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 _____ Овчинников Н.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

Цель работы

Изучить алгоритм Кнута-Морриса-Пратта поиска подстроки в строке и реализовать данный алгоритм на языке программирования C++.

Задание

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

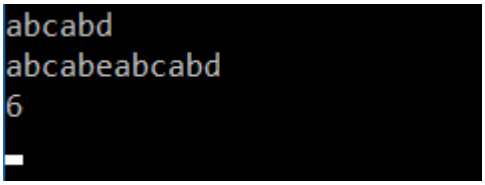
Ход работы

1. Был реализован алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке на языке программирования C++.
   a) Для корректной работы алгоритма КМП сначала была реализована префикс-функция (prefix) для подстроки, которая определяет наибольшую длину префикса, который одновременно является суффиксом для данной подстроки. Функция заполняет массив типа size_t, который хранит значения максимальных длин суффиксов, одновременно являющихся суффиксами подстроки.
   b) Далее была реализована функция, которая реализует поиск подстроки в строке (find_entrances). Функция работает следующим образом: пока не был достигнут конец строки, выполняется сравнения символов строки и подстроки. Если символы равны, индексы увеличиваются, и функция переходит следующих символов. Если символы оказались не равны и индекс подстроки не указывает на его начало, то новый индекс образа вычисляется с использованием массива полученного в ходе работы префикс функции. Если индекс подстроки равен нулю, то индекс строки инкрементируется. В ходе работы функции индексы вхождений сохраняются в вектор (result_entrances) типа size_t.
   c) После завершения работы предыдущей функции выводятся все индексы вхождений подстроки в строку. Если вхождений нет, то выводится -1.
2. Была реализована функция, которая определяет, является ли одна строка циклическим сдвигом второй строки.

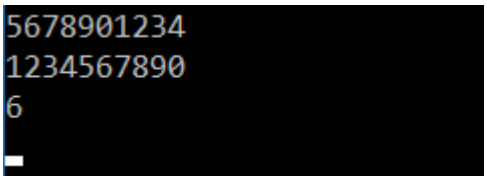
a) Сначала проверяется равенство длин строк. Если длины строк не равны, функция сразу возвращает -1.
b) Выполняется конкатенация первой строки с собой и вызывается функция, реализующая алгоритм КМП для поиска подстроки в строке из п.1 для поиска второй строки в удвоенной первой.
c) Результатом работы функции считается первый элемент вектора (result_entrances). Если вектор не содержит ни одного элемента, то выводится -1.

Пример работы

1. Поиск вхождений:



2. Определение циклического сдвига:



Вывод

В ходе выполнения лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке и успешно реализован на языке программирования C++. Также для корректной работы данного алгоритма была реализована префикс-функция, которая определяет длину наибольшего префикса подстроки, равного суффиксу такой же длины. Временная сложность вычисления префикс-функции оценивается как O(n), где n – длина подстроки. Временная сложность работы всего алгоритма оценивается как O(n+m), где m – длина строки.

Приложение: исходный код программы

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

using namespace std;

class KMP
{
private:
    string P;   //шаблон
    string T;   //текст
    size_t *pi; //массив значений префикс-функции шаблона
    vector<size_t> result_entrances;

    inline void prefix()
    {
        pi[0] = 0;
        for(size_t j=0, i=1; i<P.length();)
        {
            if(P[i]==P[j])
            {
                pi[i] = j+1;
                i++;
                j++;
            }
            else
            {
                if(j==0)
                {
                    pi[i++]=0;
                }
                else
                {
                    j = pi[j-1];
                }
            }
        }
    }

public:
    KMP(string P, string T)
    {
        this->P = P;
        this->T = T;

        pi = new size_t[P.length()];
    }

    ~KMP()
    {
        delete pi;
    }

    void find_entrances()
    {
        prefix();
        int index = -1;
        for(size_t k=0, i=0; k<T.length(); )
        {
            if(T[k] == P[i])
            {
                if(index == -1)
                    index = k;
                k++;
```

```
                i++;
                if(i == P.length())
                {
                    result_entrances.push_back(index);
                    index = -1;
                }
            }
            else
            {
                if(i==0)
                {
                    k++;
                    index = -1;
                }
                else
                {
                    i = pi[i-1];
                    index = k - i;
                }
            }
        }
    }

    void print_entrances()
    {
        if(result_entrances.size() == 0)
        {
            cout << -1 << endl;
            return;
        }
        cout << result_entrances.front();
        for(vector<size_t>::iterator it = result_entrances.begin()+1; it !=
result_entrances.end(); it++)
        {
            cout << "," << *it;
        }
        cout << endl;
    }

    int find_cyclic_shift()
    {
        if(P.length() != T.length())
            return -1;

        result_entrances.clear();
        string save_P = P;
        P = T;
        T = save_P + save_P;
        find_entrances();

        if(result_entrances.size() == 0)
            return -1;

        return result_entrances.at(0);
    }
};


int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    string P, T;
    cin >> P >> T;
    KMP kmp(P, T);
    cout << kmp.find_cyclic_shift() << endl;
//    kmp.find_entrances();
//    kmp.print_entrances();
```

```
    return a.exec();
}
```