# ESTRATEGIA PARA MEDIR SIMILITUD ENTRE CÓDIGOS EN CUALQUIER LENGUAJE DE PROGRAMACIÓN

## *STRATEGY FOR MEASURING SIMILARITY BETWEEN CODES IN ANY PROGRAMMING LANGUAGE*

***Francisco Gutiérrez Vera***
Tecnológico Nacional de México / IT de Celaya, México
*Francisco.gutierrez@itcelaya.edu.mx*

***Claudia Cristina Ortega González***
Tecnológico Nacional de México / IT de Celaya, México
*Francisco.gutierrez@itcelaya.edu.mx*

**Recepción:** 21/noviembre/2024 **Aceptación:** 20/diciembre/2024

## Resumen

En este artículo se aborda como se desarrolló una estrategia y su implementación para medir la similitud entre códigos en algún lenguaje de programación, con la finalidad de reducir el plagio de códigos en asignaturas de programación. Se implemento la estrategia en dos plataformas web, la primera para hacer pruebas globales de comparación entre 2 códigos que solucionan el mismo problema, una vez valorado el rendimiento se aplicó la estrategia en una plataforma académica en la que todo un grupo de estudiantes entregan como propios códigos. Desde la perspectiva de los autores, el plagio consensuado y no consensuado de programas que intentan medir el avance académico de un estudiante se ha disparado enormemente ya sea porque los alumnos no saben por qué están estudiando una carrera o porque no tienen claras sus decisiones de vida, o alguna otra razón, todo esto está llevando a que las capacidades de desarrollo de software ha caído y ahora un porcentaje importante de alumnos todo lo obtienen de las plataformas de inteligencia Artificial.

**Palabras Clave:** Estrategia, Medir Similitud, código, Lenguaje de programación.

## *Abstract*

*This paper discusses how a strategy was developed and implemented to measure code similarity in a programming language to measure the similarity between codes*

*in a programming language, with the purpose of reducing code to reduce code plagiarism in programming subjects. We The strategy was implemented in two web platforms, the first one to make global comparison tests between 2 codes that solve the same problem, once the performance was evaluated, the strategy was applied in an academic platform in which a group of students deliver their own codes. From the authors' perspective, consensual and non-consensual plagiarism of programs that attempt to measure a student's academic progress has skyrocketed, either because students do not know why they are studying a career or because they are not clear about their life decisions, or some other reason. Life decisions, or some other reason, all of this is leading to the fact that the software development software development skills have dropped and now a significant percentage of students get everything from the students now get everything from Artificial Intelligence platforms.*
***Keywords*****:** *Strategy, Measuring Similarity, code, Programming language.*



## 1. Introducción

Como parte de las actividades de enseñanza y aprendizaje que permitan medir el desarrollo de las habilidades de programación en estudiantes de diferentes niveles se encuentra el dejar tareas, prácticas o conjuntos de problemas, enfocándonos en ésta última, con lleva a varias situaciones, preparar la actividad, revisar que todos los programas entregados funcionen y una de las más importantes, ¿Cómo saber que ellos resolvieron el problema?, en otras palabras, ¿Cómo establecer que no plagiaron el código de una Inteligencia Artificial (IA) o algún compañero se los pasó?.

Seo et al [2011] establecen el concepto general de plagio como actividades de reproducir el trabajo de otra persona sin reconocer ni mencionar la fuente.

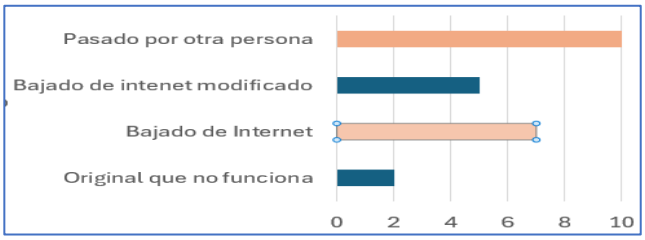
La empresa Turnitin [2024] líder mundial en análisis de plagio establece en su artículo ¿Qué es el plagio de código fuente y por qué se da cada vez más? que el plagio de código fuente es utilizar el código fuente de otra persona y adjudicárselo como propio. En la actualidad esto abarca a las IA, es decir, consultar algo a una IA y presentarlo como propio. La empresa Turnitin [2024] dispone en una de sus plataformas 5 tipos de plagio derivados de una encuesta que realizaron:

- Clonación. Cuando el documento es idéntico a otro.
- Mosaico. Cuando el documento está conformado por partes de diferentes fuentes con buena coherencia que parece inédito.
- Copiado y pegado.
- Remix. Parafrasear textos de otros autores.
- Búsqueda y reemplazar. Consiste en cambiar palabras por sinónimos, así como expresiones importantes del texto y darle el mismo sentido, pero con un lenguaje diferente.

En esta parte se detonan varios problemas, por una parte, la carga excesiva para revisar que los programas entregados cumplan con lo solicitado y por otra, la originalidad de los programas. Desde la década de los 90 del siglo pasado se han desarrollado programas que intentan verificar la autoría de obras o su originalidad, uno de los programas o algoritmos más conocidos es MOSS (Measure of Software Similarity), su funcionamiento básicamente es convertir el código fuente en tokens, pedazos representativos del código, se buscan cuáles son los rasgos más representativos de un programa y se comparan estos pedazos con los generados por códigos de otras personas. Existe una variedad importante de algoritmos para medir similitud, algunos se describen en Gutiérrez et al [2022] y Tresnawatia et al [2011], además de muchos sitios de internet que te permiten medir el nivel de plagio, los más simples sólo requieren que te registres y tienes acceso a una limitada cantidad de envíos, otros son de paga.

Retomando la parte de la problemática, una cosa es que el código funcione y otra quien es el autor original, en este segundo aspecto desde la perspectiva del profesor puede interesarle si es original, si es bajado de internet o si otro humano se lo paso. Se hizo una encuesta para ver el grado de impacto negativo de estos tres factores, la pregunta fue ¿Qué forma de obtener un programa es la más perjudicial desde su punto de vista, para el desarrollo de competencias en programación? El resultado se muestra en la Figura 1. Se puede observar notoriamente que el hecho de que una persona pase o entregue un código es considerado como lo peor, sin embargo, los estudiantes en su afán de conseguir objetivos modifican variables, cambian

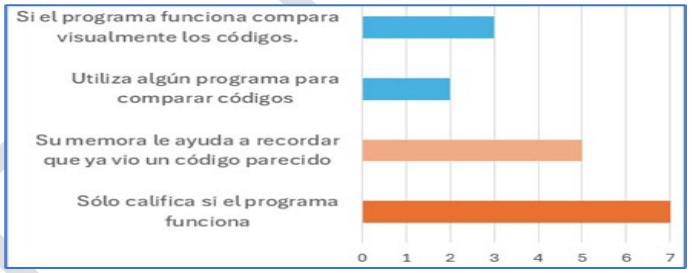
nombres, agregan comentarios y logran en algunas ocasiones engañar al profesor sobre la originalidad de lo que entregan.




*Fuente: elaboración propia.*

Figura 1 Valoración de las entregas de código.

Independientemente de que el plagio no se puede evitar debido a la naturaleza humana, se realizó otra encuesta para determinar cómo le hacen los profesores para medir el nivel de plagio, en esta encuesta se les permitió seleccionar 1 o dos respuestas, el resultado se puede ver en la Figura 2, donde la mayoría solo califica si funciona, no le da importancia al plagió, y pocos usan herramientas automatizadas para medirlo.




*Fuente: elaboración propia.*

Figura 2 Formas que utilizan los profesores para medir el plagio.

Dentro de las posibles razones para no darle importancia al plagio, los profesores dieron diversas opiniones, entre ellas el cambio generacional y que las herramientas modernas no son tan accesibles para el quehacer académico, pero esto sería tema de otra investigación. De los profesores que usan alguna herramienta para medir la similitud, expusieron que medir la similitud, tiende a ser de repente subjetiva, ya que, aunque dos personas piensan diferente, todos en programación utilizamos decisiones, ciclos, variables, clases y objetos en algunos lenguajes, y por lo tanto

siempre existirá algún grado de similitud. Basados en un grupo de estudiantes la pregunta a responder será ¿Cómo generar una herramienta-metodología que permita medir la similitud de códigos generados por un grupo de estudiantes de la asignatura X, sin importar si es bajada de internet o no? Aquí se presentan los resultados de la creación de una estrategia para medir las similitudes de código, con retroalimentación para alumnos y maestros que permita medir el posible plagio.

## 2. Métodos

### Determinación de los elementos de un lenguaje de programación

Cada lenguaje de programación contiene reglas léxicas, sintácticas y semánticas, las primeras establecen el conjunto de símbolos existentes en el lenguaje, por ejemplo, la frase en español, “El perro comió croquetas”, es una frase lexicográficamente correcta debido a que todos los elementos (tokens) son reconocidos en el idioma español, pero la frase “El prro comió croquetas”, tiene un error lexicográfico por que el token prro, no existe o nos falta contexto. La parte sintáctica tiene que ver con las reglas de cómo se escribe algo, por ejemplo, en español la siguiente frase está escrita correctamente “El perro comió croquetas”, debido a que se cumplen todas las reglas (debe empezar con mayúscula, terminar con punto, y su estructura es sujeto, verbo y predicado). Las reglas sintácticas son las más abundantes en un lenguaje y por último y no menos importante están las reglas semánticas, son las que dan sentido a las cosas, la frase “El perro voló la cometa”, cumple lexicográfica y sintácticamente, pero semánticamente tiene un problema en un sentido lógico, un perro no puede volar algo.

Traducido todo esto a un lenguaje de programación es una tarea enorme revisar que el programa cumple con todo, pero esto lo hace un compilador o interprete del lenguaje, todas estas reglas se están afectando en la actualidad por muchas contracciones que las personas hacemos al comunicarnos por ejemplo en México es común utilizar las contracciones “nomas” y “paque” entre muchas, los lenguajes de programación también incluyen en sus reglas léxicas o sintácticas contracciones por ejemplo en lenguajes derivados de C, dato=dato+1, se pude contraer así dato++. Debido a que cada lenguaje de programación tiene sus propias reglas es

algo complicado crear algoritmos genéricos que abarque todos los lenguajes, sin embargo, hay que considerar solo 2 aspectos:

- El compilador revisa todas las reglas, y como consecuencia, no interesa revisar si semánticamente es correcto o no el código y mucho menos sintácticamente.
- El aspecto léxico es el que comparten todos los lenguajes de programación, no importa si es Python, Java, PHP u otro, todos ellos tienen elementos léxicos para tomar decisiones, repeticiones, bloques de instrucciones, entre otros más.

Por tanto, crear un algoritmo que mida similitudes globales y no estructurales (como arma la solución), es una decisión razonablemente buena, además ya se mencionó que en programación todos usamos decisiones, repeticiones y variables.

**Creación del conjunto de elementos léxicos**

Generar un conjunto de elementos léxicos para todos o un grupo de lenguajes de programación llevo a analizar las reglas léxicas de algunos de ellos, se tomó como base a JAVA, PHP, PYTHON y javascript, en base a éstos lenguajes se creó un listado de los elementos representativos de cada uno de ellos, los cuales fueron paréntesis, líneas, try, variables, variint, varifloat, varilong, varichar, varidouble, variString, variboolean, varistring, variScanner, funciones, ciclFor, ciclWhile, decisiones, sino, sinoSi, bloques, seleMultiple',"foreach, case, class, variByte, varibyte, return, paraInt, paraFloat, paraDouble, paraChar, and, or, not, input, def, elif, :, match. Este conjunto permite caracterizar estos lenguajes de programación, algunas de estos elementos léxicos (características o factores) son compartidos por todos, esto facilita el análisis y la creación de una herramienta única, sin embargo, hay elementos que son muy particulares de cada uno y esto complica o aumenta los factores a medir, de tal forma que existieron ventajas y desventajas al armar el conjunto único de elementos léxico.

**Creación de la estrategia para medir la similitud**

Se revisaron algunas formas conocidas, pero se decidió por “Huella de texto o conteo estadístico”, descrita en Gutiérrez et al [2022], en donde se busca establecer

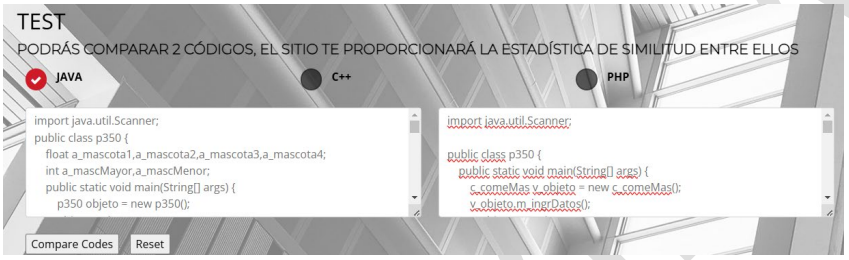
por conteo el número de veces que se utiliza un elementos léxico, dejando de la lado la forma o estructura del código, en esta estrategia, se eliminan todos los elementos no útiles y se concentra en el rasgos más prominentes o que resaltan más, el objetivo es establecer un nivel de similitud entre códigos de programación, un objetivo alternativo o que se puede lograr es reducir el que los alumnos de un mismo grupo entreguen los mismos programas, al aplicar está estrategia a un código se obtiene una huella que resalta los rasgos más prominentes. Para encontrar la similitud estadística se deben obtener las huellas de cada código y luego compararlas, se decidió crear un esquema de similitud basado en porcentajes, cada rasgo de la huella aportará un porcentaje de similitud, de tal forma que, si los códigos uno utiliza 9 factores y el otro 10, se toma el que tiene más factores y se divide entre 100 el número de factores. La estrategia que se define es crea o adaptar una plataforma de entrega de tareas-códigos, de tal forma que cada que un estudiante entrega un código, se obtenga su huella y se comparé con las demás huellas almacenadas en una base de datos.

**Implementación de la estrategia**

Se construyo una interfaz web para pruebas, en la Figura 3 se muestra cómo se ingresan los códigos, se escoge el lenguaje y se lanza la comparación, se programó en PHP el proceso de creación de las huellas y la comparativa de las huellas.

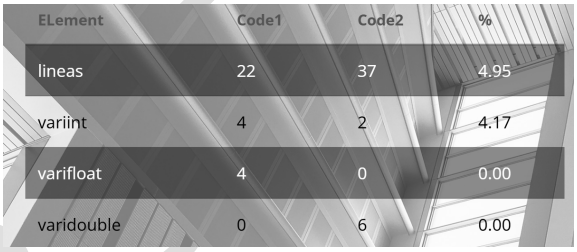
En la Figura 4 se muestran 4 factores de una prueba a dos códigos que resolvían el mismo problema, se muestra el número de líneas de ambos códigos, variables enteras, variables flotantes y variables doubles, en las flotantes y doubles se presenta el caso de que los códigos son diferentes ya que uno si maneja ese elemento y el otro no, con lo cual esos dos factores no son similares, en los otros dos factores mostrados hay una cantidad elementos que dan una similitud del $4.95$ y $4.17$ respectivamente, el $4.95$ se obtiene de la variación que existe entre los códigos, es decir, líneas 22 y 37, dividimos el dato pequeño entre el grande y obtenemos una proporción de 0.59 que ya es un grado de similitud, para obtener el $4.95$, se multiplica la proporción obtenida en el factor por el resultado de dividir 100 entre el número de factores a evaluar, en la prueba de la Figura 3 fueron 12 los

factores encontrados. Por lo tanto, cada factor podría agregar una similitud del $8.33\%$ $(0.59X8.33 = 4.95)$. La Tabla 1 muestra el detalle de la prueba con los 12 factores medidos, en donde la prueba arrojo un 47.9% de similitud, cada factor se evalúa como se indicó previamente, sumando todos los porcentajes se obtiene el valor indicado. Una vez concluida la etapa de prueba y revisión de resultados se procedió a cargar la estrategia en una plataforma educativa, en donde la plataforma les informaba a los estudiantes el grado de similitud de sus programas entregados.




*Fuente: elaboración propia.*

Figura 3 Interfaz para probar similitud.




*Fuente: elaboración propia.*

Figura 4 Prueba de similitud de 2 códigos.

Tabla 1 Porcentaje de similitud entre dos códigos, en cada factor.

| Element | Code1 | Code2 | % |
|---|---|---|---|
| lineas | 22 | 37 | 4.95 |
| variint | 4 | 2 | 4.17 |
| varifloat | 4 | 0 | 0 |
| varidouble | 0 | 6 | 0 |
| variString | 1 | 1 | 8.33 |
| variScanner | 2 | 3 | 5.56 |
| funciones | 5 | 5 | 8.33 |
| decisiones | 6 | 6 | 8.33 |
| sino | 6 | 0 | 0 |
| bloques | 6 | 13 | 3.85 |
| class | 1 | 2 | 4.17 |
| return | 8 | 0 | 0 |

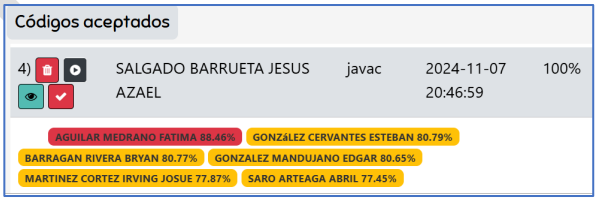
*Fuente: elaboración propia.*

## 3. Resultados

La estrategia para obtener el grado de similitud funciona bastante bien, al generar la huella del código tomando en cuenta los factores/rasgos más prominentes del conjunto de elementos léxicos se pudo crear un mecanismo de comparación entre todos los programas que entregan los estudiantes. En pruebas en donde se introdujo el mismo código y solo se cambiaban nombres de variables o funciones propias, la similitud se seguía reflejando, es decir, se establecía un alto rango de similitud. Se agregaron otras funcionalidades al algoritmo como fue eliminar los comentarios o líneas comentadas, ya que el alumno puede dejar ahí elementos léxicos que afecten la comparación y también se eliminaron las líneas vacías, es decir, líneas que no contienen código, esto permitió una medición más precisa.

En la Figura 5 se muestra la implementación final sobre la plataforma ConAcad [2024], en esta plataforma los estudiantes cargan soluciones a problemas planteados, la plataforma valida que el programa entregado funcione y de resultados, al mismo tiempo genera la huella del código y lo compara con los programas entregados anteriormente.

En la Figura 5 se puede observar que esa persona entrego su código, el cual funciono al 100%, pero tiene similitudes estadísticas con otros 5 códigos entregados, también el profesor puede ver el grado de similitud de acuerdo con la estrategia generada. En la interfaz del estudiante no le indica con quien tiene similitud, solo le indica que es similar que otros, indicando solo el grado de similitud.




*Fuente: elaboración propia.*

Figura 5 Interfaz que muestra el grado de similitud.

## 4. Discusión

Si bien es cuestionable el mecanismo de medir similitud debido a que no se toma en cuenta la forma estructural del código, es decir, las personas cuando redactamos

o creamos alguna obra literaria tenemos una forma o estilo de escribir ya sea porque recurrimos a ciertas “muletillas” o usamos muchos conectores de oraciones, al final del día dejamos algo de nuestra personalidad en una obra literaria, un programa de cómputo podría caer en esta categoría de acuerdo con el portal “Justia México” [2024], se decidió que el estilo era un factor no considerado, básicamente por que requeriría de procesar otros factores. Tampoco se está tomando en cuenta que el alumno entregue un código hecho por alguna IA. Normalmente los códigos hechos por IA tienen 2 características, funcionan bien y por otra parte utilizan muchas herramientas(librerías) que en la mayoría de los casos los profesores no enseñan en clases, pero esto tendría que analizarse más a profundidad.

El realizar un conteo estadístico como estrategia se podría aplicar en la mayoría de las plataformas que permiten cargar códigos o tareas, claro con la ayuda de algún desarrollador que incorpore la programación necesaria o los creadores de la plataforma que se vaya a utilizar, pero como modificar plataformas con lleva un proceso de ingeniería, se está optando por herramientas más “Fáciles” como las mismas IA, una de las herramientas muy utilizadas para que las personas practiquen programación son los jueces en línea, como OMEGAUP, BECROW, UVA, CODEWARS, como por ejemplo los jueces en línea, uno de los problemas de IA y jueces en línea es que su objetivo no es medir similitud, en el caso de las IA se debe entrenar una para que aprenda a medir similitud de un conjunto de programas, y los jueces en línea su objetivo es que las personas solucionen el problema cargando el código sin importar de donde proviene, como se mencionó al principio de este artículo, el problema académico es ¿Cómo mides que el alumno este desarrollando las competencias y habilidades necesarias?.

## 5. Conclusiones

Al incorporar esta estrategia en una plataforma se redujo notoriamente la entrega de códigos, esto se nota en la Figura 6 en donde se puede observar la cantidad de alumnos que están intentado resolver problemas, esa imagen corresponde a 22 alumnos de un grupo de Fundamentos de programación. Se pudo notar que los alumnos se sienten observados, ya que se preguntan cómo es que se mide la

comparación ente los códigos y por lo mismo no intentan hacer trampa, ya que la plataforma informará al profesor. Sin esta estrategia seguramente los 22 estarían entregando programas copiados, o alterados, por lo que podemos considerar que la estrategia tuvo éxito al lograr medir la similitud y alternativamente reducir el plagio.

Estadisticas Problemario

| | | Alumnos | |
|---|---|---|---|
| | Problema | 100% | Intentando |
| 1 | ¿Quien come más? | 2 | 2 |
| 2 | Área del polígono | 6 | 1 |
| 3 | Áreas Interesantes | 1 | 0 |
| 4 | De primaria | 8 | 4 |
| 5 | Dime quien soy | 13 | 3 |
| 6 | Don Chon y las estatuas | 1 | 1 |
| 7 | El código ASCII | 1 | 0 |

*Fuente: elaboración propia.*

Figura 6 Listado de alumnos intentado resolver problemas.

El efecto negativo o colateral es que al sentirse observados los alumnos dejan de intentar resolver, aunque sea preguntado cómo le hicieron los que si están resolviendo problemas. Incorporar estrategias simples que ayuden a medir el nivel de plagio viéndolo desde el aspecto positivo permite promover el desarrollo del pensamiento abstracto en los estudiantes.

## 6. Bibliografía y Referencias

[1] ConAcad, https://tigger.celeya.tecnm.mx/conacad, 2024

[2] Gutiérrez, F., Ortega, C., Fierro, G., Asato, J., Algoritmos Para Medir el Nivel de Plagio y su Posible Implementación en la Comparación de Códigos de Programación. https://pistaseducativas.celaya.tecnm.mx/index.php/pistas/article/view/2830/2175, 2022.

[3] Justia.mexico, Derechos de Autor de Programas de Computación, Programas Informáticos o Software, https://mexico.justia.com/derecho-de-la-propiedad-intelectual/derechos-de-autor-de-programas-de-computacion-programas-informaticos/, 2024.

[4] Seo, N., Sangwoo, K., Cheonyoung, J., A Lightweight Program Similarity Detection Model using XML and Levenshtein Distance. https://www.researchgate.net/publication/220843982_A_Lightweight_Program_Similarity_Detection_Model_using_XML_and_Levenshtein_Distance, 2011.

[5] Tresnawatia, D., Syaichu, A., Kuspriyantoa, R., Plagiarism Detection System Design for Programming Assignment in Virtual Classroom Based on Moodle, 2011.

[6] Turnitin, ¿Qué es el plagio de código fuente y por qué se da cada vez más?, https://latam.turnitin.com/blog/que-es-el-plagio-de-codigo-fuente-y-por-que-se-da-cada-vez-mas, 2024.