```
========================================
/Users/linxie/Documents/github/leetcodeHub/code/3Sum
========================================


3Sum

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0?
Find all unique triplets in the array which gives the sum of zero.

Note:
Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c)
The solution set must not contain duplicate triplets.
    For example, given array S = {-1 0 1 2 -1 -4},

    A solution set is:
    (-1, 0, 1)
    (-1, -1, 2)
---
```

```java
public class Solution {
    public List<List<Integer>> threeSum(int[] num) {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n<3) return re;

        Arrays.sort(num);
        for(int i=0;i<n-2;i++)
        {
            while(i!=0 && i<n-2 && num[i] == num[i-1]) i++;
            if(i==n-2) break;

            int j = i+1, k = n-1;
            while(j<k)
            {
                while(j!=i+1 && j<k && num[j]==num[j-1]) j++;
                if(j==k) break;
                while(k!=n-1 && j<k && num[k]==num[k+1]) k--;
                if(j==k) break;

                int sum = num[i]+num[j]+num[k];
                if(sum == 0)
                {
                    List<Integer> r = new ArrayList<Integer>();
                    r.add(num[i]); r.add(num[j]); r.add(num[k]);
                    re.add(r);
                    j++; k--;
                }
                else if(sum<0)  j++;
                else k--;
            }
        }
        return re;
    }
}
```

```
---
CLeaner solution:
```

```java
class Solution
{
    public List<List<Integer>> threeSum(int[] num)
    {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n<3) return re;

        Arrays.sort(num);

        for(int i=0;i<n;i++)
        {
            if(i!=0 && num[i-1]==num[i]) continue;

            int j = i+1, k = n-1;
            while(j<k)
            {
                if(j!=i+1 && num[j] == num[j-1]) j++;
                else if(k!=n-1 && num[k] ==num[k+1]) k--;
                else
                {
                    int sum = num[i] + num[j] +num[k];
                    if(sum == 0)
                    {
                        List<Integer> rr = new ArrayList<Integer>();
                        rr.add(num[i]);
                        rr.add(num[j]);
                        rr.add(num[k]);
                        re.add(rr);
                        j++; k--;
                    }
                    else if(sum<0) j++;
                    else k--;
                }
            }
        }
        return re;
    }
}
```

```
========================================
/Users/linxie/Documents/github/leetcodeHub/code/3SumClosest
========================================


3SumClosest

Given an array S of n integers, find three integers in S such that the sum is closest to a
given number, target. Return the sum of the three integers. You may assume that each input
would have exactly one solution.

    For example, given array S = {-1 2 1 -4}, and target = 1.

    The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
---
```

```java
public class Solution {
    public int threeSumClosest(int[] num, int target) {


        int n = num.length;
        if(n==0) return 0;
        if(n==1) return num[0];
        if(n==2)
        {
            int re = num[0];
            re = Math.abs(re-target)<Math.abs(num[1]-target)?re:num[1];
            re = Math.abs(re-target)<Math.abs(num[1]+num[0]-target)?re:(num[0]+num[1]);
            return re;
        }

        int diff = Integer.MAX_VALUE;
        int re = 0;
        Arrays.sort(num);
        for(int i=0;i<n;i++)
        {
            if(i!=0 && num[i-1] == num[i]) continue;
            else
            {
                int j = i+1, k = n-1;
                while(j<k)
                {
                    while(j<k && j!=i+1 && num[j] == num[j-1]) j++;
                    if(j==k) break;

                    while(j<k && k!=n-1 && num[k] ==num[k+1]) k--;
                    if(j==k) break;
                    int sum = num[i]+num[j]+num[k];
                    if( Math.abs(sum-target) < diff)
                    {
                        diff =Math.abs(num[i]+num[j]+num[k]-target);
                        re  = num[i]+num[j]+num[k];
                    }
                    if(sum == target) return target;
                    if(sum<target) j++;
                    else k--;
                }
            }
        }
        return re;


    }
}
```

```
========================================
/Users/linxie/Documents/github/leetcodeHub/code/4Sum
========================================


4Sum

Given an array S of n integers, are there elements a, b, c, and d in S such that
a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:
Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d)
The solution set must not contain duplicate quadruplets.
    For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

    A solution set is:
    (-1,  0, 0, 1)
    (-2, -1, 1, 2)
    (-2,  0, 0, 2)

----
```

```java
public class Solution {
    public List<List<Integer>> fourSum(int[] num, int target) {

        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n<4) return re;

        Arrays.sort(num);
        for(int i=0;i<n;i++)
        {
            if(i!=0 && num[i-1]==num[i]) continue;

            for(int j=i+1; j<n;j++)
            {
                if(j!=i+1 && num[j-1]==num[j]) continue;
```

```
                int k = j+1, v =n-1;
                while(k<v)
                {
                    int res = num[i] + num[j] + num[k]+num[v];
                    if(res==target)
                    {
                        List<Integer> list = new ArrayList<Integer>();
                        list.add(num[i]); list.add(num[j]); list.add(num[k]); list.add(num[v]);
                        re.add(list);
                        k++; v--;
                        while(k<v && num[k]==num[k-1]) k++;
                        if(k==v) break;
                        while(k<v && num[v]==num[v+1]) v--;
                        if(k==v) break;
                    }
                    else if(res>target)
                    {
                        v--;
                        while(k<v && num[v]==num[v+1]) v--;
                        if(k==v) break;
                    }else
                    {
                        k++;
                        while(k<v && num[k]==num[k-1]) k++;
                        if(k==v) break;
                    }
                }
            }
        }
        return re;
    }
}

---
CLeaner solution

public class Solution {
    public List<List<Integer>> fourSum(int[] num, int target) {

        int n = num.length;

        List<List<Integer>> re = new ArrayList<List<Integer>>();
        if(n<4) return re;

        Arrays.sort(num);

        for(int i=0;i<n;i++)
        {
            if(i!=0 && num[i] == num[i-1]) continue;

            for(int j = i+1; j<n; j++)
            {
                if(j!=i+1 && num[j] == num[j-1]) continue;

                int k = j+1, v = n-1;
                while(k<v)
                {
                    if(k!=j+1 && num[k] == num[k-1]) k++;
                    else if(v!=n-1 && num[v] == num[v+1]) v--;
                    else
                    {
                        int sum = num[i]+num[j]+num[k]+num[v];
                        if(sum==target)
                        {
                            List<Integer> rr  = new ArrayList<Integer>();
                            rr.add(num[i]); rr.add(num[j]); rr.add(num[k]); rr.add(num[v]);
                            re.add(rr);
                            k++; v--;
                        }
                        else if(sum<target) k++;
                        else v--;
                    }
                }
            }
        }
        return re;
    }
}




========================================
/Users/linxie/Documents/github/leetcodeHub/code/AddBinary
========================================


AddBinary

Given two binary strings, return their sum (also a binary string).

For example,
a = "11"
b = "1"
Return "100".
--

public class Solution {
    public String addBinary(String a, String b) {
        int m = a.length();
        int n = b.length();
        if(m==0) return b;
```

```
        if(n==0) return a;

        int carry = 0;
        int i = m-1, j = n-1;
        String re = "";

        while(i>=0 && j>=0)
        {
            int r = carry + (int)(a.charAt(i)-0)+(int)(b.charAt(j)-0);
            carry = r/2;
            re = Integer.toString(r%2) + re;
            i--; j--;
        }
        while(i>=0)
        {
            int r = carry + (int)(a.charAt(i)-0);
            carry = r/2;
            re = Integer.toString(r%2) + re;
            i--;
        }
        while(j>=0)
        {
            int r = carry + (int)(b.charAt(j)-0);
            carry = r/2;
            re = Integer.toString(r%2) + re;
            j--;
        }
        if(carry==1) re = "1"+re;
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/AddTwoNumbers
========================================


AddTwoNumbers

You are given two linked lists representing two non-negative numbers. The digits are stored
in reverse order and each of their nodes contain a single digit. Add the two numbers and
return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8

---

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

        if(l1==null) return l2;
        if(l2==null) return l1;
        ListNode newHead = null, tail =null;
        int carry = 0;
        while(l1!=null && l2!=null)
        {
            int re = l1.val+l2.val+carry;
            carry = re/10;
            if(carry!=0)
                re %= 10;
            ListNode cur = new ListNode(re);
            if(newHead == null)
            {
                newHead = cur; tail = cur;
            }else
            {
                tail.next = cur;
                tail = cur;
            }
            l1=l1.next; l2=l2.next;
        }

        while(l1!=null)
        {
            int re = l1.val+carry;
            carry = re/10;
            if(carry!=0) re%=10;
            ListNode cur = new ListNode(re);
            tail.next = cur;
            tail = cur;
            l1=l1.next;
        }
        while(l2!=null)
        {
            int re = l2.val+carry;
            carry = re/10;
            if(carry!=0) re%=10;
            ListNode cur = new ListNode(re);
            tail.next = cur;
            tail = cur;
            l2=l2.next;
```

```
        }
        if(carry!=0)
            tail.next = new ListNode(carry);
        return newHead;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/Anagrams
=======================================


Anagrams

Given an array of strings, return all groups of strings that are anagrams.

---
public class Solution {
    public List<String> anagrams(String[] strs) {

        List<String> re = new ArrayList<String>();
        int n = strs.length;
        if(n<=1) return re;

        HashMap<Integer, Integer> map  = new HashMap<Integer, Integer>();
        HashSet<Integer> added = new HashSet<Integer>();
        for(int i = 0; i<n; i++)
        {
            char[] c = strs[i].toCharArray();
            Arrays.sort(c);
            int key = (new String(c)).hashCode();
            if(!map.containsKey(key))
            {
                map.put(key, i);
            }
            else
            {
                if(!added.contains(map.get(key)))
                {
                    added.add(map.get(key));
                    re.add(strs[map.get(key)]);
                }
                re.add(strs[i]);
            }
        }
        return re;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/BalancedBinaryTree
=======================================


BalancedBinaryTree

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which
the depth of the two subtrees of every node never differ by more than 1.

---

/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int helper(TreeNode root)
    {
            if(root == null) return 0;
            int left = helper(root.left);
            if(left == -1) return -1;
            int right = helper(root.right);
            if(right == -1) return -1;

            if(Math.abs(left-right)<2) return Math.max(left, right)+1;
            else return -1;
    }

    public boolean isBalanced(TreeNode root)
    {
        return helper(root)!=-1;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/BestTimetoBuyandSellStock
=======================================


BestTimetoBuyandSellStock

Say you have an array for which the ith element is the price of a given stock on day i.
```

```

If you were only permitted to complete at most one transaction (ie, buy one and sell one share
    of the stock), design an algorithm to find the maximum profit.

---
public class Solution {
    public int maxProfit(int[] prices) {

        int re = 0;
        int n = prices.length;
        if(n<=1) return re;

        int min = Integer.MAX_VALUE;
        for(int i:prices)
        {
            if(i<min) min = i;
            else
            {
                if(i-min > re) re = i-min;
            }
        }
        return re;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/BestTimetoBuyandSellStockII
========================================


BestTimetoBuyandSellStockII

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like
(ie, buy one and sell one share of the stock multiple times). However, you may not engage in
multiple transactions at the same time (ie, you must sell the stock before you buy again).
--
public class Solution {
    public int maxProfit(int[] prices) {

        int n = prices.length;
        if(n<=1) return 0;

        int max = 0;
        for(int i=1;i<n;i++)
        {
            if(prices[i]>prices[i-1])
                max+=prices[i] - prices[i-1];
        }
        return max;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/BestTimetoBuyandSellStockIII
========================================


BestTimetoBuyandSellStockIII

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete at most two transactions.

Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you
 buy again).

---

public class Solution {
    public int maxProfit(int[] prices) {

        int n= prices.length;
        if(n<2) return 0;
        int[] one = new int[n];

        int min = prices[0];
        for(int i=1;i<n;i++)
        {
            if(prices[i]<min)
            {
                one[i] = one[i-1];
                min = prices[i];
            }
            else
            {
                if(prices[i] - min > one[i-1]) one[i] = prices[i] - min;
                else one[i] = one[i-1];
            }
        }

        int max = 0;
        int re = one[n-1];
        for(int i=n-1; i>0; i--)
        {
            if(prices[i]>max) max = prices[i];
            else
            {
```

```
                if(max-prices[i]+one[i-1]>re) re =max-prices[i]+one[i-1];
            }
        }
        return re;


    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeInorderTraversal
=======================================


BinaryTreeInorderTraversal

Given a binary tree, return the inorder traversal of its nodes values.

For example:
Given binary tree {1,#,2,3},
   1
    \
     2
    /
   3
return [1,3,2].

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {


        List<Integer> re = new ArrayList<Integer>();
        if(root == null) return re;

        TreeNode cur = root, tail = null, pre = null;
        root = null;
        while(cur!=null)
        {
            if(cur.left==null)
            {
                if(root == null) { root = cur; tail = cur; cur = cur.right; tail.right = null; }
                else
                {
                    tail.right = cur;
                    cur = cur.right;
                    tail = tail.right;
                    tail.right = null;
                    tail.left = null;
                }
            }
            else
            {
                pre = cur.left;
                while(pre.right!=null && pre.right!=cur) pre = pre.right;
                if(pre.right == null){ pre.right = cur; cur = cur.left; pre.right.left = null; }
                else cur.left = null;
            }
        }
        cur = root;
        while(cur!=null) { re.add(cur.val); cur = cur.right; }
        return re;


    }
}


===

class Solution
{
    public List<Integer> inorderTraversal(TreeNode root)
    {
        List<Integer> re = new ArrayList<Integer>();

        if(root == null) return re;

        TreeNode cur = root, prev = null;

        while(cur!=null)
        {
            if(cur.left==null)
            {
                re.add(cur.val);
                cur = cur.right;
            }
            else
            {
                prev = cur.left;
                while(prev.right != cur && prev.right!=null) prev = prev.right;
                if(prev.right == null)
                {
                    prev.right = cur;
```

```
                    cur = cur.left;
                }
                else
                {
                    re.add(cur.val);
                    prev.right = null;
                    cur = cur.right;
                }
            }
        }
        return re;
    }
}




========================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeLevelOrderTraversal
========================================


BinaryTreeLevelOrderTraversal

Given a binary tree, return the level order traversal of its nodes values. (ie, from left to right,
 level by level).

For example:
Given binary tree {3,9,20,#,#,15,7},
    3
   / \
  9  20
    /  \
   15   7
return its level order traversal as:
[
  [3],
  [9,20],
  [15,7]
]---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {

        List<List<Integer>> re = new ArrayList<List<Integer>>();
        if(root == null) return re;

        ArrayList<TreeNode> q = new ArrayList<TreeNode>();
        q.add(root);

        int start = 0;
        while(start!=q.size())
        {
            int siz = q.size();
            List<Integer> r = new ArrayList<Integer>();
            for(int i=start; i< siz; i++)
            {
                TreeNode cur = q.get(i);
                r.add(cur.val);
                if(cur.left!=null) q.add(cur.left);
                if(cur.right!=null) q.add(cur.right);
            }
            start = siz;
            re.add(r);
        }
        return re;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeLevelOrderTraversalII
========================================


BinaryTreeLevelOrderTraversalII

Given a binary tree, return the bottom-up level order traversal of its nodes values.
(ie, from left to right, level by level from leaf to root).

For example:
Given binary tree {3,9,20,#,#,15,7},
    3
   / \
  9  20
    /  \
   15   7
return its bottom-up level order traversal as:
[
  [15,7],
  [9,20],
  [3]
]
confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ.
```

```

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {

        List<List<Integer>> re = new ArrayList<List<Integer>>();
         if(root == null) return re;

         List<TreeNode> prev = new ArrayList<TreeNode>();
         prev.add(root);
         List<Integer> r = new ArrayList<Integer>();
         r.add(root.val);
         re.add(0, r);
         while(prev.size()!=0)
         {
             List<TreeNode> cur = new ArrayList<TreeNode>();
            r = new ArrayList<Integer>();
             for(TreeNode nd : prev)
             {
                 if(nd.left!=null)
                 {
                     cur.add(nd.left);
                     r.add(nd.left.val);
                 }
                 if(nd.right!=null){
                     cur.add(nd.right);
                     r.add(nd.right.val);
                 }
             }
             prev = cur;
             if(r.size()!=0) re.add(0,r);
         }
        return re;


    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeMaximumPathSum
========================================


BinaryTreeMaximumPathSum

Given a binary tree, find the maximum path sum.

The path may start and end at any node in the tree.

For example:
Given the below binary tree,

       1
      / \
     2   3
Return 6.

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDeep(TreeNode root, HashMap<TreeNode, Integer> deep)
    {
        if(root == null) return 0;
        if(deep.containsKey(root)) return deep.get(root);

        int re = root.val;

        if(root.left != null) re = Math.max(re, root.val+maxDeep(root.left,deep));
        if(root.right != null) re = Math.max(re, root.val+maxDeep(root.right, deep));
        deep.put(root, re);
        return re;
    }

    public int maxPathSum(TreeNode root, HashMap<TreeNode, Integer> deep)
    {
        if(root==null) return Integer.MIN_VALUE;
        int leftPath = maxPathSum(root.left, deep);
        int rightPath = maxPathSum(root.right, deep);
        int path = root.val;
        if(root.left!=null) path = Math.max(path, path+deep.get(root.left));
        if(root.right!=null) path = Math.max(path, path+deep.get(root.right));

        return Math.max(path, Math.max(leftPath, rightPath));

    }

```

```
    public int maxPathSum(TreeNode root)
    {
        if(root == null) return 0;
        HashMap<TreeNode, Integer> deep = new HashMap<TreeNode, Integer>();
        maxDeep(root, deep);
        return maxPathSum(root, deep);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreePostorderTraversal
========================================


BinaryTreePostorderTraversal

Given a binary tree, return the postorder traversal of its nodes values.

For example:
Given binary tree {1,#,2,3},
   1
    \
     2
    /
   3
return [3,2,1]
---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {

    static TreeNode add(TreeNode from, TreeNode to)
    {
        TreeNode cur = from;
        from  = null;

        while(cur!=null)
        {
            TreeNode t = cur.right;
            cur.right = from;
            from = cur;
            cur = t;
        }

        return from;
    }


    public static List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> re = new ArrayList<Integer>();
        if(root == null) return re;

        TreeNode tmp = new TreeNode(0), tail = null;
        tmp.left = root;
        root = null;
        TreeNode  cur = tmp;

        while(cur!=null)
        {

            if(cur.left == null){ cur = cur.right; }
            else
            {
                TreeNode pre = cur.left;

                while(pre.right!=null && pre.right!= cur) pre = pre.right;


                if(pre.right==null)
                {
                    pre.right = cur;
                    cur = cur.left;
                }
                else
                {
                    pre.right = null;
                    TreeNode t = cur.left;
                    cur.left = null;

                    t = add(t, pre);
                    if(root == null) root = t;
                    else    tail.right = t;

                    tail = root; while(tail.right !=null) tail = tail.right;

                    cur = cur.right;
                }
            }
        }
        for(TreeNode cc = root; cc!=null; cc = cc.right)
        {
            re.add(cc.val);
        }
        return re;
    }
}
```

```
---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> re = new ArrayList<Integer>();
        if(root == null) return re;

        Stack<TreeNode> s = new Stack<TreeNode>();
        s.push(root);
        TreeNode prev= null;
        while(!s.isEmpty())
        {
            TreeNode nd = s.peek();
            if(prev == null || prev.left == nd || prev.right == nd)
            {
                if(nd.left!=null) s.push(nd.left);
                else if(nd.right!=null) s.push(nd.right);
            }
            else
            {
                if(nd.left == prev && nd.right!=null) s.push(nd.right);
                else
                {
                    re.add(nd.val);
                    s.pop();
                }
            }
            prev = nd;
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreePreorderTraversal
========================================


BinaryTreePreorderTraversal

Given a binary tree, return the preorder traversal of its nodes values.

For example:
Given binary tree {1,#,2,3},
   1
    \
     2
    /
   3
return [1,2,3].
---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {

        List<Integer> re =new ArrayList<Integer>();

        if(root == null) return re;

        while(root!=null)
        {
            if(root.left == null){ re.add(root.val); root = root.right; continue; }

            TreeNode pre = null;
            for(pre = root.left; pre.right!=null && pre.right!=root.right; pre=pre.right);

            if(pre.right == null){ pre.right = root.right; re.add(root.val); root = root.left;}
            else
            {
                pre.right = null;  root = root.right;
            }
        }
        return re;

    }
}



========================================
/Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeZigzagLevelOrderTraversal
========================================


BinaryTreeZigzagLevelOrderTraversal

```

Given a binary tree, return the zigzag level order traversal of its nodes values.
(ie, from left to right, then right to left for the next level and alternate between).

For example:
Given binary tree {3,9,20,#,#,15,7},
```
    3
   / \
  9  20
    /  \
   15   7
```
return its zigzag level order traversal as:
```
[
  [3],
  [20,9],
  [15,7]
]
```

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

        List<List<Integer>> re = new ArrayList<List<Integer>>();
        if(root==null) return re;

        ArrayList<TreeNode> q = new ArrayList<TreeNode>();
        q.add(root);
        boolean flg = false;

        int start = 0;
        int end = q.size();
        while(start<end)
        {
            List<Integer> level = new ArrayList<Integer>();
            for(int i=start; i<end; i++)
            {
                if(!flg) level.add(q.get(i).val);
                else level.add(0,q.get(i).val);
                if(q.get(i).left!=null) q.add(q.get(i).left);
                if(q.get(i).right!=null) q.add(q.get(i).right);
            }
            start = end;
            end = q.size();
            flg = !flg;
            re.add(level);
        }
        return re;

    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/Candy
==========

Candy

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.
Children with a higher rating get more candies than their neighbors.
What is the minimum candies you must give?

```
public class Solution {
    public int candy(int[] ratings) {
        int n = ratings.length;
        if(n==0) return 0;
        if(n==1) return 1;

        int[] candy = new int[n];
        candy[0] = 1;

        for(int i=1;i<n;i++)
        {
            if(ratings[i]>ratings[i-1]) candy[i] = candy[i-1]+1;
            else candy[i] = 1;
        }
        int total = candy[n-1];
        for(int i = n-2;i>=0;i--)
        {
            if(ratings[i]>ratings[i+1] && candy[i]<=candy[i+1])
            {
                candy[i] = candy[i+1]+1;
            }
            total += candy[i];
        }
        return total;
    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/ClimbingStairs
==========

```
ClimbingStairs

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

---
public class Solution {
    public int climbStairs(int n) {
        if(n<=1) return 1;

        int first = 1, second = 1;
        int re = -1;
        for(int i=2;i<=n;i++)
        {
            re = first + second;
            first = second;
            second = re;
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/CloneGraph
========================================


CloneGraph

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.


OJs undirected graph serialization:
Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.
As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes, and therefore contains three parts as separated by #.

First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
Second node is labeled as 1. Connect node 1 to node 2.
Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.
Visually, the graph looks like the following:

       1
      / \
     /   \
    0 --- 2
         / \
         \_/
---
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     List<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new ArrayList<UndirectedGraphNode>(); }
 * };
 */
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node, HashMap<UndirectedGraphNode, UndirectedGraphNode> map)
    {
        if(node == null) return null;
        if(map.containsKey(node)) return map.get(node);

        UndirectedGraphNode nd = new UndirectedGraphNode(node.label);
        map.put(node, nd);
        for(UndirectedGraphNode N : node.neighbors)
        {
            nd.neighbors.add(cloneGraph(N,map));
        }
        return nd;
    }

    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node)
    {
        return cloneGraph(node, new HashMap<UndirectedGraphNode, UndirectedGraphNode>());
    }

}


----
public class Solution {


    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node)
    {

        if(node == null) return null;

        HashMap<UndirectedGraphNode, UndirectedGraphNode> map =
            new HashMap<UndirectedGraphNode, UndirectedGraphNode>();

        LinkedList<UndirectedGraphNode> q = new LinkedList<UndirectedGraphNode>();

        q.addLast(node);
        map.put(node, new UndirectedGraphNode(node.label));

```

```
        while(!q.isEmpty())
        {
            UndirectedGraphNode cur = q.removeFirst();
            for(UndirectedGraphNode l:cur.neighbors)
            {
                if(!map.containsKey(l))
                {
                    UndirectedGraphNode neighbor = new UndirectedGraphNode(l.label);
                    map.get(cur).neighbors.add(neighbor);
                    map.put(l,neighbor);
                    q.addLast(l);
                }
                else
                {
                    map.get(cur).neighbors.add(map.get(l));
                }
            }
        }
        return map.get(node);
     }
}

---

public class Solution {


    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node)
    {

        if(node == null) return null;

        HashMap<UndirectedGraphNode, UndirectedGraphNode> map =
            new HashMap<UndirectedGraphNode, UndirectedGraphNode>();

        Stack<UndirectedGraphNode> q = new Stack<UndirectedGraphNode>();

        UndirectedGraphNode cp = new UndirectedGraphNode(node.label);
        map.put(node, cp);

        q.push(node);
        while(!q.isEmpty())
        {
            UndirectedGraphNode cur = q.pop();

            for(UndirectedGraphNode l:cur.neighbors)
            {
                if(!map.containsKey(l))
                {
                    UndirectedGraphNode ll = new UndirectedGraphNode(l.label);
                    map.put(l,ll);
                    map.get(cur).neighbors.add(ll);
                    q.push(l);
                }
                else
                {
                    map.get(cur).neighbors.add(map.get(l));
                }
            }
        }

        return map.get(node);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/Combinations
========================================


Combinations

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

For example,
If n = 4 and k = 2, a solution is:

[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
---

public class Solution {
    public List<List<Integer>> combine(int n, int k) {
        List< List<Integer> > re = new ArrayList<List<Integer>>();
        if(n<k || k==0) return re;

        if(n==k)
        {
            List<Integer> r = new ArrayList<Integer>();
            for(int i=1; i<= n; i++)
                r.add(i);
            re.add(r);
            return re;
        }
            List<List<Integer>> rr = combine(n-1, k-1);
            if(rr.size() == 0)
            {
```

```
                List<Integer> r = new ArrayList<Integer>();
                r.add(n);
                re.add(r);
            }
            for(List<Integer> iter : rr)
            {
                iter.add(n);
                re.add(iter);
            }
            rr = combine(n-1,k);
            for(List<Integer> iter:rr)
                re.add(iter);

        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/CombinationSum
========================================


CombinationSum

Given a set of candidate numbers (C) and a target number (T), find all unique combinations
in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:
All numbers (including target) will be positive integers.
Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
The solution set must not contain duplicate combinations.
For example, given candidate set 2,3,6,7 and target 7,
A solution set is:
[7]
[2, 2, 3]

---

public class Solution {
     public static void combinationSum(int[] candidates, int ind, int target, List<List<Integer>> re, List<Integer>r)
    {

        if(target == 0)
        {
            re.add(new ArrayList<Integer>(r));
        }
        if(target <=0) return;

        if(ind>=candidates.length) return;

        int val = candidates[ind];

        combinationSum(candidates, ind+1, target, re, r);

        int siz = r.size();
        while(target>=0)
        {
            r.add(val);
            target -= val;
            combinationSum(candidates, ind+1, target, re, r);
        }
        if(r.size() > siz)
        {
            for(int i = r.size()-1; i >= siz; i--)
                r.remove(i);
        }
    }

    public static List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = candidates.length;
        if(n==0) return re;

        if(target == 0)
            re.add(new ArrayList<Integer>());
        Arrays.sort(candidates);

           combinationSum(candidates, 0, target, re, new ArrayList<Integer>());
           return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/CombinationSumII
========================================


CombinationSumII

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations
in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:
All numbers (including target) will be positive integers.
Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
The solution set must not contain duplicate combinations.
For example, given candidate set 10,1,2,7,6,1,5 and target 8,
A solution set is:
```

```
[1, 7]
[1, 2, 5]
[2, 6]
[1, 1, 6]
----

public class Solution {
    public static void combinationSum2(List<List<Integer>> re, List<Integer> r, int[] num, int ind, int target)
    {
        if(target == 0 && r.size()!=0)
        {
            List<Integer> rr = new ArrayList<Integer>(r);
            re.add(rr);
            return;
        }
        if(ind >= num.length || target < num[ind]) return;

        int n = ind;
        int val = num[ind];
        for(;n<num.length && num[n] == val;n++);
        int sum = 0;
        int siz = r.size();

        combinationSum2(re, r, num, n, target);
        for(int i=ind; i<n;i++)
        {
            sum+=val;
            r.add(val);
            combinationSum2(re, r, num, n, target-sum);
        }
        for(int i=r.size()-1; i>=siz; i--)
            r.remove(r.size()-1);
    }
    public static  List<List<Integer>> combinationSum2(int[] num, int target) {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n==0) return re;

        List<Integer> r = new ArrayList<Integer>();
        Arrays.sort(num);

        combinationSum2(re, r, num, 0, target);
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/ConstructBinaryTreefromInorderandPostorderTraversa
========================================


ConstructBinaryTreefromInorderandPostorderTraversal

Given inorder and postorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode buildTree(int[] inorder, int inLeft, int inRight, int[] postorder, int postLeft,
        int postRight)
    {
        if(inLeft <0 || inRight >= inorder.length || postLeft <0 || inLeft > inRight || (inRight- inLeft != postRight -postLeft)) re

        int val = postorder[postRight];
        int i = inLeft;
        for(;i<=inRight;i++)
        {
            if(inorder[i] == val) break;
        }
        if(i>inRight) return null;

        TreeNode root = new TreeNode(val);
        root.left = buildTree(inorder, inLeft, i-1, postorder, postLeft, i-1-inLeft+postLeft);
        root.right = buildTree(inorder, i+1, inRight, postorder, postRight+i-inRight, postRight-1);
        return root;
    }

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        int n = inorder.length;
        if(n==0 || n!= postorder.length) return null;
        return buildTree(inorder, 0, n-1, postorder, 0, n-1);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/ConstructBinaryTreefromPreorderandInorderTraversal
========================================


ConstructBinaryTreefromPreorderandInorderTraversal

```

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {

    int find(int[] arr, int l, int r, int val)
    {
        for(int i=r; i>=l; i--)
            if(arr[i] == val)
                return i;
        return -1;
    }

    public TreeNode buildTree(int[] preorder, int[] inorder, int s1, int e1, int s2, int e2)
    {
        if(s1>e1 || s2>e2 || (e1-s1) != (e2-s2) ) return null;

        TreeNode nd = new TreeNode(preorder[s1]);
        int mid = find(inorder, s2, e2, preorder[s1]);
        if(mid == -1) return null;

        nd.left = buildTree(preorder, inorder, s1+1, s1+(mid-s2), s2, mid-1);
        nd.right = buildTree(preorder, inorder, e1-e2+mid+1, e1, mid+1, e2);
        return nd;
    }

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        int n = inorder.length;
        if(n!=preorder.length) return null;

        return buildTree(preorder, inorder, 0, n-1, 0, n-1);
    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/ContainerWithMostWater
==========

ContainerWithMostWater

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai).
n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two
lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container.

```
public class Solution {
    public int maxArea(int[] height) {

        int n = height.length;
        if(n<=1) return 0;

        int l = 0, r = n-1;
        int max = Integer.MIN_VALUE;
        while(l<r)
        {
            int local = Math.min(height[l], height[r])*(r-l);
            if(local> max) max = local;

            if(height[l]<=height[r]) l++;
            else r--;
        }
        return max;

    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/ConvertSortedArraytoBinarySearchTree
==========

ConvertSortedArraytoBinarySearchTree

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
```

```
    public TreeNode sortedArrayToBST(int[] num, int l, int r) {
         int n = r-l+1;
         if(n<=0) return null;
         if(n==1) return new TreeNode(num[l]);

         int mid = n/2+l;
         TreeNode root = new TreeNode(num[mid]);
         root.left = sortedArrayToBST(num, l, mid-1);
         root.right = sortedArrayToBST(num, mid+1, r);
         return root;
    }

    public TreeNode sortedArrayToBST(int[] num) {
        return sortedArrayToBST(num, 0, num.length-1);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/ConvertSortedListtoBinarySearchTree
========================================


ConvertSortedListtoBinarySearchTree

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

---
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; next = null; }
 * }
 */
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode sortedListToBST(ListNode head) {

        if(head == null) return null;
        if(head.next == null) return new TreeNode(head.val);

        ListNode first = head, second = head, prev = null;
        int count = 0;

        while(first.next !=null)
        {
            count++;
            first = first.next;
            if(count%2 ==0){
                prev = second;
                second = second.next;
            }
        }
        if(count%2 == 0)
        {
            if(prev == null)
            {
                TreeNode root = new TreeNode(first.val);
                root.left = new TreeNode(second.val);
                return root;
            }
            prev.next = null;
            TreeNode root = new TreeNode(second.val);
            root.left    = sortedListToBST(head);
            ListNode t = second.next;
            second.next = null;
            root.right = sortedListToBST(t);
            return root;
        }
        else
        {
            ListNode t = second.next;
            second.next = null;
            TreeNode root = new TreeNode(t.val);
            root.left = sortedListToBST(head);
            root.right = sortedListToBST(t.next);
            return root;
        }
  }
}


=====

Better solution, linear:

//Node is a wrapper class, since java does not support reference.
//Head can not move forward by passing in ListNode.


class Node
{
    ListNode head;
    Node(ListNode _head)
```

```
    {
        head = _head;
    }
}

public class Solution {
    public TreeNode helper(Node head, int start, int end)
    {
        if(start>end) return null;
        int mid = start+(end-start)/2;
        TreeNode left = helper(head, start, mid-1);
        TreeNode cur = new TreeNode(head.head.val);
        cur.left = left;
        head.head = head.head.next;
        cur.right = helper(head, mid+1, end);
        return cur;
    }
    public TreeNode sortedListToBST(ListNode head)
    {
        if(head == null) return null;

        int count = 0;
        ListNode cur = head;
        while(cur.next !=null)
        {
            count++;
            cur = cur.next;
        }
        return helper(new Node(head), 0 ,count);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/CopyListwithRandomPointer
========================================


CopyListwithRandomPointer

A linked list is given such that each node contains an additional random pointer which could
point to any node in the list or null.

Return a deep copy of the list.

---
/**
 * Definition for singly-linked list with a random pointer.
 * class RandomListNode {
 *     int label;
 *     RandomListNode next, random;
 *     RandomListNode(int x) { this.label = x; }
 * };
 */
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head) {

         if(head == null) return null;

        RandomListNode cur = head;
        while(cur!=null)
        {
            RandomListNode t = cur.next;
            cur.next = new RandomListNode(cur.label);
            cur.next.next = t;
            cur = t;
        }

        cur = head;
        while(cur != null)
        {
            if(cur.next == null) System.exit(-1);
            if(cur.random == null) cur.next.random = null;
            else cur.next.random = cur.random.next;
            cur = cur.next.next;
        }
        cur = head;
        RandomListNode newHead = cur.next;

        while(cur!=null)
        {
            RandomListNode t = cur.next;
            cur.next = t.next;
            if(cur.next !=null) t.next = cur.next.next;
            else t.next = null;
            cur = cur.next;
        }
        return newHead;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/CountandSay
========================================


CountandSay

The count-and-say sequence is the sequence of integers beginning as follows:
1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.
```

11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.
Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string

---

```
public class Solution {
    public String countAndSay(int n) {

        if(n==0) return "";

        ArrayList<Integer> re = new ArrayList<Integer>();
        int count =0;
        int val = 1;
        re.add(1);
        for(int i=1;i<n;i++)
        {
            ArrayList<Integer> rr = new ArrayList<Integer>();
            for(int j=0;j<re.size();j++)
            {
                if(re.get(j) == val) count++;
                else
                {
                    rr.add(count);
                    rr.add(val);
                    val = re.get(j);
                    count = 1;
                }
            }
            rr.add(count);
            rr.add(val);
            val = rr.get(0);
            count = 0;
            re =rr;
        }
        String result = "";
        for(int i:re)
            result += Integer.toString(i);
        return result;

    }
}
```

==
/Users/linxie/Documents/github/leetcodeHub/code/DecodeWays
==

DecodeWays

A message containing letters from A-Z is being encoded to numbers using the following mapping:

A -> 1
B -> 2
...
Z -> 26
Given an encoded message containing digits, determine the total number of ways to decode it.

For example,
Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.

```
public class Solution {

    public boolean validate(String s){
        int n = s.length();
        if(n!=1 && n!=2) return false;

        try
        {
            int val = Integer.parseInt(s);
            if(val >0 && val <27)
            {
                if(n==1 || s.charAt(0)!=0) return true;
            }
        }
        catch(Exception o)
        {}
        return false;
    }

    public int numDecodings(String s, int[] map)
    {
        int n = s.length();
        if(n==0) return 1;

        if(map[n-1]!=-1) return map[n-1];

        int re = 0;
        if(validate(s.substring(n-1)))  re = numDecodings(s.substring(0, n-1), map);
        if(n-2>=0 && validate(s.substring(n-2))) re+=numDecodings(s.substring(0, n-2), map);
        map[n-1] = re;
        return re;
    }
    public int numDecodings(String s) {
        int n = s.length();
        if(n==0) return 0;
        int[] map = new int[n];
        for(int i=0;i<n;i++)
            map[i] = -1;
```

```
        return numDecodings(s,map);
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/DistinctSubsequences
=======================================


DistinctSubsequences

Given a string S and a string T, count the number of distinct subsequences of T in S.

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters

Here is an example:
S = "rabbbit", T = "rabbit"

Return 3.
---
public class Solution {
    public int numDistinct(String S, String T) {

        int n = T.length();
        if(n==0) return 1;
        int m = S.length();
        if(m==0) return 0;

        int[] re = new int[n+1];
        for(int i =1; i<n+1; i++)
            re[i] = 0;
        re[0] = 1;
        for(int i=0;i<m;i++)
        {
            for(int j = n; j>0; j--)
            {
                if(S.charAt(i)==T.charAt(j-1)) re[j] = re[j]+re[j-1];
            }
        }
        return re[n];


    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/DivideTwoIntegers
=======================================


DivideTwoIntegers

Divide two integers without using multiplication, division and mod operator.

---

public class Solution {
    public int divide(int dividend, int divisor) {

        if(dividend == 0) return 0;
        if(divisor == 0) return Integer.MAX_VALUE;

        long d1 = (long)dividend;
        long d2 = (long) divisor;
        int flg  = 0;
        if(d1 < 0){ d1=-d1; flg++; }
        if(d2  <0) { d2 = -d2; flg++; }
        if(d2>d1) return 0;
        if(d2==1) return (int)((flg%2==0)?d1:-d1);

        long ret = d2;
        int m = 1;
        while(ret<=d1)
        {
            if(ret == d1) return (flg%2==0)?m:-m;
            ret = ret<<1; m = m<<1;
        }
        ret=ret>>1; m=m>>1;
        long re = m+divide((int)(d1-ret),(int)d2);
        return  (int)((flg%2==0)?re:-re);


    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/EditDistance
=======================================


EditDistance

Given two words word1 and word2, find the minimum number of steps required to convert
word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

a) Insert a character
b) Delete a character
c) Replace a character
---
```

```java

public class Solution {
    public int minDistance(String word1, String word2) {

        char[] a = word1.toCharArray();
        char[] b = word2.toCharArray();
        int m = a.length;
        int n = b.length;
        if(m==0) return n;
        if(n==0) return m;

        int[] re = new int[m+1];
        for(int i=0;i<re.length;i++)
            re[i] = i;
        for(int i=0;i<n;i++)
        {
            int pre = -1;
            for(int j=0;j<re.length;j++)
            {
                if(j==0)
                {
                    pre = re[j];
                    re[j]++;
                }
                else
                {
                    int t = re[j];
                    if(a[j-1] != b[i]) re[j] = Math.min(re[j-1], Math.min(pre, re[j]))+1;
                    else
                        re[j] = pre;
                    pre = t;
                }
            }
        }
        return re[m];

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/EvaluateReversePolishNotation
========================================


EvaluateReversePolishNotation

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /. Each operand may be an integer or another expression.

Some examples:
  ["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
  ["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

----
public class Solution {
    public int evalRPN(String[] tokens) {

            int n = tokens.length;
            if(n==0) return 0;

            Stack<String> s = new Stack<String>();
            try{
                for(String t:tokens)
                {
                    if(t.equals("+"))
                    {
                        if(s.isEmpty()) break;
                        int a = Integer.parseInt(s.pop());
                        if(s.isEmpty()) break;
                        int b = Integer.parseInt(s.pop());
                        s.push(Integer.toString(a+b));
                    }
                    else if(t.equals("-"))
                    {
                        if(s.isEmpty()) break;
                        int a = Integer.parseInt(s.pop());
                        if(s.isEmpty()) break;
                        int b = Integer.parseInt(s.pop());
                        s.push(Integer.toString(b-a));
                    }
                    else if(t.equals("*"))
                    {
                        if(s.isEmpty()) break;
                        int a = Integer.parseInt(s.pop());
                        if(s.isEmpty()) break;
                        int b = Integer.parseInt(s.pop());
                        s.push(Integer.toString(a*b));
                    }
                    else if(t.equals("/"))
                    {
                        if(s.isEmpty()) break;
                        int a = Integer.parseInt(s.pop());
                        if(a==0 || s.isEmpty()) break;
                        int b = Integer.parseInt(s.pop());
                        s.push(Integer.toString(b/a));
                    }
                    else
                    {
                        s.push(t);
                    }
                }
            }
```

```
            catch(Exception o)
            {
                System.exit(-1);
            }
            if(!s.isEmpty()) return Integer.parseInt(s.pop());
            else return 0;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/Fibonacci
=======================================


Fibonacci_logN

import java.util.*;
import java.lang.*;

class fourNum
{
    int a0, a1, a2, a3;
    fourNum(int _a0, int _a1, int _a2, int _a3)
    { a0 = _a0; a1=_a1; a2=_a2; a3=_a3;}

    public fourNum multiple(fourNum o)
    {
        fourNum re = new fourNum(0,0,0,0);
        re.a0 = a0*o.a0 + a1*o.a2;
        re.a1 = a0*o.a1 + a1*o.a3;
        re.a2 = a2*o.a0 + a3*o.a2;
        re.a3 = a2*o.a1 + a3*o.a3;
        return re;
    }
}

class Solution
{
    public fourNum helper(int n)
    {
        if(n==0 || n==1) return new fourNum(1,1,1,0);
        fourNum t = helper(n/2);
        if(n%2==0)
        {
            return t.multiple(t);
        }
        else
        {
            return t.multiple(t).multiple(new fourNum(1,1,1,0));
        }
    }

    public int fibonacci(int n)
    {
        if(n<=1) return 1;
        return helper(n).a0;
    }
}



=======================================
/Users/linxie/Documents/github/leetcodeHub/code/findRotation
=======================================


Given a rotated array, find the number of rotation

======

    public int findTotate(int[] a)
    {
        int n = a.length;
        if(n<=1) return 0;

        int l = 0, r = n-1;
        int re = -1;
        while(l<r)
        {
            int mid = l+(r-l)/2;
            if(a[mid]>a[l] && a[mid]>=a[r])
            {
                re = mid;
                l = mid;
            }else if(a[mid]<a[l] && a[mid]<=a[r])
            {
                re=mid;
                r = mid;
            }
            else
            {
                l++;
            }
        }
        return (re==-1)?0:re;
     }


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/FirstMissingPositive
=======================================

```

FirstMissingPositive

Given an unsorted integer array, find the first missing positive integer.

For example,
Given [1,2,0] return 3,
and [3,4,-1,1] return 2.

Your algorithm should run in O(n) time and uses constant space.

```
public class Solution {
    public static int firstMissingPositive(int[] A) {
        int n = A.length;
        if(n==0) return 1;

        for(int i=0;i<n;i++)
        {
            while(A[i]>0 && A[i]<=n && A[i]!=i+1 && A[A[i]-1]!=A[i])
            {
                int t = A[i];
                A[i] = A[t-1];
                A[t-1] = t;
            }
        }

        for(int i=0;i<n;i++)
            if(A[i]!=i+1) return i+1;
        return n+1;
    }
}
```

==========

/Users/linxie/Documents/github/leetcodeHub/code/FlattenBinaryTreetoLinkedList

==========

FlattenBinaryTreetoLinkedList

Given a binary tree, flatten it to a linked list in-place.

For example,
Given

```
         1
        / \
       2   5
      / \   \
     3   4   6
```

The flattened tree should look like:

```
   1
    \
     2
      \
       3
        \
         4
          \
           5
            \
             6
```

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void flatten(TreeNode root) {
        TreeNode cur = root, pre = null, tail = null;
        while(cur!=null)
        {
            if(cur.left==null)
            {
                if(cur == root)
                {
                    tail = cur;
                }
                else
                {
                    tail.right = cur;
                    tail = cur;
                }
                TreeNode tmp = cur.right;
                cur.right = null;
                cur = tmp;
            }
            else
            {
                pre = cur.left;
                while(pre.right!=null) pre = pre.right;
                pre.right = cur.right;
                cur.right  = cur.left;
                cur.left = null;
            }
        }
        return;
```

```
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/GasStation
=======================================


GasStation

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i
to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas stations index if you can travel around the circuit once, otherwise return -1.

Note:
The solution is guaranteed to be unique.


---
public class Solution {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int n = gas.length;
        if(n!=cost.length)
            return -1;

        int[] diff = new int[n];
        int[] G = new int[n];
        for(int i=0;i<n;i++)
            diff[i] = gas[i] - cost[i];

        int min = Integer.MAX_VALUE;
        int sum = 0;

        for(int i=0;i<n;i++)
        {
            sum+=diff[i];
            if(sum<min)
                min = sum;
        }
        if(min>=0) return 0;

        G[0] = min;
        for(int i=1;i<n;i++)
        {
            G[i] = Math.min((G[i-1]-diff[i-1]), sum);
            if(G[i]>=0) return i;
        }

        return -1;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/GenerateParentheses
=======================================


GenerateParentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

---

public class Solution {
    public void helper(int n, int m, int k,  List<String> re, String r){
        if(k==n)
        {
            if(m==0)
            {
                re.add(new String(r));
            }
            return;
        }
        if(m==0)
        {
            helper(n, m+1,k+1,re, r+"(");
        }else
        {
            helper(n, m-1, k+1,re,r+")");
            helper(n, m+1, k+1,re,r+"(");
        }
    }
    public List<String> generateParenthesis(int n) {
         List<String> re = new ArrayList<String>();
         if(n==0) return re;
         helper(2*n, 0, 0, re, "");
         return re;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/GrayCode
```

```
=======================================


GrayCode

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print
the sequence of gray code. A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

00 - 0
01 - 1
11 - 3
10 - 2
---
public class Solution {
    public List<Integer> grayCode(int n) {

         List<Integer> re = new ArrayList<Integer>();
         if(n<0) return re;
         for(int i=0;i<(1<<n);i++)
         {
             re.add(i^(i>>1));
         }
         return re;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/ImplementstrStr
=======================================


ImplementstrStr

Implement strStr().

Returns a pointer to the first occurrence of needle in haystack, or null if needle is
not part of haystack.

---

public class Solution {
        public static int[] prefix(char[] a)
    {
        int n = a.length;
        if(n==0) return null;

        int[] F = new int[n+1];
        F[0] = F[1] = 0;
        for(int i=2; i<=n;i++)
        {
            int j = F[i-1];
            while(true)
            {
                if(a[j] == a[i-1]){ F[i] = j+1; break;}
                if(j==0){ F[i] = 0; break;}
                j = F[j];
            }
        }
        return F;
    }

    public String strStr(String haystack, String needle)
    {
        char[] ss = haystack.toCharArray();
        char[] pp = needle.toCharArray();
        int m = ss.length;
        int n = pp.length;
        if(ss.length<pp.length) return null;
        if(n==0 || haystack.equals(needle)) return haystack;

        int[] F = prefix(pp);

        int i=0, j=0;
        while(i<m)
        {
            if(ss[i] == pp[j])
            {
                i++; j++;
                if(j==n) return haystack.substring(i-n);
            }
            else if(j>0) j = F[j];
            else i++;
        }
        return null;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/InsertInterval
=======================================


InsertInterval

Given a set of non-overlapping intervals, insert a new interval into the intervals
(merge if necessary).

```

You may assume that the intervals were initially sorted according to their start times.

Example 1:
Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2:
Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as [1,2],[3,10],[12,16].

This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public List<Interval> insert(List<Interval> intervals, Interval newInterval) {


        int n = intervals.size();
        if(n==0)
        {
            intervals.add(newInterval);
            return intervals;
        }

        List<Interval> re = new ArrayList<Interval>();
        boolean flg = false;
        for(int i = 0;i<n;i++)
        {
            Interval in = intervals.get(i);
            if(flg || in.end<newInterval.start)
            {
                re.add(in);
            }
            else if(newInterval.end < in.start)
            {
                re.add(newInterval);
                re.add(in);
                flg = true;

            }
            else if(newInterval.start <= in.start)
            {
                if(newInterval.end<in.end)
                {
                    newInterval.end = in.end;
                }
            }else
            {
                if(newInterval.end <= in.end)
                {
                    re.add(in);
                    flg = true;
                }
                else
                {
                    newInterval.start = in.start;
                }
            }
        }
        if(!flg) re.add(newInterval);
        return re;


    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/InsertintoaCyclicSortedList
==========

InsertintoaCyclicSortedList

Given a node from a cyclic linked list which has been sorted, write a function to insert
a value into the list such that it remains a cyclic sorted list. The given node can be
any single node in the list.

--

```
class Solution
{
    public void insert(ListNode node, int x)
    {
        if(node == null)
        {
            node = new ListNode(x);
            node.next = node;
            return;
        }

        ListNode cur = node, prev = null;
        do
        {
            prev = cur;
            cur = cur.next;
```

```
            if(x>=prev.val && x<cur.val) break;
            else if(prev.val >cur.val && (x>prev.val || x<cur.val)) break;
        }
        while(cur!=node);

        prev.next = new ListNode(x);
        prev.next.next = cur;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/InsertionSortList
========================================


InsertionSortList

Sort a linked list using insertion sort
---
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode insertionSortList(ListNode head) {

        if(head == null || head.next == null) return head;

        ListNode cur = head.next;
        ListNode tail = head;
        head.next = null;

        while(cur!=null)
        {
            ListNode t = cur.next;
            cur.next = null;
            ListNode nd = head, prev = null;

            while(nd!=null && nd.val<=cur.val)
            {
                prev = nd;
                nd = nd.next;
            }
            if(nd==null)
            {
                tail.next = cur;
                tail = cur;
            }
            else
            {
                if(prev == null)
                {
                    cur.next = head;
                    head = cur;
                }
                else
                {
                    prev.next = cur;
                    cur.next = nd;
                }
            }
            cur = t;
        }
        return head;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/IntegertoRoman
========================================


IntegertoRoman

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

---

public class Solution {
    public String intToRoman(int num) {
        if(num<=0 || num>3999) return "";
        String[] m = {"M", "D", "C","L","X","V","I"};

        String re = "";
        int div = 1000;
        for(int i=0; i<=7; i+=2)
        {
            int val = num/div;
            if(val<4)
            {
                for(int j=0;j<val; j++)
                    re+=m[i];
```

```
            }else if(val <5)
            {
                re+=m[i];
                re+=m[i-1];
            }else if(val==5)
            {
                re+=m[i-1];
            }else if(val<9)
            {
                re+=m[i-1];
                for(int j=0;j<val-5; j++)
                {
                    re+=m[i];

                }
            }else if(val == 9)
            {
                re+=m[i];
                re+=m[i-2];
            }else break;
            num%=div;
            div/=10;
        }
        return re;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/InterleavingString
=======================================


InterleavingString

Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.

For example,
Given:
s1 = "aabcc",
s2 = "dbbca",

When s3 = "aadbbcbcac", return true.
When s3 = "aadbbbaccc", return false.

---
public class Solution {
    public int isInterleave(String s1, String s2, String s3, int[][] map)
    {

        int m=s1.length();
        int n=s2.length();

        if(map[m][n] !=0) return map[m][n];

        if(m+n!=s3.length())
        {
            map[m][n] = -1;
        }
        else
        {
            int re = -1;
            if(m==0)    re = (s2.equals(s3))?1:-1;
            else if(n==0)    re = (s1.equals(s3))?1:-1;
            else
            {
                int r1 = 0, r2 = 0;
                if(s1.charAt(m-1) == s3.charAt(m+n-1))
                {
                    if(m==1) r1=isInterleave("", s2, s3.substring(0,m+n-1), map);
                    else r1 = isInterleave(s1.substring(0,m-1), s2, s3.substring(0,m+n-1), map);
                }
                if(s2.charAt(n-1) == s3.charAt(m+n-1))
                {
                    if(n==1) r2=isInterleave(s1, "", s3.substring(0,m+n-1), map);
                    else r2 = isInterleave(s1,s2.substring(0,n-1), s3.substring(0,m+n-1), map);
                }
                if(r1==1 || r2==1)
                    re = 1;
                else
                    re = -1;
            }
            map[m][n] = re;
        }
        return map[m][n];
    }

    public boolean isInterleave(String s1, String s2, String s3)
    {
        int m = s1.length();
        int n=s2.length();
        if(m==0) return s2.equals(s3);
        if(n==0) return s1.equals(s3);
        int[][] map = new int[m+1][n+1];
        int re = isInterleave(s1, s2, s3, map);
        return  re== 1;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/JumpGame
=======================================

```

JumpGame

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:
A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.

```
public class Solution {
    public boolean canJump(int[] A) {
        int n = A.length;
        if(n<=1) return true;

        int max=0;
        int l=0, r=0;
        while(l<=r)
        {
            for(int i=l; i<=r; i++)
            {
                if(i+A[i] > max) max = i+A[i];
                if(max >= n-1) return true;
            }
            l = r+1;
            r = max;
        }
        return false;
    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/JumpGameII
==========

JumpGameII

Given an array of non-negative integers, you are initially positioned at the first index
of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example:
Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1,
    then 3 steps to the last index.)

```
public class Solution {
    public int jump(int[] A) {
        int n = A.length;
        if(n<=1) return 0;

        int jump = 0;
        int l = 0, r = 0;

        while(l<=r && r<n)
        {
            int max = r;
            jump++;
            for(int i = l; i<=r; i++)
            {
                if(A[i]+i > max)  max = A[i]+i;
                if(max >= n-1) return jump;
            }
            l = r+1;
            r = max;
        }
        return 0;
    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/LargestBSTinBT
==========

Given a binary tree, find the largest Binary Search Tree (BST), where largest
means BST with largest number of nodes in it. The largest BST may or may not
include all of its descendants.

```
class returnType
{
    int countNode;
    int maxNode;
    TreeNode largestBST;
    TreeNode child;
```

```
    returnType(int _countNode, int _maxNode, TreeNode _largestBST, TreeNode _child)
    {
        countNode  = _countNode;
        _maxNode = maxNode;
        _largestBST = largestBST;
        _child = child;
    }
}

class Solution
{

    TreeNode helper(TreeNode p, int min, int max, returnType cur)
    {
        if(p==null)
        {
            return new returnType(0, cur.maxNode, cur.largestBST, cur.child);
        }

        if(min<p.val && p.val<max)
        {
            returnType left = helper(p.left, min, p.val, cur);
            TreeNode re = new TreeNode(p.val);
            re.left = (left.countNode)?null:left.child;

            returnType right = helper(p.right, p.val, max, cur);
            re.right = (right.countNode)?null:right.child;

            cur.child = re;
            int total = left.countNode+right.countNode+1;
            if(total > cur.maxNode)
            {
                cur.maxNode = total;
                cur.largestBST = re;
            }
            return new returnType(total, cur.maxNode, cur.largestBST, cur.child);
        }
        else
        {
            returnType re = helper(p, Integer.MIN_VALUE Integer.MAX_VALUE, cur);
            re.countNode = 0;
            return re;
        }
    }

    TreeNode findLargestBST(TreeNode root)
    {
        returnType re =  helper(root, Integer.MIN_VALUE, Integer.MAX_VALUE,
            new returnType(0, Integer.MIN_VALUE, null, null));
        return re.largestBST;
    }

}



========================================
/Users/linxie/Documents/github/leetcodeHub/code/LargestRectangleinHistogram
========================================


LargestRectangleinHistogram

Given n non-negative integers representing the histograms bar height where the width of
 each bar is 1, find the area of largest rectangle in the histogram.


Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].


The largest rectangle is shown in the shaded area, which has area = 10 unit.

For example,
Given height = [2,1,5,6,2,3],
return 10.

--
public class Solution {
    public int largestRectangleArea(int[] height) {
        int n = height.length;
        if(n==0) return 0;
        if(n==1) return height[0];

        Stack<Integer> st = new Stack<Integer>();
        int min = 0;

        for(int i=0;i<n;i++)
        {
            if(st.isEmpty() || height[i]>=height[st.peek()])
                st.push(i);
            else{
                int tmp = height[st.pop()];
                int local = tmp* (st.isEmpty()?i:(i-st.peek()-1));
                if(local >min) min = local;
                i--;
            }
        }

        while(!st.isEmpty())
        {
            int tmp = height[st.pop()];
            int local = tmp* (st.isEmpty()?n:(n-st.peek()-1));
            if(local >min) min = local;
        }
```

```
        return min;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/LengthofLastWord
========================================


LengthofLastWord

Given a string s consists of upper/lower-case alphabets and empty space characters  ,
return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,
Given s = "Hello World",
return 5.
--

public class Solution {
    public int lengthOfLastWord(String s) {
        char[] c = s.toCharArray();
        int n = c.length;
        if(n==0) return 0;
        int len = 0;
        int i=0, j=0;
        for(;j<n;j++)
        {
            while(j<n && c[j] ==  ) j++;
            if(j==n) break;
            i = j;
            while(j<n && c[j]!= ) j++;
            len = j-i;
        }
        return len;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/LetterCombinationsofaPhoneNumber
========================================


LetterCombinationsofaPhoneNumber

Input:Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

---

public class Solution {
    public void letterCombinations(char[] digits, int i, List<String> re, char[] r, String[] map)
    {
        if(i==digits.length)
        {
            re.add(new String(r));
            return;
        }
        if(digits[i] <0 || digits[i] >9) return;
        for(char c:map[(int)(digits[i]-0)-1].toCharArray())
        {
            r[i] = c;
            letterCombinations(digits,i+1,re,r, map);
        }
    }
    public List<String> letterCombinations(String digits)
    {
         String[] map = {"","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
         List<String> re = new ArrayList<String>();

         int n = digits.length();
         if(n==0)
         {
             re.add("");
             return re;
         }

         letterCombinations(digits.toCharArray(), 0, re, new char[n], map);
         return re;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/LinkedListCycle
========================================


LinkedListCycle

Given a linked list, determine if it has a cycle in it.

Follow up:
Can you solve it without using extra space?
---
/**
 * Definition for singly-linked list.
```

```
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null || head.next == null) return false;
        if(head.next == head) return true;
        ListNode left = head, right = head;
        int count = 0;
        while(right.next != null)
        {
            right = right.next;
            count++;
            if(count%2 ==0)
            {
                left = left.next;
                if(right == left) return true;
            }
        }
        return false;
    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/LinkedListCycleII
======================================


LinkedListCycleII

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Follow up:
Can you solve it without using extra space?

---

/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode detectCycle(ListNode head) {

        if(head == null || head.next == null) return null;
        if(head.next == head) return head;
        ListNode left = head, right = head;
        int count = 0;
        while(right.next != null)
        {
            right = right.next;
            count++;
            if(count%2 ==0)
            {
                left = left.next;
                if(right == left) break;
            }
        }
        if(right.next == null) return null;
        left = head;
        while(left!=right){
            left=left.next;
            right=right.next;
        }
        return left;
    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/LongestCommonPrefix
======================================


LongestCommonPrefix

Write a function to find the longest common prefix string amongst an array of strings.

---

public class Solution {
    public static boolean longestCommonPrefix(String[] strs, int k)
    {
        String first = strs[0];
        if(first.length() <=k) return false;
        char c = first.charAt(k);
        for(String s:strs)
        {
            if(s.length()<=k || s.charAt(k)!=c) return false;
        }
```

```
        return true;
    }

    public String longestCommonPrefix(String[] strs)
    {
        int n = strs.length;
        if(n==0) return "";
        if(n==1) return strs[0];

        int i = 0;
        for(; i< strs[0].length(); i++)
        {
            if(!longestCommonPrefix(strs,i)) break;
        }
        return strs[0].substring(0, i);
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/LongestConsecutiveSequence
=======================================


LongestConsecutiveSequence

Given an unsorted array of integers, find the length of the longest consecutive
elements sequence.

For example,
Given [100, 4, 200, 1, 3, 2],
The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Your algorithm should run in O(n) complexity.

---
public class Solution {
    public int longestConsecutive(int[] num) {

        HashSet<Integer> m = new HashSet<Integer>();
        for(int i:num)
            m.add(i);

        int max =0;

        HashSet<Integer> mm = new HashSet<Integer>(m);
        for(int i: mm)
        {
            if(!m.contains(i)) continue;

            int t = i;
            int count = 1;
            m.remove(t);
            while(m.contains(++t)) count++;
            t=i;
            while(m.contains(--t)) count++;
            if(count>max) max=count;
        }
        return max;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/LongestPalindromicSubstring
=======================================


LongestPalindromicSubstring

Given a string S, find the longest palindromic substring in S. You may assume that the
maximum length of S is 1000, and there exists one unique longest palindromic substring.

---

public class Solution {
    public String longestPalindrome(String s) {
        int n = s.length();
        if(n<2) return s;

        int maxStart = 1, maxEnd = 0;
        for(int i=0;i<n;i++)
        {
            int l = i, r = i;
            while(l>=0 && r<n && s.charAt(l) == s.charAt(r)) {l--;r++;}
            if(maxEnd - maxStart < r-l-2)
            {
                maxStart = l+1;
                maxEnd = r-1;
            }
        }
        for(int i=0;i<n;i++)
        {
            int l = i, r = i+1;
            while(l>=0 && r<n && s.charAt(l) == s.charAt(r)) {l--;r++;}
            if(maxEnd - maxStart < r-l-2)
            {
                maxStart = l+1;
                maxEnd = r-1;
            }
        }
        return s.substring(maxStart, maxEnd+1);
    }
```

```
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/LongestSubstringWithoutRepeatingCharacters
========================================


LongestSubstringWithoutRepeatingCharacters

Given a string, find the length of the longest substring without repeating characters.
For example, the longest substring without repeating letters for "abcabcbb" is "abc",
which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

---

public class Solution {
    public int lengthOfLongestSubstring(String s) {

        int n=s.length();
        if(n<=1) return n;

        HashSet<Character> map = new HashSet<Character>();
        int max = 0;
        int i=0, j=0;

        for(;j<n;j++)
        {
            char c = s.charAt(j);
            if(!map.contains(c)){
                map.add(c);
            }
            else
            {
                if(max < j-i) max = j-i;
                for(;i<=j && s.charAt(i)!=c ;i++)
                {
                    map.remove(s.charAt(i));
                }
                i++;
            }
        }
        if(max < n-i) max = n-i;
        return max;


    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/LongestValidParentheses
========================================


LongestValidParentheses

Given a string containing just the characters ( and ), find the length of the longest valid
(well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()", which has length = 2.

Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

---

public class Solution {
    public int longestValidParentheses(String s) {
        int n = s.length();
        if(n<=1) return 0;

        Stack<Integer> st = new Stack<Integer>();
        boolean[] b = new boolean[n];
        for(int i=0;i<n;i++)
        {
            char c = s.charAt(i);
            if(c == ) && !st.isEmpty() && s.charAt(st.peek()) == ()
            {
                b[i] = true; b[st.peek()] = true;
                st.pop();
            }
            else
            {
                while(!st.isEmpty() && s.charAt(st.peek()) ==))    st.pop();
                st.push(i);
            }
        }
        int max = 0;
        int local = 0;
        for(int i =0; i<n; i++)
        {
            if(b[i]) local++;
            else
            {
                if(max <local) max = local;
                local = 0;
            }
        }
        if(max <local) max = local;
        return max;
    }
}
```

```
=======================================
/Users/linxie/Documents/github/leetcodeHub/code/LowestCommonAncestorofaBinaryTreePartI
=======================================


LowestCommonAncestorofaBinaryTreePartI

Given a binary tree, find the lowest common ancestor of two given nodes in the tree.

===

The solution below is a flaw one, since it hold the assumption that the tree has those two
nodes and those nodes are different. This is not true for general case.

node LCA(node root, node p, node q)
{
    if(root == null) return null;
    if(root == p || root == q) return root;

    node l = LCA(root.left, p, q);
    node r = LCA(root.right, p, q);

    if(l!=null && r!=null) return root;
    return (l!=null)?l:r;
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/LRUCache
=======================================


LRUCache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support
the following operations: get and set.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache,
otherwise return -1.
set(key, value) - Set or insert the value if the key is not already present. When the cache
reached its capacity, it should invalidate the least recently used item before inserting a new item.
----

class mem
{
    int key;
    int value;
    mem prev;
    mem next;
    mem(int _key, int _value)
    {
        key = _key;
        value = _value;
        prev = null;
        next = null;
    }

}

public class LRUCache {

    int capacity;
    int size;
    mem head;
    mem tail;
    HashMap<Integer, mem> map;

    public LRUCache(int capacity) {
        this.capacity = capacity;
        map = new HashMap<Integer, mem>();
        size = 0;
        mem head = null;
        mem tail = null;
    }

    public int get(int key) {
        if(!map.containsKey(key)) return -1;

         mem m = map.get(key);
         if(size == 1 || m == head)
         {
             return m.value;
         }

         if(m==tail)
         {
             tail = m.prev;
             m.prev = null;
             tail.next = null;

             m.next = head;
             head.prev = m;
             head = m;
         }else
         {
             m.prev.next = m.next;
             m.next.prev = m.prev;

             m.next = head;
             head.prev = m;
             m.prev = null;
             head = m;
         }
         return m.value;
```

```
    }

    public void set(int key, int value) {
        if( map.containsKey(key))
        {
            mem m = map.get(key);
            m.value = value;
            get(key);
        }
        else
        {
            if(size<capacity)
            {
                mem m = new mem(key, value);
                if(size == 0)
                {
                    head = m;
                    tail = m;
                }
                else
                {
                    m.next = head;
                    head.prev = m;
                    head = m;
                }
                map.put(key, m);
                size++;
            }
            else
            {
                int k = tail.key;
                map.remove(k);
                tail.value = value;
                tail.key = key;
                map.put(key, tail);
                get(key);
            }
        }

    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/MaximalRectangle
======================================


MaximalRectangle

Given a 2D binary matrix filled with 0s and 1s, find the largest rectangle containing all
ones and return its area
---

public class Solution {
     public static int maxHis(int[] a)
     {
         int n = a.length;
         if(n==0) return 0;
         if(n==1) return a[0];
         int max = 0;
         Stack<Integer> s = new Stack<Integer>();
         for(int i=0;i<n;i++)
         {
             if(s.isEmpty() || a[i] >= a[s.peek()]) s.push(i);
             else
             {

                 int tmp = a[s.pop()];
                 int local = tmp * (s.isEmpty()?i:i-s.peek()-1);
                 if(local > max) max = local;
                 i--;
             }
         }

         while(!s.isEmpty())
         {
             int tmp = a[s.pop()];
             int local = tmp *(s.isEmpty()?n:n-s.peek()-1);
             if(local > max) max = local;
         }
         return max;
     }

    public int maximalRectangle(char[][] matrix) {
         int m = matrix.length;
         if(m == 0) return 0;
         int n = matrix[0].length;
         if(n ==0 ) return 0;

         int[] cur = new int[n];
         for(int i=0;i<n;i++)
             cur[i] = (matrix[0][i]-0);
         int max = maxHis(cur);

         for(int i=1;i<m;i++)
         {
             for(int j = 0; j<n; j++)
             {
                 if(matrix[i][j] ==0) cur[j] = 0;
                 else cur[j]++;
             }
             int local = maxHis(cur);
             if(local > max) max = local;
```

```
        }
        return max;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/MaximumDepthofBinaryTree
=======================================


MaximumDepthofBinaryTree

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to
the farthest leaf node.
--
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDepth(TreeNode root) {

        if(root == null) return 0;
        return 1+Math.max(maxDepth(root.left), maxDepth(root.right));

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/MaximumProductSubarray
=======================================


MaximumProductSubarray

Find the contiguous subarray within an array (containing at least one number) which has
the largest product.

For example, given the array [2,3,-2,4],
the contiguous subarray [2,3] has the largest product = 6.
------
public class Solution {
    public int maxProduct(int[] A) {

        int n = A.length;
        if(n==0) return 0;

        int[] b = new int[n];
        boolean neg = false, pos = false;
        int max_neg = 0;
        int min_pos = 0;
        int prod = 1;
        int max = Integer.MIN_VALUE;
        for(int i=0;i<n;i++)
        {
            prod*=A[i];

            if(prod>max)
            {
                max = prod;
            }

            if(prod == 0)
            {
                 neg = false;
                 pos = false;
                 prod = 1;
            }
            else if(prod < 0)
            {
                if(!neg || prod > max_neg)
                {
                    if(!neg) neg = true;
                    max_neg = prod;
                }
                else
                {
                    if(prod/max_neg > max)
                    {
                        max = prod/max_neg;
                    }
                }
            }
            else
            {
                if(!pos || prod < min_pos)
                {
                    if(!pos) pos = true;
                    min_pos = prod;
                }
                else
                {
                    if(prod/min_pos > max)
                    {
                        max = prod/min_pos;
```

```
                    }
                }
            }
        }
        return max;


    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MaximumSubarray
========================================


MaximumSubarray

Find the contiguous subarray within an array (containing at least one number) which
has the largest sum.

For example, given the array [-2,1,-3,4,-1,2,1,-5,4],
the contiguous subarray [4,-1,2,1] has the largest sum = 6.

----

public class Solution {
    public int maxSubArray(int[] A) {
        int n=A.length;
        if(n==0) return 0;
        if(n==1) return A[0];

        int max = Integer.MIN_VALUE;
        int local = 0;
        for(int i:A)
        {
            local+=i;
            if(local>max) max = local;
            if(local<0) local = 0;
        }
        return max;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MaxPointsonaLine
========================================


MaxPointsonaLine

Given n points on a 2D plane, find the maximum number of points that lie on the same
straight line.
---
/**
 * Definition for a point.
 * class Point {
 *     int x;
 *     int y;
 *     Point() { x = 0; y = 0; }
 *     Point(int a, int b) { x = a; y = b; }
 * }
 */
public class Solution {
    public int maxPoints(Point[] points) {
        int n = points.length;
        if(n <= 2) return n;
        int max = 0;
        HashMap<Double, Integer> map = new HashMap<Double, Integer>();
        for(int i=0;i<n-1;i++)
        {
            map.clear();
            int vertical = 0, overlap = 0;
            int local = 0;
            for(int j=i+1; j<n;j++)
            {
                int dx = points[j].x - points[i].x;
                int dy = points[j].y - points[i].y;
                if(dx == 0)
                {
                    if(dy==0) overlap++;
                    else
                    {
                        vertical++;
                        if(vertical > local) local = vertical;
                    }
                }
                else if(dy==0)
                {
                    if(map.containsKey(0.0)) map.put(0.0,map.get(0.0)+1);
                    else    map.put(0.0,1);
                    if(map.get(0.0)>local) local = map.get(0.0);
                }
                else
                {
                    double k = 1.0*dy/dx;
                    if(map.containsKey(k)) map.put(k, map.get(k)+1);
                    else map.put(k,1);
                    if(local < map.get(k)) local = map.get(k);
                }
            }
            if(local+overlap>max) max = local+overlap;
        }
```

```
        return max+1;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MedianofTwoSortedArrays
========================================


MedianofTwoSortedArrays

There are two sorted arrays A and B of size m and n respectively. Find the median of the
two sorted arrays. The overall run time complexity should be O(log (m+n)).

---

public class Solution {
    public double findKth(int k, int[] A, int al, int ar, int[] B, int bl, int br)
    {
        int alen = ar-al+1;
        int blen = br-bl+1;
        if(alen==0)
        {
            if(blen<=k) System.exit(0);
            return B[bl+k];
        }
        if(blen==0)
        {
            if(alen<=k) System.exit(0);
            return A[al+k];
        }
        if(k==0) return A[al]<=B[bl]?A[al]:B[bl];

        int amid = k*alen/(alen+blen);
        int bmid = k-amid-1;
        amid+=al;
        bmid+=bl;
        if(A[amid]>B[bmid])
        {
            k-=bmid-bl+1;
            ar =amid;
            bl=bmid+1;
        }else
        {
            k-=amid-al+1;
            br=bmid;
            al = amid+1;
        }
        return findKth(k, A, al, ar, B, bl, br);
    }

    public double findMedianSortedArrays(int A[], int B[]) {
        int m = A.length;
        int n = B.length;
        if((m+n)%2==1) return findKth((m+n)/2, A, 0, m-1, B, 0, n-1);
        else
            return (  findKth((m+n)/2-1, A, 0, m-1, B, 0, n-1) +  findKth((m+n)/2, A, 0, m-1, B, 0, n-1))/2;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MergeIntervals
========================================


MergeIntervals

Given a collection of intervals, merge all overlapping intervals.

For example,
Given [1,3],[2,6],[8,10],[15,18],
return [1,6],[8,10],[15,18].

---
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public static boolean check(List<Interval> intervals, int[] index, int l, int r)
    {
        return intervals.get(index[l]).start <= intervals.get(index[r]).start ;
    }

    public static int partition(List<Interval> intervals, int[] index, int l, int r)
    {
        if(l>r || l<0 || r>=index.length) return -1;
        int p = l;
        int i = l-1, j = r+1;
        while(i<j)
        {
            while((++i)<j && check(intervals, index, i, p));
            while(i<=(--j) && !check(intervals, index, j, p));
            if(i>j) break;

            int t = index[i];
            index[i] = index[j];
```

```
            index[j] = t;
        }
        int t = index[l];
        index[l] = index[j];
        index[j] = t;
        return j;
    }

    public static void    sort(List<Interval> intervals, int[] index, int l, int r)
    {
        if(l>=r) return;

        int p = partition(intervals, index, l, r);
        sort(intervals, index, l, p-1);
        sort(intervals, index, p+1, r);
    }

    public List<Interval> merge(List<Interval> intervals) {
        int n = intervals.size();
        if(n<=1) return intervals;

        int[] index = new int[n];
        for(int i=0; i<n; i++)
            index[i] = i;
        sort(intervals, index, 0, n-1);
        // for(int i:index)
        //     System.out.print(intervals.get(i)+" ");
        // System.out.println();
        List<Interval> new_intervals = new ArrayList<Interval>();

        Interval prev = intervals.get(index[0]);
        new_intervals.add(prev);

        for(int i=1; i<n;i++)
        {
            Interval cur = intervals.get(index[i]);

            if(cur.start > prev.end)
            {

                new_intervals.add(cur);
                prev = cur;
            }
            else if(cur.end > prev. end)
            {

                prev.end = cur.end;
            }
        }
        return new_intervals;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MergekSortedLists
========================================


MergekSortedLists

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

---

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
class node implements Comparable
{
    ListNode nd;
    int val;
    node(ListNode _nd)
    {
        nd = _nd;
        val = _nd.val;
    }

    public int compareTo(Object o)
    {
        return val-((node)o).val;
    }
}
public class Solution {
    public ListNode mergeKLists(List<ListNode> lists)
    {
        int n = lists.size();
        if(n==0) return null;
        if(n==1) return lists.get(0);

        PriorityQueue<node> que = new PriorityQueue<node>();

        ListNode head = null, tail =null;
        for(ListNode l:lists)
        {
            if(l==null) continue;
```

```
            que.add(new node(l));
            l=l.next;
        }
        while(!que.isEmpty())
        {
            ListNode min = que.poll().nd;
            if(head == null)
            {
                head = min; tail = min;
            }
            else
            {
                tail.next = min;
                tail = min;
            }
            if(min.next!=null)    que.add(new node(min.next));
            min.next = null;
        }
        return head;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/MergeSortedArray
=======================================


MergeSortedArray

Given two sorted integer arrays A and B, merge B into A as one sorted array.

Note:
You may assume that A has enough space (size that is greater or equal to m + n) to hold
additional elements from B. The number of elements initialized in A and B are m and n respectively.

---

public class Solution {
    public void merge(int A[], int m, int B[], int n) {
        if(n==0) return;
        if(m==0)
        {
            for(int i=0;i<n;i++)
                A[i] = B[i];
            return;
        }

        int ind = m+n-1;

        int i = m-1, j = n-1;;
        while( i>=0 && j>=0)
        {
            if(A[i]>B[j])   A[ind--] = A[i--];
            else    A[ind--] = B[j--];
        }
        while(i>=0)
            A[ind--] = A[i--];
        while(j>=0)
            A[ind--] = B[j--];
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/MergeTwoSortedLists
=======================================


MergeTwoSortedLists

Merge two sorted linked lists and return it as a new list. The new list should be made by
splicing together the nodes of the first two lists.

---
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if(l1==null) return l2;
        if(l2==null) return l1;

        ListNode head = null, tail = null;

        while(l1!=null && l2!=null)
        {
            if(l1.val<=l2.val)
            {
                ListNode t = l1.next;
                l1.next = null;
                if(head == null)
                {
                    head = l1; tail = l1;
                }
                else
```

```
                {
                    tail.next = l1;
                    tail = l1;
                }
                l1 = t;
            }
            else
            {
                ListNode t = l2.next;
                l2.next = null;
                if(head == null)
                {
                    head = l2; tail = l2;
                }
                else
                {
                    tail.next = l2;
                    tail = l2;
                }
                l2 = t;
            }
        }
        if(l1!=null) tail.next = l1;
        if(l2!=null) tail.next = l2;
        return head;


    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MinimumDepthofBinaryTree
========================================


MinimumDepthofBinaryTree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down
to the nearest leaf node.

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int minDepth(TreeNode root) {
        if(root==null) return 0;
        int left = minDepth(root.left);
        int right = minDepth(root.right);
        if(left == 0) return right+1;
        if(right == 0) return left+1;
        return 1+Math.min(left, right);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MinimumPathSum
========================================


MinimumPathSum

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right
which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

--
public class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        if(m==0) return 0;
        int n = grid[0].length;
        if(n==0) return 0;

        int[] re = new int[n];

        re[0] = grid[0][0];
        for(int i=1; i<n; i++)
            re[i] = re[i-1] + grid[0][i];

        for(int i=1; i<m;i++)
        {
            re[0] += grid[i][0];
            for(int j=1; j<n; j++)
                re[j] = grid[i][j] + Math.min(re[j], re[j-1]);
        }
        return re[n-1];
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/MinimumWindowSubstring
```

```
=======================================


MinimumWindowSubstring

Given a string S and a string T, find the minimum window in S which will contain all the
characters in T in complexity O(n).

For example,
S = "ADOBECODEBANC"
T = "ABC"
Minimum window is "BANC".

Note:
If there is no such window in S that covers all characters in T, return the emtpy string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique
minimum window in S.

---
```

```java
public class Solution {
    public String minWindow(String S, String T)
    {
        int mm = S.length();
        int n = T.length();
        char[] s = S.toCharArray();
        char[] t = T.toCharArray();

        if(n==0 || mm==0 || mm<n) return "";

        HashMap<Character, Integer> map = new HashMap<Character, Integer>();
        int m = 0;
        for(char c:t)
            if(map.containsKey(c))
                map.put(c,map.get(c)+1);
            else
            {
                map.put(c, 1);
            }

        m = map.size();
        int maxStart = 0, maxEnd = Integer.MAX_VALUE;
        int i = 0, j = 0;

        for(;j<mm;j++)
        {
            if(m>0)
            {
                if(map.containsKey(s[j]))
                {
                    int val = map.get(s[j]);
                    if(val == 1) m--;
                    map.put(s[j], val-1);
                }
            }

            if(m<=0)
            {
                for(;i<=j;i++)
                {

                    char c = s[i];
                    if(map.containsKey(c))
                    {
                        int val = map.get(c);
                        map.put(c, val+1);
                        if(val==0) m++;
                        if(m>0)
                        {
                            break;
                        }
                    }
                }
                if(i>j) return T;

                if(m>0 && (j-i) < maxEnd-maxStart)
                {
                    maxStart = i;
                    maxEnd = j;
                }
                i++;
            }
        }
        if(maxEnd == Integer.MAX_VALUE) return "";
        return S.substring(maxStart, maxEnd+1);
    }
}
```

```
=======================================
/Users/linxie/Documents/github/leetcodeHub/code/MultiplyStrings
=======================================


MultiplyStrings

Given two numbers represented as strings, return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative

----
```

```java
public class Solution {
```

```
        public static void reverse(char[] s, int l, int r)
    {
        while(l<r)
        {
            char c = s[l];
            s[l] = s[r];
            s[r]=c;
            l++; r--;
        }
    }

    public String multiply(String num1, String num2) {

        char[] n1 = num1.toCharArray();
        char[] n2 = num2.toCharArray();

        if(n1.length==0) return num2;
        if(n2.length==0) return num1;

        char[] re = new char[n1.length+n2.length];
        reverse(n1, 0, n1.length-1);
        reverse(n2, 0, n2.length-1);

        int carry = 0;

        for(int i=0;i<re.length;i++)
        {
            int val = carry;
            for(int j = 0; j<=i && j<n1.length;j++)
            {
                int k = i-j;
                if( k >=0 && k<n2.length)
                {
                    val += (int)(n1[j]-0)* (int)(n2[k]-0);
                }
            }
            carry = val/10;
            re[i] = (char)(val%10 + 0);
        }
        if(re[re.length-1] == 0 && carry != 0 ) re[re.length-1] = (char)(0+carry);

        int i = re.length-1;
        while(i>=0 && re[i] == 0) i--;
        if(i<0) return "0";
        reverse(re, 0, i);
        return new String(re).substring(0,i+1);

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/N-Queens
=======================================


N-Queens

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens placement, where Q and .
both indicate a queen and an empty space respectively.

For example,
There exist two distinct solutions to the 4-queens puzzle:

[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]---

public class Solution {
     public static boolean valid(int[] a, int k){
         for(int i=0;i<k;i++)
         {
             if(a[k] == a[i] || Math.abs(i-k) == Math.abs(a[i] - a[k])) return false;

         }
         return true;
     }

     public static void solveNQueens(int n, int k, int[] a, List<int[]> re)
     {
         if(k==n)
         {
             int[] aa = new int[n];
             for(int i=0;i<n;i++)
             {
                 aa[i] = a[i];
             }
             re.add(aa);
             return;
         }

         for(int i=0;i<n;i++)
         {
             a[k]=i;
```

```
            if(valid(a,k))
                solveNQueens(n, k+1, a, re);
        }
    }

    public static List<String[]> solveNQueens(int n)
    {
        List<String[]> re = new ArrayList<String[]>();

        if(n==0) return re;
        if(n==1){
            String[] r_str = new String[n];
            r_str[0] = "Q";
            re.add(r_str);
            return re;
        }

        int[] a = new int[n];
        List<int[]> r = new  ArrayList<int[]>();
        solveNQueens(n,0,a,r);

        for(int i=0; i<r.size(); i++)
        {
            String[] r_str = new String[n];
            for(int j=0;j<n;j++)
            {
                String rr = "";
                for(int k =0; k<n; k++)
                {
                    if( r.get(i)[j] == k)
                    {
                        rr+="Q";
                    }
                    else rr+=".";
                }
                r_str[j] = rr;
            }
            re.add(r_str);
        }
        return re;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/N-QueensII
=======================================


N-QueensII

Follow up for N-Queens problem.

Now, instead outputting board configurations, return the total number of distinct solutions.

---
public class Solution {
    public boolean valid(int n, int k, int[] re)
    {
        int r = re[k];
        for(int i=0;i<k;i++)
        {
            if(re[i] == r || Math.abs(re[i]-r) == k-i) return false;
        }
        return true;
    }

    public int totalNQueens(int n, int k, int[] re)
    {
        if(n==k) return 1;

        int r = 0;
        for(int i=0; i<n;i++)
        {
            re[k] = i;
            if(valid(n,k,re))
                r+=totalNQueens(n,k+1,re);
        }
        return r;
    }
    public int totalNQueens(int n)
    {
        if(n<=1) return n;
        int[] re = new int[n];
        for(int i=0;i<n;i++)
        {
            re[i] = -1;
        }

        return totalNQueens(n, 0, re);
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/NextPermutation
=======================================


NextPermutation

Implement next permutation, which rearranges numbers into the lexicographically next greater
permutation of numbers.
```

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie,
    sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are
in the right-hand column.
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1

```
public class Solution {
    public void nextPermutation(int[] num) {
        int n=num.length;
        if(n<=1) return;

        int i = n-1;
        while(i>0 && num[i]<=num[i-1]) i--;
        if(i==0)
        {
            int l = 0, r = n-1;
            while(l<r)
            {
               int t = num[l];
                num[l] = num[r];
                num[r] = t;
                l++; r--;
            }
            return;
        }

        int j=n-1;
        while(j>=i && num[j]<=num[i-1]) j--;
        if(j<i) return;

        int t = num[i-1];
        num[i-1] = num[j];
        num[j] = t;
        int l = i, r = n-1;
        while(l<r)
        {
            t = num[l];
            num[l] = num[r];
            num[r] = t;
            l++; r--;
        }
    }
}
```

==
/Users/linxie/Documents/github/leetcodeHub/code/PalindromeNumber
==

PalindromeNumber

Determine whether an integer is a palindrome. Do this without extra space.

```
public class Solution {
    public boolean isPalindrome(int x) {


        if(x==0) return true;
        if(x==Integer.MIN_VALUE) return false;
        if(x<0) return false;

        int t = x;
        int div = 10;
        while(true)
        {
            if(t/div!=0)
            {
                if(div< Integer.MAX_VALUE/10)
                {
                    div*=10;
                    continue;
                }
            }
             else div /=10;
             break;
        }
        int l = 1, r = div;
        while(l<=r){
            if((x/l%10) != (x/r%10))
            {
                return false;
            }
            l*=10;
            r/=10;
        }
        return true;


    }
}
```

```
public class Solution {
    public boolean isPalindrome(int x) {
```

```
        if(x<0) return false;

        int div = 1;
        while(x/div>=10)
        {
            div*=10;
        }

        while(x!=0)
        {
            int l = x/div;
            int r = x%10;
            if(l!=r) return false;

            x = (x%div)/10;
            div /=100;
        }
        return true;
    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/PalindromePartitioning
======================================


PalindromePartitioning

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given s = "aab",
Return

  [
    ["aa","b"],
    ["a","a","b"]
  ]
  ---
  public class Solution {
     public static boolean valid(String s)
     {
         int n =s.length();
         if(n<=1) return true;
         if(s.charAt(0) == s.charAt(n-1)) return valid(s.substring(1,n-1));
         return false;
     }

    public static List<List<String>> partition(String s, HashMap<Integer, List<List<String>>> map)
    {
           List<List<String>> re = new ArrayList<List<String>>();
           int n = s.length();
           if(n==0) return re;
           if(map.containsKey(n)) return map.get(n);

           for(int i=n-1;i>=0;i--)
           {
               if(valid(s.substring(i)))
               {
                   List<List<String>> r = partition(s.substring(0,i),map);
                   if(r.size() == 0)
                   {
                       if(i==0)
                       {
                           List<String> rr = new ArrayList<String>();
                           rr.add(s);
                           re.add(rr);
                           continue;
                       }
                   }
                   else
                   {
                       for(List<String>l:r)
                       {
                           List<String> rr = new ArrayList<String>(l);
                           rr.add(s.substring(i));
                           re.add(rr);
                       }
                   }
               }
           }
           map.put(n,re);
           return re;
    }

    public static List<List<String>> partition(String s) {
        return partition(s, new HashMap<Integer, List<List<String>>>());
    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/PalindromePartitioningII
======================================


PalindromePartitioningII

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.
```

```

For example, given s = "aab",
Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.
---
public class Solution {
    public boolean valid(char[] c, int l, int r, int n, int[] m)
    {
        int k = l*n+r;
        if(m[k]!=-1) return m[k] == 1;

        if(l>=r) return true;

        if(c[l] == c[r] && valid(c, l+1, r-1, n, m))
        {
            m[k] = 1;
        }
        else
        {
            m[k] = 0;
        }
        return m[k]==1;
    }

    public int minCut(char[] s, int k, int n, int[] cuts, int[] m)
    {
        if(cuts[k]!=-1) return cuts[k];

        int min =  Integer.MAX_VALUE;
        for(int i=k;i>0;i--)
        {
            if(valid(s, i, k, n, m))
            {
                int prev = minCut(s, i-1, n, cuts, m);

                if(prev == -2) continue;
                if(prev!=-1 && prev<min ) min = prev;
            }
        }
        if(min!=Integer.MAX_VALUE)
        {
            cuts[k] = min+1;
        }
        else    cuts[k] = -2;

        return cuts[k];
    }

    public int minCut(String s)
    {
        int n = s.length();
        if(n<=1) return 0;
        int[] cuts = new int[n];
        for(int i=0;i<n;i++)
            cuts[i] = -1;
        int[] m = new int[n*n];
        for(int i=0;i<n*n;i++)
            m[i] = -1;
        for(int i =0;i<n;i++)
        {
            if( valid(s.toCharArray(),0, i, n, m))
            {

                cuts[i] = 0;
            }
        }

        return minCut(s.toCharArray(),n-1, n,  cuts, m);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/PartitionList
========================================


PartitionList

Given a linked list and a value x, partition it such that all nodes less than x come
before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given 1->4->3->2->5->2 and x = 3,
return 1->2->2->4->3->5.
--
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode partition(ListNode head, int x) {

        if(head == null) return null;
        ListNode less = null, lTail = null, rTail = null;
        ListNode cur = head;
```

```
        head = null;

        while(cur!=null)
        {
            ListNode t = cur.next;
            cur.next = null;
            if(cur.val < x)
            {
                if(lTail == null)
                {
                    less = cur;
                    lTail = cur;
                }
                else
                {
                    lTail.next = cur;
                    lTail = cur;
                }
            }else
            {
                if(rTail == null)
                {
                    head = cur;
                    rTail = cur;
                }
                else
                {
                    rTail.next = cur;
                    rTail = cur;
                }
            }
            cur = t;
        }
        if(lTail == null) return head;
        lTail.next = head;
        return less;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/PascalsTriangle
========================================


PascalsTriangle

Given numRows, generate the first numRows of Pascals triangle.

For example, given numRows = 5,
Return

[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
---
public class Solution {
    public List<List<Integer>> generate(int numRows) {

        List<List<Integer>> re = new ArrayList<List<Integer>>();

        if(numRows == 0) return re;
        List<Integer> r = new ArrayList<Integer>();
        r.add(1);
        re.add(r);
        if(numRows == 1) return re;

        for(int i=1;i<numRows;i++)
        {
            List<Integer> rr = re.get(i-1);
            r = new ArrayList<Integer>();
            r.add(1);
            for(int j=0;j<rr.size()-1;j++)
            {
                r.add(rr.get(j) + rr.get(j+1));
            }
            r.add(1);
            re.add(r);
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/PascalsTriangleII
========================================


PascalsTriangleII

Given an index k, return the kth row of the Pascals triangle.

For example, given k = 3,
Return [1,3,3,1].

Note:
Could you optimize your algorithm to use only O(k) extra space
---
```

```
5200
5201 public class Solution {
5202     public List<Integer> getRow(int rowIndex) {
5203
5204         List<Integer> re = new ArrayList<Integer>();
5205         if(rowIndex<0) return re;
5206
5207         int n = rowIndex+1;
5208
5209         int[] a = new int[n];
5210         a[0]=1;
5211         for(int i=1;i<n;i++)
5212         {
5213             for(int j=i;j>=0;j--)
5214             {
5215                 if(j==0 || j==i) a[j] = 1;
5216                 else
5217                 {
5218                     a[j] += a[j-1];
5219                 }
5220             }
5221         }
5222         for(int i:a)
5223             re.add(i);
5224         return re;
5225
5226     }
5227 }
5228
5229
5230 ======================================
5231 /Users/linxie/Documents/github/leetcodeHub/code/PathSum
5232 ======================================
5233
5234
5235 PathSum
5236
5237 Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that
5238 adding up all the values along the path equals the given sum.
5239
5240 For example:
5241 Given the below binary tree and sum = 22,
5242               5
5243              / \
5244             4   8
5245            /   / \
5246           11  13  4
5247          /  \      \
5248         7    2      1
5249 return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.
5250 ---
5251 /**
5252  * Definition for binary tree
5253  * public class TreeNode {
5254  *     int val;
5255  *     TreeNode left;
5256  *     TreeNode right;
5257  *     TreeNode(int x) { val = x; }
5258  * }
5259  */
5260 public class Solution {
5261     public boolean hasPathSum(TreeNode root, int sum) {
5262         if(root == null) return false;
5263         if(root.left == null && root.right == null)
5264             return root.val == sum;
5265         return hasPathSum(root.left, sum-root.val) || hasPathSum(root.right, sum-root.val);
5266     }
5267 }
5268
5269
5270 ======================================
5271 /Users/linxie/Documents/github/leetcodeHub/code/Permutations
5272 ======================================
5273
5274
5275 Permutations
5276
5277 Given a collection of numbers, return all possible permutations.
5278
5279 For example,
5280 [1,2,3] have the following permutations:
5281 [1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].
5282 ---
5283 public class Solution {
5284     public static boolean nextPerm(int[] num)
5285     {
5286         int n=num.length;
5287         if(n==0) return false;
5288         for(int i=n-2; i>=0; i--)
5289         {
5290             if(num[i]<num[i+1])
5291             {
5292                 for(int j=n-1;j>i;j--)
5293                 {
5294                     if(num[j]>num[i])
5295                     {
5296                         int t = num[i];
5297                         num[i] = num[j];
5298                         num[j] = t;
5299                         i = i+1; j = n-1;
5300                         while(i<j)
5301                         {
5302                             t = num[i];
5303                             num[i] = num[j];
```

```
                            num[j] = t;
                            i++; j--;
                        }
                        return true;
                    }
                }
            }
        }
        return false;
    }

    public List<List<Integer>> permute(int[] num) {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n==0) return re;
        List<Integer> r = new ArrayList<Integer>();
        if(n==1)
        {
            r.add(num[0]);
            re.add(r);
            return re;
        }
        Arrays.sort(num);
        for(int i:num)
        {
            r.add(i);
        }
        re.add(r);
        while(nextPerm(num))
        {
           r = new ArrayList<Integer>();
            for(int i:num)
            {
                r.add(i);
            }
            re.add(r);
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/PermutationSequence
========================================


PermutationSequence

The set [1,2,3,…,n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order,
We get the following sequence (ie, for n = 3):

"123"
"132"
"213"
"231"
"312"
"321"
Given n and k, return the kth permutation sequence.

---

public class Solution {

     public static boolean nextPerm(int[] a)
     {
         int n = a.length;
         if(n<=1) return false;

         int i=n-1, j=n-1;
         for(; i>0 && a[i]<=a[i-1];i--);
         if(i==0) return false;

         for(; j>=i && a[j]<=a[i-1];j--);
         if(j==i-1) return false;

         int t = a[i-1]; a[i-1] = a[j]; a[j] = t;

         for(int l=i, r=n-1; l<r; l++, r--)
         {
             t = a[l]; a[l] = a[r]; a[r] =t;
         }
         return true;
     }

    public static String getPermutation(int n, int k) {
        if(n==0) return "";

        int[] a= new int[n];
        for(int i=0;i<n;i++)
        {
            a[i] = i+1;
        }
        int count =0;
        for(int i=1; i<k; i++)
        {
            count++;
            if(!nextPerm(a)) return getPermutation(n, k%count);
        }
        char[] re = new char[n];
        for(int i=0;i<n;i++)
        {
```

```
            re[i] = (char)(a[i]+0);
        }
        return new String(re);
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/PermutationsII
=======================================


PermutationsII

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,
[1,1,2] have the following unique permutations:
[1,1,2], [1,2,1], and [2,1,1].

---

public class Solution {
    public boolean nextPerm(int[] num)
    {
        int n = num.length;
        if(n==0) return false;

        int j=n-1;
        for(;j>0;j--)
            if(num[j-1] < num[j]) break;
        if(j==0) return false;;

        j--;
        int i=n-1;
        for(;i>=j+1;i--)
            if(num[i]>num[j])
                break;
        int t = num[j]; num[j] = num[i]; num[i] = t;
        j++; i = n-1;
        while(j<i)
        {
            t = num[j]; num[j] = num[i]; num[i] = t;
            i--; j++;
        }
        return true;
    }

    public List<List<Integer>> permuteUnique(int[] num) {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n==0) return re;

        Arrays.sort(num);
        List<Integer> r = new ArrayList<Integer>();
        for(int i:num)
            r.add(i);
        re.add(r);
        while(nextPerm(num))
        {
            r = new ArrayList<Integer>();
            for(int i:num)
               r.add(i);
            re.add(r);
        }
        return re;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/PlusOne
=======================================


PlusOne

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.
--
public class Solution {
    public int[] plusOne(int[] digits) {
        int n = digits.length;
        if(n==0) return digits;

        int carry = 1;
        for(int i=n-1;i>=0;i--)
        {
            int re = carry+digits[i];
            carry = re/10;
            digits[i] = re%10;
        }
        int[] r;
        if(carry == 1)
        {
            r =  new int[n+1];
            r[0] = carry;
            for(int i=0;i<n;i++)
                r[i+1] = digits[i];
        }
        else
        {
            r =  new int[n];
```

```
            for(int i=0;i<n;i++)
                r[i] = digits[i];
        }
        return r;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/PopulatingNextRightPointersinEachNode
========================================


PopulatingNextRightPointersinEachNode

Given a binary tree

    struct TreeLinkNode {
      TreeLinkNode *left;
      TreeLinkNode *right;
      TreeLinkNode *next;
    }
Populate each next pointer to point to its next right node. If there is no next right node,
the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Note:

You may only use constant extra space.
You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and
    every parent has two children).
For example,
Given the following perfect binary tree,
         1
       /  \
      2    3
     / \  / \
    4  5  6  7
After calling your function, the tree should look like:
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL

---
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
 * }
 */
public class Solution {

    public static void connect(TreeLinkNode root, TreeLinkNode par)
    {
        if(root == null) return;
        if(par == null) { root.next = null;  return;}
        if(root == par.left && par.right!=null) { root.next = par.right; connect(root.next, par); return; }
        par = par.next;
        while(par!=null)
        {
            if(par.left!=null)
            {
                root.next = par.left;
                connect(root.next, par);
                break;
            }

            else if(par.right!=null)
            {
                root.next = par.right;
                connect(root.next, par);
                break;
            }
            else par = par.next;
        }
    }

    public static void connect(TreeLinkNode root) {
        TreeLinkNode  par =null;
        while(root!=null)
        {
            connect(root, par);
            while(root!=null)
            {
                if(root.left!=null)
                {
                    par = root;
                    root = root.left;
                    break;
                }
                else if(root.right != null)
                {
                    par = root;
                    root = root.right;
                    break;
                }else
                {
                    root = root.next;
                }
```

```
            }
        }
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/PopulatingNextRightPointersinEachNodeII
========================================


PopulatingNextRightPointersinEachNodeII

Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree? Would your previous solution still work?

Note:

You may only use constant extra space.
For example,
Given the following binary tree,
         1
       /  \
      2    3
     / \    \
    4   5    7
After calling your function, the tree should look like:
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \    \
    4-> 5 -> 7 -> NULL
---
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
 * }
 */
public class Solution {

    public static void connect(TreeLinkNode root, TreeLinkNode par)
    {
        if(root == null) return;
        if(par == null) { root.next = null;  return;}
        if(root == par.left && par.right!=null) { root.next = par.right; connect(root.next, par); return; }
        par = par.next;
        while(par!=null)
        {
            if(par.left!=null)
            {
                root.next = par.left;
                connect(root.next, par);
                break;
            }

            else if(par.right!=null)
            {
                root.next = par.right;
                connect(root.next, par);
                break;
            }
            else par = par.next;
        }
    }

    public static void connect(TreeLinkNode root) {
        TreeLinkNode  par =null;
        while(root!=null)
        {
            connect(root, par);
            while(root!=null)
            {
                if(root.left!=null)
                {
                    par = root;
                    root = root.left;
                    break;
                }
                else if(root.right != null)
                {
                    par = root;
                    root = root.right;
                    break;
                }else
                {
                    root = root.next;
                }
            }
        }
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/Pow(x,n)
========================================


Pow(x,n)

```

```
---
public class Solution {
    public double pow(double x, int n) {

        if(x==0 || x==1 || n==1) return x;
        if(n==0)  return 1.0;

        if(n==Integer.MIN_VALUE)
        {
            n++;
            n=-n;
            double ret = pow(x,n/2);
            if(n%2==1)
            {
                return 1/(ret*ret*x*x);
            }else
            {
                return 1/(ret*ret*x);
            }
        }

        boolean flag = false;
        if(n<0)
        {
            flag = true;
            n=-n;
        }
        double ret = pow(x,n/2);
        ret*=ret;
        if(n%2==1)
        {
            ret *= x;
        }
        if(flag)
            ret = 1/ret;
        return ret;


    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/RecoverBinarySearchTree
=======================================


RecoverBinarySearchTree

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void recoverTree(TreeNode root) {

        if(root == null) return;

        TreeNode first = null, second = null;

        Stack<TreeNode> s = new Stack<TreeNode>();
        TreeNode cur = root, prev = null;
        while(!s.isEmpty() || cur!=null)
        {

            if(cur!=null)
            {
                s.push(cur);
                cur = cur.left;
            }
            else
            {
                cur = s.pop();
                if(prev!=null && prev.val > cur.val)
                {
                    if(first ==null)
                    {
                        first = prev;
                        second = cur;
                    }
                    else
                    {
                        second = cur;
                        break;
                    }
                }
                prev = cur;
                cur = cur.right;
            }
        }
        if(first!=null && second!=null)
        {
            int t = first.val;
            first.val = second.val;
```

```
            second.val = t;
            // System.out.println(first.val+", "+second.val);
        }
        return;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/RegularExpressionMatching
=======================================


RegularExpressionMatching

Implement regular expression matching with support for . and *.

. Matches any single character.
* Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true

---

public class Solution {
    public boolean isMatch(String s, String p) {
        if(p.length()==0) return s.length()==0;

        int n = p.length();

        if(n>1 && p.charAt(1) == *)
        {
            while( s.length()!=0 && (s.charAt(0) == p.charAt(0) || p.charAt(0)==.))
            {
                if(isMatch(s,p.substring(2)))    return true;
                s=s.substring(1);
            }
            return isMatch(s,p.substring(2));
        }
        else
        {
            if(s.length()!=0 && (s.charAt(0) == p.charAt(0) || p.charAt(0)==.))
                return isMatch(s.substring(1),p.substring(1));
            return false;
        }
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedArray
=======================================


RemoveDuplicatesfromSortedArray

Given a sorted array, remove the duplicates in place such that each element appear
only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,
Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].
---

public class Solution {
    public int removeDuplicates(int[] A) {
        int n = A.length;
        if(n<=1) return n;

        int i = 1, j=i;
        while(j<n)
        {
            if(A[j]!=A[j-1])
            {
                A[i++] = A[j];
            }
            j++;
        }
        return i;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedArrayII
=======================================

```

RemoveDuplicatesfromSortedArrayII

Follow up for "Remove Duplicates":
What if duplicates are allowed at most twice?

For example,
Given sorted array A = [1,1,1,2,2,3],

Your function should return length = 5, and A is now [1,1,2,2,3].

```
public class Solution {
    public int removeDuplicates(int[] A) {

        int n = A.length;
        if(n<=2) return n;

        int i=1, j=0;
        int count =1;
        int val = A[0];

        while(i<n)
        {
            if(A[i] == A[i-1])
            {
                count++;
                if(count>2)
                {
                    break;
                }
            }
            else break;
            i++;
        }

        j=i;
        while(j<n)
        {
            if(A[j]!=val){ count =1; val = A[j]; A[i++] = A[j];}
            else
            {
                count++;
                if(count<=2)
                {
                    A[i++]=A[j];
                }
            }
            j++;
        }
        return i;

    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedList
==========

RemoveDuplicatesfromSortedList

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if(head == null || head.next ==null) return head;

        ListNode newHead = head, newTail = head;
        int tmp = head.val;
        head = head.next;
        newHead.next = null;

        while(head!=null)
        {
            if(head.val != tmp)
            {
                tmp = head.val;
                newTail.next = head;
                newTail = head;
                head = head.next;
                newTail.next = null;
            }else
            {
                ListNode t = head;
                head = head.next;
```

```
                t.next = null;
            }
        }
        return newHead;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedListII
========================================


RemoveDuplicatesfromSortedListII

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only
distinct numbers from the original list.

For example,
Given 1->2->3->3->4->4->5, return 1->2->5.
Given 1->1->1->2->3, return 2->3.
---
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
     public  ListNode remove(ListNode cur)
     {
         if(cur == null || cur.next == null) return cur;
         int val = cur.val;
         while(cur!=null)
         {
             ListNode t = cur.next;
             if(cur.val == val)
             {
                 cur.next = null;
                 cur = t;
             }else
             {
                 return cur;
             }
         }
         return null;
     }
    public ListNode deleteDuplicates(ListNode head) {
        if(head == null) return head;

        ListNode cur = head;
        head = null;
        ListNode tail = null;

        while(cur!=null)
        {
            if(cur.next == null || cur.val != cur.next.val)
            {
                ListNode t = cur.next;
                cur.next = null;
                if(tail == null)
                {
                    head = cur;
                    tail = cur;
                }
                else
                {
                    tail.next = cur;
                    tail = cur;
                }
                cur = t;
            }else
            {
                cur = remove(cur);
            }
        }
        return head;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/RemoveElement
========================================


RemoveElement

Given an array and a value, remove all instances of that value in place and return
the new length.

The order of elements can be changed. It doesnt matter what you leave beyond the
new length.

---
public class Solution {
    public int removeElement(int[] A, int elem) {

        int n=A.length;
```

```
        if(n==0) return 0;
        int i=0, j=0;
        while(i<n && A[i]!=elem) i++;
        if(i==n) return n;

        j = i;
        for(;j<n;j++)
        {
            if(A[j]==elem) continue;
            A[i++] = A[j];
        }
        return i;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/RemoveNthNodeFromEndofList
========================================


RemoveNthNodeFromEndofList

Given a linked list, remove the nth node from the end of list and return its head.

For example,

   Given linked list: 1->2->3->4->5, and n = 2.

   After removing the second node from the end, the linked list becomes 1->2->3->5.
---

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {

        if(head == null) return head;
        if(head.next == null)
        {
            if(n==1) return null;
            else return head;
        }
        int count = 0;
        ListNode cur = head, second = head, prev = null;

        while(cur.next !=null)
        {
            count++;
            cur = cur.next;
            if(count>=n)
            {
                prev = second;
                second = second.next;
            }
        }
        if(count <n-1) return head;

        if(prev == null)
        {
            head = second.next;
            second.next = null;
        }
        else
        {
            prev.next = second.next;
            second.next = null;
        }
        return head;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/ReorderList
========================================


ReorderList

Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes values.

For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.

---
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
```

```
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public static ListNode reverse(ListNode head)
    {
        ListNode cur = head;
        head = null;
        while(cur!=null)
        {
            ListNode t = cur.next;
            cur.next = head;
            head = cur;
            cur = t;
        }
        return head;
    }

    public static ListNode merge(ListNode n1, ListNode n2)
    {
        if(n1==null) return n2;
        if(n2 == null) return n1;

        ListNode head = null, tail = null;
        while(n1 != null && n2 !=null)
        {
            if(head == null)
            {
                head = n1;
                tail = n1;
                n1 = tail.next;
                tail.next = null;
            }
            else
            {
                tail.next = n1;
                tail = n1;
                n1 = n1.next;
                tail.next = null;
            }
                tail.next = n2;
                tail = n2;
                n2 = n2.next;
                tail.next = null;
        }
        if(n1!=null && tail!=null) tail.next = n1;
        return head;
    }

    public static void reorderList(ListNode head) {
        if(head == null || head.next == null) return;

        ListNode left = head, right = head;
        int count=0;
        while(right.next != null)
        {
            count++;
            right = right.next;
            if(count%2==0) left = left.next;
        }
        right = left.next;
        left.next = null;
        head = merge(head, reverse(right));
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/ReplaceStringWithPattern
========================================


Replace all occurrence of the given pattern to 'X'.
For example, given that the pattern="abc", replace
"abcdeffdfegabcabc" with "XdeffdfegX". Note that multiple
occurrences of abc's that are contiguous will be replaced
with only one 'X'

===

class Solution
{
    public boolean validate(String s, String p)
    {
        int m = s.length(), n = p.length();
        if(m<n) return false;

        int i=0;
        for(char c: p.toCharArray())
        {
            if(c!=s.charAt(i++)) return false;
        }
        return true;
    }

    public String replace(String s, String p)
    {
        int m = s.length(), n = p.length();
        if(m<n) return s;

```

```
        char[] a = s.toCharArray(), b = p.toCharArray();

        int i=0, j=0;

        while(j<m)
        {
            if(j+n<=m &&  validate(s.substring(j,j+n), p) )
            {
                a[i++] = X;
                j+=n;
            }
            else
                a[i++] = a[j++];
        }
        return new String(a).substring(0, i);
    }

}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/RestoreIPAddresses
=======================================


RestoreIPAddresses

Given a string containing only digits, restore it by returning all possible valid
IP address combinations.

For example:
Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

---

public class Solution {
     public static boolean valid(String s)
     {
         int n = s.length();
         if(n==0) return false;
         try{
             int re = Integer.parseInt(s);
             if(n>3 || (n !=1 && s.charAt(0)==0) || re>255 || re<0) return false;
             return true;
         }
         catch(Exception o)
         {
             return false;
         }
     }

     public static List<String> restoreIpAddresses(String s, int k, HashMap<Integer, List<String>> map)
     {
         int n = s.length();
         int key = 4*n+k;
         if(map.containsKey(key)) return map.get(key);

         List<String> re = new ArrayList<String>();
         if(n==0) return re;

         if(k==1)
         {
             if(valid(s))
             {
                 re.add(s);
             }
             return re;
         }

         for(int i=n-1; i>=n-3;i--)
         {
             if(i<0) break;
             if(valid(s.substring(i)))
             {
                 List<String> r = restoreIpAddresses(s.substring(0,i), k-1, map);
                 if(r.size() == 0) continue;
                 for(String ss:r)
                 {
                     re.add(ss+"."+s.substring(i));
                 }
             }
         }
         map.put(key, re);
         return re;
     }

    public static List<String> restoreIpAddresses(String s) {
        List<String> re = new ArrayList<String>();

        int n = s.length();
        if(n==0) return re;
        return restoreIpAddresses(s, 4, new HashMap<Integer, List<String>>());
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/ReverseInteger
=======================================


ReverseInteger
```

Reverse digits of an integer.

Example1: x = 123, return 321
Example2: x = -123, return -321

--

```
public class Solution {
    public int reverse(int x) {
        if(x>-10 && x<10) return x;

        boolean neg = false;
        if(x<0){ x = -x; neg = true; }

        long re = 0;
        while(x!=0)
        {
            re = 10*re + (long)(x%10);
            x/=10;
        }
        if(neg) re=(-1)*re;
        if(x>Integer.MAX_VALUE) return Integer.MAX_VALUE;
        if(x<Integer.MIN_VALUE) return Integer.MIN_VALUE;
        return (int) re;
    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/ReverseLinkedListII
==========

ReverseLinkedListII

Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:
Given 1->2->3->4->5->NULL, m = 2 and n = 4,

return 1->4->3->2->5->NULL.

Note:
Given m, n satisfy the following condition:
1 ≤ m ≤ n ≤ length of list.

--

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode reverse(ListNode nd)
    {
        if(nd == null || nd.next == null) return nd;
        ListNode newHead = null, cur = nd;

        while(cur!=null)
        {
            ListNode tmp = cur.next;
            cur.next = newHead;
            newHead = cur;
            cur = tmp;
        }
        return newHead;
    }

    public ListNode reverseBetween(ListNode head, int m, int n) {
        if(head == null) return null;
        if(n<=m) return head;

        ListNode cur = head;
        int count =0;

        while(cur.next!=null && count <n-m)
        {
            count++;
            cur = cur.next;
        }
        if(count!=n-m) return head;

        ListNode sec = head, prev = null;

        count = 1;
        while(count<m && cur.next!=null)
        {
            count++;
            cur = cur.next;
            prev = sec;
            sec = sec.next;
        }
        if(count!=m) return head;


        ListNode rd = cur.next;
        cur.next = null;
```

```
        if(prev == null)    head = reverse(sec);
        else    prev.next = reverse(sec);
        sec.next = rd;
        return head;
    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/ReverseNodesink-Group
======================================


ReverseNodesink-Group

Given a linked list, reverse the nodes of a linked list k at a time and
return its modified list.

If the number of nodes is not a multiple of k then left-out nodes in the end should
emain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,
Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

---

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {

        if(k<=1) return head;
        if(head == null) return head;
        ListNode start = head;

        head = null;
        ListNode tail = null;

        while(true)
        {
            ListNode cur = start;
            int count = 1;
            while(cur.next != null && count<k)
            {
                count++; cur = cur.next;
            }
            if(count<k)
            {
                if(tail == null)
                {
                    head = start;
                    break;
                }
                else
                {
                    tail.next = start;
                    tail = cur;
                    start = cur.next;
                    tail.next = null;
                }
            }
            else
            {
                ListNode newStart = cur.next;
                ListNode prev = null;
                cur.next = null;
                cur = start;
                start = null;
                while(cur!=null)
                {
                    if(start == null) prev = cur;
                    ListNode t = cur.next;
                    cur.next = start;
                    start = cur;
                    cur = t;
                }
                if(tail == null)
                {
                    head = start;
                    tail = prev;
                }
                else
                {
                    tail.next = start;
                    tail = prev;
                }
```

```
                start = newStart;
            }
            if(start==null) break;
        }
        return head;

    }
}


====

Using function to do reverse, cleaner:

    public ListNode reverse(ListNode head)
    {
        if(head == null || head.next == null) return head;
        ListNode cur = head, newHead = null;

        while(cur!=null)
        {
            ListNode t = cur.next;
            cur.next = newHead;
            newHead = cur;
            cur = t;
        }
        return newHead;
    }

    public ListNode reverseKGroup(ListNode head, int k)
    {
        if(k<=1) return head;
        ListNode start = head, tail = null;
        head = null;

        while(start!=null)
        {
            ListNode cur = start;
            int count = 1;
            while(cur.next!=null && count <k)
            {
                count++;
                cur = cur.next;
            }
            if(count <k)
            {
                if(tail == null)
                {
                    head = start;
                }
                else
                {
                    tail.next = start;
                }
                break;
            }
            else
            {
                ListNode t = cur.next;
                cur.next = null;
                ListNode rev = reverse(start);
                if(tail == null)
                {
                    head = rev; tail = start;
                }
                else
                {
                    tail.next = rev;
                    tail = start;
                }
                start = t;
            }
        }
        return head;
    }




========================================
/Users/linxie/Documents/github/leetcodeHub/code/ReverseWordsinaString
========================================

ReverseWordsinaString

Given an input string, reverse the string word by word.

For example,
Given s = "the sky is blue",
return "blue is sky the".

----

public class Solution {
    public void reverse(char[] s, int l, int r)
    {
        while(l<r)
        {
            char c = s[l];
            s[l] = s[r];
            s[r] = c;
            l++; r--;
```

```
        }
    }
    public String reverseWords(String s) {
        char[] c = s.toCharArray();
        int n = c.length;

        int i=0, j=0;
        for(;j<n;j++)
        {
            while(j<n && c[j] ==  ) j++;
            if(j==n) break;

            if(i!=0)    c[i++]=  ;
            while(j<n && c[j]!= ) c[i++] = c[j++];
            if(j==n) break;
        }
        if(i==0) return "";
        n=i;
        reverse(c, 0, i-1);
        i=0; j=0;
        for(;j<n;j++)
        {
            i=j;
            for(;j<n && c[j]!= ;j++);
            reverse(c,i,j-1);
        }
        char[] newC = new char[n];
        for(i=0; i<n;i++)
            newC[i] = c[i];
        return new String(newC);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/RomantoInteger
========================================


RomantoInteger

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

---

public class Solution {
    public int romanToInt(String s) {
        int n = s.length();
        if(n==0) return 0;

        HashMap<Character, Integer> m = new HashMap<Character, Integer>();
        m.put(M, 1000);
        m.put(D, 500);
        m.put(C, 100);
        m.put(L,50);
        m.put(X, 10);
        m.put(V, 5);
        m.put(I,1);
        char[] c = s.toCharArray();
        int re = m.get(c[0]);
        for(int i=1;i<n;i++)
        {
            int pre = m.get(c[i-1]);
            int cur = m.get(c[i]);
            if(cur>pre)
                re+=(cur-2*pre);
            else re+=cur;
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/RotateImage
========================================


RotateImage

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

---
public class Solution {
    public void rotate(int[][] matrix) {

        int m=matrix.length;
        if(m==0) return;
        int n = matrix[0].length;
        if(n==0) return;

        int layer = (Math.min(m,n)+1)/2;
        for(int i=0;i<layer;i++)
        {
            for(int j=i;j<n-i-1;j++)
            {
                int t = matrix[n-1-j][i];
                matrix[n-1-j][i] = matrix[n-i-1][n-1-j];
                matrix[n-i-1][n-1-j] = matrix[j][n-i-1];
                matrix[j][n-i-1] = matrix[i][j];
```

```
                matrix[i][j] = t;
            }
        }
        return;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/RotateList
========================================


RotateList

Given a list, rotate the list to the right by k places, where k is non-negative.

For example:
Given 1->2->3->4->5->NULL and k = 2,
return 4->5->1->2->3->NULL.

---

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode rotateRight(ListNode head, int n) {
        if(head == null) return null;
        if(n==0) return head;

        int count = 0;
        ListNode first = head, second = head;

        while(first.next!=null)
        {
            count++;
            first = first.next;
            if(count > n)    second = second.next;
        }

        if(count<n) return rotateRight(head, n%(count+1));

        ListNode newHead = second.next;
        second.next = null;
        first.next = head;
        return newHead;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SameTree
========================================


SameTree

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the
nodes have the same value.

---
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if(p==null)
            return q==null;
        if(q==null) return false;
        return p.val == q.val && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/ScrambleString
========================================


ScrambleString

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty
substrings recursively.

Below is one possible representation of s1 = "great":

```

```
    great
   /    \
  gr    eat
 / \    /  \
g   r  e   at
           / \
          a   t
```
To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".

```
    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t
```
We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".

```
    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta  e
       / \
      t   a
```
We say that "rgtae" is a scrambled string of "great".

Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

```
class Pair
{
    String s1;
    String s2;
    public Pair(String _s1, String _s2)
    {
        s1 = _s1; s2 = _s2;
    }

    public int hashCode()
    {
        return 31*s1.hashCode() + s2.hashCode();
    }
    public boolean equals(Object o)
    {
        if(o!=null && o instanceof Pair)
        {
            Pair _o = (Pair)o;
            return (s1.equals(_o.s1) && s2.equals(_o.s2)) ||  (s2.equals(_o.s1) && s1.equals(_o.s2));
        }
        return false;
    }
}

public class Solution {
    public boolean isScramble(String s1, String s2, HashMap<Pair, Boolean> map)
    {
        int n = s1.length();
        if(n!=s2.length()) return false;
        if(s1.equals(s2)) return true;

        Pair p = new Pair(s1, s2);
        if(map.containsKey(p)) return map.get(p);

        for(int i=1; i< n; i++)
        {
            if(( isScramble(s1.substring(0, i), s2.substring(0,i), map)
                && isScramble(s1.substring(i), s2.substring(i), map) )
                || ( isScramble(s1.substring(0, i), s2.substring(n-i), map)
                    && isScramble(s1.substring(i), s2.substring(0, n-i), map)) )
            {
                map.put(p,true);
                return true;
            }
        }
        map.put(p,false);
        return false;
    }

    public boolean isScramble(String s1, String s2) {
        return isScramble(s1, s2, new HashMap<Pair, Boolean>());
    }
}
```

```
========================================
/Users/linxie/Documents/github/leetcodeHub/code/Searcha2DMatrix
========================================
```

Searcha2DMatrix

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right.
The first integer of each row is greater than the last integer of the previous row.
For example,

Consider the following matrix:

```
[
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
Given target = 3, return true.


---
public class Solution {
     public int findRight(int[] a, int l, int r, int e)
     {
         int re = -1;
         if(r-l<0 || r-l>=a.length) return re;
         while(l<=r)
         {
             int mid = l +(r-l)/2;
             if(a[mid] == e) return mid;
             else if(a[mid]>e)
             {
                 re = mid;
                 r = mid-1;
             }
             else
                 l = mid+1;
         }
         return re;
     }

    public boolean searchMatrix(int[][] matrix, int target)
    {
        int m = matrix.length;
        if(m==0)    return false;
        int n = matrix[0].length;
        if(n==0) return false;

        int[] b = new int[m];
        for(int i=0; i<m; i++)
        {
            b[i] = matrix[i][n-1];
        }
        int r = findRight(b, 0, m-1, target);
        if(r == -1) return false;
        if(b[r] == target) return true;

        int rr = findRight(matrix[r], 0, n-1, target);
        if(rr == -1 || matrix[r][rr]!=target) return false;
        return true;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SearchforaRange
=======================================


SearchforaRange

Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithms runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example,
Given [5, 7, 7, 8, 8, 10] and target value 8,
return [3, 4].

---

public class Solution {
    public int searchLeft(int[] A, int target)
    {
        int n = A.length;
        int re = -1;
        int l = 0, r = n-1;
        while(l<=r)
        {
            int mid = l+(r-l)/2;
            if(A[mid] == target)
            {
                re = mid;
                r = mid-1;
            }
            else if(target < A[mid]) r = mid-1;
            else l = mid+1;
        }
        return re;
    }

    public int searchRight(int[] A, int target)
    {
        int n = A.length;
        int re = -1;
        int l = 0, r = n-1;
        while(l<=r)
        {
            int mid = l+(r-l)/2;
            if(A[mid] == target)
```

```
            {
                re = mid;
                l = mid+1;
            }
            else if(target < A[mid]) r = mid-1;
            else l = mid+1;
        }
        return re;
    }

    public int[] searchRange(int[] A, int target)
    {
        int n = A.length;
        int left = searchLeft(A, target);
        int right = searchRight(A, target);
        int[] re = {left, right};
        return re;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SearchinRotatedSortedArray
=======================================


SearchinRotatedSortedArray

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

---

public class Solution {
    public int search(int[] A, int target) {

        int n = A.length;
        if(n==0) return -1;
        if(n==1)
            if(A[0] == target) return 0;
            else return -1;

        int l = 0, r = n-1;

        while(l<=r)
        {
            int mid = l+(r-l)/2;
            if(A[mid] == target) return mid;

            if(A[l]<A[mid])
            {
                if(target >= A[l] && target < A[mid]) r = mid-1;
                else l = mid+1;
            }
            else if (A[l] > A[mid])
            {
                if(target > A[mid] && target <= A[r]) l = mid+1;
                else r = mid-1;
            }
            else l++;
        }
        return -1;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SearchinRotatedSortedArrayII
=======================================


SearchinRotatedSortedArrayII

Follow up for "Search in Rotated Sorted Array":
What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

---

public class Solution {
    public boolean search(int[] A, int target) {

        int n = A.length;
        if(n==0) return false;
        if(n==1) return A[0] == target;

        int l=0, r = n-1;

        while(l<=r)
        {
            int mid = l+(r-l)/2;
            if(A[mid] == target) return true;
            if(A[mid]<A[l])
            {
                if(target>A[mid] && target<=A[r]) l=mid+1;
```

```
                else r=mid-1;
            }
            else if(A[mid]>A[l])
            {
                if(target>=A[l] && target<A[mid]) r = mid-1;
                else l = mid+1;
            }
            else l++;
        }
        return false;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SearchInsertPosition
========================================


SearchInsertPosition

Given a sorted array and a target value, return the index if the target is found. If not,
return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0

--

public class Solution {
    public int searchInsert(int[] A, int target) {
        int n = A.length;
        int l = 0, r = n-1;
        int re = -1;

        while(l<=r)
        {
            int mid = l+(r-l)/2;
            if(A[mid] == target)
            {
                re = mid; break;
            }
            else if(A[mid]<target)
            {
                l = mid+1;
                re = l;
            }
            else
            {
                re = mid;
                r = mid-1;
            }
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SerializeAndDeserializeBT
========================================


SerializeAndDeserializeBT

pre-order

======
class Node
{
    String str;
    Node(String _str){ str = _str;}
}

class Solution
{

    public String serialize(TreeNode p)
    {
        if(p == null) return "#";
        String re = Integer.toString(p.val);
        re+=serialize(p.left);
        re+=serialize(p.right);
        return re;
    }

    public TreeNode deserialize(Node s)
    {
        if(s.str.length() == 0) return null;
        if(s.str.charAt(0)==#)
        {
            s.str = s.str.substring(1);
            return null;
        }
        TreeNode root = new TreeNode((int)(s.str.charAt(0)-0));
        s.str=s.str.substring(1);
        root.left = deserialize(s);
        root.right = deserialize(s);
```

```
        return root;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SetMatrixZeroes
=======================================


SetMatrixZeroes

Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place.

---
public class Solution {
    public void setZeroes(int[][] matrix) {

        int m=matrix.length;
        if(m==0) return;
        int n = matrix[0].length;
        if(n==0) return;

        boolean col =false, row = false;
        for(int i:matrix[0])
            if(i==0)
            {
                row = true;
                break;
            }
        for(int i=0;i<m;i++)
            if(matrix[i][0] == 0)
            {
                col = true;
                break;
            }

        for(int i=1; i<m;i++)
            for(int j=1;j<n;j++)
                if(matrix[i][j] == 0)
                {
                    matrix[0][j] = 0;
                    matrix[i][0] = 0;
                }

        for(int i=1;i<m;i++)
            for(int j=1;j<n;j++)
                if(matrix[i][0]==0 || matrix[0][j]==0)
                    matrix[i][j] = 0;
        if(row)
            for(int i=0;i<n;i++) matrix[0][i] = 0;
        if(col)
            for(int i=0;i<m;i++) matrix[i][0] = 0;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SimplifyPath
=======================================


SimplifyPath

Given an absolute path for a file (Unix-style), simplify it.

For example,
path = "/home/", => "/home"
path = "/a/./b/../../c/", => "/c"
click

--

public class Solution {
    public String simplifyPath(String path) {
        int n = path.length();
        if(n==0) return "";

        Stack<String> s = new Stack<String>();
        int i=0, j=0;
        String re = "";
        while(i<n)
        {
            while(i<n && path.charAt(i)==/) i++;
            if(i==n) break;

            j = i;
            while(j<n && path.charAt(j)!=/) j++;

            String val = path.substring(i,j);
            if(val.equals(".."))
            {
                if(!s.isEmpty()) s.pop();
            }
            else if(!val.equals(".")) s.push(val);
            i = j;
        }
        if(s.isEmpty()) re = "/";
        else
        {
            while(!s.isEmpty())
            {
                re = "/"+s.pop()+re;
```

```
            }
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SingleNumber
========================================


SingleNumber

Given an array of integers, every element appears twice except for one. Find that single one.

Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?
---
public class Solution {
    public int singleNumber(int[] A) {
        if(A.length==0) System.exit(-1);
        int re=0;
        for(int i:A)
            re^=i;
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SingleNumberII
========================================


SingleNumberII

Given an array of integers, every element appears three times except for one. Find that single one.

Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?
---

public class Solution {
    public int singleNumber(int[] A) {
        int n = A.length;
        if(n==1) return A[0];
        if(n==0 || n%3!=1) return 0;

        int re=0;
        for(int i=0;i<32;i++)
        {
            int count = 0;
            for(int j=0; j<n; j++)
            {
                if((A[j] & (1<<i))!=0) count++;
            }
            if(count%3!=0) re = re|(1<<i);
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SortColors
========================================


SortColors

Given an array with n objects colored red, white or blue, sort them so that objects of
the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
---

public class Solution {
    public void sortColors(int[] A) {

        int n = A.length;
        if(n<=1) return;

        int i=0, k=n-1;
        while(i<n && A[i]==0) i++;
        if(i==n) return;
        while(k>=0 && A[k]==2) k--;
        if(k<0) return;

        int j = i;
        while(j<=k)
        {
            if(A[j]==1) j++;
            else if(A[j]==0)
            {
                int t = A[i];
                A[i] = A[j];
                A[j] = t;
                i++;
            }else
            {
                int t = A[k];
                A[k] = A[j];
                A[j] = t;
```

```
                k--;
            }
            if(j<i) j++;
        }

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SortList
=======================================


SortList

Sort a linked list in O(n log n) time using constant space complexity.
---
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
     public static ListNode merge(ListNode n1, ListNode n2)
     {
         if(n1==null) return n2;
         if(n2==null) return n1;

         ListNode head = null, tail = null;
         while(n1!=null && n2!=null)
         {
             if(n1.val <=n2.val)
             {
                 if(tail == null)
                 {
                     head = n1;
                     tail = head;
                     n1 = n1.next;
                     tail.next = null;
                 }
                 else
                 {
                     tail.next = n1;
                     tail = n1;
                     n1=n1.next;
                     tail.next = null;
                 }
             }
             else
             {

                 if(tail == null)
                 {
                     head = n2;
                     tail = head;
                     n2 = n2.next;
                     tail.next = null;
                 }
                 else
                 {
                     tail.next = n2;
                     tail = n2;
                     n2=n2.next;
                     tail.next = null;
                 }

             }
         }
         if(n1!=null)
         {
             tail.next = n1;
         }
         if(n2!=null)
         {
             tail.next = n2;
         }
         return head;
     }

    public static ListNode sortList(ListNode head) {
        if(head == null || head.next == null) return head;

        int count =0;
        ListNode l = head, r = head;
        while(l.next!=null)
        {
            count++;
            l = l.next;
            if(count%2==0) r = r.next;
        }
        l = r.next;
        r.next = null;
        return merge(sortList(head), sortList(l));
    }
}


```

```
======================================
/Users/linxie/Documents/github/leetcodeHub/code/SpiralMatrix
======================================


SpiralMatrix

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix
in spiral order.

For example,
Given the following matrix:

[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
You should return [1,2,3,6,9,8,7,4,5].
---

public class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> re = new ArrayList<Integer>();
        int m = matrix.length;
        if(m==0) return re;
        int n = matrix[0].length;
        if(n==0) return re;

        int layer = (Math.min(m,n)+1)/2;
        int count =0;
        for(int i=0;i<layer;i++)
        {
            for(int j=i; j<n-i; j++)
            {
                re.add(matrix[i][j]);
                if(++count >= m*n) break;
            }
            if(count >= m*n) break;
            for(int j=i+1; j<m-i; j++)
            {
                re.add(matrix[j][n-i-1]);
                if(++count >= m*n) break;
            }
            if(count >= m*n) break;
            for(int j=n-i-2; j>=i; j--)
            {
                re.add(matrix[m-i-1][j]);
                if(++count >= m*n) break;
            }
            if(count >= m*n) break;
            for(int j=m-i-2; j>i; j--)
            {
                re.add(matrix[j][i]);
                if(++count >= m*n) break;
            }
            if(count >= m*n) break;
        }
        return re;


    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/SpiralMatrixII
======================================


SpiralMatrixII

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example,
Given n = 3,

You should return the following matrix:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
---

public class Solution {
    public int[][] generateMatrix(int n) {

        int[][] re = new int[n][n];
        if(n==0) return re;

        int layer = (n+1)/2;
        int in = 1;

        for(int i = 0; i<layer; i++)
        {
            for(int j = i; j<n-i; j++)    re[i][j] = (in++);
            for(int j = i+1; j<n-i; j++)    re[j][n-i-1] = (in++);
            for(int j = n-i-2; j>=i; j--)    re[n-i-1][j] = (in++);
            for(int j = n-i-2; j>i; j--)    re[j][i] = (in++);
        }
        return re;


```

```
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/Sqrt(x)
========================================


Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of x.


--
public class Solution {
    public int sqrt(int x) {

        if(x<0) return -1;
        if(x<=1) return x;

        int l = 1, r=x;
        int re = -1;

        while(l<=r)
        {
            int mid = l+(r-l)/2;
            long result = (long)mid*(long)mid;
            if(result==x) return mid;
            else if(result < x)
            {
                re = mid; l = mid+1;
            }
            else
            {
                r = mid-1;
            }
        }
        return re;

    }
}

===
Newton for double sqrt

public static double sqrt(double c)
{
    if(c<0) return Double.NaN;
    double EPS = 1E-15;

    double t = c;
    while(Math.abs(t-c/t) > EPS*t)
        t = (c/t+t)/2.0;
    return t;
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/StringtoInteger
========================================


StringtoInteger

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see
below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You
are responsible to gather all the input requirements up front
--

public class Solution {
    public int atoi(String str) {

        char[] s = str.toCharArray();
        int n = s.length;
        if(n==0) return 0;

        boolean  number = false;
        int sign = 0;

        long re = 0;
        for(char c : s)
        {
            if(!number && (sign == 0) && c== ) continue;

            int num = (int)(c-0);

            if(num<0 || num>9)
            {
                if( sign!=0 || (c!=+ && c!=-) ) break;

                if(c == -) sign = -1;
                else sign =1;
            }
            else
            {
                if(sign == 0) sign =1;
                number = true;
```

```
                re = re*10+(long)num;
            }
        }
        if(sign == -1) re = -re;
        if(re > Integer.MAX_VALUE) return Integer.MAX_VALUE;
        if(re < Integer.MIN_VALUE) return Integer.MIN_VALUE;
        return (int)re;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/Subsets
=======================================


Subsets

Given a set of distinct integers, S, return all possible subsets.

Note:
Elements in a subset must be in non-descending order.
The solution set must not contain duplicate subsets.
For example,
If S = [1,2,3], a solution is:

[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]---

public class Solution {

    public void subsets(int[] S, int ind, List<List<Integer>> re, List<Integer> r)
    {
        if(ind >= S.length)
        {
            List<Integer> rr = new ArrayList(r);
            re.add(rr);
            return;
        }
        int tmp = S[ind];

        int siz = r.size();
        int end = ind;
        while(end<S.length && S[end] == tmp)    end++;
        subsets(S, end, re, r);

        for(int i=ind; i<end; i++)
        {
            r.add(tmp);
            subsets(S, end, re, r);
        }

        for(int i=r.size()-1; i>=siz; i--)
            r.remove(i);
    }

    public List<List<Integer>> subsets(int[] S) {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = S.length;
        if(n==0) return re;

        List<Integer> r = new ArrayList<Integer>();

        Arrays.sort(S);
        subsets(S, 0, re, r);
        return re;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SubsetsII
=======================================


SubsetsII

Given a collection of integers that might contain duplicates, S, return all possible subsets.

Note:
Elements in a subset must be in non-descending order.
The solution set must not contain duplicate subsets.
For example,
If S = [1,2,2], a solution is:

[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
---
```

```
public class Solution {
    public void  subsetsWithDup(int[] num, int k, List<List<Integer>> re, List<Integer> r)
    {
        int n = num.length;
        if(k==n)
        {
            List<Integer> l = new ArrayList<Integer>(r);
            re.add(l);
            return;
        }

        int val = num[k];
        int i = k;
        while(i<n && num[i]==val) i++;

        int siz = r.size();
        for(int j=k; j<=i;j++)
        {
            subsetsWithDup(num, i, re, r);
            r.add(val);
        }
        int siz2 = r.size();
        for(int j=siz2-1;j>=siz;j--)
            r.remove(j);
    }

    public List<List<Integer>> subsetsWithDup(int[] num)
    {
        List<List<Integer>> re = new ArrayList<List<Integer>>();
        int n = num.length;
        if(n==0) return re;
        Arrays.sort(num);
        subsetsWithDup(num, 0, re, new ArrayList<Integer>());
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SubstringwithConcatenationofAllWords
========================================


SubstringwithConcatenationofAllWords

You are given a string, S, and a list of words, L, that are all of the same length. Find all
starting indices of substring(s) in S that is a concatenation of each word in L exactly once
and without any intervening characters.

For example, given:
S: "barfoothefoobarman"
L: ["foo", "bar"]

You should return the indices: [0,9].
(order does not matter).


---

public class Solution {
    public boolean helper(String s, HashMap<String, Integer> map, int k)
    {
        int m = map.size();
        if(m==0)
            if(s.length()==0) return true;
            else return false;

        int n=s.length();
        for(int i=0;i<=n-k;i+=k)
        {
            String str = s.substring(i,i+k);
            if(map.containsKey(str) && map.get(str) > 0)
            {
                map.put(str, map.get(str)-1);
                if(map.get(str)==0) m--;
            }else return false;
        }
        return m==0;
    }

    public List<Integer> findSubstring(String S, String[] L)
    {
        List<Integer> re =  new ArrayList<Integer>();
        int n = S.length();
        if(L.length == 0) return re;
        if(n<L.length*L[0].length()) return re;

        HashMap<String, Integer> map = new HashMap<String, Integer>();
        for(String s:L)
        {
            if(map.containsKey(s))
                map.put(s, map.get(s)+1);
            else
                map.put(s,1);
        }

        int total = L.length*L[0].length();
        for(int i=0;i<=n-total;i++)
        {
            if( helper(S.substring(i,i+total), new HashMap<String, Integer>(map), L[0].length()) )
                re.add(i);
        }
        return re;
    }
```

```
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/Sudoku Solver
========================================


Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character ..

You may assume that there will be only one unique solution.

---

public class Solution {
    public boolean valid(char[][] board, int i, int j)
    {
        for(int k=0;k<9;k++)
            if(k!=i && board[k][j] == board[i][j]) return false;
        for(int k=0;k<9;k++)
            if(k!=j && board[i][k] == board[i][j]) return false;
        int c = i/3, d = j/3;
        int ii = i%3, jj=j%3;

        for(int k=0;k<3;k++)
            for(int v = 0; v<3; v++)
            {
                if((c*3+k !=i && d*3+v!=j) && board[c*3+k][d*3+v] == board[i][j]) return false;
            }
        return true;
    }

    public boolean solveSudoku(char[][] board, int i, int j)
    {
        int n = board.length;
        if(i==n) return true;
        if(board[i][j]==.)
        {
            for(char k = 1;k<=9;k++)
            {
                board[i][j] = k;
                if(valid(board,i,j))
                {
                    if(j+1==n)
                    {
                        if(solveSudoku(board, i+1,0)) return true;
                    }
                    else
                    {
                        if(solveSudoku(board, i,j+1)) return true;
                    }
                }
            }
            board[i][j] =.;
        }
        else
        {
            if(j+1==n)
            {
                if(solveSudoku(board, i+1,0)) return true;
            }
            else
            {
                if(solveSudoku(board, i,j+1)) return true;
            }
        }
        return false;
    }

    public void solveSudoku(char[][] board)
    {
        solveSudoku(board, 0, 0);
    }

}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/SumRoottoLeafNumbers
========================================


SumRoottoLeafNumbers

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

For example,

    1
   / \
  2   3
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.

Return the sum = 12 + 13 = 25.
---
```

```java
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int sumNumbers(TreeNode root, int sum) {
        if(root == null) return 0;

        int re = sum*10 +root.val;
        if(root.left==null && root.right == null) return re;
        else    return sumNumbers(root.left, re) + sumNumbers(root.right, re);
    }

    public int sumNumbers(TreeNode root) {
        if(root == null) return 0;
        return sumNumbers(root, 0);
    }
}
```

```
=======================================
/Users/linxie/Documents/github/leetcodeHub/code/SurroundedRegions
=======================================
```

SurroundedRegions

Given a 2D board containing X and O, capture all regions surrounded by X.

A region is captured by flipping all Os into Xs in that surrounded region.

For example,
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:

X X X X
X X X X
X X X X
X O X X

```java
public class Solution {
     static void dfs(char[][] board, int i, int j){
         board[i][j] = D;
        int m = board.length;
        int n = board[0].length;
         if(i-1>=0 && board[i-1][j]==O)
             dfs(board, i-1, j);
         if(j-1>=0 && board[i][j-1]==O)
             dfs(board, i, j-1);
         if(i+1<m && board[i+1][j]==O)
             dfs(board, i+1, j);
        if(j+1<n && board[i][j+1]==O)
             dfs(board, i, j+1);

     }

      void solve(char[][] board) {

        int m = board.length;
        if(m==0) return;

        int n = board[0].length;
        if(n==0) return;

        for(int i=0; i<n; i++)
        {
            if(board[0][i] ==O)
                dfs(board, 0, i);
        }

        for(int i=0; i<m; i++)
        {
            if(board[i][n-1] ==O)
                dfs(board, i, n-1);
        }

        for(int i=n-1; i>=0; i--)
        {
            if(board[m-1][i] ==O)
                dfs(board, m-1, i);
        }
        for(int i=m-1; i>=0; i--)
        {
            if(board[i][0] ==O)
                dfs(board, i, 0);
        }
        for(int i=0;i<m;i++)
            for(int j=0;j<n;j++)
                if(board[i][j]==D)
                    board[i][j] = O;

    }
}
```

```
8319 ======================================
8320 /Users/linxie/Documents/github/leetcodeHub/code/SwapNodesinPairs
8321 ======================================
8322
8323
8324 SwapNodesinPairs
8325
8326 Given a linked list, swap every two adjacent nodes and return its head.
8327
8328 For example,
8329 Given 1->2->3->4, you should return the list as 2->1->4->3.
8330
8331 Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.
8332
8333 ---
8334
8335 /**
8336  * Definition for singly-linked list.
8337  * public class ListNode {
8338  *     int val;
8339  *     ListNode next;
8340  *     ListNode(int x) {
8341  *         val = x;
8342  *         next = null;
8343  *     }
8344  * }
8345  */
8346 public class Solution {
8347     public ListNode swapPairs(ListNode head) {
8348
8349         if(head == null || head.next == null) return head;
8350
8351         ListNode cur = head, tail = null;
8352         head = null;
8353
8354         while(cur!=null && cur.next !=null)
8355         {
8356             ListNode t = cur.next.next;
8357             cur.next.next = null;
8358
8359             if(head == null)
8360             {
8361                 head = cur.next;
8362                 tail = cur;
8363                 head.next = tail;
8364                 tail.next = null;
8365             }
8366             else
8367             {
8368                 tail.next = cur.next;
8369                 tail.next.next = cur;
8370                 cur.next = null;
8371                 tail = cur;
8372             }
8373             cur = t;
8374         }
8375         if(cur!=null) tail.next = cur;
8376         return head;
8377
8378     }
8379 }
8380
8381
8382 ======================================
8383 /Users/linxie/Documents/github/leetcodeHub/code/SymmetricTree
8384 ======================================
8385
8386
8387 SymmetricTree
8388
8389 Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).
8390
8391 For example, this binary tree is symmetric:
8392
8393     1
8394    / \
8395   2   2
8396  / \ / \
8397 3  4 4  3
8398 But the following is not:
8399     1
8400    / \
8401   2   2
8402    \   \
8403    3    3
8404 Note:
8405 Bonus points if you could solve it both recursively and iteratively.
8406
8407 ---
8408 /**
8409  * Definition for binary tree
8410  * public class TreeNode {
8411  *     int val;
8412  *     TreeNode left;
8413  *     TreeNode right;
8414  *     TreeNode(int x) { val = x; }
8415  * }
8416  */
8417 public class Solution {
8418     public boolean isSymmetric(TreeNode n1, TreeNode n2) {
8419         if(n1 == null)
8420             if(n2 == null)  return true;
8421             else return false;
8422         if(n2 == null) return false;
```

```
        if(n1.val == n2.val)
            return isSymmetric(n1.left, n2.right) && isSymmetric(n1.right, n2.left);
        return false;
    }
    public boolean isSymmetric(TreeNode root) {
        if(root == null) return true;
        return isSymmetric(root.left, root.right);
    }
}


======================================
/Users/linxie/Documents/github/leetcodeHub/code/TextJustification
======================================


TextJustification

Given an array of words and a length L, format the text such that each line has exactly L characters
and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line.
Pad extra spaces   when necessary so that each line has exactly L characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a
line do not divide evenly between words, the empty slots on the left will be assigned more spaces than
the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

For example,
words: ["This", "is", "an", "example", "of", "text", "justification."]
L: 16.

Return the formatted lines as:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
---

public class Solution {
    public List<String> fullJustify(String[] words, int L) {

        List<String> re = new ArrayList<String>();

        int cur = 0;
        int n = words.length;
        while(cur<n)
        {
            int end = cur;
            int res = 0;
            while(res<L)
            {
                if( end < n && res + words[end].length() <= L-end+cur)
                {
                    res+=words[end].length();
                    end++;
                }
                else
                {
                    break;
                }
            }
            int spaceNum = L-res;
            int num = end - cur;
            if(num == 0)
            {
                re.add("");
                return re;
            }
           String result = "";
            if(num==1)
            {
                result = words[cur];
                for(int i = 0; i<spaceNum; i++)
                    result +=" ";
            }
            else
            {
                String spaceLess = "";
                int i = 0;

                for(;i<spaceNum/(num-1); i++)
                    spaceLess += " ";

                int j = cur;
                if(end!=n)
                {
                    for(;j< cur+spaceNum%(num-1); j++)
                    {
                        result += words[j]+spaceLess+" ";
                    }
                    for(;j<end-1;j++)
                    {
                        result += words[j] + spaceLess;
                    }
                    result += words[j];
                }
                else
                {
```

```
                    for(;j<end-1;j++)
                    {
                        result += words[j] + " ";
                    }
                    result += words[j];
                    for(int k=result.length();k<L;k++)
                        result+=" ";
                }
            }

            re.add(result);
            cur = end;
        }
        return re;

    }
}



========================================
/Users/linxie/Documents/github/leetcodeHub/code/TrappingRainWater
========================================


TrappingRainWater

Given n non-negative integers representing an elevation map where the width of each bar is 1,
compute how much water it is able to trap after raining.

For example,
Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.

---

public class Solution {
    public int trap(int[] A) {
        int n = A.length;
        if(n<=2) return 0;

        int[] big = new int[n];
        big[0] = A[0];
        for(int i=1;i<n;i++)
            if(A[i]>=big[i-1]) big[i] = A[i];
            else    big[i] = big[i-1];

        int rightBig = A[n-1];
        int total = 0;
        for(int i=n-2; i>0; i--)
        {
            if(A[i] < rightBig)
            {
                if(A[i]<big[i]) total += Math.min(rightBig, big[i])-A[i];
            }
            else rightBig = A[i];
        }
        return total;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/Triangle
========================================


Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to
adjacent numbers on the row below.

For example, given the following triangle
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

Note:
Bonus point if you are able to do this using only O(n) extra space, where n is the total
number of rows in the triangle.

--

public class Solution {
    public int minimumTotal(List<List<Integer>> triangle) {

        int n = triangle.size();
        if(n==0) return 0;
        if(n==1)
        {
            int min=Integer.MAX_VALUE;
            for(int i:triangle.get(0))
                if(i<min) min = i;
            return min;
        }

        List<Integer> prev = triangle.get(n-1);
        for(int i=n-2; i>=0; i--)
        {
            List<Integer> cur = triangle.get(i);
            for(int j=0;j<cur.size();j++)
```

```
            {
                cur.set(j,Math.min(prev.get(j),prev.get(j+1))+cur.get(j));
            }
            prev = cur;
        }
        return prev.get(0);

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/TwoSum
========================================


TwoSum

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target,
 where index1 must be less than index2. Please note that your returned answers (both index1 and
    index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2

---

public class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int[] re = new int[2];
        int n = numbers.length;
        if(n<=1) return null;

        HashMap<Integer, Integer>m = new HashMap<Integer, Integer>();

        for(int i=0;i<n;i++)
        {
            int r = target - numbers[i];
            if(m.containsKey(r))
            {
                re[0]=m.get(r)+1;
                re[1]=i+1;
                break;
            }
            m.put(numbers[i],i);
        }
        return re;
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/UniqueBinarySearchTrees
========================================


UniqueBinarySearchTrees

Given n, how many structurally unique BSTs (binary search trees) that store values 1...n?

For example,
Given n = 3, there are a total of 5 unique BSTs.

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                \
   2     1         2                3

---
public class Solution {
    public int numTrees(int l, int r)
    {
        if(l==r) return 1;
        if(l>r) return 0;

        int re = 0;
        for(int i=l;i<=r;i++)
        {
            int left = numTrees(l,i-1);
            int right =numTrees(i+1, r);
            if(left == 0) re+=right;
            else if(right==0) re+=left;
            else re+=left*right;
        }
        return re;
    }
    public int numTrees(int n)
    {
        return numTrees(0,n-1);
    }

}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/UniqueBinarySearchTreesII
========================================


```

UniqueBinarySearchTreesII

Given n, generate all structurally unique BSTs (binary search trees) that store values 1...n.

For example,
Given n = 3, your program should return all 5 unique BSTs shown below.

```
   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; left = null; right = null; }
 * }
 */
public class Solution {

    public static List<TreeNode> generateTrees(HashMap<Integer, List<TreeNode>> map,int l, int r, int n) {
            int key = l*n+r;
            if(map.containsKey(key)) return map.get(key);

            List<TreeNode> re = new ArrayList<TreeNode>();
            if(l>r) return re;
            if(l==r)
            {
                TreeNode root = new TreeNode(l);
                re.add(root);
            }
            for(int i=l;i<=r;i++)
            {
                List<TreeNode> left = generateTrees(map, l, i-1, n);
                List<TreeNode> right = generateTrees(map, i+1, r, n);
                if(left.size() == 0)
                {
                    for(TreeNode nd : right)
                    {
                        TreeNode root = new TreeNode(i);
                        root.right = nd;
                        re.add(root);
                    }
                }
                else if(right.size() == 0)
                {
                    for(TreeNode nd: left)
                    {
                        TreeNode root = new TreeNode(i);
                        root.left    = nd;
                        re.add(root);
                    }
                }
                else
                {
                    for(TreeNode lnd:left)
                    {
                        for(TreeNode rnd:right)
                        {
                            TreeNode root = new    TreeNode(i);
                            root.left = lnd;
                            root.right = rnd;
                            re.add(root);
                        }
                    }
                }
            }
            map.put(key, re);
            return re;
    }
    public static List<TreeNode> generateTrees(int n) {
        List<TreeNode> re = new ArrayList<TreeNode>();
                if(n==0){
            re.add(null);
            return re;
        }
        if(n==1)
        {
            TreeNode root = new TreeNode(1);
            re.add(root);
            return re;
        }


        return generateTrees(new HashMap<Integer, List<TreeNode>>(), 1, n,n);

    }
}
```

==========
/Users/linxie/Documents/github/leetcodeHub/code/UniquePaths
==========

UniquePaths

A robot is located at the top-left corner of a m x n grid (marked Start in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked Finish in the diagram below).

How many possible unique paths are there?

```
public class Solution {
    public int uniquePaths(int m, int n) {
        if(m*n==0) return 0;
        int[] re = new int[n];
        for(int i =0;i<n;i++)
            re[i] = 1;
        for(int i=1;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(j==0) re[j] = 1;
                else re[j]+=re[j-1];
            }
        }
        return re[n-1];

    }
}
```

==
/Users/linxie/Documents/github/leetcodeHub/code/UniquePathsII
==

UniquePathsII

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example,
There is one obstacle in the middle of a 3x3 grid as illustrated below.

```
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
```
The total number of unique paths is 2.

```
public class Solution {
    public int uniquePathsWithObstacles(int[][] obstacleGrid) {
        int m = obstacleGrid.length;
        int n = obstacleGrid[0].length;
        if(m*n==0) return 0;
        int[] re = new int[n];
        boolean row = false, col = false;
        for(int i =0;i<n;i++)
        {
            if(obstacleGrid[0][i]==1) row = true;
            if(row)  re[i] = 0;
            else re[i] = 1;
        }
        for(int i=1;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(obstacleGrid[i][j]==1)
                {
                    re[j] = 0;
                    continue;
                }
                if(j==0) continue;
                else re[j]+=re[j-1];
            }
        }
        return re[n-1];


    }
}
```

==
/Users/linxie/Documents/github/leetcodeHub/code/ValidateBinarySearchTree
==

ValidateBinarySearchTree

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the nodes key.
The right subtree of a node contains only nodes with keys greater than the nodes key.
Both the left and right subtrees must also be binary search trees.

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
```

```
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public static boolean isValidBST(TreeNode root, int lower, int upper)
    {
        if(root == null) return true;
        if(root.val >= upper || root.val <= lower) return false;
        return isValidBST(root.left, lower, root.val) && isValidBST(root.right, root.val, upper);
    }

    public static boolean isValidBST(TreeNode root)
    {
        if(root == null) return true;
        return isValidBST(root.left, Integer.MIN_VALUE, root.val) && isValidBST(root.right, root.val, Integer.MAX_VALUE);
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/ValidNumber
=======================================


ValidNumber

Validate if a given string is numeric.

Some examples:
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true

---
public class Solution {
    public boolean isNumber(String s) {

        boolean number = false, e = false, dot = false, sign = false;

        int i=0;
        while(i<s.length() && s.charAt(i) ==  ) i++;
        if(i==s.length()) return false;

        int j=s.length()-1;
        while(j>=0 && s.charAt(j) ==  ) j--;
        if(j<0) return false;

        for(; i<=j; i++)
        {
            char c = s.charAt(i);
            if(c == + || c==-)
            {
                if(sign || number) return false;
                sign = true;
            }
            else if(c == e)
            {
                if(e || !number) return false;
                e = true;
                number = false;
                sign = false;
            }else if(c == .)
            {
                if(dot || e) return false;
                dot = true;
                if(!sign) sign = true;

            }else if(c>=0 && c<=9)
            {
                if(!number) number = true;
                if(!sign) sign = true;
            }
            else{
                return false;
            }
        }
        return number;


    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/ValidPalindrome
=======================================


ValidPalindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,
"A man, a plan, a canal: Panama" is a palindrome.
"race a car" is not a palindrome.

Note:
Have you consider that the string might be empty? This is a good question to ask during an interview.

```

For the purpose of this problem, we define empty string as valid palindrome.

```
public class Solution {
    public int valid(char c){
        if(c ==  ) return -1;

        int key = c-a;
        if(key>=0 && key<=25) return key;
        key = c-A;
        if(key>=0 && key<=25) return key;
        key = c-0;
        if(key>=0 && key<=9)
        {
            key+=30;
            return key;
        }
        return -1;
    }

    public boolean isPalindrome(String s) {
        int n = s.length();
        if(n<=1) return true;

        char[] c = s.toCharArray();
        int l = 0, r = n-1;
        while(l<=r)
        {
            int first = -1;
            while(l<=r)
            {
                first = valid(c[l]);
                if( first == -1) l++;
                else break;
            }
            if(l>r) break;

            int second = -1;
            while(l<=r)
            {
                second = valid(c[r]);
                if(second == -1) r--;
                else break;
            }
            if(l>r) break;

            if(first!=second) return false;
            l++;
            r--;
        }
        return true;
    }
}
```

```
========================================
/Users/linxie/Documents/github/leetcodeHub/code/ValidParentheses
========================================
```

ValidParentheses

Given a string containing just the characters (,), {, }, [and], determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

```
public class Solution {
    public boolean isValid(String s) {

        int n = s.length();
        if(n==0) return true;
        if(n%2 == 1) return false;

        char[] c = s.toCharArray();
        Stack<Character> st = new Stack<Character>();
        for(char cc :c)
        {
            if(cc==( || cc=={ || cc==[) st.push(cc);
            else if(cc == ) && (st.isEmpty() || st.pop()!=()) return false;
            else if(cc == } && (st.isEmpty() || st.pop()!={)) return false;
            else if(cc == ] && (st.isEmpty() ||st.pop()!=[)) return false;
        }
        return st.isEmpty();

    }
}
```

```
========================================
/Users/linxie/Documents/github/leetcodeHub/code/ValidSudoku
========================================
```

ValidSudoku

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

The Sudoku board could be partially filled, where empty cells are filled with the character ..

```
public class Solution {
```

```
    public boolean isValidSudoku(char[][] board) {
        int n = board.length;
        boolean[] rec = new boolean[9];


        for(int i=0;i<n;i++)
        {
            for(int j=0;j<9;j++)
                rec[j] = false;

            for(int j =0; j< n; j++)
            {
                if(board[i][j] == .) continue;
                int cur = (int)(board[i][j]-0);
                if( cur<1 || cur >9 || rec[cur-1] ) return false;
                else rec[cur-1] = true;
            }
        }

        for(int i=0;i<n;i++)
        {
            for(int j=0;j<9;j++)
                rec[j] = false;

            for(int j =0; j< n; j++)
            {
                if(board[j][i] == .) continue;
                int cur = (int)(board[j][i]-0);
                if( cur<1 || cur >9 || rec[cur-1] ) return false;
                else rec[cur-1] = true;
            }
        }

        for(int i=0; i<3;i++)
        {
            for(int j=0; j<3; j++)
            {
                for(int k=0;k<9;k++)
                    rec[k] = false;
                for(int k=0;k<3;k++)
                {
                    for(int v =0; v<3; v++)
                    {
                        int ii = i*3+k;
                        int jj = j*3+v;
                        if(board[ii][jj] == .) continue;
                        int cur = (int)(board[ii][jj]-0);
                        if( cur<1 || cur >9 || rec[cur-1] ) return false;
                        else rec[cur-1] = true;
                    }
                }
            }
        }
        return true;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/WildcardMatching
=======================================


WildcardMatching

Implement wildcard pattern matching with support for ? and *.

? Matches any single character.
* Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "*") → true
isMatch("aa", "a*") → true
isMatch("ab", "?*") → true
isMatch("aab", "c*a*b") → false

---

public class Solution {
    public boolean isMatch(String s, String p) {
        char[] a = s.toCharArray(), b = p.toCharArray();
        int m = a.length, n = b.length;
        if(n==0) return m==0;

        int i=0, j=0;
        int star = -1, ii = -1;
        while(i<m)
        {
            if(j<n && (a[i]==b[j] || b[j]==?))
            {
                i++; j++;
            }
            else if(j<n && b[j]==*)
            {
                star = j;
```

```
                j++;
                ii = i+1;
            }
            else if(star!=-1)
            {
                i = ii++;
                j=star+1;
            }
            else
            {
                return false;
            }
        }
        while(j<n && b[j]==*) j++;
        return j==n;

    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/WordBreak
=======================================


WordBreak

Given a string s and a dictionary of words dict, determine if s can be segmented into a
space-separated sequence of one or more dictionary words.

For example, given
s = "leetcode",
dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".
--------
public class Solution {
    public boolean wordBreak(String s, Set<String> dict, int[] map)
    {
        int n = s.length();
        if(n==0) return true;
        if(map[n-1]!=0) return map[n-1]==1;

        for(int i=n-1; i>0;i--)
        {
            if(dict.contains(s.substring(i)) && wordBreak(s.substring(0,i), dict, map)) return true;
        }
        map[n-1] = -1;
        return false;
    }

    public boolean wordBreak(String s, Set<String> dict) {
        int n = s.length();
        if(n==0) return false;

        int[] map = new int[n];
        for(int i=0;i<n;i++)
        {
            if(dict.contains(s.substring(0,i+1)))
                map[i] = 1;
        }
        return wordBreak(s,dict,map);
    }

}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/WordBreakII
=======================================


WordBreakII

Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where
each word is a valid dictionary word.

Return all such possible sentences.

For example, given
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].

---
public class Solution {

    public List<String> wordBreak(String s, HashMap<Integer, List<String>> map, Set<String> dict) {
        List<String> re = new ArrayList<String>();
        int n = s.length();
        if(n==0) return re;
        if(map.containsKey(n)) return map.get(n);

        for(int i=n-1; i>=0; i--)
        {
            if(dict.contains(s.substring(i)))
            {
                if(i==0)
                {
                    re.add(s);
                }
                List<String> r = wordBreak(s.substring(0,i), map, dict);
                for(String list : r)
```

```
                {
                    re.add(list+" "+s.substring(i));
                }
            }
        }
        if(re.size()>0) map.put(n, re);
        return re;
    }
    public List<String> wordBreak(String s, Set<String> dict) {
        HashMap<Integer, List<String>> map = new HashMap<Integer, List<String>>();
        return wordBreak(s, map, dict);
    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/WordLadder
========================================


WordLadder

Given two words (start and end), and a dictionary, find the length of shortest transformation
sequence from start to end, such that:

Only one letter can be changed at a time
Each intermediate word must exist in the dictionary
For example,

Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.

---
public class Solution {
    public int ladderLength(String start, String end, Set<String> dict) {

        int n = start.length();
        if(end.length()!=n) return 0;
        if(start.equals(end)) return 1;

        HashSet<String> cur = new HashSet<String>();
        cur.add(start);
        dict.remove(start);
        boolean found = false;
        int re = 1;

        while(!found)
        {
            re++;
            if(cur.size()==0) return 0;
            HashSet<String> next = new HashSet<String>();
            for(String s:cur)
            {
                char[] c = s.toCharArray();
                for(int i=0;i<n;i++)
                {
                    char val = c[i];
                    for(char cc = a;cc<=z;cc++)
                    {
                        c[i] = cc;
                        String str = new String(c);
                        if(str.equals(end)) return re;
                        if(dict.contains(str) && !next.contains(str))
                        {
                            next.add(str);
                        }
                    }
                    c[i] = val;
                }
            }
            for(String ss:next)
            {
                dict.remove(ss);
            }
            cur = next;
        }
        return re;

    }
}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/WordLadderII
========================================


WordLadderII

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s)
from start to end, such that:

Only one letter can be changed at a time
Each intermediate word must exist in the dictionary
For example,

Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

```
Return
  [
    ["hit","hot","dot","dog","cog"],
    ["hit","hot","lot","log","cog"]
  ]
Note:
All words have the same length.
All words contain only lowercase alphabetic characters.
---

class Word
{
    Set<String> dict;
    ArrayList<HashSet<Character>> letters;
    HashMap<String, HashSet<String>> map = new HashMap<String, HashSet<String>>();

    Word(Set<String> _dict, ArrayList<HashSet<Character>> _letters, String name)
    {
        dict = new HashSet<String>(_dict);
        dict.remove(name);
        letters = _letters;
    }

    public HashSet<String> getList(HashSet<String> list)
    {
        HashSet<String> re = new HashSet<String>();

        for(String w : list)
        {
            char[] ss = w.toCharArray();

            for(int i=0;i<ss.length; i++){
              char tmp = ss[i];
              for(char c : letters.get(i))
              {
                if(c == tmp) continue;
                ss[i] = c;

                String str = new String(ss);

                if(dict.contains(str))
                {
                  re.add(str);
                  if(map.containsKey(str))
                  {
                    HashSet<String> prev = map.get(str);
                    if(!prev.contains(w))
                    {
                      prev.add(w);
                      map.put(str, prev);
                    }
                  }else
                  {
                    HashSet<String> prev = new HashSet<String>();
                    prev.add(w);
                    map.put(str, prev);
                  }
                }
              }
              ss[i] = tmp;
            }
        }
        for(String ss : re)
            dict.remove(ss);
        return re;
    }

    List<List<String>> getResult(String end, boolean flg)
    {
        List<List<String>> re = new ArrayList<List<String>>();
        if(!map.containsKey(end))
        {
            List<String> result = new ArrayList<String>();
            result.add(end);
            re.add(result);
            return re;
        }


        for(String s:map.get(end))
        {
            List<List<String>> tmp = getResult(s, flg);
            Iterator<List<String>> it = tmp.iterator();
            while(it.hasNext())
            {
              re.add(it.next());
            }
        }
        Iterator<List<String>> it = re.iterator();
        while(it.hasNext())
        {
          if(flg) it.next().add(end);
          else it.next().add(0, end);
        }
        return re;
    }
}


public class Solution {

    public List<List<String>> findLadders(String start, String end, Set<String> dict) {
        List<List<String>> re = new LinkedList<List<String>>();
        int n = start.length();
```

```
        if(n!=end.length()) return re;

        dict.add(end);

        boolean found = false;

        ArrayList<HashSet<Character>> letters = new ArrayList<HashSet<Character>>();
        for(int i = 0; i<n;i++)
        {
          letters.add(new HashSet<Character>());
        }

        for(String s:dict)
        {
          if(s.length()!=n) continue;

          for(int i=0;i<n;i++)
          {
            if(!letters.get(i).contains(s.charAt(i)))
              letters.get(i).add(s.charAt(i));
          }
        }

        Word forward = new Word(dict, letters, start);
        Word backward = new Word(dict, letters, end);

        HashSet<String> forward_list = new HashSet<String>();
        HashSet<String> backward_list = new HashSet<String>();

        HashSet<String> sub_result = new HashSet<String>();

        forward_list.add(start);
        backward_list.add(end);

        while(!found)
        {
            forward_list = forward.getList(forward_list);
            if(forward_list.isEmpty())
                break;
            for(String ss : forward_list)
            {
                if(backward_list.contains(ss))
                {
                    sub_result.add(ss);
                    found = true;
                }
            }
            if(found) break;
            backward_list = backward.getList(backward_list);
            if(backward_list.isEmpty())
                break;

            for(String ss : forward_list)
            {
                if(backward_list.contains(ss))
                {
                    sub_result.add(ss);
                    found = true;
                }
            }
        }
        if(sub_result.isEmpty()) return re;

        for(String s : sub_result)
        {
            List<List<String>> first_half = forward.getResult(s,true);
            List<List<String>> back_half = backward.getResult(s,false);

            for(List<String> l : first_half){
                for(List<String> ll : back_half)
                {
                    ll.remove(s);
                    List<String> l_tmp = new ArrayList<String>(l);
                    l_tmp.addAll(ll);
                    re.add(l_tmp);
                }
            }
        }
        return re;

    }

}


========================================
/Users/linxie/Documents/github/leetcodeHub/code/WordSearch
========================================


WordSearch

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are
those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,
Given board =

[
  ["ABCE"],
  ["SFCS"],
  ["ADEE"]
```

```
]
word = "ABCCED", -> returns true,
word = "SEE", -> returns true,
word = "ABCB", -> returns false.

--
public class Solution {

    public boolean exist(char[][] board, String word, int i, int j, boolean[][] map)
    {
        if(word.length() == 0) return true;
        if(i-1>=0 && !map[i-1][j])
        {
            if(word.charAt(0) == board[i-1][j])
            {
                map[i-1][j] = true;
                if(exist(board, word.substring(1), i-1, j, map))
                    return true;
                map[i-1][j] = false;
            }
        }

        if(j-1>=0 && !map[i][j-1])
        {
            if(word.charAt(0) == board[i][j-1])
            {
                map[i][j-1] = true;
                if(exist(board, word.substring(1), i, j-1, map))
                    return true;
                map[i][j-1] = false;
            }
        }

        if(i+1<board.length && !map[i+1][j])
        {
            if(word.charAt(0) == board[i+1][j])
            {
                map[i+1][j] = true;
                if(exist(board, word.substring(1), i+1, j, map))
                    return true;
                map[i+1][j] = false;
            }
        }

        if(j+1<board[0].length && !map[i][j+1])
        {
            if(word.charAt(0) == board[i][j+1])
            {
                map[i][j+1] = true;
                if(exist(board, word.substring(1), i, j+1, map))
                    return true;
                map[i][j+1] = false;
            }
        }
        return false;
    }

    public boolean exist(char[][] board, String word) {
        int m = board.length;
        if(m==0) return false;
        int n = board[0].length;
        if(n==0) return false;
        if(word.length() == 0 ) return true;

        boolean[][] map = new boolean[m][n];

        for(int i=0;i<m;i++)
            for(int j =0; j<n; j++)
                if( board[i][j] == word.charAt(0))
                {
                    map[i][j] = true;
                    if(exist(board, word.substring(1), i, j, map))
                        return true;
                    map[i][j] = false;
                }
        return false;
    }
}


=======================================
/Users/linxie/Documents/github/leetcodeHub/code/youngTablueKMAX
=======================================


给A, B 2个array, 里面都是integer, 已经排好序了, 由大到小, 他们的长度都是N

现在从A和B里各选出一个数, 总成一个sum, 请返回前N个最大的sum




class Pair implements Comparable
{
    int val, i, j;
    Pair(int _i, int _j, int ai, int bj)
    {
        val = ai+bj;
        i = _i; j = _j;
    }

    public int compareTo(Object o)
    {
        Pair p = (Pair)o;
```

```java
        return val-p.val;
    }
}

class Solution
{
    public List<Integer> youngTablueKMAX(int[] a, int[] b, int k)
    {
        List<Integer> re = new ArrayList<Integer>();

        int m = a.length, n = b.length;
        if(k<=0 || k>=m*n) return re;

        if(k==1)
        {
            re.add(a[0]+b[0]);
            return re;
        }

        PriorityQueue<Pair> q = new PriorityQueue<Pair>();
        for(int i=0;i<n;i++)
        {
            q.add(new Pair(0,i,a[0], b[i]));
        }
        int count = 0;
        while(!q.isEmpty() && count<=k)
        {
            Pair cur = q.poll();
            if(cur == null) break;
            re.add(cur.val);

            cur.i++;
            if(cur.i == m) continue;

            q.add(new Pair(cur.i, cur.j, a[cur.i], b[cur.j]));
        }
        return re;
    }
}
```