МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 9382 _____ Субботин М. О.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2021

Цель работы.

Познакомиться с одним из часто используемых на практике алгоритма поиска подстроки в строке. Получить навыки решения задач на этот алгоритм.

Задание.

Первое 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

Sample Input:

```
ab
abab
```

Sample Output:

```
0,2
```

Второе 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.

Sample Input:

```
defabc
abcdef
```

Sample Output:

```
3
```

Описание алгоритма.

В наивном алгоритме поиска подстроки в строке рассматриваются подстроки, которые начинаются с каждого символа в строке.

Алгоритм Кнута-Морриса-Пратта позволяет не рассматривать подстроки, начинающиеся с каждого символа в строке. В случае несовпадения текущих символов строки и паттерна-подстроки нужно посмотреть на подстроку до этого символа, если в этой подстроке есть префикс, который равен его постфиксу, то можно продолжать рассматривать текущий элемент в строке, а в паттерне элемент, следующий за префиксом. Таким образом избегается рассматривание всех элементов в строке до этого несовпадающего. Для такого определения совпадающих постфиксов и префиксов заранее строится массив. В элементах этого массива находятся длины максимальных префиксов, на которые можно сдвигаться. Т.е. если не совпали элементы i из строки и j из паттерна, то в элементе j-1 массива будет храниться индекс элемента, с которого можно продолжать рассматривать паттерн дальше, при этом не меняя элемент i. Если же индекс j изначально равен 0, то следует подвинуть на 1 вправо индекс i.

Таким образом, предотвращая рассмотрения всех подстрок этот алгоритм работает намного лучше наивного.

Сложность алгоритма.

Для алгоритма строится массив префикс-функций за $O(m)$, где m – длина подстроки. Сам алгоритм работает за $O(n)$, где n – длина строки, т.к. не происходит возвращения назад в строке и в подстроке для каждого элемента строки лишь идут переходы по подстроке по уже подсчитанным ранее “путям”.

Таким образом общая сложность алгоритма по времени $O(n + m)$.

По памяти сложность будет составлять $O(m)$, где m – длина подстроки, т.к. в алгоритме выделяется память только под массив префикс-функций, все остальные операции выполняются “in-place”.

Описание функций и структур данных.

std::vector<int> KMPCompute(std::string pattern, std::string text) – функция основного алгоритма.
Аргументы:
std::string pattern – паттерн-подстрока
std::string text – основная строка
Возвращаемый объект:
std::vector<int> - вектор индексов, с которых начинаются все вхождения искомых подстрок.

void longestPrefixSuffix(std::vector<int>& lps, std::string pattern) – функция, отвечающая за построение массива префикс-функций.
Аргументы:
std::vector<int>& lps – массив префикс-функций.
std::string pattern – паттерна-подстрока
Возвращаемых объектов нет.

Тестирование.

Задание 1:

№	Входные данные	Выходные данные	Результат
1	aba abababa	0,2,4	Правильно
2	et bgasdfasdgrljja	-1	Правильно
3	ab abababababababababababab	0,2,4,6,8,10,12,14,16,18,20	Правильно
4	cd qwetrccccdgg	8	Правильно

Задание 2:

№	Входные данные	Выходные данные	Результат
1	defabc abcdef	3	Правильно

2	defabct abcdef	-1	Правильно
3	template template	0	Правильно
4	latetemp template	4	Правильно

Выводы.

Был исследован часто используемый на практике алгоритм - поиск подстроки в строке. Также были получены навыки решения задач на этот алгоритм.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

#define TASK_1
//#define TASK_2
#define DEBUG

class KMP{
public:
    std::vector<int> KMPCompute(std::string pattern, std::string text);
private:
    void longestPrefixSuffix(std::vector<int>& lps, std::string pattern);
};

#ifdef DEBUG
void printPatternAndString(std::string pattern, std::string text, int i, int j){
    for(int k = 0; k < i; k++){
        std::cout << " ";
    }
    std::cout << "i" << std::endl;

    std::cout << text << " - основная строка" << std::endl;

    for(int k = 0; k < j; k++){
        std::cout << " ";
    }
    std::cout << "j" << std::endl;

    std::cout << pattern << " - паттерн" << std::endl;
}
#endif

std::vector<int> KMP::KMPCompute(std::string pattern, std::string text) {
    std::vector<int> lps(pattern.length());
    std::vector<int> entries;

    //подсчитываем префикс-функцию от строки-паттерна
```

```
#ifdef DEBUG
    std::cout << "Считаем массив префикс-функции: " << std::endl;
#endif
    longestPrefixSuffix(lps, pattern);

    //индекс i для прохода по основной строке, а j по паттерну
    int i = 0;
    int j = 0;
#ifdef DEBUG
    std::cout << "Начальные индексы: " << std::endl;
    printPatternAndString(pattern, text, i, j);
    std::cout << std::endl;
#endif

    //пока не пройдем всю основную строку
    while ( i < text.length()){

        //если символы совпали, то просто переходим на следующие
        if(pattern[j] == text[i]){
#ifdef DEBUG
            std::cout << "Элементы с индексами i и j совпадают, двигаем i и j на
1 вправо" << std::endl;
#endif
            j++;
            i++;
        }

        /*если дошли до конца паттерна, то подстрока совпала
          записываем индекс начала подстроки
          и возвращаемся в паттерне на элемент следующий за префиксом, которой
равен постфиксу
        */
        if( j == pattern.length()){
#ifdef DEBUG
            std::cout << "Дошли до конца паттерна, строка совпала, добавляем
индекс в результат." << std::endl
                      << "Переходим j к элементу после префикса." << std::endl;
#endif
            entries.push_back(i - j);
            j = lps[j - 1];
        }
```

```
        //если не конец текста и элементы паттерна и текста не равны
        else if( i < text.length() && pattern[j] != text[i]){
#ifdef DEBUG
            std::cout << "Элементы с индексами i и j не равны." << std::endl;
#endif
            //если не первый элемент в паттерне, то возвращаемся на элемент, следующий за префиксом
            if(j!=0){
#ifdef DEBUG
                std::cout << "j не ноль, передвигаем j к элементу следующим за префиксом." << std::endl;
#endif
                j = lps[j - 1];
            }
            //иначе рассматриваем следующий элемент в тексте
            else {
#ifdef DEBUG
                std::cout << "j ноль, передвигаем i на 1 элемент вправо." << std::endl;
#endif
                i++;
            }
        }
#ifdef DEBUG
        if(!entries.empty()) {
            std::cout << "Текущие индексы вхождений: " << std::endl;
            for (auto &item : entries) {
                std::cout << item << " ";
            }
        }
        std::cout << std::endl;
        printPatternAndString(pattern, text, i, j);
        std::cout << std::endl;
#endif
    }
#ifdef DEBUG
    std::cout << "Вся строка пройдена, заканчиваем алгоритм." << std::endl;
#endif
    return entries;
}
```

```
#ifdef DEBUG
void printPrefixSuffix(std::vector<int>lps, std::string pattern, int i, int j){
    for(int i = 0; i < pattern.size(); i++){
        std::cout << i << " ";
    }
    std::cout << " - индексы" << std::endl;

    for(int k = 0; k < j; k++){
        std::cout << "  ";
    }
    std::cout << "j ";
    for(int k = 0; k < i-j-1; k++){
        std::cout << "  ";
    }
    std::cout << "i";
    std::cout << std::endl;

    for(auto& item : pattern){
        std::cout << item << " ";
    }
    std::cout << " - паттерн " << std::endl;
    for(auto& item : lps){
        std::cout << item << " ";
    }
    std::cout << " - массив префикс-функции" << std::endl;
}
#endif

void KMP::longestPrefixSuffix(std::vector<int>& lps, std::string pattern) {
    //первый элемент всегда 0
    lps[0] = 0;

    int j = 0;
    int i = 1;

#ifdef DEBUG
    std::cout << "Начальное состояние: " << std::endl;
    printPrefixSuffix(lps, pattern, i, j);
    std::cout << std::endl;
#endif
```

```
    while( i < pattern.length()){
        //если два текущих элемента в паттерне одинаковы
        //то запоминаем, что при несовпадении на элементе i + 1, следует перейти на элемент с индексом j
        if(pattern[i] == pattern[j]){
#ifdef DEBUG
            std::cout << "Элементы с индексами i и j одинаковы, запоминаем j и продвигаем оба индекса" << std::endl;
#endif
            j++;
            lps[i] = j;
            i++;
        }
        else {
#ifdef DEBUG
            std::cout << "Элементы с индексами i и j различные" << std::endl;
#endif
            if(j!=0){
#ifdef DEBUG
                std::cout << "j не ноль, меняем j на элемент с индексом j-1 в массиве префикс-функции" << std::endl;
#endif
                j = lps[j - 1];
            }
            else {
#ifdef DEBUG
                std::cout << "j -- ноль, записываем в элемент с индексом i ноль и двигаем i вправо" << std::endl;
#endif
                lps[i] = 0;
                i++;
            }
        }

#ifdef DEBUG
        std::cout << std::endl;
        printPrefixSuffix(lps, pattern, i, j);
        std::cout << std::endl;
#endif
    }
```

```
#ifdef DEBUG
    std::cout << "Индекс i достиг конца строки-паттерна, заканчиваем построение
массива префикс-функции. " << std::endl;
#endif
}

#ifdef TASK_1
void Task_1(std::string pattern, std::string text){
    KMP algo;
    std::vector<int> res = algo.KMPCompute(pattern, text);


    if(res.empty()){
        std::cout << "-1";
    }
    else {
        for( int i = 0; i < res.size(); i++){
            std::cout << res[i];
            if(i != res.size()-1)
                std::cout << ",";
        }
    }
}
#endif

#ifdef TASK_2
void Task_2(std::string pattern, std::string text){
    if(pattern.length() != text.length()){
#ifdef DEBUG
        std::cout << "Строки разной длины, значит точно нет циклического сдвига."
<< std::endl;
#endif
        std::cout << "-1";
        return;
    }

    KMP algo;
    std::vector<int> res = algo.KMPCompute(text, pattern + pattern);

    if(res.empty()){
        std::cout << "-1";
```

```
    }
    else {
        std::cout << res[0];
    }
}
#endif

int main() {
    std::string pattern, text;
    std::cin >> pattern >> text;

#ifdef TASK_1
    Task_1(pattern, text);
#endif

#ifdef TASK_2
    Task_2(pattern, text);
#endif

    return 0;
}
```