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 Кнутта-Мориса-Пратта

Студент гр. 7383 _____ Зуев Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Цель работы: ознакомиться с алгоритмом Кнута-Морриса-Пратта на примере построения алгоритма для выполнения задачи.

Формулировка задачи: Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P|≤15000) и текста T (|T|≤5000000) найдите все вхождения P в T.

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.

Для реализации алгоритма были написаны следующие функции:

preffix_for_P — вычисляет префикс функцию для первой строки, вхождения которой будут искаться во второй строке. Возвращает вектор значений префикс функции.

KMP — алгоритм Кнута-Морриса-Пратта, проходит по второй строке, совершая шаг с учетом вычисленной ранее префикс функции для первой строки. При записывает в строку индексы вхождений первой строки во вторую, возвращает полученную строку.

init_func — делит вторую строку на подстроки с учетом того, что никакие подстроки длинной, равной длине первой строки, не будут пропущены. Затем выполняет алгоритм Кнута-Морриса-Пратта для каждой из полученных подстрок. Возвращает строку — конкатенацию результатов работы алгоритма для этих подстрок.

main — длины дуг до смежных вершин, увеличивающиеся алгоритмом.

В главной функции main считываются строки и количество потоков, для подготовки к распараллеливанию на них. Выполняются задания и выводится результат их работы.

Код программы представлен в приложении Б.

Исследование сложности алгоритма.

Функция preffix_for_P вычисляет значения префикс функции для первой строки. Она проходит по всем символам этой строки, поэтому время выполнения O(|P|).

Функция KMP ища вхождения первой строки проходит по всем символам второй строки, поэтому время выполнения O(|T|).

В сумме получается сложность O(|P|+|T|).

Тестирование.

Программа была собрана в компиляторе G++ в среде разработки Qt в операционной системе Linux Ubuntu 17.10.

В ходе тестирования ошибок выявлено не было.

Корректные тестовые случаи представлены в приложении А.

Выводы.

В ходе выполнения данной работы был изучен и реализован алгоритм Кнута-Морриса-Пратта поиска всех вхождений одной строки в другой. Оценена сложность реализованного алгоритма, она составляет O(|P|+|T|).

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Выходные данные
ab abab 4	Task 1: 0,2 Task 2: -1
defabc abcdef 5	Task 1: -1 Task 2: 3
abcde abcde 1	Task 1: 0 Task 2: 0

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД

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

using namespace std;

string KMP(string T, string P, size_t begin, size_t end, vector<int> p)
{
   string result;
   int k = 0;
   for (int i = begin; i <= end; ++i)
   {
      while ((k > 0) && (P[k] != T[i]))
         k = p[k - 1];
      if (P[k] == T[i])
         ++k;
      if(k == P.length())
         result += to_string(i - P.length() + 1) + ',';
   }
   return result;
}

vector<int> preffix_for_P(string P)
{
   int len = P.length();
   vector<int> p(len);
   p[0] = 0;
   int k = 0;
   for (int i = 1; i < len; ++i)
   {
      while ((k > 0) && (P[k] != P[i]))
         k = p[k - 1];
      if (P[k] == P[i])
         ++k;
      p[i] = k;
   }
   return p;
}

string init_func(string T, string P, size_t k)
{
   string result;
   size_t s_length = T.length() / k;
   vector<int> p = preffix_for_P(P);
   for(int i = 0; i < k-1;i++)
      result+=KMP(T, P, i*s_length,(i+1)*s_length+P.length() - 2, p);
   result+=KMP(T,P, (k-1)*s_length, T.length()-1, p);
   return result;
```

```
}

int main()
{
    string P;
    string T;
    size_t k;
    cin>>P>>T>>k;
    string task1_answ = init_func(T, P, k);
    cout<< "Task 1:" << endl;
    if(task1_answ.empty())
        cout<<-1<<endl;
    else
    {
        task1_answ.pop_back();
        cout << task1_answ << endl;
    }
    cout<<"Task 2:" << endl;
    if(T.length() != P.length())
        cout<<-1<<endl;
    else
    {
        string task2_answ = init_func(T+T, P, k);
        if(task2_answ.empty())
            cout<<-1<<endl;
        else
            cout << task2_answ[0] << endl;
    }
    return 0;
}
```