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. 8303 _____ Кибардин А.Б.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма Кнута-Морриса-Пратта для поиска всех вхождений подстроки в исходной строке.

Формулировка задачи 1.

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

Вход:
Первая строка - P
Вторая строка - T
Выход:
индексы начал вхождений P в T, разделенных запятой, если P*P* не входит в T*T*, то вывести -1−1

Sample Input:

ab
abab

Sample Output:

0,2

Формулировка задачи 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.
Вход:
Первая строка - A
Вторая строка - B
Выход:
Если A является циклическим сдвигом B, индекс начала строки B в *A*, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

Sample Input:

defabc
abcdef

Sample Output:

3

Индивидуализация.

Вар. 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.

Описание алгоритма КМП.

Первым шагом алгоритм вычисляет префикс-функцию для шаблона. На следующем шаге производится сравнение шаблона и текста. При совпадении символов увеличиваем счетчики текста (k) и шаблона (l) на единицу. При несовпадении происходит откат к предыдущему совпавшему префиксу. Если счетчик шаблона стал равен длине строки шаблона, то слово найдено, его позиция k-l записывается в вектор результата, счетчик шаблона обнуляется, а текста инкрементируется, цикл повторяется. Алгоритм заканчивает работу, когда он дойдет до конца строки. Если совпадения в тексте отсутствуют, то в массив индексов записывается -1.

Сложности алгоритма.

Сложность алгоритма по операциям O(n+m), где n – количество символов в тексте, m – количество символов в шаблоне.

Сложность алгоритма O(m), где m – число символов в строке шаблоне.

Описание циклического сдвига.

Алгоритм является модификацией КМП. Аналогично КМП при несовпадении происходит изменение счетчиков. При совпадении проверяется, является ли данное совпадение первым в цепочке. Если оно первое, то позиция в промежуточный результат, инкрементируются счетчики строки сдвига (k) и исходной строки (l). Если счетчик строки сдвига равен длине строки сдвига производится проверка количество проходов по строке. Если алгоритм проходится в первый раз по строке, то счетчик сдвига обнуляется и флаг цикла становится равным true. При повторном достижении конца строки алгоритм завершает свою работу, возвращая -1.

Если на очередной итерации счетчик исходной строки равен длине исходной сроки, то алгоритм завершает свою работу, возвращая индекс начала исходной строки в строке сдвига.

Сложности алгоритма.

Сложность алгоритма по операциям O(n+n) = O(n), где n – количество символов строке сдвига.

Сложность алгоритма O(n), где n – число символов в строке сдвига.

Префикс-функция.

Префикс-функция от строки и позиции в ней – длина наибольшего собственного префикса подстроки, который одновременно является суффиксом этой подстроки. То есть, в начале подстроки длины нужно найти такой префикс максимальной длины, который был бы суффиксом данной подстроки.

В программе префикс-функция для очередного символа рассчитывается на основе предыдущего. Так, если символ с индексом равным префикс-функции и текущий символ строки равны, то префикс-функция текущего символа равна предыдущее значение префикс-функции плюс единица. В случае, когда символы не равны, рассматривается префикс предыдущего символа.

Описание функций.

std::vector<int> prefix_function (std::string s) – функция построения префикс функции. Возвращает вектор значений префикс-функции для обрабатываемой строки.

s – обрабатываемая строка.

int cyclicShift(std::string& text, std::string& shiftText) – функция проверяющая, является ли строка shifhtText циклическим сдвигом строки text. Возвращает индекс начала строки text в строки shiftText. Если строка shiftText не является циклическим сдвигом, возвращается -1.

std::vector<int> kmp(std::string& pattern, std::string& text) – функця поиска вхождений подстроки в строку с помощью алгоритма КМП. Возвращается вектор индексов, с которых найдено совпадение. Если совпадений нет, то возвращается вектор с единственным элементом -1.

pattern – строка-шаблон.

text – строка исходного текста, в которой ищется шаблон.

Тестирование.

Алгоритм КМП

```
ab
abab
0,2
Process finished with exit code 0
```

```
ba
abababaaababababbabaabbbaaa
1,3,5,9,11,14,16,21
Process finished with exit code 0
```

```
ddddd
aaaaa
-1
Process finished with exit code 0
```

```
abcabd
abcabeabcabcabd
prefix function: 0 0 0 1 2 0
found equal symbols, pattern[0] = a, text[0] = a
found equal symbols, pattern[1] = b, text[1] = b
found equal symbols, pattern[2] = c, text[2] = c
found equal symbols, pattern[3] = a, text[3] = a
found equal symbols, pattern[4] = b, text[4] = b
found different symbol, pattern[5] = d, text[5] = e
shift l, k = 5, l = 2
found different symbol, pattern[5] = c, text[2] = e
shift l, k = 5, l = 0
found different symbol, pattern[5] = a, text[0] = e
k++, k = 6, l = 0
found equal symbols, pattern[6] = a, text[0] = a
found equal symbols, pattern[7] = b, text[1] = b
found equal symbols, pattern[8] = c, text[2] = c
found equal symbols, pattern[9] = a, text[3] = a
found equal symbols, pattern[10] = b, text[4] = b
found different symbol, pattern[11] = d, text[5] = c
shift l, k = 11, l = 2
found equal symbols, pattern[11] = c, text[2] = c
found equal symbols, pattern[12] = a, text[3] = a
found equal symbols, pattern[13] = b, text[4] = b
found equal symbols, pattern[14] = d, text[5] = d
found pattern at k = 9
9
Process finished with exit code 0
```

```
gts
abcdefghgtsabcdefghgts
prefix function: 0 0 0
found different symbol, pattern[0] = g, text[0] = a
k++, k = 1, l = 0
found different symbol, pattern[1] = g, text[0] = b
k++, k = 2, l = 0
found different symbol, pattern[2] = g, text[0] = c
k++, k = 3, l = 0
found different symbol, pattern[3] = g, text[0] = d
k++, k = 4, l = 0
found different symbol, pattern[4] = g, text[0] = e
k++, k = 5, l = 0
found different symbol, pattern[5] = g, text[0] = f
k++, k = 6, l = 0
found equal symbols, pattern[6] = g, text[0] = g
found different symbol, pattern[7] = t, text[1] = h
shift l, k = 7, l = 0
found different symbol, pattern[7] = g, text[0] = h
k++, k = 8, l = 0
found equal symbols, pattern[8] = g, text[0] = g
found equal symbols, pattern[9] = t, text[1] = t
found equal symbols, pattern[10] = s, text[2] = s
found pattern at k = 8
found different symbol, pattern[11] = g, text[0] = a
k++, k = 12, l = 0
found different symbol, pattern[12] = g, text[0] = b
k++, k = 13, l = 0
found different symbol, pattern[13] = g, text[0] = c
k++, k = 14, l = 0
found different symbol, pattern[14] = g, text[0] = d
k++, k = 15, l = 0
found different symbol, pattern[15] = g, text[0] = e
k++, k = 16, l = 0
found different symbol, pattern[16] = g, text[0] = f
k++, k = 17, l = 0
found equal symbols, pattern[17] = g, text[0] = g
found different symbol, pattern[18] = t, text[1] = h
shift l, k = 18, l = 0
found different symbol, pattern[18] = g, text[0] = h
k++, k = 19, l = 0
found equal symbols, pattern[19] = g, text[0] = g
found equal symbols, pattern[20] = t, text[1] = t
found equal symbols, pattern[21] = s, text[2] = s
found pattern at k = 19
8,19
Process finished with exit code 0
```

Циклический сдвиг

```
defabc
abcdef
3
Process finished with exit code 0

abcabcabcabcabc
bcabcabcabcabca
1
Process finished with exit code 0

sbtnpwrlflb
bsbtnprwlfl
-1
Process finished with exit code 0

defabcabc
abcdefabc
prefix function: 0 0 0 0 0 0 1 2 3
found different symbol, shiftText[0] = d, text[0] = a
k++, k = 1, l = 0
found different symbol, shiftText[1] = e, text[0] = a
k++, k = 2, l = 0
found different symbol, shiftText[2] = f, text[0] = a
k++, k = 3, l = 0
found equal symbols, shiftText[3] = a, text[0] = a
found equal symbols, shiftText[4] = b, text[1] = b
found equal symbols, shiftText[5] = c, text[2] = c
found different symbol, shiftText[6] = a, text[3] = d
shift l, k = 6, l = 0
found equal symbols, shiftText[6] = a, text[0] = a
found equal symbols, shiftText[7] = b, text[1] = b
found equal symbols, shiftText[8] = c, text[2] = c
end shiftText reached , k = 0, l = 3
found equal symbols, shiftText[0] = d, text[3] = d
found equal symbols, shiftText[1] = e, text[4] = e
found equal symbols, shiftText[2] = f, text[5] = f
found equal symbols, shiftText[3] = a, text[6] = a
found equal symbols, shiftText[4] = b, text[7] = b
found equal symbols, shiftText[5] = c, text[8] = c
end text reached at k = 5
result with cycle: 6
6
Process finished with exit code 0
```

Вывод.

В ходе выполнения лабораторной работы были получены знания по работе с алгоритмом Кнута-Морриса-Пратта для поиска всех вхождений подстроки в заданную строку. Так же были написаны функции, реализующие алгоритм Кнута-Морриса-Пратта, вычисление префикс-функции и проверки является ли одна строка циклическим сдвигом другой.

Приложение А. Исходный код.

Kmp.cpp

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

#define debug

std::vector<int> prefix_function (std::string s) {
   int n =  s.length();
   std::vector<int> pi(n, 0);                    //-создаем вектор необходимого размера
   for (int i=1; i<n; ++i) {
      int j = pi[i-1];                           //-вычисляем новое значение п-ф, про
      while (j > 0 && s[i] != s[j])              //-проверяем до тех пор, пока не дойдем до 0, либо пока
         j = pi[j-1];                            //символы не совпадут
      if (s[i] == s[j])                          //-при совпадении символов увеличиваем п-ф на единицу
         ++j;
      pi[i] = j;
   }
#ifdef debug
   std::cout << "prefix function: ";
   for(auto iter : pi)
      std::cout << iter << " ";
   std::cout << std::endl;
#endif
   return pi;
}

std::vector<int> kmp(std::string& pattern, std::string& text)          //алгоритм кнута-морриса-пратта
{
   std::vector<int> result;
   std::vector<int> prefix;
   if(pattern.size() > text.size())                    //проверка на корректность длины шаблона
   {
#ifdef debug
      std::cout << "length of text less than pattern" << std::endl;
#endif
      result.push_back(-1);
      return result;
   }
   prefix = prefix_function(pattern);                  //-задаем префикс-функцию
   for(int k = 0, l = 0; k < text.size();)
      if(text[k] == pattern[l])                        //-при нахождении одинаковых символов сдвигаем счетчики
      {
#ifdef debug
         std::cout << "found equal symbols, pattern[" << k << "] = " << pattern[l] << ", " << "text[" << l << "] = " <<
text[k] << std::endl;
#endif
         k++;
         l++;
         if(l == pattern.size())                       //-обнуляем счетчик шаблона, если он найден в тексте
         {
#ifdef debug
            std::cout << "found pattern at k = " <<  k - l << std::endl;
#endif
            result.push_back(k - l);
            l = 0;
         }
      } else
      {
#ifdef debug
```

```
        std::cout << "found different symbol, pattern[" << k << "] = " << pattern[l] << ", " << "text[" << l << "] = " <<
text[k] << std::endl;
#endif
        if(l == 0)                          //-при несовпадении шаблона и текста с первого символа
        {                                   //увеличиваем счетчик текста, иначе счетчику шаблона
          k++;                              //присваиваем предыдущее значение префикс-функции
#ifdef debug
          std::cout << "k++, k = " << k << ", l = " << l << std::endl;
#endif
        } else
        {
          l = prefix[l-1];
#ifdef debug
          std::cout << "shift l, k = " << k << ", l = " << l << std::endl;
#endif
        }
      }
  if(result.empty())
  {
    result.push_back(-1);
  }
  return result;
}

int main() {
  std::vector<int> result;
  std::string P, T;
  std::cin >> P;
  std::cin >> T;
  result = kmp(P, T);
  for(auto iter = result.begin(); iter != result.end(); iter++)
  {
    std::cout << *iter;
    if(iter + 1 != result.end())
      std::cout << ",";
  }
  return 0;
}
```

cyclicShift.cpp

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

#define debug

std::vector<int> prefix_function (std::string s) {
  int n =  s.length();
  std::vector<int> pi(n, 0);                    //-создаем вектор необходимого размера
  for (int i=1; i<n; ++i) {
    int j = pi[i-1];                            //-вычисляем новое значение п-ф, про
    while (j > 0 && s[i] != s[j])               //-проверяем до тех пор, пока не дойдем до 0, либо пока
      j = pi[j-1];                              //символы не совпадут
    if (s[i] == s[j])                           //-при совпадении символов увеличиваем п-ф на единицу
      ++j;
    pi[i] = j;
  }
#ifdef debug
  std::cout << "prefix function: ";
  for(auto iter : pi)
    std::cout << iter << " ";
  std::cout << std::endl;
#endif
```

```
   return pi;
}

int cyclicShift(std::string& text, std::string& shiftText)
{
   bool cycle = false;
   int result = -1;
   std::vector<int> prefix;
   if(text.size() != shiftText.size())                    //-выходим из функции, так как дилны строк не равны
   {
#ifdef debug
      std::cout << "strings have different lengths "<< std::endl;
#endif
      return -1;
   }
   prefix = prefix_function(text);                        //-вычисляем префикс-функцию
   for(int k = 0, l = 0; k < shiftText.size();)
      if(shiftText[k] == text[l])                         //-при нахождении одинаковых символов сдвигаем счетчики
      {
         if(result == -1)                                 //-если до текущего шага не было найдено совпадений, то
            result = k;                                   //записываем индекс первого символа в сдвиге
#ifdef debug
         std::cout << "found equal symbols, shiftText[" << k << "] = " << shiftText[k] << ", " << "text[" << l << "] = "
<< text[l] << std::endl;
#endif
         k++;
         l++;
         if(l == text.size())                             //-при достижении конца строки T возвращаем индекс первого
         {                                                //символа из T в строке сдвига
#ifdef debug
            std::cout << "end text reached at k = " << k - 1 << std::endl;
            if(cycle)
            {
               std::cout << "result with cycle: " << result << std::endl;
            } else
            {
               std::cout << "result without cycle: " << result << std::endl;
            }
#endif
            return result;
         }
         if(k == shiftText.size())                        //-при достижении конца строки сдвига, обнуляем ее счетчик
         {
            k = 0;
            if(cycle)                                     //при повторном достижении конца строки завершаем работу
               return -1;
            cycle = true;
#ifdef debug
            std::cout << "end shiftText reached , k = " << k << ", l = " << l << std::endl;
#endif
         }
      } else
      {
#ifdef debug
         std::cout << "found different symbol, shiftText[" << k << "] = " << shiftText[k] << ", " << "text[" << l << "] = "
<< text[l] << std::endl;
#endif
         result = -1;                                     //-при несовпадении символов, удаляем запись о нахождении
         //первого символа
         if(l == 0)                                       //-увеличение счетчика строки сдвига
         {
            k++;
```

```
#ifdef debug
            std::cout << "k++, k = " << k << ", l = " << l << std::endl;
#endif
          } else
          {                                  //-присваиваем счетчику искомой строки предыдущее значение
            l = prefix[l-1];                        // префикс-функции
#ifdef debug
            std::cout << "shift l, k = " << k << ", l = " << l << std::endl;
#endif
          }
       }
#ifdef debug
   std::cout << "not a cycle shift " << std::endl;
#endif
   return result;
}

int main() {
   std::string shiftText, text;
   std::cin >> shiftText;
   std::cin >> text;
   std::cout <<  cyclicShift(text, shiftText);
   return 0;
}
```