**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»

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Пивоев Н. М. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2025

# Цель работы

Изучить работу алгоритма Кнута-Морриса-Пратта, с его помощью решить задачу поиска вхождений заданного шаблона в текст и определение того, является ли одна строка циклическим сдвигом другой.

# Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных шаблона и текста найдите все вхождения P в T.

Входные данные:

Первая строка – P

Вторая строка – T

Выходные данные:

Индексы начал вхождений P в T, разделенные запятой, если P не входит в T, то вывести -1.

**Sample Input:**

ab

abab

**Sample Output:**

0,2

# Задание 2

Заданы две строки и .

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

Входные данные:

Первая строка – A

Вторая строка – B

Выходные данные:

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

**Sample Input:**

defabc

abcdef

**Sample Output:**

3

## Выполнение работы

Описание алгоритма

Алгоритм Кнута-Морриса-Пратта – это алгоритм эффективного поиска подстроки в строке.

Для эффективной работы алгоритма используется префикс-функция, которая для каждого символа строки-паттерна находит соответствующий индекс, равный максимальному размеру текущего префикса и совпадающего с ним суффикса. Полученные значения используется при поиске вхождения при несоответствии текущих символов текста и паттерна, для меньшего смещения текущей позиции в паттерне.

При поиске вхождений вводятся два указателя на текст и паттерн. Затем идёт анализ текущих символов на совпадение. Если символы равны, то увеличиваются оба указателя, а если строка-паттерн была пройдена, то было найдено вхождение. При несовпадении символов, необходимо применить результат префикс-функции для того, чтобы начать дальнейший поиск не с нуля, а с определённого смещения.

Алгоритм также применим для нахождения циклического сдвига строк. Для этого необходимо удвоить строку, в которой осуществляется поиск вхождения, чтобы в ней гарантированно была искомая строка в полном виде.

Оценка сложности

Временная сложность алгоритма линейна – , где n – длина текста, а m – длина паттерна. При составлении префикс-функции временная сложность – O(m), при обработке текста – O(n), поскольку мы один раз проходимся по строке. Временная сложность поиска циклического сдвига - , поскольку длина строки, в которой ведётся поиск, удваивается.

Пространственная сложность алгоритма - поскольку нам необходимо хранить массив длин префиксов равный размеру паттерна.

Код программы содержит реализацию следующих функций:

* *std::vector<int> prefixFunc(std::string pattern)* – ставит в соответствие каждому символу паттерна число, равное максимальному суффиксу для каждого префикса паттерна.
* *std::vector<int> kmp\_algorithm(std::string text, std::string pattern)* – опираясь на результаты префикс-функции, эффективно вычисляет вхождения подстроки в тексте.
* *int shift\_algorithm(std::string a, std::string b)* – опираясь на результаты префикс-функции, эффективно вычисляет, является ли одна строка циклическим сдвигом другой.

# Тестирование

Обе программы были протестированы на различных входных данных. Соответственно составлены две таблицы:

Таблица 1. Тестирование задачи поиска шаблона.

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| ab  abab | 0,2 |
| abc  abc | 0 |
| abcd  abc | -1 |
| aa  aaaaa | 0,1,2,3 |
| abac  ababacabac | 2,6 |
| a  b | -1 |

Таблица 2. Тестирование задачи поиска циклического сдвига.

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| defabc  abcdef | 3 |
| abc  abcd | -1 |
| abc  def | -1 |
| a  a | 0 |
| abcabcabc  bcabcabca | 7 |
| aaaaabaa  baaaaaaa | 5 |
| abcdef  cdefab | 2 |

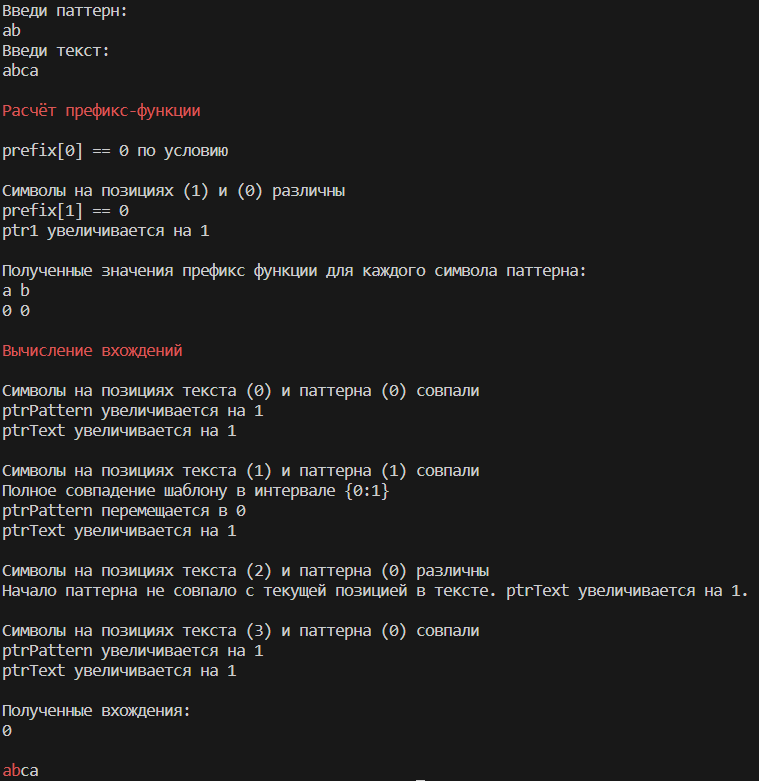


Рисунок 1 – Результат работы программы kmp.cpp

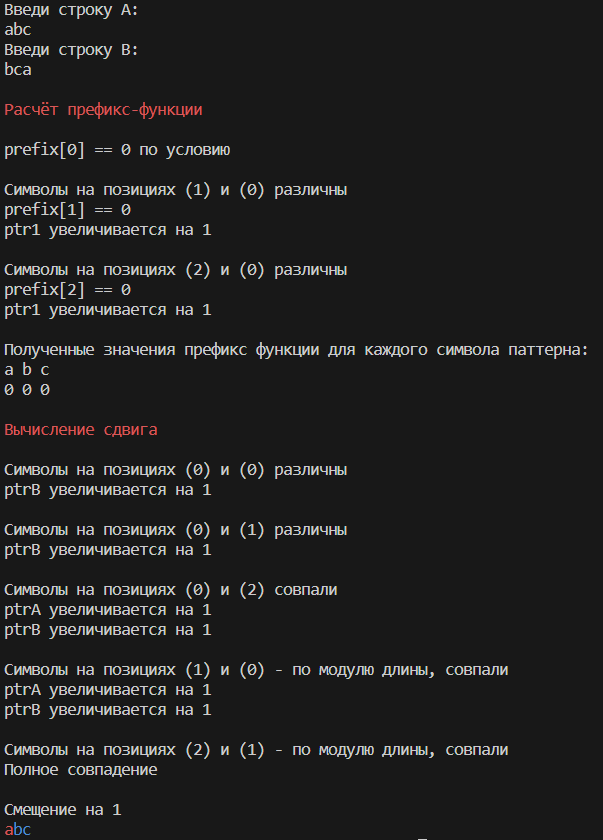


Рисунок 2 – Результат работы программы shift.cpp

# Исследование

Исследуем эффективность алгоритма Кнута-Морриса-Пратта на различных объёмах входных данных.

Таблица 3. Исследование эффективности по времени.

|  |  |  |
| --- | --- | --- |
| Размер текста | Размер паттерна | Затраченное время (с) |
| 1.000 | 1 | 0.000037 |
| 1.000 | 10 | 0.000044 |
| 100.000 | 1 | 0.002434 |
| 100.000 | 100 | 0.002972 |
| 100.000 | 10.000 | 0.003082 |
| 1.000.000 | 1 | 0.025418 |
| 1.000.000 | 1.000 | 0.028378 |
| 1.000.000 | 100.000 | 0.029789 |
| 10.000.000 | 1 | 0.249666 |
| 10.000.000 | 100 | 0.289753 |
| 10.000.000 | 10.000 | 0.292171 |
| 10.000.000 | 1.000.000 | 0.304549 |

Рисунок 3 – График зависимости времени работы алгоритма от размера текста

# Можно сделать следующие выводы по исследованию:

1. Время работы алгоритма прямо пропорционально размеру текста.
2. Время работы практически не зависит от размера паттерна, но при небольших размерах, результат вычисляется эффективнее.

# Выводы

Во время выполнения лабораторной работы, была изучена работа алгоритма Кнута-Морриса-Пратта. Решены задачи поиска вхождений заданного шаблона в текст и определение того, является ли одна строка циклическим сдвигом другой.

**ПРИЛОЖЕНИЕ**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Имя файла: kmp.cpp

#include <iostream>

#include <string>

#include <vector>

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

std::vector<int> prefix;

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

prefix.push\_back(0);

}

int ptr2 = 0;

std::cout << "\033[31m" << "Расчёт префикс-функции" << "\033[0m" << std::endl << std::endl;

std::cout << "prefix[0] == 0 по условию" << std::endl << std::endl;

for (int ptr1 = 1; ptr1 < prefix.size(); ++ptr1) {

while (ptr2 > 0 && pattern[ptr1] != pattern[ptr2]) {

std::cout << "ptr2 > 0 и символы " << pattern[ptr1] << "(" << ptr1 << ") и " << pattern[ptr2] << "(" << ptr2 << ") различны" << std::endl;

std::cout << "ptr2 заменяется на предыдущее значение массива префикс-значений == " << prefix[ptr2 - 1] << std::endl << std::endl;

ptr2 = prefix[ptr2 - 1];

}

if (pattern[ptr1] == pattern[ptr2]) {

std::cout << "Символы на позициях (" << ptr1 << ") и (" << ptr2 << ") совпали" << std::endl;

std::cout << "ptr2 увеличивается на 1" << std::endl;

++ptr2;

std::cout << "prefix[" << ptr1 << "] == " << ptr2 << std::endl;

prefix[ptr1] = ptr2;

std::cout << "ptr1 увеличивается на 1" << std::endl << std::endl;

continue;

}

std::cout << "Символы на позициях (" << ptr1 << ") и (" << ptr2 << ") различны" << std::endl;

std::cout << "prefix[" << ptr1 << "] == 0" << std::endl;

prefix[ptr1] = 0;

std::cout << "ptr1 увеличивается на 1" << std::endl << std::endl;

}

return prefix;

}

std::vector<int> kmp\_algorithm(std::string text, std::string pattern) {

std::vector<int> prefix = prefixFunc(pattern);

std::cout << "Полученные значения префикс функции для каждого символа паттерна:" << std::endl;

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

std::cout << pattern[i] << " ";

}

std::cout << std::endl;

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

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

}

std::cout << std::endl << std::endl;

int ptrPattern = 0;

int ptrText = 0;

std::vector<int> answer;

std::cout << "\033[31m" << "Вычисление вхождений" << "\033[0m" << std::endl << std::endl;

while (ptrText < text.size()) {

while (ptrPattern > 0 && text[ptrText] != pattern[ptrPattern]) {

std::cout << "ptrPattern > 0 и символы " << pattern[ptrText] << "(" << ptrText << ") и " << pattern[ptrPattern] << "(" << ptrPattern << ") различны" << std::endl;

std::cout << "ptrPattern заменяется на предыдущее значение массива префикс-значений == " << prefix[ptrPattern - 1] << std::endl << std::endl;

ptrPattern = prefix[ptrPattern - 1];

}

if (text[ptrText] == pattern[ptrPattern]) {

std::cout << "Символы на позициях текста (" << ptrText << ") и паттерна (" << ptrPattern << ") совпали" << std::endl;

if (ptrPattern == pattern.size() - 1) {

std::cout << "Полное совпадение шаблону в интервале {" << ptrText - (pattern.size() - 1) << ":" << ptrText << "}" << std::endl;

answer.push\_back(ptrText - (pattern.size() - 1));

std::cout << "ptrPattern перемещается в " << prefix[ptrPattern] << std::endl;

ptrPattern = prefix[ptrPattern];

} else {

std::cout << "ptrPattern увеличивается на 1" << std::endl;

++ptrPattern;

}

std::cout << "ptrText увеличивается на 1" << std::endl << std::endl;

++ptrText;

continue;

}

std::cout << "Символы на позициях текста (" << ptrText << ") и паттерна (" << ptrPattern << ") различны" << std::endl;

if (ptrPattern == 0) {

std::cout << "Начало паттерна не совпало с текущей позицией в тексте. ptrText увеличивается на 1." << std::endl << std::endl;

++ptrText;

continue;

}

}

return answer;

}

int main() {

std::string pattern, text;

std::cout << "Введи паттерн:" << std::endl;

std::cin >> pattern;

std::cout << "Введи текст:" << std::endl;

std::cin >> text;

std::cout << std::endl;

std::vector<int> answer = kmp\_algorithm(text, pattern);

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

std::cout << -1 << std::endl;

std::cout << "Вхождений не обнаружено." << std::endl;

return 0;

}

std::cout << "Полученные вхождения:" << std::endl;

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

if (i == answer.size() - 1) {

std::cout << answer[i];

continue;

}

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

}

std::cout << std::endl << std::endl;

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

int flag = 1;

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

if (i >= answer[j] && i < answer[j] + pattern.size()) {

std::cout << "\033[31m" << text[i] << "\033[0m";

flag = 0;

break;

}

}

if (flag) {

std::cout << text[i];

}

}

std::cout << std::endl;

}

Имя файла: shift.cpp

#include <iostream>

#include <string>

#include <vector>

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

std::vector<int> prefix;

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

prefix.push\_back(0);

}

int ptr2 = 0;

std::cout << "\033[31m" << "Расчёт префикс-функции" << "\033[0m" << std::endl << std::endl;

std::cout << "prefix[0] == 0 по условию" << std::endl << std::endl;

for (int ptr1 = 1; ptr1 < prefix.size(); ++ptr1) {

while (ptr2 > 0 && pattern[ptr1] != pattern[ptr2]) {

std::cout << "ptr2 > 0 и символы " << pattern[ptr1] << "(" << ptr1 << ") и " << pattern[ptr2] << "(" << ptr2 << ") различны" << std::endl;

std::cout << "ptr2 заменяется на предыдущее значение массива префикс-значений == " << prefix[ptr2 - 1] << std::endl << std::endl;

ptr2 = prefix[ptr2 - 1];

}

if (pattern[ptr1] == pattern[ptr2]) {

std::cout << "Символы на позициях (" << ptr1 << ") и (" << ptr2 << ") совпали" << std::endl;

std::cout << "ptr2 увеличивается на 1" << std::endl;

++ptr2;

std::cout << "prefix[" << ptr1 << "] == " << ptr2 << std::endl;

prefix[ptr1] = ptr2;

std::cout << "ptr1 увеличивается на 1" << std::endl << std::endl;

continue;

}

std::cout << "Символы на позициях (" << ptr1 << ") и (" << ptr2 << ") различны" << std::endl;

std::cout << "prefix[" << ptr1 << "] == 0" << std::endl;

prefix[ptr1] = 0;

std::cout << "ptr1 увеличивается на 1" << std::endl << std::endl;

}

return prefix;

}

int shift\_algorithm(std::string a, std::string b) {

if (a == b) {

std::cout << "Строки равны" << std::endl;

return 0;

}

if (a.size() != b.size()) {

std::cout << "Размеры строк различны" << std::endl;

return -1;

}

std::vector<int> prefix = prefixFunc(a);

std::cout << "Полученные значения префикс функции для каждого символа паттерна:" << std::endl;

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

std::cout << a[i] << " ";

}

std::cout << std::endl;

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

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

}

std::cout << std::endl << std::endl;

int ptrA = 0;

std::cout << "\033[31m" << "Вычисление сдвига" << "\033[0m" << std::endl << std::endl;

for (int ptrB = 0; ptrB < a.size() \* 2; ++ptrB) {

while (ptrA > 0 && a[ptrA] != b[ptrB % a.size()]) {

std::cout << "ptrA > 0 и символы " << a[ptrA] << "(" << ptrA << ") и " << b[ptrB % a.size()] << "(" << ptrB << ") различны" << std::endl;

std::cout << "ptrA заменяется на предыдущее значение массива префикс-значений == " << prefix[ptrA - 1] << std::endl << std::endl;

ptrA = prefix[ptrA - 1];

}

if (a[ptrA] == b[ptrB % a.size()]) {

if (ptrB >= a.size()) {

std::cout << "Символы на позициях (" << ptrA << ") и (" << ptrB % a.size() << ") - по модулю длины, совпали" << std::endl;

} else {

std::cout << "Символы на позициях (" << ptrA << ") и (" << ptrB % a.size() << ") совпали" << std::endl;

}

if (ptrA == a.size() - 1) {

std::cout << "Полное совпадение" << std::endl << std::endl;

return (a.size()\*2 - (ptrB + 1)) % a.size();

}

std::cout << "ptrA увеличивается на 1" << std::endl;

std::cout << "ptrB увеличивается на 1" << std::endl << std::endl;

++ptrA;

continue;

}

std::cout << "Символы на позициях (" << ptrA << ") и (" << ptrB % a.size() << ") различны" << std::endl;

std::cout << "ptrB увеличивается на 1" << std::endl << std::endl;

}

return -1;

}

int main() {

std::string a, b;

std::cout << "Введи строку A:" << std::endl;

std::cin >> a;

std::cout << "Введи строку B:" << std::endl;

std::cin >> b;

std::cout << std::endl;

int answer = shift\_algorithm(a, b);

if (answer == -1) {

std::cout << -1 << std::endl;

std::cout << "A не является циклическим сдвигом B" << std::endl;

return 0;

}

std::cout << "Смещение на " << answer << std::endl;

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

if (i < answer) {

std::cout << "\033[31m" << a[i] << "\033[0m";

} else {

std::cout << "\033[34m" << a[i] << "\033[0m";

}

}

std::cout << std::endl;

}