**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 Кнута-Морриса-Пратта**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Архипов В. А. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2022

**Цель работы.**

Изучение и практическое применение алгоритма поиска подстроки в строке – алгоритма Кнута-Морриса-Пратта.

**Задание 1.**

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*, разделённых запятой, если *P* не входит в *T*, то вывести -1

***Пример входных данных*** ab

abab

***Соответствующие выходные данные***

0, 2

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

﻿ ***Пример входных данных*** defabc

abcdef

***Соответствующие выходные данные***

3

**Основные теоретические положения.**

Префикс-суффиксстроки **–** какое-либо количество (возможно нулевое) символов суффикса строки, полностью совпадающих с префиксом строки и не являющееся этой строкой.

Префикс-функция от строки — массив длин наибольших префикс-

суффиксов для каждой позиции этой строки. То есть для каждого среза строки от [0: 1] до [0: n] - максимальное количество символов суффикса, совпадающее с префиксом.

Алгоритм Кнута — Морриса — Пратта — алгоритм [поиска подстроки в строке](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9D%D0%B0%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0_%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8_%D0%B2_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B5#.D0.9F.D0.BE.D1.81.D1.82.D0.B0.D0.BD.D0.BE.D0.B2.D0.BA.D0.B0_.D0.B7.D0.B0.D0.B4.D0.B0.D1.87.D0.B8). Дан текст *Т* шаблон *Р*, можно построить строку *P*#*T* где «#» - разделитель не входящий в алфавит. Тогда при взятии префикс-функции от полученной строки индексы элементов префикс-функции, равных длине |*P*|, будут являться индексами конца вхождения подстроки *P* в строку *P*#*T;* Поскольку равенство элемента массива префикс-функции длине подстроки *P* означает полное совпадение образца *P* и части строки *T*, это и будет вхождением подстроки *P* в строку *T.* Поскольку разделитель «#» больше нигде не встречается, то максимальное значение префикс-суффикса построенной строки не может превысить длины образца *P.*

**Выполнение работы.**

Описание алгоритма.

1. **Вычисление префикс функции**

Для начала создается вектор целых чисел prefix, заполненный нулями. Размер вектора равен длине переданной в него строки *str*. Вычисляться значения префикс-функции *prefix*[*i*] будут по очереди от *i =* 1 до *i = n* – 1, где *n* – длина строки *str* (значение *prefix*[*0*] = 0)*.*

Для подсчета текущего значения *prefix*[*i*] мы заводим переменную *j*, обозначающую длину текущего рассматриваемого образца (префикса строки), изначально *j* = *prefix*[*i* - 1].

Смотрим является ли префикс строки длины *j* суффиксом этой же строки. Для этого сравниваем символы *str*[*j*] и *str*[*i*]. Если они совпадают, то полагаем *prefix*[*i*] = *j* + 1 и переходим к следующему индексу *i* + 1. Если же символы отличаются, то уменьшаем длину *j,* полагая её равной *prefix*[*j* – 1], и повторяем этот шаг алгоритма сначала.

Если мы дошли до длины *j* = 0, а символы *str*[*j*] и *str*[*i*] не совпадают, то полагаем *prefix*[*i*] = 0 и переходим к следующему индексу *i* + 1 (так как в нашем случае вектор изначально заполнен нулями, то нам нужно просто перейти к следующему индексу).

1. **Алгоритм Кнута-Морриса-Пратта**

Данный алгоритм должен найти все вхождения подстроки *P* в строку *T.* Хранить индексы начала вхождений мы будем в векторе *result*. Для начала производится склеивание строк *P* и *T* через разделитель, не встречающийся в этих строках. В нашем случае таким разделителем был выбран символ «#». Получаем строку *P#T.* Для данной строки вычисляется значение префикс-функции, которое записывается в вектор *prefix*. Далее происходит итерация по вектору *prefix* (итерация начинается с индекса |*P*| + 1, где |*P*| - длина строки *P,* до конца массива *prefix*), нас будут интересовать элементы вектора *prefix*, значение которых равно длине искомой подстроки *P*. Как только такой элемент был найден, вычисляется индекс вхождения подстроки в строку (делается это по формуле *i* – 2\*|*P*|, *i –* индекс элемента, чье значение равно длине искомой подстроки, |*P*| - длина искомой подстроки), полученное значение записывается в вектор *result*. Если после перебора всех значений вектора prefix не было найдено ни одного значения, удовлетворяющего условию *prefix*[i] = |*P*|, то в вектор result записывается значение -1.

1. **Алгоритм определения циклического сдвига.**

На вход алгоритму подаются две строки a и b. Для начала проверяется равенство длин этих строк. Если строки a и b имеют разную длину, то алгоритм завершает свою работу и возвращает значение -1. Иначе эти строки склеиваются через разделитель, не встречающийся в этих строках. В нашем случае таким разделителем был выбран символ «#». Получаем строку *b#a.* Для полученной строки вычисляем значение префикс-функции, которое будет записано в вектор *prefix.* Рассмотрим значение последнего элемента вектора *prefix*, положим его равным *k*. По определению префикс функции *k* равняется количеству символов префикса строки *b*, который является суффиксом строки *a*. Отсюда можно сделать вид, что подстроки *a*[|*a*| - *k* : -1] и *b*[0 : *k*], где |*а*| - длина строки *а,* совпадают. Осталось проверить подстроки *a*[*0* : |*a*| - *k*] и  *b*[0 : *k*]. Если они совпадут, то строка *a* является циклическим сдвигом *b,* в таком случае будет возвращено число |*a*| - *k.* Если же подстроки не совпали, то *a* не является циклическим сдвигом *b*, и будет возвращено значение -1. Однако этот алгоритм выявляет, является ли строка b левым сдвигом строки a, в то время как она может являться и правым сдвигом. Поэтому после алгоритму подаются строки b и a (в обратном порядке), сделано это для того, чтобы установить, является ли строка b правым сдвигом строки a. В остальном алгоритм работает аналогично описанному выше, за исключением получения итогового ответа. Если после запуска алгоритма второй раз было получено число, отличное от -1, то для того, чтобы получить корректный ответ, необходимо от длины строки a отнять полученное при втором прогоне алгоритма число. Если же после второго запуска алгоритма было получено число -1, то это значит, что строка a не является циклическим сдвигом b.

**Оценка сложности алгоритмов по времени.**

- Алгоритм вычисления префикс функции

На каждом шаге алгоритма значение максимального префикса, совпадающего с суффиксом, увеличивается не более, чем на единицу. Таким образом максимальное значение, которого оно может достичь *n* – 1, где *n* –длина переданной строки. Во внутреннем цикле это значение только уменьшается, поэтому внутренний цикл отработает меньше, чем за *n* операций. Внешний цикл отработает строго за *n* операций. Отсюда получаем сложность O(2*n*) = O(*n*).

- Алгоритм Кнута-Морриса-Пратта.

Вычисление префикс функции для строки *P#T*: O(|*P*| + 1 + |*T*|) = O(|*P*| + |*T*|), *P* и *T* переданные шаблон и текст соответственно.

Итерация по полученному значению префикс-функции O(|*T*|)

Итого O(|*P*| + 2|*T*|) = O(|*P*|+ |*T*|)

- Алгоритм поиска циклического сдвига

Вычисление значения префикс-функции для строки *b#a*: O(|*a*| + 1 + |*b*|) = O(|*a*| + |*b*|), где *a* и *b –* переданные строки.

Сравнение префикса *а* с суффиксом *b* O(|*a*|)

Итого O(2|*a*| + |*b*|) = O(|*a*| + |*b*|)

**Оценка сложности алгоритмов по памяти.**

- Алгоритм вычисления префикс функции

Для решения выделяется массив размером *n*, где *n* – размер передаваемой на вход строки

Итого O(*n*)

- Алгоритм Кнута-Морриса-Пратта.

Создание новой строки путём склеивания через разделитель старых O(|*P*| + 1 + |*T*|) = O(|*P*| + |*T*|), *P* и *T* переданные шаблон и текст соответственно.

Нахождение префикс функции от полученной строки O(|*P*| + |*T*|)

Массив индексов вхождений O(|T|) (худший случай)

Итого O(2|*P*| + 3|*T*|) = O(|*P*| + |*T*|)

- Алгоритм поиска циклического сдвига

Создание новой строки путём склеивания через разделитель старых O(|*a*| + 1 + |*b*|) = O(|*a*| + |*b*|), *a* и *b* переданные на вход строки.

Нахождение префикс функции от полученной строки O(|*a*| + |*b*|)

Итого O(2|*a*| + 2|*b*|) = O(|*a*| + |*b*|)

**Реализованные структуры данных и функции.**

Разработанный программный код см. в приложении А.

* Вычисление префикс-функции

std::vector<int> prefix\_func(std::string str) – вычисляет значение префикс функции для переданной строки.

Аргументы:

str – строка, для которой нужно найти значение префикс-функции

Возвращаемое значение:

Вектор, содержащий значение префикс-функции для строки str (вектор длин максимальных префикс-суффиксов для каждой позиции в строке str).

* Алгоритм Кнута-Морриса-Пратта

std::vector<int> kmp (std::string pattern, std::string text) – находит все индексы вхождений подстроки pattern в строку text.

Аргументы:

pattern – образец строки, вхождения которого нужно найти

text – строка, в которой нужно искать вхождения образца pattern

Возвращаемое значение:

Вектор индексов вхождения образца pattern в строку text. Если вхождений нет, то вектор содержит -1.

* Алгоритм поиска циклического сдвига

int is\_cycle (std::string a, std::string b) – определяет индекс начала строки b в a, если a является циклическим сдвигом b

Аргументы:

a – строка, потенциально являющаяся циклическим сдвигом строки b

b – искомая строка

Возвращаемое значение:

Число – индекс начала строки b в a, если a является циклическим сдвигом b, (в противном случае будет возвращено значение -1).

* Для вывода отладочной информации предусмотрена булева глобальная переменна DEBUG (значение true соответствует выводу отладочной информации, false – выводу только ответа на задание)
* Так же для представления отладочных данных в более удобном для зрительного восприятия виде были реализованы функwии print\_tab и print\_space
* void print\_tab(int t) – печатает указанное число табуляций на экран

Аргументы:

t - число табуляций, которое нужно вывести на экран

Возвращаемое значение:

Функция ничего не возвращает

* void print\_space(int s) - печатает указанное число пробелов на экран

Аргументы:

s - число пробелов, которое нужно вывести на экран

Возвращаемое значение:

Функция ничего не возвращает

**Тестирование.**

Результаты тестирования представлены в таблица ниже (см. табл. 1).

Таблица 1- тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выход | Комментарий |
| Алгоритм Кнута-Морриса-Пратта | | | |
| 1 | Ken  MoneyKenJuicyKen | 5,13 | Программа работает корректно |
| 2 | ha  Lipsi ha Lipsi ha Lipsi give me money ha | 6,15,38 | Программа работает корректно |
| 3 | love  somewhere its not | -1 | Программа работает корректно |
| 4 | aa  aaaaaaaaaa | 1,2,3,4,5,6,7,8 | Программа работает корректно |
| 5 | FaceTime  FaceTime | 0 | Программа работает корректно |
| 6 | AG  ATTCCGTCGAG | 9 | Программа работает корректно |
| Алгоритм поиска циклического сдвига | | | |
| 1 | aaa  aaa | 0 | Программа работает корректно |
| 2 | aaa  aaaa | -1 | Программа работает корректно |
| 3 | amogus  gusamo | 3 | Программа работает корректно |
| 4 | i’m sorry  sorry for what? | -1 | Программа работает корректно |
| 5 | red marker  arkerred m | 5 | Программа работает корректно |
| 6 | slow motion  low motions | 1 | Программа работает корректно |

**Выводы**

В ходе выполнения лабораторной работы был реализован алгоритм Кнута-Морриса-Пратта для решения задачи поиска подстроки в строке и для решения задачи об определении циклического сдвига одной строки относительно другой.

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: main.cpp

#include <iostream>  
#include <vector>  
#include <string>  
#include <algorithm>  
#include "prefix\_func.h"  
#include "is\_cycle.h"  
#include "KMP.h"  
  
  
int main() {  
 std::string pattern = "Ken";  
 *//std::cin >> pattern;* std::string text = "MoneyKenJuicyKen";  
 *//std::cin >> text;  
  
// Свдиг  
// int res = is\_cycle(pattern, text);  
// if (res == -1){  
// res = is\_cycle(text,pattern);  
// if(res != -1){  
// res = pattern.length() - res;  
// }  
// }  
// std::cout << res << std::endl;  
  
 //Поиск образца* std::vector<int> result = kmp(pattern, text);  
  
  
 for(int i = 0; i < result.size() - 1; i ++){  
 std::cout << result[i] << ",";  
 }  
 std::cout << result.back() << std::endl;  
  
  
 return 0;  
}

Название файла: is\_cycle.h

#ifndef **UNTITLED\_IS\_CYCLE\_H**#define **UNTITLED\_IS\_CYCLE\_H**#include "KMP.h"  
int is\_cycle (std::string a, std::string b);  
  
#endif *//UNTITLED\_IS\_CYCLE\_H*

Название файла: kmp.h

#ifndef **UNTITLED\_KMP\_H**#define **UNTITLED\_KMP\_H**#include <vector>  
#include <string>  
#include "prefix\_func.h"  
  
std::vector<int> kmp (std::string pattern, std::string text);  
  
#endif *//UNTITLED\_KMP\_H*

Название файла: prefix\_func.h

#ifndef **UNTITLED\_PREFIX\_FUNC\_H**#define **UNTITLED\_PREFIX\_FUNC\_H**#include <vector>  
#include <iostream>  
#include <string>  
  
extern bool DEBUG;  
std::vector<int> prefix\_func(std::string str);  
void print\_tab(int t);  
void print\_space(int s);  
#endif *//UNTITLED\_PREFIX\_FUNC\_H*

Название файла: is\_cycle.cpp

#include "is\_cycle.h"  
  
int is\_cycle (std::string a, std::string b){  
  
 if(DEBUG){  
 std::cout << "Determining if the string \""<< a << "\" is a cyclic shift of the string \"" << b <<"\"\n";  
 }  
  
 *// Если переданные строки разной длины, то функция возвращает -1, так как в таком случае одна подстрока  
 // не может являться циклическим сдвигом другой* if(a.length() != b.length()){  
  
 if(DEBUG){  
 print\_tab(1);  
 std::cout << "Strings have the different length\n";  
 }  
  
 return -1;  
 }  
 *// Вычисляем префикс-функцию для строки, полученной путём склеивания строк через символ #,  
 // не встречающийся в строках* std::vector<int> prefix = prefix\_func(b + "#" + a);  
 *// Рассматриваем последний элемент значения префикс-функции  
 // Это значение будет равняться количеству символов суффикса первой строки, которые совпали с префиксом второй строки* int k = prefix.back();  
  
 if(DEBUG){  
 print\_tab(1);  
 std::cout << "The last value of prefix-function is " << k << std::endl;  
 print\_tab(1);  
 std::cout << "Matching part \"" << a.substr(a.length() - k) << "\"" << std::endl;  
 *//print\_tab(1);* }  
 *// Проверка равенства остаточного суффикса второй строки и остаточного префикса первой строки  
 // Если рассматриваемые подстроки равны, то одна строка является циклическим сдвигом другой* if(DEBUG){  
 print\_tab(2);  
 std::cout << "Compare the remain parts: \"" << b.substr(k) << "\" and \"" << a.substr(0, a.length() - k)<< "\"\n";  
 }  
  
 if(b.substr(k) == a.substr(0, a.length() - k)){  
 *//Возвращаем смещени второй строки относительно первой* if(DEBUG){  
 print\_tab(2);  
 std::cout << "Cuts are equal\n";  
 }  
  
 return a.length() - k;  
 }  
 *//Если рассматриваемые подстроки не совпали, то одна строка не является циклическим сдвигом второй  
 //Возвращаем -1* else{  
 if(DEBUG){  
 print\_tab(2);  
 std::cout << "Cuts are not equal\n";  
 }  
 return -1;  
 }  
}

Название файла: kmp.cpp

#include "KMP.h"  
  
std::vector<int> kmp (std::string pattern, std::string text){  
  
 if(DEBUG){  
 std::cout << "Search for substring " << '"'<< pattern << "\" in string " <<'"'<< text << "\"\n" ;  
 }  
 *//Склеивание образца и текста через символ #, не встречающийся ни в одной из введённых строк* std::string pattern\_text = pattern + "#" + text;  
 *//Вычисление значения префикс-функции для полученной строки* std::vector<int> prefix = prefix\_func(pattern\_text);  
  
 *// Инициализация вектора, который будет хранить индексы вхождений образца в текст* std::vector<int> result;  
  
 if(DEBUG){  
 print\_tab(1);  
 std::cout << "Iteration over the values of the prefix function\n";  
 }  
 *//Итерация по элементам значения префикс-функции* for (int i = pattern.length() + 1; i < prefix.size(); i++){  
  
 if(DEBUG){  
 if(i >= 2\*pattern.length()) {  
 print\_tab(2);  
 std::cout << "Current cut: " << text.substr(0, i - pattern.length()) << std::endl;  
 print\_tab(2);  
 print\_space(13 + i - 2\*pattern.length());  
 std::cout << pattern <<"\n";  
 if(prefix[i] != pattern.length()){  
 print\_tab(3);  
 std::cout << "No match found\n\n";  
 }  
 else{  
 print\_tab(3);  
 std::cout << "Match found at symbol " << i - pattern.length() << " (match starts with symbol "  
 << i -2\*pattern.length()<< ")" << "\n\n";  
 }  
 }  
 }  
 *// Если длина префикс-суффикса равнятеся длине образца, то образец входит в данный текст, а конец вхождения  
 // это i-ый символ* if (prefix[i] == pattern.length()){  
 *//Запись индекса начала вхождения образца в строку* result.push\_back(i - 2\*pattern.length());  
 }  
 }  
  
 if(result.empty()){  
 result.push\_back(-1);  
 }  
  
 if(DEBUG){  
 print\_tab(1);  
 std::cout << "Answer: [";  
 for(int j = 0; j < result.size() - 1; j++){  
 std::cout << result[j] << ", ";  
 }  
 std::cout << result.back() <<"]\n\n";  
 }  
 return result;  
}

Название файла: prefix\_func.cpp

#include "prefix\_func.h"  
  
  
bool DEBUG = false;  
  
void print\_tab(int t){  
 while(t > 0) {  
 std::cout << "\t";  
 t--;  
 }  
}  
  
void print\_space(int s){  
 while(s > 0) {  
 std::cout << " ";  
 s--;  
 }  
}  
  
*// Функция вычисления значения префикс функции*std::vector<int> prefix\_func (std::string str){  
 *// n - длина строки, для которой будет найдено значение префикс-функции* int n = str.length();  
  
 if(DEBUG){  
 print\_tab(1);  
 std::cout << "Calculation of the prefix function for the string: " << str << std::endl;  
 }  
 *// Инициализация массива, который будет содержать значение префикс-функции. Сейчас он заполнен нулями* std::vector<int> prefix (n, 0);  
  
 if(DEBUG){  
 print\_tab(2);  
 std::cout << "Current cut: "<< str.substr(0,1) << std::endl;  
 print\_tab(2);  
 std:: cout << "Current length of prefix-suffix for string with right border at 0th symbol: 0\n\n";  
 }  
  
 for (int i = 1; i < n; ++i){  
 *// Значение префикс-функции на предыдущем шаге* int j = prefix[i-1];  
 *// Сравнение символа строки с символом на позиции значения префикс-функции j  
 // Если сравниваемые символы не совпадают и значение префикс-функции на предыдущем шаге больше нуля,  
 // то значение префикс-функции на данном шаге будет равно значению префикс-функции на предыдущем* while (j > 0 && str[i] != str[j]){  
 j = prefix[j-1];  
 }  
 *// Если символы совпали, то значение префикс-функции на данном шаге равняется предыдущему значению  
 // увеличенному на единицу* if (str[i] == str[j]) {  
 prefix[i] = j + 1;  
 }  
  
 if(DEBUG){  
 print\_tab(2);  
 std::cout << "Current cut: "<< str.substr(0,1 + i) << std::endl;  
 print\_tab(2);  
 std:: cout << "Current length of prefix-suffix for string with right border at " << i << "th symbol: "<< prefix[i] << "\n\n";  
 }  
 }  
  
 if(DEBUG){  
 print\_tab(1);  
 std::cout << "Prefix-Suffix function value: [";  
 for(int j = 0; j < prefix.size() - 1; j++){  
 std::cout << prefix[j] << ", ";  
 }  
 std::cout << prefix.back() <<"]\n\n";  
 }  
 return prefix;  
}