

LeetCode PROBLEMS

V 0.1



August 29, 2013

bY qUN zHAO

Beijing, China

## **Two Sum**

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

The function two Sum 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

* **Solving ideas**

Basic solution uses two loop to get the results, time complexity is O(n^2), space complexity is O(1);

Another solution uses a Hash Map to store the previous numbers and there indexes, time complexity is O(n), space complexity is O(n).

* **Attentions**

There are two conditions which will complicate the problem, one is duplicate numbers in the given array, the other is integer overflow or underflow, such as given target = xx, but there are two numbers x1, x2 in the array , sum(x1, x2) will overflow / underflow of integer but the results of sum equals to target by chance.

**For overflow or underflow cases. The solution with Hash Map is valid too, but for two loop solution it will fail.**

* **Test cases**
* **Solutions**

## 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

* **Solving ideas**

Basic problem

* **Attentions**

Beware of the end status of carry, if it is not 0, we should add a node of that value to the tail.

* **Test cases**
* **Solutions**

## 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.

* **Solving ideas**

Brute force method is use double loop to traverse all positions, time complexity is O(n^ 2) , space complexity is O(1).

Another solution is use an array or Hash Map to record the exits char and their positions. Keeping a window which only contains unique character, change left and right boundary according to which character encounter. Time complexity is O(n), space complexity is O(1) ( int[256] ).

* **Attentions**

Beware of the update of the left boundary, when right is a exists character,

left = max(left, pos[c[right]])

* **Test cases**
* **Solutions**

## 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)).

* **Solving ideas**

Basic solution with two traversal, time complexity is O(m + n), space complexity is O(1).

Because the two given arrays are sorted, so we can consider whether it is possible to use binary search.

A

B

Pb

Pa

Pb = min(k / 2 , LenB);

Pa = k – pb;

If( A[Pa] > B[Pb] ) {B1 abandoned }

If(A[Pa] < B[Pb]){A1 abandoned}

If(A[Pa] == B[Pb]){return A[Pa];}

A1

A2

B1

B2

* **Attentions**

Beware of the boundary conditions.

1. When one array B is empty, then return A[k -1]
2. When K is 1 , return min (A[0], B[0])

* **Test cases**
* **Solutions**

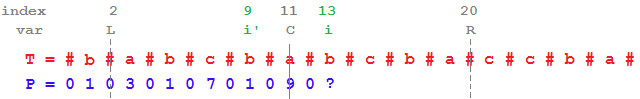
## Longest Palindromic Substring

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.

* **Solving ideas**

Basic solution use double loop to check every substring is valid palindromic string. Time complexity is O(n^2) space complexity is O(1).

Another O(n) solution utilize the symmetric of palindromic string



C is the center of a known palindromic string, so for position i, it is have the same palindromic length with the symmetric position I’, it will be min( res[I’] , R – i) and update res[i].

* **Attentions**

Should construct a new string with ‘#’ inserted between each char.

* **Test cases**
* **Solutions**

## ZigZag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N

A P L S I I G

Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

string convert(string text, int nRows);

convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

* **Solving ideas**

With the graph above, we can know that. Every (2 \* n – 2 ) character will construct a loop and for line 0 and line n-1 we should only add one number , other lines we should add two number.

* **Attentions**
* **Test cases**
* **Solutions**

## Reverse Integer

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

* **Solving ideas**

Get tail number per time and construct new integer.

* **Attentions**

Beware of negative or positive and for some integer the overflow or underflow will happen.

* **Test cases**
* **Solutions**

## 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.

Requirements for atoi:

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.

* **Solving ideas**

Check positive or negative first and for some character which is not number, ignore or return.

* **Attentions**

1. Positive or negative
2. Overflow or underflow
3. Not numeric character

* **Test cases**
* **Solutions**

For the overflow or underflow, two ways to avoid it.

A simple way is using long instead of Integer. (simple but need more extra space 8 bytes)

The other way: (num == Integer.MAX\_VALUE / 10 && cur > 7 || num > Integer.MAX\_VALUE / 10)

## Palindrome Number

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

Some hints:

Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

* Solving ideas

Two ways to solve the problems. One uses O(n) space to transform integer to string and check whether the inverse string is same to original string

The other check each position in the head and tail. Time complexity is O(n) space complexity is O(1), but will traverse two times.

Another way to solve it with a recurrence. This solution is very interesting, use DFS to check head with tail, in the following graph, only judge x and y‘s tail position.

X = 131 | Y = 131

X = 13 | Y = 131

X = 1 | Y = 131

X = 13 | Y = 13

X = 131 | Y = 1

* Attentions

Beware of the negative integers.

For solution two, beware of that when x = 10 in the while.

For solution that reverse integer, the results maybe correct, because when original number is a palindrome, it will not overflow, if it is not, it maybe will overflow, but when we got the reversed number, it is not equals to original one, so the results is correct.

* Test Cases
* Solutions

## 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("aa", "a\*") ? true

isMatch("aa", ".\*") ? true

* Solving ideas

Just consider the condition with ‘\*’, for that condition, we should return

match(is+1,ip) || match(is,ip+2) || match(is+1,ip+2).

But for a\*b and aaaaaaaab, it is efficiently to calculate match(is,ip+2) if not true, calculate match(is+1,ip+2). If no match , return match(is,ip+2)

Another solution will use dynamic programming.

* Attentions

Beware of the terminal conditions in the reccurrence

* Test cases
* Solutions

## 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.

* Solving ideas

Basic solution has O(n ^ 2) time complexity , it will get all the possible containers and calculate the maximum.

A O(n) solution is from head and tail. If height of head bigger than height of tail, head++, other tail--.

* Attentions
* Test cases
* Solutions

## Integer to Roman

Given an integer, convert it to a roman numeral.

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

## Roman to Integer

Given a roman numeral, convert it to an integer.

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

## Longest Common Prefix

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

* Solving ideas

Take strs[0] as the basic string, check from head char to tail char, if current char not exists in one of the other strs, return the substring.

* Attentions

Other strs may have lower character than strs[0], so we should check the length of strs[i] first.

## 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)

* Solving ideas

Firstly, sort the array. O(nlogn), use double loop, first pointer from head to tail -2 , second and third pointer use left array from head and tail, if sum of three number bigger than 0, high should lower, else if smaller than 0, low should higher, then add to results.

Time complexity is O(n^2).

* Attentions

Beware of the duplicated cases, first in the first loop, if I is not 0 and num[i] == num[I - 1], should ignore this number. In inner loop, same as I, if low is not I + 1 and num[low] == num[low -1], continue, the same for high

## 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).

* Solving ideas

Time complexity is O(n ^ 2). Same as 3 Sum, just record the min value to target.

* Attentions

First, should be careful with the update of min value

Math.abs(sum - target) < Math.abs(min – target)

Second, should update low and high normally.

## 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)

* Solving ideas

See graph below.

P1

P3

P4

P2

From the graph above, we should sort the array first O(nlogn) and then set four pointer, P1 from head to tail, P2 from tail to head, P3 bigger than P1 and P4 smaller than P2, when sum of the four number equals to target, record the four number, if smaller than the target, P3 add one, else P4 minus one.

* Attentions

There are some duplicated cases.

1. When number [ P1 - 1 ] equals to number[P1] .
2. Number[P3 - 1] equals to number[P3]
3. Number[P2] equals to number[p2 +1]
4. Number[p4] equals to number[p4 +1]

Totally time complexity is O(n ^ 3) . space complexity is O(1)

## 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"].

* Solving ideas

Basic DFS and there is a iterative solution

* Attentions

## Remove Nth Node From End of List

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

For example,

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

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

Note:

Given n will always be valid.

Try to do this in one pass.

* Solving ideas

Time complexit is O(n), space time complexity is O(1), and with only one pass.

Use two pointers, first point to head, second point to head + n node, and all move to next until the second pointer in the end, remove the first’s next node.

* Attentions

The second pointer maybe null, in this condition, we should remove head node.

## 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.

* Solving ideas

Use a stack to track the characters. If current character is right character, should pop out left character or false, time complexity is O(n), space complexity is O(n)

* Attentions

Check the final stack status, if empty return true else return false

## 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:

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

* Solving ideas

Fill in the character from left to right. Check current status, when current ‘(‘ number bigger than ‘)’ number, we can fill in ‘(’ or ‘)’, if equals to, we only can fill in ‘(’, do it recursive

* Attentions

Check the final status. Left character number should equals to right character number

## 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.

* Solving ideas

Check the smaller of head of l1 and head l2, link it to the result list. Time complexity is O(m+n), space complexity is O(1)

* Attentions

Beware of when l1 is null or l2 is null

Merge k Sorted Lists

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

* Solving ideas

Time complexity is O(nlogn), space complexity is O(1), like mergesort, merge each pair list until left one list.

* Attentions

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.

* Solving ideas

There are three solutions, one is recursive solution, another is use a dummy node in the head and use three pointer to swap the node pair, finally solution use no extra space, but more complex, we should check the if pre pointer is null or not.

* Attentions

For traversal solution without extra space, should beware of the status of pre pointer.

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

* Solving ideas
* Attentions

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].

* Solving ideas

Use two pointer, one point to index which can put valid number, second pointer point to current index, if current number bigger than pre number of the first pointer, set it to first pointer position and move pointer to it’s next.

Time complexity is O(n), space complexity is O(1)

* Attentions

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.

* Solving ideas

Two pointer, first pointer point to index which can set to valid number, second pointer point to current number, if current number is valid, move to first pointer’s index.

Time complexity is O(n), space complexity is O(1)

* Attentions

Implement strStr()

Implement strStr().

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

* Solving ideas

Basic solution, one pointer from head to tail, check if current

* Attentions

## Divide Two Integers

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

* **Solving ideas**

The time complexity is O(logn), for every integer such as 5, it can be expressed as 2^0 + 2^2, so for 17 / 3, we can check 3 \* 2 ^x bigger than 17 firstly, x is 3, and from 3 to 0, if left is bigger than I , and I to result

* **Attentions**

Beware of divisor is 0 and the symbol.

## Substring with Concatenation of All Words

You are given a string, S, and a list of words, L, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.

For example, given:

S: "barfoothefoobarman"

L: ["foo", "bar"]

You should return the indices: [0,9]. (order does not matter).

* Solving ideas

There is a O(n) time complexity solution, but it is so complex. My solution is O(m \* n), use a hashmap to record the list of words and from head to tail of String S, check if current position is the right position, such as, from index I , get n words and check these words whether equals to words in hashmap.

* Attentions

## 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

* Solving ideas

We should get the first position which is the higher position will be changed. Such as 1342, the first index will be changed is 1.

The position can be got by check whether pre position is bigger than it.

When got the first position, we should get the smaller number which bigger than number in this index, the number in this example is 4. And then change their position, in the last, reverse all the number left of the first postion.

Time complexity is O(n)

* Attentions

Beware of case : 4321, if we can’t get the first position, we should reverse all numbers.

## 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.

* Solving ideas

Time complexity is O(n), space complexity is O(n).

Use a stack to record the position. When current char is ‘(’, push to the stack, other check whether the stack is empty, if empty, push this char, if not ,check the top char is ‘(’ or not. If is ‘(’, pop and calculate the length and update the max length, if not, push to the stack.

* Attentions

## 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.

* Solving ideas

Time complexity is O(logn)

Use binary search to find the position. If number from low to mid is sorted and target between this interval, we set high to mid minus one. Else set low to mid plus one.

If number from mid to high is sorted, and target between this interval, we set low to mid plus one else set high to mid minus one.

* Attentions

Beware of that when middle is bigger than target, the target maybe equals to high.

There is a better way to solve this problem

## 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].

* Solving ideas

Time complexity is O(logn), use binary search to find the left target and then find the right target.

* Attentions

## 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

* Solving ideas

Time complexity is O(logn).

Use binary search.

* Attentions

Beware of the final returned position when can’t find the target, we should return low, because when low equals to high and can’t find the target, low will plus one, in this situation, we should return low.

## Valid Sudoku

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

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



A partially filled sudoku which is valid.

* Solving ideas

Time complexity is O(n^2), for one traversal, use space O(3 \* n ^ 2) to record which row, column, block contains which number, such as row[1][9] for board[1][x] = 9, that is means, number 9 is already in row one. The same to column and block.

The index of block is i – I % 3 + j /3.

Another solution only use space O(n), but should with two traversal. Check each row, column and block.

* Attentions

## 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.

* Solving ideas

Use DFS + backtracking to solve this problem.

* Attentions

Beware of that when one way is impossible we should set the current position to ‘.’ (Don’t forget the backtracking progress)

## 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 nth sequence.

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

* Solving ideas

Easy problem, just count how many character same as previous.

Time complexity is O(n^2) space complexity is O(1)

* Attentions

Beware of the 0 position

## 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 (a1, a2, … , ak) must be in non-descending order. (ie, a1 < a2 < … < ak).

The solution set must not contain duplicate combinations.

For example, given candidate set 2,3,6,7 and target 7,

A solution set is:

[7]

[2, 2, 3]

* Solving ideas

Basic dfs

* Attentions

## 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 (a1, a2, …, ak) must be in non-descending order. (ie, a1 ? a2 ? … ? ak).

The solution set must not contain duplicate combinations.

For example, given candidate set 10,1,2,7,6,1,5 and target 8,

A solution set is:

[1, 7]

[1, 2, 5]

[2, 6]

[1, 1, 6]

* Solving ideas

Basic dfs

* Attentions

Beware of the repeated number.

## 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.

* Solving ideas

Time complexity is O(n) space complexity is O(1).

If current number is valid (bigger than 0 and smaller than N), swap this value with A[value -1], do it from head to tail

* Attentions

When current value equals to A[value -1], we should stop and do the next number

## 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.



* Solving ideas

Time complexity is O(n), space complexity is O(1), with two traversal.

First find the middle of the integer, then from head to middle calculate the waters , and from tail to middle calculate the waters.

* Attentions

## Multiply Strings

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

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

* Solving ideas

Just from tail to head, get multiply results of each position and format it to right order in the final

* Attentions

Beware of the order of the number and num1 or num2 maybe ‘0’

## 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

* Solving ideas

There are three solutions, recursive solution, iterative solution and DP solution.

For recursive solution, it is easy to implement, but will cost large resource, iterative solution is more complex but efficient, DP solution will cost n \* m space.

For iterative solution, we should record last ‘\*’ position, if not match, backtrack to last index and s’s index add one.

* Attentions

Beware of two cases: aa with \*, for this case, ip will be p.length(), the other case is aa with \*\*\*, beware of ‘\*’ in the tail.

## 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.)

* Solving ideas

Time complexity is O(n), space complexity is O(1).

We should use two variable (last, max) to record the current max index we can reach.

For last, will not update when current index not bigger than it. And max will update every move, that’s means, when we move out of last, we should add one to step and update the last to current max, this is the max index we can reach in this step.

* Attentions

Max index should be updated every time.

## 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].

* Solving ideas

For this problem, it is easy to give the DFS solution, but there is another iterative solution. In some interview such as M$, the interviewer will ask you to transform recursive solution to iterative solution.

For iterative solution, it is an insert progress. For example, we have 1 now and want add 2 in. we insert 2 in the 0 position and 1 position, get 2,1 and 1,2, the same is to 3.

* Attentions

## 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].

* Solving ideas

Basic DFS solution. There is iterative solution.

* Attentions

Check the duplicated numbers.

## 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?

* Solving ideas

Solution is a recursive progress. Just like the graph below.



It is a recursive progress. Layer count is n / 2. And for each layer, we should do some transform, just like the graph shows.

* Attentions

Beware of each level’s start index and the transform function from original to new position.

## Anagrams

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

Note: All inputs will be in lower-case.

* Solving ideas

First sort the string, and use a hashmap to store all unique strings.

Use more space, but time complexity is low.

* Attentions

## Pow(x, n)

Implement pow(x, n).

* Solving ideas

There are two kind of solutions, one is recursive, the other is iterative, but they don’t share any point on solving the problem.

For recursive solution, we can calculate pow(x, n/2) and multiply itself. For odd number, we should multiply one x more in each time.

For iterative solution, we decompose n to some numbers which sum is n.

Such as 7 = 2^0 + 2^1 + 2^2. And the binary format of 7 is 0111, so we can check whether the tail bit is 1, if is we add one number to results, the number changed and n changed each time.

* Attentions

Beware of that n maybe negative number, in iterative solution. We should use long to avoid the overflow problem and in recursive solution, we should divide x when n is odd.

## 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.

* Solving ideas

The point to solve this problem is how to store the queens’ index and how to check its’ valid, we use an array to store queens’ index, each number stands for which row the queen in.

For checking, when we set a new queen, we first check whether has same number in the before index and check if queen[index] – queen[current] equals to index – current.

* Attentions

## N-Queens II

Follow up for N-Queens problem.

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



* Solving ideas

The same to N-Queens

* Attentions

## 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.

* Solving ideas

If we know the maximum subarray’s sum of index I, so for i+1, if sum(i) is below zero, the sum(I + 1) is num[i+1] else will be num[i+1] + sum(i), and from head to tail, get all the max.

* Attentions

The initial of max and current.

## Spiral Matrix

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

For example,

Given the following matrix:

[

[ 1, 2, 3 ],

[ 4, 5, 6 ],

[ 7, 8, 9 ]

]

You should return [1,2,3,6,9,8,7,4,5].

* Solving ideas

Just like rotate image, but now matrix is not a square, it’s a rectangle.

So for each level, we should add top, right, bottom, left.

* Attentions

Beware of that when there is only one row or one column

## 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.

* Solving ideas

Just record max index which we can reach in each step.

* Attentions

Initial of the max and case that can’t reach the tail.

## Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

For example,

Given [1,3],[2,6],[8,10],[15,18],

return [1,6],[8,10],[15,18].

* Solving ideas

First sort the array according to its first number and then merge intervals from first to last, there are three cases:

* Attentions

Beware of the three cases.

## Insert Interval

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

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

Example 1:

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

Example 2:

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

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

* Solving ideas

From first to last, combine the newinterval with current interval, there will be some cases:

newInterval before the current interval, for this case, add new to result and set current to new.

New after current interval, add current and continue.

Overlap, for this case, update new’s start and end.

* Attentions

Don’t forget add new in the last out of the for loop.

## 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.

* Solving ideas

Use two variable to record the length. Current update each time, when current character is blank, set current to 0 and update last, else add one to current.

* Attentions

If current is zero in the last, return last, else return current.

When character is blank in the loop, check whether last is zero, if not, don’t’ set current to last.

## Spiral Matrix II

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example,

Given n = 3,

You should return the following matrix:

[

[ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ]

]

* Solving ideas

This problem is easier than Spiral Matrix I, because it is N \* N matrix now.

Just an iterative progress and each time add one to variable.

* Attentions

Beware of that when n is odd, we should add the center of matrix to the results. Such as num[1][1] for n equals to 3.

## 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):

"123"

"132"

"213"

"231"

"312"

"321"

Given n and k, return the kth permutation sequence.

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

* Solving ideas

We can use Next permutation above to generate next each time, but it is not the best answer.

Another efficient solution is generating number in each position directly, time complexity is O(n), space complexity is O(n).

For example n = 3. And k = 3.

The first position is 3 -1 / 2! = 1 so num[1] = 2, we got the first number 2. The same to left numbers.

* Attentions

## 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.

* Solving ideas

First get the length of the list and link tail to head. Next get the real steps n %= length.

Get the length from head n = len – n -1 and keep a pointer to the n position from head.

n.next is the new head and set n.next equals to null.

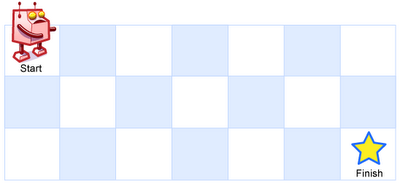
* Attentions

## 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.

* Solving ideas

F(I,j) = f(i-1,j) + f(I,j-1)

* Attentions

Start from position x = 1 and y =1, the initial of first row and first column is all one.

## 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.

* Solving ideas

In this problem, keep an array with size m \* n and if current cell is blocked (1 in the original matrix) set current value to 0, else current value equals to sum of upper and left cell.

* Attentions

## Minimum Path Sum

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

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

* Solving ideas

Easy problem, from left up to right bottom only add Min of upper or left number.

Time complexity is O(n\*m), space complexity is O(1)

* Attentions

The initial of first row and first column.

## 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.

* Solving ideas

Easy problem, just link lower number each time.

* Attentions

## Add Binary

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

For example,

a = "11"

b = "1"

Return "100".

* Solving ideas

Easy problem, just from tail to head, add two strings.

* Attentions

Don’t forget the carry in the final.

## 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.

* Solving ideas

It’s a DFA

* Attentions

## Plus One

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

* Solving ideas

Easy problem.

* Attentions

## Text Justification

Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly L characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

For example,

words: ["This", "is", "an", "example", "of", "text", "justification."]

L: 16.

Return the formatted lines as:

[

"This is an",

"example of text",

"justification. "

]

Note: Each word is guaranteed not to exceed L in length.

* Solving ideas

This problem is so tedious that needs too much code to solve.

* Attentions

## Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of x.

* Solving ideas

When x is negative, return zero or others.

Binary search, but beware of the right number condition.

* Attentions

Beware of the overflow, don’t use mid \* mid directly, but use x / mid instead.

When x / mid equals to mid or x / mid bigger than mid but x / (mid + 1) smaller than (mid + 1), we get the results.

## 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?

* Solving ideas

Basic DP, easy problem.

Same to fibnaci number, can be get in O(1) time complexity.

But easy solution is O(n) and space complexity is O(1). Use two variable to stand for f[n-1] and f[n-2] and update them each time.

* Attentions

## 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.

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".

* Solving ideas

First split the string with separator of “/”, use a stack to store the paths.

For case “.” Or “”, continue, for case “..”, if stack is not null, pop out an element.

* Attentions

Beware of the end state, when the stack is null, we should return “/”.

## Edit Distance

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

You have the following 3 operations permitted on a word:

a) Insert a character

b) Delete a character

c) Replace a character

* Solving ideas

It is a basic DP problem, but basic solution use m \* n space.

An optimizing solution use two array with size of m. and update them each time.

The best solution use one array, find details in below image.



We only need use three variable, and update them each time when calculate now from left to right.

* Attentions

The initial of dia, up, left.

## Set Matrix Zeroes

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

* Solving ideas

Basic solution use two array to record which row and column should be set to zero.

Another solution will cost no extra space. The idea is so straightforward, use first row and column to instead of the extra array.

But we should record whether we should set first row and column to zero.

* Attentions

When reset the number, we should start from x = 1, y =1.

## 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,

[

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

Given **target** = 3, return true.

* Solving ideas

See it as an array and use binary search to find the target.

* Attentions

## 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 supposed to use the library's sort function for this problem.

* Solving ideas

Use three pointer, one points to head, stands for the index which can put 0, second traverses from head to tail, stands for current number which should be processed. The third pointer points to tail, stands for first index which can put 2.

We should swap numbers of pointer one / three with second, swap 0 to head and swap 2 to tail, then 1 left to middle.

* Attentions

## Minimum Window Substring

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

For example,  
S = "ADOBECODEBANC"  
T = "ABC"

Minimum window is "BANC".

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

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

* Solving ideas

The best solution with time complexity of O(n).

We should keep a window, in the window contains all character in the given string and shrink when the window’s right border advanced.

* Attentions

## 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],

]

* Solving ideas

Basic solution use recurrence to get the results, such as n = 4, k = 2, we should select 1 and select 1 number from 2 to 4, or select 2 and select 1 number from 3 to 4, it is easy.

But there is another iterative solution.

* Attentions

## Subsets

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

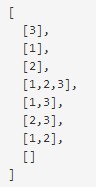
Note:

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:



* Solving ideas



* Attentions

Beware of that when add new number to exists array, we should new one array instead of the old one.

## 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.

* Solving ideas

Basic DFS problems. Use a HashSet to record which index was visited, don’t forget reset the visited number to false when can’t get result from that branch.

* Attentions

## 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].

* Solving ideas

Time complexity is O(n) and space complexity is O(1).

Just use a variable to record how many times the current number occurred.

* Attentions

## 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.

* Solving ideas

Same as previous problems. Firstly, judge which part can be abandoned.

* Attentions

Beware of the equals condition, when middle number equals to high or low number, just add one to the high or low.

## 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.

* Solving ideas

Easy problem, just check whether next node is bigger than the previous one.

* Attentions

## 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.

* Solving ideas

Just be sure of which is one valid node.

One case, next node is null, second case, next node’s value is bigger than current node’s value.

* Attentions

When current node isn’t valid, we should move pointer to next node which value is bigger than current node.

## 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.

* Solving ideas

For each bar, we should make sure its left bounder and right bounder, for example, 5 in above graph, the left bounder is 5 and right bounder is 6, so for this bar, the maximum size will be 5 \* 2 = 10.

Traverse the array and get every bar’s max size, then we get the results.

The point is how to do it in O(n).

We can use one stack to get every bar’s bounder, for right bounder, the first one which is lower than current bar, so if next bar’s height higher than current bar, push it into the stack else, we got current bar’s right bounder. For left bounder, if current bar is higher than stack’s peek element, we know the left bounder is the peek element’s height.

Beware of the condition that when stack is empty.

* Attentions

## 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.

* Solving ideas

For every line, we can see it as an array of rectangle, so easy to calculate the height of each number.

Now use the algorithm above can solve the problem, with time complexity of O(n \* m)

But above solution use one stack, if the matrix is large, it will overflow. So we use two extra array instead of the stack, for each rectangle, we only need to get it’s left one which is smaller than it and the right one which is smaller than it.

We should make use of the characteristic of the matrix to calculate the left and right bounder.

Current left bounder is the maximum of above position’s left bounder or the position which number is zero. The same to right bounder calculation.

* Attentions

## 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.

* Solving ideas

Just use two node to link the smaller node and the bigger number.

There is another easier solution, but with two extra node.

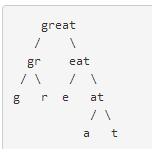
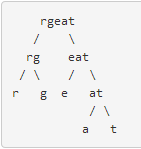
First node initialed with value -1 and link smaller node together. Another node initialed with value x and link bigger node together, in the last just link the first tail node’s next to second node’s next node.

* Attentions

## Scramble String

Given a string *s1*, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of *s1* = "great":

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".

We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".

We say that "rgtae" is a scrambled string of "great".

Given two strings *s1* and *s2* of the same length, determine if *s2* is a scrambled string of *s1*.

* Solving ideas

There are two solutions. Iterative and recursive solution. For recursive solution, we should whether A’s substring is scramble of B’s substring and do it recursively.

For iterative solution, we use DP, f(i,j,n) stands for substring from A[i] and substring from B[j] with length of n is scramble or not, and f(i, j, n) = || ((f(i, j, m) && f(i + m, j + m, n - m)) || f(i, j + n - m, m) && f(i + m, j, n - m)) for 1 < m < n.

## Merge 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 to hold additional elements from B. The number of elements initialized in A and B are m and n respectively.

* Solving ideas

Basic problem, time complexity is O(m+n), put the bigger one to end of array A.

* Attentions

## 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.

* Solving ideas

Got the formula from Wikipedia.

* Attentions

## Decode Ways

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

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.

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

The number of ways decoding "12" is 2.

* Solving ideas

Time complexity is O(n), space complexity is O(n), it’s a DP solution.

For position x, if x is not ‘0’, times[x] should add times[x-1], if x and x -1 can combine to valid number (x -1 is ‘1’ or x -1 is ‘2’ and x is smaller than ‘7’)

* Attentions

Beware of the position 0 and 1.

## 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],

[]

]

* Solving ideas

The same to Subset I, but should beware of the repeated number, for that case, we should use a variable to record last added items.

* Attentions

## 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.

* Solving ideas

One graph can show the solving ideas



We should move Head node to after position of the PreM node.

* Attentions

Update head to pre when we move head to other position.

## 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)

* Solving ideas

Just a DFS problem, beware of the valid status of the number.

* Attentions

## 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?

* Solving ideas

Easy for recursive solution, but complex for iterative solution.

For iterative solution, use a current pointer and a stack, if current pointer is null, we should pop out a node and set to current pointer and print the node’s value, set current pointer to node’s right child. If not null, just push current node and set current pointer to its left child.

* Attentions

## 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

* Solving ideas

Time complexity is O(n ^ 2), space complexity is O(n).

For n = 3. We should make sure that when root is 1, how many unique trees and same for 2 and 3.

When root is 1, we know that the left tree has no node, and right tree has 2 nodes, so if we know how many unique trees with 2 nodes, we can know how many unique trees with 3 node and root is 1, the same for root is 2 and 3.

So we calculate the number from 1 to n and get the final results.

* Attentions

## 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

* Solving ideas

A DFS problem, for n = 3, we should generate all trees with root equals to 1, 2, 3. And when root is 2, we should generate all left trees with numbers from 1 to 1 and right trees with numbers from 3 to 3. So it is an iterative progress.

* Attentions

## Interleaving String

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

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

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

* Solving ideas

An DP problem, with time complexity of O(n\*m) and space complexity of O(n)

Transfer function:

F(i, j) = f(i - 1, j) && s1.charAt(i -1) == s3.charAt(i+j-1) || f(i,j-1) && s2.charAt(j-1) == s3.charAt(i+j-1)

* Attentions

Beware of the initial of the first row and first column.

## 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.
* Solving ideas

Two solution, in-order traverse and check whether the number array is ordered, with extra O(n) space and time complexity is O(n).

Another solution without extra space, will recursive check the sub-tree of each node.

Such as for node A, for left sub-tree, A’s value will be the maximum value for left sub-tree and minimum value for right sub-tree.

* Attentions

## 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?

* Solving ideas

If with extra space, we can traverse in-order and get the ordered array, then check which two number is not in order.

Without extra space, we should record the two node which is not in order in the traversal.

For example, the two number which is not in order is 5 and 12.

2, 3, 4, 12, 7, 9, 10, 5, 13

We can see that, the first number should be 12 (pre node’s value bigger than current node) and the second number should be 5 (pre node’s value should be smaller than current node), so we when we check the false order in traversal we should set node one to pre and don’t reset again, in the next time, set node two to current node.

* Attentions

We should use a common variable to record the pre node pointer.

## 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.

* Solving ideas

Easy recursive problem.

Another solution. Use two queue to record all the left nodes and all the right nodes, and check whether they are same.

* Attentions

## 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.

* Solving ideas

Easy recursive solution. Just check whether Node A’s left is symmetric with Node B’s right and A’s right is symmetric with B’s left.

Use two Queue to implement the iterative solution. Poll out Node A from Queue A and poll out Node B from Queue B and check whether they are same, then add A’s left to Queue A and B’s right to Queue B and A’s right to Queue A and B’s left to Queue B. Beware of the null node and the initialization of the two Queue.

* Attentions

## 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]

]

* Solving ideas

Two kinds of solutions.

Use Queue and two variable (current, next) to implement the iterative solution (BFS).

DFS solution is more complex, we should record current level and print each level per time, from level 1 to max level. So get the maximum height first and print nodes in each level.

For a given level, if level is 1, add the node’s value to list, or recursive do its left and right node and level minus one.

* Attentions

For recursive solution, beware of the null situation.

## 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]

]

* Solving ideas
* Attentions

## 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.

* Solving ideas

Two solutions, one recursive solution using DFS, this solution is easy to implement.

Another solution use BFS with one Queue, just record the current level node number and next level node’s number and do it with BFS.

* Attentions

## 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.

* Solving ideas

Two solutions, one is recursive and the other is recursive, first recursive solution is easy to implement, just get root from preorder array and split the two array into left arrays and right arrays.

Iterative solution is more complex, for most recursive problems, there is a iterative solution instead. But most is more complex to implement.

* Attentions