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

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Изучить принцип работы алгоритмов на графах (алгоритм Кнута — Морриса — Пратта). Применить знания на практике, решив поставленную задачу.

Основные теоретические положения.

Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

Задание.

А. Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P| 15000|P|≤15000) и текста T (|T| 5000000|T|≤5000000) найдите все вхождения P в T.

Вход:

Первая строка - P

Вторая строка - T

Выход:

индексы начал вхождений в T, разделенных запятой, если P не входит в T, то вывести −1.

Б. Заданы две строки A (|A| 5000000|A|≤5000000) и B (|B| 5000000|B|≤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.

Вход:

Первая строка - A

Вторая строка - B

Выход:

Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

Выполнение работы.

vector<int> Function_pre(string s) – функция, вычисляющая префикс-функцию поданной на вход строки непосредственно по определению, сравнивая префиксы и суффиксы строк. Первый элемент массива по определению алгоритма, всегда равен нулю. Затем запускается цикл for, проходящий по символам поданной строки. Ищем, какой префикс-суффикс можно расширить. Вычисляем длину предыдущего префикса-суффикса. Пока расширить нельзя, берём длину меньшего префикса-суффикса. Если совпадение, то расширяем найденный префикс-суффикс. Итерация цикла for заканчивается. Возвращает массив значений префикс-функции для каждого символа строки.

А. *int KMP(string text, string pattern)* – функция, исполняющая алгоритм КМП. Сначала массиву (*vector<int> pi = Function_pre(pattern))* присваиваются значения префикс-функции подстроки. Далее в цикле for происходит посимвольное сравнение искомой подстроки со строкой, пока не достигнут последний символ строки. При совпадении всех символов индекс вхождения успешно записывается в массив (массив *int pos[MAX],* хранящий индексы вхождения подстроки в строку). При несовпадении, сравнение происходит со сдвигом тот – символ подстроки под индексом префикс-функции предыдущего символа.

Б. *int KMP(string text, string pattern*) – функция, исполняющая алгоритм КМП с модификациями, необходимыми для выполнения задания. Алгоритм теперь находит только первое «вхождение» подстроки в строку. Чтобы выявить цикличный сдвиг, дополнительно объявляется логическая переменная (*bool continiud = true*). С помощью неё, когда исходная строка подходит к концу, меняется индекс так, что мы рассматриваем её с начала. При этом, «прыжок» к началу строки выполняется только один раз, так как сразу после него флаг *continiud* устанавливается в *false*.

Код выполнен с выводом промежуточных данных.

Пример:

Входные данные для задания Б:

defabc

abcdef

KMP demonstration:

Function_pre demonstration:

d -- 0

e -- 0

f -- 0

a -- 0

b -- 0

c -- 0

d e f a b c

0 0 0 0 0 0

Pattern: defabc

Text: abcdef

Text[0] Pattern[0] ? Text[1] Pattern[0] ? Text[2] Pattern[0] ? Text[3] Pattern[0] ? Found a match!

Text[4] Pattern[1] ? Found a match!

Text[5] Pattern[2] ? Found a match!

Text[0] Pattern[3] ? Found a match!

Text[1] Pattern[4] ? Found a match!

Text[2] Pattern[5] ? Found a match!

Result:

3

Тестирование.

Тестирование проводилось с помощью встроенной библиотеки <*cassert*>. Результат не отображён в таблицах, так как команда *assert* предназначена для отлова ошибок, и выводит результат на экран только в случае провала теста.

Были написаны 3 проверяющих функции.

Работа *assert'a* заключается в следующем*:* в тесте создаются новые исходные данные. Далее assert'у передаётся результат функции и предполагаемый (верный) результат. При несовпадении на экран выводится сообщение. Пример:

Assertion failed: find('e') == 5, file

C:/Users/Serg/CLionProjects/untitled1/main.cpp, line 39

Тест *TestFunction_pre()*проверяет функцию *Function_pre*() следующим образом: задаётся массив значений, затем в *assert* передаётся этот массив и результат работы проверяемой функции, с исходными данными такими, чтобы массив значений являлся верным ответом.

Аналогично работает тест *TestKMP*(), проверяющий работу алоритма Кнута — Морриса — Пратта, для задания А. Тест для задания Б работает проще: там в assert передаётся равенство результата работы функции и значение циклического сдвига строк.

В конце каждого теста есть вывод строки – результата. Ведь если *assert* не вызвал ошибку, то программа выполняется дальше, и печатается строка, что определённый тест пройден.

Результат:

Function Function_pre: Test OK

Function KMP: Test OK

Process finished with exit code 0

Выводы.

В результате работы была написана программа, решающая поставленную задачу при использовании изученных теоретических материалов. Программа было протестирована, результаты тестов удовлетворительны.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл lb4_1.cpp

```
#include <iostream>
#include <vector>
#include <cassert>
#define MAX 55000000
using namespace std;
bool continiud = true;
int pos[MAX];
int ptr = 0;

vector<int> Function_pre(string s)
{vector<int> p(s.length());
    cout << "Function_pre demonstration:\n";
    p[0] = 0;
    cout << s[0] << " -- " << 0 << "\n";
    for (int i = 1; i < (s.length()); i++) {
        cout << s[i] << " -- ";
        int cur = p[i - 1];
        while (s[i] != s[cur] && cur > 0){
            cur = p[cur - 1];
            cout << "string[" << i << "] != string[" << cur << "] -- ";
        }

        if (s[i] == s[cur]){
            p[i] = cur + 1;
            cout << "string[" << i << "] == string[" << cur << "] -- ";
        }
        cout << cur << '\n';
    }
    for (auto i: s) {
        cout << i << ' ';
    }
    cout << '\n';
    for (auto k: p){
        cout << k << ' ';
    }
    cout << endl;
    return p;
}

void TestFunction_pre(){
    vector<int> p = {0,0,1,2,3,4,0,1,2};
    assert(Function_pre("rgrgrgyrg") ==  p);
    vector<int> p2 = {0,0,0,0,0,1,0,1,0,0,1,0};
    assert(Function_pre("abccbanasnas") ==  p2);
    vector<int> p3 =
{0,0,0,0,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,1,2,3,4,1,2,1,2,1,2,3,4,0,0};
    assert(Function_pre("qwertqwerhqweryqwerqwerqwqwqwerlk") ==  p3);
```

```
    cout << endl << "Function Function_pre: Test OK" << endl;
}

int KMP(string text,  string pattern)
{ ptr = 0;
    cout << "KMP demonstration:\n";
    vector<int> pi = Function_pre(pattern);
    int index = 0;
    cout  << "Pattern: " << pattern << "\n";
    cout  << "Text: "<< text << "\n";
    for (int i = 0; i < text.length(); i++)
    {cout  << "Text[" << i << "]  Pattern[" << index  << "] ? ";
        while (index > 0 && pattern[index] != text[i]) {
            index = pi[index - 1];
            cout  << " != \n";}
        if (pattern[index] == text[i]){
            index++;
            cout  << " Found a match!" << "\n";
        }
        if (index == pattern.length())
        {
            cout <<"Found a word! = " << i - index + 1 << "\n";
            cout <<"!!!!!!!!!!!!!!!!!\n";
            pos[ptr++] =  i - index + 1;
        }
    }
}

void TestKMP(){
    int realpos[] = {0,5,10,15,19,27};
    KMP("qwertqwerhqweryqwerqwerqwqwqwerlk", "qwer");
    assert( pos[0] == realpos[0] and pos[1] == realpos[1]
                                    and pos[2] == realpos[2]
                                        and pos[3] == realpos[3]
                                            and  pos[4] == realpos[4]
                                                 and  pos[5] == realpos[5]);

    KMP("qwertqwerhqweryqwerqwerqwqwqwerlk", "qwerty");
    assert(ptr ==  0);
    KMP("abcabcabc", "abc");
    assert(pos[0] ==  0 and pos[1] == 3 and pos[2] == 6);
    cout << endl << "Function KMP: Test OK" << endl;
}

int main() {
    string word,text;
    char choose;
    cout << "Choose an action:" << endl;
    cout << "d - enter data" << endl;
    cout << "t - run tests" << endl;
    std::cin >> choose;
    if (choose == 'd'){
```

```
        std::cin >> word;
        std::cin >> text;
        KMP(text,word);
        for(int i = 0; i < ptr; i++)
        {
            if (i) std::cout << ',';

            cout <<pos[i];
        }
        if (ptr == 0) cout << -1;
        return 0; }
    else if (choose == 't'){
        TestFunction_pre();
        TestKMP();
    }
    return 0;
}
```

Файл lb4_2.cpp

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

using namespace std;
bool continiud = true;


vector<int> Function_pre(string s)
{vector<int> p(s.length());
    cout << "Function_pre demonstration:\n";
    p[0] = 0;
    cout << s[0] << " -- " << 0 << "\n";
    for (int i = 1; i < (s.length()); i++) {
        cout << s[i] << " -- ";
        int cur = p[i - 1];
        while (s[i] != s[cur] && cur > 0){
            cur = p[cur - 1];
            cout << "string[" << i << "] != string[" << cur << "] -- ";
        }

        if (s[i] == s[cur]){
            p[i] = cur + 1;
            cout << "string[" << i << "] == string[" << cur << "] -- ";
        }
        cout << cur << '\n';
    }
    for (auto i: s) {
        cout << i << ' ';
    }
    cout << '\n';
    for (auto k: p){
        cout << k << ' ';
```

```
    }
    cout << endl;
    return p;
}

void TestFunction_pre(){
    vector<int> p = {0,0,1,2,3,4,0,1,2};
    assert(Function_pre("rgrgrgyrg") ==  p);
    vector<int> p2 = {0,0,0,0,0,1,0,1,0,0,1,0};
    assert(Function_pre("abccbanasnas") ==  p2);
    vector<int> p3 =
{0,0,0,0,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,1,2,3,4,1,2,1,2,1,2,3,4,0,0};
    assert(Function_pre("qwertqwerhqweryqwerqwerqwqwqwerlk") ==  p3);
    cout << endl << "Function Function_pre: Test OK" << endl;
}

int KMP(string text,  string pattern)
{continiud = true;
    cout << "KMP demonstration:\n";
    vector<int> pi = Function_pre(pattern);
    int index = 0;
    int i = 0;
    cout  << "Pattern: " << pattern << "\n";
    cout  << "Text: "<< text << "\n";
    for (int i = 0; i < text.length(); i++)
    { cout  << "Text[" << i << "]  Pattern[" << index  << "] ? ";
        while (index > 0 && pattern[index] != text[i]) {
            index = pi[index - 1];
            cout  << " != \n";}
        if (pattern[index] == text[i]) {
            index++;
            cout  << " Found a match!" << "\n";
        }
        if (index == pattern.length())
        {
            cout  << "Result:" << "\n";
            return abs((i - index + 1));
        }
        if (i + 1 == text.length() and continiud){
            i = -1;
            continiud = false;
        }
    }
    return -1;
}

void TestKMP(){
    assert(KMP("abcabc", "abcabc") ==  0);
    assert(KMP("abcdef","defabc") ==  3);
    assert(KMP("qwertqwerhqweryqwerqwerqwqwqwerlk",
"hqweryqwerqwerqwqwqwerlkqwertqwer") ==  24);
    cout << endl << "Function KMP: Test OK" << endl;
```

```
}

int main() {
    string word,text;
    char choose;
    cout << "Choose an action:" << endl;
    cout << "d - enter data" << endl;
    cout << "t - run tests" << endl;
    std::cin >> choose;
    if (choose == 'd'){
        std::cin >> word;
        std::cin >> text;
        if ((word.length()) != text.length()) cout << -1;
        else {
            cout <<KMP(text,word);
        } }
    else if (choose == 't'){
        TestFunction_pre();
        TestKMP();
    }
    return 0;
}
```