LeetCode

(soulmachine@gmail.com)

https://github.com/soulmachine/leetcode

2016-4-5

LeetCode Online Judge(http://leetcode.com/onlinejudge)

LeetCode Online Judge

-
-
-

[1] 65292 65292 12298 12298 plusminus[2] plusminus

GitHub

plusminushttps://github.com/soulmachine/leetcode

plusminushttp://q.weibo.com/1312378

[1] 12298 12298 plusminushttp://book.douban.com/subject/2024655/

[2] http://book.douban.com/subject/4854123/

a==b fabs(a-b) 1e-9 12290 12290

x % 2 != 0 x % 2 == 1

char char unsigned int unsigned char

vector

vector

```
int** ary = new int*[row_num];
for(int i = 0; i < row_num; ++i)
    ary[i] = new int[col_num];
```

```
vector<vector<int> > ary(row_num, vector<int>(col_num, 0));
```

2.1

2.1.1 Contains Duplicate

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

```
// LeetCode, Contains Duplicate
```

- Contains Duplicate II §**??**
- Contains Duplicate III §**??**

2.1.2 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 ,

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

```
// LeetCode, Remove Duplicates from Sorted Array
```

```
// LeetCode, Remove Duplicates from Sorted Array
```

```
// LeetCode, Remove Duplicates from Sorted Array
```

- Remove Duplicates from Sorted Array II□ 65292 □ 65292 §**??**

2.1.3 Remove Duplicates from Sorted Array II

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

For example, Given sorted array ,

Your function should return length = 5, and A is now

```
// LeetCode, Remove Duplicates from Sorted Array II
```

```
   occur < 2  occur < 3
// LeetCode, Remove Duplicates from Sorted Array II
// @author  (http://weibo.com/u/1666779725)
```

- Remove Duplicates from Sorted Array §??

2.1.4 Search in Rotated Sorted Array

Suppose a sorted array is rotated at some pivot unknown to you beforehand.
(i.e., might become).
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.

```
// LeetCode, Search in Rotated Sorted Array
```

- Search in Rotated Sorted Array II §??

2.1.5 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.

`A[m]>=A[l]`, `[l,m]` □ 12290 □ 12290
`A[m]>=A[l]`

- `A[m]>A[l]` `[l,m]`
- `A[m]==A[l]` `l++`

```
// LeetCode, Search in Rotated Sorted Array II
```

- Search in Rotated Sorted Array §**??**

2.1.6 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 $O(\log(m+n))$.

k

$O(m+n)$ k

k m `pA` `pB` `pA++` `m++`□ 65307 □ 65307 `pB++` `m++` m k $O(k)$ $O(1)$ k $m+n$ $O(m+n)$

k k k

$k/2$ $k/2$ `A[k/2-1]`□ 65289 □ 65289 $k/2$ `B[k/2-1]` k k

- `A[k/2-1] == B[k/2-1]`
- `A[k/2-1] > B[k/2-1]`
- `A[k/2-1] < B[k/2-1]`

`A[k/2-1] < B[k/2-1]` `A[0]` `A[k/2-1]` $A \cup B$ `A[k/2-1]` $A \cup B$ k

$k/2$ `A[k/2-1] > B[k/2-1]` $k/2$

`A[k/2-1] == B[k/2-1]` k `A[k/2-1]` `B[k/2-1]`

- `B[k-1]` `A[k-1]`
- `k=1` `min(A[0], B[0])`
- `A[k/2-1] == B[k/2-1]` `A[k/2-1]` `B[k/2-1]`

```
// LeetCode, Median of Two Sorted Arrays
```

-

2.1.7 Longest Consecutive Sequence

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

For example, Given , The longest consecutive elements sequence is . Return its length: 4.

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

$O(n \log n)$ $O(n)$

$O(n)$

`unordered_map<int, bool> used`

```
// Leet Code, Longest Consecutive Sequence
```

`unordered_map<int, int> map` http://discuss.leetcode.com/questions/1070/longest-consecutive-sequence

```
// Leet Code, Longest Consecutive Sequence
```

•

2.1.8 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: {}

Output:

$O(n^2)$

$O(n)$.

$O(n \log n)$ $O(n)$ $O(n \log n)$

```
//LeetCode, Two Sum
//
```

- 3Sum, §**??**
- 3Sum Closest, §**??**
- 4Sum, §**??**

2.1.9 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, $a \leq b \leq c$)
- The solution set must not contain duplicate triplets.

For example, given array $\{\}$.

A solution set is:

```
(-1, 0, 1)
(-1, -1, 2)
```

$O(n^2)$

k-sum $k-2$ $O(\max\{n \log n, n^{k-1}\})$

```
// LeetCode, 3Sum
```

- Two sum, §**??**
- 3Sum Closest, §**??**
- 4Sum, §**??**

2.1.10 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 $\{\}$, and .

The sum that is closest to the target is 2. ().

$O(n^2)$

```
// LeetCode, 3Sum Closest
```

- Two sum, §**??**
- 3Sum, §**??**
- 4Sum, §**??**

2.1.11 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, $a \leq b \leq c \leq d$)
- The solution set must not contain duplicate quadruplets.

For example, given array $\{\}$, and .

A solution set is:

```
(-1,  0, 0, 1)
(-2, -1, 1, 2)
(-2,  0, 0, 2)
```

$O(n^3)$

$O(n^3)$ 3Sum

```
// LeetCode, 4Sum
```

map

```
// LeetCode, 4Sum
//
```

multimap

```
// LeetCode, 4Sum
//  hashmap
```

```
// LeetCode, 4Sum
//
//
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        vector<vector<int>> result;
        if (nums.size() < 4) return result;
        sort(nums.begin(), nums.end());

        auto last = nums.end();
        for (auto a = nums.begin(); a < prev(last, 3);
                a = upper_bound(a, prev(last, 3), *a)) {
            for (auto b = next(a); b < prev(last, 2);
                    b = upper_bound(b, prev(last, 2), *b)) {
                auto c = next(b);
                auto d = prev(last);
                while (c < d) {
                    if (*a + *b + *c + *d < target) {
                        c = upper_bound(c, d, *c);
                    } else if (*a + *b + *c + *d > target) {
                        d = prev(lower_bound(c, d, *d));
                    } else {
                        result.push_back({ *a, *b, *c, *d });
                        c = upper_bound(c, d, *c);
                        d = prev(lower_bound(c, d, *d));
                    }
                }
            }
        }
        return result;
    }
};
```

- Two sum, §**??**
- 3Sum, §**??**
- 3Sum Closest, §**??**

2.1.12 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.

```
// LeetCode, Remove Element
```

```
// LeetCode, Remove Element
```

-

2.1.13 Next Permutation

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

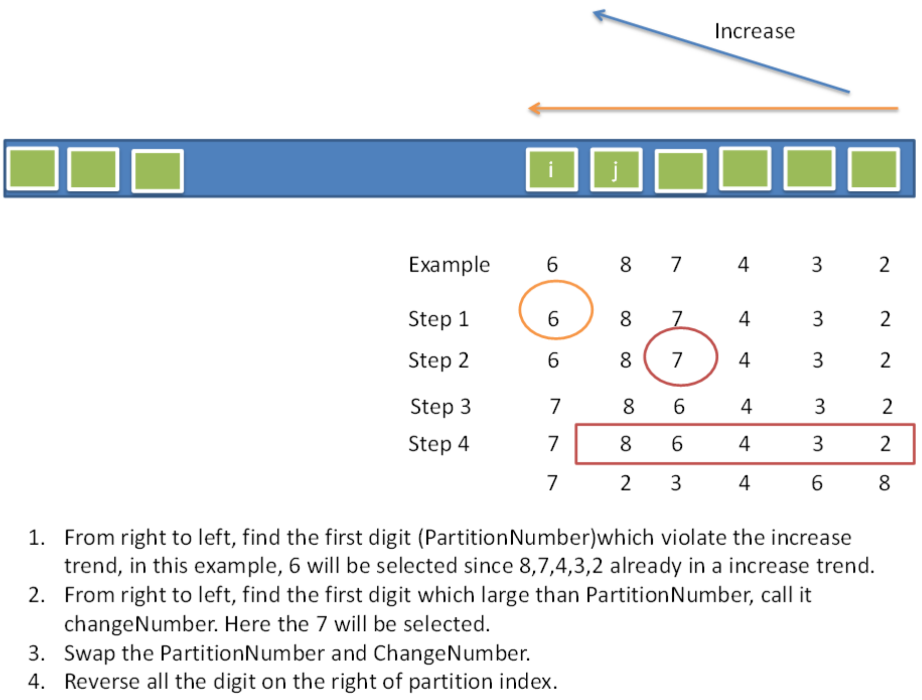
If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

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
```

?? http://fisherlei.blogspot.com/2012/12/leetcode-next-permutation.html



Example	6	8	7	4	3	2
Step 1	6	8	7	4	3	2
Step 2	6	8	7	4	3	2
Step 3	7	8	6	4	3	2
Step 4	7	8	6	4	3	2
	7	2	3	4	6	8

1. From right to left, find the first digit (PartitionNumber)which violate the increase trend, in this example, 6 will be selected since 8,7,4,3,2 already in a increase trend.
2. From right to left, find the first digit which large than PartitionNumber, call it changeNumber. Here the 7 will be selected.
3. Swap the PartitionNumber and ChangeNumber.
4. Reverse all the digit on the right of partition index.

2-1

```
// LeetCode, Next Permutation
```

- Permutation Sequence, §**??**
- Permutations, §**??**

- Permutations II, §??
- Combinations, §??

2.1.14 Permutation Sequence

The set `[1,2,3, 8230 8230 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.`

$k-1$ `next_permutation()`

k k

n k $a_1, a_2, a_3, ..., a_n$ a_1

a_1 $a_2, a_3, ..., a_n$, $n-1$ $n-1$ $(n-1)!$ $a_1 = k/(n-1)!$

$a_2, a_3, ..., a_n$

$$\begin{aligned} k_2 &= k\%(n-1)! \\ a_2 &= k_2/(n-2)! \\ &\cdots \\ k_{n-1} &= k_{n-2}\%2! \\ a_{n-1} &= k_{n-1}/1! \\ a_n &= 0 \end{aligned}$$

```
// LeetCode, Permutation Sequence
```

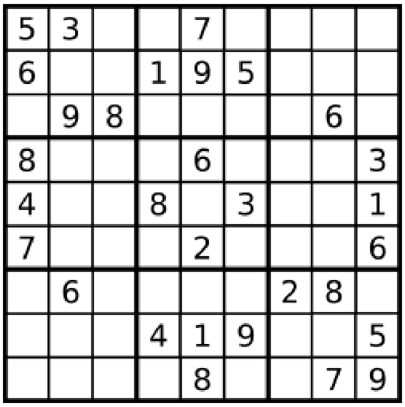
```
// LeetCode, Permutation Sequence
```

- Next Permutation, §??
- Permutations, §??
- Permutations II, §??
- Combinations, §??

2.1.15 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 '.'.



2-2 A partially filled sudoku which is valid

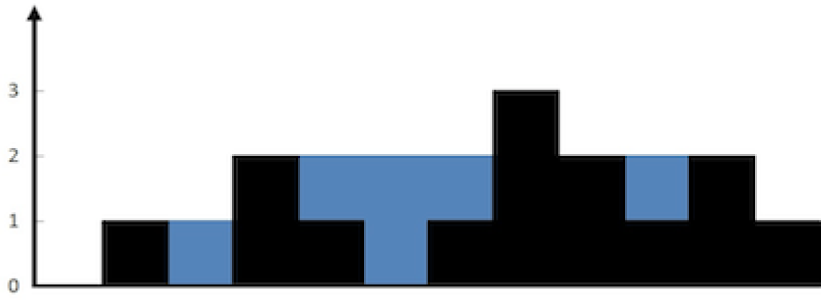
```
// LeetCode, Valid Sudoku
```

- Sudoku Solver, §??

2.1.16 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 , return 6.



2-3 Trapping Rain Water

1.
2.
3.

1.
2.
3.

```
// LeetCode, Trapping Rain Water
```

```
// LeetCode, Trapping Rain Water
// LeetCode, Trapping Rain Water
//
//
```

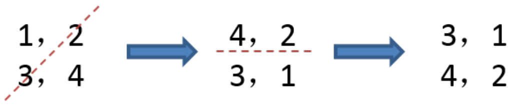
- Container With Most Water, §??
- Largest Rectangle in Histogram, §??

2.1.17 Rotate Image

You are given an $n \times n$ 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this in-place?



2-4 Rotate Image

```
// LeetCode, Rotate Image
```

```
// LeetCode, Rotate Image
```

-

2.1.18 Plus One

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

```
// LeetCode, Plus One
```

```
// LeetCode, Plus One
```

-

2.1.19 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?

$f(n)$ n n

- $n-1$
- $n-1$

$f(n)=f(n-1)+f(n-2)$

$$a_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

```
// LeetCode, Climbing Stairs
// LeetCode, Climbing Stairs
```

- Decode Ways, §??

2.1.20 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.

http://en.wikipedia.org/wiki/Gray_code

plusminus$g_0 = b_0, g_i = b_i \oplus b_{i-1}$

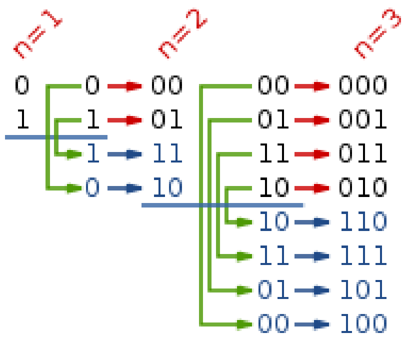
plusminus$b_0 = g_0, b_i = g_i \oplus b_{i-1}$

n $n \oplus (n/2)$

n

$0 \sim 2^n - 1$

n $n - 1$ **§??**



2-5 The first few steps of the reflect-and-prefix method.

```
// LeetCode, Gray Code
```

Reflect-and-prefix method

```
// LeetCode, Gray Code
// reflect-and-prefix method
```

•

2.1.21 Set Matrix Zeroes

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

Follow up: Did you use extra space?

A straight forward solution using $O(mn)$ space is probably a bad idea.

A simple improvement uses $O(m+n)$ space, but still not the best solution.

Could you devise a constant space solution?

$O(m+n)$

```
// LeetCode, Set Matrix Zeroes
```

```
// LeetCode, Set Matrix Zeroes
```

•

2.1.22 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.

$O(N^2)$

$O(N)$ sum total sum

```
// LeetCode, Gas Station
```

•

2.1.23 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?

```
// LeetCode, Candy
```

```
// LeetCode, Candy
```

-

2.1.24 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?

```
// LeetCode, Single Number
```

```
// LeetCode, Single Number
```

- Single Number II, §??

2.1.25 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?

Single Number

sizeof(int) count[sizeof(int)]count[i] i count[i]

one 1 two 2 one two one

```
// LeetCode, Single Number II
```

```
// LeetCode, Single Number II
```

- Single Number, §??

2.2

```
//
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) { }
};
```

2.2.1 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:

Output:

§??

```
// LeetCode, Add Two Numbers
//
```

- Add Binary, §??

2.2.2 Reverse Linked List II

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

For example: Given , m = 2 and n = 4,

return .

Note: Given m, n satisfy the following condition: $1 \leq m \leq n \leq$ length of list.

```
// LeetCode, Reverse Linked List II
```

-

2.2.3 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 and , return .

```
// LeetCode, Partition List
```

-

2.2.4 Remove Duplicates from Sorted List

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

For example,

Given , return .

Given , return .

```
// LeetCode, Remove Duplicates from Sorted List
```

```
// LeetCode, Remove Duplicates from Sorted List
```

- Remove Duplicates from Sorted List II §??

2.2.5 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 , return .

Given , return .

```
// LeetCode, Remove Duplicates from Sorted List II
```

```
// LeetCode, Remove Duplicates from Sorted List II
```

- Remove Duplicates from Sorted List §??

2.2.6 Rotate List

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

For example: Given and , return .

$len\ \ k\quad len\ \ k\% = len\ \ len - k$

```
// LeetCode, Remove Rotate List
```

-

2.2.7 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: , and n = 2.

After removing the second node from the end, the linked list becomes .

Note:

- Given n will always be valid.
- Try to do this in one pass.

p, q q n p q q p->next

```
// LeetCode, Remove Nth Node From End of List
```

-

2.2.8 Swap Nodes in Pairs

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

For example, Given , you should return the list as .

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

```
// LeetCode, Swap Nodes in Pairs
```

```
// LeetCode, Swap Nodes in Pairs
```

- Reverse Nodes in k-Group, §??

2.2.9 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:

For $k = 2$, you should return:

For $k = 3$, you should return:

```
// LeetCode, Reverse Nodes in k-Group
```

```
// LeetCode, Reverse Nodes in k-Group
```

- Swap Nodes in Pairs, §??

2.2.10 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.

```
// LeetCode, Copy List with Random Pointer
```

•

2.2.11 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?

`unordered_map<int, bool> visited` $O(n)$ $O(N)$

$O(n)$ $O(1)$ http://leetcode.com/2010/09/detecting-loop-in-singly-linked-list.html

```
//LeetCode, Linked List Cycle
```

- Linked List Cycle II, §??

2.2.12 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?

n $1 \le n$ s $2s$ s n r

$$\begin{aligned} 2s &= s + nr \\ s &= nr \end{aligned}$$

L a x

$$\begin{aligned} x + a &= nr = (n–1)r + r = (n-1)r + L - x \\ x &= (n-1)r + (L–x–a) \end{aligned}$$

$L–x–a$ $n-1$ `head` `slow2`

```
//LeetCode, Linked List Cycle II
```

- `Linked List Cycle`, §??

2.2.13 Reorder List

Given a singly linked list $L : L_0 \to L_1 \to \cdots \to L_{n-1} \to L_n$, reorder it to: $L_0 \to L_n \to L_1 \to L_{n-1} \to L_2 \to L_{n-2} \to \cdots$

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}.

$O(1)$

```
// LeetCode, Reorder List
```

-

2.2.14 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.

std::liststd::unordered_map

- $O(1)$
-
-
-
-

```
// LeetCode, LRU Cache
```

-

3.1 Valid Palindrome

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

For example,

is a palindrome.

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.

```
// Leet Code, Valid Palindrome
```

- Palindrome Number, §??

3.2 Implement strStr()

Implement strStr().

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

$O(m*n)$

```
// LeetCode, Implement strStr()
```

KMP

```
// LeetCode, Implement strStr()
```

- String to Integer (atoi) §??

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.

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, or is returned.

1. " + 413",
2.

```
// LeetCode, String to Integer (atoi)
```

- Implement strStr() §??

3.4 Add Binary

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

For example,

```
a = "11"
b = "1"
```

Return .

```
//LeetCode, Add Binary
```

- Add Two Numbers, §??

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.

$O(n^2)$

$O(n^2)$ f[i][j]

```
f[i][j] = if (i == j) S[i]
          if (S[i] == S[j] && f[i+1][j-1] == S[i+1][j-1]) S[i][j]
          else max(f[i+1][j-1], f[i][j-1], f[i+1][j])
```

$O(n^2)$ f(i,j)

$$f(i,j) = \begin{cases} true & ,i=j \\ S[i]=S[j] & ,j=i+1 \\ S[i]=S[j] \texttt{ and } f(i+1,j-1) & ,j>i+1 \end{cases}$$

$O(n)$ http://leetcode.com/2011/11/longest-palindromic-substring-part-ii.html

```
// LeetCode, Longest Palindromic Substring
//
```

```
// LeetCode, Longest Palindromic Substring
```

Manacher 8217 8217

```
// LeetCode, Longest Palindromic Substring
```

•

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
```

```
// LeetCode, Regular Expression Matching
```

- Wildcard Matching, §??

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
```

'*' p '*''*' s s s++

```
// LeetCode, Wildcard Matching
//
```

```
// LeetCode, Wildcard Matching
```

- Regular Expression Matching, §??

3.8 Longest Common Prefix

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

```
// LeetCode, Longest Common Prefix
//
```

```
// LeetCode, Longest Common Prefix
//
//
```

-

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.

strtod()

```
// LeetCode, Valid Number
// @author  (http://weibo.com/luangong)
```

```
// LeetCode, Valid Number
// @author  (http://weibo.com/lianchengzju)
```

•

3.10 Integer to Roman

Given an integer, convert it to a roman numeral.

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

```
// LeetCode, Integer to Roman
```

- Roman to Integer, §??

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.

IV = 5 - 1 VI = 5 + 1, II=1+1

```
// LeetCode, Roman to Integer
```

- Integer to Roman, §??

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 nth sequence.

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

```
// LeetCode, Count and Say
// @author  (http://weibo.com/lianchengzju)
```

•

3.13 Anagrams

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

Note: All inputs will be in lower-case.

Anagram 65288 65288 "dormitory" "dirty room" "tea" "eat"
anagrams

```
// LeetCode, Anagrams
```

•

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"
```

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".

```
// LeetCode, Simplify Path
```

-

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.

STL

```
// LeetCode, Length of Last Word
//  STL
```

```
// LeetCode, Length of Last Word
//  word
```

-

4.1

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.

```
// LeetCode, Valid Parentheses
```

- Generate Parentheses, §??
- Longest Valid Parentheses, §??

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.

```
// LeetCode, Longest Valid Parenthese
```

Dynamic Programming, One Pass

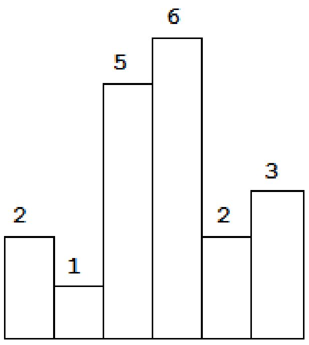
```
// LeetCode, Longest Valid Parenthese
```

```
// LeetCode, Longest Valid Parenthese
```

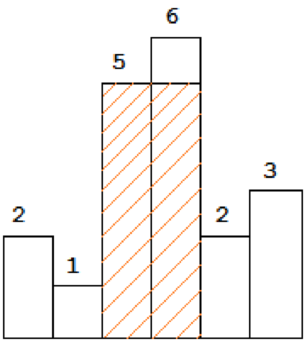
- Valid Parentheses, §??
- Generate Parentheses, §??

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.



4-1 Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].



4-2 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.

Container With Most Water(§??) $O(n^2)$

§?? $i = 4$

// LeetCode, Largest Rectangle in Histogram

- Trapping Rain Water, §??
- Container With Most Water, §??

4.1.4 Evaluate Reverse Polish Notation

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

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

Some examples:

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

```
// LeetCode, Evaluate Reverse Polish Notation
```

```
// LeetCode, Max Points on a Line
```

•

4.2

LeetCode

```
//
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) { }
};
```

5.1

(root->left->right) (left->right->root) (left->root->right)

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
```

return .

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

```
// LeetCode, Binary Tree Preorder Traversal
```

Morris

```
// LeetCode, Binary Tree Preorder Traversal
```

- Binary Tree Inorder Traversal §??
- Binary Tree Postorder Traversal §??
- Recover Binary Search Tree §??

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
```

return .

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

```
// LeetCode, Binary Tree Inorder Traversal
```

Morris

```
// LeetCode, Binary Tree Inorder Traversal
```

- Binary Tree Preorder Traversal §??
- Binary Tree Postorder Traversal §??
- Recover Binary Search Tree §??

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
```

return .

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

```
// LeetCode, Binary Tree Postorder Traversal
```

Morris

```
// LeetCode, Binary Tree Postorder Traversal
```

- Binary Tree Preorder Traversal §??
- Binary Tree Inorder Traversal §??
- Recover Binary Search Tree §??

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
```

```
return its level order traversal as:
  [
    [3],
    [9,20],
    [15,7]
  ]
```

```
// LeetCode, Binary Tree Level Order Traversal
```

```
// LeetCode, Binary Tree Level Order Traversal
```

- Binary Tree Level Order Traversal II §??
- Binary Tree Zigzag Level Order Traversal §??

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
```

```
return its bottom-up level order traversal as:
  [
    [15,7]
    [9,20],
    [3],
  ]
```

§??reverse()

```
// LeetCode, Binary Tree Level Order Traversal II
```

```
// LeetCode, Binary Tree Level Order Traversal II
```

- Binary Tree Level Order Traversal §??
- Binary Tree Zigzag Level Order Traversal §??

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
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

```
// LeetCode, Binary Tree Zigzag Level Order Traversal
```

```
// LeetCode, Binary Tree Zigzag Level Order Traversal
//
```

- Binary Tree Level Order Traversal §??
- Binary Tree Level Order Traversal II §??

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?

$O(n)$

$O(n)$

```
// LeetCode, Recover Binary Search Tree
```

- Binary Tree Inorder Traversal §??

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.

```
// LeetCode, Same Tree
// LeetCode, Same Tree
```

- Symmetric Tree §??

5.1.9 Symmetric 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.

```
// LeetCode, Symmetric Tree
```

```
// LeetCode, Symmetric Tree
```

- Same Tree §??

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.

```
// LeetCode, Balanced Binary Tree
```

•

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
```

```
// LeetCode, Flatten Binary Tree to Linked List
```

```
// LeetCode, Flatten Binary Tree to Linked List
```

```
// LeetCode, Flatten Binary Tree to Linked List
```

•

5.1.12 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
```

Populating Next Right Pointers in Each Node I.

```
// LeetCode, Populating Next Right Pointers in Each Node II
```

```
// LeetCode, Populating Next Right Pointers in Each Node II
```

• Populating Next Right Pointers in Each Node §??

5.2

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.

```
// LeetCode, Construct Binary Tree from Preorder and Inorder Traversal
```

- Construct Binary Tree from Inorder and Postorder Traversal §??

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.

```
// LeetCode, Construct Binary Tree from Inorder and Postorder Traversal
```

- Construct Binary Tree from Preorder and Inorder Traversal §??

5.3

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
```

```
 1       1           2          3       3
  \       \         / \        /       /
   3       2       1   3      2       1
  /         \                /         \
2            3              1           2
```

$1, 2, 3, ..., n$ **plusminus plusminus**

$f(i)$ $[1, i]$

$f(0) = 1$

1$f(1) = 1$

1,2

```
1             2
 \           /
  2      1
```

$$\begin{aligned} f(2) &= f(0) * f(1) \\ &+ f(1) * f(0) \end{aligned}$$

$$\begin{aligned} f(3) &= f(0) * f(2) \\ &+ f(1) * f(1) \\ &+ f(2) * f(0) \end{aligned}$$

f

$$f(i) = \sum_{k=1}^{i} f(k-1) \times f(i-k)$$

```
// LeetCode, Unique Binary Search Trees
```

- Unique Binary Search Trees II §??

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
```

```
// LeetCode, Unique Binary Search Trees II
```

- Unique Binary Search Trees §??

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.

```
// Validate Binary Search Tree
```

- Validate Binary Search Tree §??

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.

```
// LeetCode, Convert Sorted Array to Binary Search Tree
```

- Convert Sorted List to Binary Search Tree §??

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.

http://leetcode.com/2010/11/convert-sorted-list-to-balanced-binary.html

Convert Sorted Array to Binary Search Tree $O(n \log n)$

```
// LeetCode, Convert Sorted List to Binary Search Tree
//  Convert Sorted Array to Binary Search Tree
```

```
// LeetCode, Convert Sorted List to Binary Search Tree
```

- Convert Sorted Array to Binary Search Tree §??

5.4

§?? 8220 8220 plusminus

$3! = 6$ root->r->l

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.

```
// LeetCode, Minimum Depth of Binary Tree
```

```
// LeetCode, Minimum Depth of Binary Tree
```

- Maximum Depth of Binary Tree §??

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.

```
// LeetCode, Maximum Depth of Binary Tree
```

- Minimum Depth of Binary Tree §??

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 ,

```
          5
         / \
        4   8
       /   / \
      11  13  4
     /  \      \
    7    2      1
```

return true, as there exist a root-to-leaf path which sum is 22.

true false

```
// LeetCode, Path Sum
```

- Path Sum II §??

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 ,

```
          5
         / \
        4   8
       /   / \
      11  13  4
     /  \    / \
    7    2  5   1
```

return

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

```
// LeetCode, Path Sum II
```

- Path Sum §??

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.

§??

```
// LeetCode, Binary Tree Maximum Path Sum
```

- Maximum Subarray §??

5.4.6 Populating Next Right Pointers in Each Node

Given a binary tree

```
struct TreeLinkNode {
   int val;
   TreeLinkNode *left, *right, *next;
   TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
};
```

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
```

```
// LeetCode, Populating Next Right Pointers in Each Node
```

- Populating Next Right Pointers in Each Node II §??

5.4.7 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.

```
// LeetCode, Decode Ways
```

•

6.1 Merge Two Sorted Arrays

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.

```
//LeetCode, Merge Sorted Array
```

- Merge Two Sorted Lists §??
- Merge k Sorted Lists §??

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.

```
//LeetCode, Merge Two Sorted Lists
```

- Merge Sorted Array §??
- Merge k Sorted Lists §??

6.3 Merge k Sorted Lists

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

§??

```
//LeetCode, Merge k Sorted Lists
```

- Merge Sorted Array §??
- Merge Two Sorted Lists §??

6.4 Insertion Sort List

Sort a linked list using insertion sort.

```
// LeetCode, Insertion Sort List
```

- Sort List, §??

6.5 Sort List

Sort a linked list in $O(nlogn)$ time using constant space complexity.

$O(nlogn)$"Merge Two Sorted Lists"

```
// LeetCode, Sort List
```

- Insertion Sort List §??

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.

A[i]!= i+1 A[i]== A[A[i]-1]

```
// LeetCode, First Missing Positive
```

- Sort Colors, §??

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.

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?

$O(n)$ $O(1)$

partition n

```
// LeetCode, Sort Colors
// Counting Sort
```

```
// LeetCode, Sort Colors
```

```
// LeetCode, Sort Colors
// use partition()
```

```
// LeetCode, Sort Colors
//  partition()
```

- First Missing Positive, §??

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 .

For example, Given and target value 8, return .

```
// LeetCode, Search for a Range
```

lower_bound upper_bound

```
// LeetCode, Search for a Range
//  lower_bound  upper_bound
```

- Search Insert Position, §??

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
```

std::lower_bound()

```
// LeetCode, Search Insert Position
//  lower_bound
```

- Search for a Range, §??

7.3 Search a 2D Matrix

Write an efficient algorithm that searches for a value in an $m \times n$ 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.

```
// LeetCode, Search a 2D Matrix
```

-

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 , a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

8.1.1

```
// LeetCode, Subsets
```

bool selected[n]

```
// LeetCode, Subsets
```

8.1.2

```
// LeetCode, Subsets
```

i $S[i]$ `S={A,B,C,D}` `0110=6` `{B,C}`

B_1 B_2 $B_1 \cup B_2, B_1 \cap B_2, B_1 \triangle B_2$

```
// LeetCode, Subsets
```

- `Subsets II` §??

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],
  []
]
```

8.2.1

```
// LeetCode, Subsets II
// LeetCode, Subsets II
// LeetCode, Subsets II
```

8.2.2

```
// LeetCode, Subsets II
//
```

```
// LeetCode, Subsets II
```

- Subsets §??

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].

8.3.1 next_permutation()

std::next_permutation()

```
// LeetCode, Permutations
```

8.3.2

§??

```
// LeetCode, Permutations
//  next_permutation()
```

8.3.3

```
// LeetCode, Permutations
//
```

- Next Permutation, §??
- Permutation Sequence, §??
- Permutations II, §??
- Combinations, §??

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].

8.4.1 next_permutation()

std::next_permutation()

8.4.2

std::next_permutation()

8.4.3

permute() p

```
// LeetCode, Permutations II
```

- Next Permutation, §??
- Permutation Sequence, §??
- Permutations, §??
- Combinations, §??

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],
]
```

8.5.1

```
// LeetCode, Combinations
//
```

8.5.2

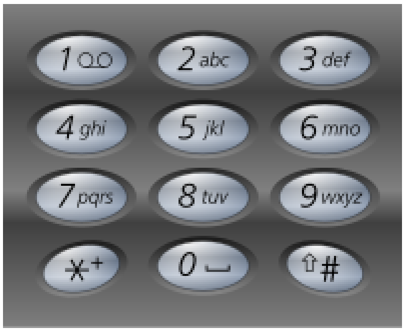
```
// LeetCode, Combinations
// use prev_permutation()
```

- Next Permutation, §??
- Permutation Sequence, §??
- Permutations, §??
- Permutations II, §??

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.



8-1 Phone Keyboard

Input: Digit string

Output: .

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

8.6.1

```
// LeetCode, Letter Combinations of a Phone Number
```

8.6.2

```
// LeetCode, Letter Combinations of a Phone Number
```

•

9.1 Word Ladder

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

For example, Given:

```
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

As one shortest transformation is , 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.

```
//LeetCode, Word Ladder
```

```
//LeetCode, Word Ladder
```

- Word Ladder II §??

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:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

For example, Given:

```
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

Return

```
[
    ["hit","hot","dot","dog","cog"],
    ["hit","hot","lot","log","cog"]
]
```

Note:

- All words have the same length.
- All words contain only lowercase alphabetic characters.

Word Ladder

```
//LeetCode, Word Ladder II
```

```
//LeetCode, Word Ladder II
```

dict

```
//LeetCode, Word Ladder II
```

- Word Ladder §??

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
```

'O'

```
// LeetCode, Surrounded Regions
```

-

9.4

9.4.1

65306 65306

9.4.2

1.
   (a)
   (b)
      i.
      ii.
2.
3.
4.
   (a)
   (b) 8220 8220 **plusminus** visited visited
   (c)
      i.
      ii. unordered_set §??
      iii.
5.

9.4.3

queue vector

1. state_t level level queue priority_queue
2. current, next level level level

bool visited[STATE_MAX] vector<bool> visited(STATE_MAX, false)) set unordered_-set

unordered_map<state_t, state_t > father state_t nodes[STATE_MAX]

bfs_common.h

```
/**  */
struct state_t {
```

bfs_template.cpp

```
#include "bfs_common.h"

/**
```

bfs_template1.cpp

```
#include "bfs_common.h"

/**
```

bfs_template.cpp

```
/**
```

bfs_template.cpp

```
#include "bfs_common.h"

/**
```

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 , Return

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

$n-1$ $O(2^{n-1})$

```
//LeetCode, Palindrome Partitioning
```

Combination Sum, Combination Sum II

```
//LeetCode, Palindrome Partitioning
```

```
// LeetCode, Palindrome Partitioning
```

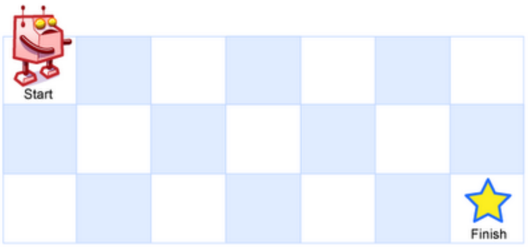
- Palindrome Partitioning II §??

10.2 Unique Paths

A robot is located at the top-left corner of a $m \times n$ 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?



10-1 Above is a 3×7 grid. How many possible unique paths are there?

Note: m and n will be at most 100.

10.2.1

```
// LeetCode, Unique Paths
//
```

10.2.2

```
// LeetCode, Unique Paths
//  +
```

10.2.3

`f[i][j]` $(1,1)$ (i,j)
`f[i][j]=f[i-1][j]+f[i][j-1]`

```
// LeetCode, Unique Paths
//
```

10.2.4

m n $m+n-2$ $m-1$ $m+n-2$ m–1 C_{m+n-2}^{m-1}

```
// LeetCode, Unique Paths
//
class Solution {
public:
    typedef long long int64_t;
    // , n!/(start-1)! n*(n-1)...start n >= 1
    static int64_t factor(int n, int start = 1) {
        int64_t  ret = 1;
        for(int i = start; i <= n; ++i)
            ret *= i;
        return ret;
    }
    //  C_n^k
    static int64_t combination(int n, int k) {
        //
        if (k == 0) return 1;
        if (k == 1) return n;

        int64_t ret = factor(n, k+1);
        ret /= factor(n - k);
        return ret;
    }

    int uniquePaths(int m, int n) {
        // max
        return combination(m+n-2, max(m-1, n-1));
    }
};
```

- `Unique Paths II` §??
- `Minimum Path Sum,` §??

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 3×3 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.

10.3.1

```
// LeetCode, Unique Paths II
//  +
class Solution {
public:
    int uniquePathsWithObstacles(const vector<vector<int> >& obstacleGrid) {
        const int m = obstacleGrid.size();
        const int n = obstacleGrid[0].size();
        if (obstacleGrid[0][0] || obstacleGrid[m - 1][n - 1]) return 0;

        f = vector<vector<int> >(m, vector<int>(n, 0));
        f[0][0] = obstacleGrid[0][0] ? 0 : 1;
        return dfs(obstacleGrid, m - 1, n - 1);
    }
private:
    vector<vector<int> > f;  //

    // @return  (0, 0)  (x, y)
    int dfs(const vector<vector<int> >& obstacleGrid,
            int x, int y) {
        if (x < 0 || y < 0) return 0; //

        // (x,y)
        if (obstacleGrid[x][y]) return 0;
        if (x == 0 and y == 0) return f[0][0]; //

        if (f[x][y] > 0) {
            return f[x][y];
        } else {
            return f[x][y] = dfs(obstacleGrid, x - 1, y) +
                dfs(obstacleGrid, x, y - 1);
        }
    }
};
```

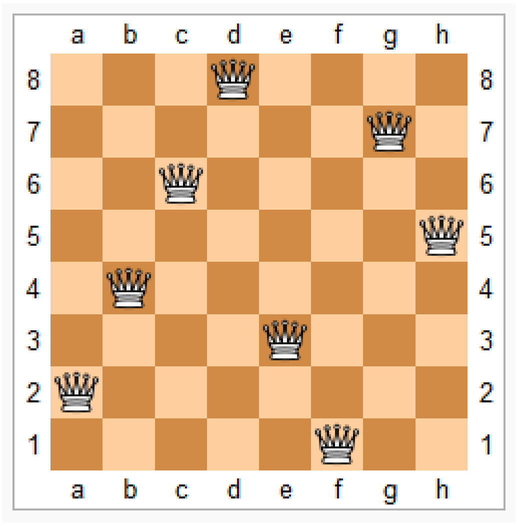
10.3.2

```
// LeetCode, Unique Paths II
//
```

- Unique Paths §??
- Minimum Path Sum, §??

10.4 N-Queens

The *n-queens puzzle* is the problem of placing n queens on an $n \times n$ chessboard such that no two queens attack each other.



10-2 Eight Queens

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

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

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

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

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

vector<int> C(n, 0), C[i] (i, C[i])

```
// LeetCode, N-Queens
//
```

```
// LeetCode, N-Queens
//
```

- N-Queens II §??

10.5 N-Queens II

Follow up for N-Queens problem.

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

```
// LeetCode, N-Queens II
//
```

```
// LeetCode, N-Queens II
//
```

- N-Queens §??

10.6 Restore IP Addresses

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

For example: Given ,

return . (Order does not matter)

```
// LeetCode, Restore IP Addresses
```

•

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, $a_1 \leq a_2 \leq ... \leq a_k$).
- 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]
```

```
// LeetCode, Combination Sum
```

- Combination Sum II §??

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]
```

```
// LeetCode, Combination Sum II
```

- Combination Sum §??

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:

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

$< n$

```
// LeetCode, Generate Parentheses
```

```
// LeetCode, Generate Parentheses
// @author  (http://weibo.com/lianchengzju)
class Solution {
public:
    vector<string> generateParenthesis (int n) {
        if (n == 0) return vector<string>(1, "");
        if (n == 1) return vector<string> (1, "()");
        vector<string> result;

        for (int i = 0; i < n; ++i)
            for (auto inner : generateParenthesis (i))
                for (auto outer : generateParenthesis (n - 1 - i))
                    result.push_back ("(" + inner + ")" + outer);

        return result;
    }
};
```

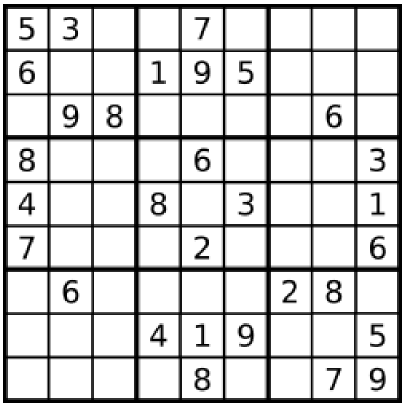
- Valid Parentheses, §??
- Longest Valid Parentheses, §??

10.10 Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

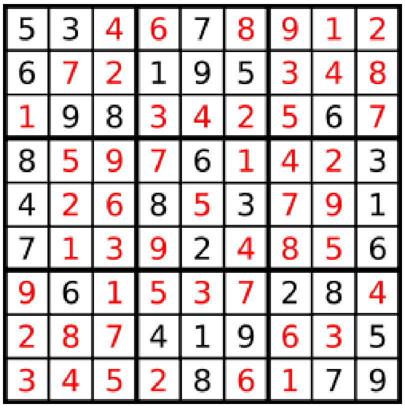
Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

10-3 A sudoku puzzle...



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

10-4 ...and its solution numbers marked in red

```
// LeetCode, Sudoku Solver
```

- Valid Sudoku, §??

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 neighbouring. 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.

```
// LeetCode, Word Search
//
```

-

10.12

10.12.1

10.12.2

1.
    (a)
    (b) `path[]`
2.
3. `struct struct`
4.
5.
6.

    `path[]`
7.
    (a)
    (b) §??
8.
    (a)
    (b)
        i.
        ii. `map unordered_map map`

10.12.3

dfs_template.cpp

```
/**
```

10.12.4

http://en.wikipedia.org/wiki/Depth_first_search http://en.wikipedia.org/wiki/Backtracking

= +

10.12.5

65292 65292

memorization 12290 12290 memorization 65288 65288 §??plusminus memorization memorization memorization memorization

11.1 Pow(x,n)

Implement pow(x, n).

$x^n = x^{n/2} \times x^{n/2} \times x^{n\%2}$

```
//LeetCode, Pow(x, n)
```

- Sqrt(x) §??

11.2 Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of x.

```
// LeetCode, Sqrt(x)
//
```

- `Pow(x)` §??

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:

, return true.

, return false.

A[i]

f[i] A[i]

$$f[i] = max(f[i-1], A[i-1]) - 1, i > 0$$

```
// LeetCode, Jump Game
```

```
// LeetCode, Jump Game
```

```
// LeetCode, Jump Game
```

- Jump Game II §??

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

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.)

```
// LeetCode, Jump Game II
```

```
// LeetCode, Jump Game II
```

- Jump Game §??

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.

m $m = 1$

```
// LeetCode, Best Time to Buy and Sell Stock
```

- Best Time to Buy and Sell Stock II §??
- Best Time to Buy and Sell Stock III §??

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).

m $m =$

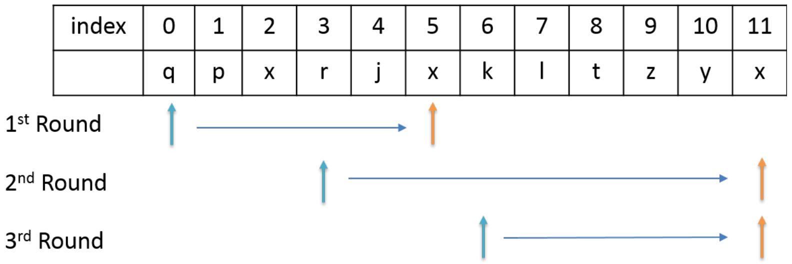
```
// LeetCode, Best Time to Buy and Sell Stock II
```

- Best Time to Buy and Sell Stock §??
- Best Time to Buy and Sell Stock III §??

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 is , which the length is 3. For the longest substring is , with the length of 1.

index+1 $O(n)$??



12-1

```
// LeetCode, Longest Substring Without Repeating Characters
```

•

12.6 Container With Most Water

Given n non-negative integers $a_1, a_2, ..., a_n$, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of line i is at (i, a_i) 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.

```
// LeetCode, Container With Most Water
```

- Trapping Rain Water, §??
- Largest Rectangle in Histogram, §??

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.

$f(i,j)$ (i,j)

$$f(i,j) = \min\left\{f(i+1,j), f(i+1,j+1)\right\} + (i,j)$$

```
// LeetCode, Triangle
```

-

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 , the contiguous subarray has the largest .

1 12289 12289

f[j] S[j]

$$\begin{aligned} f[j] &= \max\left\{f[j-1]+S[j], S[j]\right\}, \ \ 1 \leq j \leq n \\ target &= \max\left\{f[j]\right\}, \ \ 1 \leq j \leq n \end{aligned}$$

- $f[j-1]+S[j]$
- $S[j]$

- $O(n^3)$
- $O(n^2)$
- $O(n \log n)$
- $O(n^2)$ $O(n)$
-

```
// LeetCode, Maximum Subarray
```

```
// LeetCode, Maximum Subarray
```

- Binary Tree Maximum Path Sum §??

13.3 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 ,

Return 1 since the palindrome partitioning could be produced using 1 cut.

f(i,j) [i,j]

$$f(i,j) = \min\left\{f(i,k) + f(k+1,j)\right\}, i \leq k \leq j, 0 \leq i \leq j < n$$

$$f(i) = \min\left\{f(j+1) + 1\right\}, i \leq j < n$$

[i,j]

P[i][j] = true if [i,j]

```
P[i][j] = str[i] == str[j] && P[i+1][j-1]
```

```
// LeetCode, Palindrome Partitioning II
```

- Palindrome Partitioning §??

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.

```
// LeetCode, Maximal Rectangle
```

-

13.5 Best Time to Buy and Sell Stock III

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 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).

$f(i)$ $[0,i](0 \le i \le n-1)$ $g(i)$ $[i,n-1](0 \le i \le n-1)$ $\max\{f(i)+g(i)\}, 0 \le i \le n-1$

m $m=2$**https://gist.github.com/soulmachine/5906637**

```
// LeetCode, Best Time to Buy and Sell Stock III
```

- Best Time to Buy and Sell Stock §??
- Best Time to Buy and Sell Stock II §??

13.6 Interleaving String

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

For example, Given: ,

When , return true.

When , return false.

f[i][j] s1[0,i] s2[0,j] s3[0, i+j] f[i][j]=f[i-1][j] 65307 65307 f[i][j]=f[i][j-1]

```
f[i][j] = (s1[i - 1] == s3 [i + j - 1] && f[i - 1][j])
       || (s2[j - 1] == s3 [i + j - 1] && f[i][j - 1]);
```

```
// LeetCode, Interleaving String
//
class Solution {
public:
    bool isInterleave(const string& s1, const string& s2, const string& s3) {
        if (s3.length() != s1.length() + s2.length())
            return false;

        return isInterleave(begin(s1), end(s1), begin(s2), end(s2),
                begin(s3), end(s3));
    }

    template<typename InIt>
    bool isInterleave(InIt first1, InIt last1, InIt first2, InIt last2,
            InIt first3, InIt last3) {
        if (first3 == last3)
            return first1 == last1 && first2 == last2;

        return (*first1 == *first3
                && isInterleave(next(first1), last1, first2, last2,
                        next(first3), last3))
                || (*first2 == *first3
                        && isInterleave(first1, last1, next(first2), last2,
                                next(first3), last3));
    }
};
```

```
// LeetCode, Interleaving String
```

```
// LeetCode, Interleaving String
```

•

13.7 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 :

```
    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 and swap its two children, it produces a scrambled string .

```
    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t
```

We say that is a scrambled string of .

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

```
    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta  e
       / \
      t   a
```

We say that is a scrambled string of .

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

map unordered_map

f[n][i][j] n s1[i] s2[j]

```
f[n][i][j]} =  (f[k][i][j] && f[n-k][i+k][j+k])
             || (f[k][i][j+n-k] && f[n-k][i+k][j])
```

```
// LeetCode, Scramble String
//
```

```
// LeetCode, Scramble String
```

```
// LeetCode, Scramble String
//
```

```
// LeetCode, Scramble String
```

```
typedef string::const_iterator Iterator;
typedef tuple<Iterator, Iterator, Iterator> Key;
//
namespace std {
template<> struct hash<Key> {
    size_t operator()(const Key & x) const {
        Iterator first1, last1, first2;
        tie(first1, last1, first2) = x;

        int result = *first1;
        result = result * 31 + *last1;
        result = result * 31 + *first2;
        result = result * 31 + *(next(first2, distance(first1, last1)-1));
        return result;
    }
};
}

// LeetCode, Scramble String
//
```

•

13.8 Minimum Path Sum

Given a $m \times n$ 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

Unique Paths (§??)

f[i][j] $(0, 0)$ (i, j)

f[i][j]=min(f[i-1][j], f[i][j-1])+grid[i][j]

```
// LeetCode, Minimum Path Sum
//
class Solution {
public:
    int minPathSum(vector<vector<int> > &grid) {
        const int m = grid.size();
        const int n = grid[0].size();
        this->f = vector<vector<int> >(m, vector<int>(n, -1));
        return dfs(grid, m-1, n-1);
    }
private:
    vector<vector<int> > f;  //

    int dfs(const vector<vector<int> > &grid, int x, int y) {
```

```
// LeetCode, Minimum Path Sum
//
class Solution {
public:
    int minPathSum(vector<vector<int> > &grid) {
        if (grid.size() == 0) return 0;
        const int m = grid.size();
        const int n = grid[0].size();

        int f[m][n];
        f[0][0] = grid[0][0];
        for (int i = 1; i < m; i++) {
            f[i][0] = f[i - 1][0] + grid[i][0];
        }
        for (int i = 1; i < n; i++) {
            f[0][i] = f[0][i - 1] + grid[0][i];
        }

        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                f[i][j] = min(f[i - 1][j], f[i][j - 1]) + grid[i][j];
            }
        }
        return f[m - 1][n - 1];
    }
};
```

```
// LeetCode, Minimum Path Sum
//
class Solution {
public:
    int minPathSum(vector<vector<int> > &grid) {
        const int m = grid.size();
        const int n = grid[0].size();

        int f[n];
        fill(f, f+n, INT_MAX); //  INT_MAX
        f[0] = 0;

        for (int i = 0; i < m; i++) {
            f[0] += grid[i][0];
            for (int j = 1; j < n; j++) {
                //
                //
                f[j] = min(f[j - 1], f[j]) + grid[i][j];
            }
        }
        return f[n - 1];
    }
};
```

- Unique Paths, §??
- Unique Paths II, §??

13.9 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:

- Insert a character
- Delete a character
- Replace a character

f[i][j] A[0,i] B[0,j] A[0,i] str1cB[0,j] str2d

1. c==d f[i][j]=f[i-1][j-1]
2. c!=d
   (a) f[i][j]=f[i-1][j-1]+1
   (b) f[i][j]=f[i][j-1]+1
   (c) f[i][j]=f[i-1][j]+1

```
// LeetCode, Edit Distance
```

```
// LeetCode, Edit Distance
//
```

-

13.10 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.

Climbing Stairs (§??)

```
// LeetCode, Decode Ways
```

- Climbing Stairs, §??

13.11 Distinct Subsequences

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

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

$f(i,j)$ T[0,j] S[0,i] S[i] T[j] S[i] $f(i,j) = f(i-1,j)$ S[i]==T[j] S[i] $f(i,j) = f(i-1,j) + f(i-1,j-1)$

```
// LeetCode, Distinct Subsequences
//
```

-

13.12 Word Break

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given

s = "leetcode",

dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".

$f(i)$ s[0,i]

$$f(i) = any\_of(f(j) \&\& s[j+1,i] \in dict), 0 \leq j < i$$

```
// LeetCode, Word Break
//
```

```
// LeetCode, Word Break
```

- Word Break II, §??

13.13 Word Break II

Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word.

Return all such possible sentences.

For example, given

s = "catsanddog",

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

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

```
// LeetCode, Word Break II
```

- Word Break, §??

```
//
struct UndirectedGraphNode {
    int label;
    vector<UndirectedGraphNode *> neighbors;
    UndirectedGraphNode(int x) : label(x) {};
};
```

14.1 Clone Graph

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

OJ's undirected graph serialization: Nodes are labeled uniquely.

We use as a separator for each node, and as a separator for node label and each neighbour of the node. As an example, consider the serialized graph {}.

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

1. First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
2. Second node is labeled as 1. Connect node 1 to node 2.
3. Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.

Visually, the graph looks like the following:

```
       1
      / \
     /   \
    0 --- 2
         / \
         \_/
```

DFS

```
// LeetCode, Clone Graph
```

BFS

```
// LeetCode, Clone Graph
```

•

15.1 Reverse Integer

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

Have you thought about this?

Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

Throw an exception? Good, but what if throwing an exception is not an option? You would then have to re-design the function (ie, add an extra parameter).

```
//LeetCode, Reverse Integer
```

- Palindrome Number, §??

15.2 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.

Palindrome reverse()

```
//LeetCode, Palindrome Number
```

- Reverse Integer, §??
- Valid Palindrome, §??

15.3 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 , insert and merge in as .

Example 2: Given , insert and merge in as .

This is because the new interval overlaps with .

```
struct Interval {
    int start;
    int end;
    Interval() : start(0), end(0) { }
    Interval(int s, int e) : start(s), end(e) { }
};

//LeetCode, Insert Interval
```

- Merge Intervals §??

15.4 Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

For example, Given , return

```
struct Interval {
    int start;
    int end;
    Interval() : start(0), end(0) { }
    Interval(int s, int e) : start(s), end(e) { }
};

//LeetCode, Merge Interval
//
```

- Insert Interval §??

15.5 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,

Minimum window is .

Note:

- If there is no such window in S that covers all characters in T, return the emtpy string .
- If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

T

```
// LeetCode, Minimum Window Substring
```

-

15.6 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.

$2^{31} - 1 = 2147483647$

```
// LeetCode, Multiply Strings
// @author  (http://weibo.com/lianchengzju)
// LeetCode, Multiply Strings
```

•

15.7 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: .(order does not matter).

```
// LeetCode, Substring with Concatenation of All Words
```

•

15.8 Pascal's Triangle

Given $numRows$, generate the first $numRows$ of Pascal's triangle.

For example, given $numRows = 5$,

Return

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

```
// LeetCode, Pascal's Triangle
```

```
// LeetCode, Pascal's Triangle
```

- Pascal's Triangle II §??

15.9 Pascal's Triangle II

Given an index k, return the k^{th} row of the Pascal's triangle.

For example, given $k = 3$,

Return .

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

```
// LeetCode, Pascal's Triangle II
```

- Pascal's Triangle §??

15.10 Spiral Matrix

Given a matrix of $m \times n$ 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].

```
// LeetCode, Spiral Matrix
// @author  (http://weibo.com/luangong)
```

- Spiral Matrix II §??

15.11 Spiral Matrix II

Given an integer n, generate a square matrix filled with elements from 1 to n^2 in spiral order.

For example, Given $n = 3$,

You should return the following matrix:

```
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

```
// LeetCode, Spiral Matrix II
```

```
// LeetCode, Spiral Matrix II
// @author  (http://weibo.com/luangong)
```

- Spiral Matrix, §??

15.12 ZigZag Conversion

The string 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:

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

```
string convert(string text, int nRows);
```

should return .

n=4:

```
P     I     N
A   L S   I G
Y A   H R
P     I
```

n=5:

```
P       H
A     S I
Y   I   R
P L     I   G
A       N
```

$(i, j) = (j + 1) * n + i (i, j) = (j + 1) * n - i$

```
// LeetCode, ZigZag Conversion
```

•

15.13 Divide Two Integers

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

```
// LeetCode, Divide Two Integers
```

```
// LeetCode, Divide Two Integers
```

•

15.14 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:

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.

Corner Cases:

- A line other than the last line might contain only one word. What should you do in this case?
- In this case, that line should be left

```
// LeetCode, Text Justification
```

-

15.15 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.

n $\frac{1}{2}n(n+1)$ n $O(n^3)$

$O(n^2)$ $O(n)$

```
// LeetCode, Max Points on a Line
```

```
// LeetCode, Max Points on a Line
```

-