

# Informe de Laboratorio 06

## Tema: Tries

Nota

Estudiantes	Escuela	Asignatura
Andre David Delgado Allpan, Andre Hilacondo Begazo, Gonzalo Layme Mamani, Diego Llerena Tellez, Mauricio Zegarra Puma	Escuela Profesional de Ingeniería de Sistemas	Estructura de datos y algoritmos Semestre: III Código: 20231001

Laboratorio	Tema	Duración
06	Tries	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 10 Julio 2023	Al 17 Julio 2023

## 1. URL de Repositorio Github

- URL para el laboratorio 06 en el Repositorio GitHub.
- [https://github.com/GonzaloRail/Laboratorio\\_06\\_EDA.git](https://github.com/GonzaloRail/Laboratorio_06_EDA.git)

## 2. Tarea

### 2.1. Introducción

- En este informe, describiremos la implementación de una Trie con una interfaz gráfica de usuario para permitir al usuario insertar, buscar y reemplazar palabras en un texto. Una Trie es una estructura de datos eficiente para almacenar y buscar palabras o cadenas de caracteres, especialmente cuando hay muchas palabras con prefijos comunes.

### 2.2. Descripción del Trie

- La Trie se implementa utilizando una estructura de árbol, donde cada nodo del árbol representa un carácter en una palabra. Cada nodo tiene un arreglo de 26 referencias a otros nodos, correspondientes a las 26 letras del alfabeto inglés. Si una palabra tiene un carácter específico en una posición, el nodo correspondiente tendrá una referencia no nula al siguiente nodo que representa la siguiente letra en la palabra.

### 2.3. Métodos principales de la clase Trie

- `insert(String word)`: Inserta una palabra en la Trie. Comienza desde la raíz y recorre los nodos correspondientes a cada letra de la palabra. Marca el último nodo como el final de la palabra.

Listing 1: Trie.java (Metodo insert)

```
1 public void insert(String word) {
2     if (word == null || word.isEmpty()) {
3         throw new IllegalArgumentException("Invalid input");
4     }
5
6     TrieNode current = root;
7     for (char c : word.toCharArray()) {
8         int index = getIndex(c);
9         if (current.children[index] == null) {
10             current.children[index] = new TrieNode();
11         }
12         current = current.children[index];
13     }
14     current.isWord = true;
15 }
```

- `search(String word)`: Busca una palabra en la Trie. Comienza desde la raíz y recorre los nodos correspondientes a cada letra de la palabra. Si encuentra todos los nodos correspondientes y el último nodo está marcado como el final de una palabra, significa que la palabra está presente en la Trie.

Listing 2: Trie.java (Metodo search)

```
1 public boolean search(String word) {
2     if (word == null || word.isEmpty()) {
3         return false;
4     }
5
6     TrieNode current = root;
7     for (char c : word.toCharArray()) {
8         int index = getIndex(c);
9         if (current.children[index] == null) {
10             return false;
11         }
12         current = current.children[index];
13     }
14     return current.isWord;
15 }
```

- `delete(String word)`: Elimina una palabra de la Trie. Comienza desde la raíz y recorre los nodos correspondientes a cada letra de la palabra. Marca el último nodo como no el final de la palabra.

Listing 3: Trie.java (Metodo delete)

```
1 public void delete(String word) {
2     if (word == null || word.isEmpty()) {
3         throw new IllegalArgumentException("Invalid input");
4     }
5
6     TrieNode current = root;
7     for (char c : word.toCharArray()) {
8         int index = getIndex(c);
9         if (current.children[index] == null) {
10            throw new NoSuchElementException("Word not found in the Trie");
11        }
12        current = current.children[index];
13    }
14    current.isWord = false;
15 }
```

- `replace(String word, String replacement)`: Reemplaza una palabra en la Trie con otra palabra proporcionada. Primero, busca la palabra original y la reemplaza con la nueva palabra. Si la palabra original no está presente, se lanza una excepción.

Listing 4: Trie.java (Metodo replace)

```
1 public void replace(String word, String replacement) {
2     if (word == null || word.isEmpty() || replacement == null) {
3         throw new IllegalArgumentException("Invalid input");
4     }
5
6     if(word.equals(replacement)) {
7         throw new IllegalArgumentException("same words");
8     }
9
10    for (int i = 0; i < content.size(); i++) {
11        if (content.get(i).equals(word)) {
12            content.set(i, replacement);
13        }
14    }
15
16    delete(word); //eliminar del trie
17    insert(replacement); //agregar al trie
18 }
```

## 2.4. Estructura de la implementación

- Trie: La clase principal que representa la Trie y contiene los métodos para insertar, buscar y eliminar palabras.

Listing 5: Trie.java

```
1 import java.util.*;
2 public class Trie {
3     private TrieNode root;
4     private LinkedList<String> content;
5
6     public Trie() {
7         root = new TrieNode(); //root is empty
8         content = new LinkedList<>();
9     }
10
11     private class TrieNode {
12         private TrieNode[] children;
13         private boolean isWord;
14
15         public TrieNode() {
16             children = new TrieNode[26]; // storing english words - a -> z
17             isWord = false;
18         }
19     }
20
21     private int getIndex(char c) {
22         return c - 'a';
23     }
24
25     public void insert(String word) {
26         if (word == null || word.isEmpty()) {
27             throw new IllegalArgumentException("Invalid input");
28         }
29
30         TrieNode current = root;
31         for (char c : word.toCharArray()) {
32             int index = getIndex(c);
33             if (current.children[index] == null) {
34                 current.children[index] = new TrieNode();
35             }
36             current = current.children[index];
37         }
38         current.isWord = true;
39     }
40
41     public boolean search(String word) {
42         if (word == null || word.isEmpty()) {
43             return false;
44         }
45
46         TrieNode current = root;
47         for (char c : word.toCharArray()) {
48             int index = getIndex(c);
49             if (current.children[index] == null) {
50                 return false;
```

```
51         }
52         current = current.children[index];
53     }
54     return current.isWord;
55 }
56
57 public void replace(String word, String replacement) {
58     if (word == null || word.isEmpty() || replacement == null) {
59         throw new IllegalArgumentException("Invalid input");
60     }
61
62     if(word.equals(replacement)) {
63         throw new IllegalArgumentException("same words");
64     }
65
66     for (int i = 0; i < content.size(); i++) {
67         if (content.get(i).equals(word)) {
68             content.set(i, replacement);
69         }
70     }
71
72     delete(word); //eliminar del trie
73     insert(replacement); //agregar al trie
74 }
75
76 public void delete(String word) {
77     if (word == null || word.isEmpty()) {
78         throw new IllegalArgumentException("Invalid input");
79     }
80
81     TrieNode current = root;
82     for (char c : word.toCharArray()) {
83         int index = getIndex(c);
84         if (current.children[index] == null) {
85             throw new NoSuchElementException("Word not found in the Trie");
86         }
87         current = current.children[index];
88     }
89     current.isWord = false;
90 }
91
92 public String getContent() {
93     StringBuilder contentGlobal = new StringBuilder();
94     for (String word : content) {
95         contentGlobal.append(word).append(" ");
96     }
97     return contentGlobal.toString();
98 }
99
100 public void insertList(String word) {
101     content.add(word);
102 }
103
104
105 }
```

- TrieNode: Una clase interna de Trie que representa un nodo en la Trie, que contiene las referencias a otros nodos y un indicador para marcar el final de una palabra.

Listing 6: Trie.java (Líneas 11-19)

```
1 private class TrieNode {
2     private TrieNode[] children;
3     private boolean isWord;
4
5     public TrieNode() {
6         children = new TrieNode[26]; // storing english words - a -> z
7         isWord = false;
8     }
9 }
```

- TrieGUI: La clase que crea la interfaz gráfica de usuario para interactuar con la Trie.

Listing 7: TrieGUI.java

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import java.util.*;
6
7 public class TrieGUI extends JFrame implements design{
8
9     private Trie trie;
10    private JTextArea consoleTextArea;
11    private JTextField contentTextField;
12    private JTextArea contentTextArea;
13
14    public TrieGUI() {
15        trie = new Trie();
16
17        // GUI components
18        JLabel insertLabel = new JLabel("Insert a Word:");
19        JLabel outputLabel = new JLabel("Output:");
20        JLabel contentLabel = new JLabel("Content:");
21        JButton insertButton = new JButton("Insert");
22        JButton searchButton = new JButton("Search");
23        JButton replaceButton = new JButton("Replace");
24        JTextField inputField = new JTextField(40);
25
26        // Console to show output
27        consoleTextArea = new JTextArea(10, 10);
28        consoleTextArea.setEditable(false);
29        consoleTextArea.setFont(buttonFont);
30        JScrollPane scrollPane = new JScrollPane(consoleTextArea);
31
32        // Console to show output
33        contentTextArea = new JTextArea(10, 10);
34        contentTextArea.setEditable(false);
35        contentTextArea.setFont(buttonFont);
36        JScrollPane contentP = new JScrollPane(contentTextArea);
37
38        // Console to show content
```

```
39     contentTextField = new JTextField();
40     contentTextField.setEditable(false);
41     contentTextField.setFont(buttonFont);
42     JScrollPane contentPane = new JScrollPane(contentTextField);
43
44
45     // Set custom colors for buttons
46     insertButton.setBackground(buttonColor);
47     searchButton.setBackground(buttonColor);
48     replaceButton.setBackground(buttonColor);
49
50     // Set custom font for buttons
51     insertButton.setFont(buttonFont);
52     searchButton.setFont(buttonFont);
53     replaceButton.setFont(buttonFont);
54     insertLabel.setFont(buttonFont);
55     outputLabel.setFont(buttonFont);
56     contentLabel.setFont(buttonFont);
57
58
59     // Button listeners
60     insertButton.addActionListener(new ActionListener() {
61         public void actionPerformed(ActionEvent e) {
62             String word = inputField.getText();
63             try {
64                 trie.insert(word);
65                 trie.insertList(word);
66                 logToConsole("Inserted: " + word);
67                 logToContent();
68             } catch (IllegalArgumentException ex) {
69                 logToConsole("Invalid input");
70             }
71             inputField.setText("");
72         }
73     });
74
75     searchButton.addActionListener(new ActionListener() {
76         public void actionPerformed(ActionEvent e) {
77             String word = inputField.getText();
78             boolean found = trie.search(word);
79             logToConsole("Search: " + word + " -> " + found);
80             inputField.setText("");
81         }
82     });
83
84     replaceButton.addActionListener(new ActionListener() {
85         public void actionPerformed(ActionEvent e) {
86             String wordToReplace = inputField.getText();
87             String replacementWord = JOptionPane.showInputDialog(TrieGUI.this, "Enter
88                 replacement word:");
89             try {
90                 trie.replace(wordToReplace, replacementWord);
91                 logToConsole("Replaced: " + wordToReplace + " with " +
92                     replacementWord);
93                 logToContent();
94             } catch (IllegalArgumentException | NoSuchElementException ex) {
```

```
193         logToConsole(ex.getMessage());
194     }
195     inputField.setText("");
196 }
197 });
198
199 // Layout
200 JPanel buttonPanel = new JPanel();
201 buttonPanel.add(insertButton);
202 buttonPanel.add(searchButton);
203 buttonPanel.add(replaceButton);
204
205 JPanel inputPanel = new JPanel(new BorderLayout());
206 inputPanel.add(insertLabel, BorderLayout.NORTH);
207 inputPanel.add(inputField, BorderLayout.CENTER);
208
209 JPanel outputPanel = new JPanel(new BorderLayout());
210 outputPanel.add(outputLabel, BorderLayout.NORTH);
211 outputPanel.add(scrollPane, BorderLayout.CENTER);
212
213 JPanel contentPanel = new JPanel(new BorderLayout());
214 contentPanel.add(contentLabel, BorderLayout.NORTH);
215 contentPanel.add(contentP, BorderLayout.CENTER);
216
217 JPanel global = new JPanel(new BorderLayout());
218 global.add(buttonPanel, BorderLayout.NORTH);
219 global.add(inputPanel, BorderLayout.CENTER);
220 global.add(outputPanel, BorderLayout.SOUTH);
221
222 JPanel mainPanel = new JPanel(new BorderLayout());
223 mainPanel.add(global, BorderLayout.CENTER);
224 mainPanel.add(contentPanel, BorderLayout.SOUTH);
225
226 add(mainPanel);
227
228 setTitle("Insert, Search and Replace");
229 pack();
230 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
231 setLocationRelativeTo(null);
232 setVisible(true);
233 }
234
235 private void logToConsole(String message) {
236     consoleTextArea.append(message + "\n");
237     consoleTextArea.setCaretPosition(consoleTextArea.getDocument().getLength());
238 }
239
240 private void logToContent() {
241     contentTextArea.setText(trie.getContent());
242 }
243
244 public static void main(String[] args) {
245     SwingUtilities.invokeLater(new Runnable() {
246         public void run() {
247             new TrieGUI();
248         }
249     });
250 }
```

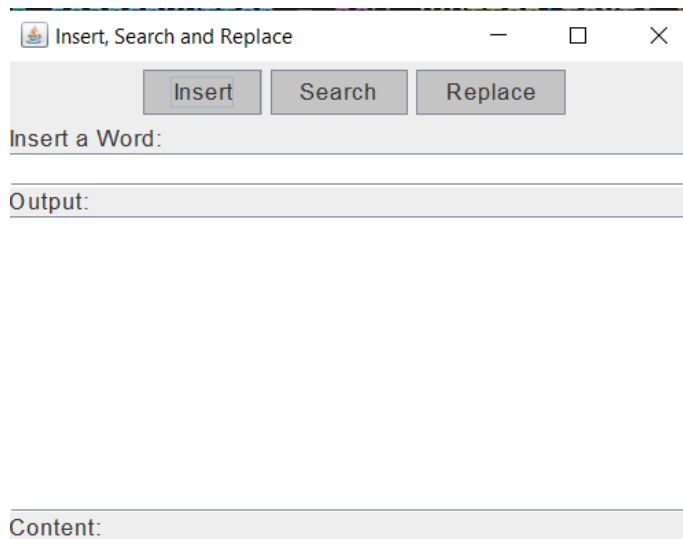


```
149     });
150   }
151 }
```

## 2.5. Descripción general de la clase TrieGUI

La clase TrieGUI crea una interfaz gráfica para permitir al usuario interactuar con la Trie. La interfaz gráfica contiene los siguientes componentes:

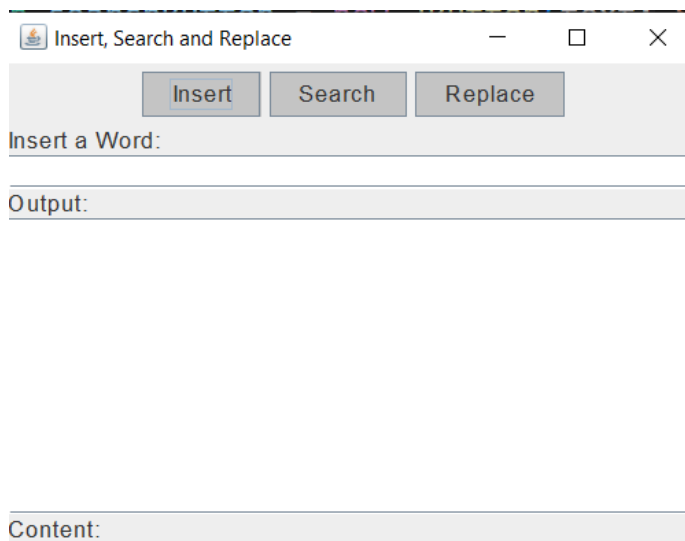
- Etiquetas: Etiquetas para indicar qué acción realizar (insertar, buscar, reemplazar).
- Botones: Botones para realizar las acciones de inserción, búsqueda y reemplazo.
- Cuadros de texto: Campos de texto para ingresar palabras y ver la salida en la consola y el contenido actual en la Trie.



## 2.6. Descripción general de la clase TrieGUI

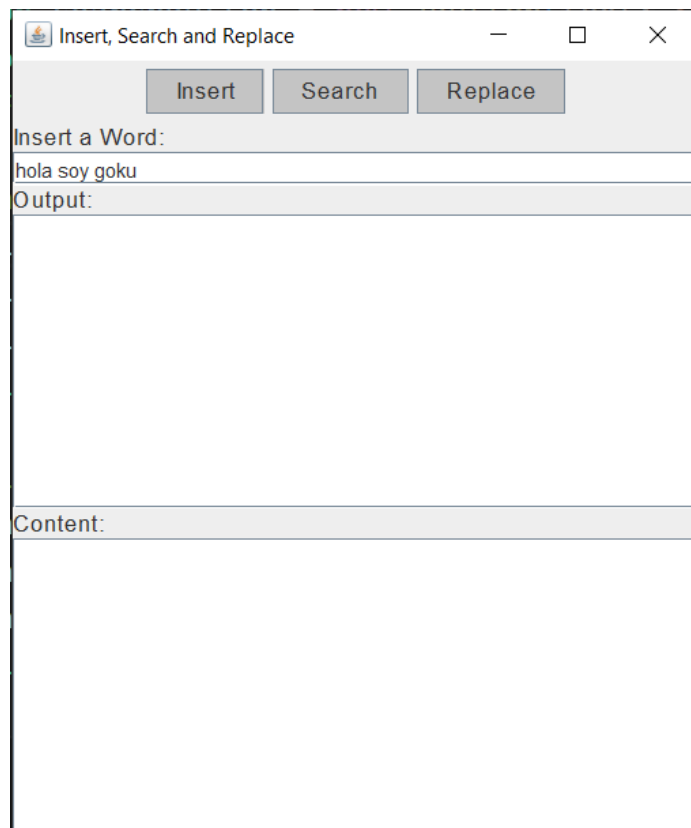
La clase TrieGUI crea una interfaz gráfica para permitir al usuario interactuar con la Trie. La interfaz gráfica contiene los siguientes componentes:

- Etiquetas: Etiquetas para indicar qué acción realizar (insertar, buscar, reemplazar).
- Botones: Botones para realizar las acciones de inserción, búsqueda y reemplazo.
- Cuadros de texto: Campos de texto para ingresar palabras y ver la salida en la consola y el contenido actual en la Trie.

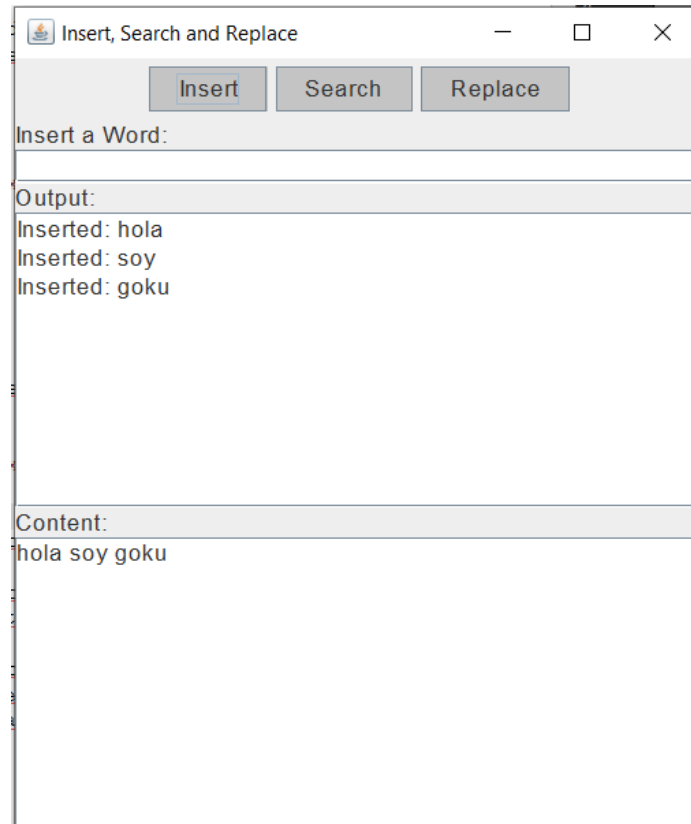


## 2.7. Funcionamiento de la aplicación

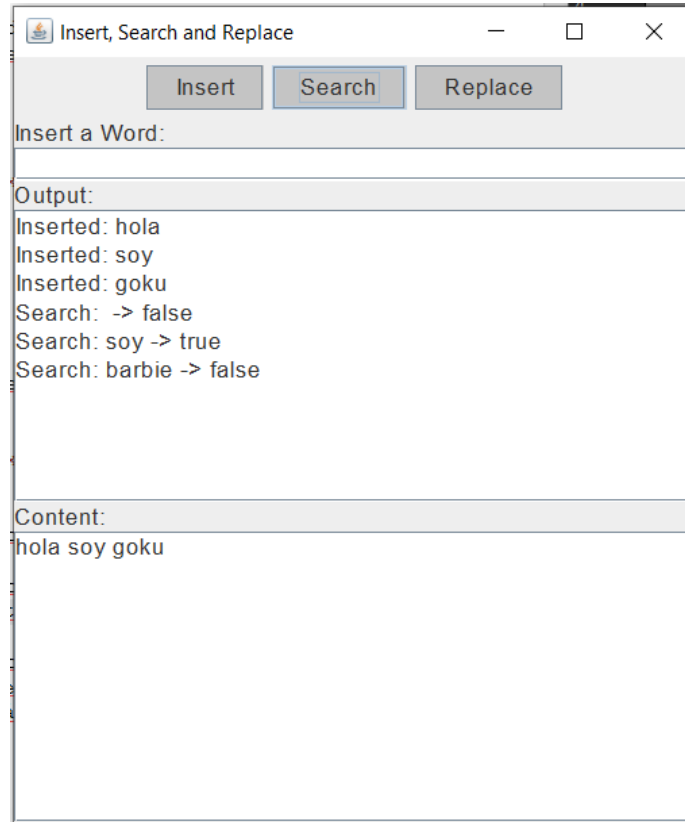
- Al ejecutar la aplicación, se mostrará una ventana con la interfaz gráfica.



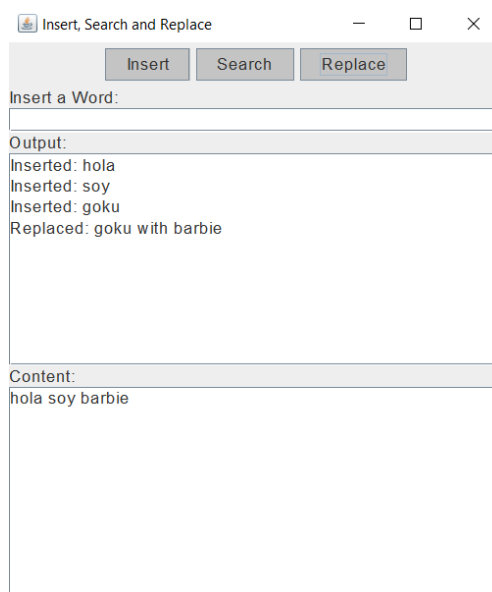
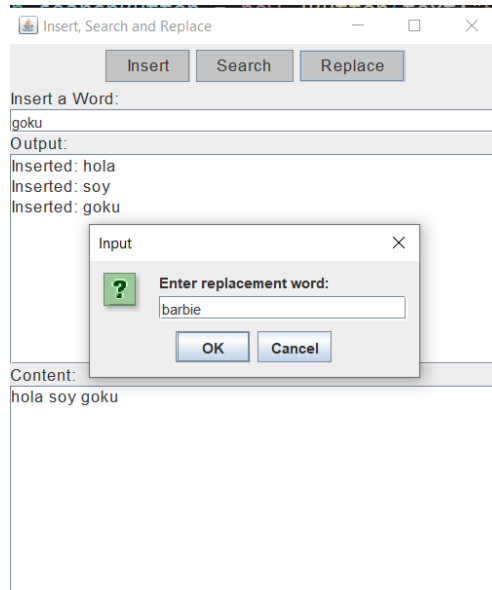
- El usuario puede ingresar una palabra en el campo de texto y luego hacer clic en el botón "Insert" para insertar la palabra en la Trie.



- El usuario puede ingresar una palabra en el campo de texto y luego hacer clic en el botón "Search" para buscar la palabra en la Trie. El resultado de la búsqueda se mostrará en la consola.



- El usuario puede ingresar una palabra en el campo de texto y luego hacer clic en el botón "eplace". Se le pedirá que ingrese una palabra de reemplazo. La aplicación reemplazará la palabra original con la nueva palabra en la Trie y mostrará el resultado actualizado en la consola y el contenido actual en la Trie.



## 2.8. Conclusiones

- En la tarea de implementar un Trie con interfaz gráfica de usuario para insertar, buscar y reemplazar palabras en un texto, hemos visto que los Trie son una estructura de datos altamente eficiente para manejar y manipular un conjunto grande de palabras, especialmente cuando existen prefijos comunes entre ellas.
- Los Trie nos permiten realizar búsquedas de palabras en tiempo lineal en función del tamaño de la palabra y no del tamaño total del conjunto de palabras. Esto los convierte en una excelente opción para implementar aplicaciones como correctores ortográficos, autocompletado de palabras y sistemas de sugerencias de búsqueda, donde la búsqueda rápida y eficiente es esencial.

## 3. Cuestionario

- **Explique. ¿Cómo se utiliza esta estructura de datos para almacenar prefijos?**

En un Trie, cada nodo representa un carácter y cada camino desde la raíz hasta un nodo hoja representa una palabra completa. La idea clave detrás del Trie es que los nodos comparten los mismos prefijos comunes. Esto significa que las palabras que comparten un prefijo comparten nodos en común en el Trie. Utilizando una clase Trie y una clase interna TrieNode. La clase TrieNode representa un nodo en el Trie y contiene un arreglo de referencias a otros nodos, correspondientes a las 26 letras del alfabeto inglés. Cada nodo también tiene un indicador booleano `isWord`, que marca si el nodo representa el final de una palabra.

Cuando se inserta una palabra en el Trie utilizando el método `insert(String word)`, el algoritmo recorre los caracteres de la palabra y crea nuevos nodos si no existen para representar cada carácter. Esto asegura que se construya un camino desde la raíz hasta el nodo que representa el último carácter de la palabra, formando así la palabra completa.

El uso de la estructura Trie para almacenar prefijos permite que las búsquedas de palabras y la comprobación de existencia de palabras sean rápidas. Dado que los prefijos comunes se comparten entre las palabras, la búsqueda puede detenerse cuando se alcanza un punto en el Trie donde ya no hay más caracteres que coincidan con el prefijo buscado.

- **Cómo realizo la funcionalidad de reemplazar texto**

La funcionalidad de reemplazar texto se ha implementado en el método `replace(String word, String replacement)` de la clase Trie. El objetivo es reemplazar una palabra `word` existente en el Trie con una nueva palabra `replacement`.

El método `replace` se divide en tres partes:

Primero, busca la palabra `word` en el Trie utilizando el algoritmo de búsqueda similar al utilizado en el método `search(String word)`. Si encuentra la palabra, procede a la siguiente etapa. Si no encuentra la palabra, se lanza una excepción.

Luego, se recorre la lista `content`, que contiene todas las palabras previamente insertadas en el Trie. Si encuentra la palabra `word` en la lista `content`, la reemplaza con la nueva palabra `replacement`.

Después de actualizar la lista `content`, el método elimina la palabra `word` del Trie utilizando el método `delete(String word)`, asegurándose de que el nodo correspondiente al último carácter de la palabra tenga `isWord` establecido en `false`. Luego, inserta la nueva palabra `replacement` en el Trie utilizando el método `insert(String word)`, creando un nuevo camino para la palabra reemplazada.

## 4. Rúbricas

### 4.1. Entregable Informe

Tabla 1: Tipo de Informe

<b>Informe</b>	
<b>Latex</b>	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.



## 4.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
<b>Puntos</b>	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
<b>2.0</b>	0.5	1.0	1.5	2.0
<b>4.0</b>	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
<b>1. GitHub</b>	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
<b>2. Commits</b>	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4			
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
<b>4. Ejecución</b>	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
<b>5. Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
<b>6. Fechas</b>	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
<b>7. Ortografía</b>	El documento no muestra errores ortográficos.	2	X	2	
<b>8. Madurez</b>	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4			
<b>Total</b>		20		12	

## 5. Referencias

- <https://www.w3schools.com/java/default.asp>
- <https://www.geeksforgeeks.org/insertion-sort/>