ALGORITMOS PARA MEDIR EL NIVEL DE PLAGIO Y SU POSIBLE IMPLEMENTACIÓN EN LA COMPARACIÓN DE CÓDIGOS DE PROGRAMACIÓN

ALGORITHMS FOR MEASURING THE LEVEL OF PLAGIARISM AND ITS POSSIBLE IMPLEMENTATION IN THE COMPARISON OF PROGRAMMING CODES

Francisco Gutiérrez Vera
Tecnológico Nacional de México / IT de Celaya, México
francisco.gv@celaya.tecnm.mx

Claudia Cristina Ortega González
Tecnológico Nacional de México / IT de Celaya, México
claudia.og@celaya.tecnm.mx

José Guillermo Fierro Mendoza
Tecnológico Nacional de México / IT de Celaya, México
guillermo.fm@celaya.tecnm.mx

Julio Armando Asato España
Tecnológico Nacional de México / IT de Celaya, México
julio.ae@celaya.tecnm.mx

Recepción: 8/junio/2022 **Aceptación:** 30/junio/2022

Resumen

El presente artículo trata acerca de algunos de los algoritmos existentes para la comparación de textos, se incluye un análisis del funcionamiento de ellos hacia la posible implementación como herramienta auxiliar en la medición del plagio de programas de computo que los estudiantes entregan a nivel universitario. Si bien es conocido que el internet es una gran herramienta de exploración, las generaciones actuales abusan al buscar de forma automática o natural cualquier tarea del área de programación, por otra parte, el plagio consensuado o sin consentimiento se está dando entre los seres humanos, para establecer un dato de partida sobre el nivel de plagio de programas de cómputo se realizó una pequeña encuesta referente a cuantas personas bajan códigos de internet y los entregan como propios o no respetando el derecho de autor. Una vez hecho el análisis de los algoritmos se agregó una reflexión del impacto de su uso de este tipo de algoritmos en plataformas

comerciales o personales, para ayudar a disminuir el nivel de plagio de programas de cómputo.

Palabras clave: Plagio, algoritmo de comparación, programas de clase.

Abstract

This article deals with some of the existing algorithms for the comparison of texts, including an analysis of how they work towards the possible implementation as an auxiliary tool in the measurement of the plagiarism of computer programs that students deliver at the university level. While it is known that the Internet is a great tool for exploration, current generations abuse when searching automatically or naturally for any task in the programming area, on the other hand, the consensual plagiarism or without consent is being given among the human beings, to establish a starting point on the level of plagiarism of computer programs was conducted a small survey concerning how many people download codes of the Internet and give them as their own or not respecting the copyright. Once the analysis of the algorithms was done, a reflection of the impact of their use of this type of algorithms was added on commercial or personal platforms, to help decrease the level of plagiarism of computer programs.

1. Introducción

El plagió de obras ya sean artísticas o literarias es una actividad realizada desde hace mucho tiempo con o sin fin de lucro, Astudillo [2006] en su trabajo el plagio intelectual, recopila varias definiciones de este concepto en donde resaltaremos la que establece que el plagió es «...el acto de ofrecer o presentar como propia, en su totalidad o en parte, la obra de otra persona, en una forma o contexto más o menos alterados...». En el medio universitario y muy particularmente en las asignaturas que pretenden desarrollar la habilidad de programación, una de las actividades más comunes por parte de los profesores es proponer a los estudiantes una serie de problemas a resolver, que normalmente son grupales, es decir, el mismo problema a todo el grupo. A fin de sustentar el trabajo de investigación presentado en este artículo, se realizó una encuesta a cien estudiantes de nivel universitario que incluyó

las siguientes preguntas: "¿Has bajado códigos de internet y entregado como propios?", "¿Le has entregado a compañeros alguno de tus programas?" "¿Has pedido que te pasen los programas que tienes que resolver? En la figura 1 se muestra el resultado de estas preguntas. Se puede observar que el 85% de los encuestados buscan los códigos en internet, en lugar de hacer un desarrollo propio, de la segunda pregunta se puede observar que el 70% comparte su obra, y de la última pregunta se observa que 50% acepto que piden códigos. Sin querer formar un juicio definitivo sobre los resultados de la encuesta y las razones de fondo, resaltamos el hecho de qué es una práctica muy común buscar códigos ya realizados en lugar de desarrollarlos.

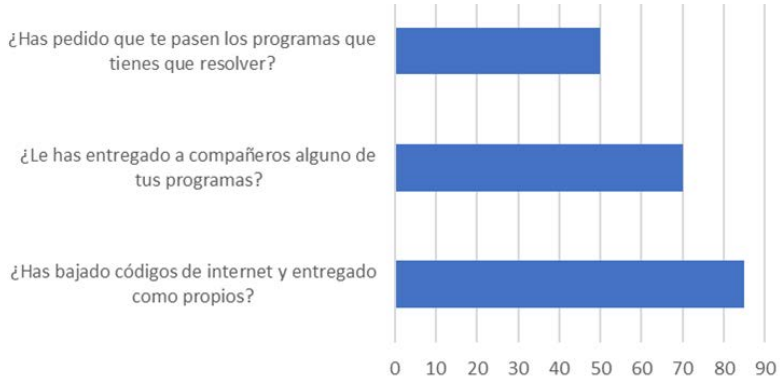


Figura 1 Encuesta sobre el nivel de traspaso de códigos entre estudiantes.

A nivel mundial hay cientos o miles de universidades que imparten licenciaturas referentes al desarrollo de programas de cómputo, si tomamos como referencia el sitio bachelorstudies.mx [KeyStone Bachelor Studies, 2022] en este sitio se mencionan 815 programas en ciencias computacionales en el mundo, al tomar como referencia los números a nivel mundial y si extrapolamos la encuesta realizada, es cuando se torna un tanto alarmante el nivel de plagio que se presenta. Debido a que el objetivo de las diversas asignaturas del área programación se enfoca en desarrollar esta habilidad, buscar o tener un mecanismo que permita a un profesor diferenciar el trabajo que presentan es una tarea compleja en muchas ocasiones muy ardua, y en ese sentido es importante tener algoritmos-programas capaces de medir el proceso de medición de las similitudes. La medición de la

eficiencia de un algoritmo o programa permite establecer su complejidad, la complejidad de un algoritmo puede ser espacial o temporal, es decir cuánto tiempo y memoria consume. Guerequeta y Vallecillo [2000] puntualizan que "El tiempo de ejecución de un algoritmo va a depender de diversos factores como son: los datos de entrada que le suministramos, la calidad del código generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo". En consecuencia, se establece que la complejidad de un algoritmo es la tasa de crecimiento del tiempo o espacio (memoria) que gasta un algoritmo en proporción a los datos que está procesando, normalmente a mayor cantidad de datos mayor cantidad de alguno de los dos recursos.

En este trabajo presentamos algunas plataformas y/o algoritmos que se han desarrollado como auxiliares en la búsqueda de orientar sobre el nivel plagio que hay en obras literarias.

2. Métodos

Se utilizó una metodología basada en 3 etapas, Revisión del nivel de plagió a nivel mundial en obras de texto, Revisión de los algoritmos comunes en la evaluación de textos duplicados (plagio) y por último la posible implementación en programas de cómputo. La primera etapa permitió tener un panorama de que organismos o personas están interesadas en este tema, encontrando que la mayoría a centrado su atención en el plagio de información científica como artículos, tesis y/o libros, encontrando plataformas especializadas en la revisión del plagio como Turnitin, Viper, PlagTracker entre muchas más, Timal y Sanchez (2017) hacen una recopilación de los aspectos legales, académicos y culturales del plagio en México y establecen los argumentos base de la importancia o impacto que está teniendo esta actividad. Para la etapa 2 se revisaron varias herramientas que pretenden ayudar u orientar sobre este tema, estableciendo cuál es la estrategia de búsqueda de la similitud del texto.

Para los algoritmos estadísticos el texto se debe analizar a partir de recuentos. En la mayoría de las aplicaciones estadísticas se encadenan cuatro grandes etapas

PROBLEMA -> DATOS -> TRATAMIENTO -> INTERPRETACIÓN [Rodríguez, Alessandrini, 2008]. Existe un debate acerca de si un programa se puede considerar una obra literaria, es decir tener en cuenta los derechos de autor como un libro o una pintura, en México el Instituto Nacional de Derechos de autor (INDAUTOR) tienen un área dedicada al registro de programas de cómputo quedando esto al margen las diversas opiniones al respecto, sobre todo de aquellos que establecen que no es una obra de arte un programa por que se hacen cientos o miles de soluciones de una misma problemática.

Algoritmos utilizados

Definitivamente establecer que una obra literaria es similar en algún grado a otra es una actividad que requiere establecer algún parámetro o indicador de comparación, es decir, ¿Cómo saber si es similar una obra a otra? Aunque la computación ha avanzado en muchas direcciones buscando hacer más fácil la vida los humanos, nada es automático o mágico, es decir, se debe procesar la información por algún medio y obtener una conclusión. En estos términos se han desarrollado varias ideas-algoritmos para medir esto.

Buscando frases en textos

La forma más elemental para establecer que dos textos tienen algún grado de similitud, es buscar que los textos sean los mismos o que tengan frases o párrafos repetidos, en el ámbito de la ingeniería en sistemas computacionales la mayoría de los lenguajes de programación contienen herramientas de búsquedas de textos dentro de otros textos, se les llama string y substring, esto derivado de que los lenguajes de programación están desarrollados en el idioma inglés; El proceso es muy simple, se toman dos textos y se busca uno dentro del otro, por ejemplo, el texto "*En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no hace mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.*" el cual forma parte del primer capítulo de la obra el Quijote de la Mancha de Miguel de Cervantes Saavedra, se puede utilizar para estos conceptos de string y subtring, buscando frases internas, en lenguaje java, por

ejemplo, si tenemos almacenado ese texto en la variable palabras = ”En un lugar .....”, con la instrucción palabras.indexOf(“rocin flaco y galgo”) se puede saber si el texto original contiene esa subfrase (substring). Como se mencionó esta es una forma simple de hacer el proceso, sin embargo, su simpleza conlleva un gran esfuerzo computacional, ya que se tendría que realizar una cantidad inmensa de búsquedas, se requieren una cantidad enorme de sub-búsquedas debido a que las personas pudieran cambiar, quitar y/o eliminar algunas palabras en diversos lugares tratando de hacer pasar el texto como algo nuevo por ejemplo ”EN UNA ZONA DE LA MANCHA..”. Ampliando el ejemplo del esfuerzo computacional tenemos la frase “El cielo es azul porque el mar es verde”, se tendrían que realizar búsquedas de:

- Toda la frase junta.
- Frases que comienzan con El Cielo, El cielo es azul, El cielo es azul porque, El cielo es azul por que el mar.
- Frases que no comiencen con EL, y así repetir el proceso con tantas frases como se puedan formar.

Como se podrá apreciar el esfuerzo de cómputo es enorme si consideramos que la mayoría de los trabajos revisados llegan a tener miles de palabras. Al final de todo este proceso se puede establecer cuantas frases coinciden con el texto original y con ello establecer un grado de similitud. Esta técnica es muy exacta pero extremadamente costosa, es decir, es un proceso con una alta complejidad temporal ya que por cada palabra y combinación de palabras se hacen búsquedas en, en otras palabras, se entra al terreno de las combinaciones y permutaciones, y para su determinación se emplean las fórmulas $C(n,r) = n!/((n-r)!\,r!)$ y $P(n,r) = n!/(n-r)$, respectivamente.

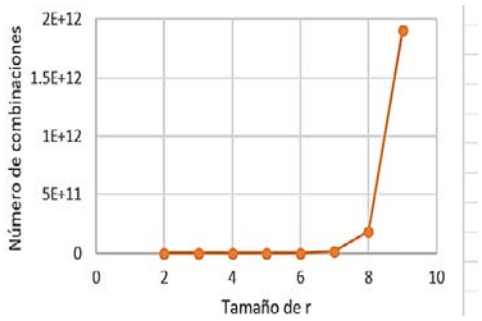
“Una combinación de orden r de un conjunto con n de elementos es un subconjunto con r elementos del conjunto. Así, por ejemplo, las combinaciones de orden 2 de las letras a, b y c son ab, ac, ad, bc, bd, y cb” [Blanco, 2010].

“Una permutación es un arreglo en un orden particular de objetos que forman un conjunto, Por ejemplo, las permutaciones de las letras a, b y c son son abc, acb, bac, bca, cab y cba es decir seis permutaciones” [Blanco, 2010]. Con el fin de

resaltar más el problema de alta complejidad temporal que se presenta en estos casos, pongamos como ejemplo que tenemos un texto con 5000 palabras, o sea $n = 5000$ y buscamos subfrases de veinte palabras $C(5000{,}20) = 5000!/(5000-20)! * 20!$ Es un dato no computable, pero si programable, el tiempo que se tardaría sería demasiado, en la tabla 1 se muestra el crecimiento de combinaciones, para esta tabla se consideró un segmento de 100 palabras (tokens) y se varió el tamaño del grupo, si consideramos que por cada combinación generado se gasta 1 milisegundo (hipotético) se puede calcular el tiempo que se tardaría una computadora en obtener lo que buscamos, existen varios algoritmos de este estilo como el Algoritmo de Knuth-Morris-Pratt, Algoritmo de Rabin-Karp y Algoritmo de Boyer-Moore.

Tabla 1 Crecimiento exponencial de combinaciones en base a una $n = 100$ y variando el tamaño del r (tamaño de grupo o token).

r, token o agrupación	$C(n,r)$
2	4950
3	161700
4	3921225
5	75287520
6	1192052400
7	1.6008E+10
8	1.8609E+11
9	1.9022E+12



Huella de texto o conteo estadístico

Otra estrategia que no se basa en la similitud de las sub frases contenidas sino más bien en la forma del uso de las palabras a través de su repetición, es una forma muy simple que permite crear un patrón de escritura de cada persona, esta técnica se basa en convertir el texto en un conteo de palabras, traduciendo el texto en un contador para llevar registro de las veces que el autor usa una palabra, en esta técnica se sugiere evitar o no considerar las preposiciones y los artículos debido a que son los elementos más utilizados al momento de redactar un texto. Con esta técnica el texto de Don Quijote Mencionado antes se traduciría en Lugar(1), Mancha(1), cuyo(1), Nombre(1), No (2), quiero(1), acordarme(1), hace(1),

mucho(1), tiempo(1), vivía(1), Hidalgo(1), Lanza(1), astillero(1), adarga(1) antigua(1), rocín(1), flaco(1), galgo(1).

El resultado generado es una huella de escritura o conteo estadístico, al tener esta huella se puede comparar con otras huellas de otros autores, y lograr establecer una similitud.

Rodríguez y Alessandrini [2008] procesaron 1400 encuestas que incluían respuestas abiertas, en la figura 1 se muestra una parte de su conteo, el conteo las columnas eran las personas encuestadas y los renglones la palabra y el número de veces utilizada.

		a01a	a02a	a03a	a04a	a05a	a06a
a	i	322.	2.	0.	10.	12.	76.
actitud	i	6.	0.	0.	0.	0.	7.
activa	i	9.	0.	0.	0.	2.	1.
actividades	i	6.	0.	0.	0.	0.	5.
adelante	i	19.	0.	0.	0.	0.	3.
además	i	8.	0.	0.	1.	0.	1.
adolescencia	i	3.	0.	0.	0.	0.	7.

Figura 1 Parte del conteo de palabras en las encuestas realizadas por Rodríguez y Alessandrini.

Esta técnica presenta una baja complejidad temporal y espacial ya que básicamente es un conteo de los términos (palabras o tokens), el crecimiento de los recursos de tiempo y espacio son muy constantes, realizamos una pequeña prueba de esta técnica con dos textos de 105,506 palabras y 240,618 palabras con el código siguiente:

```
import java.io.File; import java.util.ArrayList; import java.util.Arrays;
import java.util.Collections; import java.util.Enumeration; import
java.util.Hashtable;
import java.util.List; import java.util.Scanner;
public class Complejidad {
  File doc;  Scanner archivo;
 Hashtable<String, Integer> contenedor;
   public static void main(String[] args) {
```

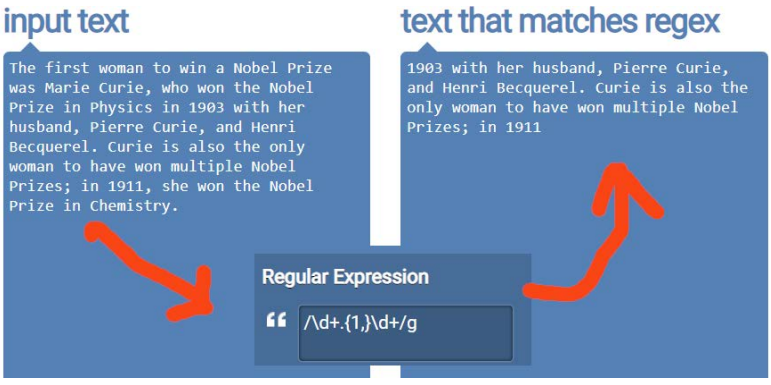
```
    Complejidad Objeto=new Complejidad();
    Objeto.doc = new File("archivo.txt");
   Objeto.contenedor= new Hashtable<String, Integer>();
    try {   Objeto.archivo = new Scanner(Objeto.doc); } catch(Exception E){};
       Objeto.procesa();
   }
void procesa (){ while (archivo.hasNextLine())  segmenta(archivo.nextLine());
muestra();    }
void segmenta(String renglon){
  String palabras[]=renglon.split(" ");
String     noValidas[]={";",",",     "al","del","lo",     "no","si","que","un","se","por"
,"en","el","la", "los" ,"las","de","para","con","a","y","es","desde","contra"};
  int valor;
List<String> prepo = new ArrayList<>(Arrays.asList(noValidas));
   for(String palabra : palabras){
   if   (!palabra.equals("")   &&   !prepo.contains(palabra.toLowerCase())   &&
palabra.length()>2 )
     if (contenedor.get(palabra)==null) contenedor.put(palabra,1);
     else { valor=contenedor.get(palabra)+1;
     contenedor.put(palabra,valor);} }}
void muestra(){
Enumeration<String> keys = contenedor.keys();
String palabra,ceros; int conteo, control;
List <String> tokens=new  ArrayList<String>();
while(keys.hasMoreElements()){
   palabra=keys.nextElement();
   control=conteo=contenedor.get(palabra);
   ceros="";
   while (control<10000)     {  ceros+="0"; control*=10; }
   tokens.add(ceros+conteo+"=>"+palabra);}
Collections.sort(tokens, Collections.reverseOrder());conteo=0;
```

for(String token:tokens) { System.out.println(token); conteo++; if (conteo>50) break; }}}

El código se programó en java, el tiempo de obtención de la huella fue de 0 segundos, claro que esto depende de las características del procesador y memoria de la computadora. El siguiente paso en términos de contemplar la identificación del plagio es establecer una metodología para comparar las huellas.

Autómatas determinísticos

Una de las formas más recientes para buscar similitudes entre textos es el uso de autómatas finitos determinísticos (AFD) y/o expresiones regulares (regexp), la estrategia consiste en convertir el texto a un AFD o una regexp, con ello se logra una huella semejante al algoritmo 2.2, pero más corto en términos del resultado obtenido. Se genera una tabla con ocurrencias y el algoritmo genera una regexp. En la figura 2 se muestra un texto base, una regexp y el resultado que arroja la REGEXP al alimentarla con el texto base. Esta técnica puede ser muy exacta en cuanto a encontrar las similitudes, los problemas que presentan son como obtener el REGEXP así como las variantes que puede tener, porque como se pudo observar en la figura 2, solo se identificó una parte del texto.



Fuente: elaboración propia.

Figura 2 Texto a REGEXP y el resultado de procesar un texto con ella.

Recientemente los sistemas anti-plagio están utilizando las teorías de autómatas para convertir un texto en su correspondiente autómata con su gramática y el

resultado es más simple de evaluar con respecto de otros textos, ya que el autómata puede medir cuántos tokens del nuevo texto corresponden con el texto original. Y a partir de esto encontrar los fragmentos plagiados.

3. Resultados

En este trabajo se revisó y presentó el funcionamiento de 3 algoritmos comunes en cuanto a la búsqueda de similitudes o textos, estas técnicas son combinadas en diversas plataformas anti-plagio, y como se mencionó en la sección anterior, esto puede requerir de una alta capacidad de cómputo de la plataforma. Turnitin, por poner un ejemplo, el análisis de los algoritmos se centró en los textos como los libros y artículos de divulgación científica. Pero, ¿qué sucede en el campo de la programación?, los textos (códigos) no son tan extensos, es decir, si recordamos la intención de este artículo es orientar este tipo de algoritmos hacia la programación y las asignaturas universitarias cuya proposito es que el alumno desarrolle habilidades en el campo de la programación; Desde la perspectiva que los códigos presentados por los alumnos son pequeños (50 líneas/130 palabras en promedio a problemas simples o triviales) tomemos como ejemplo el código 1 de este artículo que contiene 144 palabras (tokens) hacer un análisis no es costoso para una computadora, es decir, implementar alguno de los esquemas en la comparativa con los 30 o 35 alumnos inscritos normalmente en un grupo universitario, pero la misma tarea si es pesada para un humano promedio (maestro) y su apreciación del grado de similitud sería subjetivo, mientras que para un programa de cómputo sería bastante objetivo en términos de obtener datos fríos como los números arrojados de la comparativa.

4. Discusión

Buscar que los estudiantes desarrollen las habilidades del desarrollo de soluciones a problemas computables es la tarea fundamental de las asignaturas primarias en las licenciaturas de las ciencias computacionales y definitivamente es viable la aplicación de este tipo de estrategia para medir la similitud de programas entregados por alumnos sobre todo en las materias base de las licenciaturas en el

área de las ciencias computacionales, esto podría brindar una herramienta rápida y precisa del nivel de plagió que se presenta en las universidades. Pero el impacto no debe ser en el corto plazo, sino a largo plazo, es decir, desarrollar en alguna de las plataformas educativas encaminadas a la programación como los jueces en línea, las cuales son plataformas que permiten practicar la programación resolviendo diversos tipos de problemas, en estas plataformas la intención no es enseñar a programar sino la demostración de los conocimientos en el área, ejemplo de este tipo de plataformas son OMEGA UP, UVA Online Judge, URI online judge entre muchas que existen en internet, estas plataformas solo te dicen si el programa presentado cumple con lo solicitado, serian una gran herramienta académica si aparte de definir que el programa presentado por un alumno cumple con requisitos de funcionamiento y por otra parte pudiera establecer el nivel de similitud con respecto de otros programadores.

Una característica alternativa para la detección de plagio corresponde a la presencia de indicios de formato y otros aspectos no asociados con el código. Por ejemplo, la presencia de faltas de ortografía idénticas en la documentación interna del código (comentarios), semejanzas notables en los nombres de identificadores empleados o incluso, la cantidad de líneas en blanco entre segmentos de código puede servir de indicio de que dichos códigos han sido compartidos.

5. Conclusiones

Dewi, Arief y Kuspriyantoa [2022] indican que “La práctica del plagio ya no es algo extraño, sobre todo entre los estudiantes que trabajan casi todos los días en tareas asignadas por el profesor” y establecen dos grandes categorías para el análisis, la parte léxica como los algoritmos de conteo de palabras -tokens u ocurrencias o análisis estructural como el proceso de convertir textos en expresiones regulares. Implementar algún tipo de algoritmo para comparar programas no es nuevo desde 1994 la universidad de Berkeley ha estado trabajando en una herramienta para esto, conocido como MOSS (Measure of Software Similarity) de hecho tiene una plataforma bajo contrato para su uso, sin embargo es estar alimentando de códigos y esperar la respuesta, básicamente

todas estas herramientas tratan de obtener el ADN de un código, pero comparar los M programas por materia de los N grupos de un profesor y guardar el histórico para futuros grupos es algo ideal como estrategia para desanimar la compartición de códigos y fomentar que los alumnos desarrollen su propia solución, por otra parte puede beneficiar en el plagio no consensuado en programas de cómputo académicos, lamentablemente lograr implementar algún algoritmo sobre una plataforma comercial o personal es una tarea no simple.

6. Bibliografía y Referencias

[1] Astudillo, F. (2006). El Plagio Intelectual: https://www.redalyc.org/pdf/1890/189018586009.pdf.

[2] Blanco, L. (2010). Probabilidad. Editorial Universidad Nacional de Colombia.

[3] Dewi, T, Arief, S & Kuspriyantoa (2011). Plagiarism Detection System Design for Programming Assignment in Virtual Classroom Based on Moodle, Elsevier.

[4] Guerequeta, R., Vallecillo, A. (2000). Técnicas de Diseño de Algoritmos. ISBN: 84-7496-666-3 Universidad de Málaga.

[5] KeyStone Bachelor Studies (2022). 814 Programas de Grado en Ciencias de la Computación 2022: https://www.bachelorstudies.mx/Grado-Licenciatura/Ciencias-de-la-Computaci%C3%B3n/.

[6] Rodríguez, S., Alessandrini, D. (2008). Aplicación de técnicas de Análisis Estadístico de Textos a una encuesta con preguntas abiertas: comparación de los resultados obtenidos con post-codificación: http://www.iesta.edu.uy/wp-content/uploads/2015/05/Pasantia_Rodriguez_Alessandrini.pdf

[7] Timal, S., Sánchez F. . (2017). http://www.scielo.org.mx/pdf/tla/v11n42/1870-6916-tla-11-42-00048.pdf.