

**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. 8304

\_\_\_\_\_

Сергеев А.Д.

Преподаватель

\_\_\_\_\_

Размочаева Н.В.

Санкт-Петербург

2020

## Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для оптимального поиска всех вхождений подстроки в строку.

## Задание.

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

2. Заданы две строки  $A$  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.

Дополнительное задание (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).

### **Порядок выполнения работы.**

Написание работы производилось на базе операционной системы Windows 10 на языке программирования kotlin в среде программирования IntelliJ IDEA и на языке программирования c++ в среде программирования CLion.

В связи со спецификацией дополнительного задания (длина образца гораздо меньше длины строки поиска) выполнение подготовки к распараллеливанию возможно только для первой части работы, во второй в любом случае длина образца в два раза короче длины строки поиска.

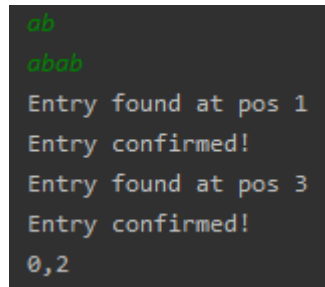
Первая программа состоит из трёх функций, расширяющих тип String. Функция `prefix` вычисляет значение префикс-функции для данной строки. Функция `findAll` реализует стандартный алгоритм Кнута-Морриса-Пратта для данной строки, строки образца и массива значений префикс-функции для данного образца. Она возвращает объект класса данных `ListAndInt`, содержащий в себе список вхождений образца в данную строку (переменная `list`) и целое число, равное длине максимального постфикса данной строки, являющегося одновременно префиксом образца (переменная `int`). Функция `findAllSync` делит данную строку на 100 частей, если образец короче её в 100 и более раз. Потом для каждой из полученных подстрок вызывается функция `findAll`, а их результаты объединяются. В случае, если одна или несколько объектов `ListAndInt`, полученных от `findAll` содержат переменную `int` больше 0, иницируется дополнительный поиск на подстроке, равной длине образца + `int`. Если образец короче данной строки менее чем в 100 раз, то вызывается функция `findAll` для всей данной строки целиком.

Вторая программа состоит из двух функций. Функция `prefix` вычисляет значение префикс-функции для данной строки. Функция `isCyclic` принимает на вход две строки и выполняет алгоритм Кнута-Морриса-Пратта, дважды проходя по строке поиска.

Описание классов в UML-виде приложено к отчету в файлах `UML.K.png` и `UML.C.png`.

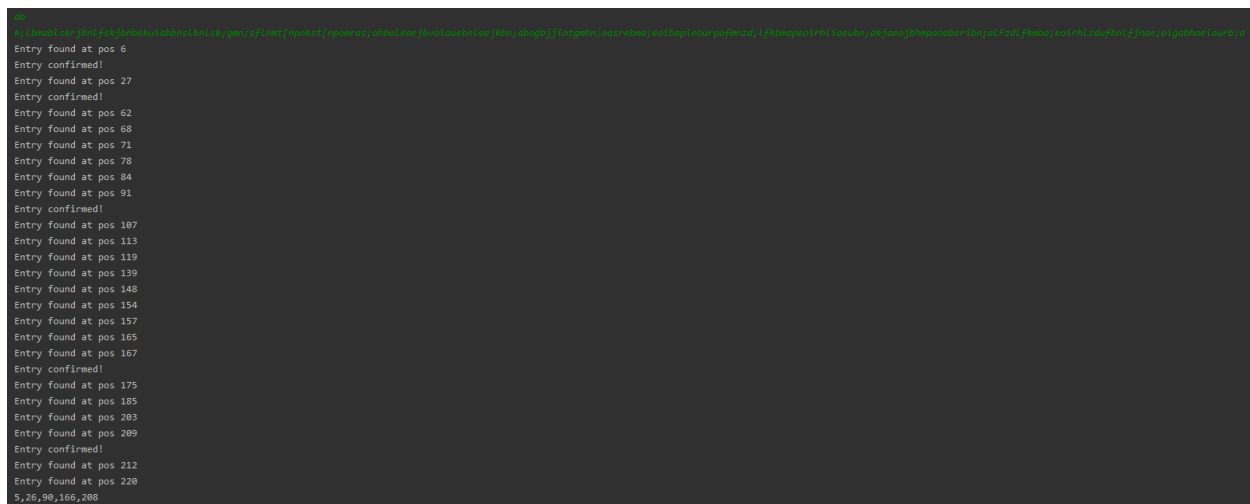
### Тестирование.

Тестирование первой программы:



```
ab
abab
Entry found at pos 1
Entry confirmed!
Entry found at pos 3
Entry confirmed!
0,2
```

Рисунок 1 - Вывод первой программы при образце короче строки поиска менее чем в 100 раз



```
ab
Entry found at pos 6
Entry confirmed!
Entry found at pos 27
Entry confirmed!
Entry found at pos 62
Entry found at pos 68
Entry found at pos 71
Entry found at pos 78
Entry found at pos 84
Entry found at pos 91
Entry confirmed!
Entry found at pos 107
Entry found at pos 113
Entry found at pos 119
Entry found at pos 139
Entry found at pos 148
Entry found at pos 154
Entry found at pos 157
Entry found at pos 165
Entry found at pos 167
Entry confirmed!
Entry found at pos 175
Entry found at pos 185
Entry found at pos 203
Entry found at pos 209
Entry confirmed!
Entry found at pos 212
Entry found at pos 220
5,26,90,166,288
```

Рисунок 2 - Вывод первой программы при образце короче строки поиска более чем в 100 раз

Тестирование второй программы:

```
defabc  
abcdef  
Entry found at pos: 4  
Entry confirmed!  
3
```

Рисунок 3 - Вывод второй программы, если A является циклическим сдвигом B

```
defabc  
abcdef  
Entry found at pos: 4  
Entry confirmed!  
3
```

Рисунок 4 - Вывод второй программы, если A не является циклическим сдвигом B

### **Вывод.**

В результате лабораторной работы были изучены алгоритм Кнута-Морриса-Пратта и префикс-функция.

## Приложение А

### Исходный код программы, файл Main.kt

```
private data class ListAndInt(var list: List<Int>, var int: Int)

fun main() {
    val mark = readLine()!!
    val base = readLine()!!
    val entryArray = base.findAllSync(mark)
    println(if (entryArray.isEmpty()) "-1" else
entryArray.joinToString(", "))
}

private fun String.prefix() : IntArray {
    val pi = IntArray(this.length)
    var k = 0
    for (q in 1 until this.length) {
        if (k > 0 && this[k] != this[q]) k = pi[k]
        if (this[k] == this[q]) k += 1
        pi[q] = k
    }
    return pi
}

private fun String.findAll(mark : String, prf : IntArray, offset
: Int) : ListAndInt {
    val answer = mutableListOf<Int>()
    var counter = 0
    for (i in this.indices) {
        while (counter > 0 && this[i] != mark[counter]) counter
= prf[counter - 1]
```

```

        if (this[i] == mark[counter]) {
            if (counter == 0) println("Entry found at pos ${i +
offset + 1}")
            if (counter == mark.length - 1) println("Entry
confirmed!")
            counter++
        }
        if (counter == mark.length) {
            answer.add(i - mark.length + 1)
            counter = prf[counter - 1]
        }
    }
    return ListAndInt(answer, counter)
}

```

```

fun String.findAllSync(mark : String) : IntArray {
    val prefixes = mark.prefix()
    return if (mark.length > this.length / 100) {
        findAll(mark, prefixes, 0).list.toIntArray()
    } else {
        val ptL = mark.length * 100
        val answer = mutableListOf<Int>()
        for (i in 0 until (this.length / ptL)) {
            var subs = this.substring(i * ptL, (i + 1) * ptL)
            var result = subs.findAll(mark, prefixes, i * ptL)
            answer.addAll(result.list.map { it + i * ptL })
            val offset = result.int
            val tailLen = offset + mark.length - 1
            subs = this.substring((i + 1) * ptL - offset, ((i +
1) * ptL - offset + tailLen).coerceAtMost(this.length))
            result = subs.findAll(mark, prefixes, (i + 1) * ptL
- offset)

```

```

        answer.addAll(result.list.map { it + i * ptL + ptL -
offset })
    }
    val result = this.substring(this.length - (this.length %
ptL)).findAll(mark, prefixes, this.length - (this.length % ptL))
    answer.addAll(result.list.map { it + this.length -
(this.length % ptL) })
    answer.toIntArray()
}
}

```



## Приложение Б

### Исходный код программы, файл main.cpp

```
#include <iostream>

void prefix(std::string& mark, int** prefix) {
    int k = 0;
    (*prefix)[0] = 0;
    for (int i = 0; i < mark.size(); ++i) {
        if (k > 0 && mark[k] != mark[i]) k = (*prefix)[k];
        if (mark[k] == mark[i]) k += 1;
        (*prefix)[i] = k;
    }
}

int isCyclic(std::string& a, std::string& b) {
    int* prf = ((int*) (calloc(b.size(), sizeof(unsigned
int*)))));
    prefix(b, &prf);
    int counter = 0;
    for (int i = 0; i < a.size() * 2; ++i) {
        if (counter > 0 && a[i % a.size()] != b[counter])
counter = prf[counter - 1];
        if (a[i % a.size()] == b[counter]) {
            if (!counter) std::cout << "Entry found at pos: " <<
i % (a.size()) + 1 << std::endl;
            if (counter == a.size() - 1) std::cout << "Entry
confirmed!" << std::endl;
            counter++;
        }
    }
    if (counter == b.size()) {
        return i - a.size() + 1;
    }
}
```

```

        }
    }
    return -1;
}

int main() {
    std::string a = std::string(), b = std::string();
    std::cin >> a;
    std::cin >> b;
    if (a.size() != b.size()) {
        std::cout << -1;
    } else {
        std::cout << isCyclic(a, b);
    }
}

```