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. 9383 _____ Нистратов Д.Г.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для нахождения вхождения строки.

Задание.

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.

Описание работы алгоритма.

Поиск вхождений в строку:

Шаг 1. Высчитывается префикс функция для строки текста.

Шаг 2. Инициализируется две переменные, выполняющие роль позиций символов двух строк.

Шаг 3. Осуществляется обход по строке текста и осуществляется поиск первого вхождения заданной строки для поиска. Если на она не была найдена, то позиция высчитывается по префикс функции. Если была найдена, то осуществляется проверка верности строки и позиция записывается в ответ.

Шаг 4. Вывод позиций вхождения строки в терминал.

Анализ алгоритма.

Сложность префикс функции O(n), где n длинна строки.

Сложность алгоритма Кнута-Морриса-Пратта для поиска вхождения в строку O(n + m), где n – длинна строки, а m – префикс функция.

Сложность алгоритма Кнута-Морриса-Пратта для поиска циклического сдвига O(2n + m), где n – длинна строки, а m – прификс функция.

Описание основных функций.

std::vector<int> prefix_function (std::string s) – высчитывает значение префикс функции для заданной строки.

std::vector<int> KMP(std::string& P, std::string& T) – алгоритм Кнута-Морриса-Пратта, возвращающий позиции вхождения подстроки в строку.

Тестирование.

Тесты проведены с помощью сторонней библиотеки catch2 и описаны в файле test_1.cpp, для поиска вхождения строки, и test_2.cpp, для нахождения циклических сдвигов.
Проверяются две основные функции алгоритма, создание префикс функции и алгоритм Кнута-Морриса-Пратта. Примеры тестов см. в таблице 1 и таблице 2.

Таблица 1. Результаты тестирования для поиска вхождения строки

Ввод	Вывод
abcd abcd	0
de abcdef	3
abacad	0,2,4

Таблица 2. Результаты тестирования для нахождения циклических циклов

Ввод	Вывод
abcd efgfs	-1
bcdefa abcdef	1
defabc abcdef	3

Выводы.

При выполнении работы был изучен и реализован алгоритм Кнута-Морриса-Прата для поиска вхождения строки в текст. На основе данного алгоритма был описан поиск позиций вхождений подстроки в строку, а также поиск циклических сдвигов. Поиск вхождения подстроки в строку реализован со сложностью O (n + m), а поиск циклических сдвигов со сложностью O (2n + m).

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

Название файла: main_1.cpp

```
#include "lab4_1.hpp"

int main(){
    std::string P, T;
    std::cin >> P >> T;
    auto answer = KMP(P, T);
    if (!answer.size()){
        std::cout << -1 << std::endl;
    }
    for (auto symbol : answer){
        if (symbol != answer.back()) std::cout << symbol << ',';
        else std::cout << symbol << std::endl;
    }
    return 0;
}
```

Название файла: lab4_1.cpp

```
#include "lab4_1.hpp"

std::vector<int> prefix_function (std::string s){
    size_t n =  s.length();
    std::vector<int> 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::string& P, std::string& T){
    std::vector<int> answer;
    std::vector<int> pi = prefix_function(P);
    int l = 0;
    for (size_t k = 0; k < T.size(); k++) {
        if (T[k] == P[l]) {
            l++;
            if (l == P.size()) {
                answer.push_back(k + 1 - P.size());
            }
        }
        else if (l != 0){
            l = pi[l-1];
            k--;
        }
    }
    return answer;
}
```

Название файла: lab4_1.hpp

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

std::vector<int> prefix_function (std::string s);
std::vector<int> KMP(std::string& P, std::string& T);
```

Название файла: main_2.cpp

```
#include "lab4_2.hpp"

int main(){
    std::string P, T;
    std::cin >> P >> T;

    if (P == T){
```

```
        std::cout << 0 << std::endl;
        return 0;
    }

    if (P.size() != T.size()){
        std::cout << -1 << std::endl;
        return 0;
    }

    auto answer = KMP(P, T);

    if (!answer.size()){
        std::cout << -1 << std::endl;
        return 0;
    }

    for (auto symbol : answer){
        if (symbol != answer.back()) std::cout << symbol << ',';
        else std::cout << symbol << std::endl;
    }

    return 0;
}
```

Название файла: lab4_2.cpp

```
#include "lab4_2.hpp"

std::vector<int> prefix_function (std::string s){
    size_t n =  s.length();
    std::vector<int> 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::string& P, std::string& T){
    P += P;
    std::vector<int> answer;
    std::vector<int> pi = prefix_function(T);
    int l = 0;
    for (size_t k = 0; k < P.size(); k++) {
        if (P[k] == T[l]) {
            l++;
            if (l == T.size()) {
                answer.push_back(k + 1 - T.size());
            }
        }
        else if (l != 0){
            l = pi[l-1];
            k--;
        }
    }
    return answer;
}
```

Название файла: lab4_2.hpp

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

std::vector<int> prefix_function (std::string s);
std::vector<int> KMP(std::string& P, std::string& T);
```