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

Вариант 4

Студент гр. 8382 Торосян Т.А.

Преподаватель Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить принцип работы алгоритма Кнута-Морриса-Пратта и алгоритм поиска циклического сдвига, реализовать их на практике для обработки строковых данных.

Алгоритм КМП

Постановка задачи

Реализуйте алгоритм КМП и с его помощью для заданных шаблона $P(|P|\leq 15000)$ и текста $T(|T|\leq 5000000)$ найдите все вхождения P в T.

Входные данные:

Первая строка — P

Вторая строка — T

Выходные данные:

индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1

Пример:

Ввод:

ab

abab

Вывод:

0, 2

Индивидуализация

Вариант 1. Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

Префикс-функция

Префикс-функция от строки представлена массивом pi, где pi[i] показывает длинумаксимального префикса строки s[0..i], совпадающего с

суффиксом той же длины.

Распараллеливание задачи

Для того, чтобы алгоритм мог работать параллельно в нескольких потоках, необходимо разбить исходный текст на несколько частей. В данной реализации число частей задается пользователем. На это количество равных частей и делится текст. Затем каждая часть кроме последней увеличивается на число символов, равное длине шаблона за вычетом единицы. Это делается для того, чтобы учесть пограничные условия при проверке на входение шаблона.

Описание алгоритма

На вход алгоритму подаются шаблон и текст. Алгоритм должен найти все вхождения шаблона.

В первую очередь вычисляется префикс-функция шаблона, которая заносится в массив pi.

Затем алгоритм начинает сравнивать шаблон и текст. Двигаясь от начал строк, при каждом совпадении символов счетчики положения шаблона и текста увеличиваются на единицу. Как только счетчик шаблона сравнялся с его длиной, что говорит о том, что вхождение найдено, в массив ответов заносится разность счетчиков шаблона и текста, что равно индексу вхождения. Если же символы не совпали, алгоритм откатывается к концу предыдущего совпавшего префикса. Алгоритм завершает работу при достижении конца текста.

Описание функций

- std::vector<int> prefixFunction (std::string s) – вычисляет префикс-функцию и возвращает в виде массива значений.
    - std::string s — входная строка
- void createPart(const std::string& text, std::string& part, int countSymbols, int index) — «вытаскивает» из исходного текста заданное количество символов и помещает их в другую строку.
    - const std::string& text — исходный текст

    - std::string& part — строка-контейнер для извлечённой части
    - int countSymbols — длина извлекаемого отрезка
    - int index — индекс первого символа извлекаемого отрезка
- std::vector<std::string> split(const std::string& text, const std::string& pattern, int lengthSmallPart, int k) — разбивает текст на необходимое кол-во отрезков для многопоточной обработки
    - const std::string& text — исходный текст
    - const std::string& pattern — строка паттерна, её длина учитывается при разбиении текста
    - int lengthSmallPart – ожидаемая длина части текста без учета длины паттерна.
    - int k — ожидаемое количество частей
- void KMP() - функция-обёртка для алгоритма КМП, осуществляющая первичный ввод-вывод и пре-инициализацию алгоритма.
- std::vector<int> KMP(const std::string& text, const std::string& pattern, const std::vector<int>& pi) — основная функция алгоритма, выполняет поиск паттерна в заданном тексте.
    - const std::string& text - участок текста, в котором осуществляется поиск
    - const std::string& pattern — строка-паттерн
    - std::vector<int>& pi — префикс-функция в виде массива
- void cycleShift() - функция-обёртка для решателя задачи циклического сдвига. Реализует ввод входных данных через консоль.
- void cycleShift(std::string a, std::string b) — функция, которая реализует поиск циклического сдвига.
    - std::string a — строка А
    - std::string b — строка Б.

Сложность алгоритма по времени

Для алгоритма КМП сложность равна O(m+n), где m и n – длины текста и шаблона соотв-нно.

Сложность алгоритма по памяти

.Сложность по памяти — O(m), т. к. по условию варианта длина шаблона незначительно мала относительно размера текста.

Тестирование

Тест 1

Enter a pattern:

test

Enter text:

ctesfestestesteeette

Enter the number of parts to search:

3

Text will be divided into 3 parts

Text is divided into 2 big parts with length = 10,

into 0 small parts with length = 9

and into final part with length = 6:

ctesfestes testesteee eeette

PrefixFunction calculation:

Pattern: t e s t

pi = [0]

s[1] != s[0] ;

Pattern: t e s t

pi = [0, 0]

s[2] != s[0] ;

Pattern: t e s t

pi = [0, 0, 0]

s[3] == s[0]

Pattern: t e s t

pi = [0, 0, 0, 1]

<) Part #1 (>

KMP for part: c t e s f e s t e s

text[0] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[1] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[2] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[3] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[4] != pattern[3] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[4] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[5] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[6] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[7] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[8] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[9] == pattern[2] :: Continuing, textIndex++; patternIndex++

<)==========(>

<) Part #2 (>

KMP for part: t e s t e s t e e e

text[0] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[1] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[2] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[3] == pattern[3] :: Continuing, textIndex++; patternIndex++

! Substring found at text[0] !

text[4] != pattern[4] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 1

text[4] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[5] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[6] == pattern[3] :: Continuing, textIndex++; patternIndex++

! Substring found at text[3] !

text[7] != pattern[4] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 1

text[7] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[8] != pattern[2] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[8] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[9] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

<)==========(>

<) Part #3 (>

KMP for part: e e e t t e

text[0] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[1] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[2] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[3] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[4] != pattern[1] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[4] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[5] == pattern[1] :: Continuing, textIndex++; patternIndex++

<)==========(>

All occurrences of the pattern in text:

7,10

Тест 2

Enter a pattern:

qweeeeeee

Enter text:

qwww

Text is less than pattern!

Тест 3

Enter a pattern:

word

Enter text:

wowooooooord

Enter the number of parts to search:

100500

Length of one part can't be less than pattern size!

Тест 4

Enter a pattern:

geez

Enter text:

bungeezzzgeeezgezgeegeezgee

Enter the number of parts to search:

3

Text will be divided into 3 parts

Text is divided into 0 big parts with length = 13,

into 2 small parts with length = 12

and into final part with length = 9:

bungeezzzgee geeezgezgeeg eegeezgee

PrefixFunction calculation:

Pattern: g e e z

pi = [0]

s[1] != s[0] ;

Pattern: g e e z

pi = [0, 0]

s[2] != s[0] ;

Pattern: g e e z

pi = [0, 0, 0]

s[3] != s[0] ;

Pattern: g e e z

pi = [0, 0, 0, 0]

<) Part #1 (>

KMP for part: b u n g e e z z z g e e

text[0] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[1] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[2] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[3] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[4] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[5] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[6] == pattern[3] :: Continuing, textIndex++; patternIndex++

! Substring found at text[3] !

text[7] != pattern[4] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[7] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[8] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[9] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[10] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[11] == pattern[2] :: Continuing, textIndex++; patternIndex++

<)==========(>

<) Part #2 (>

KMP for part: g e e e z g e z g e e g

text[0] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[1] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[2] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[3] != pattern[3] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[3] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[4] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[5] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[6] == pattern[1] :: Continuing, textIndex++; patternIndex++

```
text[7] != pattern[2] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0
text[7] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;
text[8] == pattern[0] :: Continuing, textIndex++; patternIndex++
text[9] == pattern[1] :: Continuing, textIndex++; patternIndex++
text[10] == pattern[2] :: Continuing, textIndex++; patternIndex++
text[11] != pattern[3] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0
text[11] == pattern[0] :: Continuing, textIndex++; patternIndex++

<)==========================================(>

        <) Part #3 (>

KMP for part: e e g e e z g e e
text[0] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;
text[1] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;
text[2] == pattern[0] :: Continuing, textIndex++; patternIndex++
text[3] == pattern[1] :: Continuing, textIndex++; patternIndex++
text[4] == pattern[2] :: Continuing, textIndex++; patternIndex++
text[5] == pattern[3] :: Continuing, textIndex++; patternIndex++
! Substring found at text[2] !
text[6] != pattern[4] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0
text[6] == pattern[0] :: Continuing, textIndex++; patternIndex++
text[7] == pattern[1] :: Continuing, textIndex++; patternIndex++
text[8] == pattern[2] :: Continuing, textIndex++; patternIndex++

<)==========================================(>

All occurrences of the pattern in text:
3,20
```

Циклический сдвиг

Постановка задачи

Заданы две строки

Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и A состоит из суффикса B, склеенного с префиксом B). 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.

Пример ввода:

defabc

abcdef

Пример вывода:

3

Описание алгоритма

На вход поступают две строки. Сначала сравниваются размеры строк: если они разные, на этом можно завершить работу алгоритма.

Чтобы проверить строки на наличие циклического сдвига, можно дополнить строку A её копией и применить к ней поиск с помощью КМП по шаблону B.

В результате КМП вернёт массив индексов вхождений, первое из которых и будет ответом.

Сложность алгоритма по времени

Для алгоритма КМП сложность равна O(n), n – длина строки.

Сложность алгоритма по памяти

.Сложность по памяти — O(т), т. к. хранятся только исходные строки плюс копия строки A.

Тестирование

Тест 1

Enter string a:

qwerty

Enter string b:

tyqwer

String a += a: qwertyqwerty

PrefixFunction calculation:

Pattern: t y q w e r

pi = [0]

s[1] != s[0] ;

Pattern: t y q w e r

pi = [0, 0]

s[2] != s[0] ;

Pattern: t y q w e r

pi = [0, 0, 0]

s[3] != s[0] ;

Pattern: t y q w e r

pi = [0, 0, 0, 0]

s[4] != s[0] ;

Pattern: t y q w e r

pi = [0, 0, 0, 0, 0]

s[5] != s[0] ;

Pattern: t y q w e r

pi = [0, 0, 0, 0, 0, 0]

KMP for part: q w e r t y q w e r t y

text[0] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[1] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[2] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[3] != pattern[0] :: Continuing, textIndex++, patternIndex is still 0;

text[4] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[5] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[6] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[7] == pattern[3] :: Continuing, textIndex++; patternIndex++

text[8] == pattern[4] :: Continuing, textIndex++; patternIndex++

text[9] == pattern[5] :: Continuing, textIndex++; patternIndex++

! Substring found at text[4] !

text[10] != pattern[6] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[10] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[11] == pattern[1] :: Continuing, textIndex++; patternIndex++

B string start index in A: 4

Тест 2

Enter string a:

aaa

Enter string b:

aaa

String a += a: aaaaaa

PrefixFunction calculation:

Pattern: a a a

pi = [0]

s[1] == s[0]

Pattern: a a a

pi = [0, 1]

s[2] == s[1]

Pattern: a a a

pi = [0, 1, 2]

KMP for part: a a a a a a

text[0] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[1] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[2] == pattern[2] :: Continuing, textIndex++; patternIndex++

! Substring found at text[0] !

text[3] != pattern[3] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 2

text[3] == pattern[2] :: Continuing, textIndex++; patternIndex++

! Substring found at text[1] !

text[4] != pattern[3] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 2

text[4] == pattern[2] :: Continuing, textIndex++; patternIndex++

! Substring found at text[2] !

text[5] != pattern[3] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 2

text[5] == pattern[2] :: Continuing, textIndex++; patternIndex++

! Substring found at text[3] !

B string start index in A: 0

Тест 3

Enter string a:

aabaabaa

Enter string b:

abaaaaba

String a += a: aabaabaaaabaabaa

PrefixFunction calculation:

Pattern: a b a a a a b a

pi = [0]

s[1] != s[0] ;

Pattern: a b a a a a b a

pi = [0, 0]

s[2] == s[0]

Pattern: a b a a a a b a

pi = [0, 0, 1]

s[3] != s[1] ; j => 0

Pattern: a b a a a a b a

pi = [0, 0, 1]

s[3] == s[0]

Pattern: a b a a a a b a

pi = [0, 0, 1, 1]

s[4] != s[1] ; j => 0

Pattern: a b a a a a b a

pi = [0, 0, 1, 1]

s[4] == s[0]

Pattern: a b a a a a b a

pi = [0, 0, 1, 1, 1]

s[5] != s[1] ; j => 0

Pattern: a b a a a a b a

pi = [0, 0, 1, 1, 1]

s[5] == s[0]

Pattern: a b a a a a b a

pi = [0, 0, 1, 1, 1, 1]

s[6] == s[1]

Pattern: a b a a a a b a

pi = [0, 0, 1, 1, 1, 1, 2]

s[7] == s[2]

Pattern: a b a a a a b a

pi = [0, 0, 1, 1, 1, 1, 2, 3]

KMP for part: a a b a a b a a a a b a a b a a

text[0] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[1] != pattern[1] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 0

text[1] == pattern[0] :: Continuing, textIndex++; patternIndex++

text[2] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[3] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[4] == pattern[3] :: Continuing, textIndex++; patternIndex++

text[5] != pattern[4] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 1

text[5] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[6] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[7] == pattern[3] :: Continuing, textIndex++; patternIndex++

text[8] == pattern[4] :: Continuing, textIndex++; patternIndex++

text[9] == pattern[5] :: Continuing, textIndex++; patternIndex++

text[10] == pattern[6] :: Continuing, textIndex++; patternIndex++

text[11] == pattern[7] :: Continuing, textIndex++; patternIndex++

! Substring found at text[4] !

text[12] != pattern[8] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 3

text[12] == pattern[3] :: Continuing, textIndex++; patternIndex++

text[13] != pattern[4] :: Backtracking patternIndex with prefix func: pi[patternIndex-1] == 1

text[13] == pattern[1] :: Continuing, textIndex++; patternIndex++

text[14] == pattern[2] :: Continuing, textIndex++; patternIndex++

text[15] == pattern[3] :: Continuing, textIndex++; patternIndex++

B string start index in A: 4

Вывод

В ходе выполнения лабораторной работы были изучены и реализованы алгоритмы Кнута-Морриса-Пратта и поиска циклического сдвига. Получено представление о работе префикс-функции.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

ФАЙЛ main.cpp

```
#include <iostream>
#include <vector>
#include <string>

#define OUTPUT

std::vector<int> prefixFunction (std::string s)
{
    std::vector<int> pi;
    pi.push_back(0);

#ifdef OUTPUT
    std::cout << "PrefixFunction calculation:" << std::endl;
    std::cout << "Pattern: ";
    for(auto ch: s)
        std::cout << ch << " ";
    std::cout << std::endl;
    std::cout << "pi = [0]" << std::endl;
#endif


    for(int i = 1, j = 0; i < s.size(); i++) {

        if(s[i] == s[j]) {
#ifdef OUTPUT
            std::cout << "s[" << i << "]" << " == " << "s[" << j << "] " << std::endl;
#endif
            pi.push_back(j + 1);
            j++;
        } else {
#ifdef OUTPUT
            std::cout << "s[" << i << "]" << " != " << "s[" << j << "] " << "; ";
#endif
            if(j == 0) {
                pi.push_back(0);
            } else {
```

```
#ifdef OUTPUT
                std::cout << "j => " << pi[j-1] << std::endl;
#endif
                j = pi[j-1];
                i--;
            }
        }

#ifdef OUTPUT
        std::cout << std::endl;
        std::cout << "Pattern: ";
        for(auto ch: s)
            std::cout << ch << " ";
        std::cout << std::endl;
        std::cout << "pi = [";
        for(int i = 0 ; i < pi.size() - 1; i++)
            std::cout << pi[i] << ", ";
        std::cout << pi[pi.size() - 1];
        std::cout << "]" << std::endl;
#endif
    }

    return pi;
}

void createPart(const std::string& text, std::string& part, int countSymbols, int index)
{
    for(int i = index, j = 0 ; i < index + countSymbols; i++, j++)
        part.push_back(text[i]);
}

std::vector<std::string> split(const std::string& text, const std::string& pattern, int lengthSmallPart, int k)
{
    std::vector<std::string> parts;
    if(k == 1) {
        parts.push_back(text);
        return parts;
    }
    int lengthText = text.size();
    int lengthPattern = pattern.size();

    int lengthBigPart = lengthSmallPart + 1;
```

```
    int countBigParts = lengthText % k;
    int countSmallParts = k - countBigParts;

    int lengthExtendedBigPart = lengthBigPart + (lengthPattern - 1);
    int lengthExtendedSmallPart = lengthSmallPart + (lengthPattern - 1);

    int i, indexText = 0;
    for(i = 0; i < countBigParts; i++, indexText += lengthBigPart) {
        std::string part;
        createPart(text, part, lengthExtendedBigPart, indexText);
        parts.push_back(part);
    }

    for(i = 0; i < countSmallParts - 1; i++, indexText += lengthSmallPart) {
        std::string part;
        createPart(text, part, lengthExtendedSmallPart, indexText);
        parts.push_back(part);
    }

    std::string finalPart;
    int lengthFinalPart = lengthText - indexText;
    createPart(text, finalPart, lengthFinalPart, indexText);
    parts.push_back(finalPart);

#ifdef OUTPUT
    std::cout << "Text is divided into " << countBigParts << " big parts with length = " << lengthExtendedBigPart << ","<<
std::endl;
    std::cout << "into " << countSmallParts - 1 << " small parts with length = " << lengthExtendedSmallPart << std::endl;
    std::cout << "and into final part with length = " << lengthFinalPart << ":"<< std::endl;
    for(auto part: parts)
        std::cout << part << " ";
    std::cout << std::endl;
#endif

    return parts;
}

std::vector<int> KMP(const std::string& text, const std::string& pattern, const std::vector<int>& pi)
{
    std::vector<int> answer;
#ifdef OUTPUT
```

```
    std::cout << std::endl << "KMP for part: ";
    for(auto ch: text)
        std::cout << ch << " ";
    std::cout << std::endl;
#endif
    int textIndex = 0;
    int patternIndex = 0;
    int lengthText = text.size();
    int lengthPattern = pattern.size();
    while(textIndex < lengthText) {
#ifdef OUTPUT
        std::cout << std::endl << "Part: ";
        for(auto ch: text)
            std::cout << ch << " ";
        std::cout << std::endl;
        std::cout << "Pattern: ";
        for(auto ch: pattern)
            std::cout << ch << " ";
        std::cout << std::endl;
        std::cout << "PrefixFunction of pattern: [";
        for(int i = 0; i < pi.size() - 1; i++)
            std::cout << pi[i] << ", ";
        std::cout << pi[pi.size() - 1] << "]" << std::endl << std::endl;
#endif
        if(text[textIndex] == pattern[patternIndex]) {
#ifdef OUTPUT
            std::cout << "text[" << textIndex << "] == pattern[" << patternIndex << "]" << std::endl;
#endif
            textIndex++;
            patternIndex++;
            if(patternIndex == lengthPattern) {
                answer.push_back(textIndex - patternIndex);
#ifdef OUTPUT
                std::cout << "Substring found!" << std::endl;
#endif
            }
        } else {
#ifdef OUTPUT
            std::cout << "text[" << textIndex << "] != pattern[" << patternIndex << "]" << std::endl;
#endif
            if(patternIndex == 0) {
                textIndex++;
```

```
            } else {
                patternIndex = pi[patternIndex-1];
            }
        }
    }
    return answer;
}

void KMP()
{
    std::vector<int> answer;
    int k;
    std::vector<std::string> parts;
    std::string pattern, text;

#ifdef OUTPUT
    std::cout << "Enter a pattern:" << std::endl;
#endif
    std::cin >> pattern;
#ifdef OUTPUT
    std::cout << "Enter text: " << std::endl;
#endif
    std::cin >> text;

    int lengthText = text.size();
    int lengthPattern = pattern.size();

    if(lengthText < lengthPattern) {
        std::cout << "Text less than pattern!" << std::endl;
        return;
    }
#ifdef OUTPUT
    std::cout << "Enter the number of parts to search: " << std::endl;
#endif
    std::cin >> k;

    int lengthSmallPart = lengthText / k;
    if(lengthSmallPart < pattern.size()) {
        std::cout << "Length one part can not less pattern size!" << std::endl;
        return;
    }
    if(k <= 0) {
```

```
        std::cout << "Count of parts can not equal or less 0!" << std::endl;
        return;
    }

#ifdef OUTPUT
    std::cout << "Text will be divided into " << k << " parts" << std::endl;
#endif
    parts = split(text, pattern, lengthSmallPart, k);

    std::vector<int> pi = prefixFunction(pattern);

    std::vector<int> occurrences;
    for(int i = 0, indexText = 0; i < parts.size(); i++) {
        occurrences = KMP(parts[i], pattern, pi);
        for(int j = 0; j < occurrences.size(); j++) {
            answer.push_back(occurrences[j] + indexText);
        }
        indexText += parts[i].size() - lengthPattern + 1;
    }

    if(answer.empty())
        std::cout << -1;
    else {
#ifdef OUTPUT
        std::cout << "All occurrences of the pattern in text:" << std::endl;
#endif
        for (int i = 0; i < answer.size() - 1; i++)
            std::cout << answer[i] << ",";
        std::cout << answer[answer.size() - 1] << std::endl;
    }

}

void cycleShift(std::string a, std::string b)
{
    if(a.size() != b.size()) {
#ifdef OUTPUT
        std::cout << "Lengths of strings a and b are different!" << std::endl;
#endif
        std::cout << -1;
        return;
    }
```

```
    a += a;
#ifdef OUTPUT
    std::cout << "String a += a: " << a << std::endl;
#endif

    std::vector<int> pi = prefixFunction(b);

    std::vector<int> occurrences = KMP(a, b, pi);

    if(occurrences.empty()) {
#ifdef OUTPUT
        std::cout << "A is not cyclic shift B!" << std::endl;
#endif
        std::cout << "-1";
    } else {
#ifdef OUTPUT
        std::cout << "B string start index in A: ";
#endif
        std::cout << occurrences[0];
    }
}

void cycleShift()
{
    std::string a, b;
#ifdef OUTPUT
    std::cout << "Enter string a:" << std::endl;
#endif
    std::cin >> a;
#ifdef OUTPUT
    std::cout << "Enter string b:" << std::endl;
#endif
    std::cin >> b;
    cycleShift(a, b);
}
int main()
{
    //KMP();
    cycleShift();
    return 0;
}
```