**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 КМП**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Петрова А.С. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2019

**Задание:**

**Вариант 2**

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

Оптимизация по памяти: программа должна требовать О(m) памяти, где m – длина образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

**Описание алгоритма.**

Алгоритм Кнута-Моррса-Пратта используется для поиска подстроки (образца) в строке. Кажется, что может быть проще: двигаемся по строке и сравниваем последовательно символы с образцом. Не совпало, перемещаем начало сравнения на один шаг и снова сравниваем. И так до тех пор, пока не найдем образец или не достигнем конца строки.

Рассмотрим сравнение строк на позиции i, где образец S[0,m-1] сопоставляется с частью текста T[i,i+m-1]. Предположим, что первое несовпадение произошло между T[i+j] и S[j], где 1<j<m. Тогда T[i,i+j-1]=S[0,j-1]=P и a=T[i+j]\neq S[j]=b.

При сдвиге вполне можно ожидать, что префикс (начальные символы) образца S сойдется с каким-нибудь суффиксом (конечные символы) текста P. Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки S для индекса j.

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

Алгоритм Кнута-Морриса-Пратта (КМП) позволяет находить префикс-функцию от строки за линейное время, и имеет достаточно лаконичную реализацию, по длине не превышающую наивный алгоритм.

Для начала заметим важное свойство: π[i]≤π[i−1]+1. То есть префикс-функция от следующего элемента не более чем на 1 превосходит префикс-функцию от текущего. Случай π[i]=π[i−1]+1 легко изобразить:

0 1 2 3 4 5 6

a b a c a b a

*s*[6]=*s*[2], π[5]=2, π[6]=π[5]+1

То есть верно следующее утверждение (в 0-индексации):

s[i]=s[π[i−1]]⇒π[i]=π[i−1]+ 1

Этот случай достаточно тривиален. Но что если s[i]≠s[π[i−1]]? Хотелось бы найти такую длину j,  s[0..j−1]=s[i−j..i−1], но при этом j<π[i−1]. Если s[i]=s[j], то π[i]=j+1. На самом деле, длина j уже была найдена в процессе нахождения префикс-функции. А именно, j=π[π[i−1]−1]. Графически это выглядит так:

0 1 2 3 4 5 6 7 8 9 10 11 12 13

a a b a a x x x x a a b a a

s[14]≠s[π[13]]

j=π[π[13]-1]=2

s[14]=s[2]

π[14]=j+1=3

Если же длина j также не подходит (s[i]≠s[j]), просто ещё раз уменьшим её по такой же формуле: j=π[j−1]. Таким образом будем пытаться продолжить префикс длины j, пока j не станет равно 0. В таком случае просто сравним s[i] с s[0], и в зависимости от результата присвоим π[i]=0 или 1.

**Тестирование.**

|  |  |
| --- | --- |
| ab  bcabcdbababcfdabcvs | 2,7,9,14 |
| file  filfilfilfilefilfilfielfile | 9,23 |

Рассмотрим подробный пример:

ola

lolalolalolalolalola

the result of the prefix function:

o l a

0 0 0

coincidence 1 liter o [0]

coincidence 2 liter l [1]

coincidence 3 liter a [2]

length coincided with the length of the template

coincidence 5 liter o [0] end line found

coincidence 6 liter l [1]

coincidence 7 liter a [2]

length coincided with the length of the template

coincidence 9 liter o [0] end line found

coincidence 10 liter l [1]

coincidence 11 liter a [2]

length coincided with the length of the template

coincidence 13 liter o [0] end line found

coincidence 14 liter l [1]

coincidence 15 liter a [2]

length coincided with the length of the template

coincidence 17 liter o [0] end line found

coincidence 18 liter l [1]

coincidence 19 liter a [2]

1, 5, 9, 13, 17

length coincided with the length of the template

Тестирование (циклическое вхождение)

|  |  |
| --- | --- |
| qwerty  tyqwer | 4 |
| asdfghjkl  jklasdfgh | 6 |

Рассмотрим подробный тест:

amamamama

maamamama

the result of the prefix function:

m a a m a m a m a

0 0 0 1 2 1 2 1 2

coincidence 1 liter m [0]

coincidence 2 liter a [1]

coincidence 3 liter m [0]

coincidence 4 liter a [1]

coincidence 5 liter m [0]

coincidence 6 liter a [1]

coincidence 7 liter m [0]

coincidence 8 liter a [1]

coincidence 9 liter a [2]

coincidence 10 liter m [3]

coincidence 11 liter a [4]

coincidence 12 liter m [5]

coincidence 13 liter a [6]

coincidence 14 liter m [7]

coincidence 15 liter a [8]

the entry found in 7 liter.

**Описание функций и структур данных.**

В данной программе не используется дополнительных функций и структур данных, поэтому описаны только переменные, которые используются в циклах.

string samplе – шаблон,

string text – текст,

vector<size\_t> prefix(sample.length()) – массив, хранящий префиксы для шаблона,

size\_t last\_prefix = prefix[0] = 0 – значение префикс функции,

bool was\_founded = false – переменная для отслеживания вхождений, если найдено, то значение изменяется на true,

bool was\_first = false - переменная для отслеживания вхождений, изменяется на true, если было найдено первое вхождение.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы (количество вхождений)**

#include "pch.h"

#include <string>

#include <iostream>

#include <vector>

using namespace std;

#define TEST

int main() {

string sample; // шаблон

string text; // исходный текст

cin >> sample;

cin >> text;

vector<size\_t> prefix(sample.length()); // массив который хранит префиксы для шаблона

// часть отвечающую за префикс функцию

size\_t last\_prefix = prefix[0] = 0;

// поиск префикс-функции для шаблона

for (size\_t i = 1; i < sample.length(); i++) {

while (last\_prefix > 0 && sample[last\_prefix] != sample[i])

last\_prefix = prefix[last\_prefix - 1];

if (sample[last\_prefix] == sample[i])

last\_prefix++;

prefix[i] = last\_prefix;

}

#ifdef TEST

cout << "the result of the prefix function:\n";

for (size\_t i = 0; i < prefix.size(); i++) {

cout << sample[i] << " ";

}

cout << endl;

for (size\_t i = 0; i < prefix.size(); i++) {

cout << prefix[i] << " ";

}

cout << endl;

#endif

// прогон строки и сравнение с префикс-функцией

last\_prefix = 0;

bool was\_founded = false;

bool was\_first = false;

for (size\_t i = 0; i < text.length(); i++) {

while (last\_prefix > 0 && sample[last\_prefix] != text[i]) // пока значение префикс функции >0 и символ шаблона = символу текста

last\_prefix = prefix[last\_prefix - 1];

if (sample[last\_prefix] == text[i]) // если символ шаблона от значения префикс-функции равно тексту

{

#ifdef TEST

for (size\_t j = 0; j < last\_prefix; j++) {

cout << " ";

}

cout << "coincidence " << i << " liter " << text[i] << " [" << last\_prefix << "]";

if (i && !last\_prefix&&was\_founded)

cout << " end line found";

cout << endl;

was\_founded = false;

#endif

last\_prefix++;

}

if (last\_prefix == sample.length()) { // если значение префикс-функции равно длине шаблона

#ifdef TEST

cout << "\nlength coincided with the length of the template\n";

was\_first = false;

was\_founded = true;

#endif

if (was\_first)

cout << ",";

was\_first = true; // вхождение найдено,

#ifndef TEST

cout << i + 1 - sample.length(); // выводим символ начала вхождения

#endif

}

}

if (!was\_first)

cout << -1;

}

**ПРИЛОЖЕНИЕ Б**

**Исходный код программы (циклический сдвиг)**

#include "pch.h"

#include <string>

#include <iostream>

#include <vector>

using namespace std;

#define TEST

int main() {

string sample; // шаблон

string text; // исходный текст

cin >> text;

cin >> sample;

// сравниваем длину текста и длину шаблона

// если они не равны, то выводим -1

if (text.length() != sample.length()) {

cout << -1;

return 0;

}

text += text;

vector<size\_t> prefix(sample.length()); // массив который хранит префиксы для шаблона

// часть отвечающую за префикс функцию

size\_t last\_prefix = prefix[0] = 0;

for (size\_t i = 1; i < sample.length(); i++) {

while (last\_prefix > 0 && sample[last\_prefix] != sample[i])

last\_prefix = prefix[last\_prefix - 1];

if (sample[last\_prefix] == sample[i])

last\_prefix++;

prefix[i] = last\_prefix;

}

#ifdef TEST

cout << "the result of the prefix function:\n";

for (size\_t i = 0; i < prefix.size(); i++) {

cout << sample[i] << " ";

}

cout << endl;

for (size\_t i = 0; i < prefix.size(); i++) {

cout << prefix[i] << " ";

}

cout << endl;

#endif

// прогон строки и сравнение с префикс-функцией

last\_prefix = 0;

bool was\_founded = false;

bool was\_first = false;

for (size\_t i = 0; i < text.length(); i++) {

while (last\_prefix > 0 && sample[last\_prefix] != text[i])// пока значение префикс функции >0 и символ шаблона = символу текста

last\_prefix = prefix[last\_prefix - 1];

if (sample[last\_prefix] == text[i]) // если символ шаблона от значения префикс-функции равно тексту

{

#ifdef TEST

for (size\_t j = 0; j < last\_prefix; j++) {

cout << " ";

}

cout << "coincidence " << i << " liter " << text[i] << " [" << last\_prefix << "]";

if (i && !last\_prefix&&was\_founded)

cout << " end line found";

cout << endl;

was\_founded = false;

#endif

last\_prefix++;

}

if (last\_prefix == sample.length()) { // если значение префикс-функции равно длине шаблона

#ifdef TEST

cout << "\nthe entry found in ";

was\_first = false;

was\_founded = true;

#endif

was\_first = true; // вхождение найдено

cout << i + 1 - sample.length() << " liter"; // выводим символ, с которого начинается циклическое вхождение

i = text.length();

}

}

if (!was\_first)

cout << -1 << "entry not found";

}