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

Студент гр. 8303 ____ Спирин Н.В.

Преподаватель ____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучить алгоритм Кнута-Морриса-Пратта и алгоритм циклического сдвига, а также написать программу, реализующую эти алгоритмы работы со строками.

Алгоритм Кнута-Морриса-Пратта

Задание

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.

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

Пример входных данных

ab

abab

Пример выходных данных

0, 2

Индивидуализация

Вар. 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 равна массиву pi, где pi[i] обозначает длину максимального префикса строки s[0..i], совпадающего с её суффиксом.

Разделение исходного текста на части

Согласно индивидуализации, необходимо разбить исходный текст на вводимое пользователем количество частей. Для этого вычисляется длина каждой части, равная частному размера текста и введенного числа частей. Для того, чтобы учесть вхождения шаблона на стыках полученных частей, каждая из них продлевается вперед на длину строки шаблона – 1, кроме финальной части. Полученные расширенные части сохраняются в вектор для дальнейшей обработки алгоритмом КМП.

Описание алгоритма

На вход алгоритма передается шаблон и текст. Необходимо найти все вхождения шаблона в тексте.

Сначала вычисляется префикс-функция шаблона и полученные значения заносится в вектор pi.

Затем производится сравнение шаблона и текста. Начиная с начала соответствующих строк при каждом совпадении символов происходит увеличение на единицу счётчиков текущего положения шаблона и текста. Если счетчик шаблона стал равен его размеру (то есть дошли до конца шаблона), то значит вхождение найдено и его позиция равна разности счетчика текста и шаблона. Полученный результат заносится в вектор ответов, который хранит индексы вхождений. Если очередной символ текста не совпал с символом шаблона, то происходит откат к концу предыдущего совпавшего префикса. Алгоритм заканчивает работу, когда будет достигнут конец текста.

Сложность алгоритма по операциям: O (m + n), m – длина текста, n – длина шаблона.

Сложность алгоритма по памяти: O (m), m – длина текста, длина шаблона не учитывается по условию индивидуализации.

Тестирование

1)

```
Enter a pattern:
ab
Enter text:
abab
```

```
Enter the number of parts to search:
1

Text will be divided into 1 parts
PrefixFunction calculation:
Pattern: a b
pi = [0]
s[1] != s[0] ;
Pattern: a b
pi = [0, 0]

KMP for part: a b a b

Part: a b a b
Pattern: a b
PrefixFunction of pattern: [0, 0]

text[0] == pattern[0]

Part: a b a b
Pattern: a b
PrefixFunction of pattern: [0, 0]

text[1] == pattern[1]
Substring found!

Part: a b a b
Pattern: a b
PrefixFunction of pattern: [0, 0]

text[2] != pattern[2]

Part: a b a b
Pattern: a b
PrefixFunction of pattern: [0, 0]

text[2] == pattern[0]

Part: a b a b
Pattern: a b
PrefixFunction of pattern: [0, 0]

text[3] == pattern[1]
Substring found!
All occurrences of the pattern in text:
 0, 2
```

2)

```
Enter a pattern:
queue

Enter text:
ququqeuqueq
Enter the number of parts to search:
2

Text will be divided into 2 parts
Text is divided into 1 big parts with length = 10,
into 0 small parts with length = 9
and into final part with length = 5:
ququqeuque uqueq
PrefixFunction calculation:
Pattern: q u e u e
pi = [0]
s[1] != s[0] ;
Pattern: q u e u e
pi = [0, 0]
s[2] != s[0] ;
Pattern: q u e u e
pi = [0, 0, 0]
s[3] != s[0] ;
Pattern: q u e u e
pi = [0, 0, 0, 0]
s[4] != s[0] ;
Pattern: q u e u e
pi = [0, 0, 0, 0, 0]

KMP for part: q u q u q e u q u e

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[0] == pattern[0]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[1] == pattern[1]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]
```

text[2] != pattern[2]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[2] == pattern[0]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[3] == pattern[1]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[4] != pattern[2]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[4] == pattern[0]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[5] != pattern[1]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[5] != pattern[0]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[6] != pattern[0]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[7] == pattern[0]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[8] == pattern[1]

Part: q u q u q e u q u e
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[9] == pattern[2]

KMP for part: u q u e q

Part: u q u e q
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[0] != pattern[0]

Part: u q u e q
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[1] == pattern[0]

Part: u q u e q
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[2] == pattern[1]

Part: u q u e q
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[3] == pattern[2]

Part: u q u e q
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[4] != pattern[3]

Part: u q u e q
Pattern: q u e u e
PrefixFunction of pattern: [0, 0, 0, 0, 0]

text[4] == pattern[0]

-1

Циклический сдвиг

Задание

Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$).

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 входных данных

defabc

abcdef

Пример выходных данных

3

Описание алгоритма

На вход алгоритму поступают две строки. Сначала алгоритм сравнивает размеры строк, если они не совпадают – строки не могут являться циклическим сдвигом.

Для того, чтобы вычислить, является ли строка A циклическим сдвигом

B, можно удвоить строку A и произвести алгоритм КМП, приняв за текст строку A, в которой ищется вхождение строки B.

В результате КМП вернёт вектор с вхождениями строки B в строку A, из которых для ответа будет выбрано первое.

Сложность алгоритма по операциям: O (n + n), n – длина строки

Сложность алгоритма по памяти: O (n + n), n – длина строки

Тестирование

1)

```
Enter string a:
abcdef
Enter string b:
Efabcd

String a += a: abcdefabcdef
PrefixFunction calculation:
Pattern: e f a b c d
pi = [0]
s[1] != s[0] ;
Pattern: e f a b c d
pi = [0, 0]
s[2] != s[0] ;
Pattern: e f a b c d
pi = [0, 0, 0]
s[3] != s[0] ;
Pattern: e f a b c d
pi = [0, 0, 0, 0]
s[4] != s[0] ;
Pattern: e f a b c d
pi = [0, 0, 0, 0, 0]
s[5] != s[0] ;
Pattern: e f a b c d
pi = [0, 0, 0, 0, 0, 0]

KMP for part: a b c d e f a b c d e f

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[0] != pattern[0]

Part: a b c d e f a b c d e f
```

Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[1] != pattern[0]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[2] != pattern[0]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[3] != pattern[0]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[4] == pattern[0]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[5] == pattern[1]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[6] == pattern[2]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[7] == pattern[3]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[8] == pattern[4]

Part: a b c d e f a b c d e f

Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[9] == pattern[5]
Substring found!

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[10] != pattern[6]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[10] == pattern[0]

Part: a b c d e f a b c d e f
Pattern: e f a b c d
PrefixFunction of pattern: [0, 0, 0, 0, 0, 0]

text[11] == pattern[1]

B string start index in A: 4

2)

Enter string a:

abb

abb

Enter string b:

bab

bab

String a += a: abbabb

PrefixFunction calculation:

Pattern: b a b

pi = [0]

s[1] != s[0] ;

Pattern: b a b

pi = [0, 0]

s[2] == s[0]

Pattern: b a b
pi = [0, 0, 1]

KMP for part: a b b a b b

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[0] != pattern[0]

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[1] == pattern[0]

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[2] != pattern[1]

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[2] == pattern[0]

Part: a b b a b b
Pattern: b a b

PrefixFunction of pattern: [0, 0, 1]

text[3] == pattern[1]

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[4] == pattern[2]
Substring found!

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[5] != pattern[3]

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[5] != pattern[1]

Part: a b b a b b
Pattern: b a b
PrefixFunction of pattern: [0, 0, 1]

text[5] == pattern[0]
B string start index in A: 2

Выводы

В ходе выполнения лабораторной работы был изучены и написаны алгоритм Кнута-Морриса-Пратта и алгоритм циклического сдвига.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

```
#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, 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;
    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];
    }
}


int main() {
    //KMP();
    cycleShift();
    return 0;
}
```