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. 9382 _____ Дерюгин Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта. Реализовать программу, которая ищет все вхождения шаблона в тексте. Реализовать программу, которая проверяет, является ли одна строка циклическим сдвигом другой строки.

Задание 1

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

Задание 2

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

Сначала находим префикс функции для каждой подстроки шаблона:

Префикс функция для первого элемента равна 0.

Величина префикс функции для подстроки - это максимальная длина собственного префикса подстроки, который совпадает с собственным суффиксом данной подстроки.

После этого посимвольно проходимся по тексту. До тех пор, пока текущий символ из шаблона не равен текущему символу из текста рассматриваем символ в шаблоне под индексом префикс функции предыдущего символа. Если же текущий символ в шаблоне стоит на нулевом месте, значит переходим к следующему символу текста.

Если символы из текста и шаблона равны и символ шаблона не последний, тогда проверяем следующие символы. Если же символ шаблона последний, тогда подстрока найдена в тексте.

Во втором алгоритме удваиваем строку, которая может являться циклическим сдвигом другой строки. Таким образом первая строка будет содержать вторую и все, что нужно сделать, это запустить алгоритм из предыдущего задания и найти совпадение.

Оценка сложности по памяти.

В первой программе сложность по памяти будет O(a + b), где a - длина шаблона, b - длина текста, так как нам нужно хранить лишь массивы этих символов.

Во второй программе сложность по памяти будет O(a*2+b), так как строчка a удваивается.

Оценка сложности по времени.

Вычисление префикс функции происходит за O(a) операция, где a - число символов в шаблоне. Алгоритм КМП также является линейным, так как проходит один раз по тексту длинной n. Общая сложность по времени в первом алгоритме O(a + n)

Так как во втором алгоритме шаблонная строка удваивается, количество операций для вычисления префикс функции для нее возрастет в два раза. Поэтому сложность второго алгоритма: O(2*a + n), где a - длина первой строки, b - длина второй строки.

Тестирование.

Тестирование первой программы.

№	Входные данные	Выходные данные
1	ab abab	Start calculating prefix function Suffix of 'a' on position '0' equals 0 Position of j '0'. That's why pi(1) = 0, where p[1] = b Prefix functions for substring:

		a b 0 0 Symbol 'a' was found on substring and text on positions 0 and 0 respectively Symbol 'b' was found on substring and text on positions 1 and 1 respectively substring was found on text start with position: 0 Symbol 'a' was found on substring and text on positions 0 and 2 respectively Symbol 'b' was found on substring and text on positions 1 and 3 respectively substring was found on text start with position: 2 0,2
2	af asdkfjajieofjskdjfl	-1
3	abababa abababbababababbbababababababab	7,17,19,21,23

4	a aaaaaaaaaaaaaa	0,1,2,3,4,5,6,7,8,9,10,11,12,13

Тестирование второй программы.

№	Входные данные	Выходные данные
1	abba abab	-1
2	af asdkfjajieofjskdjfl	-1
3	abbaa aaabb	3

Выводы.

В результате данной лабораторной работы были разработаны две программы, одна из которых находит все вхождения шаблона в строку, а другая программа определяет, является ли одна строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main2.cpp

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

std::vector<int> prefix(std::string p) {
    std::vector<int> pi(p.size(), 0); // sizes of prefix functions for all
substrings

    std::cout << "Start calculating prefix function\n";
    std::cout << "Suffix of '" << p[0] << "' on position '0' equals 0\n";
    for (int i = 1; i < p.size(); i++) {
        int j = pi[i - 1];
        // found equals symbols
        while (j > 0 && p[i] != p[j]) {
            std::cout << "Symbol on position " <<i<< " not equals symbol on
position " <<j<<". Looking suffix with length " << pi[j - 1]<<std::endl;
            j = pi[j - 1];
        }

        // if symbols equals add prefix function to i
        if (p[i] == p[j]) {
            std::cout << "Found equals symbols. Suffix of '"<<p[i] << "' on
position '"<<i<<"' equals "<<j+1<<std::endl;
            j++;
            pi[i] = j;
        } else {
            std::cout<< "Position of j '0'. That's why pi("<<i<<") = 0, where
p["<<i<<"] = "<< p[i]<<std::endl;
            pi[i] = 0;
        }
        std::cout<<std::endl;
    }

    std::cout<<"Prefix functions for substring:\n";
    for(int i = 0; i < pi.size(); i++){
        std::cout<<p[i]<<" ";
    }
    std::cout<<std::endl;

    for(int i = 0; i < pi.size(); i++){
        std::cout<<pi[i]<<" ";
    }
    std::cout<<std::endl;
    return pi;
}

void circle(std::string a, std::string b) {
    // if a != b
    if (a.size() != b.size()) {
        std::cout << -1;
        return;
    }
    // double string
    std::string doubleString = a + a;
    std::cout<<"String was doubled\n";
```

```
    std::vector<int> pi = prefix(b);
    int index = -1;
    int j = 0;
    for (int i = 0; i < doubleString.size(); i++) {
        while (j > 0 && doubleString[i] != b[j]) {
            std::cout<<"Return to to the index "<<pi[j-1]<<std::endl;
            j = pi[j - 1];
        }
        // if symbols equals
        if (b[j] == doubleString[i]) {
            j++;
            std :: cout << "Symbol '"<<b[j - 1]<<"' was found on substring
and text on positions "<<j - 1<<" and "<<i<< " respectively\n";
        }
        // when found substring
        if (j == b.size()) {
            std::cout<<"substring was found on text start with position: "<<i
- b.size() + 1<<std::endl;
            index = i - b.size() + 1;
            break;
        }
        std::cout<<"Go to the next symbol of text:"<< double-
String[i+1]<<std::endl;
    }

    std::cout << index;
}


int main() {
    std::string p, t;
    std::cin >> p;
    std::cin >> t;
    circle(p, t);
    return 0;
}
```

Файл main.cpp

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

std::vector<int> prefix(std::string p) {
    std::vector<int> pi(p.size(), 0); // sizes of prefix functions for all
substrings

    std::cout << "Start calculating prefix function\n";
    std::cout << "Suffix of '" << p[0] << "' on position '0' equals 0\n";
    for (int i = 1; i < p.size(); i++) {
        int j = pi[i - 1];
        // found equals symbols
        while (j > 0 && p[i] != p[j]) {
            std::cout << "Symbol on position " <<i<< " not equals symbol on
position " <<j<<". Looking suffix with length " << pi[j - 1]<<std::endl;
            j = pi[j - 1];
        }

        // if symbols equals add prefix function to i
        if (p[i] == p[j]) {
```

```
            std::cout << "Found equals symbols. Suffix of '"<<p[i] << "' on
position '"<<i<<"' equals "<<j+1<<std::endl;
            j++;
            pi[i] = j;
        } else {
            std::cout<< "Position of j '0'. That's why pi("<<i<<") = 0, where
p["<<i<<"] = "<< p[i]<<std::endl;
            pi[i] = 0;
        }
        std::cout<<std::endl;
    }

    std::cout<<"Prefix functions for substring:\n";
    for(int i = 0; i < pi.size(); i++){
        std::cout<<p[i]<<" ";
    }
    std::cout<<std::endl;

    for(int i = 0; i < pi.size(); i++){
        std::cout<<pi[i]<<" ";
    }
    std::cout<<std::endl;
    return pi;
}

void kmp(std::vector<int> pi, std::string p, std::string t) {
    int j = 0;
    std::vector<size_t> indexes;
    for (int i = 0; i < t.size(); i++) {
        while (j > 0 && p[j] != t[i]) {
            std::cout<<"Return to to the index "<<pi[j-1]<<std::endl;
            j = pi[j - 1];
        }
        // if symbols equals
        if (p[j] == t[i]) {
            j++;
            std :: cout << "Symbol '"<<p[j - 1]<<"' was found on substring
and text on positions "<<j - 1<<" and "<<i<< " respectively\n";
        }
        // when found substring
        if (j == p.size()) {
            std::cout<<"substring was found on text start with position: "<<i
- p.size() + 1<<std::endl;
            indexes.push_back(i - p.size() + 1);

        }
        std::cout<<"Go to the next symbol of text:"<< t[i+1]<<std::endl;
    }

    // print founded indexes
    if (indexes.empty()) {
        std::cout << -1;
    } else {
        for (int k = 0; k < indexes.size() - 1; k++) {
            std::cout << indexes[k] << ",";
        }
        std::cout << indexes[indexes.size() - 1];
    }
}

int main() {
```

```
    std::string p, t;
    std::cin >> p;
    std::cin >> t;
    kmp(prefix(p), p, t);
    return 0;
}
```