**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: **Алгоритм Кнута-Морриса-Пратта**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Шишкин И.В. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2020

**Цель работы.**

Изучить работу и реализовать алгоритм Кнута-Морриса-Пратта для нахождения подстроки в строке, а также написать программу для определения, является ли строка A циклическим сдвигом строки B.

**Постановка задачи.**

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.

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

Вар. 2. Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

**Описание алгоритма КМП.**

Алгоритм КМП состоит из двух этапов:

1. Формирование массива pi, используемого при сдвиге образа вдоль строки.
2. Поиск образа в строке.

Описание этапов:

1. Для формирования массива pi понадобится префикс-функция, которая для i-го символа образа возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-м символом. Это значение будет храниться в pi[i].

Например, префикс-функция для образа “abcabcd”:

Символ ‘a’: Для первого символа любого образа значение pi[0] всегда равно 0.

Символ ‘b’: ‘b’ != ‘a’, следовательно, pi[1] = 0.

Символ ‘c’: Суффикс и префикс длины 1 не совпадает (‘c’ != ‘a’), длины 2 тоже не совпадает (“ab” != “bc”). Поэтому pi[2] = 0.

Символ ‘a’: Префикс и суффикс длины 1 совпадают (‘a’ == ‘a’), остальные – нет, поэтому pi[3] = 1.

Символ ‘b’: Суффикс и префикс длины 1 не совпадает (‘b’ != ‘a’), длины 2 совпадает (“ab” == “ab”), длины 3 не совпадает (“abc” != “cab”), длины 4 не совпадает (“abca” != “bcab”), значит, pi[4] = 2.

И так далее. Итоговая префикс-функция будет: { 0, 0, 0, 1, 2, 3, 0 }.

1. На данном этапе производится непосредственно поиск образа (шаблона) в строке (тексте). Используются два индекса, которые указывают на символы в строке и образе. Когда символы совпадают – происходит смещение вправо (индексы увеличиваются на 1), когда они не совпадают – происходит использование массива pi (префикс-функции). В использовании префикс-функции и заключается отличительная особенность алгоритма КМП от наивного алгоритма. Идея состоит в том, чтобы не проверять символы, которые уже были проверены, т.е. мы не уходим назад.

Как использовать массив pi для определения того, сколько символов в строке нужно пропустить (для удобства, в пояснении используется строка text – текст (строка, в которой ищется вхождение образа), т.к. в циклическом сдвиге text используется): мы знаем, что символы pat[0..j-1] совпадают с text[i-j…i-1] (j начинается с 0 и увеличивает его только в случае совпадения). Также известно, что pi[j-1] – это количество символов pat[0… j-1], которые являются как собственным префиксом, так и суффиксом. Исходя из этого, можно сделать вывод, что нам не нужно сопоставлять символы pi[j-1] с text[i-j…i-1], потому что мы знаем, что эти символы в любом случае будут совпадать.

Пример работы алгоритма на строке “AAAAABAAABA” и образе “AAAA”:

Префикс-функция pi[] = { 0, 1, 2, 3 }.

i = 0, j = 0

text = "**AAAA**ABAAABA"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

i = 1, j = 1

text = "**AAAA**ABAAABA"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

i = 2, j = 2

text = "**AAAA**ABAAABA"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

i = 3, j = 3

text = "**AAAA**ABAAABA"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

i = 4, j = 4

Так как j == M, то запоминаем индекс вхождения i-j и сбрасываем j до значения, равного j = pi[j-1] = pi[3] = 3. Значение pi[j-1] дает индекс следующего символа в строке, которое будет сравниваться.

i = 4, j = 3

text = "A**AAAA**BAAABA"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

i = 5, j = 4

Так как j == M, то запоминаем индекс вхождения i-j и сбрасываем j до значения, равного j = pi[j-1] = pi[3] = 3.

i = 5, j = 3

text = "AA**AAAB**AAABA"

pat = "**AAAA**"

text[i] и pat[j] НЕ совпадают и j > 0, поэтому меняем только j: j = pi[j-1] = pi[2] = 2

i = 5, j = 2

text = "AAA**AABA**AABA"

pat = "**AAAA**"

text[i] и pat[j] НЕ совпадают и j > 0, поэтому меняем только j: j = pi[j-1] = pi[1] = 1

i = 5, j = 1

text = "AAAA**ABAA**ABA"

pat = "**AAAA**"

text[i] и pat[j] НЕ совпадают и j > 0, поэтому меняем только j: j = pi[j-1] = pi[0] = 0

i = 5, j = 0

text = "AAAAA**BAAA**BA"

pat = "**AAAA**"

text[i] и pat[j] НЕ совпадают и j равно 0, поэтому i++.

i = 6, j = 0

text = "AAAAAB**AAABA**"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

i = 7, j = 1

text = "AAAAAB**AAAB**A"

pat = "**AAAA**"

text[i] и pat[j] совпадают, поэтому i++, j++

И так далее.

**Описание программы поиска циклического сдвига.**

Данная программа является небольшой модификацией программы поиска подстроки в строке. Чтобы было удобнее, создается переменная string text, в которой будет храниться строка A (string pat, соответственно, строка B). Задачу поиска циклического сдвига можно переформулировать так, что нужно найти вхождение строки B в конкатенацию строк A и A. То есть, если нужно определить, является ли строка B = “defabc” циклическим сдвигом строки A = “abcdef”, то это значит, что нужно найти первое вхождение строки “defabc” в строку “abcdefabcdef”.

**Описание способов хранения частичных решений.**  Хранится string pattern – для считывания шаблона.

vector <int> answer – для вывода ответа в конце программы.

В программе поиска циклического сдвига также хранится строка string text.

**Описание функции KMPSearch.**

Функция void KMPSearch(string pat, vector <int>& answer) на вход принимает шаблон pat и ссылку на вектор answer для записи ответа в него. В начале рассчитывается значение M, равное длине строки pat. Так как по памяти программа должна требовать O(M) памяти, то считывание текста производится посимвольно с помощью переменной sym. Для образа pat вычисляется префикс-функция с помощью функции prefixFunc, результат которой записывается в вектор pi. Далее, в цикле, когда символы текста и шаблона совпадают, индексы i и j увеличиваются на 1 и считывается очередной символ текста, а когда не совпадают – если при этом индекс j не равен 0, то значению j присваивается значение pi[j-1], иначе увеличивается только индекс i на 1. Если же индекс j равен длине шаблона, то это значит, что найдено вхождение шаблона в текст, тогда индекс этого вхождения можно вычислить как i-j, которое записывается в вектор answer, и j принимает значение pi[j-1]. Цикл завершает свою работу, когда встречается конец строки или нажатие на enter.

**Описание функции prefixFunc.**

Функция void prefixFunc(string pat, int M, vector<int>& pi) на вход принимает строку pat – шаблон, M – длину шаблона, вектор pi – собственно массив пи, который и нужно заполнить.

Так как вектор в функцию подается уже обнуленный, то не нужно обнулять значение под индексом 0.

Запускается цикл от i=1 до i=M-1. Схема алгоритма:

* Для подсчёта текущего значения pi[i] заводится переменная j, обозначающая длину текущего рассматриваемого образца. При этом изначально j = pi[i-1].
* Тестируется образец длины j, для чего сравниваются символы pat[j] и pat[i]. Если они совпадают – то полагаем pi[i] = j+1 и переходим к следующему индексу i+1. Если же символы отличаются, то уменьшаем длину j, полагая её равной pi[j-1], и повторяем этот шаг алгоритма с начала.
* Если мы дошли до длины j=0 и так и не нашли совпадения, то останавливаем процесс перебора образцов и полагаем pi[i] = 0, а затем переходим к следующему индексу i+1.

**Отличия программы определения циклического сдвига от программы поиска подстроки в строке.**

Главное отличие заключается в том, что программа определения циклического сдвига хранит строку string text. При считывании, она подается в измененную функцию int KMPSearch(string& pat, string& text), которая на вход получает шаблон pat и текст text, а возвращает индекс начала строки pat в строке text. Отличий в функции prefixFunc нет.

В начале функции KMPSearch производится проверка: если длины строк, которые программа получила на вход разные, то возвращается -1. Если строки равны друг другу, то возвращается 0.

Затем переменная text приобретает значение text+text, и работа алгоритма производится с новой переменной. Остальная часть алгоритма такая же, как и в первой программе. Функция заканчивает работу раньше времени, когда находит вхождение строки, т.к. по условию задания нужно вывести лишь первое вхождение.

**Сложность алгоритмов по времени.**

Сложность алгоритма КМП по времени можно оценить, как

Так как префикс-функция вычисляется за O(M), где M – это количество символов в образе, и поскольку строка (текст) в худшем случае будет пройдена полностью 1 раз (N – длина строки).

Сложность программы определения циклического сдвига будет такой же.

**Сложность алгоритмов по памяти.**

Сложность алгоритма КМП по памяти можно оценить, как

, где M – длина образа.

Так как по условию варианта можно было хранить только образ, а текст считывать посимвольно.

Сложность программы определения циклического сдвига по памяти можно оценить, как

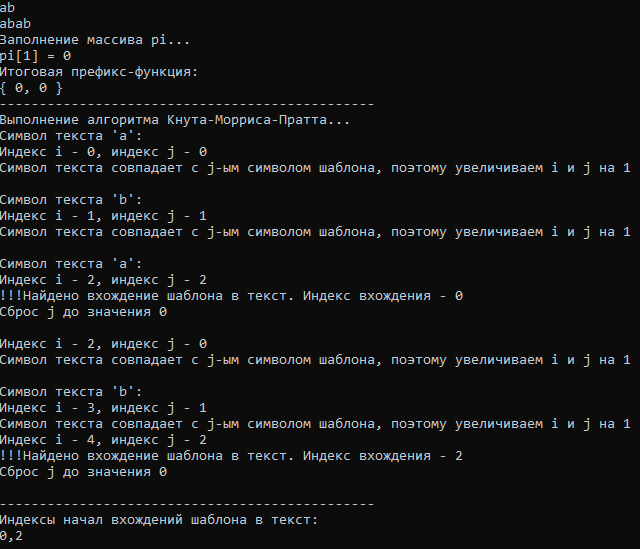
Так как программа хранит строку A длиной M и строку B длиной N.

**Спецификация программы.**

Программа написана на языке C++. Программа на вход получает две строки, причем, в случае КМП, в начале подается шаблон, а затем текст, а в случае циклического сдвига, в начале подается строка A, а затем строка B.

**Тестирование КМП.**

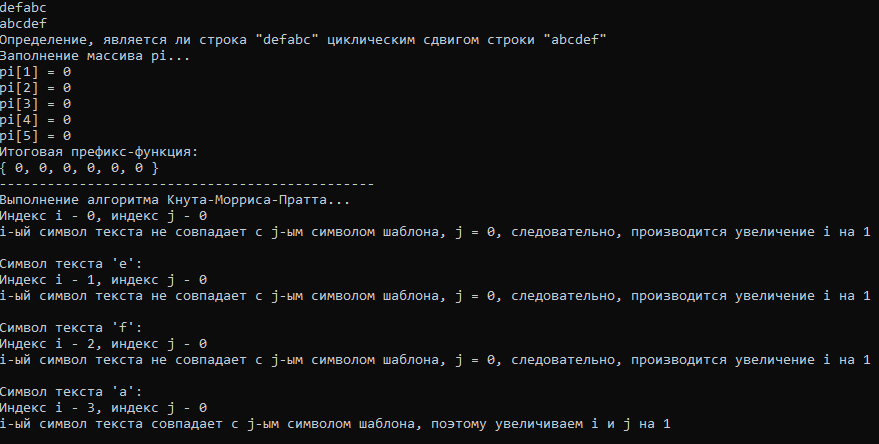
Пример вывода результата для 1-го теста.

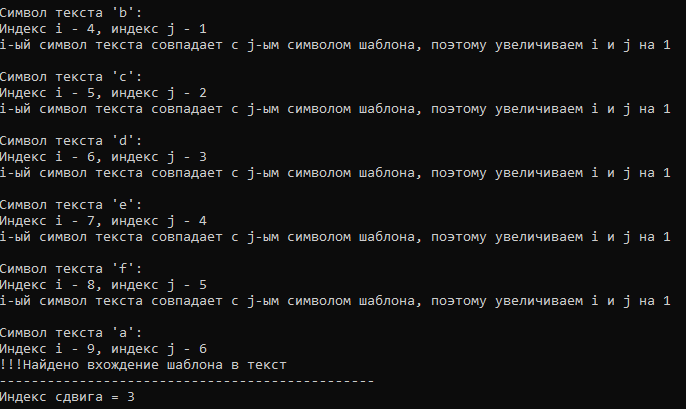


|  |  |  |
| --- | --- | --- |
| № | Input | Output |
| 1 | ab  abab | 0,2 |
| 2 | AAAA  AAAAABAAABA | 0,1 |
| 3 | ivan  ivannivaan ivann | 0,11 |

**Тестирование циклического сдвига.**

Пример вывода результата для 1-го теста.





|  |  |  |
| --- | --- | --- |
| № | Input | Output |
| 1 | defabc  abcdef | 3 |
| 2 | Ab  Ab | 0 |
| 3 | hhhhhjjjjj  jjjjhhhhhh | -1 |

**Вывод.**

В ходе выполнения лабораторной работы был реализован на языке C++

алгоритм Кнута-Морриса-Пратта для нахождения подстроки в строке.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ КМП

#include <iostream>

#include <vector>

using namespace std;

void KMPSearch(string pat, vector <int>& answer); // Выводит индексы вхождения строки pat в текст, который посимвольно вводится в этой функции

void prefixFunc(string pat, int M, vector<int>& pi); // Префикс функция

int main() {

setlocale(LC\_ALL, "RUS");

string pattern; //шаблон

cin >> pattern;

getchar(); //считывание символа '\n'

vector <int> answer;

KMPSearch(pattern, answer);

cout << "-----------------------------------------------" << endl;

cout << "Индексы начал вхождений шаблона в текст:" << endl;

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

cout << "Вхождений не было найдено, ответ: -1" << endl;

}

else {

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

if (i == 0) cout << answer[i];

else cout << "," << answer[i];

}

}

cout << endl;

return 0;

}

void KMPSearch(string pat, vector <int>& answer) {

char sym = NULL; //для посимвольного считывания текста

int M = pat.length(); //длина шаблона

vector <int> pi(M, 0); //длина максимального префикса строки, совпадающего с его суффиксом

int i = 0; //индекс для перемещения по тексту

int j = 0; //индекс для перемещения по строке pattern

sym = cin.get(); //считывание первого символа

prefixFunc(pat, M, pi); //вычисление массива pi для строки pat

cout << "Выполнение алгоритма Кнута-Морриса-Пратта..." << endl;

if (sym != '\n' && sym != EOF) cout << "Символ текста '" << sym << "': " << endl;

while (sym != '\n' && sym != EOF) {

cout << "Индекс i - " << i << ", индекс j - " << j << endl;

if (pat[j] == sym) { //если j-ый символ шаблона равен i-му символу текста

cout << "Символ текста совпадает с j-ым символом шаблона, поэтому увеличиваем i и j на 1" << endl;

j++; //j увеличивается только когда находится сходство

i++;

sym = cin.get();

if (sym != '\n' && sym != EOF) cout << endl << "Символ текста '" << sym << "': " << endl;

}

if (j == M) { //если j равен длине шаблона, то найдено вхождение шаблона в текст

cout << "Индекс i - " << i << ", индекс j - " << j << endl;

cout << "!!!Найдено вхождение шаблона в текст. Индекс вхождения - " << i - j << endl;

answer.push\_back(i - j);

cout << "Сброс j до значения " << pi[j - 1] << endl << endl;

j = pi[j - 1];

}

else if (pat[j] != sym) { //если j-ый символ шаблона не равен символу текста

cout << "Символ текста не совпадает с j-ым символом шаблона";

if (j != 0) {

cout << ". Теперь j принимает значение " << pi[j - 1] << endl;

j = pi[j - 1];

}

else {

cout << ", j = 0, следовательно, производится увеличение i на 1" << endl;

i = i + 1;

sym = cin.get();

if (sym != '\n' && sym != EOF) cout << endl << "Символ текста '" << sym << "': " << endl;

}

}

}

}

void prefixFunc(string pat, int M, vector<int>& pi) {

cout << "Заполнение массива pi..." << endl;

for (int i = 1; i < M; i++) { //вычисление массива пи от i := 1 до i := M-1

int j = pi[i - 1]; //для подсчёта текущего значения pi[i] заводится переменная j, обозначающая длину текущего рассматриваемого образца

while (j > 0 && pat[i] != pat[j]) { //сравнение символов pat[i] и pat[j]

cout << "Обработка символов '" << pat[i] << "' и '" << pat[j] << "'..." << endl;

j = pi[j - 1];

}

if (pat[i] == pat[j]) { //если символы pat[i] и pat[j] совпадают, то pi[i] = j + 1

cout << "Суффикс '" << pat[i] << "' и префикс " << pat[j] << "' равны, pi[" << i << "] увеличивается на 1" << endl;

++j;

}

cout << "pi[" << i << "] = " << j << endl;

pi[i] = j;

}

cout << "Итоговая префикс-функция:" << endl << "{ ";

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

if (i == 0) cout << pi[i];

else cout << ", " << pi[i];

}

cout << " }" << endl;

cout << "-----------------------------------------------" << endl;

}

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ ЦИКЛИЧЕСКОГО СДВИГА

#include <iostream>

#include <vector>

using namespace std;

int KMPSearch(string& pat, string& text); //КМП

void prefixFunc(string pat, int M, vector<int>& pi); // Префикс функция

int main() {

setlocale(LC\_ALL, "RUS");

string pattern; //шаблон (строка B)

string text; //текст (строка A)

cin >> text;

cin >> pattern;

int answer = KMPSearch(pattern, text);

cout << "-----------------------------------------------" << endl;

cout << "Индекс сдвига = " << answer << endl;

return 0;

}

int KMPSearch(string& pat, string& text) {

cout << "Определение, является ли строка \"" << text << "\" циклическим сдвигом строки \"" << pat << "\"" << endl;

int M = pat.length(); //длина шаблона

int N = text.length(); //длина текста

if (M != N) { //если длина текста и шаблона не равны, то возвращается -1

cout << "Длина текста и шаблона не равны" << endl;

return -1;

}

else if (pat == text) {

cout << "Строки идентичны" << endl;

return 0;

}

text += text;

N = text.length(); //новое значение длины текста

vector <int> pi(M, 0); //длина максимального префикса строки, совпадающего с его суффиксом

int i = 0; //индекс для перемещения по строке text

int j = 0; //индекс для перемещения по строке pattern

prefixFunc(pat, M, pi); //вычисление массива pi для строки pat

cout << "Выполнение алгоритма Кнута-Морриса-Пратта..." << endl;

while (i < N) {

cout << "Индекс i - " << i << ", индекс j - " << j << endl;

if (pat[j] == text[i]) { //если j-ый символ шаблона равен i-му символу текста

cout << "i-ый символ текста совпадает с j-ым символом шаблона, поэтому увеличиваем i и j на 1" << endl;

j++; //j увеличивается только когда находится сходство

i++;

cout << endl << "Символ текста '" << text[i] << "': " << endl;

}

if (j == M) { //если j равен длине шаблона, то найдено вхождение шаблона в текст

cout << "Индекс i - " << i << ", индекс j - " << j << endl;

cout << "!!!Найдено вхождение шаблона в текст" << endl;

return i - j;

}

else if (i < N && pat[j] != text[i]) { //если j-ый символ шаблона не равен символу текста

cout << "i-ый символ текста не совпадает с j-ым символом шаблона";

if (j != 0) {

cout << ". Теперь j принимает значение " << pi[j - 1] << endl;

j = pi[j - 1];

}

else {

cout << ", j = 0, следовательно, производится увеличение i на 1" << endl;

i++;

cout << endl << "Символ текста '" << text[i] << "': " << endl;

}

}

}

cout << "Строка не является циклическим сдвигом" << endl;

return -1;

}

void prefixFunc(string pat, int M, vector<int>& pi) {

cout << "Заполнение массива pi..." << endl;

for (int i = 1; i < M; i++) { //вычисление массива пи от i := 1 до i := M-1

int j = pi[i - 1]; //для подсчёта текущего значения pi[i] заводится переменная j, обозначающая длину текущего рассматриваемого образца

while (j > 0 && pat[i] != pat[j]) { //сравнение символов pat[i] и pat[j]

cout << "Обработка символов '" << pat[i] << "' и '" << pat[j] << "'..." << endl;

j = pi[j - 1];

}

if (pat[i] == pat[j]) { //если символы pat[i] и pat[j] совпадают, то pi[i] = j + 1

cout << "Суффикс '" << pat[i] << "' и префикс " << pat[j] << "' равны, pi[" << i << "] увеличивается на 1" << endl;

++j;

}

cout << "pi[" << i << "] = " << j << endl;

pi[i] = j;

}

cout << "Итоговая префикс-функция:" << endl << "{ ";

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

if (i == 0) cout << pi[i];

else cout << ", " << pi[i];

}

cout << " }" << endl;

cout << "-----------------------------------------------" << endl;

}