Trie

Trie data structure is defined as a Tree based data structure that is used for storing some collection of strings and performing efficient search operations on them. The word Trie is derived from reTRIEval, which means finding something or obtaining it.

Properties of the Trie for a set of the string:

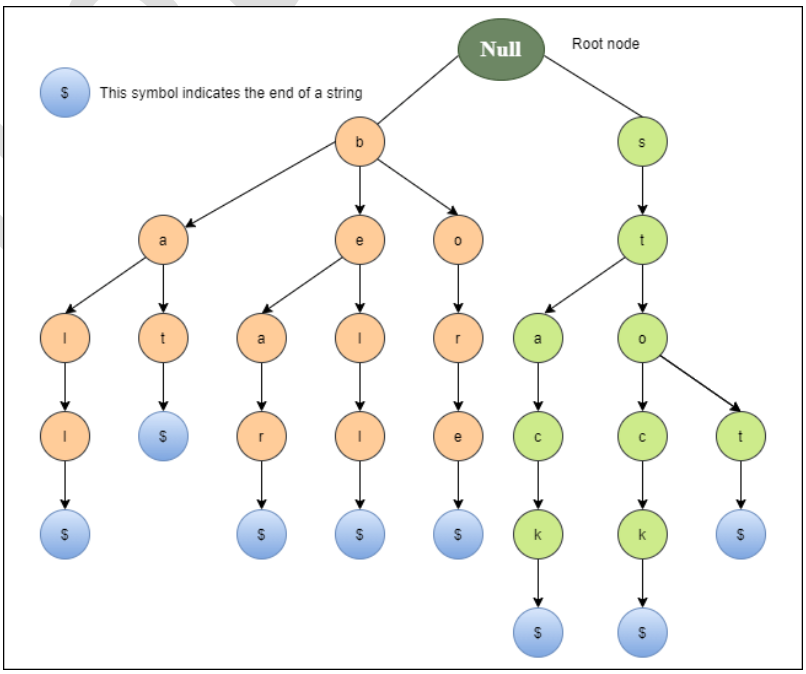
1. The root node of the trie always represents the null node.
2. Each child of nodes is sorted alphabetically.
3. Each node can have a maximum of **26** children (A to Z).
4. Each node (except the root) can store one letter of the alphabet.

The diagram below depicts a trie representation for the bell, bear, bore, bat, ball, stop, stock, and stack.

Basic operations of Trie

There are three operations in the Trie:

1. Insertion of a node
2. Searching a node
3. Deletion of a node



Insertion of string

```java
package Trie;

import Trie.Questions.Searching_element;

public class Trie_Implementation {
    static class Node {
        Node[] children;
        boolean eow;


        public Node() {
            children = new Node[26];
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
            eow = false;
        }
    }

    public static Node root = new Node();
    // insertion
    public static void insert(String word) { //O(n)
        Node curr = root;
        for (int i = 0; i < word.length(); i++) {
            int ind = word.charAt(i) - 'a';

            if (curr.children[ind] == null) {
                curr.children[ind] = new Node();
            }

            if (i == word.length() - 1) {
                curr.children[ind].eow = true;
            }
            curr = curr.children[ind];
        }
    }

    public static boolean search(String key) { //O(n)
        Node curr = root;
        for (int i = 0; i < key.length(); i++) {
            int ind = key.charAt(i) - 'a';
            Node node = curr.children[ind];

            if (node == null) {
                return false;
            }

            if (i == key.length() - 1 && curr.children[ind].eow == false) {
                return false;
            }
            curr = curr.children[ind];
        }
        return true;
    }


    public static void main(String[] args) {
        Trie_Implementation t = new Trie_Implementation();
        String[] word = {"the", "a", "there", "their", "any", "thee"};
        for (int i = 0; i < word.length; i++) {
            insert(word[i]);
            System.out.println("insert : " + word[i]);
        }

        System.out.println(search("there"));
    }
}
```

Searching in trie

```java
package Trie.Questions;

public class Searching_element {
    static class Node {
```

```
        Node[] children;
        boolean eow;


        public Node() {
            children = new Node[26];
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
            eow = false;
        }
    }


    public static Node root = new Node();
    public static void insert(String word) { //O(n)

        Node curr = root;
        for (int i = 0; i < word.length(); i++) {
            int ind = word.charAt(i) - 'a';

            if (curr.children[ind] == null) {
                curr.children[ind] = new Node();
            }

            if (i == word.length() - 1) {
                curr.children[ind].eow = true;
            }
            curr = curr.children[ind];
        }
    }


    // searching
    public static boolean search(String key) { //O(n)
        Node curr = root;
        for (int i = 0; i < key.length(); i++) {
            int ind = key.charAt(i) - 'a';
            Node node = curr.children[ind];

            if (node == null) {
                return false;
            }

            if (i == key.length() - 1 && curr.children[ind].eow == false) {
                return false;
            }
            curr = curr.children[ind];
        }
        return true;
    }


    public static void main(String[] args) {
        String[] word = {"the", "a", "there", "their", "any", "thee"};
        for (int i = 0; i < word.length; i++) {
            insert(word[i]);
//            System.out.println("insert : " + word[i]);
        }
```

```
        System.out.println(search("b"));

    }

}
```

Start with prefix

```
package Trie.Questions;

public class StartWithPrefix {
    static class Node {
        Node[] children;
        boolean eow;


        public Node() {
            children = new Node[26];
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
            eow = false;
        }
    }


    public static Node root = new Node();

    // insertion
    public static void insert(String word) { //O(n)
        Node curr = root;
        for (int i = 0; i < word.length(); i++) {
            int ind = word.charAt(i) - 'a';

            if (curr.children[ind] == null) {
                curr.children[ind] = new Node();
            }

            if (i == word.length() - 1) {
                curr.children[ind].eow = true;
            }
            curr = curr.children[ind];
        }
    }


    public static boolean startWith(String prefix) {
        Node curr = root;
        for (int i = 0; i < prefix.length(); i++) {
            int ind = prefix.charAt(i) - 'a';


            if (curr.children[ind] == null) {
                return false;
            }
            curr = curr.children[ind];
        }

        return true;
    }


    public static void main(String[] args) {
        String[] word = {"apple", "app", "mango", "man", "woman"};
        String prefix = "man";
```

```
        for (int i = 0; i < word.length; i++) {
            insert(word[i]);
        }

        System.out.println(startWith(prefix));
    }
}
```

word-break

```
package Trie.Questions;


public class WordBreak {
    static class Node {
        Node[] children;
        boolean eow;


        public Node() {
            children = new Node[26];
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
            eow = false;
        }
    }


    public static Node root = new Node();

    // insertion
    public static void insert(String word) { //O(n)
        Node curr = root;
        for (int i = 0; i < word.length(); i++) {
            int ind = word.charAt(i) - 'a';

            if (curr.children[ind] == null) {
                curr.children[ind] = new Node();
            }

            if (i == word.length() - 1) {
                curr.children[ind].eow = true;
            }
            curr = curr.children[ind];
        }
    }


    public static boolean wordBreak(String key) {
        if (key.length() == 0) {
            return true;
        }

        for (int i = 1; i <= key.length(); i++) {
            String firstPart = key.substring(0, i);
            String secondPart = key.substring(i);
```

```
            if (search(firstPart) && wordBreak(secondPart)) {
                return true;
            }
        }
        return false;
    }




    public static void main(String[] args) {
        String[] words = {"i", "like", "sam", "samsung", "mobile"};
        for (int i = 0; i < words.length; i++) {
            insert(words[i]);
        }
        String key = "ilikesamsung";

        System.out.println(wordBreak(key));

    }
}
```

Count unique string

```
package Trie.Questions;

public class CountUniqueString {
    static class Node {
        Node[] children;
        boolean eow;


        public Node() {
            children = new Node[26];
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
            eow = false;
        }
    }


    public static Node root = new Node();

    // insertion
    public static void insert(String word) { //O(n)
        Node curr = root;
        for (int i = 0; i < word.length(); i++) {
            int ind = word.charAt(i) - 'a';

            if (curr.children[ind] == null) {
                curr.children[ind] = new Node();
            }

            if (i == word.length() - 1) {
                curr.children[ind].eow = true;
            }
            curr = curr.children[ind];
        }
    }
```

```
//    public static boolean search(String key) { //O(n)
//        Node curr = root;
//        for (int i = 0; i < key.length(); i++) {
//            int ind = key.charAt(i) - 'a';
//            Node node = curr.children[ind];
//
//            if (node == null) {
//                return false;
//            }
//
//            if (i == key.length() - 1 && curr.children[ind].eow == false) {
//                return false;
//            }
//            curr = curr.children[ind];
//        }
//        return true;
//    }


    public static int countNode(Node root) {
        if (root == null) {
            return 0;
        }

        int count = 0;
        for (int i = 0; i < 26; i++) {
            if (root.children[i] != null) {
                count += countNode(root.children[i]);
            }
        }
        return count + 1;
    }


    public static void main(String[] args) {
        String str = "ababa";
        for (int i = 0; i < str.length(); i++) {
            String suffix = str.substring(i);
            insert(suffix);
        }

        System.out.println(countNode(root));
    }

}
```

Longest Word with string

```
package Trie.Questions;

public class Longest_word_with_prefix {
    static class Node {
```

```
        Node[] children;
        boolean eow;


        public Node() {
            children = new Node[26];
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
            eow = false;
        }
    }


    public static Node root = new Node();


    // insertion
    public static void insert(String word) { //O(n)
        Node curr = root;
        for (int i = 0; i < word.length(); i++) {
            int ind = word.charAt(i) - 'a';

            if (curr.children[ind] == null) {
                curr.children[ind] = new Node();
            }

            if (i == word.length() - 1) {
                curr.children[ind].eow = true;
            }
            curr = curr.children[ind];
        }
    }



    public static String ans = " ";
    public static void longestWord(Node root, StringBuilder temp) {
        if (root == null) {
            return;
        }

        for (int i = 0; i < 26; i++) {
            if (root.children[i] != null  && root.children[i].eow == true)
{
                temp.append((char)(i+'a'));
                if (temp.length() > ans.length()) {
                    ans = temp.toString();
                }

                longestWord(root.children[i], temp);
                temp.deleteCharAt(temp.length() - 1);

            }
        }
    }

    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);
    }

}
```