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`

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)

```java
public void insert(String word) {
    if (word == null || word.isEmpty()) {
     throw new IllegalArgumentException("Invalid input");
    }

    TrieNode current = root;
    for (char c : word.toCharArray()) {
        int index = getIndex(c);
        if (current.children[index] == null) {
            current.children[index] = new TrieNode();
        }
        current = current.children[index];
    }
    current.isWord = true;
}
```

- 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)

```java
public boolean search(String word) {
    if (word == null || word.isEmpty()) {
        return false;
    }

    TrieNode current = root;
    for (char c : word.toCharArray()) {
        int index = getIndex(c);
        if (current.children[index] == null) {
            return false;
        }
        current = current.children[index];
    }
    return current.isWord;
}
```

- 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)

```java
public void delete(String word) {
    if (word == null || word.isEmpty()) {
     throw new IllegalArgumentException("Invalid input");
    }

    TrieNode current = root;
    for (char c : word.toCharArray()) {
        int index = getIndex(c);
        if (current.children[index] == null) {
           throw new NoSuchElementException("Word not found in the Trie");
        }
        current = current.children[index];
    }
    current.isWord = false;
}
```

- 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)

```java
public void replace(String word, String replacement) {
    if (word == null || word.isEmpty() || replacement == null) {
     throw new IllegalArgumentException("Invalid input");
    }

    if(word.equals(replacement)) {
     throw new IllegalArgumentException("same words");
    }

    for (int i = 0; i < content.size(); i++) {
        if (content.get(i).equals(word)) {
            content.set(i, replacement);
        }
    }

    delete(word); //eliminar del trie
    insert(replacement); //agregar al trie
}
```

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

```java
import java.util.*;
public class Trie {
    private TrieNode root;
    private LinkedList<String> content;

    public Trie() {
        root = new TrieNode(); //root is empty
        content = new LinkedList<>();
    }

    private class TrieNode {
        private TrieNode[] children;
        private boolean isWord;

        public TrieNode() {
            children = new TrieNode[26]; // storing english words - a -> z
            isWord = false;
        }
    }

    private int getIndex(char c) {
        return c - 'a';
    }

    public void insert(String word) {
        if (word == null || word.isEmpty()) {
         throw new IllegalArgumentException("Invalid input");
        }

        TrieNode current = root;
        for (char c : word.toCharArray()) {
            int index = getIndex(c);
            if (current.children[index] == null) {
                current.children[index] = new TrieNode();
            }
            current = current.children[index];
        }
        current.isWord = true;
    }

    public boolean search(String word) {
        if (word == null || word.isEmpty()) {
            return false;
        }

        TrieNode current = root;
        for (char c : word.toCharArray()) {
            int index = getIndex(c);
            if (current.children[index] == null) {
                return false;
```

```
            }
            current = current.children[index];
        }
        return current.isWord;
    }

    public void replace(String word, String replacement) {
        if (word == null || word.isEmpty() || replacement == null) {
         throw new IllegalArgumentException("Invalid input");
        }

        if(word.equals(replacement)) {
         throw new IllegalArgumentException("same words");
        }

        for (int i = 0; i < content.size(); i++) {
            if (content.get(i).equals(word)) {
                content.set(i, replacement);
            }
        }

        delete(word); //eliminar del trie
        insert(replacement); //agregar al trie
    }

    public void delete(String word) {
        if (word == null || word.isEmpty()) {
         throw new IllegalArgumentException("Invalid input");
        }

        TrieNode current = root;
        for (char c : word.toCharArray()) {
            int index = getIndex(c);
            if (current.children[index] == null) {
               throw new NoSuchElementException("Word not found in the Trie");
            }
            current = current.children[index];
        }
        current.isWord = false;
    }

    public String getContent() {
        StringBuilder contentGlobal = new StringBuilder();
        for (String word : content) {
            contentGlobal.append(word).append(" ");
        }
        return contentGlobal.toString();
    }

   public void insertList(String word) {
        content.add(word);
   }


}
```

- 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)

```
    private class TrieNode {
        private TrieNode[] children;
        private boolean isWord;

        public TrieNode() {
            children = new TrieNode[26]; // storing english words - a -> z
            isWord = false;
        }
    }
```

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

Listing 7: TrieGUI.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.*;

public class TrieGUI extends JFrame implements design{

    private Trie trie;
    private JTextArea consoleTextArea;
    private JTextField contentTextField;
    private JTextArea contentTextArea;

    public TrieGUI() {
        trie = new Trie();

        // GUI components
        JLabel insertLabel = new JLabel("Insert a Word:");
        JLabel outputLabel = new JLabel("Output:");
        JLabel contentLabel = new JLabel("Content:");
        JButton insertButton = new JButton("Insert");
        JButton searchButton = new JButton("Search");
        JButton replaceButton = new JButton("Replace");
        JTextField inputField = new JTextField(40);

        // Console to show output
        consoleTextArea = new JTextArea(10, 10);
        consoleTextArea.setEditable(false);
        consoleTextArea.setFont(buttonFont);
        JScrollPane scrollPane = new JScrollPane(consoleTextArea);

     // Console to show output
        contentTextArea = new JTextArea(10, 10);
        contentTextArea.setEditable(false);
        contentTextArea.setFont(buttonFont);
        JScrollPane contentP = new JScrollPane(contentTextArea);

        // 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
                        replacement word:");
88                  try {
89                      trie.replace(wordToReplace, replacementWord);
90                      logToConsole("Replaced: " + wordToReplace + " with " +
                            replacementWord);
91                      logToContent();
92                  } catch (IllegalArgumentException | NoSuchElementException ex) {
```

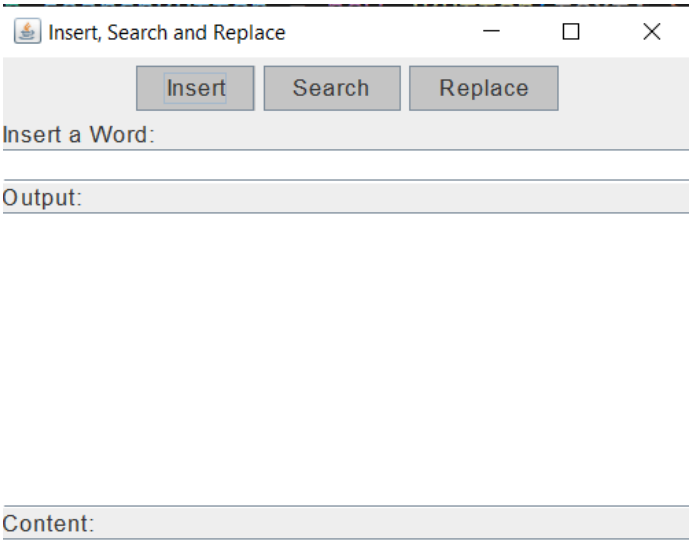
```
93                  logToConsole(ex.getMessage());
94              }
95              inputField.setText("");
96          }
97      });
98
99      // Layout
100     JPanel buttonPanel = new JPanel();
101     buttonPanel.add(insertButton);
102     buttonPanel.add(searchButton);
103     buttonPanel.add(replaceButton);
104
105     JPanel inputPanel = new JPanel(new BorderLayout());
106     inputPanel.add(insertLabel, BorderLayout.NORTH);
107     inputPanel.add(inputField, BorderLayout.CENTER);
108
109     JPanel outputPanel = new JPanel(new BorderLayout());
110     outputPanel.add(outputLabel, BorderLayout.NORTH);
111     outputPanel.add(scrollPane, BorderLayout.CENTER);
112
113     JPanel contentPanel = new JPanel(new BorderLayout());
114     contentPanel.add(contentLabel, BorderLayout.NORTH);
115     contentPanel.add(contentP, BorderLayout.CENTER);
116
117     JPanel global = new JPanel(new BorderLayout());
118     global.add(buttonPanel, BorderLayout.NORTH);
119     global.add(inputPanel, BorderLayout.CENTER);
120     global.add(outputPanel, BorderLayout.SOUTH);
121
122     JPanel mainPanel = new JPanel(new BorderLayout());
123     mainPanel.add(global, BorderLayout.CENTER);
124     mainPanel.add(contentPanel, BorderLayout.SOUTH);
125
126     add(mainPanel);
127
128     setTitle("Insert, Search and Replace");
129     pack();
130     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
131     setLocationRelativeTo(null);
132     setVisible(true);
133 }
134
135 private void logToConsole(String message) {
136     consoleTextArea.append(message + "\n");
137     consoleTextArea.setCaretPosition(consoleTextArea.getDocument().getLength());
138 }
139
140 private void logToContent() {
141     contentTextArea.setText(trie.getContent());
142 }
143
144 public static void main(String[] args) {
145     SwingUtilities.invokeLater(new Runnable() {
146         public void run() {
147             new TrieGUI();
148         }
```

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

2.5. Descripciám general de la clase TrieGUI

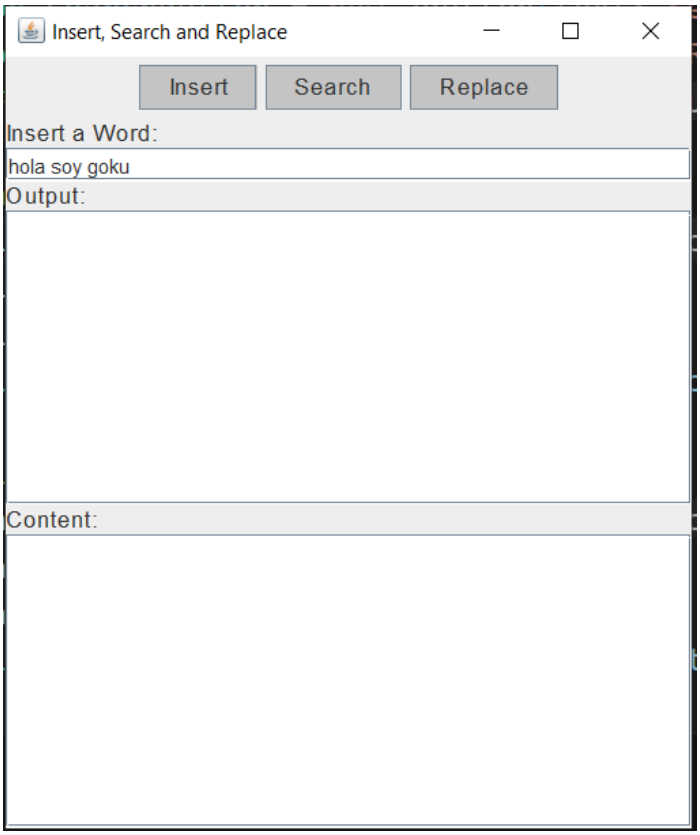
La clase TrieGUI crea una interfaz gráfica para permitir al usuario interactuar con la Trie. La interfaz gráfica contiene los siguientes componentes:

- 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.

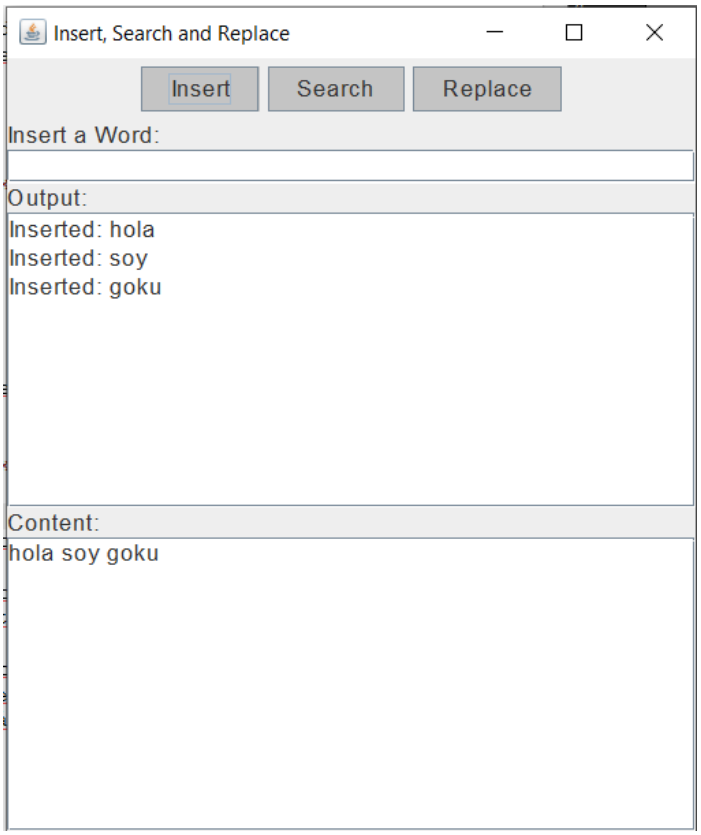


2.6. Funcionamiento de la aplicación

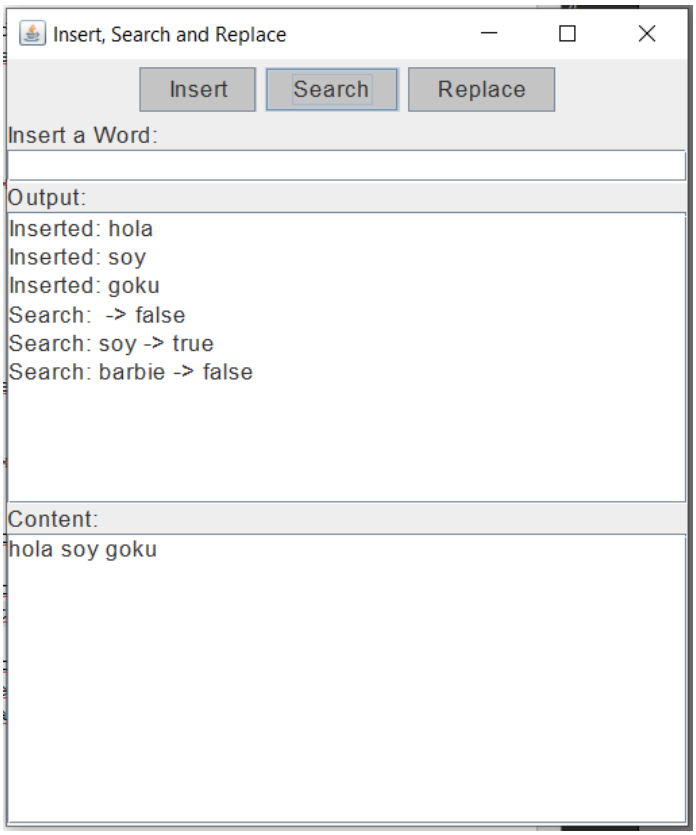
- Al ejecutar la aplicación, se mostrará una ventana con la interfaz gráfica.



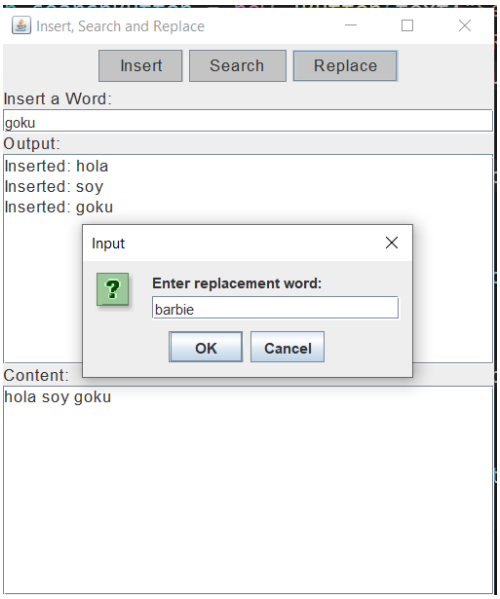
- El usuario puede ingresar una palabra en el campo de texto y luego hacer clic en el botón Ïnsert"para insertar la palabra en la Trie.

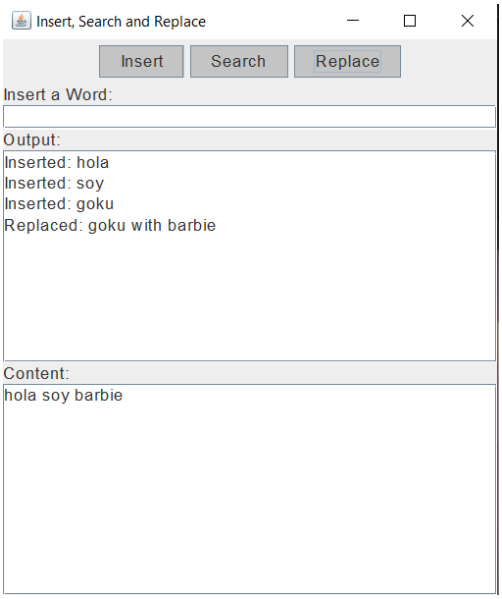


- El usuario puede ingresar una palabra en el campo de texto y luego hacer clic en el botón "Search"para buscar la palabra en la Trie. El resultado de la búsqueda se mostrará en la consola.



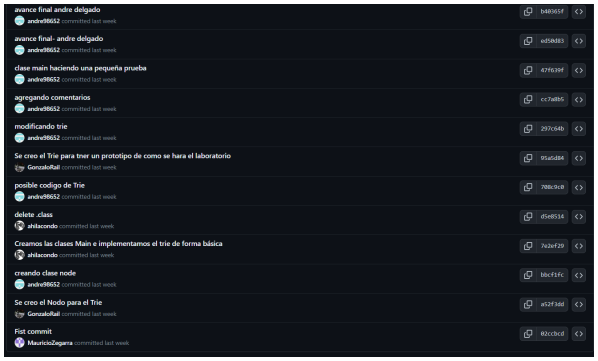
- El usuario puede ingresar una palabra en el campo de texto y luego hacer clic en el botón -eplace". Se le pedirá que ingrese una palabra de reemplazo. La aplicación reemplazará la palabra original con la nueva palabra en la Trie y mostrará el resultado actualizado en la consola y el contenido actual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

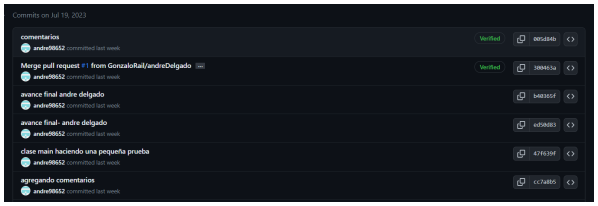


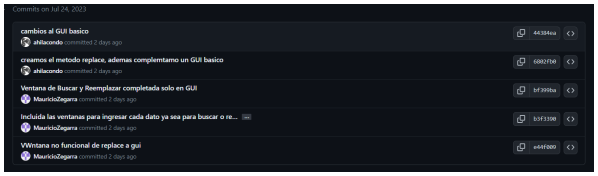


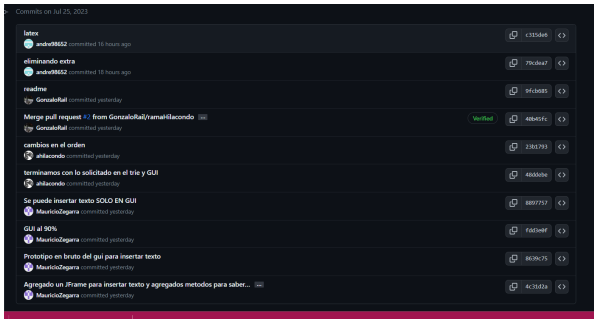
2.7. Commits

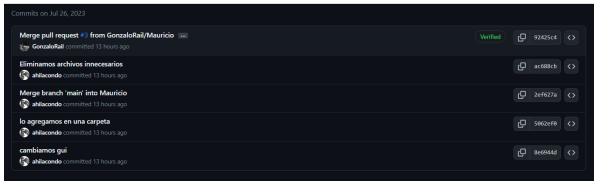
- Commits realizados











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 realizó 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

Informe	
Latex	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 cumplio 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 lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

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

Tabla 3: Rúbrica para contenido del Informe y demostración

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

5. Referencias

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