
               if (s[i] == t[j])
               {
                    i++;
                    j++;
                    if (j == m)
                    {
                         //std::cout << "finded" << std::endl;
                         res.push_back(i - m);
                         f = true;
                         //break;
                    }

               }
               else
               {
                    if (j > 0)
                    {
                         j = p[j - 1];
                    }
                    else
                    {
                         i++;
                    }
               }
          }

          if (f)
               print_v(res);
          else
               std::cout << -1;


     }



     int main()
```

```
{
	std::string s;
	std::string t;
	std::cin >> t;
	std::cin >> s;

	std::vector<int> p(t.size());
	std::vector<int> res;

	prefix_funk(t, p);
	find_template(s, t, p, res);



	return 0;
}
```

Название файла: lab4_2.cpp

```
#include <iostream>
#include <set>
#include <vector>
#include <map>
#include <list>
#include <queue>
#include <sstream>
#include <algorithm>


void print_v(std::vector<int>& p)
{
	for (int i = 0; i < p.size(); i++)
		if (i < p.size() - 1)
			std::cout << p[i] << ",";
		else
			std::cout << p[i];
	std::cout << std::endl;
}


void prefix_funk(std::string &t, std::vector<int>& p)
{
	int i = 1;
	int j = 0;

	while (i < t.size())
	{
		if (t[i] == t[j])
		{
			p[i] = j + 1;
			i += 1;
			j += 1;
```

```
            }
            else
            {
                if (j == 0)
                {
                    p[i] = 0;
                    i += 1;
                }
                else
                    j = p[j - 1];
            }
        }
    }


    void    find_template(std::string&    s,    std::string&    t,
std::vector<int>& p, std::vector<int>& res)
    {

        int m = t.size();
        int n = s.size();
        int i = 0;
        int j = 0;
        bool f = false;

        while (i < n)
        {
            if (s[i] == t[j])
            {
                i++;
                j++;
                if (j == m)
                {
                    //std::cout << "finded" << std::endl;
                    res.push_back(i - m);
                    f = true;
                    //break;
                }

            }
            else
            {
                if (j > 0)
                {
                    j = p[j - 1];
                }
                else
                {
                    i++;
                }
            }
        }
```

```
     if (f)
          print_v(res);
     else
          std::cout << -1;
}


int find_c(std::string& A, std::string& B) {
     if (A.length() != B.length()) return -1;

     std::vector<int> p1(A.length()*2+1);

     //std::vector<int> p2(A.length() * 2 + 1);

     std::string str = B + "|" + A;

     prefix_funk(str, p1);
     int a = p1.back();

     if (a == 0) return -1;
     if (a % A.length() == 0) return 0;


     std::string  str1 = A + "|" + B;



     prefix_funk(str1, p1);
     int b = p1.back();

     //std::cout << str <<" " << a << std::endl;
     //std::cout << str1 <<" " << b << std::endl;
     if (a + b >= A.length())
          return b;
     return -1;
}



int main()
{
     std::string s;
     std::string t;
     std::string a;
     std::string b = "";

     std::cin >> t;
     std::cin >> s;

     std::cout << find_c(t, s);

     return 0;
```

```
}
```