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: Кнут-Моррис-Пратт

Студентка гр. 9382 _____ Голубева В.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для того, чтобы научиться искать вхождения одной строки в другую и или проверять, что одна из них является циклическим сдвигом другой.

Задание 1

Для заданного шаблона $P = efefeftef$ вычислите значения префикс-функции.

Например, для $P = aba$ значения - 0 0 1

Значения функции в ответе разделяйте одним пробелом.

Задание 2

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 3

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

Ответ на задание 1

0 0 1 2 3 4 0 1 2

Описание алгоритма

Для того, чтобы не производить большое количество заведомо бесполезных шагов строим вектор значений префикс — функции.

Первоначально определяются значения префикс-функции для шаблона, который необходимо найти в тексте. Значение префикс-функции означает длину наибольшего совпадения префикса и суффикса в подстроке шаблона, которая рассматривается. Всего значений будет N, где первое значение – 0,

так как размер подстроки равен единице, соответственно максимальный размр префикса и суффикса этой подстроки равен единице.

В первой программе необходимо найти все вхождения шаблона в тексте. Рассматриваются символы текста до тех пор, пока не будет рассмотрен гонечный символ. Также рассматриваются символы строки-шаблона. Если символ текста и символ шаблона равен, то рассматриваются следующие символы. Если этот символ был последним символом строки-шаблона, то было найдено вхождение шаблона в тексте, индекс вхождения записывается в результат. Индекс строки-шаблона в этом случае становится значением префикс-функции под предыдущим значением индекса строки-шаблона.

Если рассматриваемые символы не равны, и рассматриваемый символ строки-шаблона был начальным, то сдвигается индекс символа, который рассматривается в тексте на единицу. Если же символ был не начальным, то индекс символа в строке-шаблоне становится равен значению префикс-функции предыдущего индекса.

Во второй программе мы склеиваем первую строку саму с собой, после этого для второй строки мы можем понять, является ли она циклически получена из первой строки. Для этого нужно просто проверить, входит ли она в исходную строку с помощью алгоритма КМП.

Оценка сложности по памяти

В первой программе храним две входные строки с длинами m и n, а также префикс функцию, длина которой равна длине одной из строк, то есть сложность по памяти равна O(m+n)

Во второй программе храним также две входные строки с длинами m и n, а также префикс функцию и строку, склеенную из одной из них с самой с собой. Таким образом, сложность по памяти также равна O(m+n).

Оценка сложности по времени

Пусть pi[i] — значение префикс-функции от строки S[0, m-1] для индекса j. Тогда после сдвига мы можем возобновить сравнения с места T[i+j] и S[pi[j]] без потери возможного местонахождения образца. Можно показать, что таблица pi может быть вычислена (амортизационно) за O(m) сравнений перед началом поиска. А поскольку строка T будет пройдена ровно один раз, суммарное время работы алгоритма будет равно O(m + n), где — длина текста.

Такая же оценка будет и для второй программы. Хоть длина одной из строк будет больше, чем в первой программе, сложность не изменится - O(m + n).

Тестирование

Результаты тестирования программы можно посмотреть в приложениях В и Г.

Выводы.

Был изучен алгоритм Кнута-Морриса-Пратта. Была реализована программа, которая осуществляет поиск вхождения одной строки в другую и или проверяет, что одна из них является циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LAB4_1

Название файла: lab4_1.cpp

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

void print_vec(std::vector<int >vec){
    for (int i = 0;i < vec.size(); i++){
        if (i == 0){
            std::cout << vec[i];
        }
        else{
            std::cout << ',' << vec[i];
        }
    }
    std::cout<<"\n";
}

std::vector<int> KnutMorrisPratt(std::string text, std::string templ)
{
    std::vector<int> prefix(templ.size() + 1, -1);
    std::vector<int> matches;

    if (templ.size() == 0){
        matches.push_back(0);
        return matches;
```

```
    }

    //compute prefix function
    for (int i = 1; i <= templ.size(); i++){
        int position = prefix[i - 1];
        while (position != -1 && templ[position] != templ[i - 1]){
            position = prefix[position];
        }
        prefix[i] = position + 1;
    }
    std::cout<< "Prefix function of template(without the -1 in
beginning): ";
    print_vec(prefix);
    std::cout << '\n';

    int textpos = 0, templpos = 0;
    while (textpos < text.size()){
        while (templpos != -1 && (templpos == templ.size() ||
templ[templpos] != text[textpos])){
            templpos = prefix[templpos];
        }
        std::cout << "templpos: " << templpos << ", textpos: " <<
textpos<<'\n';
        textpos++;
        templpos++;
        if (templpos == templ.size()){ // if found a match
            std::cout<<"find match at pos: " << textpos - templ.size()
<< '\n';
            matches.push_back(textpos - templ.size());
        }
    }
    return matches;
}

int main(){
    std::string s1;
    std::string s2;
```

```
    std::cin >> s1;
    std::cin >> s2;
    std::vector <int> match = KnutMorrisPratt(s2, s1);
    if (match.size() == 0){
        std::cout<< "Result: " << -1 << '\n';
    }
    else{
        print_vec(match);
    }
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ LAB4_2

Название файла: lab4_2.cpp

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

void print_vec(std::vector<int >vec){
    for (int i = 0;i < vec.size(); i++){
        if (i == 0){
            std::cout << vec[i];
        }
        else{
            std::cout << ", " << vec[i];
        }
    }
    std::cout<<"\n";
}

std::vector <int> prefixFunc(std::string string){
    int len_s = string.length();
    std::vector <int> prefix;
     prefix.resize(len_s);
    prefix[0] = 0;
    int ind = 0;
     for (int i = 1; i < string.length(); i++){
```

```
            while (ind > 0 && string[ind] != string[i]){
                  ind = prefix[ind - 1];
         }
            if (string[ind] == string[i]){
                  ind += 1;
         }
            prefix[i] = ind;
      }
      return prefix;
}

int isCyclicShift(std::string str1, std::string str2){
      int result = -1;
      if (str1.length() != str2.length()){ //unless exactly cyclic
shift
            std::cout << "string lengths not matched! ";
            return result;
     }

      std::string pasting_string = str1 + str1;
      std::vector <int> pref = prefixFunc(str2); // find prefix -
function
      std::cout<< "Prefix function of str2: ";
     print_vec(pref);
     std::cout << '\n';
      int ind = 0;
      for (int i = 0; i < pasting_string.size(); i++){
            //find index in str2 to match characters in str1
            while (ind > 0 && str2[ind] != pasting_string[i]){
                  ind = pref[ind - 1];
         }
            std::cout<<"compare characters in position " << i << " in
str1 and "<< ind <<" in str2"<<'\n';
            if (str2[ind] == pasting_string[i]){// check matches
            ind += 1;
         }
```

```
            if (ind == str2.length()) { // if find a substring in
rasting string
                  std::cout<<"find a match with position " << i -
str2.length() + 1<< ", finish!";
                  result = i - str2.length() + 1;
                  break;
            }
      }
      return result;
}

int main() {
      std::string s1;
    std::string s2;
    std::cin >> s1;
    std::cin >> s2;
      int res = isCyclicShift(s1, s2);
      std::cout << "\nResult: " << res << '\n';
    return 0;
}
```

ПРИЛОЖЕНИЕ В

ТЕСТИРОВАНИЕ ПРОГРАММЫ LAB4_1

Входные данные	Выходные данные
ab ababab	Prefix function of template(without the -1 in beginning): -1,0,0 templpos: 0, textpos: 0 templpos: 1, textpos: 1 find match at pos: 0 templpos: 0, textpos: 2 templpos: 1, textpos: 3 find match at pos: 2 templpos: 0, textpos: 4 templpos: 1, textpos: 5 find match at pos: 4 0,2,4
abcdfeo oabcdfe	Prefix function of template(without the -1 in beginning): -1,0,0,0,0,0,0 templpos: -1, textpos: 0 templpos: 0, textpos: 1 templpos: 1, textpos: 2 templpos: 2, textpos: 3 templpos: 3, textpos: 4 templpos: 4, textpos: 5

	templpos: 5, textpos: 6 Result: -1
qwe qwerty	0
qwert qwe	-1
pnkvsnvsklv sdnvksefljjk	-1

ПРИЛОЖЕНИЕ Г

ТЕСТИРОВАНИЕ ПРОГРАММЫ LAB4_2

Входные данные	Выходные данные
defabc abcdef	Prefix function of str2: 0, 0, 0, 0, 0, 0 compare characters in position 0 in str1 and 0 in str2 compare characters in position 1 in str1 and 0 in str2 compare characters in position 2 in str1 and 0 in str2 compare characters in position 3 in str1 and 0 in str2 compare characters in position 4 in str1 and 1 in str2 compare characters in position 5 in str1 and 2 in str2 compare characters in position 6 in str1 and 3 in str2 compare characters in position 7 in str1 and 4 in str2 compare characters in position 8 in str1 and 5 in str2 find a match with position 3, finish! Result: 3
defabc abc	string lengths not matched! Result: -1
aaaaaaaa bbbbbbbb	Result: -1
abcabcabc bcabcabca	Result: 1
abcdfeo	Result: 6

oabcdfe	