МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 9383 Мосин К.К.

Преподаватель Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Найти все вхождения подстроки в строку и определение циклического сдвига, используя алгоритм Кнута-Морриса-Пратта.

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.

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.

Выполнение работы.

Алгоритм Кнута-Морриса-Пратта использует результаты префикс-функции от подстроки. На каждой итерации происходит проверка символов строки и подстроки. В случае несовпадения, индекс подстроки перемещается на значение префикс функции для предыдущего символа и сравнивается с текущим символом строки.

Для поиска циклического сдвига, первая строка удваивается и происходит поиск вхождения второй строки в первую, индекс на выходе которого является ответом для циклического сдвига.

Тестирование.

Тестировались стандартные входные данные, пустые строки, а также мусор на входе. Результаты алгоритма поиска подстроки в строке представлены в таблице 1 и алгоритма проверки циклического сдвига в таблтце 2.

Табл. 1 - Результаты тестирования поиска подстроки

Входные данные	Выходные данные
a aba	0, 2
a bbbbbbbbbb	-1
∅ abb	-1
aas#11f1___\n\\\ aasbbbbassasbbsb	-1

Табл. 2 - Результаты тестирование циклического сдвига

Входные данные	Выходные данные
a aba	-1
aaa aaa	0
aaaaaaaaaaaaaas saaaaaaaaaaaaaa	14
∅ ∅	0

Анализ алгоритма.

Значения префикс функции высчитываются за линейное время O(n), где n - длинна подстроки. Обход основной строки выполняется один раз, откуда затраченное время тоже линейно и равно O(m), где m - длинна строки. Общее время составляет O(n+m).

Вывод

В ходе выполнения лабораторной работы был реализован алгоритм Кнута-Морриса-Пратта. Также удалось поработать с префикс-функцией, которая возвращает префикс и одновременно суффикс для текущей строки. Выяснилось, что алгоритм поиска подстроки позволяет узнать циклической сдвиг одной строки от другой.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: API.h

```
#pragma once

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

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

Название файла: API.cpp

```
#include "API.h"

std::vector<int> prefix_function(std::string s) {
    if (s.empty()) {
        return {-1};
    }

    std::vector<int> p(s.size());
    p[0] = 0;
    for (int i = 1; i < s.size(); ++i) {
        int k = p[i - 1];
        while (k > 0 && s[i] != s[k]) {
            k = p[k - 1];
        }
        if (s[i] == s[k]) {
            k++;
        }
```

```
            p[i] = k;
        }
        return p;
}

std::vector<int> KMP(std::string P, std::string T) {
    std::vector<int> indices;
    std::vector<int> prefix = prefix_function(P);
    for (int i = 0, j = 0; i < T.size();) {
        if (T[i] != P[j]) {
            if (j == 0) {
                i++;
            }
            j = prefix[j - 1];
        }
        else if (j == P.size() - 1) {
            indices.push_back(i - P.size() + 1);
            j = prefix[j - 1] + 1;
            ++i;
        }
        else {
            ++i;
            ++j;
        }
    }
    return indices;
}
```

Название файла: lab4_1.cpp

```
#include "API.h"
```

```
int main(int argc, char *argv[]) {
    std::string P, T;
    std::cout << "substring: ";
    std::cin >> P;
    std::cout << "text: ";
    std::cin >> T;

    std::vector<int> indices = KMP(P, T);
    if (indices.empty()) {
        std::cout << -1 << std::endl;
    }
    else {
        for (auto index : indices) {
            std::cout << index;
            if (index != indices.back()) {
                std::cout << ',';
            }
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Название файла: lab4_2.cpp

```
#include "API.h"

int main(int argc, char *argv[]) {
    std::string P, T;
```

```
    std::cout << "first string: ";
    std::cin >> P;
    std::cout << "second string: ";
    std::cin >> T;

    if (P == T) {
        std::cout << 0 << std::endl;
    }
    else if (P.size() != T.size()) {
        std::cout << -1 << std::endl;
    }
    else {
        P += P;
        std::vector<int> indices = KMP(T, P);
        if (indices.empty()) {
            std::cout << -1 << std::endl;
        }
        else {
            std::cout << indices[0] << std::endl;
        }
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ В
ТЕСТЫ

Название файла: test.cpp

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

#include "API.h"

TEST_CASE("Test prefix function") {
    std::vector<int> indices;
    SECTION("default values") {
        indices = {0, 0, 1, 2, 0, 1, 2, 3};
        REQUIRE(prefix_function("ababbaba") == indices);

        indices = {0, 0, 1, 2, 3, 4};
        REQUIRE(prefix_function("ababab") == indices);

        indices = {0};
        REQUIRE(prefix_function("a") == indices);
    }
    SECTION("empty value") {
        indices = {-1};
        REQUIRE(prefix_function("") == indices);
    }
    SECTION("garbage input") {
        indices = {0, 0, 0, 0, 0, 0};
        REQUIRE(prefix_function("\naaa\b\t") == indices);
        REQUIRE(prefix_function("\'\"\\A@*") == indices);
    }
}
```

```
TEST_CASE("Test KMP algorithm") {
    std::vector<int> indices;
    SECTION("default values") {
        indices = {0, 2, 7, 10};
        REQUIRE(KMP("ab", "ababaaaabaab") == indices);

        indices = {1};
        REQUIRE(KMP("a", "sas") == indices);

        REQUIRE(KMP("a",   "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb").empty()
== true);
    }
    SECTION("first arg is greater than second") {
        REQUIRE(KMP("ababa","a").empty() == true);
    }
    SECTION("empty values") {
        REQUIRE(KMP("", "a").empty() == true);
        REQUIRE(KMP("a", "").empty() == true);
        REQUIRE(KMP("", "").empty() == true);
    }
    SECTION("garbage input") {
        REQUIRE(KMP("aas#11f1__\n\\\'",
"aaaaaaaasaaaaaaababababbbsbs\n\t").empty() == true);

        indices = {1};
        REQUIRE(KMP("as", "\nas") == indices);

        indices = {12};
        REQUIRE(KMP("as", "\n\t____\n\b\t\a\a\nas\n@@@@a@sa\ns") == indices);
```

```
    }
}
```