**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математической кибернетики и информационных технологий

**Отчет по лабораторной работе №3**

По дисциплине «СиАОД»

На тему:

«Методы поиска подстроки в строке»

Выполнил: студент группы БВТ1801

Задоркин Максим Александрович

Руководитель:

Кутейников Иван Алексеевич

Москва 2020

**Цель работы:**   
Изучить основные методы поиска подстроки в строке и сравнить их скорость.

**Задача (Вариант 10):**Реализовать методы поиска:1. Реализовать на языке Java метод Кнута-Морриса-Пратта.   
2. Реализовать поиск стандартной библиотекой Java.  
Сравнить все вышеперечисленные методы по времени.

**Программа:**

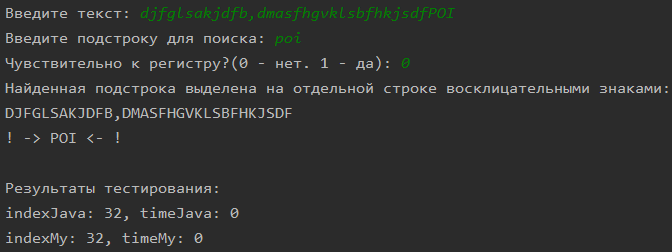
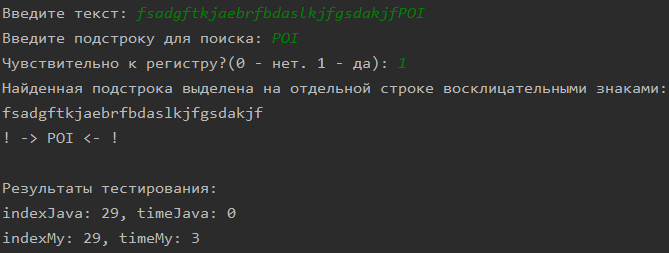
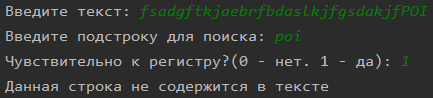
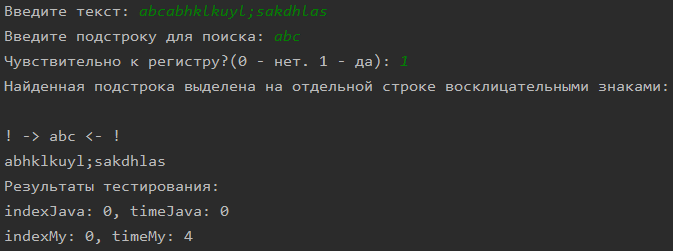
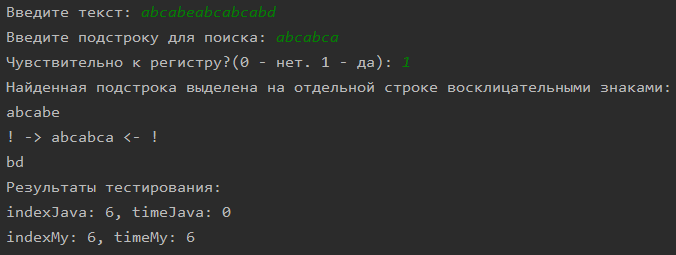
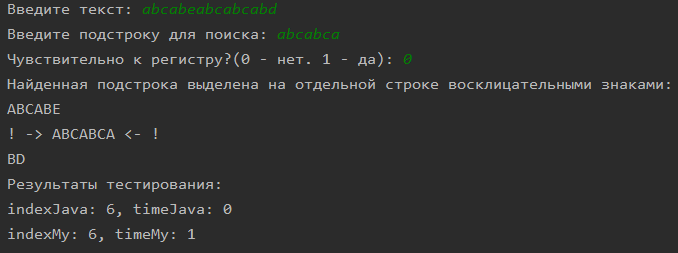
**1.** Код программы:

public class Main {  
 public static void main(String args[]) {  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.print("Введите текст: ");  
 String text = input.nextLine();  
 System.*out*.print("Введите подстроку для поиска: ");  
 String str = input.nextLine();  
 System.*out*.print("Чувствительно к регистру?(0 - нет. 1 - да): ");  
 int caseOfLetters = input.nextInt();  
  
 // чувствительность к регистру  
 if (caseOfLetters == 0) {  
 text = text.toUpperCase();  
 str = str.toUpperCase();  
 }  
  
 long time1 = System.*currentTimeMillis*();  
 // создаем префикс функцию  
 int piArray[] = new int[str.length()];  
 KMP findStr = new KMP(text, str);  
 findStr.prefixFunc(piArray);  
  
 // осуществляем поиск  
 int result = findStr.KMPFunc(piArray);  
 long timeMy = System.*currentTimeMillis*() - time1;  
  
 // выводим результат на экран  
 if (result == -1) {  
 System.*out*.println("Данная строка не содержится в тексте");

return;  
 } else {  
 // Выводим на экран  
 System.*out*.println("Найденная подстрока выделена на отдельной строке восклицательными знаками:");  
 findStr.showResultKMP(result);  
 }  
  
 time1 = System.*currentTimeMillis*();  
 int indexJava = text.indexOf(str);  
 long timeJava = System.*currentTimeMillis*() - time1;  
  
 System.*out*.println("\nРезультаты тестирования:");  
 System.*out*.println("indexJava: " + indexJava + ", timeJava: " + timeJava);  
 System.*out*.println("indexMy: " + result + ", timeMy: " + timeMy);  
  
 input.close();  
 }  
}

public class KMP {  
 public String text;  
 public String str;  
 public KMP(String textInput, String strInput) {  
 text = textInput;  
 str = strInput;  
 }  
 public void prefixFunc(int piArray[]) {  
 if (text.length() < str.length()) {  
 System.*out*.println("Ошибка. Длина подстроки должна быть меньше длины строки");  
 return;  
 }  
 piArray[0] = 0;  
 if (str.length() < 2) {  
 return;  
 }  
 for (int i = 1; i < str.length(); i += 1) {  
 int pIndex = 0;  
 for (int j = 1; j <= i; j += 1) {  
 String prefix = str.substring(0, j); // увеличиваем на каждой итерации значение префикса на один символ  
 String suffix = str.substring(i + 1 - j, i + 1); // увеличиваем значение суффикса на 1 символ каждую итерацию  
 if (prefix == suffix && pIndex < prefix.length()) {  
 pIndex = prefix.length();  
 }  
 }  
 piArray[i] = pIndex;  
 }  
 }  
 // Передаем префикс-функцию и возвращаем индекс с которого начинается подстрока в тексте  
 public int KMPFunc(int piArray[]) {  
 int index = 0;  
 while (index < text.length()) {  
 for (int j = 0; j < str.length(); j += 1) {  
 // добавить условие если первая буква не подходит  
 if (str.charAt(j) != text.charAt(index + j)) {  
 if (j == 0) { // если это первая буква  
 index += 1;  
 } else {  
 index += j - piArray[j - 1];  
 }  
 break;  
 }  
 if (j == str.length() - 1) {  
 return index;  
 }  
 }  
 }  
 return -1;  
 }  
 public void showResultKMP(int index) {  
 System.*out*.print(text.substring(0, index));  
 System.*out*.print("\n! -> " + text.substring(index, index + str.length()) + " <- !\n");  
 System.*out*.print(text.substring(index + str.length()));  
 }  
}

**2.** Тестирование программы



**4.** Для реализации поиска стандартными средствами java был выбран поиск indexOf().

Сравнение времени выполнения(скорости) методов:

|  |  |  |  |
| --- | --- | --- | --- |
| **Длина текста** | **Длина подстроки** | **Алгоритм Кнута-Морриса-Пратта** | **Поиск Java** |
| 10000 | 17 | 2 | 0 |
| 106780 | 50 | 12 | 2 |
| 364474 | 50 | 18 | 3 |
| 1008645 | 1000 | 21 | 6 |
| 10010645 | 2000 | 481 | 0 |
| 1010659 | 13 | 21 | 7 |

**Вывод:** из вышеперечисленных данных можно сделать вывод, что алгоритм КМП уступает стандартному алгоритму java и из тестов можно сделать вывод, основываясь преимущественно на последних двух тестах, что большинство проблем связанно из-за составления префикс функции, которая отнимает большую часть времени работы алгоритма.