1. **线性表**

1.1 Remove Duplicates from Sorted Array

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

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

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

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

题解：

public class Solution {

public int removeDuplicates(int[] A) {

int len = A.length;

if(len<=0) return 0;

int pos = 0;

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

{

if(A[i]!=A[pos])

{

pos ++;

A[pos] = A[i];

}

}

return pos+1;

}

}

### 1.2 Remove Duplicates from Sorted Array II

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

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

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

题解：

public class Solution {

public int removeDuplicates(int[] A) {

int len =A.length;

if(len<=0) return 0;

int pos = 0,count=1;

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

{

if(A[pos]!=A[i])

{

pos ++;

A[pos] = A[i];

count = 1;

}else{

if(count>=2)

{

continue;

}else{

pos ++;

A[pos] = A[i];

count ++;

}

}

}

return pos+1;

}

}

### 1.3 Search in Rotated Sorted Array

 Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

题解：

public class Solution {

public int search(int[] A, int target) {

int len = A.length;

if(len<=0) return -1;

int start = 0,end = len-1,middle=0;

while(start<=end)

{

middle = (start+end)/2;

if(A[middle]==target)

return middle;

if(A[start]<=A[middle])

{

if(target>=A[start] && target<=A[middle])

end = middle;

else

start = middle+1;

}else{

if(target>=A[middle] && target<=A[end])

start = middle;

else

end = middle -1;

}

}

return -1;

}

}

* 1. Search in Rotated Sorted Array II

Follow up for "Search in Rotated Sorted Array":  
What if *duplicates* are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

题解：

public class Solution {

public boolean search(int[] A, int target) {

int len = A.length;

if(len<=0) return false;

int start = 0,end = len-1,middle=0;

while(start<=end)

{

middle = (start+end)/2;

if(A[middle]==target)

return true;

if(A[start]<A[middle])

{

if(target>=A[start] && target<=A[middle])

end = middle;

else

start = middle+1;

}else if(A[start]>A[middle]){

if(target>=A[middle] && target<=A[end])

start = middle;

else

end = middle -1;

}else{

start ++; //只能顺序查找了

}

}

return false;

}

}

### 1.5 Median of Two Sorted Arrays

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

题解：

### 1.6 Longest Consecutive Sequence

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

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

Your algorithm should run in O(*n*) complexity.

题解：

public class Solution {

public int longestConsecutive(int[] num) {

int len = num.length;

if(len<=0) return 0;

HashSet<Integer> set = new HashSet<Integer>();

int longest = 1;

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

set.add(num[i]);

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

{

int temp = num[i],max = 1;

int j=1;

set.remove(temp);

while(set.contains(temp-j))

{

set.remove(temp-j);

max ++;

j ++;

}

j = 1;

while(set.contains(temp+j))

{

set.remove(temp+j);

max ++;

j ++;

}

if(max>longest)

longest = max;

}

return longest;

}

}

1.7 Two Sum

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

**Input:** numbers={2, 7, 11, 15}, target=9  
**Output:** index1=1, index2=2

题解：

public class Solution {

public int[] twoSum(int[] numbers, int target) {

int[] result = new int[2];

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

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

{

map.put(numbers[i],i+1);

}

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

{

if(map.containsKey(target-numbers[i])&&map.get(target-numbers[i])!=(i+1))

{

result[0] = (i+1)>map.get(target-numbers[i])?map.get(target-numbers[i]):(i+1);

result[1] = (i+1)<map.get(target-numbers[i])?map.get(target-numbers[i]):(i+1);

break;

}

}

return result;

}

}

### 1.8 3Sum

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

**Note:**

* Elements in a triplet (*a*,*b*,*c*) must be in non-descending order. (ie, *a* ≤ *b* ≤ *c*)
* The solution set must not contain duplicate triplets.

For example, given array S = {-1 0 1 2 -1 -4},

A solution set is:

(-1, 0, 1)

(-1, -1, 2)

题解：

public class Solution {

public List<List<Integer>> threeSum(int[] num) {

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

if(num.length<3) return result;

Arrays.sort(num);

for(int i=0;i<num.length-2;i++)

{

if(num[i]>0) break;

int s = i+1,e=num.length-1;

if(num[i]+num[s]>0) break;

while(s<e)

{

if(s!=i+1 && num[s]==num[s-1])

{

s ++;

continue; //防止对于重复的进行再一次程序运行，很重要

}

if((num[i]+num[s]+num[e])==0)

{

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

temp.add(num[i]);

temp.add(num[s]);

temp.add(num[e]);

if(!result.contains(temp))

result.add(temp);

s ++;

e --;

}else if((num[i]+num[s]+num[e])>0){

e --;

}else{

s ++;

}

}

}

return result;

}

}

### 1.9 3Sum Closest

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

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

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

题解：

public class Solution {

public int threeSumClosest(int[] num, int target) {

int min = Integer.MAX\_VALUE;

int result = 0;

Arrays.sort(num);

for(int i=0;i<num.length-2;i++)

{

int s = i+1,e=num.length-1;

while(s<e)

{

if(Math.abs((num[i]+num[s]+num[e])-target)<min)

{

min = Math.abs((num[i]+num[s]+num[e])-target);

result = num[i]+num[s]+num[e];

}

if(num[i]+num[s]+num[e]>target) e--;

else s++;

}

}

return result;

}

}

1.10 4Sum

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

**Note:**

* Elements in a quadruplet (*a*,*b*,*c*,*d*) must be in non-descending order. (ie, *a* ≤ *b* ≤ *c* ≤ *d*)
* The solution set must not contain duplicate quadruplets.

For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:

(-1, 0, 0, 1)

(-2, -1, 1, 2)

(-2, 0, 0, 2)

题解：

public class Solution {

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

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

if(num.length<4) return result;

Arrays.sort(num);

for(int j=0;j<num.length-3;j++)

{

for(int i=j+1;i<num.length-2;i++)

{

int s = i+1,e=num.length-1;

while(s<e)

{

if((num[j]+num[i]+num[s]+num[e])==target)

{

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

temp.add(num[j]);

temp.add(num[i]);

temp.add(num[s]);

temp.add(num[e]);

if(!result.contains(temp))

result.add(temp);

s ++;

e --;

}else if((num[j]+num[i]+num[s]+num[e])>target){

e --;

}else{

s ++;

}

}

}

}

return result;

}

}

### 1.11 Remove Element

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

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

题解：

(1)public class Solution {

public int removeElement(int[] A, int elem) {

int left = A.length;

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

{

if(A[i]==elem)

{

while(A[left-1]==elem)

{

if(left-1 > i)

left --;

else

break;

}

if(left-1 != i)

A[i] = A[left-1];

left --;

}

}

return left;

}

}

(2)public class Solution {

public int removeElement(int[] A, int elem) {

int len = A.length ,index = 0;

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

{

if(A[i]!=elem)

{

A[index++] = A[i];

}

}

return index;

}

}

### 1.12 Next Permutation

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

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

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

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

题解：

public class Solution {

public int minindex(int[] A,int start,int end,int value)

{

int index=0;

for(int i=end;i>=start;i--)

{

if(A[i]>value)

{

index = i;

break;

}

}

return index;

}

public int[] sort(int[] A,int start,int end)

{

int[] temp = new int[end-start+1];

for(int i=start;i<=end;i++)

{

temp[i-start] = A[i];

}

Arrays.sort(temp);

return temp;

}

public boolean isreverse(int[] A)

{

boolean flag = true;

for(int i=0;i<A.length-1;i++)

{

if(A[i]<A[i+1])

{

flag = false;

break;

}

}

return flag;

}

public void nextPermutation(int[] num) {

if(isreverse(num))

{

Arrays.sort(num);

return ;

}

for(int i=num.length-2;i>=0;i--)

{

if(num[i]<num[i+1])

{

int index= minindex(num,i+1,num.length-1,num[i]);

int temp = num[i];

num[i] = num[index];

num[index] = temp;

int[] t = sort(num,i+1,num.length-1);

for(int j=i+1;j<num.length;j++)

num[j] = t[j-i-1];

break;

}

}

}

}

1.13 Permutation Sequence

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

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

1. "123"
2. "132"
3. "213"
4. "231"
5. "312"
6. "321"

Given *n* and *k*, return the *k*th permutation sequence.

**Note:** Given *n* will be between 1 and 9 inclusive.

题解：

public class Solution {

public int factorial(int n)

{

int result = 1;

for(int i=1;i<=n;i++)

result \*= i;

return result;

}

public String getPermutation(int n, int k) {

StringBuilder s = new StringBuilder();

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

for(int i=1;i<=n;i++)

lst.add(i);

int count = n;

k --; //编码是从0开始的。很重要

while(!lst.isEmpty())

{

int t = k/factorial(count-1);

s.append(lst.remove(t));

k = k%factorial(count-1);

count --;

}

return s.toString();

}

}

### 1.14 Valid Sudoku

 Determine if a Sudoku is valid, according to: [Sudoku Puzzles - The Rules](http://sudoku.com.au/TheRules.aspx).

The Sudoku board could be partially filled, where empty cells are filled with the character '.'.

题解：

public class Solution {

public boolean check(char[][] board,int a,int b)

{

for(int i=0;i<9;i++)

{

if(board[a][i]==board[a][b] && i!=b)

return false;

}

for(int i=0;i<9;i++)

{

if(board[i][b]==board[a][b] && i!=a)

return false;

}

for(int i=(a/3)\*3;i<=(a/3)\*3+2;i++)

{

for(int j=(b/3)\*3;j<=(b/3)\*3+2;j++)

{

if(board[i][j]==board[a][b]&&i!=a&&j!=b)

return false;

}

}

return true;

}

public boolean isValidSudoku(char[][] board) {

for(int i=0;i<9;i++)

{

for(int j=0;j<9;j++)

{

if(board[i][j]!='.')

{

if(check(board,i,j)==false)

return false;

}

}

}

return true;

}

}

### 1.15 Trapping Rain Water

 Given *n* non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example,   
Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.



The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

题解：

public class Solution {

public int trap(int[] A) {

int max = Integer.MIN\_VALUE,maxindex=-1;

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

{

if(A[i]>max)

{

max = A[i];

maxindex = i;

}

}

int result = 0;

for(int i=0,temp=0;i<maxindex;i++)

{

if(A[i]>temp) temp = A[i];

else result += temp-A[i];

}

for(int i=A.length-1,temp=0;i>maxindex;i--)

{

if(A[i]>temp) temp = A[i];

else result += temp-A[i];

}

return result;

}

}

### 1.16 Rotate Image

 You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up:  
Could you do this in-place?

题解：

public class Solution {

public void rotate(int[][] matrix) {

int len = matrix.length;

for(int i=0;i<len-1;i++)

{

for(int j=0;j<len-1-i;j++)

{

int temp = matrix[len-1-j][len-1-i];

matrix[len-1-j][len-1-i] = matrix[i][j];

matrix[i][j] = temp;

}

}

for(int i=0;i<=(len-1)/2;i++)

{

for(int j=0;j<len;j++)

{

int temp = matrix[i][j];

matrix[i][j] = matrix[len-1-i][j];

matrix[len-1-i][j] = temp;

}

}

}

}

1.17 Plus One

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

The digits are stored such that the most significant digit is at the head of the list.

题解：

public class Solution {

public int[] plusOne(int[] digits) {

int len = digits.length;

int[] temp = new int[len+1];

int a = digits[len-1] + 1;

int jin = a/10;

temp[len] = a%10;

for(int i=len-2;i>=0;i--)

{

a = digits[i]+jin;

jin = a/10;

temp[i+1] = a%10;

}

if(jin==0)

{

int[] result = new int[len];

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

result[i-1] = temp[i];

return result;

}else{

temp[0] = jin;

return temp;

}

}

}

1.18 Climbing Stairs

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

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

题解：

public class Solution {

public int climbStairs(int n) {

if(n<=1) return 1;

int[] result = new int[n+1];

result[0] = result[1] = 1;

for(int i=2;i<=n;i++)

{

result[i] = result[i-1] + result[i-2];

}

return result[n];

}

}

### 1.19 Gray Code

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

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

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

00 - 0

01 - 1

11 - 3

10 - 2

**Note:**  
For a given *n*, a gray code sequence is not uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

题解：

public class Solution {

public List<Integer> grayCode(int n) {

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

int count = (int)Math.pow(2,n);

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

{

int t = (i>>1)^i;

result.add(t);

}

return result;

}

}

1.20 Set Matrix Zeroes

Given a *m* x *n* matrix, if an element is 0, set its entire row and column to 0. Do it in place.

题解：

public class Solution {

public void setZeroes(int[][] matrix) {

List<Integer> lst1 = new ArrayList<Integer>();

List<Integer> lst2 = new ArrayList<Integer>();

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

{

for(int j=0;j<matrix[0].length;j++)

{

if(matrix[i][j]==0)

{

if(!lst1.contains(i))

lst1.add(i);

if(!lst2.contains(j))

lst2.add(j);

}

}

}

for(int i=0;i<lst1.size();i++)

{

int row = lst1.get(i);

for(int j=0;j<matrix[0].length;j++)

matrix[row][j] = 0;

}

for(int i=0;i<lst2.size();i++)

{

int col = lst2.get(i);

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

matrix[j][col] = 0;

}

}

}

### 1.21 Gas Station

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

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

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

**Note:**  
The solution is guaranteed to be unique.

题解：

public class Solution {

public int canCompleteCircuit(int[] gas, int[] cost) {

int n = gas.length;

int total=0,sum=0,index=-1;

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

{

sum += gas[i]-cost[i];

total += gas[i]-cost[i];

if(sum<0)

{

index = i;

sum = 0;

}

}

return total>=0?index+1:-1;

}

}

1.22 Candy

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

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

* Each child must have at least one candy.
* Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

题解：

public class Solution {

public int candy(int[] ratings) {

int[] left = new int[ratings.length];

int[] right = new int[ratings.length];

for(int i=1;i<ratings.length;i++)

{

if(ratings[i]>ratings[i-1])

left[i] = left[i-1] + 1;

}

for(int i=ratings.length-2;i>=0;i--)

{

if(ratings[i]>ratings[i+1])

right[i] = right[i+1] + 1;

}

int result = 0;

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

{

int temp = left[i]>right[i]?left[i]:right[i];

result += temp;

}

return result+ratings.length;

}

}

### 1.23 Single Number

 Given an array of integers, every element appears *twice* except for one. Find that single one.

**Note:**  
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

题解：

public class Solution {

public int singleNumber(int[] A) {

int result = A[0];

for(int i=1;i<A.length;i++)

result ^= A[i];

return result;

}

}

1.24 Single Number II

Given an array of integers, every element appears *three* times except for one. Find that single one.

**Note:**  
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

题解：

public class Solution {

public int singleNumber(int[] A) {

int result = 0;

for(int i=0;i<32;i++)

{

int t = 1<<i,total = 0;

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

{

if((A[j]&t) !=0)

total += 1;

}

if(total%3!=0)

result |= 1<<i;

}

return result;

}

}

1. **单链表**

### 2.1 Add Two Numbers

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

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

题解：

public class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

if(l1==null && l2==null) return null;

ListNode result = new ListNode(0);

ListNode node = result;

int jin = 0;

while(l1!=null || l2!=null)

{

int a = 0;

if(l1!=null)

{

a += l1.val;

l1 = l1.next;

}

if(l2!=null)

{

a += l2.val;

l2 = l2.next;

}

a += jin;

jin = a/10;

ListNode temp = new ListNode(a%10);

node.next = temp;

node = node.next;

}

if(jin!=0)

{

ListNode temp = new ListNode(jin);

node.next = temp;

}

return result.next;

}

}

### 2.2 Reverse Linked List II

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

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

return 1->4->3->2->5->NULL.

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

题解：

public class Solution {

public ListNode reverseBetween(ListNode head, int m, int n) {

ListNode start = null,cur = null,pre=null;

int count = 0;

ListNode head2 = new ListNode(-1);

head2.next = head;

ListNode node = head2;

for(count=0;count<m-1;count++)

node = node.next;

start = node;

pre = node.next;

cur = pre.next;

for(count=m+1;count<=n;count++)

{

pre.next = cur.next;

cur.next = start.next; //

start.next = cur;

cur = pre.next;

}

return head2.next;

}

}

### 2.3 Partition List

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

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

For example,  
Given 1->4->3->2->5->2 and *x* = 3,  
return 1->2->2->4->3->5.

题解：

public class Solution {

public ListNode partition(ListNode head, int x) {

ListNode r = new ListNode(-1);

ListNode node = r;

ListNode dummy = new ListNode(-1);

dummy.next = head;

ListNode pre = dummy;

while(head!=null)

{

if(head.val<x)

{

ListNode temp = head;

node.next = temp;

node = node.next;

pre.next = head.next;

head = pre.next;

}else{

pre =head;

head = head.next;

}

}

node.next = dummy.next;

return r.next;

}

}

### 2.4 Remove Duplicates from Sorted List

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

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

题解：

public class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head==null) return null;

ListNode node = head;

while(node!=null)

{

if(node.next==null)

break;

else{

if(node.val!=node.next.val)

{

node = node.next;

}else{

ListNode temp = node;

while(temp.val==node.val)

{

temp = temp.next;

if(temp==null) break;

}

node.next = temp;

node = temp;

}

}

}

return head;

}

}

### 2.5 Remove Duplicates from Sorted List II

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

For example,  
Given 1->2->3->3->4->4->5, return 1->2->5.  
Given 1->1->1->2->3, return 2->3.

题解：

public class Solution {

public ListNode deleteDuplicates(ListNode head) {

ListNode h = new ListNode(-1);

h.next = head;

ListNode node = h;

while(head!=null)

{

if(head.next==null)

{

node.next = head;

break;

}else{

if(head.val==head.next.val)

{

ListNode temp = head;

while(temp.val==head.val)

{

temp = temp.next;

if(temp==null) break;

}

head = temp;

node.next = head;

}else{

node.next = head;

node = node.next;

head = head.next;

}

}

}

return h.next;

}

}

### 2.6 Rotate List

 Given a list, rotate the list to the right by *k* places, where *k*is non-negative.

For example:  
Given 1->2->3->4->5->NULL and *k* = 2,  
return 4->5->1->2->3->NULL.

题解：

public class Solution {

public ListNode rotateRight(ListNode head, int k) {

if(head==null) return null;

int len = 0;

ListNode node = head;

while(node!=null)

{

len ++;

node=node.next;

}

int kk = k%len;

if(kk==0) return head;

node = head;

for(int i=0;i<len-kk-1;i++)

node = node.next;

ListNode h = node.next;

node.next =null;

ListNode temp = h;

while(temp.next!=null)

temp = temp.next;

temp.next = head;

return h;

}

}

2.7 Remove Nth Node From End of List

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

For example,

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

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

**Note:**  
Given *n* will always be valid.  
Try to do this in one pass.

题解：

public class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

ListNode h = new ListNode(-1);

h.next = head;

ListNode last = h;

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

last=last.next;

ListNode left = h;

while(last.next!=null)

{

last = last.next;

left = left.next;

}

left.next = left.next.next;

return h.next;

}

}

2.8 Swap Nodes in Pairs

Given a linked list, swap every two adjacent nodes and return its head.

For example,  
Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may **not** modify the values in the list, only nodes itself can be changed.

题解：

public class Solution {

public ListNode swapPairs(ListNode head) {

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

ListNode dummy = new ListNode(-1);

dummy.next = head;

ListNode s = dummy,cur=dummy.next,ne=cur.next;

while(cur!=null&&ne!=null)

{

cur.next= ne.next;

ne.next = s.next;

s.next = ne;

s = cur;

cur = cur.next;

if(cur!=null)

ne = cur.next;

else

break;

}

return dummy.next;

}

}

2.9 Reverse Nodes in k-Group

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

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

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

Only constant memory is allowed.

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

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

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

题解：

public class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

if(k==1 || head==null ||head.next==null) return head;

ListNode dummy = new ListNode(-1);

dummy.next= head;

ListNode s = dummy,cur = s.next,ne = cur.next;

boolean flag = true;

ListNode node = head;

int count = 0;

while(node!=null)

{

count ++;

node = node.next;

}

if(count<k) flag = false;

count = 1;

while(flag)

{

if(count<k){

cur.next = ne.next;

ne.next = s.next;

s.next = ne;

ne = cur.next;

count ++;

}else{

node = cur.next;

count = 0;

while(node!=null)

{

count ++;

node = node.next;

}

if(count<k) { flag = false;break;}

else{

count = 1;

s = cur;

cur = s.next;

ne = cur.next;

}

}

}

return dummy.next;

}

}

2.10 Copy List with Random Pointer

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

Return a deep copy of the list.

题解：

public class Solution {

public RandomListNode copyRandomList(RandomListNode head) {

if(head==null) return null;

RandomListNode node = head;

while(node!=null)

{

RandomListNode temp = new RandomListNode(node.label);

temp.next = node.next;

node.next = temp;

node = node.next.next;

}

RandomListNode cur = head,ne = cur.next;

while(cur!=null)

{

ne = cur.next;

if(cur.random!=null)

ne.random = cur.random.next;

else

ne.random = null;

cur = ne.next;

}

RandomListNode dummy = new RandomListNode(-1);

node = head;

cur = dummy;

while(node!=null)

{

cur.next = node.next;

cur = cur.next;

node.next = node.next.next;

node = node.next;

}

return dummy.next;

}

}

2.11 Linked List Cycle

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

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

题解：

public class Solution {

public boolean hasCycle(ListNode head) {

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

ListNode fast = head,slow = head;

int count = 0;

while(fast!=null)

{

if(fast==slow && count!=0) //防止单节点自环

return true;

if(fast.next==null) return false;

else

fast = fast.next.next;

slow = slow.next;

count ++;

}

return false;

}

}

2.12 Linked List Cycle II

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

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

题解：

public class Solution {

public boolean isCycle(ListNode head)

{

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

ListNode fast = head,slow = head;

int count = 0;

while(fast!=null)

{

if(fast==slow && count!=0) //防止单节点自环

return true;

if(fast.next==null) return false;

else

fast = fast.next.next;

slow = slow.next;

count ++;

}

return false;

}

public ListNode detectCycle(ListNode head) {

if(isCycle(head)==false) return null;

ListNode fast = head,slow = head;

int count = 0;

while(fast!=slow || count==0)

{

fast = fast.next.next;

slow = slow.next;

if(fast==slow) break;

}

slow = head;

while(slow!=fast)

{

slow = slow.next;

fast = fast.next;

}

return fast;

}

}

### 2.13 Reorder List

 Given a singly linked list *L*: *L*0→*L*1→…→*Ln*-1→*L*n,  
reorder it to: *L*0→*Ln*→*L*1→*Ln*-1→*L*2→*Ln*-2→…

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

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

题解：

public class Solution {

public void reorderList(ListNode head) {

ListNode node = head;

int len = 0;

while(node!=null)

{

len ++;

node = node.next;

}

if(len<=2) return;

ListNode s = head;

for(int i=0;i<len-len/2-1;i++)

s = s.next;

ListNode cur = s.next,ne = cur.next;

while(ne!=null)

{

cur.next = ne.next;

ne.next= s.next;

s.next = ne;

ne = cur.next;

}

cur = head;

ne = s.next;

while(cur!=s)

{

s.next= ne.next;

ne.next = cur.next;

cur.next = ne;

cur = ne.next;

ne = s.next;

}

}

}

2.14 LRU Cache

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

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

题解：

### 3.1 Valid Palindrome

 Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,  
"A man, a plan, a canal: Panama" is a palindrome.  
"race a car" is *not* a palindrome.

**Note:**  
Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

题解：

public class Solution {

public boolean isPalindrome(String s) {

if(s.equals("")) return true;

String ss = s.toLowerCase();

StringBuilder sb = new StringBuilder();

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

{

if((ss.charAt(i)>='a' && ss.charAt(i)<='z')||(ss.charAt(i)>='0' && ss.charAt(i)<='9'))

{

String temp = ss.charAt(i)+"";

sb.append(temp);

}

}

String temp = sb.toString();

for(int i=0,j=temp.length()-1;i<=j;i++,j--)

{

if(temp.charAt(i)!=temp.charAt(j))

return false;

}

return true;

}

}

### 3.2 Implement strStr()

 Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

**Update (2014-11-02):**  
The signature of the function had been updated to return the *index* instead of the pointer. If you still see your function signature returns a char \* or String, please click the reload button  to reset your code definition.

题解：

### 3.3 String to Integer (atoi)

 Implement atoi to convert a string to an integer.

**Hint:** Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

**Notes:** It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

**Update (2015-02-10):**  
The signature of the C++ function had been updated. If you still see your function signature accepts aconst char \* argument, please click the reload button  to reset your code definition.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned. If the correct value is out of the range of representable values, INT\_MAX (2147483647) or INT\_MIN (-2147483648) is returned.

题解：

public class Solution {

public int atoi(String str) {

String s = str.trim();

if(s.equals("")) return 0;

boolean minus = true;

int count = 0,position=0;

while(s.charAt(position)=='-' || s.charAt(position)=='+')

{

count ++;

position ++;

}

if(count>=2) return 0;

if(count==0 || s.charAt(0)=='+') minus = false;

int result = 0,i=0;

if(count==0) i=0;

else i = 1;

for( ;i<s.length();i++)

{

if(s.charAt(i)>='0' && s.charAt(i)<='9')

{

if(result>Integer.MAX\_VALUE/10 || (result==Integer.MAX\_VALUE/10 &&s.charAt(i)-'0'>Integer.MAX\_VALUE%10))

return minus==true?Integer.MIN\_VALUE:Integer.MAX\_VALUE;

result = result\*10 + s.charAt(i)-'0';

}

else

break;

}

if(minus==true) result \*= -1;

return result;

}

}

3.4 Add Binary

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

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

题解：

public class Solution {

public String addBinary(String a, String b) {

StringBuilder sb = new StringBuilder();

int jin = 0;

int i = a.length()-1,j = b.length()-1;

while(i>=0 || j>=0)

{

int temp = 0;

if(i>=0) temp += a.charAt(i)-'0';

if(j>=0) temp += b.charAt(j)-'0';

temp += jin;

if(temp==3)

{

jin = 1;

sb.append("1");

}else if(temp==2){

jin = 1;

sb.append("0");

}else{

jin = 0;

sb.append(temp+"");

}

i --;

j --;

}

if(jin!=0) sb.append(jin+"");

return sb.reverse().toString();

}

}

### 3.5 Longest Palindromic Substring

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

题解：

public class Solution {

public String longestPalindrome(String s) {

int longest = Integer.MIN\_VALUE,flag=1,index=-1;

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

{

int left = i-1,right=i+1,temp = 0;

while(left>=0 && right<s.length() && s.charAt(left)==s.charAt(right))

{

temp += 2;

left --;

right ++;

}

if(temp+1 > longest) {longest = temp +1;index=i;flag=1;}

left = i;

right = i+1;

temp = 0;

while(left>=0 && right<s.length() && s.charAt(left)==s.charAt(right))

{

temp += 2;

left --;

right ++;

}

if(temp > longest) {longest = temp;flag = -1;index=i;}

}

if(flag==1)

return s.substring(index-longest/2,index+longest/2+1);

else

return s.substring(index-longest/2+1,index+longest/2+1);

}

}

### 3.6 Regular Expression Matching

 Implement regular expression matching with support for'.' and '\*'.

'.' Matches any single character.

'\*' Matches zero or more of the preceding element.

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

The function prototype should be:

bool isMatch(const char \*s, const char \*p)

Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "a\*") → true

isMatch("aa", ".\*") → true

isMatch("ab", ".\*") → true

isMatch("aab", "c\*a\*b") → true

题解：

public class Solution {

public boolean isMatch(String s, String p) {

if(p.length() == 0)

return s.length() == 0;

//p's length 1 is special case

if(p.length() == 1 || p.charAt(1) != '\*'){

if(s.length() < 1 || (p.charAt(0) != '.' && s.charAt(0) != p.charAt(0)))

return false;

return isMatch(s.substring(1), p.substring(1));

}else{

int len = s.length();

int i = -1;

while(i<len && (i < 0 || p.charAt(0) == '.' || p.charAt(0) == s.charAt(i))){

if(isMatch(s.substring(i+1), p.substring(2)))

return true;

i++;

}

return false;

}

}

}

### 3.7 Wildcard Matching

 Implement wildcard pattern matching with support for '?' and '\*'.

'?' Matches any single character.

'\*' Matches any sequence of characters (including the empty sequence).

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

The function prototype should be:

bool isMatch(const char \*s, const char \*p)

Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "\*") → true

isMatch("aa", "a\*") → true

isMatch("ab", "?\*") → true

isMatch("aab", "c\*a\*b") → false

题解：

递归的方式会超时

public class Solution {

public boolean isMatch(String s, String p) {

int a=0,b=0,star=-1,match = -1;

while(a<s.length())

{

if(b<p.length()&&(s.charAt(a)==p.charAt(b) || p.charAt(b)=='?'))

{

a ++;

b ++;

}else if(b<p.length()&&p.charAt(b)=='\*' ){

while(b<p.length()&&p.charAt(b)=='\*') b++;

if(b==p.length()) return true;

star = b;

match = a;

}else

{

if(star!=-1)

{

b = star;

match ++;

a = match;

}else

{

return false;

}

}

}

while(b<p.length()&&p.charAt(b)=='\*')

b++;

return b==p.length();

}

}

### 3.8 Longest Common Prefix

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

题解：

public class Solution {

public int longestCommon(String a,String b)

{

int i=0,j=0;

int result = 0;

while(i<a.length()&&j<b.length())

{

if(a.charAt(i)==b.charAt(j))

{

result += 1;

i ++; j ++;

}

else

break;

}

return result;

}

public String longestCommonPrefix(String[] strs) {

if(strs.length==0) return "";

if(strs.length==1) return strs[0];

int result = Integer.MAX\_VALUE;

for(int i=1;i<strs.length;i++)

{

int temp = longestCommon(strs[0],strs[i]);

if(temp<result)

result = temp;

}

return strs[0].substring(0,result);

}

}

3.9 Valid Number

Validate if a given string is numeric.

Some examples:  
"0" => true  
" 0.1 " => true  
"abc" => false  
"1 a" => false  
"2e10" => true

**Note:** It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

题解：

We start with trimming.

* If we see [0-9] we reset the number flags.
* We can only see . if we didn't see e or ..
* We can only see e if we didn't see e but we did see a number. We reset numberAfterE flag.
* We can only see + and - in the beginning and after an e
* any other character break the validation.

At the and it is only valid if there was at least 1 number and if we did see an e then a number after it as well.

So basically the number should match this regular expression:

[-+]?[0-9]\*(.[0-9]+)?(e[-+]?[0-9]+)?

public class Solution {

public boolean isNumber(String s) {

s = s.trim();

boolean pointSeen = false;

boolean eSeen = false;

boolean numberSeen = false;

boolean numberAfterE = true;

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

if('0' <= s.charAt(i) && s.charAt(i) <= '9') {

numberSeen = true;

numberAfterE = true;

} else if(s.charAt(i) == '.') {

if(eSeen || pointSeen) {

return false;

}

pointSeen = true;

} else if(s.charAt(i) == 'e') {

if(eSeen || !numberSeen) {

return false;

}

numberAfterE = false;

eSeen = true;

} else if(s.charAt(i) == '-' || s.charAt(i) == '+') {

if(i != 0 && s.charAt(i-1) != 'e') {

return false;

}

} else {

return false;

}

}

return numberSeen && numberAfterE;

}

}

3.10 Integer to Roman

3.11 Roman to Integer

Given a roman numeral, convert it to an integer.

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

题解：

### 3.12 Count and Say

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

1 is read off as "one 1" or 11.  
11 is read off as "two 1s" or 21.  
21 is read off as "one 2, then one 1" or 1211.

Given an integer *n*, generate the *n*th sequence.

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

### 题解：

### public class Solution {

### public static String read(String s)

### {

### char c = s.charAt(0);

### int count = 1;

### StringBuilder sb = new StringBuilder();

### for(int i=1;i<s.length();i++)

### {

### if(s.charAt(i)==c)

### count ++;

### else{

### sb.append(count+"");

### sb.append(c+"");

### c = s.charAt(i);

### count = 1;

### }

### }

### sb.append(count+"");

### sb.append(c+"");

### return sb.toString();

### }

### public String countAndSay(int n) {

### if(n<=0) return "";

### String s = "1";

### for(int i=1;i<n;i++)

### {

### s = read(s);

### }

### return s;

### }

### }

### 3.13 Anagrams

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

Note: All inputs will be in lower-case.

题解：

public class Solution {

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

List<String> result = new ArrayList<String>();

if(strs.length<=0) return result;

Map<String,List<String>> map = new HashMap<String,List<String>>();

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

{

char[] temp = strs[i].toCharArray();

Arrays.sort(temp);

String s = new String(temp);

if(map.get(s)==null)

{

List<String> lst = new ArrayList<String>();

lst.add(strs[i]);

map.put(s,lst);

}else{

List<String> lst = map.get(s);

lst.add(strs[i]);

map.put(s,lst);

}

}

Iterator<String> keys = map.keySet().iterator();

while(keys.hasNext())

{

String t = (String) keys.next();

List<String> lst = map.get(t);

if(lst.size()>1)

{

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

result.add(lst.get(i));

}

}

return result;

}

}

### 3.14 Simplify Path

 Given an absolute path for a file (Unix-style), simplify it.

For example,  
**path** = "/home/", => "/home"  
**path** = "/a/./b/../../c/", => "/c"

[click to show corner cases.](https://leetcode.com/problems/simplify-path/)

**Corner Cases:**

* Did you consider the case where **path** = "/../"?  
  In this case, you should return "/".
* Another corner case is the path might contain multiple slashes '/' together, such as "/home//foo/".  
  In this case, you should ignore redundant slashes and return "/home/foo".

### 题解：

### public class Solution {

### public String simplifyPath(String path) {

### List<String> lst = new ArrayList<String>();

### StringBuilder sb = new StringBuilder();

### for(int i=0;i<path.length();i++)

### {

### if(path.charAt(i)!='/')

### sb.append(path.charAt(i)+"");

### else

### {

### String s = sb.toString();

### if(!s.equals(""))

### {

### lst.add(s);

### sb = new StringBuilder();

### }

### }

### }

### String s = sb.toString();

### if(!s.equals(""))

### lst.add(s);

### s = "";

### // String[] temp = path.split("/{1,}");

### Stack<String> stack = new Stack<String>();

### for(int i=0;i<lst.size();i++)

### {

### if(lst.get(i).equals("."))

### continue;

### else if(lst.get(i).equals(".."))

### {

### if(!stack.isEmpty())

### stack.pop();

### }

### else

### stack.push(lst.get(i));

### }

### while(!stack.isEmpty())

### {

### s = "/"+stack.pop()+s;

### }

### if(s.equals(""))

### return "/";

### else

### return s;

### }

### }

### 3.15 Length of Last Word

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

If the last word does not exist, return 0.

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

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

S = “a “

Return 1.

题解：

public class Solution {

public int lengthOfLastWord(String s) {

int i = 0,len=0;

String ss = s.trim();

for(i=ss.length()-1;i>=0;i--)

{

if(ss.charAt(i)!=' ')

len ++;

else

break;

}

return len;

}

}

4 栈和队列

4.1 栈

### 4.1.1 Valid Parentheses

 Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

题解：

public class Solution {

public boolean isValid(String s) {

if(s.equals("")) return true;

Stack<Character> stack = new Stack<Character>();

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

{

if(s.charAt(i)=='('||s.charAt(i)=='['||s.charAt(i)=='{')

stack.push(s.charAt(i));

else if(s.charAt(i)==')')

{

if(!stack.isEmpty()&&stack.peek()=='(')

stack.pop();

else

return false;

}else if(!stack.isEmpty()&&s.charAt(i)==']')

{

if(stack.peek()=='[')

stack.pop();

else

return false;

}else if(!stack.isEmpty()&&s.charAt(i)=='}')

{

if(stack.peek()=='{')

stack.pop();

else

return false;

}else

return false;

}

if(stack.isEmpty()) return true;

else return false;

}

}

4.1.2 Longest Valid Parentheses

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

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

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

题解：

public class Solution {

public int longestValidParentheses(String s) {

int longest = 0;

Stack<Integer> stack = new Stack<Integer>();

int left = 0;

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

{

if(s.charAt(i)=='(')

{

stack.push(i);

}

else{

if(!stack.isEmpty())

{

stack.pop();

if(!stack.isEmpty())

longest = longest>(i-stack.peek())?longest:(i-stack.peek());

else

{

longest = longest>(i-left+1)?longest:(i-left+1);

}

}

else

left = i+1;

}

}

return longest;

}

}

4.1.3 Largest Rectangle in Histogram

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



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



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

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

题解：

public class Solution {

public int largestRectangleArea(int[] height) {

if(height.length<=0) return 0;

Stack<Integer> stack = new Stack<Integer>();

int max = 0,temp=0;

for(int i=0;i<height.length;)

{

if(stack.isEmpty()||height[i]>height[stack.peek()])

{

stack.push(i);i++;

}

else{

int left = stack.pop();

temp = stack.isEmpty()?i:(i-stack.peek()-1);

if(max<temp\*height[left]) max=temp\*height[left];

}

}

while(!stack.isEmpty())

{

int left = stack.pop();

temp = stack.isEmpty()?height.length:(height.length-stack.peek()-1);

if(max<temp\*height[left]) max=temp\*height[left];

}

return max;

}

}

### 4.1.4 Evaluate Reverse Polish Notation

 Evaluate the value of an arithmetic expression in[Reverse Polish Notation](http://en.wikipedia.org/wiki/Reverse_Polish_notation).

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

Some examples:

["2", "1", "+", "3", "\*"] -> ((2 + 1) \* 3) -> 9

["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

题解：

public class Solution {

public int evalRPN(String[] tokens) {

if(tokens.length<=0) return 0;

Stack<Integer> stack = new Stack<Integer>();

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

{

if(!tokens[i].equals("+")&&!tokens[i].equals("-")&&!tokens[i].equals("\*")&&!tokens[i].equals("/"))

stack.push(Integer.parseInt(tokens[i]));

else{

int b = stack.pop();

int a = stack.pop();

if(tokens[i].equals("+"))

{

int c = a+b;

stack.push(c);

}else if(tokens[i].equals("-")){

int c = a-b;

stack.push(c);

}else if(tokens[i].equals("\*")){

int c = a\*b;

stack.push(c);

}else {

int c = a/b;

stack.push(c);

}

}

}

return stack.pop();

}

}

5 树

5.1 二叉树的遍历

### 5.1.1 Binary Tree Preorder Traversal

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

For example:  
Given binary tree {1,#,2,3},

1

\

2

/

3

return [1,2,3].

**Note:** Recursive solution is trivial, could you do it iteratively?

题解：跌代版

/\*\*

\* Definition for binary tree

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

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

Stack<TreeNode> stack = new Stack<TreeNode>();

public void printLeft(TreeNode root){

while(root!=null)

{

lst.add(root.val);

if(root.right!=null) stack.push(root.right);

root = root.left;

}

}

public List<Integer> preorderTraversal(TreeNode root) {

printLeft(root);

while(!stack.isEmpty())

{

printLeft(stack.pop());

}

return lst;

}

}

### 5.1.2 Binary Tree Inorder Traversal

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

For example:  
Given binary tree {1,#,2,3},

1

\

2

/

3

return [1,3,2].

**Note:** Recursive solution is trivial, could you do it iteratively?

confused what "{1,#,2,3}" means? [> read more on how binary tree is serialized on OJ.](https://leetcode.com/problems/binary-tree-inorder-traversal/)

题解：

public class Solution {

Stack<TreeNode> stack = new Stack<TreeNode>();

public void addLeft(TreeNode root)

{

while(root!=null)

{

stack.push(root);

root = root.left;

}

}

public List<Integer> inorderTraversal(TreeNode root) {

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

addLeft(root);

while(!stack.isEmpty())

{

TreeNode temp = stack.pop();

lst.add(temp.val);

if(temp.right!=null)

addLeft(temp.right);

}

return lst;

}

}

5.1.3 Binary Tree Postorder Traversal

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

For example:  
Given binary tree {1,#,2,3},

1

\

2

/

3

return [3,2,1].

**Note:** Recursive solution is trivial, could you do it iteratively?

题解：

public class Solution {

public void addLeft(TreeNode root,Stack<TreeNode> stack){

while(root!=null)

{

stack.push(root);

root = root.left;

}

}

public List<Integer> postorderTraversal(TreeNode root) {

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

Stack<TreeNode> stack = new Stack<TreeNode>();

if(root==null) return lst;

TreeNode pre = null,cur=null;

addLeft(root,stack);

while(!stack.isEmpty())

{

cur = stack.pop();

if(cur.right==null||cur.right==pre)

{

lst.add(cur.val);

pre = cur;

}else{

stack.push(cur);

cur = cur.right;

addLeft(cur,stack);

}

}

return lst;

}

}

5.1.4 Binary Tree Level Order Traversal

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

For example:  
Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its level order traversal as:

[

[3],

[9,20],

[15,7]

]

confused what "{1,#,2,3}" means? [> read more on how binary tree is serialized on OJ.](https://leetcode.com/problems/binary-tree-level-order-traversal/)

题解：

public class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

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

List<TreeNode> lst = new ArrayList<TreeNode>();

if(root==null) return result;

lst.add(root);

while(!lst.isEmpty())

{

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

int size = lst.size();

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

{

TreeNode node = lst.get(0);

temp.add(node.val);

if(node.left!=null) lst.add(node.left);

if(node.right!=null) lst.add(node.right);

lst.remove(0);

}

result.add(temp);

}

return result;

}

}

5.1.5 Binary Tree Level Order Traversal II

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

For example:  
Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its bottom-up level order traversal as:

[

[15,7],

[9,20],

[3]

]

题解：

public class Solution {

public List<List<Integer>> levelOrderBottom(TreeNode root) {

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

List<TreeNode> lst = new ArrayList<TreeNode>();

if(root==null) return result;

lst.add(root);

while(!lst.isEmpty())

{

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

int size = lst.size();

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

{

TreeNode node = lst.get(0);

temp.add(node.val);

if(node.left!=null) lst.add(node.left);

if(node.right!=null) lst.add(node.right);

lst.remove(0);

}

result.add(temp);

}

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

for(int i=result.size()-1;i>=0;i--)

r.add(result.get(i));

return r;

}

}

### 5.1.6 Binary Tree Zigzag Level Order Traversal

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

For example:  
Given binary tree {3,9,20,#,#,15,7},

3

/ \

9 20

/ \

15 7

return its zigzag level order traversal as:

[

[3],

[20,9],

[15,7]

]

题解：

public class Solution {

public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

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

List<TreeNode> lst = new ArrayList<TreeNode>();

if(root==null) return result;

lst.add(root);

while(!lst.isEmpty())

{

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

int size = lst.size();

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

{

TreeNode node = lst.get(0);

temp.add(node.val);

if(node.left!=null) lst.add(node.left);

if(node.right!=null) lst.add(node.right);

lst.remove(0);

}

result.add(temp);

}

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

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

{

if(i%2!=0)

{

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

for(int j=result.get(i).size()-1;j>=0;j--)

temp.add(result.get(i).get(j));

r.add(temp);

}else{

r.add(result.get(i));

}

}

return r;

}

}

5.1.7 Recover Binary Search Tree

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

Recover the tree without changing its structure.

**Note:**  
A solution using O(*n*) space is pretty straight forward. Could you devise a constant space solution?

confused what "{1,#,2,3}" means? [> read more on how binary tree is serialized on OJ.](https://leetcode.com/problems/recover-binary-search-tree/)

**OJ's Binary Tree Serialization:**

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

1

/ \

2 3

/

4

\

5

The above binary tree is serialized as "{1,2,3,#,#,4,#,#,5}".

题解：morris方式会改变树的结构

public class Solution {

TreeNode[] tree = new TreeNode[2];

TreeNode pre = null;

public void parse(TreeNode root )

{

if(root==null) return ;

if(root.left!=null)

parse(root.left );

if(pre!=null && pre.val>root.val)

{

if(tree[0]==null)

{

tree[0] = pre;

tree[1] = root;

}else

tree[1] = root;

}

pre = root;

if(root.right!=null)

parse(root.right);

}

public void recoverTree(TreeNode root) {

if(root==null) return;

parse(root);

if(tree[0]!=null && tree[1]!=null)

{

int temp = tree[0].val;

tree[0].val = tree[1].val;

tree[1].val = temp;

}

}

}

5.1.8 Same Tree

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

题解：

public class Solution {

public boolean isSameTree(TreeNode p, TreeNode q) {

if(p!=null&&q!=null)

{

if(p.val==q.val)

return isSameTree(p.left,q.left)&&isSameTree(p.right,q.right);

else

return false;

}else if(p==null&&q!=null|| q==null&&p!=null)

return false;

else

return true;

}

}

### 5.1.9 Symmetric Tree

 Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree is symmetric:

1

/ \

2 2

/ \ / \

3 4 4 3

But the following is not:

1

/ \

2 2

\ \

3 3

**Note:**  
Bonus points if you could solve it both recursively and iteratively.

题解：

public class Solution {

public boolean Symmetric(TreeNode p,TreeNode q)

{

if(p!=null&q!=null)

{

if(p.val==q.val)

return Symmetric(p.left,q.right)&&Symmetric(p.right,q.left);

else

return false;

}else if(p==null&&q!=null ||p!=null&&q==null)

return false;

else

return true;

}

public boolean isSymmetric(TreeNode root) {

if(root==null) return true;

return Symmetric(root.left,root.right);

}

}

5.1.10 Balanced Binary Tree

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

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

题解：

public class Solution {

public int tree(TreeNode root)

{

if(root==null) return 0;

int l = tree(root.left);

int r = tree(root.right);

return l+1>r+1?l+1:r+1;

}

public boolean isBalanced(TreeNode root) {

if(root==null) return true;

int l = tree(root.left);

int r = tree(root.right);

if(Math.abs(l-r)<=1)

return isBalanced(root.left)&&isBalanced(root.right);

else

return false;

}

}

5.1.11 Flatten Binary Tree to Linked List

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

For example,  
Given

1

/ \

2 5

/ \ \

3 4 6

The flattened tree should look like:

1

\

2

\

3

\

4

\

5

\

6

题解：

public class Solution {

public void flatten(TreeNode root) {

if(root==null) return;

Stack<TreeNode> stack= new Stack<TreeNode>();

stack.push(root);

TreeNode node = null;

while(!stack.isEmpty())

{

node = stack.pop();

if(node.right!=null) stack.push(node.right);

if(node.left!=null) stack.push(node.left);

node.left = null;

if(!stack.isEmpty())

node.right = stack.peek();

}

}

}

### 5.1.12 Populating Next Right Pointers in Each Node

 Given a binary tree

struct TreeLinkNode {

TreeLinkNode \*left;

TreeLinkNode \*right;

TreeLinkNode \*next;

}

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

**Note:**

* You may only use constant extra space.
* You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example,  
Given the following perfect binary tree,

1

/ \

2 3

/ \ / \

4 5 6 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ / \

4->5->6->7 -> NULL

Populating Next Right Pointers in Each Node II

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

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

**Note:**

* You may only use constant extra space.

For example,  
Given the following binary tree,

1

/ \

2 3

/ \ \

4 5 7

After calling your function, the tree should look like:

1 -> NULL

/ \

2 -> 3 -> NULL

/ \ \

4-> 5 -> 7 -> NULL

题解：

public class Solution {

public void connect(TreeLinkNode root) {

if(root==null) return;

TreeLinkNode dummy = new TreeLinkNode(-1);

for(TreeLinkNode pre = dummy,cur=root;cur!=null;cur=cur.next)

{

if(cur.left!=null)

{

pre.next = cur.left;

pre = pre.next;

}

if(cur.right!=null)

{

pre.next = cur.right;

pre = pre.next;

}

}

connect(dummy.next); //递归每层第一个元素

}

}

5.2 二叉树的构建

### 5.2.1 Construct Binary Tree from Preorder and Inorder Traversal

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

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

题解：

public class Solution {

public TreeNode build(int[] preorder, int a,int b,int[] inorder,int c,int d)

{

if(a>b || c>d) return null;

TreeNode root = new TreeNode(preorder[a]);

int l=0,r=0;

for(int i=c;i<=d;i++)

{

if(inorder[i]==preorder[a])

{

l = i-c;

r = d-i;

break;

}

}

TreeNode left = build(preorder,a+1,a+l,inorder,c,c+l-1);

TreeNode right = build(preorder,a+l+1,b,inorder,c+l+1,d);

root.left = left;

root.right = right;

return root;

}

public TreeNode buildTree(int[] preorder, int[] inorder) {

if(preorder.length<=0 || inorder.length<=0) return null;

return build(preorder,0,preorder.length-1,inorder,0,inorder.length-1);

}

}

5.2.2 Construct Binary Tree from Inorder and Postorder Traversal

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

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

题解：

public class Solution {

public TreeNode build(int[] inorder, int a,int b,int[] postorder,int c,int d)

{

if(a>b || c>d) return null;

TreeNode root = new TreeNode(postorder[d]);

int l=0,r=0;

for(int i=a;i<=b;i++)

{

if(inorder[i]==postorder[d])

{

l = i-a;

r = b-i;

break;

}

}

TreeNode left = build(inorder,a,a+l-1,postorder,c,c+l-1);

TreeNode right = build(inorder,a+l+1,b,postorder,c+l,d-1);

root.left = left;

root.right = right;

return root;

}

public TreeNode buildTree(int[] inorder, int[] postorder) {

if(postorder.length<=0 || inorder.length<=0) return null;

return build(inorder,0,inorder.length-1,postorder,0,postorder.length-1);

}

}

5.3 二叉查找树

5.3.1 Unique Binary Search Trees

Given *n*, how many structurally unique **BST's** (binary search trees) that store values 1...*n*?

For example,  
Given *n* = 3, there are a total of 5 unique BST's.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

题解：

public class Solution {

public int numTrees(int n) {

int[] r = new int[n+1];

r[0] = 1;r[1] = 1;

for(int i=2;i<=n;i++)

{

for(int k=1;k<=i;k++)

r[i] += r[k-1]\*r[i-k];

}

return r[n];

}

}

### 5.3.2 Unique Binary Search Trees II

 Given *n*, generate all structurally unique **BST's** (binary search trees) that store values 1...*n*.

For example,  
Given *n* = 3, your program should return all 5 unique BST's shown below.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

题解：

public class Solution {

public List<TreeNode> generate(int a,int b)

{

List<TreeNode> result = new ArrayList<TreeNode>();

if(a>b){

result.add(null);

return result;

}

for(int i=a;i<=b;i++)

{

List<TreeNode> l = generate(a,i-1);

List<TreeNode> r = generate(i+1,b);

for(int j=0;j<l.size();j++)

{

for(int k=0;k<r.size();k++)

{

TreeNode temp = new TreeNode(i);

temp.left = l.get(j);

temp.right = r.get(k);

result.add(temp);

}

}

}

return result;

}

public List<TreeNode> generateTrees(int n) {

return generate(1,n);

}

}

5.3.3 Validate Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

* The left subtree of a node contains only nodes with keys **less than** the node's key.
* The right subtree of a node contains only nodes with keys **greater than** the node's key.
* Both the left and right subtrees must also be binary search trees.

题解：

可以利用中序遍历。

public class Solution {

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

public void isValid(TreeNode root)

{

if(root==null)

return;

isValid(root.left);

lst.add(root.val);

isValid(root.right);

}

public boolean isValidBST(TreeNode root) {

if(root==null)

return true;

isValid(root);

for(int i=0;i<lst.size()-1;i++)

{

if(lst.get(i)>=lst.get(i+1))

return false;

}

return true;

}

}

递归：

public class Solution {

public boolean isValid(TreeNode root,long a,long b){

if(root==null ) return true;

return root.val>a&&root.val<b&&isValid(root.left,a,root.val)&&isValid(root.right,root.val,b);

}

public boolean isValidBST(TreeNode root) {

return isValid(root,Long.MIN\_VALUE,Long.MAX\_VALUE);

}

}

### 5.3.4 Convert Sorted Array to Binary Search Tree

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

题解：

public class Solution {

public TreeNode sort(int[] num,int a,int b){

if(a>b) return null;

int mid = (a+b)/2;

TreeNode root = new TreeNode(num[mid]);

TreeNode left = sort(num,a,mid-1);

TreeNode right = sort(num,mid+1,b);

root.left = left;

root.right = right;

return root;

}

public TreeNode sortedArrayToBST(int[] num) {

return sort(num,0,num.length-1);

}

}

### 5.3.5 Convert Sorted List to Binary Search Tree

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

题解：

public class Solution {

public TreeNode partSort(ListNode node,int start,int end)

{

int middle = 0;

middle = (start+end)/2;

ListNode h = node;

for(int i=start;i<middle;i++)//应该从start开始而不是0

h = h.next;

TreeNode root = new TreeNode(h.val);

if(middle-1>=start)

root.left = partSort(node,start,middle-1);

else

root.left = null;

if(middle+1<=end)

root.right = partSort(h.next,middle+1,end);//注意开始节点的变化h.next

else

root.right = null;

return root;

}

public TreeNode sortedListToBST(ListNode head) {

if(head==null)

return null;

int len = 0;

ListNode hh = head;

while(hh!=null)

{

len++;

hh = hh.next;

}

TreeNode root = partSort(head,0,len-1);

return root;

}

}

5.4 二叉树的递归

### 5.4.1 Minimum Depth of Binary Tree

 Given a binary tree, find its minimum depth.

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

题解：

public class Solution {

public int minDepth(TreeNode root) {

if(root==null) return 0;

if(root.left==null && root.right==null) return 1;

int l=0,r=0;

if(root.left!=null)

l = minDepth(root.left);

else

l = Integer.MAX\_VALUE;

if(root.right!=null)

r = minDepth(root.right);

else

r = Integer.MAX\_VALUE;

return l<r?l+1:r+1;

}

}

5.4.2 Maximum Depth of Binary Tree

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

题解：

public class Solution {

public int maxDepth(TreeNode root) {

if(root==null) return 0;

if(root.left==null && root.right==null) return 1;

int l=0,r=0;

if(root.left!=null)

l = maxDepth(root.left);

else

l = Integer.MIN\_VALUE;

if(root.right!=null)

r = maxDepth(root.right);

else

r = Integer.MIN\_VALUE;

return l>r?l+1:r+1;

}

}

5.4.3 Path Sum

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example:  
Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

题解：

public class Solution {

public boolean hasPathSum(TreeNode root, int sum) {

if(root==null) return false;

if(root.left==null&&root.right==null)

{

if(root.val==sum) return true;

else return false;

}

boolean l=false,r=false;

if(root.left!=null)

l = hasPathSum(root.left,sum-root.val);

if(root.right!=null)

r = hasPathSum(root.right,sum-root.val);

return l||r;

}

}

### 5.4.4 Path Sum II

 Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example:  
Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

return

[

[5,4,11,2],

[5,8,4,5]

]

题解：

public class Solution {

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

public void sumTree(TreeNode root,int sum,List<Integer> lst){

if(root==null) return;

if(root.left==null&&root.right==null){

if(root.val==sum)

{

lst.add(root.val);

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

temp.addAll(lst);

r.add(temp);

lst.remove(lst.size()-1);

return;

}

}

lst.add(root.val);

sumTree(root.left,sum-root.val,lst);

sumTree(root.right,sum-root.val,lst);

int size = lst.size();

lst.remove(size-1);

}

public List<List<Integer>> pathSum(TreeNode root, int sum) {

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

sumTree(root,sum,lst);

return r;

}

}5.4.5 Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum.

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

For example:  
Given the below binary tree,

1

/ \

2 3

Return 6.

题解：

public class Solution {

int max = Integer.MIN\_VALUE;

public int maxSum(TreeNode root){

if(root==null) return 0;

int l = maxSum(root.left);

int r = maxSum(root.right);

int sum = root.val;

if(l>0) sum +=l;

if(r>0) sum += r;

if(sum>max) max=sum;

int t = root.val+l>root.val+r?root.val+l:root.val+r;

return root.val>t?root.val:t;

}

public int maxPathSum(TreeNode root) {

maxSum(root);

return max;

}

}

5.4.6 Sum Root to Leaf Numbers

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

For example,

1

/ \

2 3

The root-to-leaf path 1->2 represents the number 12.  
The root-to-leaf path 1->3 represents the number 13.

Return the sum = 12 + 13 = 25.

题解：

public class Solution {

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

public void numbers(TreeNode root,int x){

if(root==null) return;

if(root.left==null&&root.right==null)

{

int t = x\*10+root.val;

lst.add(t);

return;

}

numbers(root.left,x\*10+root.val);

numbers(root.right,x\*10+root.val);

}

public int sumNumbers(TreeNode root) {

numbers(root,0);

int result = 0;

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

result += lst.get(i);

return result;

}

}

6 排序

6.1Merge Sorted Array

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

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

题解：

public class Solution {

public void merge(int A[], int m, int B[], int n) {

int a = m-1,b=n-1,i=m+n-1;

while(a>=0&&b>=0)

{

if(A[a]>B[b])

{

A[i] = A[a];

a --;

}else{

A[i] = B[b];

b --;

}

i --;

}

for(int j=b;j>=0;j--)

{

A[i] = B[j];

i --;

}

}

}

### 6.2 Merge Two Sorted Lists

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

题解：

public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode head = new ListNode(-1);

ListNode h=head;

while(l1!=null && l2!=null)

{

ListNode temp = l1.val<l2.val?l1:l2;

h.next = temp;

h=h.next;

if(temp==l1) l1=l1.next;

else l2=l2.next;

}

if(l1==null) h.next = l2;

else h.next = l1;

return head.next;

}

}

6.3 Merge k Sorted Lists

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

题解：如果按顺序一条条merge的话会超时

public class Solution {

public ListNode merge(ListNode l1, ListNode l2) {

ListNode head = new ListNode(-1);

ListNode h=head;

while(l1!=null && l2!=null)

{

ListNode temp = l1.val<l2.val?l1:l2;

h.next = temp;

h=h.next;

if(temp==l1) l1=l1.next;

else l2=l2.next;

}

if(l1==null) h.next = l2;

else h.next = l1;

return head.next;

}

public ListNode mergeKLists(List<ListNode> lists) {

if(lists.size()==0) return null;

List<ListNode> result = new ArrayList<ListNode>();

List<ListNode> temp = new ArrayList<ListNode>();

result.addAll(lists);

while(result.size()>1)

{

temp.clear();

for(int i=0;i<result.size()/2;i++)

{

temp.add(merge(result.get(2\*i),result.get(2\*i+1)));

}

if(result.size()%2!=0)

temp.add(result.get(result.size()-1));

result.clear();

result.addAll(temp);

}

return result.get(0);

}

}

6.4 Insertion Sort List

Sort a linked list using insertion sort.

题解：

public class Solution {

public ListNode insertionSortList(ListNode head) {

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

ListNode dummy = new ListNode(-1);

dummy.next = new ListNode(head.val);

head = head.next;

while(head!=null)

{

ListNode h = dummy.next;

while(h!=null)

{

if(h.val<=head.val)

h = h.next;

else{

ListNode temp = new ListNode(h.val);

temp.next = h.next;

h.next = temp;

h.val = head.val;

break;

}

}

if(h==null)

{

h = dummy.next;

while(h.next!=null)

h = h.next;

h.next = new ListNode(head.val);

}

head = head.next;

}

return dummy.next;

}

}

6.5 Sort List

Sort a linked list in *O*(*n* log *n*) time using constant space complexity.

题解:

public class Solution {

public ListNode mergeTwo(ListNode l1, ListNode l2) {

ListNode head = new ListNode(-1);

ListNode h=head;

while(l1!=null && l2!=null)

{

ListNode temp = l1.val<l2.val?l1:l2;

h.next = temp;

h=h.next;

if(temp==l1) l1=l1.next;

else l2=l2.next;

}

if(l1==null) h.next = l2;

else h.next = l1;

return head.next;

}

public ListNode sortList(ListNode head) {

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

int len = 0;

ListNode h = head;

while(h!=null){

h = h.next;

len ++;

}

int mid = len/2;

h = head;

while(mid>1)

{

h = h.next;

mid --;

}

ListNode hh = h.next;

h.next = null;

ListNode left = sortList(head);

ListNode right = sortList(hh);

return mergeTwo(left,right);

}

}

### 6.6 First Missing Positive

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

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

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

题解：

public class Solution {

public int firstMissingPositive(int[] A) {

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

{

while(A[i]!=(i+1)){

if(A[i]<=0||A[i]>A.length||A[i]==A[A[i]-1]) break;

int temp = A[i];

A[i] = A[temp-1];

A[temp-1] = temp;

}

}

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

if(A[i]!=i+1) return i+1;

}

return A.length+1;

}

}

6.7 Sort Colors

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

**Note:**  
You are not suppose to use the library's sort function for this problem.

[click to show follow up.](https://leetcode.com/problems/sort-colors/)

**Follow up:**  
A rather straight forward solution is a two-pass algorithm using counting sort.  
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

题解：

public class Solution {

public void sortColors(int[] A) {

int red=0,blue=A.length-1;

for(int i=0;i<blue+1;)

{

if(A[i]==0){

int temp = A[i];

A[i] = A[red];

A[red] = temp;

red ++;i ++;

}else if(A[i]==2){

int temp = A[i];

A[i] = A[blue];

A[blue] = temp;

blue --;

}else{

i++;

}

}

}

}

7 查找

### 7.1 Search for a Range

 Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of *O*(log *n*).

If the target is not found in the array, return [-1, -1].

For example,  
Given [5, 7, 7, 8, 8, 10] and target value 8,  
return [3, 4].

题解：

public class Solution {

public int[] searchRange(int[] A, int target) {

int[] result = {-1,-1};

int l=0,r=A.length-1;

while(l<=r){

int mid = (l+r)/2;

if(A[mid]==target&&(mid==(A.length-1)||A[mid+1]!=target)){

result[1] = mid;

break;

}

if(A[mid]<=target){

l = mid+1;

}else if(A[mid]>target){

r = mid-1;

}

}

l=0;r=A.length-1;

while(l<=r){

int mid = (l+r)/2;

if(A[mid]==target&&(mid==0||A[mid-1]!=target)){

result[0] = mid;

break;

}

if(A[mid]<target){

l = mid+1;

}else if(A[mid]>=target){

r = mid-1;

}

}

return result;

}

}

### 7.2 Search Insert Position

 Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.  
[1,3,5,6], 5 → 2  
[1,3,5,6], 2 → 1  
[1,3,5,6], 7 → 4  
[1,3,5,6], 0 → 0

题解：

public class Solution {

//log(n)

public int searchInsert(int[] A, int target) {

int l=0,r=A.length-1,result=-1;

int mid = 0;

while(l<=r){

mid = (l+r)/2;

if(A[mid]==target){

result = mid;

break;

}else if(A[mid]<target){

l = mid+1;

}else{

r = mid-1;

}

}

if(result!=-1) return result;

else{

if(A[mid]<target) return mid+1;

else return mid;

}

}

}

7.3 Search a 2D Matrix

Write an efficient algorithm that searches for a value in an*m* x *n* matrix. This matrix has the following properties:

* Integers in each row are sorted from left to right.
* The first integer of each row is greater than the last integer of the previous row.

For example,

Consider the following matrix:

[

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

Given **target** = 3, return true.

题解：

public class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

int i=0,j=matrix[0].length-1;

while(i<matrix.length&&j>=0){

if(matrix[i][j]==target){

return true;

}else if(matrix[i][j]<target){

i ++;

}else{

j --;

}

}

return false;

}

}

8 暴力枚举法

8.1 Subsets

Given a set of distinct integers, *S*, return all possible subsets.

**Note:**

* Elements in a subset must be in non-descending order.
* The solution set must not contain duplicate subsets.

For example,  
If ***S*** = [1,2,3], a solution is:

[

[3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[]

]

题解：

public class Solution {

public List<List<Integer>> subsets(int[] S) {

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

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

result.add(temp);

if(S.length==0)

return result;

Arrays.sort(S);

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

{

int n = result.size();

for(int j=0;j<n;j++)

{

temp = new ArrayList<Integer>();

temp.addAll(result.get(j));

temp.add(S[i]);

if(!result.contains(temp))

result.add(temp);

}

}

return result;

}

}

### 8.2 Subsets II

 Given a collection of integers that might contain duplicates, *S*, return all possible subsets.

**Note:**

* Elements in a subset must be in non-descending order.
* The solution set must not contain duplicate subsets.

For example,  
If ***S*** = [1,2,2], a solution is:

[

[2],

[1],

[1,2,2],

[2,2],

[1,2],

[]

]

题解：

public class Solution {

public List<List<Integer>> subsetsWithDup(int[] num) {

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

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

result.add(temp);

if(num.length==0)

return result;

Arrays.sort(num);

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

{

int n = result.size();

for(int j=0;j<n;j++)

{

temp = new ArrayList<Integer>();

temp.addAll(result.get(j));

temp.add(num[i]);

if(!result.contains(temp))

result.add(temp);

}

}

return result;

}

}

### 8.3 Permutations

 Given a collection of numbers, return all possible permutations.

For example,  
[1,2,3] have the following permutations:  
[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].

题解：

public class Solution {

public List<List<Integer>> permutation(int[] num,int start){

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

if(start>=num.length){

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

rr.add(lst);

return rr;

}

List<List<Integer>> r = permutation(num,start+1);

for(int i=0;i<r.size();i++){

List<Integer> lst = r.get(i);

for(int j=0;j<=lst.size();j++){

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

for(int k=0;k<j;k++)

temp.add(lst.get(k));

temp.add(num[start]);

for(int k=j;k<lst.size();k++)

temp.add(lst.get(k));

rr.add(temp);

}

}

return rr;

}

public List<List<Integer>> permute(int[] num) {

return permutation(num,0);

}

}

### 8.4 Permutations II

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

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

题解：

public class Solution {

public List<List<Integer>> permutation(int[] num,int start){

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

if(start>=num.length){

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

rr.add(lst);

return rr;

}

List<List<Integer>> r = permutation(num,start+1);

for(int i=0;i<r.size();i++){

List<Integer> lst = r.get(i);

for(int j=0;j<=lst.size();j++){

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

for(int k=0;k<j;k++)

temp.add(lst.get(k));

temp.add(num[start]);

for(int k=j;k<lst.size();k++)

temp.add(lst.get(k));

if(!rr.contains(temp))

rr.add(temp);

}

}

return rr;

}

public List<List<Integer>> permuteUnique(int[] num) {

return permutation(num,0);

}

}

8.5 Combinations

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

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

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

题解：

public class Solution {

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

public void combination(int n, int k,int start,int cur,List<Integer> lst) {

if(cur==k){

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

temp.addAll(lst);

if(!result.contains(temp))

result.add(temp);

}

for(int i=start;i<=n;i++){

lst.add(i);

combination(n,k,i+1,cur+1,lst);

lst.remove(lst.size()-1);

}

}

public List<List<Integer>> combine(int n, int k) {

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

combination(n,k,1,0,lst);

return result;

}

}

8.6 Letter Combinations of a Phone Number

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.



**Input:**Digit string "23"

**Output:** ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

**Note:**  
Although the above answer is in lexicographical order, your answer could be in any order you want.

题解：

public class Solution {

List<String> combine(List<String> a,List<String> b)

{

List<String> result = new ArrayList<String>();

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

{

for(int j=0;j<b.size();j++)

{

String s = ""+a.get(i)+b.get(j);

result.add(s);

}

}

return result;

}

List<String> convert(String s)

{

List<String> result = new ArrayList<String>();

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

result.add(String.valueOf(s.charAt(i)));

return result;

}

public List<String> letterCombinations(String digits) {

Map<Integer,String> map = new HashMap<Integer,String>();

map.put(0,"");map.put(1,"");map.put(2,"abc");map.put(3,"def");map.put(4,"ghi");

map.put(5,"jkl");map.put(6,"mno");map.put(7,"pqrs");map.put(8,"tuv");map.put(9,"wxyz");

List<String> result = new ArrayList<String>();

if(digits.equals("")) return result;

if(digits.length()==0)

{

result.add("");

return result;

}

List<String> first = new ArrayList<String>();

String temp = map.get(digits.charAt(0)-'0');

first = convert(temp);

for(int i=1;i<digits.length();i++)

{

temp = map.get(digits.charAt(i)-'0');

List<String> t = convert(temp);

first = combine(first,t);

}

return first;

}

}

9 广度优先搜索

9.1 Word Ladder

Given two words (beginWord and endWord), and a dictionary, find the length of shortest transformation sequence from beginWord to endWord, such that:

Only one letter can be changed at a time

Each intermediate word must exist in the dictionary

For example,

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

Note:

Return 0 if there is no such transformation sequence.

All words have the same length.

All words contain only lowercase alphabetic characters.

题解：

public class Solution {

public int ladderLength(String beginWord, String endWord, Set<String> wordDict) {

int len = beginWord.length();

List<String> lst = new ArrayList<String>();

Map<String,Integer> map = new HashMap<String,Integer>();

lst.add(beginWord);

map.put(beginWord, 1);

int level = 0;

while(!lst.isEmpty()){

String t = lst.remove(0);

level = map.get(t);

for(int k=0;k<len;k++){

for(char j='a';j<='z';j++){

if(j!=t.charAt(k)){

StringBuilder sb=new StringBuilder(t);

sb.setCharAt(k, j);

if(sb.toString().equals(endWord))

return level+1;

if(wordDict.contains(sb.toString())&&map.get(sb.toString())==null)

{

lst.add(sb.toString());

map.put(sb.toString(), level+1);

}

}

}

}

}

return 0;

}

}

9.2 Word Ladder II

Given two words (*start* and *end*), and a dictionary, find all shortest transformation sequence(s) from *start* to *end*, such that:

1. Only one letter can be changed at a time
2. Each intermediate word must exist in the dictionary

For example,

Given:  
*start* = "hit"  
*end* = "cog"  
*dict* = ["hot","dot","dog","lot","log"]

Return

[

["hit","hot","dot","dog","cog"],

["hit","hot","lot","log","cog"]

]

**Note:**

* All words have the same length.
* All words contain only lowercase alphabetic characters.

题解：

public class Solution {

Map<String,Integer> map = new HashMap<String,Integer>();

public void ladderLength(String beginWord, String endWord, Set<String> wordDict) {

int len = beginWord.length();

List<String> lst = new ArrayList<String>();

lst.add(beginWord);

map.put(beginWord, 1);

int level = 0;

while(!lst.isEmpty()){

String t = lst.remove(0);

level = map.get(t);

for(int k=0;k<len;k++){

for(char j='a';j<='z';j++){

if(j!=t.charAt(k)){

StringBuilder sb=new StringBuilder(t);

sb.setCharAt(k, j);

if(sb.toString().equals(endWord)||wordDict.contains(sb.toString()) )

{

if(map.get(sb.toString())==null){

lst.add(sb.toString());

map.put(sb.toString(), level+1);

}

}

}

}

}

}

}

public void genPath(String start, String end, Set<String> dict, List<String> pathArray,List<List<String>> result){

//找到了，需要reverse加入的所有单词

if(start.equals(end)==true) {

pathArray.add(start);

Collections.reverse(pathArray);

result.add(pathArray);

return;

}

if(map.get(start)==null) {

return;

}

pathArray.add(start);

int nextDepth = (int)map.get(start) - 1;

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

char[] strCharArr = start.toCharArray();

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

if(strCharArr[i]==ch) {

continue;

}

strCharArr[i] = ch;

String newWord = new String(strCharArr);

//只相差一个字母同时这个单词所在的层数也是当前单词的上一层

if(map.get(newWord)!=null&&(map.get(newWord)==nextDepth)) {

ArrayList<String> newPathArray = new ArrayList<String>(pathArray);

genPath(newWord,end,dict,newPathArray,result);

}

}

}

}

public List<List<String>> findLadders(String start, String end, Set<String> dict) {

List<List<String>> result = new ArrayList<List<String>>();

ArrayList<String> path = new ArrayList<String>();

if(start==null||end==null||start.length()!=end.length()) {

return result;

}

ladderLength(start, end, dict);

genPath(end,start, dict, path, result);

return result;

}

}

### 9.3 Surrounded Regions

 Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,

X X X X

X O O X

X X O X

X O X X

After running your function, the board should be:

X X X X

X X X X

X X X X

X O X X

题解：

public class Solution {

LinkedList<Integer> queue = new LinkedList<Integer>();

public void bfs(char[][] board,int i,int j)

{

if(board[i][j]!='O')

return;

board[i][j] = 'S';

int code = i\*board[0].length+j;

queue.add(code);

while(!queue.isEmpty()){

code = queue.poll();

int row = code/board[0].length;

int col = code%board[0].length;

if(row>=1 && board[row-1][col]=='O'){

queue.add((row-1)\*board[0].length + col);

board[row-1][col]='S';

}

if(row<=board.length-2 && board[row+1][col]=='O'){

queue.add((row+1)\*board[0].length + col);

board[row+1][col]='S';

}

if(col>=1 && board[row][col-1]=='O'){

queue.add(row\*board[0].length + col-1);

board[row][col-1]='S';

}

if(col<=board[0].length-2 && board[row][col+1]=='O'){

queue.add(row\*board[0].length + col+1);

board[row][col+1]='S';

}

}

}

public void solve(char[][] board) {

if(board.length<=0) return;

for(int j=0;j<board[0].length;j++){

if(board[0][j]=='O')

bfs(board,0,j);

if(board[board.length-1][j]=='O')

bfs(board,board.length-1,j);

}

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

if(board[i][0]=='O')

bfs(board,i,0);

if(board[i][board[0].length-1]=='O')

bfs(board,i,board[0].length-1);

}

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

for(int j=0;j<board[0].length;j++){

if(board[i][j]=='O') board[i][j] ='X';

if(board[i][j]=='S') board[i][j] ='O';

}

}

}

}

10 深度优先搜索

10.1 Palindrome Partitioning

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

Return all possible palindrome partitioning of *s*.

For example, given *s* = "aab",  
Return

[

["aa","b"],

["a","a","b"]

]

题解：

public class Solution {

public static boolean isPalindrome(String s,int start,int end){

int len = end-start+1;

if(len==1) return true;

if(len%2!=0){

while(s.charAt(start)==s.charAt(end) && start<end){

start ++;

end --;

}

if(start==end) return true;

else return false;

}else{

while(s.charAt(start)==s.charAt(end) && start<end){

start ++;

end --;

}

if(start-1==end) return true;

else return false;

}

}

public static void dfs(String s,int start,List<List<String>> result,List<String> lst){

if(start==s.length()){

List<String> temp = new ArrayList<String>();

temp.addAll(lst);

result.add(temp);

return;

}

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

if(isPalindrome(s,start,i)==true){

lst.add(s.substring(start,i+1));

dfs(s,i+1,result,lst); //i+1

lst.remove(lst.size()-1);

}

}

}

public List<List<String>> partition(String s) {

List<List<String>> result = new ArrayList<List<String>>();

List<String> lst = new ArrayList<String>();

if(s.equals("")) return result;

dfs(s,0,result,lst);

return result;

}

}

Palindrome Partitioning II

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

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

For example, given *s* = "aab",  
Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.

题解：

public class Solution {

public int minCut(String s) {

int len = s.length();

int[][] p = new int[len][len];

int[] result = new int[len];

for(int i=len-1;i>=0;i--){

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

if(s.charAt(i)==s.charAt(j)&&(j-i<2||p[i+1][j-1]==1))

p[i][j]=1;

}

}

for(int i=len-2;i>=0;i--){

int min = Integer.MAX\_VALUE;

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

if(p[i][j]==1){

if(j==len-1) min=0;

else if(min>1+result[j+1])

min = 1+result[j+1];

}

}

result[i] = min;

}

return result[0];

}

}

10.2 Unique Paths

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?



Above is a 3 x 7 grid. How many possible unique paths are there?

**Note:** *m* and *n* will be at most 100.

题解：

public class Solution {

public int uniquePaths(int m, int n) {

if(m==0||n==0) return 0;

int[][] p = new int[m][n];

for(int i=0;i<n;i++) p[0][i] = 1;

for(int i=0;i<m;i++) p[i][0] = 1;

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

for(int j=1;j<n;j++)

p[i][j] = p[i-1][j]+p[i][j-1];

}

return p[m-1][n-1];

}

}

### 10.3 Unique Paths II

 Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

[

[0,0,0],

[0,1,0],

[0,0,0]

]

The total number of unique paths is 2.

**Note:** *m* and *n* will be at most 100.

题解：

public class Solution {

public int uniquePathsWithObstacles(int[][] obstacleGrid) {

int m = obstacleGrid.length;

int n = obstacleGrid[0].length;

if(m==0||n==0) return 0;

int[][] p = new int[m][n];

boolean flag = false;

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

if(obstacleGrid[0][i]==1) flag = true;

if(flag==true) p[0][i] = 0;

else p[0][i] = 1;

}

flag = false;

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

if(obstacleGrid[i][0]==1) flag = true;

if(flag==true) p[i][0] = 0;

else p[i][0] = 1;

}

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

for(int j=1;j<n;j++){

if(obstacleGrid[i][j]==1) p[i][j]=0;

else{

if(obstacleGrid[i-1][j]==1) p[i][j] = p[i][j-1];

else if(obstacleGrid[i][j-1]==1) p[i][j] = p[i-1][j];

else p[i][j] = p[i-1][j]+p[i][j-1];

}

}

}

return p[m-1][n-1];

}

}

### 10.4 N-Queens

 The *n*-queens puzzle is the problem of placing *n* queens on an *n*×*n* chessboard such that no two queens attack each other.



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

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

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

[

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

"...Q",

"Q...",

"..Q."],

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

"Q...",

"...Q",

".Q.."]

]

题解：

public class Solution {

public static boolean check(int[] pos,int x,int y)

{

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

{

if(pos[i]==y)

{

return false;

}

if(x-i==y-pos[i] || i+pos[i]==x+y)

return false;

}

return true;

}

public static void solve(int n,List<String[]> result,int k,int[] pos){

if(k==n){

char[][] c = new char[n][n];

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

for(int j=0;j<n;j++){

if(j==pos[i]) c[i][j]='Q';

else c[i][j]='.';

}

}

String[] s = new String[n];

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

StringBuilder sb = new StringBuilder();

for(int j=0;j<n;j++)

sb.append(c[i][j]+"");

s[i] = sb.toString();

}

result.add(s);

return;

}

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

if(check(pos,k,i)==true){

pos[k]=i;

solve(n,result,k+1,pos);

pos[k]= -1;

}

}

}

public List<String[]> solveNQueens(int n) {

List<String[]> result = new ArrayList<String[]>();

int[] pos = new int[n];

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

solve(n,result,0,pos);

return result;

}

}

10.5 N-Queens II

Follow up for N-Queens problem.

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



题解：

public class Solution {

int count = 0;

public boolean check(int[] pos,int x,int y)

{

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

{

if(pos[i]==y)

{

return false;

}

if(x-i==y-pos[i] || i+pos[i]==x+y)

return false;

}

return true;

}

public void solve(int n,int k,int[] pos){

if(k==n){

count ++;

return;

}

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

if(check(pos,k,i)==true){

pos[k]=i;

solve(n,k+1,pos);

pos[k]=-1;

}

}

}

public int totalNQueens(int n) {

int[] pos = new int[n];

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

solve(n,0,pos);

return count;

}

}

10.6 Restore IP Addresses

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

For example:  
Given "25525511135",

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

题解：

public class Solution {

public void restore(String s,int start,int count,List<String> result,String t){

if(count==4){

if(start==s.length()){

result.add(t.substring(1));

}

return;

}

if((s.length()-start) > (4-count)\*3 || (s.length()-start) < (4-count)) return; //剪枝

int temp = 0;

for(int i=start;i<start+3;i++){

if(i<s.length()){

temp = temp\*10+s.charAt(i)-'0';

if(temp<=255&&temp>=0){

restore(s,i+1,count+1,result,t+"."+temp);

}

}

if(temp==0) break; //不允许前缀是0，但可以是单个0

}

}

public List<String> restoreIpAddresses(String s) {

List<String> result = new ArrayList<String>();

String t = "";

restore(s,0,0,result,t);

return result;

}

}

10.7 Combination Sum

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

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

**Note:**

* All numbers (including target) will be positive integers.
* Elements in a combination (*a*1, *a*2, … , *a*k) must be in non-descending order. (ie, *a*1 ≤ *a*2 ≤ … ≤ *a*k).
* The solution set must not contain duplicate combinations.

For example, given candidate set 2,3,6,7 and target 7,   
A solution set is:   
[7]   
[2, 2, 3]

题解：

public class Solution {

public void dfs(int[] a ,int target,int pos,List<List<Integer>> results,List<Integer> result)

{

if(target<0) return;

if(target==0)

{

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

t.addAll(result);

if(!results.contains(t))

results.add(t);

return;

}

for(int i=pos;i<a.length;i++)

{

result.add(a[i]);

**dfs(a,target-a[i], i ,results,result);**

result.remove(result.size()-1);

}

}

public List<List<Integer>> combinationSum(int[] candidates, int target) {

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

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

Arrays.sort(candidates);

dfs(candidates,target,0,results,result);

return results;

}

}

10.8 Combination Sum II

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

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

**Note:**

* All numbers (including target) will be positive integers.
* Elements in a combination (*a*1, *a*2, … , *a*k) must be in non-descending order. (ie, *a*1 ≤ *a*2 ≤ … ≤ *a*k).
* The solution set must not contain duplicate combinations.

For example, given candidate set 10,1,2,7,6,1,5 and target 8,   
A solution set is:   
[1, 7]   
[1, 2, 5]   
[2, 6]   
[1, 1, 6]

题解：

public class Solution {

public void dfs(int[] a ,int target,int pos,List<List<Integer>> results,List<Integer> result)

{

if(target<0) return;

if(target==0)

{

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

t.addAll(result);

if(!results.contains(t))

results.add(t);

return;

}

for(int i=pos;i<a.length;i++)

{

result.add(a[i]);

**dfs(a,target-a[i], i+1,results,result);**

result.remove(result.size()-1);

}

}

public List<List<Integer>> combinationSum2(int[] num, int target) {

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

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

Arrays.sort(num);

dfs(num,target,0,results,result);

return results;

}

}

10.9 Generate Parentheses

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

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

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

题解：

public class Solution {

public void dfs(int left,int right,List<String> result,String s){

if(left==0&&right==0){

result.add(s);

return;

}

if(left>0) dfs(left-1,right,result,s+"(");

if(right>left) dfs(left,right-1,result,s+")");

}

public List<String> generateParenthesis(int n) {

List<String> result = new ArrayList<String>();

String s = "";

dfs(n,n,result,s);

return result;

}

}

10.10 Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.



A sudoku puzzle...



...and its solution numbers marked in red.

题解：

public class Solution {

public boolean check(int a,int b,char value,char[][] board){

for(int i=0;i<board[0].length;i++){

if(board[a][i]==value) return false;

}

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

if(board[i][b]==value) return false;

}

for(int i=(a/3)\*3;i<(a/3)\*3+3;i++){

for(int j=(b/3)\*3;j<(b/3)\*3+3;j++){

if(board[i][j]==value) return false;

}

}

return true;

}

public boolean solve(char[][] board,int pos) {

if(pos==81){

return true;

}

int i=pos/9;

int j=pos%9;

if(board[i][j]=='.'){

for(char k='1';k<='9';k++){

if(check(i,j,k,board)==true){

board[i][j]=k;

if(solve(board,pos+1)) return true;

board[i][j]='.';

}

}

return false;

}else

return solve(board,pos+1);

}

public void solveSudoku(char[][] board) {

solve(board,0);

}

}

### 10.11 Word Search

 Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,  
Given **board** =

[

["ABCE"],

["SFCS"],

["ADEE"]

]

**word** = "ABCCED", -> returns true,  
**word** = "SEE", -> returns true,  
**word** = "ABCB", -> returns false.

题解：

public class Solution {

public static boolean dfs(char[][] board,String word,int pos,int x,int y,int[][] index){

if(pos==word.length()) return true;

if(x<0||y<0||x>=board.length||y>=board[0].length) return false;

if(board[x][y]!=word.charAt(pos)) return false;

if(index[x][y]==1) return false;

index[x][y] = 1;

boolean res = dfs(board,word,pos+1,x-1,y,index)||dfs(board,word,pos+1,x,y-1,index)||dfs(board,word,pos+1,x+1,y,index)||dfs(board,word,pos+1,x,y+1,index);

index[x][y] = 0;

return res;

}

public boolean exist(char[][] board, String word) {

int[][] index = new int[board.length][board[0].length];

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

for(int j=0;j<board[0].length;j++){

if(dfs(board,word,0,i,j,index)) return true;

}

}

return false;

}

}

11 分治法

11.1 Pow(x, n)

Implement pow(*x*, *n*).

题解：

public class Solution {

public double power(double x,int n){

if(n==0) return 1.0;

double value = power(x,n/2);

if(n%2==0) return value\*value;

else return value\*value\*x;

}

public double pow(double x, int n) {

if(n>0) return power(x,n);

else

return 1.0/power(x,-n);

}

}

### 11.2 Sqrt(x)

 Implement int sqrt(int x).

Compute and return the square root of *x*.

题解：

public class Solution {

public int mySqrt(int x) {

double error = 0.000000001f;

double high = x;

double low = 0;

while(high-low> error){

double mid = (high+low)/2;

if(mid>x/mid){

high = mid;

}else {

low = mid;

}

}

return (int)Math.floor(high);

}

}

12 贪心法

### 12.1 Jump Game

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

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

Determine if you are able to reach the last index.

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

A = [3,2,1,0,4], return false.

题解：

public class Solution {

public boolean dfs(int[] nums,int pos){ //dfs会超时

if(pos>=nums.length-1) return true;

if(pos<nums.length-1&&nums[pos]==0) return false;

boolean res = false;

for(int i=1;i<=nums[pos];i++)

res = res||dfs(nums,pos+i);

return res;

}

public boolean canJump(int[] nums) {

int reach = 1;

for(int i=0;i<reach&&reach<nums.length;i++){

reach = reach>(i+1+nums[i])?reach:i+1+nums[i];

}

return reach>=nums.length;

}

}

12.2 Jump Game II

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

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

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

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

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

题解：

public class Solution {

public int jump(int[] A) {

if(A.length<=1) return 0;

int curJump=1;

int left = 0,right = A[0];

while(left<=right)

{

if (right>=A.length-1 )

return curJump;

int t=right;

while(left<=t)

{

right=right>A[left]+left?right:A[left]+left;

left ++;

}

left = t+1;

curJump++;

}

return -1;

}

}

### 12.3 Best Time to Buy and Sell Stock

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

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

题解：

public class Solution {

public int maxProfit(int[] prices) {

if(prices.length<2) return 0;

int min = prices[0];

int sum = 0;

for(int i=1;i<prices.length;i++){

sum = sum>(prices[i]-min)?sum:prices[i]-min;

if(prices[i]<min) min = prices[i];

}

return sum;

}

}

12.4 Best Time to Buy and Sell Stock II

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

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

题解：

public class Solution {

public int maxProfit(int[] prices) {

if(prices.length<2) return 0;

int sum = 0;

int t = prices[0];

for(int i=1;i<prices.length;i++){

if(prices[i]-t >0) sum += prices[i]-t;

t = prices[i];

}

return sum;

}

}

### 12.5 Longest Substring Without Repeating Characters

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

题解：

public class Solution {

public int lengthOfLongestSubstring(String s) {

int max=0,len=0,start=0;

Map<Character,Integer> map = new HashMap<Character,Integer>();

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

{

if(map.get(s.charAt(i))==null)

{

map.put(s.charAt(i),i);

len += 1;

}

else

{

int temp = map.get(s.charAt(i));

if(temp>=start)

{

start = temp+1;

len = i-temp;

}else{

len += 1;

}

map.put(s.charAt(i),i);

}

if(len>max) max = len;

}

return max;

}

}

12.6 Container With Most Water

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

Note: You may not slant the container.

题解：

public int maxArea(int[] height) {

int l=0,r=height.length-1;

int max = 0;

while(l<r){

int min = height[l]<height[r]?height[l]:height[r];

max = max>(r-l)\*min?max:(r-l)\*min;

if(height[l]<height[r]){

int pos = l+1;

while(height[pos]<height[l]) pos ++;

l = pos;

}else{

int pos = r-1;

while(height[pos]<height[r]) pos --;

r = pos;

}

}

return max;

}

13 动态规划

13.1 Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

[

[2],

[3,4],

[6,5,7],

[4,1,8,3]

]

The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

**Note:**  
Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.

题解：

public class Solution {

public int minimumTotal(List<List<Integer>> triangle) {

int m=triangle.size();

int n=triangle.get(m-1).size();

int[][] f = new int[m][n];

for(int i=0;i<n;i++) f[m-1][i]=triangle.get(m-1).get(i);

for(int i=m-2;i>=0;i--){

for(int j=0;j<triangle.get(i).size();j++){

int min = f[i+1][j]<f[i+1][j+1]?f[i+1][j]:f[i+1][j+1];

f[i][j] = min+triangle.get(i).get(j);

}

}

return f[0][0];

}

}

### 13.2 Maximum Subarray

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

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

[click to show more practice.](https://leetcode.com/problems/maximum-subarray/)

**More practice:**

If you have figured out the O(*n*) solution, try coding another solution using the divide and conquer approach, which is more subtle.

题解：

public class Solution {

public int maxSubArray(int[] nums) {

int[] f = new int[nums.length];

f[0]=nums[0];

for(int i=1;i<nums.length;i++){

f[i] = f[i-1]+nums[i]>nums[i]?f[i-1]+nums[i]:nums[i];

}

int max=Integer.MIN\_VALUE;

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

if(f[i]>max) max=f[i];

}

return max;

}

}

13.3 Palindrome PartitioningII

13.4 Maximal Rectangle

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

题解：