Solution@28-08-23

652. Find Duplicate Subtrees

```
public class Solution {
    public List<TreeNode> findDuplicateSubtrees(TreeNode root) {
        List<TreeNode> list = new ArrayList<TreeNode>();
        Map<String, Integer> map = new HashMap<String, Integer>();
        dfs(root, map, list);
        return list;
    }

    private String dfs(TreeNode root, Map<String, Integer> map, List<TreeNode> list) {
        if (root == null) {
            return "null";
        }
        String key = String.valueOf(root.val) + "," +
                dfs(root.left, map, list) + "," + dfs(root.right, map, list);
        Integer num = map.get(key);
        if (num != null) {
            if (num == 1) {
                list.add(root);
            }
            num += 1;
        } else {
            num = 1;
        }
        map.put(key, num);
        return key;
    }
}
```

701. Insert into a Binary Search Tree

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */
class Solution {
    public TreeNode insertIntoBST(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        TreeNode current = root;
        while (true) {
            if (current.val < val) {
                if (current.right != null)
                    current = current.right;
                else {
                    current.right = new TreeNode(val);
                    break;
                }
            }
            else {
                if (current.left != null)
                    current = current.left;
                else {
                    current.left = new TreeNode(val);
                    break;
                }
            }
        }

        return root;
    }
}
```

720. longest word in dictionary java

```
class
Solution
{
            public String longestWord(String[] words) {
                Arrays.sort(words);
                String result=new String();
                Set<String> set=new HashSet<>();
                for(String word:words){
                    set.add(word);
                }
                for(String word:words){
                    boolean flag=true;
                    String subWord="";
                    for(int i=0;i<word.length();i++){
                        subWord +=word.charAt(i);
                        if(!set.contains(subWord)) {
```

```
                        flag=false;
                        break;
                    }
                }
                if(flag && word.length()>result.length()){
                    result=word;
                }
            }
            return result;
        }
    }
```

897. increasing order search tree

```
class Solution {
    TreeNode cur;
    public TreeNode increasingBST(TreeNode root) {
        TreeNode ans = new TreeNode(0);
        cur = ans;
        inorder(root);
        return ans.right;
    }

    public void inorder(TreeNode node) {
        if (node == null) return;
        inorder(node.left);
        node.left = null;
        cur.right = node;
        cur = node;
```

```
      inorder(node.right);

    }

}
```

965. Univalued Binary Tree

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
  public boolean isUnivalTree(TreeNode root) {
    if (root == null) {
      // This first check is actually unnecessary to solve the
      // Leetcode problem since it specifies that the tree has at
      // least one node. However, it is useful in the general case
      // to prevent a null pointer exception.
      return true;
    } else {
      return isUnivalTree(root, root.val);
    }
  }
  public boolean isUnivalTree(TreeNode root, int val) {
    return (root == null)
        || (root.val == val
            && isUnivalTree(root.left, val)
            && isUnivalTree(root.right, val));
  }
}
```

1154. Day of the Year

```
class Solution {
    public int dayOfYear(String date) {
        int year = Integer.valueOf(date.substring(0, 4));
        int month = Integer.valueOf(date.substring(5, 7));
        int day = Integer.valueOf(date.substring(8, 10));
        int[] days = {31,28,31,30,31,30,31,31,30,31,30,31};

        // Check for leap year
        if(year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)){
            days[1] = 29;
        }

        int total = 0;
        for(int i = 0; i < month-1; i++){
            total += days[i];
        }
        return total + day;
    }
}
```

1185. Day of the Week

```
class Solution {
    public String dayOfTheWeek(int day, int month, int year) {
        int[] mon = {31, isLeapYear(year) ? 29 : 28, 31, 30,
31,30,31,31,30,31,30,31};
        String[] dayOfweek = {"Sunday", "Monday", "Tuesday",
            "Wednesday", "Thursday", "Friday", "Saturday"};
        int sum = 4;
        //day from the year
        for(int i = 1971; i < year; i++){
            sum += i % 4 == 0 ? 366 : 365;
            //Or
            //if(i % 4 == 0) sum +=366;
            //else sum = sum + 365;
        }
        //day from the month
        for(int i = 0; i < month - 1; i++) {
            isLeapYear(i);
            sum = sum + mon[i];
        }
        sum = sum + day;
        return dayOfweek[sum % 7];
    }

    private static boolean isLeapYear(int year) {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }
}
```

1455 Check if a Word Occurs As a Prefix of Any Word in a Sentence

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Rough {
    public static int isPrefixOfWord(String sentence, String searchWord) {
        String[] words = sentence.split(" ");
        for (int i = 1; i <= words.length; ++i) {
            if (words[i - 1].startsWith(searchWord)) {
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) throws IOException {
        String sentence = "i love eating burger";
        String searchWord = "burg";
        System.out.print(isPrefixOfWord(sentence, searchWord));
    }
}
```

1541. Minimum Insertions to Balance a Parentheses String java

```
class Solution {
    public int minInsertions(String s) {
        int left = 0;
        int ans = 0;
        for(int i=0; i<s.length(); i++){
            if(s.charAt(i)=='(') left++;
            else{
                if(i+1<s.length() && s.charAt(i+1)==')') i+=1;
                else ans++;
                if(left==0) ans++;
                else left--;
            }
        }
        ans += left*2;
        return ans;
    }
}
```

2022. Convert 1D Array Into 2D Array

```
class Solution {
    public int[][] construct2DArray(int[] original, int m, int n) {
        if (m * n != original.length) {
            return new int[0][0];
        }
        int[][] ans = new int[m][n];
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                ans[i][j] = original[i * n + j];
            }
        }
        return ans;
    }
}
```

2063. Vowels of All Substrings

```
class Solution {
    public long countVowels(String word) {
        long ans = 0;
        for (int i = 0, n = word.length(); i < n; ++i) {
            char c = word.charAt(i);
            if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
{
                ans += (i + 1L) * (n - i);
            }
        }
        return ans;
    }
}
```

2239 find closest number to zero

```
package com.fishercoder.solutions;

public class _2239 {
    public static class Solution1 {
        public int findClosestNumber(int[] nums) {
            int ans = nums[0];
            int minDiff = Math.abs(nums[0]);
            for (int i = 1; i < nums.length; i++) {
                int diff = Math.abs(nums[i]);
                if (diff < minDiff) {
                    minDiff = diff;
                    ans = nums[i];
                } else if (diff == minDiff && nums[i] > ans) {
                    ans = nums[i];
            }
          }
          return ans;
        }
    }
}
```

2427 . Number of Common factor

```
package com.fishercoder.solutions;


public class _2427 {
    public static class Solution1 {
        public int commonFactors(int a, int b) {
            int ans = 1;
            int num = 2;
            while (num <= a && num <= b) {
                if (a % num == 0 && b % num == 0) {
                    ans++;
                }
                num++;
            }
            return ans;
        }
    }
}
```