Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2019

Trabajo Práctico No. 1

La resolución de este trabajo práctico debe ser enviada a través del moodle de la materia `http://dc.exa.unrc.edu.ar/moodle`, antes de las **23:55 del viernes 19 de Abril de 2019**.
El código provisto como parte de la solución a los ejercicios deberá estar documentado apropiadamente (por ejemplo, con comentarios en el código). Aquellas soluciones que no requieran programación, como así también la documentación adicional de código que se desee proveer, debe entregarse en **archivos de texto convencionales o archivos en formato PDF** unicamente, con nombres que permitan identificar facilmente su contenido.
Tanto la calidad de la solución, como el código y su documentación serán considerados en la calificación. Recuerde además que los trabajos prácticos **no tienen recuperación**, y que el trabajo en la resolución de los ejercicios debe realizarse en grupos de 3 personas.

1. *String matching* es el problema de encontrar todas las ocurrencias de un patrón de caracteres (subcadena) en un texto. La estrategia de String matching juega un rol clave en muchas aplicaciones de software que van desde simples editores de texto hasta el complejo NIDS (Sistema de detección de intrusos). Existen numerosas aplicaciones conocidas que incluyen String matching como un aspecto importante de su funcionalidad, como filtros de spam, motores de búsqueda, detección de plagios, bioinformática, entre otros. Dada la relevancia y aplicabilidad de este problema, existen diferentes algoritmos que dan solución a String Matching.

   Como parte de la solución a este trabajo práctico, se pide implementar, en Java, dos algoritmos que resuelvan este problema:

   - una solución deberá ser utilizando la técnica *Fuerza Bruta* (ver ejemplo en las notas teóricas)
   - una segunda solución con el algoritmo de *Knuth-Morris-Pratt* (KMP)[1], el cual compara el patrón con el texto de izquierda a derecha de manera tal que en caso de discrepancia, la palabra en sí misma contiene información suficiente para determinar dónde podría comenzar la siguiente coincidencia, y de esta manera se omite la reexaminación de los caracteres previamente coincidentes.

   Para la solución al problema planteado, deberá proveer una API que contenga las siguientes funciones:

```
public class StringMatching{

 /**
 * Calcula String Matching usando Fuerza Bruta.
 * @param text texto a analizar
 * @param pattern patron a buscar en el texto
 * @returns indice de la primera ocurrencia de pattern en text, en caso que exista,
 *          o -1 en caso que no se encuentre.
*/
public static int match(String text, String pattern) {
  return -1; // resolver el problema.
}

/**
```

[1] en.wikipedia.org/Knuth–Morris–Pratt_algorithm

```
     * Calcula String Matching usando Knuth-Morris-Pratt.
     * @param text texto a analizar
     * @param pattern patron a buscar en el texto
     * @returns indice de la primera ocurrencia de pattern en text, en caso que exista,
     */
    public static int matchKmp(String text, String pattern) {
      return -1; // resolver el problema.
    }


}
```

2. En Ciencias de la Computación, el problema *longest repeated substring* consiste en encontrar la subcadena más larga de un texto que ocurre al menos dos veces, esto es, dado un texto t, encontrar α de mayor longitud, tal que $\alpha\alpha$ es subcadena de t. Este problema tiene una gran importancia en una variedad de aplicaciones que incluyen biología molecular computacional, minería de datos, compresión de datos y análisis de música.

   Como parte de la solución a este trabajo práctico, se pide implementar, en Java, dos algoritmos que resuelvan este problema:

   - una solución deberá ser a través de la técnica de *Fuerza Bruta*.
   - una segunda solución a través de la técnica *Divide&Conquer*.

   Para la solución al problema planteado, deberá proveer una API que contenga las siguientes funciones:

```
public class LongestRepetition{

    /**
     * Calcula Longest repetition usando Fuerza Bruta.
     * @param text texto a analizar
     * @returns subString de mayor longitud que se repite.
    */
    public static String repetition(String text) {
      return ""; // resolver el problema.
    }

    /**
     * Calcula Longest repetition usando D&C.
     * @param text texto a analizar
     * @returns subString de mayor longitud que se repite.
     */
    public static String repetitionDc(String text) {
      return ""; // resolver el problema.
    }

}
```

Importante

Además deberá incluir, como parte del TP, para los dos ejercicios detallados previamente, un análisis experimental de tiempo y corrección, para medir la performance de los programas implementados, comparando el desempeño entre ellos. Para ello se debe preparar un conjunto de casos de test que permitan observar los tiempos de ejecución en función de los parámetros de entrada, analizando la idoneidad de cada uno de los métodos programados para diferentes tipos de instancias.

Para la corrección de este trabajo práctico se considerará, no sólo si el código fuente provisto implementa la solución propuesta, sino que además debe seguir las buenas prácticas de la programación, entre ellas podemos mencionar:

- comentarios pertinentes,

- nombres de variables apropiados,
- estilo de indentación coherente
- modularización adecuada
- estructura de clases adecuada

Se pueden utilizar herramientas que asisten y ayudan a seguir convenciones, entre ellas: *checkstyle*, para asistencia de estilos para el lenguaje de programación Java, *maven*, ayuda en la estructura de proyectos Java.