**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. 9382 |  | Голубева В.П. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2021

## Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для того, чтобы научиться искать вхождения одной строки в другую и или проверять, что одна из них является циклическим сдвигом другой.

## Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных шаблона *P* (*∣P∣≤15000*) и текста *T* (*∣T∣≤5000000*) найдите 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*

**Задание 2**

Заданы две строки *A* (*∣A∣≤5000000*) и *B* (*∣B∣≤5000000*).

Определить, является ли *А* циклическим сдвигом *В*(это значит, что *А* и *В* имеют одинаковую длину и *А* состоит из суффикса *В*, склеенного с префиксом *В*). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - *A*

Вторая строка - *B*

Выход:

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

**Описание алгоритма**

Для того, чтобы не производить большое количество заведомо бесполезных шагов строим вектор значений префикс — функции.

Префикс-функция возвращает длину наибольшего префикса строки, где префикс совпадает с этой строкой. Таким образом, префикс-функция применяется для исходной строки N-1 раз, где N — длина строки.

После этого значение префикс-строки для текущей позиции в исходной строке используется следующим образом: после сдвига мы можем возобновить сравнения с шаблоном с позиции, равной значению префикс-функции от предыдущей позиции, а сравнение с шаблоном с последней текущей позиции, и при этом избегаем потери возможного местонахождения образца.

Во второй программе мы склеиваем первую строку саму с собой, после этого для второй строки мы можем понять, является ли она циклически получена из первой строки. Для этого нужно просто проверить, входит ли она в исходную строку с помощью алгоритма КМП.

**Оценка сложности по памяти**

В первой программе храним две входные строки с длинами m и n, а также префикс функцию, длина которой равна длине одной из строк, то есть сложность по памяти равна O(m+n)

Во второй программе храним также две входные строки с длинами m и n, а также префикс функцию и строку, склеенную из одной из них с самой с собой. Таким образом, сложность по памяти также равна O(m+n).

**Оценка сложности по времени**

Пусть pi[i] — значение префикс-функции от строки S[0, m-1] для индекса j. Тогда после сдвига мы можем возобновить сравнения с места T[i+j] и S[pi[j]] без потери возможного местонахождения образца. Можно показать, что таблица pi может быть вычислена (амортизационно) за O(m) сравнений перед началом поиска. А поскольку строка T будет пройдена ровно один раз, суммарное время работы алгоритма будет равно O(m + n), где — длина текста.

Такая же оценка будет и для второй программы. Хоть длина одной из строк будет больше, чем в первой программе, сложность не изменится - O(m + n).

**Тестирование**

Результаты тестирования программы можно посмотреть в приложениях В и Г.

## Выводы.

Был изучен алгоритм Кнута-Морриса-Пратта. Была реализована программа, которая осуществляет поиск вхождения одной строки в другую и или проверяет, что одна из них является циклическим сдвигом другой.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ LAB4\_1**

Название файла: lab4\_1.cpp

#include <iostream>

#include <vector>

void print\_vec(std::vector<int >vec){

for (int i = 0;i < vec.size(); i++){

if (i == 0){

std::cout << vec[i];

}

else{

std::cout << ',' << vec[i];

}

}

std::cout<<"\n";

}

std::vector<int> KnutMorrisPratt(std::string text, std::string templ)

{

std::vector<int> prefix(templ.size() + 1, -1);

std::vector<int> matches;

if (templ.size() == 0){

matches.push\_back(0);

return matches;

}

//compute prefix function

for (int i = 1; i <= templ.size(); i++){

int position = prefix[i - 1];

while (position != -1 && templ[position] != templ[i - 1]){

position = prefix[position];

}

prefix[i] = position + 1;

}

std::cout<< "Prefix function of template(without the -1 in beginning): ";

print\_vec(prefix);

std::cout << '\n';

int textpos = 0, templpos = 0;

while (textpos < text.size()){

while (templpos != -1 && (templpos == templ.size() || templ[templpos] != text[textpos])){

templpos = prefix[templpos];

}

std::cout << "templpos: " << templpos << ", textpos: " << textpos<<'\n';

textpos++;

templpos++;

if (templpos == templ.size()){ // if found a match

std::cout<<"find match at pos: " << textpos - templ.size() << '\n';

matches.push\_back(textpos - templ.size());

}

}

return matches;

}

int main(){

std::string s1;

std::string s2;

std::cin >> s1;

std::cin >> s2;

std::vector <int> match = KnutMorrisPratt(s2, s1);

if (match.size() == 0){

std::cout<< "Result: " << -1 << '\n';

}

else{

print\_vec(match);

}

}

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ПРОГРАММЫ LAB4\_2**

Название файла: lab4\_2.cpp

#include <iostream>

#include <vector>

#include <string>

void print\_vec(std::vector<int >vec){

for (int i = 0;i < vec.size(); i++){

if (i == 0){

std::cout << vec[i];

}

else{

std::cout << ", " << vec[i];

}

}

std::cout<<"\n";

}

std::vector <int> prefixFunc(std::string string){

int len\_s = string.length();

std::vector <int> prefix;

prefix.resize(len\_s);

prefix[0] = 0;

int ind = 0;

for (int i = 1; i < string.length(); i++){

while (ind > 0 && string[ind] != string[i]){

ind = prefix[ind - 1];

}

if (string[ind] == string[i]){

ind += 1;

}

prefix[i] = ind;

}

return prefix;

}

int isCyclicShift(std::string str1, std::string str2){

int result = -1;

if (str1.length() != str2.length()){ //unless exactly cyclic shift

std::cout << "string lengths not matched! ";

return result;

}

std::string pasting\_string = str1 + str1;

std::vector <int> pref = prefixFunc(str2); // find prefix - function

std::cout<< "Prefix function of str2: ";

print\_vec(pref);

std::cout << '\n';

int ind = 0;

for (int i = 0; i < pasting\_string.size(); i++){

//find index in str2 to match characters in str1

while (ind > 0 && str2[ind] != pasting\_string[i]){

ind = pref[ind - 1];

}

std::cout<<"compare characters in position " << i << " in str1 and "<< ind <<" in str2"<<'\n';

if (str2[ind] == pasting\_string[i]){// check matches

ind += 1;

}

if (ind == str2.length()) { // if find a substring in rasting string

std::cout<<"find a match with position " << i - str2.length() + 1<< ", finish!";

result = i - str2.length() + 1;

break;

}

}

return result;

}

int main() {

std::string s1;

std::string s2;

std::cin >> s1;

std::cin >> s2;

int res = isCyclicShift(s1, s2);

std::cout << "\nResult: " << res << '\n';

return 0;

}

**ПРИЛОЖЕНИЕ В**

**ТЕСТИРОВАНИЕ ПРОГРАММЫ LAB4\_1**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| ab  ababab | Prefix function of template(without the -1 in beginning): -1,0,0  templpos: 0, textpos: 0  templpos: 1, textpos: 1  find match at pos: 0  templpos: 0, textpos: 2  templpos: 1, textpos: 3  find match at pos: 2  templpos: 0, textpos: 4  templpos: 1, textpos: 5  find match at pos: 4  0,2,4 |
| abcdfeo  oabcdfe | Prefix function of template(without the -1 in beginning): -1,0,0,0,0,0,0,0  templpos: -1, textpos: 0  templpos: 0, textpos: 1  templpos: 1, textpos: 2  templpos: 2, textpos: 3  templpos: 3, textpos: 4  templpos: 4, textpos: 5  templpos: 5, textpos: 6  Result: -1 |
| qwe  qwerty | 0 |
| qwert  qwe | -1 |
| pnkvsnvsklv  sdnvksefljjk | -1 |

**ПРИЛОЖЕНИЕ Г**

**ТЕСТИРОВАНИЕ ПРОГРАММЫ LAB4\_2**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| defabc  abcdef | Prefix function of str2: 0, 0, 0, 0, 0, 0  compare characters in position 0 in str1 and 0 in str2  compare characters in position 1 in str1 and 0 in str2  compare characters in position 2 in str1 and 0 in str2  compare characters in position 3 in str1 and 0 in str2  compare characters in position 4 in str1 and 1 in str2  compare characters in position 5 in str1 and 2 in str2  compare characters in position 6 in str1 and 3 in str2  compare characters in position 7 in str1 and 4 in str2  compare characters in position 8 in str1 and 5 in str2  find a match with position 3, finish!  Result: 3 |
| defabc  abc | string lengths not matched!  Result: -1 |
| aaaaaaaa  bbbbbbbb | Result: -1 |
| abcabcabc  bcabcabca | Result: 1 |
| abcdfeo  oabcdfe | Result: 6 |