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. 7304 _____ Давыдов А.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Изучение алгоритма поиска подстроки в строке – алгоритм Кнута-Морриса-Пратта.

Теоретические сведения.

Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

Алгоритм был разработан Д. Кнутом и В. Праттом и, независимо от них, Д. Моррисом. Результаты своей работы они опубликовали совместно в 1977 году.

Алгоритм Кнута-Моррса-Пратта используется для поиска подстроки (образца) в строке. Кажется, что может быть проще: двигаемся по строке и сравниваем последовательно символы с образцом. Не совпало, перемещаем начало сравнения на один шаг и снова сравниваем. И так до тех пор, пока не найдем образец или не достигнем конца строки.

Простой случай поиска

'игла' — образец

'стогистогстогигстогстогиглстогстогигластогигластог' — строка

поиска Сложный случай поиска

aabbaab

aabaabbaaabaabaabaabaabbaabb

Функция работает и довольно шустро, если образцы и строка «хорошие». Хорошие — это когда внутренний цикл прерывается быстро (на 1-3 шаге, скажем, как в простом случае). Но если образец и строка содержат часто повторяющиеся вложенные куски (как сложном случае выше), то внутренний цикл обрывается ближе к концу образца и время поиска оценивается как O(<длина образца>*<длина строки>). Если длина строки 100тыс, а длина образца 100, то получаем O(10млн). Конечно, реально редко встретишь образец

длиной 100, но в олимпиадных задачах «на поиск» это обычное дело, поэтому там простой поиск часто не подходит.

А если строка — это длинный текст, а образец-фрагмент слова, и надо найти все вхождения этого фрагмента, причем на лету, по мере набора слова (замечали, как быстро это делают браузеры)? Алгоритм КМП находит все вхождения образца в строку и его скорость O(<длина образца>+<длина строки>), поэтому на больших текстах/образцах или на слабых процессорах (как в низкобюджетных сотовых) он вне конкуренции.

Задача.

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

   Вход:
   Первая строка - P
   Вторая строка - T
   Выход:
   индексы начал вхождений PP в T, разделенных запятой, если P не входит в T, то вывести -1

Sample Input:
ab
abab

Sample Output:
0,2

2. Заданы две строки AA ($|A|\leq 5000000$) и BB ($|B|\leq 5000000$). 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 является 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.

1. **Алгоритм Кнута-Морриса-Пратта**
    1. Объединяем строку-паттерн и текст в единую строку с символом-разделителем @.
    2. Для каждого символа последовательно вычисляем значение префикс-функции и сохраняем эти значения в массив.
    3. Ищем индексы, по которым значение префикс-функции совпадает с длиной паттерна. Эти значения являются индексами, по которым находится последний символ вхождения паттерна в объединение паттерна и текста.
    4. Зная длину паттерна вычисляем индекс начала вхождения для каждого индекса.
2. **Определение циклического сдвига.**
    1. Если текст A совпадает с текстом B или их длины не совпадают, то решение тривиально. Алгоритм заканчивает работу.
    2. Объединяем текст A и текст B в единую строку с символом разделителем @.
    3. Используем алгоритм Кнута-Морриса-Пратта для нахождения значения префикс-функции последнего символа объединенного текста.
    4. Начиная с полученного индекса следует начинать сравнение строки A со строкой B. Если, начиная с данного индекса строка A совпадает с началом строки B, то строка A является циклическим сдвигом строки B, и полученное значение префикс-функции есть индекс начала строки B в A.

Пример работы.

1. *ab*
   abab

   0,2

   baaabab
   aa
   1,2

2. *defabc*
   abcdef

   3

   aaaa
   aaaa

   0

   aaa
   bbb

   -1

Вывод

В ходе лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Данный алгоритм был реализован на C++. Идея, лежащая в основе этого алгоритма, была использована для определения является одна строка циклическим сдвигом другой.

Исходный код программы

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

using namespace std;

vector<int> prefix_function (string s)
{
    int n =  s.length();
    vector<int> pi(n);

    for (int i = 1; i < n; ++i)
    {
        int j = pi[i-1];
        while ((j > 0) && (s[i] != s[j]))
            j = pi[j-1];

        if (s[i] == s[j])
            ++j;

        pi[i] = j;
     }

     return pi;
}


// checking B is cyclic shift A
int CyclicShift(string A, string B)
{
    int idx = prefix_function(A + "|" + B).back(); //shows how prefix of A
close to suffix B
    if(idx == A.size())
        return 0;
    else if(idx + prefix_function(B + "|" + A).back() == A.size())
        return idx;
    else
        return -1;
}


string KMP(string const &S, string const &pattern)
{
    string concat = pattern + "|" + S;
    vector<int> pi = prefix_function(concat);
    int n = pattern.size();
    string result = "";

    for(int i = n + 1; i < n + S.size() + 1; ++i)
        if(pi[i] == n)
        {
            if(result.size() == 0)
                result += std::to_string(i - (n + 1) - (n - 1)); //(n + 1)
is shift in concat str
            else
                result += "," + std::to_string(i - (n + 1) - (n - 1));
        }

    if(result.size() == 0)
        result += "-1";

    return result;
}


int main()
```

```
{
    string S;
    string pattern;
    getline(cin, pattern);
    getline(cin, S);

    //cout << KMP(S, pattern);
    cout << CyclicShift(pattern, S);

    return 0;
}
```