

## 目录

1. Two Sum.....	4
2. Add Two Numbers.....	6
3. Longest Substring Without Repeating Characters★★.....	8
4. Median of Two Sorted Arrays★★.....	10
5. Longest Palindromic Substring★★.....	12
6. ZigZag Conversion.....	14
7. Reverse Integer.....	16
8. String to Integer (atoi).....	18
9. Palindrome Number.....	21
10. Regular Expression Matching★★.....	23
11. Container With Most Water.....	27
12. Integer to Roman.....	29
13. Roman to Integer.....	32
14. Longest Common Prefix.....	35
15. 3Sum.....	37
16. 3Sum Closest.....	39
17. Letter Combinations of a Phone Number.....	41
18. 4Sum.....	43
19. Remove Nth Node From End of List.....	46
20. Valid Parentheses.....	48
21. Merge Two Sorted Lists.....	50
22. Generate Parentheses.....	52
23. Merge k Sorted Lists★★.....	54
24. Swap Nodes in Pairs.....	56
25. Reverse Nodes in k-Group.....	58
26. Remove Duplicates from Sorted Array.....	60
27. Remove Element.....	62
28. Implement strStr()★★.....	64
29. Divide Two Integers★★.....	66
30. Substring with Concatenation of All Words★★.....	68
31. Next Permutation.....	70
32. Longest Valid Parentheses★★.....	72
33. Search in Rotated Sorted Array.....	75
34. Find First and Last Position of Element in Sorted Array★★.....	77
35. Search Insert Position.....	79
36. Valid Sudoku.....	81
37. Sudoku Solver.....	84
38. Count and Say.....	86
39. Combination Sum.....	88
40. Combination Sum II★★.....	90
41. First Missing Positive★★.....	92
42. Trapping Rain Water★★.....	94
43. Multiply Strings.....	96

44. Wildcard Matching ★ ★	98
45. Jump Game II	101
46. Permutations	103
47. Permutations II ★ ★	106
48. Rotate Image	110
49. Group Anagrams	113
50. Pow(x, n) ★ ★	115
51. N-Queens	117
52. N-Queens II	120
53. Maximum Subarray	123
54. Spiral Matrix	125
55. Jump Game	127
56. Merge Intervals ★ ★	129
57. Insert Interval ★ ★	131
58. Length of Last Word	133
59. Spiral Matrix II	135
60. Permutation Sequence	137
61. Rotate List	139
62. Unique Paths	141
63. Unique Paths II	143
64. Minimum Path Sum	145
65. Valid Number ★ ★	147
66. Plus One	149
67. Add Binary	151
68. Text Justification	153
69. Sqrt(x) ★ ★	156
70. Climbing Stairs	157
71. Simplify Path	159
72. Edit Distance ★ ★	162
73. Set Matrix Zeroes ★ ★	164
74. Search a 2D Matrix	167
75. Sort Colors ★ ★	169
76. Minimum Window Substring ★ ★	171
77. Combinations	173
78. Subsets	175
79. Word Search	179
80. Remove Duplicates from Sorted Array II	181
81. Search in Rotated Sorted Array II ★ ★	183
82. Remove Duplicates from Sorted List II	185
83. Remove Duplicates from Sorted List	188
84. Largest Rectangle in Histogram	190
85. Maximal Rectangle ★ ★	192
86. Partition List	194
87. Scramble String ★ ★	196

88. Merge Sorted Array.....	200
89. Gray Code.....	202
90. Subsets II ★ ★ .....	204
91. Decode Ways.....	207
92. Reverse Linked List II .....	209
93. Restore IP Addresses .....	211
94. Binary Tree Inorder Traversal .....	213
95. Unique Binary Search Trees II ★ ★ .....	215
96. Unique Binary Search Trees.....	217
97. Interleaving String.....	219
98. Validate Binary Search Tree .....	221
99. Recover Binary Search Tree ★ ★ .....	223
100. Same Tree .....	226

# 1. Two Sum

Easy

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,

return [0, 1].

```
class Solution {  
public:  
    vector<int> twoSum(vector<int>& nums, int target) {  
  
    }  
};
```

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> my_map;
        for (int i = 0; i < nums.size(); i++){
            if (my_map.count(target-nums[i]) && i != my_map[target-nums[i]]) {
                return {i, my_map[target-nums[i]]};
            }
            my_map[nums[i]] = i;
        }
    }
};
```

## 2. Add Two Numbers

### Medium

You are given two **non-empty** linked lists representing two non-negative integers. 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.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

### Example:

**Input:** (2 -> 4 -> 3) + (5 -> 6 -> 4)

**Output:** 7 -> 0 -> 8

**Explanation:** 342 + 465 = 807.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */

class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {

    }
};
```

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode *l1, ListNode *l2) {
        ListNode *res = new ListNode(-1), *p = res;
        int c = 0;
        while (l1 || l2){
            int a = l1 ? l1->val : 0;
            int b = l2 ? l2->val : 0;
            p->next = new ListNode((c + a + b) % 10);
            c = (c + a + b) / 10;
            p = p->next;
            if (l1) l1 = l1->next;
            if (l2) l2 = l2->next;
        }
        if (c > 0) p->next = new ListNode(c);
        return res->next;
    }
};
```

### 3. Longest Substring Without Repeating Characters ★ ★

#### Medium

Given a string, find the length of the **longest substring** without repeating characters.

#### Example 1:

**Input:** "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

#### Example 2:

**Input:** "bbbb"

**Output:** 1

**Explanation:** The answer is "b", with the length of 1.

#### Example 3:

**Input:** "pwwkew"

**Output:** 3

**Explanation:** The answer is "wke", with the length of 3.

Note that the answer must be a **substring**, "pwke" is a *subsequence* and not a substring.

```
class Solution {  
public:  
    int lengthOfLongestSubstring(string s) {  
  
    }  
};
```



```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        if (s.empty()) return 0;
        int n = s.length(), left = -1, res = 1;
        vector<int> m(128,-1);
        for (int i = 0; i < n; i++) {
            left = max(left, m[s[i]]);
            m[s[i]] = i;
            res = max(res, i-left);
        }
        return res;
    }
};
```

## 4. Median of Two Sorted Arrays ★ ★

### Hard

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

You may assume **nums1** and **nums2** cannot be both empty.

#### Example 1:

```
nums1 = [1, 3]
```

```
nums2 = [2]
```

The median is 2.0

#### Example 2:

```
nums1 = [1, 2]
```

```
nums2 = [3, 4]
```

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

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

    }
};
```

```

class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int m = nums1.size(), n = nums2.size();
        double res = findKth(nums1, 0, nums2, 0, (m+n+1)/2);
        if ((m+n) % 2 != 0) return res;
        res += findKth(nums1, 0, nums2, 0, (m+n+2)/2);
        return res / 2.0;
    }

private:
    int findKth(vector<int>& nums1, int i, vector<int>& nums2, int j, int k) {
        if (i >= nums1.size()) return nums2[j + k - 1];
        if (j >= nums2.size()) return nums1[i + k - 1];
        if (k == 1) return min(nums1[i], nums2[j]);
        int midVal1 = (i+k/2-1 < nums1.size()) ? nums1[i+k/2-1] : INT_MAX;
        int midVal2 = (j+k/2-1 < nums2.size()) ? nums2[j+k/2-1] : INT_MAX;
        if (midVal1 < midVal2) return findKth(nums1, i+k/2, nums2, j, k-k/2);
        else return findKth(nums1, i, nums2, j+k/2, k-k/2);
    }
};

```

## 5. Longest Palindromic Substring ★ ★

### Medium

Given a string *s*, find the longest palindromic substring in *s*. You may assume that the maximum length of *s* is 1000.

#### Example 1:

**Input:** "babad"

**Output:** "bab"

**Note:** "aba" is also a valid answer.

#### Example 2:

**Input:** "cbbd"

**Output:** "bb"

```
class Solution {  
public:  
    string longestPalindrome(string s) {  
  
    }  
};
```

```

class Solution {
public:
    string longestPalindrome(string s) {
        string T = preProcess(s);
        const int n = T.length();
        int P[n];
        int C = 0, R = 0;
        for (int i = 1; i < n - 1; i++) {
            int i_mirror = 2 * C - i;
            P[i] = (R > i) ? min(R - i, P[i_mirror]) : 0;
            while (T[i + 1 + P[i]] == T[i - 1 - P[i]]) P[i]++;
            if (i + P[i] > R) {
                C = i;
                R = i + P[i];
            }
        }
        int max_len = 0, center_index = 0;
        for (int i = 1; i < n - 1; i++) {
            if (P[i] > max_len) {
                max_len = P[i];
                center_index = i;
            }
        }
        return s.substr((center_index - 1 - max_len) / 2, max_len);
    }
private:
    string preProcess(string &s) {
        int n = s.length();
        if (n == 0) return "^$";
        string ret = "^";
        for (int i = 0; i < n; i++) ret += "#" + s.substr(i, 1);
        return ret + "#$";
    }
};

```

## 6. ZigZag Conversion

### Medium

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

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

And then read line by line: "PAHNAPLSIIGYIR"

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

```
string convert(string s, int numRows);
```

#### Example 1:

**Input:** s = "PAYPALISHIRING", numRows = 3

**Output:** "PAHNAPLSIIGYIR"

#### Example 2:

**Input:** s = "PAYPALISHIRING", numRows = 4

**Output:** "PINALSIGYAHRPI"

#### Explanation:

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

```
class Solution {
public:
    string convert(string s, int numRows) {

    }
};
```

```
class Solution {
public:
    string convert(string s, int numRows) {
        if (numRows == 1) return s;
        int n = s.length();
        string res;
        for (int i = 0; i < numRows; i++) {
            for (int j = i; j < n; j += 2*numRows-2) {
                res += s[j];
                if (i == 0 || i == numRows-1) continue;
                if (j+2*(numRows-i-1) < n) res += s[j+2*(numRows-i-1)];
            }
        }
        return res;
    }
};
```

## 7. Reverse Integer

Easy

Given a 32-bit signed integer, reverse digits of an integer.

**Example 1:**

**Input:** 123

**Output:** 321

**Example 2:**

**Input:** -123

**Output:** -321

**Example 3:**

**Input:** 120

**Output:** 21

**Note:**

Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range:  $[-2^{31}, 2^{31} - 1]$ . For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

```
class Solution {
public:
    int reverse(int x) {

    }
};
```



```
class Solution {
public:
    int reverse(int x) {
        int rev = 0;
        while (x != 0) {
            int pop = x % 10;
            x /= 10;
            if (rev > INT_MAX/10 || (rev == INT_MAX/10 && pop > 7)) return 0;
            if (rev < INT_MIN/10 || (rev == INT_MIN/10 && pop < -8)) return 0;
            rev = rev * 10 + pop;
        }
        return rev;
    }
};
```

## 8. String to Integer (atoi)

### Medium

Implement `atoi` which converts a string to an integer.

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.

#### Note:

- Only the space character ' ' is considered as whitespace character.
- Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range:  $[-2^{31}, 2^{31} - 1]$ . If the numerical value is out of the range of representable values, `INT_MAX` ( $2^{31} - 1$ ) or `INT_MIN` ( $-2^{31}$ ) is returned.

#### Example 1:

**Input:** "42"

**Output:** 42

#### Example 2:

**Input:** " -42"

**Output:** -42

**Explanation:** The first non-whitespace character is '-', which is the minus sign.

Then take as many numerical digits as possible, which gets 42.

#### Example 3:

**Input:** "4193 with words"

**Output:** 4193

**Explanation:** Conversion stops at digit '3' as the next character is not a numerical digit.

#### Example 4:

**Input:** "words and 987"

**Output:** 0

**Explanation:** The first non-whitespace character is 'w', which is not a numerical digit or a +/- sign. Therefore no valid conversion could be performed.

**Example 5:**

**Input:** "-91283472332"

**Output:** -2147483648

**Explanation:** The number "-91283472332" is out of the range of a 32-bit signed integer. Therefore  $\text{INT\_MIN} (-2^{31})$  is returned.

```
class Solution {
public:
    int myAtoi(string str) {

    }
};
```

```
class Solution {
public:
    int myAtoi(string str) {
        long res = 0;
        int sign = 1, n = str.length(), i = str.find_first_not_of(' ');
        if (str[i] == '+' || str[i] == '-') sign = str[i++] == '+' ? 1 : -1;

        while (i < n && isdigit(str[i])) {
            res = res * 10 + str[i++] - '0';
            if (res * sign > INT_MAX) return INT_MAX;
            if (res * sign < INT_MIN) return INT_MIN;
        }
        return res * sign;
    }
};
```

## 9. Palindrome Number

Easy

Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

**Example 1:**

**Input:** 121

**Output:** true

**Example 2:**

**Input:** -121

**Output:** false

**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

**Example 3:**

**Input:** 10

**Output:** false

**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

**Follow up:**

Could you solve it without converting the integer to a string?

```
class Solution {
    public:
        bool isPalindrome(int x) {

        }
}
```

```

class Solution {
public:
    bool isPalindrome(int x) {
        if (x < 0 || (x % 10 == 0 && x != 0))    return false;
        int y = 0, n = x;
        while (n) {
            if (y >= INT_MAX / 10) return false;
            y = y * 10 + n % 10;
            n /= 10;
        }
        return x == y;
    }
};

```

////////////////////////////////////

```

class Solution {
public:
    bool isPalindrome(int x) {
        // 特殊情况:
        // 如上所述, 当 x < 0 时, x 不是回文数。
        // 同样地, 如果数字的最后一位是 0, 为了使该数字为回文,
        // 则其第一位数字也应该是 0
        // 只有 0 满足这一属性
        if (x < 0 || (x % 10 == 0 && x != 0))    return false;
        int revertedNumber = 0;
        while (x > revertedNumber) {
            revertedNumber = revertedNumber * 10 + x % 10;
            x /= 10;
        }
        // 当数字长度为奇数时, 我们可以通过 revertedNumber/10 去除处于中位的数字。
        // 例如, 当输入为 12321 时, 在 while 循环的末尾我们可以得到 x = 12,
        revertedNumber = 123,
        // 由于处于中位的数字不影响回文 (它总是与自己相等), 所以我们可以简单地将其去除。
        return x == revertedNumber || x == revertedNumber/10;
    }
};

```

## 10. Regular Expression Matching ★ ★

### Hard

Given an input string (*s*) and a pattern (*p*), 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).

### Note:

- `s` could be empty and contains only lowercase letters `a-z`.
- `p` could be empty and contains only lowercase letters `a-z`, and characters like `.` or `*`.

### Example 1:

#### Input:

```
s = "aa"
```

```
p = "a"
```

**Output:** false

**Explanation:** "a" does not match the entire string "aa".

### Example 2:

#### Input:

```
s = "aa"
```

```
p = "a*"
```

**Output:** true

**Explanation:** '\*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

### Example 3:

#### Input:

```
s = "ab"
```

```
p = ".*"
```

**Output:** true

**Explanation:** `".*"` means "zero or more (\*) of any character (.)".

**Example 4:**

**Input:**

`s = "aab"`

`p = "c*a*b"`

**Output:** true

**Explanation:** c can be repeated 0 times, a can be repeated 1 time. Therefore it matches "aab".

**Example 5:**

**Input:**

`s = "mississippi"`

`p = "mis*is*p*."`

**Output:** false

```
class Solution {
public:
    bool isMatch(string s, string p) {

    }
};
```



```
class Solution {
public:
    bool isMatch(string s, string p) {
        if (p.empty()) return s.empty();
        if (p.length() > 1 && p[1] == '*') {
            return isMatch(s, p.substr(2))
                || (!s.empty() && (s[0] == p[0] || p[0] == '.')
                    && isMatch(s.substr(1), p));
        }
        else return !s.empty() && (s[0] == p[0] || p[0] == '.')
            && isMatch(s.substr(1), p.substr(1));
    }
};
```

```

//////////////////////////////////dp//////////////////////////////////
class Solution {
public:
    bool isMatch(string s, string p) {
        if (p.empty()) return s.empty();
        int m = s.size(), n = p.size(), k = 0;
        vector<vector<bool>> dp(2, vector<bool>(n + 1, false));
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                if (j == 0) dp[k][j] = i == 0;
                else if (p[j-1] == '*') {
                    dp[k][j] = dp[k][j-2]
                        || (i && dp[k^1][j]
                            && (s[i-1] == p[j-2] || p[j-2] == '.'));
                } else {
                    dp[k][j] = i && dp[k^1][j-1]
                        && (s[i-1] == p[j-1] || p[j-1] == '.');
                }
            }
            k ^= 1;
        }
        return dp[k^1][n];
    }
};

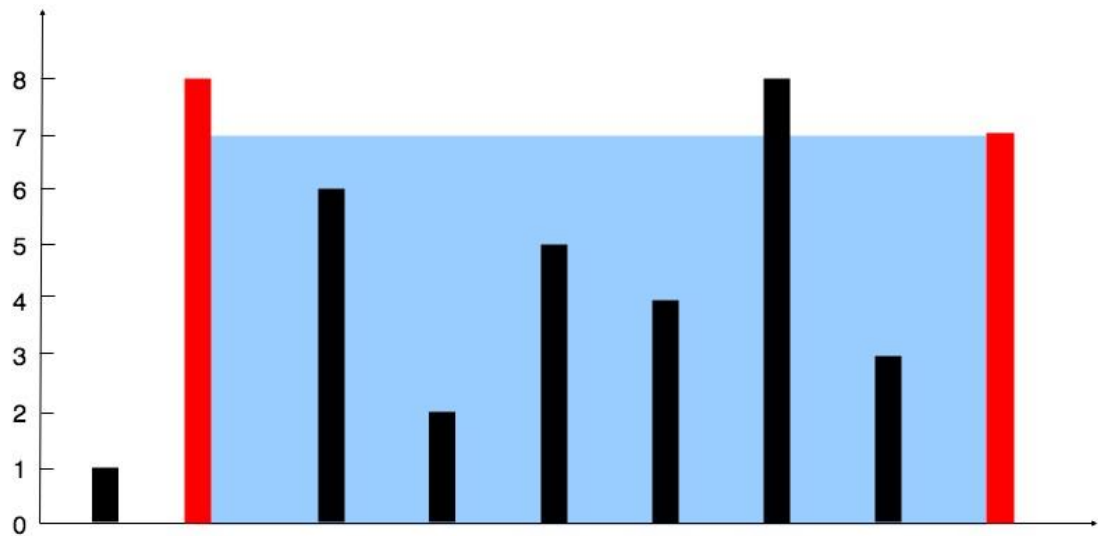
```

## 11. Container With Most Water

### Medium

Given  $n$  non-negative integers  $a_1, a_2, \dots, a_n$ , 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 and  $n$  is at least 2.



The above vertical lines are represented by array  $[1, 8, 6, 2, 5, 4, 8, 3, 7]$ . In this case, the max area of water (blue section) the container can contain is 49.

### Example:

**Input:**  $[1, 8, 6, 2, 5, 4, 8, 3, 7]$

**Output:** 49

```
class Solution {
public:
    int maxArea(vector<int>& height) {

    }
};
```

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        int MAX = -1, L = 0, R = height.size()-1;
        while (L < R){
            MAX = max(MAX, (R-L) * min(height[L], height[R]));
            height[L] > height[R] ? R-- : L++;
        }
        return MAX;
    }
};
```

## 12. Integer to Roman

### Medium

Roman numerals are represented by seven different symbols: **I**, **V**, **X**, **L**, **C**, **D** and **M**.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, two is written as **II** in Roman numeral, just two one's added together. Twelve is written as, **XII**, which is simply **X** + **II**. The number twenty seven is written as **XXVII**, which is **XX** + **V** + **II**.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not **IIII**. Instead, the number four is written as **IV**. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as **IX**. There are six instances where subtraction is used:

- I** can be placed before **V** (5) and **X** (10) to make 4 and 9.
- X** can be placed before **L** (50) and **C** (100) to make 40 and 90.
- C** can be placed before **D** (500) and **M** (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

#### Example 1:

**Input:** 3

**Output:** "III"

#### Example 2:

**Input:** 4

**Output:** "IV"

#### Example 3:

**Input:** 9

**Output:** "IX"

**Example 4:**

**Input:** 58

**Output:** "LVIII"

**Explanation:** L = 50, V = 5, III = 3.

**Example 5:**

**Input:** 1994

**Output:** "MCMXCIV"

**Explanation:** M = 1000, CM = 900, XC = 90 and IV = 4.

```
class Solution {  
public:  
    string intToRoman(int num) {  
  
    }  
};
```

```
class Solution {
public:
    string intToRoman(int num) {
        string res = "";
        vector<int> val{1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
        vector<string> str{"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X",
                           "IX", "V", "IV", "I"};
        for (int i = 0; i < val.size(); ++i) {
            while (num >= val[i]) {
                num -= val[i];
                res += str[i];
            }
        }
        return res;
    }
};
```

## 13. Roman to Integer

Easy

Roman numerals are represented by seven different symbols: **I**, **V**, **X**, **L**, **C**, **D** and **M**.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, two is written as **II** in Roman numeral, just two one's added together. Twelve is written as, **XII**, which is simply **X** + **II**. The number twenty seven is written as **XXVII**, which is **XX** + **V** + **II**.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not **IIII**. Instead, the number four is written as **IV**. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as **IX**. There are six instances where subtraction is used:

- I** can be placed before **V** (5) and **X** (10) to make 4 and 9.
- X** can be placed before **L** (50) and **C** (100) to make 40 and 90.
- C** can be placed before **D** (500) and **M** (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

**Example 1:**

**Input:** "III"

**Output:** 3

**Example 2:**

**Input:** "IV"

**Output:** 4

**Example 3:**



**Input:** "IX"

**Output:** 9

**Example 4:**

**Input:** "LVIII"

**Output:** 58

**Explanation:** L = 50, V= 5, III = 3.

**Example 5:**

**Input:** "MCMXCIV"

**Output:** 1994

**Explanation:** M = 1000, CM = 900, XC = 90 and IV = 4.

```
class Solution {  
public:  
    int romanToInt(string s) {  
  
    }  
};
```

```
class Solution {
public:
    int romanToInt(string s) {
        if (s.empty()) return 0;
        map<char, int> My_map{{'I',1},{'V',5},{'X', 10},{'L',50},
                               {'C',100},{'D',500},{'M', 1000}};
        int len = s.length(), res = My_map[s[len-1]];
        for(int i = 0; i < len-1; i++){
            if (My_map[s[i]] < My_map[s[i+1]]) res -= My_map[s[i]];
            else res += My_map[s[i]];
        }
        return res;
    }
};
```

## 14. Longest Common Prefix

Easy

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

If there is no common prefix, return an empty string `""`.

**Example 1:**

**Input:** ["flower","flow","flight"]

**Output:** "fl"

**Example 2:**

**Input:** ["dog","racecar","car"]

**Output:** ""

**Explanation:** There is no common prefix among the input strings.

**Note:**

All given inputs are in lowercase letters `a-z`.

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {

    }
};
```

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        string res = "";
        if (strs.empty()) return res;
        int len = strs[0].length();
        for (int i = 0; i < len; i++){
            char c = strs[0][i];
            for (auto &str : strs){
                if (str.length() < i || str[i] != c) return res;
            }
            res += c;
        }
        return res;
    }
};
```

## 15. 3Sum

### Medium

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

### Note:

The solution set must not contain duplicate triplets.

### Example:

Given array `nums = [-1, 0, 1, 2, -1, -4]`,

A solution set is:

```
[  
  [-1, 0, 1],  
  [-1, -1, 2]  
]
```

```
class Solution {  
public:  
    vector<vector<int>> threeSum(vector<int>& nums) {  
  
        }  
};
```

```

class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> ans;
        if (nums.size() < 3) return ans;
        sort(nums.begin(), nums.end());
        for (auto i = nums.begin(); i != nums.end()-2; i++){
            if (i > nums.begin() && *i == *(i-1)) continue;
            auto j = i + 1, k = nums.end() - 1;
            while (j < k){
                if (*i + *j + *k < 0) {
                    j++;
                    while (j < k && *j == *(j-1)) j++;
                }
                else if (*i + *j + *k > 0) {
                    k--;
                    while (j < k && *k == *(k+1)) k--;
                }
                else {
                    ans.push_back({*i,*j,*k});
                    j++; k--;
                    while (j < k && *j == *(j-1)) j++;
                    while (j < k && *k == *(k+1)) k--;
                }
            }
        }
        return ans;
    }
};

```

## 16. 3Sum Closest

### Medium

Given an array `nums` of  $n$  integers and an integer `target`, find three integers in `nums` such that the sum is closest to `target`. Return the sum of the three integers. You may assume that each input would have exactly one solution.

### Example:

Given array `nums = [-1, 2, 1, -4]`, and `target = 1`.

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

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {

    }
};
```

```

class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int ans, minn = numeric_limits<int>::max();
        for (auto i = nums.begin(); i != nums.end() - 2; i++){
            auto j = i + 1, k = nums.end() - 1;
            while (j < k) {
                int sum = *i + *j + *k;
                if (minn > abs(sum - target)) {
                    minn = abs(sum - target);
                    ans = sum;
                }
                if (sum < target)      j++;
                else if (sum > target) k--;
                else return target;
            }
        }
        return ans;
    }
};

```



## 17. Letter Combinations of a Phone Number

### Medium

Given a string containing digits from 2–9 inclusive, 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. Note that 1 does not map to any letters.



### Example:

**Input:** "23"

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

### Note:

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

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {

    }
};
```

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        vector<string> res;
        if (digits.empty()) return res;
        dfs("", 0, digits, res);
        return res;
    }
private:
    string a[8] = {"abc", "def", "ghi", "jkl",
                  "mno", "pqrs", "tuv", "wxyz"};

    void dfs(string s, int i, string &digits, vector<string> &res){
        if (i == digits.length()) res.push_back(s);
        else for (auto c : a[digits[i]-'2']) {
            dfs(s+c, i+1, digits, res);
        }
    }
};
```

## 18. 4Sum

### Medium

Given an array `nums` of  $n$  integers and an integer `target`, are there elements  $a$ ,  $b$ ,  $c$ , and  $d$  in `nums` such that  $a + b + c + d = \text{target}$ ? Find all unique quadruplets in the array which gives the sum of `target`.

### Note:

The solution set must not contain duplicate quadruplets.

### Example:

Given array `nums` = [1, 0, -1, 0, -2, 2], and `target` = 0.

A solution set is:

```
[
  [-1, 0, 0, 1],
  [-2, -1, 1, 2],
  [-2, 0, 0, 2]
]
```

```
class Solution {
public:
    vector< vector<int> > fourSum(vector<int> &nums, int target) {

    }
};
```

```

class Solution {
public:
    vector<vector<int>> fourSum(vector<int> &nums, int target) {
        vector<vector<int>> res;
        int n = nums.size();
        sort(nums.begin(), nums.end());
        for (int i = 0; i < n-3; ++i) {
            if (i > 0 && nums[i] == nums[i-1]) continue;
            for (int j = i+1; j < n-2; ++j) {
                if (j > i+1 && nums[j] == nums[j-1]) continue;
                int left = j+1, right = n-1;
                while (left < right) {
                    int sum = nums[i] + nums[j] + nums[left] + nums[right];
                    if (sum == target) {
                        res.push_back( {nums[i], nums[j],
                                        nums[left], nums[right]} );
                        while (left < right && nums[left] == nums[left+1])
                            ++left;
                        while (left < right && nums[right] == nums[right-1])
                            --right;
                        ++left; --right;
                    }
                    else if (sum < target) ++left;
                    else --right;
                }
            }
        }
        return res;
    }
};

```

```

//////////////////////////////////好O(n^2) 坏O(n^4)//////////////////////////////////
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        vector<vector<int>> res;
        unordered_map<int, vector<pair<int,int>>> My_map;
        int n = nums.size();
        if (n < 4) return res;
        sort(nums.begin(), nums.end());
        for (auto i = 0; i < n; i++){
            for (int j = i+1; j < n; j++)
                My_map[nums[i]+nums[j]].emplace_back(i, j);
        }
        for (auto &i : My_map) if (My_map.count(target-i.first)){
            auto j = My_map.find(target-i.first);
            for(auto &[a, b] : i.second){
                for(auto &[c, d] : j->second) {
                    if (b >= c) continue;
                    res.push_back({nums[a],nums[b],nums[c],nums[d]});
                }
            }
        }
        sort(res.begin(), res.end());
        res.erase(unique(res.begin(), res.end()), res.end());
        return res;
    }
};

```

## 19. Remove Nth Node From End of List

### Medium

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

#### Example:

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

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

#### Note:

Given  $n$  will always be valid.

#### Follow up:

Could you do this in one pass?

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {

    }
};
```

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *pre = dummy, *p = pre;
        While (n--) p = p->next;
        While (p->next) {
            p = p->next;
            pre = pre->next;
        }
        pre->next = pre->next->next;
        return dummy->next;
    }
};
```

## 20. Valid Parentheses

Easy

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

**Example 1:**

**Input:** "()"

**Output:** true

**Example 2:**

**Input:** "()[]{}"

**Output:** true

**Example 3:**

**Input:** "(]"

**Output:** false

**Example 4:**

**Input:** "(D]"

**Output:** false

**Example 5:**

**Input:** "{[]}"

**Output:** true

```
class Solution {
public:
    bool isValid(string s) {

    }
};
```



```
class Solution {
public:
    unordered_map<char, char> mp{{'{', '}'}, {'(', ')'}, {'[', ']'}};

    bool isValid(string s) {
        stack<char> Stack;
        for (auto c : s){
            if (mp.count(c)) Stack.push(c);
            else if (!Stack.empty() && mp[Stack.top()] == c) {
                Stack.pop();
            }
            else return false;
        }
        return Stack.empty();
    }
};
```

## 21. Merge Two Sorted Lists

Easy

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.

**Example:**

**Input:** 1->2->4, 1->3->4

**Output:** 1->1->2->3->4->4

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {

    }
};
```

```

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode *dummy = new ListNode(-1), *p = dummy;
        while (l1 && l2){
            if (l1->val < l2->val){
                p->next = l1;
                l1 = l1->next;
            }
            else {
                p->next = l2;
                l2 = l2->next;
            }
            p = p->next;
        }
        if (l1) p->next = l1;
        if (l2) p->next = l2;
        return dummy->next;
    }
};

```

## 22. Generate Parentheses

## Medium

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:

```
[
  "((0))",
  "(00)",
  "(0)0",
  "0(0)",
  "000"
]
```

```
class Solution {
public:

    vector<string> generateParenthesis(int n) {

    }

};
```

```
class Solution {
public:
    vector<string> res;
    vector<string> generateParenthesis(int n) {
        dfs(n, n, "");
        return res;
    }
    void dfs(int n, int m, string s){
        if (n < 0 || m < 0 || n > m) return;
        if (!n && !m){
            res.push_back(s);
            return;
        }
        dfs(n-1, m, s+'(');
        dfs(n, m-1, s+')');
    }
};
```

## 23. Merge k Sorted Lists★★

Hard

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

**Example:**

**Input:**

```
[
  1->4->5,
  1->3->4,
  2->6
]
```

**Output:** 1->1->2->3->4->4->5->6

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */

class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {

    }
};
```

```

class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        ListNode* dummy = new ListNode(-1), *p = dummy;
        auto cmp = [](const ListNode *lhs, const ListNode *rhs) {
            return lhs->val > rhs->val;
        };
        priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)> pq(cmp);
        for (auto &i : lists) if (i) {
            pq.push(i);
        }
        while (!pq.empty()) {
            auto t = pq.top();
            pq.pop();
            p->next = t;
            p = t;
            t = t->next;
            if (t) pq.push(t);
        }
        return dummy->next;
    }
};

```

## 24. Swap Nodes in Pairs

### Medium

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

#### Example:

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

#### Note:

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

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {

    }
};
```



```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *p = dummy, *q = p->next;
        while (q && q->next) {
            p->next = q->next;
            q->next = p->next->next;
            p->next->next = q;
            p = q;
            q = q->next;
        }
        return dummy->next;
    }
};
```

## 25. Reverse Nodes in k-Group

### Hard

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.

### 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

### Note:

- Only constant extra memory is allowed.
- You may not alter the values in the list's nodes, only nodes itself may be changed.

```
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {

    }
};
```

```

class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        if (head == nullptr || k < 2) return head;
        ListNode *head2 = head, *q;
        for (int i = 1; i < k; i++) {
            head2 = head2->next;
            if (head2 == nullptr) return head;
        }

        head2->next = reverseKGroup(head2->next, k);
        while (--k) {
            q = head->next;
            head->next = head2->next;
            head2->next = head;
            head = q;
        }
        return head2;
    }
};

```

## 26. Remove Duplicates from Sorted Array

Easy

Given a sorted array *nums*, 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 by **modifying the input array in-place** with O(1) extra memory.

### Example 1:

Given *nums* = [1,1,2],

Your function should return length = 2, with the first two elements of *nums* being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.

### Example 2:

Given *nums* = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of *nums* being modified to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.

### Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)

int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.

// using the length returned by your function, it prints the first len elements.

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

    print(nums[i]);

}
```

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.empty()) return 0;
        int index = 1;
        for (int i = 1; i < nums.size(); i++) {
            if (nums[i] != nums[index-1])
                nums[index++] = nums[i];
        }
        return index;
    }
};
```

## 27. Remove Element

Easy

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

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

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

### Example 1:

Given *nums* = [3,2,2,3], *val* = 3,

Your function should return length = 2, with the first two elements of *nums* being 2.

It doesn't matter what you leave beyond the returned length.

### Example 2:

Given *nums* = [0,1,2,2,3,0,4,2], *val* = 2,

Your function should return length = 5, with the first five elements of *nums* containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.

### Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)

int len = removeElement(nums, val);

// any modification to nums in your function would be known by the caller.

// using the length returned by your function, it prints the first len elements.

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

    print(nums[i]);

}
```

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int index = 0;
        for (int i = 0; i < nums.size(); i++){
            if (nums[i] != val)
                nums[index++] = nums[i];
        }
        return index;
    }
};
```

## 28. Implement strStr() ★ ★

Easy

Implement [strStr\(\)](#).

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

**Example 1:**

**Input:** haystack = "hello", needle = "ll"

**Output:** 2

**Example 2:**

**Input:** haystack = "aaaaa", needle = "bba"

**Output:** -1

**Clarification:**

What should we return when **needle** is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when **needle** is an empty string. This is consistent to C's [strstr\(\)](#) and Java's [indexOf\(\)](#).

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {

    }
};
```



```

class Solution {
public:
    int strStr(string haystack, string needle) {
        int m = haystack.size(), n = needle.size();
        if (!n) return 0;
        vector<int> f = get_fail(needle);
        for (int i = 0, j = 0; i < m; i++) {
            if (haystack[i] == needle[j]) {
                i++, j++;
            }
            if (j == n) return i-j;
            else if (i < m && haystack[i] != needle[j]) {
                j = j ? f[j - 1] : 0;
            }
        }
        return -1;
    }
private:
    vector<int> get_fail(string str) {
        int n = str.size();
        vector<int> f(n, 0);
        for (int i = 1, len = 0; i < n; i++) {
            if (str[i] == str[len]) f[i++] = ++len;
            else if (len) len = f[len - 1];
            else f[i++] = 0;
        }
        return f;
    }
};

```

## 29. Divide Two Integers★★

### Medium

Given two integers `dividend` and `divisor`, divide two integers without using multiplication, division and mod operator.

Return the quotient after dividing `dividend` by `divisor`.

The integer division should truncate toward zero.

#### Example 1:

**Input:** dividend = 10, divisor = 3

**Output:** 3

#### Example 2:

**Input:** dividend = 7, divisor = -3

**Output:** -2

#### Note:

- Both dividend and divisor will be 32-bit signed integers.
- The divisor will never be 0.
- Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range:  $[-2^{31}, 2^{31} - 1]$ . For the purpose of this problem, assume that your function returns  $2^{31} - 1$  when the division result overflows.

```
class Solution {
public:
    int divide(int dividend, int divisor) {

    }
};
```

```

class Solution {
public:
    int divide(int dividend, int divisor) {
        if (divisor == 0 || (dividend == INT_MIN && divisor == -1))
            return INT_MAX;
        int sign = (dividend > 0) ^ (divisor > 0) ? -1 : 1;
        long long Dividend = labs(dividend);
        long long Divisor = labs(divisor);
        int res = 0;
        while (Dividend >= Divisor) {
            long long mul = 1, temp = Divisor;
            while (Dividend > (temp << 1)) {
                mul <<= 1;
                temp <<= 1;
            }
            res += mul;
            Dividend -= temp;
        }
        return sign*res;
    }
};

```

## 30. Substring with Concatenation of All Words★★

### Hard

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.

#### Example 1:

##### Input:

```
s = "barfoothefoobarman",
```

```
words = ["foo","bar"]
```

**Output:** [0,9]

**Explanation:** Substrings starting at index 0 and 9 are "barfoo" and "foobar" respectively.

The output order does not matter, returning [9,0] is fine too.

#### Example 2:

##### Input:

```
s = "wordgoodgoodgoodbestword",
```

```
words = ["word","good","best","word"]
```

**Output:** []

```
class Solution {
public:
    vector<int> findSubstring(string s, vector<string>& words) {

    }
};
```

```

class Solution {
public:
    vector<int> findSubstring(string s, vector<string>& words) {
        vector<int> res;
        if (words.empty() || s.empty()) return res;
        unordered_map<string, int> M0, M1;
        const int N = words[0].length(), M = s.length(), K = words.size();
        for (auto &i : words) M0[i]++;
        for (int i = 0; i < N; i++) {
            int cnt = 0, left = i;
            M1.clear();
            for (int j = i; j+N <= M; j += N){
                string a = s.substr(j, N);
                if (M0.count(a)) {
                    if (++M1[a] <= M0[a]){
                        if (++cnt == K){
                            res.push_back(left);
                            cnt--; M1[s.substr(left, N)]--; left += N;
                        }
                    } else{
                        While (s.substr(left, N) != a) {
                            cnt--; M1[s.substr(left, N)]--; left += N;
                        }
                        left += N; M1[a]--;
                    }
                } else{
                    M1.clear();
                    left = j+N;
                    cnt = 0;
                }
            }
        }
        return res;
    }
};

```

## 31. Next Permutation

### Medium

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 and use only constant 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

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {

    }
};
```

```

class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        permutation(nums, 0, nums.size());
    }
private:
    bool permutation(vector<int> &nums, int begin, int end) {
        int p = end - 2;
        while (p > -1 && nums[p] >= nums[p+1]) p--;
        if (p == -1) {
            reverse(nums.begin(), nums.end());
            return false;
        }
        int c = end-1;
        while (nums[p] >= nums[c]) c--;
        swap(nums[p], nums[c]);
        reverse(nums.begin()+p+1, nums.end());
        return true;
    }
};

```

## 32. Longest Valid Parentheses ★ ★

### Hard

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

#### Example 1:

**Input:** "()"

**Output:** 2

**Explanation:** The longest valid parentheses substring is "()"

#### Example 2:

**Input:** ")()())"

**Output:** 4

**Explanation:** The longest valid parentheses substring is "()()"

```
class Solution {
public:
    int longestValidParentheses(string s) {

    }
};
```



```
/////////////////////////////////stack/////////////////////////////////
class Solution {
public:
    int longestValidParentheses(string s) {
        int res = 0, left = -1, n = s.length();
        stack<int> S;
        for (int i = 0; i < n; i++){
            if (s[i] == '(') S.push(i);
            else if (S.empty()) left = i;
            else {
                S.pop();
                if (S.empty()) res = max(res, i-left);
                else res = max(res, i-S.top());
            }
        }
        return res;
    }
};
```

```

////////////////////////////////dp////////////////////////////////////////
class Solution {
public:
    int longestValidParentheses(string s) {
        if (s.length() <= 1) return 0;
        int res = 0;
        vector<int> dp(s.size(), 0);
        //dp[i] 表示以 s[i] 结尾最长匹配括号的长度
        for (int i = 1; i < s.length(); i++){
            if (s[i] == '(') continue;
            if (s[i-1] == '(') {
                dp[i] = (i-2) >= 0 ? dp[i-2]+2 : 2;
                res = max(dp[i], res);
            } else if (i-dp[i-1]-1 >= 0 && s[i-dp[i-1]-1] == '(') {
                dp[i] = dp[i-1] + 2 + ((i-dp[i-1]-2)>=0) ? dp[i-dp[i-1]-2] : 0;
                res = max(dp[i], res);
            }
        }
        return res;
    }
};

```

## 33. Search in Rotated Sorted Array

### Medium

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]`).

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.

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

#### Example 1:

**Input:** `nums = [4,5,6,7,0,1,2]`, `target = 0`

**Output:** `4`

#### Example 2:

**Input:** `nums = [4,5,6,7,0,1,2]`, `target = 3`

**Output:** `-1`

```
class Solution {
public:
    int search(vector<int>& nums, int target) {

    }
};
```

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) return mid;
            else if (nums[mid] < nums[right]) {
                if(nums[mid] < target && nums[right] >= target) left = mid + 1;
                else right = mid - 1;
            } else {
                if(nums[left] <= target && nums[mid] > target) right = mid - 1;
                else left = mid + 1;
            }
        }
        return -1;
    }
};
```

## 34. Find First and Last Position of Element in Sorted Array★★

### Medium

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given `target` value.

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

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

#### Example 1:

**Input:** `nums = [5,7,7,8,8,10]`, `target = 8`

**Output:** `[3,4]`

#### Example 2:

**Input:** `nums = [5,7,7,8,8,10]`, `target = 6`

**Output:** `[-1,-1]`

```
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {

    }
};
```

```

class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        /*
        if (nums.empty()) return {-1, -1};
        auto p = equal_range(nums.begin(), nums.end(), target);
        if (p.first == nums.end() || *(p.first) != target) return {-1, -1};
        else return {p.first-nums.begin(), p.second-nums.begin()-1};
        */
        vector<int> res{-1, -1};
        if (nums.empty()) return res;
        int left = 0, right = nums.size();
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] < target) left = mid + 1;
            else right = mid;
        }
        if (left == nums.size() || nums[left] != target) return res;
        res[0] = left;
        right = nums.size();
        while (left < right) {
            int mid = left + (right-left) / 2;
            if (nums[mid] <= target) left = mid+1;
            else right = mid;
        }
        res[1] = left-1; //or right-1;
        return res;
    }
};

```

## 35. Search Insert Position

Easy

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.

**Example 1:**

**Input:** [1,3,5,6], 5

**Output:** 2

**Example 2:**

**Input:** [1,3,5,6], 2

**Output:** 1

**Example 3:**

**Input:** [1,3,5,6], 7

**Output:** 4

**Example 4:**

**Input:** [1,3,5,6], 0

**Output:** 0

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {

    }
};
```

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        if (nums.back() < target) return nums.size();
        int left = 0, right = nums.size();
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] < target) left = mid + 1;
            else right = mid;
        }
        return left;
    }
};
```



# 36. Valid Sudoku

## Medium

Determine if a 9x9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

- 1. Each row must contain the digits 1–9 without repetition.
- 2. Each column must contain the digits 1–9 without repetition.
- 3. Each of the 9 3x3 sub-boxes of the grid must contain the digits 1–9 without repetition.

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

A partially filled sudoku which is valid.

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

### Example 1:

#### Input:

```
[
  ["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]
```

```
[[".", ".", ".", ".", ".", "8", ".", ".", "7", "9"]  
]
```

**Output:** true

**Example 2:**

**Input:**

```
[  
  ["8", "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"]  
]
```

**Output:** false

**Explanation:** Same as Example 1, except with the **5** in the top left corner being modified to **8**. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

**Note:**

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.
- The given board contain only digits 1–9 and the character '.'.
- The given board size is always 9x9.

```
class Solution {  
public:  
    bool isValidSudoku(vector<vector<char>>& board) {  
  
    }  
};
```

```

class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        bool used[9];
        for (int i = 0; i < 9; i++){
            fill(used, used+9, false);
            for (int j = 0; j < 9; j++) {
                if (!check(board[i][j], used))
                    return false;
            }
            fill (used, used+9, false);
            for (int j = 0; j < 9; j++) {
                if (!check(board[j][i], used))
                    return false;
            }
        }

        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++) {
                fill(used, used+9, false);
                for (int a = 3*i; a < 3*i+3; a++) {
                    for (int b = 3*j; b < 3*j+3; b++)
                        if (!check(board[a][b], used))
                            return false;
                }
            }
        }
        return true;
    }

private:
    bool check(char a, bool used[]){
        if (a == '.') return true;
        if (used[a-'1']) return false;
        used[a-'1'] = true;
        return true;
    }
};

```

## 37. Sudoku Solver

### Hard

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

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits 1–9 must occur exactly once in each row.
2. Each of the digits 1–9 must occur exactly once in each column.
3. Each of the the digits 1–9 must occur exactly once in each of the 9 3×3 sub-boxes of the grid.

Empty cells are indicated by the character '.'.

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

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

...and its solution numbers marked in red.

**Note:**

- The given board contain only digits 1-9 and the character '.'.
- You may assume that the given Sudoku puzzle will have a single unique solution.
- The given board size is always 9x9.

```

class Solution {
public:
    void solveSudoku(vector<vector<char>> &board) {
        solveSudoku(board, 0, 0);
    }

private:
    bool check(vector<vector<char>> &board, int i, int j, char val) {
        int row = i-i%3, column = j-j%3;
        for (int x = 0; x < 9; x++) {
            if (board[x][j] == val || board[i][x] == val)
                return false;
        }
        for (int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                if (board[row+x][column+y] == val)
                    return false;
            }
        }
        return true;
    }

    bool solveSudoku(vector<vector<char>> &board, int i, int j){
        if (i == 9) return true;
        else if (j == 9) return solveSudoku(board, i+1, 0);
        if (board[i][j] != '.') return solveSudoku(board, i, j+1);
        else for (char c = '1'; c <= '9'; c++){
            if (check(board, i, j, c)) {
                board[i][j] = c;
                if (solveSudoku(board, i, j+1)) return true;
                board[i][j] = '.';
            }
        }
        return false;
    }
};

```

## 38. Count and Say

Easy

The count-and-say sequence is the sequence of integers with the first five terms as following:

1. 1
2. 11
3. 21
4. 1211
5. 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  $n$  where  $1 \leq n \leq 30$ , generate the  $n^{\text{th}}$  term of the count-and-say sequence.

Note: Each term of the sequence of integers will be represented as a string.

**Example 1:**

**Input:** 1

**Output:** "1"

**Example 2:**

**Input:** 4

**Output:** "1211"

```
class Solution {  
public:  
    string countAndSay(int n) {  
  
    }  
};
```

```
class Solution {
public:
    string countAndSay(int n) {
        string s = "1";
        while (--n) {
            string t;
            char c = s[0];
            int cnt = 1;
            for (int i = 1; i < s.length(); i++){
                if (s[i] == c) cnt++;
                else {
                    t += to_string(cnt)+c;
                    cnt = 1;
                    c = s[i];
                }
            }
            s = t + to_string(cnt)+c;
        }
        return s;
    }
};
```

## 39. Combination Sum

### Medium

Given a **set** of candidate numbers (`candidates`) (**without duplicates**) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

The **same** repeated number may be chosen from `candidates` unlimited number of times.

#### Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

#### Example 1:

**Input:** `candidates = [2,3,6,7]`, `target = 7`,

**A solution set is:**

```
[  
  [7],  
  [2,2,3]  
]
```

#### Example 2:

**Input:** `candidates = [2,3,5]`, `target = 8`,

**A solution set is:**

```
[  
  [2,2,2,2],  
  [2,3,3],  
  [3,5]  
]
```

```
class Solution {  
public:  
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {  
  
    }  
};
```



```
class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> combinationSum(vector<int> &candidates, int target) {
        vector<int> t;
        sort(candidates.begin(), candidates.end());
        dfs(candidates, 0, target, t);
        return res;
    }
private:
    void dfs(vector<int> &v, int cur, int target, vector<int> &t) {
        if (target == 0) res.push_back(t);
        else for(auto i = cur; i < v.size() && target-v[i] >= 0; i++) {
            t.push_back(v[i]);
            dfs(v, i, target-v[i], t);
            t.pop_back();
        }
    }
};
```

## 40. Combination Sum II ★★

### Medium

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

Each number in `candidates` may only be used **once** in the combination.

#### Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

#### Example 1:

**Input:** `candidates = [10,1,2,7,6,1,5]`, `target = 8`,

**A solution set is:**

```
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
```

#### Example 2:

**Input:** `candidates = [2,5,2,1,2]`, `target = 5`,

**A solution set is:**

```
[
  [1,2,2],
  [5]
]
```

```
class Solution {
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

    }
};
```

```

class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> combinationSum2(vector<int> &candidates, int target) {
        vector<int> t;
        sort(candidates.begin(), candidates.end());
        dfs(candidates, 0, target, t);
        return res;
    }

private:
    void dfs(vector<int> &v, int cur, int target, vector<int> &t) {
        if (target == 0) res.push_back(t);
        else for(auto i = cur; i < v.size() && target-v[i] >= 0; i++) {
            if (i != cur && v[i] == v[i-1]) continue; //不同之处 1
            t.push_back(v[i]);
            dfs(v, i+1, target-v[i], t); //不同之处 2
            t.pop_back();
        }
    }
};

```

## 41. First Missing Positive ★ ★

Hard

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

**Example 1:**

Input: [1,2,0]

Output: 3

**Example 2:**

Input: [3,4,-1,1]

Output: 2

**Example 3:**

Input: [7,8,9,11,12]

Output: 1

**Note:**

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

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {

    }
};
```

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        int n = nums.size();
        for (int i = 0; i < n; i++) {
            while (nums[i] > 0 && nums[i] <= n
                    && nums[nums[i] - 1] != nums[i]) {
                swap(nums[i], nums[nums[i] - 1]);
            }
        }
        for (int i = 0; i < n; i++) {
            if (nums[i] != i + 1)
                return i + 1;
        }
        return n + 1;
    }
};
```

## 42. Trapping Rain Water★★

Hard

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.



The above elevation map is represented by array `[0,1,0,2,1,0,1,3,2,1,2,1]`. In this case, 6 units of rain water (blue section) are being trapped. **Thanks Marcos** for contributing this image!

**Example:**

**Input:** `[0,1,0,2,1,0,1,3,2,1,2,1]`

**Output:** 6

```
class Solution {
public:
    int trap(vector<int>& height) {

    }
};
```

```
class Solution {
public:
    int trap(vector<int>& height) {
        int res = 0, l = 0, r = height.size()-1;
        while (l < r) {
            int MIN = min(height[l], height[r]);
            if (MIN == height[l]) {
                ++l;
                while (l < r && height[l] < MIN) res += MIN-height[l++];
            } else {
                --r;
                while (l < r && height[r] < MIN) res += MIN-height[r--];
            }
        }
        return res;
    }
};
```

## 43. Multiply Strings

### Medium

Given two non-negative integers `num1` and `num2` represented as strings, return the product of `num1` and `num2`, also represented as a string.

#### Example 1:

**Input:** `num1 = "2", num2 = "3"`

**Output:** `"6"`

#### Example 2:

**Input:** `num1 = "123", num2 = "456"`

**Output:** `"56088"`

#### Note:

1. The length of both `num1` and `num2` is  $< 110$ .
2. Both `num1` and `num2` contain only digits `0-9`.
3. Both `num1` and `num2` do not contain any leading zero, except the number `0` itself.
4. You **must not use any built-in BigInteger library** or **convert the inputs to integer** directly.

```
class Solution {
public:
    string multiply(string num1, string num2) {

    }
};
```



```

class Solution {
public:
    string multiply(string num1, string num2) {
        if (num1 == "0" || num2 == "0") return "0";
        int n = num1.size(), m = num2.size();
        vector<int> sum(n+m);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                sum[n+m-i-j-2] += (num1[i]-'0')*(num2[j]-'0');
            }
        }
        int carry = 0;
        string res;
        for (int i = 0; i < n+m-1; i++) {
            carry += sum[i];
            res += carry%10+'0';
            carry /= 10;
        }
        if (carry) res += carry+'0';
        reverse(res.begin(), res.end());
        return res;
    }
};

```

## 44. Wildcard Matching ★★

### Hard

Given an input string (*s*) and a pattern (*p*), 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).

### Note:

- s* could be empty and contains only lowercase letters *a-z*.
- p* could be empty and contains only lowercase letters *a-z*, and characters like ? or \*.

### Example 1:

#### Input:

*s* = "aa"

*p* = "a"

**Output:** false

**Explanation:** "a" does not match the entire string "aa".

### Example 2:

#### Input:

*s* = "aa"

*p* = "\*"

**Output:** true

**Explanation:** '\*' matches any sequence.

### Example 3:

#### Input:

*s* = "cb"

*p* = "?a"

**Output:** false

**Explanation:** '?' matches 'c', but the second letter is 'a', which does not match 'b'.

**Example 4:**

**Input:**

s = "adceb"

p = "\*a\*b"

**Output:** true

**Explanation:** The first '\*' matches the empty sequence, while the second '\*' matches the substring "dce".

**Example 5:**

**Input:**

s = "acdeb"

p = "a\*c?b"

**Output:** false

```
class Solution {
public:
    bool isMatch(string s, string p) {

    }
}
```

```

class Solution {
public:
    bool isMatch(string s, string p) {
        int slen = s.size(), plen = p.size();
        int i = 0, j = 0, iStar = -1, jStar = -1;
        while (i < slen) {
            cout << i << " " << j << "\n";
            if (j < plen && p[j] == '*') {
                //meet a new '*', update traceback i/j info
                iStar = i;
                jStar = j++;
            }
            else if (j < plen && (p[j] == s[i] || p[j] == '?')) {
                ++i, ++j;
            }
            else {
                // mismatch happens
                if (iStar < 0) return false;
                // met a '*' before, then do traceback
                i = iStar++;
                j = jStar + 1;
            }
        }
        while (j < plen && p[j] == '*') ++j;
        return j == plen;
    }
};

```

## 45. Jump Game II

### Hard

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.

### Example:

**Input:** [2,3,1,1,4]

**Output:** 2

**Explanation:** 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.

```
class Solution {  
public:  
    int jump(vector<int>& nums) {  
  
    }  
};
```

```

class Solution {
public:
    int jump(vector<int>& nums) {
        if (nums.size() <= 1) return 0;
        int res = 1, n = nums.size(), i = 0;
        while (i < n) {
            int farest = i+nums[i], temp = farest;
            if (fares >= n-1) return res;
            else for (int k = i+1; k <= farest; k++) {
                if (k+nums[k] > temp) temp = k + nums[i = k];
            }
            res++;
        }
        return res;
    }
};

```

## 46. Permutations

Medium

Given a collection of **distinct** integers, return all possible permutations.

**Example:**

**Input:** [1,2,3]

**Output:**

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

```
class Solution {  
public:  
    vector<vector<int>> permute(vector<int>& nums) {  
  
        }  
};
```

```

class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int> >res;

        do{
            res.push_back(nums);
        } while(Next_Permutation(nums));
        return res;
    }

    bool Next_Permutation(vector<int> &nums) {
        int n = nums.size(), i, j;
        for (i = n-2, j = n-1; i >= 0; i--) {
            if (nums[i] < nums[i+1]) {
                while (nums[j] < nums[i]) j--;
                swap(nums[i], nums[j]);
                int l = i+1, r = n-1;
                while(l < r) swap(nums[l++], nums[r--]);
                return true;
            }
        }
        return false;
    }
};

```



//DFS//

```
class Solution {
public:
    vector<vector<int>> > permute(vector<int> &num) {
        vector<vector<int>>> res;
        dfs(num, 0, res);
        return res;
    }

private:
    void dfs(vector<int> &num, int cur, vector<vector<int>>> &res) {
        if (cur >= num.size()) {
            res.push_back(num);
            return;
        }
        for (int i = cur; i < num.size(); i++) {
            swap(num[cur], num[i]);
            dfs(num, cur + 1, res);
            swap(num[cur], num[i]);
        }
    }
};
```

## 47. Permutations II ★ ★

Medium

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

**Example:**

**Input:** [1,1,2]

**Output:**

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

```
class Solution {  
public:  
    vector<vector<int>> permuteUnique(vector<int>& nums) {  
  
    }  
};
```

```

class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int> >res;

        do{
            res.push_back(nums);
        } while(Next_Permutation(nums));
        return res;
    }

    bool Next_Permutation(vector<int> &nums) {
        int n = nums.size(), i, j;
        for (i = n-2, j = n-1; i >= 0; i--) {
            if (nums[i] < nums[i+1]) {
                while (nums[j] <= nums[i]) j--;
                swap(nums[i], nums[j]);
                int l = i+1, r = n-1;
                while(l < r) swap(nums[l++], nums[r--]);
                return true;
            }
        }
        return false;
    }
};

```

```

/////////////////////////////////DFS/////////////////////////////////
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& num) {
        vector<vector<int>> res;
        dfs(num, 0, res);
        return res;
    }

private:
    void dfs(vector<int> &num, int cur, vector<vector<int>> &res) {
        if (cur >= num.size()) {
            res.push_back(num);
            return;
        }
        unordered_set<int> st;
        for (int i = cur; i < num.size(); i++) {
            if (st.count(num[i])) continue;
            st.insert(num[i]);
            swap(num[cur], num[i]);
            dfs(num, cur + 1, res);
            swap(num[cur], num[i]);
        }
    }
};

```

```
//////////注意 nums 为值传递//////////
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int> &nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> res;
        permute(nums, 0, res);
        return res;
    }
    void permute(vector<int> nums, int i, vector<vector<int>> &res) {
        if (i == nums.size()) res.push_back(nums);
        else for (int k = i; k < nums.size(); k++) {
            if (i != k && nums[i] == nums[k]) continue;
            swap(nums[i], nums[k]);
            permute(nums, i+1, res);
        }
    }
};
```

## 48. Rotate Image

### Medium

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

Rotate the image by 90 degrees (clockwise).

#### Note:

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly.

**DO NOT** allocate another 2D matrix and do the rotation.

#### Example 1:

Given **input matrix** =

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

rotate the input matrix **in-place** such that it becomes:

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

#### Example 2:

Given **input matrix** =

```
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
```

```
[15,14,12,16]  
],
```

rotate the input matrix **in-place** such that it becomes:

```
[  
  [15,13, 2, 5],  
  [14, 3, 4, 1],  
  [12, 6, 8, 9],  
  [16, 7,10,11]  
]
```

```
class Solution {  
public:  
    void rotate(vector<vector<int>>& matrix) {  
  
        }  
};
```

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n = matrix.size();
        int i = 0, j = n-1;
        while (i < j){
            for (int k = 0; k < n; k++) swap(matrix[k][i], matrix[k][j]);
            i++; j--;
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j + i < n-1; j++)
                swap(matrix[i][j],matrix[n-1-j][n-1-i]);
        }
    }
};
```



## 49. Group Anagrams

### Medium

Given an array of strings, group anagrams together.

#### Example:

**Input:** ["eat", "tea", "tan", "ate", "nat", "bat"],

#### Output:

```
[  
  ["ate","eat","tea"],  
  ["nat","tan"],  
  ["bat"]  
]
```

#### Note:

- All inputs will be in lowercase.
- The order of your output does not matter.

```
class Solution {  
public:  
    vector<vector<string>> groupAnagrams(vector<string>& strs) {  
  
};
```

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> m;
        for (string s : strs) {
            string t = s;
            sort(t.begin(), t.end());
            m[t].push_back(s);
        }
        vector<vector<string>> res;
        for (auto &p : m) {
            res.push_back(p.second);
        }
        return res;
    }
};
```

## 50. Pow(x, n) ★★

Medium

Implement `pow(x, n)`, which calculates  $x$  raised to the power  $n$  ( $x^n$ ).

**Example 1:**

**Input:** 2.00000, 10

**Output:** 1024.00000

**Example 2:**

**Input:** 2.10000, 3

**Output:** 9.26100

**Example 3:**

**Input:** 2.00000, -2

**Output:** 0.25000

**Explanation:**  $2^{-2} = 1/2^2 = 1/4 = 0.25$

**Note:**

- $-100.0 < x < 100.0$
- $n$  is a 32-bit signed integer, within the range  $[-2^{31}, 2^{31} - 1]$

```
class Solution {
public:
    double myPow(double x, int n) {

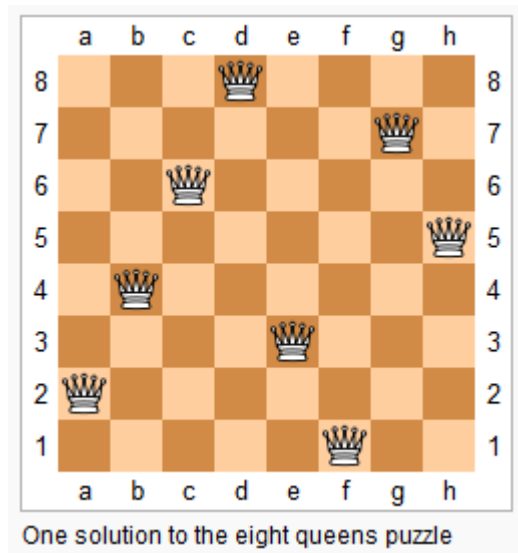
    }
};
```

```
class Solution {
public:
    double myPow(double x, int n) {
        if (x == 1 || n == 0) return 1;
        if (n < 0) x = 1/x;
        double res = 1;
        unsigned long long m = abs(n);
        while (m) {
            if (m&1) res *= x;
            x *= x;
            m >>= 1;
        }
        return res;
    }
};
```

## 51. N-Queens

Hard

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.



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.

**Example:**

**Input:** 4

**Output:** [

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

"...Q",

"Q...",

"..Q."],

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

"Q...",

"...Q",

".Q.."]

```
]
```

**Explanation:** There exist two distinct solutions to the 4-queens puzzle as shown above.

```
class Solution {  
public:  
    vector<vector<string>> solveNQueens(int n) {  
  
    }  
};
```

```

class Solution {
public:
    vector<vector<string>> res;
    vector<vector<string>> solveNQueens(int n) {
        col.resize(n, false);
        main_diag.resize(2*n-1, false);
        anti_diag.resize(2*n-1, false);
        vector<string> v;
        dfs(0, n, v);
        return res;
    }

private:
    vector<bool> col;
    vector<bool> main_diag;
    vector<bool> anti_diag;

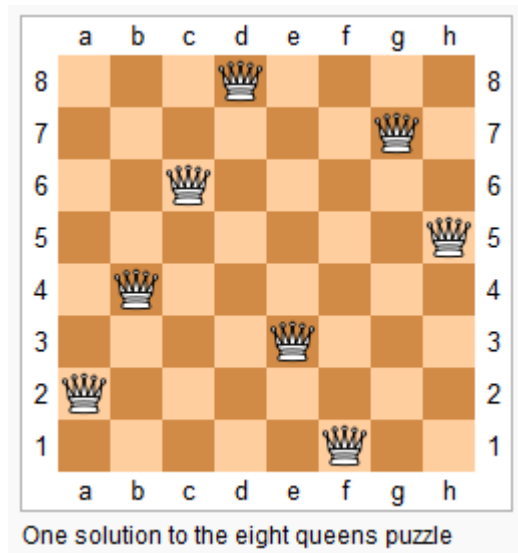
    void dfs(int x, int n, vector<string> &v) {
        if (x == n) res.push_back(v);
        else for (int y = 0; y < n; y++) {
            if (!col[y] && !main_diag[x+y] && !anti_diag[x-y+n-1]){
                col[y] = main_diag[x+y] = anti_diag[x-y+n-1] = true;
                v.push_back( string(y, '.') + 'Q' + string(n-y-1, '.') );
                dfs(x+1, n, v);
                v.pop_back();
                col[y] = main_diag[x+y] = anti_diag[x-y+n-1] = false;
            }
        }
    }
};

```

## 52. N-Queens II

Hard

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.



Given an integer  $n$ , return the number of distinct solutions to the  $n$ -queens puzzle.

**Example:**

**Input:** 4

**Output:** 2

**Explanation:** There are two distinct solutions to the 4-queens puzzle as shown below.

[

["Q..", // Solution 1

"...Q",

"Q...",

"..Q."],

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

"Q...",

"...Q",



```
"Q."]  
]
```

```
class Solution {  
public:  
    int totalNQueens(int n) {  
  
    }  
};
```

```

class Solution {
public:
    int totalNQueens(int n) {
        col.resize(n, false);
        main_diag.resize(2*n-1, false);
        anti_diag.resize(2*n-1, false);
        return dfs(0, n);
    }

private:
    vector<bool> col;
    vector<bool> main_diag;
    vector<bool> anti_diag;

    int dfs(int x, int n) {
        if (x == n) return 1;
        int res = 0;
        for (int y = 0; y < n; y++) {
            if (!col[y] && !main_diag[x+y] && !anti_diag[x-y+n-1]){
                col[y] = main_diag[x+y] = anti_diag[x-y+n-1] = true;
                res += dfs(x+1, n);
                col[y] = main_diag[x+y] = anti_diag[x-y+n-1] = false;
            }
        }
        return res;
    }
};

```

## 53. Maximum Subarray

Easy

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

**Example:**

**Input:** [-2,1,-3,4,-1,2,1,-5,4],

**Output:** 6

**Explanation:** [4,-1,2,1] has the largest sum = 6.

**Follow up:**

If you have figured out the  $O(n)$  solution, try coding another solution using the divide and conquer approach, which is more subtle.

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum = nums[0], cur = 0;
        for (auto &i : nums) {
            cur += i;
            sum = max(sum, cur);
            if (cur <= 0) cur = 0;
        }
        return sum;
    }
};
```

```

//////////divide and conquer approach//////////
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        return maxSubArray(nums, 0, nums.size()-1);
    }

private:
    int maxSubArray(vector<int>& nums, int left, int right) {
        if (right < left) return INT_MIN;
        else if (right == left) return nums[left];
        int mid = left+(right-left)/2;
        int ret = max(maxSubArray(nums, left, mid),
                      maxSubArray(nums, mid+1, right));
        int sum = nums[mid], temp = 0;
        for (int i = mid+1, t = 0; i <= right; i++) {
            temp = max(t += nums[i], temp);
        }
        sum += temp;
        temp = 0;
        for (int i = mid-1, t = 0; i >= left; i--) {
            temp = max(t += nums[i], temp);
        }
        return max(ret, sum += temp);
    }
};

```

## 54. Spiral Matrix

### Medium

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

#### Example 1:

**Input:**

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

**Output:** [1,2,3,6,9,8,7,4,5]

#### Example 2:

**Input:**

```
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
```

**Output:** [1,2,3,4,8,12,11,10,9,5,6,7]

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {

    }
};
```

```

class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> res;
        if (matrix.empty()) return res;
        int rowSize = matrix.size(), colSize = matrix[0].size();
        int lb = 0, rb = colSize-1, ub = 0, db = rowSize-1;
        // left bound, right bound, up bound, down bound;
        int direction = 0, col = 0, row = 0;
        while (lb <= rb && ub <= db) {
            res.push_back(matrix[row][col]);
            switch (direction){
                case 0: // right
                    if (col >= rb) {direction = 1; row++; ub++;}
                    else col++;
                    break;
                case 1: // down
                    if (row >= db) {direction = 2; col--; rb--;}
                    else row++;
                    break;
                case 2: // left
                    if (col <= lb) {direction = 3; row--; db--;}
                    else col--;
                    break;
                case 3: // up
                    if (row <= ub) {direction = 0; col++; lb++;}
                    else row--;
            }
        }
        return res;
    }
};

```

## 55. Jump Game

### Medium

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.

#### Example 1:

**Input:** [2,3,1,1,4]

**Output:** true

**Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

#### Example 2:

**Input:** [3,2,1,0,4]

**Output:** false

**Explanation:** You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

```
class Solution {
public:
    bool canJump(vector<int>& nums) {

    }
};
```

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        if (nums.size() <= 1) return true;
        int n = nums.size(), temp = n-1;
        for (int i = n-1; i >= 0; --i) {
            int j = i + nums[i];
            if (j >= temp) temp = i;
        }
        return temp == 0;
    }
};
```



## 56. Merge Intervals★★

Medium

Given a collection of intervals, merge all overlapping intervals.

**Example 1:**

**Input:** [[1,3],[2,6],[8,10],[15,18]]

**Output:** [[1,6],[8,10],[15,18]]

**Explanation:** Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].

**Example 2:**

**Input:** [[1,4],[4,5]]

**Output:** [[1,5]]

**Explanation:** Intervals [1,4] and [4,5] are considered overlapping.

```
class Solution {
public:
    vector<vector<int>>> merge(vector<vector<int>>& intervals) {

    }
};
```

```

class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        vector<vector<int>> res;
        if (intervals.empty()) return res;
        auto cmp = [](const vector<int> &a, const vector<int> &b){
            return a[0] < b[0];
        };
        sort(intervals.begin(), intervals.end(), cmp);
        res.push_back(intervals[0]);
        for (auto &v : intervals) {
            if (res.back()[1] < v[0]) res.push_back(v);
            else res.back()[1] = max(res.back()[1], v[1]);
        }
        return res;
    }
};

```

## 57. Insert Interval★★

### Hard

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:

**Input:** intervals = [[1,3],[6,9]], newInterval = [2,5]

**Output:** [[1,5],[6,9]]

#### Example 2:

**Input:** intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

**Output:** [[1,2],[3,10],[12,16]]

**Explanation:** Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

```
class Solution {
public:
    vector<vector<int>>> insert(vector<vector<int>>>& ins, vector<int>& newIn) {
        auto it = ins.begin();
        while (it != ins.end()) {
            if ((*it)[1] < newIn[0]) it++;
            else if ((*it)[0] > newIn[1]) {
                ins.insert(it, newIn);
                return ins;
            } else {
                newIn[0] = min(newIn[0], (*it)[0]);
                newIn[1] = max(newIn[1], (*it)[1]);
                it = ins.erase(it);
            }
        }
        ins.insert(ins.end(), newIn);
        return ins;
    }
};
```

```

class Solution {
public:
    vector<vector<int>>> insert(vector<vector<int>>& ins, vector<int>& newIn){
        auto cmp = [] (const vector<int> &lhs, const vector<int> &rhs) {
            return lhs[1] < rhs[0];
        };
        auto range = equal_range(ins.begin(), ins.end(), newIn, cmp);
        auto p = range.first, q = range.second;
        if (p == q) ins.insert(p, newIn);
        else {
            q--;
            (*q)[0] = min(newIn[0], (*p)[0]);
            (*q)[1] = max(newIn[1], (*p)[1]);
            ins.erase(p, q);
        }
        return ins;
    }
};

```

## 58. Length of Last Word

Easy

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.

**Example:**

**Input:** "Hello World"

**Output:** 5

```
class Solution {  
public:  
    int lengthOfLastWord(string s) {  
  
    }  
};
```

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        stringstream ss(s);
        string res;
        while (ss >> res);
        return res.length();
    }
};
```

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int i = s.find_last_not_of(' ');
        int res = 0;
        while (i >= 0 && s[i--] != ' ') res++;
        return res;
    }
};
```

## 59. Spiral Matrix II

Medium

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

**Example:**

**Input:** 3

**Output:**

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

```
class Solution {  
public:  
    vector<vector<int>> generateMatrix(int n) {  
  
        }  
};
```

```

class Solution {
public:
    vector<vector<int>> generateMatrix(int n) {
        vector<vector<int>> res(n, vector<int>(n, 0));
        int x = 0, y = 0, cnt = 0;
        res[x][y] = ++cnt;
        while (cnt < n*n) {
            while (y+1 < n && !res[x][y+1]) res[x][++y] = ++cnt;
            while (x+1 < n && !res[x+1][y]) res[++x][y] = ++cnt;
            while (y-1 >= 0 && !res[x][y-1]) res[x][--y] = ++cnt;
            while (x-1 >= 0 && !res[x-1][y]) res[--x][y] = ++cnt;
        }
        return res;
    }
};

```



## 60. Permutation Sequence

### Medium

The set  $[1, 2, 3, \dots, n]$  contains a total of  $n!$  unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for  $n = 3$ :

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

Given  $n$  and  $k$ , return the  $k^{\text{th}}$  permutation sequence.

#### Note:

- Given  $n$  will be between 1 and 9 inclusive.
- Given  $k$  will be between 1 and  $n!$  inclusive.

#### Example 1:

**Input:**  $n = 3, k = 3$

**Output:** "213"

#### Example 2:

**Input:**  $n = 4, k = 9$

**Output:** "2314"

```
class Solution {
public:
    string getPermutation(int n, int k) {
    }
};
```

```

class Solution {
public:
    string getPermutation(int n, int k) {
        string s(n, '0');
        for (int i = 0; i < n; ++i) s[i] += i + 1;
        // 康托编码从 0 开始
        return kth_permutation(s, n, k-1);
    }

private:
    int factorial(int n) {
        int res = 1;
        while (n) res *= n--;
        return res;
    }

    string kth_permutation(string &s, int n, int k) {
        string res;
        int base = factorial(n);
        for (int i = n; i > 0; i--) {
            base /= i;
            auto a = next(s.begin(), k / base);
            res.push_back(*a);
            s.erase(a);
            k %= base;
        }
        return res;
    }
};

```

## 61. Rotate List

### Medium

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

#### Example 1:

**Input:** 1->2->3->4->5->NULL,  $k = 2$

**Output:** 4->5->1->2->3->NULL

#### Explanation:

rotate 1 steps to the right: 5->1->2->3->4->NULL

rotate 2 steps to the right: 4->5->1->2->3->NULL

#### Example 2:

**Input:** 0->1->2->NULL,  $k = 4$

**Output:** 2->0->1->NULL

#### Explanation:

rotate 1 steps to the right: 2->0->1->NULL

rotate 2 steps to the right: 1->2->0->NULL

rotate 3 steps to the right: 0->1->2->NULL

rotate 4 steps to the right: 2->0->1->NULL

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {

    }
};
```

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (head == nullptr) return head;
        int len = 1;
        ListNode *last = head, *p = head;
        while (last->next) {
            last = last->next;
            len++;
        }
        k = len - k % len;
        while (--k) p = p->next;
        last->next = head;
        head = p->next;
        p->next = nullptr;
        return head;
    }
};
```

## 62. Unique Paths

### Medium

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?



Above is a 7 x 3 grid. How many possible unique paths are there?

**Note:**  $m$  and  $n$  will be at most 100.

#### Example 1:

**Input:**  $m = 3, n = 2$

**Output:** 3

#### Explanation:

From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right

#### Example 2:

**Input:**  $m = 7, n = 3$

**Output:** 28

```

class Solution {
public:
    int uniquePaths(int m, int n) {
        if (n > m) swap(m, n);
        long long res = 1;
        int i = 1;
        while(i < n) res = res*(m++)/i++;
        return res;
    }
};

```

```

class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<int> f(n, 1);
        while (--m) {
            for (int j = n-1; j >= 0; j--) {
                if (j != n-1) f[j] += f[j+1];
            }
        }
        return f[0];
    }
};

```

## 63. Unique Paths II

### Medium

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).

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.

**Note:**  $m$  and  $n$  will be at most 100.

### Example 1:

#### Input:

```
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
```

#### Output: 2

#### Explanation:

There is one obstacle in the middle of the 3x3 grid above.

There are two ways to reach the bottom-right corner:

1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right

```

class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        int m = obstacleGrid.size(), n = obstacleGrid[0].size();
        if (obstacleGrid[0][0] || obstacleGrid[m-1][n-1]) return 0;
        vector<long long> f(n, 0);
        f[n-1] = 1;
        for (int i = m-1; i >= 0; i--){
            for (int j = n-1; j >= 0; j--) {
                if (i == m-1 && j == n-1) continue;
                else if (obstacleGrid[i][j]) f[j] = 0;
                else if (j != n-1) f[j] += f[j+1];
            }
        }
        return f[0];
    }
};

```



## 64. Minimum Path Sum

### Medium

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.

**Example:**

**Input:**

```
[  
  [1,3,1],  
  [1,5,1],  
  [4,2,1]  
]
```

**Output:** 7

**Explanation:** Because the path  $1 \rightarrow 3 \rightarrow 1 \rightarrow 1 \rightarrow 1$  minimizes the sum.

```
class Solution {  
public:  
    int minPathSum(vector<vector<int>>& grid) {  
  
    }  
};
```

```

class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        int res = 0;
        if (grid.empty()) return res;
        int n = grid.size(), m = grid[0].size();
        vector<int> f = grid[0];
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (i == 0) f[j] += (j == 0 ? 0: f[j-1]);
                else if (j == 0) f[j] += grid[i][j];
                else f[j] = min(f[j], f[j-1]) + grid[i][j];
            }
        }
        return f[m-1];
    }
};

```

## 65. Valid Number★★

### Hard

Validate if a given string can be interpreted as a decimal number.

Some examples:

```
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
" -90e3 " => true
" 1e" => false
"e3" => false
" 6e-1" => true
" 99e2.5 " => false
"53.5e93" => true
" --6 " => false
"-+3" => false
"95a54e53" => false
```

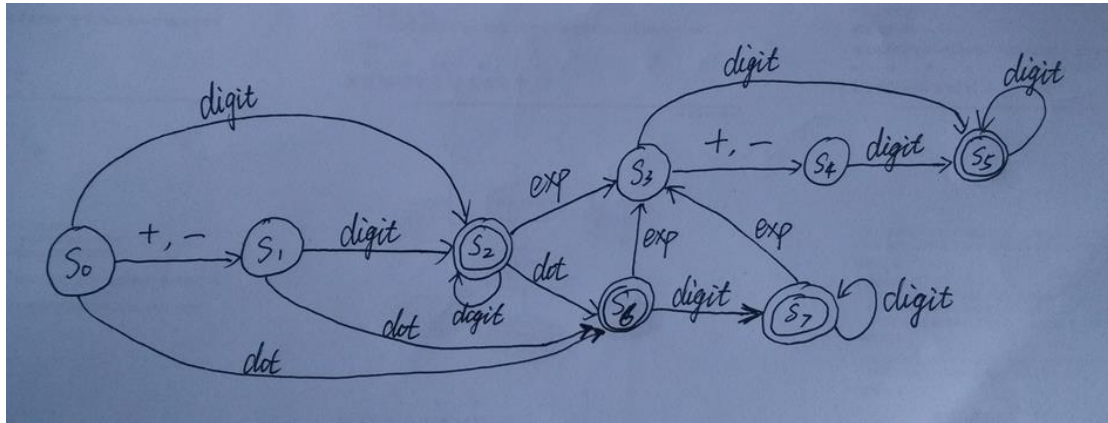
**Note:** It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one. However, here is a list of characters that can be in a valid decimal number:

- Numbers 0-9
- Exponent - "e"
- Positive/negative sign - "+"/"-"
- Decimal point - "."

Of course, the context of these characters also matters in the input.

### Update (2015-02-10):

The signature of the C++ function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button to reset your code definition.



```

class Solution {
public:
    bool isNumber(string str) {
        int state = 0, flag = 0;
        while (str[0] == ' ') str.erase(0,1);
        while (str.back() == ' ') str.erase(str.length()-1);
        for (auto &c : str){
            if (isdigit(c)) {
                flag = 1;
                if (state <= 2) state = 2;
                else state = (state <= 5) ? 5 : 7;
            }
            else if ('+' == c || '-' == c){
                if (state == 0 || state == 3) state++;
                else return false;
            }
            else if ('.' == c){
                if (state <= 2) state = 6;
                else return false;
            }
            else if ('e' == c){
                if(flag && (state == 2 || state == 6 || state == 7)) state = 3;
                else return false;
            }
            else return false;
        }
        return state == 2 || state == 5 || (flag && state == 6) || state == 7;
    }
};

```

## 66. Plus One

Easy

Given a **non-empty** array of digits representing a non-negative integer, plus one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

**Example 1:**

**Input:** [1,2,3]

**Output:** [1,2,4]

**Explanation:** The array represents the integer 123.

**Example 2:**

**Input:** [4,3,2,1]

**Output:** [4,3,2,2]

**Explanation:** The array represents the integer 4321.

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {

    }
};
```

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int c = 1;
        for(auto i = digits.rbegin(); i != digits.rend(); i++){
            c += *i;
            *i = c % 10;
            c /= 10;
        }
        if (c) digits.insert(digits.begin(), c);
        return digits;
    }
};
```

## 67. Add Binary

Easy

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

The input strings are both **non-empty** and contains only characters `1` or `0`.

**Example 1:**

**Input:** a = "11", b = "1"

**Output:** "100"

**Example 2:**

**Input:** a = "1010", b = "1011"

**Output:** "10101"

```
class Solution {  
public:  
    string addBinary(string a, string b) {  
  
    }  
};
```

```

class Solution {
public:
    string addBinary(string a, string b) {
        int c = 0;
        string res;
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        auto p = a.begin(), q = b.begin();
        while (p != a.end() || q != b.end()) {
            if (p != a.end()) {
                c += *p - '0';
                p++;
            }
            if (q != b.end()) {
                c += *q - '0';
                q++;
            }
            res += c % 2 + '0';
            c /= 2;
        }
        if (c) res += '1';
        reverse(res.begin(), res.end());
        return res;
    }
};

```



## 68. Text Justification

### Hard

Given an array of words and a width *maxWidth*, format the text such that each line has exactly *maxWidth* 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 *maxWidth* 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.

### Note:

- A word is defined as a character sequence consisting of non-space characters only.
- Each word's length is guaranteed to be greater than 0 and not exceed *maxWidth*.
- The input array `words` contains at least one word.

### Example 1:

#### Input:

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

```
maxWidth = 16
```

#### Output:

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

### Example 2:

#### Input:

```
words = ["What", "must", "be", "acknowledgment", "shall", "be"]
```

```
maxWidth = 16
```

#### Output:

```
[
    "What    must   be",
    "acknowledgment ",
    "shall be       "
]
```

**Explanation:** Note that the last line is "shall be " instead of "shall be",  
because the last line must be left-justified instead of fully-justified.

Note that the second line is also left-justified because it contains only one word.

### Example 3:

#### Input:

```
words = ["Science","is","what","we","understand","well","enough","to","explain",
        "to","a","computer.","Art","is","everything","else","we","do"]
maxWidth = 20
```

#### Output:

```
[
    "Science  is  what we",
    "understand    well",
    "enough to explain to",
    "a  computer. Art is",
    "everything  else  we",
    "do                "
]
```

```
class Solution {
public:
    vector<string> fullJustify(vector<string>& words, int maxWidth) {
        }
};
```

```

class Solution {
public:
    vector<string> fullJustify(vector<string>& words, int maxWidth) {
        vector<string> res;
        int n = words.size(), i = 0;
        while (i < n) {
            int width = 0;
            vector<string> t;
            while (i < n && width + words[i].length()
                    <= maxWidth - t.size()) {
                width += words[i].length();
                t.push_back(words[i++]);
            }
            string s = t[0];
            int space_num = maxWidth - width;
            if (t.size() == 1)    s += string(space_num, ' ');
            else if (i == n) {
                for (int j = 1; j < t.size(); ++j) {
                    s += " " + t[j];
                }
                s += string(maxWidth - s.length(), ' ');
            }
            else {
                int aver = space_num / (t.size() - 1);
                int k = space_num - (t.size() - 1) * aver;
                for (int j = 1; j < t.size(); ++j) {
                    s += string((j <= k ? aver + 1 : aver), ' ') + t[j];
                }
            }
            res.push_back(s);
        }
        return res;
    }
};

```

## 69. Sqrt(x) ★★

Easy

Implement `int sqrt(int x)`.

Compute and return the square root of  $x$ , where  $x$  is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

**Example 1:**

**Input:** 4

**Output:** 2

**Example 2:**

**Input:** 8

**Output:** 2

**Explanation:** The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

```
class Solution {
public:
    int mySqrt(int x) {
        int low = 0, high = x;
        if (x < 2) return x;
        while (low < high) {
            int mid = low + (high-low)/2;
            if (x/mid >= mid) low = mid+1;
            else high = mid;
        }
        return low-1;
    }
};
```

## 70. Climbing Stairs

Easy

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?

**Note:** Given  $n$  will be a positive integer.

**Example 1:**

**Input:** 2

**Output:** 2

**Explanation:** There are two ways to climb to the top.

1. 1 step + 1 step
2. 2 steps

**Example 2:**

**Input:** 3

**Output:** 3

**Explanation:** There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

```
class Solution {
public:
    int climbStairs(int n) {

    }
};
```

```

class Solution {
public:
    struct Matrix{
        int a[2][2] = {0};
        void init() {
            for (int i = 0; i < 2; i++)
                a[i][i] = 1;
        }
    };

    Matrix MUL(Matrix A, Matrix B){
        Matrix C;
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int k = 0; k < 2; k++) {
                    C.a[i][j] += A.a[i][k]*B.a[k][j];
                }
            }
        }
        return C;
    }

    int climbStairs(int n) {
        if (n == 1) return 1;
        if (n == 2) return 2;
        Matrix res, x;
        x.a[0][0] = x.a[0][1] = x.a[1][0] = 1;
        x.a[1][1] = 0;
        res.init();
        while (n) {
            if (n&1) res = MUL(res, x);
            x = MUL(x, x);
            n >>= 1;
        }
        return res.a[0][1]+res.a[1][1];
    }
};

```

## 71. Simplify Path

### Medium

Given an **absolute path** for a file (Unix-style), simplify it. Or in other words, convert it to the **canonical path**.

In a UNIX-style file system, a period `.` refers to the current directory. Furthermore, a double period `..` moves the directory up a level. For more information, see: [Absolute path vs relative path in Linux/Unix](#)

Note that the returned canonical path must always begin with a slash `/`, and there must be only a single slash `/` between two directory names. The last directory name (if it exists) **must not** end with a trailing `/`. Also, the canonical path must be the **shortest** string representing the absolute path.

#### Example 1:

**Input:** `"/home/"`

**Output:** `"/home"`

**Explanation:** Note that there is no trailing slash after the last directory name.

#### Example 2:

**Input:** `"/../"`

**Output:** `"/"`

**Explanation:** Going one level up from the root directory is a no-op, as the root level is the highest level you can go.

#### Example 3:

**Input:** `"/home//foo/"`

**Output:** `"/home/foo"`

**Explanation:** In the canonical path, multiple consecutive slashes are replaced by a single one.

#### Example 4:

**Input:** `"/a/./b/../../c/"`

**Output:** `"/c"`

#### Example 5:

**Input:** `"/a/../../b/../c//.//"`

**Output:** `"/c"`

**Example 6:**

**Input:** `"/a/b///c/d/././.."`

**Output:** `"/a/b/c"`

```
class Solution {  
public:  
    string simplifyPath(string path) {  
  
    }  
};
```



```

class Solution {
public:
    string simplifyPath(string path) {
        string res, tmp;
        vector<string> stk;
        stringstream ss(path);
        while (getline(ss, tmp, '/')) {
            //getline(istream &is, string &str, char delim); delim 终止符
            if (tmp == "" || tmp == ".") continue;
            else if (tmp == ".." && !stk.empty()) stk.pop_back();
            else if (tmp != "..") stk.push_back(tmp);
        }
        for (auto &str : stk) res += '/' + str;
        return res.empty() ? "/" : res;
    }
};

```

## 72. Edit Distance ★ ★

### Hard

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

### Example 1:

**Input:** word1 = "horse", word2 = "ros"

**Output:** 3

#### Explanation:

horse -> rorse (replace 'h' with 'r')

rorse -> rose (remove 'r')

rose -> ros (remove 'e')

### Example 2:

**Input:** word1 = "intention", word2 = "execution"

**Output:** 5

#### Explanation:

intention -> inention (remove 't')

inention -> enention (replace 'i' with 'e')

enention -> exention (replace 'n' with 'x')

exention -> exection (replace 'n' with 'c')

exection -> execution (insert 'u')

```

class Solution {
public:
    int minDistance(string word1, string word2) {
        int m = word1.size(), n = word2.size();
        vector<vector<int>> dp(2, vector<int>(n+1));
        int k = 0;
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                if (i == 0) dp[k][j] = j;
                else if (j == 0) dp[k][j] = i;
                else if (word1[i-1] == word2[j-1]) dp[k][j] = dp[k^1][j-1];
                else dp[k][j] = 1 + min(dp[k^1][j-1],
                                         min(dp[k][j-1], dp[k^1][j]));
            }
            k ^= 1;
        }
        return dp[k^1][n];
    }
};

```

## 73. Set Matrix Zeroes ★ ★

Medium

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

**Example 1:**

**Input:**

```
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
```

**Output:**

```
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
```

**Example 2:**

**Input:**

```
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
```

**Output:**

```
[
  [0,0,0,0],
  [0,4,5,0],
  [0,4,5,0]
]
```

```
[0,3,1,0]  
]
```

**Follow up:**

- 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?

```
class Solution {  
public:  
    void setZeroes(vector<vector<int>>& matrix) {  
  
    }  
};
```

```

class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        if (matrix.empty()) return;
        bool Row_Zero = false, Col_Zero = false;
        int N = matrix.size(), M = matrix[0].size();
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                if (!i && !matrix[i][j]) Row_Zero = true;
                if (!j && !matrix[i][j]) Col_Zero = true;
                if (!matrix[i][j]) matrix[i][0] = matrix[0][j] = 0;
            }
        }

        for (int i = 1; i < N; i++) {
            for (int j = 1; j < M; j++) {
                if (!matrix[i][0] || !matrix[0][j])
                    matrix[i][j] = 0;
            }
        }

        if (Row_Zero){
            for (int j = 0; j < M; j++) matrix[0][j] = 0;
        }
        if (Col_Zero){
            for (int i = 0; i < N; i++) matrix[i][0] = 0;
        }
    }
};

```

## 74. Search a 2D Matrix

### Medium

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.

#### Example 1:

##### Input:

```
matrix = [  
    [1, 3, 5, 7],  
    [10, 11, 16, 20],  
    [23, 30, 34, 50]  
]  
target = 3
```

**Output:** true

#### Example 2:

##### Input:

```
matrix = [  
    [1, 3, 5, 7],  
    [10, 11, 16, 20],  
    [23, 30, 34, 50]  
]  
target = 13
```

**Output:** false

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty()) return false;
        int n = matrix.size(), m = matrix[0].size();
        int l = 0, r = m*n;
        while (l < r) {
            int mid = l + (r-l)/2;
            int v = matrix[mid/m][mid%m];
            if (v == target) return true;
            else if (v < target) l = mid+1;
            else r = mid;
        }
        return false;
    }
};
```



## 75. Sort Colors★★★

### Medium

Given an array with  $n$  objects colored red, white or blue, sort them in-place 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.

### Example:

**Input:** [2,0,2,1,1,0]

**Output:** [0,0,1,1,2,2]

### 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 a one-pass algorithm using only constant space?

```
class Solution {
public:
    void sortColors(vector<int>& nums) {

    }
};
```

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int red = 0, blue = nums.size()-1, i = 0;
        while (i <= blue) {
            if (nums[i] == 0)    swap(nums[i++], nums[red++]);
            else if (nums[i] == 2)    swap(nums[i], nums[blue--]);
            else    i++;
        }
    }
};
```

## 76. Minimum Window Substring ★ ★

### Hard

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

### Example:

**Input:** *S* = "ADOBECODEBANC", *T* = "ABC"

**Output:** "BANC"

### Note:

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

```
class Solution {
public:
    string minWindow(string s, string t) {

    }
};
```

```

class Solution {
public:
    string minWindow(string s, string t) {
        string res;
        vector<int> tab(256, 0), window(256, 0);
        for(char &c: t) ++tab[c];

        int st = 0, cur = 0;
        int minlen = INT_MAX, nums = 0;

        while (1) {
            while (cur < s.length() && nums < t.length()){
                int idx = s[cur++];
                if (tab[idx] > 0 && window[idx]++ < tab[idx]) ++nums;
            }
            if (nums < t.length()) break;
            if (minlen > cur - st) {
                res = s.substr(st, cur-st);
                minlen = res.length();
            }
            int idx = s[st++];
            if (tab[idx] > 0 && --window[idx] < tab[idx]) --nums;
        }
        return res;
    }
};

```

## 77. Combinations

### Medium

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

#### Example:

**Input:**  $n = 4, k = 2$

#### Output:

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

```
class Solution {  
public:  
    vector<vector<int>> res;  
  
    }  
};
```

```
class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> combine(int n, int k) {
        vector<int> v;
        dfs(0, k, 1, n, v);
        return res;
    }
private:
    void dfs(int cur, int k, int start, int n, vector<int> &v) {
        if (cur == k) {
            res.push_back(v);
            return;
        }
        for (int i = start; i <= n; i++) {
            v.push_back(i);
            dfs(cur+1, k, i+1, n, v);
            v.pop_back();
        }
    }
};
```

## 78. Subsets

Medium

Given a set of **distinct** integers, *nums*, return all possible subsets (the power set).

**Note:** The solution set must not contain duplicate subsets.

**Example:**

**Input:** `nums = [1,2,3]`

**Output:**

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

```
class Solution {  
public:  
    vector<vector<int>> subsets(vector<int>& nums) {  
  
        }  
};
```

```

class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<int> v;
        dfs(0, nums, v);
        return res;
    }
private:
    void dfs(int cur, vector<int> &nums, vector<int> &v) {
        if (cur == nums.size()) {
            res.push_back(v);
            return;
        }
        v.push_back(nums[cur]);
        dfs(cur+1, nums, v);
        v.pop_back();
        dfs(cur+1, nums, v);
    }
};

```

```

class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<int> v;
        dfs(0, nums, v);
        return res;
    }
private:
    void dfs(int start, vector<int> &nums, vector<int> &v) {
        res.push_back(v);
        for (int i = start; i < nums.size(); i++) {
            v.push_back(nums[i]);
            dfs(i+1, nums, v);
            v.pop_back();
        }
    }
};

```



```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>> res;
        for (int i = 0; i < (1<<n) ; i++) {
            vector<int> v;
            for (int j = 0; j < n; j++) {
                if (i & (1<<j)) v.push_back(nums[j]);
            }
            res.push_back(v);
        }
        return res;
    }
};
```

```

class Solution {
public:
    vector<vector<int>>> subsets(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>>> res{{{}}};
        for(auto &x : nums){
            res.reserve(res.size()*2);
            auto half = res.begin()+res.size(); // auto half = res.end();
            copy(res.begin(), res.end(), back_inserter(res));
            for_each(res.begin(), half, [&x](decltype(res[0]) &v){
                v.push_back(x);
            });
            /*int sz = res.size();
            for(int i = 0; i < sz; i++) {
                res.push_back(res[i]);
                res.back().push_back(x);
            }*/
        }
        return res;
    }
};

```

## 79. Word Search

### Medium

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

#### Example:

```
board =  
  
[  
  ['A','B','C','E'],  
  ['S','F','C','S'],  
  ['A','D','E','E']  
]
```

Given word = "ABCCED", return **true**.

Given word = "SEE", return **true**.

Given word = "ABCB", return **false**.

```
class Solution {  
public:  
    bool exist(vector<vector<char>>& board, string word) {  
  
    }  
};
```

```

class Solution {
public:
    bool exist(vector<vector<char>>& board, string word) {
        n = board.size();
        m = board[0].size();
        visited.resize(n, vector<bool>(m, false));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (dfs(i, j, 0, word, board))
                    return true;
            }
        }
        return false;
    }

private:
    int n, m;
    vector<vector<bool>> visited;
    bool dfs(int i, int j, int cnt, string &s, vector<vector<char>>& board) {
        if(cnt == s.length()) return true;
        if (i < 0 || j < 0 || i >= n || j >= m) return false;
        else if (visited[i][j] || board[i][j] != s[cnt++]) return false;
        visited[i][j] = true;
        bool ret =    dfs(i-1, j, cnt, s, board)
                    || dfs(i+1, j, cnt, s, board)
                    || dfs(i, j-1, cnt, s, board)
                    || dfs(i, j+1, cnt, s, board);
        visited[i][j] = false;
        return ret;
    }
};

```

## 80. Remove Duplicates from Sorted Array II

### Medium

Given a sorted array *nums*, remove the duplicates **in-place** such that duplicates appeared at most *twice* and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

#### Example 1:

Given *nums* = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of *nums* being 1, 1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.

#### Example 2:

Given *nums* = [0,0,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of *nums* being modified to 0, 0, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.

#### Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {

    }
};
```

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.size() <= 2) return nums.size();
        int index = 2, n = nums.size();
        for (int i = 2; i < n; i++){
            if (nums[index-2] != nums[i]) {
                nums[index++] = nums[i];
            }
        }
        return index;
    }
};
```

## 81. Search in Rotated Sorted Array II ★ ★

### Medium

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0, 0, 1, 2, 2, 5, 6]` might become `[2, 5, 6, 0, 0, 1, 2]`).

You are given a target value to search. If found in the array return `true`, otherwise return `false`.

#### Example 1:

**Input:** `nums = [2,5,6,0,0,1,2]`, `target = 0`

**Output:** `true`

#### Example 2:

**Input:** `nums = [2,5,6,0,0,1,2]`, `target = 3`

**Output:** `false`

#### Follow up:

- This is a follow up problem to [Search in Rotated Sorted Array](#), where `nums` may contain duplicates.
- Would this affect the run-time complexity? How and why?

```
class Solution {
public:
    bool search(vector<int>& nums, int target) {

    }
};
```

```
class Solution {
public:
    bool search(vector<int>& nums, int target) {
        int l = 0, r = nums.size()-1;
        while (l <= r) {
            int mid = l + (r-l)/2;
            if (nums[mid] == target) return true;
            if (nums[mid] < nums[r]) {
                if (nums[mid] < target && target <= nums[r]) l = mid + 1;
                else r = mid - 1;
            } else if (nums[mid] > nums[r]) {
                if (nums[l] <= target && target < nums[mid]) r = mid - 1;
                else l = mid+1;
            } else r--;
        }
        return false;
    }
};
```



## 82. Remove Duplicates from Sorted List II

### Medium

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

#### Example 1:

**Input:** 1->2->3->3->4->4->5

**Output:** 1->2->5

#### Example 2:

**Input:** 1->1->1->2->3

**Output:** 2->3

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {

    }
};
```

```

class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        head = dummy;
        ListNode *p = dummy, *q = dummy->next;

        while (q) {
            bool duplicated = false;
            while (q->next && p->next->val == q->next->val) {
                duplicated = true;
                q = q->next;
            }
            if (!duplicated) {head->next = q; head = q;}
            p = q;
            q = q->next;
        }
        head->next = nullptr;
        return dummy->next;
    }
};

```

//////////////////////////////////////递归//////////////////////////////////////

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (head == nullptr || head->next == nullptr) return head;
        ListNode *p = head->next;
        if (head->val == p->val) {
            while (p && head->val == p->val) {
                ListNode *t = p;
                p = p->next;
                delete(t);
            }
            delete(head);
            return deleteDuplicates(p);
        } else {
            head->next = deleteDuplicates(p);
            return head;
        }
    }
};
```

## 83. Remove Duplicates from Sorted List

Easy

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

**Example 1:**

**Input:** 1->1->2

**Output:** 1->2

**Example 2:**

**Input:** 1->1->2->3->3

**Output:** 1->2->3

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {

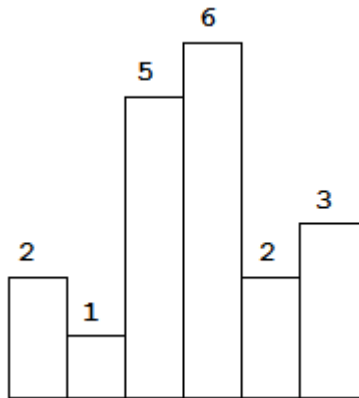
    }
};
```

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (!head) return head;
        ListNode *p = head, *q = p->next;
        while (q) {
            if (q->val != p->val) {
                p->next = q;
                p = q;
            }
            q = q->next;
        }
        p->next = nullptr;
        return head;
    }
};
```

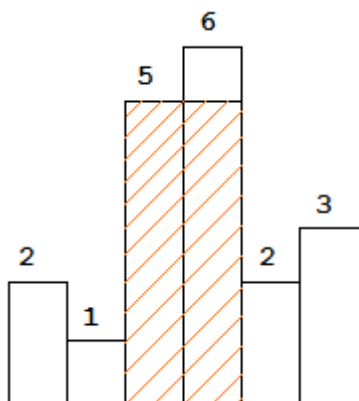
## 84. Largest Rectangle in Histogram

Hard

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.

**Example:**

**Input:** `[2,1,5,6,2,3]`

**Output:** 10

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {

    }
};
```

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        stack<int> stk;
        heights.push_back(0);
        int res = 0, n = heights.size();
        for (int i = 0; i < n; i++) {
            while (!stk.empty() && heights[i] <= heights[stk.top()]) {
                int h = heights[stk.top()];
                stk.pop();
                res = max(res,
                    h*(stk.empty() ? i : i-stk.top()-1));
            }
            stk.push(i);
        }
        return res;
    }
};
```

## 85. Maximal Rectangle★★

### Hard

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

### Example:

#### Input:

```
[  
  ["1","0","1","0","0"],  
  ["1","0","1","1","1"],  
  ["1","1","1","1","1"],  
  ["1","0","0","1","0"]  
]
```

**Output:** 6

```
class Solution {  
public:  
    int maximalRectangle(vector<vector<char>>& matrix) {  
  
    }  
};
```



```

class Solution {
public:
    int maximalRectangle(vector<vector<char>>& matrix) {
        if (matrix.empty()) return 0;
        int res = 0, n = matrix[0].size(), m = matrix.size();
        vector<int> v(n+1,0);
        for (auto &u : matrix) {
            for (int j = 0; j < n; j++) {
                if (u[j] == '1') v[j]++;
                else v[j] = 0;
            }
            res = max(res, largestRectangleArea(v));
        }
        return res;
    }

private:
    int largestRectangleArea(vector<int>& heights) {
        stack<int> stk;
        int res = 0, n = heights.size();
        for (int i = 0; i < n; i++) {
            while (!stk.empty() && heights[i] <= heights[stk.top()]) {
                int h = heights[stk.top()];
                stk.pop();
                res = max(res,
                    h*(stk.empty() ? i : i-stk.top()-1));
            }
            stk.push(i);
        }
        return res;
    }
};

```

## 86. Partition List

### Medium

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.

#### Example:

**Input:** head = 1->4->3->2->5->2,  $x = 3$

**Output:** 1->2->2->4->3->5

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {

    }
};
```

```

class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode *L_dummy = new ListNode(-1), *R_dummy = new ListNode(-1);
        ListNode *l = L_dummy, *r = R_dummy, *p = head;
        while (p) {
            if (p->val < x) {
                l->next = p;
                l = p;
            } else {
                r->next = p;
                r = p;
            }
            p = p->next;
        }
        l->next = R_dummy->next;
        r->next = NULL;
        return L_dummy->next;
    }
};

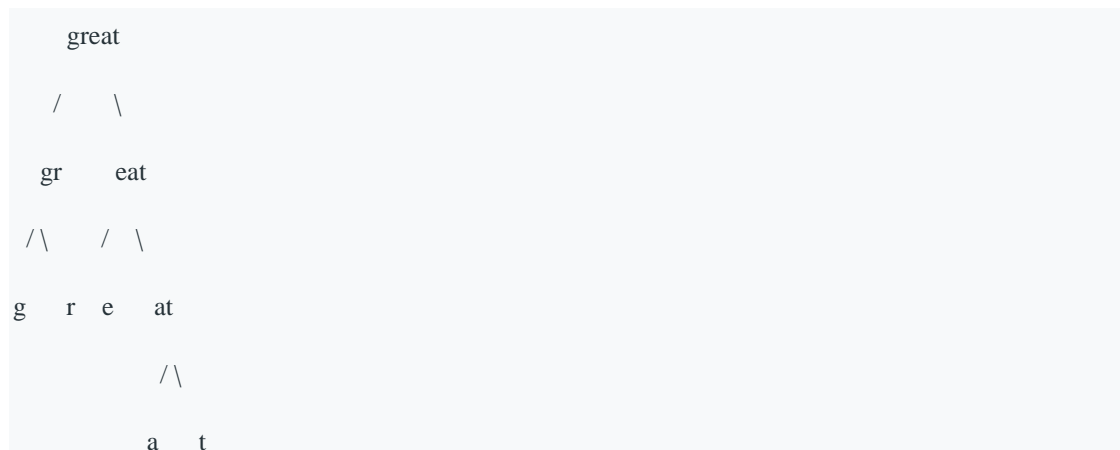
```

## 87. Scramble String ★ ★

### Hard

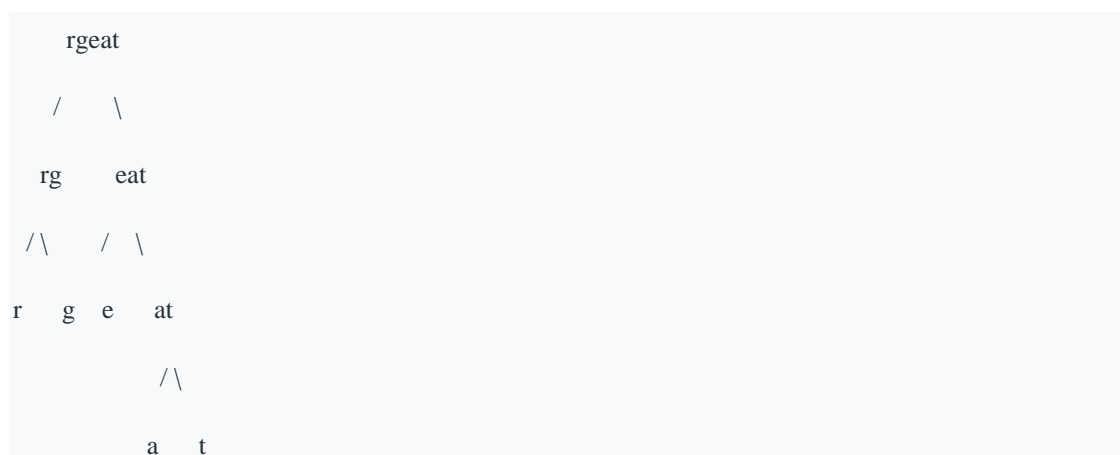
Given a string  $sI$ , we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of  $sI = \text{"great"}$ :



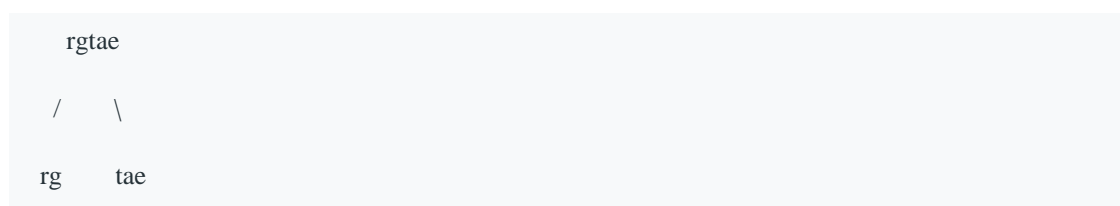
To scramble the string, we may choose any non-leaf node and swap its two children.

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



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

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



```
 /\   /  \
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  $s1$ .

**Example 1:**

**Input:**  $s1 = \text{"great"}, s2 = \text{"rgeat"}$

**Output:** true

**Example 2:**

**Input:**  $s1 = \text{"abcde"}, s2 = \text{"caebd"}$

**Output:** false

```

class Solution{
public:
    bool isScramble(string s1, string s2) {
        if (s1.length() != s2.length()) return false;
        int len = s1.length();
        vector<vector<vector<bool>>> dp
        dp.resize(len, vector<vector<bool>>(len,vector<bool>(len+1, false)));
        for (size_t i = 0; i < len; ++i) {
            for (size_t j = 0; j < len; ++j) {
                dp[i][j][1] = (s1[i] == s2[j]);
            }
        }

        for (int subLen = 2; subLen <= len; ++subLen) {
            int len0 = len - subLen;
            for (int i = 0; i <= len0; ++i) {
                for (int j = 0; j <= len0; ++j) {
                    for (int k = 0; k < subLen-1 && !dp[i][j][subLen]; ++k){
                        int len1 = k + 1, len2 = subLen - len1;
                        dp[i][j][subLen] = (dp[i][j][len1]
                                            && dp[i + len1][j + len1][len2])
                                            || (dp[i][j + len2][len1]
                                                && dp[i + len1][j][len2]);
                    }
                }
            }
        }
        return dp[0][0][len];
    }
};

```

```

class Solution {
public:
    bool isScramble(string s1, string s2) {
        if (s1 == s2) return true;

        int len = s1.length();
        int cnt[26] = {0};
        for(int i = 0; i < len; i++) {
            cnt[s1[i]-'a']++;
            cnt[s2[i]-'a']--;
        }

        for (int i : cnt) {
            if (i != 0)
                return false;
        }

        for(int i = 1; i < len; i++) {
            if (isScramble(s1.substr(0,i), s2.substr(0,i))
                && isScramble(s1.substr(i), s2.substr(i)))
                return true;
            if (isScramble(s1.substr(0,i), s2.substr(len-i))
                && isScramble(s1.substr(i), s2.substr(0,len-i)))
                return true;
        }
        return false;
    }
};

```

## 88. Merge Sorted Array

Easy

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

**Note:**

- The number of elements initialized in *nums1* and *nums2* are *m* and *n* respectively.
- You may assume that *nums1* has enough space (size that is greater or equal to  $m + n$ ) to hold additional elements from *nums2*.

**Example:**

**Input:**

`nums1 = [1,2,3,0,0,0], m = 3`

`nums2 = [2,5,6], n = 3`

**Output:** `[1,2,2,3,5,6]`

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

    }
};
```



```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int t = m+n-1, p = m-1, q = n-1;
        while (p >= 0 || q >= 0) {
            if (p >= 0 && q >= 0) {
                if (nums1[p] > nums2[q]) nums1[t--] = nums1[p--];
                else nums1[t--] = nums2[q--];
            }
            else if (q >= 0) {
                while (q >= 0) nums1[t--] = nums2[q--];
            }
            else break;
        }
    }
};
```

## 89. Gray Code

### Medium

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.

#### Example 1:

**Input:** 2

**Output:** [0,1,3,2]

**Explanation:**

00 - 0

01 - 1

11 - 3

10 - 2

For a given  $n$ , a gray code sequence may not be uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence.

00 - 0

10 - 2

11 - 3

01 - 1

#### Example 2:

**Input:** 0

**Output:** [0]

**Explanation:** We define the gray code sequence to begin with 0.

A gray code sequence of  $n$  has size  $= 2^n$ , which for  $n = 0$  the size is  $2^0 = 1$ .

Therefore, for  $n = 0$  the gray code sequence is [0].

```
class Solution {
public:
    vector<int> grayCode(int n) {
        vector<int> res{0};
        if (n == 0) return res;
        for (int i = 0; i < n; i++) {
            int bit = 1<<i;
            for(int j = res.size()-1; j >= 0; j--)
                res.push_back(res[j] | bit);
        }
        return res;
    }
};
```

## 90. Subsets II ★ ★

### Medium

Given a collection of integers that might contain duplicates, *nums*, return all possible subsets (the power set).

**Note:** The solution set must not contain duplicate subsets.

**Example:**

**Input:** [1,2,2]

**Output:**

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

```
class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {

    }
};
```

```

class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        sort(nums.begin(), nums.end()); //不同之处 1
        vector<int> v;
        dfs(0, nums, v);
        return res;
    }

private:
    void dfs(int start, vector<int> &nums, vector<int> &v) {
        res.push_back(v);
        for (int i = start; i < nums.size(); i++) {
            if (i != start && nums[i] == nums[i-1]) continue; //不同之处 2
            v.push_back(nums[i]);
            dfs(i+1, nums, v);
            v.pop_back();
        }
    }
};

```

```

class Solution {
public:
    vector<vector<int>>> res;
    vector<vector<int>>> subsetsWithDup(vector<int>& nums) {
        unordered_map<int, int> My_map;
        for (auto &i : nums) My_map[i]++;
        vector<pair<int, int>> Nums;
        for (auto &i : My_map) Nums.push_back({i.first, i.second});
        vector<int> v;
        dfs(0, Nums, v);
        return res;
    }
private:
    void dfs(int cur, vector<pair<int, int>> &Nums, vector<int> &v) {
        if (cur == Nums.size()) {
            res.push_back(v);
            return;
        }
        int x = Nums[cur].first, n = Nums[cur].second;
        dfs(cur+1, Nums, v);
        for (int i = 0; i < n; i++) {
            v.push_back(x);
            dfs(cur+1, Nums, v);
        }
        while (n-- > 0) v.pop_back();
    }
};

```

## 91. Decode Ways

### Medium

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

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given a **non-empty** string containing only digits, determine the total number of ways to decode it.

#### Example 1:

**Input:** "12"

**Output:** 2

**Explanation:** It could be decoded as "AB" (1 2) or "L" (12).

#### Example 2:

**Input:** "226"

**Output:** 3

**Explanation:** It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

```
class Solution {
public:
    int numDecodings(string s) {

    }
};
```

```
class Solution {
public:
    int numDecodings(string s) {
        int f_0 = 0, f_1 = 1, n = s.size();
        for (int i = 0; i < n; i++) {
            int f_2 = s[i] == '0' ? 0 : f_1;
            if (!i && (s[i-1]=='1' || s[i-1]=='2' && s[i]<'7')) f_2 += f_0;
            f_0 = f_1;
            f_1 = f_2;
        }
        return f_1;
    }
};
```



## 92. Reverse Linked List II

Medium

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

**Note:**  $1 \leq m \leq n \leq \text{length of list}$ .

**Example:**

**Input:** 1->2->3->4->5->NULL,  $m = 2, n = 4$

**Output:** 1->4->3->2->5->NULL

```
class Solution {  
public:  
    ListNode* reverseBetween(ListNode* head, int m, int n) {  
  
    }  
};
```

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        ListNode *dummy = new ListNode(-1), *post, *cur, *last;
        dummy->next = head;
        head = dummy;
        for (int i = 0; i < m-1; i++) head = head->next;
        last = cur = head->next;
        for (int i = m; i <= n; i++) {
            post = cur->next;
            cur->next = head->next;
            head->next = cur;
            cur = post;
        }
        last->next = cur;
        return dummy->next;
    }
};
```

## 93. Restore IP Addresses

Medium

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

**Example:**

**Input:** "25525511135"

**Output:** ["255.255.11.135", "255.255.111.35"]

```
class Solution {
public:
    vector<string> restoreIpAddresses(string s) {

    }
};
```

```

class Solution {
public:
    vector<string> restoreIpAddresses(string s) {
        vector<string> res;
        string t;
        dfs(0, s, t, 4, res);
        return res;
    }
private:
    void dfs(int cur, string &s, string t, int cnt, vector<string> &res) {
        if (cnt == 1) {
            string a = s.substr(cur);
            if (a.length() > 3 || (a[0] == '0' && a != "0")) return;
            if (stoi(a) < 256) res.push_back(t + a);
        }
        else for(int k = 1; k <= 3 && cur+k < s.length(); k++) {
            string a = s.substr(cur, k);
            if (k != 3 || stoi(a) < 256) dfs(cur+k, s, t+a+'. ', cnt-1, res);
            if (a[0] == '0') break;
        }
    }
};

```

## 94. Binary Tree Inorder Traversal

Medium

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

**Example:**

**Input:** [1,null,2,3]

```
  1
   \
    2
   /
  3
```

**Output:** [1,3,2]

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

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {

    }
};
```

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> stk;
        TreeNode *p = root;
        while (!stk.empty() || p) {
            while (p) {
                stk.push(p);
                p = p->left;
            }
            p = stk.top();
            stk.pop();
            res.push_back(p->val);
            p = p->right;
        }
        return res;
    }
};
```

## 95. Unique Binary Search Trees II ★ ★

Medium

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

**Example:**

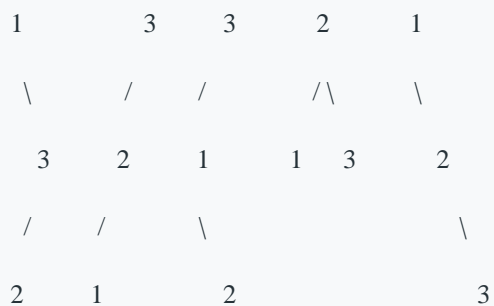
**Input:** 3

**Output:**

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

**Explanation:**

The above output corresponds to the 5 unique BST's shown below:



```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
public:
    vector<TreeNode*> generateTrees(int n) {
        if (n == 0) return vector<TreeNode*> ();
        dp.resize(n+1, vector<vector<TreeNode*>> (n+1));
        return generateTrees(1, n);
    }

private:
    vector<vector<vector<TreeNode*>>> dp;

    vector<TreeNode*> generateTrees(int start, int end) {
        vector<TreeNode*> subTree;
        if (start > end) {
            subTree.push_back(nullptr);
            return subTree;
        }
        else if (!dp[start][end].empty()) return dp[start][end];
        else for (int k = start; k <= end; ++k) {
            vector<TreeNode*> leftsubTree = generateTrees(start, k-1);
            vector<TreeNode*> rightsubTree = generateTrees(k+1, end);
            for (auto &i : leftsubTree) {
                for (auto &j : rightsubTree) {
                    TreeNode *t = new TreeNode(k);
                    t->left = i;
                    t->right = j;
                    subTree.push_back(t);
                }
            }
        }
        return dp[start][end] = subTree;
    }
};

```



## 96. Unique Binary Search Trees

Medium

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

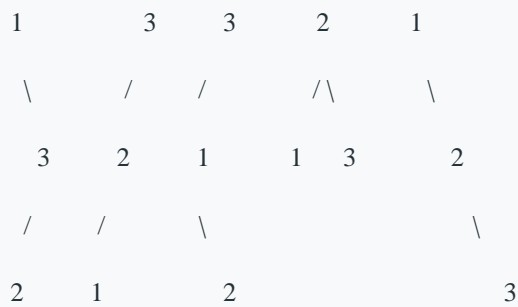
**Example:**

**Input:** 3

**Output:** 5

**Explanation:**

Given  $n = 3$ , there are a total of 5 unique BST's:



```
class Solution {
public:
    int numTrees(int n) {

    }
};
```

```
class Solution {
public:
    int numTrees(int n) {
        vector<int> f(n+1, 0);
        f[0] = f[1] = 1;
        for (int i = 2; i <= n; i++) {
            for (int k = 0; k <= i; k++) {
                f[i] += f[k]*f[i-1-k];
            }
        }
        return f[n];
    }
};
```

## 97. Interleaving String

Hard

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

**Example 1:**

**Input:**  $s1 = \text{"aabcc"}$ ,  $s2 = \text{"dbbca"}$ ,  $s3 = \text{"aadbcbcbac"}$

**Output:** true

**Example 2:**

**Input:**  $s1 = \text{"aabcc"}$ ,  $s2 = \text{"dbbca"}$ ,  $s3 = \text{"aadbbaacc"}$

**Output:** false

```

class Solution {
public:
    bool isInterleave(string s1, string s2, string s3) {
        int n = s1.length(), m = s2.length(), l = s3.length();
        if (n+m != l) return false;
        else if (n < m) return isInterleave(s2, s1, s3);
        vector<bool> f(m+1, false);
        f[0] = true;
        for (int k = 1; k <= l; k++) {
            for (int j = min(m, k); j >= max(0, k-n); j--) {
                int i = k-j;
                f[j] = (j >= 1 && s2[j-1] == s3[i+j-1] && f[j-1])
                    || (i >= 1 && s1[i-1] == s3[i+j-1] && f[j]);
            }
        }
        return f[m];
    }
};

```

## 98. Validate Binary Search Tree

### Medium

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.

#### Example 1:

**Input:**

```
    2
   /\
  1  3
```

**Output:** true

#### Example 2:

```
    5
   /\
  1  4
   /\
  3  6
```

**Output:** false

**Explanation:** The input is: [5,1,4,null,null,3,6]. The root node's value is 5 but its right child's value is 4.

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
    }
};
```

```

class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return isValidBST(root, LONG_MIN, LONG_MAX);
    }

private:
    bool isValidBST(TreeNode* root, long long Min, long long Max) {
        if (root == nullptr) return true;
        return root->val < Max && root->val > Min
            && isValidBST(root->left, Min, root->val)
            && isValidBST(root->right, root->val, Max);
    }
};

```

```

class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return isValidBST(root, nullptr, nullptr);
    }

private:
    bool isValidBST(TreeNode *root, TreeNode *Min, TreeNode *Max) {
        if (!root) return true;
        else if (Min && root->val <= Min->val || Max && root->val >= Max->val)
            return false;
        else return isValidBST(root->left, Min, root)
            && isValidBST(root->right, root, Max);
    }
};

```

## 99. Recover Binary Search Tree ★ ★

Hard

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

Recover the tree without changing its structure.

### Example 1:

**Input:** [1,3,null,null,2]

```
  1
 /
3
 \
  2
```

**Output:** [3,1,null,null,2]

```
  3
 /
1
 \
  2
```

### Example 2:

**Input:** [3,1,4,null,null,2]

```
  3
 /\
1  4
 /
  2
```

**Output:** [2,1,4,null,null,3]

```
      2
     /\
    1  4
     /
    3
```

**Follow up:**

- A solution using  $O(n)$  space is pretty straight forward.
- Could you devise a constant space solution?

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void recoverTree(TreeNode* root) {

    }
};
```



```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
public:
    void recoverTree(TreeNode* root) {
        inorder(root);
        swap(p->val, q->val);
    }

private:
    TreeNode *p = nullptr, *q = nullptr, *prev = nullptr;
    void inorder(TreeNode *root) {
        stack<TreeNode*> stk;
        while (!stk.empty() || root) {
            while (root) {
                stk.push(root);
                root = root->left;
            }
            root = stk.top();
            stk.pop();
            if (prev && root->val < prev->val) {
                if (!p) p = prev;
                q = root;
            }
            prev = root;
            root = root->right;
        }
    }
};

```

## 100. Same Tree

Easy

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

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

**Example 1:**

```
Input:      1      1
           /\    /\
          2  3   2  3
        [1,2,3], [1,2,3]
```

**Output:** true

**Example 2:**

```
Input:      1      1
           /      \
          2         2
        [1,2],   [1,null,2]
```

**Output:** false

**Example 3:**

```
Input:      1      1
           /\    /\
          2  1   1  2
        [1,2,1], [1,1,2]
```

**Output:** false

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {

    }
};
```

```

class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if (p == nullptr || q == nullptr) return p == q;
        stack<TreeNode*> s;
        s.push(p); s.push(q);
        while (!s.empty()) {
            TreeNode *p = s.top(); s.pop();
            TreeNode *q = s.top(); s.pop();
            if (p->val != q->val) return false;
            if (p->left || q->left) {
                if (!(q->left && q->left)) return false;
                s.push(p->left);
                s.push(q->left);
            }
            if (p->right || q->right) {
                if (!(p->right && q->right)) return false;
                s.push(p->right);
                s.push(q->right);
            }
        }
        return true;
    }
};

```

```

class Solution {
public:
    bool isSameTree(TreeNode *p, TreeNode *q) {
        if (p == nullptr || q == nullptr) return p == q;
        return p->val == q->val && isSameTree(p->left, q->left)
            && isSameTree(p->right, q->right);
    }
};

```