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 алгоримов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8304 _____ Матросов Д.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм Кнута-Морриса-Пратта, найти индексы вхождения подстроки в строку, а также разработать алгоритм проверки двух строк на циклический сдвиг.

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.

Ход выполнения работы:

На вход алгоритма передается строка-образец, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения.

Алгоритм сначала вычисляет префикс-функцию строки-образца.

Далее посимвольно считывается строка-текст. Переменная-счетчик изначально k = 0. При каждом совпадении k-го символа образца и i-го символа текста счетчик увеличивается на 1. Если k = размер образца, значит вхождение найдено. Если очередной символ текста не совпал с k-ым символом образца, то сдвигаем образец, причем точно знаем, что первые k символов образца совпали с символами строки и надо сравнить k+1-й символ образца (его индекс k) с i-м символом строки.

Для реализации алгоритма были разработаны следующие функции:

- prefix_function – вычисляет префикс функцию шаблона
- KMP – реализация алгоритма Кнута-Морриса-Пратта

Выводы.

В ходе работы был построен и анализирован алгоритм КМП. Код программы представлен в приложении А.

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

using namespace std;

vector<string> make_threads(string str, string subStr,
size_t count)
{
    vector<string> threads;
    unsigned threadSize = str.size() / count;
    if (str.size() % count)
    {
        threadSize += 1;
    }
    unsigned charCount = threadSize + subStr.size();
    for (unsigned i = 0; i < count; i++)
    {
        unsigned offset = i * threadSize;

        if (i < str.size() % count)
        {
            offset -= i;
        }
        else
        {
            offset -= str.size() % count;
        }
        threads.push_back(str.substr(offset, charCount));
    }
```

```
    return threads;
}

vector<size_t> prefix_function(string s)
{
    size_t n = s.length();
    vector<size_t> pi(n);
    for (size_t i = 1; i < n; ++i)
    {
        size_t j = pi[i - 1];
        while ((j > 0) && (s[i] != s[j]))
            j = pi[j - 1];

        if (s[i] == s[j])
            ++j;
        pi[i] = j;
    }
    return pi;
}
//
//std::vector<int>  KMP(std::istream&  input  =  std::cin,
std::ostream& output = std::cout) {
//
//    string Pattern, S;
//    input >> S;
//    input >> Pattern;
//
//    std::vector<size_t> pf = prefix_function(Pattern);
//
// /*   for (int i = 0; i < pf.size(); i++) {
```

```
//        std::cout << pf[i] << " ";
//    }
//    std::cout << std::endl;*/
//
//    std::vector<int> result;
//
//    int mod = 0;
//    int k, i, res;
//
//    for (k = 0, i = 0; i < Pattern.length(); ++i)
//    {
//        while ((k > 0) && (Pattern[i] != S[k]))
//            k = pf[k - 1];
//        if (Pattern[i] == S[k])
//            k++;
//
//        if (k == S.length()) {
//            res = i - S.length() + 1;
//            result.push_back(res + mod);
//        }
//    }
//    return result;
//}

std::vector<int> sufIn(string Pattern, string Suff) {
    int k, i, res;
    std::vector<int> result;

    std::vector<size_t> pf = prefix_function(Pattern);
```

```
    for (k = 0, i = 0; i < Pattern.length(); ++i)
    {
        while ((k > 0) && (Pattern[i] != Suff[k]))
            k = pf[k - 1];
        if (Pattern[i] == Suff[k])
            k++;

        if (k == Suff.length()) {
            res = i - Suff.length() + 1;
            result.push_back(res);
        }
    }
    return result;
}

int KMP(std::istream& input = std::cin, std::ostream&
output = std::cout) {

    string Pattern, Suff, Suff2, Str, tmp;
    input >> Str;
    input >> Pattern;

    std::vector<int> res1;
    std::vector<int> res2;

    int result = -1;

    if (Str.size() != Pattern.size()) return result;

    for (int i = 1; i <= Str.length()-1; i++) {
```

```
            tmp = Str;
            Suff = tmp.erase(i, Str.length());
            tmp = Str;
            Suff2 = tmp.erase(0, i);
            res1 = sufIn(Pattern, Suff);
            res2 = sufIn(Pattern, Suff2);

            if (res2.size() != 0 && res1.size() != 0) {
                for (auto j : res2) {
                    if (j == 0) {
                        for (auto k : res1) {
                            if (k == Suff2.length()) {
                                return k;
                            }
                        }
                    }
                }
            }
        }

        return result;
    }



    int main() {
        int res = KMP();
        std::cout << res;
    }
```