МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №4
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Кнута — Морриса — Пратта

Студент гр. 7383 _____ Кирсанов А.Я.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Постановка задачи.

Цель работы.

Задание 1. Реализовать алгоритм КМП и с его помощью для заданных шаблона P (|P|≤15000) и текста T (|T|≤5000000), найти все вхождения P в T.

Задание 2. Заданы две строки A (|A|≤5000000) и B (|B|≤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).

Вариант 1. Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

Реализация задачи.

Был создан класс **Kmp** со следующими полями:

string target – текст, в котором требуется найти шаблон.

string pattern – шаблон.

vector <size_t> prefix – контейнер, хранящий значение префикс-функции для шаблона.

vector <size_t> result – контейнер, индексы вхождения шаблона в строку или индекс начала строки **pattern** в **target**.

В классе **Kmp** реализованы следующие функции:

void prefixfunction () – вычисляет префикс-функцию для шаблона.

void kmpfunction (size_t from, size_t to) – функция, реализующая алгоритм Кнута-Морриса-Пратта. Проходит по строке **target** с индекса **from** до индекса **to** и ищет вхождения шаблона **pattern**, используя значения префикс-функции. Записывает индексы вхождения в контейнер **result**.

void init (size_t ch) – функция считывает текст, шаблон и количество разбиений, в зависимости от значения **ch** либо ищет все вхождения шаблона в текст, либо выводит индекс сдвига – начала строки **pattern** в **target**.

Описание работы программы.

Функция **main()** создает объект класса **Kmp**, считывает номер задания и вызывает функцию **Kmp :: init()**, которая считывает текст и шаблон, а также количество разбиений **k** текста. Рассчитывается префикс-функция для шаблона. Текст разбивается на **k** частей, для каждой из которых вызывается алгоритм КМП, при этом длина каждой части дополняется текстом, равным длине шаблона.

В зависимости от задания будут либо выведены индексы вхождений шаблона в текст, либо индекс начала шаблона в тексте.

Исходный код программы представлен в Приложении Б.

Исследование сложности алгоритма.

Функции **prefixfunction** и **kmpfunction** проходят все символы соответственно шаблона **pattern** и текста **target** один раз. Поэтому сложность алгоритма $O(|\mathbf{pattern}| + |\mathbf{target}|)$.

Тестирование.

Программа тестировалась в среде разработки Qt с помощью компилятора MinGW 5.3.0 в операционной системе Windows 10.

Тестовые случаи представлены в Приложении А.

Выводы.

В ходе выполнения задания был реализован алгоритм Кнута-Морриса-Пратта, выполнена подготовка к распараллеливанию алгоритма. Оценена сложность алгоритма, она составляет $O(|\mathbf{pattern}| + |\mathbf{target}|)$.

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Выходные данные
Task 1 Pattern, target, number of parts ab abab 2	0,2
Task 2 Pattern, target, number of parts defabcdefabc abcdefabcdef 2	3
Task 1 Pattern, target, number of parts defabc abcdef 2	-1

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

using namespace std;

class Kmp{
public:
    Kmp() = default;
    ~Kmp(){
        target.erase();
        pattern.erase();
        prefix.clear();
        result.clear();
    }

    void init(size_t ch){
        cout << "Pattern, target, number of parts" << endl;
        size_t k, partleng;
        cin >> pattern >> target >> k;
        prefix.resize(pattern.length());
        prefixfunction();
        switch (ch) {
            case 1:{
                partleng = target.length()/k;
                for (size_t i = 0; i < k - 1; i++) {
                    kmpfunction(i * partleng, (i + 1) * partleng +
pattern.length() - 2);
                }
                kmpfunction((k - 1)*partleng, target.length());
                if(result.empty()) cout << -1;
                else{
```

```
                    auto it = result.begin();
                    for (; it < result.end() - 1; it++) {
                        cout << *it << ",";
                    }
                    cout << *it;
                }
                break;
            }
            case 2:{
                if(target.length() != pattern.length()){
                    cout << -1;
                    break;
                }
                string tmp = target;
                target = pattern;
                pattern = tmp;
                target += target;
                partleng = target.length()/k;
                for (size_t i = 0; i < k - 1; i++) {
                    kmpfunction(i * partleng, (i + 1) * partleng +
pattern.length() - 2);
                }
                kmpfunction((k - 1)*partleng, target.length());
                if(!result.empty()) cout << result[0];
                else cout << -1;
                break;
            }
            default:{break;}
        }
    }

private:
    string target;
    string pattern;
```

```
    vector<size_t> prefix;
    vector<size_t> result;

    void kmpfunction(size_t from, size_t to)
    {
        for(size_t j = 0, i = from; i < to; ++i)
        {
            while ((j > 0) && (pattern[j] != target[i]))
                j = prefix[j - 1];

            if (pattern[j] == target[i])
                j++;

            if (j == pattern.length())
                result.push_back(i - pattern.length() + 1);
        }
    }

    void prefixfunction()
    {
        prefix[0] = 0;
        for (size_t j = 0, i = 1; i < pattern.length(); ++i)
        {
            while ((j > 0) && (pattern[i] != pattern[j]))
                j = prefix[j - 1];

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

            prefix[i] = j;
        }
    }
};
```

```
int main()
{
    size_t ch;
    cout << "Task" << endl;
    cin >> ch;
    Kmp k;
    k.init(ch);
    return 0;
}
```