# 1. 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

# 2. Median of Two Sorted Arrays

There are two sorted arrays nums1 and nums2 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)).

Example 1:

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

Example 2:

nums1 = [1, 2]

nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5

# 3. Merge k Sorted Lists

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

25. 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.

k is a positive integer and is less than or equal to the length of the linked 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

# 4. Substring with Concatenation of All Words

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

For example, given:

s: "barfoothefoobarman"

words: ["foo", "bar"]

You should return the indices: [0,9].

(order does not matter).

# 5. 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.

# 6. 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.

# 7. 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.

# 8. 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.

# 9. 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.)

**Note:**  
You can assume that you can always reach the last index.

# 10. 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.."]

]

# 11. N-Queens II

Follow up for N-Queens problem.

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



# 12. 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].

# 13. 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.

# 14. 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.

# 15. 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

# 16. 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.

# 17. 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 heights = [2,1,5,6,2,3],  
return 10.

# 18. Maximal Rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

For example, given the following matrix:

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

Return 6.

# 19. 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":

great

/ \

gr eat

/ \ / \

g r e at

/ \

a t

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

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

rgeat

/ \

rg eat

/ \ / \

r g e at

/ \

a t

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

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

rgtae

/ \

rg tae

/ \ / \

r g ta e

/ \

t a

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

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

# 20. 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.

# 21. 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?

# 22. Distinct Subsequences

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

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

Here is an example:

S = "rabbbit", T = "rabbit"

Return 3.

# 23. Best Time to Buy and Sell Stock III

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

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

Note:

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

124. Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain at least one node and does not need to go through the root.

For example:

Given the below binary tree,

1

/ \

2 3

Return 6.

# 24. Word Ladder II

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find all shortest transformation sequence(s) from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

For example,

Given:  
*beginWord* = "hit"  
*endWord* = "cog"  
*wordList* = ["hot","dot","dog","lot","log","cog"]

Return

[

["hit","hot","dot","dog","cog"],

["hit","hot","lot","log","cog"]

]

**Note:**

* Return an empty list if there is no such transformation sequence.
* All words have the same length.
* All words contain only lowercase alphabetic characters.
* You may assume no duplicates in the word list.
* You may assume *beginWord* and *endWord* are non-empty and are not the same.

# 25. 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.

# 26. 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.

# 27. 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?

# 28. Word Break II

Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, add spaces in s to construct a sentence where each word is a valid dictionary word. You may assume the dictionary does not contain duplicate words.

Return all such possible sentences.

For example, given

s = "catsanddog",

dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].

# 29. 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?

# 30. LRU Cache

Design and implement a data structure for [Least Recently Used (LRU) cache](https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU). It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.  
put(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.

**Follow up:**  
Could you do both operations in **O(1)** time complexity?

**Example:**

LRUCache cache = new LRUCache( 2 /\* capacity \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // returns 1

cache.put(3, 3); // evicts key 2

cache.get(2); // returns -1 (not found)

cache.put(4, 4); // evicts key 1

cache.get(1); // returns -1 (not found)

cache.get(3); // returns 3

cache.get(4); // returns 4

# 31. Max Points on a Line

Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

# 32. Find Minimum in Rotated Sorted Array

Suppose an array sorted in ascending order 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).

Find the minimum element.

You may assume no duplicate exists in the array.

# 33. Find Minimum in Rotated Sorted Array II

Follow up for "Find Minimum in Rotated Sorted Array":

What if duplicates are allowed?

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

Suppose an array sorted in ascending order 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).

Find the minimum element.

The array may contain duplicates.

# 作业要求：

1. 每个同学都应该从中选取一定的题目完成作业，否则本课程不能得分；
2. 完成题目的难度和数量较多的同学可以得到较高的分数；
3. 作业提交包括（1）题目翻译；（2）算法分析；（3）算法复杂度分析；（4）源代码

等文件，放入一个文件夹中，文件夹的名字是题目的编号；将这些文件夹放在一个统一的文件夹中，该文件夹的名字是：班级+姓名+学号；

1. 作业提交时间2017年12月31日前。