Informe de Laboratorio 06

Tema: Arbol AVL

INFORMACIÓN BÁSICA					
ASIGNATURA:	**Estructura de Datos y Algoritmos**				
TÍTULO DEL TRABAJO:	Trabajo Final: Detector de plagio				
NÚMERO DE TRABAJO:	07	**AÑO LECTIVO:**	2023-A	**NRO. SEMESTRE:**	III
FECHA DE PRESENTACIÓN:	08/08/23	**HORA DE PRESENTACIÓN:**	23:59		
INTEGRANTE (s)				**NOTA (0-20)**	
Hidalgo Chinchay, Paulo Andre **Betanzos Rosas, Taylor Anthony** **Villafuerte Ccapira Frank Alexis**					
DOCENTE(s): Mg. Edith Giovanna Cano Mamani					

Tabla 1: Mi tabla extendida

INTRODUCCIÓN

En el campo de la investigación el robo de propiedad intelectual es un problema muy grande, y la preocupación ante este problema crece aún más debido a la facilidad que nos ofrece la tecnología para acceder a la información.

En el presente trabajo presentaremos la implementación de un detector de plagio textual, el cual buscará aspectos similares entre un párrafo base y varios documentos; y a partir de esto dará un veredicto.

MARCO CONCEPTUAL

TRIE

Un trie es una estructura de datos de tipo árbol que permite la recuperación de información. La información almacenada en un trie es un conjunto de claves, donde una clave es una secuencia de símbolos pertenecientes a un alfabeto. Las claves son almacenadas en las hojas del árbol y los nodos internos son pasarelas para guiar la búsqueda. El árbol se estructura de forma que cada letra de la clave se sitúa en un nodo de forma que los hijos de un nodo representan las distintas posibilidades de símbolos diferentes que pueden continuar al símbolo representado por el nodo padre. Por tanto la búsqueda en un trie se hace de forma similar a como se hacen las búsquedas en un diccionario[1]

PLAGIO

Según la RAE, el plagio es la acción de plagiar, el cual consiste copiar obras ajenas y darlas como propias[2,3]

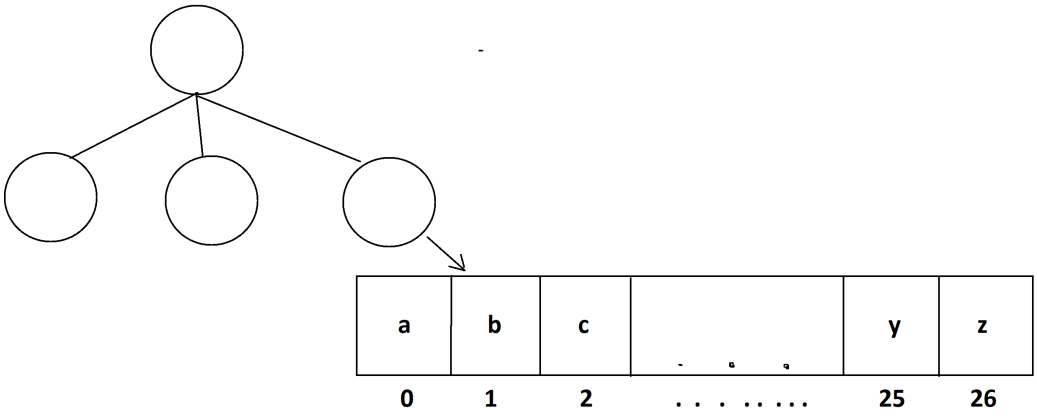
DETECTOR DE PLAGIO

Es el proceso de detectar las similitudes de contenido, el cual se realizaba mediante la detección humana. Gracias a la llegada de la tecnología, este proceso facilitó la tarea, que normalmente llevaba una gran cantidad de tiempo[4].

SOLUCIONES Y PRUEBAS

El detector de plagio requiere una comparación entre palabras del texto de referencia y el texto evaluado, por lo cual debemos almacenar las palabras del primer texto de tal manera que podamos encontrarlos con facilidad.

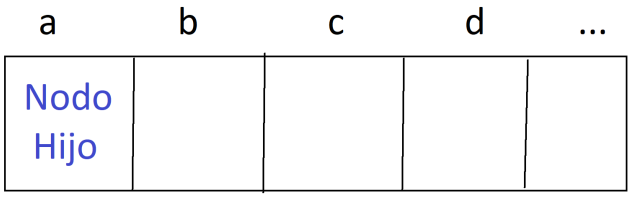
En este caso creamos un Trie, el cual es un árbol que almacenará las palabras. En nuestro código, cada Nodo del Trie tendrá un arreglo de 27 posiciones, representando a cada una de las letras del alfabeto. Según la letra que se vaya a almacenar se agrega un nodo hijo a la posicion correspondiente.



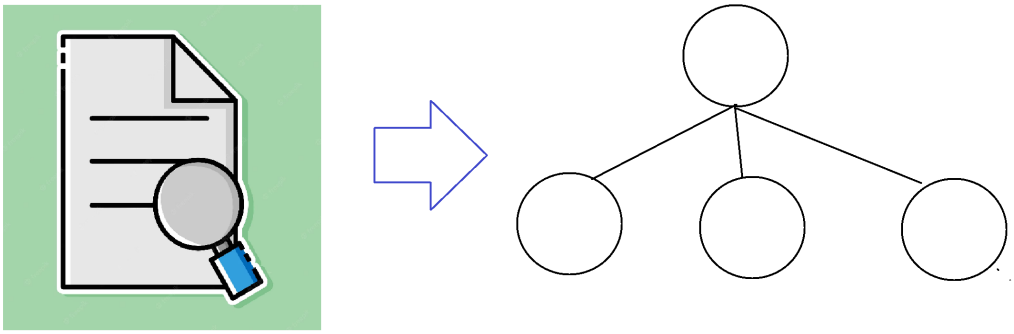
Para corresponder la posicion de las letras debemos asignar un valor numérico teniendo a la letra a como 0, b como 1 y así sucesivamente.
Para lograr esto usaremos el código ascii de cada letra, y le restaremos el valor de la ."a", así obtendremos un valor relativo a las posiciones del alfabeto.

a=97

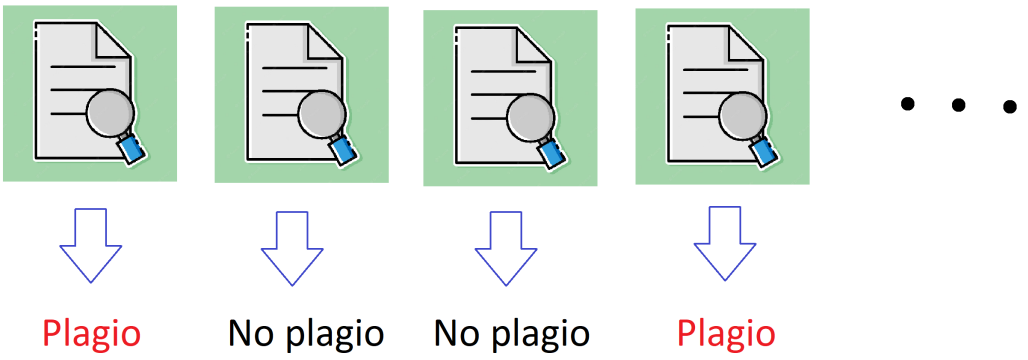
a-a=0
b-a=1
c-a=2
d-a=3
.
.
.



El procedimiento de detección de plagio requiere que tengamos un documento de referencia.
Usaremos la clase PlagiarismChecker para obtener el contenido de los documentos y agregar las palabras al Trie que será almacenado dentro de esta misma clase.
Mediante el botón de añadir archivo de la interfaz obtendremos el texto que contiene y agregaremos todas sus palabras al trie.



La métrica utilizada para el veredicto es el porcentaje de palabras similares del texto de referencia en relación a todas las palabras existentes en dicho texto. La clase ResulChacker buscará cada palabra en el trie e irá contando los resultados similares, luego lo comparará con la cantidad de palabras totales multiplicado por 0,2. Por tanto si el número de similitudes supera el 20 % el sistema alertará sobre plagio. Se compara cada uno de los documentos ingresados y la clase retornará un arreglo de booleanos que servirán a la interfaz para enviar un aviso de los resultados por cada documento.



En este caso usamos la clase ResulChacker dentro de PlagiarismChecker para poder enviarle el un ArrayList de Trie como parámetro. Un Trie por documento. El cual es igual al usado en el lab6 con un atributo más, el cual es el size, que es un entero con la cantidad de palabras que tiene el Trie.

```java
package trie;

public class Trie {
    private TrieNode root = new TrieNode();
    private int size =0;

    public Trie() {
    }

    public int getSize() {
        return size;
    }
```

Como decia los requerimientos, se necesitaba subir los archivos desde una ruta por lo cual se necesitaba del metodo auxiliar loadFile que guarda el texto de un archivo en un Trie siempre y cuando el archivo sea .txt, asimismo si ese texto tiene caracteresa especiales no son guardados y muestra un mensaje de alerta.

Listing 1: loadFile.java

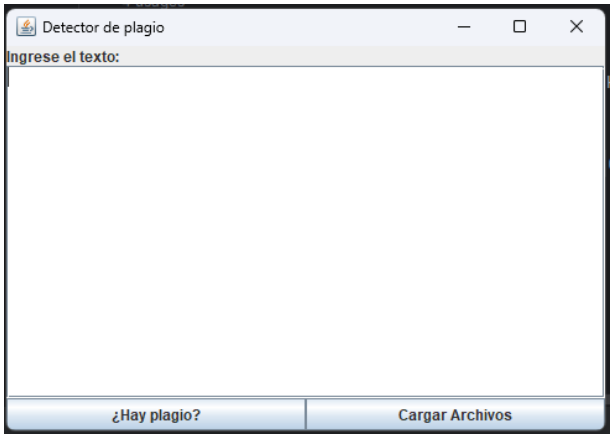
```
private boolean loadFile(String path) {
       try (BufferedReader reader = new BufferedReader(new FileReader(path))) {
           if (path.endsWith(".txt")) {
               String line;
               String msg = "";
               boolean exito = false;
               Trie trie = new Trie();
               while ((line = reader.readLine()) != null) {
                   String[] palabras = line.split("\\ss+| ");
                   for (int i = 0; i < palabras.length; i++) {
                       palabras[i] = palabras[i].trim().toLowerCase();
                       Pattern pattern = Pattern.compile("^[a-z]+$");
                       if (pattern.matcher(palabras[i]).matches()) {
                           exito = true;
                           trie.insert(palabras[i]);
                       } else {
                           msg = "hay palabras que son letras o tienen caracteres
                               especiales";
                       }
                   }
               }
               if (!msg.isEmpty())
                   JOptionPane.showMessageDialog(null, "texto no proporcionado en " + path
                       + " no es de solo palabras", msg, JOptionPane.ERROR_MESSAGE);
               if (exito) tries.add(trie);
               return exito;
           }else return false;
       } catch (Exception e) {
           return false;
       }
   }
```

Por otro lado tenemos al metodo detectPlagiarism en la clase ResulChacker que cambia el valor del arrelo del booleano dependiendo si hay plagio o no, considerando este a partir de la copia textual de el 20 por ciento o más de las palabras del alguno de los archivos

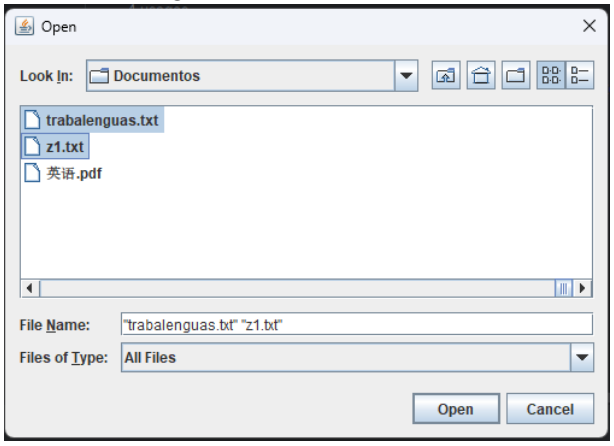
Listing 2: detectPlagiarism.java

```
private void detectPlagiarism(ArrayList<Trie> tries) {
       String[] words = str.toLowerCase().split("\\s+");
       for (int i = 0; i < resul.length; i++) {
           int numPalbrasConcidentes =0;
           Trie trie = tries.get(i);
           for (String word : words) {
               if (trie.search(word)) {
                   numPalbrasConcidentes++;
               }
           }
           System.out.println(trie.getSize());
           resul[i] = (int) (0.2 * trie.getSize()) <= numPalbrasConcidentes;
       }
   }
```

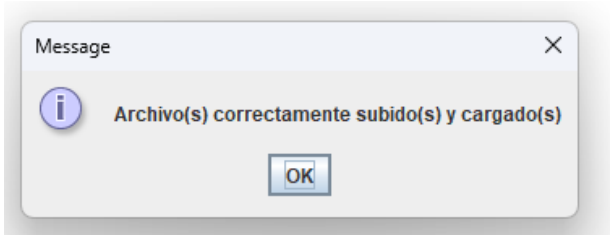
Con todo lo anterior mencionado se implementa la interfaz la cual pide que primero subamos un archivo de formato correcto para saber si el texto que ingresamos en el JTextArea se considera plagio o no. Pantalla inicial



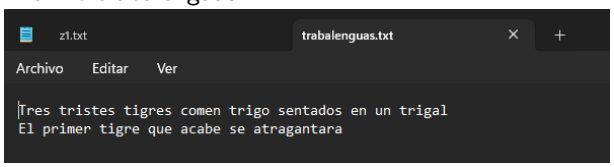
Pantalla de carga



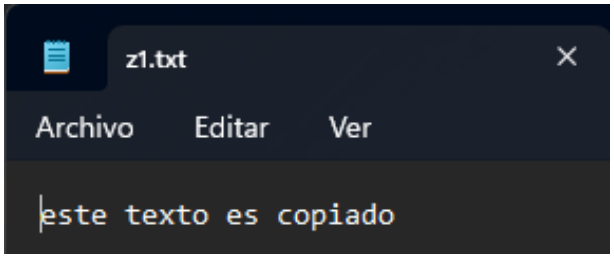
Carga correcta



Archivo trabalenguas



Archivo z1



Prueba sin plagio

Prueba con plagio

LECCIONES APRENDIDAS Y CONCLUSIONES

Durante el proceso de ejecución de este proyecto aprendimos a utilizar los árboles TRIE para facilitar la búsqueda de palabras específicas y verificar su existencia en un texto.
Ademas aprendimos, a grandes rasgos, como funciona un detector de plagio textual y cómo el usar TRIES facilita el proceso de comparación y determinar que tan similar es el documento de referencia con el párrafo evaluado.

REFERENCIAS Y BIBLIOGRAFÍA

[1]https://es.wikipedia.org/wiki/Trie
[2]https://dle.rae.es/plagio
[3]https://dle.rae.es/plagiar#CU0knYP
[4]https://www.ayudauniversitaria.com/detector-de-plagio/