

```

1 =====
2 /Users/linxie/Documents/github/leetcodeHub/code/3Sum
3 =====
4
5
6 3Sum
7
8 Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0?
9 Find all unique triplets in the array which gives the sum of zero.
10
11 Note:
12 Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c)
13 The solution set must not contain duplicate triplets.
14     For example, given array S = {-1 0 1 2 -1 -4},
15
16     A solution set is:
17     (-1, 0, 1)
18     (-1, -1, 2)
19 ---
20
21 public class Solution {
22     public List<List<Integer>> threeSum(int[] num) {
23         List<List<Integer>> re = new ArrayList<List<Integer>>();
24         int n = num.length;
25         if(n<3) return re;
26
27         Arrays.sort(num);
28         for(int i=0;i<n-2;i++)
29         {
30             while(i!=0 && i<n-2 && num[i] == num[i-1]) i++;
31             if(i==n-2) break;
32
33             int j = i+1, k = n-1;
34             while(j<k)
35             {
36                 while(j!=i+1 && j<k && num[j]==num[j-1]) j++;
37                 if(j==k) break;
38                 while(k!=n-1 && j<k && num[k]==num[k+1]) k--;
39                 if(j==k) break;
40
41                 int sum = num[i]+num[j]+num[k];
42                 if(sum == 0)
43                 {
44                     List<Integer> r = new ArrayList<Integer>();
45                     r.add(num[i]); r.add(num[j]); r.add(num[k]);
46                     re.add(r);
47                     j++; k--;
48                 }
49                 else if(sum<0) j++;
50                 else k--;
51             }
52         }
53         return re;
54     }
55 }
56
57 ---
58 Cleaner solution:
59
60 class Solution
61 {
62     public List<List<Integer>> threeSum(int[] num)
63     {
64         List<List<Integer>> re = new ArrayList<List<Integer>>();
65         int n = num.length;
66         if(n<3) return re;
67
68         Arrays.sort(num);
69
70         for(int i=0;i<n;i++)
71         {
72             if(i!=0 && num[i-1]==num[i]) continue;
73
74             int j = i+1, k = n-1;
75             while(j<k)
76             {
77                 if(j!=i+1 && num[j] == num[j-1]) j++;
78                 else if(k!=n-1 && num[k] == num[k+1]) k--;
79                 else
80                 {
81                     int sum = num[i] + num[j] +num[k];
82                     if(sum == 0)
83                     {
84                         List<Integer> rr = new ArrayList<Integer>();
85                         rr.add(num[i]);
86                         rr.add(num[j]);
87                         rr.add(num[k]);
88                         re.add(rr);
89                         j++; k--;
90                     }
91                     else if(sum<0) j++;
92                     else k--;
93                 }
94             }
95         }
96         return re;
97     }
98 }
99
100
101
102
103
```

```

104 =====
105 /Users/linxie/Documents/github/leetcodeHub/code/3SumClosest
106 =====
107
108
109 3SumClosest
110
111 Given an array S of n integers, find three integers in S such that the sum is closest to a
112 given number, target. Return the sum of the three integers. You may assume that each input
113 would have exactly one solution.
114
115     For example, given array S = {-1 2 1 -4}, and target = 1.
116
117     The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
118 ===
119
120 public class Solution {
121     public int threeSumClosest(int[] num, int target) {
122
123         int n = num.length;
124         if(n==0) return 0;
125         if(n==1) return num[0];
126         if(n==2)
127         {
128             int re = num[0];
129             re = Math.abs(re-target)<Math.abs(num[1]-target)?re:num[1];
130             re = Math.abs(re-target)<Math.abs(num[1]+num[0]-target)?re:(num[0]+num[1]);
131             return re;
132         }
133     }
134
135     int diff = Integer.MAX_VALUE;
136     int re = 0;
137     Arrays.sort(num);
138     for(int i=0;i<n;i++)
139     {
140         if(i!=0 && num[i-1] == num[i]) continue;
141         else
142         {
143             int j = i+1, k = n-1;
144             while(j<k)
145             {
146                 while(j<k && j!=i+1 && num[j] == num[j-1]) j++;
147                 if(j==k) break;
148
149                 while(j<k && k!=n-1 && num[k] == num[k+1]) k--;
150                 if(j==k) break;
151                 int sum = num[i]+num[j]+num[k];
152                 if( Math.abs(sum-target) < diff)
153                 {
154                     diff = Math.abs(num[i]+num[j]+num[k]-target);
155                     re = num[i]+num[j]+num[k];
156                 }
157                 if(sum == target) return target;
158                 if(sum<target) j++;
159                 else k--;
160             }
161         }
162     }
163     return re;
164
165 }
166 }
167 =====
168 /Users/linxie/Documents/github/leetcodeHub/code/4Sum
169 =====
170
171
172
173
174
175 4Sum
176
177 Given an array S of n integers, are there elements a, b, c, and d in S such that
178 a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.
179
180 Note:
181 Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d)
182 The solution set must not contain duplicate quadruplets.
183     For example, given array S = {1 0 -1 0 -2 2}, and target = 0.
184
185     A solution set is:
186     (-1, 0, 0, 1)
187     (-2, -1, 1, 2)
188     (-2, 0, 0, 2)
189
190 ===
191
192
193 public class Solution {
194     public List<List<Integer>> fourSum(int[] num, int target) {
195
196         List<List<Integer>> re = new ArrayList<List<Integer>>();
197         int n = num.length;
198         if(n<4) return re;
199
200         Arrays.sort(num);
201         for(int i=0;i<n;i++)
202         {
203             if(i!=0 && num[i-1]==num[i]) continue;
204
205             for(int j=i+1; j<n;j++)
206             {
207                 if(j!=i+1 && num[j-1]==num[j]) continue;

```

```

208
209     int k = j+1, v =n-1;
210     while(k<v)
211     {
212         int res = num[i] + num[j] + num[k]+num[v];
213         if(res==target)
214         {
215             List<Integer> list = new ArrayList<Integer>();
216             list.add(num[i]); list.add(num[j]); list.add(num[k]); list.add(num[v]);
217             re.add(list);
218             k++; v--;
219             while(k<v && num[k]==num[k-1]) k++;
220             if(k==v) break;
221             while(k<v && num[v]==num[v+1]) v--;
222             if(k==v) break;
223         }
224         else if(res>target)
225         {
226             v--;
227             while(k<v && num[v]==num[v+1]) v--;
228             if(k==v) break;
229         }else
230         {
231             k++;
232             while(k<v && num[k]==num[k-1]) k++;
233             if(k==v) break;
234         }
235     }
236 }
237 }
238 return re;
239 }
240 }
241 ---
242 CLeaner solution
243
244
245 public class Solution {
246     public List<List<Integer>> fourSum(int[] num, int target) {
247
248         int n = num.length;
249
250         List<List<Integer>> re = new ArrayList<List<Integer>>();
251         if(n<4) return re;
252
253         Arrays.sort(num);
254
255         for(int i=0;i<n;i++)
256         {
257             if(i!=0 && num[i] == num[i-1]) continue;
258
259             for(int j = i+1; j<n; j++)
260             {
261                 if(j!=i+1 && num[j] == num[j-1]) continue;
262
263                 int k = j+1, v = n-1;
264                 while(k<v)
265                 {
266                     if(k!=j+1 && num[k] == num[k-1]) k++;
267                     else if(v!=n-1 && num[v] == num[v+1]) v--;
268                     else
269                     {
270                         int sum = num[i]+num[j]+num[k]+num[v];
271                         if(sum==target)
272                         {
273                             List<Integer> rr = new ArrayList<Integer>();
274                             rr.add(num[i]); rr.add(num[j]); rr.add(num[k]); rr.add(num[v]);
275                             re.add(rr);
276                             k++; v--;
277                         }
278                         else if(sum<target) k++;
279                         else v--;
280                     }
281                 }
282             }
283         }
284         return re;
285     }
286 }
287
288
289
290
291 =====
292 /Users/linxie/Documents/github/leetcodeHub/code/AddBinary
293 =====
294
295
296
297 AddBinary
298
299 Given two binary strings, return their sum (also a binary string).
300
301 For example,
302 a = "11"
303 b = "1"
304 Return "100".
305 --
306
307 public class Solution {
308     public String addBinary(String a, String b) {
309         int m = a.length();
310         int n = b.length();
311         if(m==0) return b;

```

```

312     if(n==0) return a;
313
314     int carry = 0;
315     int i = m-1, j = n-1;
316     String re = "";
317
318     while(i>=0 && j>=0)
319     {
320         int r = carry + (int)(a.charAt(i)-0)+(int)(b.charAt(j)-0);
321         carry = r/2;
322         re = Integer.toString(r%2) + re;
323         i--; j--;
324     }
325     while(i>=0)
326     {
327         int r = carry + (int)(a.charAt(i)-0);
328         carry = r/2;
329         re = Integer.toString(r%2) + re;
330         i--;
331     }
332     while(j>=0)
333     {
334         int r = carry + (int)(b.charAt(j)-0);
335         carry = r/2;
336         re = Integer.toString(r%2) + re;
337         j--;
338     }
339     if(carry==1) re = "1"+re;
340     return re;
341 }
342 }
343
344
345 =====
346 /Users/linxie/Documents/github/leetcodeHub/code/AddTwoNumbers
347 =====
348
349
350 AddTwoNumbers
351
352 You are given two linked lists representing two non-negative numbers. The digits are stored
353 in reverse order and each of their nodes contain a single digit. Add the two numbers and
354 return it as a linked list.
355
356 Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
357 Output: 7 -> 0 -> 8
358
359 ---
360
361 /**
362 * Definition for singly-linked list.
363 * public class ListNode {
364 *     int val;
365 *     ListNode next;
366 *     ListNode(int x) {
367 *         val = x;
368 *         next = null;
369 *     }
370 * }
371 */
372 public class Solution {
373     public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
374
375         if(l1==null) return l2;
376         if(l2==null) return l1;
377         ListNode newHead = null, tail =null;
378         int carry = 0;
379         while(l1!=null && l2!=null)
380         {
381             int re = l1.val+l2.val+carry;
382             carry = re/10;
383             if(carry!=0)
384                 re %= 10;
385             ListNode cur = new ListNode(re);
386             if(newHead == null)
387             {
388                 newHead = cur; tail = cur;
389             }else
390             {
391                 tail.next = cur;
392                 tail = cur;
393             }
394             l1=l1.next; l2=l2.next;
395         }
396
397         while(l1!=null)
398         {
399             int re = l1.val+carry;
400             carry = re/10;
401             if(carry!=0) re%=10;
402             ListNode cur = new ListNode(re);
403             tail.next = cur;
404             tail = cur;
405             l1=l1.next;
406         }
407         while(l2!=null)
408         {
409             int re = l2.val+carry;
410             carry = re/10;
411             if(carry!=0) re%=10;
412             ListNode cur = new ListNode(re);
413             tail.next = cur;
414             tail = cur;
415             l2=l2.next;
        }
    }
}

```

```

416         }
417     if(carry!=0)
418         tail.next = new ListNode(carry);
419     return newHead;
420 }
421 }
422 }
423 }
424 =====
425 /Users/linxie/Documents/github/leetcodeHub/code/Anagrams
426 =====
427 =====
428
429
430 Anagrams
431
432 Given an array of strings, return all groups of strings that are anagrams.
433
434 ---
435 public class Solution {
436     public List<String> anagrams(String[] strs) {
437
438         List<String> re = new ArrayList<String>();
439         int n = strs.length;
440         if(n<=1) return re;
441
442         HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
443         HashSet<Integer> added = new HashSet<Integer>();
444         for(int i = 0; i<n; i++)
445         {
446             char[] c = strs[i].toCharArray();
447             Arrays.sort(c);
448             int key = (new String(c)).hashCode();
449             if(!map.containsKey(key))
450             {
451                 map.put(key, i);
452             }
453             else
454             {
455                 if(!added.contains(map.get(key)))
456                 {
457                     added.add(map.get(key));
458                     re.add(strs[map.get(key)]);
459                 }
460                 re.add(strs[i]);
461             }
462         }
463         return re;
464     }
465 }
466 }
467
468 =====
469 /Users/linxie/Documents/github/leetcodeHub/code/BalancedBinaryTree
470 =====
471 =====
472
473
474 BalancedBinaryTree
475
476 Given a binary tree, determine if it is height-balanced.
477
478 For this problem, a height-balanced binary tree is defined as a binary tree in which
479 the depth of the two subtrees of every node never differ by more than 1.
480
481 ---
482
483 /**
484 * Definition for binary tree
485 * public class TreeNode {
486 *     int val;
487 *     TreeNode left;
488 *     TreeNode right;
489 *     TreeNode(int x) { val = x; }
490 * }
491 */
492 public class Solution {
493     public int helper(TreeNode root)
494     {
495         if(root == null) return 0;
496         int left = helper(root.left);
497         if(left == -1) return -1;
498         int right = helper(root.right);
499         if(right == -1) return -1;
500
501         if(Math.abs(left-right)<2) return Math.max(left, right)+1;
502         else return -1;
503     }
504
505     public boolean isBalanced(TreeNode root)
506     {
507         return helper(root)!=-1;
508     }
509 }
510
511
512 =====
513 /Users/linxie/Documents/github/leetcodeHub/code/BestTimetoBuyandSellStock
514 =====
515
516
517 BestTimetoBuyandSellStock
518
519 Say you have an array for which the ith element is the price of a given stock on day i.

```

```

520
521 If you were only permitted to complete at most one transaction (ie, buy one and sell one share
522     of the stock), design an algorithm to find the maximum profit.
523
524 ---
525 public class Solution {
526     public int maxProfit(int[] prices) {
527
528         int re = 0;
529         int n = prices.length;
530         if(n<=1) return re;
531
532         int min = Integer.MAX_VALUE;
533         for(int i:prices)
534         {
535             if(i<min) min = i;
536             else
537             {
538                 if(i-min > re) re = i-min;
539             }
540         }
541         return re;
542
543     }
544 }
545
546
547 =====
548 /Users/linxie/Documents/github/leetcodeHub/code/BestTimetoBuyandSellStockII
549 =====
550
551
552 BestTimetoBuyandSellStockII
553
554 Say you have an array for which the ith element is the price of a given stock on day i.
555
556 Design an algorithm to find the maximum profit. You may complete as many transactions as you like
557 (ie, buy one and sell one share of the stock multiple times). However, you may not engage in
558 multiple transactions at the same time (ie, you must sell the stock before you buy again).
559 --
560 public class Solution {
561     public int maxProfit(int[] prices) {
562
563         int n = prices.length;
564         if(n<=1) return 0;
565
566         int max = 0;
567         for(int i=1;i<n;i++)
568         {
569             if(prices[i]>prices[i-1])
570                 max+=prices[i] - prices[i-1];
571         }
572         return max;
573     }
574 }
575 }
576
577
578 =====
579 /Users/linxie/Documents/github/leetcodeHub/code/BestTimetoBuyandSellStockIII
580 =====
581
582
583 BestTimetoBuyandSellStockIII
584
585 Say you have an array for which the ith element is the price of a given stock on day i.
586
587 Design an algorithm to find the maximum profit. You may complete at most two transactions.
588
589 Note:
590 You may not engage in multiple transactions at the same time (ie, you must sell the stock before you
591 buy again).
592
593 ---
594
595 public class Solution {
596     public int maxProfit(int[] prices) {
597
598         int n= prices.length;
599         if(n<2) return 0;
600         int[] one = new int[n];
601
602         int min = prices[0];
603         for(int i=1;i<n;i++)
604         {
605             if(prices[i]<min)
606             {
607                 one[i] = one[i-1];
608                 min = prices[i];
609             }
610             else
611             {
612                 if(prices[i] - min > one[i-1]) one[i] = prices[i] - min;
613                 else one[i] = one[i-1];
614             }
615         }
616
617         int max = 0;
618         int re = one[n-1];
619         for(int i=n-1; i>0; i--)
620         {
621             if(prices[i]>max) max = prices[i];
622             else
623             {

```

```

624             if(max-prices[i]+one[i-1]>re) re =max-prices[i]+one[i-1];
625         }
626     }
627     return re;
628 }
629 }
630 }
631 }
632 }
633 }
634 =====
635 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeInorderTraversal
636 =====
637
638
639 BinaryTreeInorderTraversal
640
641 Given a binary tree, return the inorder traversal of its nodes values.
642
643 For example:
644 Given binary tree {1,#,2,3},
645   1
646   \
647     2
648     /
649     3
650 return [1,3,2].
651
652 ===
653 /**
654 * Definition for binary tree
655 * public class TreeNode {
656 *     int val;
657 *     TreeNode left;
658 *     TreeNode right;
659 *     TreeNode(int x) { val = x; }
660 * }
661 */
662 public class Solution {
663     public List<Integer> inorderTraversal(TreeNode root) {
664
665
666         List<Integer> re = new ArrayList<Integer>();
667         if(root == null) return re;
668
669         TreeNode cur = root, tail = null, pre = null;
670         root = null;
671         while(cur!=null)
672         {
673             if(cur.left==null)
674             {
675                 if(root == null) { root = cur; tail = cur; cur = cur.right; tail.right = null; }
676                 else
677                 {
678                     tail.right = cur;
679                     cur = cur.right;
680                     tail = tail.right;
681                     tail.right = null;
682                     tail.left = null;
683                 }
684             }
685             else
686             {
687                 pre = cur.left;
688                 while(pre.right!=null && pre.right!=cur) pre = pre.right;
689                 if(pre.right == null){ pre.right = cur; cur = cur.left; pre.right.left = null; }
690                 else cur.left = null;
691             }
692         }
693         cur = root;
694         while(cur!=null) { re.add(cur.val); cur = cur.right; }
695         return re;
696
697     }
698 }
699 }
700
701
702 ===
703
704 class Solution
705 {
706     public List<Integer> inorderTraversal(TreeNode root)
707     {
708         List<Integer> re = new ArrayList<Integer>();
709
710         if(root == null) return re;
711
712         TreeNode cur = root, prev = null;
713
714         while(cur!=null)
715         {
716             if(cur.left==null)
717             {
718                 re.add(cur.val);
719                 cur = cur.right;
720             }
721             else
722             {
723                 prev = cur.left;
724                 while(prev.right != cur && prev.right!=null) prev = prev.right;
725                 if(prev.right == null)
726                 {
727                     prev.right = cur;
728                 }
729             }
730         }
731     }
732 }
```

```

728         cur = cur.left;
729     }
730     else
731     {
732         re.add(cur.val);
733         prev.right = null;
734         cur = cur.right;
735     }
736 }
737 }
738 return re;
739 }
740 }
741
742
743
744
745
746 =====
747 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeLevelOrderTraversal
748 =====
749
750
751 BinaryTreeLevelOrderTraversal
752
753 Given a binary tree, return the level order traversal of its nodes values. (ie, from left to right,
754 level by level).
755
756 For example:
757 Given binary tree {3,9,20,#,#,15,7},
758     3
759     / \
760    9   20
761    /   \
762   15   7
763 return its level order traversal as:
764 [
765     [3],
766     [9,20],
767     [15,7]
768 ]---
769 /**
770 * Definition for binary tree
771 * public class TreeNode {
772 *     int val;
773 *     TreeNode left;
774 *     TreeNode right;
775 *     TreeNode(int x) { val = x; }
776 * }
777 */
778 public class Solution {
779     public List<List<Integer>> levelOrder(TreeNode root) {
780
781         List<List<Integer>> re = new ArrayList<List<Integer>>();
782         if(root == null) return re;
783
784         ArrayList<TreeNode> q = new ArrayList<TreeNode>();
785         q.add(root);
786
787         int start = 0;
788         while(start!=q.size())
789         {
790             int siz = q.size();
791             List<Integer> r = new ArrayList<Integer>();
792             for(int i=start; i< siz; i++)
793             {
794                 TreeNode cur = q.get(i);
795                 r.add(cur.val);
796                 if(cur.left!=null) q.add(cur.left);
797                 if(cur.right!=null) q.add(cur.right);
798             }
799             start = siz;
800             re.add(r);
801         }
802         return re;
803     }
804 }
805 }
806
807
808 =====
809 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeLevelOrderTraversalII
810 =====
811
812
813 BinaryTreeLevelOrderTraversalII
814
815 Given a binary tree, return the bottom-up level order traversal of its nodes values.
816 (ie, from left to right, level by level from leaf to root).
817
818 For example:
819 Given binary tree {3,9,20,#,#,15,7},
820     3
821     / \
822    9   20
823    /   \
824   15   7
825 return its bottom-up level order traversal as:
826 [
827     [15,7],
828     [9,20],
829     [3]
830 ]
831 confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ.

```

```

832
833 /**
834  * Definition for binary tree
835  * public class TreeNode {
836  *     int val;
837  *     TreeNode left;
838  *     TreeNode right;
839  *     TreeNode(int x) { val = x; }
840  * }
841 */
842 */
843 public class Solution {
844     public List<List<Integer>> levelOrderBottom(TreeNode root) {
845
846         List<List<Integer>> re = new ArrayList<List<Integer>>();
847         if(root == null) return re;
848
849         List<TreeNode> prev = new ArrayList<TreeNode>();
850         prev.add(root);
851         List<Integer> r = new ArrayList<Integer>();
852         r.add(root.val);
853         re.add(0, r);
854         while(prev.size()!=0)
855         {
856             List<TreeNode> cur = new ArrayList<TreeNode>();
857             r = new ArrayList<Integer>();
858             for(TreeNode nd : prev)
859             {
860                 if(nd.left!=null)
861                 {
862                     cur.add(nd.left);
863                     r.add(nd.left.val);
864                 }
865                 if(nd.right!=null){
866                     cur.add(nd.right);
867                     r.add(nd.right.val);
868                 }
869             }
870             prev = cur;
871             if(r.size()!=0) re.add(0,r);
872         }
873         return re;
874
875     }
876 }
877 }
878
879
880 =====
881 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeMaximumPathSum
882 =====
883
884
885 BinaryTreeMaximumPathSum
886
887 Given a binary tree, find the maximum path sum.
888
889 The path may start and end at any node in the tree.
890
891 For example:
892 Given the below binary tree,
893
894      1
895      / \
896      2   3
897 Return 6.
898
899 /**
900  * Definition for binary tree
901  * public class TreeNode {
902  *     int val;
903  *     TreeNode left;
904  *     TreeNode right;
905  *     TreeNode(int x) { val = x; }
906  * }
907 */
908 */
909 public class Solution {
910     public int maxDeep(TreeNode root, HashMap<TreeNode, Integer> deep)
911     {
912         if(root == null) return 0;
913         if(deep.containsKey(root)) return deep.get(root);
914
915         int re = root.val;
916
917         if(root.left != null) re = Math.max(re, root.val+maxDeep(root.left,deep));
918         if(root.right != null) re = Math.max(re, root.val+maxDeep(root.right,deep));
919         deep.put(root, re);
920         return re;
921     }
922
923     public int maxPathSum(TreeNode root, HashMap<TreeNode, Integer> deep)
924     {
925         if(root==null) return Integer.MIN_VALUE;
926         int leftPath = maxPathSum(root.left, deep);
927         int rightPath = maxPathSum(root.right, deep);
928         int path = root.val;
929         if(root.left!=null) path = Math.max(path, path+deep.get(root.left));
930         if(root.right!=null) path = Math.max(path, path+deep.get(root.right));
931
932         return Math.max(path, Math.max(leftPath, rightPath));
933     }
934 }
935

```

```

936     public int maxPathSum(TreeNode root)
937     {
938         if(root == null) return 0;
939         HashMap<TreeNode, Integer> deep = new HashMap<TreeNode, Integer>();
940         maxDeep(root, deep);
941         return maxPathSum(root, deep);
942     }
943 }
944
945
946 =====
947 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreePostorderTraversal
948 =====
949
950
951 BinaryTreePostorderTraversal
952
953 Given a binary tree, return the postorder traversal of its nodes values.
954
955 For example:
956 Given binary tree {1,#,2,3},
957     1
958     \
959     2
960     /
961     3
962 return [3,2,1]
963 ---
964 /**
965 * Definition for binary tree
966 * public class TreeNode {
967 *     int val;
968 *     TreeNode left;
969 *     TreeNode right;
970 *     TreeNode(int x) { val = x; }
971 * }
972 */
973 public class Solution {
974
975     static TreeNode add(TreeNode from, TreeNode to)
976     {
977         TreeNode cur = from;
978         from = null;
979
980         while(cur!=null)
981         {
982             TreeNode t = cur.right;
983             cur.right = from;
984             from = cur;
985             cur = t;
986         }
987
988         return from;
989     }
990
991
992     public static List<Integer> postorderTraversal(TreeNode root) {
993         List<Integer> re = new ArrayList<Integer>();
994         if(root == null) return re;
995
996         TreeNode tmp = new TreeNode(0), tail = null;
997         tmp.left = root;
998         root = null;
999         TreeNode cur = tmp;
1000
1001         while(cur!=null)
1002         {
1003
1004             if(cur.left == null){ cur = cur.right; }
1005             else
1006             {
1007                 TreeNode pre = cur.left;
1008
1009                 while(pre.right!=null && pre.right!=cur) pre = pre.right;
1010
1011
1012                 if(pre.right==null)
1013                 {
1014                     pre.right = cur;
1015                     cur = cur.left;
1016                 }
1017                 else
1018                 {
1019                     pre.right = null;
1020                     TreeNode t = cur.left;
1021                     cur.left = null;
1022
1023                     t = add(t, pre);
1024                     if(root == null) root = t;
1025                     else tail.right = t;
1026
1027                     tail = root; while(tail.right !=null) tail = tail.right;
1028
1029                     cur = cur.right;
1030
1031                 }
1032             }
1033             for(TreeNode cc = root; cc!=null; cc = cc.right)
1034             {
1035                 re.add(cc.val);
1036             }
1037             return re;
1038     }
1039 }

```

```

1040
1041 /**
1042 * Definition for binary tree
1043 * public class TreeNode {
1044 *     int val;
1045 *     TreeNode left;
1046 *     TreeNode right;
1047 *     TreeNode(int x) { val = x; }
1048 * }
1049 */
1050 */
1051 public class Solution {
1052     public List<Integer> postorderTraversal(TreeNode root) {
1053         List<Integer> re = new ArrayList<Integer>();
1054         if(root == null) return re;
1055
1056         Stack<TreeNode> s = new Stack<TreeNode>();
1057         s.push(root);
1058         TreeNode prev= null;
1059         while(!s.isEmpty())
1060         {
1061             TreeNode nd = s.peek();
1062             if(prev == null || prev.left == nd || prev.right == nd)
1063             {
1064                 if(nd.left!=null) s.push(nd.left);
1065                 else if(nd.right!=null) s.push(nd.right);
1066             }
1067             else
1068             {
1069                 if(nd.left == prev && nd.right!=null) s.push(nd.right);
1070                 else
1071                 {
1072                     re.add(nd.val);
1073                     s.pop();
1074                 }
1075             }
1076             prev = nd;
1077         }
1078         return re;
1079     }
1080 }
1081
1082
1083 =====
1084 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreePreorderTraversal
1085 =====
1086
1087
1088 BinaryTreePreorderTraversal
1089
1090 Given a binary tree, return the preorder traversal of its nodes values.
1091
1092 For example:
1093 Given binary tree {1,#,2,3},
1094     1
1095       \
1096         2
1097         /
1098           3
1099 return [1,2,3].
1100 ---
1101 /**
1102 * Definition for binary tree
1103 * public class TreeNode {
1104 *     int val;
1105 *     TreeNode left;
1106 *     TreeNode right;
1107 *     TreeNode(int x) { val = x; }
1108 * }
1109 */
1110 public class Solution {
1111     public List<Integer> preorderTraversal(TreeNode root) {
1112
1113         List<Integer> re =new ArrayList<Integer>();
1114
1115         if(root == null) return re;
1116
1117         while(root!=null)
1118         {
1119             if(root.left == null){ re.add(root.val); root = root.right; continue; }
1120
1121             TreeNode pre = null;
1122             for(pre = root.left; pre.right!=null && pre.right!=root.right; pre=pre.right);
1123
1124             if(pre.right == null){ pre.right = root.right; re.add(root.val); root = root.left; }
1125             else
1126             {
1127                 pre.right = null; root = root.right;
1128             }
1129         }
1130         return re;
1131
1132     }
1133 }
1134
1135
1136
1137 =====
1138 /Users/linxie/Documents/github/leetcodeHub/code/BinaryTreeZigzagLevelOrderTraversal
1139 =====
1140
1141
1142 BinaryTreeZigzagLevelOrderTraversal
1143

```

```

1144 Given a binary tree, return the zigzag level order traversal of its nodes values.
1145 (ie, from left to right, then right to left for the next level and alternate between).
1146
1147 For example:
1148 Given binary tree {3,9,20,#,#,15,7},
1149     3
1150     / \
1151    9   20
1152    / \
1153   15   7
1154 return its zigzag level order traversal as:
1155 [
1156   [3],
1157   [20,9],
1158   [15,7]
1159 ]
1160 ===
1161 /**
1162 * Definition for binary tree
1163 * public class TreeNode {
1164 *     int val;
1165 *     TreeNode left;
1166 *     TreeNode right;
1167 *     TreeNode(int x) { val = x; }
1168 * }
1169 */
1170 public class Solution {
1171     public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
1172
1173         List<List<Integer>> re = new ArrayList<List<Integer>>();
1174         if(root==null) return re;
1175
1176         ArrayList<TreeNode> q = new ArrayList<TreeNode>();
1177         q.add(root);
1178         boolean flg = false;
1179
1180         int start = 0;
1181         int end = q.size();
1182         while(start<end)
1183         {
1184             List<Integer> level = new ArrayList<Integer>();
1185             for(int i=start; i<end; i++)
1186             {
1187                 if(!flg) level.add(q.get(i).val);
1188                 else level.add(0,q.get(i).val);
1189                 if(q.get(i).left!=null) q.add(q.get(i).left);
1190                 if(q.get(i).right!=null) q.add(q.get(i).right);
1191             }
1192             start = end;
1193             end = q.size();
1194             flg = !flg;
1195             re.add(level);
1196         }
1197         return re;
1198     }
1199 }
1200 }
1201
1202 =====
1203 /Users/linxie/Documents/github/leetcodeHub/code/Candy
1204 =====
1205
1206
1207
1208 Candy
1209
1210 There are N children standing in a line. Each child is assigned a rating value.
1211
1212 You are giving candies to these children subjected to the following requirements:
1213
1214 Each child must have at least one candy.
1215 Children with a higher rating get more candies than their neighbors.
1216 What is the minimum candies you must give?
1217 ---public class Solution {
1218     public int candy(int[] ratings) {
1219         int n = ratings.length;
1220         if(n==0) return 0;
1221         if(n==1) return 1;
1222
1223         int[] candy = new int[n];
1224         candy[0] = 1;
1225
1226         for(int i=1;i<n;i++)
1227         {
1228             if(ratings[i]>ratings[i-1]) candy[i] = candy[i-1]+1;
1229             else candy[i] = 1;
1230         }
1231         int total = candy[n-1];
1232         for(int i = n-2;i>=0;i--)
1233         {
1234             if(ratings[i]>ratings[i+1] && candy[i]<=candy[i+1])
1235             {
1236                 candy[i] = candy[i+1]+1;
1237             }
1238             total += candy[i];
1239         }
1240         return total;
1241     }
1242 }
1243
1244
1245 =====
1246 /Users/linxie/Documents/github/leetcodeHub/code/ClimbingStairs
1247 =====

```

```

1248
1249
1250 ClimbingStairs
1251
1252 You are climbing a stair case. It takes n steps to reach to the top.
1253
1254 Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?
1255
1256 ---
1257 public class Solution {
1258     public int climbStairs(int n) {
1259         if(n<=1) return 1;
1260
1261         int first = 1, second = 1;
1262         int re = -1;
1263         for(int i=2;i<=n;i++) {
1264             {
1265                 re = first + second;
1266                 first = second;
1267                 second = re;
1268             }
1269             return re;
1270     }
1271 }
1272
1273
1274 =====
1275 /Users/linxie/Documents/github/leetcodeHub/code/CloneGraph
1276 =====
1277
1278
1279 CloneGraph
1280
1281 Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.
1282
1283
1284 OJs undirected graph serialization:
1285 Nodes are labeled uniquely.
1286
1287 We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.
1288 As an example, consider the serialized graph {0,1,2#1,2#2,2}.
1289
1290 The graph has a total of three nodes, and therefore contains three parts as separated by #.
1291
1292 First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
1293 Second node is labeled as 1. Connect node 1 to node 2.
1294 Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.
1295 Visually, the graph looks like the following:
1296
1297
1298      1
1299      / \
1300      /   \
1301      0 --- 2
1302          / \
1303          \_/
1304 ---
1305 /**
1306 * Definition for undirected graph.
1307 * class UndirectedGraphNode {
1308 *     int label;
1309 *     List<UndirectedGraphNode> neighbors;
1310 *     UndirectedGraphNode(int x) { label = x; neighbors = new ArrayList<UndirectedGraphNode>(); }
1311 * };
1312 */
1313 public class Solution {
1314     public UndirectedGraphNode cloneGraph(UndirectedGraphNode node, HashMap<UndirectedGraphNode, UndirectedGraphNode> map)
1315     {
1316         if(node == null) return null;
1317         if(map.containsKey(node)) return map.get(node);
1318
1319         UndirectedGraphNode nd = new UndirectedGraphNode(node.label);
1320         map.put(node, nd);
1321         for(UndirectedGraphNode N : node.neighbors)
1322         {
1323             nd.neighbors.add(cloneGraph(N, map));
1324         }
1325         return nd;
1326     }
1327
1328     public UndirectedGraphNode cloneGraph(UndirectedGraphNode node)
1329     {
1330         return cloneGraph(node, new HashMap<UndirectedGraphNode, UndirectedGraphNode>());
1331     }
1332 }
1333
1334
1335 ---
1336 public class Solution {
1337
1338
1339     public UndirectedGraphNode cloneGraph(UndirectedGraphNode node)
1340     {
1341
1342         if(node == null) return null;
1343
1344         HashMap<UndirectedGraphNode, UndirectedGraphNode> map =
1345             new HashMap<UndirectedGraphNode, UndirectedGraphNode>();
1346
1347         LinkedList<UndirectedGraphNode> q = new LinkedList<UndirectedGraphNode>();
1348
1349         q.addLast(node);
1350         map.put(node, new UndirectedGraphNode(node.label));
1351

```

```

1352     while(!q.isEmpty())
1353     {
1354         UndirectedGraphNode cur = q.removeFirst();
1355         for(UndirectedGraphNode l:cur.neighbors)
1356         {
1357             if(!map.containsKey(l))
1358             {
1359                 UndirectedGraphNode neighbor = new UndirectedGraphNode(l.label);
1360                 map.get(cur).neighbors.add(neighbor);
1361                 map.put(l,neighbor);
1362                 q.addLast(l);
1363             }
1364             else
1365             {
1366                 map.get(cur).neighbors.add(map.get(l));
1367             }
1368         }
1369     }
1370     return map.get(node);
1371 }
1372 }
1373
1374 ---
1375
1376 public class Solution {
1377
1378     public UndirectedGraphNode cloneGraph(UndirectedGraphNode node)
1379     {
1380
1381         if(node == null) return null;
1382
1383         HashMap<UndirectedGraphNode, UndirectedGraphNode> map =
1384             new HashMap<UndirectedGraphNode, UndirectedGraphNode>();
1385
1386         Stack<UndirectedGraphNode> q = new Stack<UndirectedGraphNode>();
1387
1388         UndirectedGraphNode cp = new UndirectedGraphNode(node.label);
1389         map.put(node, cp);
1390
1391         q.push(node);
1392         while(!q.isEmpty())
1393         {
1394             UndirectedGraphNode cur = q.pop();
1395
1396             for(UndirectedGraphNode l:cur.neighbors)
1397             {
1398                 if(!map.containsKey(l))
1399                 {
1400                     UndirectedGraphNode ll = new UndirectedGraphNode(l.label);
1401                     map.put(l,ll);
1402                     map.get(cur).neighbors.add(ll);
1403                     q.push(l);
1404                 }
1405                 else
1406                 {
1407                     map.get(cur).neighbors.add(map.get(l));
1408                 }
1409             }
1410         }
1411     }
1412
1413     return map.get(node);
1414 }
1415 }
1416
1417 =====
1418 /Users/linxie/Documents/github/leetcodeHub/code/Combinations
1419 =====
1420 =====
1421
1422
1423 Combinations
1424
1425 Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.
1426
1427 For example,
1428 If n = 4 and k = 2, a solution is:
1429
1430 [
1431     [2,4],
1432     [3,4],
1433     [2,3],
1434     [1,2],
1435     [1,3],
1436     [1,4],
1437 ]
1438 ---
1439
1440 public class Solution {
1441     public List<List<Integer>> combine(int n, int k) {
1442         List< List<Integer> > re = new ArrayList<List<Integer>>();
1443         if(n<k || k==0) return re;
1444
1445         if(n==k)
1446         {
1447             List<Integer> r = new ArrayList<Integer>();
1448             for(int i=1; i<= n; i++)
1449                 r.add(i);
1450             re.add(r);
1451             return re;
1452         }
1453         List<List<Integer>> rr = combine(n-1, k-1);
1454         if(rr.size() == 0)
1455         {

```

```

1456         List<Integer> r = new ArrayList<Integer>();
1457         r.add(n);
1458         re.add(r);
1459     }
1460     for(List<Integer> iter : rr)
1461     {
1462         iter.add(n);
1463         re.add(iter);
1464     }
1465     rr = combine(n-1,k);
1466     for(List<Integer> iter:rr)
1467         re.add(iter);
1468
1469     return re;
1470 }
1471 }
1472
1473 =====
1474 /Users/linxie/Documents/github/leetcodeHub/code/CombinationSum
1475 =====
1476
1477
1478
1479 CombinationSum
1480
1481 Given a set of candidate numbers (C) and a target number (T), find all unique combinations
1482 in C where the candidate numbers sums to T.
1483
1484 The same repeated number may be chosen from C unlimited number of times.
1485
1486 Note:
1487 All numbers (including target) will be positive integers.
1488 Elements in a combination (a1, a2, ..., ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ ... ≤ ak).
1489 The solution set must not contain duplicate combinations.
1490 For example, given candidate set 2,3,6,7 and target 7,
1491 A solution set is:
1492 [7]
1493 [2, 2, 3]
1494
1495 ---
1496
1497 public class Solution {
1498     public static void combinationSum(int[] candidates, int ind, int target, List<List<Integer>> re, List<Integer>r)
1499     {
1500
1501         if(target == 0)
1502         {
1503             re.add(new ArrayList<Integer>(r));
1504         }
1505         if(target <=0) return;
1506
1507         if(ind>=candidates.length) return;
1508
1509         int val = candidates[ind];
1510
1511         combinationSum(candidates, ind+1, target, re, r);
1512
1513         int siz = r.size();
1514         while(target>=0)
1515         {
1516             r.add(val);
1517             target -= val;
1518             combinationSum(candidates, ind+1, target, re, r);
1519         }
1520         if(r.size() > siz)
1521         {
1522             for(int i = r.size()-1; i >= siz; i--)
1523                 r.remove(i);
1524         }
1525     }
1526
1527     public static List<List<Integer>> combinationSum(int[] candidates, int target) {
1528         List<List<Integer>> re = new ArrayList<List<Integer>>();
1529         int n = candidates.length;
1530         if(n==0) return re;
1531
1532         if(target == 0)
1533             re.add(new ArrayList<Integer>());
1534         Arrays.sort(candidates);
1535
1536         combinationSum(candidates, 0, target, re, new ArrayList<Integer>());
1537         return re;
1538     }
1539 }
1540
1541
1542 =====
1543 /Users/linxie/Documents/github/leetcodeHub/code/CombinationSumII
1544 =====
1545
1546
1547 CombinationSumII
1548
1549 Given a collection of candidate numbers (C) and a target number (T), find all unique combinations
1550 in C where the candidate numbers sums to T.
1551
1552 Each number in C may only be used once in the combination.
1553
1554 Note:
1555 All numbers (including target) will be positive integers.
1556 Elements in a combination (a1, a2, ..., ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ ... ≤ ak).
1557 The solution set must not contain duplicate combinations.
1558 For example, given candidate set 10,1,2,7,6,1,5 and target 8,
1559 A solution set is:

```

```

1560 [1, 7]
1561 [1, 2, 5]
1562 [2, 6]
1563 [1, 1, 6]
1564 ----
1565
1566 public class Solution {
1567     public static void combinationSum2(List<List<Integer>> re, List<Integer> r, int[] num, int ind, int target)
1568     {
1569         if(target == 0 && r.size()!=0)
1570         {
1571             List<Integer> rr = new ArrayList<Integer>(r);
1572             re.add(rr);
1573             return;
1574         }
1575         if(ind >= num.length || target < num[ind]) return;
1576
1577         int n = ind;
1578         int val = num[ind];
1579         for(;n<num.length && num[n] == val;n++)
1580         {
1581             int sum = 0;
1582             int siz = r.size();
1583
1584             combinationSum2(re, r, num, n, target);
1585             for(int i=ind; i<n;i++)
1586             {
1587                 sum+=val;
1588                 r.add(val);
1589                 combinationSum2(re, r, num, n, target-sum);
1590             }
1591             for(int i=r.size()-1; i>=siz; i--)
1592                 r.remove(r.size()-1);
1593         }
1594     public static List<List<Integer>> combinationSum2(int[] num, int target) {
1595         List<List<Integer>> re = new ArrayList<List<Integer>>();
1596         int n = num.length;
1597         if(n==0) return re;
1598
1599         List<Integer> r = new ArrayList<Integer>();
1600         Arrays.sort(num);
1601
1602         combinationSum2(re, r, num, 0, target);
1603         return re;
1604     }
1605
1606
1607 =====
1608 /Users/linxie/Documents/github/leetcodeHub/code/ConstructBinaryTreefromInorderandPostorderTraversa
1609 =====
1610
1611
1612 ConstructBinaryTreefromInorderandPostorderTraversa
1613
1614 Given inorder and postorder traversal of a tree, construct the binary tree.
1615
1616 Note:
1617 You may assume that duplicates do not exist in the tree.
1618
1619 ---
1620 /**
1621 * Definition for binary tree
1622 * public class TreeNode {
1623 *     int val;
1624 *     TreeNode left;
1625 *     TreeNode right;
1626 *     TreeNode(int x) { val = x; }
1627 * }
1628 */
1629 public class Solution {
1630     public TreeNode buildTree(int[] inorder, int inLeft, int inRight, int[] postorder, int postLeft,
1631                             int postRight)
1632     {
1633         if(inLeft <0 || inRight >= inorder.length || postLeft <0 || inLeft > inRight || (inRight- inLeft != postRight -postLeft)) re
1634
1635         int val = postorder[postRight];
1636         int i = inLeft;
1637         for(;i<inRight;i++)
1638         {
1639             if(inorder[i] == val) break;
1640         }
1641         if(i>inRight) return null;
1642
1643         TreeNode root = new TreeNode(val);
1644         root.left = buildTree(inorder, inLeft, i-1, postorder, postLeft, i-1-inLeft+postLeft);
1645         root.right = buildTree(inorder, i+1, inRight, postorder, postRight+i-inRight, postRight-1);
1646         return root;
1647     }
1648
1649     public TreeNode buildTree(int[] inorder, int[] postorder) {
1650         int n = inorder.length;
1651         if(n==0 || n!= postorder.length) return null;
1652         return buildTree(inorder, 0, n-1, postorder, 0, n-1);
1653     }
1654 }
1655
1656
1657 =====
1658 /Users/linxie/Documents/github/leetcodeHub/code/ConstructBinaryTreefromPreorderandInorderTraversa
1659 =====
1660
1661
1662 ConstructBinaryTreefromPreorderandInorderTraversa
1663

```

```

1664 Given preorder and inorder traversal of a tree, construct the binary tree.
1665
1666 Note:
1667 You may assume that duplicates do not exist in the tree
1668 ---
1669
1670 /**
1671 * Definition for binary tree
1672 * public class TreeNode {
1673 *     int val;
1674 *     TreeNode left;
1675 *     TreeNode right;
1676 *     TreeNode(int x) { val = x; }
1677 * }
1678 */
1679 public class Solution {
1680
1681     int find(int[] arr, int l, int r, int val)
1682     {
1683         for(int i=r; i>=l; i--)
1684             if(arr[i] == val)
1685                 return i;
1686         return -1;
1687     }
1688
1689     public TreeNode buildTree(int[] preorder, int[] inorder, int s1, int e1, int s2, int e2)
1690     {
1691         if(s1>e1 || s2>e2 || (e1-s1) != (e2-s2) ) return null;
1692
1693         TreeNode nd = new TreeNode(preorder[s1]);
1694         int mid = find(inorder, s2, e2, preorder[s1]);
1695         if(mid == -1) return null;
1696
1697         nd.left = buildTree(preorder, inorder, s1+1, s1+(mid-s2), s2, mid-1);
1698         nd.right = buildTree(preorder, inorder, e1-e2+mid+1, e1, mid+1, e2);
1699         return nd;
1700     }
1701
1702     public TreeNode buildTree(int[] preorder, int[] inorder) {
1703         int n = inorder.length;
1704         if(n!=preorder.length) return null;
1705
1706         return buildTree(preorder, inorder, 0, n-1, 0, n-1);
1707     }
1708 }
1709
1710
1711 =====
1712 /Users/linxie/Documents/github/leetcodeHub/code/ContainerWithMostWater
1713 =====
1714
1715
1716 ContainerWithMostWater
1717
1718 Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai).
1719 n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two
1720 lines, which together with x-axis forms a container, such that the container contains the most water.
1721
1722 Note: You may not slant the container.
1723
1724 ---
1725
1726 public class Solution {
1727     public int maxArea(int[] height) {
1728
1729         int n = height.length;
1730         if(n<=1) return 0;
1731
1732         int l = 0, r = n-1;
1733         int max = Integer.MIN_VALUE;
1734         while(l<r)
1735         {
1736             int local = Math.min(height[l], height[r])*(r-l);
1737             if(local> max) max = local;
1738
1739             if(height[l]<=height[r]) l++;
1740             else r--;
1741         }
1742         return max;
1743     }
1744 }
1745 }
1746
1747
1748 =====
1749 /Users/linxie/Documents/github/leetcodeHub/code/ConvertSortedArraytoBinarySearchTree
1750 =====
1751
1752
1753 ConvertSortedArraytoBinarySearchTree
1754
1755 Given an array where elements are sorted in ascending order, convert it to a height balanced BST.
1756
1757 ---
1758 /**
1759 * Definition for binary tree
1760 * public class TreeNode {
1761 *     int val;
1762 *     TreeNode left;
1763 *     TreeNode right;
1764 *     TreeNode(int x) { val = x; }
1765 * }
1766 */
1767 public class Solution {

```

```

1768     public TreeNode sortedArrayToBST(int[] num, int l, int r) {
1769         int n = r-l+1;
1770         if(n<=0) return null;
1771         if(n==1) return new TreeNode(num[1]);
1772
1773         int mid = n/2+1;
1774         TreeNode root = new TreeNode(num[mid]);
1775         root.left = sortedArrayToBST(num, l, mid-1);
1776         root.right = sortedArrayToBST(num, mid+1, r);
1777         return root;
1778     }
1779
1780     public TreeNode sortedArrayToBST(int[] num) {
1781         return sortedArrayToBST(num, 0, num.length-1);
1782     }
1783 }
1784
1785
1786 =====
1787 /Users/linxie/Documents/github/leetcodeHub/code/ConvertSortedListtoBinarySearchTree
1788 =====
1789
1790
1791 ConvertSortedListtoBinarySearchTree
1792
1793 Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.
1794
1795 ---
1796 /**
1797 * Definition for singly-linked list.
1798 * public class ListNode {
1799 *     int val;
1800 *     ListNode next;
1801 *     ListNode(int x) { val = x; next = null; }
1802 * }
1803 */
1804 /**
1805 * Definition for binary tree
1806 * public class TreeNode {
1807 *     int val;
1808 *     TreeNode left;
1809 *     TreeNode right;
1810 *     TreeNode(int x) { val = x; }
1811 * }
1812 */
1813 public class Solution {
1814     public TreeNode sortedListToBST(ListNode head) {
1815
1816         if(head == null) return null;
1817         if(head.next == null) return new TreeNode(head.val);
1818
1819         ListNode first = head, second = head, prev = null;
1820         int count = 0;
1821
1822         while(first.next !=null)
1823         {
1824             count++;
1825             first = first.next;
1826             if(count%2 ==0){
1827                 prev = second;
1828                 second = second.next;
1829             }
1830         }
1831         if(count%2 == 0)
1832         {
1833             if(prev == null)
1834             {
1835                 TreeNode root = new TreeNode(first.val);
1836                 root.left = new TreeNode(second.val);
1837                 return root;
1838             }
1839             prev.next = null;
1840             TreeNode root = new TreeNode(second.val);
1841             root.left = sortedListToBST(head);
1842             ListNode t = second.next;
1843             second.next = null;
1844             root.right = sortedListToBST(t);
1845             return root;
1846         }
1847         else
1848         {
1849             ListNode t = second.next;
1850             second.next = null;
1851             TreeNode root = new TreeNode(t.val);
1852             root.left = sortedListToBST(head);
1853             root.right = sortedListToBST(t.next);
1854             return root;
1855         }
1856     }
1857 }
1858
1859
1860 =====
1861
1862 Better solution, linear:
1863
1864 //Node is a wrapper class, since java does not support reference.
1865 //Head can not move forward by passing in ListNode.
1866
1867
1868 class Node
1869 {
1870     ListNode head;
1871     Node(ListNode _head)

```

```

1872     {
1873         head = _head;
1874     }
1875 }
1876
1877 public class Solution {
1878     public TreeNode helper(TreeNode head, int start, int end)
1879     {
1880         if(start>end) return null;
1881         int mid = start+(end-start)/2;
1882         TreeNode left = helper(head, start, mid-1);
1883         TreeNode cur = new TreeNode(head.head.val);
1884         cur.left = left;
1885         head.head = head.head.next;
1886         cur.right = helper(head, mid+1, end);
1887         return cur;
1888     }
1889     public TreeNode sortedListToBST(ListNode head)
1890     {
1891         if(head == null) return null;
1892
1893         int count = 0;
1894         ListNode cur = head;
1895         while(cur.next !=null)
1896         {
1897             count++;
1898             cur = cur.next;
1899         }
1900         return helper(new Node(head), 0 ,count);
1901     }
1902 }
1903
1904
1905 =====
1906 /Users/linxie/Documents/github/leetcodeHub/code/CopyListwithRandomPointer
1907 =====
1908
1909
1910 CopyListwithRandomPointer
1911
1912 A linked list is given such that each node contains an additional random pointer which could
1913 point to any node in the list or null.
1914
1915 Return a deep copy of the list.
1916
1917 ---
1918 /**
1919 * Definition for singly-linked list with a random pointer.
1920 * class RandomListNode {
1921 *     int label;
1922 *     RandomListNode next, random;
1923 *     RandomListNode(int x) { this.label = x; }
1924 * };
1925 */
1926 public class Solution {
1927     public RandomListNode copyRandomList(RandomListNode head) {
1928
1929         if(head == null) return null;
1930
1931         RandomListNode cur = head;
1932         while(cur!=null)
1933         {
1934             RandomListNode t = cur.next;
1935             cur.next = new RandomListNode(cur.label);
1936             cur.next.next = t;
1937             cur = t;
1938         }
1939
1940         cur = head;
1941         while(cur != null)
1942         {
1943             if(cur.next == null) System.exit(-1);
1944             if(cur.random == null) cur.next.random = null;
1945             else cur.next.random = cur.random.next;
1946             cur = cur.next.next;
1947         }
1948         cur = head;
1949         RandomListNode newHead = cur.next;
1950
1951         while(cur!=null)
1952         {
1953             RandomListNode t = cur.next;
1954             cur.next = t.next;
1955             if(cur.next !=null) t.next = cur.next.next;
1956             else t.next = null;
1957             cur = cur.next;
1958         }
1959         return newHead;
1960     }
1961 }
1962 }
1963
1964
1965 =====
1966 /Users/linxie/Documents/github/leetcodeHub/code/CountandSay
1967 =====
1968
1969
1970 CountandSay
1971
1972 The count-and-say sequence is the sequence of integers beginning as follows:
1973 1, 11, 21, 1211, 111221, ...
1974
1975 1 is read off as "one 1" or 11.

```

```

1976 11 is read off as "two 1s" or 21.
1977 21 is read off as "one 2, then one 1" or 1211.
1978 Given an integer n, generate the nth sequence.
1979
1980 Note: The sequence of integers will be represented as a string
1981
1982 ---
1983 public class Solution {
1984     public String countAndSay(int n) {
1985         if(n==0) return "";
1986
1987         ArrayList<Integer> re = new ArrayList<Integer>();
1988         int count =0;
1989         int val = 1;
1990
1991         re.add(1);
1992         for(int i=1;i<n;i++)
1993         {
1994             ArrayList<Integer> rr = new ArrayList<Integer>();
1995             for(int j=0;j<re.size();j++)
1996             {
1997                 if(re.get(j) == val) count++;
1998                 else
1999                 {
2000                     rr.add(count);
2001                     rr.add(val);
2002                     val = re.get(j);
2003                     count = 1;
2004                 }
2005             }
2006             rr.add(count);
2007             rr.add(val);
2008             val = rr.get(0);
2009             count = 0;
2010             re =rr;
2011         }
2012         String result = "";
2013         for(int i:re)
2014             result += Integer.toString(i);
2015         return result;
2016     }
2017 }
2018 }
2019
2020
2021 =====
2022 /Users/linxie/Documents/github/leetcodeHub/code/DecodeWays
2023 =====
2024
2025
2026 DecodeWays
2027
2028 A message containing letters from A-Z is being encoded to numbers using the following mapping:
2029
2030 A -> 1
2031 B -> 2
2032 ...
2033 Z -> 26
2034 Given an encoded message containing digits, determine the total number of ways to decode it.
2035
2036 For example,
2037 Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).
2038
2039 The number of ways decoding "12" is 2.
2040 ---
2041
2042 public class Solution {
2043
2044     public boolean validate(String s){
2045         int n = s.length();
2046         if(n!=1 && n!=2) return false;
2047
2048         try
2049         {
2050             int val = Integer.parseInt(s);
2051             if(val >0 && val <27)
2052             {
2053                 if(n==1 || s.charAt(0)!=0) return true;
2054             }
2055         }
2056         catch(Exception o)
2057         {}
2058         return false;
2059     }
2060
2061     public int numDecodings(String s, int[] map)
2062     {
2063         int n = s.length();
2064         if(n==0) return 1;
2065
2066         if(map[n-1]!=-1) return map[n-1];
2067
2068         int re = 0;
2069         if(validate(s.substring(n-1))) re = numDecodings(s.substring(0, n-1), map);
2070         if(n-2>=0 && validate(s.substring(n-2))) re+=numDecodings(s.substring(0, n-2), map);
2071         map[n-1] = re;
2072         return re;
2073     }
2074     public int numDecodings(String s) {
2075         int n = s.length();
2076         if(n==0) return 0;
2077         int[] map = new int[n];
2078         for(int i=0;i<n;i++)
2079             map[i] = -1;

```

```

2080         return numDecodings(s, map);
2081     }
2082 }
2083
2084 =====
2085 /Users/linxie/Documents/github/leetcodeHub/code/DistinctSubsequences
2086 =====
2087 =====
2088
2089
2090 DistinctSubsequences
2091
2092 Given a string S and a string T, count the number of distinct subsequences of T in S.
2093
2094 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
2095
2096 Here is an example:
2097 S = "rabbbit", T = "rabbit"
2098
2099 Return 3.
2100 ---
2101 public class Solution {
2102     public int numDistinct(String S, String T) {
2103         int n = T.length();
2104         if(n==0) return 1;
2105         int m = S.length();
2106         if(m==0) return 0;
2107
2108         int[] re = new int[n+1];
2109         for(int i =1; i<n+1; i++)
2110             re[i] = 0;
2111         re[0] = 1;
2112         for(int i=0;i<m;i++)
2113         {
2114             for(int j = n; j>0; j--)
2115             {
2116                 if(S.charAt(i)==T.charAt(j-1)) re[j] = re[j]+re[j-1];
2117             }
2118         }
2119     }
2120     return re[n];
2121 }
2122
2123 }
2124 }
2125
2126
2127 =====
2128 /Users/linxie/Documents/github/leetcodeHub/code/DivideTwoIntegers
2129 =====
2130
2131
2132 DivideTwoIntegers
2133
2134 Divide two integers without using multiplication, division and mod operator.
2135
2136 ---
2137
2138 public class Solution {
2139     public int divide(int dividend, int divisor) {
2140         if(dividend == 0) return 0;
2141         if(divisor == 0) return Integer.MAX_VALUE;
2142
2143         long d1 = (long)dividend;
2144         long d2 = (long) divisor;
2145         int flg = 0;
2146         if(d1 < 0){ d1=-d1; flg++; }
2147         if(d2 <0) { d2 = -d2; flg++; }
2148         if(d2>d1) return 0;
2149         if(d2==1) return (int)((flg%2==0)?d1:-d1);
2150
2151         long ret = d2;
2152         int m = 1;
2153         while(ret<=d1)
2154         {
2155             if(ret == d1) return (flg%2==0)?m:-m;
2156             ret = ret<<1; m = m<<1;
2157         }
2158         ret=ret>>1; m=m>>1;
2159         long re = m+divide((int)(d1-ret),(int)d2);
2160         return (int)((flg%2==0)?re:-re);
2161
2162
2163     }
2164 }
2165 }
2166
2167
2168 =====
2169 /Users/linxie/Documents/github/leetcodeHub/code/EditDistance
2170 =====
2171
2172
2173 EditDistance
2174
2175 Given two words word1 and word2, find the minimum number of steps required to convert
2176 word1 to word2. (each operation is counted as 1 step.)
2177
2178 You have the following 3 operations permitted on a word:
2179
2180 a) Insert a character
2181 b) Delete a character
2182 c) Replace a character
2183 ---

```

```

2184
2185 public class Solution {
2186     public int minDistance(String word1, String word2) {
2187
2188         char[] a = word1.toCharArray();
2189         char[] b = word2.toCharArray();
2190         int m = a.length;
2191         int n = b.length;
2192         if(m==0) return n;
2193         if(n==0) return m;
2194
2195         int[] re = new int[m+1];
2196         for(int i=0;i<re.length;i++)
2197             re[i] = i;
2198         for(int i=0;i<n;i++)
2199         {
2200             int pre = -1;
2201             for(int j=0;j<re.length;j++)
2202             {
2203                 if(j==0)
2204                 {
2205                     pre = re[j];
2206                     re[j]++;
2207                 }
2208                 else
2209                 {
2210                     int t = re[j];
2211                     if(a[j-1] != b[i]) re[j] = Math.min(re[j-1], Math.min(pre, re[j]))+1;
2212                     else
2213                         re[j] = pre;
2214                     pre = t;
2215                 }
2216             }
2217         }
2218         return re[m];
2219     }
2220 }
2221 }
2222
2223
2224 =====
2225 /Users/linxie/Documents/github/leetcodeHub/code/EvaluateReversePolishNotation
2226 =====
2227
2228
2229 EvaluateReversePolishNotation
2230
2231 Evaluate the value of an arithmetic expression in Reverse Polish Notation.
2232
2233 Valid operators are +, -, *, /. Each operand may be an integer or another expression.
2234
2235 Some examples:
2236 ["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
2237 ["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6
2238
2239 ----
2240 public class Solution {
2241     public int evalRPN(String[] tokens) {
2242
2243         int n = tokens.length;
2244         if(n==0) return 0;
2245
2246         Stack<String> s = new Stack<String>();
2247         try{
2248             for(String t:tokens)
2249             {
2250                 if(t.equals("+"))
2251                 {
2252                     if(s.isEmpty()) break;
2253                     int a = Integer.parseInt(s.pop());
2254                     if(s.isEmpty()) break;
2255                     int b = Integer.parseInt(s.pop());
2256                     s.push(Integer.toString(a+b));
2257                 }
2258                 else if(t.equals("-"))
2259                 {
2260                     if(s.isEmpty()) break;
2261                     int a = Integer.parseInt(s.pop());
2262                     if(s.isEmpty()) break;
2263                     int b = Integer.parseInt(s.pop());
2264                     s.push(Integer.toString(b-a));
2265                 }
2266                 else if(t.equals("*"))
2267                 {
2268                     if(s.isEmpty()) break;
2269                     int a = Integer.parseInt(s.pop());
2270                     if(s.isEmpty()) break;
2271                     int b = Integer.parseInt(s.pop());
2272                     s.push(Integer.toString(a*b));
2273                 }
2274                 else if(t.equals("/"))
2275                 {
2276                     if(s.isEmpty()) break;
2277                     int a = Integer.parseInt(s.pop());
2278                     if(a==0 || s.isEmpty()) break;
2279                     int b = Integer.parseInt(s.pop());
2280                     s.push(Integer.toString(b/a));
2281                 }
2282                 else
2283                 {
2284                     s.push(t);
2285                 }
2286             }
2287         }

```

```

2288         catch(Exception o)
2289     {
2290         System.exit(-1);
2291     }
2292     if(!s.isEmpty()) return Integer.parseInt(s.pop());
2293     else return 0;
2294 }
2295 }
2296 }
2297 =====
2298 =====
2299 /Users/linxie/Documents/github/leetcodeHub/code/Fibonacci
2300 =====
2301 =====
2302
2303
2304 Fibonacci_logN
2305
2306 import java.util.*;
2307 import java.lang.*;
2308
2309 class fourNum
2310 {
2311     int a0, a1, a2, a3;
2312     fourNum(int _a0, int _a1, int _a2, int _a3)
2313     { a0 = _a0; a1=_a1; a2=_a2; a3=_a3;}
2314
2315     public fourNum multiple(fourNum o)
2316     {
2317         fourNum re = new fourNum(0,0,0,0);
2318         re.a0 = a0*o.a0 + a1*o.a2;
2319         re.a1 = a0*o.a1 + a1*o.a3;
2320         re.a2 = a2*o.a0 + a3*o.a2;
2321         re.a3 = a2*o.a1 + a3*o.a3;
2322         return re;
2323     }
2324 }
2325
2326 class Solution
2327 {
2328     public fourNum helper(int n)
2329     {
2330         if(n==0 || n==1) return new fourNum(1,1,1,0);
2331         fourNum t = helper(n/2);
2332         if(n%2==0)
2333         {
2334             return t.multiple(t);
2335         }
2336         else
2337         {
2338             return t.multiple(t).multiple(new fourNum(1,1,1,0));
2339         }
2340     }
2341
2342     public int fibonacci(int n)
2343     {
2344         if(n<=1) return 1;
2345         return helper(n).a0;
2346     }
2347 }
2348
2349
2350
2351 =====
2352 /Users/linxie/Documents/github/leetcodeHub/code/findRotation
2353 =====
2354
2355
2356 Given a rotated array, find the number of rotation
2357
2358 =====
2359
2360     public int findTotate(int[] a)
2361     {
2362         int n = a.length;
2363         if(n<=1) return 0;
2364
2365         int l = 0, r = n-1;
2366         int re = -1;
2367         while(l<r)
2368         {
2369             int mid = l+(r-l)/2;
2370             if(a[mid]>a[l] && a[mid]>=a[r])
2371             {
2372                 re = mid;
2373                 l = mid;
2374             }else if(a[mid]<a[l] && a[mid]<=a[r])
2375             {
2376                 re=mid;
2377                 r = mid;
2378             }
2379         else
2380         {
2381             l++;
2382         }
2383     }
2384     return (re==-1)?0:re;
2385 }
2386
2387
2388 =====
2389 /Users/linxie/Documents/github/leetcodeHub/code/FirstMissingPositive
2390 =====
2391

```

```

2392
2393 FirstMissingPositive
2394
2395 Given an unsorted integer array, find the first missing positive integer.
2396
2397 For example,
2398 Given [1,2,0] return 3,
2399 and [3,4,-1,1] return 2.
2400
2401 Your algorithm should run in O(n) time and uses constant space.
2402
2403 ---
2404
2405 public class Solution {
2406     public static int firstMissingPositive(int[] A) {
2407         int n = A.length;
2408         if(n==0) return 1;
2409
2410         for(int i=0;i<n;i++)
2411         {
2412             while(A[i]>0 && A[i]<=n && A[i]!=i+1 && A[A[i]-1]!=A[i])
2413             {
2414                 int t = A[i];
2415                 A[i] = A[t-1];
2416                 A[t-1] = t;
2417             }
2418         }
2419
2420         for(int i=0;i<n;i++)
2421             if(A[i]!=i+1) return i+1;
2422         return n+1;
2423     }
2424 }
2425
2426
2427 =====
2428 /Users/linxie/Documents/github/leetcodeHub/code/FlattenBinaryTreeToLinkedList
2429 =====
2430
2431
2432 FlattenBinaryTreeToLinkedList
2433
2434 Given a binary tree, flatten it to a linked list in-place.
2435
2436 For example,
2437 Given
2438
2439      1
2440      / \
2441      2   5
2442      / \   \
2443      3   4   6
2444 The flattened tree should look like:
2445      1
2446      \
2447      2
2448      \
2449      3
2450      \
2451      4
2452      \
2453      5
2454      \
2455      6
2456 ---
2457 /**
2458 * Definition for binary tree
2459 * public class TreeNode {
2460 *     int val;
2461 *     TreeNode left;
2462 *     TreeNode right;
2463 *     TreeNode(int x) { val = x; }
2464 * }
2465 */
2466 public class Solution {
2467     public void flatten(TreeNode root) {
2468         TreeNode cur = root, pre = null, tail = null;
2469         while(cur!=null)
2470         {
2471             if(cur.left==null)
2472             {
2473                 if(cur == root)
2474                 {
2475                     tail = cur;
2476                 }
2477                 else
2478                 {
2479                     tail.right = cur;
2480                     tail = cur;
2481                 }
2482                 TreeNode tmp = cur.right;
2483                 cur.right = null;
2484                 cur = tmp;
2485             }
2486             else
2487             {
2488                 pre = cur.left;
2489                 while(pre.right!=null) pre = pre.right;
2490                 pre.right = cur.right;
2491                 cur.right = cur.left;
2492                 cur.left = null;
2493             }
2494         }
2495     }
2496 }
```

```

2496
2497     }
2498 }
2500
2501
2502 =====
2503 /Users/linxie/Documents/github/leetcodeHub/code/GasStation
2504 =====
2505
2506
2507 GasStation
2508
2509 There are N gas stations along a circular route, where the amount of gas at station i is gas[i].
2510
2511 You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i
2512 to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.
2513
2514 Return the starting gas stations index if you can travel around the circuit once, otherwise return -1.
2515
2516 Note:
2517 The solution is guaranteed to be unique.
2518
2519
2520 ---
2521 public class Solution {
2522     public int canCompleteCircuit(int[] gas, int[] cost) {
2523         int n = gas.length;
2524         if(n!=cost.length)
2525             return -1;
2526
2527         int[] diff = new int[n];
2528         int[] G = new int[n];
2529         for(int i=0;i<n;i++)
2530             diff[i] = gas[i] - cost[i];
2531
2532         int min = Integer.MAX_VALUE;
2533         int sum = 0;
2534
2535         for(int i=0;i<n;i++)
2536         {
2537             sum+=diff[i];
2538             if(sum<min)
2539                 min = sum;
2540         }
2541         if(min>=0) return 0;
2542
2543         G[0] = min;
2544         for(int i=1;i<n;i++)
2545         {
2546             G[i] = Math.min((G[i-1]-diff[i-1]), sum);
2547             if(G[i]>=0) return i;
2548         }
2549
2550         return -1;
2551     }
2552 }
2553
2554
2555 =====
2556 /Users/linxie/Documents/github/leetcodeHub/code/GenerateParentheses
2557 =====
2558
2559
2560 GenerateParentheses
2561
2562 Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.
2563
2564 For example, given n = 3, a solution set is:
2565
2566 "((()))", "(()())", "(())()", "()(())", "()()()"
2567
2568 ---
2569
2570 public class Solution {
2571     public void helper(int n, int m, int k, List<String> re, String r){
2572         if(k==n)
2573         {
2574             if(m==0)
2575             {
2576                 re.add(new String(r));
2577             }
2578             return;
2579         }
2580         if(m==0)
2581         {
2582             helper(n, m+1,k+1,re, r+"(");
2583         }else
2584         {
2585             helper(n, m-1, k+1,re,r+ ")");
2586             helper(n, m+1, k+1,re,r+ "(");
2587         }
2588     }
2589     public List<String> generateParenthesis(int n) {
2590         List<String> re = new ArrayList<String>();
2591         if(n==0) return re;
2592         helper(2*n, 0, 0, re, "");
2593         return re;
2594     }
2595 }
2596
2597
2598 =====
2599 /Users/linxie/Documents/github/leetcodeHub/code/GrayCode

```

```

2600 =====
2601
2602
2603 GrayCode
2604
2605 The gray code is a binary numeral system where two successive values differ in only one bit.
2606
2607 Given a non-negative integer n representing the total number of bits in the code, print
2608 the sequence of gray code. A gray code sequence must begin with 0.
2609
2610 For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:
2611
2612 00 - 0
2613 01 - 1
2614 11 - 3
2615 10 - 2
2616 ---
2617 public class Solution {
2618     public List<Integer> grayCode(int n) {
2619
2620         List<Integer> re = new ArrayList<Integer>();
2621         if(n<0) return re;
2622         for(int i=0; i<(1<<n); i++)
2623         {
2624             re.add(i^(i>>1));
2625         }
2626         return re;
2627     }
2628 }
2629 }
2630
2631
2632 =====
2633 /Users/linxie/Documents/github/leetcodeHub/code/ImplementstrStr
2634 =====
2635
2636
2637 ImplementstrStr
2638
2639 Implement strStr().
2640
2641 Returns a pointer to the first occurrence of needle in haystack, or null if needle is
2642 not part of haystack.
2643
2644 ---
2645
2646 public class Solution {
2647     public static int[] prefix(char[] a)
2648     {
2649         int n = a.length;
2650         if(n==0) return null;
2651
2652         int[] F = new int[n+1];
2653         F[0] = F[1] = 0;
2654         for(int i=2; i<=n; i++)
2655         {
2656             int j = F[i-1];
2657             while(true)
2658             {
2659                 if(a[j] == a[i-1]) { F[i] = j+1; break; }
2660                 if(j==0){ F[i] = 0; break; }
2661                 j = F[j];
2662             }
2663         }
2664         return F;
2665     }
2666
2667     public String strStr(String haystack, String needle)
2668     {
2669         char[] ss = haystack.toCharArray();
2670         char[] pp = needle.toCharArray();
2671         int m = ss.length;
2672         int n = pp.length;
2673         if(ss.length<pp.length) return null;
2674         if(n==0 || haystack.equals(needle)) return haystack;
2675
2676         int[] F = prefix(pp);
2677
2678         int i=0, j=0;
2679         while(i<m)
2680         {
2681             if(ss[i] == pp[j])
2682             {
2683                 i++; j++;
2684                 if(j==n) return haystack.substring(i-n);
2685             }
2686             else if(j>0) j = F[j];
2687             else i++;
2688         }
2689         return null;
2690     }
2691 }
2692
2693
2694 =====
2695 /Users/linxie/Documents/github/leetcodeHub/code/InsertInterval
2696 =====
2697
2698
2699 InsertInterval
2700
2701 Given a set of non-overlapping intervals, insert a new interval into the intervals
2702 (merge if necessary).
2703

```

```

2704 You may assume that the intervals were initially sorted according to their start times.
2705
2706 Example 1:
2707 Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].
2708
2709 Example 2:
2710 Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as [1,2],[3,10],[12,16].
2711
2712 This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].
2713
2714 ---
2715
2716 /**
2717 * Definition for an interval.
2718 * public class Interval {
2719 *     int start;
2720 *     int end;
2721 *     Interval() { start = 0; end = 0; }
2722 *     Interval(int s, int e) { start = s; end = e; }
2723 * }
2724 */
2725 public class Solution {
2726     public List<Interval> insert(List<Interval> intervals, Interval newInterval) {
2727
2728         int n = intervals.size();
2729         if(n==0)
2730         {
2731             intervals.add(newInterval);
2732             return intervals;
2733         }
2734
2735
2736         List<Interval> re = new ArrayList<Interval>();
2737         boolean flg = false;
2738         for(int i = 0;i<n;i++)
2739         {
2740             Interval in = intervals.get(i);
2741             if(flg || in.end<newInterval.start)
2742             {
2743                 re.add(in);
2744             }
2745             else if(newInterval.end < in.start)
2746             {
2747                 re.add(newInterval);
2748                 re.add(in);
2749                 flg = true;
2750
2751             }
2752             else if(newInterval.start <= in.start)
2753             {
2754                 if(newInterval.end<in.end)
2755                 {
2756                     newInterval.end = in.end;
2757                 }
2758             }else
2759             {
2760                 if(newInterval.end <= in.end)
2761                 {
2762                     re.add(in);
2763                     flg = true;
2764                 }
2765                 else
2766                 {
2767                     newInterval.start = in.start;
2768                 }
2769             }
2770         }
2771         if(!flg) re.add(newInterval);
2772         return re;
2773
2774
2775     }
2776 }
2777
2778
2779 =====
2780 /Users/linxie/Documents/github/leetcodeHub/code/InsertintoaCyclicSortedList
2781 =====
2782
2783
2784 InsertintoaCyclicSortedList
2785
2786 Given a node from a cyclic linked list which has been sorted, write a function to insert
2787 a value into the list such that it remains a cyclic sorted list. The given node can be
2788 any single node in the list.
2789
2790 --
2791
2792 class Solution
2793 {
2794     public void insert(ListNode node, int x)
2795     {
2796         if(node == null)
2797         {
2798             node = new ListNode(x);
2799             node.next = node;
2800             return;
2801         }
2802
2803         ListNode cur = node, prev = null;
2804         do
2805         {
2806             prev = cur;
2807             cur = cur.next;

```

```

2808         if(x>=prev.val && x<cur.val) break;
2809     else if(prev.val >cur.val && (x>prev.val || x<cur.val)) break;
2810   }
2811   while(cur!=node);
2812 
2813   prev.next = new ListNode(x);
2814   prev.next.next = cur;
2815 }
2816 }
2817 
2818 =====
2819 /Users/linxie/Documents/github/leetcodeHub/code/InsertionSortList
2820 =====
2821 
2822 
2823 
2824 InsertionSortList
2825 
2826 Sort a linked list using insertion sort
2827 ---
2828 /**
2829 * Definition for singly-linked list.
2830 * public class ListNode {
2831 *     int val;
2832 *     ListNode next;
2833 *     ListNode(int x) {
2834 *         val = x;
2835 *         next = null;
2836 *     }
2837 * }
2838 */
2839 public class Solution {
2840     public ListNode insertionSortList(ListNode head) {
2841         if(head == null || head.next == null) return head;
2842 
2843         ListNode cur = head.next;
2844         ListNode tail = head;
2845         head.next = null;
2846 
2847         while(cur!=null)
2848         {
2849             ListNode t = cur.next;
2850             cur.next = null;
2851             ListNode nd = head, prev = null;
2852 
2853             while(nd!=null && nd.val<=cur.val)
2854             {
2855                 prev = nd;
2856                 nd = nd.next;
2857             }
2858             if(nd==null)
2859             {
2860                 tail.next = cur;
2861                 tail = cur;
2862             }
2863             else
2864             {
2865                 if(prev == null)
2866                 {
2867                     cur.next = head;
2868                     head = cur;
2869                 }
2870                 else
2871                 {
2872                     prev.next = cur;
2873                     cur.next = nd;
2874                 }
2875             }
2876         }
2877         cur = t;
2878     }
2879     return head;
2880 }
2881 }
2882 }
2883 
2884 =====
2885 /Users/linxie/Documents/github/leetcodeHub/code/IntegertoRoman
2886 =====
2887 
2888 
2889 
2890 IntegertoRoman
2891 
2892 Given an integer, convert it to a roman numeral.
2893 
2894 Input is guaranteed to be within the range from 1 to 3999.
2895 
2896 ---
2897 
2898 public class Solution {
2899     public String intToRoman(int num) {
2900         if(num<=0 || num>3999) return "";
2901         String[] m = {"M", "D", "C", "L", "X", "V", "I"};
2902 
2903         String re = "";
2904         int div = 1000;
2905         for(int i=0; i<=7; i+=2)
2906         {
2907             int val = num/div;
2908             if(val<4)
2909             {
2910                 for(int j=0;j<val; j++)
2911                     re+=m[i];
2912             }
2913             else if(val==4)
2914             {
2915                 re+=m[i];
2916                 re+=m[i+1];
2917             }
2918             else if(val<9)
2919             {
2920                 re+=m[i];
2921                 for(int j=1;j<=val-5; j++)
2922                     re+=m[i+1];
2923             }
2924             else if(val==9)
2925             {
2926                 re+=m[i];
2927                 re+=m[i+1];
2928                 re+=m[i+1];
2929             }
2930         }
2931     }
2932 }
```

```

2912         }else if(val <5)
2913     {
2914         re+=m[i];
2915         re+=m[i-1];
2916     }else if(val==5)
2917     {
2918         re+=m[i-1];
2919     }else if(val<9)
2920     {
2921         re+=m[i-1];
2922         for(int j=0;j<val-5; j++)
2923         {
2924             re+=m[i];
2925         }
2926     }
2927 }else if(val == 9)
2928 {
2929     re+=m[i];
2930     re+=m[i-2];
2931 }else break;
2932 num%=div;
2933 div/=10;
2934 }
2935 return re;
2936 }
2937 }
2938
2939
2940 =====
2941 /Users/linxie/Documents/github/leetcodeHub/code/InterleavingString
2942 =====
2943
2944
2945 InterleavingString
2946
2947 Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.
2948
2949 For example,
2950 Given:
2951 s1 = "aabcc",
2952 s2 = "dbbca",
2953
2954 When s3 = "adbcbcbcac", return true.
2955 When s3 = "adbbbabacc", return false.
2956
2957 ---
2958 public class Solution {
2959     public int isInterleave(String s1, String s2, String s3, int[][] map)
2960     {
2961
2962         int m=s1.length();
2963         int n=s2.length();
2964
2965         if(map[m][n] !=0) return map[m][n];
2966
2967         if(m+n!=s3.length())
2968         {
2969             map[m][n] = -1;
2970         }
2971         else
2972         {
2973             int re = -1;
2974             if(m==0) re = (s2.equals(s3))?1:-1;
2975             else if(n==0) re = (s1.equals(s3))?1:-1;
2976             else
2977             {
2978                 int r1 = 0, r2 = 0;
2979                 if(s1.charAt(m-1) == s3.charAt(m+n-1))
2980                 {
2981                     if(m==1) r1=isInterleave("", s2, s3.substring(0,m+n-1), map);
2982                     else r1 = isInterleave(s1.substring(0,m-1), s2, s3.substring(0,m+n-1), map);
2983                 }
2984                 if(s2.charAt(n-1) == s3.charAt(m+n-1))
2985                 {
2986                     if(n==1) r2=isInterleave(s1, "", s3.substring(0,m+n-1), map);
2987                     else r2 = isInterleave(s1,s2.substring(0,n-1), s3.substring(0,m+n-1), map);
2988                 }
2989                 if(r1==1 || r2==1)
2990                     re = 1;
2991                 else
2992                     re = -1;
2993             }
2994             map[m][n] = re;
2995         }
2996         return map[m][n];
2997     }
2998
2999     public boolean isInterleave(String s1, String s2, String s3)
3000     {
3001         int m = s1.length();
3002         int n=s2.length();
3003         if(m==0) return s2.equals(s3);
3004         if(n==0) return s1.equals(s3);
3005         int[][] map = new int[m+1][n+1];
3006         int re = isInterleave(s1, s2, s3, map);
3007         return re== 1;
3008     }
3009 }
3010
3011
3012 =====
3013 /Users/linxie/Documents/github/leetcodeHub/code/JumpGame
3014 =====
3015

```

```

3016
3017 JumpGame
3018
3019 Given an array of non-negative integers, you are initially positioned at the first index of the array.
3020
3021 Each element in the array represents your maximum jump length at that position.
3022
3023 Determine if you are able to reach the last index.
3024
3025 For example:
3026 A = [2,3,1,1,4], return true.
3027
3028 A = [3,2,1,0,4], return false.
3029
3030 ---
3031 public class Solution {
3032     public boolean canJump(int[] A) {
3033         int n = A.length;
3034         if(n<=1) return true;
3035
3036         int max=0;
3037         int l=0, r=0;
3038         while(l<=r)
3039         {
3040             for(int i=l; i<=r; i++)
3041             {
3042                 if(i+A[i] > max) max = i+A[i];
3043                 if(max >= n-1) return true;
3044             }
3045             l = r+1;
3046             r = max;
3047         }
3048         return false;
3049     }
3050 }
3051
3052 =====
3053 /Users/linxie/Documents/github/leetcodeHub/code/JumpGameII
3054 =====
3055 =====
3056
3057
3058 JumpGameII
3059
3060 Given an array of non-negative integers, you are initially positioned at the first index
3061 of the array.
3062
3063 Each element in the array represents your maximum jump length at that position.
3064
3065 Your goal is to reach the last index in the minimum number of jumps.
3066
3067 For example:
3068 Given array A = [2,3,1,1,4]
3069
3070 The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1,
3071 then 3 steps to the last index.)
3072
3073 ---
3074
3075 public class Solution {
3076     public int jump(int[] A) {
3077         int n = A.length;
3078         if(n<=1) return 0;
3079
3080         int jump = 0;
3081         int l = 0, r = 0;
3082
3083         while(l<=r && r<n)
3084         {
3085             int max = r;
3086             jump++;
3087             for(int i = l; i<=r; i++)
3088             {
3089                 if(A[i]+i > max) max = A[i]+i;
3090                 if(max >= n-1) return jump;
3091             }
3092             l = r+1;
3093             r = max;
3094         }
3095         return 0;
3096     }
3097 }
3098
3099
3100 =====
3101 /Users/linxie/Documents/github/leetcodeHub/code/LargestBSTinBT
3102 =====
3103
3104
3105
3106 Given a binary tree, find the largest Binary Search Tree (BST), where largest
3107 means BST with largest number of nodes in it. The largest BST may or may not
3108 include all of its descendants.
3109
3110 ----
3111
3112
3113 class returnType
3114 {
3115     int countNode;
3116     int maxNode;
3117     TreeNode largestBST;
3118     TreeNode child;
3119

```

```

3120     returnType(_countNode, _maxNode, _largestBST, _child)
3121     {
3122         countNode = _countNode;
3123         _maxNode = maxNode;
3124         _largestBST = largestBST;
3125         _child = child;
3126     }
3127 }
3128
3129 class Solution
3130 {
3131
3132     TreeNode helper(TreeNode p, int min, int max, returnType cur)
3133     {
3134         if(p==null)
3135         {
3136             return new returnType(0, cur.maxNode, cur.largestBST, cur.child);
3137         }
3138
3139         if(min<p.val && p.val<max)
3140         {
3141             returnType left = helper(p.left, min, p.val, cur);
3142             TreeNode re = new TreeNode(p.val);
3143             re.left = (left.countNode)?null:left.child;
3144
3145             returnType right = helper(p.right, p.val, max, cur);
3146             re.right = (right.countNode)?null:right.child;
3147
3148             cur.child = re;
3149             int total = left.countNode+right.countNode+1;
3150             if(total > cur.maxNode)
3151             {
3152                 cur.maxNode = total;
3153                 cur.largestBST = re;
3154             }
3155             return new returnType(total, cur.maxNode, cur.largestBST, cur.child);
3156         }
3157         else
3158         {
3159             returnType re = helper(p, Integer.MIN_VALUE Integer.MAX_VALUE, cur);
3160             re.countNode = 0;
3161             return re;
3162         }
3163     }
3164
3165     TreeNode findLargestBST(TreeNode root)
3166     {
3167         returnType re = helper(root, Integer.MIN_VALUE, Integer.MAX_VALUE,
3168             new returnType(0, Integer.MIN_VALUE, null, null));
3169         return re.largestBST;
3170     }
3171
3172 }
3173
3174
3175
3176 =====
3177 /Users/linxie/Documents/github/leetcodeHub/code/LargestRectangleinHistogram
3178 =====
3179
3180
3181 LargestRectangleinHistogram
3182
3183 Given n non-negative integers representing the histograms bar height where the width of
3184 each bar is 1, find the area of largest rectangle in the histogram.
3185
3186
3187 Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].
3188
3189
3190 The largest rectangle is shown in the shaded area, which has area = 10 unit.
3191
3192 For example,
3193 Given height = [2,1,5,6,2,3],
3194 return 10.
3195
3196 --
3197 public class Solution {
3198     public int largestRectangleArea(int[] height) {
3199         int n = height.length;
3200         if(n==0) return 0;
3201         if(n==1) return height[0];
3202
3203         Stack<Integer> st = new Stack<Integer>();
3204         int min = 0;
3205
3206         for(int i=0;i<n;i++)
3207         {
3208             if(st.isEmpty() || height[i]>=height[st.peek()])
3209                 st.push(i);
3210             else{
3211                 int tmp = height[st.pop()];
3212                 int local = tmp* (st.isEmpty()?i:(i-st.peek()-1));
3213                 if(local >min) min = local;
3214                 i--;
3215             }
3216         }
3217
3218         while(!st.isEmpty())
3219         {
3220             int tmp = height[st.pop()];
3221             int local = tmp* (st.isEmpty()?n:(n-st.peek()-1));
3222             if(local >min) min = local;
3223         }

```

```

3224         return min;
3225     }
3226 }
3227
3228
3229 =====
3230 /Users/linxie/Documents/github/leetcodeHub/code/LengthofLastWord
3231 =====
3232
3233
3234 LengthofLastWord
3235
3236 Given a string s consists of upper/lower-case alphabets and empty space characters ,
3237 return the length of last word in the string.
3238
3239 If the last word does not exist, return 0.
3240
3241 Note: A word is defined as a character sequence consists of non-space characters only.
3242
3243 For example,
3244 Given s = "Hello World",
3245 return 5.
3246 --
3247
3248 public class Solution {
3249     public int lengthOfLastWord(String s) {
3250         char[] c = s.toCharArray();
3251         int n = c.length;
3252         if(n==0) return 0;
3253         int len = 0;
3254         int i=0, j=0;
3255         for(;j<n;j++)
3256         {
3257             while(j<n && c[j] == ) j++;
3258             if(j==n) break;
3259             i = j;
3260             while(j<n && c[j]!= ) j++;
3261             len = j-i;
3262         }
3263         return len;
3264     }
3265 }
3266
3267
3268 =====
3269 /Users/linxie/Documents/github/leetcodeHub/code/LetterCombinationsofaPhoneNumber
3270 =====
3271
3272
3273 LetterCombinationsofaPhoneNumber
3274
3275 Input:Digit string "23"
3276 Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
3277
3278 ---
3279
3280 public class Solution {
3281     public void letterCombinations(char[] digits, int i, List<String> re, char[] r, String[] map)
3282     {
3283         if(i==digits.length)
3284         {
3285             re.add(new String(r));
3286             return;
3287         }
3288         if(digits[i] <0 || digits[i] >9) return;
3289         for(char c:map[(int)(digits[i]-0)-1].toCharArray())
3290         {
3291             r[i] = c;
3292             letterCombinations(digits,i+1,re,r, map);
3293         }
3294     }
3295     public List<String> letterCombinations(String digits)
3296     {
3297         String[] map = {"", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
3298         List<String> re = new ArrayList<String>();
3299
3300         int n = digits.length();
3301         if(n==0)
3302         {
3303             re.add("");
3304             return re;
3305         }
3306
3307         letterCombinations(digits.toCharArray(), 0, re, new char[n], map);
3308         return re;
3309     }
3310 }
3311 }
3312
3313
3314 =====
3315 /Users/linxie/Documents/github/leetcodeHub/code/LinkedListCycle
3316 =====
3317
3318
3319 LinkedListCycle
3320
3321 Given a linked list, determine if it has a cycle in it.
3322
3323 Follow up:
3324 Can you solve it without using extra space?
3325 ---
3326 /**
3327 * Definition for singly-linked list.

```

```

3328 * class ListNode {
3329 *     int val;
3330 *     ListNode next;
3331 *     ListNode(int x) {
3332 *         val = x;
3333 *         next = null;
3334 *     }
3335 * }
3336 */
3337 public class Solution {
3338     public boolean hasCycle(ListNode head) {
3339         if(head == null || head.next == null) return false;
3340         if(head.next == head) return true;
3341         ListNode left = head, right = head;
3342         int count = 0;
3343         while(right.next != null)
3344         {
3345             right = right.next;
3346             count++;
3347             if(count%2 == 0)
3348             {
3349                 left = left.next;
3350                 if(right == left) return true;
3351             }
3352         }
3353         return false;
3354     }
3355 }
3356
3357 =====
3358 /Users/linxie/Documents/github/leetcodeHub/code/LinkedListCycleII
3360 =====
3361
3362
3363 LinkedListCycleII
3364
3365 Given a linked list, return the node where the cycle begins. If there is no cycle, return null.
3366
3367 Follow up:
3368 Can you solve it without using extra space?
3369
3370 ---
3371
3372 /**
3373 * Definition for singly-linked list.
3374 * class ListNode {
3375 *     int val;
3376 *     ListNode next;
3377 *     ListNode(int x) {
3378 *         val = x;
3379 *         next = null;
3380 *     }
3381 * }
3382 */
3383 public class Solution {
3384     public ListNode detectCycle(ListNode head) {
3385
3386         if(head == null || head.next == null) return null;
3387         if(head.next == head) return head;
3388         ListNode left = head, right = head;
3389         int count = 0;
3390         while(right.next != null)
3391         {
3392             right = right.next;
3393             count++;
3394             if(count%2 == 0)
3395             {
3396                 left = left.next;
3397                 if(right == left) break;
3398             }
3399         }
3400         if(right.next == null) return null;
3401         left = head;
3402         while(left!=right){
3403             left=left.next;
3404             right=right.next;
3405         }
3406         return left;
3407     }
3408 }
3409
3410
3411 =====
3412 /Users/linxie/Documents/github/leetcodeHub/code/LongestCommonPrefix
3413 =====
3414
3415
3416 LongestCommonPrefix
3417
3418 Write a function to find the longest common prefix string amongst an array of strings.
3419
3420 ---
3421
3422 public class Solution {
3423     public static boolean longestCommonPrefix(String[] strs, int k)
3424     {
3425         String first = strs[0];
3426         if(first.length() <=k) return false;
3427         char c = first.charAt(k);
3428         for(String s:strs)
3429         {
3430             if(s.length()<=k || s.charAt(k)!=c) return false;
3431         }

```

```

3432     return true;
3433 }
3434
3435 public String longestCommonPrefix(String[] strs)
3436 {
3437     int n = strs.length;
3438     if(n==0) return "";
3439     if(n==1) return strs[0];
3440
3441     int i = 0;
3442     for(; i< strs[0].length(); i++)
3443     {
3444         if(!longestCommonPrefix(strs,i)) break;
3445     }
3446     return strs[0].substring(0, i);
3447 }
3448 }
3449
3450
3451 =====
3452 /Users/linxie/Documents/github/leetcodeHub/code/LongestConsecutiveSequence
3453 =====
3454
3455
3456 LongestConsecutiveSequence
3457
3458 Given an unsorted array of integers, find the length of the longest consecutive
3459 elements sequence.
3460
3461 For example,
3462 Given [100, 4, 200, 1, 3, 2],
3463 The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.
3464
3465 Your algorithm should run in O(n) complexity.
3466
3467 ---
3468 public class Solution {
3469     public int longestConsecutive(int[] num) {
3470
3471         HashSet<Integer> m = new HashSet<Integer>();
3472         for(int i:num)
3473             m.add(i);
3474
3475         int max = 0;
3476
3477         HashSet<Integer> mm = new HashSet<Integer>(m);
3478         for(int i: mm)
3479         {
3480             if(!m.contains(i)) continue;
3481
3482             int t = i;
3483             int count = 1;
3484             m.remove(t);
3485             while(m.contains(++t)) count++;
3486             t=i;
3487             while(m.contains(--t)) count++;
3488             if(count>max) max=count;
3489         }
3490         return max;
3491     }
3492 }
3493 }
3494
3495
3496 =====
3497 /Users/linxie/Documents/github/leetcodeHub/code/LongestPalindromicSubstring
3498 =====
3499
3500
3501 LongestPalindromicSubstring
3502
3503 Given a string S, find the longest palindromic substring in S. You may assume that the
3504 maximum length of S is 1000, and there exists one unique longest palindromic substring.
3505
3506 ---
3507
3508 public class Solution {
3509     public String longestPalindrome(String s) {
3510         int n = s.length();
3511         if(n<2) return s;
3512
3513         int maxStart = 1, maxEnd = 0;
3514         for(int i=0;i<n;i++)
3515         {
3516             int l = i, r = i;
3517             while(l>=0 && r<n && s.charAt(l) == s.charAt(r)) {l--;r++;}
3518             if(maxEnd - maxStart < r-l-2)
3519             {
3520                 maxStart = l+1;
3521                 maxEnd = r-1;
3522             }
3523         }
3524         for(int i=0;i<n;i++)
3525         {
3526             int l = i, r = i+1;
3527             while(l>=0 && r<n && s.charAt(l) == s.charAt(r)) {l--;r++;}
3528             if(maxEnd - maxStart < r-l-2)
3529             {
3530                 maxStart = l+1;
3531                 maxEnd = r-1;
3532             }
3533         }
3534         return s.substring(maxStart, maxEnd+1);
3535     }

```

```

3536 }
3537
3538
3539 =====
3540 /Users/linxie/Documents/github/leetcodeHub/code/LongestSubstringWithoutRepeatingCharacters
3541 =====
3542
3543
3544 LongestSubstringWithoutRepeatingCharacters
3545
3546 Given a string, find the length of the longest substring without repeating characters.
3547 For example, the longest substring without repeating letters for "abcabcbb" is "abc",
3548 which the length is 3. For "bbbb" the longest substring is "b", with the length of 1.
3549
3550 ---
3551
3552 public class Solution {
3553     public int lengthOfLongestSubstring(String s) {
3554
3555         int n=s.length();
3556         if(n<=1) return n;
3557
3558         HashSet<Character> map = new HashSet<Character>();
3559         int max = 0;
3560         int i=0, j=0;
3561
3562         for(;j<n;j++)
3563         {
3564             char c = s.charAt(j);
3565             if(!map.contains(c)){
3566                 map.add(c);
3567             }
3568             else
3569             {
3570                 if(max < j-i) max = j-i;
3571                 for(;i<=j && s.charAt(i)!=c ;i++)
3572                 {
3573                     map.remove(s.charAt(i));
3574                 }
3575                 i++;
3576             }
3577         }
3578         if(max < n-i) max = n-i;
3579         return max;
3580
3581     }
3582 }
3583
3584
3585
3586 =====
3587 /Users/linxie/Documents/github/leetcodeHub/code/LongestValidParentheses
3588 =====
3589
3590
3591 LongestValidParentheses
3592
3593 Given a string containing just the characters ( and ), find the length of the longest valid
3594 (well-formed) parentheses substring.
3595
3596 For "()", the longest valid parentheses substring is "()", which has length = 2.
3597
3598 Another example is ")()()", where the longest valid parentheses substring is "()()", which has length = 4.
3599
3600 ---
3601
3602 public class Solution {
3603     public int longestValidParentheses(String s) {
3604         int n = s.length();
3605         if(n<=1) return 0;
3606
3607         Stack<Integer> st = new Stack<Integer>();
3608         boolean[] b = new boolean[n];
3609         for(int i=0;i<n;i++)
3610         {
3611             char c = s.charAt(i);
3612             if(c == ')') && !st.isEmpty() && s.charAt(st.peek()) == '('
3613             {
3614                 b[i] = true; b[st.peek()] = true;
3615                 st.pop();
3616             }
3617             else
3618             {
3619                 while(!st.isEmpty() && s.charAt(st.peek()) == ')') st.pop();
3620                 st.push(i);
3621             }
3622         }
3623         int max = 0;
3624         int local = 0;
3625         for(int i =0; i<n; i++)
3626         {
3627             if(b[i]) local++;
3628             else
3629             {
3630                 if(max <local) max = local;
3631                 local = 0;
3632             }
3633         }
3634         if(max <local) max = local;
3635         return max;
3636     }
3637 }
3638
3639

```

```

3640 =====
3641 /Users/linxie/Documents/github/leetcodeHub/code/LowestCommonAncestorofaBinaryTreePartI
3642 =====
3643
3644
3645 LowestCommonAncestorofaBinaryTreePartI
3646
3647 Given a binary tree, find the lowest common ancestor of two given nodes in the tree.
3648
3649 ===
3650
3651 The solution below is a flaw one, since it hold the assumption that the tree has those two
3652 nodes and those nodes are different. This is not true for general case.
3653
3654 node LCA(node root, node p, node q)
3655 {
3656     if(root == null) return null;
3657     if(root == p || root == q) return root;
3658
3659     node l = LCA(root.left, p, q);
3660     node r = LCA(root.right, p, q);
3661
3662     if(l!=null && r!=null) return root;
3663     return (l!=null)?l:r;
3664 }
3665
3666
3667 =====
3668 /Users/linxie/Documents/github/leetcodeHub/code/LRUCache
3669 =====
3670
3671
3672 LRUCache
3673
3674 Design and implement a data structure for Least Recently Used (LRU) cache. It should support
3675 the following operations: get and set.
3676
3677 get(key) - Get the value (will always be positive) of the key if the key exists in the cache,
3678 otherwise return -1.
3679 set(key, value) - Set or insert the value if the key is not already present. When the cache
3680 reached its capacity, it should invalidate the least recently used item before inserting a new item.
3681 ----
3682
3683 class mem
3684 {
3685     int key;
3686     int value;
3687     mem prev;
3688     mem next;
3689     mem(int _key, int _value)
3690     {
3691         key = _key;
3692         value = _value;
3693         prev = null;
3694         next = null;
3695     }
3696
3697 }
3698
3699 public class LRUCache {
3700     int capacity;
3701     int size;
3702     mem head;
3703     mem tail;
3704     HashMap<Integer, mem> map;
3705
3706     public LRUCache(int capacity) {
3707         this.capacity = capacity;
3708         map = new HashMap<Integer, mem>();
3709         size = 0;
3710         mem head = null;
3711         mem tail = null;
3712     }
3713
3714     public int get(int key) {
3715         if(!map.containsKey(key)) return -1;
3716
3717         mem m = map.get(key);
3718         if(size == 1 || m == head)
3719         {
3720             return m.value;
3721         }
3722
3723         if(m==tail)
3724         {
3725             tail = m.prev;
3726             m.prev = null;
3727             tail.next = null;
3728
3729             m.next = head;
3730             head.prev = m;
3731             head = m;
3732         }
3733     else
3734     {
3735         m.prev.next = m.next;
3736         m.next.prev = m.prev;
3737
3738         m.next = head;
3739         head.prev = m;
3740         m.prev = null;
3741         head = m;
3742     }
3743     return m.value;
3744 }

```

```

3744     }
3745
3746     public void set(int key, int value) {
3747         if( map.containsKey(key) )
3748         {
3749             mem m = map.get(key);
3750             m.value = value;
3751             get(key);
3752         }
3753     else
3754     {
3755         if(size<capacity)
3756         {
3757             mem m = new mem(key, value);
3758             if(size == 0)
3759             {
3760                 head = m;
3761                 tail = m;
3762             }
3763         else
3764         {
3765             m.next = head;
3766             head.prev = m;
3767             head = m;
3768         }
3769         map.put(key, m);
3770         size++;
3771     }
3772     else
3773     {
3774         int k = tail.key;
3775         map.remove(k);
3776         tail.value = value;
3777         tail.key = key;
3778         map.put(key, tail);
3779         get(key);
3780     }
3781 }
3782
3783 }
3784 }
3785
3786
3787 =====
3788 /Users/linxie/Documents/github/leetcodeHub/code/MaximalRectangle
3789 =====
3790
3791
3792 MaximalRectangle
3793
3794 Given a 2D binary matrix filled with 0s and 1s, find the largest rectangle containing all
3795 ones and return its area
3796 ---
3797
3798 public class Solution {
3799     public static int maxHis(int[] a)
3800     {
3801         int n = a.length;
3802         if(n==0) return 0;
3803         if(n==1) return a[0];
3804         int max = 0;
3805         Stack<Integer> s = new Stack<Integer>();
3806         for(int i=0;i<n;i++)
3807         {
3808             if(s.isEmpty() || a[i] >= a[s.peek()]) s.push(i);
3809             else
3810             {
3811                 int tmp = a[s.pop()];
3812                 int local = tmp * (s.isEmpty()?i:s.peek()-1);
3813                 if(local > max) max = local;
3814                 i--;
3815             }
3816         }
3817     }
3818
3819     while(!s.isEmpty())
3820     {
3821         int tmp = a[s.pop()];
3822         int local = tmp * (s.isEmpty()?n:n-s.peek()-1);
3823         if(local > max) max = local;
3824     }
3825     return max;
3826 }
3827
3828 public int maximalRectangle(char[][] matrix) {
3829     int m = matrix.length;
3830     if(m == 0) return 0;
3831     int n = matrix[0].length;
3832     if(n == 0 ) return 0;
3833
3834     int[] cur = new int[n];
3835     for(int i=0;i<n;i++)
3836         cur[i] = (matrix[0][i]-0);
3837     int max = maxHis(cur);
3838
3839     for(int i=1;i<m;i++)
3840     {
3841         for(int j = 0; j<n; j++)
3842         {
3843             if(matrix[i][j] ==0) cur[j] = 0;
3844             else cur[j]++;
3845         }
3846         int local = maxHis(cur);
3847         if(local > max) max = local;
3848     }
3849 }

```

```

3848         }
3849     return max;
3850 }
3851 }
3852 }
3853 }
3854 =====
3855 /Users/linxie/Documents/github/leetcodeHub/code/MaximumDepthofBinaryTree
3856 =====
3857
3858
3859 MaximumDepthofBinaryTree
3860
3861 Given a binary tree, find its maximum depth.
3862
3863 The maximum depth is the number of nodes along the longest path from the root node down to
3864 the farthest leaf node.
3865 --
3866 /**
3867 * Definition for binary tree
3868 * public class TreeNode {
3869 *     int val;
3870 *     TreeNode left;
3871 *     TreeNode right;
3872 *     TreeNode(int x) { val = x; }
3873 * }
3874 */
3875 public class Solution {
3876     public int maxDepth(TreeNode root) {
3877
3878         if(root == null) return 0;
3879         return 1+Math.max(maxDepth(root.left), maxDepth(root.right));
3880     }
3881 }
3882 }
3883
3884
3885 =====
3886 /Users/linxie/Documents/github/leetcodeHub/code/MaximumProductSubarray
3887 =====
3888
3889
3890 MaximumProductSubarray
3891
3892 Find the contiguous subarray within an array (containing at least one number) which has
3893 the largest product.
3894
3895 For example, given the array [2,3,-2,4],
3896 the contiguous subarray [2,3] has the largest product = 6.
3897 -----
3898 public class Solution {
3899     public int maxProduct(int[] A) {
3900
3901         int n = A.length;
3902         if(n==0) return 0;
3903
3904         int[] b = new int[n];
3905         boolean neg = false, pos = false;
3906         int max_neg = 0;
3907         int min_pos = 0;
3908         int prod = 1;
3909         int max = Integer.MIN_VALUE;
3910         for(int i=0;i<n;i++)
3911         {
3912             prod*=A[i];
3913
3914             if(prod>max)
3915             {
3916                 max = prod;
3917             }
3918
3919             if(prod == 0)
3920             {
3921                 neg = false;
3922                 pos = false;
3923                 prod = 1;
3924             }
3925             else if(prod < 0)
3926             {
3927                 if(!neg || prod > max_neg)
3928                 {
3929                     if(!neg) neg = true;
3930                     max_neg = prod;
3931                 }
3932                 else
3933                 {
3934                     if(prod/max_neg > max)
3935                     {
3936                         max = prod/max_neg;
3937                     }
3938                 }
3939             }
3940             else
3941             {
3942                 if(!pos || prod < min_pos)
3943                 {
3944                     if(!pos) pos = true;
3945                     min_pos = prod;
3946                 }
3947             else
3948             {
3949                 if(prod/min_pos > max)
3950                 {
3951                     max = prod/min_pos;
3952                 }
3953             }
3954         }
3955     }
3956 }

```

```

3952         }
3953     }
3954   }
3955 }
3956 return max;
3957
3958
3959 }
3960 }
3961
3962
3963 =====
3964 /Users/linxie/Documents/github/leetcodeHub/code/MaximumSubarray
3965 =====
3966
3967
3968 MaximumSubarray
3969
3970 Find the contiguous subarray within an array (containing at least one number) which
3971 has the largest sum.
3972
3973 For example, given the array [-2,1,-3,4,-1,2,1,-5,4],
3974 the contiguous subarray [4,-1,2,1] has the largest sum = 6.
3975
3976 ---
3977
3978 public class Solution {
3979     public int maxSubArray(int[] A) {
3980         int n=A.length;
3981         if(n==0) return 0;
3982         if(n==1) return A[0];
3983
3984         int max = Integer.MIN_VALUE;
3985         int local = 0;
3986         for(int i:A)
3987         {
3988             local+=i;
3989             if(local>max) max = local;
3990             if(local<0) local = 0;
3991         }
3992         return max;
3993     }
3994 }
3995
3996
3997 =====
3998 /Users/linxie/Documents/github/leetcodeHub/code/MaxPointsonaLine
3999 =====
4000
4001
4002 MaxPointsonaLine
4003
4004 Given n points on a 2D plane, find the maximum number of points that lie on the same
4005 straight line.
4006 ---
4007 /**
4008 * Definition for a point.
4009 * class Point {
4010 *     int x;
4011 *     int y;
4012 *     Point() { x = 0; y = 0; }
4013 *     Point(int a, int b) { x = a; y = b; }
4014 * }
4015 */
4016 public class Solution {
4017     public int maxPoints(Point[] points) {
4018         int n = points.length;
4019         if(n <= 2) return n;
4020         int max = 0;
4021         HashMap<Double, Integer> map = new HashMap<Double, Integer>();
4022         for(int i=0;i<n-1;i++)
4023         {
4024             map.clear();
4025             int vertical = 0, overlap = 0;
4026             int local = 0;
4027             for(int j=i+1; j<n;j++)
4028             {
4029                 int dx = points[j].x - points[i].x;
4030                 int dy = points[j].y - points[i].y;
4031                 if(dx == 0)
4032                 {
4033                     if(dy==0) overlap++;
4034                     else
4035                     {
4036                         vertical++;
4037                         if(vertical > local) local = vertical;
4038                     }
4039                 }
4040                 else if(dy==0)
4041                 {
4042                     if(map.containsKey(0.0)) map.put(0.0,map.get(0.0)+1);
4043                     else map.put(0.0,1);
4044                     if(map.get(0.0)>local) local = map.get(0.0);
4045                 }
4046                 else
4047                 {
4048                     double k = 1.0*dy/dx;
4049                     if(map.containsKey(k)) map.put(k, map.get(k)+1);
4050                     else map.put(k,1);
4051                     if(local < map.get(k)) local = map.get(k);
4052                 }
4053             }
4054             if(local+overlap>max) max = local+overlap;
4055         }

```

```

4056         return max+1;
4057     }
4058 }
4059
4060
4061 =====
4062 /Users/linxie/Documents/github/leetcodeHub/code/MedianofTwoSortedArrays
4063 =====
4064
4065
4066 MedianofTwoSortedArrays
4067
4068 There are two sorted arrays A and B of size m and n respectively. Find the median of the
4069 two sorted arrays. The overall run time complexity should be O(log (m+n)).
4070
4071 ---
4072
4073 public class Solution {
4074     public double findKth(int k, int[] A, int[] al, int[] ar, int[] B, int bl, int br)
4075     {
4076         int alen = ar-al+1;
4077         int blen = br-bl+1;
4078         if(alen==0)
4079         {
4080             if(blen<=k) System.exit(0);
4081             return B[bl+k];
4082         }
4083         if(blen==0)
4084         {
4085             if(alen<=k) System.exit(0);
4086             return A[al+k];
4087         }
4088         if(k==0) return A[al]<=B[bl]?A[al]:B[bl];
4089
4090         int amid = k*alen/(alen+blen);
4091         int bmid = k-amid-1;
4092         amid+=al;
4093         bmid+=bl;
4094         if(A[amid]>B[bmid])
4095         {
4096             k-=bmid-bl+1;
4097             ar =amid;
4098             bl=bmid+1;
4099         }else
4100         {
4101             k-=amid-al+1;
4102             br=bmid;
4103             al = amid+1;
4104         }
4105         return findKth(k, A, al, ar, B, bl, br);
4106     }
4107
4108     public double findMedianSortedArrays(int A[], int B[]) {
4109         int m = A.length;
4110         int n = B.length;
4111         if((m+n)%2==1) return findKth((m+n)/2, A, 0, m-1, B, 0, n-1);
4112         else
4113             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;
4114     }
4115 }
4116
4117
4118 =====
4119 /Users/linxie/Documents/github/leetcodeHub/code/MergeIntervals
4120 =====
4121
4122
4123 MergeIntervals
4124
4125 Given a collection of intervals, merge all overlapping intervals.
4126
4127 For example,
4128 Given [1,3],[2,6],[8,10],[15,18],
4129 return [1,6],[8,10],[15,18].
4130
4131 ---
4132 /**
4133 * Definition for an interval.
4134 * public class Interval {
4135 *     int start;
4136 *     int end;
4137 *     Interval() { start = 0; end = 0; }
4138 *     Interval(int s, int e) { start = s; end = e; }
4139 * }
4140 */
4141 public class Solution {
4142     public static boolean check(List<Interval> intervals, int[] index, int l, int r)
4143     {
4144         return intervals.get(index[l]).start <= intervals.get(index[r]).start ;
4145     }
4146
4147     public static int partition(List<Interval> intervals, int[] index, int l, int r)
4148     {
4149         if(l>r || l<0 || r>=index.length) return -1;
4150         int p = l;
4151         int i = l-1, j = r+1;
4152         while(i<j)
4153         {
4154             while((++i)<j && check(intervals, index, i, p));
4155             while(i<=(-j) && !check(intervals, index, j, p));
4156             if(i>j) break;
4157
4158             int t = index[i];
4159             index[i] = index[j];

```

```

4160         index[j] = t;
4161     }
4162     int t = index[l];
4163     index[l] = index[j];
4164     index[j] = t;
4165     return j;
4166 }
4167
4168 public static void sort(List<Interval> intervals, int[] index, int l, int r)
4169 {
4170     if(l>=r) return;
4171
4172     int p = partition(intervals, index, l, r);
4173     sort(intervals, index, l, p-1);
4174     sort(intervals, index, p+1, r);
4175 }
4176
4177 public List<Interval> merge(List<Interval> intervals) {
4178     int n = intervals.size();
4179     if(n<=1) return intervals;
4180
4181     int[] index = new int[n];
4182     for(int i=0; i<n; i++)
4183         index[i] = i;
4184     sort(intervals, index, 0, n-1);
4185     // for(int i:index)
4186     //     System.out.print(intervals.get(i)+" ");
4187     // System.out.println();
4188     List<Interval> new_intervals = new ArrayList<Interval>();
4189
4190     Interval prev = intervals.get(index[0]);
4191     new_intervals.add(prev);
4192
4193     for(int i=1; i<n;i++)
4194     {
4195         Interval cur = intervals.get(index[i]);
4196
4197         if(cur.start > prev.end)
4198         {
4199             new_intervals.add(cur);
4200             prev = cur;
4201         }
4202         else if(cur.end > prev.end)
4203         {
4204             prev.end = cur.end;
4205         }
4206     }
4207     return new_intervals;
4208 }
4209 }
4210 }
4211 }
4212
4213 =====
4214 /Users/linxie/Documents/github/leetcodeHub/code/MergekSortedLists
4215 =====
4216 =====
4217
4218
4219 MergekSortedLists
4220
4221 Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.
4222
4223 ---
4224
4225 /**
4226 * Definition for singly-linked list.
4227 * public class ListNode {
4228 *     int val;
4229 *     ListNode next;
4230 *     ListNode(int x) {
4231 *         val = x;
4232 *         next = null;
4233 *     }
4234 * }
4235 */
4236 class node implements Comparable
4237 {
4238     ListNode nd;
4239     int val;
4240     node(ListNode _nd)
4241     {
4242         nd = _nd;
4243         val = _nd.val;
4244     }
4245
4246     public int compareTo(Object o)
4247     {
4248         return val-(node)o.val;
4249     }
4250 }
4251 public class Solution {
4252     public ListNode mergeKLists(List<ListNode> lists)
4253     {
4254         int n = lists.size();
4255         if(n==0) return null;
4256         if(n==1) return lists.get(0);
4257
4258         PriorityQueue<node> que = new PriorityQueue<node>();
4259
4260         ListNode head = null, tail =null;
4261         for(ListNode l:lists)
4262         {
4263             if(l==null) continue;

```

```

4264         que.add(new Node(1));
4265         l=l.next;
4266     }
4267     while(!que.isEmpty())
4268     {
4269         ListNode min = que.poll().nd;
4270         if(head == null)
4271         {
4272             head = min; tail = min;
4273         }
4274         else
4275         {
4276             tail.next = min;
4277             tail = min;
4278         }
4279         if(min.next!=null) que.add(new Node(min.next));
4280         min.next = null;
4281     }
4282     return head;
4283 }
4284 }
4285
4286 =====
4287 /Users/linxie/Documents/github/leetcodeHub/code/MergeSortedArray
4288 =====
4289 =====
4290
4291
4292 MergeSortedArray
4293
4294 Given two sorted integer arrays A and B, merge B into A as one sorted array.
4295
4296 Note:
4297 You may assume that A has enough space (size that is greater or equal to m + n) to hold
4298 additional elements from B. The number of elements initialized in A and B are m and n respectively.
4299
4300 ---
4301
4302 public class Solution {
4303     public void merge(int A[], int m, int B[], int n) {
4304         if(n==0) return;
4305         if(m==0)
4306         {
4307             for(int i=0;i<n;i++)
4308                 A[i] = B[i];
4309             return;
4310         }
4311         int ind = m+n-1;
4312
4313         int i = m-1, j = n-1;;
4314         while( i>=0 && j>=0)
4315         {
4316             if(A[i]>B[j]) A[ind--] = A[i--];
4317             else A[ind--] = B[j--];
4318         }
4319         while(i>=0)
4320             A[ind--] = A[i--];
4321         while(j>=0)
4322             A[ind--] = B[j--];
4323
4324     }
4325 }
4326
4327
4328 =====
4329 /Users/linxie/Documents/github/leetcodeHub/code/MergeTwoSortedLists
4330 =====
4331
4332
4333 MergeTwoSortedLists
4334
4335 Merge two sorted linked lists and return it as a new list. The new list should be made by
4336 splicing together the nodes of the first two lists.
4337
4338 ---
4339 /**
4340 * Definition for singly-linked list.
4341 * public class ListNode {
4342 *     int val;
4343 *     ListNode next;
4344 *     ListNode(int x) {
4345 *         val = x;
4346 *         next = null;
4347 *     }
4348 * }
4349 */
4350 public class Solution {
4351     public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
4352         if(l1==null) return l2;
4353         if(l2==null) return l1;
4354
4355         ListNode head = null, tail = null;
4356
4357         while(l1!=null && l2!=null)
4358         {
4359             if(l1.val<=l2.val)
4360             {
4361                 ListNode t = l1.next;
4362                 l1.next = null;
4363                 if(head == null)
4364                 {
4365                     head = l1; tail = l1;
4366                 }
4367             }

```

```

4368         {
4369             tail.next = l1;
4370             tail = l1;
4371         }
4372         l1 = t;
4373     }
4374     else
4375     {
4376         ListNode t = l2.next;
4377         l2.next = null;
4378         if(head == null)
4379         {
4380             head = l2; tail = l2;
4381         }
4382         else
4383         {
4384             tail.next = l2;
4385             tail = l2;
4386         }
4387         l2 = t;
4388     }
4389 }
4390 if(l1!=null) tail.next = l1;
4391 if(l2!=null) tail.next = l2;
4392 return head;
4393
4394
4395 }
4396 }
4397
4398 =====
4399 /Users/linxie/Documents/github/leetcodeHub/code/MinimumDepthofBinaryTree
4400 =====
4401
4402
4403
4404 MinimumDepthofBinaryTree
4405
4406 Given a binary tree, find its minimum depth.
4407
4408 The minimum depth is the number of nodes along the shortest path from the root node down
4409 to the nearest leaf node.
4410
4411 ---
4412 /**
4413 * Definition for binary tree
4414 * public class TreeNode {
4415 *     int val;
4416 *     TreeNode left;
4417 *     TreeNode right;
4418 *     TreeNode(int x) { val = x; }
4419 * }
4420 */
4421 public class Solution {
4422     public int minDepth(TreeNode root) {
4423         if(root==null) return 0;
4424         int left = minDepth(root.left);
4425         int right = minDepth(root.right);
4426         if(left == 0) return right+1;
4427         if(right == 0) return left+1;
4428         return 1+Math.min(left, right);
4429     }
4430 }
4431
4432 =====
4433 =====
4434 /Users/linxie/Documents/github/leetcodeHub/code/MinimumPathSum
4435 =====
4436
4437
4438 MinimumPathSum
4439
4440 Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right
4441 which minimizes the sum of all numbers along its path.
4442
4443 Note: You can only move either down or right at any point in time.
4444
4445 --
4446 public class Solution {
4447     public int minPathSum(int[][] grid) {
4448         int m = grid.length;
4449         if(m==0) return 0;
4450         int n = grid[0].length;
4451         if(n==0) return 0;
4452
4453         int[] re = new int[n];
4454
4455         re[0] = grid[0][0];
4456         for(int i=1; i<n; i++)
4457             re[i] = re[i-1] + grid[0][i];
4458
4459         for(int i=1; i<m; i++)
4460         {
4461             re[0] += grid[i][0];
4462             for(int j=1; j<n; j++)
4463                 re[j] = grid[i][j] + Math.min(re[j], re[j-1]);
4464         }
4465         return re[n-1];
4466     }
4467 }
4468
4469
4470 =====
4471 /Users/linxie/Documents/github/leetcodeHub/code/MinimumWindowSubstring

```

```

4472 =====
4473
4474
4475 MinimumWindowSubstring
4476
4477 Given a string S and a string T, find the minimum window in S which will contain all the
4478 characters in T in complexity O(n).
4479
4480 For example,
4481 S = "ADOBECODEBANC"
4482 T = "ABC"
4483 Minimum window is "BANC".
4484
4485 Note:
4486 If there is no such window in S that covers all characters in T, return the empty string "".
4487
4488 If there are multiple such windows, you are guaranteed that there will always be only one unique
4489 minimum window in S.
4490
4491 ---
4492
4493 public class Solution {
4494     public String minWindow(String S, String T)
4495     {
4496         int mm = S.length();
4497         int n = T.length();
4498         char[] s = S.toCharArray();
4499         char[] t = T.toCharArray();
4500
4501         if(n==0 || mm==0 || mm<n) return "";
4502
4503         HashMap<Character, Integer> map = new HashMap<Character, Integer>();
4504         int m = 0;
4505         for(char c: t)
4506             if(map.containsKey(c))
4507                 map.put(c, map.get(c)+1);
4508             else
4509             {
4510                 map.put(c, 1);
4511             }
4512
4513         m = map.size();
4514         int maxStart = 0, maxEnd = Integer.MAX_VALUE;
4515         int i = 0, j = 0;
4516
4517         for(;j<mm;j++)
4518         {
4519             if(m>0)
4520             {
4521                 if(map.containsKey(s[j]))
4522                 {
4523                     int val = map.get(s[j]);
4524                     if(val == 1) m--;
4525                     map.put(s[j], val-1);
4526                 }
4527             }
4528
4529             if(m<=0)
4530             {
4531                 for(;i<=j;i++)
4532                 {
4533
4534                     char c = s[i];
4535                     if(map.containsKey(c))
4536                     {
4537                         int val = map.get(c);
4538                         map.put(c, val+1);
4539                         if(val==0) m++;
4540                         if(m>0)
4541                         {
4542                             break;
4543                         }
4544                     }
4545                 }
4546             if(i>j) return T;
4547
4548             if(m>0 && (j-i) < maxEnd-maxStart)
4549             {
4550                 maxStart = i;
4551                 maxEnd = j;
4552             }
4553             i++;
4554         }
4555     }
4556     if(maxEnd == Integer.MAX_VALUE) return "";
4557     return S.substring(maxStart, maxEnd+1);
4558 }
4559 }
4560
4561
4562 =====
4563 /Users/linxie/Documents/github/leetcodeHub/code/MultiplyStrings
4564 =====
4565
4566
4567 MultiplyStrings
4568
4569 Given two numbers represented as strings, return multiplication of the numbers as a string.
4570
4571 Note: The numbers can be arbitrarily large and are non-negative
4572
4573 ---
4574
4575 public class Solution {

```

```

4576     public static void reverse(char[] s, int l, int r)
4577     {
4578         while(l<r)
4579         {
4580             char c = s[l];
4581             s[l] = s[r];
4582             s[r]=c;
4583             l++; r--;
4584         }
4585     }
4586
4587     public String multiply(String num1, String num2) {
4588
4589         char[] n1 = num1.toCharArray();
4590         char[] n2 = num2.toCharArray();
4591
4592         if(n1.length==0) return num2;
4593         if(n2.length==0) return num1;
4594
4595         char[] re = new char[n1.length+n2.length];
4596         reverse(n1, 0, n1.length-1);
4597         reverse(n2, 0, n2.length-1);
4598
4599         int carry = 0;
4600
4601         for(int i=0;i<re.length;i++)
4602         {
4603             int val = carry;
4604             for(int j = 0; j<=i && j<n1.length;j++)
4605             {
4606                 int k = i-j;
4607                 if( k >=0 && k<n2.length)
4608                 {
4609                     val += (int)(n1[j]-0)* (int)(n2[k]-0);
4610                 }
4611             }
4612             carry = val/10;
4613             re[i] = (char)(val%10 + 0);
4614         }
4615         if(re[re.length-1] == 0 && carry != 0 ) re[re.length-1] = (char)(0+carry);
4616
4617         int i = re.length-1;
4618         while(i>=0 && re[i] == 0) i--;
4619         if(i<0) return "0";
4620         reverse(re, 0, i);
4621         return new String(re).substring(0,i+1);
4622
4623     }
4624 }
4625
4626
4627 =====
4628 /Users/linxie/Documents/github/leetcodeHub/code/N-Queens
4629 =====
4630
4631
4632 N-Queens
4633
4634 Given an integer n, return all distinct solutions to the n-queens puzzle.
4635
4636 Each solution contains a distinct board configuration of the n-queens placement, where Q and .
4637 both indicate a queen and an empty space respectively.
4638
4639 For example,
4640 There exist two distinct solutions to the 4-queens puzzle:
4641
4642 [
4643     [".Q..", // Solution 1
4644      "...Q",
4645      "Q...",
4646      "...Q."],
4647
4648     [".Q.", // Solution 2
4649      "Q...",
4650      "...Q",
4651      "...Q."]
4652 ]---
4653
4654 public class Solution {
4655     public static boolean valid(int[] a, int k){
4656         for(int i=0;i<k;i++)
4657         {
4658             if(a[k] == a[i] || Math.abs(i-k) == Math.abs(a[i] - a[k])) return false;
4659         }
4660         return true;
4661     }
4662
4663
4664     public static void solveNQueens(int n, int k, int[] a, List<int[]> re)
4665     {
4666         if(k==n)
4667         {
4668             int[] aa = new int[n];
4669             for(int i=0;i<n;i++)
4670             {
4671                 aa[i] = a[i];
4672             }
4673             re.add(aa);
4674             return;
4675         }
4676
4677         for(int i=0;i<n;i++)
4678         {
4679             a[k]=i;

```

```

4680         if(valid(a,k))
4681             solveNQueens(n, k+1, a, re);
4682     }
4683 }
4684
4685 public static List<String[]> solveNQueens(int n)
4686 {
4687     List<String[]> re = new ArrayList<String[]>();
4688
4689     if(n==0) return re;
4690     if(n==1){
4691         String[] r_str = new String[n];
4692         r_str[0] = "Q";
4693         re.add(r_str);
4694         return re;
4695     }
4696
4697     int[] a = new int[n];
4698     List<int[]> r = new ArrayList<int[]>();
4699     solveNQueens(n,0,a,r);
4700
4701     for(int i=0; i<r.size(); i++)
4702     {
4703         String[] r_str = new String[n];
4704         for(int j=0;j<n;j++)
4705         {
4706             String rr = "";
4707             for(int k =0; k<n; k++)
4708             {
4709                 if( r.get(i)[j] == k)
4710                 {
4711                     rr+="Q";
4712                 }
4713                 else rr+=".";
4714             }
4715             r_str[j] = rr;
4716         }
4717         re.add(r_str);
4718     }
4719     return re;
4720 }
4721 }
4722
4723
4724 =====
4725 /Users/linxie/Documents/github/leetcodeHub/code/N-QueensII
4726 =====
4727
4728
4729 N-QueensII
4730
4731 Follow up for N-Queens problem.
4732
4733 Now, instead outputting board configurations, return the total number of distinct solutions.
4734
4735 ---
4736 public class Solution {
4737     public boolean valid(int n, int k, int[] re)
4738     {
4739         int r = re[k];
4740         for(int i=0;i<k;i++)
4741         {
4742             if(re[i] == r || Math.abs(re[i]-r) == k-i) return false;
4743         }
4744         return true;
4745     }
4746
4747     public int totalNQueens(int n, int k, int[] re)
4748     {
4749         if(n==k) return 1;
4750
4751         int r = 0;
4752         for(int i=0; i<n;i++)
4753         {
4754             re[k] = i;
4755             if(valid(n,k,re))
4756                 r+=totalNQueens(n,k+1,re);
4757         }
4758         return r;
4759     }
4760     public int totalNQueens(int n)
4761     {
4762         if(n<=1) return n;
4763         int[] re = new int[n];
4764         for(int i=0;i<n;i++)
4765         {
4766             re[i] = -1;
4767         }
4768
4769         return totalNQueens(n, 0, re);
4770     }
4771 }
4772
4773
4774
4775 =====
4776 /Users/linxie/Documents/github/leetcodeHub/code/NextPermutation
4777 =====
4778
4779
4780 NextPermutation
4781
4782 Implement next permutation, which rearranges numbers into the lexicographically next greater
4783 permutation of numbers.

```

```

4784
4785 If such arrangement is not possible, it must rearrange it as the lowest possible order (ie,
4786 sorted in ascending order).
4787
4788 The replacement must be in-place, do not allocate extra memory.
4789
4790 Here are some examples. Inputs are in the left-hand column and its corresponding outputs are
4791 in the right-hand column.
4792 1,2,3 → 1,3,2
4793 3,2,1 → 1,2,3
4794 1,1,5 → 1,5,1
4795 ---
4796
4797 public class Solution {
4798     public void nextPermutation(int[] num) {
4799         int n=num.length;
4800         if(n<=1) return;
4801
4802         int i = n-1;
4803         while(i>0 && num[i]<=num[i-1]) i--;
4804         if(i==0)
4805         {
4806             int l = 0, r = n-1;
4807             while(l<r)
4808             {
4809                 int t = num[l];
4810                 num[l] = num[r];
4811                 num[r] = t;
4812                 l++; r--;
4813             }
4814             return;
4815         }
4816
4817         int j=n-1;
4818         while(j>i && num[j]<=num[i-1]) j--;
4819         if(j<i) return;
4820
4821         int t = num[i-1];
4822         num[i-1] = num[j];
4823         num[j] = t;
4824         int l = i, r = n-1;
4825         while(l<r)
4826         {
4827             t = num[l];
4828             num[l] = num[r];
4829             num[r] = t;
4830             l++; r--;
4831         }
4832     }
4833 }
4834
4835
4836 =====
4837 /Users/linxie/Documents/github/leetcodeHub/code/PalindromeNumber
4838 =====
4839
4840
4841 PalindromeNumber
4842
4843 Determine whether an integer is a palindrome. Do this without extra space.
4844
4845 ---
4846
4847 public class Solution {
4848     public boolean isPalindrome(int x) {
4849
4850         if(x==0) return true;
4851         if(x==Integer.MIN_VALUE) return false;
4852         if(x<0) return false;
4853
4854         int t = x;
4855         int div = 10;
4856         while(true)
4857         {
4858             if(t/div!=0)
4859             {
4860                 if(div< Integer.MAX_VALUE/10)
4861                 {
4862                     div*=10;
4863                     continue;
4864                 }
4865             }
4866         }
4867         else div /=10;
4868         break;
4869     }
4870     int l = 1, r = div;
4871     while(l<=r){
4872         if((x/l%10) != (x/r%10))
4873         {
4874             return false;
4875         }
4876         l*=10;
4877         r/=10;
4878     }
4879     return true;
4880
4881
4882 }
4883 }
4884
4885 ----
4886 public class Solution {
4887     public boolean isPalindrome(int x) {

```

```

4888
4889     if(x<0) return false;
4890
4891     int div = 1;
4892     while(x/div>=10)
4893     {
4894         div*=10;
4895     }
4896
4897     while(x!=0)
4898     {
4899         int l = x/div;
4900         int r = x%10;
4901         if(l!=r) return false;
4902
4903         x = (x%div)/10;
4904         div /=100;
4905     }
4906     return true;
4907 }
4908 }
4909
4910
4911 =====
4912 /Users/linxie/Documents/github/leetcodeHub/code/PalindromePartitioning
4913 =====
4914
4915
4916 PalindromePartitioning
4917
4918 Given a string s, partition s such that every substring of the partition is a palindrome.
4919
4920 Return all possible palindrome partitioning of s.
4921
4922 For example, given s = "aab",
4923 Return
4924
4925 [
4926     ["aa", "b"],
4927     ["a", "a", "b"]
4928 ]
4929 ---
4930 public class Solution {
4931     public static boolean valid(String s)
4932     {
4933         int n = s.length();
4934         if(n<=1) return true;
4935         if(s.charAt(0) == s.charAt(n-1)) return valid(s.substring(1,n-1));
4936         return false;
4937     }
4938
4939     public static List<List<String>> partition(String s, HashMap<Integer, List<List<String>>> map)
4940     {
4941         List<List<String>> re = new ArrayList<List<String>>();
4942         int n = s.length();
4943         if(n==0) return re;
4944         if(map.containsKey(n)) return map.get(n);
4945
4946         for(int i=n-1;i>=0;i--)
4947         {
4948             if(valid(s.substring(i)))
4949             {
4950                 List<List<String>> r = partition(s.substring(0,i),map);
4951                 if(r.size() == 0)
4952                 {
4953                     if(i==0)
4954                     {
4955                         List<String> rr = new ArrayList<String>();
4956                         rr.add(s);
4957                         re.add(rr);
4958                         continue;
4959                     }
4960                 }
4961                 else
4962                 {
4963                     for(List<String>l:r)
4964                     {
4965                         List<String> rr = new ArrayList<String>(l);
4966                         rr.add(s.substring(i));
4967                         re.add(rr);
4968                     }
4969                 }
4970             }
4971         }
4972         map.put(n,re);
4973         return re;
4974     }
4975
4976     public static List<List<String>> partition(String s) {
4977         return partition(s, new HashMap<Integer, List<List<String>>>());
4978     }
4979 }
4980
4981
4982 =====
4983 /Users/linxie/Documents/github/leetcodeHub/code/PalindromePartitioningII
4984 =====
4985
4986
4987 PalindromePartitioningII
4988
4989 Given a string s, partition s such that every substring of the partition is a palindrome.
4990
4991 Return the minimum cuts needed for a palindrome partitioning of s.

```

```

4992
4993 For example, given s = "aab",
4994 Return 1 since the palindrome partitioning ["aa", "b"] could be produced using 1 cut.
4995 ---
4996 public class Solution {
4997     public boolean valid(char[] c, int l, int r, int n, int[] m)
4998     {
4999         int k = l+n+r;
5000         if(m[k]==-1) return m[k] == 1;
5001
5002         if(l>=r) return true;
5003
5004         if(c[l] == c[r] && valid(c, l+1, r-1, n, m))
5005         {
5006             m[k] = 1;
5007         }
5008         else
5009         {
5010             m[k] = 0;
5011         }
5012         return m[k]==1;
5013     }
5014
5015     public int minCut(char[] s, int k, int n, int[] cuts, int[] m)
5016     {
5017         if(cuts[k]==-1) return cuts[k];
5018
5019         int min = Integer.MAX_VALUE;
5020         for(int i=k;i>0;i--)
5021         {
5022             if(valid(s, i, k, n, m))
5023             {
5024                 int prev = minCut(s, i-1, n, cuts, m);
5025
5026                 if(prev == -2) continue;
5027                 if(prev!=-1 && prev<min) min = prev;
5028             }
5029         }
5030         if(min!=Integer.MAX_VALUE)
5031         {
5032             cuts[k] = min+1;
5033         }
5034         else    cuts[k] = -2;
5035
5036         return cuts[k];
5037     }
5038
5039     public int minCut(String s)
5040     {
5041         int n = s.length();
5042         if(n<=1) return 0;
5043         int[] cuts = new int[n];
5044         for(int i=0;i<n;i++)
5045             cuts[i] = -1;
5046         int[] m = new int[n*n];
5047         for(int i=0;i<n*n;i++)
5048             m[i] = -1;
5049         for(int i =0;i<n;i++)
5050         {
5051             if( valid(s.toCharArray(),0, i, n, m))
5052             {
5053
5054                 cuts[i] = 0;
5055             }
5056         }
5057
5058         return minCut(s.toCharArray(),n-1, n, cuts, m);
5059     }
5060 }
5061
5062
5063 =====
5064 /Users/linxie/Documents/github/leetcodeHub/code/PartitionList
5065 =====
5066
5067
5068 PartitionList
5069
5070 Given a linked list and a value x, partition it such that all nodes less than x come
5071 before nodes greater than or equal to x.
5072
5073 You should preserve the original relative order of the nodes in each of the two partitions.
5074
5075 For example,
5076 Given 1->4->3->2->5->2 and x = 3,
5077 return 1->2->2->4->3->5.
5078 --
5079 /**
5080 * Definition for singly-linked list.
5081 * public class ListNode {
5082 *     int val;
5083 *     ListNode next;
5084 *     ListNode(int x) {
5085 *         val = x;
5086 *         next = null;
5087 *     }
5088 * }
5089 */
5090 public class Solution {
5091     public ListNode partition(ListNode head, int x) {
5092
5093         if(head == null) return null;
5094         ListNode less = null, lTail = null, rTail = null;
5095         ListNode cur = head;

```

```

5096     head = null;
5097
5098     while(cur!=null)
5099     {
5100         ListNode t = cur.next;
5101         cur.next = null;
5102         if(cur.val < x)
5103         {
5104             if(lTail == null)
5105             {
5106                 less = cur;
5107                 lTail = cur;
5108             }
5109             else
5110             {
5111                 lTail.next = cur;
5112                 lTail = cur;
5113             }
5114         }
5115     }
5116     if(rTail == null)
5117     {
5118         head = cur;
5119         rTail = cur;
5120     }
5121     else
5122     {
5123         rTail.next = cur;
5124         rTail = cur;
5125     }
5126 }
5127     cur = t;
5128 }
5129 if(lTail == null) return head;
5130 lTail.next = head;
5131 return less;
5132 }
5133 }
5134 }
5135
5136
5137 =====
5138 /Users/linxie/Documents/github/leetcodeHub/code/PascalsTriangle
5139 =====
5140
5141
5142 PascalsTriangle
5143
5144 Given numRows, generate the first numRows of Pascals triangle.
5145
5146 For example, given numRows = 5,
5147 Return
5148
5149 [
5150     [1],
5151     [1,1],
5152     [1,2,1],
5153     [1,3,3,1],
5154     [1,4,6,4,1]
5155 ]
5156 ---
5157 public class Solution {
5158     public List<List<Integer>> generate(int numRows) {
5159
5160         List<List<Integer>> re = new ArrayList<List<Integer>>();
5161
5162         if(numRows == 0) return re;
5163         List<Integer> r = new ArrayList<Integer>();
5164         r.add(1);
5165         re.add(r);
5166         if(numRows == 1) return re;
5167
5168         for(int i=1;i<numRows;i++)
5169         {
5170             List<Integer> rr = re.get(i-1);
5171             r = new ArrayList<Integer>();
5172             r.add(1);
5173             for(int j=0;j<rr.size()-1;j++)
5174             {
5175                 r.add(rr.get(j) + rr.get(j+1));
5176             }
5177             r.add(1);
5178             re.add(r);
5179         }
5180         return re;
5181     }
5182 }
5183
5184
5185 =====
5186 /Users/linxie/Documents/github/leetcodeHub/code/PascalsTriangleII
5187 =====
5188
5189
5190 PascalsTriangleII
5191
5192 Given an index k, return the kth row of the Pascals triangle.
5193
5194 For example, given k = 3,
5195 Return [1,3,3,1].
5196
5197 Note:
5198 Could you optimize your algorithm to use only O(k) extra space
5199 ---

```

```

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         }
5225     }
5226 }
5227 }
5228
5229 =====
5230 /Users/linxie/Documents/github/leetcodeHub/code/PathSum
5231 =====
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
5243         5
5244        / \
5245       4   8
5246      /   / \
5247     11  13  4
5248    / \   \
5249   7   2   1
5250 return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.
5251 ---
5252 /**
5253 * Definition for binary tree
5254 * public class TreeNode {
5255 *     int val;
5256 *     TreeNode left;
5257 *     TreeNode right;
5258 *     TreeNode(int x) { val = x; }
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 /Users/linxie/Documents/github/leetcodeHub/code/Permutations
5271 =====
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];

```

```

5304             num[j] = t;
5305             i++; j--;
5306         }
5307     }
5308 }
5309 }
5310 }
5311 }
5312 return false;
5313 }
5314
5315 public List<List<Integer>> permute(int[] num) {
5316     List<List<Integer>> re = new ArrayList<List<Integer>>();
5317     int n = num.length;
5318     if(n==0) return re;
5319     List<Integer> r = new ArrayList<Integer>();
5320     if(n==1)
5321     {
5322         r.add(num[0]);
5323         re.add(r);
5324         return re;
5325     }
5326     Arrays.sort(num);
5327     for(int i:num)
5328     {
5329         r.add(i);
5330     }
5331     re.add(r);
5332     while(nextPerm(num))
5333     {
5334         r = new ArrayList<Integer>();
5335         for(int i:num)
5336         {
5337             r.add(i);
5338         }
5339         re.add(r);
5340     }
5341     return re;
5342 }
5343 }
5344
5345
5346 =====
5347 /Users/linxie/Documents/github/leetcodeHub/code/PermutationSequence
5348 =====
5349
5350
5351 PermutationSequence
5352
5353 The set [1,2,3,...,n] contains a total of  $n!$  unique permutations.
5354
5355 By listing and labeling all of the permutations in order,
5356 we get the following sequence (ie, for  $n = 3$ ):
5357
5358 "123"
5359 "132"
5360 "213"
5361 "231"
5362 "312"
5363 "321"
5364 Given n and k, return the kth permutation sequence.
5365
5366 ---
5367
5368 public class Solution {
5369
5370     public static boolean nextPerm(int[] a)
5371     {
5372         int n = a.length;
5373         if(n<=1) return false;
5374
5375         int i=n-1, j=n-1;
5376         for(; i>0 && a[i]<=a[i-1];i--);
5377         if(i==0) return false;
5378
5379         for(; j>=i && a[j]<=a[i-1];j--);
5380         if(j==i-1) return false;
5381
5382         int t = a[i-1]; a[i-1] = a[j]; a[j] = t;
5383
5384         for(int l=i, r=n-1; l<r; l++, r--)
5385         {
5386             t = a[l]; a[l] = a[r]; a[r] = t;
5387         }
5388         return true;
5389     }
5390
5391     public static String getPermutation(int n, int k) {
5392         if(n==0) return "";
5393
5394         int[] a= new int[n];
5395         for(int i=0;i<n;i++)
5396         {
5397             a[i] = i+1;
5398         }
5399         int count =0;
5400         for(int i=1; i<k; i++)
5401         {
5402             count++;
5403             if(!nextPerm(a)) return getPermutation(n, k%count);
5404         }
5405         char[] re = new char[n];
5406         for(int i=0;i<n;i++)
5407         {

```

```

5408         re[i] = (char)(a[i]+0);
5409     }
5410     return new String(re);
5411 }
5412 }
5413
5414
5415 =====
5416 /Users/linxie/Documents/github/leetcodeHub/code/PermutationsII
5417 =====
5418
5419
5420 PermutationsII
5421
5422 Given a collection of numbers that might contain duplicates, return all possible unique permutations.
5423
5424 For example,
5425 [1,1,2] have the following unique permutations:
5426 [1,1,2], [1,2,1], and [2,1,1].
5427
5428 ---
5429
5430 public class Solution {
5431     public boolean nextPerm(int[] num)
5432     {
5433         int n = num.length;
5434         if(n==0) return false;
5435
5436         int j=n-1;
5437         for(;j>0;j--)
5438             if(num[j-1] < num[j]) break;
5439         if(j==0) return false;
5440
5441         j--;
5442         int i=n-1;
5443         for(;i>=j+1;i--)
5444             if(num[i]>num[j])
5445                 break;
5446         int t = num[j]; num[j] = num[i]; num[i] = t;
5447         j++; i = n-1;
5448         while(j<i)
5449         {
5450             t = num[j]; num[j] = num[i]; num[i] = t;
5451             i--; j++;
5452         }
5453         return true;
5454     }
5455
5456     public List<List<Integer>> permuteUnique(int[] num) {
5457         List<List<Integer>> re = new ArrayList<List<Integer>>();
5458         int n = num.length;
5459         if(n==0) return re;
5460
5461         Arrays.sort(num);
5462         List<Integer> r = new ArrayList<Integer>();
5463         for(int i:num)
5464             r.add(i);
5465         re.add(r);
5466         while(nextPerm(num))
5467         {
5468             r = new ArrayList<Integer>();
5469             for(int i:num)
5470                 r.add(i);
5471             re.add(r);
5472         }
5473         return re;
5474     }
5475 }
5476
5477
5478 =====
5479 /Users/linxie/Documents/github/leetcodeHub/code/PlusOne
5480 =====
5481
5482
5483 PlusOne
5484
5485 Given a non-negative number represented as an array of digits, plus one to the number.
5486
5487 The digits are stored such that the most significant digit is at the head of the list.
5488 --
5489 public class Solution {
5490     public int[] plusOne(int[] digits) {
5491         int n = digits.length;
5492         if(n==0) return digits;
5493
5494         int carry = 1;
5495         for(int i=n-1;i>=0;i--)
5496         {
5497             int re = carry+digits[i];
5498             carry = re/10;
5499             digits[i] = re%10;
5500         }
5501         int[] r;
5502         if(carry == 1)
5503         {
5504             r = new int[n+1];
5505             r[0] = carry;
5506             for(int i=0;i<n;i++)
5507                 r[i+1] = digits[i];
5508         }
5509         else
5510         {
5511             r = new int[n];

```

```

5512         for(int i=0;i<n;i++)
5513             r[i] = digits[i];
5514     }
5515     return r;
5516 }
5517 }
5518
5519 =====
5520 /Users/linxie/Documents/github/leetcodeHub/code/PopulatingNextRightPointersinEachNode
5521 =====
5522 =====
5523
5524
5525 PopulatingNextRightPointersinEachNode
5526
5527 Given a binary tree
5528
5529     struct TreeLinkNode {
5530         TreeLinkNode *left;
5531         TreeLinkNode *right;
5532         TreeLinkNode *next;
5533     }
5534 Populate each next pointer to point to its next right node. If there is no next right node,
5535 the next pointer should be set to NULL.
5536
5537 Initially, all next pointers are set to NULL.
5538
5539 Note:
5540
5541 You may only use constant extra space.
5542 You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and
5543 every parent has two children).
5544 For example,
5545 Given the following perfect binary tree,
5546
5547     1
5548    / \
5549   2   3
5550  / \ / \
5551 4 5 6 7
5552 After calling your function, the tree should look like:
5553
5554     1 -> NULL
5555    / \
5556   2 -> 3 -> NULL
5557  / \ / \
5558 4->5->6->7 -> NULL
5559 ---
5560 /**
5561 * Definition for binary tree with next pointer.
5562 * public class TreeLinkNode {
5563 *     int val;
5564 *     TreeLinkNode left, right, next;
5565 *     TreeLinkNode(int x) { val = x; }
5566 */
5567 public class Solution {
5568
5569     public static void connect(TreeLinkNode root, TreeLinkNode par)
5570     {
5571         if(root == null) return;
5572         if(par == null) { root.next = null; return; }
5573         if(root == par.left && par.right!=null) { root.next = par.right; connect(root.next, par); return; }
5574         par = par.next;
5575         while(par!=null)
5576         {
5577             if(par.left!=null)
5578             {
5579                 root.next = par.left;
5580                 connect(root.next, par);
5581                 break;
5582             }
5583
5584             else if(par.right!=null)
5585             {
5586                 root.next = par.right;
5587                 connect(root.next, par);
5588                 break;
5589             }
5590             else par = par.next;
5591         }
5592     }
5593
5594     public static void connect(TreeLinkNode root) {
5595         TreeLinkNode par =null;
5596         while(root!=null)
5597         {
5598             connect(root, par);
5599             while(root!=null)
5600             {
5601                 if(root.left!=null)
5602                 {
5603                     par = root;
5604                     root = root.left;
5605                     break;
5606                 }
5607                 else if(root.right != null)
5608                 {
5609                     par = root;
5610                     root = root.right;
5611                     break;
5612                 }
5613                 else
5614                 {
5615                     root = root.next;
5616                 }
5617             }
5618         }
5619     }
5620 }

```

```

5616         }
5617     }
5618 }
5619 }
5620
5621
5622 =====
5623 /Users/linxie/Documents/github/leetcodeHub/code/PopulatingNextRightPointersinEachNodeII
5624 =====
5625
5626
5627 PopulatingNextRightPointersinEachNodeII
5628
5629 Follow up for problem "Populating Next Right Pointers in Each Node".
5630
5631 What if the given tree could be any binary tree? Would your previous solution still work?
5632
5633 Note:
5634
5635 You may only use constant extra space.
5636 For example,
5637 Given the following binary tree,
5638
5639     1
5640     / \
5641     2   3
5642     / \   \
5643     4   5   7
5644 After calling your function, the tree should look like:
5645
5646     1 -> NULL
5647     / \
5648     2 -> 3 -> NULL
5649     / \
5650     4-> 5 -> 7 -> NULL
5651 ---
5652 /**
5653 * Definition for binary tree with next pointer.
5654 * public class TreeLinkNode {
5655 *     int val;
5656 *     TreeLinkNode left, right, next;
5657 *     TreeLinkNode(int x) { val = x; }
5658 * }
5659
5660 public class Solution {
5661
5662     public static void connect(TreeLinkNode root, TreeLinkNode par)
5663     {
5664         if(root == null) return;
5665         if(par == null) { root.next = null; return; }
5666         if(root == par.left && par.right!=null) { root.next = par.right; connect(root.next, par); return; }
5667         par = par.next;
5668         while(par!=null)
5669         {
5670             if(par.left!=null)
5671             {
5672                 root.next = par.left;
5673                 connect(root.next, par);
5674                 break;
5675             }
5676             else if(par.right!=null)
5677             {
5678                 root.next = par.right;
5679                 connect(root.next, par);
5680                 break;
5681             }
5682             else par = par.next;
5683         }
5684     }
5685
5686     public static void connect(TreeLinkNode root) {
5687         TreeLinkNode par =null;
5688         while(root!=null)
5689         {
5690             connect(root, par);
5691             while(root!=null)
5692             {
5693                 if(root.left!=null)
5694                 {
5695                     par = root;
5696                     root = root.left;
5697                     break;
5698                 }
5699                 else if(root.right != null)
5700                 {
5701                     par = root;
5702                     root = root.right;
5703                     break;
5704                 }
5705                 else
5706                 {
5707                     root = root.next;
5708                 }
5709             }
5710         }
5711
5712
5713 =====
5714 /Users/linxie/Documents/github/leetcodeHub/code/Pow(x,n)
5715 =====
5716
5717
5718 Pow(x,n)
5719

```

```

5720 ---
5721 public class Solution {
5722     public double pow(double x, int n) {
5723         if(x==0 || x==1 || n==1) return x;
5724         if(n==0) return 1.0;
5725         if(n==Integer.MIN_VALUE)
5726         {
5727             n++;
5728             n=-n;
5729             double ret = pow(x,n/2);
5730             if(n%2==1)
5731             {
5732                 return 1/(ret*ret*x*x);
5733             }else
5734             {
5735                 return 1/(ret*ret*x);
5736             }
5737         }
5738     }
5739 }
5740
5741     boolean flag = false;
5742     if(n<0)
5743     {
5744         flag = true;
5745         n=-n;
5746     }
5747     double ret = pow(x,n/2);
5748     ret*=ret;
5749     if(n%2==1)
5750     {
5751         ret *= x;
5752     }
5753     if(flag)
5754         ret = 1/ret;
5755     return ret;
5756 }
5757
5758 }
5759 }
5760
5761 =====
5762 /Users/linxie/Documents/github/leetcodeHub/code/RecoverBinarySearchTree
5763 =====
5764 =====
5765
5766
5767 RecoverBinarySearchTree
5768
5769 Two elements of a binary search tree (BST) are swapped by mistake.
5770
5771 Recover the tree without changing its structure.
5772
5773 ---
5774 /**
5775 * Definition for binary tree
5776 * public class TreeNode {
5777 *     int val;
5778 *     TreeNode left;
5779 *     TreeNode right;
5780 *     TreeNode(int x) { val = x; }
5781 * }
5782 */
5783 public class Solution {
5784     public void recoverTree(TreeNode root) {
5785
5786         if(root == null) return;
5787
5788         TreeNode first = null, second = null;
5789
5790         Stack<TreeNode> s = new Stack<TreeNode>();
5791         TreeNode cur = root, prev = null;
5792         while(!s.isEmpty() || cur!=null)
5793         {
5794
5795             if(cur!=null)
5796             {
5797                 s.push(cur);
5798                 cur = cur.left;
5799             }
5800             else
5801             {
5802                 cur = s.pop();
5803                 if(prev!=null && prev.val > cur.val)
5804                 {
5805                     if(first ==null)
5806                     {
5807                         first = prev;
5808                         second = cur;
5809                     }
5810                     else
5811                     {
5812                         second = cur;
5813                         break;
5814                     }
5815                 }
5816                 prev = cur;
5817                 cur = cur.right;
5818             }
5819         }
5820         if(first!=null && second!=null)
5821         {
5822             int t = first.val;
5823             first.val = second.val;

```

```

5824         second.val = t;
5825         // System.out.println(first.val+", "+second.val);
5826     }
5827     return;
5828 }
5829 }
5830 }
5831
5832 =====
5833 /Users/linxie/Documents/github/leetcodeHub/code/RegularExpressionMatching
5834 =====
5835 =====
5836
5837
5838 RegularExpressionMatching
5839
5840 Implement regular expression matching with support for . and *.
5841
5842 . Matches any single character.
5843 * Matches zero or more of the preceding element.
5844
5845 The matching should cover the entire input string (not partial).
5846
5847 The function prototype should be:
5848 bool isMatch(const char *s, const char *p)
5849
5850 Some examples:
5851 isMatch("aa", "a") → false
5852 isMatch("aa", "aa") → true
5853 isMatch("aaa", "aa") → false
5854 isMatch("aa", "a*") → true
5855 isMatch("aa", ".*") → true
5856 isMatch("ab", ".*") → true
5857 isMatch("aab", "c*a*b") → true
5858
5859 ---
5860
5861 public class Solution {
5862     public boolean isMatch(String s, String p) {
5863         if(p.length()==0) return s.length()==0;
5864
5865         int n = p.length();
5866
5867         if(n>1 && p.charAt(1) == *)
5868         {
5869             while( s.length()!=0 && (s.charAt(0) == p.charAt(0) || p.charAt(0)==.))
5870             {
5871                 if(isMatch(s,p.substring(2))) return true;
5872                 s=s.substring(1);
5873             }
5874             return isMatch(s,p.substring(2));
5875         }
5876         else
5877         {
5878             if(s.length()!=0 && (s.charAt(0) == p.charAt(0) || p.charAt(0)==.))
5879                 return isMatch(s.substring(1),p.substring(1));
5880             return false;
5881         }
5882     }
5883 }
5884
5885 =====
5886 /Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedArray
5887 =====
5888 =====
5889
5890
5891 RemoveDuplicatesfromSortedArray
5892
5893 Given a sorted array, remove the duplicates in place such that each element appear
5894 only once and return the new length.
5895
5896 Do not allocate extra space for another array, you must do this in place with constant memory.
5897
5898 For example,
5899 Given input array A = [1,1,2],
5900
5901 Your function should return length = 2, and A is now [1,2].
5902 ---
5903
5904 public class Solution {
5905     public int removeDuplicates(int[] A) {
5906         int n = A.length;
5907         if(n<=1) return n;
5908
5909         int i = 1, j=i;
5910         while(j<n)
5911         {
5912             if(A[j]!=A[j-1])
5913             {
5914                 A[i++] = A[j];
5915             }
5916             j++;
5917         }
5918         return i;
5919     }
5920 }
5921
5922 =====
5923 /Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedArrayII
5924 =====
5925 =====
5926

```

```

5927
5928 RemoveDuplicatesfromSortedArrayII
5929
5930 Follow up for "Remove Duplicates":
5931 What if duplicates are allowed at most twice?
5932
5933 For example,
5934 Given sorted array A = [1,1,1,2,2,3],
5935
5936 Your function should return length = 5, and A is now [1,1,2,2,3].
5937
5938 ---
5939
5940 public class Solution {
5941     public int removeDuplicates(int[] A) {
5942
5943     int n = A.length;
5944     if(n<=2) return n;
5945
5946     int i=1, j=1;
5947     int count =1;
5948     int val = A[0];
5949
5950     while(i<n)
5951     {
5952         if(A[i] == A[i-1])
5953         {
5954             count++;
5955             if(count>2)
5956             {
5957                 break;
5958             }
5959         }
5960         else break;
5961         i++;
5962     }
5963
5964     j=i;
5965     while(j<n)
5966     {
5967         if(A[j]!=val){ count =1; val = A[j]; A[i++] = A[j];}
5968         else
5969         {
5970             count++;
5971             if(count<=2)
5972             {
5973                 A[i++]=A[j];
5974             }
5975         }
5976         j++;
5977     }
5978     return i;
5979
5980 }
5981 }
5982
5983
5984 =====
5985 /Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedList
5986 =====
5987
5988
5989 RemoveDuplicatesfromSortedList
5990
5991 Given a sorted linked list, delete all duplicates such that each element appear only once.
5992
5993 For example,
5994 Given 1->1->2, return 1->2.
5995 Given 1->1->2->3->3, return 1->2->3.
5996
5997 ---
5998 /**
5999 * Definition for singly-linked list.
6000 * public class ListNode {
6001 *     int val;
6002 *     ListNode next;
6003 *     ListNode(int x) {
6004 *         val = x;
6005 *         next = null;
6006 *     }
6007 * }
6008 */
6009 public class Solution {
6010     public ListNode deleteDuplicates(ListNode head) {
6011         if(head == null || head.next ==null) return head;
6012
6013         ListNode newHead = head, newTail = head;
6014         int tmp = head.val;
6015         head = head.next;
6016         newHead.next = null;
6017
6018         while(head!=null)
6019         {
6020             if(head.val != tmp)
6021             {
6022                 tmp = head.val;
6023                 newTail.next = head;
6024                 newTail = head;
6025                 head = head.next;
6026                 newTail.next = null;
6027             }
6028             {
6029                 ListNode t = head;
6030                 head = head.next;

```

```

6031             t.next = null;
6032         }
6033     }
6034     return newHead;
6035 }
6036 }
6037
6038 =====
6039 /Users/linxie/Documents/github/leetcodeHub/code/RemoveDuplicatesfromSortedListII
6040 =====
6041
6042
6043
6044 RemoveDuplicatesfromSortedListII
6045
6046 Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only
6047 distinct numbers from the original list.
6048
6049 For example,
6050 Given 1->2->3->3->4->4->5, return 1->2->5.
6051 Given 1->1->1->2->3, return 2->3.
6052 ---
6053 /**
6054 * Definition for singly-linked list.
6055 */
6056 public class ListNode {
6057     int val;
6058     ListNode next;
6059     ListNode(int x) {
6060         val = x;
6061         next = null;
6062     }
6063 }
6064 public class Solution {
6065     public ListNode remove(ListNode cur)
6066     {
6067         if(cur == null || cur.next == null) return cur;
6068         int val = cur.val;
6069         while(cur!=null)
6070         {
6071             ListNode t = cur.next;
6072             if(cur.val == val)
6073             {
6074                 cur.next = null;
6075                 cur = t;
6076             }else
6077             {
6078                 return cur;
6079             }
6080         }
6081         return null;
6082     }
6083     public ListNode deleteDuplicates(ListNode head) {
6084         if(head == null) return head;
6085
6086         ListNode cur = head;
6087         head = null;
6088         ListNode tail = null;
6089
6090         while(cur!=null)
6091         {
6092             if(cur.next == null || cur.val != cur.next.val)
6093             {
6094                 ListNode t = cur.next;
6095                 cur.next = null;
6096                 if(tail == null)
6097                 {
6098                     head = cur;
6099                     tail = cur;
6100                 }
6101                 else
6102                 {
6103                     tail.next = cur;
6104                     tail = cur;
6105                 }
6106                 cur = t;
6107             }else
6108             {
6109                 cur = remove(cur);
6110             }
6111         }
6112         return head;
6113     }
6114 }
6115
6116 =====
6117 /Users/linxie/Documents/github/leetcodeHub/code/RemoveElement
6118 =====
6119
6120
6121
6122 RemoveElement
6123
6124 Given an array and a value, remove all instances of that value in place and return
6125 the new length.
6126
6127 The order of elements can be changed. It doesnt matter what you leave beyond the
6128 new length.
6129
6130 ---
6131 public class Solution {
6132     public int removeElement(int[] A, int elem) {
6133         int n=A.length;

```

```

6135     if(n==0) return 0;
6136     int i=0, j=0;
6137     while(i<n && A[i]!=elem) i++;
6138     if(i==n) return n;
6139
6140     j = i;
6141     for(;j<n;j++)
6142     {
6143         if(A[j]==elem) continue;
6144         A[i++] = A[j];
6145     }
6146     return i;
6147
6148 }
6149 }
6150
6151
6152 =====
6153 /Users/linxie/Documents/github/leetcodeHub/code/RemoveNthNodeFromEndofList
6154 =====
6155
6156
6157 RemoveNthNodeFromEndofList
6158
6159 Given a linked list, remove the nth node from the end of list and return its head.
6160
6161 For example,
6162
6163     Given linked list: 1->2->3->4->5, and n = 2.
6164
6165     After removing the second node from the end, the linked list becomes 1->2->3->5.
6166 ---
6167
6168 /**
6169 * Definition for singly-linked list.
6170 * public class ListNode {
6171 *     int val;
6172 *     ListNode next;
6173 *     ListNode(int x) {
6174 *         val = x;
6175 *         next = null;
6176 *     }
6177 * }
6178 */
6179 public class Solution {
6180     public ListNode removeNthFromEnd(ListNode head, int n) {
6181
6182         if(head == null) return head;
6183         if(head.next == null)
6184         {
6185             if(n==1) return null;
6186             else return head;
6187         }
6188         int count = 0;
6189         ListNode cur = head, second = head, prev = null;
6190
6191         while(cur.next !=null)
6192         {
6193             count++;
6194             cur = cur.next;
6195             if(count>=n)
6196             {
6197                 prev = second;
6198                 second = second.next;
6199             }
6200         }
6201         if(count <n-1) return head;
6202
6203         if(prev == null)
6204         {
6205             head = second.next;
6206             second.next = null;
6207         }
6208         else
6209         {
6210             prev.next = second.next;
6211             second.next = null;
6212         }
6213         return head;
6214
6215     }
6216 }
6217
6218
6219 =====
6220 /Users/linxie/Documents/github/leetcodeHub/code/ReorderList
6221 =====
6222
6223
6224 ReorderList
6225
6226 Given a singly linked list L: L0>L1>...>Ln-1>Ln,
6227 reorder it to: L0>Ln>L1>Ln-1>L2>Ln-2>...
6228
6229 You must do this in-place without altering the nodes values.
6230
6231 For example,
6232 Given {1,2,3,4}, reorder it to {1,4,2,3}.
6233
6234 ---
6235 /**
6236 * Definition for singly-linked list.
6237 * class ListNode {
6238 *     int val;

```

```

6239 *     ListNode next;
6240 *     ListNode(int x) {
6241 *         val = x;
6242 *         next = null;
6243 *     }
6244 */
6245 */
6246 public class Solution {
6247     public static ListNode reverse(ListNode head)
6248     {
6249         ListNode cur = head;
6250         head = null;
6251         while(cur!=null)
6252         {
6253             ListNode t = cur.next;
6254             cur.next = head;
6255             head = cur;
6256             cur = t;
6257         }
6258         return head;
6259     }
6260
6261     public static ListNode merge(ListNode n1, ListNode n2)
6262     {
6263         if(n1==null) return n2;
6264         if(n2 == null) return n1;
6265
6266         ListNode head = null, tail = null;
6267         while(n1 != null && n2 !=null)
6268         {
6269             if(head == null)
6270             {
6271                 head = n1;
6272                 tail = n1;
6273                 n1 = tail.next;
6274                 tail.next = null;
6275             }
6276             else
6277             {
6278                 tail.next = n1;
6279                 tail = n1;
6280                 n1 = n1.next;
6281                 tail.next = null;
6282             }
6283             tail.next = n2;
6284             tail = n2;
6285             n2 = n2.next;
6286             tail.next = null;
6287         }
6288         if(n1!=null && tail!=null) tail.next = n1;
6289         return head;
6290     }
6291
6292     public static void reorderList(ListNode head) {
6293         if(head == null || head.next == null) return;
6294
6295         ListNode left = head, right = head;
6296         int count=0;
6297         while(right.next != null)
6298         {
6299             count++;
6300             right = right.next;
6301             if(count%2==0) left = left.next;
6302         }
6303         right = left.next;
6304         left.next = null;
6305         head = merge(head, reverse(right));
6306     }
6307 }
6308
6309
6310 =====
6311 /Users/linxie/Documents/github/leetcodeHub/code/ReplaceStringWithPattern
6312 =====
6313
6314
6315 Replace all occurrence of the given pattern to 'X'.
6316 For example, given that the pattern="abc", replace
6317 "abcdefdfegabcabc" with "XdefdfegX". Note that multiple
6318 occurrences of abc's that are contiguous will be replaced
6319 with only one 'X'
6320
6321 ===
6322
6323 class Solution
6324 {
6325     public boolean validate(String s, String p)
6326     {
6327         int m = s.length(), n = p.length();
6328         if(m<n) return false;
6329
6330         int i=0;
6331         for(char c: p.toCharArray())
6332         {
6333             if(c!=s.charAt(i++)) return false;
6334         }
6335         return true;
6336     }
6337
6338     public String replace(String s, String p)
6339     {
6340         int m = s.length(), n = p.length();
6341         if(m<n) return s;
6342

```

```

6343     char[ ] a = s.toCharArray(), b = p.toCharArray();
6344
6345     int i=0, j=0;
6346
6347     while(j<m)
6348     {
6349         if(j+n<=m && validate(s.substring(j,j+n), p) )
6350         {
6351             a[i++] = x;
6352             j+=n;
6353         }
6354         else
6355             a[i++] = a[j++];
6356     }
6357     return new String(a).substring(0, i);
6358 }
6359
6360 }
6361
6362 =====
6363 /Users/linxie/Documents/github/leetcodeHub/code/RestoreIPAddresses
6364 =====
6365 =====
6366
6367
6368 RestoreIPAddresses
6369
6370 Given a string containing only digits, restore it by returning all possible valid
6371 IP address combinations.
6372
6373 For example:
6374 Given "25525511135",
6375
6376 return [ "255.255.11.135", "255.255.111.35" ]. (Order does not matter)
6377
6378 ---
6379
6380 public class Solution {
6381     public static boolean valid(String s)
6382     {
6383         int n = s.length();
6384         if(n==0) return false;
6385         try{
6386             int re = Integer.parseInt(s);
6387             if(n>3 || (n !=1 && s.charAt(0)==0) || re>255 || re<0) return false;
6388             return true;
6389         }
6390         catch(Exception o)
6391         {
6392             return false;
6393         }
6394     }
6395
6396     public static List<String> restoreIpAddresses(String s, int k, HashMap<Integer, List<String>> map)
6397     {
6398         int n = s.length();
6399         int key = 4*n+k;
6400         if(map.containsKey(key)) return map.get(key);
6401
6402         List<String> re = new ArrayList<String>();
6403         if(n==0) return re;
6404
6405         if(k==1)
6406         {
6407             if(valid(s))
6408             {
6409                 re.add(s);
6410             }
6411             return re;
6412         }
6413
6414         for(int i=n-1; i>=n-3;i--)
6415         {
6416             if(i<0) break;
6417             if(valid(s.substring(i)))
6418             {
6419                 List<String> r = restoreIpAddresses(s.substring(0,i), k-1, map);
6420                 if(r.size() == 0) continue;
6421                 for(String ss:r)
6422                 {
6423                     re.add(ss+s.substring(i));
6424                 }
6425             }
6426         }
6427         map.put(key, re);
6428         return re;
6429     }
6430
6431     public static List<String> restoreIpAddresses(String s) {
6432         List<String> re = new ArrayList<String>();
6433
6434         int n = s.length();
6435         if(n==0) return re;
6436         return restoreIpAddresses(s, 4, new HashMap<Integer, List<String>>());
6437     }
6438 }
6439
6440
6441 =====
6442 /Users/linxie/Documents/github/leetcodeHub/code/ReverseInteger
6443 =====
6444
6445
6446 ReverseInteger

```

```

6447
6448 Reverse digits of an integer.
6449
6450 Example1: x = 123, return 321
6451 Example2: x = -123, return -321
6452
6453 --
6454
6455 public class Solution {
6456     public int reverse(int x) {
6457         if(x>-10 && x<10) return x;
6458
6459         boolean neg = false;
6460         if(x<0){ x = -x; neg = true; }
6461
6462         long re = 0;
6463         while(x!=0)
6464         {
6465             re = 10*re + (long)(x%10);
6466             x/=10;
6467         }
6468         if(neg) re=(-1)*re;
6469         if(x>Integer.MAX_VALUE) return Integer.MAX_VALUE;
6470         if(x<Integer.MIN_VALUE) return Integer.MIN_VALUE;
6471         return (int) re;
6472     }
6473 }
6474
6475
6476 =====
6477 /Users/linxie/Documents/github/leetcodeHub/code/ReverseLinkedListII
6478 =====
6479
6480
6481 ReverseLinkedListII
6482
6483 Reverse a linked list from position m to n. Do it in-place and in one-pass.
6484
6485 For example:
6486 Given 1->2->3->4->5->NULL, m = 2 and n = 4,
6487
6488 return 1->4->3->2->5->NULL.
6489
6490 Note:
6491 Given m, n satisfy the following condition:
6492 1 ≤ m ≤ n ≤ length of list.
6493
6494 --
6495 /**
6496 * Definition for singly-linked list.
6497 * public class ListNode {
6498 *     int val;
6499 *     ListNode next;
6500 *     ListNode(int x) {
6501 *         val = x;
6502 *         next = null;
6503 *     }
6504 * }
6505 */
6506 public class Solution {
6507     public ListNode reverse(ListNode nd)
6508     {
6509         if(nd == null || nd.next == null) return nd;
6510         ListNode newHead = null, cur = nd;
6511
6512         while(cur!=null)
6513         {
6514             ListNode tmp = cur.next;
6515             cur.next = newHead;
6516             newHead = cur;
6517             cur = tmp;
6518         }
6519         return newHead;
6520     }
6521
6522     public ListNode reverseBetween(ListNode head, int m, int n) {
6523         if(head == null) return null;
6524         if(n<=m) return head;
6525
6526         ListNode cur = head;
6527         int count =0;
6528
6529         while(cur.next!=null && count <n-m)
6530         {
6531             count++;
6532             cur = cur.next;
6533         }
6534         if(count!=n-m) return head;
6535
6536         ListNode sec = head, prev = null;
6537
6538         count = 1;
6539         while(count<m && cur.next!=null)
6540         {
6541             count++;
6542             cur = cur.next;
6543             prev = sec;
6544             sec = sec.next;
6545         }
6546         if(count!=m) return head;
6547
6548
6549         ListNode rd = cur.next;
6550         cur.next = null;

```

```

6551     if(prev == null)    head = reverse(sec);
6552     else    prev.next = reverse(sec);
6553     sec.next = rd;
6554     return head;
6555 }
6556 }
6558
6559 =====
6560 /Users/linxie/Documents/github/leetcodeHub/code/ReverseNodesink-Group
6562 =====
6563
6564
6565 ReverseNodesink-Group
6566
6567 Given a linked list, reverse the nodes of a linked list k at a time and
6568 return its modified list.
6569
6570 If the number of nodes is not a multiple of k then left-out nodes in the end should
6571 remain as it is.
6572
6573 You may not alter the values in the nodes, only nodes itself may be changed.
6574
6575 Only constant memory is allowed.
6576
6577 For example,
6578 Given this linked list: 1->2->3->4->5
6579
6580 For k = 2, you should return: 2->1->4->3->5
6581
6582 For k = 3, you should return: 3->2->1->4->5
6583
6584 ---
6585
6586 /**
6587 * Definition for singly-linked list.
6588 * public class ListNode {
6589 *     int val;
6590 *     ListNode next;
6591 *     ListNode(int x) {
6592 *         val = x;
6593 *         next = null;
6594 *     }
6595 * }
6596 */
6597 public class Solution {
6598     public ListNode reverseKGroup(ListNode head, int k) {
6599
6600         if(k<=1) return head;
6601         if(head == null) return head;
6602         ListNode start = head;
6603
6604         head = null;
6605         ListNode tail = null;
6606
6607         while(true)
6608     {
6609             ListNode cur = start;
6610             int count = 1;
6611             while(cur.next != null && count<k)
6612             {
6613                 count++; cur = cur.next;
6614             }
6615             if(count<k)
6616             {
6617                 if(tail == null)
6618                 {
6619                     head = start;
6620                     break;
6621                 }
6622                 else
6623                 {
6624                     tail.next = start;
6625                     tail = cur;
6626                     start = cur.next;
6627                     tail.next = null;
6628                 }
6629             }
6630             else
6631             {
6632                 ListNode newStart = cur.next;
6633                 ListNode prev = null;
6634                 cur.next = null;
6635                 cur = start;
6636                 start = null;
6637                 while(cur!=null)
6638                 {
6639                     if(start == null) prev = cur;
6640                     ListNode t = cur.next;
6641                     cur.next = start;
6642                     start = cur;
6643                     cur = t;
6644                 }
6645                 if(tail == null)
6646                 {
6647                     head = start;
6648                     tail = prev;
6649                 }
6650                 else
6651                 {
6652                     tail.next = start;
6653                     tail = prev;
6654                 }
6655             }
6656         }
6657     }
6658 }

```

```

6655         start = newStart;
6656     }
6657     if(start==null) break;
6658   }
6659   return head;
6660 }
6661 }
6662 }
6663
6664
6665 ====
6666
6667 Using function to do reverse, cleaner:
6668
6669 public ListNode reverse(ListNode head)
6670 {
6671     if(head == null || head.next == null) return head;
6672     ListNode cur = head, newHead = null;
6673
6674     while(cur!=null)
6675     {
6676         ListNode t = cur.next;
6677         cur.next = newHead;
6678         newHead = cur;
6679         cur = t;
6680     }
6681     return newHead;
6682 }
6683
6684 public ListNode reverseKGroup(ListNode head, int k)
6685 {
6686     if(k<=1) return head;
6687     ListNode start = head, tail = null;
6688     head = null;
6689
6690     while(start!=null)
6691     {
6692         ListNode cur = start;
6693         int count = 1;
6694         while(cur.next!=null && count <k)
6695         {
6696             count++;
6697             cur = cur.next;
6698         }
6699         if(count <k)
6700         {
6701             if(tail == null)
6702             {
6703                 head = start;
6704             }
6705             else
6706             {
6707                 tail.next = start;
6708             }
6709             break;
6710         }
6711     else
6712     {
6713         ListNode t = cur.next;
6714         cur.next = null;
6715         ListNode rev = reverse(start);
6716         if(tail == null)
6717         {
6718             head = rev; tail = start;
6719         }
6720         else
6721         {
6722             tail.next = rev;
6723             tail = start;
6724         }
6725         start = t;
6726     }
6727 }
6728 return head;
6729 }
6730
6731
6732
6733
6734
6735 =====
6736 /Users/linxie/Documents/github/leetcodeHub/code/ReverseWordsinaString
6737 =====
6738
6739
6740 ReverseWordsinaString
6741
6742 Given an input string, reverse the string word by word.
6743
6744 For example,
6745 Given s = "the sky is blue",
6746 return "blue is sky the".
6747
6748 ----
6749
6750 public class Solution {
6751     public void reverse(char[] s, int l, int r)
6752     {
6753         while(l<r)
6754         {
6755             char c = s[l];
6756             s[l] = s[r];
6757             s[r] = c;
6758             l++; r--;

```

```

6759     }
6760 }
6761 public String reverseWords(String s) {
6762     char[] c = s.toCharArray();
6763     int n = c.length;
6764
6765     int i=0, j=0;
6766     for(;j<n;j++)
6767     {
6768         while(j<n && c[j] == ' ') j++;
6769         if(j==n) break;
6770
6771         if(i!=0) c[i++]= ' ';
6772         while(j<n && c[j]!= ' ') c[i++] = c[j++];
6773         if(j==n) break;
6774     }
6775     if(i==0) return "";
6776     n=i;
6777     reverse(c, 0, i-1);
6778     i=0; j=0;
6779     for(;j<n;j++)
6780     {
6781         i=j;
6782         for(;j<n && c[j]!= ' ';j++);
6783         reverse(c,i,j-1);
6784     }
6785     char[] newC = new char[n];
6786     for(i=0; i<n;i++)
6787         newC[i] = c[i];
6788     return new String(newC);
6789 }
6790 }
6791
6792
6793 =====
6794 /Users/linxie/Documents/github/leetcodeHub/code/RomantoInteger
6795 =====
6796
6797
6798 RomantoInteger
6799
6800 Given a roman numeral, convert it to an integer.
6801
6802 Input is guaranteed to be within the range from 1 to 3999.
6803
6804 ---
6805
6806 public class Solution {
6807     public int romanToInt(String s) {
6808         int n = s.length();
6809         if(n==0) return 0;
6810
6811         HashMap<Character, Integer> m = new HashMap<Character, Integer>();
6812         m.put(M, 1000);
6813         m.put(D, 500);
6814         m.put(C, 100);
6815         m.put(L, 50);
6816         m.put(X, 10);
6817         m.put(V, 5);
6818         m.put(I, 1);
6819         char[] c = s.toCharArray();
6820         int re = m.get(c[0]);
6821         for(int i=1;i<n;i++)
6822         {
6823             int pre = m.get(c[i-1]);
6824             int cur = m.get(c[i]);
6825             if(cur>pre)
6826                 re+=(cur-2*pre);
6827             else re+=cur;
6828         }
6829         return re;
6830     }
6831 }
6832
6833
6834 =====
6835 /Users/linxie/Documents/github/leetcodeHub/code/RotateImage
6836 =====
6837
6838
6839 RotateImage
6840
6841 You are given an  $n \times n$  2D matrix representing an image.
6842
6843 Rotate the image by 90 degrees (clockwise).
6844
6845 ---
6846 public class Solution {
6847     public void rotate(int[][] matrix) {
6848
6849         int m=matrix.length;
6850         if(m==0) return;
6851         int n = matrix[0].length;
6852         if(n==0) return;
6853
6854         int layer = (Math.min(m,n)+1)/2;
6855         for(int i=0;i<layer;i++)
6856         {
6857             for(int j=i;j<n-i-1;j++)
6858             {
6859                 int t = matrix[n-1-j][i];
6860                 matrix[n-1-j][i] = matrix[n-i-1][n-1-j];
6861                 matrix[n-i-1][n-1-j] = matrix[j][n-i-1];
6862                 matrix[j][n-i-1] = matrix[i][j];
6863             }
6864         }
6865     }
6866 }

```

```

6863         matrix[i][j] = t;
6864     }
6865 }
6866 return;
6867 }
6868 }
6869 }
6870
6871 =====
6872 /Users/linxie/Documents/github/leetcodeHub/code/RotateList
6873 =====
6874 =====
6875
6876
6877 RotateList
6878
6879 Given a list, rotate the list to the right by k places, where k is non-negative.
6880
6881 For example:
6882 Given 1->2->3->4->5->NULL and k = 2,
6883 return 4->5->1->2->3->NULL.
6884
6885 ---
6886
6887 /**
6888 * Definition for singly-linked list.
6889 * public class ListNode {
6890 *     int val;
6891 *     ListNode next;
6892 *     ListNode(int x) {
6893 *         val = x;
6894 *         next = null;
6895 *     }
6896 * }
6897 */
6898 public class Solution {
6899     public ListNode rotateRight(ListNode head, int n) {
6900         if(head == null) return null;
6901         if(n==0) return head;
6902
6903         int count = 0;
6904         ListNode first = head, second = head;
6905
6906         while(first.next!=null)
6907         {
6908             count++;
6909             first = first.next;
6910             if(count > n) second = second.next;
6911         }
6912
6913         if(count<n) return rotateRight(head, n%(count+1));
6914
6915         ListNode newHead = second.next;
6916         second.next = null;
6917         first.next = head;
6918         return newHead;
6919     }
6920 }
6921
6922
6923 =====
6924 /Users/linxie/Documents/github/leetcodeHub/code/SameTree
6925 =====
6926
6927
6928 SameTree
6929
6930 Given two binary trees, write a function to check if they are equal or not.
6931
6932 Two binary trees are considered equal if they are structurally identical and the
6933 nodes have the same value.
6934
6935 ---
6936 /**
6937 * Definition for binary tree
6938 * public class TreeNode {
6939 *     int val;
6940 *     TreeNode left;
6941 *     TreeNode right;
6942 *     TreeNode(int x) { val = x; }
6943 * }
6944 */
6945 public class Solution {
6946     public boolean isSameTree(TreeNode p, TreeNode q) {
6947         if(p==null)
6948             return q==null;
6949         if(q==null) return false;
6950         return p.val == q.val && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
6951     }
6952 }
6953
6954
6955 =====
6956 /Users/linxie/Documents/github/leetcodeHub/code/ScrambleString
6957 =====
6958
6959
6960 ScrambleString
6961
6962 Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty
6963 substrings recursively.
6964
6965 Below is one possible representation of s1 = "great":
6966

```

```

6967     great
6968     /   \
6969     gr   eat
6970     / \   / \
6971     g   r   e   at
6972             / \
6973             a   t
6974 To scramble the string, we may choose any non-leaf node and swap its two children.
6975
6976 For example, if we choose the node "gr" and swap its two children, it produces a scrambled
6977 string "rgeat".
6978
6979     rgeat
6980     /   \
6981     rg   eat
6982     / \   / \
6983     r   g   e   at
6984             / \
6985             a   t
6986 We say that "rgeat" is a scrambled string of "great".
6987
6988 Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a
6989 scrambled string "rgtae".
6990
6991     rgtae
6992     /   \
6993     rg   tae
6994     / \   / \
6995     r   g   ta   e
6996             / \
6997             t   a
6998 We say that "rgtae" is a scrambled string of "great".
6999
7000 Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.
7001 ---
7002
7003 class Pair
7004 {
7005     String s1;
7006     String s2;
7007     public Pair(String _s1, String _s2)
7008     {
7009         s1 = _s1; s2 = _s2;
7010     }
7011
7012     public int hashCode()
7013     {
7014         return 31*s1.hashCode() + s2.hashCode();
7015     }
7016     public boolean equals(Object o)
7017     {
7018         if(o!=null && o instanceof Pair)
7019         {
7020             Pair _o = (Pair)o;
7021             return (s1.equals(_o.s1) && s2.equals(_o.s2)) || (s2.equals(_o.s1) && s1.equals(_o.s2));
7022         }
7023         return false;
7024     }
7025 }
7026
7027 public class Solution {
7028     public boolean isScramble(String s1, String s2, HashMap<Pair, Boolean> map)
7029     {
7030         int n = s1.length();
7031         if(n!=s2.length()) return false;
7032         if(s1.equals(s2)) return true;
7033
7034         Pair p = new Pair(s1, s2);
7035         if(map.containsKey(p)) return map.get(p);
7036
7037         for(int i=1; i< n; i++)
7038         {
7039             if(( isScramble(s1.substring(0, i), s2.substring(0,i), map)
7040                 && isScramble(s1.substring(i), s2.substring(i), map) )
7041                 || ( isScramble(s1.substring(0, i), s2.substring(n-i), map)
7042                     && isScramble(s1.substring(i), s2.substring(0, n-i), map) ) )
7043             {
7044                 map.put(p,true);
7045                 return true;
7046             }
7047         }
7048         map.put(p,false);
7049         return false;
7050     }
7051
7052     public boolean isScramble(String s1, String s2) {
7053         return isScramble(s1, s2, new HashMap<Pair, Boolean>());
7054     }
7055 }
7056
7057
7058 =====
7059 /Users/linxie/Documents/github/leetcodeHub/code/Searcha2DMatrix
7060 =====
7061
7062
7063 Searcha2DMatrix
7064
7065 Write an efficient algorithm that searches for a value in an m x n matrix. This matrix
7066 has the following properties:
7067
7068 Integers in each row are sorted from left to right.
7069 The first integer of each row is greater than the last integer of the previous row.
7070 For example,

```

```

7071
7072 Consider the following matrix:
7073 [
7075   [1, 3, 5, 7],
7076   [10, 11, 16, 20],
7077   [23, 30, 34, 50]
7078 ]
7079 Given target = 3, return true.
7080
7081
7082 ---
7083
7084 public class Solution {
7085     public int findRight(int[] a, int l, int r, int e)
7086     {
7087         int re = -1;
7088         if(r-l<0 || r-l>=a.length) return re;
7089         while(l<=r)
7090         {
7091             int mid = l +(r-l)/2;
7092             if(a[mid] == e) return mid;
7093             else if(a[mid]>e)
7094             {
7095                 re = mid;
7096                 r = mid-1;
7097             }
7098             else
7099                 l = mid+1;
7100         }
7101         return re;
7102     }
7103
7104     public boolean searchMatrix(int[][] matrix, int target)
7105     {
7106         int m = matrix.length;
7107         if(m==0) return false;
7108         int n = matrix[0].length;
7109         if(n==0) return false;
7110
7111         int[] b = new int[m];
7112         for(int i=0; i<m; i++)
7113         {
7114             b[i] = matrix[i][n-1];
7115         }
7116         int r = findRight(b, 0, m-1, target);
7117         if(r == -1) return false;
7118         if(b[r] == target) return true;
7119
7120         int rr = findRight(matrix[r], 0, n-1, target);
7121         if(rr == -1 || matrix[r][rr]!=target) return false;
7122         return true;
7123     }
7124 }
7125
7126
7127 =====
7128 /Users/linxie/Documents/github/leetcodeHub/code/SearchforaRange
7129 =====
7130
7131
7132 SearchforaRange
7133
7134 Given a sorted array of integers, find the starting and ending position of a given target value.
7135
7136 Your algorithms runtime complexity must be in the order of O(log n).
7137
7138 If the target is not found in the array, return [-1, -1].
7139
7140 For example,
7141 Given [5, 7, 7, 8, 8, 10] and target value 8,
7142 return [3, 4].
7143
7144 ---
7145
7146 public class Solution {
7147     public int searchLeft(int[] A, int target)
7148     {
7149         int n = A.length;
7150         int re = -1;
7151         int l = 0, r = n-1;
7152         while(l<=r)
7153         {
7154             int mid = l+(r-l)/2;
7155             if(A[mid] == target)
7156             {
7157                 re = mid;
7158                 r = mid-1;
7159             }
7160             else if(target < A[mid]) r = mid-1;
7161             else l = mid+1;
7162         }
7163         return re;
7164     }
7165
7166     public int searchRight(int[] A, int target)
7167     {
7168         int n = A.length;
7169         int re = -1;
7170         int l = 0, r = n-1;
7171         while(l<=r)
7172         {
7173             int mid = l+(r-l)/2;
7174             if(A[mid] == target)

```

```

7175         {
7176             re = mid;
7177             l = mid+1;
7178         }
7179         else if(target < A[mid]) r = mid-1;
7180         else l = mid+1;
7181     }
7182     return re;
7183 }
7184
7185 public int[] searchRange(int[] A, int target)
7186 {
7187     int n = A.length;
7188     int left = searchLeft(A, target);
7189     int right = searchRight(A, target);
7190     int[] re = {left, right};
7191     return re;
7192 }
7193 }
7194
7195 =====
7196 /Users/linxie/Documents/github/leetcodeHub/code/SearchinRotatedSortedArray
7197 =====
7198
7199
7200
7201 SearchinRotatedSortedArray
7202
7203 Suppose a sorted array is rotated at some pivot unknown to you beforehand.
7204
7205 (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
7206
7207 You are given a target value to search. If found in the array return its index, otherwise return -1.
7208
7209 You may assume no duplicate exists in the array.
7210
7211 ---
7212
7213 public class Solution {
7214     public int search(int[] A, int target) {
7215
7216         int n = A.length;
7217         if(n==0) return -1;
7218         if(n==1)
7219             if(A[0] == target) return 0;
7220             else return -1;
7221
7222         int l = 0, r = n-1;
7223
7224         while(l<=r)
7225         {
7226             int mid = l+(r-l)/2;
7227             if(A[mid] == target) return mid;
7228
7229             if(A[l]<A[mid])
7230             {
7231                 if(target >= A[l] && target < A[mid]) r = mid-1;
7232                 else l = mid+1;
7233             }
7234             else if (A[l] > A[mid])
7235             {
7236                 if(target > A[mid] && target <= A[r]) l = mid+1;
7237                 else r = mid-1;
7238             }
7239             else l++;
7240         }
7241         return -1;
7242
7243     }
7244 }
7245
7246
7247 =====
7248 /Users/linxie/Documents/github/leetcodeHub/code/SearchinRotatedSortedArrayII
7249 =====
7250
7251
7252 SearchinRotatedSortedArrayII
7253
7254 Follow up for "Search in Rotated Sorted Array":
7255 What if duplicates are allowed?
7256
7257 Would this affect the run-time complexity? How and why?
7258
7259 Write a function to determine if a given target is in the array.
7260
7261 ---
7262
7263 public class Solution {
7264     public boolean search(int[] A, int target) {
7265
7266         int n = A.length;
7267         if(n==0) return false;
7268         if(n==1) return A[0] == target;
7269
7270         int l=0, r = n-1;
7271
7272         while(l<=r)
7273         {
7274             int mid = l+(r-l)/2;
7275             if(A[mid] == target) return true;
7276             if(A[mid]<A[l])
7277             {
7278                 if(target>A[mid] && target<=A[r]) l=mid+1;
7279             }
7280         }
7281         return false;
7282     }
7283 }
```

```

7279         else r=mid-1;
7280     }
7281     else if(A[mid]>A[1])
7282     {
7283         if(target>=A[1] && target<A[mid]) r = mid-1;
7284         else l = mid+1;
7285     }
7286     else l++;
7287 }
7288     return false;
7289 }
7290 }
7291 }
7292
7293 =====
7294 /Users/linxie/Documents/github/leetcodeHub/code/SearchInsertPosition
7295 =====
7296
7297
7298
7299 SearchInsertPosition
7300
7301 Given a sorted array and a target value, return the index if the target is found. If not,
7302 return the index where it would be if it were inserted in order.
7303
7304 You may assume no duplicates in the array.
7305
7306 Here are few examples.
7307 [1,3,5,6], 5 → 2
7308 [1,3,5,6], 2 → 1
7309 [1,3,5,6], 7 → 4
7310 [1,3,5,6], 0 → 0
7311
7312 --
7313
7314 public class Solution {
7315     public int searchInsert(int[] A, int target) {
7316         int n = A.length;
7317         int l = 0, r = n-1;
7318         int re = -1;
7319
7320         while(l<=r)
7321         {
7322             int mid = l+(r-l)/2;
7323             if(A[mid] == target)
7324             {
7325                 re = mid; break;
7326             }
7327             else if(A[mid]<target)
7328             {
7329                 l = mid+1;
7330                 re = l;
7331             }
7332             else
7333             {
7334                 re = mid;
7335                 r = mid-1;
7336             }
7337         }
7338         return re;
7339     }
7340 }
7341
7342 =====
7343 /Users/linxie/Documents/github/leetcodeHub/code/SerializeAndDeserializeBT
7344 =====
7345
7346
7347
7348 SerializeAnd DeserializeBT
7349
7350 pre-order
7351
7352 =====
7353 class Node
7354 {
7355     String str;
7356     Node(String _str){ str = _str;}
7357 }
7358
7359 class Solution
7360 {
7361
7362     public String serialize(TreeNode p)
7363     {
7364         if(p == null) return "#";
7365         String re = Integer.toString(p.val);
7366         re+=serialize(p.left);
7367         re+=serialize(p.right);
7368         return re;
7369     }
7370
7371     public TreeNode deserialize(Node s)
7372     {
7373         if(s.str.length() == 0) return null;
7374         if(s.str.charAt(0)==#)
7375         {
7376             s.str = s.str.substring(1);
7377             return null;
7378         }
7379         TreeNode root = new TreeNode((int)(s.str.charAt(0)-0));
7380         s.str=s.str.substring(1);
7381         root.left = deserialize(s);
7382         root.right = deserialize(s);

```

```

7383     return root;
7384 }
7385 }
7386
7387 =====
7388 /Users/linxie/Documents/github/leetcodeHub/code/SetMatrixZeroes
7389 =====
7390
7391
7392
7393 SetMatrixZeroes
7394
7395 Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place.
7396
7397 ---
7398 public class Solution {
7399     public void setZeroes(int[][] matrix) {
7400
7401         int m=matrix.length;
7402         if(m==0) return;
7403         int n = matrix[0].length;
7404         if(n==0) return;
7405
7406         boolean col =false, row = false;
7407         for(int i:matrix[0])
7408             if(i==0)
7409             {
7410                 row = true;
7411                 break;
7412             }
7413         for(int i=0;i<m;i++)
7414             if(matrix[i][0] == 0)
7415             {
7416                 col = true;
7417                 break;
7418             }
7419
7420         for(int i=1; i<m;i++)
7421             for(int j=1;j<n;j++)
7422                 if(matrix[i][j] == 0)
7423                 {
7424                     matrix[0][j] = 0;
7425                     matrix[i][0] = 0;
7426                 }
7427
7428         for(int i=1;i<m;i++)
7429             for(int j=1;j<n;j++)
7430                 if(matrix[i][0]==0 || matrix[0][j]==0)
7431                     matrix[i][j] = 0;
7432
7433         if(row)
7434             for(int i=0;i<n;i++) matrix[0][i] = 0;
7435         if(col)
7436             for(int i=0;i<m;i++) matrix[i][0] = 0;
7437     }
7438 }
7439
7440
7441 =====
7442 /Users/linxie/Documents/github/leetcodeHub/code/SimplifyPath
7443 =====
7444
7445
7446 SimplifyPath
7447
7448 Given an absolute path for a file (Unix-style), simplify it.
7449
7450 For example,
7451 path = "/home/", => "/home"
7452 path = "/a//b/../../c/", => "/c"
7453 click
7454
7455 ---
7456
7457 public class Solution {
7458     public String simplifyPath(String path) {
7459         int n = path.length();
7460         if(n==0) return "";
7461
7462         Stack<String> s = new Stack<String>();
7463         int i=0, j=0;
7464         String re = "";
7465         while(i<n)
7466         {
7467             while(i<n && path.charAt(i)==/) i++;
7468             if(i==n) break;
7469
7470             j = i;
7471             while(j<n && path.charAt(j)!=/) j++;
7472
7473             String val = path.substring(i,j);
7474             if(val.equals(".."))
7475             {
7476                 if(!s.isEmpty()) s.pop();
7477             }
7478             else if(!val.equals(".")) s.push(val);
7479             i = j;
7480         }
7481         if(s.isEmpty()) re = "/";
7482         else
7483         {
7484             while(!s.isEmpty())
7485             {
7486                 re = "/" + s.pop() + re;

```

```

7487         }
7488     }
7489     return re;
7490 }
7491 }
7492
7493
7494 =====
7495 /Users/linxie/Documents/github/leetcodeHub/code/SingleNumber
7496 =====
7497
7498
7499 SingleNumber
7500
7501 Given an array of integers, every element appears twice except for one. Find that single one.
7502
7503 Note:
7504 Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?
7505 ---
7506 public class Solution {
7507     public int singleNumber(int[] A) {
7508         if(A.length==0) System.exit(-1);
7509         int re=0;
7510         for(int i:A)
7511             re^=i;
7512         return re;
7513     }
7514 }
7515
7516
7517 =====
7518 /Users/linxie/Documents/github/leetcodeHub/code/SingleNumberII
7519 =====
7520
7521
7522 SingleNumberII
7523
7524 Given an array of integers, every element appears three times except for one. Find that single one.
7525
7526 Note:
7527 Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?
7528 ---
7529
7530 public class Solution {
7531     public int singleNumber(int[] A) {
7532         int n = A.length;
7533         if(n==1) return A[0];
7534         if(n==0 || n%3!=1) return 0;
7535
7536         int re=0;
7537         for(int i=0;i<32;i++)
7538         {
7539             int count = 0;
7540             for(int j=0; j<n; j++)
7541             {
7542                 if((A[j] & (1<<i))!=0) count++;
7543             }
7544             if(count%3!=0) re = re|(1<<i);
7545         }
7546         return re;
7547     }
7548 }
7549
7550
7551 =====
7552 /Users/linxie/Documents/github/leetcodeHub/code/SortColors
7553 =====
7554
7555
7556 SortColors
7557
7558 Given an array with n objects colored red, white or blue, sort them so that objects of
7559 the same color are adjacent, with the colors in the order red, white and blue.
7560
7561 Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
7562 ---
7563
7564 public class Solution {
7565     public void sortColors(int[] A) {
7566
7567         int n = A.length;
7568         if(n<=1) return;
7569
7570         int i=0, k=n-1;
7571         while(i<n && A[i]==0) i++;
7572         if(i==n) return;
7573         while(k>=0 && A[k]==2) k--;
7574         if(k<0) return;
7575
7576         int j = i;
7577         while(j<=k)
7578         {
7579             if(A[j]==1) j++;
7580             else if(A[j]==0)
7581             {
7582                 int t = A[i];
7583                 A[i] = A[j];
7584                 A[j] = t;
7585                 i++;
7586             }else
7587             {
7588                 int t = A[k];
7589                 A[k] = A[j];
7590                 A[j] = t;
7591             }
7592         }
7593     }
7594 }
```

```

7591             k--;
7592         }
7593     if(j < i) j++;
7594   }
7595 }
7596 }
7597 }
7598 =====
7600 /Users/linxie/Documents/github/leetcodeHub/code/SortList
7601 =====
7602 =====
7603
7604
7605 SortList
7606
7607 Sort a linked list in  $O(n \log n)$  time using constant space complexity.
7608 ---
7609 /**
7610 * Definition for singly-linked list.
7611 * class ListNode {
7612 *     int val;
7613 *     ListNode next;
7614 *     ListNode(int x) {
7615 *         val = x;
7616 *         next = null;
7617 *     }
7618 * }
7619 */
7620 public class Solution {
7621     public static ListNode merge(ListNode n1, ListNode n2)
7622     {
7623         if(n1==null) return n2;
7624         if(n2==null) return n1;
7625
7626         ListNode head = null, tail = null;
7627         while(n1!=null && n2!=null)
7628         {
7629             if(n1.val <=n2.val)
7630             {
7631                 if(tail == null)
7632                 {
7633                     head = n1;
7634                     tail = head;
7635                     n1 = n1.next;
7636                     tail.next = null;
7637                 }
7638                 else
7639                 {
7640                     tail.next = n1;
7641                     tail = n1;
7642                     n1=n1.next;
7643                     tail.next = null;
7644                 }
7645             }
7646             else
7647             {
7648                 if(tail == null)
7649                 {
7650                     head = n2;
7651                     tail = head;
7652                     n2 = n2.next;
7653                     tail.next = null;
7654                 }
7655             }
7656             else
7657             {
7658                 tail.next = n2;
7659                 tail = n2;
7660                 n2=n2.next;
7661                 tail.next = null;
7662             }
7663
7664         }
7665     }
7666     if(n1!=null)
7667     {
7668         tail.next = n1;
7669     }
7670     if(n2!=null)
7671     {
7672         tail.next = n2;
7673     }
7674     return head;
7675 }
7676
7677 public static ListNode sortList(ListNode head) {
7678     if(head == null || head.next == null) return head;
7679
7680     int count =0;
7681     ListNode l = head, r = head;
7682     while(l.next!=null)
7683     {
7684         count++;
7685         l = l.next;
7686         if(count%2==0) r = r.next;
7687     }
7688     l = r.next;
7689     r.next = null;
7690     return merge(sortList(head), sortList(l));
7691 }
7692 }
7693
7694

```

```

7695 =====
7696 /Users/linxie/Documents/github/leetcodeHub/code/SpiralMatrix
7697 =====
7698
7699
7700 SpiralMatrix
7701
7702 Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix
7703 in spiral order.
7704
7705 For example,
7706 Given the following matrix:
7707
7708 [
7709 [ 1, 2, 3 ],
7710 [ 4, 5, 6 ],
7711 [ 7, 8, 9 ]
7712 ]
7713 You should return [1,2,3,6,9,8,7,4,5].
7714 ---
7715
7716 public class Solution {
7717     public List<Integer> spiralOrder(int[][] matrix) {
7718
7719         List<Integer> re = new ArrayList<Integer>();
7720         int m = matrix.length;
7721         if(m==0) return re;
7722         int n = matrix[0].length;
7723         if(n==0) return re;
7724
7725         int layer = (Math.min(m,n)+1)/2;
7726         int count = 0;
7727         for(int i=0;i<layer;i++)
7728         {
7729             for(int j=i; j<n-i; j++)
7730             {
7731                 re.add(matrix[i][j]);
7732                 if(++count >= m*n) break;
7733             }
7734             if(count >= m*n) break;
7735             for(int j=i+1; j<m-i; j++)
7736             {
7737                 re.add(matrix[j][n-i-1]);
7738                 if(++count >= m*n) break;
7739             }
7740             if(count >= m*n) break;
7741             for(int j=n-i-2; j>=i; j--)
7742             {
7743                 re.add(matrix[m-i-1][j]);
7744                 if(++count >= m*n) break;
7745             }
7746             if(count >= m*n) break;
7747             for(int j=m-i-2; j>i; j--)
7748             {
7749                 re.add(matrix[j][i]);
7750                 if(++count >= m*n) break;
7751             }
7752             if(count >= m*n) break;
7753         }
7754         return re;
7755
7756
7757     }
7758 }
7759
7760
7761 =====
7762 /Users/linxie/Documents/github/leetcodeHub/code/SpiralMatrixII
7763 =====
7764
7765
7766 SpiralMatrixII
7767
7768 Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.
7769
7770 For example,
7771 Given n = 3,
7772
7773 You should return the following matrix:
7774 [
7775 [ 1, 2, 3 ],
7776 [ 8, 9, 4 ],
7777 [ 7, 6, 5 ]
7778 ]
7779 ---
7780
7781 public class Solution {
7782     public int[][] generateMatrix(int n) {
7783
7784         int[][] re = new int[n][n];
7785         if(n==0) return re;
7786
7787         int layer = (n+1)/2;
7788         int in = 1;
7789
7790         for(int i = 0; i<layer; i++)
7791         {
7792             for(int j = i; j<n-i; j++) re[i][j] = (in++);
7793             for(int j = i+1; j<n-i; j++) re[j][n-i-1] = (in++);
7794             for(int j = n-i-2; j>=i; j--) re[n-i-1][j] = (in++);
7795             for(int j = n-i-2; j>i; j--) re[j][i] = (in++);
7796         }
7797         return re;
7798

```

```

7799     }
7800 }
7801
7802 =====
7803 /Users/linxie/Documents/github/leetcodeHub/code/Sqrt(x)
7804 =====
7805 =====
7806
7807
7808 Sqrt(x)
7809
7810 Implement int sqrt(int x).
7811
7812 Compute and return the square root of x.
7813
7814 --
7815 --
7816 public class Solution {
7817     public int sqrt(int x) {
7818
7819         if(x<0) return -1;
7820         if(x<=1) return x;
7821
7822         int l = 1, r=x;
7823         int re = -1;
7824
7825         while(l<=r)
7826         {
7827             int mid = l+(r-l)/2;
7828             long result = (long)mid*(long)mid;
7829             if(result==x) return mid;
7830             else if(result < x)
7831             {
7832                 re = mid; l = mid+1;
7833             }
7834             else
7835             {
7836                 r = mid-1;
7837             }
7838         }
7839         return re;
7840     }
7841 }
7842 }
7843 ===
7844 Newton for double sqrt
7845
7846
7847 public static double sqrt(double c)
7848 {
7849     if(c<0) return Double.NaN;
7850     double EPS = 1E-15;
7851
7852     double t = c;
7853     while(Math.abs(t-c/t) > EPS*t)
7854         t = (c/t+t)/2.0;
7855     return t;
7856 }
7857
7858 =====
7859 /Users/linxie/Documents/github/leetcodeHub/code/StringtoInteger
7860 =====
7861 =====
7862
7863
7864 StringtoInteger
7865
7866 Implement atoi to convert a string to an integer.
7867
7868 Hint: Carefully consider all possible input cases. If you want a challenge, please do not see
7869 below and ask yourself what are the possible input cases.
7870
7871 Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You
7872 are responsible to gather all the input requirements up front
7873 --
7874
7875 public class Solution {
7876     public int atoi(String str) {
7877
7878         char[] s = str.toCharArray();
7879         int n = s.length;
7880         if(n==0) return 0;
7881
7882         boolean number = false;
7883         int sign = 0;
7884
7885         long re = 0;
7886         for(char c : s)
7887         {
7888             if(!number && (sign == 0) && c== ) continue;
7889
7890             int num = (int)(c-0);
7891
7892             if(num<0 || num>9)
7893             {
7894                 if( sign!=0 || (c!+= && c!=-) ) break;
7895
7896                 if(c == -) sign = -1;
7897                 else sign =1;
7898             }
7899             else
7900             {
7901                 if(sign == 0) sign =1;
7902                 number = true;

```

```

7903         re = re*10+(long)num;
7904     }
7905 }
7906 if(sign == -1) re = -re;
7907 if(re > Integer.MAX_VALUE) return Integer.MAX_VALUE;
7908 if(re < Integer.MIN_VALUE) return Integer.MIN_VALUE;
7909 return (int)re;
7910
7911 }
7912 }
7913
7914 =====
7915 /Users/linxie/Documents/github/leetcodeHub/code/Subsets
7916 =====
7917 =====
7918
7919
7920 Subsets
7921
7922 Given a set of distinct integers, S, return all possible subsets.
7923
7924 Note:
7925 Elements in a subset must be in non-descending order.
7926 The solution set must not contain duplicate subsets.
7927 For example,
7928 If S = [1,2,3], a solution is:
7929
7930 [
7931     [3],
7932     [1],
7933     [2],
7934     [1,2,3],
7935     [1,3],
7936     [2,3],
7937     [1,2],
7938     []
7939 ]---
7940
7941 public class Solution {
7942
7943     public void subsets(int[] S, int ind, List<List<Integer>> re, List<Integer> r)
7944     {
7945         if(ind >= S.length)
7946         {
7947             List<Integer> rr = new ArrayList(r);
7948             re.add(rr);
7949             return;
7950         }
7951         int tmp = S[ind];
7952
7953         int siz = r.size();
7954         int end = ind;
7955         while(end<S.length && S[end] == tmp)    end++;
7956         subsets(S, end, re, r);
7957
7958         for(int i=ind; i<end; i++)
7959         {
7960             r.add(tmp);
7961             subsets(S, end, re, r);
7962         }
7963
7964         for(int i=r.size()-1; i>=siz; i--)
7965             r.remove(i);
7966     }
7967
7968     public List<List<Integer>> subsets(int[] S) {
7969         List<List<Integer>> re = new ArrayList<List<Integer>>();
7970         int n = S.length;
7971         if(n==0) return re;
7972
7973         List<Integer> r = new ArrayList<Integer>();
7974
7975         Arrays.sort(S);
7976         subsets(S, 0, re, r);
7977         return re;
7978     }
7979 }
7980
7981 =====
7982 /Users/linxie/Documents/github/leetcodeHub/code/SubsetsII
7983 =====
7984 =====
7985
7986
7987 SubsetsII
7988
7989 Given a collection of integers that might contain duplicates, S, return all possible subsets.
7990
7991 Note:
7992 Elements in a subset must be in non-descending order.
7993 The solution set must not contain duplicate subsets.
7994 For example,
7995 If S = [1,2,2], a solution is:
7996
7997 [
7998     [2],
7999     [1],
8000     [1,2,2],
8001     [2,2],
8002     [1,2],
8003     []
8004 ]
8005 ---
```

```

8007 public class Solution {
8008     public void subsetsWithDup(int[] num, int k, List<List<Integer>> re, List<Integer> r)
8009     {
8010         int n = num.length;
8011         if(k==n)
8012         {
8013             List<Integer> l = new ArrayList<Integer>(r);
8014             re.add(l);
8015             return;
8016         }
8017
8018         int val = num[k];
8019         int i = k;
8020         while(i<n && num[i]==val) i++;
8021
8022         int siz = r.size();
8023         for(int j=k; j<=i;j++)
8024         {
8025             subsetsWithDup(num, i, re, r);
8026             r.add(val);
8027         }
8028         int siz2 = r.size();
8029         for(int j=siz2-1;j>=siz;j--)
8030             r.remove(j);
8031     }
8032
8033     public List<List<Integer>> subsetsWithDup(int[] num)
8034     {
8035         List<List<Integer>> re = new ArrayList<List<Integer>>();
8036         int n = num.length;
8037         if(n==0) return re;
8038         Arrays.sort(num);
8039         subsetsWithDup(num, 0, re, new ArrayList<Integer>());
8040         return re;
8041     }
8042 }
8043
8044
8045 =====
8046 /Users/linxie/Documents/github/leetcodeHub/code/SubstringwithConcatenationofAllWords
8047 =====
8048
8049
8050 SubstringwithConcatenationofAllWords
8051
8052 You are given a string, S, and a list of words, L, that are all of the same length. Find all
8053 starting indices of substring(s) in S that is a concatenation of each word in L exactly once
8054 and without any intervening characters.
8055
8056 For example, given:
8057 S: "barfoothefoobarman"
8058 L: ["foo", "bar"]
8059
8060 You should return the indices: [0,9].
8061 (order does not matter).
8062
8063
8064 ---
8065
8066 public class Solution {
8067     public boolean helper(String s, HashMap<String, Integer> map, int k)
8068     {
8069         int m = map.size();
8070         if(m==0)
8071             if(s.length()==0) return true;
8072             else return false;
8073
8074         int n=s.length();
8075         for(int i=0;i<=n-k;i+=k)
8076         {
8077             String str = s.substring(i,i+k);
8078             if(map.containsKey(str) && map.get(str) > 0)
8079             {
8080                 map.put(str, map.get(str)-1);
8081                 if(map.get(str)==0) m--;
8082             }else return false;
8083         }
8084         return m==0;
8085     }
8086
8087     public List<Integer> findSubstring(String S, String[] L)
8088     {
8089         List<Integer> re = new ArrayList<Integer>();
8090         int n = S.length();
8091         if(L.length == 0) return re;
8092         if(n<L.length*L[0].length()) return re;
8093
8094         HashMap<String, Integer> map = new HashMap<String, Integer>();
8095         for(String s:L)
8096         {
8097             if(map.containsKey(s))
8098                 map.put(s, map.get(s)+1);
8099             else
8100                 map.put(s,1);
8101         }
8102
8103         int total = L.length*L[0].length();
8104         for(int i=0;i<=n-total;i++)
8105         {
8106             if( helper(S.substring(i,i+total), new HashMap<String, Integer>(map), L[0].length() ) )
8107                 re.add(i);
8108         }
8109         return re;
8110     }

```

```

8111 }
8112
8113
8114 =====
8115 /Users/linxie/Documents/github/leetcodeHub/code/Sudoku Solver
8116 =====
8117
8118
8119 Sudoku Solver
8120
8121 Write a program to solve a Sudoku puzzle by filling the empty cells.
8122
8123 Empty cells are indicated by the character ..
8124
8125 You may assume that there will be only one unique solution.
8126
8127 ---
8128
8129 public class Solution {
8130     public boolean valid(char[][] board, int i, int j)
8131     {
8132         for(int k=0;k<9;k++)
8133             if(k!=i && board[k][j] == board[i][j]) return false;
8134         for(int k=0;k<9;k++)
8135             if(k!=j && board[i][k] == board[i][j]) return false;
8136         int c = i/3, d = j/3;
8137         int ii = i%3, jj=j%3;
8138
8139         for(int k=0;k<3;k++)
8140             for(int v = 0; v<3; v++)
8141             {
8142                 if((c*3+k != i && d*3+v != j) && board[c*3+k][d*3+v] == board[i][j]) return false;
8143             }
8144         return true;
8145     }
8146
8147     public boolean solveSudoku(char[][] board, int i, int j)
8148     {
8149         int n = board.length;
8150         if(i==n) return true;
8151         if(board[i][j]==.)
8152         {
8153             for(char k = 1;k<=9;k++)
8154             {
8155                 board[i][j] = k;
8156                 if(valid(board,i,j))
8157                 {
8158                     if(j+1==n)
8159                     {
8160                         if(solveSudoku(board, i+1,0)) return true;
8161                     }
8162                     else
8163                     {
8164                         if(solveSudoku(board, i,j+1)) return true;
8165                     }
8166                 }
8167             }
8168             board[i][j] =.;
8169         }
8170         else
8171         {
8172             if(j+1==n)
8173             {
8174                 if(solveSudoku(board, i+1,0)) return true;
8175             }
8176             else
8177             {
8178                 if(solveSudoku(board, i,j+1)) return true;
8179             }
8180         }
8181         return false;
8182     }
8183
8184     public void solveSudoku(char[][] board)
8185     {
8186         solveSudoku(board, 0, 0);
8187     }
8188
8189 }
8190
8191
8192 =====
8193 /Users/linxie/Documents/github/leetcodeHub/code/SumRoottoLeafNumbers
8194 =====
8195
8196
8197 SumRoottoLeafNumbers
8198
8199 Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.
8200
8201 An example is the root-to-leaf path 1->2->3 which represents the number 123.
8202
8203 Find the total sum of all root-to-leaf numbers.
8204
8205 For example,
8206
8207     1
8208     / \
8209     2   3
8210 The root-to-leaf path 1->2 represents the number 12.
8211 The root-to-leaf path 1->3 represents the number 13.
8212
8213 Return the sum = 12 + 13 = 25.
8214 ---

```

```

8215 /**
8216 * Definition for binary tree
8217 * public class TreeNode {
8218 *     int val;
8219 *     TreeNode left;
8220 *     TreeNode right;
8221 *     TreeNode(int x) { val = x; }
8222 * }
8223 */
8224 public class Solution {
8225     public int sumNumbers(TreeNode root, int sum) {
8226         if(root == null) return 0;
8227
8228         int re = sum*10 +root.val;
8229         if(root.left==null && root.right == null) return re;
8230         else    return sumNumbers(root.left, re) + sumNumbers(root.right, re);
8231     }
8232
8233     public int sumNumbers(TreeNode root) {
8234         if(root == null) return 0;
8235         return sumNumbers(root, 0);
8236     }
8237 }
8238
8239
8240 =====
8241 /Users/linxie/Documents/github/leetcodeHub/code/SurroundedRegions
8242 =====
8243
8244
8245 SurroundedRegions
8246
8247 Given a 2D board containing X and O, capture all regions surrounded by X.
8248
8249 A region is captured by flipping all Os into Xs in that surrounded region.
8250
8251 For example,
8252 X X X X
8253 X O O X
8254 X X O X
8255 X O X X
8256 After running your function, the board should be:
8257
8258 X X X X
8259 X X X X
8260 X X X X
8261 X O X X
8262 ---
8263
8264 public class Solution {
8265     static void dfs(char[][] board, int i, int j){
8266         board[i][j] = D;
8267         int m = board.length;
8268         int n = board[0].length;
8269         if(i-1>=0 && board[i-1][j]==O)
8270             dfs(board, i-1, j);
8271         if(j-1>=0 && board[i][j-1]==O)
8272             dfs(board, i, j-1);
8273         if(i+1<m && board[i+1][j]==O)
8274             dfs(board, i+1, j);
8275         if(j+1<n && board[i][j+1]==O)
8276             dfs(board, i, j+1);
8277     }
8278 }
8279
8280     void solve(char[][] board) {
8281
8282         int m = board.length;
8283         if(m==0) return;
8284
8285         int n = board[0].length;
8286         if(n==0) return;
8287
8288         for(int i=0; i<n; i++)
8289         {
8290             if(board[0][i] ==O)
8291                 dfs(board, 0, i);
8292         }
8293
8294         for(int i=0; i<m; i++)
8295         {
8296             if(board[i][n-1] ==O)
8297                 dfs(board, i, n-1);
8298         }
8299
8300         for(int i=n-1; i>=0; i--)
8301         {
8302             if(board[m-1][i] ==O)
8303                 dfs(board, m-1, i);
8304         }
8305         for(int i=m-1; i>=0; i--)
8306         {
8307             if(board[i][0] ==O)
8308                 dfs(board, i, 0);
8309         }
8310         for(int i=0;i<m;i++)
8311             for(int j=0;j<n;j++)
8312                 if(board[i][j]==D)
8313                     board[i][j] = O;
8314
8315     }
8316 }
8317
8318

```

```

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   3
8398 But the following is not:
8399
8400      1
8401      /
8402      2    2
8403      \    \
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;

```

```

8423
8424
8425     if(n1.val == n2.val)
8426         return isSymmetric(n1.left, n2.right) && isSymmetric(n1.right, n2.left);
8427     return false;
8428 }
8429 public boolean isSymmetric(TreeNode root) {
8430     if(root == null) return true;
8431     return isSymmetric(root.left, root.right);
8432 }
8433 }
8434
8435
8436 =====
8437 /Users/linxie/Documents/github/leetcodeHub/code/TextJustification
8438 =====
8439
8440
8441 TextJustification
8442
8443 Given an array of words and a length L, format the text such that each line has exactly L characters
8444 and is fully (left and right) justified.
8445
8446 You should pack your words in a greedy approach; that is, pack as many words as you can in each line.
8447 Pad extra spaces when necessary so that each line has exactly L characters.
8448
8449 Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a
8450 line do not divide evenly between words, the empty slots on the left will be assigned more spaces than
8451 the slots on the right.
8452
8453 For the last line of text, it should be left justified and no extra space is inserted between words.
8454
8455 For example,
8456 words: ["This", "is", "an", "example", "of", "text", "and", "justification."]
8457 L: 16.
8458
8459 Return the formatted lines as:
8460 [
8461     "This    is    an",
8462     "example  of text",
8463     "and       justification."
8464 ]
8465 ---
8466
8467 public class Solution {
8468     public List<String> fullJustify(String[] words, int L) {
8469
8470         List<String> re = new ArrayList<String>();
8471
8472         int cur = 0;
8473         int n = words.length;
8474         while(cur<n)
8475         {
8476             int end = cur;
8477             int res = 0;
8478             while(res<L)
8479             {
8480                 if( end < n && res + words[end].length() <= L-end+cur)
8481                 {
8482                     res+=words[end].length();
8483                     end++;
8484                 }
8485                 else
8486                 {
8487                     break;
8488                 }
8489             }
8490             int spaceNum = L-res;
8491             int num = end - cur;
8492             if(num == 0)
8493             {
8494                 re.add("");
8495                 return re;
8496             }
8497             String result = "";
8498             if(num==1)
8499             {
8500                 result = words[cur];
8501                 for(int i = 0; i<spaceNum; i++)
8502                     result += " ";
8503             }
8504             else
8505             {
8506                 String spaceLess = "";
8507                 int i = 0;
8508
8509                 for(;i<spaceNum/(num-1); i++)
8510                     spaceLess += " ";
8511
8512                 int j = cur;
8513                 if(end!=n)
8514                 {
8515                     for(;j< cur+spaceNum%(num-1); j++)
8516                     {
8517                         result += words[j]+spaceLess+" ";
8518                     }
8519                     for(;j<end-1;j++)
8520                     {
8521                         result += words[j] + spaceLess;
8522                     }
8523                     result += words[j];
8524                 }
8525             else
8526             {

```

```

8527             for(;j<end-1;j++)
8528             {
8529                 result += words[j] + " ";
8530             }
8531             result += words[j];
8532             for(int k=result.length();k<L;k++)
8533                 result+=" ";
8534         }
8535     }
8536 }
8537     re.add(result);
8538     cur = end;
8539 }
8540 return re;
8541 }
8542 }
8543 }
8544
8545
8546
8547 =====
8548 /Users/linxie/Documents/github/leetcodeHub/code/TrappingRainWater
8549 =====
8550
8551
8552 TrappingRainWater
8553
8554 Given n non-negative integers representing an elevation map where the width of each bar is 1,
8555 compute how much water it is able to trap after raining.
8556
8557 For example,
8558 Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.
8559
8560 ---
8561
8562 public class Solution {
8563     public int trap(int[] A) {
8564         int n = A.length;
8565         if(n<=2) return 0;
8566
8567         int[] big = new int[n];
8568         big[0] = A[0];
8569         for(int i=1;i<n;i++)
8570             if(A[i]>big[i-1]) big[i] = A[i];
8571             else big[i] = big[i-1];
8572
8573         int rightBig = A[n-1];
8574         int total = 0;
8575         for(int i=n-2; i>0; i--)
8576         {
8577             if(A[i] < rightBig)
8578             {
8579                 if(A[i]<big[i]) total += Math.min(rightBig, big[i])-A[i];
8580             }
8581             else rightBig = A[i];
8582         }
8583         return total;
8584     }
8585 }
8586
8587
8588 =====
8589 /Users/linxie/Documents/github/leetcodeHub/code/Triangle
8590 =====
8591
8592
8593 Triangle
8594
8595 Given a triangle, find the minimum path sum from top to bottom. Each step you may move to
8596 adjacent numbers on the row below.
8597
8598 For example, given the following triangle
8599 [
8600     [2],
8601     [3,4],
8602     [6,5,7],
8603     [4,1,8,3]
8604 ]
8605 The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).
8606
8607 Note:
8608 Bonus point if you are able to do this using only O(n) extra space, where n is the total
8609 number of rows in the triangle.
8610
8611 --
8612
8613 public class Solution {
8614     public int minimumTotal(List<List<Integer>> triangle) {
8615
8616         int n = triangle.size();
8617         if(n==0) return 0;
8618         if(n==1)
8619         {
8620             int min=Integer.MAX_VALUE;
8621             for(int i:triangle.get(0))
8622                 if(i<min) min = i;
8623             return min;
8624         }
8625
8626         List<Integer> prev = triangle.get(n-1);
8627         for(int i=n-2; i>=0; i--)
8628         {
8629             List<Integer> cur = triangle.get(i);
8630             for(int j=0;j<cur.size();j++)

```

```

8631         {
8632             cur.set(j,Math.min(prev.get(j),prev.get(j+1))+cur.get(j));
8633         }
8634         prev = cur;
8635     }
8636     return prev.get(0);
8637 }
8639 }
8640
8641
8642 =====
8643 /Users/linxie/Documents/github/leetcodeHub/code/TwoSum
8644 =====
8645
8646
8647 TwoSum
8648
8649 Given an array of integers, find two numbers such that they add up to a specific target number.
8650
8651 The function twoSum should return indices of the two numbers such that they add up to the target,
8652 where index1 must be less than index2. Please note that your returned answers (both index1 and
8653 index2) are not zero-based.
8654
8655 You may assume that each input would have exactly one solution.
8656
8657 Input: numbers={2, 7, 11, 15}, target=9
8658 Output: index1=1, index2=2
8659
8660 ---
8661
8662 public class Solution {
8663     public int[] twoSum(int[] numbers, int target) {
8664         int[] re = new int[2];
8665         int n = numbers.length;
8666         if(n<=1) return null;
8667
8668         HashMap<Integer, Integer>m = new HashMap<Integer, Integer>();
8669
8670         for(int i=0;i<n;i++)
8671         {
8672             int r = target - numbers[i];
8673             if(m.containsKey(r))
8674             {
8675                 re[0]=m.get(r)+1;
8676                 re[1]=i+1;
8677                 break;
8678             }
8679             m.put(numbers[i],i);
8680         }
8681         return re;
8682     }
8683 }
8684
8685
8686 =====
8687 /Users/linxie/Documents/github/leetcodeHub/code/UniqueBinarySearchTrees
8688 =====
8689
8690
8691 UniqueBinarySearchTrees
8692
8693 Given n, how many structurally unique BSTs (binary search trees) that store values 1...n?
8694
8695 For example,
8696 Given n = 3, there are a total of 5 unique BSTs.
8697
8698     1         3         3         2         1
8699     \       /       /         /   \
8700      3     2     1     1   3     2
8701     /   /   \           \
8702    2   1     2           3
8703
8704 ---
8705 public class Solution {
8706     public int numTrees(int l, int r)
8707     {
8708         if(l==r) return 1;
8709         if(l>r) return 0;
8710
8711         int re = 0;
8712         for(int i=l;i<=r;i++)
8713         {
8714             int left = numTrees(l,i-1);
8715             int right =numTrees(i+1, r);
8716             if(left == 0) re+=right;
8717             else if(right==0) re+=left;
8718             else re+=left*right;
8719         }
8720         return re;
8721     }
8722     public int numTrees(int n)
8723     {
8724         return numTrees(0,n-1);
8725     }
8726
8727 }
8728
8729
8730 =====
8731 /Users/linxie/Documents/github/leetcodeHub/code/UniqueBinarySearchTreesII
8732 =====
8733
8734

```

```

8735 UniqueBinarySearchTreesII
8736
8737 Given n, generate all structurally unique BSTs (binary search trees) that store values 1...n.
8738
8739 For example,
8740 Given n = 3, your program should return all 5 unique BSTs shown below.
8741
8742      1          3          3          2          1
8743      \          /          /          / \         \
8744      3          2          1          1   3        2
8745      /          /          \          \           \
8746      2          1          2          3           3
8747 ---
8748
8749 /**
8750 * Definition for binary tree
8751 * public class TreeNode {
8752 *     int val;
8753 *     TreeNode left;
8754 *     TreeNode right;
8755 *     TreeNode(int x) { val = x; left = null; right = null; }
8756 * }
8757 */
8758 public class Solution {
8759
8760     public static List<TreeNode> generateTrees(HashMap<Integer, List<TreeNode>> map, int l, int r, int n) {
8761         int key = l*n+r;
8762         if(map.containsKey(key)) return map.get(key);
8763
8764         List<TreeNode> re = new ArrayList<TreeNode>();
8765         if(l>r) return re;
8766         if(l==r)
8767         {
8768             TreeNode root = new TreeNode(l);
8769             re.add(root);
8770         }
8771         for(int i=l;i<=r;i++)
8772         {
8773             List<TreeNode> left = generateTrees(map, l, i-1, n);
8774             List<TreeNode> right = generateTrees(map, i+1, r, n);
8775             if(left.size() == 0)
8776             {
8777                 for(TreeNode nd : right)
8778                 {
8779                     TreeNode root = new TreeNode(i);
8780                     root.right = nd;
8781                     re.add(root);
8782                 }
8783             }
8784             else if(right.size() == 0)
8785             {
8786                 for(TreeNode nd: left)
8787                 {
8788                     TreeNode root = new TreeNode(i);
8789                     root.left = nd;
8790                     re.add(root);
8791                 }
8792             }
8793             else
8794             {
8795                 for(TreeNode lnd:left)
8796                 {
8797                     for(TreeNode rnd:right)
8798                     {
8799                         TreeNode root = new TreeNode(i);
8800                         root.left = lnd;
8801                         root.right = rnd;
8802                         re.add(root);
8803                     }
8804                 }
8805             }
8806         }
8807         map.put(key, re);
8808         return re;
8809     }
8810     public static List<TreeNode> generateTrees(int n) {
8811         List<TreeNode> re = new ArrayList<TreeNode>();
8812         if(n==0){}
8813         re.add(null);
8814         return re;
8815     }
8816     if(n==1)
8817     {
8818         TreeNode root = new TreeNode(1);
8819         re.add(root);
8820         return re;
8821     }
8822
8823
8824     return generateTrees(new HashMap<Integer, List<TreeNode>>(), 1, n,n);
8825
8826 }
8827 }
8828
8829 =====
8830 =====
8831 /Users/linxie/Documents/github/leetcodeHub/code/UniquePaths
8832 =====
8833
8834
8835 UniquePaths
8836
8837 A robot is located at the top-left corner of a m x n grid (marked Start in the diagram below).
8838

```

```

8839 The robot can only move either down or right at any point in time. The robot is trying to reach
8840 the bottom-right corner of the grid (marked Finish in the diagram below).
8841
8842 How many possible unique paths are there?
8843
8844 ---
8845
8846 public class Solution {
8847     public int uniquePaths(int m, int n) {
8848         if(m*n==0) return 0;
8849         int[] re = new int[n];
8850         for(int i =0;i<n;i++)
8851             re[i] = 1;
8852         for(int i=1;i<m;i++)
8853         {
8854             for(int j=0;j<n;j++)
8855             {
8856                 if(j==0) re[j] = 1;
8857                 else re[j]+=re[j-1];
8858             }
8859         }
8860         return re[n-1];
8861     }
8862 }
8863 }
8864
8865
8866 =====
8867 /Users/linxie/Documents/github/leetcodeHub/code/UniquePathsII
8868 =====
8869
8870
8871 UniquePathsII
8872
8873 Follow up for "Unique Paths":
8874
8875 Now consider if some obstacles are added to the grids. How many unique paths would there be?
8876
8877 An obstacle and empty space is marked as 1 and 0 respectively in the grid.
8878
8879 For example,
8880 There is one obstacle in the middle of a 3x3 grid as illustrated below.
8881
8882 [
8883     [0,0,0],
8884     [0,1,0],
8885     [0,0,0]
8886 ]
8887 The total number of unique paths is 2.
8888
8889 ---
8890
8891 public class Solution {
8892     public int uniquePathsWithObstacles(int[][] obstacleGrid) {
8893         int m = obstacleGrid.length;
8894         int n = obstacleGrid[0].length;
8895         if(m*n==0) return 0;
8896         int[] re = new int[n];
8897         boolean row = false, col = false;
8898         for(int i =0;i<n;i++)
8899         {
8900             if(obstacleGrid[0][i]==1) row = true;
8901             if(row) re[i] = 0;
8902             else re[i] = 1;
8903         }
8904         for(int i=1;i<m;i++)
8905         {
8906             for(int j=0;j<n;j++)
8907             {
8908                 if(obstacleGrid[i][j]==1)
8909                 {
8910                     re[j] = 0;
8911                     continue;
8912                 }
8913                 if(j==0) continue;
8914                 else re[j]+=re[j-1];
8915             }
8916         }
8917         return re[n-1];
8918     }
8919 }
8920 }
8921 }
8922
8923
8924 =====
8925 /Users/linxie/Documents/github/leetcodeHub/code/ValidateBinarySearchTree
8926 =====
8927
8928
8929 ValidateBinarySearchTree
8930
8931 Given a binary tree, determine if it is a valid binary search tree (BST).
8932
8933 Assume a BST is defined as follows:
8934
8935 The left subtree of a node contains only nodes with keys less than the nodes key.
8936 The right subtree of a node contains only nodes with keys greater than the nodes key.
8937 Both the left and right subtrees must also be binary search trees.
8938 ---
8939 /**
8940 * Definition for binary tree
8941 * public class TreeNode {
8942 *     int val;

```

```

8943 *      TreeNode left;
8944 *      TreeNode right;
8945 *      TreeNode(int x) { val = x; }
8946 *
8947 */
8948 public class Solution {
8949     public static boolean isValidBST(TreeNode root, int lower, int upper)
8950     {
8951         if(root == null) return true;
8952         if(root.val >= upper || root.val <= lower) return false;
8953         return isValidBST(root.left, lower, root.val) && isValidBST(root.right, root.val, upper);
8954     }
8955 }
8956
8957 public static boolean isValidBST(TreeNode root)
8958 {
8959     if(root == null) return true;
8960     return isValidBST(root.left, Integer.MIN_VALUE, root.val) && isValidBST(root.right, root.val, Integer.MAX_VALUE);
8961 }
8962
8963
8964 =====
8965 /Users/linxie/Documents/github/leetcodeHub/code/ValidNumber
8966 =====
8967
8968
8969 ValidNumber
8970
8971 Validate if a given string is numeric.
8972
8973 Some examples:
8974 "0" => true
8975 " 0.1 " => true
8976 "abc" => false
8977 "1 a" => false
8978 "2e10" => true
8979
8980 ---
8981
8982 public class Solution {
8983     public boolean isNumber(String s) {
8984
8985         boolean number = false, e = false, dot = false, sign = false;
8986
8987         int i=0;
8988         while(i<s.length() && s.charAt(i) == ' ') i++;
8989         if(i==s.length()) return false;
8990
8991         int j=s.length()-1;
8992         while(j>=0 && s.charAt(j) == ' ') j--;
8993         if(j<0) return false;
8994
8995         for( i=j; i++ )
8996         {
8997             char c = s.charAt(i);
8998             if(c == '+' || c=='-')
8999             {
9000                 if(sign || number) return false;
9001                 sign = true;
9002             }
9003             else if(c == 'e')
9004             {
9005                 if(e || !number) return false;
9006                 e = true;
9007                 number = false;
9008                 sign = false;
9009             }else if(c == '.')
9010             {
9011                 if(dot || e) return false;
9012                 dot = true;
9013                 if(!sign) sign = true;
9014             }
9015             }else if(c>=0 && c<=9)
9016             {
9017                 if(!number) number = true;
9018                 if(!sign) sign = true;
9019             }
9020             else{
9021                 return false;
9022             }
9023         }
9024         return number;
9025
9026
9027     }
9028 }
9029
9030
9031 =====
9032 /Users/linxie/Documents/github/leetcodeHub/code/ValidPalindrome
9033 =====
9034
9035
9036 ValidPalindrome
9037
9038 Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.
9039
9040 For example,
9041 "A man, a plan, a canal: Panama" is a palindrome.
9042 "race a car" is not a palindrome.
9043
9044 Note:
9045 Have you consider that the string might be empty? This is a good question to ask during an interview.
9046

```

```

9047 For the purpose of this problem, we define empty string as valid palindrome.
9048
9049 ---
9050 public class Solution {
9051     public int valid(char c){
9052         if(c == ' ') return -1;
9053
9054         int key = c-a;
9055         if(key>=0 && key<=25) return key;
9056         key = c-A;
9057         if(key>=0 && key<=25) return key;
9058         key = c-0;
9059         if(key>=0 && key<=9)
9060         {
9061             key+=30;
9062             return key;
9063         }
9064         return -1;
9065     }
9066
9067     public boolean isPalindrome(String s) {
9068         int n = s.length();
9069         if(n<=1) return true;
9070
9071         char[] c = s.toCharArray();
9072         int l = 0, r = n-1;
9073         while(l<=r)
9074         {
9075             int first = -1;
9076             while(l<=r)
9077             {
9078                 first = valid(c[l]);
9079                 if( first == -1) l++;
9080                 else break;
9081             }
9082             if(l>r) break;
9083
9084             int second = -1;
9085             while(l<=r)
9086             {
9087                 second = valid(c[r]);
9088                 if(second == -1) r--;
9089                 else break;
9090             }
9091             if(l>r) break;
9092
9093             if(first!=second) return false;
9094             l++;
9095             r--;
9096         }
9097         return true;
9098     }
9099 }
9100
9101
9102 =====
9103 /Users/linxie/Documents/github/leetcodeHub/code/ValidParentheses
9104 =====
9105
9106
9107 ValidParentheses
9108
9109 Given a string containing just the characters (' ', '{', '}', '[ and ], determine if the input string is valid.
9110
9111 The brackets must close in the correct order, "()" and "{}[]{}" are all valid but "[]" and "{[}]" are not.
9112
9113 ---
9114
9115 public class Solution {
9116     public boolean isValid(String s) {
9117
9118         int n = s.length();
9119         if(n==0) return true;
9120         if(n%2 == 1) return false;
9121
9122         char[] c = s.toCharArray();
9123         Stack<Character> st = new Stack<Character>();
9124         for(char cc :c)
9125         {
9126             if(cc==' ' || cc=={ || cc==[) st.push(cc);
9127             else if(cc == ' ') && (st.isEmpty() || st.pop()!=()) return false;
9128             else if(cc == }) && (st.isEmpty() || st.pop()!={) return false;
9129             else if(cc == ]) && (st.isEmpty() || st.pop()!=[) return false;
9130         }
9131         return st.isEmpty();
9132
9133     }
9134 }
9135
9136
9137 =====
9138 /Users/linxie/Documents/github/leetcodeHub/code/ValidSudoku
9139 =====
9140
9141
9142 ValidSudoku
9143
9144 Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.
9145
9146 The Sudoku board could be partially filled, where empty cells are filled with the character ..
9147
9148 ---
9149
9150 public class Solution {

```

```

9151     public boolean isValidSudoku(char[][] board) {
9152         int n = board.length;
9153         boolean[] rec = new boolean[9];
9154
9155         for(int i=0;i<n;i++)
9156         {
9157             for(int j=0;j<9;j++)
9158                 rec[j] = false;
9159
9160             for(int j =0; j< n; j++)
9161             {
9162                 if(board[i][j] == '.') continue;
9163                 int cur = (int)(board[i][j]-0);
9164                 if( cur<1 || cur >9 || rec[cur-1] ) return false;
9165                 else rec[cur-1] = true;
9166             }
9167         }
9168     }
9169
9170     for(int i=0;i<n;i++)
9171     {
9172         for(int j=0;j<9;j++)
9173             rec[j] = false;
9174
9175         for(int j =0; j< n; j++)
9176         {
9177             if(board[j][i] == '.') continue;
9178             int cur = (int)(board[j][i]-0);
9179             if( cur<1 || cur >9 || rec[cur-1] ) return false;
9180             else rec[cur-1] = true;
9181         }
9182     }
9183
9184     for(int i=0; i<3;i++)
9185     {
9186         for(int j=0; j<3; j++)
9187         {
9188             for(int k=0;k<9;k++)
9189                 rec[k] = false;
9190             for(int k=0;k<3;k++)
9191             {
9192                 for(int v =0; v<3; v++)
9193                 {
9194                     int ii = i*3+k;
9195                     int jj = j*3+v;
9196                     if(board[ii][jj] == '.') continue;
9197                     int cur = (int)(board[ii][jj]-0);
9198                     if( cur<1 || cur >9 || rec[cur-1] ) return false;
9199                     else rec[cur-1] = true;
9200                 }
9201             }
9202         }
9203     }
9204     return true;
9205 }
9206 }
9207
9208 =====
9209 /Users/linxie/Documents/github/leetcodeHub/code/WildcardMatching
9210 =====
9211 =====
9212
9213
9214 WildcardMatching
9215
9216 Implement wildcard pattern matching with support for ? and *.
9217
9218 ? Matches any single character.
9219 * Matches any sequence of characters (including the empty sequence).
9220
9221 The matching should cover the entire input string (not partial).
9222
9223 The function prototype should be:
9224 bool isMatch(const char *s, const char *p)
9225
9226 Some examples:
9227 isMatch("aa","a") → false
9228 isMatch("aa","aa") → true
9229 isMatch("aaa","aa") → false
9230 isMatch("aa", "*") → true
9231 isMatch("aa", "a*") → true
9232 isMatch("ab", "?*") → true
9233 isMatch("aab", "c*a*b") → false
9234
9235 ---
9236
9237 public class Solution {
9238     public boolean isMatch(String s, String p) {
9239         char[] a = s.toCharArray(), b = p.toCharArray();
9240         int m = a.length, n = b.length;
9241         if(n==0) return m==0;
9242
9243         int i=0, j=0;
9244         int star = -1, ii = -1;
9245         while(i<m)
9246         {
9247             if(j<n && (a[i]==b[j] || b[j]==?))
9248             {
9249                 i++; j++;
9250             }
9251             else if(j<n && b[j]==*)
9252             {
9253                 star = j;

```

```

9254         j++;
9255         ii = i+1;
9256     }
9257     else if(star!=-1)
9258     {
9259         i = ii++;
9260         j=star+1;
9261     }
9262     else
9263     {
9264         return false;
9265     }
9266 }
9267 while(j<n && b[j]==*) j++;
9268 return j==n;
9269 }
9270 }
9271 }
9272 =====
9273 /Users/linxie/Documents/github/leetcodeHub/code/WordBreak
9274 =====
9275 WordBreak
9276 =====
9277 Given a string s and a dictionary of words dict, determine if s can be segmented into a
9278 space-separated sequence of one or more dictionary words.
9279
9280 For example, given
9281 s = "leetcode",
9282 dict = ["leet", "code"].
9283
9284 Return true because "leetcode" can be segmented as "leet code".
9285 -----
9286 public class Solution {
9287     public boolean wordBreak(String s, Set<String> dict, int[] map)
9288     {
9289         int n = s.length();
9290         if(n==0) return true;
9291         if(map[n-1]!=0) return map[n-1]==1;
9292
9293         for(int i=n-1; i>0;i--)
9294         {
9295             if(dict.contains(s.substring(i)) && wordBreak(s.substring(0,i), dict, map)) return true;
9296         }
9297         map[n-1] = -1;
9298         return false;
9299     }
9300 }
9301
9302 public boolean wordBreak(String s, Set<String> dict) {
9303     int n = s.length();
9304     if(n==0) return false;
9305
9306     int[] map = new int[n];
9307     for(int i=0;i<n;i++)
9308     {
9309         if(dict.contains(s.substring(0,i+1)))
9310             map[i] = 1;
9311     }
9312     return wordBreak(s,dict,map);
9313 }
9314
9315 }
9316
9317 }
9318 }
9319
9320 =====
9321 /Users/linxie/Documents/github/leetcodeHub/code/WordBreakII
9322 =====
9323 WordBreakII
9324
9325 Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where
9326 each word is a valid dictionary word.
9327
9328 Return all such possible sentences.
9329
9330 For example, given
9331 s = "catsanddog",
9332 dict = ["cat", "cats", "and", "sand", "dog"].
9333
9334 A solution is ["cats and dog", "cat sand dog"].
9335
9336 ---
9337 public class Solution {
9338
9339     public List<String> wordBreak(String s, HashMap<Integer, List<String>> map, Set<String> dict) {
9340         List<String> re = new ArrayList<String>();
9341         int n = s.length();
9342         if(n==0) return re;
9343         if(map.containsKey(n)) return map.get(n);
9344
9345         for(int i=n-1; i>=0; i--)
9346         {
9347             if(dict.contains(s.substring(i)))
9348             {
9349                 if(i==0)
9350                 {
9351                     re.add(s);
9352                 }
9353                 List<String> r = wordBreak(s.substring(0,i), map, dict);
9354                 for(String list : r)
9355                 {
9356                     re.add(list + " " + s.substring(i+1));
9357                 }
9358             }
9359         }
9360         map.put(n, re);
9361     }
9362 }
```

```

9358         {
9359             re.add(list+" "+s.substring(i));
9360         }
9361     }
9362 }
9363 if(re.size()>0) map.put(n, re);
9364 return re;
9365 }
9366 public List<String> wordBreak(String s, Set<String> dict) {
9367     HashMap<Integer, List<String>> map = new HashMap<Integer, List<String>>();
9368     return wordBreak(s, map, dict);
9369 }
9370 }
9371
9372 =====
9373 /Users/linxie/Documents/github/leetcodeHub/code/WordLadder
9375 =====
9376
9377
9378 WordLadder
9379
9380 Given two words (start and end), and a dictionary, find the length of shortest transformation
9381 sequence from start to end, such that:
9382
9383 Only one letter can be changed at a time
9384 Each intermediate word must exist in the dictionary
9385 For example,
9386
9387 Given:
9388 start = "hit"
9389 end = "cog"
9390 dict = ["hot", "dot", "dog", "lot", "log"]
9391 As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
9392 return its length 5.
9393
9394 ---
9395 public class Solution {
9396     public int ladderLength(String start, String end, Set<String> dict) {
9397
9398         int n = start.length();
9399         if(end.length()!=n) return 0;
9400         if(start.equals(end)) return 1;
9401
9402         HashSet<String> cur = new HashSet<String>();
9403         cur.add(start);
9404         dict.remove(start);
9405         boolean found = false;
9406         int re = 1;
9407
9408         while(!found)
9409         {
9410             re++;
9411             if(cur.size()==0) return 0;
9412             HashSet<String> next = new HashSet<String>();
9413             for(String s:cur)
9414             {
9415                 char[] c = s.toCharArray();
9416                 for(int i=0;i<n;i++)
9417                 {
9418                     char val = c[i];
9419                     for(char cc = a;cc<=z;cc++)
9420                     {
9421                         c[i] = cc;
9422                         String str = new String(c);
9423                         if(str.equals(end)) return re;
9424                         if(dict.contains(str) && !next.contains(str))
9425                         {
9426                             next.add(str);
9427                         }
9428                     }
9429                     c[i] = val;
9430                 }
9431             }
9432             for(String ss:next)
9433             {
9434                 dict.remove(ss);
9435             }
9436             cur = next;
9437         }
9438         return re;
9439
9440     }
9441 }
9442
9443
9444 =====
9445 /Users/linxie/Documents/github/leetcodeHub/code/WordLadderII
9446 =====
9447
9448
9449 WordLadderII
9450
9451 Given two words (start and end), and a dictionary, find all shortest transformation sequence(s)
9452 from start to end, such that:
9453
9454 Only one letter can be changed at a time
9455 Each intermediate word must exist in the dictionary
9456 For example,
9457
9458 Given:
9459 start = "hit"
9460 end = "cog"
9461 dict = ["hot", "dot", "dog", "lot", "log"]

```

```

9462 Return
9463   [
9464     ["hit", "hot", "dot", "dog", "cog"],
9465     ["hit", "hot", "lot", "log", "cog"]
9466   ]
9467 Note:
9468 All words have the same length.
9469 All words contain only lowercase alphabetic characters.
9470 ---
9471
9472 class Word
9473 {
9474   Set<String> dict;
9475   ArrayList<HashSet<Character>> letters;
9476   HashMap<String, HashSet<String>> map = new HashMap<String, HashSet<String>>();
9477
9478 Word(Set<String> _dict, ArrayList<HashSet<Character>> _letters, String name)
9479 {
9480   dict = new HashSet<String>(_dict);
9481   dict.remove(name);
9482   letters = _letters;
9483 }
9484
9485 public HashSet<String> getList(HashSet<String> list)
9486 {
9487   HashSet<String> re = new HashSet<String>();
9488
9489   for(String w : list)
9490   {
9491     char[] ss = w.toCharArray();
9492
9493     for(int i=0;i<ss.length; i++)
9494     {
9495       char tmp = ss[i];
9496       for(char c : letters.get(i))
9497       {
9498         if(c == tmp) continue;
9499         ss[i] = c;
9500
9501         String str = new String(ss);
9502
9503         if(dict.contains(str))
9504         {
9505           re.add(str);
9506           if(map.containsKey(str))
9507           {
9508             HashSet<String> prev = map.get(str);
9509             if(!prev.contains(w))
9510             {
9511               prev.add(w);
9512               map.put(str, prev);
9513             }
9514           }
9515           else
9516           {
9517             HashSet<String> prev = new HashSet<String>();
9518             prev.add(w);
9519             map.put(str, prev);
9520           }
9521           ss[i] = tmp;
9522         }
9523     }
9524     for(String ss : re)
9525       dict.remove(ss);
9526   }
9527 }
9528
9529 List<List<String>> getResult(String end, boolean flg)
9530 {
9531   List<List<String>> re = new ArrayList<List<String>>();
9532   if(!map.containsKey(end))
9533   {
9534     List<String> result = new ArrayList<String>();
9535     result.add(end);
9536     re.add(result);
9537     return re;
9538   }
9539
9540
9541   for(String s:map.get(end))
9542   {
9543     List<List<String>> tmp = getResult(s, flg);
9544     Iterator<List<String>> it = tmp.iterator();
9545     while(it.hasNext())
9546     {
9547       re.add(it.next());
9548     }
9549   }
9550   Iterator<List<String>> it = re.iterator();
9551   while(it.hasNext())
9552   {
9553     if(flg) it.next().add(end);
9554     else it.next().add(0, end);
9555   }
9556   return re;
9557 }
9558 }
9559
9560
9561 public class Solution {
9562
9563   public List<List<String>> findLadders(String start, String end, Set<String> dict) {
9564     List<List<String>> re = new LinkedList<List<String>>();
9565     int n = start.length();

```

```

9566     if(n!=end.length()) return re;
9567
9568     dict.add(end);
9569
9570     boolean found = false;
9571
9572     ArrayList<HashSet<Character>> letters = new ArrayList<HashSet<Character>>();
9573     for(int i = 0; i<n;i++)
9574     {
9575         letters.add(new HashSet<Character>());
9576     }
9577
9578     for(String s:dict)
9579     {
9580         if(s.length()!=n) continue;
9581
9582         for(int i=0;i<n;i++)
9583         {
9584             if(!letters.get(i).contains(s.charAt(i)))
9585                 letters.get(i).add(s.charAt(i));
9586         }
9587     }
9588
9589     Word forward = new Word(dict, letters, start);
9590     Word backward = new Word(dict, letters, end);
9591
9592     HashSet<String> forward_list = new HashSet<String>();
9593     HashSet<String> backward_list = new HashSet<String>();
9594
9595     HashSet<String> sub_result = new HashSet<String>();
9596
9597     forward_list.add(start);
9598     backward_list.add(end);
9599
9600     while(!found)
9601     {
9602         forward_list = forward.getList(forward_list);
9603         if(forward_list.isEmpty())
9604             break;
9605         for(String ss : forward_list)
9606         {
9607             if(backward_list.contains(ss))
9608             {
9609                 sub_result.add(ss);
9610                 found = true;
9611             }
9612         }
9613         if(found) break;
9614         backward_list = backward.getList(backward_list);
9615         if(backward_list.isEmpty())
9616             break;
9617
9618         for(String ss : forward_list)
9619         {
9620             if(backward_list.contains(ss))
9621             {
9622                 sub_result.add(ss);
9623                 found = true;
9624             }
9625         }
9626     }
9627     if(sub_result.isEmpty()) return re;
9628
9629     for(String s : sub_result)
9630     {
9631         List<List<String>> first_half = forward.getResult(s,true);
9632         List<List<String>> back_half = backward.getResult(s,false);
9633
9634         for(List<String> l : first_half){
9635             for(List<String> ll : back_half)
9636             {
9637                 ll.remove(s);
9638                 List<String> l_tmp = new ArrayList<String>(l);
9639                 l_tmp.addAll(ll);
9640                 re.add(l_tmp);
9641             }
9642         }
9643     }
9644     return re;
9645
9646 }
9647
9648 }
9649
9650
9651 =====
9652 /Users/linxie/Documents/github/leetcodeHub/code/WordSearch
9653 =====
9654
9655
9656 WordSearch
9657
9658 Given a 2D board and a word, find if the word exists in the grid.
9659
9660 The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are
9661 those horizontally or vertically neighboring. The same letter cell may not be used more than once.
9662
9663 For example,
9664 Given board =
9665
9666 [
9667     [ "ABCE" ],
9668     [ "SFCS" ],
9669     [ "ADEE" ]

```

```

9670 ]
9671 word = "ABCED", -> returns true,
9672 word = "SEE", -> returns true,
9673 word = "ABCB", -> returns false.
9674
9675 --
9676 public class Solution {
9677
9678     public boolean exist(char[][] board, String word, int i, int j, boolean[][] map)
9679     {
9680         if(word.length() == 0) return true;
9681         if(i-1>=0 && !map[i-1][j])
9682         {
9683             if(word.charAt(0) == board[i-1][j])
9684             {
9685                 map[i-1][j] = true;
9686                 if(exist(board, word.substring(1), i-1, j, map))
9687                     return true;
9688                 map[i-1][j] = false;
9689             }
9690         }
9691
9692         if(j-1>=0 && !map[i][j-1])
9693         {
9694             if(word.charAt(0) == board[i][j-1])
9695             {
9696                 map[i][j-1] = true;
9697                 if(exist(board, word.substring(1), i, j-1, map))
9698                     return true;
9699                 map[i][j-1] = false;
9700             }
9701         }
9702
9703         if(i+1<board.length && !map[i+1][j])
9704         {
9705             if(word.charAt(0) == board[i+1][j])
9706             {
9707                 map[i+1][j] = true;
9708                 if(exist(board, word.substring(1), i+1, j, map))
9709                     return true;
9710                 map[i+1][j] = false;
9711             }
9712         }
9713
9714         if(j+1<board[0].length && !map[i][j+1])
9715         {
9716             if(word.charAt(0) == board[i][j+1])
9717             {
9718                 map[i][j+1] = true;
9719                 if(exist(board, word.substring(1), i, j+1, map))
9720                     return true;
9721                 map[i][j+1] = false;
9722             }
9723         }
9724     return false;
9725 }
9726
9727     public boolean exist(char[][] board, String word) {
9728         int m = board.length;
9729         if(m==0) return false;
9730         int n = board[0].length;
9731         if(n==0) return false;
9732         if(word.length() == 0 ) return true;
9733
9734         boolean[][] map = new boolean[m][n];
9735
9736         for(int i=0;i<m;i++)
9737             for(int j = 0; j<n; j++)
9738                 if( board[i][j] == word.charAt(0))
9739                 {
9740                     map[i][j] = true;
9741                     if(exist(board, word.substring(1), i, j, map))
9742                         return true;
9743                     map[i][j] = false;
9744                 }
9745     return false;
9746 }
9747 }
9748
9749
9750 =====
9751 /Users/linxie/Documents/github/leetcodeHub/code/youngTableKMAX
9752 =====
9753
9754
9755 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
9756
9757 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
9758
9759
9760
9761
9762 class Pair implements Comparable
9763 {
9764     int val, i, j;
9765     Pair(int _i, int _j, int ai, int bj)
9766     {
9767         val = ai+bj;
9768         i = _i; j = _j;
9769     }
9770
9771     public int compareTo(Object o)
9772     {
9773         Pair p = (Pair)o;

```

```
9774     return val-p.val;
9775 }
9776 }
9777
9778 class Solution
9779 {
9780     public List<Integer> youngTableKMAX(int[] a, int[] b, int k)
9781     {
9782         List<Integer> re = new ArrayList<Integer>();
9783
9784         int m = a.length, n = b.length;
9785         if(k<=0 || k>=m*n) return re;
9786
9787         if(k==1)
9788         {
9789             re.add(a[0]+b[0]);
9790             return re;
9791         }
9792
9793         PriorityQueue<Pair> q = new PriorityQueue<Pair>();
9794         for(int i=0;i<n;i++)
9795         {
9796             q.add(new Pair(0,i,a[0], b[i]));
9797         }
9798         int count = 0;
9799         while(!q.isEmpty() && count<=k)
9800         {
9801             Pair cur = q.poll();
9802             if(cur == null) break;
9803             re.add(cur.val);
9804
9805             cur.i++;
9806             if(cur.i == m) continue;
9807
9808             q.add(new Pair(cur.i, cur.j, a[cur.i], b[cur.j]));
9809         }
9810         return re;
9811     }
9812 }
```