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. 7304 _____ Ажель И.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Изучить и реализовать на языке программирования c++ алгоритм Кнута-Морриса-Пратта, который осуществляет поиск подстроки в строке.

Формулировка задания.

1) 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.

2) 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.

Теоретические сведения.

Префикс-функция от строки S и позиции i в ней — длина k наибольшего префикса подстроки S[1..i], который одновременно является суффиксом этой подстроки. То есть в начале подстроки S[1..i] длиной i нужно найти такой префикс максимальной длины k<i, который был бы суффиксом данной подстроки S[1..k]==S[(i-k+1)..i].

Например, для строки «abcdabscabcdabia» префикс-функция будет такой: π(abcdabscabcdabia)=«0000120012345601».

Вычисление префикс-функции.

Значения префикс-функции вычисляются по очереди: от i=1 до i=n-1 (значение π[0] = 0). Для вычисления значения π[i] создаётся переменная j, обозначающая длину текущего рассматриваемого образца. Изначально j = π[i-1]. Тестируется образец длины j. Сравниваются символы s[j] и s[i]. Если они совпадают — то π[i] = j+1. Переход к следующему индексу i+1. Если же символы отличаются, то уменьшаем длину j, полагая её равной π[j-1], и снова сравниваются s[j] и s[i]. Если j = 0, то останавливается процесс перебора образцов, π[i]=0 и переход к следующему индексу i+1.

Алгоритм.

1) КМП.

Шаг 1: конкатенация S+@+T, где S – подстрока, T – текст, @ – символ разделитель, + – конкатенация.

Шаг 2: вычисляются значения префикс функции для этой строки.

Шаг 3: проход по строке и просмотр значений префикс функции, как только встречается значение, равное длине подстроки, то этот символ является последним символом вхождения подстроки в строке, алгоритм заканчивает работу, когда заканчивается строка.

2) Циклический сдвиг.

Шаг 1: конкатенация A+@+B, где A – первая строка, B – вторая строка, @ – символ разделитель, + – конкатенация.

Шаг 2: смотрится значение префикс функции для последнего элемента этой строки.

Шаг 3: если это значение равно длине строки A, то выводится индекс начала строки B в A = 0 и заканчивается работа. Шаг 4: конкатенация B+@+A.

Шаг 5: смотрится значение префикс функции для последнего элемента этой строки.

Шаг 6: складываются значения, полученные на шаге 2 и 5.

Шаг 7: если это значение равно длине строки A, то значение, полученное на шаге 2 будет искомым индексом, иначе строка B не является циклическим сдвигом A.

Пример работы программы.

1) КМП.

Входные данные:

ab
abab

Выходные данные:

0,2

2) Циклический сдвиг.

Входные данные:

defabc
abcdef

Выходные данные:

3

Выводы.

В ходе выполнения данной лабораторной работы были изучен и реализован на языке программирования c++ алгоритм Кнута-Морриса-Пратта, результатом работы которого являются индексы вхождений подстроки в тексте. Для работы этого алгоритма понадобилась префикс-функция, которая вычисляла насколько нужно сместить счётчик L, который показывает с каким символом из подстроки мы работаем, при расхождении символов из строки и подстроки. Сложность данного алгоритма линейна и пропорциональна сумме длин строки и подстроки O(n + m).

Приложение А.

Исходный код.

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

using namespace std;

vector<int> prefixF(string str)
{
    size_t n = str.size();
    vector<int> result(n);
    for(int i = 1, j = 0; i < n; i++)
    {
        while((j > 0) && (str[i] != str[j]))
            j =  result[j-1];
        if(str[i] == str[j])
            j++;
        result[i] = j;
     }
     return result;
}

void searchKMP(string substr, string str)
{
    vector<int> P = prefixF(substr + "@" +
    str); size_t n = substr.size(); bool flag =
    false;
    for(size_t i = n + 1; i < n + str.size() + 1; i++)
    {
        if(P[i] == n)
        {
            if(flag)
                cout << ",";
            cout << (i - (n + 1) - (n - 1));
            flag = true;
        }
    }
    if(!flag)
        cout<< -1 <<endl;
}

void Cycle(string A, string B)
{
    int index = prefixF(A + "@" +
    B).back(); if(index == A.size())
        cout << 0 << endl;
    else
        if(index + prefixF(B + "@" + A).back() == A.size())
            cout << index << endl;
        else
            cout << -1 << endl;
}

int main()
{
    string substr;
    string str;
    getline(cin, substr);
    getline(cin, str);
    //searchKMP(substr, str);
```

```
    Cycle(substr, str);
    return 0;
}
```