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

Студент гр. 9383 _____ Крейсманн К.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта, реализовать с его помощью 2 программы, первая из которых ищет все вхождения подстроки в строке, а вторая проверяет является ли одна строка циклическим сдвигом другой.

Задание.

1) Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P|<=15000) и текста T (|T|<=5000000) найдите все вхождения P в T.

Вход:

Первая строка – P

Вторая строка – T

Выход:

Индексы начал вхождения P в T, разделенных запятой, если P не входит в T, то вывести -1.

Пример входных данных:

ab

abab

Пример выходных данных:

0,2

2) Заданы две строки A и B.

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).

Вход:

Первая строка – A

Вторая строка – B

Выход:

Если A является 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

Теория

Префикс строки – это какое-то количество первых символов строки.

Суффикс строки – это какое-то количество символов конца строки.

Префикс-функция – это массив чисел, вычисляющийся, как наибольшая длина суффикса, совпадающего с ее префиксом.

Алгоритм Кнута-Морриса-Пратта – эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объема входных данных. Алгоритм был разработан Д. Кнутом и В. Праттом и , независимо от них, Д.Моррисом в 1977 году.

Алгоритм вычисления префикс-функции

1) Переменным-индексам i и j присваиваются значения 1 и 0 соответственно. i «отвечает» за суффикс, j – за префикс.

2) Если i-ый и j-ый элементы строк совпадают, то i-му элементу префикс-функции присваивается значение j+1 . К индексам прибавляется по единице.

3) Если i-ый и j-ый элементы не совпадают, то, если j=0 -> i-му элементу префикс-функции присваивается значение 0, к индексу i прибавляется единица.

4) Если j!=0, то j присваивается значение префикс-функции от j-1. (то есть сравнение продолжается не сначала префикса, а с какого-то его элемента).

5) 2-4 повторяется, пока не прошлись по всей строке.

Вычислительная сложность алгоритма O(n), где n – количество символов в строке. Т.к. алгоритм проходится по всей строке один раз, при этом

с индексом j в худшем случае может совершиться n операций, без продвижения по строке (т.е. максимальное количество операций - 2n).

Алгоритма поиска подстрок в строке

Дана подстрока P и строка T.

1) Строим строку P_T, состоящую из строк P и T, разделенных пробелами. В данном случае префиксом будет являться шаблон P, а суффиксом – текущая часть строки T.

2) Считаем префикс-функцию от данной строки.

3) Проходимся по полученному массиву.

4) Если элемент равен длине шаблона, то это означает, что найдена подстрока P в строке T, а найденный элемент является последним символом в подстроке.

Вычислительная сложность алгоритма O(P_T)=O(P+T). Так как префикс функция считается за O(P_T), проход по полученному массиву также осуществляется за O(P_T).

Алгоритм проверки циклического сдвига

Даны две строки A и B.

1) Проверяем, что длины A и B совпадают

2) Строим строку состоящую из строк B , A и A. T.e. B_AA. Например, для строк “qwerty” и “ertyqw” построим строку “qwerty_erty**qwerty**qw”.

3) Считаем префикс-функцию от полученной строки.

4) Если A является циклическим сдвигом B, то подстрока B должна содержаться в строке AA.

5) Проходимся по полученному массиву, если находим элемент, равный длине B, то это означает, что A является циклическим сдвигом B.

Вычислительная сложность алгоритм O(B+A). Аналогично предыдущему алгоритму.

Описание основных функций и структур данных

Input() – ввод входных данных.

Output() – вывод результата.

Prefix() – вычисление префикс-функции. (алгоритм описан выше)

KMP () в task1 – ищет подстроки по шаблону P в строке T. Возвращает массив, содержащий индексы, с которых шаблон входит в строку.

KMP () в task2 – проверяет, является ли строка A циклическим сдвигом строки B. Если A является циклическим сдвигом B, возвращает первый индекс начала строки B в A, иначе -1.

Тестирование (задание 1)

Входные данные	Выходные данные
bcd abcdabcdabcd	1,5,9
lolo lolololo	0,2,4
abcd abcefdgabdefal	-1
a lafaoapsalsoaiwaqafaxazajamalapaa	1,3,5,8,12,15,17,19,21,23,25,27,29,31,32

Тестирование (задание 2)

Входные данные	Выходные данные
woow owwo	2
helloworld heellowo	-1
abrakadabra arbadakarba	-1
qwweeerrrrtyyuuuiiiiop rrtyyuuuiiiiopqwweeerr	8

Вывод.

В процессе выполнения лабораторной работы был изучен и реализован алгоритм Кнута-Морриса-Пратта. С помощью данного алгоритма построены программы для поиска всех вхождений подстроки в строку и для определения циклического сдвига двух строк.

Приложение А

main.cpp

```
#include "task1.hpp"

int main()
{
   std::string P, T;
   input(P, T, std::cin);
   Result result = KMP(P, T);
   output(result, std::cout);
   return 0;
}
```

task1.hpp

```
#include <iostream>
#include <string>
#include <vector>
using Result = std::vector<int>;

void input(std::string& str1, std::string& str2, std::istream& in);
Result KMP(std::string P, std::string T);
void output(Result result, std::ostream& out);
std::vector<int> prefix(std::string &str);
```

task1.cpp

```
#include "task1.hpp"
void input(std::string& str1, std::string& str2, std::istream& in)
{
   in >> str1 >> str2;
}

std::vector<int> prefix(std::string &str)
{
   std::vector<int> pi(str.length());
   pi[0]=0;
   int i = 1, j = 0;
   while (i < str.length())
   {
      if (str[i] == str[j])
      {
         pi[i] = j + 1;
         i++;
         j++;
      }
      else if (j == 0)
      {
         pi[i] = 0;
         i++;
```

```
        }
        else
        {
            j = pi[j - 1];
        }
    }
/*
     std::cout<<"Префикс-функция:"<<'\n';
    for(auto i:str)
    {
        std::cout<<i<<' ';
    }
    std::cout<<'\n';
    for(auto i:pi)
    {
        std::cout<<i<<' ';
    }
    std::cout<<'\n';
*/
    return pi;
}

Result KMP(std::string P, std::string T)
{
    std::string P_T = P + " " + T;
    Result result;
    std::vector<int> pi = prefix(P_T);

    for (int k = P.length() + 1; k < P_T.size(); k++)
    {
        if (pi[k] == P.length())
        {
            result.push_back(k -2* P.length());
        }
    }
    return result;
}

void output(Result result, std::ostream& out)
{
    if(result.size()==0)
    {
        out<<-1;
        return;
    }
    for (int i = 0; i < result.size()-1; i++)
    {
        out<< result[i] << ',';
    }
    out << result[result.size() - 1];
}
```

main.cpp

```
#include "task2.hpp"

int main()
{
   std::string A, B;
   input(A, B, std::cin);
   int result = KMP(A, B);
   std::cout<<result;
   return 0;
}
```

task2.hpp

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

void input(std::string& str1, std::string& str2, std::istream& in);
int KMP(std::string& A, std::string& B);
std::vector<int> prefix(std::string &str);
```

task2.cpp

```
#include "task2.hpp"

void input(std::string& str1, std::string& str2, std::istream& in)
{
   in >> str1 >> str2;
}

std::vector<int> prefix(std::string &str)
{
   std::vector<int> pi(str.length());
   pi[0]=0;
   int i = 1, j = 0;
   while (i < str.length())
   {
      if (str[i] == str[j])
      {
         pi[i] = j + 1;
         i++;
         j++;
      }
      else if (j == 0)
      {
         pi[i] = 0;
         i++;
      }
      else
      {
```

```
            j = pi[j - 1];
        }
    }

    /*std::cout<<"Префикс-функция:"<<'\n';
    for(auto i:str)
    {
        std::cout<<i<<' ';
    }
    std::cout<<'\n';
    for(auto i:pi)
    {
        std::cout<<i<<' ';
    }
    std::cout<<'\n';*/

    return pi;
}

int KMP(std::string& A, std::string& B)
{
    int result = -1;
    if(A.size()!=B.size())
    {
        return result;
    }
     B += " "+A + A;
    std::vector<int> pi = prefix(B);

    for (int k = A.length() + 1; k < B.length(); k++)
    {
        if (pi[k] == A.length())
        {
            result = k -2 * A.length();
            break;
        }
    }
    return result;
}
```