

Informe de Laboratorio 06

Tema: Arbol AVL

INFORMACIÓN BÁSICA

ASIGNATURA:	Estructura de Datos y Algoritmos				
TÍTULO DEL TRABAJO:	Tries				
NÚMERO DE TRABAJO:	06	AÑO LECTIVO:	2023-A	NRO. SEMESTRE:	III
FECHA DE PRESENTACIÓN:	23/07/23	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

Se implementaran los diferentes metodos con los que cuentan los Tries como son la insercion, busqueda y eliminacion, con la finalidad de crear un programa que permita ingresar palabras, buscarlas y reemplazarlas.

MARCO CONCEPTUAL

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]

SOLUCIONES Y PRUEBAS

Para el metodo insert se iteraba con un for letra por letra de la palabra a insertar para luego obtener su valor en el codigo ACSSI, con el valor resultante con se veia si el hijo del TrieNode actual era nulo, si era nulo se creaba un nuevo TrieNode con un valor de endOfWord en falso, hasta llegar a la penultima letra de la palabra, en la cual se terminaba el for y se creaba un nuevo TrieNode con el valor de endOfWord en verdadero.

```
public void insert(String palabra) {
    TrieNode actual = this.root;

    int j;
    for(int i = 0; i < palabra.length() - 1; ++i) {
        j = palabra.charAt(i) - 'a';
        actual.insertChildren(j, eok: false);
        if (actual.getChildren(j) != null) actual = actual.getChildren(j);
    }

    j = palabra.charAt(palabra.length() - 1) - 'a';
    actual.insertChildren(j, eok: true);
}
```

Para el metodo search se iteraba caracter por caracter de la palabra buscada al ifual que en el metodo insert, sin embargo, si aqui el hijo del nodo actual era nulo retornaba false y si no continuaba con el ciclo hasta la penultima letra. La ultima letra a parte de existir debia tambien cumplir en ser endOfWord retornando asi la union de estos 2 valores.

```
public boolean search(String palabra) {
    TrieNode actual = this.root;

    int j;
    for(int i = 0; i < palabra.length() - 1; ++i) {
        j = palabra.charAt(i) - 'a';
        if (actual.getChildren(j) == null) {
            return false;
        }

        actual = actual.getChildren(j);
    }

    j = palabra.charAt(palabra.length() - 1) - 'a';
    actual = actual.getChildren(j);
    return actual != null && actual.isEndOfWord();
}
```

El método delete sirve para eliminar una palabra el cual era un metodo recursivo que retornaba un boolean, lo cual permitia saber si se debe o no eliminar un TrieNode con esto se eliminaba de la ultima letra hasta la 1ra, en caso no hubiera un endOfWord en el camino, en caso fuera asi, dejaba de eliminar.

```
public boolean search(String palabra) {
    TrieNode actual = this.root;

    int j;
    for(int i = 0; i < palabra.length() - 1; ++i) {
        j = palabra.charAt(i) - 'a';
        if (actual.getChildren(j) == null) {
            return false;
        }

        actual = actual.getChildren(j);
    }

    j = palabra.charAt(palabra.length() - 1) - 'a';
    actual = actual.getChildren(j);
    return actual != null && actual.isEndOfWord();
}
```

Para obtener el máximo solo se retorna la raíz, ya que es un AVL máximo.

```
private boolean delete(TrieNode actual, String palabra, int indice) {
    if (indice == palabra.length()) {
        if (!actual.isEndOfWord()) {
            return false;
        }
        actual.setEndOfWord(false);
        return isEmpty(actual);
    }

    int indiceTrie = palabra.charAt(indice) - 'a';
    TrieNode nodo = actual.getChildren(indiceTrie);
    if (nodo == null) {
        return false;
    }

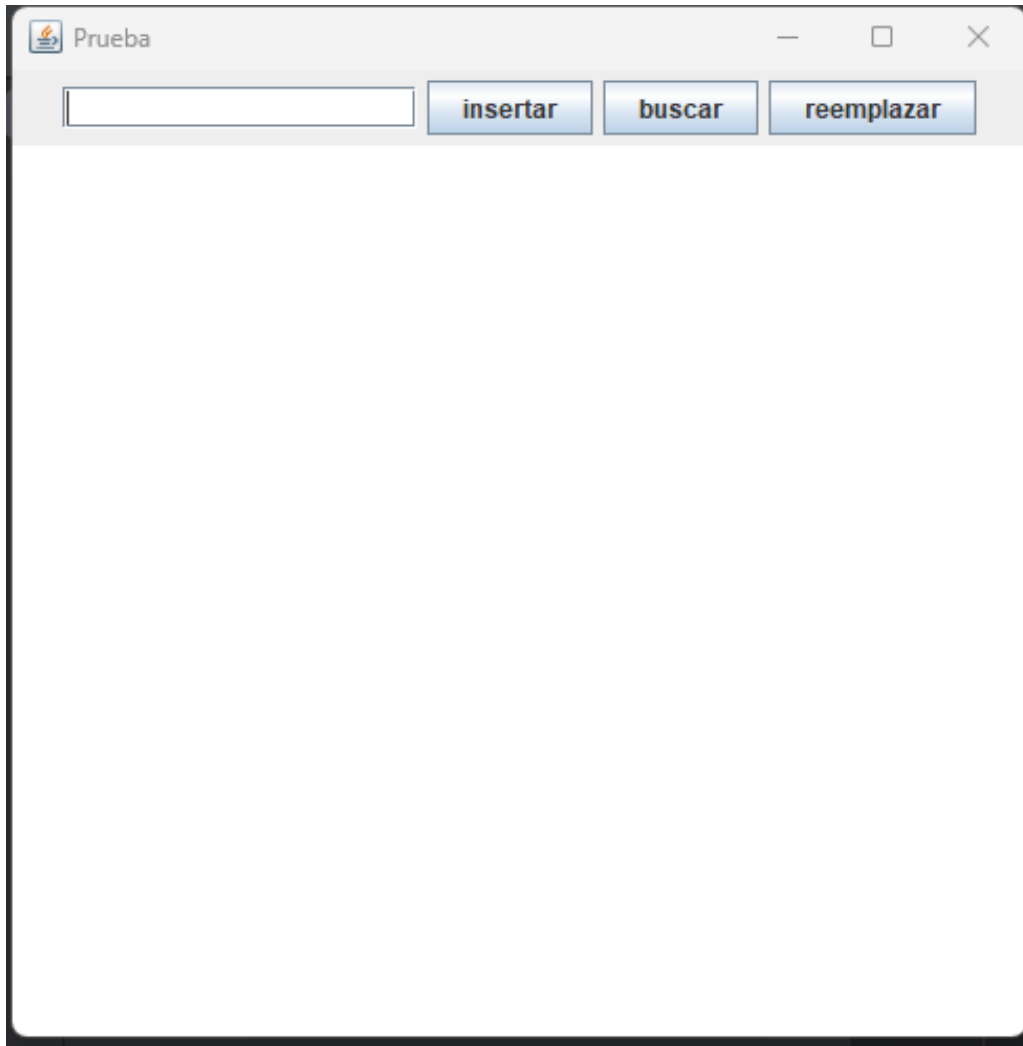
    boolean deberiaEliminarNodoActual = delete(nodo, palabra, indice + 1);
    if (deberiaEliminarNodoActual) {
        actual.setChildren( new TrieNode[26], indiceTrie);
        return isEmpty(actual);
    }

    return false;
}
```


El metodo replace, sirve para reemplazar un palabra por otra, por lo cual elimina una palabra y pone a la otra en su lugar, para ello utiliza los metodos delete e insert respectivamente.

```
public void reemplazar(String palabraAntigua, String palabraNueva) {
    delete(palabraAntigua);
    insert(palabraNueva);
}
```

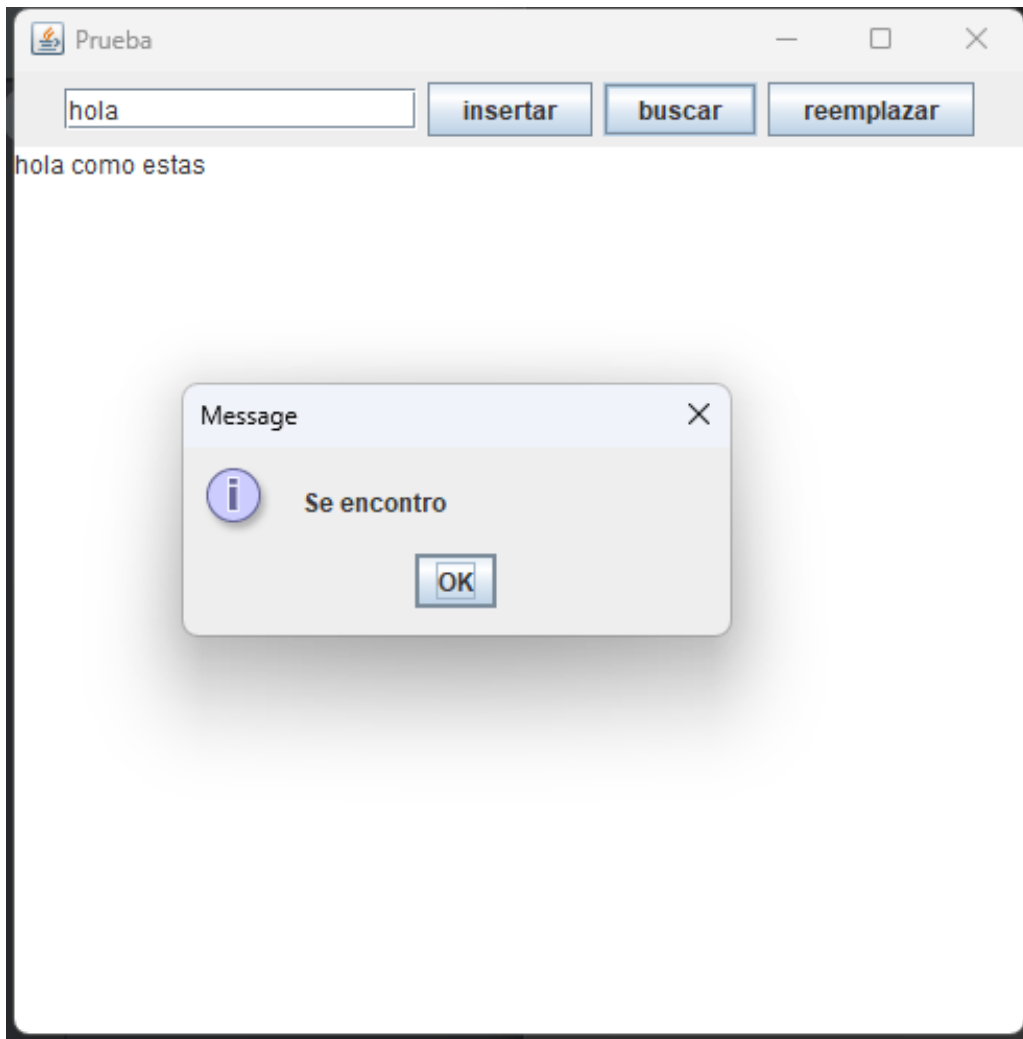
Ejecucion del programa:

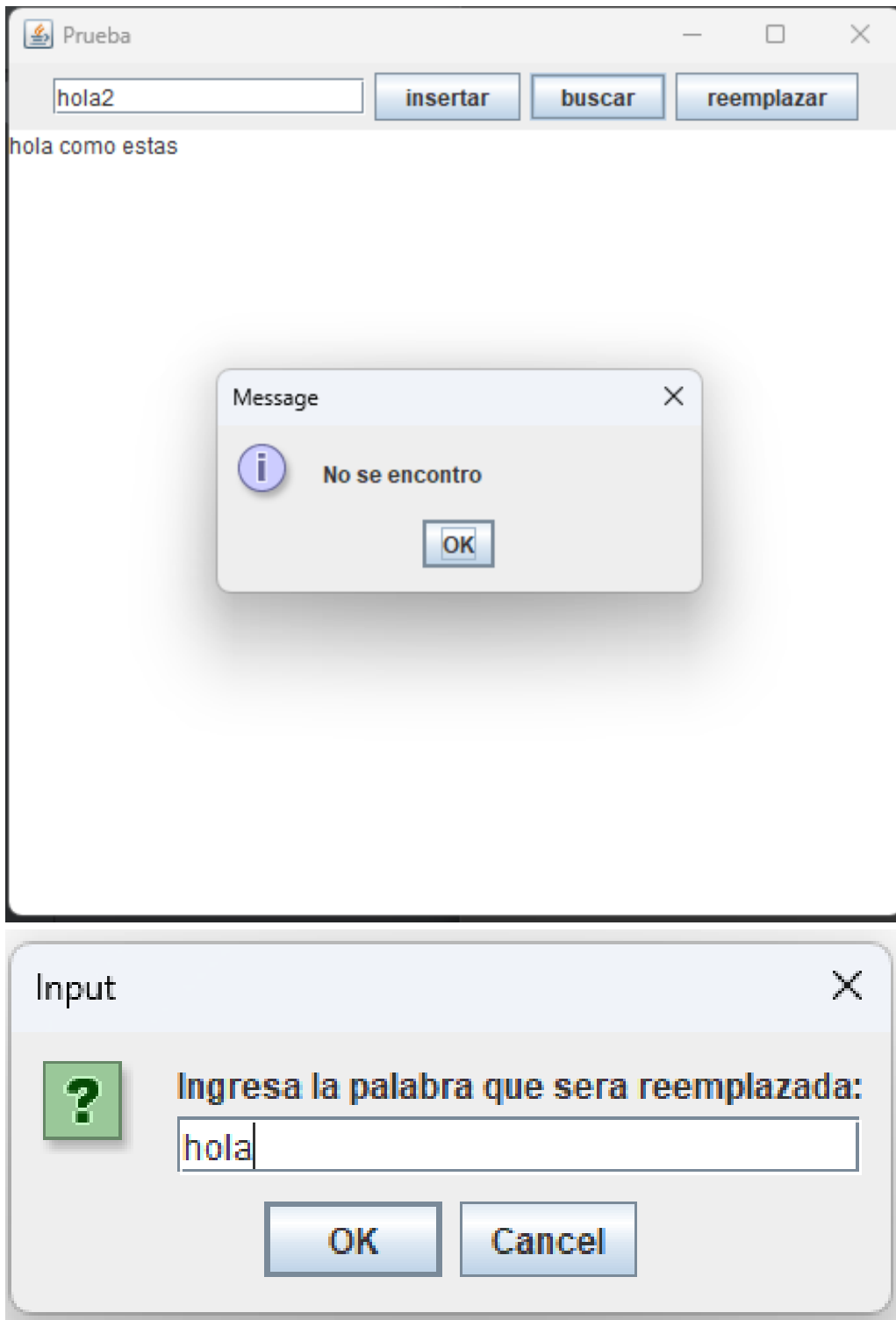


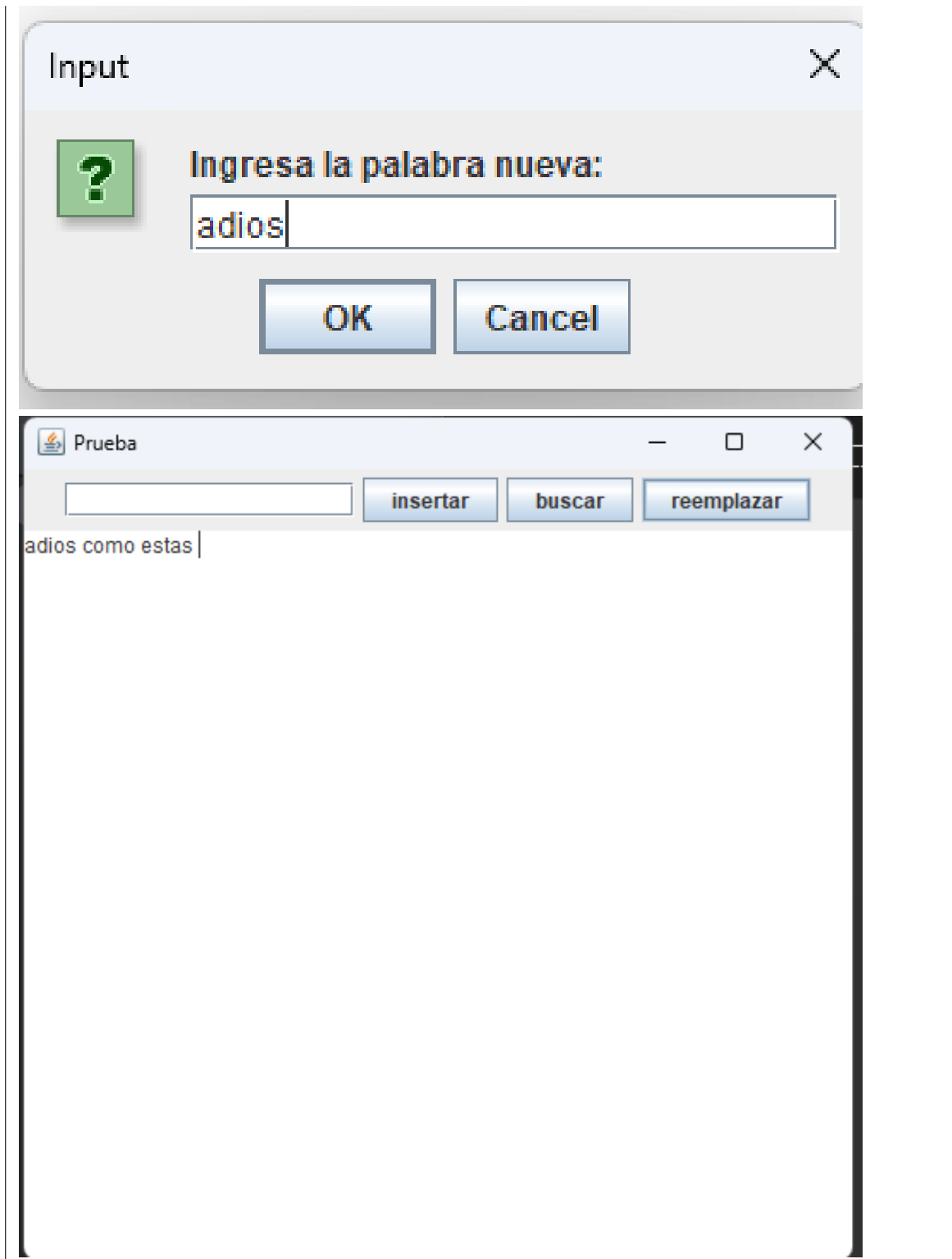
The image shows a screenshot of a Java Swing window titled "Prueba". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar, there is a text input field on the left and three buttons labeled "insertar", "buscar", and "reemplazar" on the right. The main area of the window is empty.

 **Prueba** — □ ×

hola como estas







LECCIONES APRENDIDAS Y CONCLUSIONES

Se aprendió a como implementar los principales metodos de los Tries como lo son la insercion, busqueda y eliminacion. Al igual que hacer un recuerdo del uso de intefaces graficas.

REFERENCIAS Y BIBLIOGRAFÍA

[1]<https://es.wikipedia.org/wiki/Trie>