01_两数和

执行结果: 通过 显示详情 >

执行用时: 7 ms , 在所有 Java 提交中击败了 42.20% 的用户

内存消耗: 39 MB , 在所有 Java 提交中击败了 26.89% 的用户

炫耀一下:

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, List<Integer>> map = new HashMap<>();

        for(int i = 0; i < nums.length; i++){
            if(!map.containsKey(nums[i]))
                map.put(nums[i], new ArrayList<>());
            map.get(nums[i]).add(i);
        }

        for(int i = 0; i < nums.length - 1; i++){
            int one   = nums[i];
            int other = target - nums[i];

            if(map.containsKey(other)){
                if(one == other){
                    if(map.get(one).size() == 1)
                        continue;
                    else
                        return new int[]{map.get(one).get(0), map.get(one).get(1)};
                }else{
                    return new int[]{i, map.get(other).get(0)};
                }
            }
        }

        return new int[]{-1, -1};
    }
}
```

02_链表相加

```java
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode cur1 = l1;
        ListNode cur2 = l2;

        while(cur1 != null && cur2 != null){
            int sum = cur1.val + cur2.val;

            cur1.val = sum;
            cur2.val = sum;
            cur1     = cur1.next;
            cur2     = cur2.next;
        }
        ListNode trav1 = cur1 == null ? l2 : l1;

        while(trav1 != null){
            if(trav1.val >= 10){
                trav1.val -= 10;
                if(trav1.next == null){
                    trav1.next = new ListNode(1);
                }else{
                    trav1.next.val += 1;
                }
            }

            trav1 = trav1.next;
        }

        return cur1 == null ? l2 : l1;
    }
}
```

03_最大长度不重复字符

执行结果： 通过 显示详情 >

执行用时： 12 ms , 在所有 Java 提交中击败了 27.33% 的用户

内存消耗： 38.7 MB , 在所有 Java 提交中击败了 58.64% 的用户

炫耀一下：

```java
    //滑动窗口
    public int lengthOfLongestSubstring(String s) {
        int left = 0, right = 0;

        int res = 0;
        HashMap<Character, Integer> map = new HashMap<>();
        while(right < s.length()){
            char ch = '$';
            while(right < s.length() && map.getOrDefault(s.charAt(right), 0) <= 1){
                map.put(s.charAt(right), map.getOrDefault(s.charAt(right), 0) + 1);

                if(map.get(s.charAt(right)) > 1){
                    ch = s.charAt(right);
                    right++;
                    break;
                }
                else{
                    res = Math.max(res, right - left + 1);
                    right++;
                }
            }

            if(right == s.length()) break;
            if(ch == '$')   continue;

            while(left < right && map.get(ch) > 1){
                map.put(s.charAt(left), map.get(s.charAt(left)) - 1);
                left++;
            }
        }
        return res;
    }
```

这个代码写的比我的那个简洁

```java
public int lengthOfLongestSubstring(String s) {
    int left = 0, right = 0;
    HashMap<Character, Integer> map = new HashMap<>();
    int res = 0;
```

```
    while(right < s.length())
    {
        while(right < s.length() && map.getOrDefault(s.charAt(right), 0) < 1)
        {
            map.put(s.charAt(right), map.getOrDefault(s.charAt(right), 0) + 1);
            right++;
        }
        res = Math.max(res, right - left);

        while(right < s.length() && map.getOrDefault(s.charAt(right), 0) >= 1)
        {
            map.put(s.charAt(left), map.getOrDefault(s.charAt(left), 0)-1);
            left++;
        }
    }
    return res;
}
```

04_寻找两个有序数组的中位数O(logn)

初始思路：没有好的思路，最多就是时间复杂度为m+n，做不到题目要求时间复杂度。考虑了treeSet但是么思路

笨办法：虽然可以通过编译，但是并不正确

```
 public static void solution2(int[] nums1, int[] nums2){
        List<Integer> list = new ArrayList<>();
        for(int value:nums1){
            list.add(value);
        }
        for(int value:nums2){
            list.add(value);
        }
        Collections.sort(list);
        int mid = (list.size())/2;
        if(list.size() %2 == 0){
            System.out.println((list.get(mid-1) + list.get(mid))*1.0/2);
        }else{
            System.out.println(list.get(mid/2)*1.0);
        }
    }
```

优化思路：一般看到logn的时间复杂度，都要想起二分法。十分类似于找第k小的数字

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
```

```
    int n = nums1.length;
    int m = nums2.length;
    int left = (n + m + 1) / 2;
    int right = (n + m + 2) / 2;
    //将偶数和奇数的情况合并，如果是奇数，会求两次同样的 k 。
    return (getKth(nums1, 0, n - 1, nums2, 0, m - 1, left) + getKth(nums1, 0, n - 1, nums2, 0, m - 1, right)) * 0.5;
}
private int getKth(int[] nums1, int start1, int end1, int[] nums2, int start2, int end2, int k) {
    int len1 = end1 - start1 + 1;
    int len2 = end2 - start2 + 1;
    //让 len1 的长度小于 len2，这样就能保证如果有数组空了，一定是 len1
    if (len1 > len2) return getKth(nums2, start2, end2, nums1, start1, end1, k);
    if (len1 == 0) return nums2[start2 + k - 1];

    if (k == 1) return Math.min(nums1[start1], nums2[start2]);

    int i = start1 + Math.min(len1, k / 2) - 1;
    int j = start2 + Math.min(len2, k / 2) - 1;

    if (nums1[i] > nums2[j]) {
        return getKth(nums1, start1, end1, nums2, j + 1, end2, k - (j - start2 + 1));
    }
    else {
        return getKth(nums1, i + 1, end1, nums2, start2, end2, k - (i - start1 + 1));
    }
}
```

(未完)05_

006 ZigZag Conversion

6. ZigZag Conversion

难度 中等 731

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

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

And then read line by line: `"PAHNAPLSIIGYIR"`

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

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

Example 1:

```
Input: s = "PAYPALISHIRING", numRows = 3
Output: "PAHNAPLSIIGYIR"
```

Example 2:

```
Input: s = "PAYPALISHIRING", numRows = 4
Output: "PINALSIGYAHRPI"
Explanation:

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

通过次数 143,608 | 提交次数 297,778

在真实的面试中遇到过这道题?

执行结果: 通过 显示详情 >

执行用时: 91 ms , 在所有 Java 提交中击败了 5.07% 的用户

内存消耗: 42.1 MB , 在所有 Java 提交中击败了 8.33% 的用户

炫耀一下:

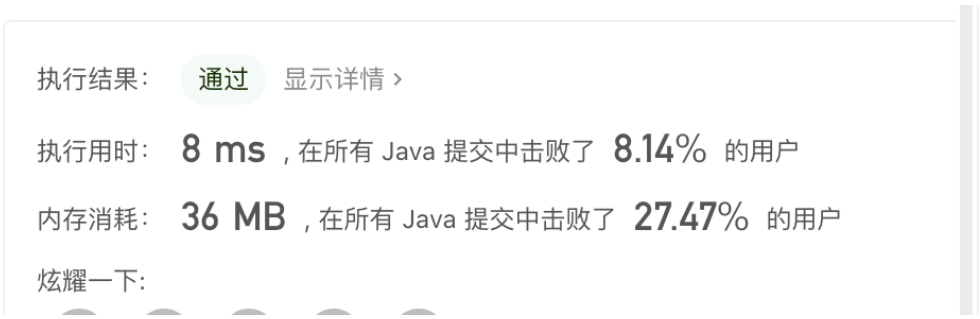
```
/*
  感觉像脑筋急转弯
  没什么技术含量
  就是来回游走
*/
```

```java
public String convert(String s, int numRows) {
    int len = s.length();
    if(numRows == 1)        return s;
    if(s.length() == 0)     return s;
    char[][] board = new char[numRows][len];
    for(char[] ch : board)
        Arrays.fill(ch, '1');

    int index = 0;
    int row = 0, column = 0;

    while(index != s.length())
    {
        while(row != numRows && index != s.length())
        {
            board[row++][column] = s.charAt(index++);
        }

        row = numRows-1;


        while(row >= 1 && index != s.length())
        {
            board[--row][++column] = s.charAt(index++);
        }
        row++;
    }

    StringBuilder res = new StringBuilder();
    for(int i = 0; i < numRows; i++)
        for(int j = 0; j < len; j++)
            if(board[i][j] != '1')
                res.append(board[i][j]);
    return res.toString();
}
```

```java
/*
  优化后:
  使用StringBuild,  不局限于题目给的
  实际上最后StringBuilder 的形状是
  PAHN
  APLSIIG
  YIR

  然后用个遍历把它们收集来就可以
  执行用时: 6 ms, 在所有 Java 提交中击败了80.16%的用户
  内存消耗: 40.1 MB, 在所有 Java 提交中击败了8.33%的用户
*/
public String convert(String s, int numRows) {
```

```
    if(numRows == 1)        return s;
    if(s.length() == 0)     return s;

    StringBuilder[] sb = new StringBuilder[numRows];
    for(int i = 0; i  < numRows; i++) sb[i] = new StringBuilder();

    int index = 0;
    int row = 0;


    while(index < s.length())
    {
        while(row < numRows && index < s.length())
            sb[row++].append(s.charAt(index++));

        row--;

        while(row >= 1 && index < s.length())
            sb[--row].append(s.charAt(index++));

        row++;
    }

    StringBuilder res = new StringBuilder();
    for(StringBuilder sa : sb)
        res.append(sa);
    return res.toString();
}
```

007 整数反转




```
    public int reverse(int x) {
        boolean neg = x < 0;
        if(neg) x = -x;

```

```
        String num = x + "";

        int res = 0;
        int index = num.length() - 1;
        while(index >= 0 && num.charAt(index) == '0')
            index--;
        if(index < 0)   return 0;
        for(int i = index; i >= 0; i--){

            int n = num.charAt(i) - '0';
            if(res > Integer.MAX_VALUE / 10 ||
                (res == Integer.MAX_VALUE / 10 && n > Integer.MAX_VALUE % 10))
                return 0;

            res *= 10;
            res += n;
        }
        return neg ? -res : res;
    }
```

```
/*
  如果 rev > intMax / 10 ，那么没的说，此时肯定溢出了。

    如果 rev == intMax / 10 = 2147483647 / 10 = 214748364 ，此时 rev * 10 就是 2147483640 如果 pop 大于 7 ，那么就一定溢出了。但是！如果假设 pop 等于 8，那么意味着原数 x 是 8463847412 了，输入的是 int ，而此时是溢出的状态，所以不可能输入，所以意味着 pop 不可能大于 7 ，也就意味着 rev == intMax / 10 时不会造成溢出。

    如果 rev < intMax / 10 ，意味着 rev 最大是 214748363 ， rev * 10 就是 2147483630 ，此时再加上 pop ，一定不会溢出。
*/
public int reverse(int x) {
    int rev = 0;
    while (x != 0) {
        int pop = x % 10;
        x /= 10;
        if (rev > Integer.MAX_VALUE/10 ) return 0;
        if (rev < Integer.MIN_VALUE/10 ) return 0;
        rev = rev * 10 + pop;
    }
    return rev;
}
```

008_字符串转整数

8. String to Integer (atoi)

难度 中等 741

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

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

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

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

If no valid conversion could be performed, a zero value is returned.

Note:

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

Example 1:

```
Input: "42"
Output: 42
```

Example 2:

```
Input: "   -42"
Output: -42
Explanation: The first non-whitespace character is '-',
which is the minus sign.
             Then take as many numerical digits as
possible, which gets 42.
```

Example 3:

```
Input: "4193 with words"
Output: 4193
Explanation: Conversion stops at digit '3' as the next
character is not a numerical digit.
```

Example 4:

```
Input: "words and 987"
Output: 0
Explanation: The first non-whitespace character is 'w',
which is not a numerical
             digit or a +/- sign. Therefore no valid
conversion could be performed.
```

Example 5:

```
Input: "-91283472332"
Output: -2147483648
Explanation: The number "-91283472332" is out of the
range of a 32-bit signed integer.
```

执行结果： 通过 显示详情 >

执行用时： 2 ms , 在所有 Java 提交中击败了 99.29% 的用户

内存消耗： 38.4 MB , 在所有 Java 提交中击败了 79.69% 的用户

炫耀一下:

```java
class Solution {
    public int myAtoi(String s) {
        if(s.length() == 0) return 0;
        int index = 0;

        //to eliminate the space
        while(index < s.length() && s.charAt(index) == ' ')
            index++;
        if(index == s.length())     return 0;

        boolean neg = false;
        if(s.charAt(index) > '9' || s.charAt(index) < '0')
            if(s.charAt(index) != '+' && s.charAt(index) != '-')
                return 0;

        if(s.charAt(index) == '+' || s.charAt(index) == '-'){
            neg = s.charAt(index) == '-';
            index++;
        }

        int res = 0;
        while(index < s.length() && s.charAt(index) >= '0' &&  s.charAt(index) <= '9'){
            int num = s.charAt(index++) - '0';

```

```
            if(!neg && (res > Integer.MAX_VALUE / 10 || (res == Integer.MAX_VALUE / 10 && num > Integer.MAX_VALUE % 10)
            ))
                return Integer.MAX_VALUE;
            if(neg && (res > -(Integer.MIN_VALUE / 10) || (res == -(Integer.MIN_VALUE / 10)  && num > -(Integer.MIN_VALUE % 10))))
                return Integer.MIN_VALUE;

            res *= 10;
            res += num;
        }

        return neg ? -res : res;
    }
}
```

```
   public int myAtoi(String str) {
        int res = 0;
        int pop = 0;
        int sign = 1;
        boolean hasSign = false;

        for(int i = 0; i < str.length(); i++)
        {
            char ch = str.charAt(i);
            if(ch == ' ' && !hasSign)
                continue;
            if(ch == '+' && !hasSign)
            {
                hasSign = true;
                continue;
            }
            if(ch == '-' && !hasSign)
            {
                hasSign = true;
                sign = -1;
                continue;
            }

            if(ch >= '0' && ch <= '9')
            {
                hasSign = true;
                pop = ch  - '0';
                if(res*sign > Integer.MAX_VALUE / 10 || (res*sign == Integer.MAX_VALUE / 10 && pop*sign > 7))
```

```
                    return Integer.MAX_VALUE;
                if(res*sign < Integer.MIN_VALUE / 10 || (res*sign == Integer.MIN_VALUE / 10 && pop*sign < -8))
                    return Integer.MIN_VALUE;

                res = res * 10 + pop;
            }
            else
                return res * sign;
        }
        return res * sign;
    }
```

```
/*
  DFA解法，日后回来研究
*/
class Solution {

    class Automaton {
    final String START = "start";
    final String SIGNED = "signed";
    final String IN_NUM = "in_number";
    final String END = "end";
    String state = START;
    Map<String, String[]> map;
    public int sign = 1;
    public long ans = 0;

    public Automaton() {
        map = new HashMap<>();
        map.put(START, new String[]{START, SIGNED, IN_NUM, END});
        map.put(SIGNED, new String[]{END, END, IN_NUM, END});
        map.put(IN_NUM, new String[]{END, END, IN_NUM, END});
        map.put(END, new String[]{END, END, END, END});
    }

    public int get_col(char c) {
        if (c == ' ') return 0;
        if (c == '+' || c == '-') return 1;
        if (c >= '0' && c <= '9') return 2;
        return 3;
    }

    public void get(char c) {
        state = map.get(state)[get_col(c)];
        if (state.equals(IN_NUM)) {
            ans = ans * 10 + c - '0';
```

```
            if (sign == 1) {
                ans = Math.min(ans, Integer.MAX_VALUE);
            } else {
                // -(long)Integer.MIN_VALUE，这个操作有点东西，不然越界
                ans = Math.min(ans, -(long)Integer.MIN_VALUE);
            }
        } else if (state.equals(SIGNED))
            sign = c == '+' ? 1 : -1;
    }
    }

    public int myAtoi(String str) {
        Automaton automaton = new Automaton();
        char[] c = str.toCharArray();
        for (char ch : c) {
            automaton.get(ch);
        }
        return automaton.sign * ((int) automaton.ans);
    }
}
```

09_回文数的判断

执行结果: 通过 显示详情 >

执行用时: 10 ms , 在所有 Java 提交中击败了 78.98% 的用户

内存消耗: 37.6 MB , 在所有 Java 提交中击败了 91.70% 的用户

炫耀一下:

```
//思路就是一半一半看
public boolean isPalindrome(int x) {
        if(x < 0)       return false;

        int num = x;
        int target = 0;
        while(num > 0){

            target *= 10;
            target += num % 10;

            num /= 10;
        }

```

```
        return target == x;
    }
```

题目：

一开始想到的思路绝对是将整数转换为字符串进行求解，十分好想且简单

```
public static boolean isPalindrome(int x) {
    if(x<0){
        return false;
    }
    StringBuilder revStr = new StringBuilder(x + "").reverse();
    StringBuilder oriStr = new StringBuilder(x+ "");

    if(oriStr.toString().equals(revStr.toString())){
        return true;
    }else{
        return false;
    }
}
```

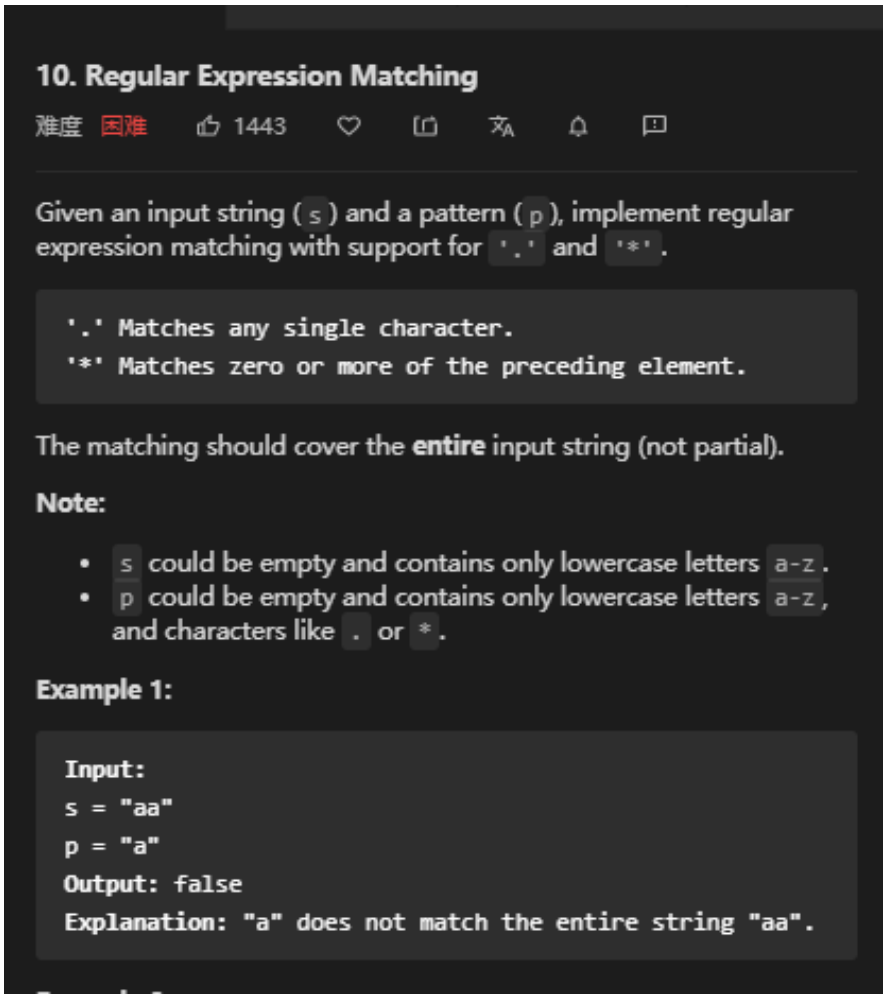
官方给出题解：

因为是回文数，所以只要比较一半的数字即可，反正剩下的也相同，因此通过除也好，取模也好，拿到该数的右边一半，然后和左边一半进行比较即可

```
public class Solution {
    public bool IsPalindrome(int x) {
        // 特殊情况：
        // 如上所述，当 x < 0 时，x 不是回文数。
        // 同样地，如果数字的最后一位是 0，为了使该数字为回文，
        // 则其第一位数字也应该是 0
        // 只有 0 满足这一属性
        if(x < 0 || (x % 10 == 0 && x != 0)) {
            return false;
        }
  int revertedNumber = 0;
    while(x > revertedNumber) {
        revertedNumber = revertedNumber * 10 + x % 10;
```

```
        x /= 10;
    }

    // 当数字长度为奇数时，我们可以通过 revertedNumber/10 去除处于中位的数字。
    // 例如，当输入为 12321 时，在 while 循环的末尾我们可以得到 x = 12，revertedNumber =
123,
    // 由于处于中位的数字不影响回文（它总是与自己相等），所以我们可以简单地将其去除。
    return x == revertedNumber || x == revertedNumber/10;
  }
}
```

010 RegExp



10. Regular Expression Matching

难度 困难 1443

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".
```

```
//递归求解
public boolean isMatch(String text, String pattern) {
        if (pattern.isEmpty()) return text.isEmpty();

        boolean first_match = (!text.isEmpty() &&
                               (pattern.charAt(0) == text.charAt(0) ||
pattern.charAt(0) == '.'));
```

```
7
8
9       //只有pattern长度大于等于 2 的时候，才考虑 *
10      /*两种情况
11          pattern 直接跳过两个字符。表示 * 前边的字符出现 0 次
12          pattern 不变，例如 text = aa , pattern = a*，第一个 a 匹配
13          然后 text 的第二个 a 接着和 pattern 的第一个 a 进行匹配。表示 * 用前一个字符替代。
14      */
15       if (pattern.length() >= 2 && pattern.charAt(1) == '*')
16              return (isMatch(text, pattern.substring(2)) ||
17                      (first_match && isMatch(text.substring(1), pattern)));
18      else
19          return first_match && isMatch(text.substring(1), pattern.substring(1));
20
21  }
22  https://leetcode.wang/leetCode-10-Regular-Expression-Matching.html
```

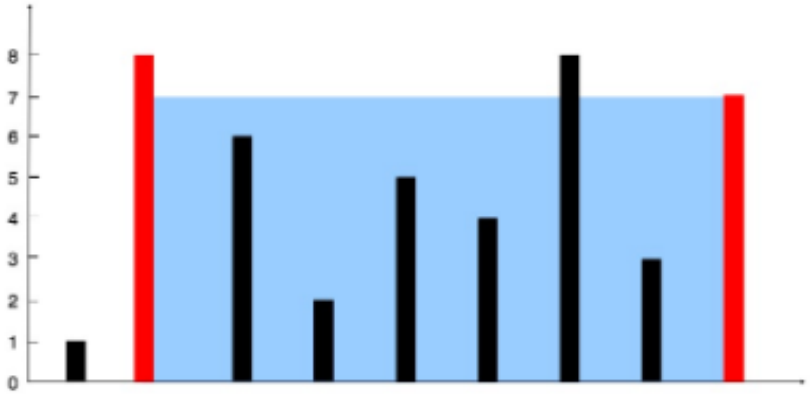
11_盛水最多

11. 盛最多水的容器

难度 中等 1441

给你 n 个非负整数 a_1，a_2，...，a_n，每个数代表坐标中的一个点 (i, a_i) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, a_i) 和 $(i, 0)$。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2。



图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例：

输入：[1,8,6,2,5,4,8,3,7]
输出：49

```
class Solution {
    public int maxArea(int[] height) {
        int res = 0;
        for(int i = 0; i < height.length - 1; i++)
            for(int j = i + 1; j < height.length; j++)
                res = Math.max(Math.min(height[i], height[j]) * (j - i), res);

        return res;
    }
}
```

初始思路

一定是暴力解法，但是提交系统中，时间复杂度不符合要求

```java
public static int solution1(int[] height){

    List<Integer> list = new ArrayList<>();
    for(int i=0;i<height.length;i++){
        for(int j=i+1;j<height.length;j++){
            int min = Math.min(height[i],height[j]);
            list.add(min*(j-i));
        }
    }
    Collections.sort(list);
    return list.get(list.size()-1);
}
```

优化解法

快慢指针，一个在头，一个在尾

迭代条件是：如果碰到下一个的比自己高，那就迭代，反之不动。

如果都没碰到，防止死循环，主动让j--

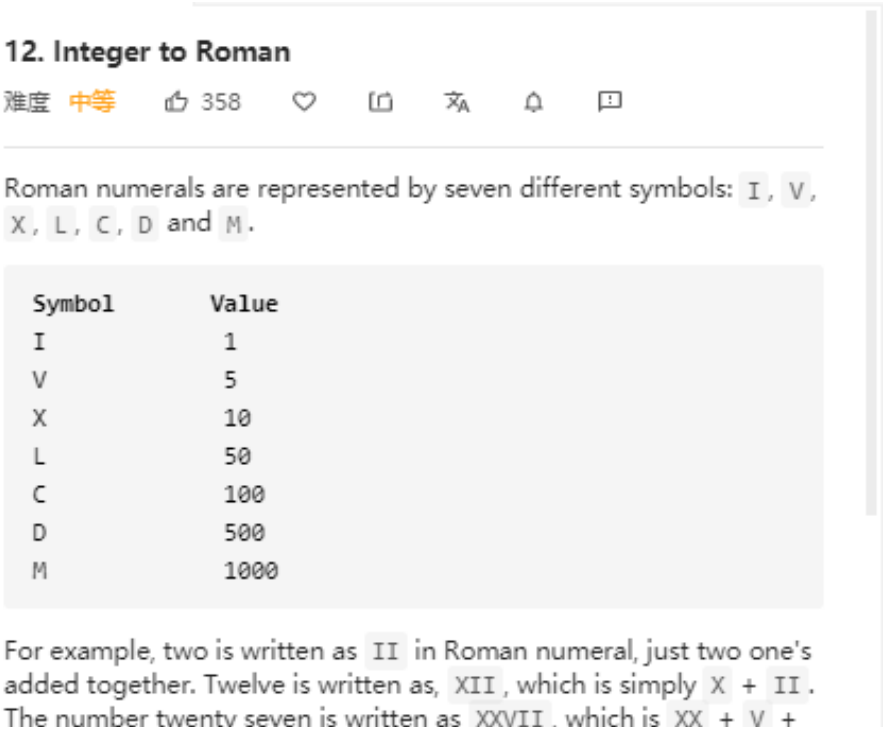
```java
public static int maxArea(int[] height) {
    public int maxArea( int[] height){
        int i = 0;
        int j = height.length - 1;
        int max = (j - i) * Math.min(height[i], height[j]);
        while (i < j) {
            if (height[j] < height[i]) {
                int res = (j - 1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

思路：

- **算法流程：** 设置双指针 i,j 分别位于容器壁两端，根据规则移动指针（后续说明），并且更新面积最大值 `res`，直到 `i == j` 时返回 `res`。
- **指针移动规则与证明：** 每次选定围成水槽两板高度 $h[i]$,$h[j]$ 中的短板，向中间收窄 1 格。以下证明：
  - 设每一状态下水槽面积为 $S(i,j)$,$(0 <= i < j < n)$，由于水槽的实际高度由两板中的短板决定，则可得面积公式 $S(i,j) = min(h[i],h[j]) \times (j-i)$。
  - 在每一个状态下，无论长板或短板收窄 1 格，都会导致水槽 **底边宽度** -1：
    - 若向内移动短板，水槽的短板 $min(h[i],h[j])$ 可能变大，因此水槽面积 $S(i,j)$ 可能增大。
    - 若向内移动长板，水槽的短板 $min(h[i],h[j])$ 不变或变小，下个水槽的面积一定小于当前水槽面积。
  - 因此，向内收窄短板可以获取面积最大值。换个角度理解：
    - 若不指定移动规则，所有移动出现的 $S(i,j)$ 的状态数为 $C(n,2)$，即暴力枚举出所有状态。
    - 在状态 $S(i,j)$ 下向内移动短板至 $S(i+1,j)$（假设 $h[i] < h[j]$），则相当于消去了 $S(i,j-1),S(i,j-2),...,S(i,i+1)$ 状态集合。而所有消去状态的面积一定 $<= S(i,j)$：
      - 短板高度：相比 $S(i,j)$ 相同或更短（$<= h[i]$）；
      - 底边宽度：相比 $S(i,j)$ 更短。
    - 因此**所有消去的状态的面积都** $< S(i,j)$。通俗的讲，我们每次向内移动短板，所有的消去状态都**不会导致丢失面积最大值**。
- **复杂度分析：**

https://leetcode-cn.com/problems/container-with-most-water/solution/container-with-most-water-shuang-zhi-zhen-fa-yi-do/

12_整数转换罗马数字



12. Integer to Roman

难度 中等 358

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

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

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

`II`.

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

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

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

Example 1:

```
Input: 3
Output: "III"
```

Example 2:

```
Input: 4
Output: "IV"
```

Example 3:

```
Input: 9
Output: "IX"
```

Example 4:

```
Input: 58
Output: "LVIII"
Explanation: L = 50, V = 5, III = 3.
```

Example 5:

```
Input: 1994
```

执行结果: 通过 显示详情 >

执行用时: 9 ms , 在所有 Java 提交中击败了 19.29% 的用户

内存消耗: 39.9 MB , 在所有 Java 提交中击败了 5.55% 的用户

炫耀一下:

```
1 public String intToRoman(int num) {
```

```java
    List<String> path = new LinkedList<>();
    int thousand = num / 1000;
    int hundred = (num - thousand*1000) / 100;
    int tenth = (num - thousand*1000 - hundred *100)/10;
    int single = num % 10;
    HashMap<Integer, String> map = new HashMap<>();
    makeDict(map);

    if(single == 4 || single == 9)
        path.add(map.get(single));
    else if(single == 5)
        path.add(map.get(single));
    else
    {
        StringBuilder sb = new StringBuilder();
        if(single > 4)
        {
            sb.append("V");
            single -= 5;
        }
        for(int i = 0; i < single; i++)
            sb.append("I");
        path.add(sb.toString());
    }

    if(tenth == 4 || tenth == 9)
        path.add(map.get(tenth*10));
    else if(tenth == 5)
        path.add(map.get(tenth*10));
    else
    {
        StringBuilder sb = new StringBuilder();
        if(tenth > 4)
        {
            tenth -= 5;
            sb.append("L");
        }
        for(int i = 0; i < tenth; i++)
            sb.append("X");
        path.add(sb.toString());
    }

    if(hundred == 4 || hundred == 9)
        path.add(map.get(hundred*100));
    else if(hundred == 5)
        path.add(map.get(hundred*100));
    else
    {
        StringBuilder sb = new StringBuilder();
```

```
        if(hundred > 4)
        {
            hundred -= 5;
            sb.append("D");
        }
        for(int i = 0; i < hundred; i++)
            sb.append("C");
        path.add(sb.toString());
    }

    if(thousand != 0)
    {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < thousand; i++) {
            sb.append("M");
        }
        path.add(sb.toString());
    }

    StringBuilder res=  new StringBuilder();
    for(int i = path.size() - 1; i >= 0; i--)
        res.append(path.get(i));
    return res.toString();

    //        System.out.println(single+ " " + tenth+ " " + hundred+ " " +
thousand);

}

private void makeDict(HashMap<Integer, String> map)
{
    map.put(1, "I");
    map.put(5, "V");
    map.put(10, "X");
    map.put(50, "L");
    map.put(100, "C");
    map.put(500, "D");
    map.put(1000, "M");

    map.put(4, "IV");
    map.put(9, "IX");
    map.put(40, "XL");
    map.put(90, "XC");
    map.put(400, "CD");
    map.put(900, "CM");
}
```

思路采用贪心算法，刚接触这种题难免会有bug。

这种解法真巧妙

```java
int[] values = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
String[] symbols = {"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};

public String intToRoman(int num) {
    StringBuilder sb = new StringBuilder();
    // Loop through each symbol, stopping if num becomes 0.
    for (int i = 0; i < values.length && num >= 0; i++) {
        // Repeat while the current symbol still fits into num.
        while (values[i] <= num) {
            num -= values[i];
            sb.append(symbols[i]);
        }
    }
    return sb.toString();
}
```

```java
/*
  回溯法
*/
```

13_罗马数字转整数

13. Roman to Integer

难度 简单 943

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

执行结果: 通过 显示详情 >

执行用时: 11 ms , 在所有 Java 提交中击败了 13.76% 的用户

内存消耗: 40.1 MB , 在所有 Java 提交中击败了 5.56% 的用户

炫耀一下:

```java
/*
  这个解法比下面的回溯法会好一些
  结构清晰，严谨且代码易读
*/
public int romanToInt(String s) {

    int index = 0;
    int res = 0;
    HashMap<String, Integer> map = new HashMap<>();
```

```
    makeDict(map);

    while(index < s.length())
    {
        if(index + 2 <= s.length() && map.containsKey(s.substring(index,index+2)))
        {
            res += map.get(s.substring(index, index+2));
            index += 2;
            continue;
        }
        if(map.containsKey(s.substring(index, index+1)))
        {
            res += map.get(s.substring(index, index+1));
            index++;
        }
    }
    return res;
}

private void makeDict(HashMap<String, Integer> map)
{
    map.put("I",1);
    map.put("V",5);
    map.put("X",10);
    map.put("L",50);
    map.put("C",100);
    map.put("D",500);
    map.put("M",1000);

    map.put("IV", 4);
    map.put("IX", 9);
    map.put("XL",40);
    map.put("XC",90);
    map.put("CD",400);
    map.put("CM",900);
}
```

```
/*
  回溯法解
  执行用时：660 ms, 在所有 Java 提交中击败了5.22%的用户
  内存消耗：40.4 MB, 在所有 Java 提交中击败了5.56%的用户
*/
    public int res;
    public HashMap<String, Integer> map = new HashMap<>();
    StringBuilder sample = new StringBuilder();
    List<Integer> list = new ArrayList<>();
    String s;
```

```java

    public int romanToInt(String s) {
        makeDict(map);
        this.res = 0;
        this.s = s;
        char[] charArray = s.toCharArray();
        StringBuilder path = new StringBuilder();
        List<Integer> jar  = new ArrayList<>();
        StringBuilder sample = new StringBuilder();
        backtrack(charArray,  0, jar);

        int min = list.get(0);
        /*
          注意这里，  只取可能结果的最小值
        */
        for(Integer i : list)
            min = Math.min(min, i);
        return min;
    }

    private void backtrack(char[] charArray,  int start, List<Integer> jar) {
        if(start > charArray.length)        return;
        if(start == charArray.length && sample.toString().equals(s))
        {
            //有很多解释可能，只取最小值
            res = 0;
            for(Integer i : jar)
                res += i;
            list.add(res);
            return;
        }

        for(int i = start; i < charArray.length; i++)
        {
            StringBuilder path = new StringBuilder();
            for(int j = i; j <= i+1; j++)
            {
                if(j == charArray.length)       return;
                path.append(charArray[j]);

                if(map.containsKey(path.toString()))
                {
                    jar.add(map.get(path.toString()));
                    sample.append(path);

                    backtrack(charArray, j+1, jar);

                    jar.remove(map.get(path.toString()));
                    sample.setLength(sample.length() - path.length());
```

```
                }
            }

        }
    }
    private void makeDict(HashMap<String, Integer> map)
    {
        map.put("I",1);
        map.put("V",5);
        map.put("X",10);
        map.put("L",50);
        map.put("C",100);
        map.put("D",500);
        map.put("M",1000);

        map.put("IV", 4);
        map.put("IX", 9);
        map.put("XL",40);
        map.put("XC",90);
        map.put("CD",400);
        map.put("CM",900);
    }
```

14_最长公共前缀

执行结果： 通过 显示详情 >

执行用时： 1 ms ，在所有 Java 提交中击败了 86.67% 的用户

内存消耗： 36.2 MB ，在所有 Java 提交中击败了 96.48% 的用户

炫耀一下:

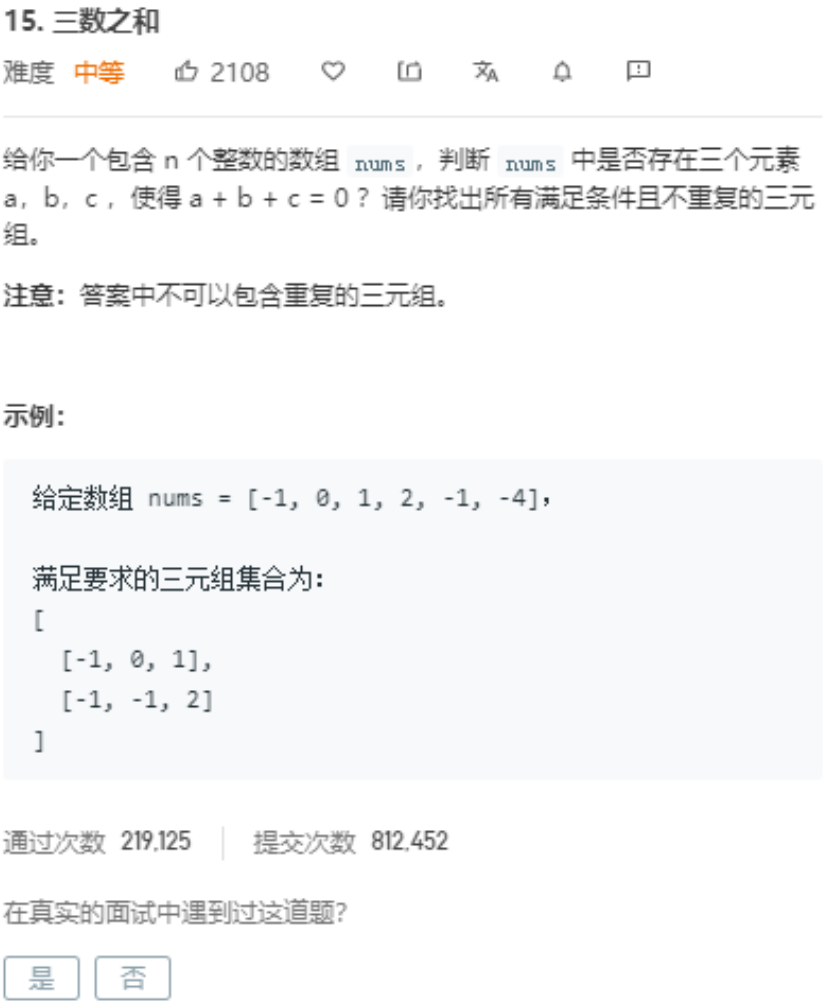
```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if(strs.length == 0)    return "";
        String minStr = strs[0];
        for(String str : strs)
            if(str.length() < minStr.length())
                minStr = str;

        String res = "";

        int index = 0;
        while(index < minStr.length()){
            for(int i = 0; i < strs.length; i++){
```

```
                if(minStr.charAt(index) != strs[i].charAt(index))
                    return minStr.substring(0, index);
            }
            index++;
        }


        return minStr.substring(0, index);
    }
}
```

思路：就是简单的匹配，并没有用算法

```
public String longestCommonPrefix(String[] strs) {
    if(strs.length == 0 || (strs.length == 1 && strs[0] == "")){
        return "";
    }
    //上面的代码防止意外情况发生
    //下面开始拿到最小的长度，和最小的字符串，因为最后的String不可能比他小

    int minLength = strs[0].length();
    int minIndex = 0;
    for (int i = 0; i < strs.length; i++) {
        if (strs[i].length() < minLength) {
            minIndex = i;
            minLength = strs[i].length();
        }

    }
    boolean flag = true;
    StringBuilder sb = new StringBuilder();
    //对字符数组中剩余元素进行匹配
    for (int i = 0; i < minLength; i++) {
        char a = strs[minIndex].charAt(i);
        for (int j = 0; j < strs.length; j++) {
            if (a != strs[j].charAt(i)) {
                flag = false;
                break;
            }
        }
        if (flag) {
            sb.append(a);
        }
    }
    return sb.toString();
}
```

15_threeSum



15. 三数之和

难度 中等 2108

给你一个包含 n 个整数的数组 `nums`，判断 `nums` 中是否存在三个元素 a，b，c，使得 a + b + c = 0？请你找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。

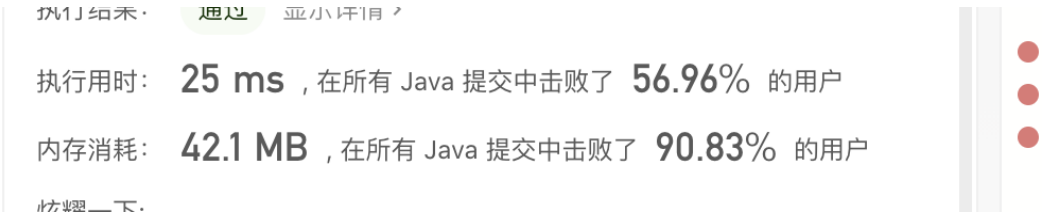
示例：

```
给定数组 nums = [-1, 0, 1, 2, -1, -4],

满足要求的三元组集合为:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

通过次数 219,125 | 提交次数 812,452

在真实的面试中遇到过这道题?

是 否



执行用时： 25 ms ，在所有 Java 提交中击败了 56.96% 的用户

内存消耗： 42.1 MB ，在所有 Java 提交中击败了 90.83% 的用户

```java
//更加快的方法
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res=  new ArrayList<>();

        for(int i = 0; i < nums.length - 2; i++){
            if(i > 0 && nums[i] == nums[i-1])   continue;

            int l = i + 1, r = nums.length - 1;
            while(l < r){
```

```
                int sum = nums[i] + nums[l] + nums[r];

                if(sum == 0){
                    res.add(Arrays.asList(nums[i], nums[l], nums[r]));
                    while(l < r && nums[l] == nums[l + 1]) l++;
                    while(l < r && nums[r] == nums[r - 1]) r--;
                    l++; r--;
                }else if(sum > 0)
                    r--;
                else
                    l++;
            }
        }


        return res;
    }

```


```

class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        HashMap<Integer, List<Integer>> map = new HashMap<>();
        HashSet<List<Integer>> filter = new HashSet<>();

        for(int i = 0; i < nums.length; i++){
            if(!map.containsKey(nums[i]))
                map.put(nums[i], new ArrayList<>());

              //这里进行优化
          //[0,0,0,0,0,0,0,0,0]
            if(map.get(nums[i]).size() <= 2)
                map.get(nums[i]).add(i);
        }

        for(int i = 0; i < nums.length - 2; i++)
            for(int j = i + 1; j < nums.length - 1; j++)
```

```
                if(map.containsKey(-(nums[i]+nums[j]))){
                    List<Integer> path = map.get(-(nums[i] + nums[j]));
                    for(Integer index : path)
                        if(index > i && index > j){
                            List<Integer> temp = new ArrayList<Integer>();
                            temp.add(nums[i]); temp.add(nums[j]);
temp.add(nums[index]);
                            Collections.sort(temp);
                            filter.add(temp);
                            break;
                        }

                }
        return new ArrayList<>(filter);
    }

}
```

时间复杂度为n2

```
public static List<List<Integer>> threeSum1(int[] nums){
        List<List<Integer>> lists = new ArrayList<>();
        //排序
        Arrays.sort(nums);
        //双指针
        int len = nums.length;
        for(int i = 0;i < len;++i) {
            //如果第一个元素上来就是大于0的,
            //直接没戏，因为我们已经对于nums排好序了
            if(nums[i] > 0) return lists;

            if(i > 0 && nums[i] == nums[i-1]) continue;
       //排除重复情况
            //如果这一个元素和下一个元素一样，那就可能有重复的
            int curr = nums[i];
            int L = i+1, R = len-1;
            //从fixed元素开始，一个往fixed +1 ，一个从末尾往前走
            while (L < R) {
                int tmp = curr + nums[L] + nums[R];
                if(tmp == 0) {
                    List<Integer> list = new ArrayList<>();
                    list.add(curr);
                    list.add(nums[L]);
                    list.add(nums[R]);
                    lists.add(list);
                    //排除重复元素
```

```
                    while(L < R && nums[L+1] == nums[L]) ++L;
                    while (L < R && nums[R-1] == nums[R]) --R;
                    ++L;
                    --R;
                } else if(tmp < 0) {
                    ++L;
                } else {
                    --R;
                }
            }
        }
        return lists;
}
```

16_threeSumClosest

16. 3Sum Closest

难度 中等 424

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

Example:

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

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

通过次数 100,701 | 提交次数 228,837

执行结果: 通过 显示详情 >

执行用时: 6 ms , 在所有 Java 提交中击败了 81.97% 的用户

内存消耗: 38.1 MB , 在所有 Java 提交中击败了 65.88% 的用户

炫耀一下:

```
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int res = Integer.MAX_VALUE - target;

```

```
        for(int i = 0; i < nums.length - 2; i++){
            int l = i + 1, r = nums.length - 1;

            while(l < r){
                int sum = nums[i] + nums[l] + nums[r];
                if(sum == target){
                    return sum;
                }else if(sum > target){
                   if(Math.abs(sum - target) < Math.abs(res - target))
                        res = sum;
                    r--;
                }else{
                    if(Math.abs(sum - target) < Math.abs(res - target))
                        res = sum;
                    l++;
                }
            }
        }

        return  res;
    }

}
```

与上一题异曲同工之妙

```
public static int threeSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    int ans = nums[0] + nums[1] + nums[2];
    for(int i=0;i<nums.length;i++) {
        int start = i+1, end = nums.length - 1;
        while(start < end) {
            int sum = nums[start] + nums[end] + nums[i];
            if(Math.abs(target - sum) < Math.abs(target - ans))
                ans = sum;
            if(sum > target)
                end--;
            else if(sum < target)
                start++;
            else
                return ans;
        }
    }
    return ans;
```

```
}
```

注意，游标是从0-leng-1 出发，同时快慢指针均位于游标右侧，同时进行迭代

17_LetterCombinationOfPhoneNumber

17. Letter Combinations of a Phone Number

难度 中等 709

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example:

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

Note:

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

```java
//典型回溯思路，  秒了
class Solution {
    HashMap<Integer, String> map = new HashMap<>();
    public List<String> letterCombinations(String digits) {
        List<String> res = new ArrayList<>();
        if(digits.equals(""))   return res;
        //1. map different numbers


        //2. for each digits, dfs them
        map.put(2, "abc");
        map.put(3, "def");
        map.put(4, "ghi");
```

```
        map.put(5, "jkl");
        map.put(6, "mno");
        map.put(7, "pqrs");
        map.put(8, "tuv");
        map.put(9, "wxyz");

        dfs(res, digits, 0, new StringBuilder());
        return res;
    }

    private void dfs(List<String> res, String digits, int curPos, StringBuilder
path) {
        if(curPos == digits.length()){
            res.add(path.toString());
            return;
        }


        String str = map.get(digits.charAt(curPos) - '0');
        for(int i = 0; i < str.length(); i++){
            path.append(str.charAt(i));

            dfs(res, digits, curPos + 1, path);

            path.setLength(path.length() - 1);
        }
    }
}
```

这个题很好的解释了为什么可以循环的地方一定可以用递归，但是可以用递归的地方不一定能用循环。

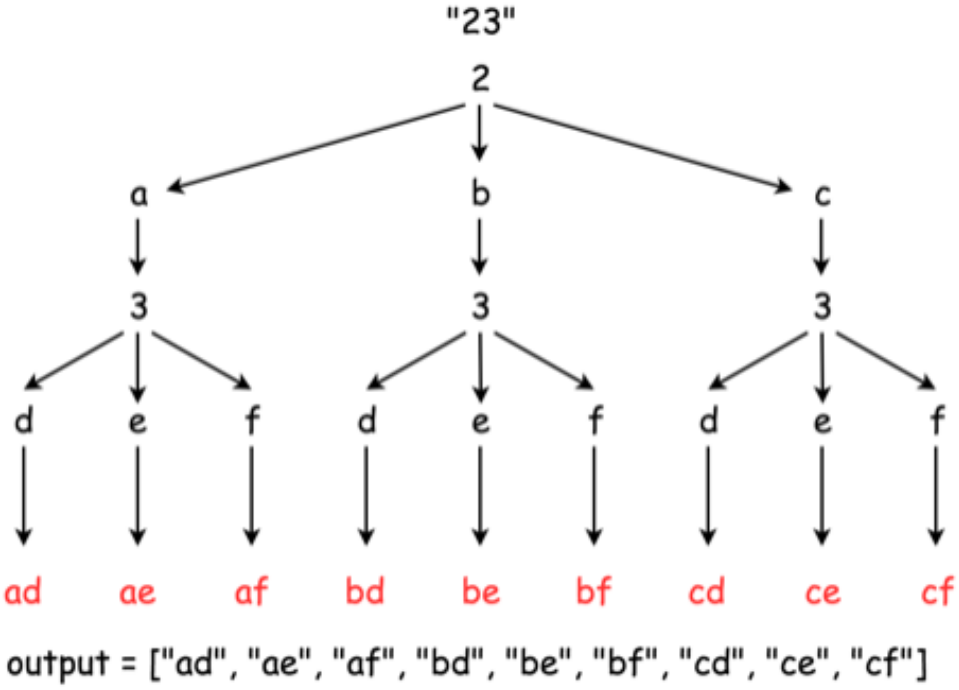
因为在本题中，无法确定for具体循环几轮，说的再细一点，就是比如有3个字符比如"132"，那么就循环3次进行求解，有4个字符比如"3456"，就需要循环四次进行求解。显然这个循环体是不太好进行确定的

因此这个地方采用递归的方式来解决

方法一：递归调用解决循环无法具象化问题

```
class Solution {
    //一个映射表，第二个位置是"abc“,第三个位置是"def"。。。
    //这里也可以用map，用数组可以更节省点内存
    String[] letter_map = {"
","*","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
    public List<String> letterCombinations(String digits) {
        //注意边界条件
        if(digits==null || digits.length()==0) {
```

```
            return new ArrayList<>();
        }
        iterStr(digits, "", 0);
        return res;
    }
    //最终输出结果的list
    List<String> res = new ArrayList<>();

    //递归函数
    void iterStr(String str, String letter, int index) {
        //递归的终止条件，注意这里的终止条件看上去跟动态演示图有些不同，主要是做了点优化
        //动态图中是每次截取字符串的一部分，"234"，变成"23"，再变成"3"，最后变成""，这样性能不佳
        //而用index记录每次遍历到字符串的位置，这样性能更好
        if(index == str.length()) {
            res.add(letter);
            return;
        }
        //获取index位置的字符，假设输入的字符是"234"
        //第一次递归时index为0所以c=2，第二次index为1所以c=3，第三次c=4
        //subString每次都会生成新的字符串，而index则是取当前的一个字符，所以效率更高一点
        char c = str.charAt(index);
        //map_string的下表是从0开始一直到9， c-'0'就可以取到相对的数组下标位置
        //比如c=2时候，2-'0'，获取下标为2,letter_map[2]就是"abc"
        int pos = c - '0';
        String map_string = letter_map[pos];
        //遍历字符串，比如第一次得到的是2，页就是遍历"abc"
        for(int i=0;i<map_string.length();i++) {
            //调用下一层递归，用文字很难描述，请配合动态图理解
            iterStr(str, letter+map_string.charAt(i), index+1);
            //以字符串"23"举例，在进行‘2’的循环时，letter == "", mapString.charAt(i) = ‘a’ 'b' 'c'
            //并由此，将其传递下去作为新一轮循环的letter,
            //同时，新一轮的map_string。charAt(i)就代表的是 3 对应的字符集 'd' 'e' 'f'
            //并进行新一轮的组合，ab, ae, af, bd,be,bf
            //当再进入下一轮的时候，发现Index == length 直接将现有的Letter进行放入res中并返回
            //结束计算
            //本过程十分类似树结构，把它放到下图中
        }
    }
}
```



方案二：采用队列

和递归道理类似，但是利用了队列的数据结构

以 "23 " 为例

首先往队列里面放入一个空字符串，然后进行放入2对应的"abc"

之后每次取出一个字符，并和3 对应的"def"进行组合，最终拿到一个结果

```
class Solution {
    public List<String> letterCombinations(String digits) {
        if(digits==null || digits.length()==0) {
            return new ArrayList<String>();
        }
        //一个映射表，第二个位置是"abc“,第三个位置是"def"。。。
        //这里也可以用map，用数组可以更节省点内存
        String[] letter_map = {
                " ","*","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"
        };
        List<String> res = new ArrayList<>();
        //先往队列中加入一个空字符
        res.add("");
        for(int i=0;i<digits.length();i++) {
            //由当前遍历到的字符，取字典表中查找对应的字符串
            String letters = letter_map[digits.charAt(i)-'0'];
            int size = res.size();
            //计算出队列长度后，将队列中的每个元素挨个拿出来
            for(int j=0;j<size;j++) {
                //每次都从队列中拿出第一个元素
```

```
                String tmp = res.remove(0);
                //然后跟"def"这样的字符串拼接，并再次放到队列中
                for(int k=0;k<letters.length();k++) {
                    res.add(tmp+letters.charAt(k));
                }
            }
        }
        return res;
    }
}
```

18_fourSum

18. 4Sum

难度 中等 461

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

Note:

The solution set must not contain duplicate quadruplets.

Example:

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

A solution set is:
[
  [-1,  0, 0, 1],
  [-2, -1, 1, 2],
  [-2,  0, 0, 2]
]
```

执行结果： 通过 显示详情 >

执行用时： 19 ms , 在所有 Java 提交中击败了 42.81% 的用户

内存消耗： 38.8 MB , 在所有 Java 提交中击败了 79.70% 的用户

炫耀一下:

```
```

```java
class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        Arrays.sort(nums);
        List<List<Integer>> res = new ArrayList<>();

        for(int i = 0; i < nums.length - 3; i++){
            if(i > 0 && nums[i] == nums[i - 1])
                continue;
            for(int j = i + 1; j < nums.length - 2; j++){
                if(j > i + 1 && nums[j] == nums[j - 1])
                    continue;
                int l = j + 1, r = nums.length - 1;

                while(l < r){
                    int sum = nums[i] + nums[j] + nums[l] + nums[r];
                    if(sum == target) {
                        res.add(Arrays.asList(nums[i], nums[j], nums[l], nums[r]));
                        while(l < r && nums[l] == nums[l + 1]) l++;
                        while(l < r && nums[r] == nums[r - 1]) r--;

                        l++; r--;
                    }else if(sum > target)
                        r--;
                    else
                        l++;
                }
            }
        }

        return res;

    }
}
```

思路十分类似threeSum,只不过是每次固定两个数字，同时注意数组下标不要越界

以及特殊情况比如{0,0,0,0,0,0,0,0,0,0}等的处理

```
public static List<List<Integer>> fourSum(int[] nums, int target) {
    List<List<Integer>> res = new ArrayList<>();
    if(nums == null || nums.length<4){
        return res;
    }
    //-2 -1 0 0 1 2
    //[-1,  0, 0, 1],
    // [-2, -1, 1, 2],
    // [-2,  0, 0, 2]
    // 0 ~ nums.length -1  -1 -1 -1

    Arrays.sort(nums);

    for (int i = 0; i <= nums.length - 4; i++) {
        for (int j = i + 1; j <= nums.length - 3; j++) {
            int leftIndex = j + 1;
            int rightIndex = nums.length - 1;
            while (leftIndex<=nums.length-2 && leftIndex < rightIndex) {

                if (nums[i] + nums[j] + nums[leftIndex] + nums[rightIndex] >
target) {
                    rightIndex--;
                } else if (nums[i] + nums[j] + nums[leftIndex] + nums[rightIndex] <
target) {
                    leftIndex++;
                } else {
                    List<Integer> list = new ArrayList<>();
                    list.add(nums[i]);
                    list.add(nums[j]);
                    list.add(nums[leftIndex]);
                    list.add(nums[rightIndex]);
                    res.add(list);
```

```
                    while (leftIndex < rightIndex && nums[rightIndex] ==
nums[rightIndex - 1]) {
                        rightIndex--;
                    }
                    while (leftIndex < rightIndex && nums[leftIndex] ==
nums[leftIndex + 1]) {
                        leftIndex++;
                    }
                    rightIndex--;
                }
            }
            while(j+1<nums.length && nums[j] == nums[j+1]){
                j++;
            }
        }
        while(i+1<nums.length && nums[i] == nums[i+1]){
            i++;
        }
    }

    return res;
}
```

19_RemoveNth Node from end of the list

19. Remove Nth Node From End of List

难度 中等 816

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

Example:

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

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

Note:

Given n will always be valid.

Follow up:

Could you do this in one pass?

通过次数 169,741 | 提交次数 439,966

执行结果： 通过 显示详情 >

执行用时： 0 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 36.2 MB , 在所有 Java 提交中击败了 73.71% 的用户

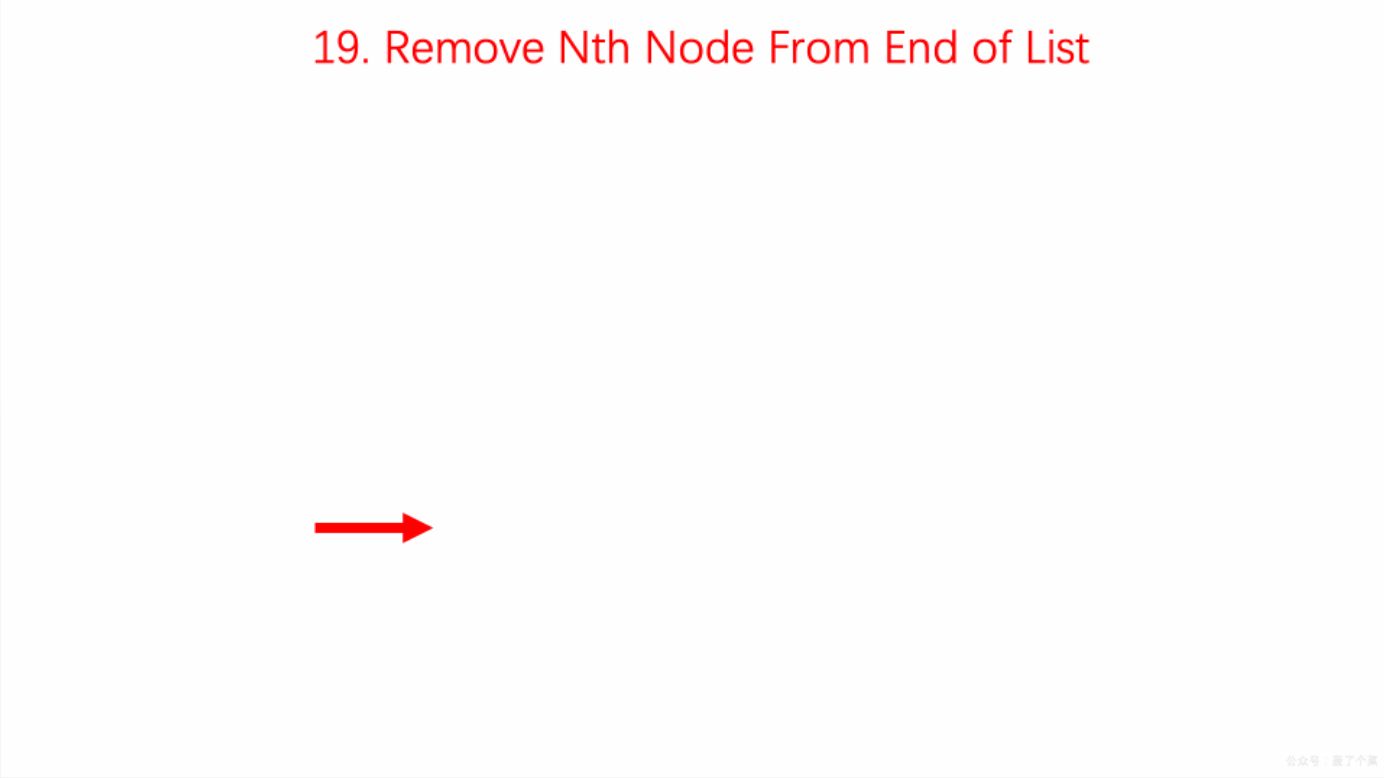
炫耀一下:

```java
class Solution {
    int res = 0;
    public ListNode removeNthFromEnd(ListNode head, int n) {
        res = n - 1;

        head = helper(head, n);

        return head;
    }

    public ListNode helper(ListNode head, int n){
        if(head == null)        return null;

        head.next = helper(head.next, n);
//        System.out.println(res);
        if(res == 0) {
            res = Integer.MAX_VALUE;
            return head.next;
        }

        res--;

        return head;
    }
}
```


```java
public static ListNode removeNthFromEnd(ListNode head, int n) {
    if(head == null)
        return null;

    ListNode cur = head;
    List<ListNode> list = new ArrayList<>();
    while(cur!=null){
        list.add(cur);
        cur = cur.next;
    }
    list.remove(list.size()-n);
    if(list.size() == 0){
        return null;
    }
    ListNode newHead =  new ListNode(list.get(0).val);
    list.remove(0);
    cur = newHead;

    while(list.size()!=0){
        ListNode newNode = new ListNode(list.get(0).val);
        cur.next = newNode;
        cur = newNode;
        list.remove(0);
    }
    return newHead;
}
```

常规链表题，取出倒数第n个节点，注意在做的时候，尽量新new一个节点，否则如果用原来的，会有next指向下一个的风险

巧妙办法:采用快慢指针，起始都在头结点，当相隔距离是n时，慢指针才会移动，直到快指针为空，去除慢指针的下一个元素




```cpp
class Solution {
    public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* dummyHead = new ListNode(0);
        dummyHead->next = head;

        ListNode* p = dummyHead;
        ListNode* q = dummyHead;
        for( int i = 0 ; i < n + 1 ; i ++ ){
            q = q->next;
        }

        while(q){
            p = p->next;
            q = q->next;
        }

        ListNode* delNode = p->next;
        p->next = delNode->next;
        delete delNode;

        ListNode* retNode = dummyHead->next;
        delete dummyHead;

```

```
25          return retNode;
26
27      }
28  };
```

20_Valid Parentheses

20. Valid Parentheses

难度 简单 1573

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

An input string is valid if:

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

Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
```

执行结果： 通过 显示详情 >

执行用时： 2 ms , 在所有 Java 提交中击败了 85.09% 的用户

内存消耗： 37.9 MB , 在所有 Java 提交中击败了 5.48% 的用户

炫耀一下：

看到Parentheses 一定会想到栈的使用，匹配就OK

```
public boolean isValid(String s) {
    Map<Character,Character> map = new HashMap<>();
    map.put(')','(');
    map.put(']','[');
    map.put('}','{');

    if(s.length() == 0){
        return true;
    }
    char[] ch = s.toCharArray();
    Stack<Character> stack = new Stack<>();
    for(int i=0;i<ch.length;i++){
        if(stack.isEmpty()){
            stack.push(ch[i]);
        }else{
            if(ch[i] == '(' || ch[i] == '[' || ch[i] == '{'){
                stack.push(ch[i]);
            }else{
                //ch[i] == }
                char c = stack.pop();
                if(map.get(ch[i]) == c){

                }else{
                    return false;
                }
            }
        }

    }
    return stack.size() == 0;
}
```

021 merge two sorted linked list

常规题型，没什么好说的

```java
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode curl1 = l1;
    ListNode curl2 = l2;
    ListNode newHead = new ListNode(0);
    ListNode newCur = newHead;
    if(l1 == null && l2 == null){
        return null;
    }
    if(l1 == null)
        return l2;
    if(l2 == null)
        return l1;

    while(curl1 != null && curl2 != null){
        if(curl1.val <= curl2.val){
            ListNode newNode  = new ListNode(curl1.val);
            newCur.next = newNode;
            newCur = newNode;
            curl1 = curl1.next;
        }else{
```

```
            ListNode newNode  = new ListNode(curl2.val);
            newCur.next = newNode;
            newCur = newNode;
            curl2 = curl2.next;
        }
    }
    if (curl1 !=null){
        newCur.next= curl1;
    }
    if(curl2 != null){
        newCur.next = curl2;
    }

    return newHead.next;
}
```

022 Generate Parentheses (Recursion应用)

22. Generate Parentheses

难度 中等 1022

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:

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

通过次数 127,666 | 提交次数 169,357

在真实的面试中遇到过这道题?

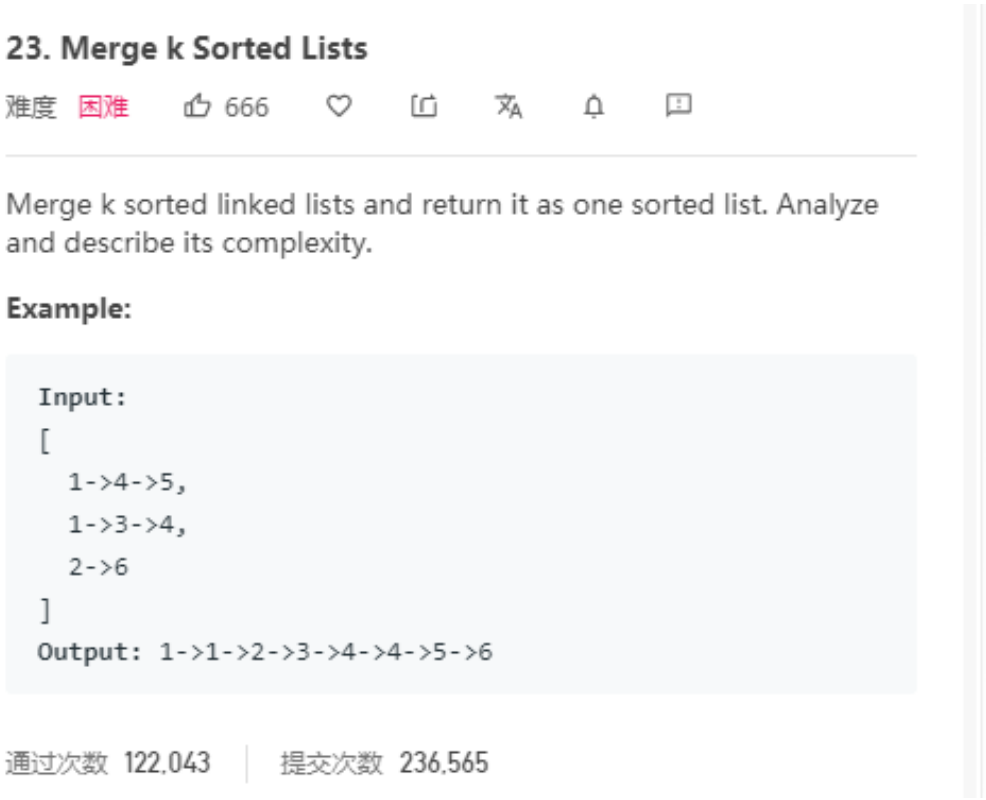
是 否

真巧妙

```
public List<String> generateParenthesis(int n) {
    List<String> res = new ArrayList<String>();
    int lc= 0;
    int rc = 0;
```

```
    //lc 代表左括号， rc 代表右括号
    dfs(res, "" , n, lc, rc);
    return res;
}
//采用深度遍历的方法
public void dfs(List<String> res, String path, int n, int lc, int rc){
     //如果右括号的数量比左括号多，或者两个已经全部遍历完毕，就返回
    if(lc > n || rc > n || rc > lc){
        return;
    }
  //如果找到合适的，就把它们放入res中
    if(lc == rc && lc == n){
        res.add(path);
        return;
    }

    dfs(res,path+'(',n,lc+1,rc);
    dfs(res,path+')',n,lc,rc+1);
}
```

023 Merged K sorted linkedlist



23. Merge k Sorted Lists

难度 困难 666

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

Example:

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

通过次数 122,043 | 提交次数 236,565

思路极其类似两个有序链表的合并，就是多了一步类似赫夫曼树构建时的相似动作

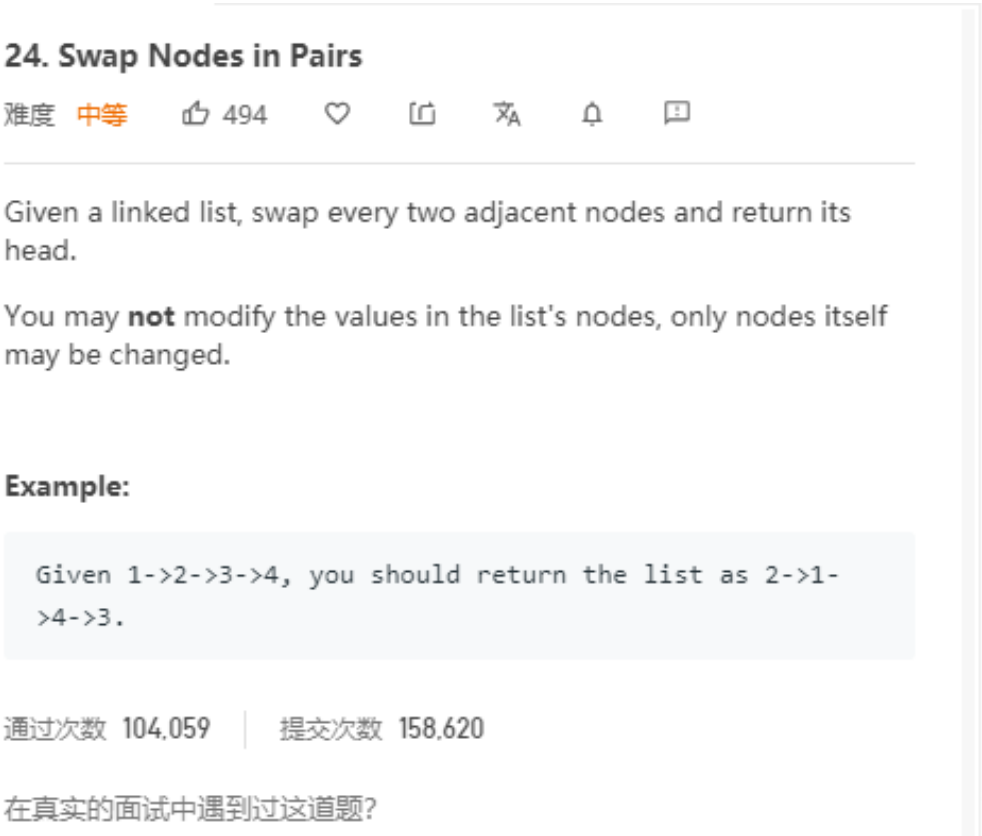
```java
public static ListNode mergeKLists(ListNode[] lists) {
        if (lists.length == 0)
            return null;

        if (lists.length == 1)
            return lists[0];
        //note the LLO < LLT
        ListNode newHead = new ListNode(0);

        List<ListNode> container = new ArrayList<>(Arrays.asList(lists));

        while (container.size() > 1) {
            ListNode headOne = container.get(0);
            ListNode headTwo = container.get(1);

            container.remove(headOne);
            container.remove(headTwo);

            newHead = merge(headOne, headTwo);
            container.add(newHead);
        }

        return newHead;
    }

    private static ListNode merge(ListNode headOne, ListNode headTwo) {
        ListNode newHead = new ListNode(0);

        if (headOne == null && headTwo == null)
            return null;
        if (headOne == null) {
            newHead.next = headTwo;
            return newHead.next;
        }
        if (headTwo == null) {
```

```
            newHead.next = headOne;
            return newHead.next;
        }
        ListNode curOne = headOne;
        ListNode curTwo = headTwo;
        ListNode curHead = newHead;
        while (curOne != null && curTwo != null) {
            if (curOne.val <= curTwo.val) {
                curHead.next = curOne;
                curHead = curOne;
                curOne = curOne.next;
            } else {
                curHead.next = curTwo;
                curHead = curTwo;
                curTwo = curTwo.next;
            }
        }

        if (curOne != null)
            curHead.next = curOne;

        if (curTwo != null)
            curHead.next = curTwo;

        return newHead.next;
}
```

```
/*
  归并排序
*/
public ListNode mergeKLists(ListNode[] lists) {
    if (lists == null || lists.length == 0) return null;
    return merge(lists, 0, lists.length - 1);
}

private ListNode merge(ListNode[] lists, int left, int right) {
    if (left == right) return lists[left];
    int mid = left + (right - left) / 2;
    ListNode l1 = merge(lists, left, mid);
    ListNode l2 = merge(lists, mid + 1, right);
    return mergeTwoLists(l1, l2);
}

private 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;
    } else {
        l2.next = mergeTwoLists(l1,l2.next);
        return l2;
    }
}

作者：powcai
链接：https://leetcode-cn.com/problems/merge-k-sorted-lists/solution/leetcode-23-he-bing-kge-pai-xu-lian-biao-by-powcai/
```

024 Swap nodes in Pairs



24. Swap Nodes in Pairs

难度 中等 494

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

You may **not** modify the values in the list's nodes, only nodes itself may be changed.

Example:

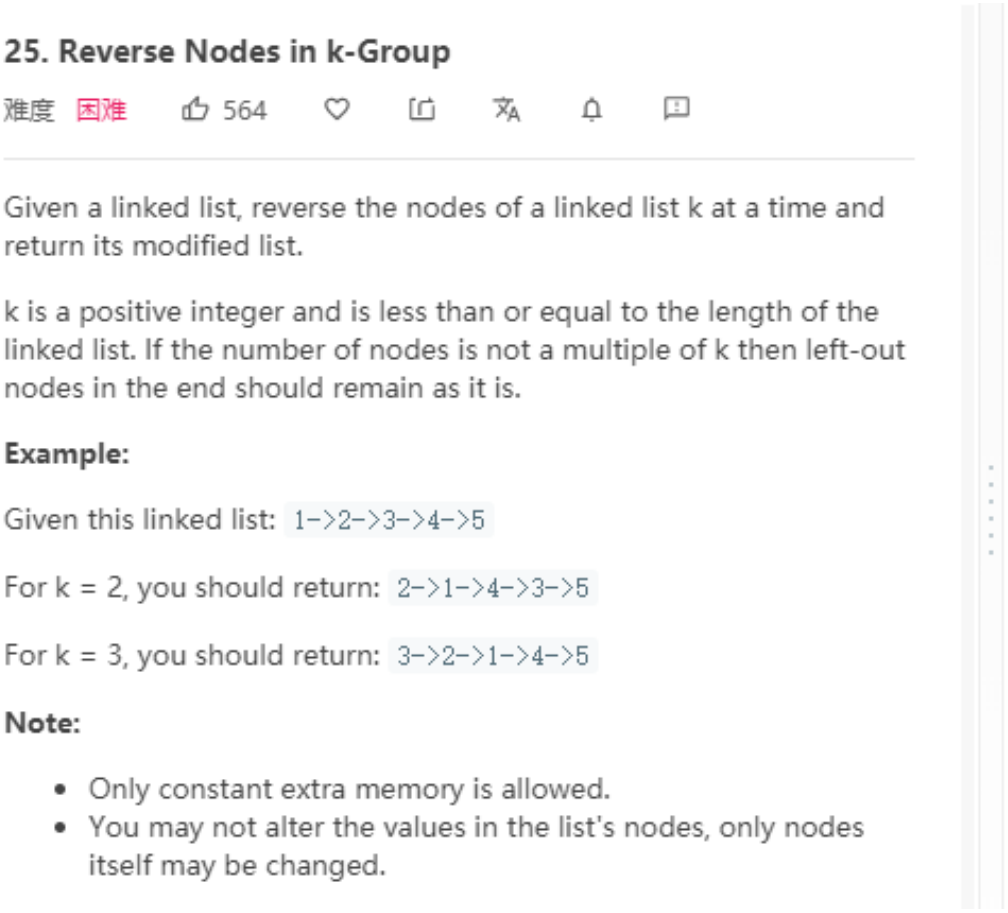
```
Given 1->2->3->4, you should return the list as 2->1->4->3.
```

通过次数 104,059 | 提交次数 158,620

在真实的面试中遇到过这道题?


```java
public static ListNode swapPairs(ListNode head) {
    if(head == null || head.next == null)
        return head;

    ListNode cur=head;
    ListNode post = head.next;
    ListNode pre;
    ListNode newHead = head.next;


    while (cur != null && post != null){
        pre = cur;
        cur.next = cur.next.next;
        post.next = cur;

        if(cur.next!=null){
            post = cur.next.next;
        }else{
            post = null;
        }
        cur = cur.next;
        if(post!=null){
            pre.next = post;
        }
        else{
            pre.next = cur;
        }
    }
    return newHead;
}
```

025 Reverse nodes in k-group



25. Reverse Nodes in k-Group

难度 困难 564

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

Example:

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

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

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

Note:

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

```java
//贡献一个我自己比较漂亮的解法
public ListNode reverseKGroup(ListNode head, int k) {
        if(head == null)
            return null;

        int count = 0;
        ListNode cur = head;
        while(cur != null){
            count++;
            cur = cur.next;
        }

        int turns = count / k;

        ListNode before = null, after = null;
        ListNode res = null;

        cur = head;
        for(int i = 0; i < turns; i++){
            ListNode pre = null;

            ListNode thisHead = null;
```

```
            ListNode thisTail = null;
            for(int j = 0; j < k; j++){
                if(j == 0)
                    thisTail = cur;
                if(j == k - 1)
                    thisHead = cur;

                ListNode temp = cur.next;
                cur.next      = pre;
                pre           = cur;

                cur = temp;
            }


            if(before == null && after == null){
                before = thisHead;
                after = thisTail;

                res = before;
            }else{
                after.next = thisHead;
                after = thisTail;
            }

            after.next = null;
        }

        after.next = cur;

        return res;
    }

```

执行结果: 通过 显示详情 >

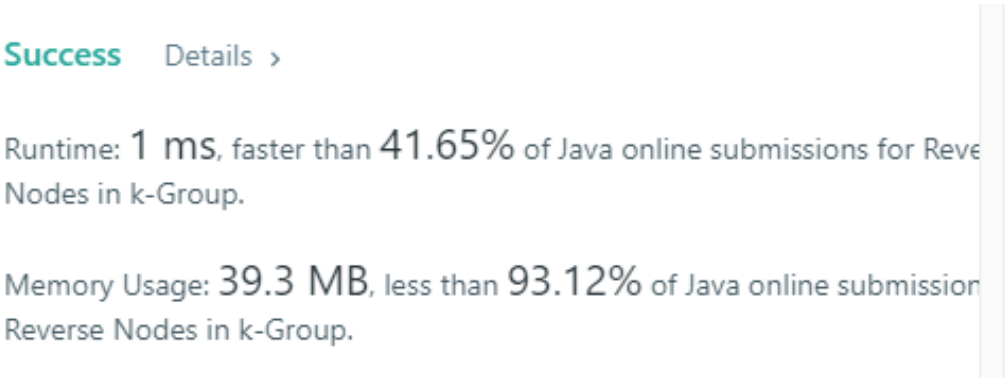
执行用时: 1 ms , 在所有 Java 提交中击败了 39.92% 的用户

内存消耗: 38.9 MB , 在所有 Java 提交中击败了 23.90% 的用户

炫耀一下:

```
    public ListNode reverseKGroup(ListNode head, int k) {
        int count = 0;

```

```
        ListNode cur = head;
        while(cur != null){
            count++;
            cur = cur.next;
        }

        //记录反转次数
        int numFlip = count / k;
        cur = head;
        ListNode res = head;

        ListNode start = cur, end = null;
        ListNode pre = null;
        for(int i = 0; i < numFlip; i++){
            if(end == null)
                end = cur;
            start = cur;

            //局部反转
            pre = null;
            int index = k;
            while(index != 0){
                ListNode temp = cur.next;
                cur.next = pre;
                pre = cur;
                cur = temp;

                index--;
            }
            //记录要返回的node
            if(i == 0)      res = pre;
            //拼接反转的部分
            if(i != 0){
                end.next = pre;
                end = start;
            }
        }

        if(cur != null)
            end.next = cur;

        return res;
    }
```




```java
class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {
        if(head == null || head.next == null)        return head;

        ListNode dummyHead = new ListNode(0);
        dummyHead.next = head;
        ListNode cur = head;
        int count = 0;
        while(cur != null)
        {
            cur = cur.next;
            count++;
        }

        int timesToFlip = count / k;

        cur= head;
        ListNode before = dummyHead;
        ListNode after  = null;
        for(int i = 0; i < timesToFlip; i++)
        {
            int numNodes = k;
            ListNode temp = null, pre = null;

            while(numNodes != 0)
            {
                numNodes--;

                temp = cur.next;
                cur.next = pre;
                pre = cur;
                cur = temp;
                after = cur;
            }

            before.next = pre;
            while(pre.next != null)
```

```
                pre = pre.next;
            pre.next = after;

            before = pre;
            cur = before.next;
        }

        return dummyHead.next;
    }
}
```

思路二： 尾插法 (后两种思路比较难理解，尤其涉及交换部分)

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode pre = dummy;
        ListNode tail = dummy;
        while (true) {
            int count = 0;
            while (tail != null && count != k) {
                count++;
                tail = tail.next;
            }
            if (tail == null) break;
            ListNode head = pre.next;
            //reverse 操作，就是尾插法，需要细致体会
            while (pre.next != tail) {
                ListNode cur = pre.next;
                pre.next = cur.next;
                cur.next = tail.next;
                tail.next = cur;
            }
            pre = head;
            tail = head;
        }
        return dummy.next;
    }
}
```

思路三： 递归操作

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode cur = head;
        int count = 0;
        while (cur != null && count != k) {
            cur = cur.next;
            count++;
        }
        if (count == k) {
            cur = reverseKGroup(cur, k);
            while (count != 0) {
                count--;
                ListNode tmp = head.next;
                head.next = cur;
                cur = head;
                head = tmp;
            }
            head = cur;
        }
        return head;
    }


```

026 Removed Duplicated Elements

26. Remove Duplicates from Sorted Array

难度 简单 1441

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

Do not allocate extra space for another array, you must do this by

modifying the input array in-place with O(1) extra memory.

Example 1:

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

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

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

Example 2:

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

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

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

Clarification:

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

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

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without
making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be
known by the caller.
// using the length returned by your function, it
prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

常规操作

```java
  public static int removeDuplicates(int[] nums) {
        int count = 0;
        int index = 0;
        int distinctElem = 0;
        while(index < nums.length){
            if(index+1 < nums.length &&nums[index] == nums[index+1]){
                index++;
            }else{
                nums[distinctElem++] = nums[index];
                count++;
                index++;
            }
        }
        System.out.println(Arrays.toString(nums));
        return count;

    }
```

```java
//2020 12 29
    public int removeDuplicates(int[] nums) {
        int left = 0, right = 0;
        int index = 0;

        while(right < nums.length){
            while(right < nums.length && nums[right] == nums[left])
                right++;
            nums[index++] = nums[left];
            left = right;
        }

        return index;
```

```
14      }
```

027 Remove Element

27. Remove Element

难度 简单 554

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

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

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

Example 1:

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

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

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

Example 2:

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

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

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

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

Clarification:

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

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

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a
copy)
int len = removeElement(nums, val);

// any modification to nums in your function would be known by
the caller.
// using the length returned by your function, it prints the
first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

通过次数 177,666 | 提交次数 305,407

在真实的面试中遇到过这道题? 是 否


```java
public static  int removeElement(int[] nums, int val) {
    int count =0;
    for(int i=0;i<nums.length;i++)
        if(nums[i] == val)
            count++;

    for(int i=0;i<nums.length;i++){
        int index = i;
        while(index <nums.length && index+1<nums.length&&nums[index] == val){
            index++;
        }
        if(nums[i] == val){
            int temp = nums[i];
            nums[i] = nums[index];
            nums[index] = temp;
```

```
        }
    }
    System.out.println(Arrays.toString(nums));
    return nums.length-count;
}
```

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        if (nums.empty()) return 0;
        int n = nums.size();

        int cur = 0;
        for (int i = 0; i < n; i ++) {
            if (nums[i] != val) {
                nums[cur] = nums[i];
                cur++;
            }
        }
        return cur;
    }
};
```

028 Implement strStr() --- KMP的典型应用

28. Implement strStr()

难度 简单 442

Implement strStr().

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

Example 1:

```
Input: haystack = "hello", needle = "ll"
Output: 2
```

Example 2:

```
Input: haystack = "aaaaa", needle = "bba"
Output: -1
```

Clarification:

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

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


```java
/*
  典型KMP 算法
*/
public int strStr(String haystack, String needle) {
    if(needle.length() == 0)      return 0;
    KMP kmp = new KMP(needle);
    return kmp.search(haystack);
}

public class KMP
{
    private String pat;
```

```
    private int[][] dfa;
    public KMP(String pat)
    {
        this.pat = pat;
        int M = pat.length();
        int R = 256;
        dfa = new int[R][M];
        dfa[pat.charAt(0)][0] = 1;

        for(int X = 0, j = 1; j < M; j++)
        {
            for(int c = 0; c < R; c++)
                dfa[c][j] = dfa[c][X];
            dfa[pat.charAt(j)][j] = j+1;
            X = dfa[pat.charAt(j)][X];
        }
    }

    public int search(String txt)
    {
        int i, j, N = txt.length(), M = pat.length();
        for(i = 0, j = 0; i < N && j < M; i++)
            j = dfa[txt.charAt(i)][j];
        if(j == M)      return i-M;
        else            return -1;
    }

}
```

029 Divide two integers

29. Divide Two Integers

难度 中等 337

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

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

The integer division should truncate toward zero, which means losing its fractional part. For example, `truncate(8.345) = 8` and `truncate(-2.7335) = -2`.

Example 1:

```
Input: dividend = 10, divisor = 3
Output: 3
Explanation: 10/3 = truncate(3.33333..) = 3.
```

Example 2:

```
Input: dividend = 7, divisor = -3
Output: -2
Explanation: 7/-3 = truncate(-2.33333..) = -2.
```

Note:

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

通过次数 46,905 | 提交次数 238,519

在真实的面试中遇到过这道题?

```go
func divide(dividend int, divisor int) int {
    if dividend == -2147483648 && divisor == -1{
        return 2147483647
    }
  sign := 1

  if (divisor > 0 && dividend < 0) || (dividend > 0 && divisor < 0){
    sign = -1
  }
```

```
10
11    if dividend > 0{
12      dividend = -dividend
13    }
14
15    if divisor > 0{
16      divisor = -divisor
17    }
18
19    res := getRes(dividend, divisor)
20    if sign == -1{
21      return -res
22    }
23
24    return res
25  }
26
27  func getRes(dividend int, divisor int) int{
28    if divisor < dividend{
29      return 0
30    }
31
32    tb := divisor
33    count := 1
34
35    for ; tb + tb < 0 && dividend < tb + tb; {
36      tb += tb
37      count += count
38    }
39
40    return count + getRes(dividend -  tb, divisor)
41  }
42
```

思路：递归实现

```
1   /*
2       思路就是：举例 17 / 5
3
4       递归开始   将两个数字转化为负数 -17 -5
5       div(-17, -5)->  div(-7, -5)  -> div(-2, -5) = 0 发现出口，递归返回
6          2 + 1+ 0 <-     1 + 0 <-  0
7
8       原理和上述一样，就不再赘述了
9
10      太巧妙了
```

```java
*/
public int divide(int dividend, int divisor) {
    if(divisor == -1 && dividend == Integer.MIN_VALUE)
        return Integer.MAX_VALUE;

    int sign = 1;
    if((dividend > 0 && divisor < 0) || (dividend < 0 && divisor > 0))
        sign = -1;

  // 都改为负号是因为int 的范围是[-2^32, 2^32-1]，如果a是-2^32，转为正数时将会溢出
    int a = dividend > 0 ? -dividend : dividend;
    int b = divisor  > 0 ? -divisor  : divisor;

    int res = div(a, b);
    return sign == 1 ? res : sign * res;
}

//坚定不移的相信， 本函数最后返回我们想要的结果
//举例 -17 / -5
private int div(int a, int b)
{
    //递归出口
    if(a > b)       return 0;

    //本轮操作
    int tb = b;
    int count = 1;
    while(tb + tb >= a && tb + tb < 0)
    {
        count += count;
        tb += tb;
    }

  //进入下一轮操作
    return count + div(a - tb, b);
}
```

030 SubString with Concatenation of all words

30. Substring with Concatenation of All Words

难度 困难 262

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

Example 1:

```
Input:
  s = "barfoothefoobarman",
  words = ["foo","bar"]
Output: [0,9]
Explanation: Substrings starting at index 0 and 9 are
"barfoo" and "foobar" respectively.
The output order does not matter, returning [9,0] is
fine too.
```

Example 2:

```
Input:
  s = "wordgoodgoodgoodbestword",
  words = ["word","good","best","word"]
Output: []
```

```java
/*
  因为单词长度固定，可以计算出截取字符串的个数和words 里面是否相等
  借用哈比表
  - 哈希表map  words
  - 哈希表tmp 截取的字符串
  比较两个哈希表是否相等
*/
public List<Integer> findSubstring(String s, String[] words) {
    List<Integer> res = new ArrayList<>();
    if (s == null || s.length() == 0 || words == null || words.length == 0) return
res;
    HashMap<String, Integer> map = new HashMap<>();

    int one_word = words[0].length();
    int word_num = words.length;
    int all_len = one_word * word_num;

```

```
    for (String word : words) {
        map.put(word, map.getOrDefault(word, 0) + 1);
    }

    for (int i = 0; i < s.length() - all_len + 1; i++) {
        String tmp = s.substring(i, i + all_len);
        HashMap<String, Integer> tmp_map = new HashMap<>();
        for (int j = 0; j < all_len; j += one_word) {
            String w = tmp.substring(j, j + one_word);
            tmp_map.put(w, tmp_map.getOrDefault(w, 0) + 1);
        }
        if (map.equals(tmp_map)) res.add(i);
    }

    return res;
}


作者：powcai
链接：https://leetcode-cn.com/problems/substring-with-concatenation-of-all-words/solution/chuan-lian-suo-you-dan-ci-de-zi-chuan-by-powcai/
```

```
/*
  滑动窗口解法
*/

class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
        List<Integer> res = new ArrayList<>();
        if (s == null || s.length() == 0 || words == null || words.length == 0) return res;
        HashMap<String, Integer> map = new HashMap<>();
        int one_word = words[0].length();
        int word_num = words.length;
        int all_len = one_word * word_num;

        for (String word : words) {
            map.put(word, map.getOrDefault(word, 0) + 1);
        }

        for (int i = 0; i < one_word; i++) {
            int left = i, right = i, count = 0;
            HashMap<String, Integer> tmp_map = new HashMap<>();

            while (right + one_word <= s.length()) {
                String w = s.substring(right, right + one_word);
                tmp_map.put(w, tmp_map.getOrDefault(w, 0) + 1);
```

```
                right += one_word;
                count++;

                while (tmp_map.getOrDefault(w, 0) > map.getOrDefault(w, 0)) {
                    String t_w = s.substring(left, left + one_word);
                    count--;
                    tmp_map.put(t_w, tmp_map.getOrDefault(t_w, 0) - 1);
                    left += one_word;
                }
                if (count == word_num) res.add(left);

            }
        }

        return res;
    }
}
作者：powcai
链接：https://leetcode-cn.com/problems/substring-with-concatenation-of-all-words/solution/chuan-lian-suo-you-dan-ci-de-zi-chuan-by-powcai/
```

031 next permutation

31. Next Permutation

难度 中等 492

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

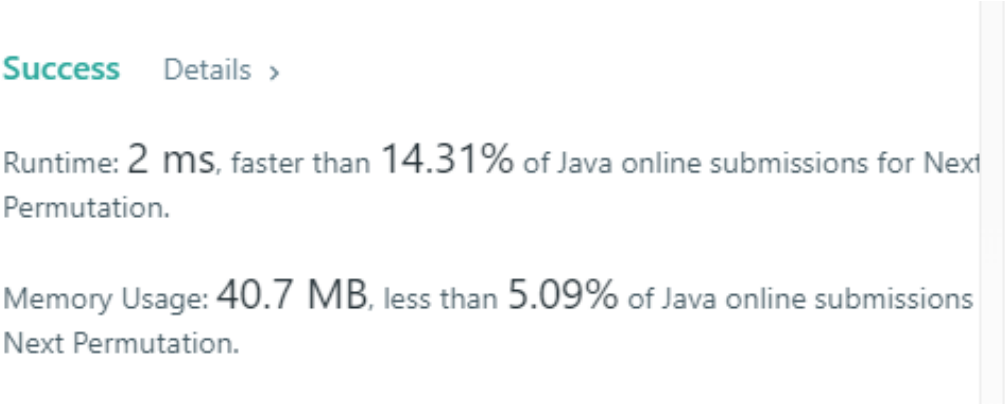
If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be **in-place** and use only constant extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

`1,2,3` → `1,3,2`
`3,2,1` → `1,2,3`
`1,1,5` → `1,5,1`

通过次数 61,903 | 提交次数 184,534




```java
public void nextPermutation(int[] nums) {
    int index = nums.length - 1;

    while(index >= 1 && nums[index-1] >= nums[index])
        index--;

    if(index == 0)
    {
        Arrays.sort(nums);
        return;
    }

    Arrays.sort(nums, index, nums.length);
    index--;

    int right = index + 1;
    while(right < nums.length && nums[right] <= nums[index])
        right++;

    if(right >= nums.length)        return;

    int temp = nums[right];
    nums[right] = nums[index];
    nums[index] = temp;
}
```

032 Longest Valid Parenthese

```java
public int longestValidParentheses(String s) {
    if(s.length() <= 1)     return 0;
    char[] chars = s.toCharArray();
    //dp[i] means from s[0...i] 有效括号, 包含s[i]
    int[] dp = new int[chars.length];
    dp[1] = chars[0] == '(' && chars[1] == ')' ? 2 : 0;

    int max = dp[1];
    for(int i = 2; i < chars.length; i++)
    {
        if(chars[i] == '(')     dp[i] = 0;
        else
        {
            if(chars[i-1] == '(')
                dp[i] = dp[i-2] + 2;

            else
            {
                if(i - dp[i-1] - 1 < 0 || chars[i - dp[i-1] -1] == ')')
                    dp[i] = 0;
                else
                    dp[i] = i - dp[i-1] - 1 >= 1 ?  2 + dp[i-1] + dp[i - dp[i-1] - 2]: 2 + dp[i-1];
            }
        }

        max = Math.max(max, dp[i]);
    }

    return max;
}

```

033 Search in Rotated Array

33. Search in Rotated Sorted Array

难度 中等 735

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

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

You are given a target value to search. If found in the array return its index, otherwise return `-1`.

You may assume no duplicate exists in the array.

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

Example 1:

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

Example 2:

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

不难发现：将待搜索区间从中间一分为二，`mid` **一定会落在其中一个有序区间里。**

情况 1：旋转点在中间

说明：请忽略“旋转点”的具体定义，题目中没有具体说哪个是“旋转点”，但不影响我们解题。

思路：看到logn就一定是二分查找了

如果nums[start] <= nums[mid] 说明前段有序

如果nums[mid] <= nums[end] 说明后段有序

再根据 target的值进行具体判定

```java
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;

        while(left <= right){
            int mid = (left + right) / 2;
            if(nums[mid] == target) return mid;

            //left in order
            if(nums[left] <= nums[mid]){
                if(nums[left] <= target && target <= nums[mid])
                    right = mid;
                else
```

```
                    left = mid + 1;
            }else{
                if(nums[mid] <= target && target <= nums[right])
                    left = mid;
                else
                    right = mid - 1;
            }
        }

        return -1;
    }
```

千万不要怕未知情况，同时注意取等号情况

034 Find First and Last Position of Element in Sorted Array

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

难度 中等 431

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

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

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

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```


```java
public int[] searchRange(int[] nums, int target) {
    int[] res = new int[2];
      res[0] = -1;
      res[1] = -1;
      if(nums.length == 0 || nums == null) return res;
      int start = 0;
      int end = nums.length-1;

      while(start <= end){
          int mid = (start+end)/2;
          if(nums[mid] == target){
              int left = mid;
              int right = mid;

              while(left >=1 && nums[left-1] == target)
                  left--;
              while(right <nums.length-1 && nums[right+1] == target)
                  right++;

              res[0] = left;
              res[1] = right;
              return res;
          }

          if(target < nums[mid]){
              end = mid-1;
          }else{
              start = mid+1;
          }
      }
      return res;
  }
```

//每次记得范围检查，尤其是left -1 与 right+1 那个地方

035 Search Insert Position

35. Search Insert Position

难度 简单 519

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

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

Example 2:

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

Example 3:

```
Input: [1,3,5,6], 7
Output: 4
```

Example 4:

```
Input: [1,3,5,6], 0
Output: 0
```

执行结果： 通过 显示详情 >

执行用时： 0 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 39.1 MB , 在所有 Java 提交中击败了 11.11% 的用户

炫耀一下:

思路：利用二分查找，进行target finding, 如果没找到，调整位置

关键在于nums[0] 的处理

```java
public int searchInsert(int[] nums, int target) {
      int start = 0;
      int end = nums.length-1;

```

```
        while(start <= end){
            int mid = (start+end)/2;

            if(nums[mid] == target) return mid;

            if(target < nums[mid]) end = mid-1;
            else                   start  = mid+1;
        }
        if(start > ((start+end)/2)) start = end;

        if(start == 0 && nums[start] < target)  return start+1;
        if(start == 0 && nums[start] > target)  return start;

        if(nums[start] < target)
            return start+1;
        else
            return start;
    }
```

036 Valid Sudoku

36. Valid Sudoku

难度 中等 330

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

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

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

A partially filled sudoku which is valid.

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

Example 1:

```
Input:
[
  ["5","3",".",".","7",".",".",".",".","."],
  ["6",".",".","1","9","5",".",".",".","."],
  [".","9","8",".",".",".",".",".","6","."],
  ["8",".",".",".","6",".",".",".",".","3"],
  ["4",".",".","8",".","3",".",".",".","1"],
  ["7",".",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".",".","2","8","."],
  [".",".",".","4","1","9",".",".",".","5"],
  [".",".",".",".",".","8",".",".","7","9"]
]
Output: true
```

Example 2:

```
Input:
[
  ["8","3",".",".","7",".",".",".",".","."],
  ["6",".",".","1","9","5",".",".",".","."],
  [".","9","8",".",".",".",".",".","6","."],
  ["8",".",".",".","6",".",".",".",".","3"],
  ["4",".",".","8",".","3",".",".",".","1"],
  ["7",".",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".",".","2","8","."],
  [".",".",".","4","1","9",".",".",".","5"],
  [".",".",".",".",".","8",".",".","7","9"]
]
Output: false
Explanation: Same as Example 1, except with the 5 in
the top left corner being
    modified to 8. Since there are two 8's in the top
left 3x3 sub-box, it is invalid.
```

Note:

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

通过次数 72,572 | 提交次数 121,989

在真实的面试中遇到过这道题?


```java
public boolean isValidSudoku(char[][] board) {
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            if(board[i][j] != '.')
                if(!isValidSudoku(board, i, j))
                    return false;
    return true;
}

private boolean isValidSudoku(char[][] board, int i , int j)
{
    for(int k = 0; k < 9; k++)
    {
        if(board[i][j] == board[i][k] && j != k)  return false;
        if(board[i][j] == board[k][j] && i != k)  return false;
        int row = (i/3)*3+k/3;
        int col = (j/3)*3 + k%3;
        if(board[i][j] == board[row][col]
           && row !=i && col != j)    return false;
    }

    return true;
}
```

037 Sudoku Solver

038 Count and Say

38. Count and Say

难度 简单 467

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

```
1.     1
2.     11
3.     21
4.     1211
5.     111221
```

`1` is read off as `"one 1"` or `11`.
`11` is read off as `"two 1s"` or `21`.
`21` is read off as `"one 2"`, then `one 1"` or `1211`.

Given an integer n where 1 ≤ n ≤ 30, generate the n^{th} term of the count-and-say sequence. You can do so recursively, in other words from the previous member read off the digits, counting the number of digits in groups of the same digit.

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

Example 1:

```
Input: 1
Output: "1"
Explanation: This is the base case.
```

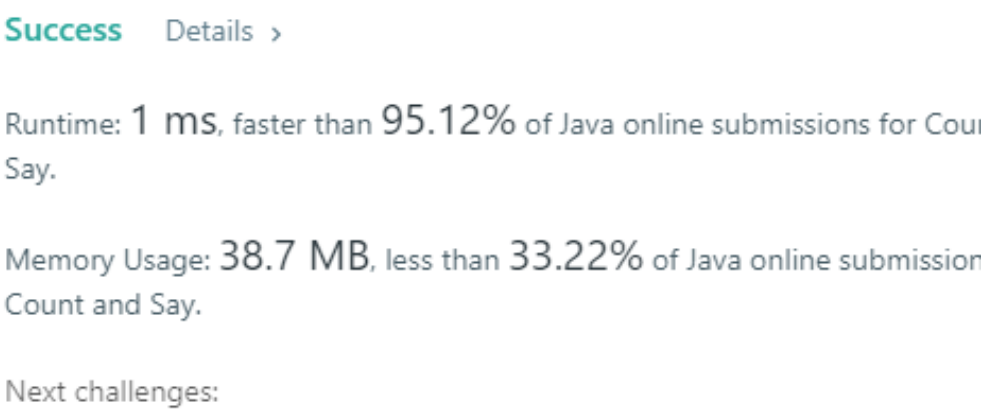
Example 2:

```
Input: 4
Output: "1211"
Explanation: For n = 3 the term was "21" in which we
have two groups "2" and "1", "2" can be read as "12"
which means frequency = 1 and value = 2, the same way
"1" is read as "11", so the answer is the
concatenation of "12" and "11" which is "1211".
```

通过次数 96,738 | 提交次数 175,035

在真实的面试中遇到过这道题?

是 否




```java
public String countAndSay(int n) {
    String res = "1";
    for(int i = 1; i < n; i++)
        res = count(res);

    return res;
}

private String count(String str)
{
    int left = 0, right = 0;
    StringBuilder res = new StringBuilder();

    while(right < str.length())
    {
        while(right < str.length() && str.charAt(right) == str.charAt(left))
            right++;
        res.append(right - left).append(str.charAt(left));
        left = right;
    }

    return res.toString();
}
```

039 Combination Sum

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

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

Note:

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

Example 1:

```
Input: candidates = [2,3,6,7], target = 7,
A solution set is:
[
  [7],
  [2,2,3]
]
```

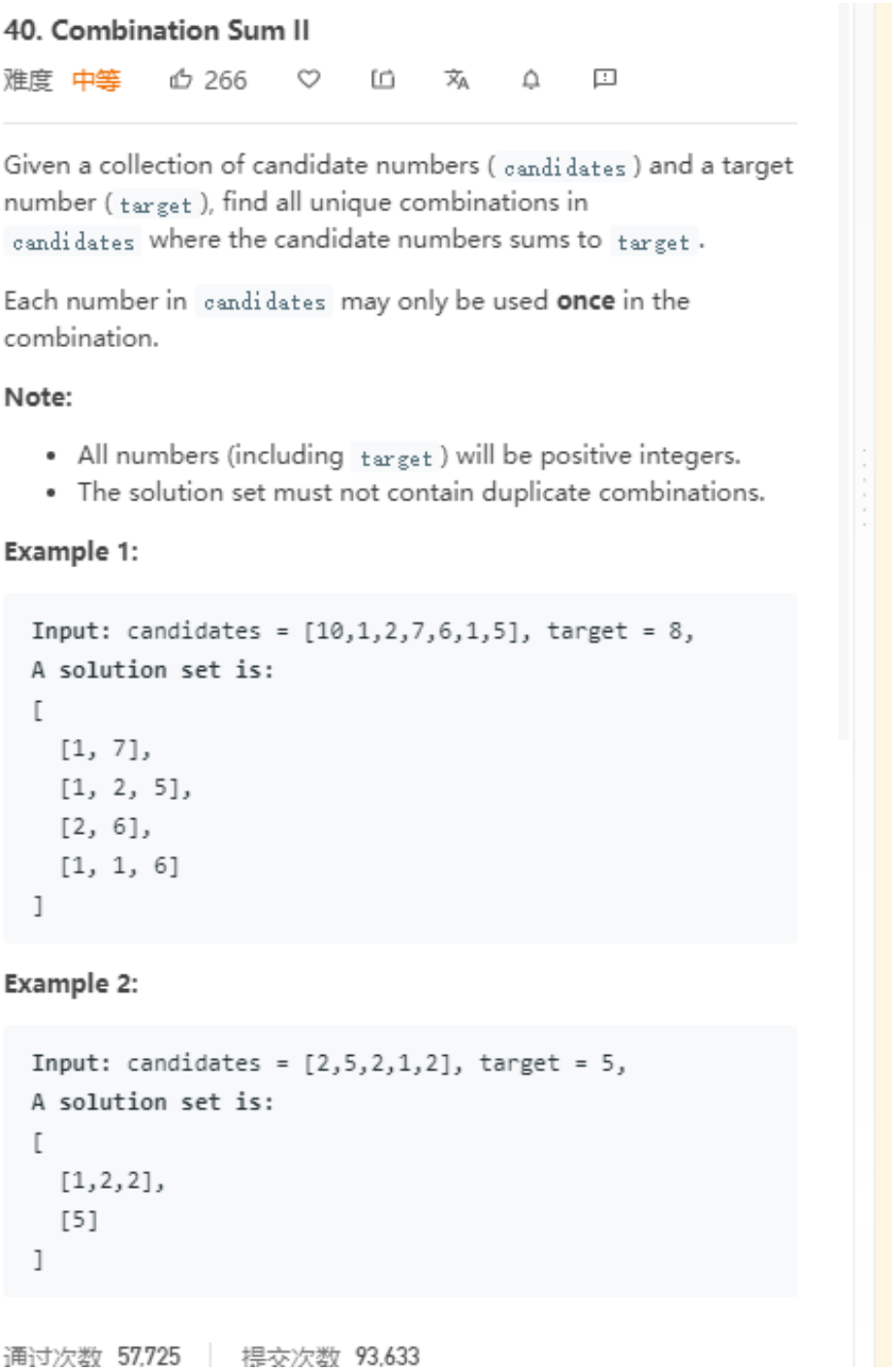
Example 2:

```
Input: candidates = [2,3,5], target = 8,
A solution set is:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

```java
  private  List<List<Integer>> res = new ArrayList<>();
   public  List<List<Integer>> combinationSum(int[] candidates, int target) {
        Arrays.sort(candidates);
        DFS(res,candidates,target,0,new ArrayList<>());
        return res;
    }
  /*
  * residue 用来看每次到底剩余多少
  */
    private  void DFS(List<List<Integer>> res, int[] candidates, int residue, int start, List<Integer> path) {
        //注意这里每次拷贝一个path的副本，因为path是全局公用的一个，否则最后结果会出问题
        //因为每次添加入res 的是path， 修改path最终会导致res里面的path会修改
        if(residue == 0)     {res.add(new ArrayList<>(path)); return;}

        for(int i =start;i<candidates.length;i++){
            if(residue - candidates[i] < 0)  break;

```

```
18              path.add(path.size(),candidates[i]);
19              DFS(res,candidates,residue-candidates[i],i,path);
20              path.remove(path.size()-1);
21          }
22      }
```

040 Combination Sum II



40. Combination Sum II

难度 中等 266

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

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

Note:

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

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8,
A solution set is:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
```

Example 2:

```
Input: candidates = [2,5,2,1,2], target = 5,
A solution set is:
[
  [1,2,2],
  [5]
]
```

通过次数 57,725 | 提交次数 93,633


```java
class Solution {
    private  List<List<Integer>> res = new ArrayList<>();
    public  List<List<Integer>> combinationSum2(int[] candidates, int target) {
        Arrays.sort(candidates);
        DFS(candidates,target,0,res, new ArrayList<Integer>());
        return res;
    }

    private  void DFS(int[] candidates, int residue, int start, List<List<Integer>> res, ArrayList<Integer> path) {
        if(residue == 0)    {res.add(new ArrayList<>(path)); return;}

        for(int i = start; i < candidates.length;i++){
            if(residue - candidates[i] < 0) break;

            path.add(path.size(),candidates[i]);
            DFS(candidates,residue-candidates[i],i+1,res,path);
            path.remove(path.size()-1);
            //这里多加了一步，目的是为了去掉重复的元素
            //比如 1 1 2 5 6 7 10
            //当第一个1进行完深度遍历后，直接跳过下一个1进行next distinct number's 遍历
            while(i+1 < candidates.length && candidates[i+1] == candidates[i])
                i++;
        }
    }

}
```

041 First Missing Positive

41. First Missing Positive

难度 困难 662

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

Example 1:

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

Example 2:

```
Input: [3,4,-1,1]
Output: 2
```

Example 3:

```
Input: [7,8,9,11,12]
Output: 1
```

Note:

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

执行结果: 通过 显示详情 >

执行用时: 2 ms , 在所有 Java 提交中击败了 9.48% 的用户

内存消耗: 37.7 MB , 在所有 Java 提交中击败了 8.33% 的用户

炫耀一下:

```java
/*
  使用蛮力解决
*/
public int firstMissingPositive(int[] nums) {
    if(nums.length == 0)        return 1;
    Arrays.sort(nums);

    int pos_pointer = 0;
    while(pos_pointer < nums.length && nums[pos_pointer] <= 0)
        pos_pointer++;

    HashSet<Integer> set = new HashSet<>();
    for(int i = pos_pointer; i <nums.length; i++)
        set.add(nums[i]);

    for(int i = 1;;i++)
```

```
        if(!set.contains(i))
            return i;

    }
}
```

```
/*
  优化
  原地哈希
*/

public class Solution {

    public int firstMissingPositive(int[] nums) {
        int len = nums.length;

        for (int i = 0; i < len; i++) {
            while (nums[i] > 0 && nums[i] <= len && nums[nums[i] - 1] != nums[i]) {
                // 满足在指定范围内、并且没有放在正确的位置上，才交换
                // 例如：数值 3 应该放在索引 2 的位置上
                swap(nums, nums[i] - 1, i);
            }
        }

        // [1, -1, 3, 4]
        for (int i = 0; i < len; i++) {
            if (nums[i] != i + 1) {
                return i + 1;
            }
        }
        // 都正确则返回数组长度 + 1
        return len + 1;
    }

    private void swap(int[] nums, int index1, int index2) {
        int temp = nums[index1];
        nums[index1] = nums[index2];
        nums[index2] = temp;
    }
}

作者：liweiwei1419
链接：https://leetcode-cn.com/problems/first-missing-positive/solution/tong-pai-xu-python-dai-ma-by-liweiwei1419/
```

042 Trapping rain water 单调栈的应用

42. Trapping Rain Water

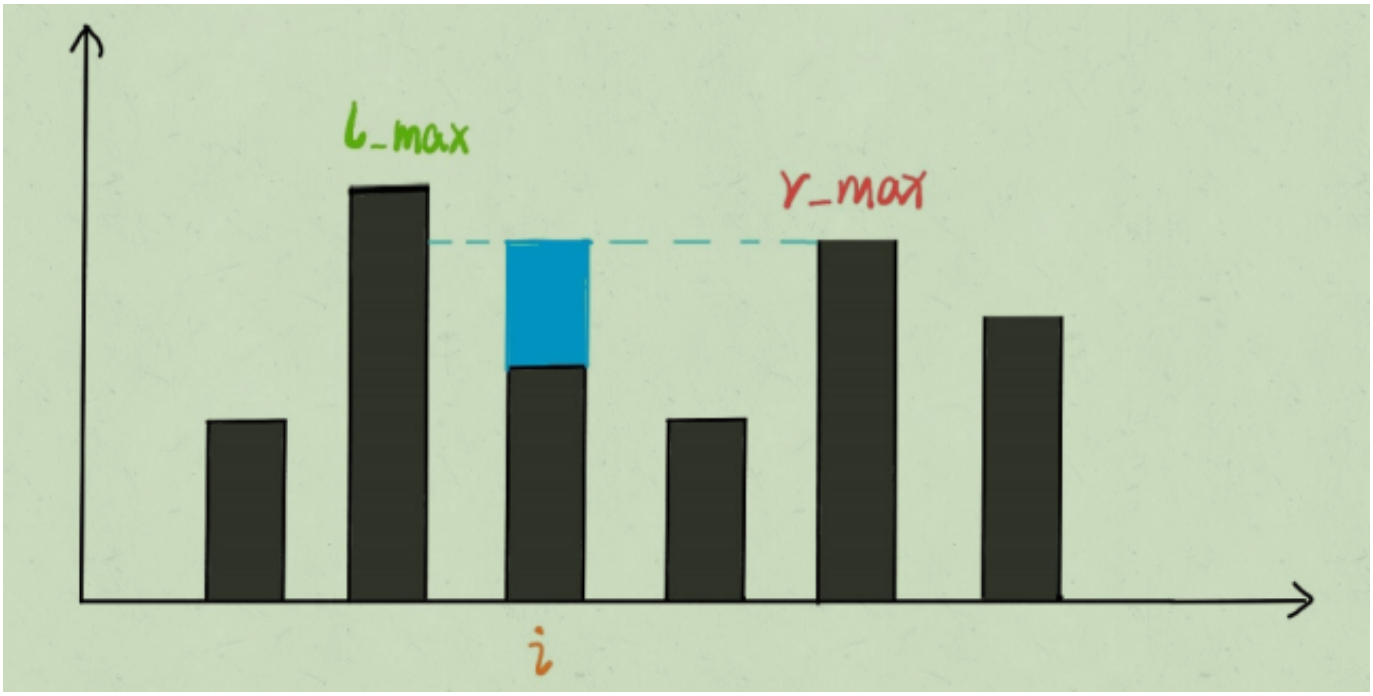
难度 困难 1291

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

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

> 思路一 动态规划

```
/*
  位置i最大的水柱高度就是min(l_max, r_max)
  也就是左右两侧能到达的最高柱子
  water[i] = min(max(height[0...i]), max(height[i...end])) - height[i];
*/
```



```java
/*
  然后开两个数组 r_max, l_max 充当备忘录
*/

public int trap(int[] height) {
    int len = height.length;
    if(len == 0)        return 0;
    int[] l_max = new int[len];
    int[] r_max = new int[len];

    l_max[0] = height[0];
    r_max[len-1] = height[len-1];

    for(int i = 1; i < len; i++)
        l_max[i] = Math.max(l_max[i-1], height[i]);
    for(int i = len-2; i >= 0; i--)
        r_max[i] = Math.max(r_max[i+1], height[i]);

    int res = 0;
    for(int i = 0; i < len; i++)
        res += Math.min(l_max[i], r_max[i]) - height[i];
    return res;
}
```

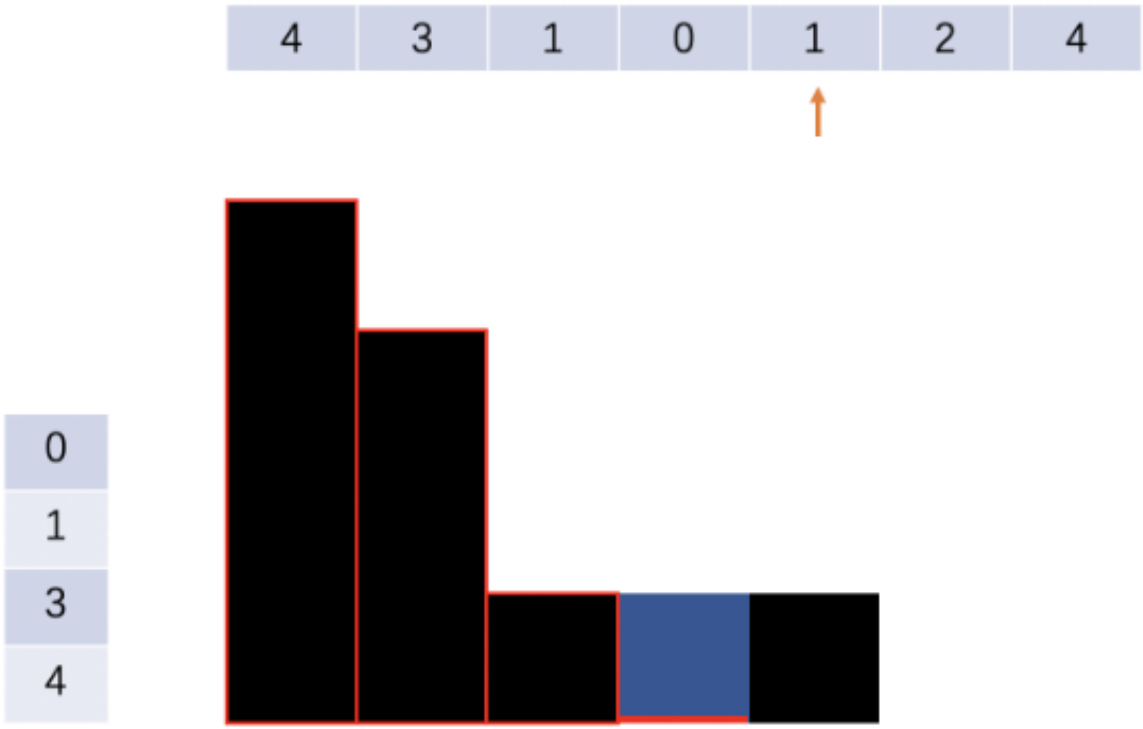
思路二：单调栈

单调栈比普通栈多一个性质，就是维护栈内元素单调递增或者递减

本案例中，维持栈单调递减，以数组[4,3,1,0,1,2,4]举例，首先将4,3,1,0，泵入栈中，

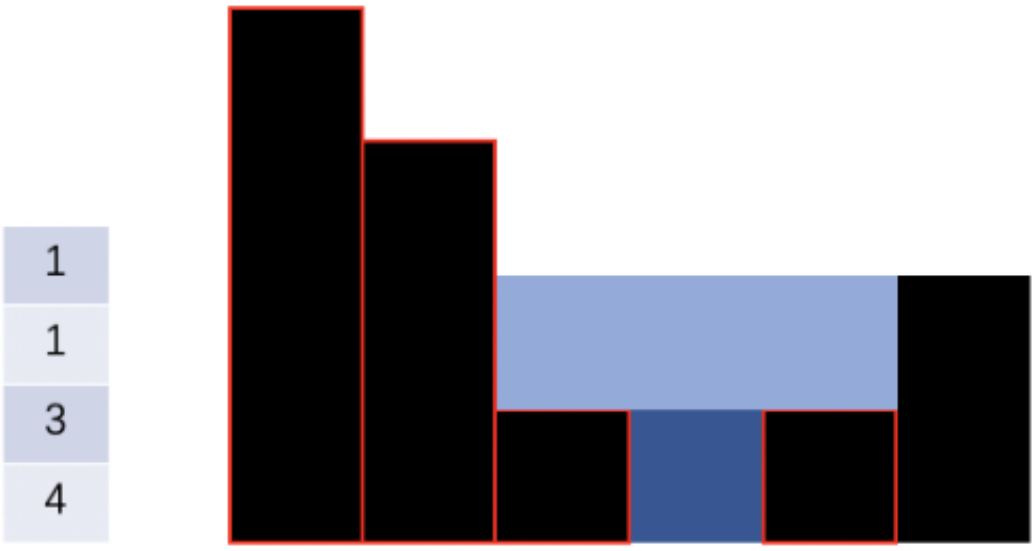
然后准备压入1，发现不满足单调栈的性质。就可以接雨水

雨水的量的高度是栈顶元素和左右两边形成的高度差的min。宽度是`1`。雨水为 1*1



接下来准备压入2，将1弹出，同时进行上步操作。雨水为1*3

循环往复




```java
public int trap6(int[] height) {
    int sum = 0;
    Stack<Integer> stack = new Stack<>();
    int current = 0;
    while (current < height.length) {
        while (!stack.empty() && height[current] > height[stack.peek()]) {
            int h = height[stack.peek()]; //取出要出栈的元素
```

```
            stack.pop(); //出栈

            if (stack.empty()) // 栈空就出去
                break;

            int distance = current - stack.peek() - 1; //两堵墙之前的距离。
            int min = Math.min(height[stack.peek()], height[current]);
            sum = sum + distance * (min - h);
        }

        stack.push(current); //当前指向的墙入栈
        current++; //指针后移
    }
    return sum;
}
作者：windliang
链接：https://leetcode-cn.com/problems/trapping-rain-water/solution/xiang-xi-tong-su-de-si-lu-fen-xi-duo-jie-fa-by-w-8/
```

043 Multiply String

43. Multiply Strings

难度 中等 339

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

Example 1:

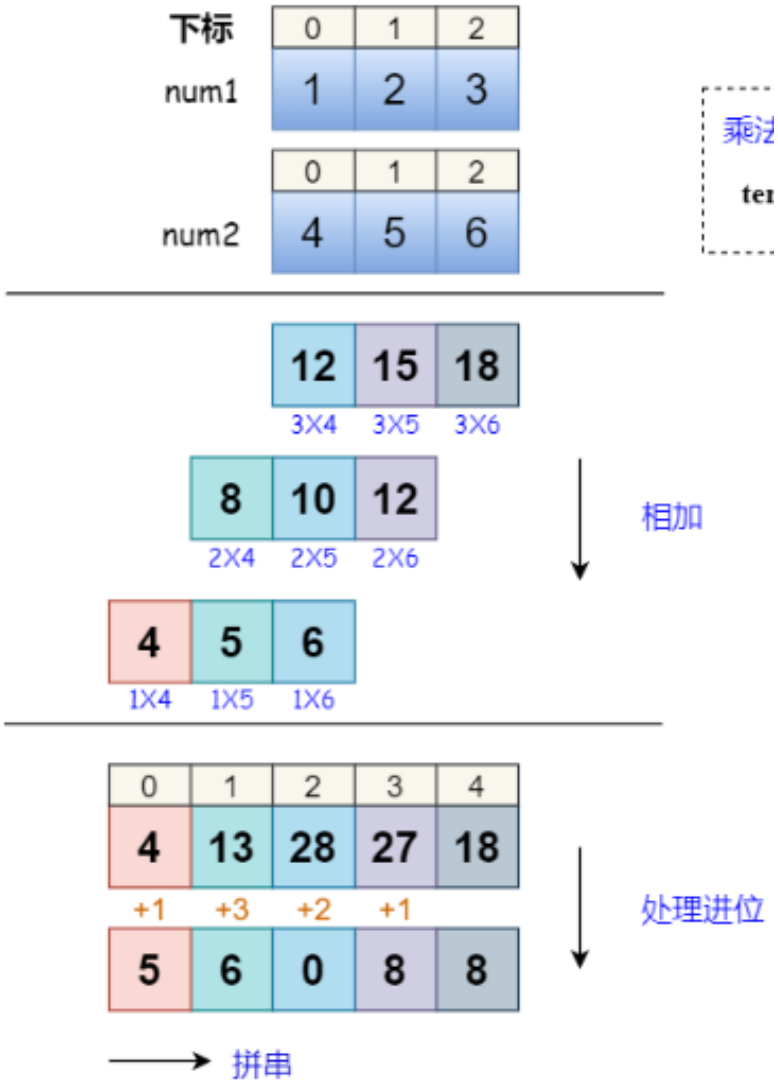
```
Input: num1 = "2", num2 = "3"
Output: "6"
```

Example 2:

```
Input: num1 = "123", num2 = "456"
Output: "56088"
```

Note:

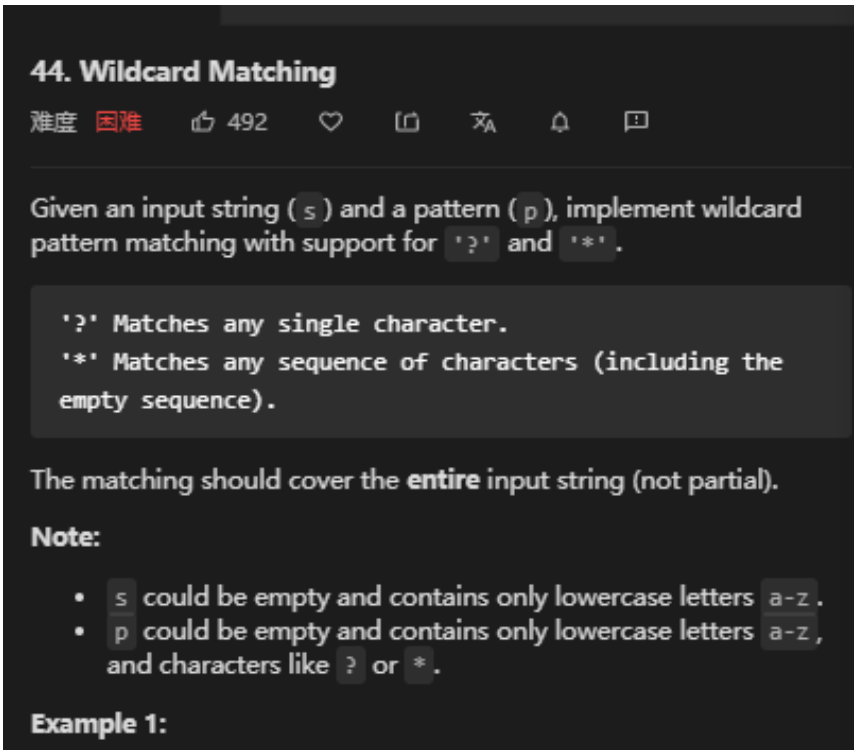
1. The length of both `num1` and `num2` is < 110.
2. Both `num1` and `num2` contain only digits `0-9`.
3. Both `num1` and `num2` do not contain any leading zero, except the number 0 itself.
4. You **must not use any built-in BigInteger library** or **convert the inputs to integer** directly.




```
![image-20200811143509178](leecode 001-050.assets/image-20200811143509178.png)public String multiply(String num1, String num2) {
    if(num1.equals("0") || num2.equals("0"))        return "0";

    int len1 = num1.length();
    int len2 = num2.length();
    int len = len1 + len2;
    int[] res = new int[len];   //3 bits * 3 bits 最多6位

    for(int i = 0; i < len1; i++)
        for(int j = 0;  j< len2; j++)
        {
            int temp = (num1.charAt(i) - '0') * (num2.charAt(j) - '0');
            res[i + j + 1] += temp;
        }

    for(int i = len-1; i>=1; i--){
        int temp = res[i] / 10;
        res[i-1] += temp;
        res[i]   -= temp*10;
```

```
    }

    StringBuilder path = new StringBuilder();
    for(int i = 0 ; i < len; i++)
    {
        //如果首尾是0, 不添加
        if(i == 0 && res[i] == 0)       continue;
        path.append(res[i]);
    }
    return path.toString();
}
作者：xfzhao
链接：https://leetcode-cn.com/problems/multiply-strings/solution/zi-fu-chuan-xiang-cheng-jian-dan-qing-xi-tu-jie-by/
```

044 WildCard Matching



44. Wildcard Matching

难度 困难 492

Given an input string (`s`) and a pattern (`p`), implement wildcard pattern matching with support for `'?'` and `'*'`.

```
'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).
```

The matching should cover the **entire** input string (not partial).

Note:

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

Example 1:

```
public boolean isMatch(String s, String p) {
    boolean[][] dp = new boolean[s.length()+1][p.length()+1];
    dp[s.length()][p.length()] = true;

    for(int i = s.length(); i >= 0; i--)
        for(int j = p.length(); j >= 0; j--)
        {
            if(i == s.length() && j == p.length())      continue;

            boolean firstMatch = (i < s.length() && j < p.length()) &&
```

```
                (s.charAt(i) == p.charAt(j) || p.charAt(j) == '?' || p.charAt(j) == '*');


            if(j < p.length() && p.charAt(j) == '*')
                    // 不使用*     ||  使用 *
                dp[i][j] = dp[i][j+1] ||(firstMatch && dp[i+1][j]);
            else
                dp[i][j] = firstMatch && dp[i+1][j+1];

        }
    return dp[0][0];
}
```

045 Jump Game II

45. Jump Game II

难度 困难 613

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

Example:

```
Input: [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the
last index is 2.
    Jump 1 step from index 0 to 1, then 3 steps to the
last index.
```

Note:

You can assume that you can always reach the last index.

执行结果： 通过 显示详情 >

执行用时： 492 ms ，在所有 C++ 提交中击败了 6.86% 的用户

内存消耗： 21.9 MB ，在所有 C++ 提交中击败了 5.06% 的用户

炫耀一下：

```cpp
class Solution {
public:
    int jump(vector<int>& nums) {
        if(nums.size() == 1)
            return 0;

        queue<int> myQueue;
        unordered_set<int> set;
        myQueue.push(0);
        set.insert(0);

        int count = 0;
        while(!myQueue.empty()){
            int size = myQueue.size();
            for(int i = 0; i < size; i++){
                int curPos = myQueue.front();
                myQueue.pop();
                int steps = nums[curPos];

                for(int i = 1; i <= steps; i++){
                    int nextPos = curPos + i;

                    if(nextPos == nums.size() - 1)
                        return count + 1;

                    if(set.count(nextPos) == 0){
                        myQueue.push(nextPos);
                        set.insert(nextPos);
                    }
                }
            }

            count++;
        }

        return -1;
    }
};
```

提交时间	提交结果	运行时间	内存消耗	
13 分钟前	通过	11 ms	42 MB	J

```java
class Solution {

    //bfs
   public int jump(int[] nums) {
        if (nums.length <= 1) return 0;

        Deque<Integer> queue     = new ArrayDeque<>();
        HashSet<Integer> visited = new HashSet<>();

        queue.add(0);

        int count = 0;
        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i = 0; i < size; i++){
                int curPos = queue.removeFirst();

                while(nums[curPos] != 0){
                    if(visited.contains(curPos + nums[curPos])) {
                        nums[curPos]--;
                        continue;
                    }

                    if(curPos + nums[curPos] == nums.length - 1)    return count + 1;
                    visited.add(curPos + nums[curPos]);
                    queue.addLast(curPos + nums[curPos]);
                    nums[curPos]--;
                }
            }

            count++;
        }

        return count;
    }
}
```

```
/*
  通过91/92 个测试案例
```

```
  超出时间限制
*/
public int jump(int[] nums) {
    int len = nums.length;
    if(len <= 1)        return 0;

    //dp[i] means Min steps from nums[0...i-1]
    int[] dp = new int[len];
    Arrays.fill(dp, Integer.MAX_VALUE);
    dp[0] = 0;

    for(int i = 1; i < len; i++)
        for(int j = 0; j < i; j++)
            if(nums[j] + j >= i)
                dp[i] = Math.min(dp[i], dp[j] + 1);

    return dp[len-1];
}
```


```
/*
  dP 优化,  加了一个break
*/
public int jump(int[] nums) {
    int len = nums.length;
    if(len <= 1)        return 0;

    //dp[i] means Min steps from nums[0...i-1]
    int[] dp = new int[len];
    Arrays.fill(dp, Integer.MAX_VALUE);
    dp[0] = 0;

    for(int i = 1; i < len; i++)
    {
        for(int j = 0; j < i; j++)
        {
            if(nums[j] + j >= i)
            {
                dp[i] = Math.min(dp[i], dp[j] + 1);
                break;
```

```
                //意思就是说，前面的跳跃次数总是要小于等于后面的跳跃次数----->as long as 相同的跳跃目标
            }
        }
    }

    return dp[len-1];
}
```

046 Permutations

46. Permutations

难度 中等 715

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

Example:

```
Input: [1,2,3]
Output:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

048 Rotate Image

48. Rotate Image

难度 中等 481

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Note:

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

```
Given input matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],

rotate the input matrix in-place such that it becomes:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```

Example 2:

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

rotate the input matrix in-place such that it becomes:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

通过次数 81,514 | 提交次数 118,657

```
/*
  作弊玩法
*/
```

```java

public void rotate(int[][] matrix) {
    int len = matrix.length;
    if(len <= 1)        return;

    int[][] copy_matrix = new int[len][len];
    int row_index = 0;
    int column_index = 0;
    for(int j = 0; j < len; j++)
    {
        for(int i = len-1; i >= 0; i--)
        {
            copy_matrix[row_index][column_index++] = matrix[i][j];
        }
        column_index = 0;
        row_index++;
        if(row_index == len){
            reWrite(matrix, copy_matrix);
            return;
        }
    }
}

private void reWrite(int[][] matrix, int[][] copy_matrix) {
    int len = matrix.length;
    for(int i = 0; i < len; i++)
        for(int j = 0; j < len; j++)
            matrix[i][j] = copy_matrix[i][j];

}
```

```java
/*
最直接的想法是先转置矩阵，然后翻转每一行。这个简单的方法已经能达到最优的时间复杂度O(N^2)
*/
class Solution {
  public void rotate(int[][] matrix) {
    int n = matrix.length;

    // transpose matrix
    for (int i = 0; i < n; i++) {
      for (int j = i; j < n; j++) {
        int tmp = matrix[j][i];
        matrix[j][i] = matrix[i][j];
        matrix[i][j] = tmp;
      }
    }
    // reverse each row
```

```
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n / 2; j++) {
        int tmp = matrix[i][j];
        matrix[i][j] = matrix[i][n - j - 1];
        matrix[i][n - j - 1] = tmp;
      }
    }
  }
}

作者：LeetCode
链接：https://leetcode-cn.com/problems/rotate-image/solution/xuan-zhuan-tu-xiang-by-leetcode/
```

049 Group Anagrams

49. Group Anagrams

难度 中等 385

Given an array of strings, group anagrams together.

Example:

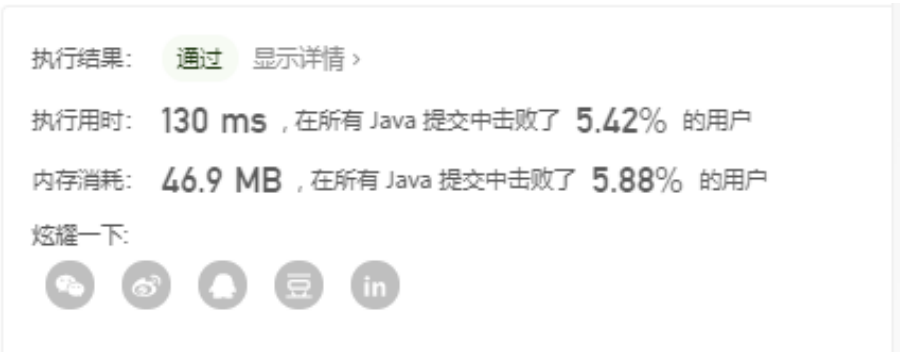
```
Input: ["eat", "tea", "tan", "ate", "nat", "bat"],
Output:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

Note:

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

```
/*
  100 / 101 个通过测试用例
  超出时间限制
*/
public List<List<String>> groupAnagrams(String[] strs) {
    List<List<String>> res = new ArrayList<>();
    HashSet<HashMap<Character,Integer>> lookup = new HashSet<>();

```

```
    for(int i = 0; i < strs.length; i++)
    {
        HashMap<Character, Integer> map = scanWord(strs[i]);
        lookup.add(map);
    //扫描每一个字符串，拿到对应的map，  也就是每个串子的字符数量
    }

    for(HashMap<Character,Integer> map : lookup)
    {
        List<String> path = new ArrayList<>();
        for(int i  = 0; i < strs.length; i++)
        {
            if(map.equals(scanWord(strs[i]))) //如果发现该串和我们比较的map一样，说明是Anagram
                path.add(strs[i]);
        }
        res.add(path);
    }
    return res;
}
//helper function
public HashMap<Character, Integer> scanWord(String s)
{
    HashMap<Character, Integer> map = new HashMap<>();
    for(int i = 0; i < s.length(); i++)
        map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);
    return map;
}
```




```
/*
  将上面的两步循环，合成一步，边找边加
*/
public List<List<String>> groupAnagrams(String[] strs) {
    List<List<String>> res = new ArrayList<>();
    HashMap<HashMap<Character,Integer>, List<String>> lookup = new HashMap<>();

//anagram  ID
    for(int i = 0; i < strs.length; i++)
    {
```

```
        HashMap<Character, Integer> map = scanWord(strs[i]);
        if(!lookup.containsKey(map))
            lookup.put(map, new ArrayList<>());

        lookup.get(map).add(strs[i]);
    }

    for(List<String> ls : lookup.values())
        res.add(ls);

    return res;
}


public HashMap<Character, Integer> scanWord(String s)
{
    HashMap<Character, Integer> map = new HashMap<>();
    for(int i = 0; i < s.length(); i++)
        map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);
    return map;
}
```

```
//另外一种方法
public List<List<String>> groupAnagrams(String[] strs) {
    if (strs == null || strs.length == 0) return new ArrayList<>();
    Map<String, List<String>> map = new HashMap<>();
    for (String s : strs) {
        char[] ca = new char[26];
        for (char c : s.toCharArray()) ca[c - 'a']++;
        String keyStr = String.valueOf(ca);
        if (!map.containsKey(keyStr)) map.put(keyStr, new ArrayList<>());
        map.get(keyStr).add(s);
    }
    return new ArrayList<>(map.values());
}
```

050 Pow(x, n) 幂的递归式写法

50. Pow(x, n)

难度 中等 435

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

Example 1:

```
Input: 2.00000, 10
Output: 1024.00000
```

Example 2:

```
Input: 2.10000, 3
Output: 9.26100
```

Example 3:

```
Input: 2.00000, -2
Output: 0.25000
Explanation: 2^-2 = 1/2^2 = 1/4 = 0.25
```

Note:

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

执行结果： 通过 显示详情 >

执行用时： 1 ms , 在所有 Java 提交中击败了 99.00% 的用户

内存消耗： 37.5 MB , 在所有 Java 提交中击败了 67.73% 的用户

```java
//n 刷 2021/03/25
class Solution {
    public double myPow(double x, int n) {
        if(n == Integer.MIN_VALUE)
            return (x == 1 || x == -1) ? 1 : 0;
        else if(n == 0)
            return 1;

        if(n < 0)
            return myPow(1 / x, -n);
        if(n % 2 == 0)
            return myPow(x * x, n / 2);
        else
            return myPow(x, n - 1) * x;

    }
}
```

```java
class Solution {
    //该函数可以返回我们想要的result
    public double quickMul(double x, long N) {
        if (N == 0) {
            return 1.0;
        }
        double y = quickMul(x, N / 2);
        return N % 2 == 0 ? y * y : y * y * x;
    }

    public double myPow(double x, int n) {
        long N = n;
        return N >= 0 ? quickMul(x, N) : 1.0 / quickMul(x, -N);
    }
}

作者：LeetCode-Solution
链接：https://leetcode-cn.com/problems/powx-n/solution/powx-n-by-leetcode-solution/
```

054 Spiral Matrix

54. Spiral Matrix

难度 中等 412

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

Example 1:

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

Example 2:

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

```java
/*
--首先设定上下左右边界
--其次向右移动到最右，此时第一行因为已经使用过了，可以将其从图中删去，体现在代码中就是重新定义上边界
--判断若重新定义后，上下边界交错，表明螺旋矩阵遍历结束，跳出循环，返回答案
--若上下边界不交错，则遍历还未结束，接着向下向左向上移动，操作过程与第一，二步同理
--不断循环以上步骤，直到某两条边界交错，跳出循环，返回答案
*/
public List<Integer> spiralOrder(int[][] matrix) {
    if(matrix.length == 0 || matrix[0].length == 0)     return new ArrayList<>();
    List<Integer> res = new ArrayList<>();
    int upperBound = 0;
    int lowerBound = matrix.length - 1;
    int leftBound  = 0;
    int rightBound = matrix[0].length - 1;

    while(upperBound <= lowerBound && leftBound <= rightBound)
    {
        for(int i = leftBound; i <= rightBound; i++)
            res.add(matrix[upperBound][i]);
        upperBound++;
```

```
21          if(upperBound > lowerBound)     break;
22
23          for(int i = upperBound; i <= lowerBound; i++)
24              res.add(matrix[i][rightBound]);
25          rightBound--;
26          if(rightBound < leftBound)      break;
27
28          for(int i = rightBound; i >= leftBound; i--)
29              res.add(matrix[lowerBound][i]);
30          lowerBound--;
31          if(upperBound > lowerBound)     break;
32
33          for(int i = lowerBound; i >= upperBound; i--)
34              res.add(matrix[i][leftBound]);
35          leftBound++;
36          if(rightBound < leftBound)      break;
37      }
38
39      return res;
40  }
```

055 Jump Game

55. Jump Game

难度 中等 726

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Example 2:

```
Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.
```

Constraints:

- `1 <= nums.length <= 3 * 10^4`
- `0 <= nums[i][j] <= 10^5`

通过次数 126,175 提交次数 312,258

```java
/*
  通过70/75个测试案例
  超出时间限制
*/

public boolean flag;
public boolean canJump(int[] nums) {
    int len = nums.length;
    flag = false;
    if(len <= 1)        return true;

    dfs(nums, 0);
    return flag;
```

```
}

private void dfs(int[] nums, int start) {
    if(start >= nums.length - 1)
    {
        flag = true;
        return;
    }

    if(nums[start] == 0)        return;

    for(int i = 1; i <= nums[start]; i++)
    {
        int nextStart = start + i;

        dfs(nums, nextStart);
    }
}
```


```
/*
  dp 优化过后
*/
public boolean canJump(int[] nums) {
    int len = nums.length;
    if (len <= 1) return true;

    //dp[i] 代表从nums[0...i-1]一路跳过来，是否能到达
    boolean[] dp = new boolean[len];

    //base case
    dp[0] = true;

    for (int i = 1; i < len; i++) {
        for (int j = 0; j < i; j++) {
            if (dp[j] && j + nums[j] >= i) {
                dp[i] = true;
                break;
            }
        }
```

```
    }

    return dp[len - 1];
}
```

056 Merge Intervals 双指针

56. Merge Intervals

难度 中等 489

Given a collection of intervals, merge all overlapping intervals.

Example 1:

```
Input: [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlaps,
merge them into [1,6].
```

Example 2:

```
Input: [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered
overlapping.
```

NOTE: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

执行结果: 通过 显示详情 > 添加

执行用时: 16 ms , 在所有 C++ 提交中击败了 97.30% 的用户

内存消耗: 13.9 MB , 在所有 C++ 提交中击败了 39.59% 的用户

炫耀一下:

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        vector<vector<int>> res;
     sort(intervals.begin(), intervals.end(), [](vector<int>& v1, vector<int>& v2)
{return v1[0] < v2[0]; });
```

```
        int left = 0, right = 0;
        while(right < intervals.size()){
            int leftBound = intervals[left][0];
            int rightBound = intervals[left][1];
            while(right < intervals.size() && intervals[right][0] <= rightBound){
                rightBound = max(rightBound, intervals[right][1]);
                right++;
            }

            res.push_back({leftBound, rightBound});
            if(right == intervals.size())
                break;

            left = right;
        }

        return res;
    }

    int max(int i, int j){
        return i >= j ? i : j;
    }
};
```


```
public int[][] merge(int[][] intervals) {
    Arrays.sort(intervals, new Comparator<>(){
        @Override
        public int compare(int[] o1, int[] o2)
        {   return o1[0] - o2[0];}
```

```
    });

    List<int[]> path = new ArrayList<>();
    int left = 0, right = 0;


    while(right < intervals.length)
    {
        int leftBound = intervals[left][0];
        int rightBound = intervals[right][1];

        while(right < intervals.length && intervals[right][0] <= rightBound)
        {
            rightBound = Math.max(intervals[right][1], rightBound);
            right++;
        }

        path.add(new int[]{leftBound, rightBound});
        left = right;
    }

    int[][] res = new int[path.size()][2];
    for(int i = 0; i < path.size(); i++)
    {
        res[i][0] = path.get(i)[0];
        res[i][1] = path.get(i)[1];
    }

    return res;
}
```

057 Insert Interval

57. Insert Interval

难度 困难 159

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

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

Example 1:

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

Example 2:

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]],
newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps
with [3,5],[6,7],[8,10].
```

NOTE: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

058 Length of Last Word

58. Length of Last Word

难度 简单 210

Given a string s consists of upper/lower-case alphabets and empty space characters `' '`, return the length of last word (last word means the last appearing word if we loop from left to right) in the string.

If the last word does not exist, return 0.

Note: A word is defined as a **maximal substring** consisting of non-space characters only.

Example:

```
Input: "Hello World"
Output: 5
```


```java
public int lengthOfLastWord(String s) {
    String[] strs = s.split(" ");
    if(strs.length == 0)        return 0;
    return strs[strs.length - 1].length();
}
```

059 Spiral Matrix

59. Spiral Matrix II

难度 中等 201

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

Example:

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

```java
public int[][] generateMatrix(int n) {
    int[][] res = new int[n][n];
    int upperBound = 0;
    int lowerBound = n - 1;
    int leftBound  = 0;
    int rightBound = n - 1;
    int index = 1;

    while(index <= n * n){
        for(int k = leftBound; k <= rightBound; k++)
            res[upperBound][k] = index++;
        upperBound++;
        if(upperBound > lowerBound)     break;

        for(int k = upperBound; k <= lowerBound; k++)
            res[k][rightBound] = index++;
        rightBound--;
        if(rightBound < leftBound)      break;

        for(int k = rightBound; k >= leftBound; k--)
            res[lowerBound][k] = index++;
        lowerBound--;
        if(lowerBound < upperBound)     break;

        for(int k = lowerBound; k >= upperBound; k--)
            res[k][leftBound] = index++;
        leftBound++;
        if(leftBound > rightBound)     break;
    }
    return res;
}
```

061 Rotate List

61. Rotate List

难度 中等 291

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

Example 1:

```
Input: 1->2->3->4->5->NULL, k = 2
Output: 4->5->1->2->3->NULL
Explanation:
rotate 1 steps to the right: 5->1->2->3->4->NULL
rotate 2 steps to the right: 4->5->1->2->3->NULL
```

Example 2:

```
Input: 0->1->2->NULL, k = 4
Output: 2->0->1->NULL
Explanation:
rotate 1 steps to the right: 2->0->1->NULL
rotate 2 steps to the right: 1->2->0->NULL
rotate 3 steps to the right: 0->1->2->NULL
rotate 4 steps to the right: 2->0->1->NULL
```

执行结果: 通过 显示详情 >

执行用时: 3 ms , 在所有 Java 提交中击败了 7.88% 的用户

内存消耗: 39.5 MB , 在所有 Java 提交中击败了 5.41% 的用户

炫耀一下:

```java
/*
  本题是对环形队列的考察
*/
public ListNode rotateRight(ListNode head, int k) {
    if(k == 0 || head == null)      return head;

    ListNode cur = head;
    List<Integer> list = new LinkedList<>();
    int num_node = 0;
    while(cur != null)
    {
        list.add(cur.val);
        num_node++;
        cur = cur.next;
    }
```

```
16      if(k > num_node)    k = k % num_node;
17
18      int[] queue = new int[list.size()];
19      for(int i = 0 ;i < list.size();i++)
20          queue[i] = list.get(i);
21
22      HashSet<Integer> visited = new HashSet<>();
23      int[] res   = new int[list.size()];
24      ListNode first = new ListNode(queue[(list.size()-k) % list.size()]);
25      ListNode curNode = first;
26      visited.add((list.size()-k) % list.size());
27
28      for(int i = (list.size()-k+1)%list.size();; i = (i+1)%list.size())
29      {
30          if(visited.contains(i))
31              break;
32          visited.add(i);
33          curNode.next = new ListNode(queue[i]);
34          curNode = curNode.next;
35      }
36
37      return first;
38  }
```

065 Valid Number -- DFA的应用

65. Valid Number

难度 困难 129

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

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

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

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

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

Update (2015-02-10):
The signature of the `C++` function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button to reset your code definition.

066 Plus One

66. Plus One

难度 简单 495

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

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

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

Example 1:

```
Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
```

Example 2:

```
Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
```

执行结果: 通过 显示详情 >

执行用时: 0 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗: 38.1 MB , 在所有 Java 提交中击败了 5.63% 的用户

炫耀一下:

```java
public int[] plusOne(int[] digits) {
    digits[digits.length-1]++;
    boolean carryFlag = false;
    int[] newDigits = new int[digits.length+1];

    for(int i = digits.length-1; i>= 0; i--)
    {
        if(i == 0 && digits[i] > 9)
        {
            carryFlag = true;
            newDigits[0] = 1;
            digits[i] %= 10;
            System.arraycopy(digits, 0, newDigits, 1, digits.length);
            break;
        }

        if(digits[i] > 9)
```

```
        {
            digits[i] %= 10;
            digits[i-1]++;
        }

    }

    return carryFlag ? newDigits : digits;
}
```

067 Add Binary

67. Add Binary

难度 简单 431

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

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

Example 1:

```
Input: a = "11", b = "1"
Output: "100"
```

Example 2:

```
Input: a = "1010", b = "1011"
Output: "10101"
```

Constraints:

- Each string consists only of `'0'` or `'1'` characters.
- `1 <= a.length, b.length <= 10^4`
- Each string is either `"0"` or doesn't contain any leading zero.

执行结果: 通过 显示详情 >

执行用时: 1 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗: 39.7 MB , 在所有 Java 提交中击败了 7.69% 的用户

炫耀一下:

```java
/*
  和上面的plusOne一样
  只不过先把字符串转换为int数组
  然后从后往前，记得标识carry flag 的进位
*/
public String addBinary(String a, String b) {
    int maxlen = Math.max(a.length(), b.length());
    int[] as = new int[maxlen+1];
    int[] bs = new int[maxlen+1];
    int[] res = new int[maxlen+1];
    int as_index = maxlen;
    int bs_index = maxlen;
    for(int i = a.length()-1; i >= 0; i--)
        as[as_index--] = a.charAt(i) - '0';

    for(int j = b.length() - 1; j >= 0; j--)
        bs[bs_index--] = b.charAt(j)- '0';

    boolean carry_flag = false;

    for(int m = maxlen; m >= 1; m--)
    {
        res[m] += as[m] + bs[m];
        if(res[m] > 1)
        {
            res[m] %= 2;
            res[m-1]++;
            if(m == 1)      carry_flag = true;
        }
    }
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i <= maxlen; i++)
    {
        if(i == 0 && carry_flag)
            sb.append(res[i]);
        else if(i == 0)
            continue;
        else
            sb.append(res[i]);
    }
    return sb.toString();
}
```

068 Text Justification

68. Text Justification

难度 困难 84

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

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

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

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

Note:

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

Example 1:

```
Input:
words = ["This", "is", "an", "example", "of", "text",
"justification."]
maxWidth = 16
Output:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

Example 2:

```
Input:
words =
["What","must","be","acknowledgment","shall","be"]
maxWidth = 16
Output:
[
  "What   must   be",
  "acknowledgment  ",
  "shall be        "
]
Explanation: Note that the last line is "shall be    "
instead of "shall     be",
             because the last line must be left-
justified instead of fully-justified.
```

```
justified instead of fully justified.
             Note that the second line is also left-
justified becase it contains only one word.
```

Example 3:

```
Input:
words =
["Science","is","what","we","understand","well","enough","t

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

```java
/*

We start with left being the first word.

findRight: Then we greedily try to go as far right as possible until we fill our current line.

Then we justify one line at a time.

justify: In all cases we pad the right side with spaces until we reach max width for the line;

    If it's one word then it is easy, the result is just that word.
    If it's the last line then the result is all words separated by a single space.
    Otherwise we calculate the size of each space evenly and if there is a remainder we distribute an extra space until it is gone.
*/
public List<String> fullJustify(String[] words, int maxWidth) {
    int left = 0;
    List<String> result = new ArrayList<>();

    while (left < words.length) {
        int right = findRight(left, words, maxWidth);
        result.add(justify(left, right, words, maxWidth));
        left = right + 1;
```

```java
    }

    return result;
}

private int findRight(int left, String[] words, int maxWidth) {
    int right = left;
    int sum = words[right++].length();

    while (right < words.length && (sum + 1 + words[right].length()) <= maxWidth)
        sum += 1 + words[right++].length();

    return right - 1;
}

private String justify(int left, int right, String[] words, int maxWidth) {
    if (right - left == 0) return padResult(words[left], maxWidth);

    boolean isLastLine = right == words.length - 1;
    int numSpaces = right - left;
    int totalSpace = maxWidth - wordsLength(left, right, words);

    String space = isLastLine ? " " : blank(totalSpace / numSpaces);
    int remainder = isLastLine ? 0 : totalSpace % numSpaces;

    StringBuilder result = new StringBuilder();
    for (int i = left; i <= right; i++)
        result.append(words[i])
        .append(space)
        .append(remainder-- > 0 ? " " : "");

    return padResult(result.toString().trim(), maxWidth);
}

private int wordsLength(int left, int right, String[] words) {
    int wordsLength = 0;
    for (int i = left; i <= right; i++) wordsLength += words[i].length();
    return wordsLength;
}

private String padResult(String result, int maxWidth) {
    return result + blank(maxWidth - result.length());
}

private String blank(int length) {
    return new String(new char[length]).replace('\0', ' ');
}
```

069 Sqrt(X)

69. Sqrt(x)

难度 简单 439

Implement `int sqrt(int x)`.

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

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

Example 1:

```
Input: 4
Output: 2
```

Example 2:

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

执行用时： 1 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 35.7 MB ，在所有 Java 提交中击败了 39.50% 的用户

炫耀一下:

```java
//保证不溢出
public int mySqrt(int x) {
        if(x == 0)  return 0;
        if(x <= 3)  return 1;

        int left = 1, right = 46340;
        int ans = 0;
        while(left <= right){
            int mid = left + (right - left) / 2;

            int cur = mid * mid;
            if(cur == x)  return mid;
            else if(cur > x)    right = mid - 1;
```

```
            else{
                ans = mid;
                left = mid + 1;
            }

        }

        return ans;
    }
```


```
public int mySqrt(int x) {
    long xl = x;
    if(xl == 0)       return 0;
    if(xl <= 3)       return 1;
    long res = 0;
    for(long i = 2;; i++)
        if(i*i <= xl)
            res = Math.max(res, i);
    else
        break;
    return (int)res;
}
```

leecode 071-080

071 Simplify Path

71. Simplify Path

难度 中等 158

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

In a UNIX-style file system, a period `.` refers to the current directory. Furthermore, a double period `..` moves the directory up a level.

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

Example 1:

```
Input: "/home/"
Output: "/home"
Explanation: Note that there is no trailing slash after the last directory name.
```

Example 2:

```
Input: "/../"
Output: "/"
Explanation: Going one level up from the root directory is a no-op, as the root level is the highest level you can go.
```

```go
func simplifyPath(path string) string {
  paths := strings.Split(path, "/")

  stack := make([]string, 0)

  for i := 0; i < len(paths); i++ {
        if paths[i] == ""{
            continue
        }

    if paths[i] == "." {
            continue
    } else if paths[i] == ".." {
      if len(stack) != 0 {
        stack = stack[0 : len(stack)-1]
      }
    } else {
```

```
      stack = append(stack, paths[i])
    }
  }

  res := "/"
  for i := 0; i < len(stack); i++ {
    res += stack[i]

    if i != len(stack)-1 {
      res += "/"
    }
  }

  return res
}
```

```
class Solution {
    public String simplifyPath(String path) {
        Deque<String> queue = new ArrayDeque<>();
        String[] strs = path.split("/");

        for(int i = 0; i < strs.length; i++)
        {
            if(strs[i].equals(".") || strs[i].equals(""))
                continue;
            else if(strs[i].equals( ".."))
            {
                if(!queue.isEmpty())
                    queue.removeLast();
            }
            else
                queue.addLast(strs[i]);
        }
        if(queue.isEmpty())       return "/";
        StringBuilder sb = new StringBuilder();
        while(!queue.isEmpty())
            sb.append(queue.removeFirst()).append("/");

        sb.setLength(sb.length()-1);
        sb.insert(0, "/");
        return sb.toString();
    }
}

作者：venturekwok
```

```
链接：https://leetcode-cn.com/problems/simplify-path/solution/java-si-lu-qing-xi-cai-yong-dui-lie-shi-pin-jiang-/
来源：力扣（LeetCode）
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

072 Edit Distance

72. Edit Distance

难度 困难 1332

Given two strings `word1` and `word2`, return *the minimum number of operations required to convert* `word1` *to* `word2`.

You have the following three operations permitted on a word:

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

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

```
```

073 Set Matrix Zeroes

73. Set Matrix Zeroes

难度 中等 247

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

Example 1:

```
Input:
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
Output:
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
```

Example 2:

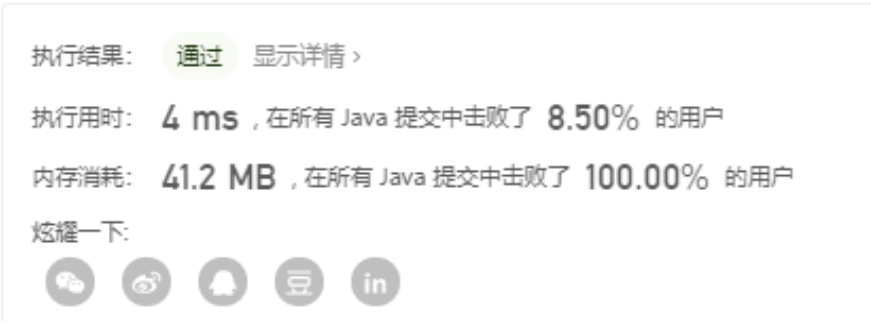
```
Input:
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
Output:
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
```

Follow up:

- A straight forward solution using O(mn) space is probably a bad idea.
- A simple improvement uses O(m + n) space, but still not the best solution.
- Could you devise a constant space solution?

通过次数 44,919 | 提交次数 80,767

在真实的面试中遇到过这道题?




```java
/*
  笨办法：统计我们需要变成0的行和列，最后设置为0
*/
public void setZeroes(int[][] matrix) {
    HashSet<Integer> targetRow = new HashSet<>();
    HashSet<Integer> targetCol = new HashSet<>();

    int row = matrix.length;
    if(row == 0)        return;
    int column = matrix[0].length;

    for(int i = 0; i < row; i++)
        for(int j = 0; j < column; j++)
            if(matrix[i][j] == 0)
            {
                targetRow.add(i);
                targetCol.add(j);
            }
    setRow(matrix, targetRow);
    setCol(matrix, targetCol);
}

private void setCol(int[][] matrix, HashSet<Integer> targetCol) {
    for(Integer i : targetCol)
        for(int j = 0; j < matrix.length; j++)
            matrix[j][i] = 0;
}

private void setRow(int[][] matrix, HashSet<Integer> targetRow) {
    for(Integer i : targetRow)
        for(int j = 0; j < matrix[0].length;j++)
            matrix[i][j] = 0;
}
```

```java
void setZeroes(vector<vector<int>>& matrix) {
        bool isZeroCol = false;
        bool isZeroRow = false;
```

```
4
5           for (int i = 0; i < matrix.size(); i++) { //check the first column
6               if (matrix[i][0] == 0) {
7                   isZeroCol = true;
8                   break;
9               }
10          }
11          for (int i = 0; i < matrix[0].size(); i++) { //check the first row
12              if (matrix[0][i] == 0) {
13                  isZeroRow = true;
14                  break;
15              }
16          }
17
 //=========================================================================
==
18          for (int i = 1; i < matrix.size(); i++) { //check except the first row and
column
19              for (int j = 1; j < matrix[0].size(); j++)
20                  if (matrix[i][j] == 0) {
21                      matrix[i][0] = 0;
22                      matrix[0][j] = 0;
23                  }
24          }
25          for (int i = 1; i < matrix.size(); i++) { //process except the first row
and column
26             for (int j = 1; j < matrix[0].size(); j++)
27                 if (matrix[i][0] == 0 || matrix[0][j] == 0)
28                     matrix[i][j] = 0;
29          }
30
 //=========================================================================
==
31          if (isZeroCol) { //handle the first column
32              for (int i = 0; i < matrix.size(); i++)
33                  matrix[i][0] = 0;
34          }
35          if (isZeroRow) { //handle the first row
36              for (int i = 0; i < matrix[0].size(); i++)
37                  matrix[0][i] = 0;
38          }
39  }
40  https://leetcode.com/problems/set-matrix-zeroes/discuss/26014/Any-shorter-O(1)-
space-solution
```

074 Search a 2D Matrix

74. Search a 2D Matrix

难度 中等 204

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.

Example 1:

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

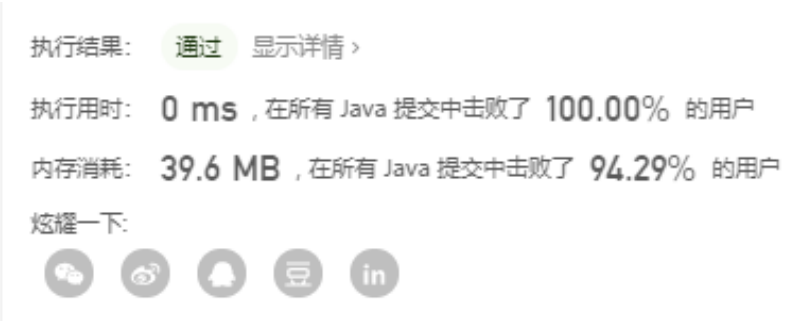
Example 2:

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

通过次数 50,239 | 提交次数 131,024

```go
func searchMatrix(matrix [][]int, target int) bool {
  left, right := 0, len(matrix) - 1
  col := len(matrix[0])

  targetRow := -1
  for ;left <= right;{
    mid := (left + right) / 2

    if matrix[mid][0] <= target && target <= matrix[mid][col - 1]{
      targetRow = mid
      break
    }

    if matrix[mid][0] > target{
      right = mid - 1
    }else if matrix[mid][col - 1] < target{
```

```
      left = mid + 1
    }
  }

  if targetRow == -1{
    return false
  }

  left, right = 0, col - 1
  for ;left <= right;{
    mid := (left + right) / 2

    if matrix[targetRow][mid] == target{
      return true
    }else if matrix[targetRow][mid] > target{
      right = mid - 1
    }else{
      left = mid + 1
    }
  }

  return false

}
```




```
public boolean searchMatrix(int[][] matrix, int target) {

    int row = matrix.length;
    if(row == 0)        return false;
    int column = matrix[0].length;
    if(column <= 0)     return false;

    int left = 0;
    int right = row-1;

    while(left <= right)
    {
```

```
        int mid = (left + right) / 2;
        if(matrix[mid][0] == target)
            return true;
        else if (matrix[mid][0] >  target)
            right = mid-1;
        else if(matrix[mid][0] < target)
            if(matrix[mid][column-1] >= target)
                break;
        else
            left = mid+1;
    }

    int fixed_row = (left + right) / 2;
    left = 0;
    right = column-1;
    while(left <= right)
    {
        int mid = (left + right) / 2;
        if(matrix[fixed_row][mid] == target)
            return true;
        else if(matrix[fixed_row][mid] > target)
            right = mid-1;
        else
            left = mid+1;
    }

    return false;
}
```

075 Sort Colors

75. Sort Colors

难度 中等 489

Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

Example:

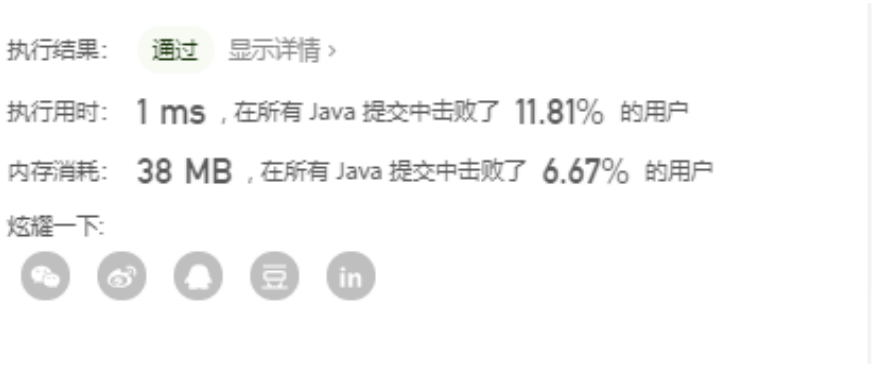
```
Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Follow up:

- A rather straight forward solution is a two-pass algorithm using counting sort.
  First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.
- Could you come up with a one-pass algorithm using only constant space?

```java
public void sortColors(int[] nums) {
    int zero_position = 0;
    int two_position = nums.length - 1;
    for (int i = 0; i <= two_position; i++) {
        if (nums[i] == 0) {
            //将当前位置的数字保存
            int temp = nums[zero_position];
            //把 0 存过来
            nums[zero_position] = 0;
            //把之前的数换过来
            nums[i] = temp;
            //当前指针后移
            zero_position++;
        } else if (nums[i] == 2) {
            //将当前位置的数字保存
            int temp = nums[two_position];
            //把 2 存过来
            nums[two_position] = 2;
            //把之前的数换过来
            nums[i] = temp;
            //当前指针前移
            two_position--;
```

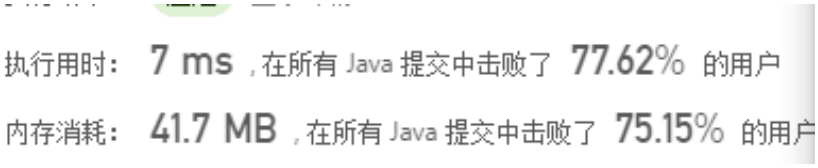
```
            //这里一定要注意，因为我们把后边的数字换到了第 i 个位置，
            //这个数字我们还没有判断它是多少，外层的 for 循环会使得 i++ 导致跳过这个元素
            //所以要 i--
            //而对于上边 zero_position 的更新不需要考虑，因为它是从前边换过来的数字
            //在之前已经都判断过了
            i--;
        }
    }
}

```




```java
public void sortColors(int[] nums) {
    int n = nums.length;
    int h = 1;

    while(h < n/3)  h = 3*h + 1;

    while(h >= 1)
    {
        for(int i = h; i < n; i++)
            for(int j = i; j >= h && less(nums[j],nums[j-h]); j-= h)
                exch(nums, j, j-h);

        h = h/3;
    }

}

private boolean less(int num, int num1) {
    return num - num1 < 0;
}

private void exch(int[] nums, int j, int i) {
```

```
    int temp = nums[i];
    nums[i] = nums[j];
    nums[j] = temp;
}
```

79 Word Search 记得用返回值及时结束递归




```
char[][] board;
int row = 0, column = 0;
int[][] directions = {{0,1},{1,0},{0,-1},{-1,0}};
boolean[][] visited;
public boolean exist(char[][] board, String word) {
    this.board = board;
    row = board.length; column = row == 0 ? 0 :board[0].length;
    if(column == 0)     return word.length() == 0;

    visited = new boolean[row][column];
    for(int i = 0; i < row; i++)
        for(int j = 0; j < column; j++)
            if(dfs(i, j, word, 0))
                return true;
    return false;
}

private boolean dfs(int x, int y, String word, int checkpoint)
{
    if(checkpoint == word.length() - 1)
    {   return board[x][y] == word.charAt(checkpoint);}

    if(word.charAt(checkpoint) == board[x][y])
    {
        visited[x][y] = true;
        for(int k = 0; k < 4; k++)
        {
            int newX = x + directions[k][0];
            int newY = y + directions[k][1];
            if(isInRange(newX, newY) && !visited[newX][newY])
                if(dfs(newX, newY, word, checkpoint+1))
                    return true;
        }

        visited[x][y] = false;
```

```
36      }
37
38      return false;
39  }
40
41  private boolean isInRange(int x, int y)
42  {   return x >= 0 && y >= 0 && x < row && y < column;}
```

080 Remove Duplicates from Sorted Arrays II

80. Remove Duplicates from Sorted Array II

难度 中等 247

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

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

Example 1:

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

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

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

Example 2:

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

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

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

Clarification:

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

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

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making
a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be
known by the caller.
// using the length returned by your function, it prints
the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

通过次数 49,345　　提交次数 88,461

```java
//
public int removeDuplicates(int[] nums) {
      int left = 0, right = 0;
      int len = nums.length;
      int index = 0;
      while(right < len){
          while(right < len && nums[left] == nums[right])
              right++;

          for(int i = 0; i < Math.min(right - left, 2); i++)
              nums[index++] = nums[left];

          left = right;
      }

      return index;
  }
```


```java
public int removeDuplicates(int[] nums) {
      int len = nums.length;
      int res = 0;
      int left = 0;
```

```
        int right = left;
        int index = 0;

        while(right <= len-1)
        {
            while(right <= len-1 && nums[right] == nums[left])
            {
                if(right - left < 2)
                    nums[index++] = nums[right];
                right++;
            }

            if(right - left >= 2)
                res += 2;
            else
                res += right - left ;

            left = right;
        }
        return res;
    }
```

081 Search In Rotated Sorted Array II

81. Search in Rotated Sorted Array II

难度 中等 183

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

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

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

Example 1:

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

Example 2:

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

Follow up:

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

执行结果: 通过 显示详情 >

执行用时: 2 ms , 在所有 Java 提交中击败了 64.91% 的用户

内存消耗: 39.5 MB , 在所有 Java 提交中击败了 33.33% 的用户

炫耀一下:

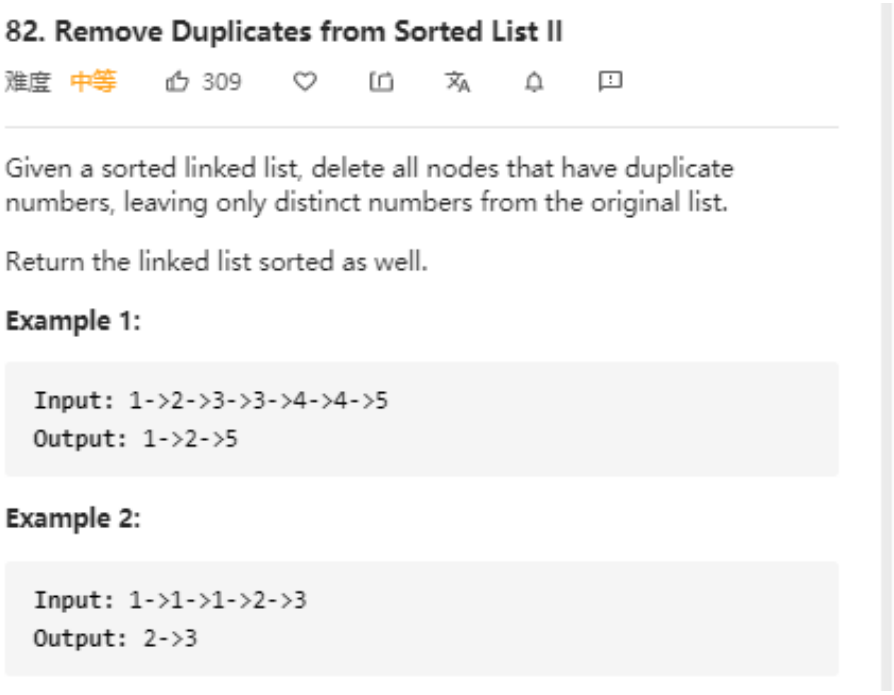
```java
/*
  作弊玩法，仅娱乐
*/
public boolean search(int[] nums, int target) {
    int len = nums.length;
    int left = 0;
    int right = len-1;

    Arrays.sort(nums);

    while(left <= right)
    {
        int mid = (left + right) / 2;

        if      (nums[mid] == target)   return true;
        else if (nums[mid] > target)    right = mid - 1;
```

```
        else                                left = mid+1;

    }

    return false;
}
```

```
/*
  正经解法：
  本题是需要使用二分查找，怎么分是关键，举个例子：

第一类
    1011110111 和 1110111101 这种。
    此种情况下 nums[start] == nums[mid]，分不清到底是前面有序还是后面有序，此时 start++ 即可。
    相当于去掉一个重复的干扰项。
第二类
    2 3 4 5 6 7 1 这种，也就是 nums[start] < nums[mid]。此例子中就是 2 < 5；
    这种情况下，前半部分有序。
    因此如果 nums[start] <=target<nums[mid]，则在前半部分找，否则去后半部分找。
第三类
    6 7 1 2 3 4 5 这种，也就是 nums[start] > nums[mid]。此例子中就是 6 > 2；
    这种情况下，后半部分有序。因此如果 nums[mid] <target<=nums[end]。则在后半部分找，否则去前半部分找。

*/

public boolean search(int[] nums, int target) {
        if (nums == null || nums.length == 0) {
            return false;
        }
        int start = 0;
        int end = nums.length - 1;
        int mid;

        while (start <= end) {
            mid = start + (end - start) / 2;
            if (nums[mid] == target) {
                return true;
            }
            if (nums[start] == nums[mid]) {
                start++;
                continue;
            }

            //前半部分有序
            if (nums[start] < nums[mid]) {
```

```
                //target在前半部分
                if (nums[mid] > target && nums[start] <= target) {
                    end = mid - 1;
                } else {  //否则，去后半部分找
                    start = mid + 1;
                }
            }

            else {
                //后半部分有序
                //target在后半部分
                if (nums[mid] < target && nums[end] >= target) {
                    start = mid + 1;
                } else {  //否则，去后半部分找
                    end = mid - 1;

                }
            }
        }

        //一直没找到，返回false
        return false;
    }

作者：reedfan
链接：https://leetcode-cn.com/problems/search-in-rotated-sorted-array-ii/solution/zai-javazhong-ji-bai-liao-100de-yong-hu-by-reedfan/
```

082 Remove Duplicates From Sorted List II



82. Remove Duplicates from Sorted List II

难度 中等 309

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

Return the linked list sorted as well.

Example 1:

```
Input: 1->2->3->3->4->4->5
Output: 1->2->5
```

Example 2:

```
Input: 1->1->1->2->3
Output: 2->3
```

```go
func deleteDuplicates(head *ListNode) *ListNode {
  head = delete(head)
  return head
}

func delete(head *ListNode) *ListNode{
  if head == nil{
    return nil
  }

  var cur *ListNode
  cur = head

  for ;cur.Next != nil && cur.Next.Val == head.Val; {
    cur = cur.Next
  }

  if cur == head{
    cur.Next = delete(cur.Next)
    return cur
  }else{
    return delete(cur.Next)
  }
}

```

```java
public ListNode deleteDuplicates(ListNode head) {
    if(head == null || head.next == null)   return head;
    ListNode newHead = new ListNode(Integer.MAX_VALUE);

    ListNode cur = newHead;
    ListNode cc  = head;
    while(cc != null)
    {
        int count = 0;
        while(cc.next != null && cc.val == cc.next.val)
        {
            count++;
            cc = cc.next;
        }

        if(count == 0)
        {
            cur.next = cc;
            cur = cur.next;
```

```
            cc = cc.next;
            cur.next = null;
        }
        else
            cc = cc.next;
    }
    return newHead.next;
}
```

083 Remove Duplicates From Sorted List

83. Remove Duplicates from Sorted List

难度 简单 342

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

Example 1:

```
Input: 1->1->2
Output: 1->2
```

Example 2:

```
Input: 1->1->2->3->3
Output: 1->2->3
```

执行结果: 通过 显示详情 >

执行用时: 1 ms , 在所有 Java 提交中击败了 72.16% 的用户

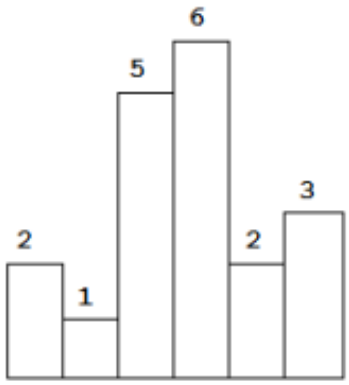
内存消耗: 39.8 MB , 在所有 Java 提交中击败了 5.97% 的用户

炫耀一下:

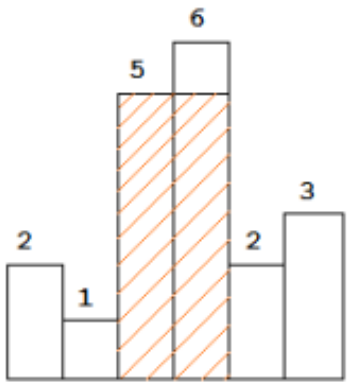
```java
public ListNode deleteDuplicates(ListNode head) {
    if(head == null)        return head;
    ListNode cur = head;
    while(cur != null && cur.next != null)
    {
        if(cur.val == cur.next.val)
            cur.next = cur.next.next;
        else
            cur = cur.next;
    }

    return head;
}
```

084 Largest Rectangle In Histogram

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.



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



The largest rectangle is shown in the shaded area, which has area = `10` unit.

Example:

```
Input: [2,1,5,6,2,3]
Output: 10
```

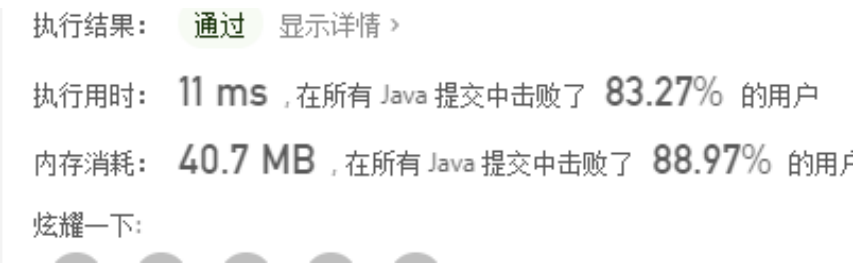
通过次数 69,787 | 提交次数 169,929

执行结果： 通过 显示详情 ›

执行用时： 941 ms , 在所有 Java 提交中击败了 20.01% 的用户

内存消耗： 41.6 MB , 在所有 Java 提交中击败了 10.32% 的用户

炫耀一下:




```java
/*
  可以通过，但是需要调整方法
*/
public int largestRectangleArea(int[] heights) {
        if(heights.length == 0)     return 0;
        int res = 0;
        int len = heights.length;
        int[] arr = new int[len + 2];
        for(int i = 0; i < len; i++)
            arr[i+1] = heights[i];

        for(int i = 1; i < arr.length - 1; i++)
        {
            int left = i, right = i;
            while(left >= 0 && arr[left] >= arr[i])
                left--;
            while(right < arr.length && arr[right] >= arr[i])
                right++;

            res = Math.max((right - left - 1) * arr[i], res);
        }

        return res;
    }

```

```java
// 单调栈
public int largestRectangleArea(int[] heights) {
        if(heights.length == 0)     return 0;
        int res = 0;
        int len = heights.length;
        int[] arr = new int[len + 2];
        for(int i = 0; i < len; i++)
            arr[i+1] = heights[i];

        Deque<Integer> stack = new ArrayDeque<>();
        int index = 1;
        stack.push(0);
        while(index < arr.length)
        {
            while(index < arr.length && arr[index] >= arr[stack.peek()])
```

```
                stack.push(index++);

            while(index < arr.length && arr[index] < arr[stack.peek()])
            {
                int curHeight = arr[stack.pop()];
                res = Math.max(res, curHeight * (index - stack.peek() - 1));
            }

            stack.push(index);
        }

        return res;
    }
```

086 Partition List

86. Partition List

难度 中等 222

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

Example:

```
Input: head = 1->4->3->2->5->2, x = 3
Output: 1->2->2->4->3->5
```

执行结果: 通过 显示详情 >

执行用时: 1 ms , 在所有 Java 提交中击败了 23.91% 的用户

内存消耗: 39.1 MB , 在所有 Java 提交中击败了 5.55% 的用户

炫耀一下:

```
public ListNode partition(ListNode head, int x) {
    ListNode lessHead = new ListNode(0);
    ListNode moreHead = new ListNode(0);

    if(head == null)        return null;

```

```
    ListNode cur = head;
    ListNode curLess = lessHead;
    ListNode curMore = moreHead;

    while(cur != null)
    {
        if(cur.val < x)
        {
            curLess.next = new ListNode(cur.val);
            curLess = curLess.next;
            cur = cur.next;
        }
        else
        {
            curMore.next = new ListNode(cur.val);
            curMore = curMore.next;
            cur = cur.next;
        }
    }
    curLess.next = moreHead.next;
    return lessHead.next;
}
```

087 Scramble String

87. Scramble String

难度 困难 137

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

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

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

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

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

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

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

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

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

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

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

Example 1:

```
Input: s1 = "great", s2 = "rgeat"
Output: true
```

Example 2:

```
Input: s1 = "abcde", s2 = "caebd"
```

```java
//递归法
public boolean isScramble(String s1, String s2) {
    if (s1.equals(s2)) return true;

    int[] letters = new int[26];
    for (int i=0; i<s1.length(); i++) {
        letters[s1.charAt(i)-'a']++;
        letters[s2.charAt(i)-'a']--;
    }
    for (int i=0; i<26; i++) if (letters[i]!=0) return false;

    for (int i=1; i<s1.length(); i++) {
        if (isScramble(s1.substring(0,i), s2.substring(0,i))
            && isScramble(s1.substring(i), s2.substring(i))) return true;
        if (isScramble(s1.substring(0,i), s2.substring(s2.length()-i))
            && isScramble(s1.substring(i), s2.substring(0,s2.length()-i))) return
true;
```

```
17      }
18      return false;
19  }
```

088 Merge Sorted Array

88. Merge Sorted Array

难度 简单 557

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

Note:

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

Example:

```
Input:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6],       n = 3

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

Constraints:

- `-10^9 <= nums1[i], nums2[i] <= 10^9`
- `nums1.length == m + n`
- `nums2.length == n`

执行结果: 通过 显示详情 >

执行用时: 1 ms , 在所有 Java 提交中击败了 23.80% 的用户

内存消耗: 39.9 MB , 在所有 Java 提交中击败了 5.06% 的用户

炫耀一下:

```java
public void merge(int[] nums1, int m, int[] nums2, int n) {
    int index = 0;
    for(int i = m; i < m+n; i++)
        nums1[i] = nums2[index++];

    Arrays.sort(nums1);
}
```

089 Gray Code

```java
public List<Integer> grayCode(int n) {
    List<Integer> res = new ArrayList<>();
    res.add(0);
    if(n == 0)      return res;

    int c = 1;
    while(c <= n)
    {
        int index = res.size()-1;
        while(index >= 0)
            res.add(res.get(index--) + (int)Math.pow(2,c-1));

        c++;
    }
    return res;
}
```

091-100

092 Reverse Linked List II

92. Reverse Linked List II

难度 中等 418

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

Note: 1 ≤ m ≤ n ≤ length of list.

Example:

```
Input: 1->2->3->4->5->NULL, m = 2, n = 4
Output: 1->4->3->2->5->NULL
```

```java
public ListNode reverseBetween(ListNode head, int left, int right) {
    ListNode before = head, after = head;
    ListNode tail = null;

    ListNode cur = head;
    if(left != 1) {
        for (int i = 1; i < left - 1; i++) {
            cur = cur.next;
        }

        before = cur.next;
    }

    ListNode prev = null;
    for(int i = left; i <= right; i++){
        ListNode temp = before.next;
        before.next = prev;
        prev = before;

        if(i == left)
            tail = before;

        if(i != right)
            before = temp;
        else{
            after = temp;
            cur.next = before;
        }
    }

    tail.next = after;
    if(left == 1)
        return before;
    return head;
}

```

```java
class Solution {
    public ListNode reverseBetween(ListNode head, int m, int n) {
        ListNode newHead = new ListNode(0);
        newHead.next = head;

        //base case
        ListNode cur = newHead;
        if(head == null || head.next == null)        return head;

        //before is the (m-1)th Node
        //after is the (n+1)th Node
        ListNode before = null, after = null;

        //encounter the node we need to reverse first
        for(int i = 1; i <= m-1; i++)
            cur = cur.next;
        before = cur;
        cur = cur.next;

        //begin to reverse the list from m to n
        before.next   = null;
        ListNode pre  = null;
        ListNode temp = null;

        for(int i = m; i <= n; i++)
        {
            if(cur == null)    break;
            if(i == n)         after = cur.next;

            temp = cur.next;
            cur.next = pre;
            pre = cur;
            cur = temp;
        }

        //get them binding
        before.next = pre;
        while(pre.next != null)
            pre = pre.next;
        pre.next = after;

        return newHead.next;
    }
}
```

93 Restore IP Address

执行用时： 0 ms ，在所有 Go 提交中击败了 100.00% 的用户

内存消耗： 2.4 MB ，在所有 Go 提交中击败了 15.92% 的用户

```go
func restoreIpAddresses(s string) []string {
    res := make([]string, 0)
    if len(s) > 12{
        return res
    }

  temp := ""
  backtrack(&res, s, 0, temp, 0)

  return res
}

func backtrack(res *[]string, s string, index int, path string, num int){
  if index == len(s){
    if num == 4{
      *res = append(*res, path)
    }
    return
  }

  for i := index; i < len(s); i++{
    frac := s[index : i + 1]

    if number, err := strconv.Atoi(frac); err == nil{
      if frac[0] == '0' && len(frac) != 1{
        continue
      }

      if 0 <= number && number <= 255{
        length := len(path)

        if num == 3{
          path += strconv.Itoa(number)
        }else{
          path += strconv.Itoa(number) + "."
        }


```

```
39          backtrack(res, s, i + 1, path, num + 1)
40
41          path = path[0 : length]
42        }
43      }
44    }
45
46  }
```

095 Unique Binary Search Tree II

95. Unique Binary Search Trees II

难度 中等 456

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

Example:

```
Input: 3
Output:
[
  [1,null,3,2],
  [3,2,null,1],
  [3,1,null,null,2],
  [2,1,3],
  [1,null,2,null,3]
]
Explanation:
The above output corresponds to the 5 unique BST's shown below:

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

```go
type TreeNode struct {
      Val int
      Left *TreeNode
      Right *TreeNode
  }

func generateTrees(n int) []*TreeNode {
  return getRes(1, n)
}

func getRes(start int, end int) []*TreeNode{
  if start > end{
    return []*TreeNode{nil}
  }

  allTrees := []*TreeNode{}

  for i := start; i <= end; i++{
    leftTrees  := getRes(0, i - 1)
    rightTrees := getRes(i + 1, end)

    for _, left := range leftTrees{
      for _, right := range rightTrees{
        root  := &TreeNode{i, nil, nil}
        root.Left = left
        root.Right = right
        allTrees = append(allTrees, root)
      }
    }
  }

  return allTrees
}

```

```java
public List<TreeNode> generateTrees(int n) {
    List<TreeNode> ans = new ArrayList<>();
    if(n == 0)      return ans;

    return getAns(1,n);
}

private List<TreeNode> getAns(int start, int end) {
    List<TreeNode> ans = new ArrayList<>();
    if (start > end) {
        ans.add(null);

```

```
        return ans;
    }

    if (start == end)
    {
        TreeNode tree = new TreeNode(start);
        ans.add(tree);
        return ans;
    }

    for(int i = start; i <= end; i++)
    {
        List<TreeNode> leftTrees = getAns(start, i-1);
        List<TreeNode> rightTrees = getAns(i+1, end);

        for(TreeNode leftTree : leftTrees)
        {
            for(TreeNode rightTree : rightTrees)
            {
                TreeNode root = new TreeNode(i);
                root.left = leftTree;
                root.right = rightTree;
                ans.add(root);
            }
        }
    }
    return ans;
}
```

096 Unique Binary Search Trees

1 分钟前　　通过　　0 ms　　35.1 MB　　Ja

```
class Solution {
    HashMap<Integer, Integer> map;
    public int numTrees(int n) {
        map = new HashMap<>();
        map.put(0, 1);
        map.put(1, 1);
        map.put(2, 2);
        map.put(3, 5);

```

```
        helper(n);
        return map.get(n);
    }

    /*
    *   用来求 到底有多少种
    * */
    private int helper(int n) {
        if(map.containsKey(n))
            return map.get(n);


        int count = 0;
        for(int i = 1; i <= n; i++){
            int left = helper(i - 1);
            int right = helper(n - i);

            count += left * right;
        }
        map.put(n, count);
        return count;
    }


}
```

097 Interleaving String

97. Interleaving String

难度 困难 196

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

Example 1:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
Output: true
```

Example 2:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
Output: false
```

执行结果: 通过 显示详情 >

执行用时: 9 ms , 在所有 Java 提交中击败了 21.15% 的用户

内存消耗: 37.2 MB , 在所有 Java 提交中击败了 10.32% 的用户

炫耀一下:

```java
class Solution {
    public boolean isInterleave(String s1, String s2, String s3) {
        int len1 =  s1.length(), len2 = s2.length(), len3 = s3.length();
        if(len1 + len2 != len3) return false;
        /*
        *   s1 [0...i]
        *   s2 [0...j]  --> s3
        * */
        boolean[][] dp = new boolean[s1.length() + 1][s2.length() + 1];
        dp[0][0] = true;



        for(int i = 0; i < len1 + 1; i++)
            for(int j = 0; j < len2 + 1; j++){
                if(i == 0 && j == 0)
                    dp[i][j]  = true;
                else if(j == 0)
                    dp[i][j] = s1.substring(0, i).equals(s3.substring(0, i));
                else if(i == 0)
                    dp[i][j] = s2.substring(0, j).equals(s3.substring(0, j));
                else
                    dp[i][j] = (s1.charAt(i - 1) == s3.charAt(i + j - 1) && dp[i-1][j])
```

```
                            || (s2.charAt(j - 1) == s3.charAt(i + j - 1) && dp[i][j-1]);
            }


        return dp[s1.length()][s2.length()];
    }
}
```

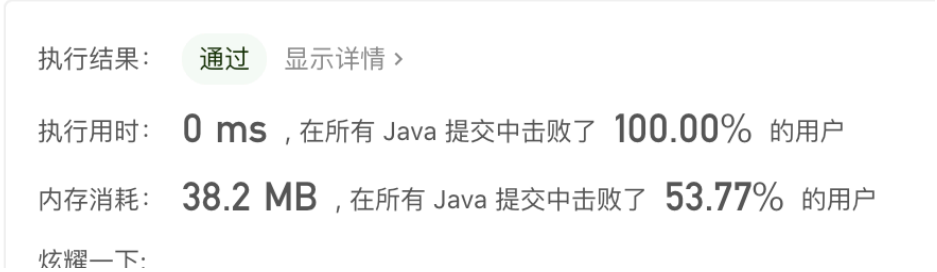
状态方程分析

```
dp[i][j] 代表使用s1 [0...i] 和s2[0...j] 个字符，组成s3的 i+j+2个字符
从而进行判断
```

```
public boolean isInterleave(String s1, String s2, String s3) {
    int len1 = s1.length();
    int len2 = s2.length();

    if(len1 + len2 != s3.length())      return false;
    boolean[][] dp= new boolean[len1+1][len2+1];

    for(int i = 0; i <= len1; i++)
        for(int j = 0; j <= len2; j++)
        {
            if(i == 0 && j == 0)
                dp[i][j] = true;
            else if(i == 0)
                dp[i][j] = s2.charAt(j-1) == s3.charAt(i+j-1) && dp[i][j-1]? true : false;
            else if(j == 0)
                dp[i][j] = s1.charAt(i-1) == s3.charAt(i+j-1) && dp[i-1][j]? true : false;
            else
                dp[i][j] =   s2.charAt(j-1) == s3.charAt(i+j-1) && dp[i][j-1]
                || s1.charAt(i-1) == s3.charAt(i+j-1) && dp[i-1][j];
        }
    return dp[len1][len2];
}
```

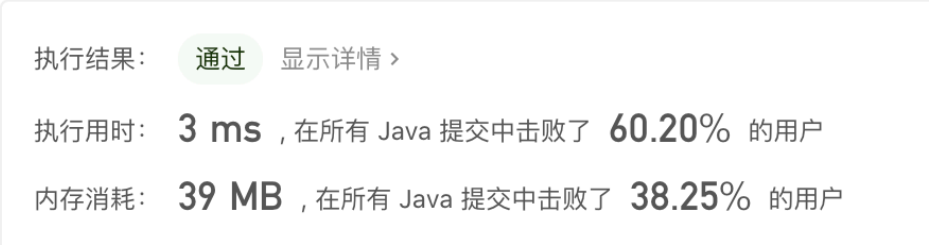
```java
/*
  TLE
*/
    public boolean isInterleave(String s1, String s2, String s3) {
        if(s1.length() + s2.length() != s3.length())
            return false;

        if(s1.length() == 0 || s2.length() == 0)
        {
            if(s1.length() == 0)
                return s2.equals(s3);
            else if(s2.length() == 0)
                return s1.equals(s3);
        }

        if(s1.charAt(0) == s3.charAt(0) && s2.charAt(0) == s3.charAt(0))
            return isInterleave(s1.substring(1), s2, s3.substring(1)) ||
                isInterleave(s1, s2.substring(1), s3.substring(1));

        else if(s1.charAt(0) == s3.charAt(0))
            return isInterleave(s1.substring(1), s2, s3.substring(1));

        else if(s2.charAt(0) == s3.charAt(0))
            return isInterleave(s1, s2.substring(1), s3.substring(1));
        else
            return false;
    }
}
```

98 Validate Binary Search Tree



```java
//核心是中序遍历
//可以直接递归，  不需要helper function
long pre = Long.MIN_VALUE;
    public boolean isValidBST(TreeNode root) {
        if(root == null)    return true;

        boolean flag1 = isValidBST(root.left);

        if(pre != Long.MIN_VALUE){
            if(pre >= root.val)
                return false;
        }

        pre = root.val;

        boolean flag2 =  isValidBST(root.right);

        return flag1 && flag2;
    }

```

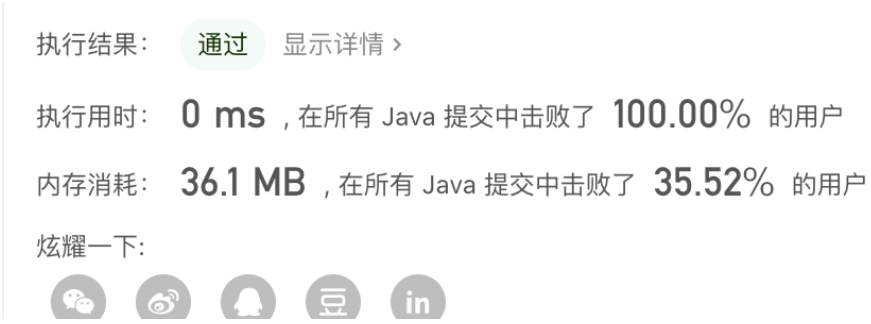
99 Recover Binary Search Tree




```java
/*
  思路就是中序的非递归遍历
*/
class Solution {
    TreeNode firstError  = null;
    TreeNode secondError = null;
    TreeNode pre = new TreeNode(Integer.MIN_VALUE);
    boolean flag = true;
    public void recoverTree(TreeNode root) {

```

```
        Deque<TreeNode> stack = new ArrayDeque<>();

        while(root != null || !stack.isEmpty()){
            while(root != null){
                stack.add(root);
                root = root.left;
            }

            root = stack.pollLast();
            //res.add(cur.val);
            if(pre.val != Integer.MIN_VALUE){
                if(pre.val >= root.val && flag){
                    firstError  = pre;
                    secondError = root;
                    flag = false;
                }else if(pre.val >= root.val && !flag){
                    secondError = root;
                }
            }

            pre = root;
            root = root.right;
        }

        int temp = firstError.val;
        firstError.val = secondError.val;
        secondError.val = temp;
    }
}
```

100 Same Tree




```
//简单题 秒了
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if(p == null || q == null){
```

```
            if(p == null && q == null)
                return true;
            else
                return false;
        }

        if(p.val != q.val)  return false;

        return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```