Practical -22

Aim:- . Write a Program to Implement Search, insert, and Remove in Trie

What is a Trie?

A Trie (pronounced as "try") is a tree-like data structure that is used to store a dynamic set of strings, where the keys are usually strings. It is particularly useful for tasks such as autocomplete and spell checking. Each node in a Trie represents a single character of a string, and the path from the root to a node represents a prefix of the string.

Algorithm:-

1. **Insert**:
    - Start from the root node.
    - For each character in the string, check if the character exists in the current node's children.
    - If it does not exist, create a new node for that character.
    - Move to the child node corresponding to the character.
    - After processing all characters, mark the last node as the end of a word.
2. **Search**:
    - Start from the root node.

- For each character in the string, check if the character exists in the current node's children.
- If it does not exist, return false (the word is not in the Trie).
- If all characters are found, check if the last node is marked as the end of a word.

3. **Remove**:
   - Start from the root node and check if the word exists using the search method.
   - If it exists, recursively delete the nodes from the last character to the root.
   - If a node has no children after deletion, remove it from its parent.

Program:-

```
import java.util.HashMap;


class TrieNode {
    HashMap<Character, TrieNode> children;
    boolean isEndOfWord;


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

```
class Trie {
    private TrieNode root;

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

    // Insert a word into the Trie
    public void insert(String word) {
        TrieNode currentNode = root;
        for (char c : word.toCharArray()) {
            currentNode.children.putIfAbsent(c, new TrieNode());
            currentNode = currentNode.children.get(c);
        }
        currentNode.isEndOfWord = true;
    }

    // Search for a word in the Trie
    public boolean search(String word) {
        TrieNode currentNode = root;
        for (char c : word.toCharArray()) {
            if (!currentNode.children.containsKey(c)) {
                return false;
```

```
        }
        currentNode = currentNode.children.get(c);
    }
    return currentNode.isEndOfWord;
}

// Remove a word from the Trie
public boolean remove(String word) {
    return removeHelper(root, word, 0);
}

private boolean removeHelper(TrieNode currentNode, String word,
int index) {
    if (index == word.length()) {
        if (!currentNode.isEndOfWord) {
            return false; // Word not found
        }
        currentNode.isEndOfWord = false; // Unmark the end of word
        return currentNode.children.isEmpty(); // Return true if no
children
    }

    char c = word.charAt(index);
    TrieNode node = currentNode.children.get(c);
```

```
        if (node == null) {
            return false; // Word not found
        }

        boolean shouldDeleteCurrentNode = removeHelper(node, word,
index + 1);

        if (shouldDeleteCurrentNode) {
            currentNode.children.remove(c);
            return currentNode.children.isEmpty() &&
!currentNode.isEndOfWord;
        }
        return false;
    }
}

public class TrieExample {
    public static void main(String[] args) {
        Trie trie = new Trie();
        trie.insert("hello");
        trie.insert("world");

        System.out.println(trie.search("hello")); // true
        System.out.println(trie.search("world")); // true
```

```
        System.out.println(trie.search("hell"));  // false


        trie.remove("hello");
        System.out.println(trie.search("hello")); // false
    }
}
```

Output:-

```
true
true
false
false
```