Informe de Laboratorio 07
Tema: Trie

Nota

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca, Armando Steven Cuno Cahuari vmaldonadov@unsa.edu.pe, acunoc@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Estructura de Datos y Algoritmos Semestre: III Código: 1702122

Tarea	Tema	Duración
07	Trie	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 04/07/24 – 09:42am	Al 04/07/24 – 23:59pm

1. Introducción

Los árboles Trie son estructuras de datos eficientes para almacenar y recuperar cadenas de caracteres, especialmente útiles para operaciones rápidas de búsqueda y autocompletado. En este documento, se explora la implementación de un árbol Trie en Java, enfocándose en su diseño y operaciones fundamentales.

2. Objetivos

- **Implementación del Árbol Trie:** Desarrollar un árbol Trie eficiente en Java, incluyendo métodos para inserción, búsqueda y eliminación de cadenas.
- **Análisis de Complejidad:** Evaluar la complejidad temporal y espacial de las operaciones del árbol Trie implementado.
- **Aplicaciones Prácticas:** Explorar aplicaciones como autocompletado de texto y gestión de listas de palabras clave usando árboles Trie.

3. Tarea

- Elabore un informe paso a paso de la implementación un Trie para insertar, buscar y reemplazar palabras en un texto.
- Encuentra las primeras 'k' palabras que ocurren con mayor frecuencia en un conjunto dado de cadenas(que se insertaron previamente en el Trie)

4. Entregables

- Informe hecho en Latex.
- URL: Repositorio GitHub.
- Archivos Java.

5. Equipos, materiales y temas utilizados

- Trie
- Git
- notepad++
- Latex
- Java

6. URL de Repositorio Github

- Link: GitHub.
- `https://github.com/Victor-Gonzalo-Maldonado-Vilca/EDA_lab07.git`

7. Desarrollo del trabajo

7.1. TrieNode

- **Descripción:** La clase TrieNode representa un nodo en un trie, una estructura de datos utilizada para almacenar cadenas de caracteres. Cada nodo tiene dos atributos: children, que es un mapa que asocia cada carácter a su nodo hijo correspondiente, y frequency, un entero que cuenta cuántas veces una palabra termina en este nodo. El constructor inicializa el mapa de hijos como un HashMap vacío y la frecuencia a 0. Esta clase es fundamental para construir y operar sobre un trie.
- **Código:**

Listing 1: Clase TrieNode

```
import java.util.*;
class TrieNode {
    Map<Character, TrieNode> children;
    int frequency;

    public TrieNode() {
        children = new HashMap<>();
        frequency = 0;
    }
}
```

7.2. Trie

7.2.1. Definición de clase Trie

- **Descripción:** Esta sección define la clase Trie y su constructor. La clase contiene un nodo raíz de tipo TrieNode.
- **Código:**

Listing 2: Clase Trie

```
import java.util.*;
class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }
```

7.2.2. Método Insertar

- **Descripción:** Este método inserta una palabra en el trie. Recorre cada carácter de la palabra y actualiza los nodos correspondientes. Si un carácter no existe, se crea un nuevo nodo.
- **Código:**

Listing 3: Método Insertar

```
public void insert(String word) {
    TrieNode node = root;
    for (char c : word.toCharArray()) {
        if (!node.children.containsKey(c)) {
            node.children.put(c, new TrieNode());
        }
        node = node.children.get(c);
    }
    node.frequency++;
}
```

7.2.3. Método Search

- **Descripción:** Este método busca una palabra en el trie. Devuelve true si la palabra existe (es decir, si su frecuencia es mayor que 0), de lo contrario, devuelve false.
- **Código:**

Listing 4: Método Search

```
public boolean search(String word) {
    TrieNode node = root;
    for (char c : word.toCharArray()) {
        if (!node.children.containsKey(c)) {
            return false;
        }
        node = node.children.get(c);
    }
    return node.frequency > 0;
```

```
    }
```

7.2.4. Método replaceWords

- **Descripción:** Este método reemplaza todas las palabras en un texto que se encuentran en el trie con una palabra de reemplazo dada. Devuelve el texto modificado.
- **Código:**

Listing 5: Método replaceWords

```
    public String replaceWords(String text, String replacement) {
        String[] words = text.split("\\s+");
        StringBuilder result = new StringBuilder();

        for (String word : words) {
            if (search(word)) {
                result.append(replacement).append(" ");
            } else {
                result.append(word).append(" ");
            }
        }

        return result.toString().trim();
    }
```

7.2.5. Método getTopKFrequentWords

- **Descripción:** Este método devuelve una lista con las k palabras más frecuentes en el trie. Utiliza un mapa para almacenar las frecuencias y las ordena para encontrar las palabras más comunes.
- **Código:**

Listing 6: Método getTopKFrequentWords

```
    public List<String> getTopKFrequentWords(int k) {
        // Usamos un mapa para almacenar las frecuencias de las palabras
        Map<String, Integer> wordFrequencyMap = new HashMap<>();
        collectWords(root, "", wordFrequencyMap);

        // Convertimos el mapa a una lista de entradas y ordenamos por frecuencia
        List<Map.Entry<String, Integer>> entries = new
            ArrayList<>(wordFrequencyMap.entrySet());
        entries.sort((a, b) -> b.getValue() - a.getValue());

        // Obtenemos las primeras k palabras mas frecuentes
        List<String> topKWords = new ArrayList<>();
        for (int i = 0; i < k && i < entries.size(); i++) {
            topKWords.add(entries.get(i).getKey());
        }

        return topKWords;
    }
```

7.2.6. Método collectWords

- **Descripción:** Este método auxiliar recolecta todas las palabras en el trie junto con sus frecuencias. Se usa recursivamente para recorrer todos los nodos del trie y actualizar el mapa de frecuencias.
- **Código:**

Listing 7: Método collectWords

```
    private void collectWords(TrieNode node, String prefix, Map<String, Integer>
        wordFrequencyMap) {
        if (node == null) {
            return;
        }
        if (node.frequency > 0) {
            wordFrequencyMap.put(prefix, node.frequency);
        }
        for (Map.Entry<Character, TrieNode> entry : node.children.entrySet()) {
            collectWords(entry.getValue(), prefix + entry.getKey(), wordFrequencyMap);
        }

    }
```

7.3. TopKFrequendtWords

- **Descripción:** El código define una clase TopKFrequentWords que utiliza la estructura de datos Trie para realizar varias operaciones relacionadas con palabras y sus frecuencias. La clase Trie se utiliza para insertar, buscar y reemplazar palabras, así como para obtener las palabras más frecuentes. El método main de la clase TopKFrequentWords se encarga de demostrar estas funcionalidades.
- **Código:**

Listing 8: Clase Principal

```
    import java.util.*;
    public class TopKFrequentWords {
        public static void main(String[] args) {
            Trie trie = new Trie();
            trie.insert("will");
            trie.insert("win");
            trie.insert("wish");
            trie.insert("war");
            trie.insert("war");
            trie.insert("war");
            trie.insert("war");
            trie.insert("want");
            trie.insert("warp");
            trie.insert("warp");
            trie.insert("warp");
            trie.insert("want");
            trie.insert("warp");
            trie.insert("wee");
            trie.insert("wee");
```

```
            System.out.println("Search 'will': " + trie.search("will"));
            System.out.println("Search 'wee': " + trie.search("wee"));
            System.out.println("Search 'hi': " + trie.search("hi"));
            System.out.println("Search 'wanted': " + trie.search("wanted"));

            String text = "hello";
            String replacedText = trie.replaceWords(text, "warp");
            System.out.println("Replaced Text: " + replacedText); // REPLACED
                REPLACED REPLACED hola

            List<String> topWords = trie.getTopKFrequentWords(2);
            System.out.println("Top 2 Frequent Words: " + topWords); // [hello, world]
        }
    }
```

- **Ejecución:**

```
D:\UNSA\EDA\Tree\Ej2\Ej2\programa>javac TopKFrequentWords.java

D:\UNSA\EDA\Tree\Ej2\Ej2\programa>java TopKFrequentWords.java
Search 'will': true
Search 'wee': true
Search 'hi': false
Search 'wanted': false
Replaced Text: hello
Top 2 Frequent Words: [war, warp]
```

Figura 1: Trie – Ejecución

8. Conclusiones

1. **Eficiencia**: Los tries son eficientes para buscar y almacenar cadenas, con operaciones de inserción, búsqueda y eliminación en tiempo proporcional a la longitud de la cadena.
2. **Autocompletado**: Son ideales para autocompletado y sugerencias de palabras, permitiendo búsquedas rápidas de prefijos.
3. **Memoria**: Pueden consumir más memoria debido a múltiples nodos y punteros, pero evitan la duplicación de prefijos comunes.
4. **Frecuencias**: Pueden manejar frecuencias de palabras para análisis estadísticos y encontrar palabras comunes.
5. **Aplicaciones**: Se utilizan en corrección ortográfica, procesamiento de lenguaje natural y compresión de datos.
6. **Flexibilidad**: Su implementación es simple y se puede extender para incluir información adicional, como frecuencias de palabras.

8.1. 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 1: 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 2: 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	X	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	X	4	
Total		20		20	

9. Referencias

- https://www.baeldung.com/trie-java
- https://www.youtube.com/watch?v=fUpZ05dNZdE
- https://www.w3schools.com/java/
- https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-ide-enterprise-java-and-web
- https://docs.oracle.com/javase/7/docs/api/java/util/List.html
- https://docs.oracle.com/javase/tutorial/java/generics/types.html