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 _____ Деркач Н.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке.

Формулировка задачи КМП.

Реализуйте 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.

Вход:

Первая строка – P

Вторая строка – T

Выход:

Индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1

Sample input:

ab

abab

Sample output:

0, 2

Формулировка задачи сдвига.

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

Вариант — 2. Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца.

Суть и сложность алгоритмов.

Алгоритм КМП.

Считывается строка-образец и подаётся на вход префикс-функции. Префикс функция имеет два индекса: один для обхода префикса строки, второй для обхода суффикса. Если символы с этими индексами совпадают, то индексы смещаются дальше по строке и в ячейку массива значений префикс функции с индексом, равным индексу суффикса, записывается значение префикса + 1. Если же символы не равны, и индекс префикса стоит в начале строки, то под символом текущего суффикса значение префикс функции записывается равное 0 и суффикс расширяется. Если символы не равны и индекс префикса не нулевой, то индекс префикса меняется на значение, записанное в массиве префикс функции под индексом предыдущего символа строки в префиксе. Когда префикс функция заполнила массив значениями, алгоритм начинает считывать по одному символу строки-шаблона из потока. Два индекса указывают на текущий символ в образце и в строке поиска. Если очередной считанный символ из строки поиска не совпадает с текущим символом в образце, то текущим символом в образце становится символ с

индексом, равным значению префикс функции в предыдущем обработанном символе образца. Когда образец был пройден до конца, его вхождение в строку поиска было найдено. Оптимизация заключается в том, что строка поиска не хранится в памяти, а считывается из потока посимвольно, и хранится только один очередной символ.

Сложность алгоритма по времени – O(N + M), где N – длина строки поиска, M – длина образца.

Сложность алгоритма по памяти – O(M), где M – длина образца.

Циклический сдвиг.

Из потока считываются две строки, первая заносится в общий массив символов один раз, вторая — два раза. Этот массив отправляется на вход префикс-функции, которая заносит свои значения в вектор pFunc. Далее происходит обход этого вектора в цикле и если в нем встречается значение префикс функции, равное длине одной из строк(обе строки должны быть равными по длине, иначе одна из них не может быть циклическим сдвигом другой), то найден циклический сдвиг на значение индекса в векторе префикс-функции + 1 - двойная длина одной из строк.

Сложность алгоритма по времени — O(4*N), где N – длина строки

Сложность алгоритма по памяти — O(3*N), где N – длина строки

Описание функций и структур.

void **prefixFunction** (vector<int>& pFunc, string& string) — префикс-функция для алгоритма КМП. vector<int>& pFunc — ссылка на вектор, в который заносятся значения префикс-функции. string& string — ссылка на строку-образец.

vector <int> **prefixFunction**(vector <int> pFunc, char* string, int size) — префикс-функция для нахождения циклического сдвига. vector <int> pFunc — вектор значений префикс-функции. char* string — указатель на начало строки результата. int size — размер строки результата.

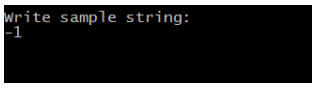
vector <int> **KMP**(string subStr) — функция нахождения всех вхождений подстроки в строку. string subStr — строка-образец. Функция возвращает вектор индексов всех вхождений подстроки в строку.

void **cyclic**(string& str1, string& str2) — функция проверяет, является ли вторая строка циклическим сдвигом первой. Если нет, то выводит значение -1. Если да, то выводит значение сдвига. string& str1 — первая строка, string& str2 — вторая строка.

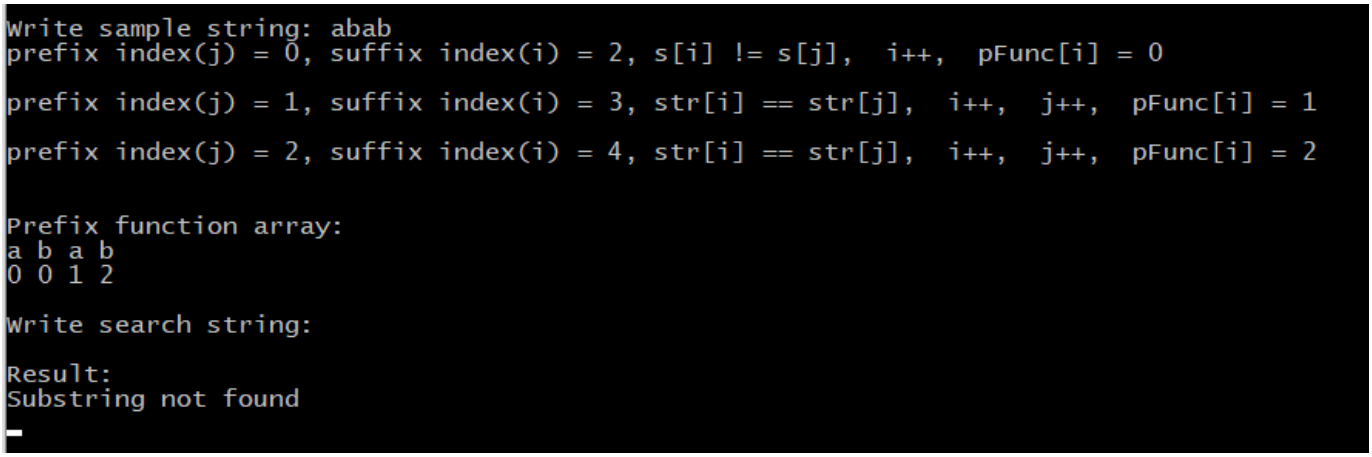
Тестирование.

Алгоритм КМП:

Тест 1(отутствие строки-образца)

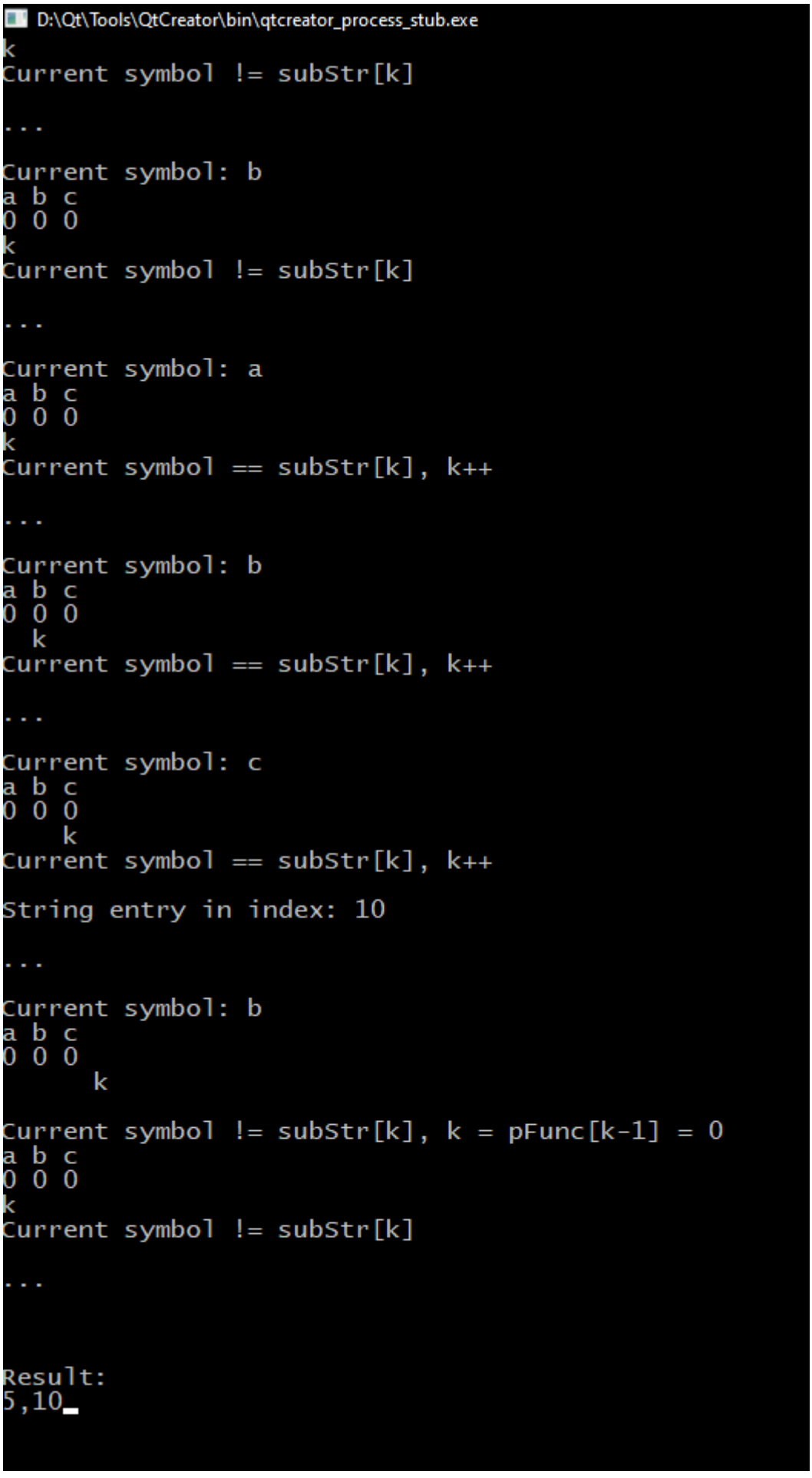


Тест 2(отутствие строки поиска)



Тест 3

```
D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Write sample string: abc
prefix index(j) = 0, suffix index(i) = 2, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 3, s[i] != s[j],  i++,  pFunc[i] = 0


Prefix function array:
a b c
0 0 0

Write search string: acbadabccbabcba
Current symbol: a
a b c
0 0 0
k
Current symbol == subStr[k], k++

...

Current symbol: c
a b c
0 0 0
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b c
0 0 0
k
Current symbol != subStr[k]

...

Current symbol: b
a b c
0 0 0
k
Current symbol != subStr[k]

...

Current symbol: a
a b c
0 0 0
k
Current symbol == subStr[k], k++

...

Current symbol: d
a b c
0 0 0
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b c
0 0 0
k
Current symbol != subStr[k]

...

Current symbol: a
a b c
```


D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
k
Current symbol != subStr[k]
...
Current symbol: b
a b c
0 0 0
k
Current symbol != subStr[k]
...
Current symbol: a
a b c
0 0 0
k
Current symbol == subStr[k], k++
...
Current symbol: b
a b c
0 0 0
k
Current symbol == subStr[k], k++
...
Current symbol: c
a b c
0 0 0
k
Current symbol == subStr[k], k++
String entry in index: 10
...
Current symbol: b
a b c
0 0 0
k
Current symbol != subStr[k], k = pFunc[k-1] = 0
a b c
0 0 0
k
Current symbol != subStr[k]
...
Result:
5,10

Тест 4:

```
Write sample string: abrakadabra
prefix index(j) = 0, suffix index(i) = 2, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 3, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 1, suffix index(i) = 4, str[i] == str[j],  i++,  j++,  pFunc[i] = 1

prefix index(j) = 0, suffix index(i) = 4, s[i] != s[j],  j = 0

prefix index(j) = 0, suffix index(i) = 5, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 1, suffix index(i) = 6, str[i] == str[j],  i++,  j++,  pFunc[i] = 1

prefix index(j) = 0, suffix index(i) = 6, s[i] != s[j],  j = 0

prefix index(j) = 0, suffix index(i) = 7, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 1, suffix index(i) = 8, str[i] == str[j],  i++,  j++,  pFunc[i] = 1

prefix index(j) = 2, suffix index(i) = 9, str[i] == str[j],  i++,  j++,  pFunc[i] = 2

prefix index(j) = 3, suffix index(i) = 10, str[i] == str[j],  i++,  j++,  pFunc[i] = 3

prefix index(j) = 4, suffix index(i) = 11, str[i] == str[j],  i++,  j++,  pFunc[i] = 4


Prefix function array:
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4

Write search string: brarabadarabrakadabradrbadarab
Current symbol: b
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...

Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
```

```
Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...

Current symbol: b
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k
Current symbol == subStr[k], k++

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
    k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...

Current symbol: d
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...

Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k
```

```
Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
          k
Current symbol == subStr[k], k++

...

Current symbol: d
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
            k
Current symbol == subStr[k], k++

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
              k
Current symbol == subStr[k], k++

...

Current symbol: b
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
                k
Current symbol == subStr[k], k++

...

Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
                  k
Current symbol == subStr[k], k++

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
                    k
Current symbol == subStr[k], k++

String entry in index: 10

...

Current symbol: d
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
                      k

Current symbol != subStr[k], k = pFunc[k-1] = 4
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
        k

Current symbol != subStr[k], k = pFunc[k-1] = 1
a b r a k a d a b r a
```

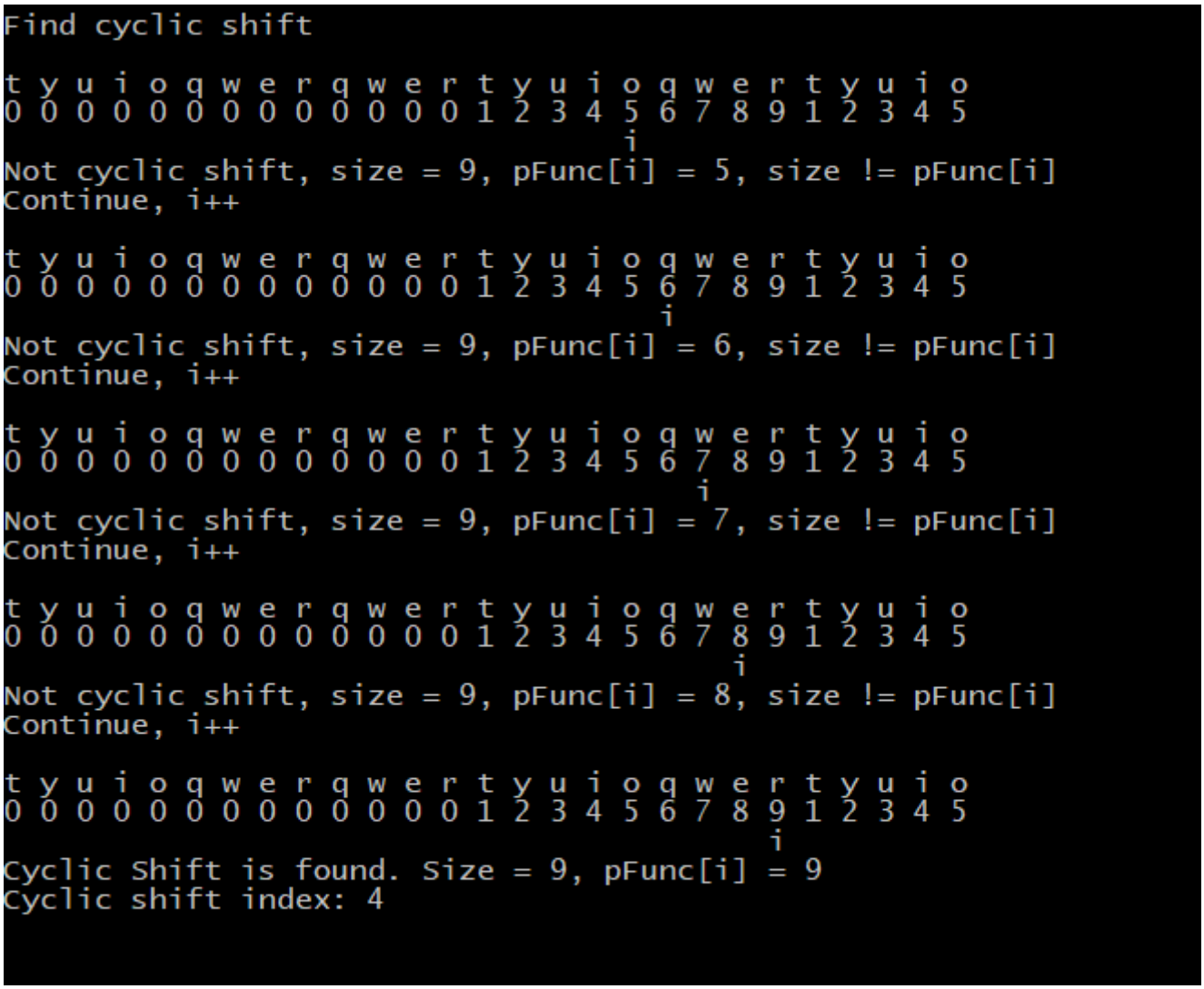
```
Current symbol != subStr[k], k = pFunc[k-1] = 1
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: b
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...

Current symbol: d
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...

Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k
```

```
Current symbol: r
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
  k

Current symbol != subStr[k], k = pFunc[k-1] = 0
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol != subStr[k]

...

Current symbol: a
a b r a k a d a b r a
0 0 0 1 0 1 0 1 2 3 4
k
Current symbol == subStr[k], k++

...



Result:
10
```

Циклический сдвиг:

Тест 1

```
Write first string: qwertyuio
Write second string: tyuioqwer
prefix index(j) = 0, suffix index(i) = 2, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 3, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 4, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 5, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 6, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 7, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 8, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 9, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 10, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 11, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 12, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 13, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 1, suffix index(i) = 14, str[i] == str[j],  i++,  j++,  pFunc[i] = 1

prefix index(j) = 2, suffix index(i) = 15, str[i] == str[j],  i++,  j++,  pFunc[i] = 2

prefix index(j) = 3, suffix index(i) = 16, str[i] == str[j],  i++,  j++,  pFunc[i] = 3

prefix index(j) = 4, suffix index(i) = 17, str[i] == str[j],  i++,  j++,  pFunc[i] = 4

prefix index(j) = 5, suffix index(i) = 18, str[i] == str[j],  i++,  j++,  pFunc[i] = 5

prefix index(j) = 6, suffix index(i) = 19, str[i] == str[j],  i++,  j++,  pFunc[i] = 6

prefix index(j) = 7, suffix index(i) = 20, str[i] == str[j],  i++,  j++,  pFunc[i] = 7

prefix index(j) = 8, suffix index(i) = 21, str[i] == str[j],  i++,  j++,  pFunc[i] = 8

prefix index(j) = 9, suffix index(i) = 22, str[i] == str[j],  i++,  j++,  pFunc[i] = 9

prefix index(j) = 0, suffix index(i) = 22, s[i] != s[j],  j = 0

prefix index(j) = 1, suffix index(i) = 23, str[i] == str[j],  i++,  j++,  pFunc[i] = 1

prefix index(j) = 2, suffix index(i) = 24, str[i] == str[j],  i++,  j++,  pFunc[i] = 2

prefix index(j) = 3, suffix index(i) = 25, str[i] == str[j],  i++,  j++,  pFunc[i] = 3

prefix index(j) = 4, suffix index(i) = 26, str[i] == str[j],  i++,  j++,  pFunc[i] = 4

prefix index(j) = 5, suffix index(i) = 27, str[i] == str[j],  i++,  j++,  pFunc[i] = 5


Prefix function:
t y u i o q w e r q w e r t y u i o q w e r t y u i o
0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5

Find cyclic shift
```


Find cyclic shift
t y u i o q w e r q w e r t y u i o q w e r t y u i o
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5
i
Not cyclic shift, size = 9, pFunc[i] = 5, size != pFunc[i]
Continue, i++
t y u i o q w e r q w e r t y u i o q w e r t y u i o
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5
i
Not cyclic shift, size = 9, pFunc[i] = 6, size != pFunc[i]
Continue, i++
t y u i o q w e r q w e r t y u i o q w e r t y u i o
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5
i
Not cyclic shift, size = 9, pFunc[i] = 7, size != pFunc[i]
Continue, i++
t y u i o q w e r q w e r t y u i o q w e r t y u i o
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5
i
Not cyclic shift, size = 9, pFunc[i] = 8, size != pFunc[i]
Continue, i++
t y u i o q w e r q w e r t y u i o q w e r t y u i o
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5
i
Cyclic Shift is found. Size = 9, pFunc[i] = 9
Cyclic shift index: 4

Тест 2

```
Write first string: qasxcvbgfd321
Write second string: bgfd321qasxcv
prefix index(j) = 0, suffix index(i) = 2, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 3, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 4, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 5, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 6, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 7, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 8, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 9, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 10, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 11, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 12, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 13, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 14, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 15, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 16, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 17, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 18, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 0, suffix index(i) = 19, s[i] != s[j],  i++,  pFunc[i] = 0
prefix index(j) = 1, suffix index(i) = 20, str[i] == str[j],  i++,  j++,  pFunc[i] = 1
prefix index(j) = 2, suffix index(i) = 21, str[i] == str[j],  i++,  j++,  pFunc[i] = 2
prefix index(j) = 3, suffix index(i) = 22, str[i] == str[j],  i++,  j++,  pFunc[i] = 3
prefix index(j) = 4, suffix index(i) = 23, str[i] == str[j],  i++,  j++,  pFunc[i] = 4
prefix index(j) = 5, suffix index(i) = 24, str[i] == str[j],  i++,  j++,  pFunc[i] = 5
prefix index(j) = 6, suffix index(i) = 25, str[i] == str[j],  i++,  j++,  pFunc[i] = 6
prefix index(j) = 7, suffix index(i) = 26, str[i] == str[j],  i++,  j++,  pFunc[i] = 7
prefix index(j) = 8, suffix index(i) = 27, str[i] == str[j],  i++,  j++,  pFunc[i] = 8
prefix index(j) = 9, suffix index(i) = 28, str[i] == str[j],  i++,  j++,  pFunc[i] = 9
prefix index(j) = 10, suffix index(i) = 29, str[i] == str[j],  i++,  j++,  pFunc[i] = 10
prefix index(j) = 11, suffix index(i) = 30, str[i] == str[j],  i++,  j++,  pFunc[i] = 11
prefix index(j) = 12, suffix index(i) = 31, str[i] == str[j],  i++,  j++,  pFunc[i] = 12
prefix index(j) = 13, suffix index(i) = 32, str[i] == str[j],  i++,  j++,  pFunc[i] = 13
```

```
Not cyclic shift, size = 13, pFunc[i] = 12, size != pFunc[i]
Continue, i++

b g f d 3 2 1 q a s x c v q a s x c v b g f d 3 2 1 q a s x c v b g f d 3 2 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 1 2 3 4 5 6 7
                                                                  i

Cyclic Shift is found. Size = 13, pFunc[i] = 13
Cyclic shift index: 6
```

Тест 3

```
D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Write first string: shalash
Write second string: shalash
prefix index(j) = 0, suffix index(i) = 2, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 3, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 4, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 0, suffix index(i) = 5, s[i] != s[j],  i++,  pFunc[i] = 0

prefix index(j) = 1, suffix index(i) = 6, str[i] == str[j],  i++,  j++,  pFunc[i] = 1

prefix index(j) = 2, suffix index(i) = 7, str[i] == str[j],  i++,  j++,  pFunc[i] = 2

prefix index(j) = 0, suffix index(i) = 7, s[i] != s[j],  j = 0
```

```
Prefix function:
s h a l a s h s h a l a s h s h a l a s h
0 0 0 0 0 1 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Find cyclic shift

s h a l a s h s h a l a s h s h a l a s h
0 0 0 0 0 1 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14
                        i
Cyclic Shift is found. Size = 7, pFunc[i] = 7
Cyclic shift index: 0
```

Тест 4

```
D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Write first string: abcd
Write second string: bc
-1
```

Выводы.

В лабораторной работе был изучен алгоритм Кнута-Морриса-Пратта нахождения подстроки в строке путём написания программы на языке C++.

Приложение А. Исходный код.

КМП:

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

using namespace std;

void prefixFunction (vector<int>& pFunc, string& string) {
    pFunc[0] = 0;
    int j = 0;            // индекс префикса
    int i = 1;            // индекс суффикса

    while(i < string.size()){       //Пока не пройдены все символы образца
        if (string[i] == string[j]){     //Если символы i и j индекса совпадают, то
оба смещаются вправо, изменяется массив префикс-функций
            pFunc[i] = j+1;
            i++;
            j++;
            cout << "prefix index(j) = " << j << ", suffix index(i) = " << i << ",
str[i] == str[j],  i++,  j++,  pFunc[i] = " << j << endl << endl;
        }
        else{
            if (j==0){                  //Иначе, если j указывает на начальный символ
строки, то на i символ значение префикс-функции 0
                pFunc[i] = 0;
                i++;                    //i-тый указатель смещается вправо
                cout << "prefix index(j) = " << j << ", suffix index(i) = " << i <<
", s[i] != s[j],  i++,  pFunc[i] = 0" << endl << endl;
            }
            else{
                j = pFunc[j-1];            //Если указатель j не указывает на
начальный символ, то сравниваются префиксы и суффиксы меньшего размера
                cout <<  "prefix index(j) = " << j << ", suffix index(i) = " << i <<
", s[i] != s[j],  j = " << pFunc[j] << endl << endl;
            }
        }

    }
    cout << "\nPrefix function array:\n";
    for (char k : string)
```

```
        cout << k << " ";
    cout << endl;
    for (int i=0; i<string.size(); i++)
        cout << pFunc[i] << " ";
    cout <<"\n\n";
}

vector <int> KMP(string subStr) {
    vector <int> result;                            // Вектор индексов вхождения
подстроки
    vector <int> pFunc(subStr.size());             // Вектор значений префикс-функций
    prefixFunction(pFunc, subStr);
    int k = 0;                          // Индекс текущего элемента в строке-образце
    int l = 1;                          // Счетчик символов в строке поиска
    char ch;
    cout << "Write search string: ";
    cin >> ch;
    while (cin.peek() != '\n') {
        cout << "Current symbol: " << ch << endl;
        for (int i = 0; i < subStr.size(); i++){
            char sym = subStr[i];
            cout << sym << " ";
        }
        cout << endl;
        for (int i = 0; i < pFunc.size(); i++){
            cout << pFunc[i] <<" ";
        }
        cout << endl;
        for (int i=0; i<k; i++)
            cout << "  ";
        cout << "k" << endl;
        while ((k > 0) && (subStr[k] != ch)) {             //Если считанный символ не
удовлетворяет условию, то изменяется индекс k
            k = pFunc[k-1];
            cout << "\nCurrent symbol != subStr[k], k = pFunc[k-1] = " << k << endl;
            for (int i = 0; i < subStr.size(); i++){
                char sym = subStr[i];
                cout << sym << " ";
            }
            cout << endl;
            for (int i = 0; i < pFunc.size(); i++){
                cout << pFunc[i] <<" ";
            }
            cout << endl;
            for (int i=0; i<k; i++)
```

```
                cout << "   ";
            cout << "k" << endl;
        }

        if (subStr[k] == ch) {          // Если считанный символ совпадает с текущим в
образце, то индекс k сдвигается вправо
            k++;
            cout << "Current symbol == subStr[k], k++" << endl << endl;
        }
        else{
            cout << "Current symbol != subStr[k]"<< endl <<endl;
        }

        if (k == subStr.size()) {           // Если найдена подстрока, то индекс
заносится в массив результата
            result.push_back(l - subStr.size());
            cout << "String entry in index: " << l - subStr.size() << endl<< endl;
        }
        l++;
        cout << "...\n" << endl;
        cin >> ch;
    }

    if (result.empty()) {                   //Если не было найдено подстроки
        result.push_back(-1);
    }
    return result;
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    string subStr;
    cout << "Write sample string: ";
    cin >> subStr;
    vector<int> result = KMP(subStr);           //Запуск алгоритма Кнута-Морриса-
Пратта
    cout << "\n\nResult:" << endl;      //Вывод результата
    if (result[0] == -1)
        cout << "Substring not found" << endl;
    else {
        for (int i = 0; i < result.size(); i++) {
            cout << result[i];
            if (i != result.size() - 1) {
                cout << ',';
```

```
            }
        }
    }
    return a.exec();
}
```

Циклический сдвиг:

```
#include <QCoreApplication>
#include <iostream>
#include <fstream>

using namespace std;

vector <int> prefixFunction(vector <int> pFunc, char* string, int size) {
    for (int i=0; i<size; i++)
        pFunc.push_back(0);
    int j = 0;            // индекс префикса
    int i = 1;            // индекс суффикса
    while(i < size){       //Пока не пройдены все символы образца
        if (string[i] == string[j]){     //Если символы i и j индекса совпадают, то
оба смещаются вправо, изменяется массив префикс-функций
            pFunc[i] = j+1;
            i++;
            j++;
            cout << "prefix index(j) = " << j << ", suffix index(i) = " << i << ",
str[i] == str[j],  i++,  j++,  pFunc[i] = " << j << endl << endl;
        }
        else{
            if (j==0){                  //Иначе, если j указывает на начальный символ
строки, то на i символ значение префикс-функции 0
                pFunc[i] = 0;
                i++;                    //индекс суффикса смещается вправо
                cout << "prefix index(j) = " << j << ", suffix index(i) = " << i <<
", s[i] != s[j],  i++,  pFunc[i] = 0" << endl << endl;
            }
            else{
                j = pFunc[j-1];            //Если индекс префикса не указывает на
начальный символ, то сравниваются префиксы и суффиксы меньшего размера
                cout <<  "prefix index(j) = " << j << ", suffix index(i) = " << i <<
", s[i] != s[j],  j = " << pFunc[j] << endl << endl;
            }
        }
```

```
    }

    cout << "\nPrefix function:\n";
    for (int i=0; i<size; i++)
        cout << string[i] << " ";
    cout << endl;
    for (int i=0; i<size; i++)
        cout << pFunc[i] << " ";
    cout <<"\n\n";
    return pFunc;
}

void cyclic(string& str1, string& str2) {
    if (str1.size() != str2.size()) {
        cout << -1;                                          //Если строки не равны
по длине, то одна не может быть циклическим сдвигом другой
        return;
    }

    int size = str1.size();

    char* temp = new char[3 * size];                         // Массив для поиска
результата

    int i = 0;
    while (i < size) {
        temp[i] = str2[i];          // Записывается первая строка
        ++i;
    }

    for (int k = 0; k < 2; ++k) {                            // Два раза записывается
вторая строка
        for (int j = 0; j < size; ++j) {
            temp[i++] = str1[j];
        }
    }
    vector <int> pFunc;
    pFunc = prefixFunction(pFunc, temp, size * 3);           // Получается массив их
префикс функций

    cout << "Find cyclic shift" << endl<<endl;
    for (int i = 2 * size - 1; i < 3 * size; ++i) {          // Обход массива
результата

        for (int k = 0; k < size*3; k++)                 // Вывод промежуточных
результатов
```

```
            cout << temp[k] << " ";
        cout << endl;
        for (int k = 0; k < size*3; k++)
            cout << pFunc[k] <<" ";
        cout << endl;
        for (int k=0; k < i; k++)
            cout << "  ";
        cout << "i ";
        if (pFunc[i] == size) {                                  //Если значение
префикс функции на символе равно размеру введенной строки, то найден циклический
сдвиг
            cout << "\nCyclic Shift is found. Size = " << size << ", pFunc[i] = "
<<pFunc[i] <<endl;
            cout <<"Cyclic shift index: "<< i + 1 - 2 * size;    //Вывод результата
            delete [] temp;
            return;
        }
        else{
            cout << "\nNot cyclic shift, size = " << size << ", pFunc[i] = "
<<pFunc[i] << ", size != pFunc[i]"<<endl;
            cout << "Continue, i++" << endl << endl;
        }
    }
    cout << -1;                              //Если циклический сдвиг не найден
    delete [] temp;
}

int main(int argc, char *argv[]) {
    QCoreApplication a(argc, argv);
    string str1;
    string str2;
    cout << "Write first string: ";
    cin >> str1;
    cout << "Write second string: ";
    cin >> str2;

    cyclic(str1, str2);  //Запуск алгоритма поиска циклического сдвига

    return a.exec();
}
```