Tries

1. Count Unique Substring

```java
public class CountUniqueSubstring {
    //Creating a Trie
    static class Node{
        Node children[] = new Node[26];
        boolean eow = false;
        Node(){
            for(int i = 0; i < 26; i++){
                children[i] = null;
            }
        }
    }
    public static Node root = new Node();   // root is empty
    //Insert in Trie - O(L)
    public static void insert(String word){
        Node curr = root;
        for(int level = 0; level < word.length(); level++){
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    public static int countNodes(Node root){
        if (root == null) {
            return 0;
        }
        int count = 0;
        for(int i=0; i<26; i++){
            if (root.children[i] != null) {
                count += countNodes(root.children[i]);
            }
        }
        return count + 1;
    }
    public static void main(String[] args) {
        String str = "ababa";
        //suffix -> insert in trie
        for(int i=0; i<str.length(); i++){
            String suffix = str.substring(i);
            insert(suffix);
        }
        System.out.println("Total no. of Unique Substring of given String: " + countNodes(root));
    }
}
```

2. Group Anagrams Together

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

```java
class TrieNode {
    List<String> data;
    TrieNode[] children;
    boolean isEnd;

    TrieNode() {
        data = new ArrayList<>();
        children = new TrieNode[26];
        isEnd = false;
    }
}

public class GroupAnagramsTogether {
    private TrieNode root;
    private List<List<String>> ans;

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

    public List<List<String>> groupAnagrams(String[] args) {
        ans = new ArrayList<>();
        for (String word : args) {
            build(word);
        }
        dfs(root);
        return ans;
    }

    private void build(String s) {
        TrieNode temp = root;
        char[] word = s.toCharArray();
        Arrays.sort(word); // Sorting the characters of the string
        for (char c : word) {
            if (temp.children[c - 'a'] == null) {
                temp.children[c - 'a'] = new TrieNode();
            }
            temp = temp.children[c - 'a'];
        }
        temp.isEnd = true;
        temp.data.add(s);
    }

    private void dfs(TrieNode rt) {
        if (rt.isEnd) {
            ans.add(rt.data);
        }
        for (int i = 0; i < 26; i++) {
            if (rt.children[i] != null) {
                dfs(rt.children[i]);
            }
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        GroupAnagramsTogether solution = new GroupAnagramsTogether();
        String[] input = {"eat", "tea", "tan", "ate", "nat", "bat"};
        List<List<String>> result = solution.groupAnagrams(input);
        System.out.println(result);
    }
}
```

3. Longest Word in Dictionary

```java
public class LongestWordInDictionary {
    private static class Node {
        private String word;
        private boolean isEnd;
        private Node[] children;

        public Node() {
            this.word = null;
            this.isEnd = false;
            this.children = new Node[26];
        }
    }

    private Node root = new Node();
    private String ans = "";

    private void insert(String word) {
        Node curr = this.root;
        for (int i = 0; i < word.length(); i++) {
            int childIdx = word.charAt(i) - 'a';
            if (curr.children[childIdx] == null) {
                curr.children[childIdx] = new Node();
            }
            curr = curr.children[childIdx];
        }
        curr.isEnd = true;
        curr.word = word;
    }

    public String longestWord(String[] words) {
        for (String word : words) {
            insert(word);
        }
        dfs(root);
        return ans;
    }

    private void dfs(Node node) {
        if (node == null) {
            return;
        }

        if (node.word != null) {
            if (node.word.length() > ans.length() ||
                (node.word.length() == ans.length() && node.word.compareTo(ans) < 0)) {
                ans = node.word;
            }
        }

        for (Node child : node.children) {
            if (child != null && child.word != null) {
                dfs(child);
            }
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        LongestWordInDictionary solution = new LongestWordInDictionary();
        String[] words = {"a", "banana", "app", "appl", "ap", "apply", "apple"};
        //Both "apply" & "apple" can be built from other words in the dictionary. However, "apple" 
is lexicographically smaller than "apply".
        System.out.println(solution.longestWord(words)); // Output: "world"
    }
}
```

4. Longest Word with all Prefixes

```java
public class LongestWordWithAllPrefixes {
    //Creating a Trie
    static class Node{
        Node children[] = new Node[26];
        boolean eow = false;
        Node(){
            for(int i = 0; i < 26; i++){
                children[i] = null;
            }
        }
    }
    public static Node root = new Node();   // root is empty
    //Insert in Trie - O(L)
    public static void insert(String word){
        Node curr = root;
        for(int level = 0; level < word.length(); level++){
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    public static String ans = " ";
    public static void longestWord(Node root, StringBuilder temp){
        if (root == null) {
            return;
        }
        for(int i=0; i<26; i++){            //for(int i=25; i>=0; i--) - Lexicographic large
            // Lexicographic small
            if (root.children[i] != null && root.children[i].eow == true) {
                char ch = (char)(i+'a');
                temp.append(ch);
                if (temp.length() > ans.length()) {
                    ans = temp.toString();
                }
                longestWord(root.children[i], temp);
                temp.deleteCharAt(temp.length() - 1);   //Backtracking
            }
        }
    }
    public static void main(String[] args) {
        String words[] = {"a", "banana", "app", "appl", "ap", "apply", "apple"};
        for(int i=0; i<words.length; i++){
            insert(words[i]);
        }
        longestWord(root, new StringBuilder(" "));
        System.out.println(ans);
    }
}
```

5. Prefix Problem

```java
public class PrefixProblem {
    static class Node{
        Node children[] = new Node[26];
        boolean eow = false;
        int freq;
        public Node(){
            for(int i=0; i<children.length; i++){
                children[i] = null;
            }
            freq = 1;
        }
    }
    public static Node root = new Node();
    public static void insert(String word){
        Node curr = root;
        for(int i=0; i<word.length(); i++){
            int idx = word.charAt(i) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            } else {
                curr.children[idx].freq++;
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    public static void findPrefix(Node root, String ans){
        if (root == null) {
            return;
        }
        if (root.freq == 1) {
            System.out.println(ans);
            return;
        }
        for(int i=0; i<root.children.length; i++){
            if (root.children[i] != null) {
                findPrefix(root.children[i], ans + (char)(i+'a'));
            }
        }
    }
    public static void main(String[] args) {
        String arr[] = {"zebra", "dog", "duck", "dove"};
        for(int i=0; i<arr.length; i++){
            insert(arr[i]);
        }
        root.freq = -1;
        findPrefix(root, " ");  //output will be in alphabetical order and only unique prefixes
    }
}
```

6. Starts with Problem

```java
public class StartswithProblem {
    //Creating a Trie
    static class Node{
        Node children[] = new Node[26];
        boolean eow = false;
        Node(){
            for(int i = 0; i < 26; i++){
                children[i] = null;
            }
        }
    }
    public static Node root = new Node();   // root is empty
    //Insert in Trie - O(L)
    public static void insert(String word){
        Node curr = root;
        for(int level = 0; level < word.length(); level++){
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    public static boolean startsWith(String prefix){
        Node curr = root;
        for(int i=0; i<prefix.length(); i++){
            int idx = prefix.charAt(i)-'a';
            if (curr.children[idx] == null) {
                return false;
            }
            curr = curr.children[idx];
        }
        return true;
    }
    public static void main(String[] args) {
        String words[] = {"apple", "app", "mango", "man", "woman"};
        String prefix1 = "app";
        String prefix2 = "moon";
        for(int i=0; i<words.length; i++){
            insert(words[i]);
        }
        System.out.println(startsWith(prefix1));    //true
        System.out.println(startsWith(prefix2));    //false
    }
}
```

7. Trie Implementation

```
public class TrieImplementation {
    //Creating a Trie
    static class Node{
        Node children[] = new Node[26];
        boolean eow = false;
        Node(){
            for(int i = 0; i < 26; i++){
                children[i] = null;
            }
        }
    }
    public static Node root = new Node();   // root is empty
    //Insert in Trie - O(L)
    public static void insert(String word){
        Node curr = root;
        for(int level = 0; level < word.length(); level++){
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    //Search in Trie - O(L)
    public static boolean search(String key){
        Node curr = root;
        for(int level=0; level < key.length(); level++){
            int idx = key.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                return false;
            }
            curr = curr.children[idx];
        }
        return curr.eow == true;
    }
    public static void main(String[] args) {
        String words[] = {"the", "a", "there", "their", "any", "thee"};
        for(int i=0; i < words.length; i++){
            insert(words[i]);
        }
        System.out.println(search("thee")); //true
        System.out.println(search("thor")); //false
    }
}
```

8. Word Break Problem

```java
public class WordBreakProblem {
    //Creating a Trie
    static class Node{
        Node children[] = new Node[26];
        boolean eow = false;
        Node(){
            for(int i = 0; i < 26; i++){
                children[i] = null;
            }
        }
    }
    public static Node root = new Node();   // root is empty
    //Insert in Trie - O(L)
    public static void insert(String word){
        Node curr = root;
        for(int level = 0; level < word.length(); level++){
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    //Search in Trie - O(L)
    public static boolean search(String key){
        Node curr = root;
        for(int level=0; level < key.length(); level++){
            int idx = key.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                return false;
            }
            curr = curr.children[idx];
        }
        return curr.eow == true;
    }
    public static boolean wordBreak(String key){
        if (key.length() == 0) {
            return true;
        }
        for(int i = 1; i <= key.length(); i++){
            if (search(key.substring(0, i)) && wordBreak(key.substring(i))) {
                return true;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        String arr[] = {"i", "like", "sam", "samsung", "mobile", "ice"};
        for(int i=0; i<arr.length; i++){
            insert(arr[i]);
        }
        String key1 = "ilikesamsung";
        System.out.println(wordBreak(key1));    // Characters - Yes, EndOfWord - Yes
        String key2 = "ilikesam";
        System.out.println(wordBreak(key2));    // Characters - Yes, EndOfWord - Yes
        String key3 = "ilikesung";
        System.out.println(wordBreak(key3));    // Characters - No, EndOfWord - No
        String key4 = "ilik";
        System.out.println(wordBreak(key4));    // Characters - Yes, EndOfWord - No
    }
}
```