UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMATICA

TESIS DE GRADO

DISEÑO DE UN ALGORITMO PARA LA DETECCIÓN DE SIMILITUD ENTRE CÓDIGOS FUENTE

Para optar por el Título de Licenciatura en Informática

Mención: Ciencias de la Computación

Postulante: Univ. Edson Eddy Lecoña Zarate

Tutor: M.Sc. Jorge Humberto Terán Pomier

La Paz - Bolivia

2022

Capítulo 1

Marco Referencial

1.1. Introducción

[Cheers *et al.*, 2021] Explica que la identificación de similitud entre códigos fuente puede servir para varios propósitos, entre ellos están el estudio de la evolución de código fuente de un proyecto, detección de prácticas de plagio, detección de prácticas de re utilización, extracción de código para "re factorización" del mismo y seguimiento de defectos para su corrección.

Los estudiantes durante su proceso de formación elaboran trabajos, proyectos, tareas y ejercicios de programación escritos en un lenguaje de programación, estas actividades se realiza de forma individual o grupal, cuando estas actividades se desarrollan de forma individual los estudiantes deben ser conscientes de que todo trabajo entregado debe ser de su autoría, pues tiene como función medir la capacidad de resolución de problemas y el enfoque lógico y otros. Por ello encontrar similitud en trabajos de programación presentados por los estudiantes, puede ser identificado como plagio.

Una de las formas para detectar el plagio en los trabajos de programación consiste en realizar la comparación entre los trabajos entregados por los estudiantes de la materia. De a modo de obtener una lista de estudiantes que tienen trabajos similares. Realizar la comparación de los trabajos manualmente puede llegar a ser un trabajo moroso, por lo cual contar con una herramienta de software que realice las comparaciones de trabajos de forma automática es de gran utilidad.

En la actualidad existen diferentes herramientas de software que aplican métodos para detección de similitud entre códigos fuente, a partir de las características se pudo evidenciar que estas presentan deficiencias como ser la obsolescencia, sistemas cerrados (sin código abierto), un proceso complejo de evaluación de similitud, la incapacidad para utilizar o no una gran base de información. Esto se debe a que fueron diseñadas para detectar plagio entre un grupo pequeño de archivos de código fuente.

Por lo cual contar con un sistema para la detección de similitud entre códigos fuente llega a ser útil para detectar plagio en trabajos de cátedra presentados por estudiantes. El presente trabajo de tesis se centra en el diseño e implementación de un algoritmo para la detección de similitud entre códigos fuente que tenga un buen desempeño, en términos de tiempo de ejecución, espacio de memoria ocupado y precisión.

1.2. Problema

1.2.1. Antecedentes del problema

Se analizaron herramientas para la detección de similitud entre código fuente más populares, a continuación se dará breve explicación de las características que tienen estas.

Sherlock

Es una herramienta de código abierto, que trabaja con código escrito en los lenguajes Java, C y texto natural. Esta herramienta no cuenta con una interfaz gráfica. Fue desarrollada por la Universidad de Sydney. Los resultados arrojados se basan en un porcentaje que se corresponde con las similitudes encontradas. El porcentaje 0 % significa que no hay similitudes, y 100 % significa que hay muchas posibilidades de que tengan partes iguales. Al no utilizar el documento en su totalidad, no se puede afirmar que sean completamente iguales. Sólo trabaja con archivos locales, no busca similitudes en Internet [Díaz *et al.*, 2007].

Simian

Esta herramienta no es de código abierto. Identifica la duplicación de códigos escritos en JAVA, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML, Visual Basic y texto natural. Fue desarrollada por una consultora de Australia llamada REDHILL. No cuenta con una interfaz gráfica, por lo tanto, el ingreso de los parámetros es a través de línea de comando. Sólo trabaja con archivos locales, no busca similitudes en Internet [Díaz *et al.*, 2007].

Jplag

Esta herramienta no es de código abierto, sólo permite la programación del cliente, pero el servidor es el que realiza las operaciones de comparación; se encuentra en la Universidad de Karlsruhe de Alemania, en la cual fue desarrollada. Trabaja con los siguientes lenguajes C, C++, Java, Scheme y texto natural. Fue desarrollada por la Universidad de Karlsruhe de Alemania. Analiza la estructura y sintaxis del código, no comparando texto. Este sistema sólo trabaja con archivos locales, no busca similitudes en Internet. Su arquitectura de trabajo es Cliente/Servidor. La interfaz es a través de línea de comando, o, puede implementarse un cliente propio. Para el manejo de los archivos enviados, puede utilizarse el cliente brindado por la universidad que lo desarrolló o bien crear un cliente propio, como mencionamos anteriormente [Díaz *et al.*, 2007].

Tester SIM

Es una herramienta de código abierto. Trabaja con los lenguajes C, JAVA, Lisp, Modula 2, Pascal y texto natural. Está desarrollada por la Universidad de Amsterdam. No cuenta con una interfaz gráfica, se ingresan los distintos parámetros por línea de comando. Los resultados brindados por la misma aparecen separados en dos columnas, mostrando en cada una las porciones de código iguales. Sólo trabaja con búsquedas de similitudes en archivos

locales. También se puede procesar un conjunto de archivos viejos contra un conjunto de archivos nuevos [Díaz *et al.*, 2007].

MOSS

Software creado por el profesor Alex Aiken en la universidad de Stanford, es el primer servicio que inició en la web, siendo una referencia a nivel mundial. Este, permite comparar hasta 250 archivos en 25 lenguajes de programación [Hage y Rademaker, 2010]. Aunque no tiene licencia como software libre, su uso dentro del ambiente académico es gratuito y ofrecido desde un servidor de Stanford. Es de difícil su configuración y tiene poca documentación [Pachón, 2019].

1.2.2. Planteamiento del problema

Los estudiantes durante su proceso de formación elaboran trabajos, proyectos, tareas y ejercicios de programación escritos en un lenguaje de programación, estas actividades se realiza de forma individual o grupal, cuando estas actividades se desarrollan de forma individual los estudiantes deben ser conscientes de que todo trabajo entregado debe ser de su autoría, pues tiene como función medir la capacidad de resolución de problemas y el enfoque lógico y otros. Por ello encontrar similitud en trabajos de programación presentados por los estudiantes, puede ser identificado como plagio.

Una de las formas para detectar el plagio en los trabajos de programación consiste en realizar la comparación entre los trabajos entregados por los estudiantes de la materia. De a modo de obtener una lista de estudiantes que tienen trabajos similares. Realizar la comparación de los trabajos manualmente puede llegar a ser un trabajo moroso, por lo cual contar con una herramienta de software que realice las comparaciones de trabajos de forma automática es de gran utilidad.

En la actualidad existen diferentes herramientas de software que aplican métodos para detección de similitud entre códigos fuente, a partir de las características se pudo evidenciar que estas presentan deficiencias como ser la obsolescencia, sistemas cerrados (sin código abierto), un proceso complejo de evaluación de similitud, la incapacidad para utilizar o no una gran base de información. Esto se debe a que fueron diseñadas para detectar plagio entre un grupo pequeño de archivos de código fuente.

1.2.3. Formulación del problema

¿El algoritmo JTEL tiene mejor desempeño frente a otras herramientas en la detección de similitud de codigos fuente en trabajos de cátedra?

1.3. Objetivos

1.3.1. Objetivo general

Diseñar e implementar el algoritmo JTEL para la detección de similitud entre códigos fuente.

1.3.2. Objetivo específico

- Estudiar los métodos para la detección de similitud entre códigos fuente existentes.
- Estudiar los algoritmos utilizados en los métodos para la detección de similitud de código fuente.
- Redactar las especificaciones para el algoritmo JTEL.
- Evaluar el desempeño del algoritmo JTEL realizando las pruebas en trabajos de programación presentado por estudiantes.

1.4. Hipótesis

El algoritmo JTEL implementado para la detección de similitud entre códigos fuente obtiene mejores resultados frente a otras herramientas para la detección de similitud.

1.4.1. Variables Independientes

- El algoritmo JTEL para la detección de similitud entre códigos fuente.

1.4.2. Variables Dependientes

- Resultados más precisos respecto a la detección de similitud entre códigos fuente de trabajos de cátedra frente a otras herramientas.

1.5. Justificaciones

1.5.1. Justificación Social

El algoritmo JTEL ahorrará el tiempo de docentes de instituciones académicas en la detección de plagio en trabajos de programación presentados por estudiantes, evitando que los docentes realicen la comparación de trabajos de programación de forma manual.

1.5.2. Justificación Económica

El algoritmo JTEL para la detección de similitud entre códigos fuente sera de código abierto, por lo cual permitirá que se desarrollen otros software a bajo costo.

1.5.3. Justificación Tecnológica

El algoritmo JTEL para la detección de similitud entre códigos fuente, se puede implementar en jueces de programación para identificar los envíos similares de los usuarios del juez.

1.5.4. Justificación Científica

Con la utilización de trabajos de programación presentados por estudiantes se medirá el desempeño de los métodos existentes para la detección de similitud entre códigos fuente. Con los resultados obtenidos se determinará cuál es el método más eficiente en términos de tiempo de ejecución, espacio de memoria ocupado y precisión.

1.6. Alcances

1.6.1. Alcance Sustancial

- Se diseñará e implementará en Python un algoritmo JTEL para la detección de similitud entre códigos fuente.
- Se realizaran las pruebas de trabajos de cátedra de código fuente en Python.
- Se dejará de lado la comprobación de correctitud de los códigos fuente.
- Se realizarán pruebas para medir el tiempo, espacio de memoria ocupado y eficiencia del algoritmo.
- Se dejará de lado el estudio de métodos para la detección de similitud que aplican técnicas de aprendizaje automático vinculadas a la lingüística computacional, tales como minería de datos sobre texto y procesamiento del lenguaje natural.

1.6.2. Alcance Espacial

- Se realizaran las pruebas del algoritmo en una computadora Intel i3 de Decima generación con 8GB de RAM y 512GB de Disco Duro.

1.6.3. Alcance Temporal

- El tiempo de ejecución que el algoritmo estará limitado a un minuto. Por cada conjunto de trabajos evaluados.

1.7. Metodología

1.7.1. Metodología Experimental

Una de las metodologías que se utilizará es la experimental, de modo que al seguir las siguientes etapas ayudara a cumplir con los objetivos propuestos.

1. Recopilación de la información.
2. Diseño del algoritmo.
3. Pruebas de funcionamiento del algoritmo en trabajos de cátedra.

4. Análisis de los resultados obtenidos.
5. Conclusiones.

Las etapas consistirán en:

- En la primera etapa se recopilara información necesaria y estudiara los temas.
- En la segunda etapa se diseñara el algoritmo tomando en cuenta los alcances.
- En la tercera etapa se Realizara pruebas en trabajos de cátedra presentados por estudiantes.
- En la cuarta etapa se analizaran los datos y se los comparara frente a otras herramientas.
- En la quinta etapa se realizara las conclusiones, se presentaran los resultados finales, y recomendaciones respecto a la investigación.

1.7.2. Metodología Cuantitativa

Otra de las metodologías que se utilizará es la cuantitativa, al tratarse de un algoritmo, las pruebas arrojan resultados cuantitativos. De tal forma en la primera fase se medirá el desempeño de métodos para la detección de similitud estudiados, de modo que se escogerá algoritmo el que obtenga mejores resultados, luego de ello y según los resultados obtenidos en la fase previa, se realizarán ajustes al diseño del algoritmo seleccionado, de modo que garantice la solución al problema planteado. El diseño del algoritmo es la fase más importante y se le debe dar el mayor esfuerzo, haciendo uso de todos los conocimientos adquiridos en fases previas, para obtener una mejor solución para el problema planteado. Después se programará el algoritmo en un lenguaje de programación para visualizar y entregar resultados obtenidos.

Capítulo 2

Marco Teorico

2.1. Código fuente

El código fuente de un programa es un conjunto de líneas de texto, donde en cada línea contiene instrucciones del programa.

El código fuente de un programa está escrito por un programador en algún lenguaje de programación, pero en este primer estado no es directamente ejecutable por la computadora, sino que debe ser traducido a otro lenguaje o código binario, así será más fácil para la máquina interpretarlo (lenguaje máquina o código objeto que sí pueda ser ejecutado por el hardware de la computadora). Para esta traducción se usan los llamados compiladores, ensambladores, intérpretes y otros sistemas de traducción [Wikipedia, 2021d].

2.2. Plagio de código fuente

[Wikipedia, 2021c] La Real Academia Española define como plagio a la acción de copiar en lo sustancial obras ajenas, dándolas como propias.

El plagio de código fuente consiste en utilizar el código fuente de otra persona y adjudicarse como propio.

2.3. Ofuscación de código fuente

La ofuscación se refiere a encubrir el significado de una comunicación haciéndola más confusa y complicada de interpretar. En computación, la ofuscación se refiere al acto deliberado de realizar un cambio no destructivo, ya sea en el código fuente de un programa informático, en el código intermedio (bytecodes) o en el código máquina cuando el programa está en forma compilada o binaria. Es decir, se cambia el código se .[en] revesa"manteniendo el funcionamiento original, para dificultar su entendimiento. De esta forma se dificultan los intentos de ingeniería inversa y desensamblado que tienen la intención de obtener una forma de código fuente cercana a la forma original [Wikipedia, 2021b].

2.4. Métodos de ofuscación

Existen muchos métodos de ofuscación utilizados por estudiantes para ocultar la similitud a continuación se mencionan algunos:

- [Marzieh *et al.*, 2011] Mencionan cambios visuales en el formato del código, por lo que parece diferente a primera vista, esto generalmente incluye la modificación de espacios en blanco como sangrías, espacios, nuevas líneas, etc.

- [Marzieh *et al.*, 2011] Mencionan cambios en los comentarios del código.

- [Donaldson *et al.*, 1981] Mencionan el cambio de los nombres de los identificadores. como nombres de variables, nombres de constantes, nombres de funciones, nombres de clases, etc.

- [Donaldson *et al.*, 1981] Mencionan re ordenar las líneas del código para las que el pedido no marca ninguna diferencia. Estos incluyen cambiar el orden de las declaraciones de variables, cambiando el orden de las declaraciones dentro de bloques de código como funciones, re ordenación de bloques de código o funciones, re ordenación de clases internas, etc.

2.5. Representación de código fuente

2.5.1. Árboles de sintaxis abstracta

Los árboles de sintaxis abstracta es un modelo que representa el código fuente, por lo cual son utilizados por diferentes algoritmos para analizar el código fuente. Los árboles de sintaxis abstracta son árboles donde cada nodo es una construcción del código fuente.

2.5.2. Tokens

Es un enfoque basado en cadenas con la diferencia de que utiliza un analizador léxico, para convertir el programa a tokens, existen muchos algoritmos para medir la similitud entre secuencias de tokens.

2.5.3. Grafo de control de flujo

En ciencias de la computación, un grafo de control de flujo (CFG) es una representación, en forma de grafo dirigido, de todos los caminos que pueden ser atravesados a través de un programa durante su ejecución [Wikipedia, 2021a].

2.6. Detección de plagio en código fuente

2.6.1. Algoritmos utilizados para la detección de similitud entre códigos fuente

[Novak *et al.*, 2019] Identificó diferentes algoritmos a continuación se mencionan algunos de ellos: Recuento de atributos, huella digital, coincidencia de cadena, texto base, estructura base, estilo, semántico, n-gramas, árboles, grafos, etc. Algunos de estos fueron inventados en la década de 1980. También hace mención a que los enfoques basados en estructuras son mucho mejores y que la mayoría de las herramientas de detección de similitud combinan más de un tipo de algoritmo.

A continuación se mencionan algunos métodos para la detección de similitud:

- Métodos que utilizan flujo del código fuente.
- Métodos que utilizan alineación de cadenas y tokens.
- Métodos que utilizan técnicas de programación dinámica.
- Métodos que utilizan técnicas codiciosas.

2.7. Medidas de comparación

[Novak *et al.*, 2019] Explica que las métricas de comparación más utilizadas son: Selectividad, detección de exceso, índice de rendimiento, sensibilidad, velocidad, precisión. En el presente trabajo de tesis se consideran las métricas de velocidad y precisión.

Bibliografía

[Cheers *et al.*, 2021] Cheers, H., Lin, Y., y Smith, S. (2021). Academic source code plagiarism detection by measuring program behavioural similarity.

[Donaldson *et al.*, 1981] Donaldson, J. L., Lancaster, A.-M., y Sposato, P. H. (1981). A plagiarism detection system. pp. 21–25.

[Díaz *et al.*, 2007] Díaz, J., Banchoff, L., y Rodríguez, L. (2007). "herramientas para la detección de plagio de software. un caso de estudio en trabajos de cátedra".

[Hage y Rademaker, 2010] Hage, J. y Rademaker, P. (2010). A comparison of plagiarism detection tools.

[Marzieh *et al.*, 2011] Marzieh, A., Mahmoudabadi, E., y Khodadadi, F. (2011). Pattern of plagiarism in novice students generated programs: An experimental approach. *The Journal of Information Technology Education*, 10.

[Novak *et al.*, 2019] Novak, M., Joy, M., y Kermek, D. (2019). Source-code similarity detection and detection tools used in academia: A systematic review. *ACM Trans. Comput. Educ.*, 19(3).

[Pachón, 2019] Pachón, H. (2019). *Generación de un algoritmo para el análisis de similitudes de código fuente en lenguajes Java y Python*. Uniandes.

[Wikipedia, 2021a] Wikipedia (2021a). Control-flow graph — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Control-flow%20graph&oldid=1050421841`. [Online; accessed 02-December-2021].

[Wikipedia, 2021b] Wikipedia (2021b). Obfuscation (software) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Obfuscation%20(software)&oldid=1056756228`. [Online; accessed 02-December-2021].

[Wikipedia, 2021c] Wikipedia (2021c). Plagiarism — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=1058139870`. [Online; accessed 02-December-2021].

[Wikipedia, 2021d] Wikipedia (2021d). Source Code — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Source%20Code&oldid=1057866572`. [Online; accessed 02-December-2021].