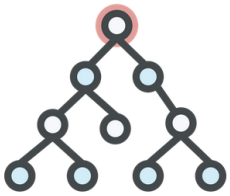
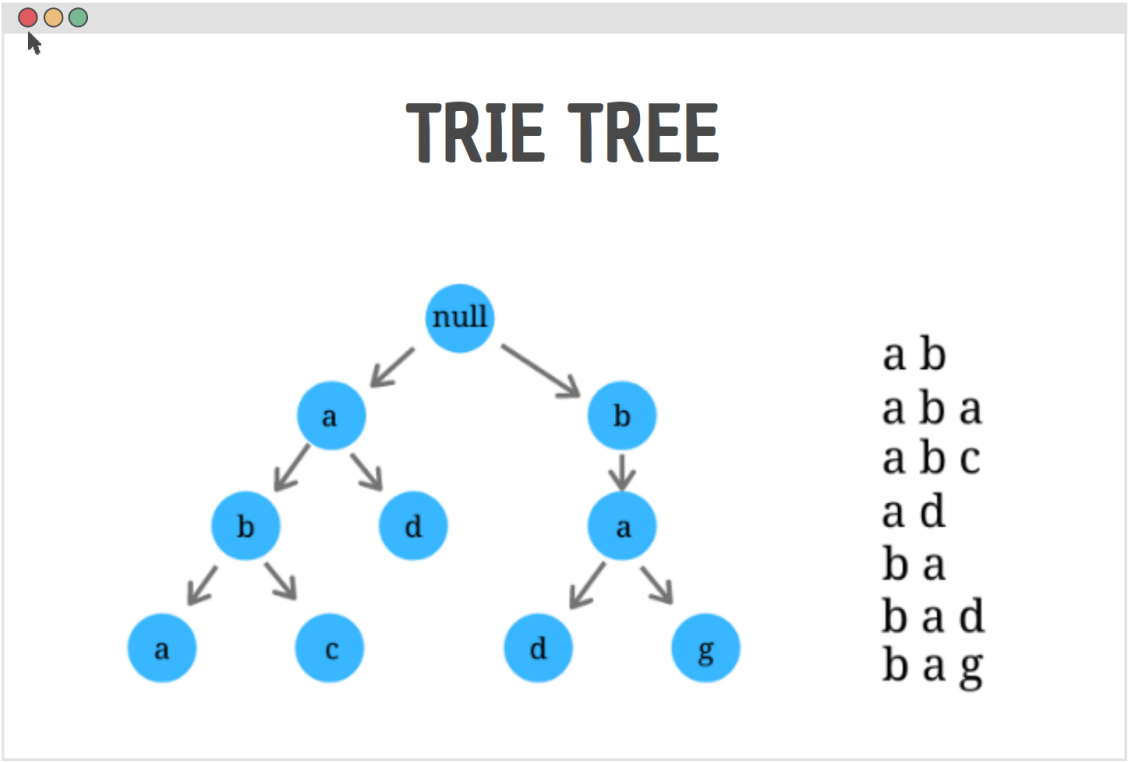


ESTRUCTURAS DE DATOS Y ALGORITMOS con Java



Profesor(a):
Edson Luque Mamani

Estudiantes:
Jorge Luis Mamani Huarsaya
Victor Narciso Mamani Anahua

27 de junio, 2024

Implementación de un Trie Tree en Java

Un Trie es una estructura de datos especializada que se utiliza para almacenar un conjunto de cadenas, generalmente para realizar búsquedas rápidas de palabras. Se detallará el funcionamiento de cada método y la importancia de los campos establecidos en la clase.

Estructura de la Clase

A continuación se presenta la implementación de la clase `Trie`:

```java
import java.util.HashMap;
import java.util.Map;

public class Trie implements ITrie {
  private TrieNode root;

  public Trie() {
    // El nodo raiz que representa el caracter nulo
    root = new TrieNode('\0');
  }

  private static class TrieNode {
    int count = 0;
    char ch;
    Map<Character, TrieNode> children;
    boolean isEndOfWord;

    public TrieNode(char ch) {
      children = new HashMap<>();
      isEndOfWord = false;
      this.ch = ch;
    }
  }
```

Campos de la Clase

- `root`: Es la raíz del Trie, un nodo especial que no representa ningún carácter (`'\0'`).
- `TrieNode`: Clase estática interna que representa un nodo en el Trie.
  - `count`: Cuenta las veces que una palabra termina en este nodo.
  - `ch`: Caracter que representa el nodo.
  - `children`: Mapa de hijos, donde la clave es un carácter y el valor es otro nodo `TrieNode`.
  - `isEndOfWord`: Indica si este nodo es el final de una palabra.

Métodos de la Clase

La clase `Trie` implementa los métodos definidos en la interfaz `ITrie`. A continuación se detallan cada uno de estos métodos.

Inserción de Palabras

```java
@Override
public void insert(String word) {
  TrieNode current = root;
  for (char ch : word.toCharArray())
    current = current.children.computeIfAbsent(ch, c -> new TrieNode(c));

  current.isEndOfWord = true;
  current.count++;
}
```

Este método inserta una palabra en el Trie. Recorre cada carácter de la palabra y lo inserta en el Trie si no está presente. Utiliza el método `computeIfAbsent` de `HashMap` para agregar un nuevo nodo si el carácter no existe en los hijos actuales. Al final de la palabra, marca el nodo como el final de una palabra (`isEndOfWord`) y aumenta el contador (`count`).

Búsqueda de Palabras

```java
@Override
public boolean contains(String word) {
  TrieNode current = root;
  for (char ch : word.toCharArray()) {
    current = current.children.get(ch);
    if (current == null)
      return false;
  }
  return current.isEndOfWord;
}
```

Este método verifica si una palabra está contenida en el Trie. Recorre cada carácter de la palabra y verifica si existe en el Trie utilizando el método `get` de `HashMap`. Si al final de la palabra, el nodo actual es el final de una palabra, retorna `true`, de lo contrario, `false`.

Obtención de Palabras

```java
@Override
public String get(String word) {
  TrieNode current = root;
  for (char ch : word.toCharArray()) {
    current = current.children.get(ch);
```

```
    if (current == null)
      return null;
  }
  return current.isEndOfWord ? word : null;
}
```

Este método retorna la palabra si está contenida en el Trie. Sigue la misma lógica que el método `contains`, pero retorna la palabra en lugar de un valor booleano.

Eliminación de Palabras

```
@Override
public boolean remove(String word) {
  return delete(root, word, 0);
}

private boolean delete(TrieNode current, String word, int index) {
  if (index == word.length()) {
    if (!current.isEndOfWord)
      return false;

    current.isEndOfWord = false;
    return current.children.isEmpty();
  }

  char ch = word.charAt(index);
  TrieNode node = current.children.get(ch);
  if (node == null)
    return false;

  boolean shouldDeleteCurrentNode = delete(node, word, index + 1);
  if (shouldDeleteCurrentNode) {
    current.children.remove(ch);
    return current.children.isEmpty();
  }
  return false;
}
```

El método `remove` elimina una palabra del Trie. Utiliza un método recursivo `delete` que elimina los nodos si ya no son necesarios.

El Método `delete`

El método `delete` es el núcleo del proceso de eliminación. A continuación se explica paso a paso su funcionamiento:

- **Caso Base:** Si el índice ha alcanzado la longitud de la palabra (`index == word.length()`), se verifica si el nodo actual marca el final de una palabra (`current.isEndOfWord`). Si no

es así, significa que la palabra no está en el Trie y se retorna `false`. Si lo es, se desmarca como el final de una palabra y se retorna `true` si el nodo no tiene hijos, indicando que este nodo puede ser eliminado.

- **Recursión:** Se obtiene el siguiente carácter de la palabra y el nodo hijo correspondiente. Si el nodo hijo no existe, se retorna `false`, indicando que la palabra no está en el Trie.
- **Eliminación Condicional:** Se llama recursivamente al método `delete` con el nodo hijo, la palabra y el índice incrementado. Si esta llamada retorna `true`, significa que el nodo hijo puede ser eliminado. Por lo tanto, se elimina el nodo hijo del mapa de hijos del nodo actual y se retorna `true` si el nodo actual no tiene más hijos.

Impresión del Trie

```
1  public void printTrie() {
2    printTrie(root, "", true);
3  }
4
5  private void printTrie(TrieNode node, String prefix, boolean isTail) {
6    if (node != null) {
7      System.out.println(prefix + (isTail ? " " : " ") +
8                         (node.ch != '\0' ? node.ch : "") +
9                         (node.isEndOfWord ? "(" + node.count + ")" : ""));
10
11     int children = node.children.size();
12     int i = 0;
13     for (Map.Entry<Character, TrieNode> entry : node.children.entrySet()) {
14       printTrie(entry.getValue(), prefix + (isTail ? "    " : "   "), ++i ==
         ↪  children);
15     }
16   }
17 }
```

Este método imprime el Trie de manera visual utilizando caracteres especiales. Muestra la jerarquía de los nodos y la cuenta de palabras que terminan en cada nodo.

El Método `printTrie`

El método `printTrie` es una representación visual del Trie. Utiliza caracteres como y para mostrar la estructura jerárquica de los nodos.

- **Lógica de Impresión:** El método recursivo `printTrie` toma un nodo, un prefijo y un booleano que indica si el nodo es la cola (el último hijo) de su padre. Imprime el prefijo seguido del carácter del nodo y, si es el final de una palabra, el contador de esa palabra.
- **Recorrido Recursivo:** Recorre recursivamente todos los hijos del nodo actual, ajustando el prefijo según si el nodo actual es la cola o no, y llamándose a sí mismo para cada hijo.

Prueba del programa

Para poder probar el programa necesitamos usar algún emulador de terminal, en este caso vamos a utilizar **Wezterm**.

```
# Compilar el programa
javac TestTree.java

# Ejecutar el programa
java TestTree
```

Figure 1: Trie Tree