**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Гельман П.Е. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

## Цель работы.

Цель данной лабораторной работы состоит в изучении алгоритма Кнута-Морриса-Пратта и его реализации.

## Задание.

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

**Sample Input:**

ab

abab

**Sample Output:**

0,2

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.

**Sample Input:**

defabc

abcdef

**Sample Output:**

3

## Выполнение работы.

Алгоритм Кнута-Морриса-Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке, используя то, что при возникновении несоответствия само слово содержит достаточно информации, чтобы определить, где может начаться следующее совпадение, минуя лишние проверки.

В ходе выполнения работы были реализованы следующие функции:

* vector<int> prefixFunc(const string &Pattern) – префиксная функция, вычисляет для каждого i-го символа строки длину наибольшего собственного префикса подстроки от первого символа до i-го, который также является её суффиксом.
* vector<int> KMT(string P, string T) – реализует алгоритм Кнута-Морриса-Пратта, который эффективно ищет вхождения подстроки P в строке T, используя предварительно вычисленную префикс-функцию. Сначала вычисляется префикс функция, далее сканируем текст T, сверяя символы с P. Если символы совпали – двигаемся дальше. Если несовпадение – используем π[j-1], чтобы избежать лишних сравнений. Тем самым, мы сдвигаемся не на один символ, а на число, лежащее в π[j-1], которое говорит, что этот набор π[j-1] символов относительно первого несовпавшего равен этому же набору первых π[j-1] символов, а они в свою очередь совпадают с последними π[j-1] символами в исходном тексте до первого несовпадающего. Если дошли до конца P, значит, нашли вхождение.
* int KMPcycle(const string& Pattern, const string& Text) - реализует алгоритм КМП для поиска подстроки Pattern в строке Text с учетом циклических сдвигов. В начале работы вычисляется префикс-функция для строки Pattern. Далее выполняется проход по удвоенной длине Text, используя индекс i % t\_len для обеспечения циклического поиска. Если текущий символ строки Text совпадает с соответствующим символом Pattern, индекс j увеличивается, и поиск продолжается. В случае несовпадения выполняется откат индекса j по значениям из префикс-функции pi. Если индекс j достигает длины Pattern, значит, найдено полное совпадение, и функция возвращает индекс начала совпадения, который соответствует количеству циклических сдвигов. Если подстрока не найдена, функция возвращает -1.

Оценка сложности.

1. Общая сложность алгоритма КМП по времени – О(n+m), где n – количество символов в тексте, m – количество символов в подстроке. Эта оценка объясняется тем, что для нахождения значений префикс-функции требуется О(m) времени; для прохода по тексту в функции KMP, чтобы найти вхождения подстроки, необходимо О(n) времени.  
   Сложность по памяти – О(m), так как мы храним вектор pi длин префиксов, равный длине m подстроки.
2. Сложность алгоритма КМП для поиска циклического сдвига – О(2\*n+m), так как требуется в худшем случае дважды проходить по всей длине n входного текста, а также один раз по длине m подстроки для получения вектора префикс-функции. Однако, для определения циклического сдвига необходимо сравнивать текст и подстроку одинаковой длины, иначе в этом не будет смысла, соответственно алгоритм будет работать за О(3n). Сложность по памяти также равна О(m), объясняется хранением вектора длины m.

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | aba  abacaba | 0,4 | Верно |
|  | ksk  mvccs | Подстрока не обнаружена в тексте. [-1] | Верно |
|  | aaaaaa  aaaaaaaaaaaaabaaaaaaaaaa | 0,1,2,3,4,5,6,7,14,15,16,17,18 | Верно |
| 4. | aba  baa | Количество необходимых сдвигов - 1 | Верно |
| 5. | Flower  monkey | Число в ответе больше длины исходного текста. Ошибка, ответ - -1 | Верно |
| 6. | j  ddfjf | Строки разных размеров, А - не циклический сдвиг В: -1 | Верно |

## Выводы.

В ходе лабораторной работы был реализован алгоритм Кнута-Морриса-Пратта для поиска вхождений подстроки в текст, проанализирована его временная сложность и сложность по памяти.

# Приложение А Исходный код программы

Название файла: kmp.cpp

#include <iostream>

#include <string>

#include <vector>

#include <windows.h>

using namespace std;

vector<int> prefixFunc(const string &P) {

cout << " \n[ШАГ 1] Нахождение значения функции pi для строки: " << P << endl;

int p\_len =(int)P.size();

vector<int> pi(p\_len, 0);

int j = 0;

cout << "\n pi[0] = 0 независимо от строки\n";

for (int i = 1; i < p\_len; i++) {

cout << "\n Анализ символа '" << P[i] << "' (позиция " << i << ") в шаблоне." << endl;

while (j > 0 && P[i] != P[j]) {

cout << " Нашли различающиеся символы('" << P[i] <<"' - позиция " << i <<", '" << P[j] <<"' - позиция " << j << "), возвращаемся к предыдущему индексу.\n";

j = pi[j - 1];

cout << " После отката j = " << j << endl;

}

if (P[i] == P[j]) {

cout << " Символы совпадают ('" << P[i] <<"' - позиция " << i <<", '" << P[j] <<"' - позиция " << j << "), увеличиваем индекс j (количество символов с текущем найденном префиксе).\n";

j++;

cout << " j = " << j << endl;

}

cout << " Присваиваем значение индекса j в pi[" << i << "] = " << j << endl;

pi[i] = j;

}

cout << " \n[РЕЗУЛЬТАТ] Префикс-функция вычислена: ";

for (int val : pi) cout << val << " ";

cout << endl;

return pi;

}

vector<int> KMT(string P, string T){

cout << "======= Алгоритм Кнута-Морриса-Пратта. ========\n";

int t\_len = (int) T.size(), p\_len = (int)P.size();

vector<int> pi = prefixFunc(P);

vector<int> ans;

cout << '\n';

int j = 0;

cout << "\n[ШАГ 2] ";

cout << "Продолжаем алгоритм КМП. \n";

for (int i = 0; i < t\_len; i++) {

cout << " Итерация i = " << i << ", символ в строке поиска: " << T[i % t\_len] << endl;

while (j > 0 && T[i] != P[j]) {

cout << "\n Найдены отличающиеся символы в строке и подстроке"

"('" << T[i] <<"' - позиция " << i <<", '" << P[j] <<"' - позиция " << j << "), возвращаемся к индексу"

", который лежит в pi[" << j-1 << "]\n";

j = pi[j - 1];

cout << " Индекс j после отката = " << j << "\n\n";

}

if (T[i] == P[j]) {

cout << " Сравниваемые символы совпадают ('" << T[i] <<"' - позиция " << i <<", '" << P[j] <<"' - позиция " << j << "), производим поиск дальше. ";

j++;

cout << "Текущий индекс подстроки j - " << j << endl;

}

if (j == p\_len) {

cout << "\n Длина входной подстроки совпала с найденной подстрокой, "

"записываем индекс начала подстроки в тексте в результат.\n";

ans.push\_back(i - p\_len + 1);

j = pi[j - 1];

cout << " Новое j = " << j << " (pi[" <<j-1 <<"])\n\n";

}

}

if(ans.empty()){

cout << "[РЕЗУЛЬТАТ] Подстрока не обнаружена в тексте. [-1]\n";

return {-1};

}

return ans;

}

int main() {

SetConsoleOutputCP(CP\_UTF8);

cout << "Введите подстроку и строку, в которой будет производиться поиск: ";

string P;

string T;

cin >> P;

cin >> T;

vector<int> res = KMT(P, T);

if(res[0] != -1) {

cout << "\n[РЕЗУЛЬТАТ] Элементы результирующего вектора: ";

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

if (i > 0) cout << ",";

cout << res[i];

}

cout << endl;

}

return 0;

}

Название файла: KMPcycle.cpp

#include <iostream>

#include <string>

#include <vector>

#include <windows.h>

using namespace std;

// Функция для нахождения префикс-функции для строки Pattern (часть алгоритма КМП)

vector<int> prefixFunc(const string &Pattern) {

cout << " \n[ШАГ 1] Нахождение значения функции pi для строки: " << Pattern << endl;

int p\_len =(int)Pattern.size();

vector<int> pi(p\_len, 0); // вектор pi, который будет хранить значения префикс-функции

int j = 0; // длина текущего префикса, совпадающего с суффиксом

for (int i = 1; i < p\_len; ++i) {

cout << "\n Анализ символа '" << Pattern[i] << "' (позиция " << i << ") в шаблоне." << endl;

while (j > 0 && Pattern[i] != Pattern[j]) {

cout << " Нашли различающиеся символы('" << Pattern[i] <<"' - позиция " << i <<", '" << Pattern[j] <<"' - позиция " << j << "), возвращаемся к предыдущему индексу.\n";

j = pi[j - 1];

cout << " После отката j = " << j << endl;

}

if (Pattern[i] == Pattern[j]) {

cout << " Символы совпадают ('" << Pattern[i] <<"' - позиция " << i <<", '" << Pattern[j] <<"' - позиция " << j << "), увеличиваем индекс j (количество символов с текущем найденном префиксе).\n";

j++;

cout << " j = " << j << endl;

}

cout << " Присваиваем значение индекса j в pi[" << i << "] = " << j << endl;

pi[i] = j;

}

cout << " \n[РЕЗУЛЬТАТ] Префикс-функция вычислена: ";

for (int val : pi) cout << val << " ";

cout << endl;

return pi;

}

// Алгоритм Кнута-Морриса-Пратта

int KMPcycle(const string& Pattern, const string& Text){

cout << "======= Алгоритм Кнута-Морриса-Пратта. ========\n";

int t\_len = (int) Text.size(), p\_len = (int)Pattern.size();

vector<int> pi = prefixFunc(Pattern);

cout << '\n';

int ans = -1;

int j = 0;

cout << "\n\n[ШАГ 2] \n";

cout << "Продолжаем алгоритм КМП. "

"Проходим по удвоенной длине входного текста, далее"

" берем значение индекса по модулю длины исходного текста\n";

for (int i = 0; i < t\_len \* 2; ++i) {

cout << "\n Итерация i = " << i << ", символ в A: " << Text[i % t\_len] << endl;

while (j > 0 && Text[i%t\_len] != Pattern[j]) {

cout << "\n Найдены отличающиеся символы в строке и подстроке"

"('" << Text[i%t\_len] <<"' - позиция " << i%t\_len <<" в А, '" << Pattern[j] <<"' - "

"позиция " << j << " в В), возвращаемся к индексу"

", который лежит в pi[" << j-1 << "]\n";

j = pi[j - 1];

cout << " Индекс j после отката = " << j << endl;

}

if (Text[i%t\_len] == Pattern[j]) {

cout << " Сравниваемые символы совпадают ('" << Text[i%t\_len] <<"' - позиция " << i%t\_len <<" "

"в А, '" << Pattern[j] <<"' - позиция " << j << " в В), производим поиск дальше. ";

j++;

cout << "\n Текущий индекс увеличился j = " << j << endl;

}

if (j == p\_len) {

cout << "\n Длина текущей найденной подстроки равна длине искомой строки,"

"возвращаем индекс, с которого началось совпадение - количество циклических сдвигов.\n";

return (i - p\_len + 1);

}

}

return ans;

}

int main() {

SetConsoleOutputCP(CP\_UTF8);

cout << "Введите две строки A и B: ";

string A, B;

cin >> A >> B;

if (A.size() != B.size()){

cout << "Строки разных размеров, А - не циклический сдвиг В: ";

cout << -1 << endl;

return 0;

}

if (A == B){

cout << "Строки идентичны; idx = " << 0 << endl;

return 0;

}

int ans = KMPcycle(B, A);

if(ans > A.size()){

cout << "Число в ответе больше длины исходного текста. Ошибка, ответ - ";

cout << -1 << endl;

return 0;

}

cout << "Количество необходимых сдвигов - " << ans << endl;

return 0;

}