**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. 7382 |  | Головина Е.С. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2019

**Цель работы.**

Изучить алгоритм Кнута-Морриса-Пратта, научиться применять его для поиска количества вхождений шаблона в строку и для проверки на циклический сдвиг.

**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*, разделенных запятой, если *P* не входит в *T*, то вывести −1

Sample Input:

ab

abab

Sample Output:

0,2

**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.

**Индивидуальный вариант №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 Кнута-Морриса-Пратта (для задания 4.1).**

Алгоритм КМП реализован в функции:

void knuthMorrisPratt(std::string pattern).

На вход принимается строка pattern, содержащая шаблон. Текст считывается посимвольно. Для алгоритма используются вспомогательные переменные:

size\_t size\_p - хранящая размер строки pattern;

char sym — хранящая считываемый символ;

bool one\_found — отвечающая за то, найдено ли хотя бы одно вхождение;

size\_t counter — счетчик считанных символов из текста;

std::vector<int> prefix — вектор размера size\_p — хранящий значение префикс функции для каждого символа из pattern;

size\_t first,last — для движения по шаблону, при расчете префикс-функции.

1. Начинается подсчет префикс функции для шаблона. Индексные переменные first и last встают на 0 и 1 позицию в шаблоне соответственно. Подсчет выполняется, пока last не достигнет конца шаблона. После подсчета префикс функции п.5.

2. Если символы шаблона на позициях first и last совпадают — префикс функция для символа на позиции last заполняется значением first+1, first и last переходят вперед. Иначе п.3.

3. Если first на позиции 0 (иначе п.4.) — префикс функция для символа на позиции last заполняется 0 и last переходит вперед. Далее возврат в п.2.

4. first переходит на позицию, которую берет из префикс функции для символа на позиции first-1. Далее возврат в п.2.

5. Подсчет префикс функции завершен. Приступаем к самому алгоритму. Запускаем цикл, который прервется, когда будет больше нечего считывать. Считаем первый символ текста. first будет двигаться по шаблону, изначально поставим эту переменную на позицию 0.

6. Если считанный символ и символ из шаблона на позиции first равны, то происходит переход вперед по шаблону и по тексту. Переход по тексту осуществляется считыванием нового символа. Если достигнут конец шаблона, то шаблон в тексте найден и индекс начала шаблона в строке выведем на экран, а по шаблону перейдем на позицию, найденную по префикс функции для предыдущего к first символа.

7. Если считанный символ не равен символу из шаблона на позиции first, то, если first не стоит на 0, то происходит переход по префикс функции для предыдущего к first символа, если first стоит на 0 — переходим вперед по тексту (считываем новый символ).

8. Дойдя до конца текста алгоритм завершает свою работу и если не найдено ни одно вхождение шаблона — печатает -1.

Сложность алгоритма по памяти: O(P), где P — количество символов в шаблоне. Такая сложность, потому что сохраняется только шаблон.

Сложность алгоритма по операциям: O(P+T), где P — количество символов в шаблоне, T — количество символов в тексте. Такая сложность, потому что в ходе алгоритма движемся вперед по T и по P, когда находим совпадения с текстом, благодаря префикс функции, не приходится возвращаться и проверять совпадения заново.

**Описание алгоритма Кнута-Морриса-Пратта (для задания 4.2).**

Алгоритм КМП реализован в функции:

void knuthMorrisPratt(std::string pattern, std::string text).

На вход принимается строка pattern, содержащая строку B (шаблон) и строку text, содержащую строку A (текст). Для алгоритма используются вспомогательные переменные:

size\_t size\_p - хранящая размер строки A (и соответственно строки B, так как они одной длины);

size\_t checker — счетчик считанных символов из шаблона во время сравнения совпадающих;

std::vector<int> prefix — вектор размера size\_p — хранящий значение префикс функции для каждого символа из pattern;

size\_t first,last — для движения по шаблону, при расчете префикс-функции.

1. Начинается подсчет префикс функции для шаблона (строки B). Индексные переменные first и last встают на 0 и 1 позицию в шаблоне соответственно. Подсчет выполняется, пока last не достигнет конца шаблона. После подсчета префикс функции п.5.

2. Если символы шаблона на позициях first и last совпадают — префикс функция для символа на позиции last заполняется значением first+1, first и last переходят вперед. Иначе п.3.

3. Если first на позиции 0 (иначе п.4.) — префикс функция для символа на позиции last заполняется 0 и last переходит вперед. Далее возврат в п.2.

4. first переходит на позицию, которую берет из префикс функции для символа на позиции first-1. Далее возврат в п.2.

5. Подсчет префикс функции завершен. Приступаем к самому алгоритму. Запускаем цикл, который прервется, когда будет достигнут конец текста (шаблона A). first будет двигаться по тексту B, last по тексту A, изначально поставим их на позицию 0 в текстах B и A соответственно.

6. Если текущие символы из текста B и A равны, то запускается цикл, который будет длиться, пока символы будут равны. Иначе п.8.

7. Внутри второго цикла переходим вперед (циклически) по тексту A и B. В checker считаем количество сверенных символов, когда оно достигнет количества символов в строках — значит A — циклический сдвиг B. И выведем индекс начала B в A. Если на каком-то этапе символы не совпадают — п.8.

8. Если символы не совпадают. Если в тексте B мы сейчас на 0 позиции, то переходим вперед по тексту A, иначе перейдем по тексту B на позицию, найденную по префикс функции для предыдущего к first символа.

Сложность алгоритма по памяти: O(P), где P — количество символов в каждом слове. Такая сложность, потому что обе строки одного размера.

Сложность алгоритма по операциям: O(P+P), где P — количество символов в слове B, T — количество символов в слове A. Такая сложность, потому что в ходе алгоритма движемся вперед по T и по P, когда находим совпадения со строкой A, благодаря префикс функции, не приходится возвращаться и проверять совпадения заново.

Описание функций и структур данных (4.1):

1. void knuthMorrisPratt(std::string pattern)

Функция, реализующая подсчет префикс функции и алгоритм КМП.

На вход принимает:

std::string pattern — строка шаблона.

Описание функций и структур данных (4.2):

1. void knuthMorrisPratt(std::string pattern, std::string text)

Функция, реализующая подсчет префикс функции и алгоритм КМП.

На вход принимает:

std::string pattern — строка текста B;

std::string text — строка текста A.

Тестирование 4.1.

Было сделано 5 тестов для демонстрации и проверки работы программы.

|  |  |  |
| --- | --- | --- |
| Тест № | Данные | Результат |
| 1 | ab  abababba | 0, 2, 4 |
| 2 | abd  abdbabasbbabdab | 0, 10 |
| 3 | abaab  ababaabaabdf | 2, 5 |
| 4 | 121  121212123 | 0, 2, 4 |
| 5 | ab  aaaaaaaaaaaaaaab | 14 |

Выберем 3 тест для детального обзора.

abaab

ababaabaabdf

Префикс функция для шаблона:

0 0 1 1 2

a b a a b

1. Шаблон на символе a, текст на символе a. Символы совпадают. Движемся вперед.

2. Шаблон на символе b, текст на символе b. Символы совпадают. Движемся вперед.

3. Шаблон на символе a, текст на символе a. Символы совпадают. Движемся вперед.

4. Шаблон на символе a, текст на символе b. Символы не совпадают. Так как шаблон не на первом символе происходит переход по префикс функции предыдущего символа, а именно на позицию 1.

5. Шаблон на символе b, текст на символе b. Символы совпадают. Движемся вперед.

6. Шаблон на символе a, текст на символе a. Символы совпадают. Движемся вперед.

7. Шаблон на символе a, текст на символе a. Символы совпадают. Движемся вперед.

8. Шаблон на символе b, текст на символе b. Символы совпадают. Шаблон найден, в тексте он начинается с позиции 2. По префикс функции в шаблоне переходим на позицию 2 Идем дальше.

9. Шаблон на символе a, текст на символе a. Символы совпадают. Движемся вперед.

10. Шаблон на символе a, текст на символе a. Символы совпадают. Движемся вперед.

11. Шаблон на символе b, текст на символе b. Символы совпадают. Шаблон найден, в тексте он начинается с позиции 5. По префикс функции в шаблоне переходим на позицию 2. Идем дальше.

12. Шаблон на символе a, текст на символе d. Символы не совпадают. Так как шаблон не на первом символе происходит переход по префикс функции предыдущего символа, а именно на позицию 0.

13. Шаблон на символе a, текст на символе d. Символы не совпадают. Так как шаблон на первом символе происходит переход по вперед только по тексту.

14. Шаблон на символе a, текст на символе f. Символы не совпадают.

15. Достигнут конец строки текста. Найденные позиции вхождения шаблона в текст: 2 и 5.

Тестирование 4.2.

Было сделано 5 тестов для демонстрации и проверки работы программы.

|  |  |  |
| --- | --- | --- |
| Тест № | Данные | Результат |
| 1 | aeaeae  eaeaea | 1 |
| 2 | defabc  abcdef | 3 |
| 3 | abaca  acaab | 2 |
| 4 | badca  bcaba | -1 |
| 5 | afdcb  bafdc | 4 |

Выберем 1 тест для детального обзора.

aeaeae

eaeaea

Префикс функция для строки B:

e a e a e a

0 0 1 2 3 4

1. Текст B на символе e, текст A на символе a. Символы не совпадают. Так как текст B на первом символе происходит переход по вперед только по тексту A.

2. Текст B на символе e, текст A на символе e. Символы совпадают. Движемся вперед.

3. Текст B на символе a, текст A на символе a. Символы совпадают. Движемся вперед.

4. Текст B на символе e, текст A на символе e. Символы совпадают. Движемся вперед.

5. Текст B на символе a, текст A на символе a. Символы совпадают. Движемся вперед.

6. Текст B на символе e, текст A на символе e. Символы совпадают. Движемся вперед.

7. Текст B на символе a, текст A на символе a. Символы совпадают. Движемся вперед.

8. Совпадений столько же сколько символов в каждом тексте – A циклический сдвиг B с позиции 1.

**Вывод**

В ходе лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта и найдены способы его применения для поиска количества вхождений шаблона в строку и для проверки на циклический сдвиг.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД 4.1**

#include <iostream>

#include <string>

#include <vector>

#include <cctype>

void knuthMorrisPratt(std::string pattern){

size\_t size\_p = pattern.size();

char sym;

bool one\_found=false;

size\_t counter=0;

std::vector<int> prefix(size\_p,0);

//вычисление префикс функции для всех символов в pattern

size\_t first=0,last=1;

while (last<size\_p){

if (pattern[first] == pattern[last]){

prefix[last]=first+1;

first++;

last++;

}

else{

if (first==0){

prefix[last]=0;

last++;

}

else first=prefix[first-1];

}

}

counter = first = 0;

std::cin>>sym;

while (1){

//если символы в строке и шаблоне совпали,

//двигаемся вперед по строке и по шаблону

if (pattern[first] == sym ){

first++;

counter++;

//выводим результат, если шаблон найден

if (first == size\_p){

if (!one\_found){

std::cout<<counter-size\_p;

one\_found=true;

}

else std::cout<<", "<<counter-size\_p;

first = prefix[first-1];

}

if (!(std::cin >> sym)) break;

}

else{

//если символы не совпали переход по префикс функции

//только если не находимся на начале шаблона

if (first != 0) first=prefix[first-1];

else{

counter++;

if (!(std::cin >> sym)) break;

}

}

}

//если не найдено вхождений вывод -1

if (!one\_found) std::cout<<-1;

}

int main(){

std::string pattern;

std::cin >> pattern;

knuthMorrisPratt(pattern);

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД 4.2**

#include <iostream>

#include <string>

#include <vector>

#include <cstddef>

ptrdiff\_t knuthMorrisPratt(std::string pattern, std::string text){

size\_t size\_p = pattern.size();

std::vector<int> prefix(size\_p,0);

//вычисление префикс функции для всех символов в pattern(text B)

size\_t first=0,last=1;

while (last<size\_p){

if (pattern[first] == pattern[last]){

prefix[last]=first+1;

first++;

last++;

}

else{

if (first==0){

prefix[last]=0;

last++;

}

else first=prefix[first-1];

}

}

//индексы по словам A и В изначально на нуле

last = first = 0;

size\_t checker=0;

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

//если в словах текущие буквы совпали - запуск цикла

while (pattern[first] == text[last]){

checker++;

//переходим вперед без выхода за границы слов

first=(first+1) % size\_p;

last=(last+1) % size\_p;

//checker считает количество букв и если их

//становится равно длине слова - B циклический сдвиг А

if (checker == size\_p) return (last+size\_p-checker) % size\_p;

//чтоб корректно идти по слову A после выхода из цикла while

i++;

}

checker=0;

//если текущие буквы в словах не совпали - переход

//по префикс функции слова B

if (pattern[first] != text[last]){

if (first != 0){

first=prefix[first-1];

//т.к. не движемся по слову A, для корректности

i--;

}

//если не с чем больше сравнить текущую букву

//в слове А, движемся по нему вперед

else last++;

}

}

//если не циклический сдвиг возвращаем -1

return -1;

}

int main(){

std::string text1,text2;

std::cin >> text2 >> text1;

if (text1.size() != text2.size()){

std::cout << -1;

return 0;

}

std::cout << knuthMorrisPratt(text1,text2) << '\n';

return 0;

}