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

Студентка гр. 7383 _____ Иолшина В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Содержание

Цель работы

Исследовать и реализовывать задачу поиска вхождения подстроки в строке, используя алгоритм Кнута – Морриса – Пратта.

Формулировка задачи: необходимо разработать программу, которая реализует 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. Если P не входит в T, то вывести −1.

Также следует разработать программу для решения следующей задачи: 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. Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести −1.

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.

Входные данные: в первой строке указывается строка шаблона P, а во второй текст, в котором ищем подстроки.

Выходные данные: индексы вхождения подстроки в строку.

Реализация задачи

В данной работе используются главная функция main() и дополнительные функции void p_func(string &pattern, vector <int> &prefix), void KMP(string &pattern, vector <int> &prefix) и int shift(string &A, string &B, vector <int> &prefix).

Параметр, передаваемый функциям pattern, является строкой шаблона. В функции main() считывается строка pattern, а затем вызывается функция p_func, в которой заполняется вектор prefix для строки шаблона значениями

префикс-функции (максимальная длина совпадающих префикса и суффикса в подстроке). Далее вызывается алгоритм KMP, который посимвольно считывает строку и сравнивает её с шаблоном, используя полученные значения префикс-функции.

Для задания с циклическим сдвигом производится деление с остатком, позволяющее переходить на начало шаблона, префикс-функция для данной задачи остаётся такой же, как и для первой.

Тестирование

Программа собрана в операционной системе Ubuntu 16.04.2 LTS", с использованием компилятора g++ (Ubuntu 5.4.0-6ubuntu1~16.04.5). В других ОС и компиляторах тестирование не проводилось.

Тестовые случаи представлены в Приложении А.

Исследование

В начале работы программа вычисляет значения префикс функции для каждого символа первой строки. Если m – длина первой строки, а n – длина второй, то сложность алгоритма по времени будет составлять O(m + n). По памяти сложность алгоритма составляет O(m).

Выводы

В ходе написания лабораторной работы был изучен алгоритм поиска подстроки в строке, используя алгоритм Кнута – Морриса – Пратта. Был написан код на языке программирования C++, который применял этот метод для поставленной задачи. Полученный алгоритм имеет линейную сложность как по времени, так и по памяти.

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

Ввод	Вывод	Верно?
ab abab	0,2	Да
Aabaa aabaabaabaabaaaaaaaaa	0,3,6,9	Да
Gh weiojt	-1	Да

ПРИЛОЖЕНИЕ Б

Исходный код программы

Lr4_1.cpp

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

void p_func(string &pattern, vector <int> &prefix)
{
  prefix.assign(pattern.size(), 0);
  prefix[0] = 0;
  int i = 1, j = 0;
  while (i < pattern.size())
  {
    if(pattern[i] == pattern[j])
    {
      prefix[i] = j + 1;
      i++;
      j++;
    }
    if(pattern[i] != pattern[j])
      if(j == 0)
      {
        prefix[i] = 0;
        i++;
      }
      else j = prefix[j - 1];
  }
}

void KMP(string &pattern, vector <int> &prefix)
{
  char c;
```

```
  cin >> c;
  int i = 0;  //переход по тексту
  int j = 0; //переход по шаблону
  bool flag = false;
  p_func(pattern, prefix);
  while(true)
  {
    if(c == pattern[j])
    {
      i++; j++;
      if(j == pattern.size())
      {
        if(flag) cout << "," << i - pattern.size();
        else
        {
          flag = true;
          cout << i - pattern.size();
        }
        j = prefix[j - 1];
      }
      if(!(cin >> c)) break;
    }
    else if(j == 0)
    {
      i++;
      if(!(cin >> c)) break;
    }
    else j = prefix[j - 1];
  }
  if(!flag) cout << -1;
  return;
}

int main()
{
  string pattern;
  cin >> pattern;
```

```
  vector <int> prefix (pattern.size());
  KMP(pattern, prefix);
  cout << endl;
  return 0;
}
```

Lr4_2.cpp

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

void p_func(string &pattern, vector <int> &prefix)
{
  prefix.assign(pattern.size(), 0);
  prefix[0] = 0;
  int i = 1, j = 0;
  while (i < pattern.size())
  {
    if(pattern[i] == pattern[j])
    {
      prefix[i] = j + 1;
      i++;
      j++;
    }
    if(pattern[i] != pattern[j])
      if(j == 0)
      {
        prefix[i] = 0;
        i++;
      }
      else j = prefix[j - 1];
  }
}

int shift(string &A, string &B, vector <int> &prefix)
```

```
{
  p_func(A, prefix);
  int k = 0, l = 0;
  int flag = 0; //совпадения
  for (int i = 0; i < A.size(); i++)
  {
    while (A[k] == B[l])
    {
      flag++;
      k = (k + 1) % A.size();
      l = (l + 1) % A.size();
      if (flag == A.size())
        return (l % A.size());
      i++;
    }
    flag = 0;
    if (A[k] != B[l])
      if (k != 0)
      {
        k =  prefix[k - 1];
        i--;
      }
      else l++;
  }
  return -1;
}

int main()
{
  string A, B;
  vector <int> prefix (B.size());
  getline(cin, A);
  getline(cin, B);
  if (A.size() - B.size())
  {
    cout << -1;
    return 0;
```

```
  }
  cout << shift(A, B, prefix) << endl;
  return 0;
}
```