**Tries**

**Java Code**

**Trie : Implementations, Insert & Search**

public class Tries {

static class Node {

Node[] children = new Node[26];

boolean eow;

public Node() {

for (int i=0; i<26; i++) {

children[i] = null;

}

}

}

public static Node root = new Node();

public static void insert(String word) { //O(n)

int level = 0;

int len = word.length();

int idx = 0;

Node curr = root;

for(; level<len; level++) {

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 search(String key) { //O(n)

int level = 0;

int len = key.length();

int idx = 0;

Node curr = root;

for(; level<len; level++) {

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 (String word : words) {

insert(word);

System.out.println("inserted " + word);

}

System.out.println("thee -> " + search("thee"));

System.out.println("thor -> " + search("thor"));

System.out.println(startsWith("the"));

System.out.println(startsWith("thi"));

}

}

**Question 1**

public static boolean wordBreak(String key) {

int len = key.length();

if(len == 0) {

return true;

}

for(int i=1; i<=len; i++) {

if( search(key.substring(0, i)) &&

wordBreak(key.substring(i)) ) {

return true;

}

}

return false;

}

**Question 2**

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;

}

**Question 3**

public static void longestWord(Node root, StringBuilder curr) {

for(int i=0; i<26; i++) {

if(root.children[i] != null && root.children[i].eow == true) {

curr.append((char)(i+'a'));

if(curr.length() > ans.length()) {

ans = curr.toString();

}

longestWord(root.children[i], curr);

curr.deleteCharAt(curr.length()-1);

}

}

}

public static String ans = "";

**Question 4**

public static void buildTrie(String str) {

//insert all suffixes to Trie

root = new Node();

for(int i=0; i<str.length(); i++) {

insert(str.substring(i));

}

}

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 1+count; //extra one for the self node

}