LEETCODE 题目精选

Selected Solutions

1.00

Felomeng

Contents

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**].

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; ++i) {
        if (map.containsKey(nums[i]))
            return new int[] { map.get(nums[i]), i };
        map.put(target - nums[i], i);
    }
    throw new RuntimeException("No Solution");
}
```

思路：求解过程可以分为两步：

a. 取数组中某一数a（情况总共有n种，因为总共n个数，遍历一遍）
b. 查找target - a是否在数组中。可以在遍历的同时建立索引，因为到目前为止（比如i位置），如果map中没有对应结果，那么说明对应结果在没有遍历过的部分，因为map中已经索引了到i位置的所有元素对应值。所以答案在未索引部分，故只要遍历一遍就可以。

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

```
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    ListNode pre = new ListNode(0), cur = new ListNode(0);
    pre = cur;
    int carry = 0, num = 0;
    while (l1 != null || l2 != null) {
        num = carry;
        if (l1 != null) {
            num += l1.val;
            l1 = l1.next;
        }
        if (l2 != null) {
            num += l2.val;
            l2 = l2.next;
        }
        cur.next = new ListNode(num % 10);
        cur = cur.next;
        carry = num / 10;
    }
    if (carry > 0)
        cur.next = new ListNode(carry);
    return pre.next;
}
```

思路：数字进位问题，该位有效值为值%10，进位值为值/10。可以使用一个变量记录进位值。这种下次状态和本次状态有关的处理，在循环结束后要检查是否要处理最后一次的结果（本题为进位）。

3. Longest Substring Without Repeating Characters Medium

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

Examples:

Given "abcabcbb", the answer is "abc", which the length is 3.

Given "bbbbb", the answer is "b", with the length of 1.

Given "pwwkew", 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.

```
public int lengthOfLongestSubstring(String s) {
    char[] sc = s.toCharArray();
    Set<Character> cs = new HashSet<>();
    int j = 0, maxLen = 0;
    for (int i = 0; i < sc.length; ++i) {
        char cur = sc[i];
        if (!cs.add(cur)) {
            maxLen = Math.max(i - j, maxLen);
            while (sc[j++] != cur)
                cs.remove(sc[j - 1]);
        }
    }
    return Math.max(sc.length - j, maxLen);
}
```

思路：滑动窗口。以j计左界限，每次出现重复字符后j移动到该字后一位。核心操作其实是向前检索重复字符（此处利用索引）。需要注意的是最后循环完成后，需要再算一下没有计算的那段的长度，在这些子段中取最长的。O(n)

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)).

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
```

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    int m = nums1.length, n = nums2.length, m1, m2;
    int k = (m + n + 1) >> 1;
    m1 = findKth(nums1, 0, nums2, 0, k);
    k = (m + n + 2) >> 1;
    m2 = findKth(nums1, 0, nums2, 0, k);
    return (m1 + m2) / 2.0;
}

int findKth(int[] nums1, int s1, int[] nums2, int s2, int k) {
    if (s1 == nums1.length)
        return nums2[s2 + k - 1];
    if (s2 == nums2.length)
        return nums1[s1 + k - 1];
    if (k == 1)
        return Math.min(nums1[s1], nums2[s2]);
    Integer m1 = Integer.MAX_VALUE, m2 = Integer.MAX_VALUE;
```

```
    if (s1 + k / 2 <= nums1.length)
        m1 = nums1[s1 + k / 2 - 1];
    if (s2 + k / 2 <= nums2.length)
        m2 = nums2[s2 + k / 2 - 1];
    if (m1 < m2)
        return findKth(nums1, s1 + k / 2, nums2, s2, k - k / 2);
    else
        return findKth(nums1, s1, nums2, s2 + k / 2, k - k / 2);
}
```

思路：中位数(Median)定义为中间数值（奇数），或中间两数的平均值（偶数）。所以可以转化为 2 个求第 k 个数的问题，最中间两数的平均即是要求值。奇数时，两数都是最中间那个数，平均值是自己，偶数时是中间两数平均值。

求两排序数组第 k 个数的方法(第 k 个数的索引是 k-1，隐含条件 k >=1)：

推理条件：如果单个数组 k/2 位存在，则移 k/2（记为临时中位数），临时中位数较小的向前移 k/2

结束条件：A 或 B 结束，从另一个数组中取第 k 个数；或，k=1，取两者中较小的数。

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:

Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.

Example:

Input: "cbbd"
Output: "bb"

```
public String longestPalindrome(String s) {
    if (s.length() <= 1)
        return s;
    int max = 0, start = 0, end = 0;
    int[] p = new int[2];
    for (int i = 0; i < s.length() - 1; ++i) {
        p = getPalindrome(s, i, i + 1);
        if (p[1] - p[0] > max) {
            max = p[1] - p[0];
            start = p[0];
            end = p[1];
        }
        p = getPalindrome(s, i, i);
        if (p[1] - p[0] > max) {
            max = p[1] - p[0];
            start = p[0];
            end = p[1];
        }
    }
    return s.substring(start, end);
}

int[] getPalindrome(String s, int i, int j) {
    while (i >= 0 && j < s.length()) {
        if (s.charAt(i) != s.charAt(j))
            break;
        ++j;
        --i;
    }
    return new int[] { i + 1, j };
```

}
思路：穷举法，从中间（注意中间为空或中间为一个元素两种情况）扩展，存最大可能。注意 substring 区间前闭后开[start,end)

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 text, int nRows);
```

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

```
public String convert(String s, int numRows) {
    StringBuilder[] sbs = new StringBuilder[numRows];
    for (int j = 0; j < sbs.length; ++j)
        sbs[j] = new StringBuilder();
    char[] cs = s.toCharArray();
    int n = cs.length, i = 0;
    while (i < n) {
        for (int j = 0; j < numRows && i < n; ++j)
            sbs[j].append(cs[i++]);
        for (int j = numRows - 2; j > 0 && i < n; --j)
            sbs[j].append(cs[i++]);
    }
    StringBuilder ans = new StringBuilder();
    for (StringBuilder sb : sbs)
        ans.append(sb);
    return ans.toString();
}
```

思路：正反反复读即可把 ZigZag 读出，存入 StringBuilder（每行一个），然后按顺序拼接 StringBuilder。注意反向时需要忽略首尾行。

7. Reverse Integer Easy

Reverse digits of an integer.
Example1: x = 123, return 321
Example2: x = -123, return -321
click to show spoilers.
Have you thought about this?
Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!
If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.
Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?
For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.
Note:
The input is assumed to be a 32-bit signed integer. Your function should **return 0 when the reversed integer overflows**.

```
public int reverse(int x) {
    int rev = 0, last = 0;
    int sign = x < 0 ? -1 : 1;
```

```
    x = Math.abs(x);
    while (x > 0) {
        rev = last * 10 + x % 10;
        if (rev / 10 != last)
            return 0;
        x /= 10;
        last = rev;
    }
    return rev * sign;
}
```

思路：从个位取（通过取余%10），逐位取，前面位*10+当前位。最终转换完成。溢出处理：当*10 时有可能溢出，只要判断/10 是否等于*10 之前的数。

8. String to Integer (atoi) Medium

Implement atoi to convert a string to an integer.

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

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

spoilers alert... click to show requirements for atoi.

Requirements for atoi:

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

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

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

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

```
public int myAtoi(String str) {
    str = str.trim();
    if (str.length() == 0)
        return 0;
    long num = 0;
    int sign = 1, cur = 0;
    char first = str.charAt(cur);
    if (first == '-') {
        sign = -1;
        ++cur;
    } else if (first == '+') {
        ++cur;
    }
    while (cur < str.length()) {
        char c = str.charAt(cur++);
        if (!Character.isDigit(c))
            return (int) num * sign;
        num = num * 10 + (c - '0');
        if (sign > 0 && num > Integer.MAX_VALUE)
            return Integer.MAX_VALUE;
        if (sign < 0 && num * sign < Integer.MIN_VALUE)
            return Integer.MIN_VALUE;
    }
    return (int) num * sign;
}
```

思路：考虑正负和溢出，从前到后逐一扫转即可。技巧：转成 char 后和'0'的差值即是该位数值。

9. Palindrome Number Easy

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

Some hints:

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

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

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

There is a more generic way of solving this problem.

```
public boolean isPalindrome(int x) {
    if (x < 0 || (x != 0 && x % 10 == 0))
        return false;
    int rev = 0;
    while (x > rev) {
        rev = rev * 10 + x % 10;
        x /= 10;
    }
    return rev == x || rev / 10 == x;
}
```

思路：先判定特殊情况：负数、零和个位数为 0。如果够长，则把后面一半逆位反过来（止于逆反数大于前面一半时）逆反数（或逆反数/10）和前面一半对比即可。这时逆反数是不可能越界的。

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 precedeng 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

```
public boolean isMatch(String s, String p) {
    int m = s.length(), n = p.length();
    boolean[][] dp = new boolean[m + 1][n + 1];
    dp[0][0] = true;
    for (int j = 2; j <= n; ++j)
        if (p.charAt(j - 1) == '*')
            dp[0][j] = dp[0][j - 2];
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
            if (match(s.charAt(i - 1), p.charAt(j - 1)))
                dp[i][j] = dp[i - 1][j - 1];
            else if (p.charAt(j - 1) == '*') {
                dp[i][j] = dp[i][j - 2];
                if (match(s.charAt(i - 1), p.charAt(j - 2)))
                    dp[i][j] |= dp[i - 1][j];
            }

    return dp[m][n];
```

```
}

boolean match(char a, char b) {
    return a == b || b == '.';
}
```

思路：动态规划。递推表达式：

1. 默认全部为假
2. 两字符相等或 p 为'.'，结果等于去掉当前两字符，dp[i][j]=dp[i-1][j-1];
3. P 为'*'，有两种情况
    a. '*'前字符没有出现过，则结果与去掉*和 *前字符结果相等，dp[i][j]=dp[i][j-2]
    b. 如果'*'前字符与 s 当前字符匹配，则结果与去掉 s 当前字符结果一致，即 s 当前字符出现与否结果等价，没有出现的情况已经判断过，直接利用没出现的情况作为此情况的值，if(match(s.charAt(i-1), p.charAt(j-2)) {dp[i][j] |= dp[i-1][j]};

11.Container With Most Water Medium

Given *n* non-negative integers a_1, a_2, ..., a_n, where each represents a point at coordinate (i, a_i). *n* vertical lines are drawn such that the two endpoints of line *i* is at (i, a_i) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and *n* is at least 2.

```
public int maxArea(int[] height) {
    int i = 0, j = height.length - 1, max = 0;
    while (i < j) {
        max = Math.max(max, Math.min(height[i], height[j]) * (j - i));
        if (height[i] > height[j])
            --j;
        else
            ++i;
    }
    return max;
}
```

思路：注意题意，每柱间隔相等，所以使用夹逼法，从两边每次缩小一格（保留较长的柱子）。

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: C = 100, L = 50, XXX = 30 and III = 3.

Example 5:

Input: 1994

Output: "MCMXCIV"

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

```
public String intToRoman(int num) {
    String M[] = { "", "M", "MM", "MMM" };
    String C[] = { "", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM" };
    String X[] = { "", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC" };
    String I[] = { "", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX" };
    return M[num / 1000] + C[(num % 1000) / 100] + X[(num % 100) / 10] + I[num % 10];s
}
```

思路：穷举每一位上所有数值的可能，然后将他们放入数组，0 位置放空，通过取余+整除取出每一位的值连接即得结果。

13.Roman to Integer Easy

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

```
public int romanToInt(String s) {
    int sum = 0;
    if (s.contains("IV"))
        sum -= 2;
    if (s.contains("IX"))
        sum -= 2;
    if (s.contains("XL"))
        sum -= 20;
    if (s.contains("XC"))
        sum -= 20;
    if (s.contains("CD"))
        sum -= 200;
    if (s.contains("CM"))
        sum -= 200;
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == 'M')
            sum += 1000;
        if (s.charAt(i) == 'D')
            sum += 500;
        if (s.charAt(i) == 'C')
            sum += 100;
        if (s.charAt(i) == 'L')
            sum += 50;
        if (s.charAt(i) == 'X')
            sum += 10;
        if (s.charAt(i) == 'V')
            sum += 5;
        if (s.charAt(i) == 'I')
            sum += 1;
    }
    return sum;
}
```

思路：把所有出现的字符按次数相加即可。特殊情况：4、40 之类的可使结果多出 2、20，注意减去。因为限定了数值为 1 到 3999，所以每种特殊情况只会出现一次。

14.Longest Common Prefix Easy

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

```
public String longestCommonPrefix(String[] strs) {
    if (strs.length == 0)
        return "";
    StringBuilder ans = new StringBuilder();
    int n = Integer.MAX_VALUE;
    boolean done = false;
    for (String str : strs)
        n = Math.min(n, str.length());
    for (int i = 0; i < n; ++i) {
        Character cur = null;
        for (String str : strs) {
            if (cur == null)
                cur = str.charAt(i);
            else if (str.charAt(i) != cur) {
                done = true;
```

```
                break;
            }
        }
        if (done)
            break;
        ans.append(cur);
    }
    return ans.toString();
}
```

思路：先求最短字符串的长度 n，然后判断逐一字符判断各个数组前 n 个是否相等，并存入 StringBuilder，不等或到 n 时返回结果。

15.3Sum Medium

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

Note: The solution set must not contain duplicate triplets.

> For example, given array S = [-1, 0, 1, 2, -1, -4],
> A solution set is:
> [[-1, 0, 1],
> [-1, -1, 2]]

```
public List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> ans = new ArrayList<>();
    Arrays.sort(nums);
    int n = nums.length, sum = 0;
    for (int i = 0; i < n - 2;) {
        if (nums[i] > 0)
            break;
        int j = i + 1, k = n - 1;
        while (j < k) {
            sum = nums[i] + nums[j] + nums[k];
            if (sum == 0) {
                ans.add(Arrays.asList(nums[i], nums[j], nums[k]));
            }
            if (sum <= 0) {
                while (nums[j] == nums[++j] && k > j)
                    ;
            }
            if (sum >= 0) {
                while (nums[k] == nums[--k] && k > j)
                    ;
            }
        }
        while (nums[i] == nums[++i] && i < n - 2)
            ;
    }
    return ans;
}
```

思路1：先排序O(nlogn)，然后查每一个数后面两数之和为当前数相反数的所有可能（夹逼法）。$O(n^2)$

思路2（未实现）：不排序，先做2数和的倒排索引，查每一个数是否有相应的2数和为其相反数。$O(n^3)$

16.3Sum Closest Medium

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

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

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

```
public int threeSumClosest(int[] nums, int target) {
    int sum = 0, s = Integer.MAX_VALUE, ans = 0;
    Arrays.sort(nums);
    int n = nums.length, j = 0, k = 0;
    for (int i = 0; i < n - 2; ++i) {
        j = i + 1;
        k = n - 1;
        while (j < k) {
            sum = nums[i] + nums[j] + nums[k];
            if (sum == target)
                return sum;
            if (Math.abs(target - sum) < s) {
                s = Math.abs(target - sum);
                ans = sum;
            }
            if (sum < target)
                while (nums[j] == nums[++j] && j < k)
                    ;
            else
                while (nums[k] == nums[--k] && j < k)
                    ;
        }
    }
    return ans;
}
```

思路：与上一题一致，不同处：在判断 Sum 时不是看是不是=0，而是看是不是更接近 target。这时注意不能再以某元素与目标元素的大小判定是否有可能有结果。

17.Letter Combinations of a Phone Number Medium

Given a digit string, return all possible letter combinations that the number could represent.
A mapping of digit to letters (just like on the telephone buttons) is given below.

Input: Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

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

```
public List<String> letterCombinations(String digits) {
    char[][] cs = new char[8][];
    cs[0] = new char[] { 'a', 'b', 'c' };
    cs[1] = new char[] { 'd', 'e', 'f' };
    cs[2] = new char[] { 'g', 'h', 'i' };
    cs[3] = new char[] { 'j', 'k', 'l' };
    cs[4] = new char[] { 'm', 'n', 'o' };
    cs[5] = new char[] { 'p', 'q', 'r', 's' };
    cs[6] = new char[] { 't', 'u', 'v' };
    cs[7] = new char[] { 'w', 'x', 'y', 'z' };
    List<String> ans = new ArrayList<>();
    bt(digits, 0, cs, ans, new StringBuilder());
    return ans;
}
```

```
void bt(String digits, int i, char[][] cs, List<String> ans, StringBuilder can) {
    if (i == digits.length()) {
        if (i != 0)
            ans.add(can.toString());
        return;
    }
    for (char c : cs[digits.charAt(i) - '2']) {
        StringBuilder nc = new StringBuilder(can);
        nc.append(c);
        bt(digits, i + 1, cs, ans, nc);
    }
}
```

思路：回溯法，每键创建一个新的List，使用之前的组合添加每一个可能的字符，每一种新结果添加到新List中。O(n^k)

18.4Sum Medium

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

Note: The solution set must not contain duplicate quadruplets.

> For example, given array S = [1, 0, -1, 0, -2, 2], and target = 0.
> A solution set is:
> [[-1, 0, 0, 1],
> [-2, -1, 1, 2],
> [-2, 0, 0, 2]]

```
public List<List<Integer>> fourSum(int[] nums, int target) {
    List<List<Integer>> ans = new ArrayList<>();
    int n = nums.length, sum = 0;
    Arrays.sort(nums);
    for (int i = 0; i < n - 3;) {
        for (int j = i + 1; j < n - 2;) {
            int k = j + 1, l = n - 1;
            while (k < l) {
                sum = nums[i] + nums[j] + nums[k] + nums[l];
                if (sum == target)
                    ans.add(Arrays.asList(nums[i], nums[j], nums[k], nums[l]));
                if (sum <= target)
                    while (k < l && nums[k] == nums[++k])
                        ;
                if (sum >= target)
                    while (k < l && nums[l] == nums[--l])
                        ;
            }
            while (nums[j] == nums[++j] && j < n - 2)
                ;
        }
        while (nums[i] == nums[++i] && i < n - 3)
            ;
    }
    return ans;
}
```

思路：同 3Sum，转化为更低阶处理。首先排序，当 k>2 时逐一试看是否有结果（注意排重和中断不可能的情况），当 k=2 时，使用夹逼法。

注：性能可以优化，比如当前 m 个数已经大于 Target 时，则无须进入下一循环，本处从略。

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.
For example,

> Given linked list: **1->2->3->4->5**, and ***n* = 2**.
> After removing the second node from the end, the linked list becomes **1->2->3->5**.

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

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode pre = new ListNode(0);
    pre.next = head;
    ListNode i = pre, j = head;
    for (int k = 0; k < n; ++k)
        j = j.next;
    while (j != null) {
        i = i.next;
        j = j.next;
    }
    if (i.next == head)
        head = head.next;
    else
        i.next = i.next.next;
    return head;
}
```

思路：双指针法，相隔 n 前移后指针。特殊情况：n 正好为数组长度，这时需要移动 head

20.Valid Parentheses Easy

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

```
public boolean isValid(String s) {
    Map<Character, Character> m = new HashMap<>();
    m.put('(', ')');
    m.put('[', ']');
    m.put('{', '}');
    Stack<Character> st = new Stack<>();
    for (char c : s.toCharArray()) {
        if (m.containsKey(c))
            st.push(m.get(c));
        else if (st.isEmpty() || st.pop() != c)
            return false;
    }
    return st.isEmpty();
}
```

思路：典型的压栈出栈操作，遇左括号则压右括号，右则出栈且对比。对比有不同或最后栈不为空，则失败。

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.

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    if (l1 == null)
        return l2;
    if (l2 == null)
        return l1;
    if (l1.val < l2.val) {
```

```
        l1.next = mergeTwoLists(l1.next, l2);
        return l1;
    }
    l2.next = mergeTwoLists(l1, l2.next);
    return l2;
}
```

思路：类似归并排序，每次前移一个较小的数字。也可以用非递归方式，思路一样：每次把较小的数值的元素所在链表前进到下一位。

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:

```
[ "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"]
```

```
public List<String> generateParenthesis(int n) {
    List<String> list = new ArrayList<String>();
    backtrack(list, "", 0, 0, n);
    return list;
}

private static void backtrack(List<String> list, String str, int open, int close, int max) {
    if (str.length() == max * 2) {
        list.add(str);
        return;
    }
    if (open < max)
        backtrack(list, str + "(", open + 1, close, max);
    if (close < open)
        backtrack(list, str + ")", open, close + 1, max);
}
```

思路：回溯法。逐一添加括号，规则：尽量多添加左，不能多添加右（close < open），够长则记录。

23.Merge k Sorted Lists Hard

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

```
public ListNode mergeKLists(ListNode[] lists) {
    if (lists == null || lists.length == 0)
        return null;
    ListNode dummy = new ListNode(0);
    ListNode cur = dummy;
    return merge(lists, 0, lists.length - 1);
}

ListNode merge(ListNode[] lists, int i, int j) {
    if (j == i)
        return lists[i];
    ListNode l = merge(lists, i, (i + j) / 2);
    ListNode r = merge(lists, (i + j) / 2 + 1, j);
    return merge(l, r);
}

ListNode merge(ListNode l, ListNode r) {
    ListNode dummy = new ListNode(0);
```

```
    ListNode cur = dummy;
    while (l != null || r != null) {
        if (l == null) {
            cur.next = r;
            break;
        }
        if (r == null) {
            cur.next = l;
            break;
        }
        if (l.val <= r.val) {
            cur.next = l;
            l = l.next;
        } else {
            cur.next = r;
            r = r.next;
        }
        cur = cur.next;
    }
    return dummy.next;
}
```

思路：分治法。把所有链表分成一对一对的，两两组合，然后再把新的链表队列两两组合，直到只剩 1 条链表。

另：不进行分组，直接顺序两两组合也能通过，不过慢很多，因为比较次数增加了很多。

24.Swap Nodes in Pairs Medium

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

For example,

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

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

```
public ListNode swapPairs(ListNode head) {
    ListNode p = head;
    if (p == null || p.next == null)
        return head;
    ListNode newHead = p.next;
    p.next = p.next.next;
    newHead.next = p;
    p = newHead.next.next;
    newHead.next.next = swapPairs(p);
    return newHead;
}
```

思路：递归，每次 Swap 两个，然后指向下一部分，最终返回当前部分的头给上一部分的尾。

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.

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

Only constant memory is allowed.

For example,

Given this linked list: `1->2->3->4->5`

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

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

```
public ListNode reverseKGroup(ListNode head, int k) {
```

```
    if (head == null || head.next == null || k < 2)
        return head;
    ListNode pre = new ListNode(0), cur, tail, dummy = new ListNode(0);
    dummy.next = head;
    pre = dummy;
    tail = dummy;
    while (true) {
        int n = 0;
        while (n++ < k && tail != null)
            tail = tail.next;
        if (tail == null)
            break;
        head = pre.next;
        while (pre.next != tail) {
            cur = pre.next;
            pre.next = cur.next;
            cur.next = tail.next;
            tail.next = cur;
        }
        pre = head;
        tail = head;
    }
    return dummy.next;
}
```

思路：每次处理 k 个结点，三个指针，分别指头、尾和当前元素(temp)，使用 temp 不断把头部元素移动到尾部（头尾分别向中间前进一位），直到头尾指到一处，再进行下一个 k 的处理。每次 k 结点的开头（pre）总是上 k 个的结尾结点，即未翻转时的头结点。

26.Remove Duplicates from Sorted Array Easy

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

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

For example,

Given input array *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 new length.

```
public int removeDuplicates(int[] nums) {
    if (nums == null || nums.length == 0)
        return 0;
    int temp = nums[0], cur;
    int count = 1;
    for (int i = 1; i < nums.length; ++i) {
        cur = nums[i];
        if (temp != cur)
            nums[count++] = cur;
        temp = cur;
    }
    return count;
}
```

思路：顺序找即可，先检查为空情况。如果不空，保留第一元素（计数 1），顺序处理后续元素，如遇与当前不同则记入原数组头部并计数。

27.Remove Element Easy

Given an array and a value, 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 in place with constant memory.

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

Example:
Given input array *nums* = [3,2,2,3], *val* = 3
Your function should return length = 2, with the first two elements of *nums* being 2.

```
public int removeElement(int[] nums, int val) {
    int j = 0;
    for (int i = 0; i < nums.length; ++i)
        if (val != nums[i])
            nums[j++] = nums[i];
    return j;
}
```

思路：与上一题相同，顺序找与给定字符不同的，将其放到当前指针处，并计数。

28.Implement strStr() Easy

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

```
public int strStr(String haystack, String needle) {
    if (haystack == null || needle == null)
        return -1;
    int m = haystack.length();
    int n = needle.length();
    Set<String> ns = new HashSet<String>() {
        {
            add(needle);
        }
    };
    for (int i = 0; i < m - n + 1; ++i) {
        String cur = haystack.substring(i, i + n);
        if (ns.contains(cur))
            return i;
    }
    return -1;
}
```

思路：取得haystack中每一个与needle等长的子串，取Hash与needle相比，然后再计算是否相等。 O(n)
另：字符串匹配算法有很多，参考：字符串匹配算法综述：BF、RK、KMP、BM、Sunday

29.Divide Two Integers Medium

Divide two integers without using multiplication, division and mod operator.
If it is overflow, return MAX_INT.

```
public int divide(int dividend, int divisor) {
    if (divisor == 0)
        throw new java.lang.ArithmeticException("/ by zero");
    long result = divideLong(dividend, divisor);
    return result > Integer.MAX_VALUE ? Integer.MAX_VALUE : (int) result;
}

public long divideLong(long dividend, long divisor) {
    boolean negative = dividend < 0 != divisor < 0;
    if (dividend < 0)
        dividend = -dividend;
    if (divisor < 0)
        divisor = -divisor;
    if (dividend < divisor)
        return 0;
    long sum = divisor;
    long divide = 1;
    while ((sum + sum) <= dividend) {
        sum += sum;
```

```
            divide += divide;
        }
        return negative ? -(divide + divideLong((dividend - sum), divisor))
                : (divide + divideLong((dividend - sum), divisor));
    }
```

思路：模拟除法，每次被除数(dividend)加倍，则除的结果加倍（从 1 开始，2、4、8……）。O((lgn)^2)
除数/被除数=结果，根据等式原理，两边同时乘以相同数仍然相等。例：
55/5 = 11
每次加倍，所以 5->10->20->40 (对应结果 1->2->4->8)，注意 1*5=5, 2*5=10, 4*5=20, 8*5=40

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.
For example, given:
s: "barfoothefoobarman"
words: ["foo", "bar"]
You should return the indices: [0,9].
(order does not matter).

```
public List<Integer> findSubstring(String s, String[] words) {
    if (words.length == 0 || words[0].length() == 0)
        return new ArrayList<>();
    HashMap<String, Integer> map = new HashMap<>();
    for (String word : words)
        map.put(word, map.getOrDefault(word, 0) + 1);
    List<Integer> list = new ArrayList<>();
    int gap = words[0].length();
    int nlen = words.length * gap;
    for (int k = 0; k < gap; k++) {
        HashMap<String, Integer> wordmap = new HashMap<>(map);
        for (int startIndex = k, shift = 0; startIndex < s.length() - nlen + 1 && startIndex +
shift <= s.length() - gap;) {
            String temp = s.substring(startIndex + shift, startIndex + shift + gap);
            if (wordmap.containsKey(temp)) {
                wordmap.put(temp, wordmap.get(temp) - 1);
                if (wordmap.get(temp) == 0)
                    wordmap.remove(temp);
                if (wordmap.isEmpty())
                    list.add(startIndex);
                shift += gap;
            } else {
                if (shift == 0)
                    startIndex += gap;
                else {
                    wordmap.put(s.substring(startIndex, startIndex + gap),
wordmap.getOrDefault(s.substring(startIndex, startIndex + gap), 0) + 1);
                    startIndex += gap;
                    shift -= gap;
                }
            }
        }
    }
    return list;
}
```

思路：滑动窗口。先把词存入词典，一词一词进行判定（使用起始坐标 startIndex 和游标 shift 标记查看过的内容），看有没有在列表中，有就词典中次数-1，如果次数为 0 就从词典中删除，如果词典为空，则找到一个解。如果下一词不在词典中，则把前面的词收回到词典，并且游标移动到当前。

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, do not allocate extra memory.
Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1

```
public void nextPermutation(int[] nums) {
    int i;
    for (i = nums.length - 1; i > 0; --i)
        if (nums[i] > nums[i - 1])
            break;
    if (i > 0)
        for (int j = nums.length - 1; j >= i; --j)
            if (nums[j] > nums[i - 1]) {
                swap(nums, j, i - 1);
                break;
            }
    reverse(nums, i, nums.length - 1);
}

private static void swap(int[] nums, int i, int j) {
    int temp = nums[i];
    nums[i] = nums[j];
    nums[j] = temp;
}

private static void reverse(int[] nums, int i, int j) {
    while (i < j)
        swap(nums, i++, j--);
}
```

思路：找下一个排列中略大于现在数的数。如果这个数存在，则数组中一定有一个低位数大于某一高位数。从个位开始向高位逐一查找，看相邻两数是否高位的较小（注意相邻两数的关系已经决定了数列的单调性），如果是，那么需要找一个大于它的数和它置换。可以证明高位之后的数字是递减排列，所以从最低位找到第一个大于该高位的数与之置换，则新数一定大于当前数，且该高位后为递减排列，这时需要把这部分倒置为递增排列。如果这个数不存在，可以证明，整个数组是递减排列，整体倒置即可。

32.Longest Valid Parentheses Hard

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.
For "(()", the longest valid parentheses substring is "()", which has length = 2.
Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

```
public int longestValidParentheses(String s) {
    Stack<Integer> stack = new Stack<Integer>();
    int max = 0;
    int left = -1;
    for (int j = 0; j < s.length(); j++) {
        if (s.charAt(j) == '(')
            stack.push(j);
        else {
            if (stack.isEmpty())
                left = j;
            else {
                stack.pop();
```

```
                if (stack.isEmpty())
                    max = Math.max(max, j - left);
                else
                    max = Math.max(max, j - stack.peek());
            }
        }
    }
    return max;
}
```

思路：典型压栈题目。遇左括号压栈，右括号判断是否合法，合法弹出判定最大长度，不合法则把左边界右移。特殊情况：栈不为空时也有可能取得最大长度。

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.

```
public int search(int[] A, int target) {
    int lo = 0;
    int hi = A.length - 1;
    if (hi < 0)
        return -1;
    while (lo < hi) {
        int mid = (lo + hi) / 2;
        if (A[mid] == target)
            return mid;
        if (A[lo] <= A[mid]) {
            if (target >= A[lo] && target < A[mid])
                hi = mid - 1;
            else
                lo = mid + 1;
        } else {
            if (target > A[mid] && target <= A[hi])
                lo = mid + 1;
            else
                hi = mid - 1;
        }
    }
    return A[lo] == target ? lo : -1;
}
```

思路：二分查找其实并不需要整个数组有序，其中有一半有序即可。一个变形的二分查找，判断条件：先和 0 点比，可以判断是怎么 Rotate 的，然后再判断应该落在哪个区间。O(lgn)

34.Search for a Range Medium

Given an array of integers 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]`.
For example,
Given `[5, 7, 7, 8, 8, 10]` and target value 8,
return `[3, 4]`.

```
public int[] searchRange(int[] nums, int target) {
    int[] ans = { -1, -1 };
    ans[0] = firstPlaceLeft(nums, target);
    ans[1] = firstPlaceRight(nums, target);
    return ans;
}
```

```
private static int firstPlaceLeft(int[] nums, int target) {
    int l = 0, r = nums.length - 1, mid = 0;
    while (l <= r) {
        mid = (l + r) / 2;
        if (target > nums[mid])
            l = mid + 1;
        else {
            if (nums[mid] == target && (mid == l || (mid > l && nums[mid - 1] != target))) {
                return mid;
            }
            r = mid - 1;
        }
    }
    return -1;
}

private static int firstPlaceRight(int[] nums, int target) {
    int l = 0, r = nums.length - 1, mid = 0;
    while (l <= r) {
        mid = (l + r) / 2;
        if (target < nums[mid])
            r = mid - 1;
        else {
            if (nums[mid] == target && (mid == r || (mid < r && nums[mid + 1] != target))) {
                return mid;
            }
            l = mid + 1;
        }
    }
    return -1;
}
```

思路：使用两个变形的二分查找，分别找第一个、最后一个等于给定数的位置。

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.

Here are few examples.

[1,3,5,6], 5 → 2

[1,3,5,6], 2 → 1

[1,3,5,6], 7 → 4

[1,3,5,6], 0 → 0

```
public int searchInsert(int[] nums, int target) {
    int n = nums.length;
    if (n == 0) {
        return 0;
    }
    int left = 0, right = n - 1;
    int mid = 0;
    while (left <= right) {
        mid = left + (right - left) / 2;
        if (target > nums[mid])
            left = Math.min(mid + 1, right);
        else if (target < nums[mid])
            right = Math.max(mid - 1, left);
        else
            return mid;
        if (left == right) {
```

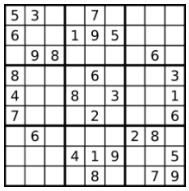
```
            if (target <= nums[left])
                return left;
            return right + 1;
        }
    }
    throw new RuntimeException("not possible");
}
```

思路：二分查找，但是允许左右指针指向同一坐标，如果找不到则左右指针指向同一坐标，如果这个数小于左面的数，则应该插入当前位置，否则应该插入下一位置。

36.Valid Sudoku Medium

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

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



A partially filled sudoku which is valid.

Note:

A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated.

```
public boolean isValidSudoku(char[][] board) {
    Set<String> seen = new HashSet<>();
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) {
            char number = board[i][j];
            if (number != '.')
                if (!seen.add(number + " in row " + i)
                        || !seen.add(number + " in column " + j)
                        || !seen.add(number + " in block " + i / 3 + "-" + j / 3))
                    return false;
        }
    }
    return true;
}
```

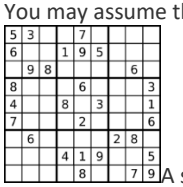
思路：只要每行、列、九宫格没有重复即可。行、列容易，九宫技巧：i/3,j/3

37.Sudoku Solver Hard

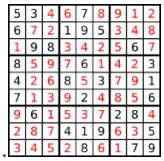
Write a program to solve a Sudoku puzzle by filling the empty cells.

Empty cells are indicated by the character '.'.

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



A sudoku puzzle...



...and its solution numbers marked in red.

```
public void solveSudoku(char[][] board) {
    if (board == null || board.length == 0)
        return;
    solve(board);
}
```

```
private boolean solve(char[][] board) {
    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[0].length; j++) {
            if (board[i][j] == '.') {
                for (char c = '1'; c <= '9'; c++)
                    if (isValid(board, i, j, c)) {
                        board[i][j] = c;
                        if (solve(board))
                            return true;
                        else
                            board[i][j] = '.';
                    }
                return false;
            }
        }
    }
    return true;
}

private boolean isValid(char[][] board, int row, int col, char c) {
    for (int i = 0; i < 9; i++) {
        if (board[i][col] != '.' && board[i][col] == c)
            return false;
        if (board[row][i] != '.' && board[row][i] == c)
            return false;
        if (board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] != '.'
                && board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
            return false;
    }
    return true;
}
```

思路：回溯法，每一个未填位置试图填入 1-9 每个数，然后再试下一位置。失败时记得把当前值回置为”.”。

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*, generate the n^{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"

```
public String countAndSay(int n) {
    if (n == 1)
```

```
        return "1";
    return countAndSay(countAndSay(n - 1));
}

private String countAndSay(String str) {
    StringBuilder sb = new StringBuilder();
    char[] cs = str.toCharArray();
    int i = 0, j = 0;
    for (; i < cs.length; i = j) {
        char cur = cs[i];
        j = i + 1;
        while (j < cs.length && cur == cs[j])
            ++j;
        sb.append(j - i);
        sb.append(cur);
    }
    return sb.toString();
}
```

思路：此题非常典型，只要掌握的递归的思路，迎刃而解。递归求解，每次把问题降低一个数，直到数值为“1”。然后代入上一级 Count。O(n(m^2))，m 为数字长度。

39.Combination Sum Medium

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

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

Note:

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

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

A solution set is:

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

```
public List<List<Integer>> combinationSum(int[] candidates, int target) {
    List<List<Integer>> r = new ArrayList<List<Integer>>();
    int n = candidates.length;
    for (int i = 0; i < n; ++i) {
        List<Integer> candidateL = new ArrayList<Integer>();
        candidateL.add(candidates[i]);
        backTrack(candidates, i, candidates[i], candidateL, target, r);
    }
    return r;
}

private static void backTrack(int[] nums, int i, int sum, List<Integer> candidateL, int target,
        List<List<Integer>> r) {
    if (sum > target)
        return;
    if (sum == target) {
        List<Integer> newL = new ArrayList<Integer>();
        newL.addAll(candidateL);
        r.add(newL);
        return;
    }
    for (int j = i; j < nums.length; ++j) {
        candidateL.add(Integer.valueOf(nums[j]));
        sum += nums[j];
        backTrack(nums, j, sum, candidateL, target, r);
```

```
            candidateL.remove(Integer.valueOf(nums[j]));
            sum -= nums[j];
        }
    }
```
思路：回溯法。尝试每种组合。结束条件：如果和已经大于等于目标值。O(n^k)

40.Combination Sum II Medium

Given a collection of candidate numbers (***C***) and a target number (***T***), find all unique combinations in ***C*** where the candidate numbers sums to ***T***.
Each number in ***C*** may only be used **once** in the combination.
Note:

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

For example, given candidate set [10, 1, 2, 7, 6, 1, 5] and target 8,
A solution set is:

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

```
public List<List<Integer>> combinationSum2(int[] cand, int target) {
    Arrays.sort(cand);
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<>();
    dfsCom(cand, 0, target, path, res);
    return res;
}

void dfsCom(int[] cand, int cur, int target, List<Integer> path, List<List<Integer>> res) {
    if (target == 0) {
        res.add(new ArrayList<>(path));
        return;
    }
    if (target < 0)
        return;
    for (int i = cur; i < cand.length; i++) {
        if (i > cur && cand[i] == cand[i - 1])
            continue;
        path.add(path.size(), cand[i]);
        dfsCom(cand, i + 1, target - cand[i], path, res);
        path.remove(path.size() - 1);
    }
}
```
思路：回溯法，尝试每一种情况，符合条件则记入结果。去重方法：如果下一位置数与当前数相等，则本次循环中跳过，因为进入下一层递归还会再计算一次。

41.First Missing Positive Hard

Given an unsorted integer array, find the first missing positive integer.
For example,
Given [1,2,0] return 3,
and [3,4,-1,1] return 2.
Your algorithm should run in *O*(*n*) time and uses constant space.

```
public int firstMissingPositive(int[] nums) {
    int i = 0;
```

```
    while (i < nums.length) {
        if (nums[i] == i + 1 || nums[i] <= 0 || nums[i] > nums.length)
            i++;
        else if (nums[nums[i] - 1] != nums[i])
            swap(nums, i, nums[i] - 1);
        else
            i++;
    }
    i = 0;
    while (i < nums.length && nums[i] == i + 1)
        i++;
    return i + 1;
}

private void swap(int[] A, int i, int j) {
    int temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
```

思路：两遍法。第一遍，nums[index] == index + 1 的和超出范围的（<=0，>数组长度）略过；值不在相应位置，且与相应位置值不同时置换；值不在相应位置但是相应位置已经有其值，则指针直接前移（第三点是性能优化，也可以不管三七二十一置换）。第二遍返回第一个不在相应位置的正数。

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.

For example,

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

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

```
public int trap(int[] A) {
    if (A.length < 3)
        return 0;
    int ans = 0;
    int l = 0, r = A.length - 1;
    while (l < r && A[l] <= A[l + 1])
        l++;
    while (l < r && A[r] <= A[r - 1])
        r--;
    while (l < r) {
        int left = A[l];
        int right = A[r];
        if (left <= right)
            while (l < r && left >= A[++l])
                ans += left - A[l];
        else
            while (l < r && A[--r] <= right)
                ans += right - A[r];
    }
    return ans;
}
```

思路：夹逼法。两边先去除比中间小的。然后两边哪边小哪边就往中间移动，因为另一侧高，水位不会高于较小的一侧。移动中碰到更大的则停止当前轮，碰到更小的则添加相应的值到结果中，左边的就是左边起始点的值-当前值，右边的是右边起始点的值-当前值。

43.Multiply Strings Medium

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

Note:

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

```
public String multiply(String num1, String num2) {
    int m = num1.length(), n = num2.length();
    int[] pos = new int[m + n];
    for (int i = m - 1; i >= 0; i--)
        for (int j = n - 1; j >= 0; j--) {
            int mul = (num1.charAt(i) - '0') * (num2.charAt(j) - '0');
            int p1 = i + j, p2 = i + j + 1;
            int sum = mul + pos[p2];
            pos[p1] += sum / 10;
            pos[p2] = (sum) % 10;
        }
    StringBuilder sb = new StringBuilder();
    for (int p : pos)
        if (!(sb.length() == 0 && p == 0))
            sb.append(p);
    return sb.length() == 0 ? "0" : sb.toString();
}
```

思路：模仿竖式法，逐位计算。结果位数为 m+n，其中每个结果所在索引为(i+j+1)，进位到(i+j)。复杂度 O(mn)

44.Wildcard Matching Hard

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

'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).
The matching should cover the **entire** input string (not partial).
The function prototype should be:
bool isMatch(const char *s, const char *p)
Some examples:
isMatch("aa","a") ? false
isMatch("aa","aa") ? true
isMatch("aaa","aa") ? false
isMatch("aa", "*") ? true
isMatch("aa", "a*") ? true
isMatch("ab", "?*") ? true
isMatch("aab", "c*a*b") ? false

```
public boolean isMatch(String s, String p) {
    int m = s.length(), n = p.length();
    boolean[][] dp = new boolean[m + 1][n + 1];
    dp[0][0] = true;
    for (int j = 1; j <= n; j++) {
        if (p.charAt(j - 1) == '*')
            dp[0][j] = true;
```

```
            else
                break;
        }
        for (int i = 1; i <= m; i++)
            for (int j = 1; j <= n; j++)
                if (p.charAt(j - 1) != '*')
                    dp[i][j] = dp[i - 1][j - 1]
                            && (s.charAt(i - 1) == p.charAt(j - 1)
                            || p.charAt(j - 1) == '?');
                else
                    dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
        return dp[m][n];
    }
```

思路：动态规划。推理条件：如果 p 是*，则与 p 或 s 上一位置结果相同；如果不是*，则看当前字符是否 match。

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.
For example:
Given array A = [2,3,1,1,4]
The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)
Note:
You can assume that you can always reach the last index.

```
public int jump(int[] nums) {
    int step = 0;
    int curMaxP = 0;
    int farthest = 0;
    if (nums.length == 1)
        return step;
    for (int i = 0; i < nums.length;) {
        if (nums[i] + i >= nums.length - 1)
            return step + 1;
        for (int j = i; j < nums.length && j <= nums[i] + i; ++j)
            if (farthest <= nums[j] + j) {
                farthest = nums[j] + j;
                curMaxP = j;
            }
        ++step;
        i = curMaxP;
    }
    return -1;
}
```

思路：步步为营，每次跳到下一步可以跳最远的位置，因为别的位置都比这个位置在下一步的选择少。

46.Permutations Medium

Given a collection of **distinct** numbers, return all possible permutations.
For example,
[1,2,3] have the following permutations:

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

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

```
public List<List<Integer>> permute(int[] nums) {
    List<List<Integer>> ans = new ArrayList<List<Integer>>();
    if (nums == null || nums.length == 0)
        return ans;
    backtrack(nums, new boolean[nums.length], new ArrayList<>(), ans);
    return ans;
}

private static void backtrack(int[] nums, boolean[] visited, List<Integer> cl, List<List<Integer>>
ans) {
    if (cl.size() == nums.length) {
        ans.add(new ArrayList<>(cl));
        return;
    }
    for (int i = 0; i < nums.length; ++i) {
        if (visited[i])
            continue;
        cl.add(nums[i]);
        visited[i] = true;
        backtrack(nums, visited, cl, ans);
        cl.remove(cl.size() - 1);
        visited[i] = false;
    }
}
```

思路：回溯法。O(n^n)

47.Permutations II Medium

Given a collection of numbers that might contain duplicates, return all possible unique permutations.
For example,
[1,1,2] have the following unique permutations:

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

```
public List<List<Integer>> permuteUnique(int[] nums) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, new boolean[nums.length]);
    return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int[] nums, boolean[]
used) {
    if (tempList.size() == nums.length)
        list.add(new ArrayList<>(tempList));
    else
        for (int i = 0; i < nums.length; i++) {
            if (used[i] || i > 0 && nums[i - 1] == nums[i] && !used[i - 1])
                continue;
            tempList.add(nums[i]);
            used[i] = true;
            backtrack(list, tempList, nums, used);
            used[i] = false;
            tempList.remove(tempList.size() - 1);
        }
}
```

思路：回溯法。去重方法：使用数组记录某元素是否已经使用，已使用则重复；如果当前元素与前元素相等，如果前面元素未被使用，则此元素也不应使用，即相同元素序列只使用第一个元素作为开始，这样可以避免相同元素造成的重复。O(n^n)

48.Rotate Image Medium

You are given an *n* x *n* 2D matrix representing an image.
Rotate the image by 90 degrees (clockwise).
Follow up:
Could you do this in-place?

```
public void rotate(int[][] matrix) {
    int n = matrix.length;
    int first, last, os, temp;
    for (int i = 0; i < n / 2; ++i) {
        first = i;
        last = n - i - 1;
        for (int j = first; j < last; ++j) {
            os = j - i;
            temp = matrix[first][j];
            matrix[first][j] = matrix[last - os][first];
            matrix[last - os][first] = matrix[last][last - os];
            matrix[last][last - os] = matrix[j][last];
            matrix[j][last] = temp;
        }
    }
}
```

思路：四边从外向里逐个元素移动，每次移动一个元素，完成一次循环（圈）后，行和列都减少 2（因为首尾都用过了），利用 Offset（os）达到此目的。最多需要移动 n/2 次。

49.Group Anagrams Medium

Given an array of strings, group anagrams together.
For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],
Return:

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

Note: All inputs will be in lower-case.

```
public List<List<String>> groupAnagrams(String[] strs) {
    List<List<String>> r = new ArrayList<List<String>>();
    Map<String, List<String>> result = new HashMap<String, List<String>>();
    for (String str : strs) {
        char[] strC = str.toCharArray();
        Arrays.sort(strC);
        String key = new String(strC);
        List<String> l = result.getOrDefault(key, new ArrayList<String>());
        l.add(str);
        result.put(key, l);
    }
    r.addAll(result.values());
    return r;
}
```

思路：先把每一个词转成 char[]排序，然后利用词典把 Anagram 都放在同一 List 中。

50.Pow(x, n) Medium

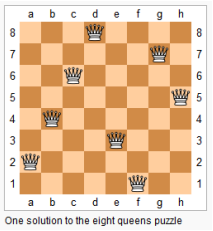
Implement pow(*x*, *n*).

```
public double myPow(double x, int n) {
    if (x == 0)
        return 0;
    long m = (long) n;
    if (n == 0)
        return 1;
    if (n < 0) {
        m = -m;
        x = 1 / x;
    }
    return (m % 2 == 0) ? myPow(x * x, (int) (m / 2)) : x * myPow(x * x, (int) (m / 2));
}
```

思路：每次自相乘（2 或 3 次方根据指数是否是 2 的倍数），同时指数/2，如果 n 小于 0，则将 n 转正，且 x 变为 1/x。结束条件：x=0 或 n=0。

51.N-Queens Hard

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



One solution to the eight queens puzzle

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

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

For example,

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

```
[[".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],
 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]]
```

```
public List<List<String>> solveNQueens(int n) {
    List<List<String>> r = new ArrayList<List<String>>();
    boolean[] col = new boolean[n];
    boolean[] lr = new boolean[n * 2];
    boolean[] rl = new boolean[n * 2];
    backtrack(0, n, col, lr, rl, r, new ArrayList<String>());
    return r;
}

private static void backtrack(int row, int n, boolean[] col, boolean[] lr, boolean[] rl,
List<List<String>> r,
        List<String> l) {
    if (row == n) {
```

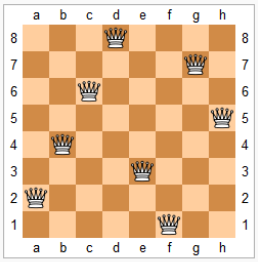
```
            r.add(new ArrayList<String>(l));
        }
        if (row == n)
            return;
        for (int i = 0; i < n; ++i) {
            int lrp = row - i + n;
            int rlp = n * 2 - i - row - 1;
            if (col[i] || lr[lrp] || rl[rlp])
                continue;
            StringBuilder sb = new StringBuilder();
            for (int i1 = 0; i1 < n; ++i1) {
                if (i == i1) {
                    sb.append("Q");
                    continue;
                }
                sb.append(".");
            }
            l.add(sb.toString());
            col[i] = true;
            lr[lrp] = true;
            rl[rlp] = true;
            backtrack(row + 1, n, col, lr, rl, r, l);
            l.remove(l.size() - 1);
            col[i] = false;
            lr[lrp] = false;
            rl[rlp] = false;
        }
    }
```

思路：回溯法。逐行（逐位）尝试，每次置“Q”后，将其所对应的列（行已经处理了）、两个纵行（row-i+n, n*2-i-row-1）设置为已访问。O((n^2)^2)

52.N-Queens II Hard

Follow up for N-Queens problem.

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



One solution to the eight queens puzzle

```
int res = 0;

public int totalNQueens(int n) {
    int[] col = new int[n];
    int[] diag1 = new int[2 * n];
    int[] diag2 = new int[2 * n];

    dfs(0, n, col, diag1, diag2);
    return res;

}

public void dfs(int i, int n, int[] col, int[] diag1, int[] diag2) {
    if (i == n) {
        res++;
```

```
            return;
        }

        for (int j = 0; j < n; j++) {
            if (col[j] == 0 && diag1[j-i+n] == 0 && diag2[n*2-i-j-1] == 0) {
                col[j] = 1;
                diag1[j-i+n] = 1;
                diag2[n*2-i-j-1] = 1;
                dfs(i + 1, n, col, diag1, diag2);
                col[j] = 0;
                diag1[j-i+n] = 0;
                diag2[n*2-i-j-1] = 0;
            }

        }
    }
```

思路：与上一题相同，统计所有解的数量。

53.Maximum Subarray Easy

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.
For example, given the array [-2,1,-3,4,-1,2,1,-5,4],
the contiguous subarray [4,-1,2,1] has the largest sum = 6.
click to show more practice.

More practice:

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

```
// Get the ID of the bucket from element value x and bucket width w
public int maxSubArray(int[] nums) {
    int max = Integer.MIN_VALUE;
    int sum = 0;
    for (int i = 0; i < nums.length; ++i) {
        int cur = nums[i];
        sum += cur;
        max = Math.max(max, sum);
        if (sum < 0)
            sum = 0;
    }
    return max;
}
```

思路 1：类似股票题目，逐一求和，如果小于 0 则再从 0 开始。O(n)

```
private class ArrayContext {
    int max;
    int lMax;
    int rMax;
    int sum;
}

public ArrayContext getArrayContext(int[] nums, int l, int r) {
    ArrayContext ctx = new ArrayContext();
    if (l == r) {
        ctx.max = nums[l];
        ctx.lMax = nums[l];
```

```
            ctx.rMax = nums[l];
            ctx.sum = nums[l];
        } else {
            int m = (l + r) / 2;
            ArrayContext lCtx = getArrayContext(nums, l, m);
            ArrayContext rCtx = getArrayContext(nums, m + 1, r);
            ctx.max = Math.max(Math.max(lCtx.max, rCtx.max),
                    lCtx.rMax + rCtx.lMax);
            ctx.lMax = Math.max(lCtx.lMax, lCtx.sum + rCtx.lMax);
            ctx.rMax = Math.max(rCtx.rMax, rCtx.sum + lCtx.rMax);
            ctx.sum = lCtx.sum + rCtx.sum;
        }
        return ctx;
    }

    public int maxSubArray(int[] nums) {
        if (nums.length == 0)
            return 0;
        ArrayContext ctx = getArrayContext(nums, 0, nums.length - 1);
        return ctx.max;
    }
```

思路 2：二分法。结束条件：全部指向同一元素。每次把数组分成两部分，分别求左、右和当前的 max，再取其中最大者。

54.Spiral Matrix Medium

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

For example,

Given the following matrix:

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

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

```
public List<Integer> spiralOrder(int[][] matrix) {
    List<Integer> res = new ArrayList<Integer>();
    if (matrix.length == 0)
        return res;
    int rowBegin = 0;
    int rowEnd = matrix.length - 1;
    int colBegin = 0;
    int colEnd = matrix[0].length - 1;
    while (rowBegin <= rowEnd && colBegin <= colEnd) {
        for (int j = colBegin; j <= colEnd; j++)
            res.add(matrix[rowBegin][j]);
        rowBegin++;
        for (int j = rowBegin; j <= rowEnd; j++)
            res.add(matrix[j][colEnd]);
        colEnd--;
```

```
        if (rowBegin <= rowEnd)
            for (int j = colEnd; j >= colBegin; j--)
                res.add(matrix[rowEnd][j]);
        rowEnd--;
        if (colBegin <= colEnd)
            for (int j = rowEnd; j >= rowBegin; j--)
                res.add(matrix[j][colBegin]);
        colBegin++;
    }
    return res;
}
```

思路：逐个读取，在达到中心前，先左到右（上），再上到下（右），再右到左（下），再下到上（左），每次注意调整移动边界。

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.
For example:
A = [2,3,1,1,4], return true.
A = [3,2,1,0,4], return false.

```
public boolean canJump(int[] nums) {
    int max = 0;
    for (int i = 0; i < nums.length; i++) {
        if (i > max)
            return false;
        max = Math.max(nums[i] + i, max);
    }
    return true;
}
```

思路：步步为营。每次看所有当前元素所能达到的最远位置，并缓存入 max，如果 i 超过 max 则表明无法达到。

56.Merge Intervals Medium

Given a collection of intervals, merge all overlapping intervals.
For example,
Given [1,3],[2,6],[8,10],[15,18],
return [1,6],[8,10],[15,18].

```
public int[][] merge(int[][] intervals) {
    if (intervals.length < 2)
        return intervals;
    List<int[]> ans = new ArrayList<>();
    int m = intervals.length;
    int[] start = new int[m];
    int[] end = new int[m];
    for (int i = 0; i < m; ++i) {
        start[i] = intervals[i][0];
        end[i] = intervals[i][1];
    }
    Arrays.sort(start);
    Arrays.sort(end);
    for (int i = 0, j = 0; i < m; ++i)
        if (i == m - 1 || start[i + 1] > end[i]) {
            ans.add(new int[] { start[j], end[i] });
            j = i + 1;
        }
```

```
    int n = ans.size();
    int[][] result = new int[n][2];
    ans.toArray(result);
    return result;
}
```

思路 1：最终要得到的是一系列的[start, end]，而中间的 start, end 其实是没有用的。所以把 start 和 end 分别排序，然后把有效的[start, end]对找出来。这里要注意如果有需要 merge 的情况，那么 start 和 end 一定是成对 merge。同时，当前 start 对应的 end 一定是当前 end 或其后的 end。所以，只要逐一查找有效的起始点（即下一 start 比当前 end 大的点）即可，这时当前 start 到当前 end 就是结果中的一对。这时可以证明，此 end 前所有的 interval 都在这个范围内。同时下一 start 也成功被定位。O(nlogn)

```
public int[][] merge(int[][] intervals) {
    if(intervals.length < 2)
        return intervals;
    Arrays.sort(intervals, (a,b)->(a[0] - b[0]));
    List<int[]> ans = new ArrayList<>();
    int start = intervals[0][0];
    int end = intervals[0][1];
    for(int[] interval : intervals) {
        if(interval[0] <= end)
            end = Math.max(end, interval[1]);
        else {
            ans.add(new int[]{start, end});
            start = interval[0];
            end = interval[1];
        }
    }
    ans.add(new int[] {start, end});
    int n = ans.size();
    int[][] result = new int[n][2];
    for(int i = 0;i < n; ++i) {
        result[i] = ans.get(i);
    }
    return result;
}
```

思路 2：把 interval 按首位排序，然后比较前一个的 end 和后一个的 start，如果有交错，则取两者较大的 end 为新 end，继续和下一个比较。直到不交错或达到最后一个元素。O(nlogn)

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:

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

Example 2:

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

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

```
public List<Interval> insert(List<Interval> intervals, Interval newInterval) {
    intervals.add(newInterval);
    int n = intervals.size();
    int[] start = new int[n];
    int[] end = new int[n];
    for (int i = 0; i < n; i++) {
        start[i] = intervals.get(i).start;
        end[i] = intervals.get(i).end;
    }
    Arrays.sort(start);
    Arrays.sort(end);

    List<Interval> res = new ArrayList<Interval>();
```

```
    for (int i = 0, j = 0; i < n; i++) {
        if (i == n - 1 || start[i + 1] > end[i]) {
            res.add(new Interval(start[j], end[i]));
            j = i + 1;
        }
    }
    return res;
}
```

思路 1：同上题，只不过需要先添加一个 Interval。

```
public int[][] insert(int[][] intervals, int[] newInterval) {
    List<int[]> res = new ArrayList<>();
    boolean consumed = false;
    for (int i = 0; i < intervals.length; ++i) {
        int[] il = intervals[i];
        if (consumed)
            res.add(il);
        else if (il[1] < newInterval[0]) {
            res.add(il);
        } else if (il[0] > newInterval[1]) {
            res.add(newInterval);
            consumed = true;
            res.add(il);
        } else {
            newInterval[0] = Math.min(il[0], newInterval[0]);
            newInterval[1] = Math.max(il[1], newInterval[1]);
        }
    }
    if (!consumed)
        res.add(newInterval);
    return res.toArray(new int[res.size()][2]);
}
```

思路 2：原 Interval 数组已经有序，实际只需要找到新 Interval 的插入点或与哪些点合并。遍历 Interval 数组，在新 Interval 前（il[1] < newInterval[0]）和后(consumed || il[0] > newInterval[1])的元素直接添加。如果有交叉，则 newInterval 不断与当前元素结合为新 Interval。当前元素首次在新 Interval 后面时(il[0] > newInterval[1])先添加新 Interval。如果循环结束后新 Interval 还没有使用，直接添加到结果中。

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.

For example,

Given *s* = "Hello World",

return 5.

```
public int lengthOfLastWord(String s) {
    s = s.trim();
    return s.length() - s.lastIndexOf(" ") - 1;
}
```

思路 **1**：通过 API，先去两头空格，然后从最后找空格的 Index，则字符串长度-该 index-1 就是最后一个词的长度。

```
public int lengthOfLastWord(String s) {
    int lenIndex = s.length() - 1;
    int len = 0;
    for (int i = lenIndex; i >= 0 && s.charAt(i) == ' '; i--)
        lenIndex--;
    for (int i = lenIndex; i >= 0 && s.charAt(i) != ' '; i--)
        len++;
    return len;
```

}
思路 **2**：从最后开始移动指针直到该位置元素不为空格。然后继续移动并记录长度直到该位置元素为空格，这时长度就是所求解。

59.Spiral Matrix II Medium

Given an integer *n*, generate a square matrix filled with elements from 1 to n^2 in spiral order.
For example,
Given *n* = 3,
You should return the following matrix:

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

```java
public int[][] generateMatrix(int n) {
    int[][] ret = new int[n][n];
    int left = 0, top = 0;
    int right = n - 1, down = n - 1;
    int count = 1;
    while (left <= right) {
        for (int j = left; j <= right; j++)
            ret[top][j] = count++;
        top++;
        for (int i = top; i <= down; i++)
            ret[i][right] = count++;
        right--;
        for (int j = right; j >= left; j--)
            ret[down][j] = count++;
        down--;
        for (int i = down; i >= top; i--)
            ret[i][left] = count++;
        left++;
    }
    return ret;
}
```

思路：与 55 题类似，逐一填入即可，从左到右（上），从上到下（右），从下到左（下），从下到上（左），每次注意移动边界。

60.Permutation Sequence Medium

The set [1,2,3,…,*n*] contains a total of *n*! unique permutations.
By listing and labeling all of the permutations in order,
We get the following sequence (ie, for *n* = 3):

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

Given *n* and *k*, return the k_{th} permutation sequence.
Note: Given *n* will be between 1 and 9 inclusive.

```java
public String getPermutation(int n, int k) {
    StringBuilder sb = new StringBuilder();
    List<Integer> numbers = new ArrayList<Integer>();
    int[] factorial = new int[n + 1];
    factorial[0] = 1;
```

```
    int sum = 1;
    for (int i = 1; i <= n; ++i) {
        sum *= i;
        factorial[i] = sum;
        numbers.add(i);
    }
    k--;
    for (int i = 1; i <= n; ++i) {
        int index = k / factorial[n - i];
        sb.append(numbers.get(index));
        numbers.remove(index);
        k -= index * factorial[n - i];
    }
    return sb.toString();
}
```

思路 1（略）：回溯法。按升序逐一列举所有可能，直到碰到第 k 个可能。
思路 2：根据全排列公式：f(n)=n!。到第几位就有多少种可能。先把可能排列的数量存入一个数组（factorial），如果 k / factorial[n - i]=0，说明当前所有数的全排列可能性大于 k，则当前值直接放入结果并从候选项中删除（O(n)），即这个数不参与排列，因为它一旦参与，则为 factorial[n - i]个。剩下数全排列的可能为(n-1)!。当 k / factorial[n - i]不是 0 时（注意 i，即每次要去除一个位的可能性），则消耗这么多种可能，从结果中摘除即可。重复上面过程直到使用完所有数。$O(n^2)$
消耗原理：因为可能性为阶乘，第 4 位时，可能性为 1x2x3x4，即 24 种，假设找第 21 种可能，则第一位必为 4（index=3），这时 index 算法为 k/factorial(3)=3，k=[7-23]的情况，都要以 4 开头，因为 4*6=24。最初时 k-1，就是为了避免出现越界，而且本题解法中数组是从 1 开始的。

61.Rotate List Medium

Given a list, rotate the list to the right by *k* places, where *k* is non-negative.
For example:
Given 1->2->3->4->5->NULL and *k* = 2,
return 4->5->1->2->3->NULL.

```
public ListNode rotateRight(ListNode head, int k) {
    if (head == null || head.next == null)
        return head;
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode fast = dummy, slow = dummy;
    int n;
    for (n = 0; fast.next != null; n++)
        fast = fast.next;
    for (int j = n - k % n; j > 0; j--)
        slow = slow.next;
    fast.next = dummy.next;
    dummy.next = slow.next;
    slow.next = null;
    return dummy.next;
}
```

思路：s 计算数组元素长度 n，然后新指针指到 n-k%n-1 位置，也 s 就是新的头。让链表首尾相连，并让其从新的尾部断开，返回新头。注意：模可以避免出界。

62.Unique Paths Medium

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).
How many possible unique paths are there?

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

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

```
public int uniquePaths(int m, int n) {
    int table[][] = new int[m][n];
    for (int i = 0; i < m; i++)
        table[i][0] = 1;
    for (int i = 1; i < n; i++)
        table[0][i] = 1;
    for (int row = 1; row < m; row++)
        for (int column = 1; column < n; column++)
            table[row][column] = table[row - 1][column] + table[row][column - 1];
    return table[m - 1][n - 1];
}
```

思路：动态规划，如果沿某边走则只有一种可能（初始化），然后内侧逐 1 添加可能性，当前可能性是上一行和上一列相应位置的和。

63.Unique Paths II Medium

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

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

The total number of unique paths is 2.

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

```
public int uniquePathsWithObstacles(int[][] obstacleGrid) {
    int m = obstacleGrid.length;
    int n = obstacleGrid[0].length;
    int[][] dp = new int[m][n];
    if (obstacleGrid[0][0] == 1)
        dp[0][0] = 0;
    else
        dp[0][0] = 1;
    for (int i = 1; i < m; ++i)
        if (obstacleGrid[i][0] == 1)
            dp[i][0] = 0;
        else
            dp[i][0] = dp[i - 1][0];

    for (int i = 1; i < n; ++i)
        if (obstacleGrid[0][i] == 1)
            dp[0][i] = 0;
        else
            dp[0][i] = dp[0][i - 1];
    for (int i = 1; i < m; ++i)
        for (int j = 1; j < n; ++j)
            if (obstacleGrid[i][j] == 1)
                dp[i][j] = 0;
            else
                dp[i][j] = dp[i][j - 1] + dp[i - 1][j];
```

```
    return dp[m - 1][n - 1];
}
```
思路：动态规划。与上题解法类似，多一点碰障碍置零。

64.Minimum Path Sum Medium

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

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

```
public int minPathSum(int[][] grid) {
    int m = grid.length, n = grid[0].length;
    int[][] dp = new int[m][n];
    dp[0][0] = grid[0][0];
    for(int i = 1;i < m; ++i)
        dp[i][0] = dp[i-1][0] + grid[i][0];
    for(int i = 1;i < n; ++i)
        dp[0][i] = dp[0][i-1] + grid[0][i];
    for(int i = 1;i < m; ++i)
        for(int j = 1;j < n; ++j)
            dp[i][j] = Math.min(dp[i][j-1], dp[i-1][j]) + grid[i][j];
    return dp[m-1][n-1];
}
```
思路：动态规划。边界上只能直接加上一位置数值到当前位置，非边界的则取上行或列中较小的加到当前数值中，最终位置数值即最小。

65.Valid Number Hard

Validate if a given string is numeric.

Some examples:

"0" => true

" 0.1 " => true

"abc" => false

"1 a" => false

"2e10" => true

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

```
public boolean isNumber(String s) {
    s = s.trim().toLowerCase();
    int n = s.length(), sign = 0;
    boolean hasNum = false, hasDot = false, hasE = false;
    char[] cs = s.toCharArray();
    for (int i = 0; i < n; ++i) {
        char c = cs[i];
        if (!isValid(c))
            return false;
        if (c == '+' || c == '-')
            if (i == n - 1 || sign == 2 || hasNum || (i > 0 && cs[i - 1] != 'e'))
                return false;
            else
                sign++;
        else if (c == '.')
            if (hasE || hasDot || (i == n - 1 && !hasNum))
                return false;
            else
                hasDot = true;
        else if (c == 'e') {
            if (!hasNum || hasE || i == n - 1)
```

```
                return false;
            hasE = true;
            hasNum = false;
        } else
            hasNum = true;

    }
    return hasNum;
}

boolean isValid(char c) {
    return c == '.' || c == '+' || c == '-' || c == 'e' || c >= '0' && c <= '9';
}
```

思路：先问清楚可能出现的字符集，本题中只会出现“.”，“+”，“-”，“e”和 0 到 9。然后根据数字的常识，标记每种情况出现与否，有不应该出现在前的字符出现，则返回假，全部通过返回真。O(n)

66.Plus One Easy

Given a non-negative integer represented as a **non-empty** array of digits, plus one to the integer.
You may assume the integer do not contain any leading zero, except the number 0 itself.
The digits are stored such that the most significant digit is at the head of the list.

```
public int[] plusOne(int[] digits) {
    int carry = 1, n = digits.length;
    for (int i = n - 1; i >= 0; --i) {
        digits[i] += carry;
        carry = digits[i] / 10;
        digits[i] %= 10;
        if (carry == 0)
            return digits;
    }
    int[] ans = new int[n + 1];
    ans[0] = 1;
    return ans;
}
```

思路：就是加法进位。如果不需要进位则结束。如果最后还需要进位，则新数形式必为 1000…。

67.Add Binary Easy

Given two binary strings, return their sum (also a binary string).
For example,
a = "11"
b = "1"
Return "100".

```
public String addBinary(String a, String b) {
    StringBuilder sb = new StringBuilder();
    int i = a.length() - 1, j = b.length() - 1, carry = 0;
    while (i >= 0 || j >= 0) {
        int sum = carry;
        if (j >= 0)
            sum += b.charAt(j--) - '0';
        if (i >= 0)
            sum += a.charAt(i--) - '0';
        sb.append(sum % 2);
        carry = sum / 2;
    }
    if (carry != 0)
        sb.append(carry);
    return sb.reverse().toString();
}
```

思路：小学加法竖式，从低位加到高位（考虑进位）。

68.Text Justification Hard

Given an array of words and a length *L*, format the text such that each line has exactly *L* characters and is fully (left and right) justified.
You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces `' '` when necessary so that each line has exactly *L* characters.
Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.
For the last line of text, it should be left justified and no extra space is inserted between words.
For example,
words: ["This", "is", "an", "example", "of", "text", "justification."]
L: 16.
Return the formatted lines as:

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

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

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

```
public List<String> fullJustify(String[] words, int maxWidth) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < maxWidth; ++i)
        sb.append(" ");
    String pads = sb.toString();
    List<String> res = new ArrayList<String>();
    for (int i = 0, sum = 0, j = 0; i < words.length; i = j) {
        for (j = i + 1, sum = words[i].length(); j < words.length
                && sum + j - i + words[j].length() <= maxWidth; ++j)
            sum += words[j].length();
        StringBuilder line = new StringBuilder();
        int spacePosCount = j - 1 - i;
        int spaceAfterWord = (j == words.length || 0 == spacePosCount) ? 1 : ((maxWidth - sum) /
spacePosCount);
        int extraSpace = (j == words.length) ? 0 : (maxWidth - sum - spaceAfterWord *
spacePosCount);
        for (int k = i; k < j - 1; ++k) {
            line.append(words[k]);
            line.append(pads.substring(0, (k - i < extraSpace) ? spaceAfterWord + 1 :
spaceAfterWord));
        }
        line.append(words[j - 1]);
        if (j == words.length || 0 == spacePosCount) {
            line.append(pads.substring(0, maxWidth - sum - spacePosCount));
        }

        res.add(line.toString());
    }
    return res;
}
```

思路：滑动窗口。按行处理，先从词表中把尽量多的并且可以组成一行的词取出（spacePosCount+1 个词，长度为 sum），这时就可以计算出所需要的总空格数（maxWidth - sum）及其相应位置的数量(spacePosCount 个位置可以放空格，每位置平均放 spaceAfterWord 或 spaceAfterWord+1 个)，当空格不能被 spacePosCount 除尽

时，在前面相应的加（extraSpace-1）多加一空格（即 spaceAfterWord+1 的情况）逐一添加到 StringBuilder 中。

69.Sqrt(x) Easy

Implement int sqrt(int x).
Compute and return the square root of *x*.

```
public int mySqrt(int x) {
    if(x == 0)
        return 0;
    int l = 1, r = x, mid;
    while(l <= r) {
        mid = (l+r)/2;
        if(mid <= x/mid && (mid+1) > x/(mid+1)) //aka. (mid^2) <= x && ((mid+1)^2) > x
            return mid;
        if(mid < x / mid)
            l = mid + 1;
        else
            r = mid - 1;
    }
    return -1;
}
```

思路：二分法寻找平方根，平方根*平方根=目标数。

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.

```
public int climbStairs(int n) {
    if (n <= 1)
        return 1;
    if (n == 2)
        return 2;
    int[] ways = new int[n];
    ways[0] = 1;
    ways[1] = 2;
    for (int i = 2; i < n; ++i)
        ways[i] = ways[i - 1] + ways[i - 2];
    return ways[n - 1];
}
```

思路：动态规划（一维）。第一步可能为 1，第二步可能为 2。后面当前步是前两步可能的和。

71.Simplify Path Medium

Given an absolute path for a file (Unix-style), simplify it.
For example,
path = "/home/", => "/home"
path = "/a/./b/../../c/", => "/c"
Corner Cases:

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

```
public String simplifyPath(String path) {
    int n = path.length();
    String[] p = path.split("/");
```

```
    StringBuilder sb = new StringBuilder();
    LinkedList<String> s = new LinkedList<>();
    for (String pa : p)
        if (pa.equals("..")) {
            if (!s.isEmpty())
                s.removeLast();
        } else if (pa.equals("/") || pa.equals("."))
            continue;
        else if (pa.length() > 0)
            s.add(pa);
    while (!s.isEmpty())
        sb.append("/").append(s.pollFirst());
    if (sb.length() == 0)
        sb.append("/");
    return sb.toString();
}
```

思路：逐一加入 List，再逐一输出到 StringBuilder 中即可，注意每一级间需要/。特殊情况：//，先去除所有//。“.”，不预理睬。“..”向上一级目录（如果存在则在 List 中删除）。

72.Edit Distance Hard

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

You have the following 3 operations permitted on a word:

a) Insert a character

b) Delete a character

c) Replace a character

```
spublic int minDistance(String word1, String word2) {
    char[] wc1 = word1.toCharArray();
    char[] wc2 = word2.toCharArray();
    int m = wc1.length, n = wc2.length;
    if (m == 0 || n == 0)
        return m + n;
    int[][] ans = new int[m + 1][n + 1];
    ans[0][0] = 0;
    for (int i = 1; i <= n; ++i)
        ans[0][i] = ans[0][i - 1] + 1;
    for (int i = 1; i <= m; ++i)
        ans[i][0] = ans[i - 1][0] + 1;
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
            ans[i][j] = Math.min(Math.min(ans[i - 1][j], ans[i][j - 1]),
                    ans[i - 1][j - 1] - (wc1[i - 1] == wc2[j - 1] ? 1 : 0)) + 1;
    return ans[m][n];
}
```

思路：动态规划。递推条件：极端情况，某一个串为空，则另一个串需要进行 n+m（一个是 0）次插入。如果不是都空，那么递推方法：

1. 当前字符不等，当前位置需要进行的操作数为行或列或行+列上一位置+1，即插入、删除或替换操作。
2. 当前字符相等，则无须操作。

73.Set Matrix Zeroes Medium

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

Follow up:

Did you use extra space?

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

A simple improvement uses O(*m* + *n*) space, but still not the best solution.
Could you devise a constant space solution?

```
public void setZeroes(int[][] matrix) {
    boolean fr = false, fc = false;
    for (int i = 0; i < matrix.length; i++)
        for (int j = 0; j < matrix[0].length; j++)
            if (matrix[i][j] == 0) {
                if (i == 0)
                    fr = true;
                if (j == 0)
                    fc = true;
                matrix[0][j] = 0;
                matrix[i][0] = 0;
            }
    for (int i = 1; i < matrix.length; i++)
        for (int j = 1; j < matrix[0].length; j++)
            if (matrix[i][0] == 0 || matrix[0][j] == 0)
                matrix[i][j] = 0;
    if (fr)
        for (int j = 0; j < matrix[0].length; j++)
            matrix[0][j] = 0;
    if (fc)
        for (int i = 0; i < matrix.length; i++)
            matrix[i][0] = 0;
}
```

思路：利用首行、列存储归零信息。对于首行、列是否归零则存在两个 Boolean 变量中。然后依次处理剩余点和首行、列。O(mn)

74.Search a 2D Matrix Medium

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

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

For example,
Consider the following matrix:

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

Given **target** = 3, return true.

```
public boolean searchMatrix(int[][] matrix, int target) {
    int n = matrix.length;
    if (n == 0)
        return false;
    int m = matrix[0].length;
    if (m == 0)
        return false;
    int l = 0, r = m * n - 1;
    while (l != r) {
        int mid = (l + r - 1) >> 1;
        if (matrix[mid / m][mid % m] < target)
            l = mid + 1;
        else
            r = mid;
    }
    return matrix[r / m][r % m] == target;
}
```

思路：变形的二分查找，模拟二分查找，只不过左右边界可能是落在矩阵中而不是一个数组中。

75.Sort Colors Medium

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.
Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
Note:
You are not suppose to use the library's sort function for this problem.
Follow up:
A rather straight forward solution is a two-pass algorithm using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.
Could you come up with an one-pass algorithm using only constant space?

```
public void sortColors(int[] nums) {
    int r = 0;
    int w = 0;
    for (int i = 0; i < nums.length; ++i) {
        if (0 == nums[i])
            ++r;
        if (1 == nums[i])
            ++w;
    }
    for (int i = 0; i < r; ++i)
        nums[i] = 0;
    for (int i = r; i < r + w; ++i)
        nums[i] = 1;
    for (int i = r + w; i < nums.length; ++i)
        nums[i] = 2;
}
```

思路：因共三种颜色，所以统计其中两种颜色填好前半段后，剩下部分全部置为第三种颜色即可。

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).
For example,
S = "ADOBECODEBANC"
T = "ABC"
Minimum window is "BANC".
Note:
If there is no such window in S that covers all characters in T, return the empty string "".
If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

```
public String minWindow(String s, String t) {
    int[] map = new int[256];
    char[] sc = s.toCharArray();
    int d = Integer.MAX_VALUE, l = 0, r = 0, count = 0, i = 0, j = 0;
    for(char c : t.toCharArray()) {
        ++map[c];
        ++count;
    }
    while(r < sc.length) {
        if(map[sc[r++]]-- > 0)
            --count;
        while(count == 0) {
            if(r-l < d) {
                d = r-l;
                i = l;
                j = r;
```

```
            }
            d = Math.min(r-l+1, d);
            if(map[sc[l++]]++ == 0)
                ++count;
        }
    }
    return s.substring(i,j);
}
```

思路：滑动窗口（Sliding Window），先找到包含 T 的第一个字串，然后把左边界缩小到最小但包含 T 的状态，这是第一个 Candidate。然后左边界前移，重复第一步，找到所有 Candidates。最终找到最小值并返回。
O(n)

77.Combinations Medium

Given two integers *n* and *k*, return all possible combinations of *k* numbers out of 1 ... *n*.
For example,
If *n* = 4 and *k* = 2, a solution is:

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

```
public List<List<Integer>> combine(int n, int k) {
    List<List<Integer>> combs = new ArrayList<List<Integer>>();
    if (k > n)
        return combs;
    combine(combs, new ArrayList<Integer>(), 1, n, k);
    return combs;
}

public static void combine(List<List<Integer>> combs, List<Integer> comb, int start, int n, int k)
{
    if (k < 0)
        return;
    if (k == 0) {
        combs.add(new ArrayList<Integer>(comb));
        return;
    }
    for (int i = start; i <= n - k + 1; i++) {
        comb.add(i);
        combine(combs, comb, i + 1, n, k - 1);
        comb.remove(comb.size() - 1);
    }
}
```

思路：回溯法。通过剪枝优化性能（n-k+1）。

78.Subsets Medium

Given a set of **distinct** integers, *nums*, return all possible subsets.
Note: The solution set must not contain duplicate subsets.
For example,
If ***nums*** = [1,2,3], a solution is:

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

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

```
public List<List<Integer>> subsets(int[] nums) {
    List<List<Integer>> list = new ArrayList<>();
    backtrack(list, new ArrayList<>(), nums, 0);
    return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int[] nums, int start) {
    list.add(new ArrayList<>(tempList));
    for (int i = start; i < nums.length; i++) {
        tempList.add(nums[i]);
        backtrack(list, tempList, nums, i + 1);
        tempList.remove(tempList.size() - 1);
    }
}
```

思路：回溯查找所有可能情况。O(n^n)

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.
For example,
Given **board** =

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

word = "ABCCED", -> returns true,
word = "SEE", -> returns true,
word = "ABCB", -> returns false.

```
public boolean exist(char[][] board, String word) {
    if (word == null || word.length() == 0)
        return true;
    char[] chs = word.toCharArray();
    for (int i = 0; i < board.length; i++)
        for (int j = 0; j < board[0].length; j++)
            if (dfs(board, chs, 0, i, j))
                return true;
    return false;
}

private boolean dfs(char[][] board, char[] words, int idx, int x, int y) {
    if (idx == words.length)
        return true;
    if (x < 0 || x == board.length || y < 0 || y == board[0].length)
        return false;
    if (board[x][y] != words[idx])
        return false;
    board[x][y] ^= 256;
    boolean exist = dfs(board, words, idx + 1, x, y + 1) || dfs(board, words, idx + 1, x, y - 1)
            || dfs(board, words, idx + 1, x + 1, y) || dfs(board, words, idx + 1, x - 1, y);
```

```
    board[x][y] ^= 256;
    return exist;
}
```
思路：深度优先检索（DFS）四向查找结果，使用过的点记录为^256，完成该轮再^256 恢复。O(m^2*n^2)

80.Remove Duplicates from Sorted Array II Medium

Follow up for "Remove Duplicates":
What if duplicates are allowed at most *twice*?
For example,
Given sorted array *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. It doesn't matter what you leave beyond the new length.

```
public int removeDuplicates(int[] nums) {
    int i = 0;
    for (int n : nums)
        if (i < 2 || n > nums[i - 2])
            nums[i++] = n;
    return i;
}
```
思路：因为允许重复两次，所以只要和前面间隔一位的数字比，大了的话加入，小了的话跳过。

81.Search in Rotated Sorted Array II Medium

Follow up for "Search in Rotated Sorted Array":
What if *duplicates* are allowed?
Would this affect the run-time complexity? How and why?
Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.
(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
Write a function to determine if a given target is in the array.
The array may contain duplicates.

```
public boolean search(int[] nums, int target) {
    int start = 0, end = nums.length - 1, mid = -1;
    while (start <= end) {
        mid = (start + end) / 2;
        if (nums[mid] == target)
            return true;
        if (nums[mid] < nums[end] || nums[mid] < nums[start])
            if (target > nums[mid] && target <= nums[end])
                start = mid + 1;
            else
                end = mid - 1;
        else if (nums[mid] > nums[start] || nums[mid] > nums[end])
            if (target < nums[mid] && target >= nums[start])
                end = mid - 1;
            else
                start = mid + 1;
        else
            end--;
    }
    return false;
}
```
思路：与 33 题核心思路一致，主要还是二分查找中判断哪边已经排序，多一种可能就是中间数和两头都相等，那么任意一边往中间移一步。最坏情况下绝大多数数字都是同一值，故其复杂度可达 O(n)，平均 O(logn)。注意在判断左或右有序时使用或条件判断大于或小于首尾元素，或成立时是中间元素与某一侧元素相等的情况。

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.

For example,

Given `1->2->3->3->4->4->5`, return `1->2->5`.

Given `1->1->1->2->3`, return `2->3`.

```
public ListNode deleteDuplicates(ListNode head) {
    if (head == null)
        return null;
    ListNode dummyNode = new ListNode(0);
    dummyNode.next = head;
    ListNode pre = dummyNode;
    ListNode cur = head;
    while (cur != null) {
        while (cur.next != null && cur.val == cur.next.val)
            cur = cur.next;
        if (pre.next == cur)
            pre = pre.next;
        else
            pre.next = cur.next;
        cur = cur.next;
    }
    return dummyNode.next;
}
```

思路：双指针法，一个指针（pre）先指向头前面，另一个指针作为游标，如果遇到值相等的则直接删除后面元素，不相等或到结尾，再判断头元素和当前元素是否相等，相等则需要删除头元素，不等则 pre 指向下一个元素，重复上面过程。另：递归，简单很多，略。

83.Remove Duplicates from Sorted List Easy

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

For example,

Given `1->1->2`, return `1->2`.

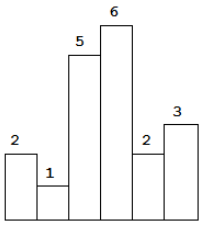
Given `1->1->2->3->3`, return `1->2->3`.

```
public ListNode deleteDuplicates(ListNode head) {
    ListNode current = head;
    while (current != null && current.next != null) {
        if (current.val == current.next.val)
            current.next = current.next.next;
        else
            current = current.next;
    }
    return head;
}
```

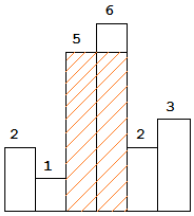
思路：与上题方法一样，只是不需要对头元素进行处理。

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.

For example,

Given heights = [2,1,5,6,2,3],

return 10.

```
public int largestRectangleArea(int[] heights) {
    if (heights == null || heights.length == 0)
        return 0;
    int[] lMostHiPos = new int[heights.length];
    int[] rMostHiPos = new int[heights.length];
    rMostHiPos[heights.length - 1] = heights.length;
    lMostHiPos[0] = -1;
    for (int i = 1; i < heights.length; i++) {
        int p = i - 1;
        while (p >= 0 && heights[p] >= heights[i])
            p = lMostHiPos[p];
        lMostHiPos[i] = p;
    }
    for (int i = heights.length - 2; i >= 0; i--) {
        int p = i + 1;
        while (p < heights.length && heights[p] >= heights[i])
            p = rMostHiPos[p];
        rMostHiPos[i] = p;
    }
    int maxArea = 0;
    for (int i = 0; i < heights.length; i++)
        maxArea = Math.max(maxArea, heights[i] * (rMostHiPos[i] - lMostHiPos[i] - 1));
    return maxArea;
}
```

思路 1：联合查找。对于每一个柱子，其极值就是它和它两边相连接的所有的>=它高度的柱子形成的长方形。所以只要找到每一个柱子相连的最远的不低于它的柱子坐标（使用联合查找），然后就可以求出当前柱子极值，所有极值中求最大值。求左右边界时，每一位的比较次数是 O(1)，所以整体复杂度 O(n)

```
public int largestRectangleArea(int[] heights) {
    Stack<Integer> stack = new Stack<>();
    stack.push(-1);
    int maxarea = 0;
    for (int i = 0; i < heights.length; ++i) {
        while (stack.peek() != -1 && heights[stack.peek()] >= heights[i])
            maxarea = Math.max(maxarea, heights[stack.pop()] * (i - stack.peek() - 1));
        stack.push(i);
    }
    while (stack.peek() != -1)
        maxarea = Math.max(maxarea, heights[stack.pop()] * (heights.length - stack.peek() - 1));
    return maxarea;
```

```
}
```
思路 2：利用栈缓存较高的柱子位置，碰到更小的柱子，则可能是一个极值，不断弹出所以可能，取极大值，直到栈内柱子较矮。最后栈内如果还有元素（-1 为占位符），则不断弹出求最值。所有元素都出入栈一次，O(n)。
```
public int calculateArea(int[] heights, int start, int end) {
    if (start > end)
        return 0;
    int minindex = start;
    for (int i = start; i <= end; i++)
        if (heights[minindex] > heights[i])
            minindex = i;
    return Math.max(heights[minindex] * (end - start + 1),
            Math.max(calculateArea(heights, start, minindex - 1), calculateArea(heights, minindex
+ 1, end)));
}

public int largestRectangleArea(int[] heights) {
    return calculateArea(heights, 0, heights.length - 1);
}
```
思路 3：分治法：数组中最矮的柱子*数组长度是一个极值，最矮柱子两边的子集也分别求极极，最终取最大值。因为是二分，所以经过 logn 次分隔，每次找最小值需要耗时 n（其实比 n 小），整体复杂度 O(nlogn)。

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.
For example, given the following matrix:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Return 6.
```
public int maximalRectangle(char[][] matrix) {
    if (matrix.length == 0 || matrix[0].length == 0)
        return 0;
    int m = matrix.length, n = matrix[0].length, max = 0, cl, cr;
    int[] l = new int[n], r = new int[n], h = new int[n];
    Arrays.fill(r, n);
    for (int i = 0; i < m; ++i) {
        cl = 0;
        cr = n;
        for (int j = n - 1; j >= 0; --j)
            if (matrix[i][j] == '1')
                r[j] = Math.min(cr, r[j]);
            else {
                cr = j;
                r[j] = n;
            }
        for (int j = 0; j < n; ++j)
            if (matrix[i][j] == '1') {
                l[j] = Math.max(cl, l[j]);
                h[j] += 1;
                max = Math.max(max, h[j] * (r[j] - l[j]));
            } else {
                cl = j + 1;
                l[j] = 0;
                h[j] = 0;
            }
    }
```

```
    return max;
}
```
思路：动态规划。对于每一个点，如果在长方形中，确定其左右最远位置和高即可。所以逐点判断是否是同一长方形，如果是，则使用相同的位置和高，如果不是则重置。

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

```
public ListNode partition(ListNode head, int x) {
    ListNode dummy1 = new ListNode(0), dummy2 = new ListNode(0);
    ListNode curr1 = dummy1, curr2 = dummy2;
    while (head != null) {
        if (head.val < x) {
            curr1.next = head;
            curr1 = head;
        } else {
            curr2.next = head;
            curr2 = head;
        }
        head = head.next;
    }
    curr2.next = null;
    curr1.next = dummy2.next;
    return dummy1.next;
}
```
思路：使用两个链表缓存结果，小于给定值的放一起，大于等于的放一起，最后连接即可。

87.Scramble String Hard

Given a string *s1*, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.
Below is one possible representation of *s1* = "great":

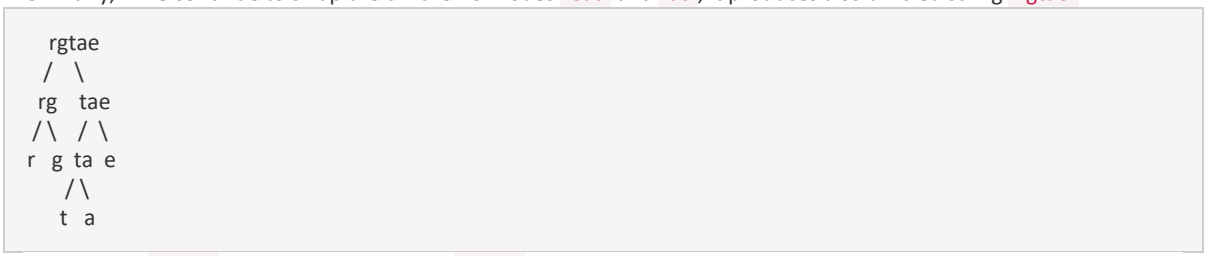
```
  great
 /  \
 gr  eat
/\  / \
g  r e  at
     /\
     a  t
```

To scramble the string, we may choose any non-leaf node and swap its two children.
For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".

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

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

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



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*.

```
public boolean isScramble(String s1, String s2) {
    int n = s1.length();
    if (n == 1)
        return s1.equals(s2);
    int[] map1 = new int[128], map2 = new int[128];
    for (int i = 0, cnt1 = 0, cnt2 = 0; i < n - 1; i++) {
        if (map1[s1.charAt(i)]++ < 0)
            cnt1++;
        if (map1[s2.charAt(i)]-- > 0)
            cnt1++;
        if (cnt1 == i + 1 && isScramble(s1.substring(0, cnt1), s2.substring(0, cnt1))
                && isScramble(s1.substring(cnt1), s2.substring(cnt1)))
            return true;
        if (map2[s1.charAt(i)]++ < 0)
            cnt2++;
        if (map2[s2.charAt(n - 1 - i)]-- > 0)
            cnt2++;
        if (cnt2 == i + 1 && isScramble(s1.substring(0, cnt2), s2.substring(n - cnt2))
                && isScramble(s1.substring(cnt2), s2.substring(0, n - cnt2)))
            return true;
    }
    return false;
}
```

思路：递归求解。结束条件：如果两字串相等则为真，如果两字串中字符数不同，则必为假。暴力计算所有可能的组合，注意左右两个字串可以头尾互换。性能调优：只有当对应的子串所含字符数相等时，才有可能是scramble。

88.Merge Sorted Array Easy

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

Note:

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

```
public void merge(int[] nums1, int m, int[] nums2, int n) {
    for (int i = nums1.length - 1; i >= n; --i)
        nums1[i] = nums1[i - n];
    int k = 0;
    for (int i = n, j = 0; i < m + n || j < n;)
        if (i >= m + n)
            nums1[k++] = nums2[j++];
        else if (j >= n)
            nums1[k++] = nums1[i++];
        else
            nums1[k++] = nums1[i] > nums2[j] ? nums2[j++] : nums1[i++];
}
```

思路：逐一将较小的数存入 num1 即可。O(m+n)

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

```
00 - 0
01 - 1
11 - 3
10 - 2
```

Note:
For a given *n*, a gray code sequence is not uniquely defined.
For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.
For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

```
public List<Integer> grayCode(int n) {
    List<Integer> result = new LinkedList<>();
    for (int i = 0; i < 1 << n; i++)
        result.add(i ^ i >> 1);
    return result;
}
```

思路：逐一生成结果，生成原理如下：

binary num	gray code
0=000	000 = 000 ^ (000>>1)
1=001	001 = 001 ^ (001>>1)
2=010	011 = 010 ^ (010>>1)
3=011	010 = 011 ^ (011>>1)
4=100	100 ^010
...	
7=111	100 = 111 ^ (111>>1)

90.Subsets II Medium

Given a collection of integers that might contain duplicates, ***nums***, return all possible subsets.
Note: The solution set must not contain duplicate subsets.
For example,
If ***nums*** = [1,2,2], a solution is:

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

```
public List<List<Integer>> subsetsWithDup(int[] nums) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, 0);
    return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int[] nums, int start) {
    list.add(new ArrayList<>(tempList));
    for (int i = start; i < nums.length; i++) {
        if (i > start && nums[i] == nums[i - 1])
```

```
            continue;
        tempList.add(nums[i]);
        backtrack(list, tempList, nums, i + 1);
        tempList.remove(tempList.size() - 1);
    }
}
```

思路：与 78 题类似，不过多一步去重。去重方法就是先排序，然后遇到相同的数时跳过。

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 an encoded message containing digits, determine the total number of ways to decode it.
For example,
Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).
The number of ways decoding "12" is 2.

```
public int numDecodings(String s) {
    int len = s.length();
    if (len == 0)
        return 0;
    int[] dp = new int[len + 1];
    dp[0] = 1;
    dp[1] = s.charAt(0) == '0' ? 0 : 1;
    for (int i = 2; i <= len; i++) {
        char c0 = s.charAt(i - 1);
        char c1 = s.charAt(i - 2);
        if (c0 != '0')
            dp[i] += dp[i - 1];
        if (c1 == '1' || (c1 == '2' && c0 <= '6'))
            dp[i] += dp[i - 2];
    }
    return dp[len];
}
```

思路：动态规划。初始化：没有数字时仅一种解，有一个数字时，如果该数字可解（!=0），则其可能性与没有数字时一样，即 1+dp[0]；不可解时 0+dp[0]。

1. 单数字情况：每次看当前数是否为 0，如果不是 0，则有一种可能，即 1*dp[i-1]；如果是 0，则无可能，即 0+dp[i-1]。
2. 相邻两数的组合情况，两数可以组合，仅当前数为 1 时或前数为 2 且后数不大于 6 时。可以组合时，其可能性为 1+dp[i-2]；否则 0+dp[-2]。O(n)

92.Reverse Linked List II Medium

Reverse a linked list from position *m* to *n*. Do it in-place and in one-pass.
For example:
Given 1->2->3->4->5->NULL, *m* = 2 and *n* = 4,
return 1->4->3->2->5->NULL.
Note:
Given *m*, *n* satisfy the following condition:
1 < *m* < *n* < length of list.

```
public ListNode reverseBetween(ListNode head, int m, int n) {
    if (head == null)
        return null;
```

```
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode pre = dummy;
    for (int i = 0; i < m - 1; i++)
        pre = pre.next;
    ListNode start = pre.next;
    ListNode then = start.next;
    for (int i = 0; i < n - m; i++) {
        start.next = then.next;
        then.next = pre.next;
        pre.next = then;
        then = start.next;
    }
    return dummy.next;
}
```

思路：与 25 题类似。不同的是不断地把第二个元素插到第一个元素前面元素（pre）后面。先移动到 m-1 位（头），然后后面元素到 n 依次插入到 m 位后，即完成指定反转。

93.Restore IP Addresses Medium

Given a string containing only digits, restore it by returning all possible valid IP address combinations.
For example:
Given "25525511135",
return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

```
public List<String> restoreIpAddresses(String s) {
    List<String> ret = new ArrayList<>();
    StringBuilder ip = new StringBuilder();
    for (int a = 1; a < 4; ++a)
        for (int b = 1; b < 4; ++b)
            for (int c = 1; c < 4; ++c)
                for (int d = 1; d < 4; ++d)
                    if (a + b + c + d == s.length()) {
                        int n1 = Integer.parseInt(s.substring(0, a));
                        int n2 = Integer.parseInt(
                            s.substring(a, a + b));
                        int n3 = Integer.parseInt(
                            s.substring(a + b, a + b + c));
                        int n4 = Integer.parseInt(
                            s.substring(a + b + c));
                        if (n1 <= 255 && n2 <= 255 && n3 <= 255
                            && n4 <= 255) {
                                ip.append(n1).append('.').append(n2)
                                  .append('.').append(n3).append('.')
                                  .append(n4);
                            if (ip.length() == s.length() + 3)
                                ret.add(ip.toString());
                            ip.delete(0, ip.length());
                        }
                    }
    return ret;
}
```

思路 1：IP 每位长度 0-4，所以逐一尝试看是否能凑足 4 个值在 0-255 间的数，如果满足则记录，首数为零时只能单数成位，所以最后比较成串的长度与原串长度就可以知道是否合理。O(1)
思路 2：回溯法，与思路 1 一样，递归实现。每级只要考虑有 1 个数，2 个数，3 个数的情况，其中大于 1 个数字时首位不能为 0。O(1)

94.Binary Tree Inorder Traversal Medium

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

For example:
Given binary tree [1,null,2,3],

```
 1
 \
  2
 /
 3
```

return [1,3,2].
Note: Recursive solution is trivial, could you do it iteratively?

```
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> r = new ArrayList<Integer>();
    if (root != null)
        inorder(root, r);
    return r;
}

private static void inorder(TreeNode root, List<Integer> r) {
    if (root.left != null)
        inorder(root.left, r);
    r.add(root.val);
    if (root.right != null)
        inorder(root.right, r);
}
```

思路：中序遍历，不输出结果，存到 List 中。

95.Unique Binary Search Trees II Medium

Given an integer *n*, generate all structurally unique **BST's** (binary search trees) that store values 1...*n*.
For example,
Given *n* = 3, your program should return all 5 unique BST's shown below.

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

```
public List<TreeNode> generateTrees(int n) {
    if (n < 1)
        return new ArrayList<TreeNode>();
    return genTrees(1, n);
}

public List<TreeNode> genTrees(int start, int end) {
    List<TreeNode> list = new ArrayList<TreeNode>();
    if (start > end) {
        list.add(null);
        return list;
    }
    if (start == end) {
        list.add(new TreeNode(start));
        return list;
    }
    List<TreeNode> left, right;
    for (int i = start; i <= end; i++) {
        left = genTrees(start, i - 1);
        right = genTrees(i + 1, end);
        for (TreeNode lnode : left)
```

```
            for (TreeNode rnode : right) {
                TreeNode root = new TreeNode(i);
                root.left = lnode;
                root.right = rnode;
                list.add(root);
            }
        }
        return list;
    }
```

思路：回溯法。因结点是顺序排列的，所以头结点两侧分别为左右子树，所以左右子树和当前结点组成的树为答案。递归结束条件：start 与 end 指向同一点，则说明该点为叶子结点；start>end，说明该子树不存在。

96.Unique Binary Search Trees Medium

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

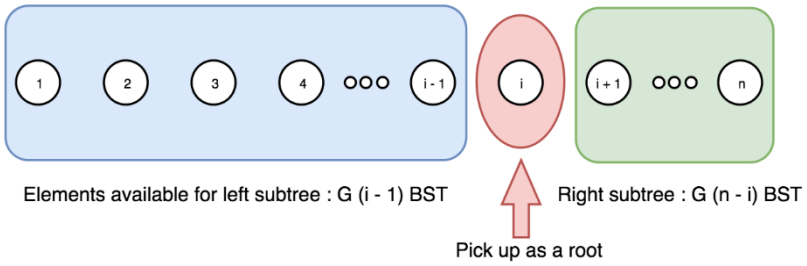
For example,

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

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

```
public int numTrees(int n) {
    if (n <= 1)
        return 1;
    int[] c = new int[n + 1];
    c[0] = c[1] = 1;
    for (int i = 2; i <= n; ++i)
        for (int j = 1; j <= i; ++j)
            c[i] += c[j - 1] * c[i - j];
    return c[n];
}
```

思路：动态规划。推导方程：每一结点（j）作为根的可能性=左面每一结点作为根的数量之和+右面每一结点作为根的数量之和，即 c[n]=c[j-1]*c[n-j]，可以从 n=2 开始推导直到 n=n。详解



97.Interleaving String Hard

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

For example,

Given:

s1 = "aabcc",

s2 = "dbbca",

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

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

```
public boolean isInterleave(String s1, String s2, String s3) {
    int m = s1.length(), n = s2.length(), o = s3.length();
    if (m + n != o)
        return false;
    boolean[][] res = new boolean[m + 1][n + 1];
    res[0][0] = true;
    for (int i = 1; i <= m; ++i)
        res[i][0] = res[i - 1][0] && s1.charAt(i - 1) == s3.charAt(i - 1);
    for (int i = 1; i <= n; ++i)
        res[0][i] = res[0][i - 1] && s2.charAt(i - 1) == s3.charAt(i - 1);
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
            res[i][j] = (res[i - 1][j] && s1.charAt(i - 1) == s3.charAt(i + j - 1)) || (res[i][j -
1] && s2.charAt(j - 1) == s3.charAt(i + j - 1));
    return res[m][n];
}
```

思路：动态规划。推导方程：当前 s2（s1）匹配，则匹配结果与上一步 s1（s2）匹配与否一致。

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:

```
  2
 /\
1  3
```

Binary tree [2,1,3], return true.
Example 2:

```
  1
 /\
2  3
```

Binary tree [1,2,3], return false.

```
public boolean isValidBST(TreeNode root) {
    return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
}

public boolean isValidBST(TreeNode root, long minVal, long maxVal) {
    if (root == null)
        return true;
    if (root.val >= maxVal || root.val <= minVal)
        return false;
    return isValidBST(root.left, minVal, root.val) && isValidBST(root.right, root.val, maxVal);
}
```

思路：根据二叉搜索树的条件，递归查找不符合条件的，如有则不是，遍历完成没有则是。O(n)

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

```
TreeNode firstElement, secondElement, prevElement;

public void recoverTree(TreeNode root) {
    traverse(root);
    int temp = firstElement.val;
    firstElement.val = secondElement.val;
    secondElement.val = temp;
}

private void traverse(TreeNode root) {
    if (root == null)
        return;
    traverse(root.left);
    if (prevElement != null) {
        if (firstElement == null && prevElement.val >= root.val)
            firstElement = prevElement;
        if (firstElement != null && prevElement.val >= root.val)
            secondElement = root;
    }
    prevElement = root;
    traverse(root.right);
}
```

思路：中序遍历。右子树中的结点（root）一定大于当前结点（preElement），找到两个不符合二叉搜索树的元素，并将其置换。

100. Same Tree Easy

Given two binary trees, write a function to check if they are equal or not.
Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

```
public boolean isSameTree(TreeNode p, TreeNode q) {
    if (p == null && q == null)
        return true;
    if (p == null || q == null || p.val != q.val)
        return false;
    return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
}
```

思路：递归求解。每次判断当前结点是否相等：当前两结点都为空则结果为真；某一为空或元素值不同则为假。逐一判断左、右子树是否相等。

101. Symmetric Tree Easy

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).
For example, this binary tree [1,2,2,3,4,4,3] is symmetric:

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

But the following [1,2,2,null,3,null,3] is not:

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

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

```
public boolean isSymmetric(TreeNode root) {
    return root == null || isSymmetricHelp(root.left, root.right);
}

private boolean isSymmetricHelp(TreeNode left, TreeNode right) {
    if (left == null || right == null)
        return left == right;
    if (left.val != right.val)
        return false;
    return isSymmetricHelp(left.left, right.right) && isSymmetricHelp(left.right, right.left);
}
```

思路 **1**：符合条件的树的特点：中心对称两侧结点相等，递归对比即得解。结束条件：对应结点都为空，则为真；对应结点有一个为空或值不相当，则为假。推导方程：左（右）结点的左、右（右、左）孩子应该和右（左）结点的右、左（左、右）孩子相等。

```
public boolean isSymmetric(TreeNode root) {
    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);
    q.add(root);
    while (!q.isEmpty()) {
        TreeNode t1 = q.poll();
        TreeNode t2 = q.poll();
        if (t1 == null && t2 == null) continue;
        if (t1 == null || t2 == null) return false;
        if (t1.val != t2.val) return false;
        q.add(t1.left);
        q.add(t2.right);
        q.add(t1.right);
        q.add(t2.left);
    }
    return true;
}
```

思路 **2**：与思路 1 一致，用队列模拟递归。这里有一个观察是：需要比较的对象总是成对出现，并且后面需要比较的对和当前对都有关系，即左左对右右，左右对右左。

102. Binary Tree Level Order Traversal Medium

Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).
For example:
Given binary tree [3,9,20,null,null,15,7],

```
  3
 / \
9  20
  /  \
 15   7
```

return its level order traversal as:

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

```
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> ans = new ArrayList<List<Integer>>();
    Queue<TreeNode> q = new LinkedList<>(), nq = new LinkedList<>();
    if (root != null)
        q.offer(root);
```

```
    List<Integer> l = new ArrayList<>();
    while (!q.isEmpty()) {
        TreeNode cur = q.poll();
        if (cur.left != null)
            nq.offer(cur.left);
        if (cur.right != null)
            nq.offer(cur.right);
        l.add(cur.val);
        if (q.isEmpty()) {
            ans.add(l);
            l = new ArrayList<Integer>();
            if (!nq.isEmpty()) {
                Queue<TreeNode> temp = q;
                q = nq;
                nq = temp;
            }
        }
    }
    return ans;
}
```

思路：使用两个 Queue（LinkedList）缓存当前行（q）和下一行（nq）的结点。第一行只有根结点，第二行是第二行的结点。把当前行所有元素逐一弹出到 List 的同时，把其子节点全部存入下一行的队列中。当当前行队列为空时，就是一行的结果存入 ans，然后 q、nq 互换直到所有行都被处理。

103. Binary Tree Zigzag Level Order Traversal Medium

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

For example:

Given binary tree [3,9,20,null,null,15,7],

```
  3
 / \
9  20
  / \
 15  7
```

return its zigzag level order traversal as:

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

```
public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
    List<List<Integer>> sol = new ArrayList<>();
    travel(root, sol, 0);
    return sol;
}

private void travel(TreeNode curr, List<List<Integer>> sol, int level) {
    if (curr == null)
        return;
    if (sol.size() <= level) {
        List<Integer> newLevel = new LinkedList<>();
        sol.add(newLevel);
    }
    LinkedList<Integer> collection = (LinkedList<Integer>)sol.get(level);
    if (level % 2 == 0)
        collection.add(curr.val);
    else
```

```
            collection.addFirst(curr.val);
        travel(curr.left, sol, level + 1);
        travel(curr.right, sol, level + 1);
}
```

思路 1：利用一个 List 嵌套数组保存每一层的元素，到相应层则把相应层元素存入对应数组。先序遍历则每层元素都是从左到右的。要成为 ZigZag，只要在双数层上反向插入 List（collection.add(0, curr.val);）即可。
思路 2：同 102，广度优先，每层新建一个 List，双数层时反射添加（可以利用 LinkedList addFirst() API）。所有 List 都添加到 sol List 中

104. Maximum Depth of Binary Tree Easy

Given a binary tree, find its maximum depth.
The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

```
public int maxDepth(TreeNode root) {
    return maxDepth(root, 0);
}

public int maxDepth(TreeNode root, int level) {
    if (root == null)
        return level;
    return Math.max(maxDepth(root.left, level + 1), maxDepth(root.right, level + 1));
}
```

思路：递归求解。递推方程：左面或右面较大者为解。结束条件：结点为空，则为当前结点深度。

105. Construct Binary Tree from Preorder and Inorder Traversal Medium

Given preorder and inorder traversal of a tree, construct the binary tree.
Note:
You may assume that duplicates do not exist in the tree.

```
public TreeNode buildTree(int[] preorder, int[] inorder) {
    return helper(0, 0, inorder.length - 1, preorder, inorder);
}

public TreeNode helper(int preStart, int inStart, int inEnd, int[] preorder, int[] inorder) {
    if (inStart > inEnd)
        return null;
    TreeNode root = new TreeNode(preorder[preStart]);
    int inIndex = 0;
    for (int i = inStart; i <= inEnd; i++)
        if (inorder[i] == root.val)
            inIndex = i;
    root.left = helper(preStart + 1, inStart, inIndex - 1, preorder, inorder);
    root.right = helper(preStart + inIndex - inStart + 1, inIndex + 1, inEnd, preorder, inorder);
    return root;
}
```

思路 1：递归求解。先序遍历每子树第一元素必为树的根结点，到下一根结点前，都是其左子树元素；而中序遍历的根结点前，必为其左子树元素；两者左子树结点数相同。
思路 2（略）：与思路 1 类似，建立一个查找中序结点 Index 的索引提高效率。

```
int pre, in;

public TreeNode buildTree(int[] preorder, int[] inorder) {
    return buildTree(preorder, inorder, Integer.MAX_VALUE);
}

TreeNode buildTree(int[] preorder, int[] inorder, int t) {
    if (in >= inorder.length || t == inorder[in])
        return null;
```

```
    TreeNode root = new TreeNode(preorder[pre++]);
    root.left = buildTree(preorder, inorder, root.val);
    ++in;
    root.right = buildTree(preorder, inorder, t);
    return root;
}
```

思路 3：与思路 1 相似，注意观察会发现同一级的左右根结点间的元素，都在一个区块。而在 pre 数组中碰到 in 的对应元素前，都为其左子树元素，之后为其右子树。故，查找 pre 中根元素对应 in 数组中的位置这一过程可以略去，即可以通过不断移动 pre 指针实现：碰到 in 当前元素时，则为当前级别右子树，碰不到时，为左子树。

106. Construct Binary Tree from Inorder and Postorder Traversal Medium

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

Note:

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

```
int pInorder;

int pPostorder;

public TreeNode buildTree(int[] inorder, int[] postorder) {
    pInorder = inorder.length - 1;
    pPostorder = postorder.length - 1;
    return helper(inorder, postorder, Integer.MAX_VALUE);
}

public TreeNode helper(int[] inorder, int[] postorder, int end) {
    if (pPostorder < 0 || inorder[pInorder] == end) {
        return null;
    }
    TreeNode root = new TreeNode(postorder[pPostorder--]);
    root.right = helper(inorder, postorder, root.val);
    pInorder--;
    root.left = helper(inorder, postorder, end);
    return root;
}
```

思路：与上题思路一致，只不过后序遍历的根结点总在尾部，而中序遍历根结点后总是右子树。

107. Binary Tree Level Order Traversal II Easy

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

For example:

Given binary tree [3,9,20,null,null,15,7],

```
  3
 / \
9  20
  /  \
 15   7
```

return its bottom-up level order traversal as:

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

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

```
    List<List<Integer>> ans = new ArrayList<List<Integer>>();
    Queue<TreeNode> q = new LinkedList<>(), nq = new LinkedList<>();
    List<Integer> can = new ArrayList<>();
    if (root != null)
        q.offer(root);
    while (!q.isEmpty()) {
        TreeNode c = q.poll();
        can.add(c.val);
        if (c.left != null)
            nq.offer(c.left);
        if (c.right != null)
            nq.offer(c.right);
        if (q.isEmpty()) {
            ans.add(0, new ArrayList(can));
            can = new ArrayList<>();
            if (!nq.isEmpty()) {
                Queue<TreeNode> temp = q;
                q = nq;
                nq = temp;
            }
        }
    }
    return ans;
}
```

思路：与 102 题思路相同，只不过这次每次添加新层 List 时总是插入到 0 位置，从而实现倒置。

108. Convert Sorted Array to Binary Search Tree Easy

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

```
public TreeNode sortedArrayToBST(int[] nums) {
    if (nums.length == 0)
        return null;
    TreeNode head = helper(nums, 0, nums.length - 1);
    return head;
}

private static TreeNode helper(int[] nums, int low, int high) {
    if (low > high)
        return null;
    int mid = (low + high) / 2;
    TreeNode node = new TreeNode(nums[mid]);
    node.left = helper(nums, low, mid - 1);
    node.right = helper(nums, mid + 1, high);
    return node;
}
```

思路：最优搜索二叉树是平衡的，即根结点为所有结点中的中位数，小于它的数为左子树，大于它的为右子树。递归求解，每次先找中位数，然后左侧元素为左子树，右侧元素为右子树。结束条件：越界时结点为 null。

109. Convert Sorted List to Binary Search Tree Medium

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

```
private ListNode node;

public TreeNode sortedListToBST(ListNode head) {
    if (head == null)
        return null;
    int size = 0;
    ListNode runner = head;
    node = head;
```

```
    while (runner != null) {
        runner = runner.next;
        size++;
    }
    return inorderHelper(0, size - 1);
}

public TreeNode inorderHelper(int start, int end) {
    if (start > end)
        return null;
    int mid = (start + end) / 2;
    TreeNode left = inorderHelper(start, mid - 1);
    TreeNode treenode = new TreeNode(node.val);
    treenode.left = left;
    node = node.next;
    TreeNode right = inorderHelper(mid + 1, end);
    treenode.right = right;
    return treenode;
}
```

思路：与上题思路相同。只不过因为链表没有线性表的位置信息，需要通过遍历来切分链表，同时双利用中间点为当前根，左右各为其子树，把问题分解成子问题求解。本解中使用中序建立根结点的方法，巧妙利用到达当前根结点前，所有左子树都会先被处理的特点，减少运算量。

110. Balanced Binary Tree Easy

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

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

```
public boolean isBalanced(TreeNode root) {
    return height(root) != -1;
}

public int height(TreeNode node) {
    if (node == null)
        return 0;
    int lH = height(node.left);
    if (lH == -1)
        return -1;
    int rH = height(node.right);
    if (rH == -1)
        return -1;
    if (Math.abs(lH - rH) > 1)
        return -1;
    return Math.max(lH, rH) + 1;
}
```

思路：递归求解。分别查看左右子树的深度，然后取大者+1 为当前子树的深度。结束条件：如果有左右子树的深度差超过 1，则表示不平衡，这时以-1 计，并返回，查子树过程中碰到-1 则返回-1；结点为空返回 0。

111. Minimum Depth of Binary Tree Easy

Given a binary tree, find its minimum depth.

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

```
public int minDepth(TreeNode root) {
    if (root == null)
        return 0;
    int left = minDepth(root.left);
    int right = minDepth(root.right);
    return (left == 0 || right == 0) ? left + right + 1 : Math.min(left, right) + 1;
```

}
思路：递归求解。结束条件：结点为空，深度 0。：分别求解左右子树深度。取较小者+1 即为本结点树的最小深度。特殊情况，有一子树不存在，则其深度为另一子树深度+1。

112. Path Sum Easy

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.
For example:
Given the below binary tree and sum = 22,

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

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

```
public boolean hasPathSum(TreeNode root, int sum) {
    return hasPathSum(root, 0, sum);
}

private boolean hasPathSum(TreeNode root, int curSum, int sum) {
    if (root == null)
        return false;
    curSum += root.val;
    if (root.left == null && root.right == null)
        return curSum == sum;
    return hasPathSum(root.left, curSum, sum) || hasPathSum(root.right, curSum, sum);
}
```

思路：递归求解。递推条件：左子树为真或右子树为真。结束条件：途中不断添加当前元素值，如果到叶子结点且和达到目标值，则为真；如果当前元素为空则为假；

113. Path Sum II Medium

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.
For example:
Given the below binary tree and sum = 22,

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

return

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

```
public List<List<Integer>> pathSum(TreeNode root, int sum) {
    List<List<Integer>> ans = new ArrayList<List<Integer>>();
    dfs(root, 0, sum, ans, new ArrayList<Integer>());
```

```
    return ans;
}

void dfs(TreeNode root, int cur, int sum, List<List<Integer>> ans, List<Integer> can) {
    if (root == null)
        return;
    if (root.left == null && root.right == null && cur + root.val == sum) {
        List<Integer> an = new ArrayList(can);
        an.add(root.val);
        ans.add(an);
    } else {
        can.add(root.val);
        dfs(root.left, cur + root.val, sum, ans, can);
        dfs(root.right, cur + root.val, sum, ans, can);
        can.remove(can.size() - 1);
    }
}
```

思路：与上题思路一致，只不过过程中通过创建新 List 不断记录新路径。如果存在目标路径，则把该 List 添加到结果中。

114. Flatten Binary Tree to Linked List Medium

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

```
    1
   /\
  2  5
 /\  \
3  4  6
```

The flattened tree should look like:

```
 1
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

Hints:
If you notice carefully in the flattened tree, each node's right child points to the next node of a pre-order traversal.

```
TreeNode pre = null;

public void flatten(TreeNode root) {
    helper(root);
}

void helper(TreeNode root) {
    if (root == null)
        return;
```

```
    TreeNode left = root.left;
    TreeNode right = root.right;
    if (pre != null) {
        pre.left = null;
        pre.right = root;
    }
    pre = root;
    helper(left);
    helper(right);
}
```

思路 1（略）：如提示，其实就是（可以用队列或递归）按先序遍历把树改成结点间仅有右子树的树。
思路 2：先序遍历就是所需要的顺序，所以在先序的过程中，保留上一结点位置信息(pre)，总是让上一位置的元素左结点为空，右结点为下一结点（即当前结点）。

115. Distinct Subsequences Hard

Given a string **S** and a string **T**, count the number of distinct subsequences of **S** which equals **T**.
A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).
Here is an example:
S = "rabbbit", **T** = "rabbit"
Return 3.

```
public int numDistinct(String s, String t) {
    int m = t.length(), n = s.length();
    int[][] dp = new int[m + 1][n + 1];
    for (int j = 0; j <= n; ++j)
        dp[0][j] = 1;
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j) {
            dp[i][j] = dp[i][j - 1];
            if (t.charAt(i - 1) == s.charAt(j - 1))
                dp[i][j] += dp[i - 1][j - 1];
        }
    return dp[m][n];
}
```

思路：动态规划。初始情况：空串是任何字符串的子序列，并且只是一次；非空串不是空串的子序列。递推方程：如果当前字符不同，则结果与当前字符没有出现过一样；如果当前字符相同，则再加上两串都不含当前字符的可能性。详解如下表：总是看当前行的上一个状态，为到目前字符的解的个数。每一行都可以捕获当前字符是否出现在子序列中，如果第一个字符出现在 T 中，则其后所有值都非 0，出现次数累加（每次加上一行上一列的值，对第 1 行来说，全部是 1）。到了下面行情况类似：如果当前字符完全不出现，则全部是零；如果出现，则 T 不含当前字符的个数加上 T 和 S 均不含当前字符的数量。

T\S		0	1	2	3	4
			a	b	b	c
0		1	1	1	1	1
1	a	0	1	1	1	1
2	b	0	0	1	2	2
3	c	0	0	0	0	2

116. Populating Next Right Pointers in Each Node Medium

Given a binary tree

```
struct TreeLinkNode {
  TreeLinkNode *left;
```

```
    TreeLinkNode *right;
    TreeLinkNode *next;
  }
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.
Initially, all next pointers are set to NULL.
Note:

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

For example,
Given the following perfect binary tree,

```
     1
   /  \
   2   3
  / \ / \
  4 5 6 7
```

After calling your function, the tree should look like:

```
     1 -> NULL
   /  \
   2 -> 3 -> NULL
  / \ / \
  4->5->6->7 -> NULL
```

```
public Node connect(Node root) {
    if (root == null)
        return root;
    Node pre = root;
    Node cur = null;
    while (pre.left != null) {
        cur = pre;
        while (cur != null) {
            cur.left.next = cur.right;
            if (cur.next != null)
                cur.right.next = cur.next.left;
            cur = cur.next;
        }
        pre = pre.left;
    }
    return root;
}
```

思路：通过一层的父结点的 next 关系，可以为下一层指定 next 指向（left 指向 right，right 指向 next.next 的 left……），然后到下一层，此时该层所有元素已经由 next 连接，同样方法把下一层连接，再到下一层。

117. Populating Next Right Pointers in Each Node II Medium

Follow up for problem "*Populating Next Right Pointers in Each Node*".
What if the given tree could be any binary tree? Would your previous solution still work?
Note:

- You may only use constant extra space.

For example,
Given the following binary tree,

```
     1
   /  \
  2    3
 / \    \
4   5    7
```

After calling your function, the tree should look like:

```
     1 -> NULL
   /  \
  2 -> 3 -> NULL
 / \    \
4-> 5 -> 7 -> NULL
```

```
public Node connect(Node root) {
    Node pre = null, cur = root, head = null;
    while (cur != null) {
        while (cur != null) {
            if (cur.left != null) {
                if (pre != null)
                    pre.next = cur.left;
                else
                    head = cur.left;
                pre = cur.left;
            }
            if (cur.right != null) {
                if (pre != null)
                    pre.next = cur.right;
                else
                    head = cur.right;
                pre = cur.right;
            }
            cur = cur.next;
        }
        cur = head;
        pre = null;
        head = null;
    }
    return root;
}
```

思路：与上题思路相同，多一步判断每一结点是否有左右子结点，如果没有就顺延 next 指向。在过程中需要问题保存两个信息，上一层的当前元素 cur，当前层的当前指针 pre。

118. Pascal's Triangle Easy

Given *numRows*, generate the first *numRows* of Pascal's triangle.
For example, given *numRows* = 5,
Return

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

```
public List<List<Integer>> generate(int numRows) {
```

```
    List<List<Integer>> allrows = new ArrayList<>();
    ArrayList<Integer> row = new ArrayList<>();
    for (int i = 0; i < numRows; i++) {
        row.add(0, 1);
        for (int j = 1; j < row.size() - 1; j++)
            row.set(j, row.get(j) + row.get(j + 1));
        allrows.add(new ArrayList<Integer>(row));
    }
    return allrows;
}
```

思路：每一元素是上一行对应元素和上一行前一元素（如果存在）之和。注意观察，每行元素数比上一行多 1，并且每行首元素都是 1，是不会改变的，所以每次先插入首元素 1（即元素数+1），然后设置 1 到 n-2 的值为 Row 中当前元素和下一元素之和（因为元素都右移了，所以相当于上一行的同位元素及其前一元素之和）。

119. Pascal's Triangle II Easy

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

For example, given *k* = 3,

Return [1,3,3,1].

Note:

Could you optimize your algorithm to use only *O*(*k*) extra space?

```
public List<Integer> getRow(int rowIndex) {
    ArrayList<Integer> row = new ArrayList<>();
    for (int i = 0; i < rowIndex + 1; i++) {
        row.add(0, 1);
        for (int j = 1; j < row.size() - 1; j++)
            row.set(j, row.get(j) + row.get(j + 1));
    }
    return row;
}
```

思路 1：与上题思路一致，只不过不缓存所有行，直接返回最后一行。

```
public List<Integer> getRow(int rowIndex) {
    Integer[] rowList = new Integer[rowIndex + 1];
    rowList[0] = 1;
    for (int i = 1; i < rowList.length; i++) {
        rowList[i] = (int) ((long) rowList[i - 1] * (rowIndex - i + 1) / i);
    }
    return Arrays.asList(rowList);
}
```

思路 2：注意观察，每一行的所有位置的值都可通过当前行的行数计算得出。

120. Triangle Medium

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

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

The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

Note:

Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.

```
public int minimumTotal(List<List<Integer>> triangle) {
```

```
    int min = Integer.MAX_VALUE;
    int[] can = new int[triangle.size() + 1];
    for (int i = triangle.size() - 1; i >= 0; --i) {
        List<Integer> cur = triangle.get(i);
        for (int j = 0; j < cur.size(); ++j) {
            can[j] = cur.get(j) + Math.min(can[j], can[j + 1]);
        }
    }
    return can[0];
}
```

思路：动态规划。逐行计算到当前元素的最小值，因为只能选上一行相邻的两个路径，所以易推导，逆向推导（从叶子到根）则在根处得到的最小值就是答案。利用叶子层个数的数组存中间结果，比 List 效率高很多。

121. Best Time to Buy and Sell Stock Easy

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.
If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Example 1:

Input: [7, 1, 5, 3, 6, 4]
Output: 5
max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)

Example 2:

Input: [7, 6, 4, 3, 1]
Output: 0
In this case, no transaction is done, i.e. max profit = 0.

```
public int maxProfit(int[] prices) {
    int maxCur = 0, maxSoFar = 0;
    for (int i = 1; i < prices.length; i++) {
        maxCur = Math.max(0, maxCur += prices[i] - prices[i - 1]);
        maxSoFar = Math.max(maxCur, maxSoFar);
    }
    return maxSoFar;
}
```

思路 1：从头到尾不断累加收益，如果小于零则归零（此时值小于上一个累加起点的值），每次累加的结果是一个极大值，取极大值中的最大值。中间虽然有时会相对上一天亏本，但是整体赢利，可以证明，只要整体赚，那么仍以该元素开始才能得到极值。O(n)

```
public int maxProfit(int[] prices) {
    int maxProfit = 0;
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < prices.length; i++)
        if (prices[i] < min)
            min = prices[i];
        else if (prices[i] - min > maxProfit)
            maxProfit = prices[i] - min;
    return maxProfit;
}
```

思路 2：因为只交易一次，所以在高点卖出即得一个极值。所以只要不断用当前值-当前点前的最小值作为极值，在极值中取最值即可。

122. Best Time to Buy and Sell Stock II Easy

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

```
public int maxProfit(int[] prices) {
    int cur = 0, max = 0;
    for (int i = 1; i < prices.length; ++i) {
        cur = prices[i] - prices[i - 1];
        max = max + (cur > 0 ? cur : 0);
    }
    return max;
}
```

思路：因为条件限制，只能获取一股收益，所以累加所有相邻天的正收益就是所求解。

123. Best Time to Buy and Sell Stock III Hard

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.
Design an algorithm to find the maximum profit. You may complete at most *two* transactions.
Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

```
public int maxProfit(int[] prices) {
    int balance1 = Integer.MIN_VALUE, balance2 = Integer.MIN_VALUE;
    int profit1 = 0, profit2 = 0;
    for (int price : prices) {
        balance1 = Math.max(balance1, -price);
        profit1 = Math.max(profit1, balance1 + price);
        balance2 = Math.max(balance2, profit1 - price);
        profit2 = Math.max(profit2, balance2 + price);
    }
    return profit2;
}
```

思路 1：动态规划。因无须回溯，所以只须记录最后一次的状态，永远以现金流为指标，最后一次卖所能得到的最大收益就是解。

```
public int maxProfit(int[] prices) {
    int len = prices.length, k = 2;
    int[] balance = new int[k + 1];
    Arrays.fill(balance, Integer.MIN_VALUE);
    int[] profit = new int[k + 1];

    for (int price : prices) {
        for (int j = 1; j <= k; j++) {
            balance[j] = Math.max(profit[j - 1] - price, balance[j]);
            profit[j] = Math.max(balance[j] + price, profit[j]);
        }
    }

    return profit[k];
}
```

思路 2：动态规划，同思路 1，注意到两次的操作方法一致，所以可以引入循环把解扩展到 k 个的情况。引入 balance 和 profit 数组分别为 k+1 个，balance 全部预设为 Integer.MIN_VALUE，从 k=1 开始计算，k=k 位置上的 profit 就是解。

124. Binary Tree Maximum Path Sum Hard

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

For example:
Given the below binary tree,

```
   1
  / \
 2  3
```

Return 6.

```
int maxValue;

public int maxPathSum(TreeNode root) {
    maxValue = Integer.MIN_VALUE;
    maxPathDown(root);
    return maxValue;
}

private int maxPathDown(TreeNode node) {
    if (node == null)
        return 0;
    int left = Math.max(0, maxPathDown(node.left));
    int right = Math.max(0, maxPathDown(node.right));
    maxValue = Math.max(maxValue, left + right + node.val);
    return Math.max(left, right) + node.val;
}
```

思路：遍历时记录每一个点的最值。最值有三种，从左、右子树到当前和从左子树经过当前再到右子树。O(n)

125. Valid Palindrome Easy

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.
For example,
"A man, a plan, a canal: Panama" is a palindrome.
"race a car" is *not* a palindrome.
Note:
Have you consider that the string might be empty? This is a good question to ask during an interview.
For the purpose of this problem, we define empty string as valid palindrome.

```
public boolean isPalindrome(String s) {
    if (s.isEmpty()) {
        return true;
    }
    int head = 0, tail = s.length() - 1;
    char cHead, cTail;
    while (head < tail) {
        cHead = s.charAt(head);
        cTail = s.charAt(tail);
        if (!Character.isLetterOrDigit(cHead)) {
            head++;
        } else if (!Character.isLetterOrDigit(cTail)) {
            tail--;
        } else {
            if (Character.toLowerCase(cHead) != Character.toLowerCase(cTail))
                return false;
            head++;
            tail--;
        }
    }
    return true;
}
```

思路：从两头进行比较，忽略非字母数字的字符和大小写（面试时注意讨论）。

126. Word Ladder II Hard

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

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

For example,

Given:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log","cog"]

Return

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

Note:

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

```
public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {
    Set<String> start = new HashSet<>();
    Set<String> end = new HashSet<>();
    Set<String> dict = new HashSet<>();
    start.add(beginWord);
    end.add(endWord);
    dict.addAll(wordList);
    HashMap<String, List<String>> map = new HashMap<>();
    List<List<String>> res = new ArrayList<>();
    if (!dict.contains(endWord))
        return res;
    buildMap(start, end, false, dict, map);
    List<String> path = new ArrayList<>();
    path.add(beginWord);
    genPath(beginWord, endWord, res, map, path);
    return res;
}

private void genPath(String start, String end, List<List<String>> ans, HashMap<String,
List<String>> map,
        List<String> temp) {
    if (start.equals(end)) {
        ans.add(new ArrayList<>(temp));
        return;
    }
    if (!map.containsKey(start))
        return;
    for (String s : map.get(start)) {
        temp.add(s);
        genPath(s, end, ans, map, temp);
        temp.remove(temp.size() - 1);
    }
}

private void buildMap(Set<String> start, Set<String> end, boolean reverse, Set<String> dict,
        HashMap<String, List<String>> map) {
    if (start.size() == 0)
```

```
            return;
        if (start.size() > end.size()) {
            buildMap(end, start, !reverse, dict, map);
            return;
        }
        dict.removeAll(start);
        boolean finished = false;
        HashSet<String> next = new HashSet<>();
        for (String word : start) {
            char[] arr = word.toCharArray();
            for (int i = 0; i < arr.length; i++) {
                char old = arr[i];
                for (char c = 'a'; c <= 'z'; c++) {
                    if (c == old)
                        continue;
                    arr[i] = c;
                    String newString = new String(arr);
                    if (dict.contains(newString)) {
                        if (end.contains(newString))
                            finished = true;
                        else
                            next.add(newString);
                        String parent = reverse ? newString : word;
                        String child = reverse ? word : newString;
                        List<String> neighbor = map.getOrDefault(parent, new ArrayList<String>());
                        neighbor.add(child);
                        map.put(parent, neighbor);
                    }
                }
                arr[i] = old;
            }
        }
        if (!finished)
            buildMap(next, end, reverse, dict, map);
    }
```

思路：BFS、回溯。先建立词表，每一词相邻的所有词都放在其关键字的 List 中。从 BeginWord 开始找，逐一试探看其相邻的里面有没有可以通过连续相邻达到 EndWord。建立词表方法：修改一个词中的任一字母，如果该 word 存在，则添加到该关键字的 List 中。技巧（提高性能）：相邻词从哪边开始到另一边通过的方法是一致的，所以每次取结点数较少的一边开始找。

127. Word Ladder Medium

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

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

For example,

Given:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log","cog"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.

- You may assume *beginWord* and *endWord* are non-empty and are not the same.

```
public int ladderLength(String beginWord, String endWord, List<String> wordList) {
    if (beginWord == null || endWord == null || wordList == null)
        return 0;
    if (beginWord.equals(endWord))
        return 1;
    return findShortestLength(beginWord, endWord, wordList);
}

private int findShortestLength(String beginWord, String endWord, List<String> wordList) {
    Set<String> dict = new HashSet<>(wordList);
    if (!dict.contains(endWord))
        return 0;
    Queue<String> from = new LinkedList<>(), to = new LinkedList<>();
    from.add(beginWord);
    to.add(endWord);

    Set<String> fromVisited = new HashSet<>(), toVisited = new HashSet<>();
    fromVisited.add(beginWord);
    toVisited.add(endWord);

    int trans = 1;
    while (!from.isEmpty() && !to.isEmpty()) {
        trans++;
        if (from.size() <= to.size()) {
            if (canTransform(from, fromVisited, toVisited, dict))
                return trans;
        } else {
            if (canTransform(to, toVisited, fromVisited, dict))
                return trans;
        }
    }

    return 0;
}

private boolean canTransform(Queue<String> from, Set<String> fromVisited, Set<String> toVisited,
Set<String> dict) {
    int size = from.size();
    while (size-- > 0) {
        String word = from.poll();
        List<String> trans = transform(word, dict);
        for (String next : trans) {
            if (toVisited.contains(next))
                return true;
            if (fromVisited.contains(next))
                continue;
            from.add(next);
            fromVisited.add(next);
        }
    }

    return false;
}

private List<String> transform(String word, Set<String> dict) {
    char[] s = word.toCharArray();
    List<String> words = new ArrayList<>();

    for (int i = 0; i < s.length; i++) {
        char c = s[i];
        for (char n = 'a'; n <= 'z'; n++) {
            if (n != c) {
```

```
                s[i] = n;
                String trans = String.valueOf(s);
                if (dict.contains(trans)) {
                    words.add(trans);
                }
            }
        }
        s[i] = c;
    }

    return words;
}
```

思路：BFS（广度优先搜索）因为只要找最短的可能，所以每次只找下一层的所有可能，再一层层递进，直到找到。和广度优先遍历类似，使用一个 Set 缓存下一层可能的词，然后到下一层逐一试。

128. Longest Consecutive Sequence Hard

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.
For example,
Given [100, 4, 200, 1, 3, 2],
The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.
Your algorithm should run in O(*n*) complexity.

```
public int longestConsecutive(int[] nums) {
    int max = 0, peak = 0;
    Set<Integer> dict = new HashSet<>();
    for (int num : nums)
        dict.add(num);
    for (int num : nums) {
        if (dict.remove(num)) {
            peak = 1;
            int l = num - 1, r = num + 1;
            while (dict.remove(l--))
                ++peak;
            while (dict.remove(r++))
                ++peak;
            max = Math.max(peak, max);
        }
    }
    return max;
}
```

思路：模拟数数的方法。把所有数字先存入 Set，然后逐一遍历所有数字，如果 Set 中还有该数字，那么把该数字及其相邻数字全部从 Set 中删除，并记其长度 max（极植），取所有极值中的最值。

129. Sum Root to Leaf Numbers Medium

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.
An example is the root-to-leaf path 1->2->3 which represents the number 123.
Find the total sum of all root-to-leaf numbers.
For example,

```
  1
 /\
2  3
```

The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Return the sum = 12 + 13 = 25.
Example 2:

Input: [4,9,0,5,1]

```
  4
 / \
9  0
/ \
5  1
```

Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.

```
public int sumNumbers(TreeNode root) {
    return sum(root, 0);
}

public int sum(TreeNode n, int s) {
    if (n == null)
        return 0;
    if (n.right == null && n.left == null)
        return s * 10 + n.val;
    return sum(n.left, s * 10 + n.val) + sum(n.right, s * 10 + n.val);
}
```

思路：DFS，过程中每入一级前当前 Sum 值都*10 即可。

130. Surrounded Regions Medium

Given a 2D board containing 'X' and 'O' (the **letter** O), capture all regions surrounded by 'X'.
A region is captured by flipping all 'O's into 'X's in that surrounded region.
For example,

```
X X X X
X O O X
X X O X
X O X X
```

After running your function, the board should be:

```
X X X X
X X X X
X X X X
X O X X
```

```
public void solve(char[][] board) {
    int m = board.length;
    if (m == 0)
        return;
    int n = board[0].length;
    for (int i = 0; i < m; ++i) {
        if (board[i][0] == 'O')
            dfs(board, i, 0);
        if (board[i][n - 1] == 'O')
            dfs(board, i, n - 1);
    }
    for (int j = 0; j < n; ++j) {
        if (board[0][j] == 'O')
            dfs(board, 0, j);
        if (board[m - 1][j] == 'O')
```

```
            dfs(board, m - 1, j);
    }
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (board[i][j] == 'O')
                board[i][j] = 'X';
            else if (board[i][j] == '+')
                board[i][j] = 'O';
        }
    }
}

void dfs(char[][] board, int i, int j) {
    if (i < 0 || i == board.length || j < 0 || j == board[0].length || board[i][j] == 'X' ||
board[i][j] == '+')
        return;
    board[i][j] = '+';
    dfs(board, i + 1, j);
    dfs(board, i - 1, j);
    dfs(board, i, j + 1);
    dfs(board, i, j - 1);
}
```

思路：先把不被包围的 O 全部置成特殊字符（这里使用*），然后再逐一扫描元素，是*的置回 O，是 O 的置成 X。

131. Palindrome Partitioning Medium

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of *s*.

For example, given *s* = "aab",

Return

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

```
public List<List<String>> partition(String s) {
    List<List<String>> ans = new ArrayList<>();
    char[] cs = s.toCharArray();
    backTrack(cs, 0, new ArrayList<String>(), ans);
    return ans;
}

void backTrack(char[] cs, int j, List<String> can, List<List<String>> ans) {
    if (j == cs.length) {
        ans.add(new ArrayList(can));
        return;
    }
    for (int i = j; i < cs.length; ++i) {
        if (!isP(cs, j, i))
            continue;
        can.add(new String(cs, j, i - j + 1));
        backTrack(cs, i + 1, can, ans);
        can.remove(can.size() - 1);
    }
}

boolean isP(char[] cs, int l, int r) {
    while (l < r) {
        if (cs[l++] != cs[r--])
            return false;
    }
```

```
    return true;
}
```
思路：回溯法。结束条件：到达字符串结尾。每次从上次结束位置开始搜寻可能的串，如有就添加到中间结果中，一轮回溯后删除。

132. Palindrome Partitioning II Hard

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.
Return the minimum cuts needed for a palindrome partitioning of *s*.
For example, given *s* = "aab",
Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.

```
public int minCut(String s) {
    int n = s.length();
    char[] t = s.toCharArray();
    int[] dp = new int[n + 1];
    Arrays.fill(dp, Integer.MAX_VALUE);
    dp[0] = -1;
    int i = 0;
    while (i < n) {
        expandAround(t, i, i, dp);
        expandAround(t, i, i + 1, dp);
        i++;
    }
    return dp[n];
}

public void expandAround(char[] t, int i, int j, int[] dp) {
    while (i >= 0 && j < t.length && t[i] == t[j]) {
        dp[j + 1] = Math.min(dp[j + 1], dp[i] + 1);
        i--;
        j++;
    }
}
```
思路：动态规划。与 125 思路类似第一点及其周边的可能中取最小值。其中已经计算过的部分可以存入 dp。
$O(n^2)$

```
public int minCut(String s) {
    int len = s.length();
    char[] c = s.toCharArray();
    int[] dp = new int[len + 1];
    boolean[][] p = new boolean[len + 1][len + 1];
    for (int i = 1; i <= len; i++) {
        dp[i] = i;
        for (int j = 1; j <= i; j++) {
            if ((c[j - 1] == c[i - 1]) && (j + 1 > i - 1 || p[j + 1][i - 1])) {
                p[j][i] = true;
                dp[i] = Math.min(dp[i], dp[j - 1] + 1);
            }
        }
    }
    return dp[len] - 1;
}
```
思路：动态规划。最坏情况：所有字母单独成串。推导方程：当前最好情况，是前面最好情况+1（如果前面是 Palindrome，如果前面不是，那么情况不会比单字母成串好）。

133. Clone Graph Medium

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.
OJ's undirected graph serialization:
Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.
As an example, consider the serialized graph {0,1,2#1,2#2,2}.
The graph has a total of three nodes, and therefore contains three parts as separated by #.

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

Visually, the graph looks like the following:

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

```
public Node cloneGraph(Node node) {
    Node root = new Node(node.val, new ArrayList<Node>());
    Queue<Node> q = new LinkedList<>();
    Map<Integer, Node> map = new HashMap<Integer, Node>();
    map.put(node.val, root);
    q.offer(node);
    while (!q.isEmpty()) {
        Node cur = q.poll();
        for (Node nb : cur.neighbors) {
            if (!map.containsKey(nb.val)) {
                map.put(nb.val, new Node(nb.val, new ArrayList<Node>()));
                q.offer(nb);
            }
            map.get(cur.val).neighbors.add(map.get(nb.val));
        }
    }
    return root;
}
```

思路 1：BFS，每次把新结点放入将访问队列，通过 Map 记录已访问结点，避免重复访问。O(nk)，n 结点数，k 邻居数。前提：结点中的值唯一。

```
Map<Node, Node> map = new HashMap<>();

public Node cloneGraph(Node node) {
    if (node == null)
        return null;
    if (map.containsKey(node))
        return map.get(node);
    Node newNode = new Node(node.val, new ArrayList<>());
    map.put(node, newNode);
    for (Node next : node.neighbors) {
        Node neighborNode = cloneGraph(next);
        newNode.neighbors.add(neighborNode);
    }
    return newNode;
}
```

思路 2：DFS，思路同思路 1，只不过深度优先，好处是结点访问次数减少。

134. Gas Station Medium

There are *N* gas stations along a circular route, where the amount of gas at station *i* is gas[i].
You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station *i* to its next station (*i*+1). You begin the journey with an empty tank at one of the gas stations.
Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Note:
The solution is guaranteed to be unique.

```
public int canCompleteCircuit(int[] gas, int[] cost) {
    int start = gas.length, end = 0, sum = 0;
    do
        sum += sum > 0 ? gas[end] - cost[end++] : gas[--start] - cost[start];
    while (start != end);
    return sum >= 0 ? start : -1;
}
```

思路：从最后一站开始试，把终点设置为第一站，如果达不到终点则前移一站，如果达到终点则终点后移，直到起点和终点相遇，这时如果 Sum>0，则该起始点可以环游。

135. Candy Hard

There are *N* children standing in a line. Each child is assigned a rating value.
You are giving candies to these children subjected to the following requirements:

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

What is the minimum candies you must give?

```
public int candy(int[] ratings) {
    int n = ratings.length, sum = 0;
    int[] left = new int[n], right = new int[n];
    left[0] = 1;
    right[n - 1] = 1;
    for (int i = 1; i < n; ++i) {
        if (ratings[i] > ratings[i - 1])
            left[i] = left[i - 1];
        left[i] += 1;
        if (ratings[n - i - 1] > ratings[n - i])
            right[n - i - 1] = right[n - i];
        right[n - i - 1] += 1;
    }
    for (int i = 0; i < n; ++i) {
        sum += Math.max(left[i], right[i]);
    }
    return sum;
}
```

思路 1：左右各扫一遍，给较大分值的孩子比前一孩子多分一块糖，如果不比前一孩子分值高，则仅第一次分一块。左右分别求每一点所需要的值，各点中取较大的值作为结果。O(N)

```
public int candy(int[] ratings) {
    if (ratings.length == 0)
        return 0;
    int result = 1;
    int last = 1;
    for (int i=1; i<ratings.length;) {
        if (ratings[i] > ratings[i-1]) {
            last++;
            result += last;
            i++;
        } else if (ratings[i] == ratings[i-1]) {
            last = 1;
            result++;
            i++;
        } else {
            int count = 0;
            while (i < ratings.length && ratings[i] < ratings[i-1]) {
                count++;
                i++;
            }
            if (count >= last) {
```

```
                result += (1+count)*count/2 + count + 1 - last;
                last = 1;
            } else {
                result += (1+count)*count/2;
                last = 1;
            }
        }
    }
    return result;
}
```

思路 2：因为每人至少一块糖，较高的多一块糖，如果数列递增（减），则所需要糖的数量=(n+1)*n/2。把数列划分为 m 个递增（减）和相等数列分别计算即可。当当前值小于等于前一值时，当前值置 1 就符合条件（只说了值大时需要糖多，没有说值小（等）时需要糖少（等），同（小）值的给 1 就行。递增、减时要考虑数列旁边值的情况做相应的增减（可能是数列中每一个值，也可能是和数列相邻的单值需要变化）。经验，递增时使用公式和使用答案中的算法，效率一样。

136. Single Number Easy

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

Note:

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

```
public int singleNumber(int[] nums) {
    int ans = 0;
    for (int num : nums)
        ans ^= num;
    return ans;
}
```

思路：利用异或操作特点，与自己异或就可以得到 0，与别的数字异或再与自己异或则得到别的数字。

137. Single Number II Medium

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

Note:

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

```
public int singleNumber(int[] nums) {
    int ones = 0, twos = 0;
    for (int num : nums) {
        ones = (ones ^ num) & ~twos;
        twos = (twos ^ num) & ~ones;
    }
    return ones;
}
```

思路 1：利用异或特点，出现一次的会缓存在 ones 里面，出现两次的会缓存在 twos 里面，出现 3 次会被中和掉。

```
public int singleNumber(int[] nums) {
    int sum, mask = 1, res = 0;
    for (int i = 0; i < 32; ++i) {
        sum = 0;
        for (int num : nums) {
            sum += (num >> i) & mask;
            sum %= 3;
        }
        res |= sum << i;
    }
    return res;
}
```

思路 2：借位返还。计算所有数中第一位出现的次数，因为 Integer 类型为 32 位，所以循环 32 次，每次取一位，计算出现次数，因为除一个数出现了一次，其余都出现 3 次，故把每位出现次数%3 后，即剩下只出现一次的数的数位，恢复即得结果。

138. Copy List with Random Pointer Medium

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

Return a deep copy of the list.

```
public Node copyRandomList(Node head) {
    Map<Node, Node> map = new HashMap<>();
    Node node = head, ans, newNode;
    while (node != null) {
        map.put(node, new Node());
        node = node.next;
    }
    node = head;
    newNode = map.get(node);
    ans = newNode;
    while (node != null) {
        newNode.val = node.val;
        newNode.next = map.get(node.next);
        newNode.random = map.get(node.random);
        newNode = newNode.next;
        node = node.next;
    }
    return ans;
}
```

思路 1：类似 133 题。遍历两次原链表，第一次利用 Map 记录新 Node，第二次利用 Map 为新 Node 添加 next 和 random 指向。O(n)

```
public Node copyRandomList(Node head) {
    if (head == null)
        return null;
    Node current = head;
    while (current != null) {
        Node temp = current.next;
        current.next = new Node(current.val, temp, null);
        current = temp;
    }
    current = head;
    while (current != null) {
        if (current.random != null)
            current.next.random = current.random.next;
        current = current.next.next;
    }
    current = head;
    Node copyList = head.next;
    while (current != null) {
        Node copy = current.next;
        Node next = current.next.next;
        current.next = copy.next;
        if (next != null)
            copy.next = next.next;
        current = next;
    }
    return copyList;
}
```

思路 2：不使用额外空间。遍历三次，第一次把拷贝原数组元素，并把相应元素插入到原元素后面。第一次遍历后，next 和 random 的 next 都会指向新的 next 和 random。第二次，让新元素的 random 指向原 random.next。第三次，把新数组独立出来。

139. Word Break Medium

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words. You may assume the dictionary does not contain duplicate words.

For example, given

s = "leetcode",

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

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

```
public boolean wordBreak(String s, List<String> wordDict) {
    boolean[] dp = new boolean[s.length() + 1];
    dp[0] = true;
    for (int i = 1; i <= s.length(); ++i) {
        for (int j = 0; j <= i; ++j) {
            if (dp[j] && wordDict.contains(s.substring(j, i))) {
                dp[i] = true;
                break;
            }
        }
    }
    return dp[s.length()];
}
```

思路 1：动态规划，每次都利用已处理部分可分信息看到当前位置是否可分，如果可分置 True。$O(n^2k)$

```
public boolean wordBreak(String s, List<String> wordDict) {
    return dfs(s, wordDict, 0, new int[s.length()]);
}

private boolean dfs(String s, List<String> dict, int begin, int[] memo) {
    if (begin >= s.length() || memo[begin] == -1)
        return false;
    for (int i = 0; i < dict.size(); ++i) {
        int end = isSubStr(s, begin, dict.get(i));
        if (end != -1) {
            if (end == s.length() - 1 || dfs(s, dict, end + 1, memo))
                return true;
        }
    }
    memo[begin] = -1;
    return false;
}

private int isSubStr(String s, int begin, String p) {
    if (begin + p.length() > s.length())
        return -1;
    for (int i = 0; i < p.length(); i++) {
        if (p.charAt(i) != s.charAt(begin + i))
            return -1;
    }
    return begin + p.length() - 1;
}
```

思路 2：DFS。用字典中所有可能的词向前推进，如果有词在词典中，则前进一定距离，如果没有或长度超过字符串长度则返回否。

140. Word Break II Hard

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

Return all such possible sentences.

For example, given
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].
A solution is ["cats and dog", "cat sand dog"].

```
public List<String> wordBreak(String s, List<String> wordDict) {
    return dfs(s, wordDict, new HashMap<>());
}

List<String> dfs(String s, List<String> dict, Map<String, List<String>> mem) {
    if (mem.containsKey(s)) {
        return mem.get(s);
    }
    List<String> ans = new ArrayList<String>();
    if (s.length() == 0) {
        ans.add("");
        return ans;
    }
    for (String word : dict) {
        if (s.startsWith(word)) {
            List<String> canList = dfs(s.substring(word.length()), dict, mem);
            for (String can : canList)
                ans.add(word + (can.length() == 0 ? "" : " ") + can);
        }
    }
    mem.put(s, ans);
    return ans;
}
```

思路 1：DFS+回溯，每次试用词典中每一个词作为下一段开始的一种可能。O（n^2k）。

```
public List<String> wordBreak(String s, List<String> wordDict) {
    int min = Integer.MAX_VALUE, max = 0;
    HashSet<String> set = new HashSet<String>();
    for (String word : wordDict) {
        set.add(word);
        int curLen = word.length();
        min = (curLen < min) ? curLen : min;
        max = (curLen > max) ? curLen : max;
    }
    List<String> result = new ArrayList<String>();
    boolean[] invalid = new boolean[s.length()]; // invalid[i]: [i:] is unbreakable
    seperate(s, result, new StringBuilder(), 0, set, invalid, min, max);
    return result;
}

public boolean seperate(String s, List<String> res, StringBuilder tmp, int index, HashSet<String>
set,
        boolean[] invalid, int min, int max) {
    if (index == s.length()) {
        res.add(tmp.toString().trim());
        return true;
    }
    boolean breakable = false;
    int prelen = tmp.length();
    int rightbound = Math.min(s.length(), index + max);
    for (int end = index + min; end <= rightbound; end++) {
        int curLen = end - index;
        if (end < s.length() && invalid[end])
            continue;
        String cur = s.substring(index, end);
        if (set.contains(cur)) {

            tmp.append(" ").append(cur);
            breakable |= seperate(s, res, tmp, end, set, invalid, min, max);
```

```
                tmp.setLength(prelen);
            }
        }
        invalid[index] = !breakable;
        return breakable;
    }
```

思路 2：与思路 1 类似，不是词去匹配字符串，而是字符串先切分，再看分出来的部分是不是在词典中。

141. Linked List Cycle Easy

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

Follow up:

Can you solve it without using extra space?

```
public boolean hasCycle(ListNode head) {
    ListNode walker = head, runner = head;
    while (runner != null && runner.next != null && runner.next.next != null) {
        walker = walker.next;
        runner = runner.next.next;
        if (runner == walker)
            return true;
    }
    return false;
}
```

思路：双指针法。如果链表中有环，则快指针（每次两步）和慢指针（每次一步）必有相遇的时候。

142. Linked List Cycle II Medium

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

Note: Do not modify the linked list.

Follow up:

Can you solve it without using extra space?

```
public ListNode detectCycle(ListNode head) {
    ListNode walker = head, runner = head;
    boolean isCycled = false;
    while (runner != null && runner.next != null && runner.next.next != null) {
        walker = walker.next;
        runner = runner.next.next;
        if (walker == runner) {
            walker = head;
            while (walker != runner) {
                walker = walker.next;
                runner = runner.next;
            }
            return walker;
        }
    }
    return null;
}
```

思路：双指针技术。假设慢指针到达 Loop 起点时，走了 k 步，则快指针已经在 Loop 中 k 步。两点首次相遇，离开 Loop 起点 loop size – k 步，距离 loop 起点 k=(L- (L-k))步，所以再用一指针从头开始，与慢指针一起一步一步移动，再相遇时就是起点。

143. Reorder List Medium

Given a singly linked list L: $L_0?L_1?\dots?L_{n-1}?L_n$,

reorder it to: $L_0?L_n?L_1?L_{n-1}?L_2?L_{n-2}?\dots$

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

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

```
public void reorderList(ListNode head) {
    if (head == null || head.next == null || head.next.next == null)
        return;
    ListNode walker = head, runner = head;
    while (runner != null && runner.next != null && runner.next.next != null) {
        walker = walker.next;
        runner = runner.next.next;
    }
    if (runner.next != null)
        runner = runner.next;
    ListNode pre = new ListNode(0), cur = walker.next;
    pre.next = cur;
    walker.next = null;
    while (cur != runner) {
        pre.next = cur.next;
        cur.next = runner.next;
        runner.next = cur;
        cur = pre.next;
    }
    pre.next = head;
    cur = new ListNode(0);
    while (head != null || runner != null) {
        if (head != null) {
            cur.next = head;
            head = head.next;
            cur = cur.next;
            cur.next = null;
        }
        if (runner != null) {
            cur.next = runner;
            runner = runner.next;
            cur = cur.next;
            cur.next = null;
        }
    }
    head = pre.next;
}
```

思路：三步走，先找到中点，然后把后半个链表倒置，然后再合并得到结果。O(n)

144. Binary Tree Preorder Traversal Medium

Given a binary tree, return the *preorder* traversal of its nodes' values.
For example:
Given binary tree {1,#,2,3},

```
 1
   \
    2
   /
  3
```

return [1,2,3].
Note: Recursive solution is trivial, could you do it iteratively?

```
public List<Integer> preorderTraversal(TreeNode root) {
    List<Integer> ans = new ArrayList<>();
    Stack<TreeNode> s = new Stack<>();
    if(root != null)
        s.push(root);
```

```
    while(!s.isEmpty()) {
        TreeNode cur = s.pop();
        while(cur != null) {
            ans.add(cur.val);
            if(cur.right != null)
                s.push(cur.right);
            cur = cur.left;
        }
    }
    return ans;
}
```

思路：使用栈模拟前序遍历，注意先添加右子树，后添加左子树，因为左子树要先弹出。

145. Binary Tree Postorder Traversal Hard

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

For example:

Given binary tree {1,#,2,3},

```
 1
 \
  2
 /
3
```

return [3,2,1].

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

```
public List<Integer> postorderTraversal(TreeNode root) {
    List<Integer> ans = new ArrayList<>();
    Stack<TreeNode> s = new Stack<>();
    if (root != null)
        s.push(root);
    while (!s.isEmpty()) {
        root = s.pop();
        while (root != null) {
            ans.add(0, root.val);
            if (root.left != null)
                s.push(root.left);
            root = root.right;
        }
    }
    return ans;
}
```

思路：与上题思路一样，利用栈缓存中间结果，注意后序遍历特点是先右子树，再左子树，再根。

146. LRU Cache Hard

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

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

put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

Follow up:

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

Example:

```
LRUCache cache = new LRUCache( 2 /* capacity */ );
cache.put(1, 1);
```

```
cache.put(2, 2);
cache.get(1);      // returns 1
cache.put(3, 3);   // evicts key 2
cache.get(2);      // returns -1 (not found)
cache.put(4, 4);   // evicts key 1
cache.get(1);      // returns -1 (not found)
cache.get(3);      // returns 3
cache.get(4);      // returns 4
```

```
class LRUCache {
    private LinkedHashMap<Integer, Integer> map;
    private int CAPACITY;

    public LRUCache(int capacity) {
        CAPACITY = capacity;
        map = new LinkedHashMap<Integer, Integer>(capacity, 0.75f, true) {
            @Override
            protected boolean removeEldestEntry(Map.Entry eldest) {
                return size() > CAPACITY;
            }
        };
    }

    public int get(int key) {
        return map.getOrDefault(key, -1);
    }

    public void put(int key, int value) {
        map.put(key, value);
    }
}
```

思路：利用LinkedHashMap的移除最老元素。如果不能用该数据结构，可使用双链表+HashMap(key->node)相结合的方式。每次put中get时，同时把相应词移动（或添加）到队尾，并且把原来位置的前后两元素相联。

147. Insertion Sort List Medium

Sort a linked list using insertion sort.

```
public ListNode insertionSortList(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode dummy = new ListNode(0);
    ListNode pre = dummy;
    ListNode cur = head;
    while (cur != null) {
        ListNode next = cur.next;
        while (pre.next != null && pre.next.val < cur.val) {
            pre = pre.next;
        }
        cur.next = pre.next;
        pre.next = cur;
        pre = dummy;
        cur = next;
    }

    return dummy.next;
}
```

思路 1：模拟扑克起牌动作，每次拿下一张牌放入对应的位置。这里需要注意的地方是 dummy 结点，本身的值是永远不变的，但是因为 pre 指向它并更改了 next，其 next 也会随之改变。

思路 2：

148. Sort List Medium

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

```
public ListNode sortList(ListNode head) {
    if (head == null || head.next == null)
        return head;
    ListNode slow = head, fast = head;
    while (fast != null && fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    fast = slow.next;
    slow.next = null;
    ListNode left = sortList(head);
    ListNode right = sortList(fast);
    return merge(left, right);
}

ListNode merge(ListNode l, ListNode r) {
    ListNode dummy = new ListNode(0), cur = dummy;
    while (l != null || r != null) {
        if (l == null) {
            cur.next = r;
            break;
        }
        if (r == null) {
            cur.next = l;
            break;
        }
        if (l.val < r.val) {
            cur.next = l;
            l = l.next;
        } else {
            cur.next = r;
            r = r.next;
        }
        cur = cur.next;
    }
    return dummy.next;
}
```

思路：归并排序。分治法的典型场景，先把链表分成基本等长的两个链表，分别排序，排好后 Merge，Merge 时因两个链表都已排序，方法同 21 题，复杂度 n，因为进行了 logn 次拆分，故总复杂度 O(nlogn)。

149. Max Points on a Line Hard

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

```
public int maxPoints(int[][] points) {
    if (points == null || points.length < 3)
        return points.length;
    int res = 0;
    for (int i = 0; i < points.length; i++) {
        int[] a = points[i];
        for (int k = i + 1; k < points.length; ++k) {
            int[] b = points[k];
            int x = a[0];
            int y = a[1];
            long dx = x - b[0];
            long dy = y - b[1];
            int count = 0;
            if (dx == 0 && dy == 0) {
                for (int j = 0; j < points.length; j++) {
```

```
                    if (points[j][0] == x && points[j][1] == y) {
                        count++;
                    }
                }
            } else {
                for (int j = 0; j < points.length; j++) {
                    if ((points[j][0] - x) * dy == (points[j][1] - y) * dx) {
                        count++;
                    }
                }
            }
            res = Math.max(res, count);
        }
    }
    return res;
}
```

思路：遍历所有两点的组合（直线），判断所有点是否在该线上。分两种情况：同一点和斜率相同。$O(n^3)$

150. Evaluate Reverse Polish Notation Medium

Evaluate the value of an arithmetic expression in Reverse Polish Notation.
Valid operators are +, -, *, /. Each operand may be an integer or another expression.
Some examples:

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

```
public int evalRPN(String[] tokens) {
    Stack<Integer> s = new Stack<>();
    int a, b;
    for (String t : tokens) {
        if ("+".equals(t))
            s.push(s.pop() + s.pop());
        else if ("-".equals(t))
            s.push(0 - (s.pop() - s.pop()));
        else if ("*".equals(t))
            s.push(s.pop() * s.pop());
        else if ("/".equals(t)) {
            a = s.pop();
            b = s.pop();
            s.push(b / a);
        } else
            s.push(Integer.valueOf(t));
    }
    return s.pop();
}
```

思路：无须考虑括号，典型栈操作即可完成，如果碰到数字就入栈，碰到操作符就出栈两个并进行相应操作后再入栈，最后弹出栈中结果。

151. Reverse Words in a String Medium

Given an input string, reverse the string word by word.
For example,
Given s = "the sky is blue",
return "blue is sky the".
Update (2015-02-12):
For C programmers: Try to solve it *in-place* in *O*(1) space.
Clarification:

- What constitutes a word?
  A sequence of non-space characters constitutes a word.
- Could the input string contain leading or trailing spaces?
  Yes. However, your reversed string should not contain leading or trailing spaces.
- How about multiple spaces between two words?
  Reduce them to a single space in the reversed string.

```
public String reverseWords(String s) {
    String[] strs = s.split(" ");
    StringBuilder sb = new StringBuilder();
    for (int i = strs.length - 1; i >= 0; --i) {
        if (strs[i].length() > 0) {
            sb.append(strs[i]);
            sb.append(" ");
        }
    }
    return sb.toString().trim();
}
```

思路 **1**：使用 API。有很多方法(lastIndexOf，indexOf，split 等)。这里只给出利用 split 把词用空格拆分，然后逆序添加到 StringBuilder 中，每个词后加空格（除最后一个）即得结果。

```
public String reverseWords(String s) {
    if (s == null)
        return null;
    char[] a = s.toCharArray();
    int n = a.length;
    reverse(a, 0, n - 1);
    reverseWords(a, n);
    return cleanSpaces(a, n);
}

void reverseWords(char[] a, int n) {
    int i = 0, j = 0;
    while (i < n) {
        while (i < j || i < n && a[i] == ' ')
            i++;
        while (j < i || j < n && a[j] != ' ')
            j++;
        reverse(a, i, j - 1);
    }
}

String cleanSpaces(char[] a, int n) {
    int i = 0, j = 0;
    while (j < n) {
        while (j < n && a[j] == ' ')
            j++;
        while (j < n && a[j] != ' ')
            a[i++] = a[j++];
        while (j < n && a[j] == ' ')
            j++;
        if (j < n)
            a[i++] = ' ';
    }
    return new String(a).substring(0, i);
}

private void reverse(char[] a, int i, int j) {
    while (i < j) {
        char t = a[i];
        a[i++] = a[j];
        a[j--] = t;
    }
```

}
思路 **2**：先把字符串整体反转，然后再把被空格分开的每一部分反转即得结果。

152. Maximum Product Subarray Medium

Find the contiguous subarray within an array (containing at least one number) which has the largest product.
For example, given the array [2,3,-2,4],
the contiguous subarray [2,3] has the largest product = 6.

```
public int maxProduct(int[] nums) {
    int ans = nums[0];
    for (int i = 1, max = nums[0], min = nums[0]; i < nums.length; ++i) {
        if (nums[i] < 0) {
            int tmp = max;
            max = min;
            min = tmp;
        }
        max = Math.max(max * nums[i], nums[i]);
        min = Math.min(min * nums[i], nums[i]);
        ans = Math.max(max, ans);
    }
    return ans;
}
```

思路：动态规划。如果首数是正数，那么只要

1. 如果当前数为负数，交换最大累积(max)和最小累积(min)，以保证最大累积中永远保存最大累积
2. 每次保存 max(当前数或前面最大累积*当前数)和 min(当前数或前最小累积*当前数)。以最大为例，如果前面累积乘到当前还不如当前大，那么说明中间必然出现了<=0 的数，重新从当前位置寻找极值即可。

如果首数是负数，那么比较难理解一些，那么就是三种情况：

1. 当前为 0，那么重新开始计算
2. 当前为负，那么负负得正
3. 当前为正，那么舍弃之前累积，因为它*当前数<当前数

153. Find Minimum 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).
Find the minimum element.
You may assume no duplicate exists in the array.
思路（略）：同下一题，只不过不需要考虑重复情况。

154. Find Minimum in Rotated Sorted Array II Hard

Follow up for "Find Minimum in Rotated Sorted Array":
What if *duplicates* are allowed?
Would this affect the run-time complexity? How and why?
Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.
(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
Find the minimum element.
The array may contain duplicates.

```
public int findMin(int[] nums) {
    int l = 0, r = nums.length - 1, mid;
    while (l < r) {
        mid = (l + r) / 2;
        if (nums[mid] < nums[r])
            r = mid;
```

```
        else if (nums[mid] > nums[r])
            l = mid + 1;
        else
            r--;
    }
    return nums[l];
}
```

思路：同二分查找，只不过只有当 l=r 时才结束。缩小范围方法：中间值小于最右侧值，说明最小值在中间值左面或就是中间值（r=mid）；中间值大于最右侧值，说明最小值在中间值右面（l=mid+1）；等于最右值则缩小范围（r--）。最坏情况 O(n)，全部数值都相等，相当于线性查找。

155. Min Stack Easy

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Example:

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> Returns -3.
minStack.pop();
minStack.top();      --> Returns 0.
minStack.getMin();   --> Returns -2.
```

```
class MinStack {

    int min = Integer.MAX_VALUE;
    Stack<Integer> stack;

    /** initialize your data structure here. */
    public MinStack() {
        stack = new Stack<>();
    }

    public void push(int x) {
        if (x <= min) {
            stack.push(min);
            min = x;
        }
        stack.push(x);
    }

    public void pop() {
        if (stack.pop() == min) {
            min = stack.pop();
        }
    }

    public int top() {
        return stack.peek();
    }

    public int getMin() {
        return min;
```

```
    }
}
```
思路1：每次碰到最小值后，就把最小值入栈两次，弹出时如果碰到最小值，就弹出两次。
思路2（略）：使用另一个Stack存当前最小值，与原栈同时出、入。

156. Binary Tree Upside Down Medium

Given a binary tree where all the right nodes are either leaf nodes with a sibling (a left node that shares the same parent node) or empty, flip it upside down and turn it into a tree where the original right nodes turned into left leaf nodes. Return the new root.
For example:
Given a binary tree {1,2,3,4,5},

```
  1
 /\
 2 3
/\
4 5
```

return the root of the binary tree [4,5,2,#,#,3,1].

```
 4
/\
5 2
 /\
 3 1
```

confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ.

```
public TreeNode upsideDownBinaryTree(TreeNode root) {
    if (root == null || root.left == null)
        return root;
    TreeNode newRoot = upsideDownBinaryTree(root.left);
    root.left.left = root.right;
    root.left.right = root;
    root.left = null;
    root.right = null;
    return newRoot;
}
```

思路 **1**：递归求解。结束条件：左结点不存在或当前结点为空。递推方程：每次按要求把右结点作为新的左子结点。

```
public TreeNode upsideDownBinaryTree(TreeNode root) {
    TreeNode cur = root;
    TreeNode next = null, tmp = null, pre = null;
    while (cur != null) {
        next = cur.left;
        cur.left = tmp;
        tmp = cur.right;
        cur.right = pre;
        pre = cur;
        cur = next;
    }
    return pre;
}
```

思路 **2**：同思路 1，只不过通过循环顺着左结点系列进行变换。

157. Read N Characters Given Read4 Medium

The API: `int read4(char *buf)` reads 4 characters at a time from a file.
The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.
By using the `read4` API, implement the function `int read(char *buf, int n)` that reads *n* characters from the file.
Note:
The `read` function will only be called once for each test case.

```
public int read(char[] buf, int n) {
    int ans = 0;
    char[] b = new char[4];
    while (ans < n) {
        int cur = read4(b);
        int i = 0;
        for (; i < cur && i + ans < n; ++i)
            buf[ans + i] = b[i];
        ans += i;
        if (cur < 4 || cur == 0)
            break;
    }
    return ans;
}
```

思路：每次读 4 位或更少存入 buf 中，达到 n 或某次读取不足 4 时结束。

158. Read N Characters Given Read4 II - Call multiple times Hard

The API: `int read4(char *buf)` reads 4 characters at a time from a file.
The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.
By using the `read4` API, implement the function `int read(char *buf, int n)` that reads *n* characters from the file.
Note:
The `read` function may be called multiple times.
Example 1:

```
Given buf = "abc"
read("abc", 1) // returns "a"
read("abc", 2); // returns "bc"
read("abc", 1); // returns ""
```

Example 2:

```
Given buf = "abc"
read("abc", 4) // returns "abc"
read("abc", 1); // returns ""
```

```
char[] b = new char[4];
int i = 4;
int cur = 4;

public int read(char[] buf, int n) {
    int ans = 0;
    int j = 0;
    while (ans < n) {
        if (i == 4) {
            cur = read4(b);
            i = 0;
        }
        int count = 0;
        for (; i < cur && ans + count < n; ++i) {
```

```
            buf[j++] = b[i];
            ++count;
        }
        ans += count;
        if (ans == n || i != 4)
            break;
    }
    return ans;
}
```

思路：不断读取直到达到规定数量 n，二次读取从上次读取位置开始，故保存每次读取完成时的位置，上次位置不=0 时不需要重新读取，直接读缓存。O(n)

159. Longest Substring with At Most Two Distinct Characters Hard

Given a string, find the length of the longest substring T that contains at most 2 distinct characters.
For example, Given s = “eceba”,
T is "ece" which its length is 3.

```
public int lengthOfLongestSubstringTwoDistinct(String s) {
    int ans = 0;
    char[] c = s.toCharArray();
    Map<Character, Integer> m = new HashMap<>();
    for (int i = 0, j = 0, n = c.length; j < n; ++j) {
        if (!m.containsKey(c[j]) && m.keySet().size() == 2) {
            while (m.keySet().size() == 2) {
                char l = c[i];
                m.put(c[i], m.get(c[i]) - 1);
                if (m.get(c[i]) == 0)
                    m.remove(c[i]);
                ++i;
            }
            m.put(c[j], 1);
        } else {
            m.put(c[j], m.getOrDefault(c[j], 0) + 1);
            ans = Math.max(ans, j - i + 1);
        }

    }
    return ans;
}
```

思路 1：滑动窗口。保持窗口内最多两个字母，不断右面扩展，如果变成三个，左面收缩到两个，再进行右面扩展。因为只有两个字母，所以可以省略掉字典，使用两个变量保存这两个字母。

```
public int lengthOfLongestSubstringTwoDistinct(String s) {
    int[] map = new int[256];
    int count = 0, res = 0;
    for (int right = 0, left = 0; right < s.length(); right++) {
        if (map[s.charAt(right)]++ == 0) {
            count++;
        }
        while (count > 2) {
            if (map[s.charAt(left++)]-- == 1) {
                count--;
            }
        }
        res = Math.max(res, right - left + 1);
    }
    return res;
}
```

思路 2：滑动窗口。假设只有 ASKII 字符集，所以可以用一个 256 大小的数组保存每一个字符出现的次数，并且可以计算出当前窗口内字符数。根据字符数做长度收放。

160. Intersection of Two Linked Lists Easy

Write a program to find the node at which the intersection of two singly linked lists begins.
For example, the following two linked lists:

```
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:    b1 → b2 → b3
```

begin to intersect at node c1.

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    if (headA == null || headB == null)
        return null;
    ListNode a = headA;
    ListNode b = headB;
    while (a != b) {
        a = a == null ? headB : a.next;
        b = b == null ? headA : b.next;
    }
    return a;
}
```

思路 1：两指针走到尾后，则再从另一个链表头开始走，如果相遇，则 a=b 且为相遇点。如果不交叉，那么每二轮将同时到达尾，则 a=b=null。
思路 2（略）：双指针。遍历计算两数组长度，较长的先前移多出的位数，两指针同时前移直到相遇或都为 null。

161. One Edit Distance Medium

Given two strings S and T, determine if they are both one edit distance apart.

```
public boolean isOneEditDistance(String s, String t) {
    for (int i = 0; i < Math.min(s.length(), t.length()); i++) {
        if (s.charAt(i) != t.charAt(i)) {
            if (s.length() == t.length())
                return s.substring(i + 1).equals(t.substring(i + 1));
            else if (s.length() < t.length())
                return s.substring(i).equals(t.substring(i + 1));
            else
                return t.substring(i).equals(s.substring(i + 1));
        }
    }
    return Math.abs(s.length() - t.length()) == 1;
}
```

思路：遍历所有对应字符，如果相等则继续；如果不等，则如果长度相同，则看剩下部分是否相同；如果一个短，则看长的当前元素（不含）后和短的当前元素（含）后部分是否相等。

162. Find Peak Element Medium

A peak element is an element that is greater than its neighbors.
Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.
The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.
You may imagine that num[-1] = num[n] = -∞.
For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.
Note:
Your solution should be in logarithmic complexity.

```
public int findPeakElement(int[] nums) {
    int l = 0, r = nums.length - 1, mid;
    while (l < r) {
        mid = (l + r) / 2;
        if (nums[mid] > nums[mid + 1])
            r = mid;
        else
            l = mid + 1;
    }
    return l;
}
```

思路：二分查找。拆分条件：如果中间值大于右值，则最值在中间或左面，否则在右面。

163. Missing Ranges Medium

Given a sorted integer array where **the range of elements are in the inclusive range [*lower*, *upper*]**, return its missing ranges.
For example, given [0, 1, 3, 50, 75], *lower* = 0 and *upper* = 99, return ["2", "4->49", "51->74", "76->99"].

```
public List<String> findMissingRanges(int[] nums, int lower, int upper) {
    List<String> ans = new ArrayList<>();
    int preMiss = lower;
    for(int num : nums) {
        if(num == Integer.MAX_VALUE) {
            if(num == upper)
                --upper;
            break;
        }
        preMiss = preMiss(preMiss, num, ans);
    }
    if(preMiss == upper)
        ans.add(preMiss + "");
    else if(preMiss < upper)
        ans.add(preMiss + "->" + upper);
    return ans;
}

int preMiss(int preMiss, int num, List<String> ans) {
    if(preMiss < num || preMiss == Integer.MAX_VALUE)
        if(preMiss + 1 == num || preMiss == Integer.MAX_VALUE)
            ans.add(preMiss + "");
        else
            ans.add(preMiss + "->" + (num - 1));
    return num + 1;
}
```

思路：逐一遍历所有数字，如果数字大于当前最小边界，则根据大的情况添加失去的范围并把最小边界前移到当前数字+1；如果不大于则直接把最小边界移动到当前数字+1。遍历完成后，根据最小边界和最大边界情况添加或不添加失去的范围。

164. Maximum Gap Hard

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.
Try to solve it in linear time/space.
Return 0 if the array contains less than 2 elements.
You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.

```
public int maximumGap(int[] nums) {
    if (nums == null || nums.length < 2)
        return 0;
    int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
    for (int num : nums) {
        max = Math.max(max, num);
        min = Math.min(min, num);
    }
    int gap = (int) Math.ceil((double) (max - min) / (nums.length - 1));
    int[] minBuckets = new int[nums.length - 1];
    int[] maxBuckets = new int[nums.length - 1];
    Arrays.fill(minBuckets, Integer.MAX_VALUE);
    Arrays.fill(maxBuckets, Integer.MIN_VALUE);
    for (int num : nums) {
        if (num == max || num == min)
            continue;
        int i = (num - min) / gap;
        minBuckets[i] = Math.min(minBuckets[i], num);
        maxBuckets[i] = Math.max(maxBuckets[i], num);
    }
    int ans = Integer.MIN_VALUE, prev = min;
    for (int i = 0; i < minBuckets.length; ++i) {
        if (minBuckets[i] == Integer.MAX_VALUE || maxBuckets[i] == Integer.MIN_VALUE)
            continue;
        ans = Math.max(minBuckets[i] - prev, ans);
        prev = maxBuckets[i];
    }
    ans = Math.max(ans, max - prev);
    return ans;
}
```

思路：桶排序（bucket sort），最小间隔为最大和最小数的差值/（数的个数-1），桶的数量为间隔数。可以证明，要求的间隔一定大于最小间隔。每一个数的索引就是它和最小值差/间隔长度。利用最大值和最小值两个桶，存储最大最小值。最后测算相邻桶中的最大差值（上一桶中的最大值和这一桶中的最小值之间的差）。桶的个数可以参考取模的原理。

165. Compare Version Numbers Medium

Compare two version numbers *version1* and *version2*.
If *version1* > *version2* return 1, if *version1* < *version2* return -1, otherwise return 0.
You may assume that the version strings are non-empty and contain only digits and the `.` character.
The `.` character does not represent a decimal point and is used to separate number sequences.
For instance, `2.5` is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.
Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

```
public int compareVersion(String version1, String version2) {
    String[] v1 = version1.split("\\.");
    String[] v2 = version2.split("\\.");
    int n = v1.length, m = v2.length;
    for (int i = 0; i < Math.min(m, n); ++i) {
        if (!v1[i].equals(v2[i]))
```

```
            return Integer.valueOf(v1[i]).compareTo(Integer.valueOf(v2[i]));
    }
    Integer sv1 = 0, sv2 = 0;
    for (int i = Math.min(m, n); i < Math.max(m, n); ++i) {
        if (v1.length > i)
            sv1 += Integer.valueOf(v1[i]);
        if (v2.length > i)
            sv2 += Integer.valueOf(v2[i]);
    }
    return sv1.compareTo(sv2);
}
```

思路：先用 String.split("\\.")把各个阶的版本号分开，然后从大版本号开始对比。

166. Fraction to Recurring Decimal Medium

Given two integers representing the numerator and denominator of a fraction, return the fraction in string format.
If the fractional part is repeating, enclose the repeating part in parentheses.
For example,

- Given numerator = 1, denominator = 2, return "0.5".
- Given numerator = 2, denominator = 1, return "2".
- Given numerator = 2, denominator = 3, return "0.(6)".

```
public String fractionToDecimal(int numerator, int denominator) {
    if (numerator == 0)
        return "0";
    int sign = numerator < 0 != denominator < 0 ? -1 : 1;
    long dividend = Math.abs((long) numerator);
    long divisor = Math.abs((long) denominator);
    long cur = dividend / divisor;
    long remainder = dividend % divisor;
    StringBuilder ans = new StringBuilder();
    if (remainder == 0) {
        ans.append(cur * sign);
        return ans.toString();
    }
    ans.append((sign < 0 ? "-" : "")).append(cur).append(".");
    Map<Long, Integer> map = new HashMap<>();
    while (remainder > 0) {
        if (map.containsKey(remainder)) {
            ans.insert(map.get(remainder), "(");
            ans.append(")");
            break;
        }
        map.put(remainder, ans.length());
        remainder *= 10;
        cur = remainder / divisor;
        ans.append(cur);
        remainder %= divisor;
    }
    return ans.toString();
}
```

思路：先判断是否为负添加负号。然后把整数部分加上，看有没有小数部分（余数），如果有，余数存入Map（以寻找重复的小数值），然后余数*10（即到下一位）后再除除数得到第一位小数，再取余，重复这个过程，如果碰到重复的余数，则加上括号。

167. Two Sum II - Input array is sorted Easy

Given an array of integers that is already ***sorted in ascending order***, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.
You may assume that each input would have *exactly* one solution and you may not use the *same* element twice.
Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2

```
public int[] twoSum(int[] numbers, int target) {
    int l = 0, r = numbers.length - 1, sum;
    while (l < r) {
        sum = numbers[l] + numbers[r];
        if (sum == target)
            return new int[] { l + 1, r + 1 };
        if (sum > target)
            r--;
        else
            l++;
    }
    return new int[] { -1, -1 };
}
```

思路：夹逼法。两边开始，如果两数和大于目标，则右侧指针左移，反之左侧指针右移。

168. Excel Sheet Column Title Easy

Given a positive integer, return its corresponding column title as appear in an Excel sheet.
For example:

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB
```

```
public String convertToTitle(int n) {
    if (n == 0)
        return "";
    return convertToTitle(--n / 26) + (char) ('A' + n % 26);
}
```

思路：大于 26 的部分需要递归求解，小于 26 的部分看其与 A 的距离确定用哪个字母。

169. Majority Element Easy

Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** ⌊ n/2 ⌋ times.
You may assume that the array is non-empty and the majority element always exist in the array.

```
public int majorityElement(int[] nums) {
    int count = 0, candidateIdx = -1;
    for (int i = 0; i < nums.length; i++) {
        if (count == 0) {
            candidateIdx = i;
        }
        if (nums[i] == nums[candidateIdx]) {
            count++;
        } else {
            count--;
        }
    }
    return nums[candidateIdx];
```

}
思路：非主元素的个数和必少于主元素的个数，所以统计任一元素个数，如果碰到与其相等就+1，不等就-1，数量归零时就换下一元素，最后所剩元素必为主元素。只需要移动 index，最后一次性取值。

170. Two Sum III - Data structure design Easy

Design and implement a TwoSum class. It should support the following operations: add and find.
add - Add the number to an internal data structure.
find - Find if there exists any pair of numbers which sum is equal to the value.
For example,

```
add(1); add(3); add(5);
find(4) -> true
find(7) -> false
```

```
class TwoSum {
    private int min;
    private int max;
    Set<Integer> set;
    List<Integer> list;

    /** Initialize your data structure here. */
    public TwoSum() {
        min = Integer.MAX_VALUE;
        max = Integer.MIN_VALUE;
        list = new ArrayList<>();
    }

    /** Add the number to an internal data structure.. */
    public void add(int number) {
        list.add(number);
        min = Math.min(min, number);
        max = Math.max(max, number);
    }

    /** Find if there exists any pair of numbers which sum is equal to the value. */
    public boolean find(int value) {
        if (value < min * 2 || value > max * 2)
            return false;
        set = new HashSet<>();
        for (Integer i : list) {
            if (set.contains(value - i))
                return true;
            set.add(i);
        }
        return false;
    }
}
```

思路：用 list 记录所有数字，查找时遍历所有数字，看有无合适的解，方法如第 1 题。

171. Excel Sheet Column Number Easy

Related to question Excel Sheet Column Title
Given a column title as appear in an Excel sheet, return its corresponding column number.
For example:

```
  A -> 1
  B -> 2
  C -> 3
```

```
...
Z -> 26
AA -> 27
AB -> 28
```

```
public int titleToNumber(String s) {
    int ans = 0;
    for (char c : s.toCharArray()) {
        ans = ans * 26 + (c - 'A' + 1);
    }
    return ans;
}
```

思路：从高位向低位逐位处理即可，与 A 的距离+1 就是其值，每次进位*26。

172. Factorial Trailing Zeroes Easy

Given an integer *n*, return the number of trailing zeroes in *n*!.

Note: Your solution should be in logarithmic time complexity.

```
public int trailingZeroes(int n) {
    return n == 0 ? 0 : n / 5 + trailingZeroes(n / 5);
}
```

思路：其实是数 5 的个数，因为在阶乘中有 5 必有 2。所以对于 n!，n/5 得到 n 中 5 的个数，但是如果本身是 5 的倍数，那么 5 的个数还将加倍。例如 25=25/5+5/5 = 6。如果 n=125，那么其第二阶队列为 25,24,23…，再一阶为 5,4,3…。

21	22	23	24	25	25/5=5
16	17	18	19	20	20/5=4
11	12	13	14	15	15/5=3
6	7	8	9	10	10/5=2
1	2	3	4	5	5/5=1

173. Binary Search Tree Iterator Medium

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

Note: next() and hasNext() should run in average O(1) time and uses O(*h*) memory, where *h* is the height of the tree.

```
class BSTIterator {
    private Stack<TreeNode> stack = new Stack<TreeNode>();

    public BSTIterator(TreeNode root) {
        pushAll(root);
    }

    public boolean hasNext() {
        return !stack.isEmpty();
    }

    public int next() {
        TreeNode tmpNode = stack.pop();
        pushAll(tmpNode.right);
        return tmpNode.val;
    }

    private void pushAll(TreeNode node) {
        for (; node != null; stack.push(node), node = node.left)
            ;
    }
```

}
思路：二叉搜索树特点：左孩子总是较小，右孩子总是较大，所以每次要到最远的左结点，缓存到Stack中，如果弹出当前元素，则将对应右结点及相应的左枝缓存入Stack中（这个枝上的结点值仍然小于其祖结点）。

174. Dungeon Game Hard

The demons had captured the princess (**P**) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (**K**) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.
The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.
Some of the rooms are guarded by demons, so the knight loses health (*negative* integers) upon entering these rooms; other rooms are either empty (*0's*) or contain magic orbs that increase the knight's health (*positive* integers).
In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.
Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.
For example, given the dungeon below, the initial health of the knight must be at least **7** if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.

-2 (K)	-3	3
-5	-10	1
10	30	-5 (P)

Notes:

- The knight's health has no upper bound.
- Any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

```
public int calculateMinimumHP(int[][] dungeon) {
    int m = dungeon.length, n = dungeon[0].length;
    int[][] h = new int[m][n];
    h[m - 1][n - 1] = Math.max(1, 1 - dungeon[m - 1][n - 1]);
    for (int i = m - 2; i >= 0; --i)
        h[i][n - 1] = Math.max(1, h[i + 1][n - 1] - dungeon[i][n - 1]);
    for (int j = n - 2; j >= 0; --j)
        h[m - 1][j] = Math.max(1, h[m - 1][j + 1] - dungeon[m - 1][j]);
    for (int i = m - 2; i >= 0; --i)
        for (int j = n - 2; j >= 0; --j)
            h[i][j] = Math.min(Math.max(1, h[i + 1][j] - dungeon[i][j]),
Math.max(1, h[i][j + 1] - dungeon[i][j]));
    return h[0][0];
}
```

思路：动态规划。因为要找起始点最小值，最终结点开始往回找，看每一点从下面和从右面回来所需要的最小生命力，取最小值，逐步往前推，到[0,0]位置就得到解。

175. Combine Two Tables Easy

```
Create table If Not Exists Person (PersonId int, FirstName varchar(255), LastName varchar(255));
Create table If Not Exists Address (AddressId int, PersonId int, City varchar(255), State varchar(255));
Truncate table Person;
insert into Person (PersonId, LastName, FirstName) values ('1', 'Wang', 'Allen');
```

```
Truncate table Address;
insert into Address (AddressId, PersonId, City, State) values ('1', '2', 'New York City', 'New York');
```

Table: Person

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| PersonId    | int     |
| FirstName   | varchar |
| LastName    | varchar |
+-------------+---------+
```

PersonId is the primary key column for this table.

Table: Address

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| AddressId   | int     |
| PersonId    | int     |
| City        | varchar |
| State       | varchar |
+-------------+---------+
```

AddressId is the primary key column for this table.

Write a SQL query for a report that provides the following information for each person in the Person table, regardless if there is an address for each of those people:

```
FirstName, LastName, City, State
```

```
SELECT Person.FirstName,
 Person.LastName,
 Address.City,
 Address.State
FROM Person
LEFT JOIN Address
ON Person.PersonId = Address.PersonId
```

思路：利用 PersonId 连接两表。

176. Second Highest Salary Easy

```
Create table If Not Exists Employee (Id int, Salary int);
Truncate table Employee;
insert into Employee (Id, Salary) values ('1', '100');
insert into Employee (Id, Salary) values ('2', '200');
insert into Employee (Id, Salary) values ('3', '300');
```

Write a SQL query to get the second highest salary from the Employee table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
```

```
| 3  | 300   |
+----+--------+
```

For example, given the above Employee table, the query should return 200 as the second highest salary. If there is no second highest salary, then the query should return null.

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| 200                 |
+---------------------+
```

```
SELECT MAX(Salary) SecondHighestSalary
FROM Employee
WHERE Salary <
 (SELECT MAX(Salary) FROM Employee)
```

思路：先找最大，然后找小于最大的最大（即第二大）。

177. Nth Highest Salary Medium

Write a SQL query to get the n^{th} highest salary from the Employee table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the n^{th} highest salary where $n = 2$ is 200. If there is no n^{th} highest salary, then the query should return null.

```
+------------------------+
| getNthHighestSalary(2) |
+------------------------+
| 200                    |
+------------------------+
```

```
CREATE FUNCTION getNthHighestSalary(N IN NUMBER) RETURN NUMBER IS
result NUMBER;
BEGIN
  /* Write your PL/SQL query statement below */
  SELECT Salary into result
  FROM
  (
    SELECT Salary, rank() over (order by Salary desc) rn
        FROM (SELECT distinct Salary from Employee)
  ) where rn = N;
  RETURN result;
END;
```

思路：取最大的 n 个，最小的一个就是。

178. Rank Scores Medium

```
Create table If Not Exists Scores (Id int, Score DECIMAL(3,2));
Truncate table Scores;
insert into Scores (Id, Score) values ('1', '3.5');
insert into Scores (Id, Score) values ('2', '3.65');
insert into Scores (Id, Score) values ('3', '4.0');
insert into Scores (Id, Score) values ('4', '3.85');
insert into Scores (Id, Score) values ('5', '4.0');
insert into Scores (Id, Score) values ('6', '3.65');
```

Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no "holes" between ranks.

```
+----+-------+
| Id | Score |
+----+-------+
| 1  | 3.50  |
| 2  | 3.65  |
| 3  | 4.00  |
| 4  | 3.85  |
| 5  | 4.00  |
| 6  | 3.65  |
+----+-------+
```

For example, given the above Scores table, your query should generate the following report (order by highest score):

```
+-------+------+
| Score | Rank |
+-------+------+
| 4.00  | 1    |
| 4.00  | 1    |
| 3.85  | 2    |
| 3.65  | 3    |
| 3.65  | 3    |
| 3.50  | 4    |
+-------+------+
```

```
SELECT SCORE,
DENSE_RANK () OVER (ORDER BY SCORE DESC) as RANK
FROM Scores
```

179. Largest Number Medium

Given a list of non negative integers, arrange them such that they form the largest number.

For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330.

Note: The result may be very large, so you need to return a string instead of an integer.

```
public String largestNumber(int[] nums) {
    String[] array = Arrays.stream(nums).mapToObj(String::valueOf).toArray(String[]::new);
    Arrays.sort(array, (String s1, String s2) -> (s2 + s1).compareTo(s1 + s2));
    return Arrays.stream(array).reduce((x, y) -> x.equals("0") ? y : x + y).get();
}
```

思路：先转换成字符串数组，然后按拼接后大小排序，然后连接到一起。

180. Consecutive Numbers Medium

```
Create table If Not Exists Logs (Id int, Num int);
Truncate table Logs;
insert into Logs (Id, Num) values ('1', '1');
insert into Logs (Id, Num) values ('2', '1');
insert into Logs (Id, Num) values ('3', '1');
insert into Logs (Id, Num) values ('4', '2');
insert into Logs (Id, Num) values ('5', '1');
insert into Logs (Id, Num) values ('6', '2');
insert into Logs (Id, Num) values ('7', '2');
```

Write a SQL query to find all numbers that appear at least three times consecutively.

```
+----+-----+
| Id | Num |
+----+-----+
| 1  |  1  |
| 2  |  1  |
| 3  |  1  |
| 4  |  2  |
| 5  |  1  |
| 6  |  2  |
| 7  |  2  |
+----+-----+
```

For example, given the above Logs table, 1 is the only number that appears consecutively for at least three times.

```
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
+-----------------+
```

```
SELECT DISTINCT l1.Num ConsecutiveNums
FROM Logs l1,
  Logs l2,
  Logs l3
WHERE l1.Id=l2.Id-1
AND l2.Id  =l3.Id-1
AND l1.Num =l2.Num
AND l2.Num =l3.Num
```

思路：ID 之间相差 1 点数值相等的所有数

181. Employees Earning More Than Their Managers Easy

```
Create table If Not Exists Employee (Id int, Name varchar(255), Salary int, ManagerId int);
Truncate table Employee;
insert into Employee (Id, Name, Salary, ManagerId) values ('1', 'Joe', '70000', '3');
insert into Employee (Id, Name, Salary, ManagerId) values ('2', 'Henry', '80000', '4');
insert into Employee (Id, Name, Salary, ManagerId) values ('3', 'Sam', '60000', 'None');
insert into Employee (Id, Name, Salary, ManagerId) values ('4', 'Max', '90000', 'None');
```

The Employee table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

```
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

```
SELECT E1.Name Employee
FROM Employee AS E1,
 Employee   AS E2
WHERE E1.ManagerId = E2.Id
AND E1.Salary     > E2.Salary
```

思路：通过 ManagerId 来找对应 Manager，条件就是员工工资大于 Manager 工资。

182. Duplicate Emails Easy

```
Create table If Not Exists Person (Id int, Email varchar(255));
Truncate table Person;
insert into Person (Id, Email) values ('1', 'a@b.com');
insert into Person (Id, Email) values ('2', 'c@d.com');
insert into Person (Id, Email) values ('3', 'a@b.com');
```

Write a SQL query to find all duplicate emails in a table named Person.

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

For example, your query should return the following for the above table:

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

Note: All emails are in lowercase.

```
select distinct p1.email
from Person p1, Person p2
```

```
where p1.email = p2.email
and p1.id != p2.id
```
思路：取 Id 不等但 email 相等的 email 并取唯一。

183. Customers Who Never Order Easy

```
Create table If Not Exists Customers (Id int, Name varchar(255));
Create table If Not Exists Orders (Id int, CustomerId int);
Truncate table Customers;
insert into Customers (Id, Name) values ('1', 'Joe');
insert into Customers (Id, Name) values ('2', 'Henry');
insert into Customers (Id, Name) values ('3', 'Sam');
insert into Customers (Id, Name) values ('4', 'Max');
Truncate table Orders;
insert into Orders (Id, CustomerId) values ('1', '3');
insert into Orders (Id, CustomerId) values ('2', '1');
```

Suppose that a website contains two tables, the Customers table and the Orders table. Write a SQL query to find all customers who never order anything.
Table: Customers.

```
+----+-------+
| Id | Name  |
+----+-------+
| 1  | Joe   |
| 2  | Henry |
| 3  | Sam   |
| 4  | Max   |
+----+-------+
```

Table: Orders.

```
+----+------------+
| Id | CustomerId |
+----+------------+
| 1  | 3          |
| 2  | 1          |
+----+------------+
```

Using the above tables as example, return the following:

```
+-----------+
| Customers |
+-----------+
| Henry     |
| Max       |
+-----------+
```

```
select customers.name as 'Customers'
from customers
where customers.id not in
(
  select customerid from orders
)
```
思路：查找在 Order 表中没有出现的 Customer，通过 CustomerId 连接两表。

184. Department Highest Salary Medium

```
Create table If Not Exists Employee (Id int, Name varchar(255), Salary int, DepartmentId int);
Create table If Not Exists Department (Id int, Name varchar(255));
Truncate table Employee;
insert into Employee (Id, Name, Salary, DepartmentId) values ('1', 'Joe', '70000', '1');
insert into Employee (Id, Name, Salary, DepartmentId) values ('2', 'Henry', '80000', '2');
insert into Employee (Id, Name, Salary, DepartmentId) values ('3', 'Sam', '60000', '2');
insert into Employee (Id, Name, Salary, DepartmentId) values ('4', 'Max', '90000', '1');
Truncate table Department;
insert into Department (Id, Name) values ('1', 'IT');
insert into Department (Id, Name) values ('2', 'Sales');
```

The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
```

The Department table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, Max has the highest salary in the IT department and Henry has the highest salary in the Sales department.

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| Sales      | Henry    | 80000  |
+------------+----------+--------+
```

```
SELECT d.Name Department, e.Name Employee, e.salary Salary
FROM Employee e, Department d
WHERE e.DepartmentId = d.id
and (e.DepartmentId,e.salary) in
(select departmentId, MAX(salary)
from Employee GROUP BY departmentId)
```

思路：通过 Employee 表可以取得部分最高工资及部门 ID，再与部门表连接得到部门名称。

185. Department Top Three Salaries Hard

```
Create table If Not Exists Employee (Id int, Name varchar(255), Salary int, DepartmentId int);
Create table If Not Exists Department (Id int, Name varchar(255));
```

```
Truncate table Employee;
insert into Employee (Id, Name, Salary, DepartmentId) values ('1', 'Joe', '70000', '1');
insert into Employee (Id, Name, Salary, DepartmentId) values ('2', 'Henry', '80000', '2');
insert into Employee (Id, Name, Salary, DepartmentId) values ('3', 'Sam', '60000', '2');
insert into Employee (Id, Name, Salary, DepartmentId) values ('4', 'Max', '90000', '1');
Truncate table Department;
insert into Department (Id, Name) values ('1', 'IT');
insert into Department (Id, Name) values ('2', 'Sales');
```

The Employee table holds all employees. Every employee has an Id, and there is also a column for the department Id.

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Henry | 80000  | 2            |
| 3  | Sam   | 60000  | 2            |
| 4  | Max   | 90000  | 1            |
| 5  | Janet | 69000  | 1            |
| 6  | Randy | 85000  | 1            |
+----+-------+--------+--------------+
```

The Department table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows.

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| IT         | Randy    | 85000  |
| IT         | Joe      | 70000  |
| Sales      | Henry    | 80000  |
| Sales      | Sam      | 60000  |
+------------+----------+--------+
```

```
SELECT
  d.Name AS 'Department', e1.Name AS 'Employee', e1.Salary
FROM
  Employee e1
    JOIN
  Department d ON e1.DepartmentId = d.Id
WHERE
  3 > (SELECT
      COUNT(DISTINCT e2.Salary)
    FROM
      Employee e2
    WHERE
```

```
        e2.Salary > e1.Salary
          AND e1.DepartmentId = e2.DepartmentId
    )
```

186. Reverse Words in a String II Medium

Given an input string, reverse the string word by word. A word is defined as a sequence of non-space characters.
The input string does not contain leading or trailing spaces and the words are always separated by a single space.
For example,
Given s = "the sky is blue",
return "blue is sky the".
Could you do it *in-place* without allocating extra space?

```
public void reverseWords(char[] s) {
    reverse(s, 0, s.length - 1);
    int start = 0;
    for (int i = 0; i < s.length; i++) {
        if (s[i] == ' ') {
            reverse(s, start, i - 1);
            start = i + 1;
        }
    }
    reverse(s, start, s.length - 1);
}

public void reverse(char[] s, int start, int end) {
    while (start < end) {
        char temp = s[start];
        s[start] = s[end];
        s[end] = temp;
        start++;
        end--;
    }
}
```

思路：先把每个单词反转，再把字符串整体反转。

187. Repeated DNA Sequences Medium

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.
Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.
For example,

```
Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",
Return:
["AAAAACCCCC", "CCCCCAAAAA"].
```

```
public List<String> findRepeatedDnaSequences(String s) {
    Set seen = new HashSet(), repeated = new HashSet();
    for (int i = 0; i + 9 < s.length(); i++) {
        String ten = s.substring(i, i + 10);
        if (!seen.add(ten))
            repeated.add(ten);
    }
    return new ArrayList(repeated);
}
```

思路：取每十个连续字符作为一个索引，如果有重复的则为一个候选项。利用 Set 的 add 返回假判断是否已经出现过。

188. Best Time to Buy and Sell Stock IV Hard

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.
Design an algorithm to find the maximum profit. You may complete at most **k** transactions.
Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

```
public int maxProfit(int k, int[] prices) {
    int n = prices.length;
    if (n <= 1)
        return 0;
    if (k >= n / 2)
        return maxCum(prices);
    int[] balance = new int[k + 1];
    Arrays.fill(balance, Integer.MIN_VALUE);
    int[] profit = new int[k + 1];
    for (int price : prices) {
        for (int j = 1; j <= k; j++) {
            balance[j] = Math.max(profit[j - 1] - price, balance[j]);
            profit[j] = Math.max(balance[j] + price, profit[j]);
        }
    }
    return profit[k];
}

int maxCum(int[] prices) {
    int ans = 0;
    for (int i = 0; i < prices.length - 1; ++i)
        if (prices[i + 1] > prices[i])
            ans += prices[i + 1] - prices[i];
    return ans;
}
```

思路：同 122 和 123 题。如果次数不小于天数一半，则可任意多次交易，只要上涨就计入即为最大收益。如果次数不到天数一半，二维动态规。O(kn)

189. Rotate Array Easy

Rotate an array of *n* elements to the right by *k* steps.
For example, with *n* = 7 and *k* = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].
Note:
Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
Hint:
Could you do it in-place with O(1) extra space?
Related problem: Reverse Words in a String II

```
public void rotate(int[] nums, int k) {
    int n = nums.length;
    k %= n;
    reverse(nums, 0, n - 1);
    reverse(nums, 0, k - 1);
    reverse(nums, k, n - 1);
}

void reverse(int[] nums, int i, int j) {
    while (i < j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
        i++;
        j--;
    }
}
```

思路：先整体翻转，再对各部分翻转。O(n)

190. Reverse Bits Easy

Reverse bits of a given 32 bits unsigned integer.
For example, given input 43261596 (represented in binary as **00000010100101000001111010011100**), return 964176192 (represented in binary as **00111001011110000010100101000000**).
Follow up:
If this function is called many times, how would you optimize it?
Related problem: Reverse Integer

```
public int reverseBits(int n) {
    int result = 0;
    for (int i = 0; i < 32; i++) {
        result += n & 1;
        n >>>= 1;
        if (i < 31)
            result <<= 1;
    }
    return result;
}
```

思路：与第 7 题一样，只不过现在通过与操作取最低位值，然后再位移反转。注意最后一次无须位移。

191. Number of 1 Bits Easy

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).
For example, the 32-bit integer '11' has binary representation `00000000000000000000000000001011`, so the function should return 3.

```
public int hammingWeight(int n) {
    int bits = 0;
    int mask = 1;
    for (int i = 0; i < 32; i++) {
        if ((n & mask) != 0) {
            bits++;
        }
        mask <<= 1;
    }
    return bits;
}
```

思路：就是数 1 的个数，通过与 1 按位与即可得。每次把 1 左移一位看下一位是否是 1（也可以数字右移）。

192. Word Frequency Medium

Write a bash script to calculate the frequency of each word in a text file `words.txt`.
For simplicity sake, you may assume:

- `words.txt` contains only lowercase characters and space `' '` characters.
- Each word must consist of lowercase characters only.
- Words are separated by one or more whitespace characters.

For example, assume that `words.txt` has the following content:

```
the day is sunny the the
the sunny is is
```

Your script should output the following, sorted by descending frequency:

```
the 4
```

```
is 3
sunny 2
day 1
```

Note:
Don't worry about handling ties, it is guaranteed that each word's frequency count is unique.
Hint:
Could you write it in one-line using Unix pipes?
Solution
cat words.txt | **tr -s** ' ' '\n' | **sort** | uniq -c | **sort** -r | awk '{ print $2, $1 }'
tr -s: truncate the string with target string, but only remaining one instance (e.g. multiple whitespaces)
sort: To make the same string successive so that uniq could count the same string fully and correctly.
uniq -c: uniq is used to filter out the repeated lines which are successive, -c means counting
sort -r: -r means sorting in descending order
awk '{ print $2, $1 }': To format the output, see here.

193. Valid Phone Numbers Easy

Given a text file file.txt that contains list of phone numbers (one per line), write a one liner bash script to print all valid phone numbers.
You may assume that a valid phone number must appear in one of the following two formats: (xxx) xxx-xxxx or xxx-xxx-xxxx. (x means a digit)
You may also assume each line in the text file must not contain leading or trailing white spaces.
For example, assume that file.txt has the following content:

```
987-123-4567
123 456 7890
(123) 456-7890
```

Your script should output the following valid phone numbers:

```
987-123-4567
(123) 456-7890
```

Solution
Using grep:
grep -P '^(\d{3}-|\(\d{3}\))\d{3}-\d{4}$' file.txt
Using sed:
sed -n -r '/^([0-9]{3}-|\([0-9]{3}\))[0-9]{3}-[0-9]{4}$/p' file.txt
Using awk:
awk '/^([0-9]{3}-|\([0-9]{3}\))[0-9]{3}-[0-9]{4}$/' file.txt

194. Transpose File Medium

Given a text file file.txt, transpose its content.
You may assume that each row has the same number of columns and each field is separated by the ' ' character.
For example, if file.txt has the following content:

```
name age
alice 21
ryan 30
```

Output the following:

```
name alice ryan
```

```
age 21 30
```

Solution

```
awk '
{
  for (i = 1; i <= NF; i++) {
    if(NR == 1) {
      s[i] = $i;
    } else {
      s[i] = s[i] " " $i;
    }
  }
}
END {
  for (i = 1; s[i] != ""; i++) {
    print s[i];
  }
}' file.txt
```

Bash:

```
ncol=`head -n1 file.txt | wc -w`
for i in `seq 1 $ncol`
do
  echo `cut -d' ' -f$i file.txt`
done
```

195. Tenth Line Easy

How would you print just the 10th line of a file?

For example, assume that file.txt has the following content:

```
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

Your script should output the tenth line, which is:

```
Line 10
```

Hint:

1. If the file contains less than 10 lines, what should you output?
2. There's at least three different solutions. Try to explore all possibilities.

Solution

```
# Solution 1
cnt=0
while read line && [ $cnt -le 10 ]; do
 let 'cnt = cnt + 1'
 if [ $cnt -eq 10 ]; then
  echo $line
```

```
    exit 0
  fi
done < file.txt
# Solution 2
awk 'FNR == 10 {print }'  file.txt
# OR
awk 'NR == 10' file.txt
# Solution 3
sed -n 10p file.txt
# Solution 4
tail -n+10 file.txt|head -1
```

196. Delete Duplicate Emails Easy

Write a SQL query to delete all duplicate email entries in a table named Person, keeping only unique emails based on its *smallest* **Id**.

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
+----+------------------+
Id is the primary key column for this table.
```

For example, after running your query, the above Person table should have the following rows:

```
+----+------------------+
| Id | Email            |
+----+------------------+
| 1  | john@example.com |
| 2  | bob@example.com  |
+----+------------------+
```

```
DELETE p1 FROM Person p1, Person p2
WHERE
   p1.Email = p2.Email AND p1.Id > p2.Id
```

思路：根据 Email 相等且 Id 不等删除冗余 Email。

197. Rising Temperature Easy

```
Create table If Not Exists Weather (Id int, Date date, Temperature int);
Truncate table Weather;
insert into Weather (Id, Date, Temperature) values ('1', '2015-01-01', '10');
insert into Weather (Id, Date, Temperature) values ('2', '2015-01-02', '25');
insert into Weather (Id, Date, Temperature) values ('3', '2015-01-03', '20');
insert into Weather (Id, Date, Temperature) values ('4', '2015-01-04', '30');
```

Given a Weather table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

```
+---------+------------+------------------+
| Id(INT) | Date(DATE) | Temperature(INT) |
+---------+------------+------------------+
```

```
|    1 | 2015-01-01 |          10 |
|    2 | 2015-01-02 |          25 |
|    3 | 2015-01-03 |          20 |
|    4 | 2015-01-04 |          30 |
+------+------------+-----------------+
```

For example, return the following Ids for the above Weather table:

```
+----+
| Id |
+----+
| 2 |
| 4 |
+----+
```

SELECT w2.id **FROM** Weather w1, Weather w2 **WHERE** w2.Temperature>w1.Temperature **AND** (w2.RecordDate-w1.RecordDate) = 1

思路：同表使用两个别名，则可对两条记录间进行对比。温度高且天数大一的为解。

198. House Robber Easy

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

```
public int rob(int[] nums) {
    int n = nums.length;
    if (n < 1)
        return 0;
    int rob = nums[0], nRob = 0, lastNr = 0;
    for (int i = 1; i < n; ++i) {
        nRob = Math.max(rob, lastNr);
        rob = lastNr + nums[i];
        lastNr = nRob;
    }
    return Math.max(rob, nRob);
}
```

思路：动态规划。每次记录当前房子抢（需要上次不抢的最值 preMax）和不抢的结果。

199. Binary Tree Right Side View Medium

Given a binary tree, imagine yourself standing on the *right* side of it, return the values of the nodes you can see ordered from top to bottom.

For example:

Given the following binary tree,

```
  1      <---
/ \
2   3     <---
\   \
 5   4    <---
```

You should return [1, 3, 4].

```
public List<Integer> rightSideView(TreeNode root) {
    List<Integer> ans = new ArrayList<>();
```

```
    Queue<TreeNode> q = new LinkedList<>(), nq = new LinkedList<>();
    if (root != null)
        q.offer(root);
    while (!q.isEmpty()) {
        root = q.poll();
        if (root.left != null)
            nq.offer(root.left);
        if (root.right != null)
            nq.offer(root.right);
        if (q.isEmpty()) {
            ans.add(root.val);
            Queue<TreeNode> temp = q;
            q = nq;
            nq = temp;
        }
    }
    return ans;
}
```

思路 1：与第 102 题思路一致。不同就是只存每行最后一个结点的值。

```
int max = -1;
public List<Integer> rightSideView(TreeNode root) {
    List<Integer> ans = new ArrayList<>();
    if (root != null)
        dfs(root, 0, ans);
    return ans;
}
public void dfs(TreeNode root, int level, List<Integer> ans) {
    if (level > max) {
        max = level;
        ans.add(root.val);
    }
    if (root.right != null)
        dfs(root.right, level + 1, ans);
    if (root.left != null)
        dfs(root.left, level + 1, ans);
}
```

思路 2：递归。因为总是先考虑右面元素，所以先右后左深度优先前序遍历，如果碰到新层元素，则入结果，为了判断是否新层元素，引入 max。

200. Number of Islands Medium

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
11110
11010
11000
00000
```

Answer: 1

Example 2:

```
11000
11000
00100
00011
```

Answer: 3

```
public int numIslands(char[][] grid) {
    int ans = 0, m = grid.length;
    if (m == 0)
        return ans;
    int n = grid[0].length;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (grid[i][j] == '1') {
                bfs(grid, i, j);
                ++ans;
            }
        }
    }
    return ans;
}

void dfs(char[][] grid, int i, int j) {
    if (i == -1 || j == grid[0].length || j == -1 || i == grid.length || grid[i][j] != '1')
        return;
    grid[i][j] = '2';
    dfs(grid, i + 1, j);
    dfs(grid, i - 1, j);
    dfs(grid, i, j + 1);
    dfs(grid, i, j - 1);
}
```

思路：遍历矩阵中所有点，如果当前点不为 0，则是一个岛，DFS 求解当前岛，并把遇到的位置全部置 0，岛数量+1。O(n)

201. Bitwise AND of Numbers Range Medium

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.
For example, given the range [5, 7], you should return 4.

```
public int rangeBitwiseAnd(int m, int n) {
    int bit = 0;
    while (m != n) {
        m >>= 1;
        n >>= 1;
        ++bit;
    }
    return n << bit;
}
```

思路：m 到 n 的数中，在 m 与 n 不同值的最高位以其后续位（二进制表示时），都有是零的情况，所以最终一定是 m 和 n 高位相同而后面都为零的数。也可以逐位求直到找到这个位：while(n > m) n != n-1;这时 n 就是解

202. Happy Number Easy

Write an algorithm to determine if a number is "happy".
A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.
Example: 19 is a happy number

- $1^2 + 9^2 = 82$
- $8^2 + 2^2 = 68$
- $6^2 + 8^2 = 100$
- $1^2 + 0^2 + 0^2 = 1$

```
public boolean isHappy(int n) {
    Set<Integer> sean = new HashSet<>();
    while (sean.add(n)) {
        int cur = n;
        n = 0;
        while (cur > 0) {
            int a = cur % 10;
            n += a * a;
            cur /= 10;
        }
        if (n == 1)
            return true;
    }
    return false;
}
```

思路：通过取余和取模计算每一位的平方然后加起来就形成下一个数，如果有数重复则停止返回假（利用 Set.add 返回假），如果和为 1 则为真。

203. Remove Linked List Elements Easy

Remove all elements from a linked list of integers that have value ***val***.

Example

Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, ***val*** = 6

Return: 1 --> 2 --> 3 --> 4 --> 5

```
public ListNode removeElements(ListNode head, int val) {
    if (head == null)
        return head;
    head.next = removeElements(head.next, val);
    return head.val == val ? head.next : head;
}
public ListNode removeElements(ListNode head, int val) {
    ListNode pre = new ListNode(0), cur = pre;
    pre.next = head;
    while (cur.next != null) {
        if (cur.next.val == val)
            cur.next = cur.next.next;
        else
            cur = cur.next;

    }
    return pre.next;
}
```

思路 **1**：递归求解。递推方程：下一点为下一点（如果下一点与目标值不相同）或再下一点（如果下一点值与目标值相同）。结束条件：结点为空。

思路 **2**：双指针法。一前一后两指针，后指针先指向头元素前，前指针指向头元素。如果头元素值与目标值一致，则后指针指向第二个元素前，前指针前移到第二个元素。重复直到遍历链表。

204. Count Primes Easy

Description:

Count the number of prime numbers less than a non-negative number, ***n***.

```
public int countPrimes(int n) {
    boolean[] np = new boolean[n];
    int m = 0;
    for (int i = 2; i < n; ++i) {
        if (!np[i])
            ++m;
        for (int j = 2; i * j < n; ++j) {
            np[i * j] = true;
        }
```

```
    }
    return m;
}
```
思路：根据素数（也称质数，Prime Number）定义：除 1 和它本身外没有其他除数的数。那么非 Prime 一定有除 1 和它本身（即[2,n-1]）另一个因子。所以可以通过排除法，用一个布尔数组保存某位是否不是素数（默认否，即是 Prime），默认全部为素数，通过乘法标记所有非素数位置为真，即非素数。过程中只要计数是素数的个数即可。

205. Isomorphic Strings Easy

Given two strings ***s*** and ***t***, determine if they are isomorphic.
Two strings are isomorphic if the characters in ***s*** can be replaced to get ***t***.
All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.
For example,
Given "egg", "add", return true.
Given "foo", "bar", return false.
Given "paper", "title", return true.
Note:
You may assume both ***s*** and ***t*** have the same length.

```
public boolean isIsomorphic(String s, String t) {
    int[] m = new int[512];
    for (int i = 0; i < s.length(); i++) {
        if (m[s.charAt(i)] != m[t.charAt(i) + 256])
            return false;
        m[s.charAt(i)] = m[t.charAt(i) + 256] = i + 1;
    }
    return true;
}
public boolean isIsomorphic(String s, String t) {
    Map<Character, Integer> m = new HashMap<>(), o = new HashMap<>();
    int n = s.length();
    for (int i = 0; i < n; ++i) {
        if (!m.getOrDefault(s.charAt(i), 0).equals(o.getOrDefault(t.charAt(i), 0))) {
            return false;
        }
        m.put(s.charAt(i), i + 1);
        o.put(t.charAt(i), i + 1);
    }
    return true;
}
```
思路：假设两字符串中只有 ASCII 编码（256）。那么可以把它们的值的位置 Map 到数组中，如果再次遇到相应的字符，看他们 Map 到的位置值是否相同即可。也可以用两个 Map 模拟。

206. Reverse Linked List Easy

Reverse a singly linked list.
Hint:
A linked list can be reversed either iteratively or recursively. Could you implement both?

```
public ListNode reverseList(ListNode head) {
    if (head == null)
        return head;
    ListNode cur = head, next = head.next, temp = null;
    cur.next = null;
    while (next != null) {
        temp = next.next;
        next.next = cur;
        cur = next;
        next = temp;
```

```
    }
    return cur;
}
public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null)
        return head;
    ListNode next = reverseList(head.next);
    head.next.next = head;
    head.next = null;
    return next;
}
```

思路 1：逐一反向指针。O(n)

思路 2：也是逐一反向指针，只不过是从链表尾部开始。O(n)

207. Course Schedule Medium

There are a total of *n* courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

For example:

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

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

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

There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Note:

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about how a graph is represented.
2. You may assume that there are no duplicate edges in the input prerequisites.

Hints:

1. This problem is equivalent to finding if a cycle exists in a directed graph. If a cycle exists, no topological ordering exists and therefore it will be impossible to take all courses.
2. Topological Sort via DFS - A great video tutorial (21 minutes) on Coursera explaining the basic concepts of Topological Sort.
3. Topological sort could also be done via BFS.

```
public boolean canFinish(int numCourses, int[][] prerequisites) {
    int[] preCount = new int[numCourses];
    int finishable = 0;
    Map<Integer, List<Integer>> m = new HashMap<>();
    for (int[] p : prerequisites) {
        List<Integer> c = m.getOrDefault(p[1], new ArrayList<>());
        c.add(p[0]);
        m.put(p[1], c);
        ++preCount[p[0]];
    }
    Queue<Integer> q = new LinkedList<>();
    for (int i = 0; i < numCourses; ++i)
        if (preCount[i] == 0)
            q.offer(i);
    while (!q.isEmpty()) {
        int cur = q.poll();
        ++finishable;
        for (int n : m.getOrDefault(cur, new ArrayList<>())) {
            --preCount[n];
```

```
            if (preCount[n] == 0)
                q.offer(n);
        }
    }
    return finishable == numCourses;
}
```

思路：用一个 Map 记录课程的后续课程，用一个数组记录每门课前提课程的门数。把前提门数为零的放入一个队列，然后逐个元素看能达到的所有课程，如果达到某课，则该课的前提减 1，如果该课前提为零，则加入队列。重复直到队列为空，如果队列中累计弹出的课程数=总课程数，则表示可以完成。

208. Implement Trie (Prefix Tree) Medium

Implement a trie with insert, search, and startsWith methods.

Note:

You may assume that all inputs are consist of lowercase letters a-z.

```
class TrieNode {
    boolean isWord;
    TrieNode[] children = new TrieNode[26];
}

class Trie {
    TrieNode root;

    /** Initialize your data structure here. */
    public Trie() {
        root = new TrieNode();
    }

    /** Inserts a word into the trie. */
    public void insert(String word) {
        TrieNode cur = root;
        for (char c : word.toCharArray()) {
            if (cur.children[c - 'a'] == null)
                cur.children[c - 'a'] = new TrieNode();
            cur = cur.children[c - 'a'];
        }
        cur.isWord = true;
    }

    /** Returns if the word is in the trie. */
    public boolean search(String word) {
        TrieNode cur = searchHelper(root, word);
        return cur != null && cur.isWord;
    }

    /**
     * Returns if there is any word in the trie that starts with the given prefix.
     */
    public boolean startsWith(String prefix) {
        TrieNode cur = searchHelper(root, prefix);
        return cur != null;
    }

    private TrieNode searchHelper(TrieNode root, String word) {
        TrieNode cur = root;
        for (char c : word.toCharArray()) {
            if (cur == null)
                break;
            cur = cur.children[c - 'a'];
        }
        return cur;
    }
```

}
思路：使用自字义 Class 存储 Trie 树，每个结点包含两个信息：是否是词以及其子结点的可能（26 个字母）。添加新词就建立对应的 Trie 树或把该词插入已有树，并标记结尾为词。Search 只要看有无路径及对应路径 s 结尾处是不是词，StartWith 只是看路径是否存在。

209. Minimum Size Subarray Sum Medium

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum >= **s**. If there isn't one, return 0 instead.
For example, given the array [2,3,1,2,4,3] and s = 7,
the subarray [4,3] has the minimal length under the problem constraint.
More practice:
If you have figured out the *O*(*n*) solution, try coding another solution of which the time complexity is *O*(*n* log *n*).

```
public int minSubArrayLen(int s, int[] nums) {
    int ans = Integer.MAX_VALUE, i = 0, j = 0, sum = 0;
    while (j < nums.length) {
        sum += nums[j++];
        while ((sum >= s)) {
            ans = Math.min(ans, j - i);
            sum -= nums[i++];
        }
    }
    return ans == Integer.MAX_VALUE ? 0 : ans;
}
```

思路：滑动窗口。因为需要连续且大于指定数，所以从头开始累加，大于则成为一个极值，再从头减去，小于后再接着往后累加。最后取极值中的最佳。O(n)。
O(nlogn)方法，建立一个累和数组 sums，每个元素都是到当前位置为止前面元素的和，所以是一个递增数列。要找某区别内和>=s 的 i、j 的值，就是找 sums[j]-sum[i]>=s 的所有 j-i 中最小的也就是在 i 位置，找 sums[j]>=s+sum[i]的最小 j 值，可以用二分查找实现。

210. Course Schedule II Medium

There are a total of *n* courses you have to take, labeled from 0 to n - 1.
Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]
Given the total number of courses and a list of prerequisite **pairs**, return the ordering of courses you should take to finish all courses.
There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.
For example:

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

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1]

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

There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].
Note:

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about how a graph is represented.
2. You may assume that there are no duplicate edges in the input prerequisites.

Hints:

1. This problem is equivalent to finding the topological order in a directed graph. If a cycle exists, no topological ordering exists and therefore it will be impossible to take all courses.
2. Topological Sort via DFS - A great video tutorial (21 minutes) on Coursera explaining the basic concepts of Topological Sort.
3. Topological sort could also be done via BFS.

```
public int[] findOrder(int numCourses, int[][] prerequisites) {
    List<List<Integer>> adj = new ArrayList<List<Integer>>(numCourses);
    for (int i = 0; i < numCourses; i++)
        adj.add(i, new ArrayList<>());
    for (int i = 0; i < prerequisites.length; i++)
        adj.get(prerequisites[i][1]).add(prerequisites[i][0]);
    int[] visited = new int[numCourses];
    Stack<Integer> stack = new Stack<>();
    for (int i = 0; i < numCourses; i++)
        if (!topologicalSort(adj, i, stack, visited))
            return new int[0];
    int i = 0;
    int[] result = new int[numCourses];
    while (!stack.isEmpty()) {
        result[i++] = stack.pop();
    }
    return result;
}

private boolean topologicalSort(List<List<Integer>> adj, int v, Stack<Integer> stack, int[]
visited) {
    if (visited[v] == 2)
        return true;
    if (visited[v] == 1)
        return false;
    visited[v] = 1;
    for (Integer u : adj.get(v)) {
        if (!topologicalSort(adj, u, stack, visited))
            return false;
    }
    visited[v] = 2;
    stack.push(v);
    return true;
}
```

思路：拓扑排序。建立图，从根结点开始，拓扑排序看有没有循环依赖（需要两次访问已访问结点），如果有，则无法完成；如果没有，给出一种排序即可。过程中有 3 个状态：0 是未访问，1 是已访问但是未结束（可能死循环），2 是已访问且结束。

211. Add and Search Word - Data structure design Medium

Design a data structure that supports the following two operations:

```
void addWord(word)
bool search(word)
```

search(word) can search a literal word or a regular expression string containing only letters `a-z` or `.`. A `.` means it can represent any one letter.

For example:

```
addWord("bad")
addWord("dad")
addWord("mad")
search("pad") -> false
search("bad") -> true
search(".ad") -> true
```

```
search("b..") -> true
```

Note: You may assume that all words are consist of lowercase letters a-z.
Hint: You should be familiar with how a Trie works. If not, please work on this problem: Implement Trie (Prefix Tree) first.

```
class TrieNode {
    boolean isWord;
    TrieNode[] children;

    public TrieNode() {
        this.isWord = false;
        children = new TrieNode[26];
    }
}

class WordDictionary {

    TrieNode root;

    public WordDictionary() {
        root = new TrieNode();
    }

    public void addWord(String word) {
        TrieNode cur = root;
        for (char c : word.toCharArray()) {
            if (cur.children[c - 'a'] == null)
                cur.children[c - 'a'] = new TrieNode();
            cur = cur.children[c - 'a'];
        }
        cur.isWord = true;
    }

    public boolean search(String word) {
        return searchHelper(word.toCharArray(), 0, root);
    }

    boolean searchHelper(char[] cs, int i, TrieNode root) {
        if (i == cs.length) {
            return root.isWord;
        }
        if (cs[i] == '.') {
            for (TrieNode child : root.children)
                if (child != null)
                    if (searchHelper(cs, i + 1, child))
                        return true;
        } else
            return root.children[cs[i] - 'a'] != null && searchHelper(cs, i + 1,
root.children[cs[i] - 'a']);
        return false;
    }
}
```

思路：使用Trie树存字典便于查找。Trie对建立参见第208题。

212. Word Search II **Hard**

Given a 2D board and a list of words from the dictionary, find all words in the board.
Each word must 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 in a word.
For example,
Given **words** = ["oath","pea","eat","rain"] and **board** =

```
[ ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']]
```

Return ["eat","oath"].

Note:

You may assume that all inputs are consist of lowercase letters a-z.

You would need to optimize your backtracking to pass the larger test. Could you stop backtracking earlier?

If the current candidate does not exist in all words' prefix, you could stop backtracking immediately. What kind of data structure could answer such query efficiently? Does a hash table work? Why or why not? How about a Trie? If you would like to learn how to implement a basic trie, please work on this problem: Implement Trie (Prefix Tree) first.

```
class Solution {
    public List<String> findWords(char[][] board, String[] words) {
        List<String> res = new ArrayList<>();
        TrieNode root = buildTrie(words);
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                dfs(board, i, j, root, res);
            }
        }
        return res;
    }

    public void dfs(char[][] board, int i, int j, TrieNode p, List<String> res) {
        char c = board[i][j];
        if (c == '#' || p.next[c - 'a'] == null)
            return;
        p = p.next[c - 'a'];
        if (p.word != null) {
            res.add(p.word);
            p.word = null;
        }
        board[i][j] = '#';
        if (i > 0)
            dfs(board, i - 1, j, p, res);
        if (j > 0)
            dfs(board, i, j - 1, p, res);
        if (i < board.length - 1)
            dfs(board, i + 1, j, p, res);
        if (j < board[0].length - 1)
            dfs(board, i, j + 1, p, res);
        board[i][j] = c;
    }

    public TrieNode buildTrie(String[] words) {
        TrieNode root = new TrieNode();
        for (String w : words) {
            TrieNode p = root;
            for (char c : w.toCharArray()) {
                int i = c - 'a';
                if (p.next[i] == null)
                    p.next[i] = new TrieNode();
                p = p.next[i];
            }
            p.word = w;
        }
        return root;
    }
```

```
    class TrieNode {
        TrieNode[] next = new TrieNode[26];
        String word;
    }
}
```

思路：先用 Trie 树存然后从每个点出发查找有无在树中的词，如有则添加入结果并标记该词为空（去重）。深度优先遍历，四个方向都有，走过的点标记为#。结束条件：到达边界或碰到#。

213. House Robber II Medium

Note: This is an extension of House Robber.

After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

```
public int rob(int[] nums) {
    int n = nums.length;
    if (n == 1)
        return nums[0];
    return Math.max(rob(nums, 0, n - 1), rob(nums, 1, n));
}

int rob(int[] nums, int j, int k) {
    int max = 0, preMax = 0, noRob, rob;
    for (int i = j; i < k; ++i) {
        int cur = nums[i];
        rob = preMax + cur;
        noRob = max;
        max = Math.max(rob, noRob);
        preMax = noRob;
    }
    return max;
}
```

思路：与第 198 题思路一致，只不过要从 0 到 n-1，1 到 n 中取较大的。因为 n 和 0 相邻。

214. Shortest Palindrome Hard

Given a string S, you are allowed to convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

For example:

Given "aacecaaa", return "aaacecaaa".

Given "abcd", return "dcbabcd".

```
public String shortestPalindrome(String s) {
    int j = 0;
    for (int i = s.length() - 1; i >= 0; i--)
        if (s.charAt(i) == s.charAt(j))
            ++j;
    if (j == s.length())
        return s;
    String suffix = s.substring(j);
    return new StringBuilder(suffix).reverse().toString() + shortestPalindrome(s.substring(0, j))
+ suffix;
}
```

思路：双指针法。通过不断移动右指针（从最右侧开始到最左侧），如果遇到与左指针相同的则左指针右移。这个过程中可以保证：如果有开头部分有互文结构，那么左指针一定指向互文结构后一个位置或者更远位置。这里把左指针右面的字符串翻转拼接到左面，则外围一定是互文结构，原字串[0,j]部分再进行同样的操作（递归）当本身就是互文结构，则 j 达到终点，这时递归结束。

215. Kth Largest Element in an Array Medium

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example,

Given [3,2,1,5,6,4] and k = 2, return 5.

Note:

You may assume k is always valid, 1 ≤ k ≤ array's length.

```
public int findKthLargest(int[] nums, int k) {
    final int N = nums.length;
    Arrays.sort(nums);
    return nums[N - k];
}
```

思路 **1**：排序后取第 k 大。O(nlogn)

```
public int findKthLargest(int[] nums, int k) {
    final Queue<Integer> pq = new PriorityQueue<>();
    for (int val : nums) {
        pq.offer(val);
        if (pq.size() > k)
            pq.poll();
    }
    return pq.peek();
}
```

思路 **2**：维护一个前 k 大的队列，最后弹出第一个元素。O(nlogk)

思路 3（略）：类似快速排序，先找出前 k 个数，然后取最大。$O(n^2)$，平均 O(n)

216. Combination Sum III Medium

Find all possible combinations of ***k*** numbers that add up to a number ***n***, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Example 1:

Input: ***k*** = 3, ***n*** = 7

Output:

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

Example 2:

Input: ***k*** = 3, ***n*** = 9

Output:

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

```
public List<List<Integer>> combinationSum3(int k, int n) {
    List<List<Integer>> ans = new ArrayList<>();
    combination(ans, new ArrayList<Integer>(), k, 1, n);
    return ans;
}

private void combination(List<List<Integer>> ans, List<Integer> comb, int k, int start, int n) {
    if (comb.size() == k && n == 0) {
        List<Integer> li = new ArrayList<Integer>(comb);
        ans.add(li);
        return;
    }
    for (int i = start; i <= 9; i++) {
        comb.add(i);
        combination(ans, comb, k, i + 1, n - i);
        comb.remove(comb.size() - 1);
    }
}
```

思路：回溯法。逐一查看所有可能，符合条件的加入结果。

217. Contains Duplicate Easy

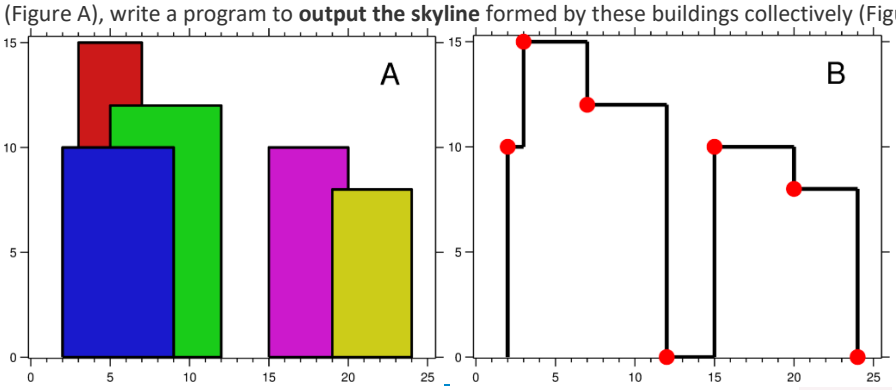
Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

```
public boolean containsDuplicate(int[] nums) {
    Set<Integer> numSet = new HashSet<Integer>();
    for (int num : nums)
        if (!numSet.add(num))
            return true;
    return false;
}
```

思路：建立词典查找。

218. The Skyline Problem Hard

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Now suppose you are **given the locations and height of all the buildings** as shown on a cityscape photo (Figure A), write a program to **output the skyline** formed by these buildings collectively (Figure B).



The geometric information of each building is represented by a triplet of integers [Li, Ri, Hi], where Li and Ri are the x coordinates of the left and right edge of the ith building, respectively, and Hi is its height. It is guaranteed that 0 ≤ Li, Ri ≤ INT_MAX, 0 < Hi ≤ INT_MAX, and Ri - Li > 0. You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

For instance, the dimensions of all buildings in Figure A are recorded as: [[2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8]].

The output is a list of "**key points**" (red dots in Figure B) in the format of [[x1,y1], [x2, y2], [x3, y3], ...] that uniquely defines a skyline. **A key point is the left endpoint of a horizontal line segment**. Note that the last key point, where the rightmost building ends, is merely used to mark the termination of the skyline, and always has zero height. Also, the ground in between any two adjacent buildings should be considered part of the skyline contour.

For instance, the skyline in Figure B should be represented as:[[2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0]].

Notes:

- The number of buildings in any input list is guaranteed to be in the range [0, 10000].
- The input list is already sorted in ascending order by the left x position Li.
- The output list must be sorted by the x position.
- There must be no consecutive horizontal lines of equal height in the output skyline. For instance, [...[2 3], [4 5], [7 5], [11 5], [12 7]...] is not acceptable; the three lines of height 5 should be merged into one in the final output as such: [...[2 3], [4 5], [12 7], ...]

```
public List<int[]> getSkyline(int[][] buildings) {
    List<int[]> heights = new ArrayList<>();
    for (int[] b : buildings) {
        heights.add(new int[] { b[0], -b[2] });
```

```
            heights.add(new int[] { b[1], b[2] });
        }
        Collections.sort(heights, (a, b) -> (a[0] == b[0]) ? a[1] - b[1] : a[0] - b[0]);
        TreeMap<Integer, Integer> heightMap = new TreeMap<>(Collections.reverseOrder());
        heightMap.put(0, 1);
        int prevHeight = 0;
        List<int[]> skyLine = new LinkedList<>();
        for (int[] h : heights) {
            if (h[1] < 0) {
                heightMap.put(-h[1], heightMap.getOrDefault(-h[1], 0) + 1);
            } else {
                Integer cnt = heightMap.get(h[1]);
                if (cnt == 1) {
                    heightMap.remove(h[1]);
                } else {
                    heightMap.put(h[1], cnt - 1);
                }
            }
            int currHeight = heightMap.firstKey();
            if (prevHeight != currHeight) {
                skyLine.add(new int[] { h[0], currHeight });
                prevHeight = currHeight;
            }
        }
        return skyLine;
    }
```

思路：从左到右追踪顶点，上升则记录，下降则看是否有高于它的线存在，不存在时记录，形成 List 返回。实现方法：用一个数组保存所有顶点（从左到右），其中左顶点对应的高记录为负值以区分。遍历数组，判断是否为起点，起点记入 TreeMap（逆向排序，保证到目前位置为止，比较高的在树的最前），终点从 Map 中减去，取 Map 中最大值，若与前 height 不同，则新有效点出来（高或低）。

219. Contains Duplicate II Easy

Given an array of integers and an integer *k*, find out whether there are two distinct indices *i* and *j* in the array such that **nums[i] = nums[j]** and the **absolute** difference between *i* and *j* is at most *k*.

```
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        Set<Integer> set = new HashSet<Integer>();
        for (int i = 0; i < nums.length; i++) {
            if (i > k)
                set.remove(nums[i - k - 1]);
            if (!set.add(nums[i]))
                return true;
        }
        return false;
    }
```

思路：滑动窗口，一直在 k 范围内寻找有无相同的值。

220. Contains Duplicate III Medium

Given an array of integers, find out whether there are two distinct indices *i* and *j* in the array such that the **absolute** difference between **nums[i]** and **nums[j]** is at most *t* and the **absolute** difference between *i* and *j* is at most *k*.

```
    private long getID(long x, long w) {
        return x < 0 ? (x + 1) / w - 1 : x / w;
    }

    public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {
        if (t < 0)
            return false;
        Map<Long, Long> d = new HashMap<>();
```

```
    long w = (long) t + 1;
    for (int i = 0; i < nums.length; ++i) {
        long m = getID(nums[i], w);
        if (d.containsKey(m))
            return true;
        if (d.containsKey(m - 1) && Math.abs(nums[i] - d.get(m - 1)) < w)
            return true;
        if (d.containsKey(m + 1) && Math.abs(nums[i] - d.get(m + 1)) < w)
            return true;
        d.put(m, (long) nums[i]);
        if (i >= k)
            d.remove(getID(nums[i - k], w));
    }
    return false;
}
```

思路：滑动窗口+桶排序。桶的大小为 t+1（使用 Long 避免溢出），使用一个 Map 保存 k 范围内的数字（滑动窗口）。当分子绝对值小于分母时，无论正负都会落入索引为 0 的桶，为区分，把小于 0 的数全部左移一位，即（(x+1)/w – 1）。每次加入新数时判断是否同索引桶里有数字，如果有，那么这对数一定是一个解。再判断临桶有没有数且距离在 k 以内并且数值相差小于 t。

221. Maximal Square Medium

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.
For example, given the following matrix:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Return 4.

```
public int maximalSquare(char[][] matrix) {
    if (matrix.length == 0)
        return 0;
    int m = matrix.length, n = matrix[0].length, result = 0;
    int[][] b = new int[m + 1][n + 1];
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            if (matrix[i - 1][j - 1] == '1') {
                b[i][j] = Math.min(Math.min(b[i][j - 1], b[i - 1][j - 1]), b[i - 1][j]) + 1;
                result = Math.max(b[i][j], result); // update result
            }
    return result * result;
}
```

思路：动态规划。要求出最大的正方形面积，只须求出最大正方形的边长。累加方法：如果当前值是 1，那么当前点所在 Square 的边长=左、上及左上中最小的值+1。

222. Count Complete Tree Nodes Medium

Given a **complete** binary tree, count the number of nodes.
Definition of a complete binary tree from Wikipedia:
In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h.

```
int height(TreeNode root) {
    return root == null ? -1 : 1 + height(root.left);
}

public int countNodes(TreeNode root) {
    int nodes = 0, h = height(root);
```

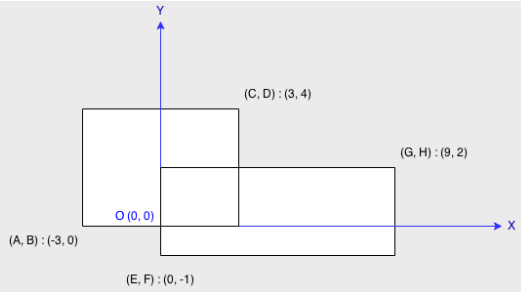
```
    while (root != null) {
        if (height(root.right) == h - 1) {
            nodes += 1 << h;
            root = root.right;
        } else {
            nodes += 1 << h - 1;
            root = root.left;
        }
        h--;
    }
    return nodes;
}
```

思路：根据完全二叉树特点：最多右树缺少叶子结点（高度最多差 1）。本题中总是根据左枝来进行高度计算，所以：

1. 如果右子树的高度与原树高度差 1，那么左子树必为满 2 叉树，且高为 h-1，故左子树叶子数为 2^h。继续统计右子树结点即可，此时整数的高度-1（因为已经进入第二层）。
2. 如果右子树高度与原树差不为 1（应该是 2），那么右子树必为满二叉树，且其高为 h-2，所以右子树叶子数为 2^{h-2}。继续统计左子树结点数即可，同样，整个树的高度-1。

223. Rectangle Area Medium

Find the total area covered by two **rectilinear** rectangles in a **2D** plane.
Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



Assume that the total area is never beyond the maximum possible value of **int**.

```
public int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {
    int left = Math.max(A, E), right = Math.max(Math.min(C, G), left);
    int bottom = Math.max(B, F), top = Math.max(Math.min(D, H), bottom);
    return (C - A) * (D - B) - (right - left) * (top - bottom) + (G - E) * (H - F);
}
```

224. Basic Calculator Hard

Implement a basic calculator to evaluate a simple expression string.
The expression string may contain open (and closing parentheses), the plus + or minus sign -, **non-negative** integers and empty spaces .
You may assume that the given expression is always valid.
Some examples:

```
"1 + 1" = 2
" 2-1 + 2 " = 3
"(1+(4+5+2)-3)+(6+8)" = 23
```

Note: Do not use the eval built-in library function.

```
public int calculate(String s) {
    char[] cs = s.toCharArray();
    int result = 0, sign = 1;
    Stack<Integer> stack = new Stack<>();
```

```
    for (int i = 0; i < cs.length; ++i) {
        char c = cs[i];
        switch (c) {
        case '+':
            sign = 1;
            break;
        case '-':
            sign = -1;
            break;
        case '(':
            stack.push(result);
            stack.push(sign);
            sign = 1;
            result = 0;
            break;
        case ')':
            result = stack.pop() * result + stack.pop();
            break;
        default:
            int cur = 0;
            if (Character.isDigit(c)) {
                cur = c - '0';
                while (i + 1 < cs.length && Character.isDigit(cs[i + 1]))
                    cur = cur * 10 + (cs[++i] - '0');
            }
            result += cur * sign;
            break;
        }
    }
    return result;
}
```

思路：只有加减法，所以括号里面部分可以当加数处理和符号相乘。碰到括号时压栈（包括符号和当前数值，并重置 Result 和符号），反括号出栈，第一个是符号，第二个是压栈前的结果，符号乘当时 Result+压栈前结果就是当前结果。

225. Implement Stack using Queues Easy

Implement the following operations of a stack using queues.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

Notes:

- You must use *only* standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
- You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

```
public class MyStack {
    Queue<Integer> queue;

    public MyStack() {
        this.queue = new LinkedList<Integer>();
    }

    public void push(int x) {
        queue.add(x);
```

```
        for (int i = 0; i < queue.size() - 1; i++) {
            queue.add(queue.poll());
        }
    }

    public int pop() {
        return queue.poll();
    }

    public int top() {
        return queue.peek();
    }

    public boolean empty() {
        return queue.isEmpty();
    }
}
```

思路：为了先进后出，每次添加新元素后都反转前面所有元素，即逐个重新入栈。

226. Invert Binary Tree Easy

Invert a binary tree.

```
  4
 / \
 2  7
/\ /\
1 36 9
```

to

```
  4
 / \
 7  2
/\ /\
9 63 1
```

Trivia:

This problem was inspired by this original tweet by Max Howell:

Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so fuck off.

```
public TreeNode invertTree(TreeNode root) {
    if (root == null)
        return null;
    TreeNode right = invertTree(root.right);
    TreeNode left = invertTree(root.left);
    root.left = right;
    root.right = left;
    return root;
}
```

思路，递归求解，左右互换。

227. Basic Calculator II Medium

Implement a basic calculator to evaluate a simple expression string.

The expression string contains only **non-negative** integers, `+`, `-`, `*`, `/` operators and empty spaces . The integer division should truncate toward zero.

You may assume that the given expression is always valid.

Some examples:

```
"3+2*2" = 7
" 3/2 " = 1
" 3+5 / 2 " = 5
```

Note: Do not use the eval built-in library function.

```
public int calculate(String s) {
    int len;
    if (s == null || (len = s.length()) == 0)
        return 0;
    Stack<Integer> stack = new Stack<Integer>();
    int num = 0;
    char sign = '+';
    for (int i = 0; i < len; i++) {
        if (Character.isDigit(s.charAt(i))) {
            num = num * 10 + s.charAt(i) - '0';
        }
        if ((!Character.isDigit(s.charAt(i)) && ' ' != s.charAt(i)) || i == len - 1) {
            if (sign == '-') {
                stack.push(-num);
            }
            if (sign == '+') {
                stack.push(num);
            }
            if (sign == '*') {
                stack.push(stack.pop() * num);
            }
            if (sign == '/') {
                stack.push(stack.pop() / num);
            }
            sign = s.charAt(i);
            num = 0;
        }
    }
    int re = 0;
    for (int i : stack) {
        re += i;
    }
    return re;
}
```

思路：如果是数字，则视为当前数字缓存在 num 中并不断进位，如果是空格则略过。加减的话把对应符号的 num 压栈。因为乘除优先级高，所以在压栈时同时进行与上一结果进行相应运算再压栈（当前结果），注意符号出现时先不进行运算，等到空格或下一符号出现时才运算，因为符号后置。还要注意最后弹出栈中所有元素并求和。

228. Summary Ranges Medium

Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

```
public List<String> summaryRanges(int[] nums) {
    List<String> ans = new ArrayList<String>();
    int first, last;
    for (int i = 0; i < nums.length; ++i) {
        first = last = nums[i];
        while (i < nums.length - 1 && last + 1 == nums[i + 1]) {
            last = nums[++i];
        }
        if (last == first)
            ans.add(String.valueOf(last));
        else
            ans.add(first + "->" + last);
```

```
    }
    return ans;
}
```
思路：逐一找出相差为 1 的数字子数组（使用 first, last 分别记录起始和结束值），如果 first==last 则说明为单数字，这时单独成行，否则为 first + "->" + last。

229. Majority Element II Medium

Given an integer array of size *n*, find all elements that appear more than ⌊ n/3 ⌋ times. The algorithm should run in linear time and in O(1) space.

```
public List<Integer> majorityElement(int[] nums) {
    if (nums == null || nums.length == 0)
        return new ArrayList<Integer>();
    List<Integer> result = new ArrayList<Integer>();
    int number1 = nums[0], number2 = nums[0], count1 = 0, count2 = 0, len = nums.length;
    for (int i = 0; i < len; i++)
        if (nums[i] == number1)
            count1++;
        else if (nums[i] == number2)
            count2++;
        else if (count1 == 0) {
            number1 = nums[i];
            count1 = 1;
        } else if (count2 == 0) {
            number2 = nums[i];
            count2 = 1;
        } else {
            count1--;
            count2--;
        }
    count1 = 0;
    count2 = 0;
    for (int i = 0; i < len; i++)
        if (nums[i] == number1)
            count1++;
        else if (nums[i] == number2)
            count2++;
    if (count1 > len / 3)
        result.add(number1);
    if (count2 > len / 3)
        result.add(number2);
    return result;
}
```
思路：找出现最多的的两个数，再各自统计其数目，看其是否符合条件，因为至少占 1/3 强，所以最多有两个数。

230. Kth Smallest Element in a BST Medium

Given a binary search tree, write a function kthSmallest to find the **k**th smallest element in it.

Note:

You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Follow up:

What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

```
public int kthSmallest(TreeNode root, int k) {
    int count = countNodes(root.left);
    if (k <= count)
        return kthSmallest(root.left, k);
    else if (k > count + 1)
        return kthSmallest(root.right, k - 1 - count);
```

```
    return root.val;
}

public int countNodes(TreeNode n) {
    if (n == null)
        return 0;
    return 1 + countNodes(n.left) + countNodes(n.right);
}
```
思路：递归求解，利用二叉搜索树的特点（左子树结点都小于根结点），每次根据 note 数量进入不同分枝。

231. Power of Two Easy

Given an integer, write a function to determine if it is a power of two.

```
public boolean isPowerOfTwo(int n) {
    if (n <= 0)
        return false;
    return (n & (n - 1)) == 0;
}
public boolean isPowerOfTwo(int n) {
    return n > 0 && Integer.bitCount(n) == 1;//排除负号
}
```
思路：2 的幂共同特点：二进制只有一位有值。

232. Implement Queue using Stacks Easy

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

Notes:

- You must use *only* standard operations of a stack -- which means only push to top, peek/pop from top, size, and is emptyoperations are valid.
- Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

```
public class MyQueue {
    private int front;
    private Stack<Integer> s1 = new Stack<>();
    private Stack<Integer> s2 = new Stack<>();

    public MyQueue() {
    }

    public void push(int x) {
        if (s1.empty())
            front = x;
        s1.push(x);
    }

    public int pop() {
```

```
        if (s2.isEmpty())
            while (!s1.isEmpty())
                s2.push(s1.pop());
        return s2.pop();
    }

    public int peek() {
        if (!s2.isEmpty())
            return s2.peek();
        return front;
    }

    public boolean empty() {
        return s1.isEmpty() && s2.isEmpty();
    }
}
```

思路：栈是先入后出，所以只要借助另一个栈就能达到先进先出。

233. Number of Digit One Hard

Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.
For example:
Given n = 13,
Return 6, because digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

```
public int countDigitOne(int n) {
    int countr = 0;
    for (long i = 1; i <= n; i *= 10) {
        long divider = i * 10;
        countr += (n / divider) * i + Math.min(Math.max(n % divider - i + 1, 0L), i);
    }
    return countr;
}
```

思路：暴力算法速度慢。可以利用数学特点，计算结果。这类题基本思路是先举一些例子，找规律，找到规律后实现。

234. Palindrome Linked List Easy

Given a singly linked list, determine if it is a palindrome.
Follow up:
Could you do it in O(n) time and O(1) space?

```
public boolean isPalindrome(ListNode head) {
    ListNode fast = head, slow = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    if (fast != null) // odd nodes: let right half smaller
        slow = slow.next;
    slow = reverse(slow);
    fast = head;
    while (slow != null) {
        if (fast.val != slow.val)
            return false;
        fast = fast.next;
        slow = slow.next;
    }
```

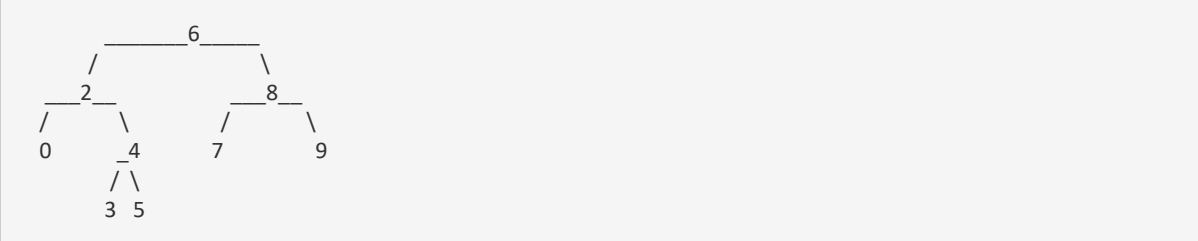
```
    return true;
}

public ListNode reverse(ListNode head) {
    ListNode prev = null;
    while (head != null) {
        ListNode next = head.next;
        head.next = prev;
        prev = head;
        head = next;
    }
    return prev;
}
```

思路：把链表平分成两半，一半反转，看两链表是否相同（尾元素可差 1）。

235. Lowest Common Ancestor of a Binary Search Tree Easy

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow **a node to be a descendant of itself**).”



For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

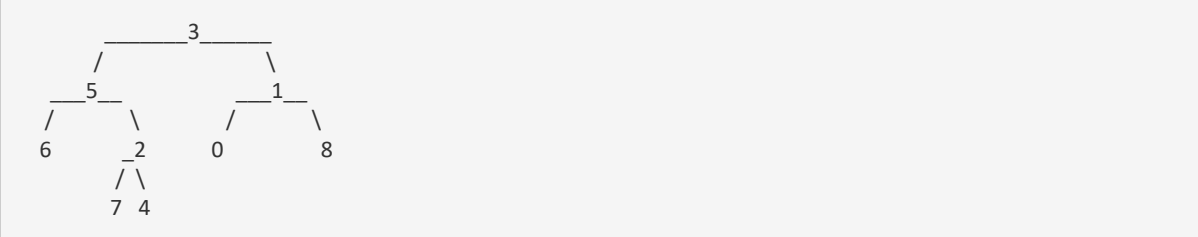
```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    while ((root.val - p.val) * (root.val - q.val) > 0)
        root = p.val < root.val ? root.left : root.right;
    return root;
}
```

思路：根据二叉树特点，共同祖先必然大于其一，小于另一。不断根据与其中 1 结点的大小关系，从根部一直到找到符合条件的共同祖先。

236. Lowest Common Ancestor of a Binary Tree Medium

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow **a node to be a descendant of itself**).”



For example, the lowest common ancestor (LCA) of nodes 5 and 1 is 3. Another example is LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
```

```
    if (root == null || root == p || root == q)
        return root;
    TreeNode left = lowestCommonAncestor(root.left, p, q);
    TreeNode right = lowestCommonAncestor(root.right, p, q);
    return left == null ? right : right == null ? left : root;
}
```

思路：深度优先查找，找到其中之一返回，如果一侧为空，则另一侧为解（单结点祖先），都不为空则当前结点为最低共同祖先。

237. Delete Node in a Linked List Easy

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.
Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

```
public void deleteNode(ListNode node) {
    node.val = node.next.val;
    node.next = node.next.next;
}
```

思路：拷贝后面元素值到当前元素，删除后面元素。

238. Product of Array Except Self Medium

Given an array of *n* integers where *n* > 1, nums, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].
Solve it **without division** and in O(*n*).
For example, given [1,2,3,4], return [24,12,8,6].
Follow up:
Could you solve it with constant space complexity? (Note: The output array **does not** count as extra space for the purpose of space complexity analysis.)

```
public int[] productExceptSelf(int[] nums) {
    int n = nums.length;
    int[] res = new int[n];
    res[0] = 1;
    for (int i = 1; i < n; i++)
        res[i] = res[i - 1] * nums[i - 1];
    int right = 1;
    for (int i = n - 1; i >= 0; i--) {
        res[i] *= right;
        right *= nums[i];
    }
    return res;
}
```

思路：注意观察，每位结果是左面所有位置数的积*右面所有数的积，所以左右各扫一遍记录相应的积并相乘。注意问一下结果溢出的情况。

239. Sliding Window Maximum Hard

Given an array *nums*, there is a sliding window of size *k* which is moving from the very left of the array to the very right. You can only see the *k* numbers in the window. Each time the sliding window moves right by one position.
For example,
Given *nums* = [1,3,-1,-3,5,3,6,7], and *k* = 3.

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
```

```
 1 3 -1 -3 [5 3 6] 7    6
 1 3 -1 -3 5 [3 6 7]    7
```

Therefore, return the max sliding window as [3,3,5,5,6,7].

Note:

You may assume *k* is always valid, ie: 1 ≤ k ≤ input array's size for non-empty array.

Follow up:

Could you solve it in linear time?

```
public int[] maxSlidingWindow(int[] nums, int k) {
    int n = nums.length, ri = 0;
    if(n == 0)
        return new int[0];
    int[] ans = new int[n-k+1];
    Deque<Integer> q = new ArrayDeque<>();
    for(int i = 0;i < n; ++i) {
        if(!q.isEmpty() && q.peek() < i - k + 1)
            q.poll();
        while(!q.isEmpty() && nums[q.peekLast()] < nums[i])
            q.pollLast();
        q.offer(i);
        if(i >= k - 1)
            ans[ri++] = nums[q.peek()];
    }
    return ans;
}
```

思路：滑动窗口。注意每个窗口内只有最大值有用，所以可以从左到右扫，遇到更大的数就把之前小一些的数删除，出范围的也删除。这样最终窗口内最大值永远是第一个。

240. Search a 2D Matrix II Medium

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

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

For example,

Consider the following matrix:

```
[ [1,  4,  7, 11, 15],
  [2,  5,  8, 12, 19],
  [3,  6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]]
```

Given **target** = 5, return true.

Given **target** = 20, return false.

```
public boolean searchMatrix(int[][] matrix, int target) {
    if (matrix == null || matrix.length < 1 || matrix[0].length < 1) {
        return false;
    }
    int col = matrix[0].length - 1;
    int row = 0;
    while (col >= 0 && row <= matrix.length - 1) {
        if (target == matrix[row][col]) {
            return true;
        } else if (target < matrix[row][col]) {
            col--;
        } else if (target > matrix[row][col]) {
            row++;
        }
    }
```

```
    return false;
}
```

思路：注意不能使用二分查找，因为第一列的尾部数有可能大于第二列的头部数。从右上角开始找，如果小于当前值，则往左移一位（行内也可使用 2 分查找），这样如果在当前行，必能找到。如果不在当前行，在任意时刻目标值大于当前值了，下移一行。因为本行中右侧值已经大于当前值，可以证明：右侧所有列已经不存在解。

241. Different Ways to Add Parentheses Medium

Given a string of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. The valid operators are `+`, `-` and `*`.

Example 1

Input: `"2-1-1"`.

```
((2-1)-1) = 0
(2-(1-1)) = 2
```

Output: `[0, 2]`

Example 2

Input: `"2*3-4*5"`

```
(2*(3-(4*5))) = -34
((2*3)-(4*5)) = -14
((2*(3-4))*5) = -10
(2*((3-4)*5)) = -10
(((2*3)-4)*5) = 10
```

Output: `[-34, -14, -10, -10, 10]`

```
public List<Integer> diffWaysToCompute(String input) {
    List<Integer> ret = new LinkedList<Integer>();
    for (int i = 0; i < input.length(); i++) {
        if (input.charAt(i) == '-' || input.charAt(i) == '*' || input.charAt(i) == '+') {
            String part1 = input.substring(0, i);
            String part2 = input.substring(i + 1);
            List<Integer> part1Ret = diffWaysToCompute(part1);
            List<Integer> part2Ret = diffWaysToCompute(part2);
            for (Integer p1 : part1Ret) {
                for (Integer p2 : part2Ret) {
                    int c = 0;
                    switch (input.charAt(i)) {
                    case '+':
                        c = p1 + p2;
                        break;
                    case '-':
                        c = p1 - p2;
                        break;
                    case '*':
                        c = p1 * p2;
                        break;
                    }
                    ret.add(c);
                }
            }
        }
    }
    if (ret.size() == 0) {
        ret.add(Integer.valueOf(input));
```

```
    }
    return ret;
}
```
思路：括号本质上是把左、右侧运算的优先级提高。递归求解，每次分左右，然后再根据当前符号进行相应运算。每次完整完成是一个解。最后别忘记检查解数为 0 的情况（单数）。

242. Valid Anagram Easy

Given two strings *s* and *t*, write a function to determine if *t* is an anagram of *s*.
For example,
s = "anagram", *t* = "nagaram", return true.
s = "rat", *t* = "car", return false.
Note:
You may assume the string contains only lowercase alphabets.
Follow up:
What if the inputs contain unicode characters? How would you adapt your solution to such case?

```
public boolean isAnagram(String s, String t) {
    if (s.length() != t.length()) {
        return false;
    }
    int[] counter = new int[26];
    for (int i = 0; i < s.length(); i++) {
        counter[s.charAt(i) - 'a']++;
        counter[t.charAt(i) - 'a']--;
    }
    for (int count : counter) {
        if (count != 0) {
            return false;
        }
    }
    return true;
}
```
思路：只要统计两者的各个字符数是否相等。

243. Shortest Word Distance Easy

Given a list of words and two words *word1* and *word2*, return the shortest distance between these two words in the list.
For example,
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].
Given *word1* = “coding”, *word2* = “practice”, return 3.
Given *word1* = "makes", *word2* = "coding", return 1.
Note:
You may assume that *word1* **does not equal to** *word2*, and *word1* and *word2* are both in the list.

```
public int shortestDistance(String[] words, String word1, String word2) {
    int i1 = -1, i2 = -1;
    int minDistance = words.length;
    for (int i = 0; i < words.length; i++) {
        if (words[i].equals(word1)) {
            i1 = i;
        } else if (words[i].equals(word2)) {
            i2 = i;
        }
        if (i1 != -1 && i2 != -1) {
            minDistance = Math.min(minDistance, Math.abs(i1 - i2));
        }
    }
    return minDistance;
}
```

思路：不断找两词的距离，然后取其中的最小值。

244. Shortest Word Distance II Medium

This is a **follow up** of Shortest Word Distance. The only difference is now you are given the list of words and your method will be called *repeatedly* many times with different parameters. How would you optimize it?
Design a class which receives a list of words in the constructor, and implements a method that takes two words *word1* and *word2* and return the shortest distance between these two words in the list.
For example,
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].
Given *word1* = “coding”, *word2* = “practice”, return 3.
Given *word1* = "makes", *word2* = "coding", return 1.
Note:
You may assume that *word1* **does not equal to** *word2*, and *word1* and *word2* are both in the list.

```
public class WordDistance {
    private Map<String, List<Integer>> map;

    public WordDistance(String[] words) {
        map = new HashMap<String, List<Integer>>();
        for (int i = 0; i < words.length; i++) {
            String w = words[i];
            if (map.containsKey(w)) {
                map.get(w).add(i);
            } else {
                List<Integer> list = new ArrayList<Integer>();
                list.add(i);
                map.put(w, list);
            }
        }
    }

    public int shortest(String word1, String word2) {
        List<Integer> list1 = map.get(word1);
        List<Integer> list2 = map.get(word2);
        int ret = Integer.MAX_VALUE;
        for (int i = 0, j = 0; i < list1.size() && j < list2.size();) {
            int index1 = list1.get(i), index2 = list2.get(j);
            if (index1 < index2) {
                ret = Math.min(ret, index2 - index1);
                i++;
            } else {
                ret = Math.min(ret, index1 - index2);
                j++;
            }
        }
        return ret;
    }
}
```

思路：将词和对应的坐标位置（List）存入词典，查找时把所有位置拿出来寻找差最小的即可。

245. Shortest Word Distance III Medium

This is a **follow up** of Shortest Word Distance. The only difference is now *word1* could be the same as *word2*.
Given a list of words and two words *word1* and *word2*, return the shortest distance between these two words in the list.
word1 and *word2* may be the same and they represent two individual words in the list.
For example,
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given *word1* = “makes”, *word2* = “coding”, return 1.
Given *word1* = "makes", *word2* = "makes", return 3.
Note:
You may assume *word1* and *word2* are both in the list.

```
public int shortestWordDistance(String[] words, String word1, String word2) {
    long dist = Integer.MAX_VALUE, i1 = dist, i2 = -dist;
    for (int i = 0; i < words.length; i++) {
        if (words[i].equals(word1))
            i1 = i;
        if (words[i].equals(word2)) {
            if (word1.equals(word2))
                i1 = i2;
            i2 = i;
        }
        dist = Math.min(dist, Math.abs(i1 - i2));
    }
    return (int) dist;
}
```

思路：与 243 题思路相似，只不过第二词如果与第一词相等，则第一词的坐标移到另一词的位置。

246. Strobogrammatic Number Easy

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).
Write a function to determine if a number is strobogrammatic. The number is represented as a string.
For example, the numbers "69", "88", and "818" are all strobogrammatic.

```
public boolean isStrobogrammatic(String num) {
    for (int i = 0, j = num.length() - 1; i <= j; i++, j--)
        if (!"00 11 88 696".contains(num.charAt(i) + "" + num.charAt(j)))
            return false;
    return true;
}
```

思路：数字翻转后必须为自身，且必须以中间元素（或空）为轴对称。这样的元素有 0、1、8 和 69、96（注意后两个必须成对出现）。

247. Strobogrammatic Number II Medium

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).
Find all strobogrammatic numbers that are of length = n.
For example,
Given n = 2, return ["11","69","88","96"].

```
public List<String> findStrobogrammatic(int n) {
    List<String> ans = new ArrayList<>();
    return findStrobogrammatic(n, n);
}

public List<String> findStrobogrammatic(int m, int n) {
    if (m == 0)
        return Arrays.asList(new String[] { "" });
    if (m == 1)
        return Arrays.asList(new String[] { "0", "1", "8" });
    List<String> lst = findStrobogrammatic(m - 2, n);
    List<String> ans = new ArrayList<String>();
    for (String ic : lst) {
        if (m != n)
            ans.add("0" + ic + "0");
        ans.add("1" + ic + "1");
        ans.add("6" + ic + "9");
        ans.add("8" + ic + "8");
        ans.add("9" + ic + "6");
    }
```

```
    return ans;
}
```
思路：递归求解。递推：每级去除两侧对称元素对，最终只有 0 或 1 个元素时退栈，并把对称对加回。

248. Strobogrammatic Number III Hard

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).
Write a function to count the total strobogrammatic numbers that exist in the range of low <= num <= high.
For example,
Given low = "50", high = "100", return 3. Because 69, 88, and 96 are three strobogrammatic numbers.
Note:
Because the range might be a large number, the *low* and *high* numbers are represented as string.

```
private static final char[][] pairs = { { '0', '0' }, { '1', '1' }, { '6', '9' }, { '8', '8' },
{ '9', '6' } };

public int strobogrammaticInRange(String low, String high) {
    int[] count = { 0 };
    for (int len = low.length(); len <= high.length(); len++) {
        char[] c = new char[len];
        dfs(low, high, c, 0, len - 1, count);
    }
    return count[0];
}

public void dfs(String low, String high, char[] c, int left, int right, int[] count) {
    if (left > right) {
        String s = new String(c);
        if ((s.length() == low.length() && s.compareTo(low) < 0)
                || (s.length() == high.length() && s.compareTo(high) > 0)) {
            return;
        }
        count[0]++;
        return;
    }
    for (char[] p : pairs) {
        c[left] = p[0];
        c[right] = p[1];
        if (c.length != 1 && c[0] == '0') {// 以 0 开头的数字不合法
            continue;
        }
        if (left == right && p[0] != p[1]) {// 同一字符时，只能使用自身对称字符，即 6、9 要排除
            continue;
        }
        dfs(low, high, c, left + 1, right - 1, count);
    }
}
```
思路：逐长度列出所有可能，思路见上题。注意在与低和高位长度相等时要比较大小去掉不符合条件的。

249. Group Shifted Strings Medium

Given a string, we can "shift" each of its letter to its successive letter, for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:

```
"abc" -> "bcd" -> ... -> "xyz"
```

Given a list of strings which contains only lowercase alphabets, group all strings that belong to the same shifting sequence.
For example, given: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"],
A solution is:

```
[ ["abc","bcd","xyz"],
 ["az","ba"],
 ["acef"],
 ["a","z"]]
```

```
public List<List<String>> groupStrings(String[] strings) {
    HashMap<String, ArrayList<String>> map = new HashMap<String, ArrayList<String>>();
    for (String s : strings) {
        StringBuilder keySb = new StringBuilder();
        for (int i = 1; i < s.length(); i++)
            keySb.append(String.format("%2d", (26 + s.charAt(i) - s.charAt(i - 1)) % 26));
        String key = keySb.toString();
        if (!map.containsKey(key))
            map.put(key, new ArrayList<String>());
        map.get(key).add(s);
    }
    return new ArrayList<List<String>>(map.values());
}
```

思路：这种词有共同点是词内字母间间距一致，故用间隔拼接起来作为关键字，把相关词组合起来就可以了。

250. Count Univalue Subtrees Medium

Given a binary tree, count the number of uni-value subtrees.

A Uni-value subtree means all nodes of the subtree have the same value.

For example:

Given binary tree,

```
      5
     /\
    1  5
   /\   \
  5  5   5
```

return 4.

```
private int count = 0;

public int countUnivalSubtrees(TreeNode root) {
    countUnivalueSubTreesHelper(root);
    return count;
}

private boolean countUnivalueSubTreesHelper(TreeNode node) {
    if (node == null)
        return true;
    boolean left = countUnivalueSubTreesHelper(node.left);
    boolean right = countUnivalueSubTreesHelper(node.right);
    if (left && right) {
        if ((node.left != null && node.val != node.left.val) || (node.right != null && node.val !=
node.right.val))
            return false;
        count++;
        return true;
    }
    return false;
}
```

思路：递归求解。递推方程：左右子树分别是并且左右子树树结点值与当前结点值相等，则当前树也是，否则不是。

251. Flatten 2D Vector Medium

Implement an iterator to flatten a 2d vector.
For example,
Given 2d vector =

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

By calling *next* repeatedly until *hasNext* returns false, the order of elements returned by *next* should be: [1,2,3,4,5,6].
Follow up:
As an added challenge, try to code it using only iterators in C++ or iterators in Java.

```
public class Vector2D implements Iterator<Integer> {
    private Iterator<List<Integer>> i;
    private Iterator<Integer> j;

    public Vector2D(List<List<Integer>> vec2d) {
        i = vec2d.iterator();
    }

    @Override
    public Integer next() {
        hasNext();
        return j.next();
    }

    @Override
    public boolean hasNext() {
        while ((j == null || !j.hasNext()) && i.hasNext())
            j = i.next().iterator();
        return j != null && j.hasNext();
    }
}
```

思路：每次把最外面一层的 Iterator 拿出，用完后再换下一，直到全部用尽。

252. Meeting Rooms Easy

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] ($s_i < e_i$), determine if a person could attend all meetings.
For example,
Given [[0, 30],[5, 10],[15, 20]],
return false.

```
public boolean canAttendMeetings(Interval[] intervals) {
    if (intervals == null)
        return false;
    Arrays.sort(intervals, (a, b) -> (a.start - b.start));
    for (int i = 1; i < intervals.length; i++)
        if (intervals[i].start < intervals[i - 1].end)
            return false;
    return true;
}
```

思路：排序间隔后看前一个结束时间是否晚于下一个开始时间。

253. Meeting Rooms II Medium

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] ($s_i < e_i$), find the minimum number of conference rooms required.

For example,
Given [[0, 30],[5, 10],[15, 20]],
return 2.

```
public int minMeetingRooms(Interval[] intervals) {
    int[] starts = new int[intervals.length];
    int[] ends = new int[intervals.length];
    for (int i = 0; i < intervals.length; i++) {
        starts[i] = intervals[i].start;
        ends[i] = intervals[i].end;
    }
    Arrays.sort(starts);
    Arrays.sort(ends);
    int rooms = 0;
    int endsItr = 0;
    for (int i = 0; i < starts.length; i++) {
        if (starts[i] < ends[endsItr])
            rooms++;
        else
            endsItr++;
    }
    return rooms;
}
```

思路：如果需要更多会议室，则结束点大于另一个会议的起始点。排序起始和终止点，找出结束点大于起始点（除自身）的个数。O(nlogn)

254. Factor Combinations Medium

Numbers can be regarded as product of its factors. For example,

```
8 = 2 x 2 x 2;
  = 2 x 4.
```

Write a function that takes an integer *n* and return all possible combinations of its factors.

Note:

1. You may assume that *n* is always positive.
2. Factors should be greater than 1 and less than *n*.

Examples:

input: 1
output:

```
[]
```

input: 37
output:

```
[]
```

input: 12
output:

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

input: 32
output:

```
[ [2, 16],
  [2, 2, 8],
  [2, 2, 2, 4],
  [2, 2, 2, 2, 2],
  [2, 4, 4],
  [4, 8]]
```

```
public List<List<Integer>> getFactors(int n) {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    helper(result, new ArrayList<Integer>(), n, 2);
    return result;
}

public void helper(List<List<Integer>> result, List<Integer> item, int n, int start) {
    if (n <= 1) {
        if (item.size() > 1)
            result.add(new ArrayList<Integer>(item));
        return;
    }
    for (int i = start; i <= n; ++i) {
        if (n % i == 0) {
            item.add(i);
            helper(result, item, n / i, i);
            item.remove(item.size() - 1);
        }
    }
}
```

思路：回溯法。利用%运算判断是否能被某数除尽，找到所以可能。

255. Verify Preorder Sequence in Binary Search Tree Medium

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Follow up:

Could you do it using only constant space complexity?

```
public boolean verifyPreorder(int[] preorder) {
    int low = Integer.MIN_VALUE;
    Stack<Integer> path = new Stack();
    for (int p : preorder) {
        if (p < low)
            return false;
        while (!path.empty() && p > path.peek())
            low = path.pop();
        path.push(p);
    }
    return true;
}
```

O（1）空间：

```
public boolean verifyPreorder(int[] preorder) {
    int low = Integer.MIN_VALUE, i = -1;
    for (int p : preorder) {
        if (p < low)
            return false;
        while (i >= 0 && p > preorder[i])
            low = preorder[i--];
        preorder[++i] = p;
    }
    return true;
}
```

思路：左子树的所有点必小于当前点，所以遇到较大点后，弹出的小于该点所有数都应小于后继点。压缩空间方法：利用原数组内元素。每次遇到更大的则存于

256. Paint House Easy

There are a row of *n* houses, each house can be painted with one of the three colors: red, blue or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a *n* x *3* cost matrix. For example, costs[0][0] is the cost of painting house 0 with color red; costs[1][2] is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

Note:

All costs are positive integers.

```
public int minCost(int[][] costs) {
    if (costs == null || costs.length == 0) {
        return 0;
    }
    for (int i = 1; i < costs.length; i++) {
        costs[i][0] += Math.min(costs[i - 1][1], costs[i - 1][2]);
        costs[i][1] += Math.min(costs[i - 1][0], costs[i - 1][2]);
        costs[i][2] += Math.min(costs[i - 1][1], costs[i - 1][0]);
    }
    int n = costs.length - 1;
    return Math.min(Math.min(costs[n][0], costs[n][1]), costs[n][2]);
}
```

思路：动态规划。每种 Color 的记录在一个里面，分别计算使用另外两种 Color 的最小成本，最后取三种中的最小。

257. Binary Tree Paths Easy

Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:

```
  1
/ \
2  3
\
 5
```

All root-to-leaf paths are:

```
["1->2->5", "1->3"]
```

```
public List<String> binaryTreePaths(TreeNode root) {
    List<String> answer = new ArrayList<String>();
    if (root != null)
        searchBT(root, "", answer);
    return answer;
}

private void searchBT(TreeNode root, String path, List<String> answer) {
    if (root.left == null && root.right == null)
        answer.add(path + root.val);
    if (root.left != null)
        searchBT(root.left, path + root.val + "->", answer);
    if (root.right != null)
        searchBT(root.right, path + root.val + "->", answer);
}
```

思路：前序遍历。

258. Add Digits Easy

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.
For example:
Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.
Follow up:
Could you do it without any loop/recursion in O(1) runtime?
Ref: Digit Root: https://en.wikipedia.org/wiki/Digital_root#Congruence_formula

```
public int addDigits(int num) {
    return 1 + (num - 1) % 9;
}
```

259. 3Sum Smaller Medium

Given an array of *n* integers *nums* and a *target*, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.
For example, given *nums* = [-2, 0, 1, 3], and *target* = 2.
Return 2. Because there are two triplets which sums are less than 2:

```
[-2, 0, 1]
[-2, 0, 3]
```

Follow up:
Could you solve it in $O(n^2)$ runtime?

```
public int threeSumSmaller(int[] nums, int target) {
    Arrays.sort(nums);
    int sum = 0;
    for (int i = 0; i < nums.length - 2; i++)
        sum += twoSumSmaller(nums, i + 1, target - nums[i]);
    return sum;
}

private int twoSumSmaller(int[] nums, int startIndex, int target) {
    int sum = 0;
    int left = startIndex;
    int right = nums.length - 1;
    while (left < right) {
        if (nums[left] + nums[right] < target) {
            sum += right - left;// 所有小于 Right 大于 Left 的坐标和 Left 坐标的结合都行
            left++;
        } else
            right--;
    }
    return sum;
}
```

思路：先排序，然后逐一数字看有没有另外两个数字可以满足条件。$O(n^2)$

260. Single Number III Medium

Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.
For example:
Given nums = [1, 2, 1, 3, 2, 5], return [3, 5].
Note:

1. The order of the result is not important. So in the above example, [5, 3] is also correct.
2. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

```
public int[] singleNumber(int[] nums) {
    int diff = 0;
```

```
    for (int num : nums)
        diff ^= num;
    diff &= -diff;
    int[] rets = { 0, 0 };
    for (int num : nums) {
        if ((num & diff) == 0)
            rets[0] ^= num;
        else
            rets[1] ^= num;
    }
    return rets;
}
```

思路：已知单数求解方法，只要将所有数分区，让两数分别落入不同分区。抑或可以保存两数的不同位的值，该值与自己相反数之按位与可得到两数最右面一位。其实取其中任一位都可以用以分区，可以证明：任一位是 1 都说明两数该位不同（取反）。分区后按位抑或或得到两数。

261. Graph Valid Tree Medium

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

For example:

Given n = 5 and edges = [[0, 1], [0, 2], [0, 3], [1, 4]], return true.

Given n = 5 and edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]], return false.

Note: you can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0]and thus will not appear together in edges.

```
public boolean validTree(int n, int[][] edges) {
    int[] islands = new int[n];
    for (int i = 0; i < n; ++i)
        islands[i] = i;
    for (int i = 0; i < edges.length; ++i) {
        int left = find(islands, edges[i][0]);
        int right = find(islands, edges[i][1]);
        if (left == right)
            return false;
        islands[left] = islands[right];
        --n;
    }
    return n == 1;
}

int find(int[] islands, int i) {
    if (islands[i] == i)
        return i;
    return find(islands, islands[i]);
}
```

思路：并查集。要成树，不能有环，且边数=结点数-1。

262. Trips and Users Hard

```
Create table If Not Exists Trips (Id int, Client_Id int, Driver_Id int,
City_Id int, Status ENUM('completed', 'cancelled_by_driver', 'cancelled_by_client'), Request_at varchar(50));
Create table If Not Exists Users (Users_Id int, Banned varchar(50), Role ENUM('client', 'driver', 'partner'));
Truncate table Trips;
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('1', '1', '10', '1', 'completed', '2013-10-01');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status,
Request_at) values ('2', '2', '11', '1', 'cancelled_by_driver', '2013-10-01');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('3', '3', '12', '6', 'completed', '2013-10-01');
```

```
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status,
Request_at) values ('4', '4', '13', '6', 'cancelled_by_client', '2013-10-01');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('5', '1', '10', '1', 'completed', '2013-10-
02');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('6', '2', '11', '6', 'completed', '2013-10-
02');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('7', '3', '12', '6', 'completed', '2013-10-
02');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('8', '2', '12', '12', 'completed', '2013-
10-03');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status, Request_at) values ('9', '3', '10', '12', 'completed', '2013-
10-03');
insert into Trips (Id, Client_Id, Driver_Id, City_Id, Status,
Request_at) values ('10', '4', '13', '12', 'cancelled_by_driver', '2013-10-03');
Truncate table Users;
insert into Users (Users_Id, Banned, Role) values ('1', 'No', 'client');
insert into Users (Users_Id, Banned, Role) values ('2', 'Yes', 'client');
insert into Users (Users_Id, Banned, Role) values ('3', 'No', 'client');
insert into Users (Users_Id, Banned, Role) values ('4', 'No', 'client');
insert into Users (Users_Id, Banned, Role) values ('10', 'No', 'driver');
insert into Users (Users_Id, Banned, Role) values ('11', 'No', 'driver');
insert into Users (Users_Id, Banned, Role) values ('12', 'No', 'driver');
insert into Users (Users_Id, Banned, Role) values ('13', 'No', 'driver');
```

The Trips table holds all taxi trips. Each trip has a unique Id, while Client_Id and Driver_Id are both foreign keys to the Users_Id at the Users table. Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').

```
+----+-----------+-----------+---------+--------------------+----------+
| Id | Client_Id | Driver_Id | City_Id |       Status       |Request_at|
+----+-----------+-----------+---------+--------------------+----------+
| 1  |    1      |    10     |    1    |     completed      |2013-10-01|
| 2  |    2      |    11     |    1    | cancelled_by_driver|2013-10-01|
| 3  |    3      |    12     |    6    |     completed      |2013-10-01|
| 4  |    4      |    13     |    6    | cancelled_by_client|2013-10-01|
| 5  |    1      |    10     |    1    |     completed      |2013-10-02|
| 6  |    2      |    11     |    6    |     completed      |2013-10-02|
| 7  |    3      |    12     |    6    |     completed      |2013-10-02|
| 8  |    2      |    12     |    12   |     completed      |2013-10-03|
| 9  |    3      |    10     |    12   |     completed      |2013-10-03|
| 10 |    4      |    13     |    12   | cancelled_by_driver|2013-10-03|
+----+-----------+-----------+---------+--------------------+----------+
```

The Users table holds all users. Each user has an unique Users_Id, and Role is an ENUM type of ('client', 'driver', 'partner').

```
+----------+--------+--------+
| Users_Id | Banned |  Role  |
+----------+--------+--------+
|    1     |   No   | client |
|    2     |  Yes   | client |
|    3     |   No   | client |
|    4     |   No   | client |
|    10    |   No   | driver |
|    11    |   No   | driver |
|    12    |   No   | driver |
```

```
|  13   |  No   | driver |
+----------+--------+--------+
```

Write a SQL query to find the cancellation rate of requests made by unbanned clients between **Oct 1, 2013** and **Oct 3, 2013**. For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to *two* decimal places.

```
+------------+-------------------+
|    Day     | Cancellation Rate |
+------------+-------------------+
| 2013-10-01 |      0.33         |
| 2013-10-02 |      0.00         |
| 2013-10-03 |      0.50         |
+------------+-------------------+
```

```
select
t.Request_at Day,
round(sum(case when t.Status like 'cancelled_%' then 1 else 0 end)/count(*),2) Rate
from Trips t
inner join Users u
on t.Client_Id = u.Users_Id and u.Banned='No'
where t.Request_at between '2013-10-01' and '2013-10-03'
group by t.Request_at
```

263. Ugly Number Easy

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7.

Note that 1 is typically treated as an ugly number.

```
public boolean isUgly(int num) {
    for (int i = 2; i < 6 && num > 0; i++)
        while (num % i == 0)
            num /= i;
    return num == 1;
}
```

思路：利用%运算，小于 6 的（2,3,5，4 其实是 2*2，所以不会出现，因为 2 时已经把 2 用完了）逐一去掉可能的因子。

264. Ugly Number II Medium

Write a program to find the n-th ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of the first 10 ugly numbers.

Note that 1 is typically treated as an ugly number, and *n* **does not exceed 1690**.

```
public int nthUglyNumber(int n) {
    int[] ugly = new int[n];
    ugly[0] = 1;
    int index2 = 0, index3 = 0, index5 = 0;
    int factor2 = 2, factor3 = 3, factor5 = 5;
    for (int i = 1; i < n; i++) {
        int min = Math.min(Math.min(factor2, factor3), factor5);
        ugly[i] = min;
        if (factor2 == min)
            factor2 = 2 * ugly[++index2];
        if (factor3 == min)
            factor3 = 3 * ugly[++index3];
        if (factor5 == min)
```

```
            factor5 = 5 * ugly[++index5];
    }
    return ugly[n - 1];
}
```

思路：动态规划。和 2、3、5 的积才有可能是解，所以只要找到这些积的可能性。最小值乘相应数后，就会变大，下一个最小值暴露。详解

265. Paint House II Hard

There are a row of *n* houses, each house can be painted with one of the *k* colors. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a *n* x *k* cost matrix. For example, costs[0][0] is the cost of painting house 0 with color 0; costs[1][2] is the cost of painting house 1 with color 2, and so on... Find the minimum cost to paint all houses.

Note:

All costs are positive integers.

Follow up:

Could you solve it in *O*(*nk*) runtime?

```
public int minCostII(int[][] costs) {
    int n = costs.length;
    if (n == 0)
        return 0;
    int k = costs[0].length, last = 0, min = Integer.MAX_VALUE, secondMin = Integer.MAX_VALUE;
    for (int j = 0; j < k; ++j) {
        if (costs[0][j] < min) {
            secondMin = min;
            min = costs[0][j];
            last = j;
        } else if (costs[0][j] < secondMin)
            secondMin = costs[0][j];
    }
    for (int i = 1; i < n; ++i) {
        int fm = Integer.MAX_VALUE, sm = Integer.MAX_VALUE, cur = 0;
        ;
        for (int j = 0; j < k; ++j) {
            int val = costs[i][j] + (last == j ? secondMin : min);
            if (val < fm) {
                sm = fm;
                fm = val;
                cur = j;
            } else if (val < sm) {
                sm = val;
            }
        }
        min = fm;
        secondMin = sm;
        last = cur;
    }
    return min;
}
```

思路 1（略）：二维动态规划，例举所有可能，只要跳开上一颜色与当前颜色同色的情况即可。O(nkk)

思路：动态规划。仔细观察，其实不需要记录所有中间状态，只须知道当前和上一步状态，并且只需要知道上位位最小及其颜色以及次小的值，那么在当前步时，只须把上一步最小+当前最小即得，其中如果当前与上步最小颜色相同时，则使用上步的次小。

266. Palindrome Permutation Easy

Given a string, determine if a permutation of the string could form a palindrome.

For example,
"code" -> False, "aab" -> True, "carerac" -> True.

```
public boolean canPermutePalindrome(String s) {
    int[] map = new int[128];
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        map[s.charAt(i)]++;
        if (map[s.charAt(i)] % 2 == 0)
            count--;
        else
            count++;
    }
    return count <= 1;
}
```

思路：因为要形成 Palindrome，则必须字母成对出现，最多有一个单字母（中心）。所以只要数一数看成对的是不是占了全部或全部-1。

267. Palindrome Permutation II Medium

Given a string s, return all the palindromic permutations (without duplicates) of it. Return an empty list if no palindromic permutation could be form.
For example:
Given s = "aabb", return ["abba", "baab"].
Given s = "abc", return [].

```
void getPerm(List<Character> list, String mid, boolean[] used, StringBuilder sb, List<String> res)
{
    if (sb.length() == list.size()) {
        res.add(sb.toString() + mid + sb.reverse().toString());
        sb.reverse();
        return;
    }
    for (int i = 0; i < list.size(); i++) {
        if (i > 0 && list.get(i) == list.get(i - 1) && !used[i - 1])
            continue;
        if (!used[i]) {
            used[i] = true;
            sb.append(list.get(i));
            getPerm(list, mid, used, sb, res);
            used[i] = false;
            sb.deleteCharAt(sb.length() - 1);
        }
    }
}
```

思路：回溯法。准备工作与上题类似，先把出现两次的数字存入一个数组（可以重复），并记录中间位那个单独数字（如果有）。然后通过回溯组合所有可能。因为偶数的一定是以中间数字轴对称，所以只要排列一侧即可。

268. Missing Number Easy

Given an array containing *n* distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.
For example,
Given *nums* = [0, 1, 3] return 2.
Note:
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

```
public int missingNumber(int[] nums) {
    int n = nums.length + 1;
```

```
    int res = 0;
    for (int i = 1; i < n; ++i) {
        res ^= i ^ nums[i - 1];
    }
    return res;
}
```

思路：一个数和它本^为 0，所以只要与前 n 个数都取^，而一个数和 0^得它本身。O(n)

269. Alien Dictionary Hard

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where **words are sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

Example 1:

Given the following words in dictionary,

```
[ "wrt",
  "wrf",
  "er",
  "ett",
  "rftt"]
```

The correct order is: `"wertf"`.

Example 2:

Given the following words in dictionary,

```
[ "z",
  "x"]
```

The correct order is: `"zx"`.

Example 3:

Given the following words in dictionary,

```
[ "z",
  "x",
  "z"]
```

The order is invalid, so return `""`.

Note:

1. You may assume all letters are in lowercase.
2. You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return any one of them is fine.

```
private final int N = 26;

public String alienOrder(String[] words) {
    boolean[][] adj = new boolean[N][N];
    int[] visited = new int[N];
    buildGraph(words, adj, visited);

    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < N; i++) {
        if (visited[i] == 0) { // unvisited
            if (!dfs(adj, visited, sb, i))
                return "";
```

```
        }
    }
    return sb.reverse().toString();
}

public boolean dfs(boolean[][] adj, int[] visited, StringBuilder sb, int i) {
    visited[i] = 1; // 1 = visiting
    for (int j = 0; j < N; j++) {
        if (adj[i][j]) { // connected
            if (visited[j] == 1)
                return false; // 1 => 1, cycle
            if (visited[j] == 0) { // 0 = unvisited
                if (!dfs(adj, visited, sb, j))
                    return false;
            }
        }
    }
    visited[i] = 2; // 2 = visited
    sb.append((char) (i + 'a'));
    return true;
}

public void buildGraph(String[] words, boolean[][] adj, int[] visited) {
    Arrays.fill(visited, -1); // -1 = not even existed
    for (int i = 0; i < words.length; i++) {
        for (char c : words[i].toCharArray())
            visited[c - 'a'] = 0;
        if (i > 0) {
            String w1 = words[i - 1], w2 = words[i];
            int len = Math.min(w1.length(), w2.length());
            for (int j = 0; j < len; j++) {
                char c1 = w1.charAt(j), c2 = w2.charAt(j);
                if (c1 != c2) {
                    adj[c1 - 'a'][c2 - 'a'] = true;
                    break;
                }
            }
        }
    }
}
```

思路：拓扑排序。字母顺序可由相邻词第一个同位不等字母决定，由此建立邻接矩阵。遍历时，三种状态，未访问、访问中和已访问。每次任选一点出发，则其后所有点都会触及，如果恰好是根，则所有结点都可一次遍历完成。如果不是，则先完成一棵子树（倒序）。根结点所在树必然最后一个完成，所以全部保存到 StringBuilder 中，最后逆序就得结果。

270. Closest Binary Search Tree Value Easy

Given a non-empty binary search tree and a target value, find the value in the BST that is closest to the target.

Note:

- Given target value is a floating point.
- You are guaranteed to have only one unique value in the BST that is closest to the target.

```
public int closestValue(TreeNode root, double target) {
    int a = root.val;
    TreeNode kid = target < a ? root.left : root.right;
    if (kid == null)
        return a;
    int b = closestValue(kid, target);
    return Math.abs(a - target) < Math.abs(b - target) ? a : b;
}
```

思路：递归求解。先逐一探底找最近的，直到叶子结点，然后再在路径中取较近的那个值。

271. Encode and Decode Strings Medium

Design an algorithm to encode **a list of strings** to **a string**. The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

```
string encode(vector<string> strs) {
  // ... your code
  return encoded_string;
}
```

Machine 2 (receiver) has the function:

```
vector<string> decode(string s) {
  //... your code
  return strs;
}
```

So Machine 1 does:

```
string encoded_string = encode(strs);
```

and Machine 2 does:

```
vector<string> strs2 = decode(encoded_string);
```

strs2 in Machine 2 should be the same as strs in Machine 1.

Implement the encode and decode methods.

Note:

- The string may contain any possible characters out of 256 valid ascii characters. Your algorithm should be generalized enough to work on any possible characters.
- Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.
- Do not rely on any library method such as eval or serialize methods. You should implement your own encode/decode algorithm.

```
public class Codec {

    // Encodes a list of strings to a single string.
    public String encode(List<String> strs) {
        StringBuilder sb = new StringBuilder();
        for (String str : strs)
            sb.append(str.length()).append(":").append(str);
        return sb.toString();
    }

    // Decodes a single string to a list of strings.
    public List<String> decode(String s) {
        List<String> ans = new ArrayList<>();
        StringBuilder sb = new StringBuilder();
        char[] cs = s.toCharArray();
        for (int i = 0, j = 0; i < cs.length; ++i) {
            char c = cs[i];
            if (j == 0) {
                if (c != ':')
                    sb.append(c);
                else {
                    j = Integer.valueOf(sb.toString());
                    if (j == 0)
                        ans.add("");
```

```
                    else
                        j += i;
                    sb = new StringBuilder();
                }
            } else if (j == i) {
                sb.append(c);
                ans.add(sb.toString());
                sb = new StringBuilder();
                j = 0;
            } else {
                sb.append(c);
            }
        }
        return ans;
    }
}
```

思路：本质是把字符串数组变成一个字符串，再转回数组，转成字符串时记录每一个字符的长度，然后用一特殊字符表示字串开始，这样取时先把数字取出，然后按长度取字串就可以了。

272. Closest Binary Search Tree Value II Hard

Given a non-empty binary search tree and a target value, find *k* values in the BST that are closest to the target.

Note:

- Given target value is a floating point.
- You may assume *k* is always valid, that is: *k* ≤ total nodes.
- You are guaranteed to have only one unique set of *k* values in the BST that are closest to the target.

Follow up:

Assume that the BST is balanced, could you solve it in less than *O*(*n*) runtime (where *n* = total nodes)?

```
public List<Integer> closestKValues(TreeNode root, double target, int k) {
    List<Integer> res = new ArrayList<>();
    Stack<Integer> preStack = new Stack<>();
    Stack<Integer> sucStack = new Stack<>();
    inorder(root, target, false, preStack);
    inorder(root, target, true, sucStack);
    while (k-- > 0) {
        if (preStack.isEmpty())
            res.add(sucStack.pop());
        else if (sucStack.isEmpty())
            res.add(preStack.pop());
        else if (Math.abs(preStack.peek() - target) < Math.abs(sucStack.peek() - target))
            res.add(preStack.pop());
        else
            res.add(sucStack.pop());
    }
    return res;
}

void inorder(TreeNode root, double target, boolean reverse, Stack<Integer> stack) {
    if (root == null)
        return;
    inorder(reverse ? root.right : root.left, target, reverse, stack);
    if ((reverse && root.val <= target) || (!reverse && root.val > target))
        return;
    stack.push(root.val);
    inorder(reverse ? root.left : root.right, target, reverse, stack);
}
```

思路：把大于和小于目标值的值分别存入不同的 Stack，离目标值越近的越后添加，这样只要对比两个栈内元素值，按次序出栈就可以了。用中序遍历进行查找。特殊情况：一侧栈为空时，则都在另一侧栈内。

273. Integer to English Words Hard

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than $2^{31} - 1$. For example,

```
123 -> "One Hundred Twenty Three"
12345 -> "Twelve Thousand Three Hundred Forty Five"
1234567 -> "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
```

```
private final String[] LESS_THAN_20 = { "", "One", "Two", "Three", "Four", "Five", "Six", "Seven",
"Eight", "Nine",
        "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen",
"Eighteen",
        "Nineteen" };
private final String[] TENS = { "", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty",
"Seventy", "Eighty",
        "Ninety" };
private final String[] THOUSANDS = { "", "Thousand", "Million", "Billion" };

public String numberToWords(int num) {
    if (num == 0)
        return "Zero";
    int i = 0;
    String words = "";
    while (num > 0) {
        if (num % 1000 != 0)
            words = helper(num % 1000) + THOUSANDS[i] + " " + words;
        num /= 1000;
        i++;
    }
    return words.trim();
}

private String helper(int num) {
    if (num == 0)
        return "";
    else if (num < 20)
        return LESS_THAN_20[num] + " ";
    else if (num < 100)
        return TENS[num / 10] + " " + helper(num % 10);
    else
        return LESS_THAN_20[num / 100] + " Hundred " + helper(num % 100);
}
```

思路：根据英语特点，1000 及以上 3 个数分一组，1000 以内的 100、20 分别是两条线。每组对应几种情况。
O(n)

274. H-Index Medium

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index *h* if *h* of his/her *N* papers have **at least** *h* citations each, and the other *N* − *h* papers have **no more than** *h* citations each."

For example, given citations = [3, 0, 6, 1, 5], which means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively. Since the researcher has 3 papers with **at least** 3 citations each and the remaining two with **no more than** 3 citations each, his h-index is 3.

Note: If there are several possible values for h, the maximum one is taken as the h-index.

```
public int hIndex(int[] citations) {
    int n = citations.length;
    int[] papers = new int[n + 1];
    for (int c : citations)
        papers[Math.min(n, c)]++;
```

```
    int k = n;
    for (int s = papers[n]; k > s; s += papers[k])
        k--;
    return k;
}
```

思路：计数排序（Counting Sort）。先按引用次数进行计数排序（次数大于 n 的全部放到 n 位置，因为 h 必不大于 n），然后从最右面开始找，直到个数大于 k（每次 k-1，并把该位置的数字加到 s 中进行比较），这样最后 k 右面的引用总数 s>k。

275. H-Index II Medium

Follow up for H-Index: What if the citations array is sorted in ascending order? Could you optimize your algorithm?

思路：二分查找。

276. Paint Fence Easy

There is a fence with n posts, each post can be painted with one of the k colors.

You have to paint all the posts such that no more than two adjacent fence posts have the same color.

Return the total number of ways you can paint the fence.

Note:

n and k are non-negative integers.

```
public int hIndex(int[] citations) {
    if (citations == null || citations.length == 0)
        return 0;
    int l = 0, r = citations.length;
    int n = citations.length;
    while (l < r) {
        int mid = l + (r - l) / 2;
        if (citations[mid] == n - mid)
            return n - mid;
        if (citations[mid] < citations.length - mid)
            l = mid + 1;
        else
            r = mid;
    }
    return n - l;
}
```

思路：动态规划。同 198 题。

277. Find the Celebrity Medium

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

Note: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

```
public int findCelebrity(int n) {
    int candidate = 0;
    for (int i = 1; i < n; i++) {
        if (knows(candidate, i))
            candidate = i;
    }
    for (int i = 0; i < n; i++) {
        if (i != candidate && (knows(candidate, i) || !knows(i, candidate)))
            return -1;
```

```
    }
    return candidate;
}
```
思路：所有人都知道 c，而 c 不知道任何人，所以如果 0 是 c，则它不可能知道任何其他人，如果 0 不是 c，则他一定知道 c，以此类推，找到唯一的一个可能是 c 的。然后判断是否符合条件。O(n)

278. First Bad Version Easy

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

```
public int firstBadVersion(int n) {
    int left = 1;
    int right = n;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (isBadVersion(mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}
```
思路：典型的二分查找。O(lgn)

279. Perfect Squares Medium

Given a positive integer *n*, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to *n*.

For example, given *n* = 12, return 3 because 12 = 4 + 4 + 4; given *n* = 13, return 2 because 13 = 4 + 9.

```
public int numSquares(int n) {
    int[] dp = new int[n+1];
    Arrays.fill(dp, Integer.MAX_VALUE);
    dp[0] = 0;
    dp[1] = 1;
    for(int i = 1;i <= n; ++i) {
        for(int j = 1;j*j <= i; ++j)
            dp[i] = Math.min(dp[i], dp[i-j*j] + 1);
    }
    return dp[n];
}
```
思路：动态规划。寻找 dp[i]最小的可能性，只要寻找 i 减去平方为 i 及以下的所有可能平方数（i-j*j）的可能性+1 中最小的值。O(n^2)

280. Wiggle Sort Medium

Given an unsorted array nums, reorder it **in-place** such that nums[0] <= nums[1] >= nums[2] <= nums[3]....

For example, given nums = [3, 5, 2, 1, 6, 4], one possible answer is [1, 6, 2, 5, 3, 4].

```
public void wiggleSort(int[] nums) {
    for (int i = 0; i < nums.length - 1; i++)
        if ((i % 2 == 0) == (nums[i] > nums[i + 1]))
            swap(nums, i, i + 1);
}
```

```
private static void swap(int[] nums, int i, int j) {
    int temp = nums[i];
    nums[i] = nums[j];
    nums[j] = temp;
}
```
思路：索引为双数的应小于后面的数，反之大于，利用这个规则对相近的两数进行置换。

281. Zigzag Iterator Medium

Given two 1d vectors, implement an iterator to return their elements alternately.
For example, given two 1d vectors:

```
v1 = [1, 2]
v2 = [3, 4, 5, 6]
```

By calling *next* repeatedly until *hasNext* returns false, the order of elements returned by *next* should be: [1, 3, 2, 4, 5, 6].
Follow up: What if you are given k 1d vectors? How well can your code be extended to such cases?
Clarification for the follow up question - Update (2015-09-18):
The "Zigzag" order is not clearly defined and is ambiguous for k > 2 cases. If "Zigzag" does not look right to you, replace "Zigzag" with "Cyclic". For example, given the following input:

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

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

```
public class ZigzagIterator implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> list;

    public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
        list = new LinkedList<Iterator<Integer>>();
        if (!v1.isEmpty())
            list.add(v1.iterator());
        if (!v2.isEmpty())
            list.add(v2.iterator());
    }

    @Override
    public Integer next() {
        Iterator<Integer> poll = list.remove();
        int result = (Integer) poll.next();
        if (poll.hasNext())
            list.add(poll);
        return result;
    }

    @Override
    public boolean hasNext() {
        return !list.isEmpty();
    }
}
```
思路：利用一个 List 存两个 List 的 Iterator，每次用完一个元素，则把两个的位置换一下（拿出第一个，再加入）。

282. Expression Add Operators Hard

Given a string that contains only digits 0-9 and a target value, return all possibilities to add **binary** operators (not unary) +, -, or *between the digits so they evaluate to the target value.
Examples:

```
"123", 6 -> ["1+2+3", "1*2*3"]
"232", 8 -> ["2*3+2", "2+3*2"]
"105", 5 -> ["1*0+5","10-5"]
"00", 0 -> ["0+0", "0-0", "0*0"]
"3456237490", 9191 -> []
```

```
public List<String> addOperators(String num, int target) {
    List<String> rst = new ArrayList<String>();
    if (num == null || num.length() == 0)
        return rst;
    helper(rst, "", num, target, 0, 0, 0);
    return rst;
}

private void helper(List<String> rst, String path, String num, int target, int pos, long eval,
long multed) {
    if (pos == num.length()) {
        if (target == eval)
            rst.add(path);
        return;
    }
    for (int i = pos; i < num.length(); i++) {
        if (i != pos && num.charAt(pos) == '0')// 数字不能以 0 开头
            break;
        long cur = Long.parseLong(num.substring(pos, i + 1));
        if (pos == 0) {
            helper(rst, path + cur, num, target, i + 1, cur, cur);
        } else {
            helper(rst, path + "+" + cur, num, target, i + 1, eval + cur, cur);
            helper(rst, path + "-" + cur, num, target, i + 1, eval - cur, -cur);
            helper(rst, path + "*" + cur, num, target, i + 1, eval - multed + multed * cur, multed
* cur);
        }
    }
}
```

思路：回溯法，走每一种可能的路径，每次都需要注意 0，记录下次乘的被乘数。注意溢出情况。O(n!)

283. Move Zeroes Easy

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

Note:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

```
public void moveZeroes(int[] nums) {
    int temp;
    for (int lastNonZeroFoundAt = 0, cur = 0; cur < nums.length; cur++) {
        if (nums[cur] != 0) {
            temp = nums[lastNonZeroFoundAt];
            nums[lastNonZeroFoundAt++] = nums[cur];
            nums[cur] = temp;
        }
    }
}
```

思路：双指针。重点是非零数字，把他们依次移到前面，原位置置为零即可，如果本来就是相应位置，则不动。O(n)

284. Peeking Iterator Medium

Given an Iterator class interface with methods: next() and hasNext(), design and implement a PeekingIterator that support the peek() operation -- it essentially peek() at the element that will be returned by the next call to next().

Here is an example. Assume that the iterator is initialized to the beginning of the list: [1, 2, 3].
Call next() gets you 1, the first element in the list.
Now you call peek() and it returns 2, the next element. Calling next() after that ***still*** return 2.
You call next() the final time and it returns 3, the last element. Calling hasNext() after that should return false.
Follow up: How would you extend your design to be generic and work with all types, not just integer?

```
class PeekingIterator implements Iterator<Integer> {

    private Integer next = null;
    private Iterator<Integer> iter;

    public PeekingIterator(Iterator<Integer> iterator) {
        iter = iterator;
        if (iter.hasNext())
            next = iter.next();
    }

    // Returns the next element in the iteration without advancing the iterator.
    public Integer peek() {
        return next;
    }

    // hasNext() and next() should behave the same as in the Iterator interface.
    // Override them if needed.
    @Override
    public Integer next() {
        Integer res = next;
        next = iter.hasNext() ? iter.next() : null;
        return res;
    }

    @Override
    public boolean hasNext() {
        return next != null;
    }
}
```

285. Inorder Successor in BST Medium

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.
Note: If the given node has no in-order successor in the tree, return null.

```
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
    if (root == null)
        return null;
    if (root.val <= p.val) {
        return inorderSuccessor(root.right, p);
    } else {
        TreeNode left = inorderSuccessor(root.left, p);
        return (left != null) ? left : root;
    }
}
```

思路：BST 中序遍历结果是升序数组。中序遍历结果是第一个比该值大的值是其后续。O(n)

286. Walls and Gates Medium

You are given a *m x n* 2D grid initialized with these three possible values.

1. -1 - A wall or an obstacle.

2. 0 - A gate.
3. INF - Infinity means an empty room. We use the value $2^{31} - 1 = 2147483647$ to represent INF as you may assume that the distance to a gate is less than 2147483647.

Fill each empty room with the distance to its *nearest* gate. If it is impossible to reach a gate, it should be filled with INF.

For example, given the 2D grid:

```
INF -1 0 INF
INF INF INF -1
INF -1 INF -1
 0 -1 INF INF
```

After running your function, the 2D grid should be:

```
 3 -1 0 1
 2 2 1 -1
 1 -1 2 -1
 0 -1 3 4
```

```java
public void wallsAndGates(int[][] rooms) {
    if (rooms.length == 0 || rooms[0].length == 0)
        return;
    Queue<int[]> queue = new LinkedList<>();
    for (int i = 0; i < rooms.length; i++) {
        for (int j = 0; j < rooms[0].length; j++) {
            if (rooms[i][j] == 0)
                queue.add(new int[]{i, j});
        }
    }
    while (!queue.isEmpty()) {
        int[] top = queue.remove();
        int row = top[0], col = top[1];
        if (row > 0 && rooms[row - 1][col] == Integer.MAX_VALUE) {
            rooms[row - 1][col] = rooms[row][col] + 1;
            queue.add(new int[]{row - 1, col});
        }
        if (row < rooms.length - 1 && rooms[row + 1][col] == Integer.MAX_VALUE) {
            rooms[row + 1][col] = rooms[row][col] + 1;
            queue.add(new int[]{row + 1, col});
        }
        if (col > 0 && rooms[row][col - 1] == Integer.MAX_VALUE) {
            rooms[row][col - 1] = rooms[row][col] + 1;
            queue.add(new int[]{row, col - 1});
        }
        if (col < rooms[0].length - 1 && rooms[row][col + 1] == Integer.MAX_VALUE) {
            rooms[row][col + 1] = rooms[row][col] + 1;
            queue.add(new int[]{row, col + 1});
        }
    }
}
```

思路：先扫一遍得到所有门，然后从每个门 BFS 更新周围的所有房间。因为是 BFS，所以不会覆盖较小的值（访问过的标记为非 Max）。

287. Find the Duplicate Number Medium

Given an array *nums* containing *n* + 1 integers where each integer is between 1 and *n* (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Note:

1. You **must not** modify the array (assume the array is read only).

2. You must use only constant, *O*(1) extra space.
3. Your runtime complexity should be less than $O(n^2)$.
4. There is only one duplicate number in the array, but it could be repeated more than once.

```
public int findDuplicate(int[] nums) {
    Set<Integer> s = new HashSet<Integer>();
    for (int num : nums) {
        if (!s.add(num)) {
            return num;
        }
    }
    throw new RuntimeException("no duplicate!");
}
```

思路：借助 Set 找重复字符。

288. Unique Word Abbreviation Medium

An abbreviation of a word follows the form <first letter><number><last letter>. Below are some examples of word abbreviations:

```
a) it                --> it    (no abbreviation)
   1
b) d|o|g             --> d1g
       1   1 1
   1---5----0----5--8
c) i|nternationalizatio|n  --> i18n
       1
   1---5----0
d) l|ocalizatio|n        --> l10n
```

Assume you have a dictionary and given a word, find whether its abbreviation is unique in the dictionary. A word's abbreviation is unique if no *other* word from the dictionary has the same abbreviation.

Example:

```
Given dictionary = [ "deer", "door", "cake", "card" ]
isUnique("dear") -> false
isUnique("cart") -> true
isUnique("cane") -> false
isUnique("make") -> true
```

```
public class ValidWordAbbr {
    private final Map<String, Boolean> abbrDict = new HashMap<>();
    private final Set<String> dict;

    public ValidWordAbbr(String[] dictionary) {
        dict = new HashSet<>(Arrays.asList(dictionary));
        for (String s : dict) {
            String abbr = toAbbr(s);
            abbrDict.put(abbr, !abbrDict.containsKey(abbr));
        }
    }

    public boolean isUnique(String word) {
        String abbr = toAbbr(word);
        Boolean hasAbbr = abbrDict.get(abbr);
        return hasAbbr == null || (hasAbbr && dict.contains(word));
    }

    private String toAbbr(String s) {
        int n = s.length();
```

```
        if (n <= 2) {
            return s;
        }
        return s.charAt(0) + Integer.toString(n - 2) + s.charAt(n - 1);
    }
}
```

思路：添加过程中设置该缩写是否重复。

289. Game of Life Medium

According to the Wikipedia's article: "The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a *board* with *m* by *n* cells, each cell has an initial state *live* (1) or *dead* (0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population..
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state.

Follow up:

1. Could you solve it in-place? Remember that the board needs to be updated at the same time: You cannot update some cells first and then use their updated values to update other cells.
2. In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches the border of the array. How would you address these problems?

```
public void gameOfLife(int[][] board) {
    if (board == null || board.length == 0)
        return;
    int m = board.length, n = board[0].length;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            int lives = liveNeighbors(board, m, n, i, j);
            if (board[i][j] == 1 && lives >= 2 && lives <= 3) {
                board[i][j] = 3;
            }
            if (board[i][j] == 0 && lives == 3) {
                board[i][j] = 2;
            }
        }
    }
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            board[i][j] >>= 1;
}

public int liveNeighbors(int[][] board, int m, int n, int i, int j) {
    int lives = 0;
    for (int x = Math.max(i - 1, 0); x <= Math.min(i + 1, m - 1); x++)
        for (int y = Math.max(j - 1, 0); y <= Math.min(j + 1, n - 1); y++)
            lives += board[x][y] & 1;
    lives -= board[i][j] & 1;
    return lives;
}
```

思路：借助两个位表示现在和将来两个状态。详解

290. Word Pattern Easy

Given a `pattern` and a string `str`, find if `str` follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** word in str.

Examples:

1. pattern = "abba", str = "dog cat cat dog" should return true.
2. pattern = "abba", str = "dog cat cat fish" should return false.
3. pattern = "aaaa", str = "dog cat cat dog" should return false.
4. pattern = "abba", str = "dog dog dog dog" should return false.

Notes:

You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

```
public boolean wordPattern(String pattern, String str) {
    String[] words = str.split(" ");
    if (words.length != pattern.length())
        return false;
    Map<Object, Integer> index = new HashMap<Object, Integer>();
    for (Integer i = 0; i < words.length; ++i)
        if (index.put(pattern.charAt(i), i) != index.put(words[i], i))
            return false;
    return true;
}
```

思路：按词典对应。词典特点：put 时如有重复，返回上次 put 同 key 时的 value。

291. Word Pattern II Hard

Given a pattern and a string str, find if str follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** substring in str.

Examples:

1. pattern = "abab", str = "redblueredblue" should return true.
2. pattern = "aaaa", str = "asdasdasdasd" should return true.
3. pattern = "aabb", str = "xyzabcxzyabc" should return false.

Notes:

You may assume both pattern and str contains only lowercase letters.

```
public boolean wordPatternMatch(String pattern, String str) {
    Map<Character, String> map = new HashMap<>();
    Set<String> set = new HashSet<>();
    return isMatch(str, 0, pattern, 0, map, set);
}

boolean isMatch(String str, int i, String pat, int j, Map<Character, String> map, Set<String> set)
{
    if (i == str.length() && j == pat.length())
        return true;
    if (i == str.length() || j == pat.length())
        return false;
    char c = pat.charAt(j);
    if (map.containsKey(c)) {
        String s = map.get(c);
        if (!str.startsWith(s, i))
            return false;
        return isMatch(str, i + s.length(), pat, j + 1, map, set);
    }
    for (int k = i; k < str.length(); k++) {
        String p = str.substring(i, k + 1);
        if (set.contains(p))
            continue;
        map.put(c, p);
        set.add(p);
        if (isMatch(str, k + 1, pat, j + 1, map, set))
```

```
            return true;
        map.remove(c);
        set.remove(p);
    }
    return false;
}
```
思路：回溯法。

292. Nim Game Easy

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.
Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.
For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

```
public boolean canWinNim(int n) {
    return (n % 4 != 0);
}
```
思路：只要是 4 的倍数，第一个人必胜。

293. Flip Game Easy

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: `+` and `-`, you and your friend take turns to flip two **consecutive** `"++"` into `"--"`. The game ends when a person can no longer make a move and therefore the other person will be the winner.
Write a function to compute all possible states of the string after one valid move.
For example, given `s = "++++"`, after one move, it may become one of the following states:

```
[ "--++",
 "+--+",
 "++--"]
```

If there is no valid move, return an empty list `[]`.

```
public List<String> generatePossibleNextMoves(String s) {
    List<String> list = new ArrayList<>();
    for (int i = -1; (i = s.indexOf("++", i + 1)) >= 0;)
        list.add(s.substring(0, i) + "--" + s.substring(i + 2));
    return list;
}
```
思路：把所有可能的连续双加号换成减号的找出来。其实就是不断找下一个双加号的位置，替换并添加到结果 List 中。

294. Flip Game II Medium

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: `+` and `-`, you and your friend take turns to flip two **consecutive** `"++"` into `"--"`. The game ends when a person can no longer make a move and therefore the other person will be the winner.
Write a function to determine if the starting player can guarantee a win.
For example, given `s = "++++"`, return true. The starting player can guarantee a win by flipping the middle `"++"` to become `"+--+"`.
Follow up:
Derive your algorithm's runtime complexity.

```
public boolean canWin(String s) {
    if (s == null || s.length() < 2)
        return false;
    Map<String, Boolean> map = new HashMap<>();
```

```
    return canWin(s, map);
}

public boolean canWin(String s, Map<String, Boolean> map) {
    if (map.containsKey(s))
        return map.get(s);
    for (int i = 0; i < s.length() - 1; i++) {
        if (s.charAt(i) == '+' && s.charAt(i + 1) == '+') {
            String opponent = s.substring(0, i) + "--" + s.substring(i + 2);
            if (!canWin(opponent, map)) {
                map.put(s, true);
                return true;
            }
        }
    }
    map.put(s, false);
    return false;
}
```

思路：回溯法。通过词典记录已经出现过的情况，已经出现过的直接返回结果。每一级结果和上一级相反（因为对方赢自己就输）。

295. Find Median from Data Stream Hard

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

Examples:

[2,3,4] , the median is 3

[2,3], the median is (2 + 3) / 2 = 2.5

Design a data structure that supports the following two operations:

- void addNum(int num) - Add a integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

For example:

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

```
public class MedianFinder {
    private Queue<Long> small = new PriorityQueue<>(), large = new PriorityQueue<>();

    public void addNum(int num) {
        large.add((long) num);
        small.add(-large.poll());
        if (large.size() < small.size())
            large.add(-small.poll());
    }

    public double findMedian() {
        return large.size() > small.size() ? large.peek() : (large.peek() - small.peek()) / 2.0;
    }
}
```

思路：把新加入的数平均放到两个优先队列中（先放大，再把大的中首元素移小，保证顺序），这样加入时耗时 logn。

296. Best Meeting Point Hard

A group of two or more people wants to meet and minimize the total travel distance. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group. The distance is calculated using Manhattan Distance, where distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|.
For example, given three people living at (0,0), (0,4), and (2,2):

```
1 - 0 - 0 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

The point (0,2) is an ideal meeting point, as the total travel distance of 2+2+2=6 is minimal. So return 6.

```
public int minTotalDistance(int[][] grid) {
    List<Integer> rows = collectRows(grid);
    List<Integer> cols = collectCols(grid);
    return minDistance1D(rows) + minDistance1D(cols);
}

private int minDistance1D(List<Integer> points) {
    int distance = 0;
    int i = 0;
    int j = points.size() - 1;
    while (i < j) {
        distance += points.get(j) - points.get(i);
        i++;
        j--;
    }
    return distance;
}

private List<Integer> collectRows(int[][] grid) {
    List<Integer> rows = new ArrayList<>();
    for (int row = 0; row < grid.length; row++)
        for (int col = 0; col < grid[0].length; col++)
            if (grid[row][col] == 1)
                rows.add(row);
    return rows;
}

private List<Integer> collectCols(int[][] grid) {
    List<Integer> cols = new ArrayList<>();
    for (int col = 0; col < grid[0].length; col++)
        for (int row = 0; row < grid.length; row++)
            if (grid[row][col] == 1)
                cols.add(col);
    return cols;
}
```

思路：纵向距离和横向距离互不干扰，故降成一维处理。

297. Serialize and Deserialize Binary Tree Hard

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.
Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.
For example, you may serialize the following tree

```
  1
 /\
2  3
  /\
 4  5
```

as "[1,2,3,null,null,4,5]", just the same as how LeetCode OJ serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

```
public class Codec {
    static int i = 0;

    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
        serialize(root, sb);
        return sb.toString().trim();
    }

    private static void serialize(TreeNode root, StringBuilder sb) {
        if (root == null) {
            sb.append("# ");
            return;
        }
        sb.append(root.val);
        sb.append(" ");
        serialize(root.left, sb);
        serialize(root.right, sb);
    }

    public TreeNode deserialize(String data) {
        String[] vals = data.split(" ");
        i = 0;
        return deserialize(vals);
    }

    private static TreeNode deserialize(String[] vals) {
        String val = vals[i++];
        if ("#".equals(val))
            return null;
        TreeNode root = new TreeNode(Integer.valueOf(val));
        if (i < vals.length - 1)
            root.left = deserialize(vals);
        if (i < vals.length - 1)
            root.right = deserialize(vals);
        return root;
    }
}
```

思路：先序遍历存入数组输出，每到叶结点添加#以区分。恢复时也使用先序遍历，如果遇到#则说明上个结点为叶子结点。O(n)

298. Binary Tree Longest Consecutive Sequence Medium

Given a binary tree, find the length of the longest consecutive sequence path.

The path refers to any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The longest consecutive path need to be from parent to child (cannot be the reverse).

For example,

```
  1
```

```
  \
   3
  /\
 2  4
     \
      5
```

Longest consecutive sequence path is 3-4-5, so return 3.

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

Longest consecutive sequence path is 2-3,not3-2-1, so return 2.

```
public int longestConsecutive(TreeNode root) {
    return dfs(root, null, 0);
}

private int dfs(TreeNode p, TreeNode parent, int length) {
    if (p == null)
        return length;
    length = (parent != null && p.val == parent.val + 1) ? length + 1 : 1;
    return Math.max(length, Math.max(dfs(p.left, p, length), dfs(p.right, p, length)));
}
```

思路：递归求解。深度优先检索，每次传递当前和上一结点元素，以便比较是否正好差 1，止于当前结点为空。

299. Bulls and Cows Medium

You are playing the following Bulls and Cows game with your friend: You write down a number and ask your friend to guess what the number is. Each time your friend makes a guess, you provide a hint that indicates how many digits in said guess match your secret number exactly in both digit and position (called "bulls") and how many digits match the secret number but locate in the wrong position (called "cows"). Your friend will use successive guesses and hints to eventually derive the secret number.

For example:

```
Secret number:  "1807"
Friend's guess: "7810"
```

Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.)

Write a function to return a hint according to the secret number and friend's guess, use A to indicate the bulls and B to indicate the cows. In the above example, your function should return "1A3B".

Please note that both secret number and friend's guess may contain duplicate digits, for example:

```
Secret number:  "1123"
Friend's guess: "0111"
```

In this case, the 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow, and your function should return "1A1B".

You may assume that the secret number and your friend's guess only contain digits, and their lengths are always equal.

```
public String getHint(String secret, String guess) {
    int bulls = 0;
    int cows = 0;
    int[] numbers = new int[10];
```

```
    for (int i = 0; i < secret.length(); i++) {
        if (secret.charAt(i) == guess.charAt(i))
            bulls++;
        else {
            if (numbers[secret.charAt(i) - '0']++ < 0)// <0 说明在 guess 里已经出现过
                cows++;
            if (numbers[guess.charAt(i) - '0']-- > 0)
                cows++;
        }
    }
    return bulls + "A" + cows + "B";
}
```

思路：同位字符 cows+1，不同位往正负两个方向统计，只要与当前值与当前方向值不一致，则说明另一数字上已经有多于当前数字的该字符，故 cows+1，两边字符相等时该值归零。

300. Longest Increasing Subsequence Medium

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example,

Given [10, 9, 2, 5, 3, 7, 101, 18],

The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n^2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

```
public int lengthOfLIS(int[] nums) {
    int[] dp = new int[nums.length];
    int len = 0;
    for (int num : nums) {
        int i = Arrays.binarySearch(dp, 0, len, num);
        if (i < 0)
            i = -(i + 1);
        dp[i] = num;
        if (i == len)
            len++;
    }
    return len;
}
```

思路：将所有数重新排序，逐一数字查找索引位置，并插入 dp 对应的位置，当索引位置与 dp 长度相等时，则说明与 dp 数组中后面一位连续。

301. Remove Invalid Parentheses Hard

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses (and).

Examples:

```
"()())()" -> ["()()()", "(())()"]
"(a)())()" -> ["(a)()()", "(a())()"]
")(" -> [""]
```

```
public List<String> removeInvalidParentheses(String s) {
    List<String> ans = new ArrayList<>();
    remove(s, ans, 0, 0, new char[] { '(', ')' });
    return ans;
}

public void remove(String s, List<String> ans, int last_i, int last_j, char[] par) {
    for (int stack = 0, i = last_i; i < s.length(); ++i) {
```

```
            if (s.charAt(i) == par[0])
                stack++;
            if (s.charAt(i) == par[1])
                stack--;
            if (stack >= 0)// 左括号够多无须进行多余操作
                continue;
            for (int j = last_j; j <= i; ++j)
                if (s.charAt(j) == par[1] && (j == last_j || s.charAt(j - 1) != par[1]))
                    remove(s.substring(0, j) + s.substring(j + 1, s.length()), ans, i, j, par);
            return;
        }
        String reversed = new StringBuilder(s).reverse().toString();
        if (par[0] == '(')
            remove(reversed, ans, 0, 0, new char[] { ')', '(' });
        else
            ans.add(reversed);
    }
```

思路：回溯法。去右括号：如果右括号数量大于左括号数量，则从上一层 j（即去掉右括号后一个位置开始）到 i 位置，尝试去掉每一个右括号。去掉后，则左右括号相等，把所有右括号都尝试一遍，除了连续出现的只去每一个（因为除其中任意一个结果相同），每层 j 开始位置（last_j）不受上一位不是右括号的影响。去左括号方法：把字符串反转，左右括号判别符相换，方法同右括号方法。

302. Smallest Rectangle Enclosing Black Pixels Hard

An image is represented by a binary matrix with 0 as a white pixel and 1 as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location (x, y) of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.
For example, given the following image:

```
[ "0010",
  "0110",
  "0100"]
```

and x = 0, y = 2,
Return 6.

```
public int minArea(char[][] image, int x, int y) {
    if (image == null)
        return 0;
    int m = image.length;
    if (m == 0)
        return 0;
    int n = image[0].length;
    if (n == 0)
        return 0;
    int[] corners = new int[4];
    corners[1] = corners[3] = Integer.MIN_VALUE;
    corners[0] = corners[2] = Integer.MAX_VALUE;
    dfs(image, x, y, m, n, corners);
    return (corners[1] - corners[0] + 1) * (corners[3] - corners[2] + 1);
}

static void dfs(char[][] image, int x, int y, int m, int n, int[] corners) {
    if (x < 0 || x >= m || y < 0 || y >= n || image[x][y] != '1')
        return;
    image[x][y] = 2;
    corners[0] = Math.min(x, corners[0]);
    corners[1] = Math.max(x, corners[1]);
    corners[2] = Math.min(y, corners[2]);
    corners[3] = Math.max(y, corners[3]);
    dfs(image, x + 1, y, m, n, corners);
```

```
    dfs(image, x - 1, y, m, n, corners);
    dfs(image, x, y + 1, m, n, corners);
    dfs(image, x, y - 1, m, n, corners);
}
```

思路 **1**：深度优先查找左、右、上、下边界。O（mn）。

```
public int minArea(char[][] image, int x, int y) {
    int m = image.length, n = image[0].length;
    int left = searchColumns(image, 0, y, 0, m, true);
    int right = searchColumns(image, y + 1, n, 0, m, false);
    int top = searchRows(image, 0, x, left, right, true);
    int bottom = searchRows(image, x + 1, m, left, right, false);
    return (right - left) * (bottom - top);
}

private int searchColumns(char[][] image, int i, int j, int top, int bottom, boolean whiteToBlack)
{
    while (i != j) {
        int k = top, mid = (i + j) / 2;
        while (k < bottom && image[k][mid] == '0')
            ++k;
        if (k < bottom == whiteToBlack)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}

private int searchRows(char[][] image, int i, int j, int left, int right, boolean whiteToBlack) {
    while (i != j) {
        int k = left, mid = (i + j) / 2;
        while (k < right && image[mid][k] == '0')
            ++k;
        if (k < right == whiteToBlack)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
```

思路 **2**：二分查找。与 一维不同的是，mid 落到某一列（行）时，都要把该列（行）每一行（列）上查找一遍。可以这样做的前提是本搜索中只关注有或没有。找到值为 1 的 x 和 y 的最大、小值，然后计算面积。

O(mlogn + nlogm)

303. Range Sum Query - Immutable **Easy**

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.

Example:

```
Given nums = [-2, 0, 3, -5, 2, -1]
sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
```

Note:

1. You may assume that the array does not change.
2. There are many calls to *sumRange* function.

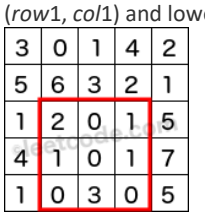
```
class NumArray {
    private int[] sum;

    public NumArray(int[] nums) {
```

```
        sum = new int[nums.length + 1];
        for (int i = 0; i < nums.length; i++) {
            sum[i + 1] = sum[i] + nums[i];
        }
    }

    public int sumRange(int i, int j) {
        return sum[j + 1] - sum[i];
    }
}
```

思路：区间差=两区间不重合的部分。

304. Range Sum Query 2D - Immutable Medium

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).



The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

Example:

```
Given matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]]
sumRegion(2, 1, 4, 3) -> 8
sumRegion(1, 1, 2, 2) -> 11
sumRegion(1, 2, 2, 4) -> 12
```

Note:

1. You may assume that the matrix does not change.
2. There are many calls to *sumRegion* function.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

```
public class NumMatrix {
    private int[][] dp;

    public NumMatrix(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0)
            return;
        dp = new int[matrix.length + 1][matrix[0].length + 1];
        for (int r = 0; r < matrix.length; r++) {
            for (int c = 0; c < matrix[0].length; c++) {
                dp[r + 1][c + 1] = dp[r + 1][c] + dp[r][c + 1] + matrix[r][c] - dp[r][c];
            }
        }
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        return dp[row2 + 1][col2 + 1] - dp[row1][col2 + 1] - dp[row2 + 1][col1] + dp[row1][col1];
    }
```

}
思路：同上题。2D 的区间差，注意把多减的部分加回来。

305. Number of Islands II **Hard**

A 2d grid map of `m` rows and `n` columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each *addLand* operation**. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

Given `m = 3, n = 3`, `positions = [[0,0], [0,1], [1,2], [2,1]]`.

Initially, the 2d grid `grid` is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

```
1 1 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

```
1 1 0
0 0 1   Number of islands = 2
0 0 0
```

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

```
1 1 0
0 0 1   Number of islands = 3
0 1 0
```

We return the result as an array: `[1, 1, 2, 3]`

Challenge:

Can you do it in time complexity O(k log mn), where k is the length of the `positions`?

```
int[][] dirs = { { 0, 1 }, { 1, 0 }, { -1, 0 }, { 0, -1 } };

public List<Integer> numIslands2(int m, int n, int[][] positions) {
    List<Integer> result = new ArrayList<>();
    if (m <= 0 || n <= 0)
        return result;
    int count = 0;
    int[] roots = new int[m * n];
    Arrays.fill(roots, -1);
    for (int[] p : positions) {
        int root = n * p[0] + p[1];
        roots[root] = root;
        count++;
        for (int[] dir : dirs) {
            int x = p[0] + dir[0];
```

```
            int y = p[1] + dir[1];
            int nb = n * x + y;
            if (x < 0 || x >= m || y < 0 || y >= n || roots[nb] == -1)
                continue;
            int rootNb = findIsland(roots, nb);
            if (root != rootNb) {// 在岛上并且不是当前点
                roots[root] = rootNb;// 当前点可通向之前点
                root = rootNb;
                count--;
            }
        }
        result.add(count);
    }
    return result;
}

public int findIsland(int[] roots, int id) {
    while (id != roots[id])
        id = roots[id];
    return id;
}
```

思路：并查集。二维转一维的方法：纵向放在十位，横向放在个位（以矩阵长度为进制）。没有岛时，增加点就增加了一个岛。有岛时增加一个点，需要判断新增加的是孤岛还是和别的岛相连（Union Find）。只有周边的点为 1 时才需要判断，如果是同有的话，Union。

306. Additive Number Medium

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

For example:

"112358" is an additive number because the digits can form an additive sequence: 1, 1, 2, 3, 5, 8.

```
1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8
```

"199100199" is also an additive number, the additive sequence is: 1, 99, 100, 199.

```
1 + 99 = 100, 99 + 100 = 199
```

Note: Numbers in the additive sequence **cannot** have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

Given a string containing only digits '0'-'9', write a function to determine if it's an additive number.

Follow up:

How would you handle overflow for very large input integers?

```
public boolean isAdditiveNumber(String num) {
    int n = num.length();
    for (int i = 1; i <= n / 2; ++i)
        for (int j = 1; Math.max(j, i) <= n - i - j; ++j)
            if (isValid(i, j, num))
                return true;
    return false;
}

private boolean isValid(int i, int j, String num) {
    if (num.charAt(0) == '0' && i > 1)
        return false;
    if (num.charAt(i) == '0' && j > 1)
        return false;
    String sum;
    Long x1 = Long.parseLong(num.substring(0, i));
    Long x2 = Long.parseLong(num.substring(i, i + j));
```

```
    for (int start = i + j; start != num.length(); start += sum.length()) {
        x2 = x2 + x1;
        x1 = x2 - x1;
        sum = x2.toString();
        if (!num.startsWith(sum, start))
            return false;
    }
    return true;
}
```

思路：根据给定规则，先找出前两数的某种可能，然后逐数字向后推导，如果出现违规则为假。

307. Range Sum Query - Mutable Medium

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.
The *update(i, val)* function modifies *nums* by updating the element at index *i* to *val*.

Example:

```
Given nums = [1, 3, 5]
sumRange(0, 2) -> 9
update(1, 2)
sumRange(0, 2) -> 8
```

Note:

1. The array is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRange* function is distributed evenly.

```
public class NumArray {
    int[] tree;
    int n;

    public NumArray(int[] nums) {
        if (nums.length > 0) {
            n = nums.length;
            tree = new int[n * 2];
            buildTree(nums);
        }
    }

    private void buildTree(int[] nums) {
        for (int i = n, j = 0; i < 2 * n; i++, j++)
            tree[i] = nums[j];
        for (int i = n - 1; i > 0; --i)
            tree[i] = tree[i * 2] + tree[i * 2 + 1];
    }

    public void update(int pos, int val) {
        pos += n;
        tree[pos] = val;
        while (pos > 0) {
            int left = pos;
            int right = pos;
            if (pos % 2 == 0)
                right = pos + 1;
            else
                left = pos - 1;
            tree[pos / 2] = tree[left] + tree[right];
            pos /= 2;
        }
    }

    public int sumRange(int l, int r) {
        l += n;
```

```
        r += n;
        int sum = 0;
        while (l <= r) {
            if ((l % 2) == 1) {
                sum += tree[l];
                l++;
            }
            if ((r % 2) == 0) {
                sum += tree[r];
                r--;
            }
            l /= 2;
            r /= 2;
        }
        return sum;
    }
}
```

思路：线段树（Segment Tree）。线段树本身记录区间和信息，更新后只更新该结点的祖先结点。线段树用数组表示，原数组值全部在叶子结点。叶子结点未必在一层，但是特点是一个结点的左右孩子必相邻且为左右子结点。根据这个特点初始化树。更新时，如果给定索引可被 2 整除，则说明其为父结点的左结点（否则为右结点），所以其右兄弟为其索引+1。不断更新父结点直到索引指到 0（0 位元素不参与计算）。

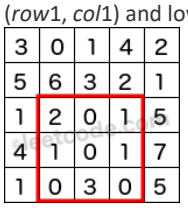
sumRange 时：

1. 如果索引的起、结恰好为某一满二叉树的左右端叶子结点，则只需要找到其根结点并返回。根据数组表示二叉树特点，逐步找父结点（每次除 2），中间所有状态，都不会构成添加值到 Sum 的条件。直到左右两值相等时（为父结点），这时必为奇或偶，所以得到其值。
2. 如果不然，则把某满二叉树外的分支单独计算再和满二叉部分合并。

思路 2：线段和（Range Sum）。每次更新，则更新所有大于 i 位置的和。

308. Range Sum Query 2D - Mutable Medium

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).



The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

Example:

```
Given matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]]
sumRegion(2, 1, 4, 3) -> 8
update(3, 2, 2)
sumRegion(2, 1, 4, 3) -> 10
```

Note:

1. The matrix is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRegion* function is distributed evenly.

3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

```
public class NumMatrix {
    int[][] tree;
    int[][] nums;
    int m;
    int n;

    public NumMatrix(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0)
            return;
        m = matrix.length;
        n = matrix[0].length;
        tree = new int[m + 1][n + 1];
        nums = new int[m][n];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                update(i, j, matrix[i][j]);
    }

    public void update(int row, int col, int val) {
        if (m == 0 || n == 0)
            return;
        int delta = val - nums[row][col];
        nums[row][col] = val;
        for (int i = row + 1; i <= m; i += i & (-i))
            for (int j = col + 1; j <= n; j += j & (-j))
                tree[i][j] += delta;
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        if (m == 0 || n == 0)
            return 0;
        return sum(row2 + 1, col2 + 1) + sum(row1, col1) - sum(row1, col2 + 1) - sum(row2 + 1,
col1);
    }

    public int sum(int row, int col) {
        int sum = 0;
        for (int i = row; i > 0; i -= i & (-i)) {
            for (int j = col; j > 0; j -= j & (-j)) {
                sum += tree[i][j];
            }
        }
        return sum;
    }
}
```

思路：2D Range Sum。

309. Best Time to Buy and Sell Stock with Cooldown Medium

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.
Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

Example:

```
prices = [1, 2, 3, 0, 2]
maxProfit = 3
```

```
transactions = [buy, sell, cooldown, buy, sell]
```

```
public int maxProfit(int[] prices) {
    int maxBuy = Integer.MIN_VALUE, maxSell = 0, preBuy = 0, preSell =
    for (int price : prices) {
        preBuy = maxBuy;
        maxBuy = Math.max(preBuy, preSell - price);
        preSell = maxSell;
        maxSell = Math.max(preSell, preBuy + price);
    }
    return maxSell;
}
```

思路：同 198 题。

310. Minimum Height Trees Medium

For a undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

Format

The graph contains `n` nodes which are labeled from `0` to `n - 1`. You will be given the number `n` and a list of undirected `edges` (each edge is a pair of labels).

You can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in `edges`.

Example 1:

Given `n = 4`, `edges = [[1, 0], [1, 2], [1, 3]]`

```
   0
   |
   1
  / \
 2   3
```

return `[1]`

Example 2:

Given `n = 6`, `edges = [[0, 3], [1, 3], [2, 3], [4, 3], [5, 4]]`

```
  0 1 2
   \|/
    3
    |
    4
    |
    5
```

return `[3, 4]`

Note:

(1) According to the definition of tree on Wikipedia: “a tree is an undirected graph in which any two vertices are connected by *exactly* one path. In other words, any connected graph without simple cycles is a tree.”

(2) The height of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

```
public List<Integer> findMinHeightTrees(int n, int[][] edges) {
    if (n == 1)
        return Collections.singletonList(0);
    List<Integer> leafList = new ArrayList<>();
    List<List<Integer>> adjList = new ArrayList<>();
    for (int i = 0; i < n; ++i)
        adjList.add(new ArrayList<>());
    for (int[] edge : edges) {
```

```
            adjList.get(edge[0]).add(edge[1]);
            adjList.get(edge[1]).add(edge[0]);
        }
        for (int i = 0; i < n; ++i)
            if (adjList.get(i).size() == 1)
                leafList.add(i);
        while (n > 2) {
            List<Integer> newLeafList = new ArrayList<>();
            for (int leaf : leafList) {
                int adjNode = adjList.get(leaf).get(0);
                adjList.get(adjNode).remove(Integer.valueOf(leaf));
                if (adjList.get(adjNode).size() == 1)
                    newLeafList.add(adjNode);
                n--;
            }
            leafList = newLeafList;
        }
        return leafList;
    }
```

思路：剥忽法。把叶子结点一层一层地去掉，直到只有两个或一个结点，就是解。最小深度树的特点：四周的树高最多差 1，根最多有两种选择。因为两者之间最多一度，否则无法成树。隐含条件：图中无环。

311. Sparse Matrix Multiplication Medium

Given two sparse matrices **A** and **B**, return the result of **AB**.

You may assume that **A**'s column number is equal to **B**'s row number.

Example:

```
A = [
 [ 1, 0, 0],
 [-1, 0, 3]
]
B = [
 [ 7, 0, 0 ],
 [ 0, 0, 0 ],
 [ 0, 0, 1 ]
]
   | 1 0 0 |   | 7 0 0 |   | 7 0 0 |
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |
        | 0 0 1 |
```

```
    public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length, n = A[0].length, nB = B[0].length;
        int[][] C = new int[m][nB];
        for (int i = 0; i < m; i++)
            for (int k = 0; k < n; k++)
                if (A[i][k] != 0) {
                    for (int j = 0; j < nB; j++)
                        if (B[k][j] != 0)
                            C[i][j] += A[i][k] * B[k][j];
                }
        return C;
    }
```

思路：暴力，无它。

312. Burst Balloons **Hard**

Given n balloons, indexed from 0 to n-1. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get nums[left] * nums[i] * nums[right] coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Note:

(1) You may imagine nums[-1] = nums[n] = 1. They are not real therefore you can not burst them.

(2) 0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100

Example:

Given [3, 1, 5, 8]

Return 167

```
   nums = [3,1,5,8] --> [3,5,8] -->   [3,8]   -->  [8]  --> []
  coins =  3*1*5    + 3*5*8   + 1*3*8    + 1*8*1  = 167
```

```
public int maxCoins(int[] nums) {
    int[] nnums = new int[nums.length + 2];
    int n = 1;
    for (int x : nums)
        if (x > 0)
            nnums[n++] = x;
    nnums[0] = nnums[n++] = 1;
    int[][] memo = new int[n][n];
    return burst(memo, nnums, 0, n - 1);
}

public int burst(int[][] memo, int[] nums, int left, int right) {
    if (left + 1 == right)
        return 0;
    if (memo[left][right] > 0)
        return memo[left][right];
    int ans = 0;
    for (int i = left + 1; i < right; ++i)
        ans = Math.max(ans,
                nums[left] * nums[i] * nums[right] + burst(memo, nums, left, i) + burst(memo,
nums, i, right));
    memo[left][right] = ans;
    return ans;
}
```

思路 1：回溯+动态规划(top down)。从每一个点作为起点，尝试所有可能，得到某一段的可能最值保存下来（备忘录）。结束条件：左右指针差 1 时，无法再爆破，返回 0；memo 中已有情况，返回该值。递推方程：当前最大可能为当前爆破值+左、右爆破最大值之和。遍历计算所有位置作为当前的爆破极值，取最大。

```
public int maxCoins(int[] nums) {
    int n = nums.length;
    int len = n + 2;
    int[] a = new int[len];
    System.arraycopy(nums, 0, a, 1, n);
    a[0] = a[len - 1] = 1;
    int[][] dp = new int[len][len];
    for (int gap = 2; gap < len; gap++) {
        for (int left = 0; left < len - gap; left++) {
            int cur = 0;
            int right = left + gap;
            for (int i = left + 1; i < right; i++)
                cur = Math.max(cur, dp[left][i] + dp[i][right] + a[left] * a[i] * a[right]);
            dp[left][right] = cur;
        }
    }
    return dp[0][len - 1];
}
```

思路 2：自底向上，中心点从 index=1 开始（位置 2），逐渐右移，这样可以利用已经计算过的部分计算未计算的部分。

313. Super Ugly Number Medium

Write a program to find the n^{th} super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list primes of size k. For example, [1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32] is the sequence of the first 12 super ugly numbers given primes = [2, 7, 13, 19] of size 4.

Note:

(1) 1 is a super ugly number for any given primes.

(2) The given numbers in primes are in ascending order.

(3) $0 < k \le 100$, $0 < n \le 10^6$, $0 < primes[i] < 1000$.

(4) The n^{th} super ugly number is guaranteed to fit in a 32-bit signed integer.

```
public int nthSuperUglyNumber(int n, int[] primes) {
    int[] ugly = new int[n];
    ugly[0] = 1;
    int[] pointer = new int[primes.length];
    for (int i = 1; i < n; i++) {
        int min = Integer.MAX_VALUE;
        int minIndex = 0;
        for (int j = 0; j < primes.length; j++) {
            if (ugly[pointer[j]] * primes[j] < min) {
                min = ugly[pointer[j]] * primes[j];
                minIndex = j;
            } else if (ugly[pointer[j]] * primes[j] == min) {
                pointer[j]++;
            }
        }
        ugly[i] = min;
        pointer[minIndex]++;
    }
    return ugly[n - 1];
}
```

思路：动态规划。穷举法取得第一位的最小丑数。每一个质数在使用之后，则与其相乘的最小因子是使用时丑数的下一个丑数。比如 7 和 1 乘后，则 7 下次使用时，必然需要与 2 相乘（2 为 1 的下一个丑数），对于每一个质数都保存一个这样的指针（pointer）。每次都遍历所有质数与其相应位丑数的积并取最小为当前丑数，并注意移动其指针。特殊情况，当有两个以上为最小值时，都需要移动指针。详解

314. Binary Tree Vertical Order Traversal Medium

Given a binary tree, return the *vertical order* traversal of its nodes' values. (ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from **left to right**.

Examples:

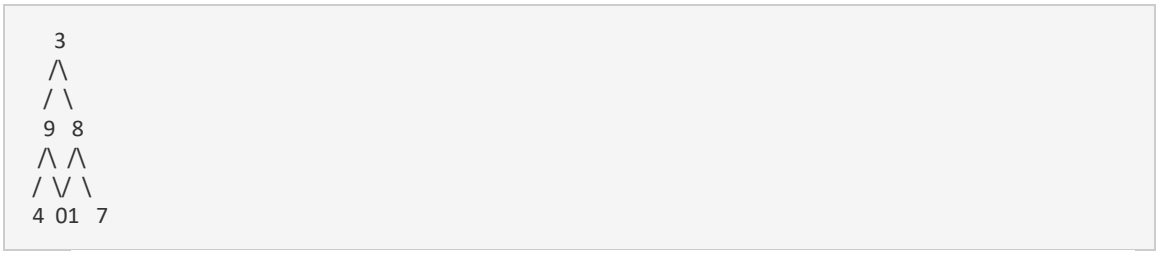
Given binary tree [3,9,20,null,null,15,7],

```
   3
  /\
 / \
 9  20
   /\
  / \
  15  7
```

return its vertical order traversal as:

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

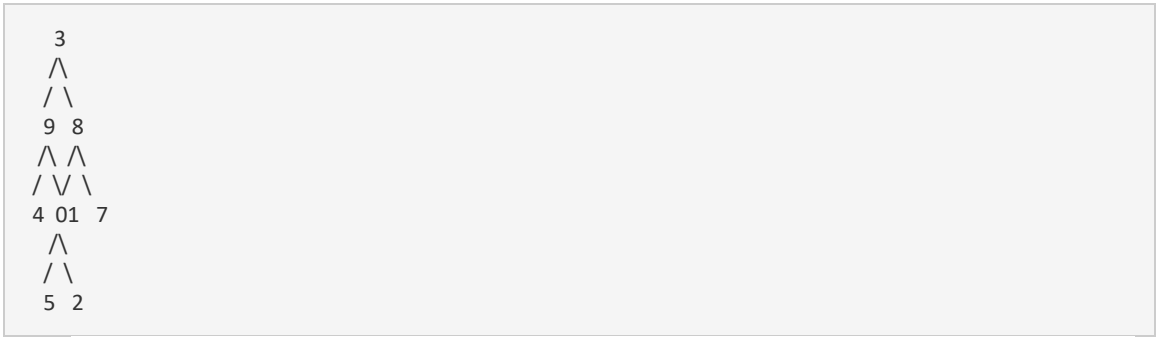
Given binary tree [3,9,8,4,0,1,7],



return its vertical order traversal as:

```
[ [4],
  [9],
  [3,0,1],
  [8],
  [7]]
```

Given binary tree [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5),



return its vertical order traversal as:

```
[ [4],
  [9,5],
  [3,0,1],
  [8,2],
  [7]]
```

```
public List<List<Integer>> verticalOrder(TreeNode root) {
    List<List<Integer>> cols = new ArrayList<>();
    if (root == null)
        return cols;
    int[] range = new int[] { 0, 0 };
    getRange(root, range, 0);
    for (int i = range[0]; i <= range[1]; i++)
        cols.add(new ArrayList<Integer>());
    Queue<TreeNode> queue = new LinkedList<>();
    Queue<Integer> colQueue = new LinkedList<>();
    queue.add(root);
```

```
    colQueue.add(-range[0]);
    while (!queue.isEmpty()) {
        TreeNode node = queue.poll();
        int col = colQueue.poll();
        cols.get(col).add(node.val);
        if (node.left != null) {
            queue.add(node.left);
            colQueue.add(col - 1);
        }
        if (node.right != null) {
            queue.add(node.right);
            colQueue.add(col + 1);
        }
    }
    return cols;
}

public void getRange(TreeNode root, int[] range, int col) {
    if (root == null) {
        return;
    }
    range[0] = Math.min(range[0], col);
    range[1] = Math.max(range[1], col);
    getRange(root.left, range, col - 1);
    getRange(root.right, range, col + 1);
}
```

思路：假设根为 0，则左/右结点为-1/1，深度遍历找出最左和最右的范围，建立相应的 List（此时最左边索引为 0，依次右推，根为-range[0]，最右则为-rang[0]+rang[1]）。再广度优先遍历（自上而下），把每个结点放入对应的 List。

315. Count of Smaller Numbers After Self Hard

You are given an integer array *nums* and you have to return a new *counts* array. The *counts* array has the property where counts[i] is the number of smaller elements to the right of nums[i].

Example:

> Given *nums* = [5, 2, 6, 1]
> To the right of 5 there are **2** smaller elements (2 and 1).
> To the right of 2 there is only **1** smaller element (1).
> To the right of 6 there is **1** smaller element (1).
> To the right of 1 there is **0** smaller element.

Return the array [2, 1, 1, 0].

```
class Node {
    Node left, right;
    int val, sum, dup = 1;
    public Node(int v, int s) {
        val = v;
        sum = s;
    }
}

public List<Integer> countSmaller(int[] nums) {
    Integer[] ans = new Integer[nums.length];
    Node root = null;
    for (int i = nums.length - 1; i >= 0; i--) {
        root = insert(nums[i], root, ans, i, 0);
    }
    return Arrays.asList(ans);
}
```

```
private Node insert(int num, Node node, Integer[] ans, int i, int preSum) {
    if (node == null) {
        node = new Node(num, 0);
        ans[i] = preSum;
    } else if (node.val == num) {
        node.dup++;
        ans[i] = preSum + node.sum;
    } else if (node.val > num) {
        node.sum++;
        node.left = insert(num, node.left, ans, i, preSum);
    } else {
        node.right = insert(num, node.right, ans, i, preSum + node.dup + node.sum);
    }
    return node;
}
```

思路：BFS。从后往前处理，生成二叉搜索树。比当前结点小的数都在左子树，个数为已经处理过的比当前数小的个数，所以可以根据当前结点左面个数进行判断。把结点从后往前依次插入树中，小于该树时往左走时++当前结点 sum。注意重复情况（小于该数的数+重复数=比该数小的数的个数）。

```
public List<Integer> countSmaller(int[] nums) {
    List<Integer> res = new LinkedList<>();
    if (nums == null || nums.length == 0) {
        return res;
    }
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for (int ele : nums) {
        if (min > ele) {
            min = ele;
        }
    }
    for (int i = 0; i < nums.length; i++) {
        nums[i] = nums[i] - min;
        if (max < nums[i]) {
            max = nums[i];
        }
    }
    int[] bit = new int[max + 1];
    int[] ans = new int[nums.length];
    for (int i = nums.length - 1; i >= 0; i--) {
        ans[i] = search(bit, nums[i]);
        insert(bit, nums[i]);
    }
    for (int i : ans) {
        res.add(i);
    }
    return res;
}
public int search(int[] bit, int index) {
    int res = 0;
    while (index > 0) {
        res += bit[index];
        index -= index & (-index);
    }
    return res;
}
public void insert(int[] bit, int index) {
    index++;
    while (index < bit.length) {
        bit[index]++;
        index += index & (-index);
    }
}
```

思路：基本思路和上一方法一样，从后往前依次处理。不同的是不用二叉搜索树进行排列和计数，而是直接计数。判断数大小的方法：先把数组简化，因为只需要处理大小关系，所以先找出最小数，作为 0。然后每一个数-最小数作为该数的新值。利用 Bit 存储数字出现的次数，ans 存储该数后小于该数的个数。两个数组的大小为最大值-最小值+1。每次把一个数处理时，相应数个数+1，然后从最小位开始逐一次加上所有有效位（单位置上对应数值）所形成的数（只要还在范围内），每个数对应位上个数都+1。每加入一个数时，先看该数有没有出现过，出现过多少次。然后去掉最小一位有效数字，再次计算该数出现次数。再去掉最小一位有效数字，计数，直到数字为 0。这时所有计数的和就是已经出现过的比这个数小的数的个数。可以证明，每个较小的数通过+最小有效位对应数（多个数），只有一个会与大于该数的任一数逐一减于最小有效位所对应的数（多个数）重合，也就是计算小于一个数个数的基础。

316. Remove Duplicate Letters Hard

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example:

Given "bcabc"
Return "abc"
Given "cbacdcbc"
Return "acdb"

```
public String removeDuplicateLetters(String s) {
    final int N = 26;
    int[] count = new int[N];
    boolean[] inQueue = new boolean[N];
    Deque<Integer> dq = new LinkedList<>();
    for (int i = 0; i < s.length(); i++) {
        count[s.charAt(i) - 'a']++;
    }
    for (int i = 0; i < s.length(); i++) {
        int idx = (int) (s.charAt(i) - 'a');
        count[idx]--;
        if (inQueue[idx]) {
            continue;
        }
        while (!dq.isEmpty() && dq.peekLast() > idx && count[dq.peekLast()] > 0) {// 利用 Count 信息把较大的字母尽量往后排，如果数量仍然大于 0，则说明后面还会出现，所以可以现在不必排入队列。
            inQueue[dq.pollLast()] = false;
        }
        inQueue[idx] = true;
        dq.offerLast(idx);
    }
    StringBuilder sb = new StringBuilder();
    while (!dq.isEmpty()) {
        sb.append((char) (dq.pollFirst() + 'a'));
    }
    return sb.toString();
}
```

思路：因为只能是小写字母（共 26 个），先统计每个字母出现次数，然后按出现次序入队列，期间每到一字母则其频次-1。每次加入新字母时与前面字母依次比较，直到找到位置。比较时如果碰到前面字母次序在后且出现频次仍大于 0（后面还会出现），则先让前面字母出队，即在后面再排前面的字母。这样可以保证尽量按字母序排序。

317. Shortest Distance from All Buildings Hard

You want to build a house on an *empty* land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values **0**, **1** or **2**, where:

- Each **0** marks an empty land which you can pass by freely.

- Each **1** marks a building which you cannot pass through.
- Each **2** marks an obstacle which you cannot pass through.

For example, given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2):

```
1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

The point (1,2) is an ideal empty land to build a house, as the total travel distance of 3+3+1=7 is minimal. So return 7.

Note:

There will be at least one building. If it is not possible to build such house according to the above rules, return -1.

```
public int shortestDistance(int[][] grid) {
    if (grid == null || grid.length == 0 || grid[0].length == 0)
        return -1;
    int m = grid.length;
    int n = grid[0].length;
    int[][] visitedCount = new int[m][n];
    int[] lr = new int[] { 1, -1, 0, 0 };
    int[] ud = new int[] { 0, 0, 1, -1 };
    int buildingCount = 0;
    Queue<int[]> q = new LinkedList<>();
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (grid[i][j] == 1) {
                ++buildingCount;
                q.add(new int[] { i, j });
                bfs(grid, q, m, n, lr, ud, buildingCount, visitedCount);
            }
        }
    }
    int ans = Integer.MIN_VALUE;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (grid[i][j] > 0 || visitedCount[i][j] != buildingCount)
                continue;
            ans = Math.max(grid[i][j], ans);
        }
    }
    return ans == Integer.MIN_VALUE ? -1 : -ans;
}

void bfs(int[][] grid, Queue<int[]> q, int m, int n, int[] lr, int[] ud, int buildingCount, int[][
] visitedCount) {
    int distance = -1;
    Queue<int[]> newQ = new LinkedList<>();
    while (!q.isEmpty()) {
        int[] pos = q.poll();
        for (int k = 0; k < 4; ++k) {
            int i = pos[0] + ud[k];
            int j = pos[1] + lr[k];
            if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] > 0 || visitedCount[i][j] != buil
dingCount - 1)
                continue;
            grid[i][j] += distance;
            ++visitedCount[i][j];
            newQ.add(new int[] { i, j });
        }
        if (q.isEmpty()) {
            --distance;
```

```
            q = newQ;
            newQ = new LinkedList<>();
        }
    }
}
```

思路：从每一个建筑物（Building）出发广度优先为每一点计算到各个建筑物距离的和（极值），然后取最小的那个。技巧：因为建筑物都是正数，空地都是 0，所以可以把距离用负数表示。统计过程中，只统计方部过次数比当前建筑物序号（总建筑物数）少 1 的点，因为别的点没有意义（少 2 或以上的代表有的建筑物不能到达，相等的代表已经访问过了）。

318. Maximum Product of Word Lengths Medium

Given a string array `words`, find the maximum value of `length(word[i]) * length(word[j])` where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1:

Given `["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]`

Return `16`

The two words can be `"abcw", "xtfn"`.

Example 2:

Given `["a", "ab", "abc", "d", "cd", "bcd", "abcd"]`

Return `4`

The two words can be `"ab", "cd"`.

Example 3:

Given `["a", "aa", "aaa", "aaaa"]`

Return `0`

No such pair of words.

```
public static int maxProduct(String[] words) {
    if (words == null || words.length == 0)
        return 0;
    int len = words.length;
    int[] value = new int[len];
    int maxProduct = 0;
    for (int i = 0; i < len; i++) {
        String tmp = words[i];
        value[i] = 0;
        for (int j = 0; j < tmp.length(); j++) {
            value[i] |= 1 << (tmp.charAt(j) - 'a');
        }
        for (int j = 0; j < i; j++) {
            if ((value[i] & value[j]) == 0 && (words[i].length() * words[j].length() >
maxProduct))
                maxProduct = words[i].length() * words[j].length();
        }
    }
    return maxProduct;
}
```

思路：因为只可能有 26 个字母，所以利用 int 存储一个词中各字母出现情况，然后通过按位与看有没有重复，如果没有，则计算极值，极值中取最大。

319. Bulb Switcher Medium

There are *n* bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the *i*th round, you toggle every *i* bulb. For the *n*th round, you only toggle the last bulb. Find how many bulbs are on after *n* rounds.

Example:

Given *n* = 3.
At first, the three bulbs are **[off, off, off]**.
After first round, the three bulbs are **[on, on, on]**.
After second round, the three bulbs are **[on, off, on]**.
After third round, the three bulbs are **[on, off, off]**.
So you should return 1, because there is only one bulb is on.

```
public int bulbSwitch(int n) {
    return (int) Math.sqrt(n);
}
```

思路：相当于求平方根。因为只有一个数是某数的平方时，才可能被切换偶数次。详解

320. Generalized Abbreviation Medium

Write a function to generate the generalized abbreviations of a word.
Example:
Given word = "word", return the following list (order does not matter):

```
["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]
```

```
public List<String> generateAbbreviations(String word) {
    List<String> ans = new ArrayList<>();
    backtrack(ans, new StringBuilder(), word.toCharArray(), 0, 0);
    return ans;
}

private static void backtrack(List<String> ans, StringBuilder sb, char[] word, int i, int k) {
    if (i == word.length) {
        if (k != 0)
            sb.append(k);
        ans.add(sb.toString());
    } else {
        backtrack(ans, new StringBuilder(sb), word, i + 1, k + 1);
        if (k != 0)
            sb.append(k);
        sb.append(word[i]);
        backtrack(ans, sb, word, i + 1, 0);
    }
}
```

思路：回溯法。逐一字符扫瞄，每到一字母有两个分支：缩写该字母和不缩写该字母。当扫瞄到最后一位的后面一位时，结束，把结果存起来。

```
public List<String> generateAbbreviations(String word) {
    if (word == null || word.length() == 0)
        return Arrays.asList("");

    int n = word.length();
    char[] workspace = new char[n];
    List<String> result = new ArrayList<>();

    dfs(word, 0, n, 0, workspace, result);

    return result;
}

private void dfs(String word, int offset, int n, int wpOffset, char[] workspace, List<String> resu
lt) {
    if (offset == n) {
        result.add(new String(workspace, 0, wpOffset));
        return;
    }
```

```
    workspace[wpOffset] = word.charAt(offset);
    dfs(word, offset + 1, n, wpOffset + 1, workspace, result);

    for (int i = offset; i < n; i++) {
        int length = i - offset + 1;
        int abbLength = 1;
        if (length < 10) {
            workspace[wpOffset] = (char) (length + '0');
        } else {
            String abbString = String.valueOf(length);
            abbLength = abbString.length();
            for (int j = 0; j < abbLength; j++)
                workspace[wpOffset + j] = abbString.charAt(j);
        }
        if (i + 1 < n) {
            workspace[wpOffset + abbLength] = word.charAt(offset + length);
            dfs(word, offset + length + 1, n, wpOffset + abbLength + 1, workspace, result);
        } else
            dfs(word, offset + length, n, wpOffset + abbLength, workspace, result);
    }
}
```

思路：DFS。思路与上一方法类似，深度优先每到一级遍历该级位置之后的紧邻的所有字母缩写情况。第一种情况都有两个分支：使用当前字符和不使用当前字符。过程中使用 workspace 数组缓存结果，使用 wpOffset 记录结果字符串长度。

321. Create Maximum Number Hard

Given two arrays of length `m` and `n` with digits `0-9` representing two numbers. Create the maximum number of length `k <= m + n`from digits of the two. The relative order of the digits from the same array must be preserved. Return an array of the `k` digits. You should try to optimize your time and space complexity.

Example 1:

nums1 = `[3, 4, 6, 5]`
nums2 = `[9, 1, 2, 5, 8, 3]`
k = `5`
return `[9, 8, 6, 5, 3]`

Example 2:

nums1 = `[6, 7]`
nums2 = `[6, 0, 4]`
k = `5`
return `[6, 7, 6, 0, 4]`

Example 3:

nums1 = `[3, 9]`
nums2 = `[8, 9]`
k = `3`
return `[9, 8, 9]`

```
public int[] maxNumber(int[] nums1, int[] nums2, int k) {
    int n = nums1.length;
    int m = nums2.length;
    int[] ans = new int[k];
    for (int i = Math.max(0, k - m); i <= k && i <= n; ++i) {
        int[] candidate = merge(maxArray(nums1, i), maxArray(nums2, k - i), k);
        if (greater(candidate, 0, ans, 0))
            ans = candidate;
    }
    return ans;
}

private int[] merge(int[] nums1, int[] nums2, int k) {
```

```
    int[] ans = new int[k];
    for (int i = 0, j = 0, r = 0; r < k; ++r)
        ans[r] = greater(nums1, i, nums2, j) ? nums1[i++] : nums2[j++];
    return ans;
}

public boolean greater(int[] nums1, int i, int[] nums2, int j) {
    while (i < nums1.length && j < nums2.length && nums1[i] == nums2[j]) {
        i++;
        j++;
    }
    return j == nums2.length || (i < nums1.length && nums1[i] > nums2[j]);
}

public int[] maxArray(int[] nums, int k) {
    int n = nums.length;
    int[] ans = new int[k];
    for (int i = 0, j = 0; i < n; ++i) {
        while (n - i + j > k && j > 0 && ans[j - 1] < nums[i])
            j--;
        if (j < k)
            ans[j++] = nums[i];
    }
    return ans;
}
```

思路：在一个数组中取 i 个最大数，另一个中取 k-i 个，两者归并后即为 k，取所有可能的 i 的情况。详解

```
public int[] maxNumber(int[] nums1, int[] nums2, int k) {
    SegmentTreeMax st1 = new SegmentTreeMax(nums1), st2 = new SegmentTreeMax(nums2);
    int[] maxN = new int[k];
    int cur1 = 0, cur2 = 0, remained = nums1.length + nums2.length - k;
    for (int i = 0; i != k; ++i) {
        int to1 = remained - cur2 + i, to2 = remained - cur1 + i;
        int picked1 = st1.reduceClosedGuarded(cur1, to1);
        int picked2 = st2.reduceClosedGuarded(cur2, to2);
        int comparePicked = compareIdx(nums1, picked1, nums2, picked2);
        if (comparePicked == 0) {
            int p1 = picked1, p2 = picked2;
            for (int j = 1; comparePicked == 0; ++j) {
                p1 = st1.reduceClosedGuarded(p1 + 1, to1 + j);
                p2 = st2.reduceClosedGuarded(p2 + 1, to2 + j);
                comparePicked = compareIdx(nums1, p1, nums2, p2);
            }
            if (p1 == -1 && p2 == -1) {
                comparePicked = 0;
                p1 = picked1;
                p2 = picked2;
                while (comparePicked == 0) {
                    p1 = st1.reduceClosedGuarded(cur1, p1 - 1);
                    p2 = st2.reduceClosedGuarded(cur2, p2 - 1);
                    comparePicked = -compareIdx(nums1, p1, nums2, p2);
                }
            }
        }
        if (comparePicked == 1) {
            maxN[i] = nums1[picked1];
            cur1 = picked1 + 1;
        } else {
            assert comparePicked == -1;
            maxN[i] = nums2[picked2];
            cur2 = picked2 + 1;
        }
    }
    return maxN;
```

```
}

private int compareIdx(int[] nums1, int i, int[] nums2, int j) {
    if (i == -1)
        return -1;
    if (j == -1)
        return 1;
    return Integer.compare(nums1[i], nums2[j]);
}
}

class SegmentTreeMax {
private final int[] data;
private final int[] origin;

public SegmentTreeMax(int[] origin) {
    data = new int[origin.length << 1];
    this.origin = origin;
    for (int i = origin.length; i != data.length; ++i)
        data[i] = i - origin.length;
    for (int i = origin.length - 1; i > 0; --i)
        data[i] = reducer4Build(data[i << 1], data[(i << 1) | 1]);
}

private int reducer4Build(int a, int b) {
    return (origin[a] < origin[b]) ? b : a;
}

private int reducer(int a, int b) {
    if (a > b) {
        a = a ^ b;
        b = a ^ b;
        a = a ^ b;
    }
    return (origin[a] < origin[b]) ? b : a;
}

public int reduceClosedGuarded(int from, int to) {
    to = Math.min(to, origin.length - 1);
    if (from > to)
        return -1;
    int maxima = from;
    for (int i = from + origin.length, j = to + origin.length; i <= j; i >>= 1, j >>= 1) {
        if ((i & 1) == 1)
            maxima = reducer(maxima, data[i++]);
        if ((j & 1) == 0)
            maxima = reducer(maxima, data[j--]);
    }
    return maxima;
}
```

思路：

322. Coin Change Medium

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Example 1:

coins = [1, 2, 5], amount = 11

return 3 (11 = 5 + 5 + 1)

Example 2:
coins = [2], amount = 3
return -1.
Note:
You may assume that you have an infinite number of each kind of coin.

```
public int coinChange(int[] coins, int amount) {
    int max = amount + 1;
    int[] dp = new int[max];
    Arrays.fill(dp, max);
    dp[0] = 0;
    for (int i = 1; i <= amount; ++i) {
        for (int coin : coins)
            if (i >= coin)
                dp[i] = Math.min(dp[i], dp[i - coin] + 1);
    }
    return dp[amount] > amount ? -1 : dp[amount];
}
```

思路：动态规划。使用和不使用每一种 Coin 对于每一种 Amount 的情况逐渐累加，直到达到 Amount。

```
public int min = Integer.MAX_VALUE;

public int coinChange(int[] coins, int amount) {
    if (amount == 0) {
        return 0;
    }

    if (coins.length == 0) {
        return -1;
    }
    Arrays.sort(coins);
    helper(coins, amount, coins.length - 1, 0);
    return min == Integer.MAX_VALUE ? -1 : min;
}

public void helper(int[] coins, int remain, int index, int count) {
    if (index < 0)
        return;
    for (int i = remain / coins[index]; i >= 0; i--) {

        int rem = remain - coins[index] * i;
        int curcount = count + i;

        if (rem > 0 && curcount + 1 < min)
            helper(coins, rem, index - 1, curcount);
        else {
            if (rem == 0 && curcount < min)
                min = curcount;
            return;
        }
    }
}
```

思路：贪心+回溯。先从最大硬币入手，尝试使用 n 到 0 个最大数，其中 n 为最多的可能数，然后把剩余的用同样的方法回溯到较小的一个硬币，如此直到可以得到目标钱数（即 rem=0）。

323. Number of Connected Components in an Undirected Graph Medium

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.
Example 1:

```
  0      3
```

```
  |       |
  1 --- 2   4
```

Given n = 5 and edges = [[0, 1], [1, 2], [3, 4]], return 2.

Example 2:

```
  0       4
  |       |
  1 --- 2 --- 3
```

Given n = 5 and edges = [[0, 1], [1, 2], [2, 3], [3, 4]], return 1.

Note:

You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

```
public int countComponents(int n, int[][] edges) {
    int[] roots = new int[n];
    for (int i = 0; i < n; i++)
        roots[i] = i;
    for (int[] e : edges) {
        int root1 = find(roots, e[0]);
        int root2 = find(roots, e[1]);
        if (root1 != root2) {
            roots[root1] = root2; // union
            n--;
        }
    }
    return n;
}

public int find(int[] roots, int id) {
    while (roots[id] != id) {
        roots[id] = roots[roots[id]]; // optional: path compression
        id = roots[id];
    }
    return id;
}
```

思路：并查集(Disjoint Set, Union Find)。每边两侧点未连通的，因为 2 点并为 1 点。已连通的无须管，因为已经合并。

324. Wiggle Sort II Medium

Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3]....

Example:

(1) Given nums = [1, 5, 1, 1, 6, 4], one possible answer is [1, 4, 1, 5, 1, 6].

(2) Given nums = [1, 3, 2, 2, 3, 1], one possible answer is [2, 3, 1, 3, 1, 2].

Note:

You may assume all input has valid answer.

Follow Up:

Can you do it in O(n) time and/or in-place with O(1) extra space?

```
public void wiggleSort(int[] nums) {
    int[] clone = nums.clone();
    Arrays.sort(clone);
    int index = fill(nums.length - 2 + nums.length % 2, nums, clone, 0);
    fill(nums.length - 1 - nums.length % 2, nums, clone, index);
}

private int fill(int start, int[] nums, int[] clone, int index) {
    for (int i = start; i >= 0; i -= 2)
        nums[i] = clone[index++];
```

```
    return index;
}
```
思路 1：隐含条件：每个数至多出现 2 次。先排序，然后从后到前分配，先分配基数列，分配完成后，剩下部分最小的数大于等于已分配最大的数，从后到前分配剩下的数到偶数列，则新数列一定满足条件。
思路 2(未实现)：先找到中位数，把数字分成三份（中位数、小于中位数的、大于中位数的），然后再根据需求重新安排。平均可达 O(n)，最差还是 O(nlogn)

325. Maximum Size Subarray Sum Equals k Medium

Given an array *nums* and a target value *k*, find the maximum length of a subarray that sums to *k*. If there isn't one, return 0 instead.

Note:

The sum of the entire *nums* array is guaranteed to fit within the 32-bit signed integer range.

Example 1:

Given *nums* = [1, -1, 5, -2, 3], *k* = 3,
return 4. (because the subarray [1, -1, 5, -2] sums to 3 and is the longest)

Example 2:

Given *nums* = [-2, -1, 2, 1], *k* = 1,
return 2. (because the subarray [-1, 2] sums to 1 and is the longest)

Follow Up:

Can you do it in O(*n*) time?

```
public int maxSubArrayLen(int[] nums, int k) {
int maxLen = 0, sum = 0;
Map<Integer, Integer> sumMap = new HashMap<>();
for (int i = 0; i < nums.length; ++i) {
    sum += nums[i];
    if (sum == k)
        maxLen = Math.max(maxLen, i + 1);
    else {
        int j = sumMap.getOrDefault(sum - k, -1);
        if (j != -1)
            maxLen = Math.max(maxLen, i - j);
    }
    if (!sumMap.containsKey(sum))
        sumMap.put(sum, i);
}
return maxLen;
}
```
思路：用 Map 保存从 0 到 i 的 sum 及 i（重复的不再保存，因为只需要最长的，复制的只在减法中当被减数用到，故保留最短情况。如果 Sum 是 k，则为 1 个候选结果；如果 Sum-k 已经在 Map 中，则从 Sum-k 结束到 i 为一个候选，取所有候选中的最大长度。O(n)

```
public int maxSubArrayLen(int[] nums, int k) {
    int[] sum = new int[nums.length + 1];
    Map<Integer, Integer> longest = new HashMap<>(nums.length + 1);
    longest.put(0, 0);
    for (int i = 0; i < nums.length; ++i) {
        sum[i + 1] = sum[i] + nums[i];
        longest.put(sum[i + 1], i + 1);
    }
    int len = 0;
    for (int i = 0; i < nums.length; ++i) {
        Integer l = longest.get(k + sum[i]);
        if (l != null)
            len = Math.max(len, l - i);
    }
    return len;
}
```
思路：与上一思路类似。分两步：第一步，把所有 Index 存入 Map，所有 Sum 存入 sum 数组。第二步，寻找所有可以构成 k 的 SUM 值，如找到则更新最值。比上一方法的优化在于不再检查是否已经存入数组，因为在

查检和值是否符合条件的时候，有重复的情况会分别计算。但是必须分两步，因为需要知道所有 sum 值时才能进行检测。

326. Power of Three Easy

Given an integer, write a function to determine if it is a power of three.

Follow up:

Could you do it without using any loop / recursion?

```
public boolean isPowerOfThree(int n) {
    if (n < 1)
        return false;
    while (n % 3 == 0)
        n /= 3;
    return n == 1;
}
```

思路 **1**：循环除求余判断。

```
public boolean isPowerOfThree(int n) {
    return n > 0 && Math.pow(3, (int) Math.floor(Math.log(Integer.MAX_VALUE) / Math.log(3))) % n =
= 0;
}
```

思路 **2**：如果为 3 的倍数，那么一定可以被最大的三的倍数除尽，而别的数无法被其除尽（因为是素数）。所以找到 3 的最大倍数（Integer 范围内），看有无余数。

327. Count of Range Sum Hard

Given an integer array nums, return the number of range sums that lie in [lower, upper] inclusive.

Range sum S(i, j) is defined as the sum of the elements in nums between indices i and j (i ? j), inclusive.

Note:

A naive algorithm of $O(n^2)$ is trivial. You MUST do better than that.

Example:

Given *nums* = [-2, 5, -1], *lower* = -2, *upper* = 2,

Return 3.

The three ranges are : [0, 0], [2, 2], [0, 2] and their respective sums are: -2, -1, 2.

```
public int countRangeSum(int[] nums, int lower, int upper) {
    int n = nums.length;
    long[] sums = new long[n + 1];
    for (int i = 0; i < n; ++i)
        sums[i + 1] = sums[i] + nums[i];
    return countWhileMergeSort(sums, 0, n + 1, lower, upper);
}

private int countWhileMergeSort(long[] sums, int start, int end, int lower, int upper) {
    if (end - start <= 1)
        return 0;
    int mid = (start + end) / 2;
    int count = countWhileMergeSort(sums, start, mid, lower, upper)
            + countWhileMergeSort(sums, mid, end, lower, upper);
    int j = mid, k = mid, t = mid;
    long[] cache = new long[end - start];
    for (int i = start, r = 0; i < mid; ++i, ++r) {
        while (k < end && sums[k] - sums[i] < lower)
            k++;
        while (j < end && sums[j] - sums[i] <= upper)
            j++;
        while (t < end && sums[t] < sums[i])
            cache[r++] = sums[t++];
        cache[r] = sums[i];
        count += j - k;
    }
```

```
    System.arraycopy(cache, 0, sums, start, t - start);
    return count;
}
```
思路：预处理后得到和的数组，然后问题就和 315 题类似，在归并排序过程中数个数。根据归并排序特点（分治策略），当合并时，mid 左右的子数组已经排序，所以只要找两子数组各取一个数符合条件的情况，这样所有的 sum 差都考虑且只考虑了一次。使用开区间处理尾数，所以回归条件为相差不大于 1，好处是对右侧数组不需要进行 mid+1 操作。

328. Odd Even Linked List Medium

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.
You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.
Example:
Given `1->2->3->4->5->NULL`,
return `1->3->5->2->4->NULL`.
Note:
The relative order inside both the even and odd groups should remain as it was in the input.
The first node is considered odd, the second node even and so on ...
```
public ListNode oddEvenList(ListNode head) {
    if (head == null)
        return null;
    ListNode odd = head, even = head.next, evenHead = even;
    while (even != null) {
        odd.next = even.next;
        if (odd.next == null)
            break;
        odd = odd.next;
        even.next = odd.next;
        even = even.next;
    }
    odd.next = evenHead;
    return head;
}
```
思路：奇偶各自放入一个数组后连接。

329. Longest Increasing Path in a Matrix Hard

Given an integer matrix, find the length of the longest increasing path.
From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).
Example 1:
```
nums = [
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
```
Return `4`
The longest increasing path is `[1, 2, 6, 9]`.
Example 2:
```
nums = [
  [3,4,5],
  [3,2,6],
  [2,2,1]
```

```
]
```

Return 4
The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

```
private static final int[][] dirs = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };
private int m, n;

public int longestIncreasingPath(int[][] matrix) {
    if (matrix.length == 0)
        return 0;
    m = matrix.length;
    n = matrix[0].length;
    int[][] cache = new int[m][n];
    int ans = 0;
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j)
            ans = Math.max(ans, dfs(matrix, i, j, cache));
    return ans;
}

private int dfs(int[][] matrix, int i, int j, int[][] cache) {
    if (cache[i][j] != 0)
        return cache[i][j];
    for (int[] d : dirs) {
        int x = i + d[0], y = j + d[1];
        if (0 <= x && x < m && 0 <= y && y < n && matrix[x][y] > matrix[i][j])
            cache[i][j] = Math.max(cache[i][j], dfs(matrix, x, y, cache));
    }
    return ++cache[i][j];
}
```

思路：DFS+缓存。通过深度查找找每一点的最大增长路径，已经查找过的路径可以缓存。小的性能优化：先判断一个方向是否可移动，再移动。这里选择更好的代码重用，效率略低。

330. Patching Array Hard

Given a sorted positive integer array *nums* and an integer *n*, add/patch elements to the array such that any number in range [1, n]inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

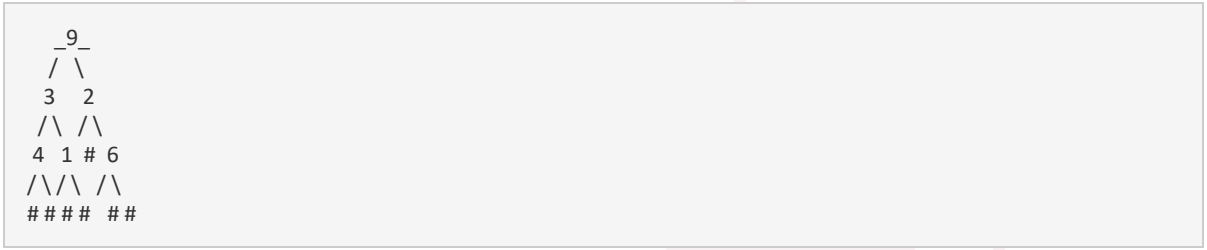
Example 1:
nums = [1, 3], *n* = 6
Return 1.
Combinations of *nums* are [1], [3], [1,3], which form possible sums of: 1, 3, 4.
Now if we add/patch 2 to *nums*, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3].
Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6].
So we only need 1 patch.
Example 2:
nums = [1, 5, 10], *n* = 20
Return 2.
The two patches can be [2, 4].
Example 3:
nums = [1, 2, 2], *n* = 5
Return 0.

```
public int minPatches(int[] nums, int n) {
    int patches = 0, i = 0;
    long miss = 1;
    while (miss <= n) {
        if (i < nums.length && nums[i] <= miss)
            miss += nums[i++];
        else {
```

```
            miss += miss;
            patches++;
        }
    }
    return patches;
}
```
思路：贪心法。假设从 1 开始 miss，逐一数对比，碰到比 miss 大的数，说明 miss 数的确 miss，这时需要补 miss（较小数），补完后，可得和范围由[1,miss)扩展到了[1,miss+miss)，所以下一个 miss 值为 miss+miss。如果遇到数小于 miss，则说明该数已经被覆盖了，则说明 miss+该数范围内都被覆盖了。

331. Verify Preorder Serialization of a Binary Tree Medium

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as #.



For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as "1,,3".

Example 1:

"9,3,4,#,#,1,#,#,2,#,6,#,#"

Return true

Example 2:

"1,#"

Return false

Example 3:

"9,#,#,1"

Return false

```
public boolean isValidSerialization(String preorder) {
    String[] nodes = preorder.split(",");
    int diff = 1;
    for (String node : nodes) {
        if (--diff < 0)
            return false;
        if (!node.equals("#"))
            diff += 2;
    }
    return diff == 0;
}
```
思路：利用结点和叶子外#数量关系进行验证。

332. Reconstruct Itinerary Medium

Given a list of airline tickets represented by pairs of departure and arrival airports [from, to], reconstruct the itinerary in order. All of the tickets belong to a man who departs from JFK. Thus, the itinerary must begin with JFK.

Note:

1. If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string. For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].
2. All airports are represented by three capital letters (IATA code).
3. You may assume all tickets form at least one valid itinerary.

Example 1:
tickets = [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]
Return ["JFK", "MUC", "LHR", "SFO", "SJC"].
Example 2:
tickets = [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]
Return ["JFK","ATL","JFK","SFO","ATL","SFO"].
Another possible reconstruction is ["JFK","SFO","ATL","JFK","ATL","SFO"]. But it is larger in lexical order.

```
Map<String, PriorityQueue<String>> map;
LinkedList<String> path;

public List<String> findItinerary(List<List<String>> tickets) {
    map = new HashMap<>();
    path = new LinkedList<>();
    for (List<String> str : tickets) {
        map.putIfAbsent(str.get(0), new PriorityQueue<>());
        map.get(str.get(0)).add(str.get(1));
    }
    dfs("JFK");
    return path;
}

public void dfs(String dep) {
    PriorityQueue<String> hp = map.get(dep);
    while (hp != null && !hp.isEmpty()) {
        dfs(hp.poll());
    }
    path.addFirst(dep);
}
```

思路：假设没有重复，非常容易，只要建立词典，从 JFK 开始逐一机票使用。现在是有可能有重复，不过题中还给出了另一个条件：如果有重复，则按到站的字典顺排序。所以这里只有到站是需要排序的。排序问题可以想到优先队列。有了上述的准备工作，则使用 DFS 找结果，最后把结果存入数组。Path.addFirst 是因为总是靠近队尾的元素更先到该句（因为是 DFS）。如果有答案，可以证明：重复到站先走哪个都应该可以完成。

333. Largest BST Subtree Medium

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.
Note:
A subtree must include all of its descendants.
Here's an example:

```
  10
  /\
 5 15
/\ \
1 8 7
```

The Largest BST Subtree in this case is the highlighted one.
The return value is the subtree's size, which is 3.
Follow up:
Can you figure out ways to solve it with O(n) time complexity?

```
public int largestBSTSubtree(TreeNode root) {
    if (root == null)
```

```
        return 0;
    if (isBST(root, null, null))
        return countNode(root);
    return Math.max(largestBSTSubtree(root.left), largestBSTSubtree(root.right));
}

private static boolean isBST(TreeNode root, Integer i, Integer j) {
    if (root == null)
        return true;
    if (i != null && root.val <= i)
        return false;
    if (j != null && root.val >= j)
        return false;
    return isBST(root.left, i, root.val) && isBST(root.right, root.val, j);
}

private static int countNode(TreeNode root) {
    if (root == null)
        return 0;
    return 1 + countNode(root.left) + countNode(root.right);
}
```

思路 **1**：DFS 自顶而下逐一验证是否二叉搜索树，如果是，同级别最大的数一定是答案。

```
class Signal {
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    int num = 0;
    boolean isBST = true;
}

public int largestBSTSubtree(TreeNode root) {
    return largestBST(root).num;
}

private Signal largestBST(TreeNode root) {
    Signal signal = new Signal();
    if (root == null)
        return signal;
    if (root.left == null && root.right == null) {
        signal.max = signal.min = root.val;
        signal.num = 1;
        return signal;
    }
    Signal left = largestBST(root.left);
    Signal right = largestBST(root.right);
    if (left.isBST && right.isBST && root.val > left.max && root.val < right.min) {
        signal.num = left.num + right.num + 1;
        signal.max = Math.max(right.max, root.val);
        signal.min = Math.min(left.min, root.val);
    } else {
        signal.num = Math.max(left.num, right.num);
        signal.isBST = false;
    }
    return signal;
}
```

思路 **2**（O(n)）：备忘录，每次计算完结果都缓存。每一个结点共记录 4 个信息，该结点为根是否为 BST，该结点之下最大的 BST 的数值，该结点以下（如果是 BST）的最值（用于上层判断是否 BST）。

334. Increasing Triplet Subsequence Medium

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.
Formally the function should:

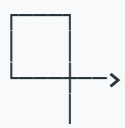
Return true if there exists *i, j, k*
such that *arr[i] < arr[j] < arr[k]* given 0 ≤ *i* < *j* < *k* ≤ *n*-1 else return false.
Your algorithm should run in O(*n*) time complexity and O(*1*) space complexity.
Examples:
Given [1, 2, 3, 4, 5],
return true.
Given [5, 4, 3, 2, 1],
return false.

```
public boolean increasingTriplet(int[] nums) {
    int small = Integer.MAX_VALUE, big = Integer.MAX_VALUE;
    for (int n : nums) {
        if (n <= small) {
            small = n;
        } else if (n <= big) {
            big = n;
        } else
            return true;
    }
    return false;
}
```

思路：顺序查看数组中的数字，不断更新第一小和第二小的数，如果出现第三小的数则为真。

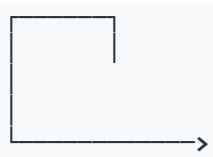
335. Self Crossing Hard

You are given an array *x* of n positive numbers. You start at point (0,0) and moves x[0] metres to the north, then x[1] metres to the west, x[2] metres to the south, x[3] metres to the east and so on. In other words, after each move your direction changes counter-clockwise.
Write a one-pass algorithm with O(1) extra space to determine, if your path crosses itself, or not.
Example 1:



```
Input: [2,1,1,2]
Output: true
```

Example 2:



```
Input: [1,2,3,4]
Output: false
```

Example 3:

```
Input: [1,1,1,1]
Output: true
```

```
public boolean isSelfCrossing(int[] x) {
    for (int i = 3, l = x.length; i < l; i++) {
        if (x[i] >= x[i - 2] && x[i - 1] <= x[i - 3])
            return true; // Case 1: current line crosses the line 3 steps ahead of it
        else if (i >= 4 && x[i - 1] == x[i - 3] && x[i] + x[i - 4] >= x[i - 2])
            return true; // Case 2: current line crosses the line 4 steps ahead of it
```

```
        else if (i >= 5 && x[i - 2] >= x[i - 4] && x[i] + x[i - 4] >= x[i - 2] && x[i - 1] <= x[i
- 3] && x[i - 1] + x[i - 5] >= x[i - 3])
            return true; // Case 3: current line crosses the line 6 steps ahead of it
    }
    return false;
}
```

思路：第四步及以后才有可能重合，而且重合只有三种可能：第四条穿过第一条；第五条与第一条重合（同一直线上且相连）；第六条穿过第一条（第六条还有可能空过第 6-3=3 条，即第一种情况）。分析中，执笔画一画可能穿越的情况很容易找出答案。

336. Palindrome Pairs Hard

Given a list of **unique** words, find all pairs of ***distinct*** indices (i, j) in the given list, so that the concatenation of the two words, i.e. words[i] + words[j] is a palindrome.

Example 1:

Given words = ["bat", "tab", "cat"]

Return [[0, 1], [1, 0]]

The palindromes are ["battab", "tabbat"]

Example 2:

Given words = ["abcd", "dcba", "lls", "s", "sssll"]

Return [[0, 1], [1, 0], [3, 2], [2, 4]]

The palindromes are ["dcbaabcd", "abcddcba", "slls", "llssssll"]

```
private List<String> allValidPrefixes(String word) {
    List<String> validPrefixes = new ArrayList<>();
    for (int i = 0; i < word.length(); i++)
        if (isPalindromeBetween(word, i, word.length() - 1))
            validPrefixes.add(word.substring(0, i));
    return validPrefixes;
}

private List<String> allValidSuffixes(String word) {
    List<String> validSuffixes = new ArrayList<>();
    for (int i = 0; i < word.length(); i++)
        if (isPalindromeBetween(word, 0, i))
            validSuffixes.add(word.substring(i + 1, word.length()));
    return validSuffixes;
}

private boolean isPalindromeBetween(String word, int front, int back) {
    while (front < back) {
        if (word.charAt(front) != word.charAt(back))
            return false;
        front++;
        back--;
    }
    return true;
}

public List<List<Integer>> palindromePairs(String[] words) {
    Map<String, Integer> wordSet = new HashMap<>();
    for (int i = 0; i < words.length; i++)
        wordSet.put(words[i], i);
    List<List<Integer>> solution = new ArrayList<>();
    for (String word : wordSet.keySet()) {
        int currentWordIndex = wordSet.get(word);
        String reversedWord = new StringBuilder(word).reverse().toString();
        if (wordSet.containsKey(reversedWord) && wordSet.get(reversedWord) != currentWordIndex)
            solution.add(Arrays.asList(currentWordIndex, wordSet.get(reversedWord)));
        for (String suffix : allValidSuffixes(word)) {
            String reversedSuffix = new StringBuilder(suffix).reverse().toString();
```

```
            if (wordSet.containsKey(reversedSuffix))
                solution.add(Arrays.asList(wordSet.get(reversedSuffix), currentWordIndex));
        }
        for (String prefix : allValidPrefixes(word)) {
            String reversedPrefix = new StringBuilder(prefix).reverse().toString();
            if (wordSet.containsKey(reversedPrefix))
                solution.add(Arrays.asList(currentWordIndex, wordSet.get(reversedPrefix)));
        }
    }
    return solution;
}
```

思路 1（略）：暴力算法。把任意两个 index 可能的组合拿出来（String），判断是不是回文结构。
思路 2：字典法。要形成回文结构，共有两种可能：两单词等长；两单词不等长。等长情况很容易判断，即反过来一样。不等长情况，容易发展，较短的反过来，正好是较长的尾部；而较长的除尾部外的部分，本身一定是回文结构。根据这些特别，容易建立索引（通过 HashMap 或者 Trie）。本解法中使用 HashMap，使用 Trie 的思路是一样的。不等长的头-尾互换共两种情况都要考虑进去。

337. House Robber III Medium

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.
Determine the maximum amount of money the thief can rob tonight without alerting the police.
Example 1:

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

Maximum amount of money the thief can rob = 3 + 3 + 1 = **7**.
Example 2:

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

Maximum amount of money the thief can rob = 4 + 5 = **9**.

```
public int rob(TreeNode root) {
    int[] ans = robT(root);
    return Math.max(ans[0], ans[1]);
}

private int[] robT(TreeNode root) {
    int[] ans = { 0, 0 };
    if (root == null)
        return ans;
    int[] left = robT(root.left);
    int[] right = robT(root.right);
    ans[1] = left[0] + right[0] + root.val;
    ans[0] = Math.max(left[0], left[1]) + Math.max(right[0], right[1]);
    return ans;
}
```

思路：动态规划。与之前的偷窥题无本质区别。递推方程：偷或不偷当前房子累加最大。偷当前层的则不能偷下一层；不偷当前层则可偷也可不偷下一层。

338. Counting Bits Medium

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example:

For num = 5 you should return [0,1,1,2,1,2].

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

```
public int[] countBits(int num) {
    int[] ans = new int[num + 1];
    for (int i = 1; i <= num; ++i)
        ans[i] = ans[i & (i - 1)] + 1;
    return ans;
}
```

思路：相邻两数总是+1 bit：如果上数末位是 0，则直接+1；如果上数末位是 1，则需要看是否进位。因为两者仅差 1 bit，所以此数所需要的 bit 的数量是上数左侧相同，右侧第一个零位置往后全部置 0 对应的数+1，该数为 i&(i-1)（假设当前数为 1）。

思路 2：当前数总是在后面或前面+1 bit 所得，递推公式：P(x+b)=P(x)+1,b=2^m>x 或 P(x)=P(x/2)+(x mod 2) 或

339. Nested List Weight Sum Easy

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Given the list [[1,1],2,[1,1]], return **10**. (four 1's at depth 2, one 2 at depth 1)

Example 2:

Given the list [1,[4,[6]]], return **27**. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4*2 + 6*3 = 27)

```
public interface NestedInteger {
  // Constructor initializes an empty nested list.
  public NestedInteger();

  // Constructor initializes a single integer.
  public NestedInteger(int value);

  // @return true if this NestedInteger holds a single integer, rather than a nested list.
  public boolean isInteger();

  // @return the single integer that this NestedInteger holds, if it holds a single integer
  // Return null if this NestedInteger holds a nested list
  public Integer getInteger();

  // Set this NestedInteger to hold a single integer.
  public void setInteger(int value);

  // Set this NestedInteger to hold a nested list and adds a nested integer to it.
  public void add(NestedInteger ni);
```

```
  // @return the nested list that this NestedInteger holds, if it holds a nested list
  // Return null if this NestedInteger holds a single integer
  public List<NestedInteger> getList();
}
public int depthSum(List<NestedInteger> nestedList) {
    return depthSum(nestedList, 1);
}

private static int depthSum(List<NestedInteger> nestedList, int depth) {
    int sum = 0;
    for (NestedInteger nestedInteger : nestedList)
        if (nestedInteger.isInteger())
            sum += nestedInteger.getInteger() * depth;
        else
            sum += depthSum(nestedInteger.getList(), depth + 1);
    return sum;
}
```

思路：根据深度加权重

340. Longest Substring with At Most K Distinct Characters Hard

Given a string, find the length of the longest substring T that contains at most *k* distinct characters.
For example, Given s = “eceba” and k = 2,
T is "ece" which its length is 3.

```
public int lengthOfLongestSubstringKDistinct(String s, int k) {
    int[] count = new int[256];
    char[] cs = s.toCharArray();
    int n = cs.length, ans = 0;
    for (int i = 0, j = 0, num = 0; i < n; ++i) {
        if (count[cs[i]]++ == 0)
            ++num;
        if (num > k) {
            while (--count[cs[j++]] > 0)
                ;
            --num;
        }
        ans = Math.max(i - j + 1, ans);
    }
    return ans;
}
```

思路：滑动窗口。不断记录当前窗口内数字个数并更新最大值，如果超出窗口限制，则左边界右移。此题中假设为 ASCII 集，如果不是，可使用 Map，这里的 Count 数组实际上是在模拟 Map 操作。

341. Flatten Nested List Iterator Medium

Given a nested list of integers, implement an iterator to flatten it.
Each element is either an integer, or a list -- whose elements may also be integers or other lists.
Example 1:
Given the list [[1,1],2,[1,1]],
By calling *next* repeatedly until *hasNext* returns false, the order of elements returned by *next* should be: [1,1,2,1,1].
Example 2:
Given the list [1,[4,[6]]],
By calling *next* repeatedly until *hasNext* returns false, the order of elements returned by *next* should be: [1,4,6].

```
public class NestedIterator implements Iterator<Integer> {
    Queue<Integer> flatList = new LinkedList<>();

    public NestedIterator(List<NestedInteger> nestedList) {
        append(nestedList);
    }
```

```
    private void append(List<NestedInteger> nestedList) {
        for (NestedInteger ni : nestedList)
            if (ni.isInteger())
                flatList.add(ni.getInteger());
            else
                append(ni.getList());
    }

    @Override
    public Integer next() {
        return flatList.poll();
    }

    @Override
    public boolean hasNext() {
        return flatList.size() > 0;
    }
}
```

思路 1：初始化时将结果存入一维数组。

```
public class NestedIterator implements Iterator<Integer> {
    List<NestedInteger> l = new LinkedList<>();

    public NestedIterator(List<NestedInteger> nestedList) {
        l = nestedList;
    }

    @Override
    public Integer next() {
        while (!l.isEmpty()) {
            NestedInteger cur = l.remove(0);
            if (cur.isInteger())
                return cur.getInteger();
            for (int i = cur.getList().size() - 1; i >= 0; --i) {
                NestedInteger v = cur.getList().get(i);
                l.add(0, v);
            }
        }
        throw new NullPointerException();
    }

    @Override
    public boolean hasNext() {
        while (!l.isEmpty()) {
            NestedInteger cur = l.get(0);
            if (cur.isInteger())
                return true;
            l.remove(0);
            for (int i = cur.getList().size() - 1; i >= 0; --i) {
                NestedInteger v = cur.getList().get(i);
                l.add(0, v);
            }
        }
        return false;
    }
}
```

思路 2：next 前把当前元素（如果是数组）重新按元素顺序放到缓存数组最前。

342. Power of Four Easy

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example:
Given num = 16, return true. Given num = 5, return false.
Follow up: Could you solve it without loops/recursion?

```
public boolean isPowerOfFour(int num) {
    return Integer.toString(num, 4).matches("10*");
}
```

思路：转化成 4 进制数，这样只要看是不是 100000……这样的形式就好了。

343. Integer Break Medium

Given a positive integer *n*, break it into the sum of **at least** two positive integers and maximize the product of those integers. Return the maximum product you can get.
For example, given *n* = 2, return 1 (2 = 1 + 1); given *n* = 10, return 36 (10 = 3 + 3 + 4).
Note: You may assume that *n* is not less than 2 and not larger than 58.

```
public int integerBreak(int n) {
    if (n == 2)
        return 1;
    if (n == 3)
        return 2;
    int num_3 = n / 3;
    int remainder = n % 3;
    if (remainder == 1) {
        remainder = 4;
        num_3--;
    } else if (remainder == 0)
        remainder = 1;
    return (int) Math.pow(3, num_3) * remainder;
}
```

思路：3 的倍数*3 的余数就是最大乘积数。详解，因为大于四的因子拆成小于四后的积更大而 3*3>2*2*2。

344. Reverse String Easy

Write a function that takes a string as input and returns the string reversed.
Example:
Given s = "hello", return "olleh".

```
public void reverseString(char[] s) {
    if (s == null)
        return;
    int i = 0, j = s.length - 1;
    while (i < j) {
        char t = s[i];
        s[i] = s[j];
        s[j] = t;
        i++;
        j--;
    }
}
```

思路：双指针法。从两头向中间一步步移动，每步都转换两头元素，直到两指针相遇。

345. Reverse Vowels of a String Easy

Write a function that takes a string as input and reverse only the vowels of a string.
Example 1:
Given s = "hello", return "holle".
Example 2:
Given s = "leetcode", return "leotcede".
Note:
The vowels does not include the letter "y".

```
public String reverseVowels(String s) {
```

```
    char[] chars = s.toCharArray();
    int i = 0, j = chars.length - 1;
    while (i < j) {
        while (i < j && !isVowel(chars[i]))
            ++i;
        while (j > i && !isVowel(chars[j]))
            --j;
        char temp = chars[i];
        chars[i++] = chars[j];
        chars[j--] = temp;
    }
    return new String(chars);
}

private boolean isVowel(char c) {
    c = Character.toLowerCase(c);
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}
```

思路：双指针法。思路与上一题一致，只不过需要跳过非元音字母。

346. Moving Average from Data Stream Easy

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.
For example,

```
MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3
```

```
class MovingAverage {
    int[] cache;
    int sum = 0, i = 0, n = 0, m = 0;

    public MovingAverage(int size) {
        cache = new int[size];
        n = size;
    }

    public double next(int val) {
        sum += val;
        i = ++i % n;
        sum -= cache[i];
        cache[i] = val;
        if (m < n)
            ++m;
        return (double) sum / m;
    }
}
```

思路：维护一个和值，每次出入元素时减或加该元素值。

347. Top K Frequent Elements Medium

Given a non-empty array of integers, return the ***k*** most frequent elements.
For example,
Given [1,1,1,2,2,3] and k = 2, return [1,2].
Note:

- You may assume *k* is always valid, 1 ≤ k ≤ number of unique elements.
- Your algorithm's time complexity **must be** better than O(*n* log *n*), where *n* is the array's size.

```java
public int[] topKFrequent(int[] nums, int k) {
    int n = nums.length;
    int[] ans = new int[k];
    List<Integer>[] freq = new List[n];
    Map<Integer, Integer> freqMap = new HashMap<>();
    for (int num : nums)
        freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
    for (int num : freqMap.keySet()) {
        int i = freqMap.get(num) - 1;
        if (freq[i] == null)
            freq[i] = new ArrayList<>();
        freq[i].add(num);
    }
    for (int i = n - 1, j = 0; i >= 0 && j < k; --i)
        if (freq[i] != null)
            for (int num : freq[i])
                ans[j++] = num;
    return ans;
}
```

思路：桶排序。因为最大词频不可能超过数组长度，所以所有的频率都会在这个范围内，利用这个数组统计每一个频次上出现的词的个数，取最大的 k 个（注意要把没有词频信息的排除）。

348. Design Tic-Tac-Toe Medium

Design a Tic-tac-toe game that is played between two players on a *n* x *n* grid.

You may assume the following rules:

1. A move is guaranteed to be valid and is placed on an empty block.
2. Once a winning condition is reached, no more moves is allowed.
3. A player who succeeds in placing *n* of their marks in a horizontal, vertical, or diagonal row wins the game.

Example:

```
Given n = 3, assume that player 1 is "X" and player 2 is "O" in the board.
TicTacToe toe = new TicTacToe(3);
toe.move(0, 0, 1); -> Returns 0 (no one wins)
|x| | |
| | | |   // Player 1 makes a move at (0, 0).
| | | |
toe.move(0, 2, 2); -> Returns 0 (no one wins)
|x| |o|
| | | |   // Player 2 makes a move at (0, 2).
| | | |
toe.move(2, 2, 1); -> Returns 0 (no one wins)
|X| |O|
| | | |   // Player 1 makes a move at (2, 2).
| | |X|
toe.move(1, 1, 2); -> Returns 0 (no one wins)
|X| |O|
| |O| |   // Player 2 makes a move at (1, 1).
| | |X|
toe.move(2, 0, 1); -> Returns 0 (no one wins)
|X| |O|
| |O| |   // Player 1 makes a move at (2, 0).
|X| |X|
toe.move(1, 0, 2); -> Returns 0 (no one wins)
|X| |O|
|O|O| |   // Player 2 makes a move at (1, 0).
|X| |X|
```

```
toe.move(2, 1, 1); -> Returns 1 (player 1 wins)
|X| |O|
|O|O| |    // Player 1 makes a move at (2, 1).
|X|X|X|
```

```
public class TicTacToe {
    private int[] rows;
    private int[] cols;
    private int diagonal;
    private int antiDiagonal;

    public TicTacToe(int n) {
        rows = new int[n];
        cols = new int[n];
    }

    public int move(int row, int col, int player) {
        int toAdd = player == 1 ? 1 : -1;
        rows[row] += toAdd;
        cols[col] += toAdd;
        if (row == col)
            diagonal += toAdd;
        if (col == (cols.length - row - 1))
            antiDiagonal += toAdd;
        int size = rows.length;
        if (Math.abs(rows[row]) == size || Math.abs(cols[col]) == size || Math.abs(diagonal) == si
ze
                || Math.abs(antiDiagonal) == size) {
            return player;
        }
        return 0;
    }
}
```

思路：与 n 后问题思路一致，重点在判断横、竖、斜四个方向上是否已经达到指定数量的相应棋子。

349. Intersection of Two Arrays Easy

Given two arrays, write a function to compute their intersection.

Example:

Given *nums1* = [1, 2, 2, 1], *nums2* = [2, 2], return [2].

Note:

- Each element in the result must be unique.
- The result can be in any order.

```
public int[] intersection(int[] nums1, int[] nums2) {
    Set<Integer> set = new HashSet<>();
    Set<Integer> intersect = new HashSet<>();
    for (int i = 0; i < nums1.length; i++)
        set.add(nums1[i]);
    for (int i = 0; i < nums2.length; i++)
        if (set.contains(nums2[i]))
            intersect.add(nums2[i]);
    int[] result = new int[intersect.size()];
    int i = 0;
    for (Integer num : intersect)
        result[i++] = num;
    return result;
}
```

思路：用 Set 记录一个数组中出现的所有数字，再在另一数组中找重现的并记录到另一 Set，后一 Set 的最终结果就是答案集合。

350. Intersection of Two Arrays II Easy

Given two arrays, write a function to compute their intersection.

Example:

Given *nums1* = [1, 2, 2, 1], *nums2* = [2, 2], return [2, 2].

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if *nums1*'s size is small compared to *nums2*'s size? Which algorithm is better?
- What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

```
public int[] intersect(int[] nums1, int[] nums2) {
    Map<Integer, Integer> set = new HashMap<>();
    Map<Integer, Integer> intersect = new HashMap<>();
    int count = 0;
    for (int i = 0; i < nums1.length; i++) {
        set.put(nums1[i], set.getOrDefault(nums1[i], 0) + 1);
    }
    for (int i = 0; i < nums2.length; i++) {
        if (set.containsKey(nums2[i]) && set.get(nums2[i]) > intersect.getOrDefault(nums2[i], 0))
{
            intersect.put(nums2[i], intersect.getOrDefault(nums2[i], 0) + 1);
            ++count;
        }
    }
    int[] result = new int[count];
    int j = 0;
    for (Integer num : intersect.keySet()) {
        for (int i = 0; i < intersect.get(num); ++i) {
            result[j++] = num;
        }
    }
    return result;
}
```

思路：用 Map 记录某一个数组里每个数出现的次数，遍历另一个数组时，如果出现重复则存入另一个 Map(计数不超过第一个 Map)。O(m+n)

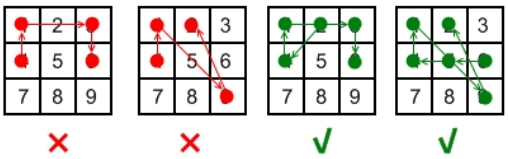
思路 2（略）：先把数组都排序，类似归并的方法找相同的数。O(nlogn)

351. Android Unlock Patterns Medium

Given an Android **3x3** key lock screen and two integers **m** and **n**, where 1 ≤ m ≤ n ≤ 9, count the total number of unlock patterns of the Android lock screen, which consist of minimum of **m** keys and maximum **n** keys.

Rules for a valid pattern:

1. Each pattern must connect at least **m** keys and at most **n** keys.
2. All the keys must be distinct.
3. If the line connecting two consecutive keys in the pattern passes through any other keys, the other keys must have previously selected in the pattern. No jumps through non selected key is allowed.
4. The order of keys used matters.



Explanation:

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

Invalid move: `4 - 1 - 3 - 6`
Line 1 - 3 passes through key 2 which had not been selected in the pattern.
Invalid move: `4 - 1 - 9 - 2`
Line 1 - 9 passes through key 5 which had not been selected in the pattern.
Valid move: `2 - 4 - 1 - 3 - 6`
Line 1 - 3 is valid because it passes through key 2, which had been selected in the pattern
Valid move: `6 - 5 - 4 - 1 - 9 - 2`
Line 1 - 9 is valid because it passes through key 5, which had been selected in the pattern.
Example:
Given **m** = 1, **n** = 1, return 9.

```
public int numberOfPatterns(int m, int n) {
    int[][] preSet = new int[9][9];
    preSet[0][2] = preSet[2][0] = 1;
    preSet[0][6] = preSet[6][0] = 3;
    preSet[3][5] = 4;
    preSet[5][3] = preSet[1][7] = preSet[7][1] = preSet[0][8] = preSet[8][0] = preSet[2][6] = preS
et[6][2] = 4;
    preSet[6][8] = preSet[8][6] = 7;
    preSet[2][8] = preSet[8][2] = 5;
    boolean[] visited = new boolean[9];
    int ans = 0;
    for (int i = m - 1; i < n; ++i) {
        ans += count(0, i, visited, preSet) * 4;
        ans += count(1, i, visited, preSet) * 4;
        ans += count(4, i, visited, preSet);
    }
    return ans;
}

private int count(int cur, int remain, boolean[] visited, int[][] preSet) {
    if (remain < 0)
        return 0;
    if (remain == 0)
        return 1;
    visited[cur] = true;
    int ans = 0;
    for (int i = 0; i < 9; ++i)
        if (!visited[i] && (preSet[cur][i] == 0 || visited[preSet[cur][i]]))
            ans += count(i, remain - 1, visited, preSet);
    visited[cur] = false;
    return ans;
}
```

思路：回溯法。深度优先遍历每种情况，对称情况可以只计算一次，然后乘对称数量。

352. Data Stream as Disjoint Intervals Hard

Given a data stream input of non-negative integers a_1, a_2, ..., a_n, ..., summarize the numbers seen so far as a list of disjoint intervals.
For example, suppose the integers from the data stream are 1, 3, 7, 2, 6, ..., then the summary will be:

```
[1, 1]
[1, 1], [3, 3]
[1, 1], [3, 3], [7, 7]
[1, 3], [7, 7]
```

[1, 3], [6, 7]

Follow up:
What if there are lots of merges and the number of disjoint intervals are small compared to the data stream's size?

```
public class SummaryRanges {
    TreeMap<Integer, int[]> tree;

    public SummaryRanges() {
        tree = new TreeMap<>();
    }

    public void addNum(int val) {
        if (tree.containsKey(val))
            return;
        Integer l = tree.lowerKey(val);
        Integer h = tree.higherKey(val);
        if (l != null && h != null && tree.get(l)[1] + 1 == val && h == val + 1) {
            tree.get(l)[1] = tree.get(h)[1];
            tree.remove(h);
        } else if (l != null && tree.get(l)[1] + 1 >= val) {
            tree.get(l)[1] = Math.max(tree.get(l)[1], val);
        } else if (h != null && h == val + 1) {
            tree.put(val, new int[] { val, tree.get(h)[1] });
            tree.remove(h);
        } else
            tree.put(val, new int[] { val, val });
    }

    public int[][] getIntervals() {
        return tree.values().toArray(new int[tree.values().size()][]);
    }
}
```

思路 1：用红黑树（TreeMap）存储区间。在红黑树的键值上存储区间的左值，在内部存储区间。每次先查找距离最近的 Intervals（左、右两侧的），然后判断新元素是否并入其 1，或与两个同时合并。取时只要把 Values 取来就是答案。

```
public class SummaryRanges {
    List<int[]> list;

    public SummaryRanges() {
        list = new ArrayList<>();
    }

    public void addNum(int val) {
        if (list.isEmpty()) {
            list.add(new int[] { val, val });
            return;
        }
        int i = 0;
        while (i < list.size() && list.get(i)[1] < val - 1)
            i++;
        if (i == list.size()) {
            list.add(new int[] { val, val });
            return;
        }
        if (val == list.get(i)[0] - 1)
            list.get(i)[0] = val;
        else if (val < list.get(i)[0] - 1)
            list.add(i, new int[] { val, val });
        else if (list.get(i)[1] == val - 1)
            list.get(i)[1] = val;
        if (i < list.size() - 1 && list.get(i)[1] + 1 == list.get(i + 1)[0]) {
            list.get(i)[1] = list.get(i + 1)[1];
```

```
            list.remove(i + 1);
        }
    }

    public int[][] getIntervals() {
        return list.toArray(new int[list.size()][]);
    }
}
```

思路 2：顺序查找。使用 List<int[]>作为存储。方法与红黑树一致，只不过找 Key 不再利用 Tree 的 Index，而是顺序查找每个 Interval 的左值。

353. Design Snake Game Medium

Design a Snake game that is played on a device with screen size = *width* x *height*. Play the game online if you are not familiar with the game.
The snake is initially positioned at the top left corner (0,0) with length = 1 unit.
You are given a list of food's positions in row-column order. When a snake eats the food, its length and the game's score both increase by 1.
Each food appears one by one on the screen. For example, the second food will not appear until the first food was eaten by the snake.
When a food does appear on the screen, it is guaranteed that it will not appear on a block occupied by the snake.
Example:

```
Given width = 3, height = 2, and food = [[1,2],[0,1]].
Snake snake = new Snake(width, height, food);
Initially the snake appears at position (0,0) and the food at (1,2).
|S| | |
| | |F|
snake.move("R"); -> Returns 0
| |S| |
| | |F|
snake.move("D"); -> Returns 0
| | | |
| |S|F|
snake.move("R"); -> Returns 1 (Snake eats the first food and right after that, the second food appears at (0,1) )
| |F| |
| |S|S|
snake.move("U"); -> Returns 1
| |F|S|
| | |S|
snake.move("L"); -> Returns 2 (Snake eats the second food)
| |S|S|
| | |S|
snake.move("U"); -> Returns -1 (Game over because snake collides with border)
```

```
public class SnakeGame {
    Set<Integer> set;
    Deque<Integer> body;
    int score;
    int[][] food;
    int foodIndex;
    int width;
    int height;

    public SnakeGame(int width, int height, int[][] food) {
        this.width = width;
        this.height = height;
```

```
        this.food = food;
        set = new HashSet<>();
        set.add(0);
        body = new LinkedList<>();
        body.offerLast(0);
    }

    public int move(String direction) {
        int rowHead = body.peekFirst() / width;
        int colHead = body.peekFirst() % width;
        switch (direction) {
            case "U":
                rowHead--;
                break;
            case "D":
                rowHead++;
                break;
            case "L":
                colHead--;
                break;
            default:
                colHead++;
        }
        int head = rowHead * width + colHead;
        set.remove(body.peekLast());
        if (rowHead < 0 || rowHead == height || colHead < 0 || colHead == width || set.contains(he
ad))
            return score = -1;
        set.add(head);
        body.offerFirst(head);
        if (foodIndex < food.length && rowHead == food[foodIndex][0] && colHead == food[foodIndex]
[1]) {
            set.add(body.peekLast());
            foodIndex++;
            return ++score;
        }
        body.pollLast();
        return score;
    }
}
```

思路：先搞清楚场景：初始状态；死亡状态；移动状态。然后设计主要动作（move）。二维图可以转为一维数组表示，用 Deque 保存蛇体，并用 Set 做索引，因为总是先进后出，且不会有重复数字（注意吃东西时会有暂时的重复，些时需要先从 Set 中移除尾巴，如果没有问题且吃了东西，再把尾巴加回来。

354. Russian Doll Envelopes Hard

You have a number of envelopes with widths and heights given as a pair of integers (w, h). One envelope can fit into another if and only if both the width and height of one envelope is greater than the width and height of the other envelope.

What is the maximum number of envelopes can you Russian doll? (put one inside other)

Example:

Given envelopes = [[5,4],[6,4],[6,7],[2,3]], the maximum number of envelopes you can Russian doll is 3 ([2,3] => [5,4] => [6,7]).

```
public int maxEnvelopes(int[][] envelopes) {
    if (envelopes == null || envelopes.length == 0 || envelopes[0] == null || envelopes[0].length
!= 2)
        return 0;
    Arrays.sort(envelopes, new Comparator<int[]>() {
        public int compare(int[] arr1, int[] arr2) {
            if (arr1[0] == arr2[0])
                return arr2[1] - arr1[1];
```

```
            else
                return arr1[0] - arr2[0];
        }
    });
    int dp[] = new int[envelopes.length];
    int len = 0;
    for (int[] envelope : envelopes) {
        int index = Arrays.binarySearch(dp, 0, len, envelope[1]);
        if (index < 0)
            index = -(index + 1);
        dp[index] = envelope[1];
        if (index == len)
            len++;
    }
    return len;
}
```

思路：动态规划+插入排序+二分查找。先把宽顺序排序，然后顺序遍历信封数组，这时只要后面的信封高度更高就是一个 Russian Doll。找更高的方法：插入排序+二分查找。逐一把高度插入新数组。在插入新元素前，二分查找其索引。如果该数不存在，则把它放在第一个大于它的数的位置（保留较小的）；如果存在，则无操作。如果插入位置是数组现有数的后面，则说明该高高于所有已知高，即出现了一个 Russian Doll。类似 300 题。`Arrays.binarySearch` 如果关键字在数列中，则返回其索引；如果不在，则返回应插入位置的相反数。

355. Design Twitter Medium

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets in the user's news feed. Your design should support the following methods:

1. **postTweet(userId, tweetId)**: Compose a new tweet.
2. **getNewsFeed(userId)**: Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent.
3. **follow(followerId, followeeId)**: Follower follows a followee.
4. **unfollow(followerId, followeeId)**: Follower unfollows a followee.

Example:

```
Twitter twitter = new Twitter();
// User 1 posts a new tweet (id = 5).
twitter.postTweet(1, 5);
// User 1's news feed should return a list with 1 tweet id -> [5].
twitter.getNewsFeed(1);
// User 1 follows user 2.
twitter.follow(1, 2);
// User 2 posts a new tweet (id = 6).
twitter.postTweet(2, 6);
// User 1's news feed should return a list with 2 tweet ids -> [6, 5].
// Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5.
twitter.getNewsFeed(1);
// User 1 unfollows user 2.
twitter.unfollow(1, 2);
// User 1's news feed should return a list with 1 tweet id -> [5],
// since user 1 is no longer following user 2.
twitter.getNewsFeed(1);
```

```
public class Twitter {
    private static int timeStamp = 0;
    private Map<Integer, User> userMap;

    private class Tweet {
```

```
        public int id;
        public int time;
        public Tweet next;

        public Tweet(int id) {
            this.id = id;
            time = timeStamp++;
            next = null;
        }
    }

    public class User {
        public int id;
        public Set<Integer> followed;
        public Tweet tweet_head;

        public User(int id) {
            this.id = id;
            followed = new HashSet<>();
            follow(id); // first follow itself
            tweet_head = null;
        }

        public void follow(int id) {
            followed.add(id);
        }

        public void unfollow(int id) {
            followed.remove(id);
        }

        public void post(int id) {
            Tweet t = new Tweet(id);
            t.next = tweet_head;
            tweet_head = t;
        }
    }

    public Twitter() {
        userMap = new HashMap<Integer, User>();
    }

    public void postTweet(int userId, int tweetId) {
        if (!userMap.containsKey(userId)) {
            User u = new User(userId);
            userMap.put(userId, u);
        }
        userMap.get(userId).post(tweetId);
    }

    public List<Integer> getNewsFeed(int userId) {
        List<Integer> res = new LinkedList<>();
        if (!userMap.containsKey(userId))
            return res;
        Set<Integer> users = userMap.get(userId).followed;
        PriorityQueue<Tweet> q = new PriorityQueue<Tweet>(users.size(), (a, b) -> (b.time - a.time
));
        for (int user : users) {
            Tweet t = userMap.get(user).tweet_head;
            if (t != null)
                q.add(t);
        }
        int n = 0;
```

```
        while (!q.isEmpty() && n < 10) {
            Tweet t = q.poll();
            res.add(t.id);
            n++;
            if (t.next != null)
                q.add(t.next);
        }
        return res;
    }

    public void follow(int followerId, int followeeId) {
        if (!userMap.containsKey(followerId)) {
            User u = new User(followerId);
            userMap.put(followerId, u);
        }
        if (!userMap.containsKey(followeeId)) {
            User u = new User(followeeId);
            userMap.put(followeeId, u);
        }
        userMap.get(followerId).follow(followeeId);
    }

    public void unfollow(int followerId, int followeeId) {
        if (!userMap.containsKey(followerId) || followerId == followeeId)
            return;
        userMap.get(followerId).unfollow(followeeId);
    }
}
```

356. Line Reflection Medium

Given n points on a 2D plane, find if there is such a line parallel to y-axis that reflect the given points.

Example 1:

Given *points* = [[1,1],[-1,1]], return true.

Example 2:

Given *points* = [[1,1],[-1,-1]], return false.

Follow up:

Could you do better than O(n^2)?

```
public boolean isReflected(int[][] points) {
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    HashSet<String> set = new HashSet<>();
    for (int[] p : points) {
        max = Math.max(max, p[0]);
        min = Math.min(min, p[0]);
        String str = p[0] + "a" + p[1];
        set.add(str);
    }
    int sum = max + min;
    for (int[] p : points) {
        String str = (sum - p[0]) + "a" + p[1];
        if (!set.contains(str))
            return false;
    }
    return true;
}
```

思路：每一个点都要有与 Y 轴对称的点，所以必有(max(x)+min(x) – xi, yi)与(xi,yi)同时存在，即必以max(x)+min(x)为轴。

357. Count Numbers with Unique Digits Medium

Given a **non-negative** integer n, count all numbers with unique digits, x, where $0 \le x < 10^n$.

Example:

Given n = 2, return 91. (The answer should be the total numbers in the range of $0 \le x < 100$, excluding [11,22,33,44,55,66,77,88,99])

```
public int countNumbersWithUniqueDigits(int n) {
    if (n == 0)
        return 1;
    int res = 10;
    int uniqueDigits = 9;
    int availableNumber = 9;
    while (n-- > 1 && availableNumber > 0) {
        uniqueDigits *= availableNumber;
        res += uniqueDigits;
        availableNumber--;
    }
    return res;
}
```

思路：排列题。从少向多推理，假设只有一位数，则最多有十个[0-9]。当有两位或以上位数时，最后一位数需要和前面的数组合，末位不能和前面已有数相同，所以每个数有 9 个数可以组合（availableNumber）。而两位数必须以[1-9]开头，所以有 9 种可能，故两位数的可能个数为 9*9。到了三位数时，只能在两位数的基础上加新的数，所以开头数就是 81 种可能，而每一个数可能与其组合的数的个数为 8，因为已经用过 2 个数，所以还有 8 种可能。依次类推。

358. Rearrange String k Distance Apart Hard

Given a non-empty string **s** and an integer **k**, rearrange the string such that the same characters are at least distance **k** from each other.

All input strings are given in lowercase letters. If it is not possible to rearrange the string, return an empty string "".

Example 1:

```
s = "aabbcc", k = 3
Result: "abcabc"
The same letters are at least distance 3 from each other.
```

Example 2:

```
s = "aaabc", k = 3
Answer: ""
It is not possible to rearrange the string.
```

Example 3:

```
s = "aaadbbcc", k = 2
Answer: "abacabcd"
Another possible answer is: "abcabcda"
The same letters are at least distance 2 from each other.
```

```
public String rearrangeString(String str, int k) {
    int length = str.length();
    int[] count = new int[26];
    int[] valid = new int[26];
    for (int i = 0; i < length; i++)
        count[str.charAt(i) - 'a']++;
    StringBuilder sb = new StringBuilder();
    for (int index = 0; index < length; index++) {
        int candidatePos = findValidMax(count, valid, index);
        if (candidatePos == -1)
            return "";
```

```
            count[candidatePos]--;
            valid[candidatePos] = index + k;
            sb.append((char) ('a' + candidatePos));
        }
        return sb.toString();
    }

    private int findValidMax(int[] count, int[] valid, int index) {
        int max = Integer.MIN_VALUE;
        int candidatePos = -1;
        for (int i = 0; i < count.length; i++) {
            if (count[i] > 0 && count[i] > max && index >= valid[i]) {
                max = count[i];
                candidatePos = i;
            }
        }
        return candidatePos;
    }
```

思路：先记录每一个字母出现的频次，通过另一数组记录每次出现字母的下一个有效位，逐一检索下一个可能的字母，每次都找数量最多且符合间隔要求的字母，这样是最大可能性，因为如果先用少的则少的用完了间隔自然变小。如果找不到则表明失败。如果找到，更新下一个有效位值，继续找。

359. Logger Rate Limiter Easy

Design a logger system that receive stream of messages along with its timestamps, each message should be printed if and only if it is **not printed in the last 10 seconds**.

Given a message and a timestamp (in seconds granularity), return true if the message should be printed in the given timestamp, otherwise returns false.

It is possible that several messages arrive roughly at the same time.

Example:

```
Logger logger = new Logger();
// logging string "foo" at timestamp 1
logger.shouldPrintMessage(1, "foo"); returns true;
// logging string "bar" at timestamp 2
logger.shouldPrintMessage(2,"bar"); returns true;
// logging string "foo" at timestamp 3
logger.shouldPrintMessage(3,"foo"); returns false;
// logging string "bar" at timestamp 8
logger.shouldPrintMessage(8,"bar"); returns false;
// logging string "foo" at timestamp 10
logger.shouldPrintMessage(10,"foo"); returns false;
// logging string "foo" at timestamp 11
logger.shouldPrintMessage(11,"foo"); returns true;
```

```
class Logger {
    private Map<String, Integer> logMap;

    public Logger() {
        logMap = new HashMap<>();
    }

    public boolean shouldPrintMessage(int timestamp, String message) {
        if (timestamp < logMap.getOrDefault(message, 0))
            return false;
        logMap.put(message, timestamp + 10);
        return true;
    }
}
```

360. Sort Transformed Array Medium

Given a **sorted** array of integers *nums* and integer values *a*, *b* and *c*. Apply a function of the form f(*x*) = $ax^2 + bx + c$ to each element *x* in the array.
The returned array must be in **sorted order**.
Expected time complexity: **O(*n*)**
Example:

```
nums = [-4, -2, 2, 4], a = 1, b = 3, c = 5,
Result: [3, 9, 15, 33]
nums = [-4, -2, 2, 4], a = -1, b = 3, c = 5
Result: [-23, -5, 1, 7]
```

```
public int[] sortTransformedArray(int[] nums, int a, int b, int c) {
    int n = nums.length;
    int[] sorted = new int[n];
    int i = 0, j = n - 1;
    int index = a >= 0 ? n - 1 : 0;
    while (i <= j) {
        if (a >= 0) {
            sorted[index-
-] = quad(nums[i], a, b, c) >= quad(nums[j], a, b, c) ? quad(nums[i++], a, b, c)
                    : quad(nums[j--], a, b, c);
        } else {
            sorted[index++] = quad(nums[i], a, b, c) >= quad(nums[j], a, b, c) ? quad(nums[j-
-], a, b, c)
                    : quad(nums[i++], a, b, c);
        }
    }
    return sorted;
}

private int quad(int x, int a, int b, int c) {
    return a * x * x + b * x + c;
}
```

思路：二次函数的图形为抛物线，a 大于 0 时开口向上，小于 0 时开口向下。因数组本已排序，故两端的点必为最值。根据这一特点，从两端逐一找最值，最终得到排好序的数组。

361. Bomb Enemy Medium

Given a 2D grid, each cell is either a wall `'W'`, an enemy `'E'` or empty `'0'` (the number zero), return the maximum enemies you can kill using one bomb.
The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.
Note that you can only put the bomb at an empty cell.
Example:

```
For the given grid
0 E 0 0
E 0 W E
0 E 0 0
return 3. (Placing a bomb at (1,1) kills 3 enemies)
```

```
public int maxKilledEnemies(char[][] grid) {
    int m = grid.length, n = m == 0 ? 0 : grid[0].length;
    int result = 0, rowHits = 0, colHits[] = new int[n];
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (j == 0 || grid[i][j - 1] == 'W') {
                rowHits = 0;
```

```
                // reset rowHits when hit wall, re calculate enemy after
                for (int k = j; k < n && grid[i][k] != 'W'; k++) {
                    rowHits += grid[i][k] == 'E' ? 1 : 0;
                    ;
                }
            }
            if (i == 0 || grid[i - 1][j] == 'W') {
                colHits[j] = 0;
                for (int k = i; k < m && grid[k][j] != 'W'; k++) {
                    colHits[j] += grid[k][j] == 'E' ? 1 : 0;
                }
            }
            if (grid[i][j] == '0') {
                result = Math.max(result, rowHits + colHits[j]);
            }
        }
    }
    return result;
}
```

思路：动态规划，左上到右下一次+右下到左上一次。记录每一行到当前行杀敌数量，如遇墙或另起一行则重置为 0。记录当前列杀敌数量到列杀敌数组，遇墙或另起一列则归 0。如果遇 0 就计算当前极值。

362. Design Hit Counter Medium

Design a hit counter which counts the number of hits received in the past 5 minutes.
Each function accepts a timestamp parameter (in seconds granularity) and you may assume that calls are being made to the system in chronological order (ie, the timestamp is monotonically increasing). You may assume that the earliest timestamp starts at 1.
It is possible that several hits arrive roughly at the same time.
Example:

```
HitCounter counter = new HitCounter();
// hit at timestamp 1.
counter.hit(1);
// hit at timestamp 2.
counter.hit(2);
// hit at timestamp 3.
counter.hit(3);
// get hits at timestamp 4, should return 3.
counter.getHits(4);
// hit at timestamp 300.
counter.hit(300);
// get hits at timestamp 300, should return 4.
counter.getHits(300);
// get hits at timestamp 301, should return 3.
counter.getHits(301);
```

Follow up:
What if the number of hits per second could be very large? Does your design scale?

```
public class HitCounter {
    private int[] times;
    private int[] hits;

    public HitCounter() {
        times = new int[300];
        hits = new int[300];
    }

    public void hit(int timestamp) {
```

```
        int index = timestamp % 300;
        if (times[index] != timestamp) {
            times[index] = timestamp;
            hits[index] = 1;
        } else
            hits[index]++;
    }

    public int getHits(int timestamp) {
        int total = 0;
        for (int i = 0; i < 300; i++) {
            if (timestamp - times[i] < 300) {
                total += hits[i];
            }
        }
        return total;
    }
}
```

思路：利用 5 分钟只有 300 个秒，把时间平均分派到 300 个索引中去，如果索引中的时间与传入时间不同，则说明旧的时间已经过时，置 0 且+1。

363. Max Sum of Rectangle No Larger Than K Hard

Given a non-empty 2D matrix *matrix* and an integer *k*, find the max sum of a rectangle in the *matrix* such that its sum is no larger than *k*.

Example:

```
Given matrix = [
  [1,  0, 1],
  [0, -2, 3]]
k = 2
```

The answer is 2. Because the sum of rectangle [[0, 1], [-2, 3]] is 2 and 2 is the max number no larger than k (k = 2).

Note:

1. The rectangle inside the matrix must have an area > 0.
2. What if the number of rows is much larger than the number of columns?

```
public int maxSumSubmatrix(int[][] matrix, int k) {
    if (matrix == null || matrix.length == 0 || matrix[0] == null || matrix[0].length == 0)
        return 0;
    int row = matrix.length;
    int col = matrix[0].length;
    int res = Integer.MIN_VALUE;

    for (int c = 0; c < col; c++) {
        int[] sum = new int[row];
        for (int l = c; l < col; l++) {
            for (int r = 0; r < row; r++)
                sum[r] += matrix[r][l];
            int cur = 0, max = sum[0];
            for (int ele : sum) {
                cur = Math.max(cur + ele, ele);
                max = Math.max(max, cur);
                if (max == k)
                    return k;
            }
            if (max < k)
                res = Math.max(res, max);
            else {// max larger than k, check every vertical subarray
                for (int i = 0; i < row; i++) {
                    int curSum = 0;
                    for (int j = i; j < row; j++) {
```

```
                    curSum += sum[j];
                    if (curSum <= k)
                        res = Math.max(res, curSum);
                }
            }
            if (res == k)
                return k;
        }
    }
}
return res;
}
```

思路：累加和。既然有可能是从中间开始，那么就把这种情况考虑进去。外面两重循环，都是列循环，外层从 0 到列数，内侧从当前列到列数，这样就有了任意两列的组合。累加还是以一维数组为基础，所有相同首尾的列的可以共享一个一维数组，这时数组中为每行的所有列的和。故只需要每次外层循环重置（这里是定义）数组。在里面，首先判定使用所有行的情况，这时取得极值，如果极值不到目标值，则取极值中的最值为备选答案；如果极值已大于目标值，则结果有可能不是从 0 行开始，故再行判断从某行开始加和的情况。

364. Nested List Weight Sum II Medium

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.
Each element is either an integer, or a list -- whose elements may also be integers or other lists.
Different from the previous question where weight is increasing from root to leaf, now the weight is defined from bottom up. i.e., the leaf level integers have weight 1, and the root level integers have the largest weight.

Example 1:

Given the list [[1,1],2,[1,1]], return **8**. (four 1's at depth 1, one 2 at depth 2)

Example 2:

Given the list [1,[4,[6]]], return **17**. (one 1 at depth 3, one 4 at depth 2, and one 6 at depth 1; 1*3 + 4*2 + 6*1 = 17)

```
public int depthSumInverse(List<NestedInteger> nestedList) {
    return depthSum(nestedList, getDepth(nestedList, 0));
}

private static int getDepth(List<NestedInteger> nestedList, int dep) {
    int depth = dep + 1;
    int max = depth;
    for (NestedInteger n : nestedList)
        if (!n.isInteger())
            max = Math.max(max, getDepth(n.getList(), depth));
    return max;
}

public int depthSum(List<NestedInteger> list, int depth) {
    int sum = 0;
    for (NestedInteger n : list) {
        if (n.isInteger())
            sum += n.getInteger() * depth;
        else
            sum += depthSum(n.getList(), depth - 1);
    }
    return sum;
}
```

思路 1：同 339 题，只不过权重是递减，需要先算出第一层的权重，即总层数。

```
public int depthSumInverse(List<NestedInteger> nestedList) {
    Queue<NestedInteger> queue = new LinkedList<>();
    for (NestedInteger i : nestedList)
        queue.offer(i);
    int sum = 0, tmp = 0;
    while (!queue.isEmpty()) {
        int size = queue.size();
        for (int i = 0; i < size; i++) {
```

```
            NestedInteger curr = queue.poll();
            if (curr.isInteger())
                tmp += curr.getInteger();
            else
                for (NestedInteger n : curr.getList())
                    queue.offer(n);
        }
        sum += tmp;
    }
    return sum;
}
```

思路 2：累加法：把权重大的累加多次。第一层的权重，并且=总层数 n，故把它累加 n 次。次一级的累加 n-1 次……，最里层仅加一次。

365. Water and Jug Problem Medium

You are given two jugs with capacities *x* and *y* litres. There is an infinite amount of water supply available. You need to determine whether it is possible to measure exactly *z* litres using these two jugs.

If *z* liters of water is measurable, you must have *z* liters of water contained within **one or both buckets** by the end.

Operations allowed:

- Fill any of the jugs completely with water.
- Empty any of the jugs.
- Pour water from one jug into another till the other jug is completely full or the first jug itself is empty.

Example 1: (From the famous *"Die Hard"* example)

```
Input: x = 3, y = 5, z = 4
Output: True
```

Example 2:

```
Input: x = 2, y = 6, z = 5
Output: False
```

```
public boolean canMeasureWater(int x, int y, int z) {
    if (x + y < z)
        return false;
    if (x == z || y == z || x + y == z)
        return true;
    return z % gcd(x, y) == 0;
}

public int gcd(int a, int b) {
    if (a == 0)
        return b;
    return gcd(b % a, as);
}
```

思路：只要 z 是 x、y 最大公约数的倍数，就可以做到。原理：根据更相减损法可知，最大公约数就是用两数可得到的最小单位量。而能得到的更大的单位量都是该量的倍数。

366. Find Leaves of Binary Tree Medium

Given a binary tree, collect a tree's nodes as if you were doing this: Collect and remove all leaves, repeat until the tree is empty.

Example:

Given binary tree

```
    1
   / \
```

```
  2  3
 / \
4  5
```

Returns [4, 5, 3], [2], [1].

Explanation:

1. Removing the leaves [4, 5, 3] would result in this tree:

```
  1
 /
2
```

2. Now removing the leaf [2] would result in this tree:

```
  1
```

3. Now removing the leaf [1] would result in the empty tree:

```
  []
```

Returns [4, 5, 3], [2], [1].

```
public List<List<Integer>> findLeaves(TreeNode root) {
    List<List<Integer>> list = new ArrayList<>();
    findLeavesHelper(list, root);
    return list;
}

private int findLeavesHelper(List<List<Integer>> list, TreeNode root) {
    if (root == null)
        return -1;
    int leftLevel = findLeavesHelper(list, root.left);
    int rightLevel = findLeavesHelper(list, root.right);
    int level = Math.max(leftLevel, rightLevel) + 1;
    if (list.size() == level)
        list.add(new ArrayList<>());
    list.get(level).add(root.val);
    return level;
}
```

思路：深度优先。根据深度添加到相应的 List 中，每个结点的深度=该结点到最远的叶子结点的深度+1，所以叶子结点深度为 0；叶子结点的子结点（null）为-1。

367. Valid Perfect Square Easy

Given a positive integer *num*, write a function which returns True if *num* is a perfect square else False.

Note: Do not use any built-in library function such as sqrt.

Example 1:

```
Input: 16
Returns: True
```

Example 2:

```
Input: 14
Returns: False
```

```
public boolean isPerfectSquare(int num) {
    int low = 1, high = num;
    while (low <= high) {
        long mid = (low + high) >>> 1;
        if (mid * mid == num)
```

```
            return true;
        else if (mid * mid < num)
            low = (int) mid + 1;
        else
            high = (int) mid - 1;

    }
    return false;
}
```
思路：二分查找。一个数有整数作为平方根，则该整数一定不大于它本身。

368. Largest Divisible Subset Medium

Given a set of **distinct** positive integers, find the largest subset such that every pair (S_i, S_j) of elements in this subset satisfies: S_i % S_j = 0 or S_j % S_i = 0.
If there are multiple solutions, return any subset is fine.
Example 1:

nums: [1,2,3]
Result: [1,2] (of course, [1,3] will also be ok)

Example 2:

nums: [1,2,4,8]
Result: [1,2,4,8]

```
public List<Integer> largestDivisibleSubset(int[] nums) {
    List<Integer> ans = new ArrayList<Integer>();
    Arrays.sort(nums);
    int n = nums.length;
    int[] count = new int[n];
    int[] preNum = new int[n];
    Arrays.fill(count, 1);
    Arrays.fill(preNum, -1);
    int max = Integer.MIN_VALUE, index = -1;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (nums[i] % nums[j] == 0 && count[j] + 1 > count[i]) {
                count[i] = count[j] + 1;
                preNum[i] = j;
            }
        }
        if (count[i] > max) {
            max = count[i];
            index = i;
        }
    }
    for (int i = index; i != -1; i = preNum[i])
        ans.add(nums[i]);
    return ans;
}
```
思路：动态规划。能被较大的数整除的一定可以被较小的数整除，递推方程：能被较小数整除，且该较小数的因子个数+1 大于当前数因子个数，则为新的当前极值，更新前值索引和当前最大因子数。

369. Plus One Linked List Medium

Given a non-negative integer represented as **non-empty** a singly linked list of digits, plus one to the integer.
You may assume the integer do not contain any leading zero, except the number 0 itself.
The digits are stored such that the most significant digit is at the head of the list.
Example:

```
Input:
1->2->3
Output:
1->2->4
```

```
public ListNode plusOne(ListNode head) {
    if (DFS(head) == 0)
        return head;
    else {
        ListNode newHead = new ListNode(1);
        newHead.next = head;
        return newHead;
    }
}

private static int DFS(ListNode head) {
    if (head == null)
        return 1;
    int res = DFS(head.next);
    int sum = res + head.val;
    head.val = sum % 10;
    return sum / 10;
}
```

思路：深度优先，直到遇到 null，返回进位 1。这时在最后一个结点后方。退回到最后结点后+上该 1，再保存当前位的值并返回当前进位。注意最后进位不为零时，需要再次进位（添加一个新头，值为 1）。

370. Range Addition Medium

Assume you have an array of length ***n*** initialized with all **0**'s and are given ***k*** update operations.
Each operation is represented as a triplet: **[startIndex, endIndex, inc]** which increments each element of subarray **A[startIndex ... endIndex]** (startIndex and endIndex inclusive) with **inc**.
Return the modified array after all ***k*** operations were executed.
Example:

```
Given:
  length = 5,
  updates = [
    [1,  3,  2],
    [2,  4,  3],
    [0,  2, -2]]
Output:
  [-2, 0, 3, 5, 3]
```

Explanation:

```
Initial state:
[ 0, 0, 0, 0, 0 ]
After applying operation [1, 3, 2]:
[ 0, 2, 2, 2, 0 ]
After applying operation [2, 4, 3]:
[ 0, 2, 5, 5, 3 ]
After applying operation [0, 2, -2]:
[-2, 0, 3, 5, 3 ]
```

```
public int[] getModifiedArray(int length, int[][] updates) {
    int[] ans = new int[length];
    for (int[] update : updates)
        for (int i = update[0]; i <= update[1]; ++i)
```

```
            ans[i] += update[2];
    return ans;
}
```

思路 1：暴力。

```
public int[] getModifiedArray(int length, int[][] updates) {
    int[] ans = new int[length];
    for (int[] update : updates) {
        ans[update[0]] += update[2];
        if (update[1] + 1 < length)
            ans[update[1] + 1] -= update[2];
    }
    for (int i = 1; i < length; ++i)
        if (ans[i] == 0)
            ans[i] = ans[i - 1];
        else
            ans[i] += ans[i - 1];
    return ans;
}
```

思路 2：缓存加值，减少运算次数。可以把每一个 update 看成从起始 index 开始，每位都+该 update 数，从结束 index 之后一位开始，每位都-该 update 数。这样在结果数组中可以只记录起始位置要+的值和结束位置要+的值（结束位计为负）。最后汇总计算时每位数最多计算一次。而且可以看出，每次如果位置上是 0，则说明与上一位的值相同，不是 0 则该位上数+前一位上的数值即为该位值。

371. Sum of Two Integers Easy

Calculate the sum of two integers *a* and *b*, but you are **not allowed** to use the operator `+` and `-`.

Example:

Given *a* = 1 and *b* = 2, return 3.

```
public int getSum(int a, int b) {
    if (a == 0)
        return b;
    if (b == 0)
        return a;
    while (b != 0) {
        int carry = a & b;
        a = a ^ b;
        b = carry << 1;
    }
    return a;
}
```

思路： A^B，能够得到没有进位的加法。A&B，能够得到相加之后，能够进位的位置的信息。向左移动一位，就是两个二进制数相加之后的进位信息。所以，(A&B)<<1 就是两个二进制数相加得到的“进位结果”。步骤三、将前两步的结果相加。相加的过程就是步骤一和步骤二，直到不再产生进位为止。

372. Super Pow Medium

Your task is to calculate a^b mod 1337 where *a* is a positive integer and *b* is an extremely large positive integer given in the form of an array.

Example1:

```
a = 2
b = [3]
Result: 8
```

Example2:

```
a = 2
b = [1,0]
```

Result: 1024

```
public int superPow(int a, int[] b) {
    if (a % 1337 == 0)
        return 0;
    int p = 0;
    for (int i : b)
        p = (p * 10 + i) % 1140;// 6*190=1140, 7*191=1337
    if (p == 0)
        p += 1440;
    return power(a, p, 1337);
}

public int power(int a, int n, int mod) {
    a %= mod;
    int ret = 1;
    while (n != 0) {
        if ((n & 1) != 0)
            ret = ret * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return ret;
}
```

思路：欧拉定理。详解

373. Find K Pairs with Smallest Sums Medium

You are given two integer arrays **nums1** and **nums2** sorted in ascending order and an integer **k**.
Define a pair **(u,v)** which consists of one element from the first array and one element from the second array.
Find the k pairs **$(u_1,v_1),(u_2,v_2)$ ...(u_k,v_k)** with the smallest sums.

Example 1:

Given nums1 = [1,7,11], nums2 = [2,4,6], k = 3
Return: [1,2],[1,4],[1,6]
The first 3 pairs are returned from the sequence:
[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]

Example 2:

Given nums1 = [1,1,2], nums2 = [1,2,3], k = 2
Return: [1,1],[1,1]
The first 2 pairs are returned from the sequence:
[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]

Example 3:

Given nums1 = [1,2], nums2 = [3], k = 3
Return: [1,3],[2,3]
All possible pairs are returned from the sequence:
[1,3],[2,3]

```
class Pair implements Comparable<Pair> {
    int x, y, sum;

    public Pair(int x, int y, int sum) {
        this.x = x;
        this.y = y;
        this.sum = sum;
    }
```

```
    @Override
    public int compareTo(Pair that) {
        return this.sum - that.sum;
    }
}

public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k) {
    List<List<Integer>> result = new ArrayList<>();
    if (nums1 == null || nums2 == null || nums1.length == 0 || nums2.length == 0)
        return result;

    PriorityQueue<Pair> pQueue = new PriorityQueue<>();

    int len1 = nums1.length, len2 = nums2.length;
    for (int i = 0; i < len2; i++) {
        pQueue.offer(new Pair(0, i, nums1[0] + nums2[i]));
    }
    for (int i = 0; i < Math.min(k, len1 * len2); i++) {
        Pair p = pQueue.poll();
        List<Integer> list = new ArrayList<>();
        list.add(nums1[p.x]);
        list.add(nums2[p.y]);
        result.add(list);
        if (p.x == len1 - 1)
            continue;
        pQueue.offer(new Pair(p.x + 1, p.y, nums1[p.x + 1] + nums2[p.y]));
    }
    return result;
}
```

思路：最小的组合一定是排的较靠前的数字的组合，用最小堆保存最小值，遍历所有可能。先取一个数组的首元素和另一数组每个元素两两结合，放入最小堆，可以证明堆中是小和值就是所有和值中最小。然后每次取出当前对后，把第一个数组对应的值换成其下一个数，组成新对再放入堆。这个和值就是第二个数组对应数与所有第一个数组中数和值中仅小于当前第一数组对应值的和值。

374. Guess Number Higher or Lower Easy

We are playing the Guess Game. The game is as follows:
I pick a number from **1** to ***n***. You have to guess which number I picked.
Every time you guess wrong, I'll tell you whether the number is higher or lower.
You call a pre-defined API guess(int num) which returns 3 possible results (-1, 1, or 0):

```
-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
```

Example:

```
n = 10, I pick 6.
Return 6.
```

```
public int guessNumber(int n) {
    int low = 1;
    int high = n;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        int res = guess(mid);
        if (res == 0)
            return mid;
        else if (res < 0)
            high = mid - 1;
```

```
            else
                low = mid + 1;
        }
        return -1;
    }
```
思路：二分查找。

375. Guess Number Higher or Lower II Medium

We are playing the Guess Game. The game is as follows:
I pick a number from **1** to **n**. You have to guess which number I picked.
Every time you guess wrong, I'll tell you whether the number I picked is higher or lower.
However, when you guess a particular number x, and you guess wrong, you pay **$x**. You win the game when you guess the number I picked.
Example:

```
n = 10, I pick 8.
First round:  You guess 5, I tell you that it's higher. You pay $5.
Second round: You guess 7, I tell you that it's higher. You pay $7.
Third round:  You guess 9, I tell you that it's lower. You pay $9.
Game over. 8 is the number I picked.
You end up paying $5 + $7 + $9 = $21.
```

Given a particular **n ≥ 1**, find out how much money you need to have to guarantee a **win**.

```
public int getMoneyAmount(int n) {
    int[][] dp = new int[n + 1][n + 1];
    return minCost(dp, 1, n);
}

private int minCost(int[][] dp, int l, int h) {
    if (l >= h)
        return 0;
    if (l + 1 == h)
        return l;
    if (dp[l][h] != 0)
        return dp[l][h];
    int minCost = Integer.MAX_VALUE;
    int mid = (h + l) / 2;
    for (int i = h - 1; i >= mid; i -= 2)
        minCost = Math.min(minCost, i + Math.max(minCost(dp, l, i - 1), minCost(dp, i + 1, h)));
    dp[l][h] = minCost;
    return minCost;
}
```
思路：动态规划。递推方程：当前错误时，其开销=当前值+前一半和后一半的最大值。因在后一半搜索时代价比前一半大，所以只要在后一半搜索就可以求出所需要的最多钱数。

376. Wiggle Subsequence Medium

A sequence of numbers is called a **wiggle sequence** if the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a wiggle sequence.
For example, [1,7,4,9,2,5] is a wiggle sequence because the differences (6,-3,5,-7,3) are alternately positive and negative. In contrast, [1,4,7,2,5] and [1,7,4,5,5] are not wiggle sequences, the first because its first two differences are positive and the second because its last difference is zero.
Given a sequence of integers, return the length of the longest subsequence that is a wiggle sequence. A subsequence is obtained by deleting some number of elements (eventually, also zero) from the original sequence, leaving the remaining elements in their original order.
Examples:

Input: [1,7,4,9,2,5]
Output: 6
The entire sequence is a wiggle sequence.
Input: [1,17,5,10,13,15,10,5,16,8]
Output: 7
There are several subsequences that achieve this length. One is [1,17,10,13,10,16,8].
Input: [1,2,3,4,5,6,7,8,9]
Output: 2

Follow up:
Can you do it in O(*n*) time?

```
public int wiggleMaxLength(int[] nums) {
    if (nums.length < 2)
        return nums.length;
    int down = 1, up = 1;
    for (int i = 1; i < nums.length; i++) {
        if (nums[i] > nums[i - 1])
            up = down + 1;
        else if (nums[i] < nums[i - 1])
            down = up + 1;
    }
    return Math.max(down, up);
}
```

思路：动态规划。必须一上一下，初始都为 1，遇到上则上=下+1，或反过来，因为要保证一上一下。最后取两者中较大值。

377. Combination Sum IV Medium

Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.
Example:

nums = [1, 2, 3] ***target*** = 4
The possible combination ways are:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
Note that different sequences are counted as different combinations.
Therefore the output is ***7***.

Follow up:
What if negative numbers are allowed in the given array?
How does it change the problem?
What limitation we need to add to the question to allow negative numbers?

```
int[] dp;
public int combinationSum4(int[] nums, int target) {
    dp = new int[target + 1];
    Arrays.fill(dp, -1);
    dp[0] = 1;
    return combinationSum(nums, target);
}s
private int combinationSum(int[] nums, int target) {
    if (dp[target] != -1)
```

```
        return dp[target];
    int ans = 0;
    for (int num : nums)
        if (num <= target)
            ans += combinationSum(nums, target - num);
    dp[target] = ans;
    return ans;
}
```

思路：动态规划（备忘录）：递推方程：总的可能数=所有去掉一个数字后的可能数之和。借助 dp 缓存已计算结果。详解

378. Kth Smallest Element in a Sorted Matrix Medium

Given a *n* x *n* matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:

```
matrix = [
  [ 1,  5,  9],
  [10, 11, 13],
  [12, 13, 15]],
k = 8,
return 13.
```

Note:

You may assume k is always valid, $1 \le k \le n^2$.

```
public int kthSmallest(int[][] matrix, int k) {
    int n = matrix.length;
    PriorityQueue<Tuple> pq = new PriorityQueue<Tuple>();
    for (int j = 0; j <= n - 1; j++)
        pq.offer(new Tuple(0, j, matrix[0][j]));
    for (int i = 0; i < k - 1; i++) {
        Tuple t = pq.poll();
        if (t.x == n - 1)
            continue;
        pq.offer(new Tuple(t.x + 1, t.y, matrix[t.x + 1][t.y]));
    }
    return pq.poll().val;
}

class Tuple implements Comparable<Tuple> {
    int x, y, val;

    public Tuple(int x, int y, int val) {
        this.x = x;
        this.y = y;
        this.val = val;
    }

    @Override
    public int compareTo(Tuple that) {
        return this.val - that.val;
    }
}
```

思路 1：利用最小值堆特性，先把第一行加入堆。然后进行 K-1 次操作：每次拿出最小值，取其下一元素加入堆。最后堆顶就是要求的解。

```
public int kthSmallest(int[][] matrix, int k) {
    int lo = matrix[0][0], hi = matrix[matrix.length - 1][matrix[0].length - 1] + 1;// [lo, hi)
    while (lo < hi) {
```

```
        int mid = lo + (hi - lo) / 2;
        int count = 0, j = matrix[0].length - 1;
        for (int i = 0; i < matrix.length; i++) {
            while (j >= 0 && matrix[i][j] > mid)
                j--;
            count += (j + 1);
        }
        if (count < k)
            lo = mid + 1;
        else
            hi = mid;
    }
    return lo;
}
```

思路 2：二分查找。一维二分查找，另一维线性查找。

379. Design Phone Directory Medium

Design a Phone Directory which supports the following operations:

1. get: Provide a number which is not assigned to anyone.
2. check: Check if a number is available or not.
3. release: Recycle or release a number.

Example:

```
// Init a phone directory containing a total of 3 numbers: 0, 1, and 2.
PhoneDirectory directory = new PhoneDirectory(3);
// It can return any available phone number. Here we assume it returns 0.
directory.get();
// Assume it returns 1.
directory.get();
// The number 2 is available, so return true.
directory.check(2);
// It returns 2, the only number that is left.
directory.get();
// The number 2 is no longer available, so return false.
directory.check(2);
// Release number 2 back to the pool.
directory.release(2);
// Number 2 is available again, return true.
directory.check(2);
```

```
public class PhoneDirectory {
    Set<Integer> used = new HashSet<Integer>();
    Queue<Integer> available = new LinkedList<Integer>();
    int max;

    public PhoneDirectory(int maxNumbers) {
        max = maxNumbers;
        for (int i = 0; i < maxNumbers; i++)
            available.offer(i);
    }

    public int get() {
        Integer ret = available.poll();
        if (ret == null)
            return -1;
        used.add(ret);
        return ret;
    }
```

```
    public boolean check(int number) {
        if (number >= max || number < 0)
            return false;
        return !used.contains(number);
    }

    public void release(int number) {
        if (used.remove(number))
            available.offer(number);
    }
}
```

380. Insert Delete GetRandom O(1) Medium

Design a data structure that supports all following operations in *average* **O(1)** time.

1. insert(val): Inserts an item val to the set if not already present.
2. remove(val): Removes an item val from the set if present.
3. getRandom: Returns a random element from current set of elements. Each element must have the **same probability** of being returned.

Example:

```
// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();
// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);
// Returns false as 2 does not exist in the set.
randomSet.remove(2);
// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);
// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();
// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);
// 2 was already in the set, so return false.
randomSet.insert(2);
// Since 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
```

```
public class RandomizedSet {
    ArrayList<Integer> nums;
    HashMap<Integer, Integer> locs;
    java.util.Random rand = new java.util.Random();

    public RandomizedSet() {
        nums = new ArrayList<Integer>();
        locs = new HashMap<Integer, Integer>();
    }

    public boolean insert(int val) {
        boolean contain = locs.containsKey(val);
        if (contain)
            return false;
        locs.put(val, nums.size());
        nums.add(val);
        return true;
    }

    public boolean remove(int val) {
```

```
        boolean contain = locs.containsKey(val);
        if (!contain)
            return false;
        int loc = locs.get(val);
        if (loc < nums.size() - 1) {
            int lastone = nums.get(nums.size() - 1);
            nums.set(loc, lastone);
            locs.put(lastone, loc);
        }
        locs.remove(val);
        nums.remove(nums.size() - 1);
        return true;
    }

    public int getRandom() {
        return nums.get(rand.nextInt(nums.size()));
    }
}
```

思路：用一个数组保存数值本身，用一个 Map 保存数在数组中的位置，当删除元素时，把末元与被删除元交换，删除末元。

381. Insert Delete GetRandom O(1) - Duplicates allowed Hard

Design a data structure that supports all following operations in *average* **O(1)** time.

Note: Duplicate elements are allowed.

1. insert(val): Inserts an item val to the collection.
2. remove(val): Removes an item val from the collection if present.
3. getRandom: Returns a random element from current collection of elements. The probability of each element being returned is **linearly related** to the number of same value the collection contains.

Example:

```
// Init an empty collection.
RandomizedCollection collection = new RandomizedCollection();
// Inserts 1 to the collection. Returns true as the collection did not contain 1.
collection.insert(1);
// Inserts another 1 to the collection. Returns false as the collection contained 1. Collection now contains [1,1].
collection.insert(1);
// Inserts 2 to the collection, returns true. Collection now contains [1,1,2].
collection.insert(2);
// getRandom should return 1 with the probability 2/3, and returns 2 with the probability 1/3.
collection.getRandom();
// Removes 1 from the collection, returns true. Collection now contains [1,2].
collection.remove(1);
// getRandom should return 1 and 2 both equally likely.
collection.getRandom();
```

```
public class RandomizedCollection {
    ArrayList<Integer> nums;
    HashMap<Integer, Set<Integer>> locs;
    java.util.Random rand = new java.util.Random();

    public RandomizedCollection() {
        nums = new ArrayList<Integer>();
        locs = new HashMap<Integer, Set<Integer>>();
    }

    public boolean insert(int val) {
        boolean contain = locs.containsKey(val);
        if (!contain)
```

```
            locs.put(val, new LinkedHashSet<Integer>());
        locs.get(val).add(nums.size());
        nums.add(val);
        return !contain;
    }

    public boolean remove(int val) {
        boolean contain = locs.containsKey(val);
        if (!contain)
            return false;
        int loc = locs.get(val).iterator().next();
        locs.get(val).remove(loc);
        if (loc < nums.size() - 1) {
            int lastone = nums.get(nums.size() - 1);
            nums.set(loc, lastone);
            locs.get(lastone).remove(nums.size() - 1);
            locs.get(lastone).add(loc);
        }
        nums.remove(nums.size() - 1);
        if (locs.get(val).isEmpty())
            locs.remove(val);
        return true;
    }

    public int getRandom() {
        return nums.get(rand.nextInt(nums.size()));
    }
}
```

思路：使用数组保存数值，使用 Map<int,list>保存数的位置，删除时，把要删除元素与最后一个元素互换，删除最后一个元素。

382. Linked List Random Node Medium

Given a singly linked list, return a random node's value from the linked list. Each node must have the **same probability** of being chosen.

Follow up:

What if the linked list is extremely large and its length is unknown to you? Could you solve this efficiently without using extra space?

Example:

```
// Init a singly linked list [1,2,3].
ListNode head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
Solution solution = new Solution(head);
// getRandom() should return either 1, 2, or 3 randomly. Each element should have equal probability of returning.
solution.getRandom();
```

```
ListNode head = null;
Random r = new Random();

public Solution(ListNode head) {
    this.head = head;
}

public int getRandom() {
    int result = this.head.val;
    ListNode node = this.head.next;
    int k = 1;
    int i = 1;
```

```
    while (node != null) {
        double x = r.nextDouble();
        double y = k / (k + i * 1.0);
        if (x <= y)
            result = node.val;
        i++;
        node = node.next;
    }
    return result;
}
```

思路：蓄水池。可以证明，第 i 个数，如果随机数<1/i，则为解。这里我们用<=1/(1+i)。详解

383. Ransom Note Easy

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.
Each letter in the magazine string can only be used once in your ransom note.

Note:

You may assume that both strings contain only lowercase letters.

```
canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true
```

```
public boolean canConstruct(String ransomNote, String magazine) {
    int[] freq = new int[26];
    int cnt = 0;
    for (char c : ransomNote.toCharArray()) {
        ++freq[c - 'a'];
        ++cnt;
    }
    for (char c : magazine.toCharArray()) {
        if (freq[c - 'a'] > 0) {
            --freq[c - 'a'];
            --cnt;
            if (cnt == 0)
                return true;
        }
    }
    return cnt == 0;
}
```

思路：词频计算问题。一旦 Magazine 中的词频小于 Note 中的，则无法完成。先计 ransomNote 中的词频，然后遍历 magazine 中的词，如果在 Note 中有，且词频数大于 0，则词频-1，总词频也-1。总词频为 0 时说明 Magazine 涵盖了所有词。

384. Shuffle an Array Medium

Shuffle a set of numbers without duplicates.

Example:

```
// Init an array with set 1, 2, and 3.
int[] nums = {1,2,3};
Solution solution = new Solution(nums);
// Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3] must equally likely to be returned.
solution.shuffle();
// Resets the array back to its original configuration [1,2,3].
solution.reset();
// Returns the random shuffling of array [1,2,3].
```

```
solution.shuffle();
```

```
private int[] nums;
private Random random;

public Solution(int[] nums) {
    this.nums = nums;
    random = new Random();
}

/** Resets the array to its original configuration and return it. */
public int[] reset() {
    return nums;
}

/** Returns a random shuffling of the array. */
public int[] shuffle() {
    if (nums == null)
        return null;
    int[] a = nums.clone();
    for (int j = 1; j < a.length; j++) {
        int i = random.nextInt(j + 1);
        swap(a, i, j);
    }
    return a;
}

private void swap(int[] a, int i, int j) {
    int t = a[i];
    a[i] = a[j];
    a[j] = t;
}
```

思路：不停地把前 j 个数中随机一个与第 j 个数进行 Shuffle。有几个数，则新数到每个位置的概率为 1/j。

385. Mini Parser Medium

Given a nested list of integers represented as a string, implement a parser to deserialize it.
Each element is either an integer, or a list -- whose elements may also be integers or other lists.
Note: You may assume that the string is well-formed:

- String is non-empty.
- String does not contain white spaces.
- String contains only digits 0-9, [, - ,,].

Example 1:

```
Given s = "324",
You should return a NestedInteger object which contains a single integer 324.
```

Example 2:

```
Given s = "[123,[456,[789]]]",
Return a NestedInteger object containing a nested list with 2 elements:
1. An integer containing value 123.
2. A nested list containing two elements:
   i.  An integer containing value 456.
   ii. A nested list with one element:
       a. An integer containing value 789.
```

```
public NestedInteger deserialize(String s) {
    if (!s.startsWith("["))
        return new NestedInteger(Integer.valueOf(s));
```

```
    NestedInteger ni = new NestedInteger();
    int i = 0;
    int open = 0;
    int start = -1;

    while (i < s.length()) {
        char c = s.charAt(i);
        i++;
        if (open == 1 && (Character.isDigit(c) || c == '-')) {
            int num = 0;
            if (c != '-')
                num = c - '0';
            while (i < s.length() && Character.isDigit(s.charAt(i))) {
                num = num * 10 + s.charAt(i) - '0';
                i++;
            }
            if (c == '-')
                num = -num;
            ni.add(new NestedInteger(num));
        } else if (c == '[') {
            if (open == 1)
                start = i - 1;
            open++;
        } else if (c == ']') {
            open--;
            if (open == 1)
                ni.add(deserialize(s.substring(start, i)));
        }
    }

    return ni;
}
```

思路：典型压栈题，每一级元素放在同一 List 中。递归实现，只有是根级括号（open == 1）才具体计算，否则压栈把子串进行相同的运算并作为当前数组的一个值（子序列）。

386. Lexicographical Numbers Medium

Given an integer *n*, return 1 ~ *n* in lexicographical order.

For example, given 13, return: [1,10,11,12,13,2,3,4,5,6,7,8,9].

Please optimize your algorithm to use less time and space. The input size may be as large as 5,000,000.

```
public List<Integer> lexicalOrder(int n) {
    List<Integer> list = new ArrayList<>(n);
    int curr = 1;
    for (int i = 0; i < n; i++) {
        list.add(curr);
        if (curr * 10 <= n) {
            curr *= 10;
        } else if (curr % 10 != 9 && curr + 1 <= n) {
            curr++;
        } else {
            while ((curr / 10) % 10 == 9) {
                curr /= 10;
            }
            curr = curr / 10 + 1;
        }
    }
    return list;
}
```

思路 1：按字典顺序生成数字序列。特殊点在每次+1 前，要把以该数字开头的更多位数的数字罗列出来。总共添加 n 个数，所以遍历 n 次。每次所需要的数从 1 开始，如果*10 还在范围内，则下一个数是*10 的结果；

否则只要末位不是 1，直接+1 就是下一个数；否则（末位是 9），不断去除最小位直到末位不是 9，然后/10+1。因为是 9 的话，+1 是 10 的倍数，已经在*10 时 Cover 了。

```
public List<Integer> lexicalOrder(int n) {
    List<Integer> res = new ArrayList<>();
    for (int i = 1; i < 10; ++i)
        dfs(i, n, res);
    return res;
}

public void dfs(int cur, int n, List<Integer> res) {
    if (cur > n) {
        return;
    } else {
        res.add(cur);
        for (int i = 0; i < 10; ++i) {
            if (10 * cur + i > n)
                return;
            dfs(10 * cur + i, n, res);
        }
    }
}
```

思路 2：深度优先。以 1-9 开头的所有值都深度优先刻有，因为同一数开头的值应该都在一起并且按位数和大小排序。

387. First Unique Character in a String Easy

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.
s = "loveleetcode",
return 2.
```

Note: You may assume the string contain only lowercase letters.

```
public int firstUniqChar(String s) {
    int freq[] = new int[26];
    for (int i = 0; i < s.length(); i++)
        freq[s.charAt(i) - 'a']++;
    for (int i = 0; i < s.length(); i++)
        if (freq[s.charAt(i) - 'a'] == 1)
            return i;
    return -1;
}
```

思路：两遍法。第一遍统计词频，第二遍输出第一个单频的位置。

388. Longest Absolute File Path Medium

Suppose we abstract our file system by a string in the following manner:

The string "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext" represents:

```
dir
  subdir1
  subdir2
    file.ext
```

The directory dir contains an empty sub-directory subdir1 and a sub-directory subdir2 containing a file file.ext.

The string "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext" represents:

```
dir
  subdir1
    file1.ext
    subsubdir1
  subdir2
    subsubdir2
      file2.ext
```

The directory dir contains two sub-directories subdir1 and subdir2. subdir1 contains a file file1.ext and an empty second-level sub-directory subsubdir1. subdir2 contains a second-level sub-directory subsubdir2 containing a file file2.ext.

We are interested in finding the longest (number of characters) absolute path to a file within our file system. For example, in the second example above, the longest absolute path is "dir/subdir2/subsubdir2/file2.ext", and its length is 32 (not including the double quotes).

Given a string representing the file system in the above format, return the length of the longest absolute path to file in the abstracted file system. If there is no file in the system, return 0.

Note:

- The name of a file contains at least a . and an extension.
- The name of a directory or sub-directory will not contain a ..

Time complexity required: O(n) where n is the size of the input string.

Notice that a/aa/aaa/file1.txt is not the longest file path, if there is another path aaaaaaaaaaaaaaaaaaaa/sth.png.

```
public int lengthLongestPath(String input) {
    String[] paths = input.split("\n");
    int[] stack = new int[paths.length + 1];
    int maxLen = 0;
    for (String path : paths) {
        int level = path.lastIndexOf("\t") + 1;
        stack[level + 1] = stack[level] + path.length() - level + 1;
        if (path.contains("."))
            maxLen = Math.max(maxLen, stack[level + 1] - 1);
    }
    return maxLen;
}
```

思路：如题中示例所示，路径间关系由\t 数量决定。其实是压栈问题，进入子文件夹压栈，出来时退栈，这样可以知道在什么位置，过程中记录路径长度即可。使用 Array 模拟栈更高效，技巧：让 Array 多一位，避开对位置-1 的处理（所有位置都+1）。

389. Find the Difference Easy

Given two strings ***s*** and ***t*** which consist of only lowercase letters.

String ***t*** is generated by random shuffling string ***s*** and then add one more letter at a random position.

Find the letter that was added in ***t***.

Example:

```
Input:
s = "abcd"
t = "abcde"
Output:e
Explanation:
'e' is the letter that was added.
```

```
public char findTheDifference(String s, String t) {
    int charCodeS = 0, charCodeT = 0;
    for (int i = 0; i < s.length(); ++i)
        charCodeS += (int) s.charAt(i);
    for (int i = 0; i < t.length(); ++i)
```

```
        charCodeT += (int) t.charAt(i);
    return (char) (charCodeT - charCodeS);
}
```
思路：只是多一个字符，所以转化成两数和差。也可以用 Map 来做。

390. Elimination Game Medium

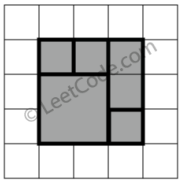
There is a list of sorted integers from 1 to *n*. Starting from left to right, remove the first number and every other number afterward until you reach the end of the list.
Repeat the previous step again, but this time from right to left, remove the right most number and every other number from the remaining numbers.
We keep repeating the steps again, alternating left to right and right to left, until a single number remains.
Find the last number that remains starting with a list of length *n*.
Example:

```
Input:
n = 9,
1 2 3 4 5 6 7 8 9
2 4 6 8
2 6
6
Output:6
```

```
public int lastRemaining(int n) {
    boolean left = true;
    int step = 1, head = 1;
    while (n > 1) {
        if (left || n % 2 == 1)
            head = head + step;
        n /= 2;
        step *= 2;
        left = !left;
    }
    return head;
}
```
思路：模拟删除过程。直到只剩一个数时就是答案，所以 n 大于 1 就继续；过程中每次删除一半，所以 n/2；当从左面删除或数字个数为奇数时，中心点需要前进，因为每次减少一半的数，所以中心点的位置前进 2 的指数倍，指数为已经进行删除操作的次数。详解

391. Perfect Rectangle Hard

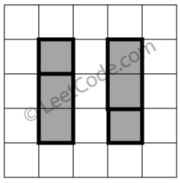
Given N axis-aligned rectangles where N > 0, determine if they all together form an exact cover of a rectangular region.
Each rectangle is represented as a bottom-left point and a top-right point. For example, a unit square is represented as [1,1,2,2]. (coordinate of bottom-left point is (1, 1) and top-right point is (2, 2)).



Example 1:

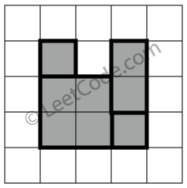
```
rectangles = [
  [1,1,3,3],
  [3,1,4,2],
```

[3,2,4,4],
[1,3,2,4],
[2,3,3,4]]
Return true. All 5 rectangles together form an exact cover of a rectangular region.



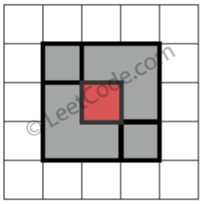
Example 2:

rectangles = [
[1,1,2,3],
[1,3,2,4],
[3,1,4,2],
[3,2,4,4]]
Return false. Because there is a gap between the two rectangular regions.



Example 3:

rectangles = [
[1,1,3,3],
[3,1,4,2],
[1,3,2,4],
[3,2,4,4]]
Return false. Because there is a gap in the top center.



Example 4:

rectangles = [
[1,1,3,3],
[3,1,4,2],
[1,3,2,4],
[2,2,4,4]]
Return false. Because two of the rectangles overlap with each other.

```java
public boolean isRectangleCover(int[][] rectangles) {
    if (rectangles.length == 0 || rectangles[0].length == 0)
        return false;
```

```
    int x1 = Integer.MAX_VALUE;
    int x2 = Integer.MIN_VALUE;
    int y1 = Integer.MAX_VALUE;
    int y2 = Integer.MIN_VALUE;
    Set<String> set = new HashSet<String>();
    int area = 0;
    for (int[] rect : rectangles) {
        x1 = Math.min(rect[0], x1);
        y1 = Math.min(rect[1], y1);
        x2 = Math.max(rect[2], x2);
        y2 = Math.max(rect[3], y2);
        area += (rect[2] - rect[0]) * (rect[3] - rect[1]);
        String s1 = rect[0] + " " + rect[1];
        String s2 = rect[0] + " " + rect[3];
        String s3 = rect[2] + " " + rect[3];
        String s4 = rect[2] + " " + rect[1];
        if (!set.add(s1))
            set.remove(s1);
        if (!set.add(s2))
            set.remove(s2);
        if (!set.add(s3))
            set.remove(s3);
        if (!set.add(s4))
            set.remove(s4);
    }
    if (!set.contains(x1 + " " + y1) || !set.contains(x1 + " " + y2) || !set.contains(x2 + " " + y
1)
            || !set.contains(x2 + " " + y2) || set.size() != 4)
        return false;
    return area == (x2 - x1) * (y2 - y1);
}
```

思路：如果所有小矩形正好形成大矩形，则大矩形面积=所有小矩形面积之和，且只有四个角的点出现一次，别的都出现偶数次。且出现一次的点就是大矩形的四顶点。

392. Is Subsequence Medium

Given a string **s** and a string **t**, check if **s** is subsequence of **t**.

You may assume that there is only lower case English letters in both **s** and **t**. **t** is potentially a very long (length ~= 500,000) string, and **s**is a short string (<=100).

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

Example 1:

s = "abc", **t** = "ahbgdc"

Return true.

Example 2:

s = "axc", **t** = "ahbgdc"

Return false.

Follow up:

If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you want to check one by one to see if T has its subsequence. In this scenario, how would you change your code?

```
public boolean isSubsequence(String s, String t) {
    if (s.length() == 0)
        return true;
    int indexS = 0, indexT = 0;
    while (indexT < t.length()) {
        if (t.charAt(indexT++) == s.charAt(indexS)) {
            indexS++;
            if (indexS == s.length())
                return true;
```

```
        }
    }
    return false;
}
```

思路：双指针法。

393. UTF-8 Validation Medium

A character in UTF8 can be from **1 to 4 bytes** long, subjected to the following rules:

1. For 1-byte character, the first bit is a 0, followed by its unicode code.
2. For n-bytes character, the first n-bits are all one's, the n+1 bit is 0, followed by n-1 bytes with most significant 2 bits being 10.

This is how the UTF-8 encoding would work:

```
Char. number range  |        UTF-8 octet sequence
   (hexadecimal)    |              (binary)
--------------------+---------------------------------------------
0000 0000-0000 007F | 0xxxxxxx
0000 0080-0000 07FF | 110xxxxx 10xxxxxx
0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Given an array of integers representing the data, return whether it is a valid utf-8 encoding.

Note:

The input is an array of integers. Only the **least significant 8 bits** of each integer is used to store the data. This means each integer represents only 1 byte of data.

Example 1:

data = [197, 130, 1], which represents the octet sequence: **11000101 10000010 00000001**.
Return **true**.
It is a valid utf-8 encoding for a 2-bytes character followed by a 1-byte character.

Example 2:

data = [235, 140, 4], which represented the octet sequence: **11101011 10001100 00000100**.
Return **false**.
The first 3 bits are all one's and the 4th bit is 0 means it is a 3-bytes character.
The next byte is a continuation byte which starts with 10 and that's correct.
But the second continuation byte does not start with 10, so it is invalid.

```
public boolean validUtf8(int[] data) {
    int cnt = 0;
    for (int d : data) {
        if (cnt == 0) {
            if ((d >> 5) == 0b110)
                cnt = 1;
            else if ((d >> 4) == 0b1110)
                cnt = 2;
            else if ((d >> 3) == 0b11110)
                cnt = 3;
            else if ((d >> 7) != 0)
                return false;
        } else {
            if ((d >> 6) != 0b10)
                return false;
            cnt--;
        }
    }
    return cnt == 0;
```

}
思路：观察得，单字节的是以 0 开头，否则开头有几个 1 则表示是几个字节（且其后需要紧跟 0），除第一字节外，后面字节都以 10 开头。

394. Decode String Medium

Given an encoded string, return it's decoded string.
The encoding rule is: k[encoded_string], where the *encoded_string* inside the square brackets is being repeated exactly *k* times. Note that *k* is guaranteed to be a positive integer.
You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, *k*. For example, there won't be input like 3a or 2[4].
Examples:

```
s = "3[a]2[bc]", return "aaabcbc".
s = "3[a2[c]]", return "accaccacc".
s = "2[abc]3[cd]ef", return "abcabccdcdcdef".
```

```
public String decodeString(String s) {
    var ans = new StringBuilder();
    char[] c = s.toCharArray();
    int n = c.length;
    Stack<Integer> cnt = new Stack<Integer>();
    Stack<String> str = new Stack<String>();
    for (int i = 0; i < n; ++i) {
        if (Character.isDigit(c[i])) {
            int num = 0;
            while (Character.isDigit(c[i]))
                num = num * 10 + (c[i++] - '0');
            cnt.push(num);
            --i;
        } else if (c[i] == '[') {
            str.push(ans.toString());
            ans = new StringBuilder();
        } else if (c[i] == ']') {
            StringBuilder tmp = new StringBuilder(str.pop());
            int rt = cnt.pop();
            while (rt-- > 0)
                tmp.append(ans);
            ans = tmp;
        } else {
            ans.append(c[i]);
        }
    }
    return ans.toString();
}
```

思路：通过 Stack 解析表达式，深层次的先运算，每层重复相应的次数。

395. Longest Substring with At Least K Repeating Characters Medium

Find the length of the longest substring ***T*** of a given string (consists of lowercase letters only) such that every character in ***T*** appears no less than *k* times.
Example 1:

```
Input:s = "aaabb", k = 3
Output:3
The longest substring is "aaa", as 'a' is repeated 3 times.
```

Example 2:

Input:s = "ababbc", k = 2
Output:5
The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.

```
public int longestSubstring(String s, int k) {
    char[] str = s.toCharArray();
    int[] counts = new int[26];
    int h, i, j, idx, max = 0, unique, noLessThanK;
    for (h = 1; h <= 26; h++) {
        Arrays.fill(counts, 0);
        i = 0;
        j = 0;
        unique = 0;
        noLessThanK = 0;
        while (j < str.length) {
            if (unique <= h) {
                idx = str[j] - 'a';
                if (counts[idx] == 0)
                    unique++;
                counts[idx]++;
                if (counts[idx] == k)
                    noLessThanK++;
                j++;
            } else {
                idx = str[i] - 'a';
                if (counts[idx] == k)
                    noLessThanK--;
                counts[idx]--;
                if (counts[idx] == 0)
                    unique--;
                i++;
            }
            if (unique == h && unique == noLessThanK)
                max = Math.max(j - i, max);
        }
    }
    return max;
}
```

思路：滑动窗口。检查从 1 个字母到 n 个字母的情况，每种情况都只考虑有 i 个字母的情况，如果找到符合条件的，则记录其长度，作为一个极值。

396. Rotate Function Medium

Given an array of integers A and let *n* to be its length.
Assume B_k to be an array obtained by rotating the array A *k* positions clock-wise, we define a "rotation function" F on A as follow:
$F(k) = 0 * B_k[0] + 1 * B_k[1] + ... + (n-1) * B_k[n-1]$.
Calculate the maximum value of F(0), F(1), ..., F(n-1).
Note:
n is guaranteed to be less than 10^5.
Example:

A = [4, 3, 2, 6]
F(0) = (0 * 4) + (1 * 3) + (2 * 2) + (3 * 6) = 0 + 3 + 4 + 18 = 25
F(1) = (0 * 6) + (1 * 4) + (2 * 3) + (3 * 2) = 0 + 4 + 6 + 6 = 16
F(2) = (0 * 2) + (1 * 6) + (2 * 4) + (3 * 3) = 0 + 6 + 8 + 9 = 23
F(3) = (0 * 3) + (1 * 2) + (2 * 6) + (3 * 4) = 0 + 2 + 12 + 12 = 26
So the maximum value of F(0), F(1), F(2), F(3) is F(3) = 26.

```
public int maxRotateFunction(int[] A) {
    if (A.length == 0)
        return 0;
    int sum = 0, iteration = 0, len = A.length;
    for (int i = 0; i < len; i++) {
        sum += A[i];
        iteration += (A[i] * i);
    }
    int max = iteration;
    for (int j = 1; j < len; j++) {
        iteration = iteration - sum + A[j - 1] * len;
        max = Math.max(max, iteration);
    }
    return max;
}
```

思路：方程为每一位开始向右（到头则从数组头部重新开始），遍历每一个数，而每一个数的加权为 0，1，2……。算出从 0 开始的结果 S 后，从 1 开始的结果为 S-sum + A[1]*A.length。把所有索引开始的情况都算一遍，取最大值。

397. Integer Replacement Medium

Given a positive integer *n* and you can do operations as follow:

1. If *n* is even, replace *n* with *n*/2.
2. If *n* is odd, you can replace *n* with either *n* + 1 or *n* - 1.

What is the minimum number of replacements needed for *n* to become 1?

Example 1:

Input:8
Output:3
Explanation:
8 -> 4 -> 2 -> 1

Example 2:

Input:7
Output:4
Explanation:
7 -> 8 -> 4 -> 2 -> 1 or 7 -> 6 -> 3 -> 2 -> 1

```
public int integerReplacement(int n) {
    if (n == Integer.MAX_VALUE)
        return 32;
    int count = 0;
    while (n > 1) {
        if (n % 2 == 0)
            n /= 2;
        else {
            if ((n + 1) % 4 == 0 && (n - 1 != 2))
                n++;
            else
                n--;
        }
        count++;
    }
    return count;
}
```

思路：是偶数非常容易，直接除 2，不是偶数则看+1 是否 4 的倍数且-1 不是 2，是则+1，否则-1。因为是 4 的倍数时，+1 后可以连续使用 2+次除法

398. Random Pick Index Medium

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

Note:

The array size can be very large. Solution that uses too much extra space will not pass the judge.

Example:

```
int[] nums = new int[] {1,2,3,3,3};
Solution solution = new Solution(nums);
// pick(3) should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning.
solution.pick(3);
// pick(1) should return 0. Since in the array only nums[0] is equal to 1.
solution.pick(1);
```

```
public class Solution {
    int[] nums;
    Random rnd;

    public Solution(int[] nums) {
        this.nums = nums;
        this.rnd = new Random();
    }

    public int pick(int target) {
        int result = -1;
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != target)
                continue;
            if (rnd.nextInt(++count) == 0)
                result = i;
        }
        return result;
    }
}
```

思路：水塘抽样（Reservoir Sampling）。顺序查找，每次随机找 0 位置，如果得到 0 位置，则更新候选，可以证明其概率相等。

399. Evaluate Division Medium

Equations are given in the format `A / B = k`, where `A` and `B` are variables represented as strings, and `k` is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return `-1.0`.

Example:

Given `a / b = 2.0, b / c = 3.0.`
queries are: `a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .`
return `[6.0, 0.5, -1.0, 1.0, -1.0 ].`
The input is: `vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries` , where `equations.size() == values.size()`, and the values are positive. This represents the equations.
Return `vector<double>`.
According to the example above:

```
equations = [ ["a", "b"], ["b", "c"] ],
values = [2.0, 3.0],
queries = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].
```

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

```
public double[] calcEquation(List<List<String>> equations, double[] values, List<List<String>> que
ries) {
    Map<String, List<String>> pairs = new HashMap<>();
    Map<String, List<Double>> valPairs = new HashMap<>();
    for (int i = 0; i < values.length; i++) {
        List<String> str = equations.get(i);
        String src = str.get(0), des = str.get(1);
        if (pairs.containsKey(src) == false) {
            pairs.put(src, new ArrayList<String>());
            valPairs.put(src, new ArrayList<Double>());
        }
        if (pairs.containsKey(des) == false) {
            pairs.put(des, new ArrayList<String>());
            valPairs.put(des, new ArrayList<Double>());
        }
        pairs.get(src).add(des);
        pairs.get(des).add(src);
        valPairs.get(src).add(values[i]);
        valPairs.get(des).add(1 / values[i]);
    }

    int N = queries.size();
    double[] res = new double[N];
    for (int i = 0; i < N; i++) {
        String src = queries.get(i).get(0), des = queries.get(i).get(1);
        double tmp = dfs(pairs, valPairs, src, des, new HashSet<>());
        res[i] = tmp == 0.0 ? -1 : tmp;
    }
    return res;
}

private double dfs(Map<String, List<String>> pairs, Map<String, List<Double>> valPairs, String src
, String des,
        Set<String> visited) {
    if (pairs.containsKey(src) == false || pairs.containsKey(des) == false)
        return 0.0;
    if (src.equals(des))
        return 1.0;
    if (visited.contains(src))
        return 0.0;
    visited.add(src);
    List<String> l1 = pairs.get(src);
    List<Double> l2 = valPairs.get(src);
    for (int i = 0; i < l1.size(); i++) {
        String n = l1.get(i);
        double t = dfs(pairs, valPairs, n, des, visited);
        if (t != 0.0) {
            return l2.get(i) * t;
        }
    }

    visited.remove(src);
    return 0.0;
}
```

思路：先建立除法（及其倒数）的词典，然后对查询进行广度优先查找以期得到结果。

400. Nth Digit Easy

Find the n^{th} digit of the infinite integer sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

Note:

n is positive and will fit within the range of a 32-bit signed integer ($n < 2^{31}$).

Example 1:

Input:3
Output:3

Example 2:

Input:11
Output:0
Explanation:
The 11th digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.

```
public int findNthDigit(int n) {
    int len = 1;
    long count = 9;
    int start = 1;
    while (n > len * count) {// 1
        n -= len * count;
        len += 1;
        count *= 10;
        start *= 10;
    }
    start += (n - 1) / len;// 2
    String s = Integer.toString(start);
    return Character.getNumericValue(s.charAt((n - 1) % len));// 3
}
```

思路：三步：1. 先找到第 n 个数字位来自第几个数字，每次步进 10 的倍数，首数为 1，10，100……；2. 找到该数字，该数为 start+(n-1)/len，start 本身是一个数，所以 n-1；3. 找到该数字位并返回，该数位为每(n-1)%len 位。