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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студентка гр. 7383 _____ Чемова К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Исследовать и реализовать задачу поиска вхождения подстроки в строке, используя алгоритм Кнута-Морриса-Пратта.

Формулировка задачи.

Необходимо разработать программу, которая реализует алгоритм Кнута-Морриса-Пратта и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найти 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. Если возможно несколько сдвигов вывести первый индекс. Входные данные:

Первая строка - P.

Вторая строка - T.

Выполнение работы.

В данной работе используются функции void kmp, void prefix. В функции main() считываются строки P и T. Проходит проверка на возможность вхождения P в T, путем сравнения длин строк. Если это возможно, тогда вызывается функция void kmp. Вызывается функция void prefix, которая ищет значения префикс-функции для строки P. После того, как был заполнен массив pi, содержащий значения префикс-функции для строки P, начинается поиск первого индекса вхождения P в T.

Префикс-функция вычисляет значения для первой строки: функция находит наибольшую длину наибольшего собственного суффикса подстроки, совпадающего с ее префиксом. Значение для первого символа всегда полагается равным 0. Когда вычислены все значения префикс-функции для первой строки, то начинается поиск первой строки во второй: начинаем

сравнение символов с начала в обеих строках, если символы равны – продолжаем сравнение пока не дойдем до первого символа, не совпадающего с символом в первой строке, или пока не дойдем до конца первой строки. Если дошли до конца первой строки, значит первый индекс вхождения первой строки во вторую найден, вычисляем первый индекс, запоминаем и продолжаем операцию сравнения для следующего символа во второй строке и для последнего символа, лежащего по адресу, хранящемуся для данного символа в массиве pi, в первой строке. Аналогично, если символы не совпадают.

Для выполнения задания с циклическим сдвигом производилось склеивание двух строк с разделителем $: P + $ + T. Далее использовалась префикс-функция для всей получившейся строки. После использовалась функция void kmp, которая осуществляла поиск циклического сдвига, используя данные префикс-функции.

Тестирование.

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении А.

Исследование алгоритма.

В начале работы программа вычисляет значения префикс функции для каждого символа первой строки. Длина первой строки P, второй – T. После начинается сравнение каждого символа второй строки с символами первой строки. Тогда сложность алгоритма по времени будет составлять O(P + T).

По памяти сложность алгоритма составляет O(P), т.к. вычисляются значения префикс-функции только для первой строки.

Выводы.

В ходе лабораторной работы был изучен алгоритм Кнута-Морриса-Пратта. Была написана программа на языке C++, реализующая данный алгоритм; исследована сложность алгоритма, по результатам сложность алгоритма линейна по памяти и времени.

ПРИЛОЖЕНИЕ А
ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Результат
aabaabaabaabaaaaaa aabaa	0
bsjkdhfvnd df	-1
abcdef htyuifabcef	-1
defabc abcdef	3

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПОИСКА ПОДСТРОКИ

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

void prefix(std::string pat, int* pi) {
  pi[0] = 0;
  int l = 0;
  for (int i = 1; i < pat.length();){
    if (pat[i] == pat[l]){
      pi[i] = l+1;
      i++;
      l++;
    }
    else{
      if (l != 0)
        l = pi[l - 1];
      else{
        pi[i] = l;
        i++;
      }
    }
  }
}

void kmp(std::string pat, std::string text, int* pi){
  prefix(pat, pi);
  vector <int> answer;
  int i = 0, j = 0;
  while (i < text.length()){
    if (text[i] == pat[j]){
      i++;
      j++;
    }
    if (j == pat.length()){
      answer.push_back(i - j);
      j = pi[j - 1];
    }
    else if (text[i] != pat[j]){
      if (j != 0)
        j = pi[j - 1];
        else
          i++;
    }
  }
  if (answer.empty())
    std::cout << "-1";
  else
    for (int m = 0; m < answer.size(); m++){
      std::cout << answer[m];
      if (m != answer.size() - 1)
        std::cout << ',';
    }
  std::cout << endl;
}
```

```
int main(){
  std::string pat, text;
  std::cin >> pat >> text;
  int pi[pat.length()];
  if (pat.length() > text.length())
    std::cout << "-1" << endl;
  else
    kmp(pat, txt, pi);
  return 0;
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ЦИКЛИЧЕСКОГОПОИСКА

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

void prefix(std::string pat, int* pi){
  pi[0] = 0;
  int l = 0;
  for (int i = 1; i < pat.length();){
    if (pat[i] == pat[l]){
      pi[i] = l+1;
      i++;
      l++;
    }
    else{
      if (l != 0)
        l = pi[l - 1];
      else{
        pi[i] = l;
        i++;
      }
    }
  }
}

void kmp(std::string pat, std::string text, int* pi, int pat_len){
  prefix(pat, pi);
  int i = 0, j = pi[pat.length() - 1];
  while (j <= pat_len){
    if (text[i] == pat[j]){
      i++;
      j++;
    }
    if (j == pat_len){
      int dif;
      dif = pat_len - i;
      std::cout << text.length() - dif << std::endl;
      return;
    }
    else if (text[i] != pat[j]){
      break;
    }
  }
  std::cout << "-1" << std::endl;
}

int main(){
  std::string pat, text;
  std::cin >> text >> pat;
  int pat_len = pat.length();
  pat = pat + '$' + text;
  int* pi = new int[pat.length()];
  if(pat_len > text.length())
    std::cout << "-1" << std::endl;
  else
    kmp(pat, text, pi, pat_len);
  delete [] pi;
```

```
  return 0;
}
```