Q1. Word Ladder: (JAVA SOLUTION)

class Solution{

public int ladderLength(String beginWord, String endWord, List<String> wordList)

{

// create set of word list so lookup is faster

HashSet<String> set = new HashSet();

for (String word : wordList)

{

set.add(word);

}

if(!set.contains(endWord)) return 0;

Queue<String> queue = new LinkedList<String>();

queue.add(beginWord);

int level = 0;

while (!queue.isEmpty())

{

int size = queue.size();

// loop over the queue

for (int i = 0; i < size; i++)

{

String cur = queue.remove();

if (cur.equals(endWord))

return level + 1;

for (int j = 0; j < cur.length(); j++)

{

// cur = "hit"

char[] word = cur.toCharArray();

// find words that are 1 edit distance apart of the current element and present in the word list

for(char ch = 'a'; ch <= 'z'; ch++)

{

word[j] = ch;

//for current word hit => ait, hat, hia || bit, hbt, bit and so on..... there will be word.length \* 26 character total combinations

String check = new String(word);

if(!check.equals(cur) && set.contains(check))

{

// formed word != current word && should be present in Set to be qualified

queue.add(check);

// for beginning word hit => hot will be added as that's 1 edit apart and present in Set as well

// remove from Set to avoid dup check as now we have already processed this element

set.remove(check);

}

}

}

}

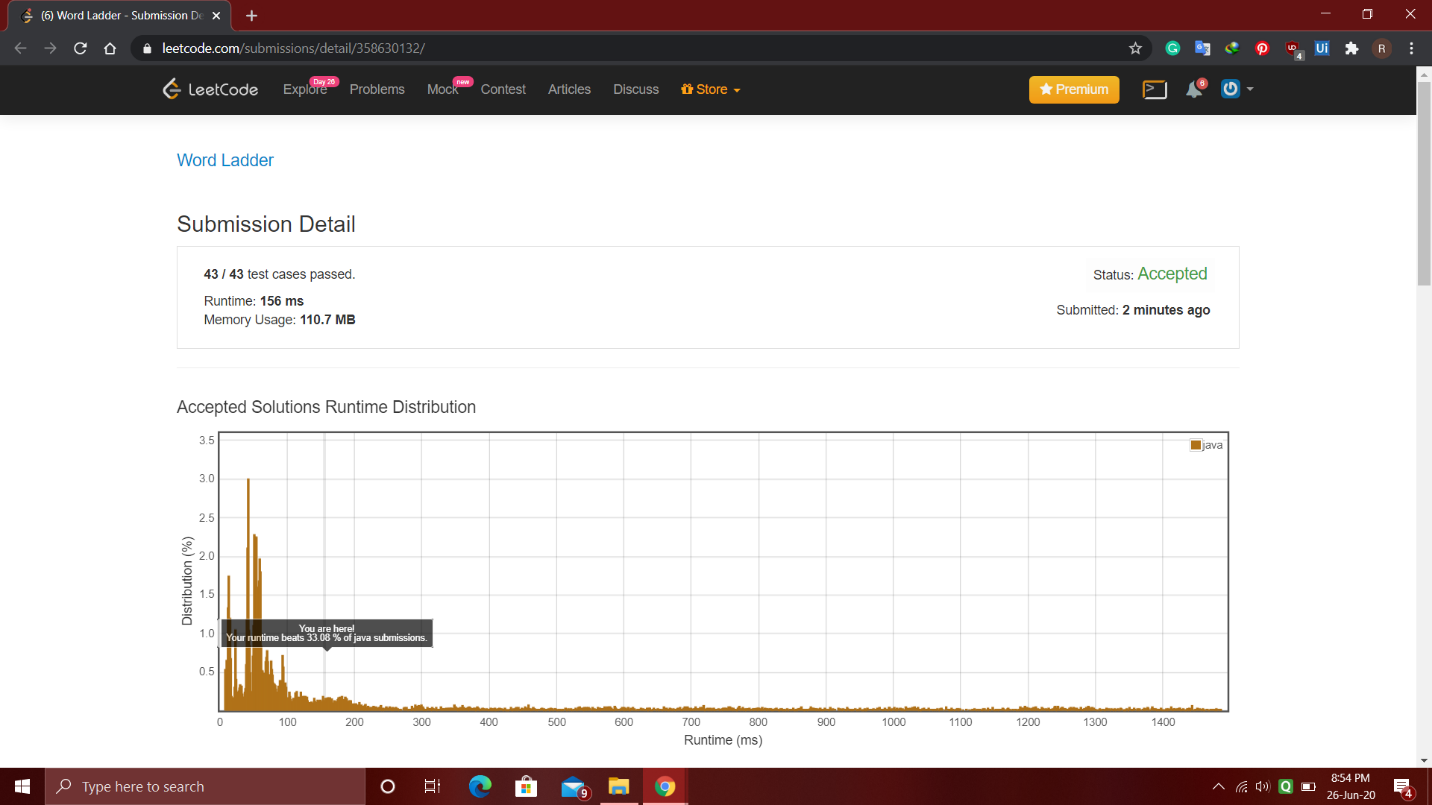
level++;

}

return 0;

}

}



Q2. Tries Problem: (JAVA SOLUTION)

class TrieNode {

TrieNode[] arr;

boolean isEnd;

// Initialize the structure here.

public TrieNode() {

this.arr = new TrieNode[26];

}

}

public class Trie {

private TrieNode root;

public Trie() {

root = new TrieNode();

}

// Inserting word into trie.

public void insert(String word) {

TrieNode p = root;

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

char c = word.charAt(i);

int index = c-'a';

if(p.arr[index]==null){

TrieNode temp = new TrieNode();

p.arr[index]=temp;

p = temp;

}else{

p=p.arr[index];

}

}

p.isEnd=true;

}

// Returns if the word is in the trie.

public boolean search(String word) {

TrieNode p = searchNode(word);

if(p==null){

return false;

}else{

if(p.isEnd)

return true;

}

return false;

}

// Returns if there is any word in the trie

// that starts with the given prefix.

public boolean startsWith(String prefix) {

TrieNode p = searchNode(prefix);

if(p==null){

return false;

}else{

return true;

}

}

public TrieNode searchNode(String s){

TrieNode p = root;

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

char c= s.charAt(i);

int index = c-'a';

if(p.arr[index]!=null){

p = p.arr[index];

}else{

return null;

}

}

if(p==root)

return null;

return p;

}

}

