МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
“ЛЭТИ” ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ВЭМ

ОТЧЕТ

По лабораторной работе №4

По дисциплине “Построение и анализ алгоритмов”

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8382 Гордиенко А.М.

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для нахождения подстроки в строке. Научиться применять алгоритм.

Постановка задачи.

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.

   Пример входных данных.

   ab

   abab

   Пример выходных данных.

   0,2

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 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

Индивидуализация.

Вариант 2. Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

Описание алгоритма 1.

В программа был реализован алгоритм Кнута-Морриса-Пратта. В начале алгоритм считает префикс-функции для всех префиксов строки-образца и сохраняет их. Алгоритм должен вернуть все индексы, с которых строка T содержит образец P.

Затем алгоритм считает значение префикс функции для всех префиксов объединенной строки P&#&T, где # - символ разделения, не содержащийся в обеих строках, чтобы не нарушать работу алгоритма, а & - операция объединения строк. Если на каком-то шаге алгоритма значение префикс функции объединенной строки становится равен длине образца, то значит найдено некоторое вхождение образца в строку.

Сложность алгоритма.

По памяти.

Сложность составляет O(|P|), где |P| - длина шаблона.

По операциям.

Так как алгоритм проходится по строке один раз и на каждом шаге выполняет операцию за O(1), а длина строки составляет |P| + |T|, то сложность составит O(|P| + |T|).

Описание алгоритма 2.

Алгоритм мало чем отличается от первого. Так для проверки на цикличную запись было решено удвоить одну из строк, причем не важно какую, и получить объединенную строку вида A&#&B&B.

Алгоритм также находит префикс-функцию строки.

Сложность алгоритма.

По памяти.

Также память является линейной O(2*|P|), избавляясь от константности получаем O(|P|).

По операциям.

Так как алгоритм всего лишь считает префикс функцию, которая выполняется линейно, то имеем сложность O(|P|).

Описание функций.

Функция выводы значения префикс функции на экран - void printPrefixFunc(vector<int>& p, const string& T), где p - само значение префикс функции, T - строка, в которой считалась префикс функция. Значения p выводятся на экран поочередно.

Функция расчета префикс функции vector<int>prefixFunction(string P) считает префикс функцию для строки P и возвращает вектор, содержащий результат выполнения функции.

Функция проверки на циклическую запись строки int checkCircle(const string& P, const string& T), проверяет, является ли строка P циклическим сдвигом представления строки T.

Функция, реализующая работу алгоритма Кнута-Морриса-Пратта, vector<int> KMP(const string& P, const string& T), находит все вхождения строки P в строку T.

Тестирование.

КМП.

Входные данные.

1) ab abab

2) cc abcdefgh

Выходные данные.

1) 0,2

2) -1

Циклический сдвиг.

Входные данные.

1) qweqwe eqweqw

2) hahahahaha gegegegege

Выходные данные.

1) 2

2) -1

Выводы.

В ходе работы был изучен алгоритм Кнута-Морриса-Пратта и применен для поиска подстроки в строке, а также для проверки, является ли строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include <iostream>

#include <vector>

#include <string>

#define DBG

using std::cin;

using std::cout;

using std::endl;

using std::string;

using std::vector;

vector<int> prefixFunction(string P) // функция считает префикс
функцию от всех префиксов строки и записывет значение для всех
префиксов в массив p

{

    vector<int> p;

    p.resize(P.size() + 1);

    p[0] = 0;

    for (size_t i = 1; i < P.size() + 1; i++)

    {

        int k = p[i - 1];               // получение значение
максимальной прификс-функции, от строки s[0 ... i - 1]

        while (k > 0 && P[i] != P[k])   // перебираем все строки,
которые являются прификсами и суфиксами

        {                               // строки s[0 ... i - 1] и
пытаемся расширить их символом s[i]

            k = p[k - 1];

        }

        if (P[i] == P[k])               // если символ s[i] совпал с
s[k] значит префикс удалось расширить

            k++;

        p[i] = k;                       // запоминаем длину прифекс-
функции для строки s[0 ... i]
```

```
    }
    return p;
}
int checkCicle(const string& P, const string& T){
    vector<int> p;
    p = prefixFunction(T); // для всех префиксов строки T
#ifdef DBG
    cout << "Prefix function of " << T << " : ";
    for (size_t i = 1;i < p.size(); i++)
        cout << p[i] << ' ';
    cout << endl;
#endif
    int k = 0;
    for (size_t i = 0; i < P.size(); i++){ // алгоритм считает
значение префикс функции для всех префиксов строки T + P + P,
        while (k > 0 && T[k] != P[i]) // чтобы не использовать
дополнительной памяти и не сохранять нигде строку P + P алгоритм два
раза проходит по
            k = p[k - 1]; // строке P. Вычисление префикс функции
начинается с начала строки A так как для строки T все уже вычислено
        if (T[k] == P[i]) // если значение прификс функции для
строки P + P стало равно длине строки T значит, алгоритм нашел
значением циклического сдвига
            k++;
        if (k == T.size())
            return (static_cast<int>(i) - static_cast<int>(T.size())
+ 1);
    }
    for (size_t i = P.size(); i < P.size() + P.size(); i++){
        while (k > 0 && T[k] != P[i - P.size()])
            k = p[k - 1];
        if (T[k] == P[i - P.size()])
            k++;
        if (k == T.size())
```

```
            return (static_cast<int>(i) - static_cast<int>(T.size())
+ 1);
    }
    return -1;
}
vector<int> KMP(const string& P, const string& T){ // функция ищет
все вхождение строки P в строку T и возвращает массив индексов этих
вхождений
    vector<int> ans;
    vector<int> p;
    p = prefixFunction(P);              // для всех префиксов строки
P
#ifdef DBG
    cout << "Prefix function of " << P << " : ";
    for (size_t i = 0;i < p.size(); i++)
        cout << p[i] << ' ';
    cout << endl;
#endif
    int k = 0;
    for(size_t i = 0; i < T.size(); i++){  // считаем значение
префикс-функции для всех префиксов строки P + T
        while (k > 0 && P[k] != T[i])
            k = p[k - 1];
        if (P[k] == T[i])
            k++;
#ifdef DBG
        cout << "Prefix function of \""
            << P.substr(0, k)
            << '|' + P.substr(k, P.size()-k) <<
            "\" and \"" + T.substr(0, i-k+1) + '|'
            << T.substr(i-k+1, k)
            << "\" = " << k << endl;
#endif
```

```
        if (k == P.size())                // если длинна префикс-функции совпало с длинной строки P значит в строке T была найдена строка P
            ans.push_back(i - P.size() + 1);
    }
    return ans;
}
int main(){
    string P;
    string T;
    cin >> P >> T;
    std::vector<int> a = KMP(P, T);
    if (a.empty()){
        std::cout << -1;
    }else {
        for(size_t i = 0; i < a.size(); i++) {
            std::cout << a[i];
            if (i + 1 != a.size()) std::cout << ',';
        }
    }
    std::cout << std::endl;
    return 0;
}
```