**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: Поиск подстроки в строке**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Лобова Е. И. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2025

## Цель работы

Целью работы является изучение алгоритма Кнута-Морриса-Пратта и практическое его применение на задаче о том, является ли одна строка циклическим сдвигом другой.

## Задание

**Задание 1**

Заданы две строки A (|A| <= 5000000) и B (|B| <= 5000000).

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

Sample Input:

defabc

abcdef

Sample Output:

3

**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 А является циклическим сдвигом В, индекс начала строки В в А, иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

Sample Input:

defabc

abcdef

Sample Output:

3

## Описание алгоритмов

Для решения задания лабораторной работы использовались алгоритмы префикс-функции и Кнута-Морриса-Пратта.

Алгоритм префикс-функции осуществляется в функции *vector<int> prefixFunction(const string& s)* - функция, которая вычисляет префикс-функцию для заданной строки s. Префикс-функция представляет собой массив *p*, где *p[i]* - это длина наибольшего собственного префикса подстроки *s[0...i]*, который одновременно является её суффиксом. Алгоритм итерирует по строке, меняя переменную *index*, представляющую длину текущего кандидата в наибольший префикс. Если текущий символ строки совпадает с символом по индексу index, то index увеличивается (то есть был продолжен текущим символом префикс для предыдущего символа). Если не совпадает, *index* уменьшается, беря предыдущее значение из массива *p*, пока не найдет совпадение или не станет равным нулю. Сравнение только определенных символов избавляет нас от затратных сравнений строк полностью. Значение *index* после каждого шага присваивается *p[i]*.

Алгоритм Кнута-Морриса-Пратта (KMP) осуществляется в функции *vector<int> search(string &pat, string &txt)* - функция, которая ищет все вхождения подстроки pat в строку txt. Функция возвращает вектор, содержащий индексы начала всех найденных вхождений. Для подстроки считается префикс-функция, чтобы в случае несовпадения символов не с начала проверять подстроку, а использовать ее префиксы. В цикле проходимся по всем символом строки и сравниваем их с символами подстроки, если они совпадают, то в обеих строках двигаемся дальше или в случае окончания подстроки добавляем индекс начала вхождения подстроки в строку в результирующий массив. Если символы не совпадают, то есть 2 пути: если не совпал первый символ подстроки, то просто переходим к следующему символу в строке; если не совпал не первый символ, то вычисляем новый индекс подстроки с использованием префикс-функции.

Также для выполнения лабораторной работы был написан алгоритм определения является ли строка *А* циклическим сдвигом строки *B* на основе алгоритма КМП. Этот алгоритм осуществляется функцией - int *cycle\_search(string &pat, string &txt)*. Алгоритм аналогичен тому, что представлен в функции search, но с модификацией, что за строку мы берем *A+A* (нереально выполняем конкатенацию строк, чтобы не увеличивать расходы на память, а индекс два раза проходит по строке *A*), и в этой строке находим первое вхождение подстроки *B*. Также перед циклом выполняется проверка на то, что у строк *A* и *B* совпадает длина.

Сложность алгоритма для задания 1:

* По памяти: О(n+2m)

O(n+m) для хранения строк. O(m) для хранения массива префикс-функций.

* По времени: O(n+m)

O(m) - построение префикс-функции, О(n) - итоговое количество итераций цикла while, то есть проход по строке.

Сложность алгоритма для задания 2:

* По памяти: О(n+m), если строки разной длины (т. к. проверка на различие длин производится до построения префикс функции), иначе O(3n):

O(n+n) для хранения строк. O(n) для хранения массива префикс-функций.

* По времени: O(1), если строки разной длины (есть проверка до вхождения в цикл), иначе О(3n):

O(n) - построение префикс-функции, O(2n) - количество итераций цикла while, то есть двойной проход по строке A.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | aaaaaaaa  aa | 0,1,2,3,4,5,6 | КМП: Для пересекающихся вхождений подстрок работает корректно. |
|  | abababcababc  ababc | 2, 7 | КМП: Для простого случая подстроки и строки работает корректно. |
|  | ababbbc  ababbbc | 0 | КМП: Для одинаковых подстроки и строки работает корректно. |
|  | abababc  cdab | -1 | КМП: Для строки, в которой нет ни одного вхождения подстроки работает корректно. |
|  | defabc  abcdef | 3 | Цикл. сдвиг: Для строк одинаковой длины и являющихся циклическим сдвигом одна другой работает корректно. |
|  | abcdef  ab | -1 | Цикл. сдвиг: Для строк разной длины работает корректно. |
| 7 | abcdef  defacc | -1 | Цикл. сдвиг: Для строк одинаковой длины и не являющихся циклическим сдвигом одна другой работает корректно. |

## 

## Выводы

В ходе работы были успешно изучены алгоритмы построения префикс-функции и Кнута-Морриса-Пратта. Они были успешно применены в задаче о том, является ли одна строка циклическим сдвигом другой.

# Приложение А Исходный код программы

Название файла: kmp.cpp

#include <iostream>

#include <vector>

#include <string>

#define RED "\033[1;31m"

#define BASIC "\033[0m"

using namespace std;

vector<int> prefixFunction(const string& s) {

cout << "Вычисление префикс-функции - массивa p, где p[i] - длина наибольшего собственного префикса подстроки s[0...i], совпадающего с её суффиксом:\n\n";

int n = s.length();

vector<int> p(n);

p[0] = 0;

cout << "p[0] = 0, так как у первого символа есть только тривиальный префикс\n\n";

for (int i = 1; i < n; ++i) {

cout << "Вычисление p[" << i << "]\n";

int index = p[i - 1];

cout << "За начальный индекс для сравнения(index) возьмем длину наибольшего префикса подстроки s[0..." << i-1 << "], то есть p["<< i-1 <<"] = " << index << "\n";

while (index > 0 && s[i] != s[index]) {

cout << "Символ s[" << i << "] = " << s[i] << " НЕ совпал с символом s[" << index << "] = " << s[index] << "\n";

cout << "Переходим к более короткому префиксу. За индекс сравнения принимаем предыдущее значение массива префикс-значений по индексу [index - 1] = " << p[index-1] << "\n";

index = p[index - 1];

}

if (s[i] == s[index]) {

cout << "Символ s[" << i << "] = " << s[i] << " совпал с символом s[" << index << "] = " << s[index] << "\n";

cout << "То есть можно продолжить текущим символом префикс для p[" << i-1 << "]\n";

index++;

}

if (index > 0){

cout << "Был найден наибольший собственный префикс, совпадающий с суффиксом, длины " << index << "\n";

} else {

cout << "Подстрока не имеет собственного префикса, совпадающиего с суффиксом\n";

}

cout << "Следовательно, p[" << i << "] = " << index << "\n\n";

p[i] = index;

}

return p;

}

vector<int> search(string &pat, string &txt) {

int n = txt.length();

int m = pat.length();

vector<int> res;

vector<int> prefix\_array = prefixFunction(pat);

cout << "Начинаем поиск подстроки " << pat << " в строке " << txt << "\n";

int i = 0;

int j = 0;

cout << "Сравнение символов начинаем с начала строки и подстроки, то есть с нулевых индексов\n";

while (i < n) {

if (txt[i] == pat[j]) {

cout << "Символ строки text[" << i << "] = " << txt[i] << " совпал с символом подстроки pattern[" << j << "] = " << pat[j] << "\n";

i++;

j++;

cout << "Поэтому переходим к следующим символам в обеих строках: индекс в строке увеличиваем до " << i << ", индекс с подстроке - " << j << "\n";

if (j == m) {

cout << RED << "Вхождение подстроки найдено!!! Индекс начала подстроки в строке " << i - j << BASIC << "\n";

res.push\_back(i - j);

j = prefix\_array[j - 1];

cout << "Индекс в подстроке сбрасываем до " << j << "\n";

}

}

else {

cout << "Символ строки text[" << i << "] = " << txt[i] << " НЕ совпал с символом подстроки pattern[" << j << "] = " << pat[j] << "\n";

if (j != 0){

j = prefix\_array[j - 1];

cout << "Индекс в подстроке сбрасываем до " << j << "\n";

}else{

i++;

cout << "Переходим к следующему символу в строке\n";

}

}

}

cout << "Достигнут конец строки, поиск завершён\n";

return res;

}

int main() {

string pattern, text;

cin >> text;

cin >> pattern;

vector<int> res = search(pattern, text);

if (res.empty()) {

cout << -1 << endl;

} else {

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

cout << res[i];

if (i < res.size() - 1) {

cout << ",";

}

}

cout << endl;

}

return 0;

}

Название файла: cycle\_search.cpp

#include <iostream>

#include <vector>

#include <string>

#define RED "\033[1;31m"

#define BASIC "\033[0m"

using namespace std;

vector<int> prefixFunction(const string& s) {

cout << "Вычисление префикс-функции - массивa p, где p[i] - длина наибольшего собственного префикса подстроки s[0...i], совпадающего с её суффиксом.:\n\n";

int n = s.length();

vector<int> p(n);

p[0] = 0;

cout << "p[0] = 0, так как у первого символа есть только тривиальный префикс\n\n";

for (int i = 1; i < n; ++i) {

cout << "Вычисление p[" << i << "]\n";

int index = p[i - 1];

cout << "За начальный индекс для сравнения(index) возьмем длину наибольшего префикса подстроки s[0..." << i-1 << "], то есть p["<< i-1 <<"] = " << index << "\n";

while (index > 0 && s[i] != s[index]) {

cout << "Символ s[" << i << "] = " << s[i] << " НЕ совпал с символом s[" << index << "] = " << s[index] << "\n";

cout << "Переходим к более короткому префиксу. За индекс сравнения принимаем предыдущее значение массива префикс-значений по индексу [index - 1] = " << p[index-1] << "\n";

index = p[index - 1];

}

if (s[i] == s[index]) {

cout << "Символ s[" << i << "] = " << s[i] << " совпал с символом s[" << index << "] = " << s[index] << "\n";

cout << "То есть можно продолжить текущим символом префикс для p[" << i - 1 << "]\n";

index++;

} else {

cout << "Символ s[" << i << "] = " << s[i] << " НЕ совпал с символом s[" << index << "] = " << s[index] << "\n";

}

if (index > 0){

cout << "Был найден наибольший собственный префикс, совпадающий с суффиксом, длины " << index << "\n";

} else {

cout << "Подстрока не имеет собственного префикса, совпадающиего с суффиксом\n";

}

cout << "Следовательно, p[" << i << "] = " << index << "\n\n";

p[i] = index;

}

return p;

}

int cycle\_search(string &pat, string &txt){

int n = txt.length();

int m = pat.length();

if (n!=m) return -1;

n \*= 2;

int index = -1;

vector<int> prefix\_array = prefixFunction(pat);

cout << "Определим является ли " << txt << " циклическим сдвигом строки " << pat << "\n";

int i = 0;

int double\_i = 0;

int j = 0;

cout << "Сравнение символов начинаем с начала строки и подстроки, то есть с нулевых индексов\n";

while (double\_i < n) {

i = double\_i % m;

if (txt[i] == pat[j]) {

cout << "Символ строки A[" << i << "] = " << txt[i] << " совпал с символом строки B" << j << "] = " << pat[j] << "\n";

double\_i++;

j++;

cout << "Поэтому переходим к следующим символам в обеих строках: индекс в строке A увеличиваем до " << i << ", индекс с строке B - " << j << "\n";

if (j == m) {

index = double\_i - j;

cout << RED << "Строка A является циклическим сдвигом строки B! Индекс начала B в A - " << index << BASIC << "\n";

break;

}

}

else {

cout << "Символ строки A[" << i << "] = " << txt[i] << " НЕ совпал с символом строки B" << j << "] = " << pat[j] << "\n";

if (j != 0){

j = prefix\_array[j - 1];

cout << "Индекс в строке B сбрасываем до " << j << "\n";

}else{

double\_i++;

cout << "Переходим к следующему символу в строке A\n";

}

}

}

cout << "Вся строка A просмотрена, алгоритм завершен\n";

return index;

}

int main() {

string B, A;

cin >> A;

cin >> B;

int res = cycle\_search(B, A);

cout << res << endl;

return 0;

}