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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 7383 _____ Васильев А.И.

Преподаватель _____ Жангиров Т.Р.

г. Санкт-Петербург

2019

Цель работы.

Исследовать и реализовать задачу поиска вхождения подстроки в строке, используя алгоритм Кнута-Морриса-Пратта.

Формулировка задачи.

Необходимо разработать программу, которая реализует алгоритм Кнута-Морриса-Пртта и с его помощью для заданных шаблона P ($|P|\leq 15000$) и текста T ($|T|\leq 5000000$) найти 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 .

Реализация задачи.

Описание функций и их параметров представлено в таблице 1.

Таблица 1 — Описание функций.

Функция	Описание функции	Список и описание параметров
Int main()	Функция считывает шаблон *P* и текст *T*. Затем создает вектор find, в котором будут хранится индексы начал вхождений подстроки в строку. После этого в функции реализованы КМП алгоритм, описанный ниже и вывод результатов на экран.	-
Int* prefix (string s)	Функция определяет значение префик-функции для каждого символа переданной ей строки и записывает эти значения в массив pi.	String s — строка-шаблон, в котором необходимо установить значение префикс-функции для каждого символа. Функция возвращает массив pi значений префикс-функции.

Алгоритм вычисления префикс-функции: функция находит наибольшую длину наибольшего собственного суффикса подстроки, совпадающего с ее префиксом. Значение для первого символа всегда принимается за 0.

Алгоритм Кнута-Морриса-Пратта: когда вычислены все значения префикс-функций для шаблона, начинается поиск вхождения этого шаблона в текст: начинается сравнение символов с начала в обеих строках, если символы равны - продолжается сравнение пока не находится первый символ, не совпадающий с символом в первой строке (шаблоне) или пока не встречается конец этой строки. Если дошли до конца первой строки, значит первый индекс вхождения первой строки (шаблона) во вторую (текст) найден и вычисляется этот индекс, запоминается этот символ (сохраняется в массив find) и продолжается операция сравнения для следующего символа во второй строке и для последнего символа, лежащего по адресу, хранящемуся для данного символа в массиве префикс-функций, в первой строке.

Код программы представлен в приложении Б.

Тестирование.

Программа собрана в операционной системе Linux Ubuntu 18.04 компилятором g++. В других ОС и компиляторах тестирование не проводилось. В результате тестирования случаев некорректной работы программы не выявлено. Тестовые случаи представлены в приложении А.

Исследование.

В начале работы программа вычисляет значения префикс-функций для каждого символа первой строки. Примем за P длину первой строки (шаблона). После начинается сравнение каждого символа второй строки с символами первой строки. Примем за T длину второй строки. Тогда сложность алгоритма по времени будет равна O(P+T).

По памяти сложность алгоритма составляет O(P), так как вычисляются значения префикс-функций только для первой строки.

Вывод.

В ходе выполнения лабораторной работы была решена задача поиска подстроки в строке с помощью алгоритма Кнута-Морриса-Пратта на языке C++, и исследован алгоритм Кнута-Морриса-Пратта. Полученный алгоритм имеет линейную сложность как по памяти, так и по времени.

ПРИЛОЖЕНИЕ А.

Тестовые случаи.

Тестовые случаи представлены в табл. 2.

Таблица 2 — Тестовые случаи.

№	Входные данные	Выходные данные
1	abcde abcdaeadadabcdeabcd	8
2	324 324324543324	0,3,9
3	3245698 324	-1
4	abc abc	0

ПРИЛОЖЕНИЕ Б.

Исходный код программы.

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

using namespace std;

int * prefix(string s)
{
    int *pi= new int[s.length()];
    pi[0]=0;
    int j = 0;
    for (int i=1; i < s.length(); i++ )
    {
        if (s[i] == s[j])
        {
            pi[i] = j+1;
            j++;
        }
        else if (j==0)
                pi[i] = 0;
            else
            {
                j = pi[j-1];
                i--;
            }
    }
    return pi;
}

int main()
{
    string p, t; //p-podstring
```

```
    cin >> p;
    cin >> t;
    vector <size_t> find;
    int * pi = prefix(p);
    for (int k = 0, l = 0; k < t.length(); k++)
    {
        if (t[k] == p[l])
        {
            l++;
            if (l == p.length())
                find.push_back(k + 1 - p.length());
        }
        else if(l != 0)
        {
            l = pi[l - 1];
            k--;
        }
    }

    if(find.empty())
        cout << "-1";
    else {
        for (int i = 0; i < find.size()-1; i++)
            cout << find[i] << ",";
        cout << find[find.size() - 1];
    }

    return 0;
}
```