401-500

401 Binary Watch

401. Binary Watch

难度 简单 194

A binary watch has 4 LEDs on the top which represent the **hours** (**0-11**), and the 6 LEDs on the bottom represent the **minutes** (**0-59**).

Each LED represents a zero or one, with the least significant bit on the right.

For example, the above binary watch reads "3:25".

Given a non-negative integer *n* which represents the number of LEDs that are currently on, return all possible times the watch could represent.

Example:

```
Input: n = 1
Return: ["1:00", "2:00", "4:00", "8:00", "0:01",
```

执行用时： 18 ms , 在所有 Java 提交中击败了 13.25% 的用户

内存消耗： 38.4 MB , 在所有 Java 提交中击败了 35.52% 的用户

```java
class Solution {
    public List<String> readBinaryWatch(int num) {
        Map<Integer, List<String>> hour = new HashMap<>();
        Map<Integer, List<String>> min  = new HashMap<>();

        for(int i = 0; i < 12; i++){
            int howManyOnes = binaryOne(i);
            hour.putIfAbsent(howManyOnes, new ArrayList<>());
            hour.get(howManyOnes).add(i + "");
        }

        for(int i = 0; i <= 59; i++){
            int howManyOnes = binaryOne(i);
            min.putIfAbsent(howManyOnes, new ArrayList<>());

            if(i < 10)
                min.get(howManyOnes).add("0" + i);
            else
                min.get(howManyOnes).add(i + "");
        }

        List<String> res = new ArrayList<>();
        for(int i = 0; i <= num; i++){
            if(!hour.containsKey(i) || !min.containsKey(num - i)) {
                continue;
            }
            for(String h : hour.get(i)){
                for(String m : min.get(num - i)){
                    res.add(h + ":" + m);
                }
            }
        }

        return res;
    }

    private int binaryOne(int num){
        int counter = 0;
        while(num != 0){
            if((num & 1)== 1)
                counter++;

            num >>= 1;
        }

        return counter;
    }
}
```

```java
/*
  author: 就是一道常规打表题
  将不同情况提前计算好， 然后直接上就完事了
*/
class Solution {
    public List<String> readBinaryWatch(int num) {
        List<String> res = new ArrayList<>();

        int[] hours = {0,1,2,3,4,5,6,7,8,9,10,11};
        int[] mins  = new int[60];
        for(int i = 0; i < mins.length; i++)
            mins[i] = i;

        HashMap<Integer, ArrayList<Integer>> hMap = new HashMap<>();
        HashMap<Integer, ArrayList<Integer>> mMap = new HashMap<>();

        getRes(hMap, hours);
        getRes(mMap, mins);

        for(int i = 0; i <= num; i++){
            if(i >= hMap.size() || num - i >= mMap.size())    continue;
            for(Integer hour : hMap.get(i))
                for(Integer minute : mMap.get(num - i))
                    res.add(hour + ":" + (minute < 10 ? ("0" + minute) : minute + ""));
        }


        return res;
    }

    private static void getRes(HashMap<Integer, ArrayList<Integer>> map, int[] nums) {
        int mode = nums.length == 12 ? 4 : 6;
        for(int num : nums){
            int res = getOnes(num, mode);
            if(!map.containsKey(res))
                map.put(res, new ArrayList<>());
            map.get(res).add(num);
        }
    }

    private static int getOnes(int num, int mode){
        int count = 0;
        for(int i = 0; i < mode; i++){
```

```
44              count += ((num & 1) == 1 ? 1 : 0);
45              num >>= 1;
46          }
47
48          return count;
49      }
50  }
```

402 Remove K digits 单调栈的应用

402. Remove K Digits

难度 中等 480

Given a non-negative integer *num* represented as a string, remove *k* digits from the number so that the new number is the smallest possible.

Note:

- The length of *num* is less than 10002 and will be ≥ *k*.
- The given *num* does not contain any leading zero.

Example 1:

```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2
to form the new number 1219 which is the smallest.
```

Example 2:

```
Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number
is 200. Note that the output must not contain
leading zeroes.
```

Example 3:

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number
and it is left with nothing which is 0.
```

执行用时： 4 ms , 在所有 C++ 提交中击败了 80.87% 的用户

内存消耗： 7 MB , 在所有 C++ 提交中击败了 45.24% 的用户

炫耀一下:

```cpp
class Solution {
public:
    string removeKdigits(string num, int k) {
        int index = 0;
        deque<char> myQueue;
        while(index < num.size()){
```

```
            while(index < num.size() && (myQueue.empty() || myQueue.back() <= num[index])){
                myQueue.push_back(num[index]);
                index++;
            }

            if(index == num.size())
                break;

            while(index < num.size() && !myQueue.empty()&&myQueue.back() > num[index] && myQueue.size() + num.size() - index - 1 >= num.size() - k){
                myQueue.pop_back();
            }

            myQueue.push_back(num[index]);
            index++;
        }

        string res = "";
        bool seeZero = true;
        for(int i = 0; i < num.size() - k; i++){

            res += myQueue.front();
            myQueue.pop_front();
        }

        if(res == "" || res == "0")
            return "0";

        index = 0;
        if(res[0] == '0'){
            for(int i = 0; i < res.size(); i++){
                if(res[i] == '0'){
                    index = i;
                }else{
                    break;
                }
            }
            auto check = [&](){
                for(char ch : res){
                    if(ch != '0')
                        return false;
                }
                    return true;
                };
            return check() ? "0" : res.substr(index + 1);
        }else{
             return res;
        }
```

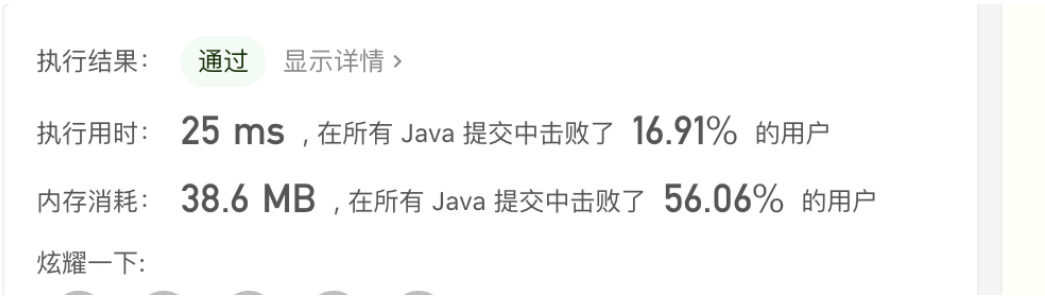
```
    }
};
```

执行用时： 13 ms ，在所有 Java 提交中击败了 28.36% 的用户

内存消耗： 38.8 MB ，在所有 Java 提交中击败了 44.38% 的用户

```java
/*
    二刷
*/
public String removeKdigits(String num, int k) {
        char[] chars = num.toCharArray();

        Deque<Character> stack = new ArrayDeque<>();
        stack.addLast('0');

        int index = 0;
        int count = num.length() - k;
        if(count == 0)
            return "0";

        while(index < num.length()){
            while(index < num.length() && stack.peekLast() > chars[index]
                    && stack.size() - 1  + num.length() - index - 1 >= count){
                stack.removeLast();
            }

            if(index == num.length())
                break;

            stack.addLast(chars[index++]);
        }

        StringBuilder sb = new StringBuilder();
        while(!stack.isEmpty() && stack.peekFirst() == '0')
            stack.removeFirst();
        if(stack.isEmpty())
            return "0";

        for(int i = 0; !stack.isEmpty() && i < count; i++)
            sb.append(stack.removeFirst());

```

```
        return sb.toString();
    }

```




```java
/*
    这种算法可以，但是太慢了
*/
public String removeKdigits(String num, int k) {
  if(num.length() == k)   return "0";

  String res = num;
  for(int i = 0; i < k; i++){

    StringBuilder sb = new StringBuilder();
    int j = 0;
    while(j < res.length() - 1)
      if(res.charAt(j) <= res.charAt(j + 1))
        j++;
    else
      break;
    int p = 0;
    while(res.charAt(p) == '0') p++;
    sb.append(res.substring(p, j)).append(res.substring(j + 1));
    res = sb.toString();
  }

  int p = 0;
  while(p < res.length() && res.charAt(p) == '0') p++;
  return p == res.length() ? "0" : res.substring(p);
}
```

```java
/*
    典型单调栈的应用

    单调递增栈
*/
public String removeKdigits(String num, int k) {
  if(num.length() == k)       return "0";

  StringBuilder stack = new StringBuilder();
  int remains = num.length() - k;

  for(int i = 0; i < num.length(); i++){
    char ch = num.charAt(i);
    while(k > 0 && stack.length() != 0 && stack.charAt(stack.length() - 1) > ch){
      stack.setLength(stack.length() - 1);
      k--;
    }
    if(ch == '0' && stack.length() == 0)    continue;

    stack.append(ch);
  }
  String res = stack.substring(0, stack.length() - k).toString();
  return res.length() == 0 ? "0" : res;
}
```

403 Frog Jump

```java
//二刷 优化， 采用记忆化
class Solution {
    Map<String, Boolean> map = new HashMap<>();
    public boolean canCross(int[] stones) {
        return dfs(stones, 0, 0);
    }

    private boolean dfs(int[] stones, int start, int lastJump){
        if(start == stones.length - 1){
            return true;
        }

        String symbol = (start + "@" + lastJump);
        if(map.containsKey(symbol))
            return map.get(symbol);

```

```
        for(int i = start + 1; i < stones.length; i++){
            int distance = stones[i] - stones[start];
            if(distance >= lastJump - 1 && distance <= lastJump + 1){
                if(dfs(stones, i, distance))
                    return true;
            }
        }

        map.put(symbol, false);
        return false;
    }
}
```

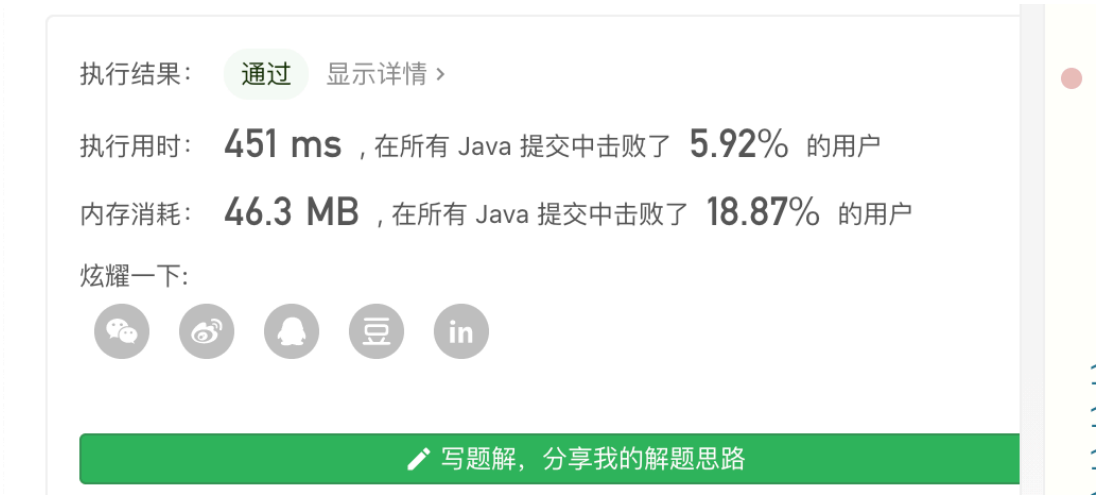
```
//二刷， 回溯超时 16/45
public boolean canCross(int[] stones) {
     return dfs(stones, 0, 0);
    }

    private boolean dfs(int[] stones, int start, int lastJump){
        if(start == stones.length - 1){
            return true;
        }

        for(int i = start + 1; i < stones.length; i++){
            int distance = stones[i] - stones[start];
            if(distance >= lastJump - 1 && distance <= lastJump + 1){
                if(dfs(stones, i, distance))
                    return true;
            }
        }

        return false;
    }
```

```
/*
  过肯定过不去， 超时了
*/
public boolean canCross(int[] stones) {
  if(stones.length < 2)        return true;
  if(stones[1] != 1)           return false;
```

```
  return backtrack(stones, 1, 1);
}

private boolean backtrack(int[] stones, int start, int steps) {
  if(start == stones.length - 1)
    return true;

  boolean b1 = false, b2 = false, b3 = false;
  for(int i = start + 1; i < stones.length; i++)
    if(stones[i] - stones[start] == steps)
      b1 = backtrack(stones, i, steps);
    else if(stones[i] - stones[start] == steps + 1)
      b2 = backtrack(stones, i, steps + 1);
    else if(stones[i] - stones[start] == steps - 1)
      b3 = backtrack(stones, i, steps - 1);

  return b1 || b2 || b3;
}
```


```
/*
    哈哈我是没想到真能过，但是太慢了，找个好方法
*/
class Solution {
    HashMap<String, Boolean> map = new HashMap<String, Boolean>();
    public boolean canCross(int[] stones) {
        if(stones.length < 2)        return true;
        if(stones[1] != 1)           return false;
        return backtrack(stones, 1, 1);
    }

    private boolean backtrack(int[] stones, int start, int steps) {
        if(start == stones.length - 1)
            return true;
        StringBuilder sb = new StringBuilder();
        String res = sb.append(start).append("@").append(steps).toString();
```

```
        if(map.containsKey(res))    return map.get(res);

        boolean b1 = false, b2 = false, b3 = false;
        for(int i = start + 1; i < stones.length; i++)
            if(stones[i] - stones[start] == steps)
                b1 = backtrack(stones, i, steps);
            else if(stones[i] - stones[start] == steps + 1)
                b2 = backtrack(stones, i, steps + 1);
            else if(stones[i] - stones[start] == steps - 1)
                b3 = backtrack(stones, i, steps - 1);


        map.put(res, b1 || b2 || b3);
        return b1 || b2 || b3;
    }
}
```




```
/*

  dp 解法
Exapme 1:

index:         0   1   2   3   4   5   6   7
             +---+---+---+---+---+---+---+---+
stone pos:   | 0 | 1 | 3 | 5 | 6 | 8 | 12| 17|
             +---+---+---+---+---+---+---+---+
k:           | 1 | 0 | 1 | 1 | 0 | 1 | 3 | 4 |
             |   | 1 | 2 | 2 | 1 | 2 | 4 | 5 |
             |   | 2 | 3 | 3 | 2 | 3 | 5 | 6 |
             |   |   |   |   | 3 | 4 |   |   |
             |   |   |   |   | 4 |   |   |   |
             |   |   |   |   |   |   |   |   |

  */
```

```java
public boolean canCross(int[] stones) {
  int N = stones.length;
  //1st-arg: index
  //2st-arg: available steps
  HashMap<Integer, HashSet<Integer>> map = new HashMap<>();
  map.put(0, new HashSet<>());
  map.get(0).add(1);

  for(int i = 1; i < N; i++){
    map.put(i, new HashSet<>());
    for(int j = 0; j < i; j++){
      int diff = stones[i] - stones[j];
      if(map.get(j).contains(diff)){
        map.get(i).add(diff);
        if(diff + 1 < N)    map.get(i).add(diff + 1);
        if(diff - 1 >= 0)   map.get(i).add(diff - 1);
      }
    }
  }

  return map.get(N - 1).size() != 0;
}
```

404 Sum of Left Leaves

```go
func sumOfLeftLeaves(root *TreeNode) int {
  sum := 0
  dfs(root, &sum)
  return sum
}

func dfs(root *TreeNode, sumP *int) {
  if root == nil{
    return
  }

  if root.Left == nil && root.Right == nil{
    return
  }

  if root.Left != nil{
    if root.Left.Left == nil && root.Left.Right == nil{
      *sumP += root.Left.Val
    }
  }

```

```
  dfs(root.Left, sumP)
  dfs(root.Right, sumP)
}
```

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 55.69% 的用户

炫耀一下:

```java
//二刷
class Solution {
    int res = 0;
    public int sumOfLeftLeaves(TreeNode root) {
        inorder(root) ;
        return res;
    }

    private void inorder(TreeNode root){
        if(root == null)
            return;

        if(root.left != null && root.left.left == null && root.left.right == null)
            res += root.left.val;

        inorder(root.left);
        inorder(root.right);

    }
}
```


```java
class Solution {
    int total = 0;
    public int sumOfLeftLeaves(TreeNode root) {
        dfs(root);
        return total;
    }

    public void dfs(TreeNode root){
        if(root == null)    return;

        if(root.left != null)
            if(root.left.left == null && root.left.right == null)
                total += root.left.val;

        dfs(root.left);
        dfs(root.right);
    }
}
```

405 Convert a Number to Hexadecimal

```java
//二刷
class Solution {
    public String toHex(int num) {
        if(num == 0)
            return "0";
        String[] strs = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"};

```

```
        /*

        */
        StringBuilder sb = new StringBuilder();
        while(num != 0){
            int temp = num & (0xF);
            sb.append(strs[temp]);
            num >>>= 4;
        }

        return sb.reverse().toString();
    }

}
```


```
public String toHex(int num) {
  if(num == 0)        return "0";
  String[] s = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"};
  StringBuilder sb = new StringBuilder();

  while(num != 0){
    int c = num % 16;
    sb.append(s[c >= 0 ? c : c + 16]);
    num >>>= 4;
  }

  return sb.reverse().toString();
}
```

406 Queue Reconstruction by Height 很强的排序题目 多关注一下

406. Queue Reconstruction by Height

难度 中等 716

You are given an array of people, `people`, which are the attributes of some people in a queue (not necessarily in order). Each `people[i] = [hi, ki]` represents the `i^th` person of height `hi` with **exactly** `ki` other people in front who have a height greater than or equal to `hi`.

Reconstruct and return *the queue that is represented by the input array* `people`. The returned queue should be formatted as an array `queue`, where `queue[j] = [hj, kj]` is the attributes of the `j^th` person in the queue (`queue[0]` is the person at the front of the queue).

Example 1:

```
Input: people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]
Output: [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]
Explanation:
Person 0 has height 5 with no other people taller or the same height in front.
Person 1 has height 7 with no other people taller or the same height in front.
Person 2 has height 5 with two persons taller or the same height in front, which is person 0 and 1.
Person 3 has height 6 with one person taller or the same height in front, which is person 1.
Person 4 has height 4 with four people taller or the same height in front, which are people 0, 1,
```

```
1  /**
2   * 解题思路：先排序再插入
3   * 1.排序规则：按照先H高度降序，K个数升序排序
4   * 2.遍历排序后的数组，根据K插入到K的位置上
5   *
6   * 核心思想：高个子先站好位，矮个子插入到K位置上，前面肯定有K个高个子，矮个子再插到前面也满足K的要求
7   *
8   * @param people
9   * @return
10  *
11  同时注意， 一般看到这种对数对的处理， 还涉及排序的， 根据第一个元素正向排序， 第二个元素反向排序
```

```
     或者返回来，  都可以简化解题过程
     */
    public int[][] reconstructQueue(int[][] people) {
        // [7,0], [7,1], [6,1], [5,0], [5,2], [4,4]
        // 再一个一个插入。
        // [7,0]
        // [7,0], [7,1]
        // [7,0], [6,1], [7,1]
        // [5,0], [7,0], [6,1], [7,1]
        // [5,0], [7,0], [5,2], [6,1], [7,1]
        // [5,0], [7,0], [5,2], [6,1], [4,4], [7,1]
        Arrays.sort(people, (o1, o2) -> o1[0] == o2[0] ? o1[1] - o2[1] : o2[0] - o1[0]);

        LinkedList<int[]> list = new LinkedList<>();
        for (int[] i : people) {
            list.add(i[1], i);
        }

        return list.toArray(new int[list.size()][2]);
    }


作者：pphdsny
链接：https://leetcode-cn.com/problems/queue-reconstruction-by-height/solution/406-gen-ju-shen-gao-zhong-jian-dui-lie-java-xian-p/
```

407 Trapping Rain Water 3D

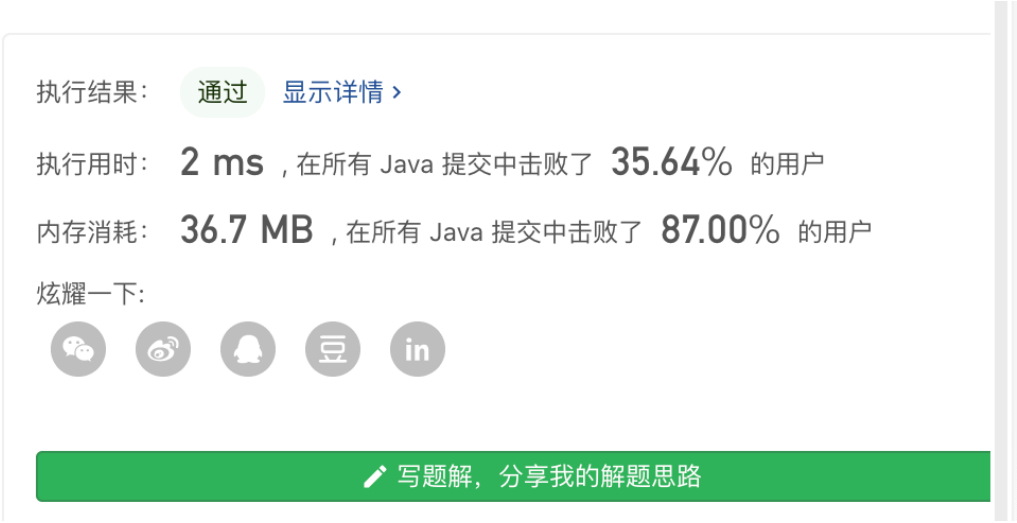
408 Valid Word Abbreviation

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： 36.6 MB , 在所有 Java 提交中击败了 90.43% 的用户

```
    public boolean validWordAbbreviation(String word, String abbr) {
        int up = 0, down = 0;
```

```
        int len1 = word.length();
        int len2 = abbr.length();

        while(up != len1 && down != len2){
            while(up != len1 && down != len2 && word.charAt(up) ==
abbr.charAt(down)){
                up++;
                down++;
            }

            if(up == len1 || down == len2)
                break;

            int index = down;
            if(isAlpha(abbr, down)){
                while(down < len2 && isAlpha(abbr, down)){
                    down++;
                }

                int num = Integer.parseInt(abbr.substring(index, down));
                if((abbr.charAt(index) == '0' && num != 0) || (num == 0))
                    return false;
                up += num;
            }else{
                return false;
            }
        }

        return up == len1 && down == len2;
    }

    private boolean isAlpha(String string, int index){
        return string.charAt(index) >= '0' && string.charAt(index) <= '9';
    }
```




```java
class Solution {
    public boolean validWordAbbreviation(String word, String abbr) {
        int index = 0;
        for(int i  = 0 ; i < abbr.length();){
            int j = i;
            if(j < abbr.length() && abbr.charAt(j) >= '1' && abbr.charAt(j) <= '9')
                while(j < abbr.length() && abbr.charAt(j) >= '0' && abbr.charAt(j) <= '9')
                    j++;

            if(index >= word.length())      return false;
            if(j != i){
                index += Integer.parseInt(abbr.substring(i, j));
                if(index > word.length())  return false;
                i = j;
            }
            else{
                if(abbr.charAt(j) != word.charAt(index))
                    return false;
                index++;
                i++;
            }
        }

        return index == word.length();
    }
}
```

409 Longest Palindrome

执行结果： 通过 显示详情 >

执行用时： 8 ms ，在所有 Java 提交中击败了 30.68% 的用户

内存消耗： 37 MB ，在所有 Java 提交中击败了 28.61% 的用户

炫耀一下：

```java
//二刷
public int longestPalindrome(String s) {
        Map<Character, Integer> map = new HashMap<>();
        for(char ch : s.toCharArray())
            map.put(ch, map.getOrDefault(ch, 0) + 1);

        boolean seenOdd = false;


        int len = 0;
        for(Character ch : map.keySet()){
            if(map.get(ch) % 2 != 0){
                seenOdd = true;
                len += (map.get(ch) - 1);
            }else if(map.get(ch) % 2 == 0){
                len += map.get(ch);
            }
        }

        return len + (seenOdd ? 1 : 0);
    }
```

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 20.01% 的用户

内存消耗： 36.5 MB ，在所有 Java 提交中击败了 92.00% 的用户

炫耀一下：

写题解，分享我的解题思路

```java
class Solution {
    public int longestPalindrome(String s) {
        int len = 0;
```

```
        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);

        boolean odd = false;
        for(Map.Entry<Character, Integer> entry : map.entrySet()) {
            if (entry.getValue() % 2 == 0 || entry.getValue() > 1)
                len += entry.getValue() % 2 == 0 ? entry.getValue() :
entry.getValue() - 1;

            if (!odd && entry.getValue() % 2 != 0) {
                len += 1;
                odd = !odd;
            }
        }

        return len;
    }
}
```

410 Split Array Largest Sum 二分查找应用 非排序数组 没有彻底搞懂，需要再研究

410. Split Array Largest Sum

难度 困难 392

Given an array `nums` which consists of non-negative integers and an integer `m`, you can split the array into `m` non-empty continuous subarrays.

Write an algorithm to minimize the largest sum among these `m` subarrays.

Example 1:

```
Input: nums = [7,2,5,10,8], m = 2
Output: 18
Explanation:
There are four ways to split nums into two subarrays.
The best way is to split it into [7,2,5] and [10,8],
where the largest sum among the two subarrays is only 18.
```

Example 2:

```
Input: nums = [1,2,3,4,5], m = 2
Output: 9
```

Example 3:

```
Input: nums = [1,4,4], m = 3
Output: 4
```

```java
/*
    对本题的理解
    https://leetcode-cn.com/problems/split-array-largest-sum/solution/bai-hua-er-fen-cha-zhao-by-xiao-yan-gou/
      有一个数组，  需要分割成 m 份，  每一份都有一个和
      让这些和最大值最小
*/
public class Solution {

    public int splitArray(int[] nums, int m) {
        int max = 0;
        int sum = 0;

```

```java
        // 计算「子数组各自的和的最大值」的上下界
        for (int num : nums) {
            max = Math.max(max, num);
            sum += num;
        }

        // 使用「二分查找」确定一个恰当的「子数组各自的和的最大值」，
        // 使得它对应的「子数组的分割数」恰好等于 m
        int left = max;
        int right = sum;
        while (left < right) {
            int mid = left + (right - left) / 2;

            int splits = split(nums, mid);
            if (splits > m) {
                // 如果分割数太多，说明「子数组各自的和的最大值」太小，此时需要将「子数组各自的和的最大值」调大
                // 下一轮搜索的区间是 [mid + 1, right]
                left = mid + 1;
            } else {
                // 下一轮搜索的区间是上一轮的反面区间 [left, mid]
                right = mid;
            }
        }
        return left;
    }

    /***
     *
     * @param nums 原始数组
     * @param maxIntervalSum 子数组各自的和的最大值
     * @return 满足不超过「子数组各自的和的最大值」的分割数
     */
    private int split(int[] nums, int maxIntervalSum) {
        // 至少是一个分割
        int splits = 1;
        // 当前区间的和
        int curIntervalSum = 0;
        for (int num : nums) {
            // 尝试加上当前遍历的这个数，如果加上去超过了「子数组各自的和的最大值」，就不加这个数，另起炉灶
            if (curIntervalSum + num > maxIntervalSum) {
                curIntervalSum = 0;
                splits++;
            }
            curIntervalSum += num;
        }
        return splits;
    }
```

```
}

作者：liweiwei1419
链接：https://leetcode-cn.com/problems/split-array-largest-sum/solution/er-fen-cha-zhao-by-liweiwei1419-4/
```

412 Fuzz Bizz


```java
class Solution {
    public List<String> fizzBuzz(int n) {
        List<String> res = new ArrayList<>();

        for(int i = 1; i <= n; i++){
            if(i % 3 == 0 || i % 5 == 0){
                if(i % 3 != 0)
                    res.add("Buzz");
                else if(i % 5 != 0)
                    res.add("Fizz");
                else
                    res.add("FizzBuzz");

            }else{
                res.add(i + "");
            }

        }

        return res;
    }
}
```

413 Arithmetic Slices

```java
//二刷
class Solution {
    public int numberOfArithmeticSlices(int[] nums) {
        int len = nums.length;

        int[] dp = new int[len];
        int count = 0;
        for(int i = 2; i < len; i++){
            if(nums[i] + nums[i - 2] == 2 * nums[i - 1])
                dp[i] = 1 + dp[i - 1];
            count += dp[i];
        }

        return count;
    }
}
```


```java
//笨办法， 可以通过
public int numberOfArithmeticSlices(int[] A) {
        int count = 0;
        for(int i = 0; i <= A.length - 3; i++)
            for(int j = i + 2; j < A.length; j++)
                if(isArithmetic(A, i, j))
                    count++;

        return count;
    }

    public boolean isArithmetic(int[] A, int i, int j){
```

```
        int diff = A[j] - A[j-1];
        for(int k = i + 1; k <= j; k++)
            if(diff != A[k] - A[k - 1])
                return false;

        return true;

    }
```

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 7.94% 的用户

内存消耗: 36.7 MB , 在所有 Java 提交中击败了 24.64% 的用户

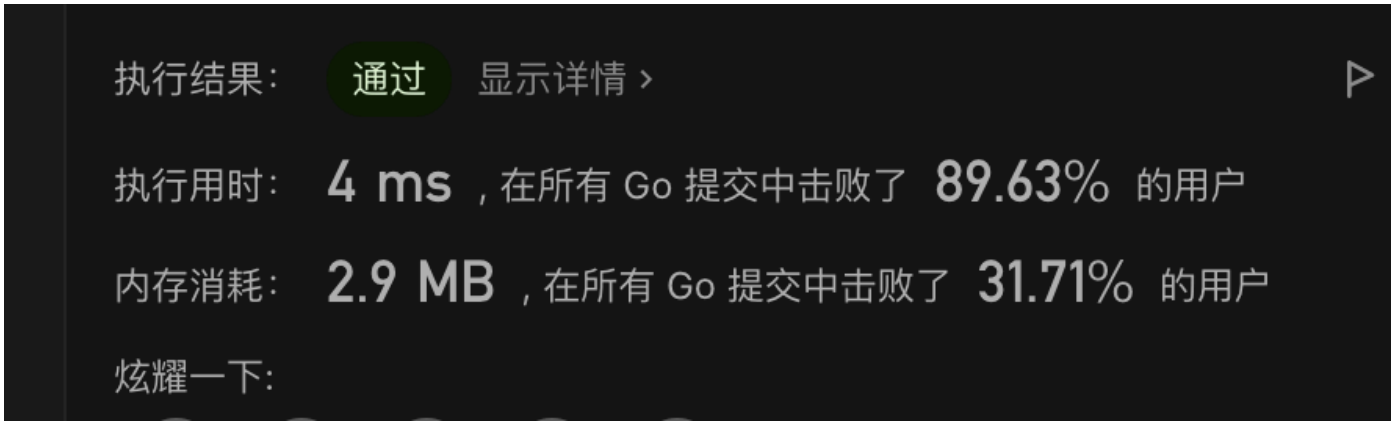
炫耀一下:

```

/*
    采用递归
*/
class Solution {

    public int numberOfArithmeticSlices(int[] A) {
        if(A.length == 0)   return 0;
        int[] diff = new int[A.length - 1];

        for(int i = 0; i < A.length - 1; i++)
            diff[i] = A[i + 1] - A[i];

        return getNumber(diff, 0, diff.length - 1);
    }

    public int getNumber(int[] diff, int start, int end){
        if(end - start < 1)
            return 0;

        int longestArithNumber = 0;
        int left = start, right = start;
        int l = 0, r = 0;
        while(right <= end){
            while(right <= end && diff[right] == diff[left])
                right++;
            if(right - left > longestArithNumber){
```

```
                longestArithNumber = right -left;
                l = left;
                r = right;
            }

            left = right;
        }
        longestArithNumber++;
        int res = (((longestArithNumber - 2) + 1) * (longestArithNumber - 2))/ 2;
        return res + getNumber(diff, start, l - 1) + getNumber(diff, r, end);
    }
}
```

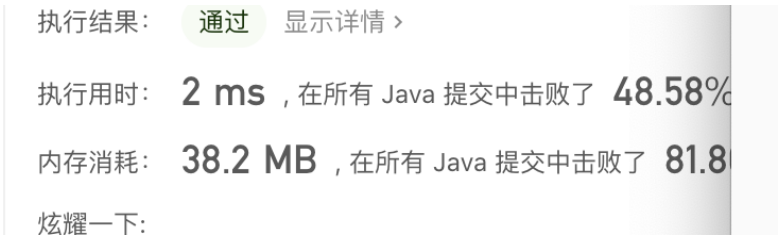
```
/*
    蛮不错的方法

    采用等差数列的性质
*/
class Solution {
public:
    int numberOfArithmeticSlices(vector<int>& A) {
        const int N = A.size();
        if(N < 3){
            return 0;
        }
        int ans = 0;
        vector<int> d(N,0);
        for(int i=2;i<N;++i){
            if(A[i]-A[i-1] == A[i-1]-A[i-2]){
                d[i]=1+d[i-1];
            }
            ans += d[i];
        }
        return ans;
    }
};

作者：jason-2
链接：https://leetcode-cn.com/problems/arithmetic-slices/solution/dong-tai-gui-hua-by-jason-2-a69j/
```

414 第三大的数




```go
func thirdMax(nums []int) int {
  FirstMax  := -2147483649
  SecondMax := -2147483649
  ThirdMax  := -2147483649


  for i := 0; i < len(nums); i++{
    if FirstMax == -2147483649{
      FirstMax = nums[i]
    }else if FirstMax > nums[i]{
      if SecondMax == -2147483649{
        SecondMax = nums[i]
      }else if SecondMax > nums[i]{
        ThirdMax = max(ThirdMax, nums[i])
      }else if SecondMax < nums[i]{
        ThirdMax = max(SecondMax, ThirdMax)
        SecondMax = nums[i]
      }
    }else if FirstMax < nums[i]{
      ThirdMax = SecondMax
      SecondMax = FirstMax
      FirstMax = nums[i]
    }
  }

  if ThirdMax == -2147483649{
    return FirstMax
  }else{
    return ThirdMax
  }
}


func max(a int, b int) int {
  if a > b{
    return a
  }else{
    return b
```

```
    }
}
```




```java
//二刷
public int thirdMax(int[] nums) {
        long firstBig = Long.MIN_VALUE;
        long secondBig = Long.MIN_VALUE;
        long thirdBig = Long.MIN_VALUE;

        for(int i = 0; i < nums.length; i++){
            if(firstBig == nums[i] || secondBig == nums[i] || thirdBig == nums[i])
                continue;

            if(firstBig == Long.MIN_VALUE){
                firstBig = (long)nums[i];
            }else if(nums[i] > firstBig){
                if(secondBig == Long.MIN_VALUE){
                    secondBig = firstBig;
                    firstBig = (long)nums[i];
                }else{
                    thirdBig = secondBig;
                    secondBig = firstBig;
                    firstBig = (long)nums[i];
                }
            }else if(nums[i] > secondBig){
                if(secondBig == Long.MIN_VALUE){
                    secondBig = (long)nums[i];
                }else{
                    thirdBig = secondBig;
                    secondBig = (long)nums[i];
                }
            }else if(nums[i] > thirdBig){
                    thirdBig = (long)nums[i];
            }
        }

        return thirdBig == Long.MIN_VALUE ? (int)firstBig : (int)thirdBig;
    }
```

```java
class Solution {
  public int thirdMax(int[] nums) {
        if (nums.length == 1) {
            return nums[0];
        }
        if (nums.length == 2) {
            return nums[0] > nums[1] ? nums[0] : nums[1];
        }

        long firstMax = Long.MIN_VALUE;
        long secondMax = Long.MIN_VALUE;
        long thirdMax = Long.MIN_VALUE;
        for (int n : nums) {
            if (n > firstMax) {
                thirdMax = secondMax;
                secondMax = firstMax;
                firstMax = n;
            } else if (firstMax == n){
                continue;
            }else  if (n > secondMax) {
                thirdMax = secondMax;
                secondMax = n;
            } else if (n == secondMax) {
                continue;
            } else if (n > thirdMax) {
                thirdMax = n;
            }
        }
        return thirdMax == Long.MIN_VALUE ? (int)firstMax : (int)thirdMax;
    }
}

作者：wang-xue-lei-2
链接：https://leetcode-cn.com/problems/third-maximum-number/solution/1msda-bai-93jian-yi-bu-yao-pai-xu-by-wan-rhqu/
```

415字符串相加

执行结果： 通过 显示详情 >

执行用时： 4 ms , 在所有 Java 提交中击败了 23.74% 的用户

内存消耗： 38.5 MB , 在所有 Java 提交中击败了 71.36% 的用户

炫耀一下:

```java
    public static String addStrings(String num1, String num2) {
        int carry = 0;
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < num1.length() && i < num2.length(); i++){
            int c = num1.charAt(num1.length() - 1 - i) - '0' +
num2.charAt(num2.length() - 1 - i) - '0' + carry;
            int div  = c % 10;
            carry    = c / 10;
            sb.insert(0, div);
        }

        if(num1.length() > num2.length()){
            for(int i = num1.length() - num2.length() - 1; i >= 0; i--){
                int c = num1.charAt(i) - '0' + carry;
                int div  = c % 10;
                carry    = c / 10;
                sb.insert(0, div);
            }
        }else if(num2.length() > num1.length()){
            for(int i = num2.length() - num1.length() - 1; i >= 0; i--){
                int c = num2.charAt(i) - '0' + carry;
                int div  = c % 10;
                carry    = c / 10;
                sb.insert(0, div);
            }
        }else{
            if(carry != 0)
                sb.insert(0, carry);
            carry = 0;
        }
        if(carry != 0)  sb.insert(0, carry);
        return sb.toString();
    }
```

416 Patition Equal Subset Sum 典型背包问题

「力扣」上的 0-1 背包问题：

「力扣」第 416 题：分割等和子集（中等）；
「力扣」第 474 题：一和零（中等）；
「力扣」第 494 题：目标和（中等）；
「力扣」第 879 题：盈利计划（困难）；
「力扣」上的 完全背包问题：

「力扣」第 322 题：零钱兑换（中等）；
「力扣」第 518 题：零钱兑换 II（中等）；
「力扣」第 1449 题：数位成本和为目标值的最大数字（困难）。

416. Partition Equal Subset Sum

难度 中等 622

Given a **non-empty** array `nums` containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Example 1:

```
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1,
5, 5] and [11].
```

Example 2:

```
Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned into
equal sum subsets.
```

Constraints:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 100`

```java
//回溯，  超时
class Solution {
    boolean flag = false;
    public boolean canPartition(int[] nums) {
        int sum = 0;
        for(int num : nums)
            sum += num;

        HashSet<Integer> visited = new HashSet<>();
        backtrack(sum, nums, 0, visited);

        return flag;
    }

    public void backtrack(int sum, int[] nums, int target, HashSet<Integer>
visited){
        if(sum - target == target){
            flag = true;
            return;
        }

        for(int i = 0; i < nums.length; i++){
            if(!visited.contains(i)){
                visited.add(i);
                target += nums[i];

                backtrack(sum, nums, target, visited);

                target -= nums[i];
                visited.remove((Integer)i);

            }
        }
    }
}
```

```cpp
/*
dp[i][j] = x 表示，对于前 i 个物品，当前背包的容量为 j 时，
  若 x 为 true，则说明可以恰好将背包装满，
  若 x 为 false，则说明不能恰好将背包装满
*/
bool canPartition(vector<int>& nums) {
    int sum = 0;
    for (int num : nums) sum += num;
    // 和为奇数时，不可能划分成两个和相等的集合
    if (sum % 2 != 0) return false;
```

```
    int n = nums.size();
    sum = sum / 2;
    vector<vector<bool>>
        dp(n + 1, vector<bool>(sum + 1, false));

    // base case
    for (int i = 0; i <= n; i++)
        dp[i][0] = true;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (j - nums[i - 1] < 0) {
               // 背包容量不足，不能装入第 i 个物品
                dp[i][j] = dp[i - 1][j];
            } else {
                //         不装                        装入背包
                dp[i][j] = dp[i - 1][j] || dp[i - 1][j-nums[i-1]];
            }
        }
    }
    return dp[n][sum];
}

作者：labuladong
链接：https://leetcode-cn.com/problems/partition-equal-subset-sum/solution/0-1-bei-bao-wen-ti-bian-ti-zhi-zi-ji-fen-ge-by-lab/
```

417 Pacific Atlantic Water Flow 典型回溯

417. Pacific Atlantic Water Flow

难度 中等 183

Given an `m x n` matrix of non-negative integers representing the height of each unit cell in a continent, the "Pacific ocean" touches the left and top edges of the matrix and the "Atlantic ocean" touches the right and bottom edges.

Water can only flow in four directions (up, down, left, or right) from a cell to another one with height equal or lower.

Find the list of grid coordinates where water can flow to both the Pacific and Atlantic ocean.

Note:

1. The order of returned grid coordinates does not matter.
2. Both *m* and *n* are less than 150.

Example:

```
Given the following 5x5 matrix:

  Pacific ~   ~   ~   ~   ~
       ~  1   2   2   3  (5) *
       ~  3   2   3  (4) (4) *
       ~  2   4  (5)  3   1  *
       ~ (6) (7)  1   4   5  *
       ~ (5)  1   1   2   4  *
          *   *   *   *   * Atlantic
```

```java
/*
    回溯解法
*/

class Solution {
    int row;
    int col;
    int[][] dirc = {{1, 0},{0, 1},{-1, 0},{0, -1}};
    public List<List<Integer>> pacificAtlantic(int[][] matrix) {
        row = matrix.length;
        col = row == 0 ? 0 : matrix[0].length;
        List<List<Integer>> res = new ArrayList<>();
        if (col == 0) return res;

        int[][] pa = new int[row][col];
        int[][] at = new int[row][col];
```

```
17
18          for(int i = 0; i < row; i++){
19              dfs(matrix, i, 0, pa);
20              dfs(matrix, i, col - 1, at);
21          }
22          for(int j = 0; j < col; j++){
23              dfs(matrix, 0, j, pa);
24              dfs(matrix, row - 1, j, at);
25          }
26
27          for (int i = 0; i < row; ++i)
28              for (int j = 0; j < col; ++j)
29                  if (pa[i][j] == 1 && at[i][j] == 1)
30                      res.add(Arrays.asList(i, j));
31
32          return res;
33      }
34
35      private void dfs(int[][] matrix, int i, int j, int[][] temp) {
36          temp[i][j] = 1;
37          for(int k = 0; k < 4; k++){
38              int newX = i + dirc[k][0];
39              int newY = j + dirc[k][1];
40              if(isInRange(newX, newY) && matrix[i][j] <= matrix[newX][newY] ){
41                //注意这句话去重复，否则StackOverFlow
42                  if(temp[newX][newY] != 1)
43                      dfs(matrix, newX, newY, temp);
44              }
45          }
46
47      }
48
49
50
51      public boolean isInRange(int i, int j){
52          return i >= 0 && j >= 0 && i < row && j < col;
53      }
54
55  }
```

418 Sentence Screen Fitting 取模解法真好!

418. Sentence Screen Fitting

难度 中等 43

Given a `rows x cols` screen and a sentence represented by a list of **non-empty** words, find **how many times** the given sentence can be fitted on the screen.

Note:

1. A word cannot be split into two lines.
2. The order of words in the sentence must remain unchanged.
3. Two consecutive words **in a line** must be separated by a single space.
4. Total words in the sentence won't exceed 100.
5. Length of each word is greater than 0 and won't exceed 10.
6. 1 ≤ rows, cols ≤ 20,000.

Example 1:

```
Input:
rows = 2, cols = 8, sentence = ["hello", "world"]

Output:
1

Explanation:
hello---
world---

The character '-' signifies an empty space on the screen.
```

Example 2:

```java
//可以通过全部案例， 但是超时间了
class Solution {
    public int wordsTyping(String[] sentence, int rows, int cols) {
        int r = 0, c = 0;
        int count = 0;

        while(r < rows){
            for(int i = 0; i < sentence.length; i++){
                if(sentence[i].length() > cols) return 0;
                if(c + sentence[i].length() > cols){
                    r++;
                    if(r == rows)   return count;
                    c = sentence[i].length() + 1;
```

```
                }else{
                    c += sentence[i].length() + 1;
                }
            }

            count++;
        }

        return count;
    }
}
```

```
//这个解法蛮新颖的
/*
    模 相当于 给sb 拓展了无限长度
    解法真好!
*/
public class Solution {
    public int wordsTyping(String[] sentence, int rows, int cols) {
        StringBuilder sb = new StringBuilder();
        for (String s : sentence) {
            sb.append(s).append(" ");
        }
        int index = 0, len = sb.length();   //(index % len)指针指向的是sb中的某个字符
        for (int i = 0; i < rows; i++) {
            index += cols;  //一行可以包含cols个字符
            if (sb.charAt(index % len) == ' ') {    //(index % len)指针指向的字符是空格
                //此时该行最末单元填充的恰好是某一单词的最后一个字符，下一个单词会从下一行行首开始填充，而无需再添加空格
                //从另一个角度来看，相对于该空格被忽略了，即无需填充该空格
                index++;
            } else {    //(index % len)指针指向的字符不是空格
                //此时该行无法完整填充当前单词，当前单词会从下一行行首开始填充
                //需要回退index指针的位置直到(index - 1 % len)指针指向的字符是空格
                while (index > 0 && sb.charAt((index - 1) % len) != ' ') {
                    index--;
                }
            }
        }
        return index / len;
    }
}

作者：617076674
```

```
链接：https://leetcode-cn.com/problems/sentence-screen-fitting/solution/ping-mu-ke-xian-shi-ju-zi-de-shu-liang-by-61707667/
```

419 Battleships in a Board 存在牛逼解法

419. Battleships in a Board

难度 中等 79

Given an 2D board, count how many battleships are in it. The battleships are represented with `'X'`s, empty slots are represented with `'.'`s. You may assume the following rules:

- You receive a valid board, made of only battleships or empty slots.
- Battleships can only be placed horizontally or vertically. In other words, they can only be made of the shape `1xN` (1 row, N columns) or `Nx1` (N rows, 1 column), where N can be of any size.
- At least one horizontal or vertical cell separates between two battleships - there are no adjacent battleships.

Example:

```
X..X
...X
...X
```

In the above board there are 2 battleships.
Invalid Example:

```
...X
XXXX
...X
```

This is an invalid board that you will not receive - as battleships will always have a cell separating between them.

Follow up:
Could you do it in **one-pass**, using only **O(1) extra memory** and

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 99.24% 的用户

内存消耗： 38.1 MB ，在所有 Java 提交中击败了 55.08% 的用户

炫耀一下：

```java
//二刷
    int row = 0;
    int col = 0;
    public int countBattleships(char[][] board) {
        row = board.length;
        col = board[0].length;

        int count = 0;
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(board[i][j] == 'X'){
                    if(i > 0 && board[i - 1][j] == 'X')
                        continue;
                    else if(j > 0 && board[i][j - 1] == 'X')
                        continue;

                    count++;
                }
            }
        }

        return count;
    }

    private boolean isInRange(int x,int y){
        return x >= 0 && y >= 0 && x < row && y < col;
    }
```

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 99.82% 的用户

内存消耗： 38.3 MB ，在所有 Java 提交中击败了 45.09% 的用户

炫耀一下：

```java
/*
    常规DFS 解法
*/
class Solution {
    int row;
    int col;
    int res = 0;
    public int countBattleships(char[][] board) {
        row = board.length;
        col = row == 0 ? 0 : board[0].length;

        if(col == 0)        return 0;

        for(int i = 0; i < row; i++)
            for(int j = 0; j < col; j++)
                if(board[i][j] == 'X'){
                    dfs(board, i, j);
                    res++;
                }


        return res;
    }

    public void dfs(char[][] board, int i, int j){
        if(isInRange(i, j) && board[i][j] == 'X'){

            board[i][j] = '@';
            dfs(board, i + 1, j);
            dfs(board, i - 1, j);
            dfs(board, i, j + 1);
            dfs(board, i, j - 1);


        }
    }

    public boolean isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
}
```

```java
//https://leetcode.com/problems/battleships-in-a-board/discuss/90902/Simple-Java-Solution
//牛逼解法

/*
```

```
  . X . . .
  . . . . .
  . . X . .
  . . X . X
  X . . . X

  1 + 1  + 1 + 1
*/
    public int countBattleships(char[][] board) {
        int m = board.length;
        if (m==0) return 0;
        int n = board[0].length;

        int count=0;

        for (int i=0; i<m; i++) {
            for (int j=0; j<n; j++) {
                if (board[i][j] == '.') continue;
                if (i > 0 && board[i-1][j] == 'X') continue;
                if (j > 0 && board[i][j-1] == 'X') continue;
                count++;
            }
        }

        return count;
    }
```

420 Strong Password Checker 未完成

420. Strong Password Checker

难度 困难 56

A password is considered strong if the below conditions are all met:

- It has at least `6` characters and at most `20` characters.
- It contains at least **one lowercase** letter, at least **one uppercase** letter, and at least **one digit**.
- It does not contain three repeating characters in a row (i.e., `"...aaa..."` is weak, but `"...aa...a..."` is strong, assuming other conditions are met).

Given a string `password`, return *the minimum number of steps required to make `password` strong. if `password` is already strong, return `0`*.

In one step, you can:

- Insert one character to `password`,
- Delete one character from `password`, or
- Replace one character of `password` with another character.

Example 1:

```
Input: password = "a"
Output: 5
```

Example 2:

```
Input: password = "aA1"
Output: 3
```

Example 3:

421Maximum XOR of Two Numbers in an Array

421. Maximum XOR of Two Numbers in an Array

难度 中等 195

Given an integer array `nums`, return *the maximum result of* `nums[i] XOR nums[j]`, where `0 ≤ i ≤ j < n`.

Follow up: Could you do this in `O(n)` runtime?

Example 1:

```
Input: nums = [3,10,5,25,2,8]
Output: 28
Explanation: The maximum result is 5 XOR 25 = 28.
```

Example 2:

```
Input: nums = [0]
Output: 0
```

Example 3:

```
Input: nums = [2,4]
Output: 6
```

Example 4:

```
Input: nums = [8,10,2]
Output: 10
```

Example 5:

执行结果： 通过 显示详情 >

执行用时： 510 ms ，在所有 Java 提交中击败了 5.35% 的用户

内存消耗： 39.2 MB ，在所有 Java 提交中击败了 89.56% 的用户

炫耀一下：

```java
//暴力解法
public int findMaximumXOR(int[] nums) {
  int maxRes = 0;
  for(int i = 0; i < nums.length - 1; i++)
    for(int j = i + 1; j < nums.length; j++)
      if((nums[i] ^ nums[j]) > maxRes)
        maxRes = nums[i] ^ nums[j];

  return maxRes;
}
```

```java
/*
    首先明白一个性质
    如果 a ^ b = c 成立，那么 a ^ c = b, b ^ c = a 成立

    可以把异或运算看作是 不进位的 加法运算
*/
import java.util.HashSet;
import java.util.Set;

public class Solution {

    // 先确定高位，再确定低位（有点贪心算法的意思），才能保证这道题的最大性质
    // 一位接着一位去确定这个数位的大小
    // 利用性质： a ^ b = c ，则 a ^ c = b，且 b ^ c = a

    public int findMaximumXOR(int[] nums) {
        int res = 0;
        int mask = 0;
        for (int i = 30; i >= 0; i--) {
            // 注意点1：注意保留前缀的方法，mask 是这样得来的
            // 用异或也是可以的 mask = mask ^ (1 << i);
            mask = mask | (1 << i);

            // System.out.println(Integer.toBinaryString(mask));
```

```
            Set<Integer> set = new HashSet<>();
            for (int num : nums) {
                // 注意点2：这里使用 & ，保留前缀的意思（从高位到低位）
                set.add(num & mask);
            }

            // 这里先假定第 n 位为 1 ，前 n-1 位 res 为之前迭代求得
            int temp = res | (1 << i);
            for (Integer prefix : set) {
                if (set.contains(prefix ^ temp)) {
                    res = temp;
                    break;
                }
            }
        }
        return res;
    }
}

作者：liweiwei1419
链接：https://leetcode-cn.com/problems/maximum-xor-of-two-numbers-in-an-array/solution/li-yong-yi-huo-yun-suan-de-xing-zhi-tan-xin-suan-f/t
```

422 Valid Word Square

执行结果： 通过 显示详情 >

执行用时： 13 ms ，在所有 Java 提交中击败了 49.25% 的用户

内存消耗： 38.1 MB ，在所有 Java 提交中击败了 76.12% 的用户

炫耀一下:

```
    public boolean validWordSquare(List<String> words) {
        int len = words.size();

        StringBuilder[] sb = new StringBuilder[len];
        for(int i = 0; i < len; i++){
            sb[i] = new StringBuilder();
        }
```

```
        for(int i = 0; i < words.get(0).length(); i++){
            for(int j = 0; j < len; j++){
                if(i < words.get(j).length() && i < len)
                    sb[i].append(words.get(j).charAt(i));
            }
        }

        for(int i = 0; i < len; i++){
            if(!words.get(i).equals(sb[i].toString()))
                return false;
        }

        return true;
    }

```

执行结果: 通过 显示详情 >

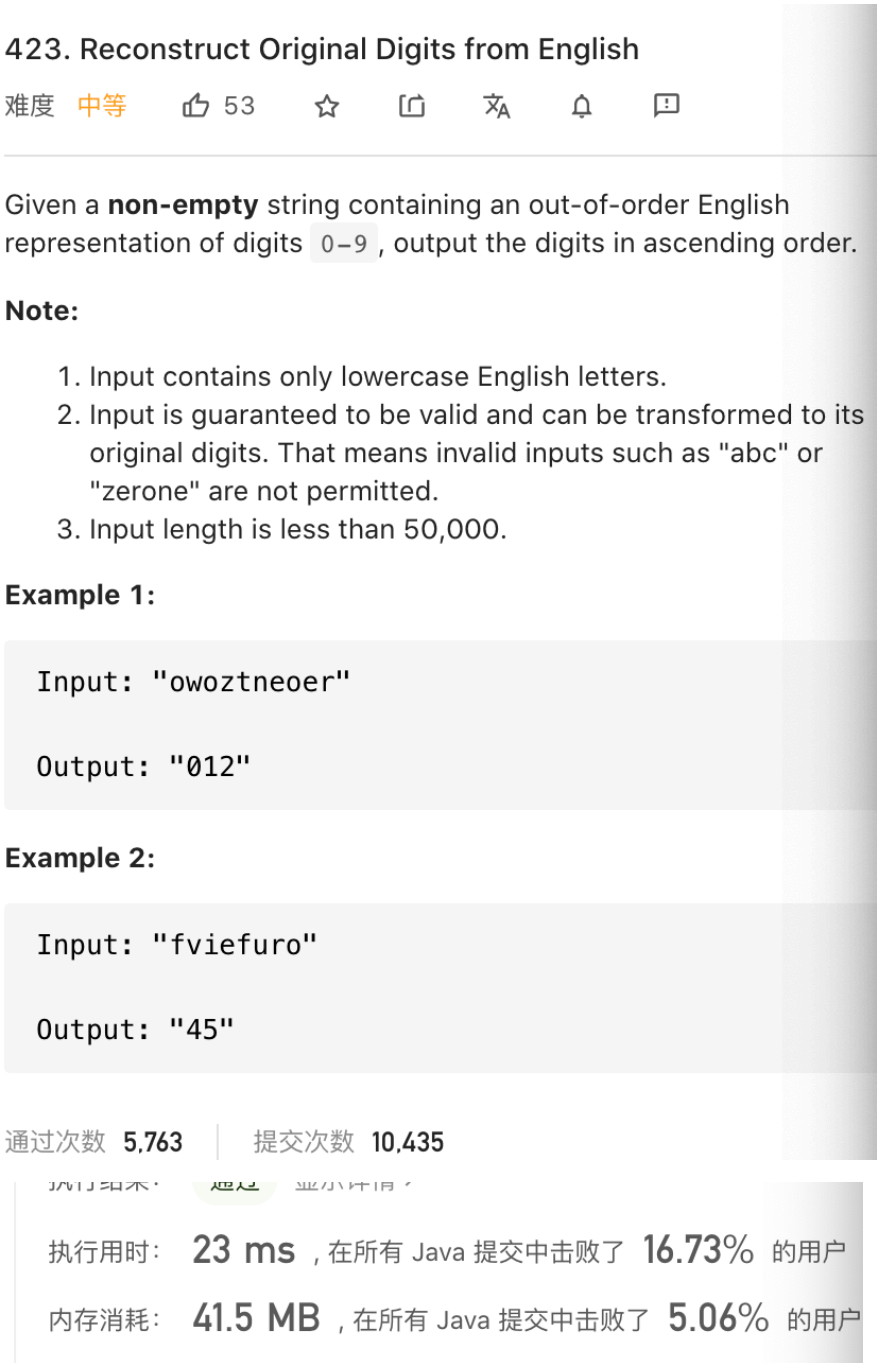
执行用时: 16 ms , 在所有 Java 提交中击败了 14.49% 的用户

内存消耗: 38.3 MB , 在所有 Java 提交中击败了 63.23% 的用户

炫耀一下:

```
class Solution {
    public boolean validWordSquare(List<String> words) {
        int len = words.size();

        for(int i = 0; i < len; i++)
            for(int j = 0; j < words.get(i).length(); j++)
                if(j >= len || words.get(j).length() <= i)
                    return false;
                else if(words.get(i).charAt(j) !=  words.get(j).charAt(i))
                    return false;

        return true;
    }
}
```

423 Reconstruct Original Digits from English



423. Reconstruct Original Digits from English

难度 中等 53

Given a **non-empty** string containing an out-of-order English representation of digits `0-9`, output the digits in ascending order.

Note:

1. Input contains only lowercase English letters.
2. Input is guaranteed to be valid and can be transformed to its original digits. That means invalid inputs such as "abc" or "zerone" are not permitted.
3. Input length is less than 50,000.

Example 1:

```
Input: "owoztneoer"

Output: "012"
```

Example 2:

```
Input: "fviefuro"

Output: "45"
```

通过次数 5,763 | 提交次数 10,435

执行用时： 23 ms ，在所有 Java 提交中击败了 16.73% 的用户

内存消耗： 41.5 MB ，在所有 Java 提交中击败了 5.06% 的用户

```java
//二刷
    public String originalDigits(String s) {
        Map<Character, Integer> map = new HashMap<>();
        for(char ch : s.toCharArray())
            map.put(ch, map.getOrDefault(ch, 0) + 1);

        Map<Integer, Integer> res = new HashMap<>();

        if(map.getOrDefault('z', 0) != 0){
            int num = map.get('z');
            res.put(0, res.getOrDefault(0, 0) + num);
            reduceFreq("zero", num, map);
```

```
        }

        if(map.getOrDefault('x', 0) != 0){
            int num = map.get('x');
            res.put(6, res.getOrDefault(6, 0) + num);
            reduceFreq("six", num, map);
        }

        if(map.getOrDefault('s', 0) != 0){
            int num = map.get('s');
            res.put(7, res.getOrDefault(7, 0) + num);

            reduceFreq("seven", num, map);
        }

        if(map.getOrDefault('g', 0) != 0){
            int num = map.get('g');
            res.put(8, res.getOrDefault(8, 0) + num);

            reduceFreq("eight", num, map);
        }

        if(map.getOrDefault('h', 0) != 0){
            int num = map.get('h');
            res.put(3, res.getOrDefault(3, 0) + num);

            reduceFreq("three", num, map);
        }

        if(map.getOrDefault('w', 0) != 0){
            int num = map.get('w');
            res.put(2, res.getOrDefault(2, 0) + num);

            reduceFreq("two", num, map);
        }

        if(map.getOrDefault('r', 0) != 0){
            int num = map.get('r');
            res.put(4, res.getOrDefault(4, 0) + num);

            reduceFreq("four", num, map);
        }

        if(map.getOrDefault('o', 0) != 0){
            int num = map.get('o');
            res.put(1, res.getOrDefault(1, 0) + num);

            reduceFreq("one", num, map);
        }
```

```
62
63          if(map.getOrDefault('v', 0) != 0){
64              int num = map.get('v');
65              res.put(5, res.getOrDefault(5, 0) + num);
66
67              reduceFreq("five", num, map);
68          }
69
70          if(map.getOrDefault('i', 0) != 0){
71              int num = map.get('i');
72              res.put(9, res.getOrDefault(9, 0) + num);
73
74              reduceFreq("nine", num, map);
75          }
76
77
78
79
80          StringBuilder ans = new StringBuilder();
81          for(int i = 0; i <= 9; i++){
82              for(int j = 0; j < res.getOrDefault(i, 0); j++)
83                  ans.append(i);
84          }
85
86          return ans.toString();
87      }
88
89      private void reduceFreq(String str, int freq, Map<Character, Integer> map){
90          for(char ch : str.toCharArray()){
91              map.put(ch, map.getOrDefault(ch, 0) - freq);
92          }
93      }
94
```


```
1   import java.lang.reflect.Constructor;
2   import java.lang.reflect.InvocationTargetException;
```

```java
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.List;
import java.util.PriorityQueue;

public class Solution {

        public String originalDigits(String s) {
        /*
            0 zero      z
            1 one
            2 two       w
            3 three
            4 four
            5 five
            6 six       x
            7 seven
            8 eight     g
            9 nine
        */
            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);

            StringBuilder sb = new StringBuilder();
            PriorityQueue<Integer> pq = new PriorityQueue<>();
        /*
            deal with 0 2 6 8
        */
            while(map.getOrDefault('z', 0) > 0 || map.getOrDefault('w', 0) > 0
                    || map.getOrDefault('x', 0) > 0 || map.getOrDefault('g',0) >
0){
                while(map.getOrDefault('z',0) > 0) {
                    pq.add(0);
                    minusMap(map, "zero");
                }

                while(map.getOrDefault('w',0) > 0){
                    pq.add(2);
                    minusMap(map, "two");
                }

                while(map.getOrDefault('x',0) > 0){
                    pq.add(6);
                    minusMap(map, "six");
                }

                while(map.getOrDefault('g',0) > 0){
                    pq.add(8);
```

```
                    minusMap(map, "eight");
                }

            }

            //3
            while(map.getOrDefault('t',0) > 0){
                while(map.getOrDefault('t',0) > 0){
                    pq.add(3);
                    minusMap(map, "three");
                }


            }

            while(map.getOrDefault('r',0) > 0 || map.getOrDefault('s',0) > 0){
                while(map.getOrDefault('r',0) > 0){
                    pq.add(4);
                    minusMap(map, "four");
                }

                while(map.getOrDefault('s',0) > 0){
                    pq.add(7);
                    minusMap(map, "seven");
                }
            }

            //////


            while(map.getOrDefault('v',0) > 0|| map.getOrDefault('o',0) > 0){

                while(map.getOrDefault('o',0) > 0){
                    pq.add(1);
                    minusMap(map, "one");

                }

                while(map.getOrDefault('v',0) > 0){
                    pq.add(5);
                    minusMap(map, "five");
                }
            }


            while(map.getOrDefault('n',0) > 0){
                pq.add(9);
                minusMap(map, "nine");
            }
```

```
100
101
102
103
104
105             while(!pq.isEmpty())
106                 sb.append(pq.poll());
107
108             return sb.toString();
109         }
110
111         public void minusMap(HashMap<Character, Integer> map, String num){
112             for(int i = 0; i < num.length(); i++)
113                 map.put(num.charAt(i), map.get(num.charAt(i)) - 1);
114         }
115
116
117 }
118
119
```

424 Longest Repeating Character replacement

424. Longest Repeating Character Replacement

难度 中等 205

Given a string `s` that consists of only uppercase English letters, you can perform at most `k` operations on that string.

In one operation, you can choose **any** character of the string and change it to any other uppercase English character.

Find the length of the longest sub-string containing all repeating letters you can get after performing the above operations.

Note:
Both the string's length and *k* will not exceed 10^4.

Example 1:

```
Input:
s = "ABAB", k = 2

Output:
4

Explanation:
Replace the two 'A's with two 'B's or vice versa.
```

Example 2:

```
Input:
s = "AABABBA", k = 1

Output:
```

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 13.07% 的用户

内存消耗: 38.2 MB , 在所有 Java 提交中击败了 99.03% 的用户

炫耀一下:

```java
int[] map = new int[26];
public int characterReplacement(String s, int k) {
    /*
        in one window,
        there will be only  k distinct charactes, that will be great
    */
```

```java
        int res = 0;
        int left = 0, right = 0;
        int len = s.length();
        while(right < len){
            while(right < len && canBeReplaceInKOps(right - left, k, map)){
                res = Math.max(right - left, res);
                char ch = s.charAt(right);
                map[ch - 'A']++;
                right++;
            }

            if(right == len) {
                if(canBeReplaceInKOps(right - left, k, map))
                    res = Math.max(right - left, res);
                break;
            }
            while(left < len && !canBeReplaceInKOps(right - left, k, map)){
                char ch = s.charAt(left);
                map[ch - 'A']--;
                left++;
            }
        }

        return res;
    }

    private boolean canBeReplaceInKOps(int strLen, int k, int[] map){
        int counter = 0;
        for(int i = 0; i < 26; i++){
            if(map[i] != 0)
                break;
            counter++;
        }

        if(counter == 26)
            return true;

        int maxFreq = 0;
        int totalFreq = 0;
        for(int i = 0; i < 26; i++){
            maxFreq = Math.max(maxFreq, map[i]);
            totalFreq += map[i];
        }

        return totalFreq - maxFreq <= k;
    }

```

```java
/*
  典型回溯会超时
*/
class Solution {
    int maxLen = 0;
    public int characterReplacement(String s, int k) {
        char[] chars = s.toCharArray();
        maxLen = countRepeat(chars);

        backtrack(chars, k);
        return maxLen;
    }

    private void backtrack(char[] chars, int remainingOps){
        if(remainingOps == 0){
            return;
        }

        for(int i = 0; i < chars.length; i++){
            for(int j = 0; j < 26; j++){
                char ch = chars[i];
                if(ch == (char)('A' + j))   continue;

                chars[i] = (char)('A' + j);
                maxLen = Math.max(maxLen, countRepeat(chars));

                backtrack(chars, remainingOps - 1);

                chars[i] = ch;
            }
        }
    }

    private int countRepeat(char[] chars){
        int res = 0;
        int left = 0, right = 0;
        while(right < chars.length){
            while(right < chars.length && chars[right] == chars[left])
                right++;

            res = Math.max(right - left, res);
            left = right;
        }

        return res;
    }

```

```

}
```

```
/*
    滑动窗口 变式
*/
class Solution {
    private int[] map = new int[26];

    public int characterReplacement(String s, int k) {
        if (s == null) {
            return 0;
        }
        char[] chars = s.toCharArray();
        int left = 0;
        int right = 0;
        int historyCharMax = 0;
        for (right = 0; right < chars.length; right++) {
            int index = chars[right] - 'A';
            map[index]++;
            historyCharMax = Math.max(historyCharMax, map[index]);
            if (right - left + 1 > historyCharMax + k) {
                map[chars[left] - 'A']--;
                left++;
            }
        }
        return chars.length - left;
    }
}

作者：migoo
链接：https://leetcode-cn.com/problems/longest-repeating-character-replacement/solution/tong-guo-ci-ti-liao-jie-yi-xia-shi-yao-shi-hua-don/
```

425 Word Squares

425. Word Squares

难度 困难 44

Given a set of words **(without duplicates)**, find all word squares you can build from them.

A sequence of words forms a valid word square if the k^{th} row and column read the exact same string, where $0 \le k < \max(\text{numRows}, \text{numColumns})$.

For example, the word sequence `["ball","area","lead","lady"]` forms a word square because each word reads the same both horizontally and vertically.

```
b a l l
a r e a
l e a d
l a d y
```

Note:

1. There are at least 1 and at most 1000 words.
2. All words will have the exact same length.
3. Word length is at least 1 and at most 5.
4. Each word contains only lowercase English alphabet `a-z`.

Example 1:

```
Input:
["area","lead","wall","lady","ball"]

Output:
[
  [ "wall",
```

```cpp
class Solution {
public:
    map<string, vector<string> > lookup;
    vector<vector<string>> res;
    vector<vector<string>> wordSquares(vector<string>& words) {
        for(string& s : words){
            for(int i = 0; i <= s.length(); i++){
                lookup[s.substr(0, i)].push_back(s);
            }
        }

        vector<string> path;
```

```
        backtrack(words, path);
        return res;
    }

    void backtrack(vector<string>& words, vector<string> path){
        if(path.size() == words[0].size()){
            res.push_back(path);
            return;
        }

        string prefix;
        for(int i = 0; i < path.size(); i++)
            prefix += path[i][path.size()];

        if(lookup.find(prefix) == lookup.end())
            return;

        vector<string> choice = lookup[prefix];
        for(string s : choice){
            path.push_back(s);

            backtrack(words, path);

            path.erase(path.end() - 1);
        }

    }
};
```

```
//回溯，  通过 13 / 16 个案例
//耗时 3000ms 左右
    String[] words;
    List<List<String>> res = new ArrayList<>();
    public List<List<String>> wordSquares(String[] words) {
        this.words = words;

        if(words.length == 0)   return res;

        for(int i = 0; i < words.length; i++){
            List<String> path = new ArrayList<>();
            path.add(words[i]);

            backtrack(words[i], path);

        }

        return res;
```

```
    }

    public boolean isValid(List<String> words) {
        char [][]arr = new char[words.size()][words.size()];
        for(int i=0; i<words.size(); i++){
            char []temp = words.get(i).toCharArray();
            //如果列数大于行数直接返回false
            if(temp.length > words.size())
                return false;
            for(int j=0; j<temp.length; j++){
                arr[i][j] = temp[j];
            }
        }
        for(int i=0; i<words.size(); i++){
            for(int j=0; j<i; j++){
                if(arr[i][j] != arr[j][i])
                    return false;
            }
        }
        return true;
    }


    private void backtrack(String firstWord, List<String> path){
        if(isValid(path)){
            res.add(new ArrayList<>(path));
            return;
        }

        if(path.size() >= firstWord.length())  return;

        for(int i = 0; i < words.length; i++){
            if(firstWord.charAt(path.size()) == words[i].charAt(0)){

                path.add(words[i]);

                backtrack(firstWord, path);

                path.remove(path.size() - 1);
            }

        }
    }

```

```
class Solution {
    public List<List<String>> wordSquares(String[] words) {
```

```
        int n = words[0].length();
        Map<String, List<String>> map = new HashMap<>();
        for (String word : words) {
            for (int i = 0; i < n; i++) {
                String prefix = word.substring(0, i);
                map.putIfAbsent(prefix, new ArrayList<>());
                map.get(prefix).add(word);
            }
        }
        List<String> list = new ArrayList<>();
        List<List<String>> lists = new ArrayList<>();
        dfs(map, n, list, lists);
        return lists;
    }

    private void dfs(Map<String, List<String>> map, int n, List<String> list, List<List<String>> lists) {
        if (list.size() == n) {
            lists.add(new ArrayList<>(list));
            return;
        }
        int m = list.size();
        char[] chars = new char[m];
        for (int i = 0; i < m; i++) {
            chars[i] = list.get(i).charAt(m);
        }
        String prefix = new String(chars);
        if (!map.containsKey(prefix)) return;
        List<String> words = map.get(prefix);
        for (String word : words) {
            list.add(word);
            dfs(map, n, list, lists);
            list.remove(list.size() - 1);
        }
    }
}
```

426 Convert Binary Search Tree to Sorted Doubly Linked List

426. Convert Binary Search Tree to Sorted Doubly Linked List

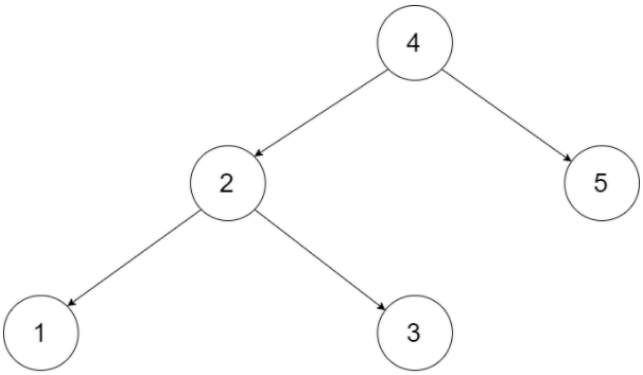
难度 中等 77

Convert a **Binary Search Tree** to a sorted **Circular Doubly-Linked List** in place.

You can think of the left and right pointers as synonymous to the predecessor and successor pointers in a doubly-linked list. For a circular doubly linked list, the predecessor of the first element is the last element, and the successor of the last element is the first eleme

We want to do the transformation **in place**. After the transformation, the left pointer of the tree node should point to its predecessor, and the right pointer should point to its successor. You should return the pointer to the smallest element of the linked list.

Example 1:



Input: root = [4,2,5,1,3]

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: 37.8 MB , 在所有 Java 提交中击败了 76.43% 的用户

炫耀一下:

```java
/*
    本质上是中序遍历，  因为最后要的结果是中序
*/
class Solution {
    private Node head = null;
    private Node tail = null;
```

```
    public Node treeToDoublyList(Node root) {
        if(root == null)    return root;

        inorder(root);

      //处理头尾
        head.left = tail;
        tail.right = head;

        return head;
    }

    private void inorder(Node root){
        if(root == null)    return;

        inorder(root.left);

        if(head == null){
            head = root;
        }else{
            root.left = tail;
            tail.right = root;
        }

        tail = root;
        inorder(root.right);
    }
}
```

427 Construct Quad Tree

427. Construct Quad Tree

难度 中等 47

Given a `n * n` matrix `grid` of `0's` and `1's` only. We want to represent the `grid` with a Quad-Tree.

Return *the root of the Quad-Tree* representing the `grid`.

Notice that you can assign the value of a node to **True** or **False** when `isLeaf` is **False**, and both are **accepted** in the answer.

A Quad-Tree is a tree data structure in which each internal node has exactly four children. Besides, each node has two attributes:

- `val`: True if the node represents a grid of 1's or False if the node represents a grid of 0's.
- `isLeaf`: True if the node is leaf node on the tree or False if the node has the four children.

```
class Node {
    public boolean val;
    public boolean isLeaf;
    public Node topLeft;
    public Node topRight;
    public Node bottomLeft;
    public Node bottomRight;
}
```

We can construct a Quad-Tree from a two-dimensional area using the following steps:

1. If the current grid has the same value (i.e all `1's` or all `0's`) set `isLeaf` True and set `val` to the value of the grid and set the four children to Null and stop.

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: 38.9 MB, 在所有 Java 提交中击败了 55.66% 的用户

炫耀一下:

```
1
2 /*
```

```
    题不难，  就是有点繁琐
*/

class Solution {
    public Node construct(int[][] grid) {
        //left, right, up, down
        return helper(grid, 0, grid.length - 1, 0, grid[0].length - 1);
    }

    private Node helper(int[][] grid, int left, int right, int up, int down){
        Node root  = new Node();
        if(isSame(grid, left, right, up, down)){
            root.val = grid[up][left] == 1 ? true : false;
            root.isLeaf = true;
            return root;
        }

        root.isLeaf = false;
        root.topLeft     = helper(grid, left,(right + left) / 2,  up, (up + down) / 2);
        root.topRight    = helper(grid, (right + left) / 2 + 1 , right, up, (up + down) / 2);
        root.bottomLeft  = helper(grid, left, (right + left) / 2 , (up + down) / 2 + 1, down);
        root.bottomRight = helper(grid, (right + left) / 2 + 1, right, (up + down) / 2 + 1, down);

        return root;
    }

    private boolean isSame(int[][] grid, int left, int right, int up, int down){
        int pre = -1;
        for(int i = up; i <= down; i++)
            for(int j = left; j <= right; j++)
                if(pre != -1){
                    if(pre != grid[i][j])
                        return false;
                }else
                    pre = grid[i][j];


        return true;
    }
}


作者：venturekwok
链接：https://leetcode-cn.com/problems/construct-quad-tree/solution/java-chao-yue-100-chang-gui-si-lu-by-ven-j8na/
```

```
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。
```

428 Serialize and Deserialize N-ary Tree

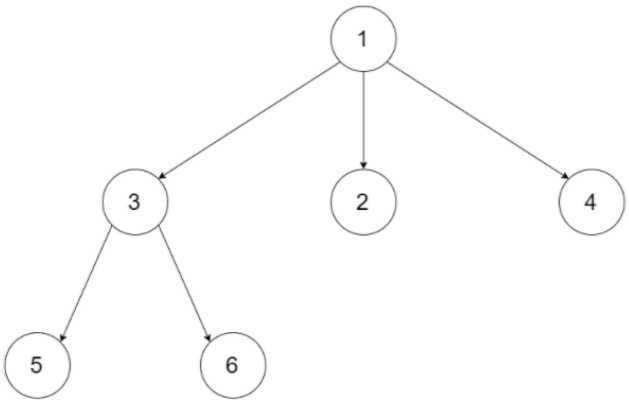
428. Serialize and Deserialize N-ary Tree

难度 困难 43

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize an N-ary tree. An N-ary tree is a rooted tree in which each node has no more than N children. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that an N-ary tree can be serialized to a string and this string can be deserialized to the original tree structure.

For example, you may serialize the following `3-ary` tree



as `[1 [3[5 6] 2 4]]`. Note that this is just an example, you do nc necessarily need to follow this format.

Or you can follow LeetCode's level order traversal serialization forma where each group of children is separated by the null value.

429 N-ary Tree Level Order Traversal

执行结果： 通过 显示详情 >

执行用时： 4 ms , 在所有 Java 提交中击败了 48.69% 的用户

内存消耗： 39.4 MB , 在所有 Java 提交中击败了 75.26% 的用户

炫耀一下:

```java
public List<List<Integer>> levelOrder(Node root) {
    List<List<Integer>> res = new ArrayList<>();
    if(root == null)        return res;
    Deque<Node> deque = new ArrayDeque<>();
    deque.add(root);
    while(!deque.isEmpty()){
        int size = deque.size();
        List<Integer> path = new ArrayList<>();
        for(int i = 0; i < size; i++){
            Node cur = deque.pollFirst();
            path.add(cur.val);
            for(Node n : cur.children)
                deque.add(n);
        }

        res.add(path);
    }

    return res;
}
```

430 Flatten a Multilevel Doubly Linked List

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： 36.3 MB , 在所有 Java 提交中击败了 91.98% 的用户

炫耀一下:

```java
//二刷， 不需要遍历到末尾
class Solution {
    Node tail = null;
    public Node flatten(Node head) {
```

```
        if(head == null)
            return null;

        Node cur = head;
        while(cur != null){
            if(cur.next == null)
                tail = cur;

            if(cur.child != null){
                Node temp = cur.next;
                Node child = flatten(cur.child);
                cur.next = child;
                child.prev = cur;
                cur.child = null;

                if(tail != null && temp != null){
                    tail.next = temp;
                    temp.prev = tail;
                    tail = null;
                }

                cur = temp;
            }else{
                cur = cur.next;
            }
        }

        return head;
    }
}
```


```
class Solution {
    public Node flatten(Node head) {
        if(head == null)   return head;

        Node cur = head;
        while(cur != null){
            if(cur.child != null){

                Node temp = cur.next;
```

```
                Node child    = flatten(cur.child);
                Node childEnd = getEnd(child);


                childEnd.next = cur.next;
                child.prev = cur;
                if(cur.next != null)
                    cur.next.prev = childEnd;
                cur.next = child;
                //[1,null,2,null,3,null
                //1 -> null

                //2

                //3

                cur.child = null;
                cur = temp;
            }else{
                cur = cur.next;
            }
        }


        //System.out.println(head.val);
        return head;
    }


    public Node getEnd(Node child){
        while(child.next != null)
            child = child.next;

        return child;
    }
}

```

431 Encode N-ary Tree to Binary Tree

```
/*
// Definition for a Node.
class Node {
```

```
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/

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

class Codec {
public:
    // Encodes an n-ary tree to a binary tree.
    TreeNode* encode(Node* root) {
        if(root == nullptr)
            return nullptr;

        TreeNode* newRoot = new TreeNode(root->val);
        vector<Node*> children = root->children;
        if(children.size() == 0)
            return newRoot;
        else{
            TreeNode* leftChild = encode(children[0]);
            newRoot->left = leftChild;

            if(children.size() == 1)
                return newRoot;

            TreeNode* cur = leftChild;
            for(int i = 1; i < children.size(); i++){
                cur->right = encode(children[i]);
                cur = cur->right;
```

```
            }
        }

        return newRoot;
    }

    // Decodes your binary tree to an n-ary tree.
    Node* decode(TreeNode* root) {
        if(root == nullptr)
            return nullptr;

        Node* newRoot = new Node(root->val);
        TreeNode* cur = root->left;
        if(cur == nullptr)
            return newRoot;
        else{
            while(cur != nullptr){
                newRoot->children.push_back(decode(cur));
                cur = cur->right;
            }
        }

        return newRoot;
    }
};

// Your Codec object will be instantiated and called as such:
// Codec codec;
// codec.decode(codec.encode(root));
```

432 All O(1) Data Structure

432. All O`one Data Structure

难度 困难 77

Implement a data structure supporting the following operations:

1. Inc(Key) - Inserts a new key with value 1. Or increments an existing key by 1. Key is guaranteed to be a **non-empty** string.
2. Dec(Key) - If Key's value is 1, remove it from the data structure. Otherwise decrements an existing key by 1. If the key does not exist, this function does nothing. Key is guaranteed to be a **non-empty** string.
3. GetMaxKey() - Returns one of the keys with maximal value. If no element exists, return an empty string `""`.
4. GetMinKey() - Returns one of the keys with minimal value. If no element exists, return an empty string `""`.

Challenge: Perform all these in O(1) time complexity.

通过次数 4,818 | 提交次数 13,502

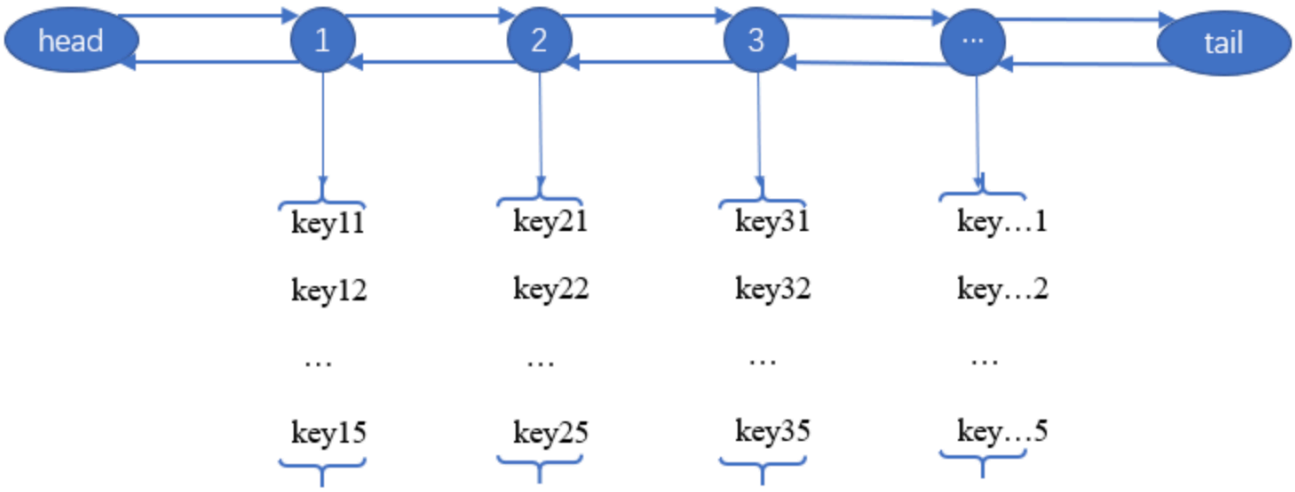
```java
//二刷
class AllOne {
    private DoubleLinkedList dll;
    private Map<String, Node> map;
    /** Initialize your data structure here. */
    public AllOne() {
        map = new HashMap<>();
        dll = new DoubleLinkedList();
    }
    /*
     * 1. node
     * 2. dll
     * 3. map
     */
    /** Inserts a new key <Key> with value 1. Or increments an existing key by 1.
*/
    public void inc(String key) {
        if(!map.containsKey(key)){
            if(dll.head.next.val == 1){
                dll.head.next.set.add(key);
                map.put(key, dll.head.next);
            }else{
                Node newNode = new Node(1);
                newNode.set.add(key);

                Node next    = dll.head.next;
```

```
                dll.addFront(next, newNode);

                map.put(key, newNode);
            }
        }else{
            Node node = map.get(key);
            Node next = node.next;
            if(node.next.val != node.val + 1){
                next = new Node(node.val + 1);
                dll.addBack(node, next);
            }

            node.set.remove(key);
            next.set.add(key);

            map.put(key, next);
            if(node.set.size() == 0)
                dll.delete(node);
        }
    }

    /** Decrements an existing key by 1. If Key's value is 1, remove it from the data structure. */
    public void dec(String key) {
        if(!map.containsKey(key))
            return;

        Node node = map.get(key);
        if(node.val == 1){
            node.set.remove(key);

            if(node.set.size() == 0)
                dll.delete(node);
            map.remove(key);
            return;
        }

        Node prev = node.prev;
        if(prev.val + 1 != node.val){
            prev = new Node(node.val - 1);
            dll.addFront(node, prev);
        }

        node.set.remove(key);
        prev.set.add(key);

        if(node.set.size() == 0)
            dll.delete(node);

```

```
        map.put(key, prev);
    }

    /** Returns one of the keys with maximal value. */
    public String getMaxKey() {
        return dll.head.next == dll.tail ? "" :
dll.tail.prev.set.iterator().next();
    }

    /** Returns one of the keys with Minimal value. */
    public String getMinKey() {
        return dll.head.next == dll.tail ? "" :
dll.head.next.set.iterator().next();
    }
}

class DoubleLinkedList{
    public Node head;
    public Node tail;

    public DoubleLinkedList() {
        this.head = new Node(-5);
        this.tail = new Node(-5);

        head.next = tail;
        tail.prev = head;
    }

    public void addFront(Node cur, Node node){
        node.next = cur;
        node.prev = cur.prev;

        cur.prev.next = node;
        cur.prev      = node;
    }

    public void addBack(Node cur, Node node){
        node.next = cur.next;
        node.prev = cur;

        cur.next.prev = node;
        cur.next      = node;
    }

    public void delete(Node node){
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }
}
```

```
121
122 class Node{
123     /* freq */
124     public int val;
125     public Set<String> set;
126     public Node prev;
127     public Node next;
128
129     public Node(int val) {
130         this.val = val;
131         this.set = new HashSet<>();
132     }
133 }
```

题目中说以 O(1)的时间，我们可以采用双向链表，如下图所示



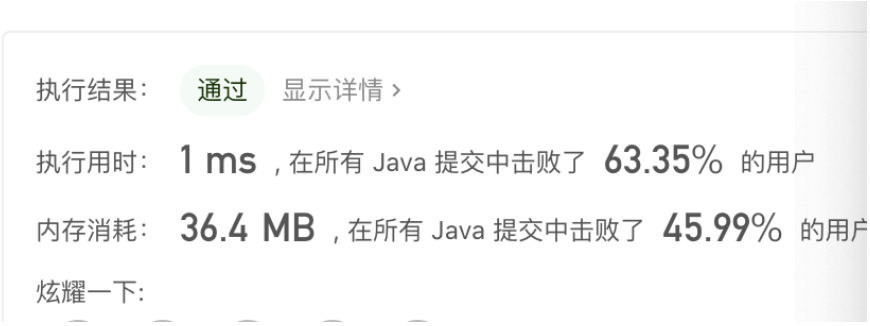

```
1  /*
2      思路类似于我们的 LRU
3
4      想到 O（1） 去拿到我们的元素， 那么就采用map
5      同时删除，拿到最大，最小，那么就维护我们的这个链表有序
6
7      保证head < node < tail
8
9      同时注意， node 的结构设计是， val 代表出现次数
10     Set<String> 里面代表的是 出现同一次数的String集合
11
12     结构如上图所示
13
14
15     如果是inc 操作，那就先看看存不存在
16       不存在， 那就从头加起
```

```
17
18          存在， 那就从中间开始更新
19
20      如果是 des 操作 先看看map 中有没有这个key
21      没有就 return
22
23      有的话， 就对应删除，注意边界情况的处理
24  */
25  public class AllOne {
26      HashMap<String, Node> map;
27      DoubleLinkedList dll;
28
29      /** Initialize your data structure here. */
30      public AllOne() {
31          map  = new HashMap<>();
32          dll = new DoubleLinkedList();
33      }
34
35      /** Inserts a new key <Key> with value 1. Or increments an existing key by 1.
    */
36      public void inc(String key) {
37          if(map.containsKey(key)){
38              Node p = map.get(key);
39              Node pn = p.next;
40
41              if(pn.val != p.val + 1){
42                  pn = new Node(p.val + 1);
43                  dll.insertBack(p, pn);
44              }
45
46              pn.content.add(key);
47              p.content.remove(key);
48
49              if(p.content.isEmpty()) dll.delete(p);
50              map.put(key, pn);
51          }else{
52              Node p = dll.head.next;
53              if(p.val != 1){
54                  p = new Node(1);
55                  dll.insertBack(dll.head, p);;
56              }
57
58              p.content.add(key);
59              map.put(key, p);
60          }
61      }
62
63      /** Decrements an existing key by 1. If Key's value is 1, remove it from the
    data structure. */
```

```
    public void dec(String key) {
        if(!map.containsKey(key))   return;

        Node p = map.get(key);
        if(p.val == 1){
            p.content.remove(key);
            if(p.content.isEmpty()){
                dll.delete(p);
                map.remove(key);
            }
        }else{
            Node pr = p.prev;
            if(pr.val != p.val - 1){
                pr = new Node(p.val - 1);
                dll.insertFront(p, pr);
            }

            pr.content.add(key);
            p.content.remove(key);
            if(p.content.isEmpty()) dll.delete(p);
            map.put(key, pr);
        }

    }

    /** Returns one of the keys with maximal value. */
    public String getMaxKey() {
        return dll.tail.prev == dll.head ? "" :
dll.tail.prev.content.iterator().next();
    }

    /** Returns one of the keys with Minimal value. */
    public String getMinKey() {
        return dll.head.next == dll.tail ? "" :
dll.head.next.content.iterator().next();
    }
}

class DoubleLinkedList{
    public Node head;
    public Node tail;

    public DoubleLinkedList(){
        head = new Node(0);
        tail = new Node(0);

        head.next = tail;
        tail.prev = head;
    }
```

```
111
112     //插到 cur 前
113     void insertFront(Node cur, Node p){
114         p.prev = cur.prev;
115         p.next = cur;
116
117         cur.prev.next = p;
118         cur.prev      = p;
119     }
120
121     void insertBack(Node cur, Node p){
122         p.next = cur.next;
123         p.prev = cur;
124
125         cur.next.prev = p;
126         cur.next      = p;
127     }
128
129     void delete(Node cur){
130         cur.prev.next = cur.next;
131         cur.next.prev = cur.prev;
132     }
133
134
135 }
136
137 class Node {
138     public int val;     //represent the value
139     public Set<String> content;
140     public Node prev;
141     public Node next;
142
143     public Node(int val){
144         this.val = val;
145         content = new HashSet<>();
146     }
147 }
148
149
```

433 常规 BFS 题目




```java
//二刷
class Solution {
    public int minMutation(String start, String end, String[] bank) {
        Set<String> set = new HashSet<>();
        for(String str : bank)
            set.add(str);

        Deque<String> queue = new ArrayDeque<>();
        Set<String> visited = new HashSet<>();
        queue.addLast(start);
        visited.add(start);

        int res = 0;
        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i = 0; i < size; i++){
                String cur = queue.removeFirst();
                if(cur.equals(end)){
                    return res;
                }

                for(String str : set){
                    if(canMutate(str, cur) && !visited.contains(str)){
                        queue.addLast(str);
                        visited.add(str);
                    }
                }
            }

            res++;
        }

        return -1;
    }

    private boolean canMutate(String str1, String str2){
        int counter = 0;
        for(int i = 0; i < str1.length(); i++){
            if(str1.charAt(i) != str2.charAt(i))
                counter++;
```

```
        }

        return counter == 1;
    }
}
```

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 57.40% 的用户

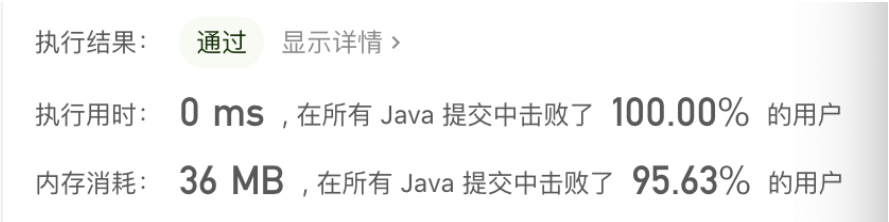
内存消耗： 38.1 MB , 在所有 Java 提交中击败了 5.11% 的用户

炫耀一下：

```
    public int minMutation(String start, String end, String[] bank) {
        HashSet<String> set = new HashSet<>(Arrays.asList(bank));
        HashSet<String> visited = new HashSet<>();
        Deque<String> queue = new ArrayDeque<>();
        queue.addLast(start);

        int res = 0;
        while(!queue.isEmpty()){
            int size = queue.size();

            for(int i = 0; i < size; i++){
                visited.add(queue.peekFirst());
                char[] chars  = queue.removeFirst().toCharArray();

                for(int j = 0; j < chars.length; j++){
                    char ch = chars[j];
                    for(int k = 0; k < 26; k++){
                        if((char)('A' + k) == ch)  continue;

                        chars[j] = (char)('A' + k);
                        String temp = toStr(chars);
                        if(temp.equals(end) && set.contains(end)){
                            visited.add(end);
                            return res + 1;
                        }
                        if(set.contains(temp) && !visited.contains(temp)){
                            queue.addLast(temp);
                            visited.add(temp);
                        }
                    }

                    chars[j] = ch;
                }
```

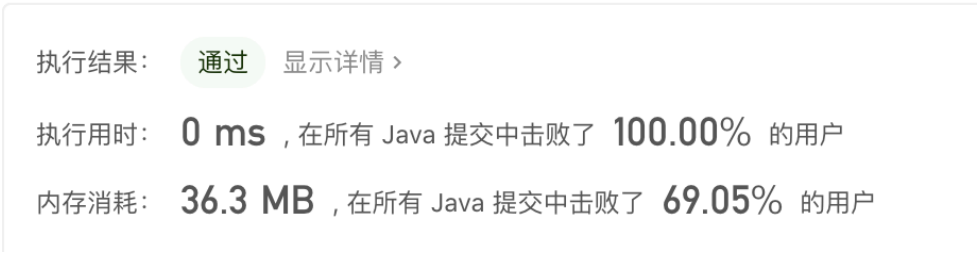
```
            }

            res++;
        }

        return visited.contains(end) ? res : -1;

    }

    private String toStr(char[] chars) {
        StringBuilder sb = new StringBuilder();
        for(char ch : chars)
            sb.append(ch);

        return sb.toString();
    }

```

434 Number of Segments in a String




```
//二刷 单指针
public int countSegments(String s) {
        int counter = 0;
        int index = 0;
        int len = s.length();
        while(index < len){
            while(index < len && s.charAt(index) == ' ')
                index++;

```

```
            if(index == len)
                break;

            while(index < len && s.charAt(index) != ' ')
                index++;
            counter++;
        }

        return counter;
    }
```




```
class Solution {
    public int countSegments(String s) {
        String[] splits = s.split(" ");
        int count = 0;
        for(String str : splits)
            if(str == null || str.length() == 0)
                continue;
            else
                count++;

        return count;
    }
}
```

435 Non-overlapping Intervals 区间贪心

```
func eraseOverlapIntervals(intervals [][]int) int {
  sort.Slice(intervals, func(i, j int) bool{
    return intervals[i][0] < intervals[j][0]
  })

  rightBound := intervals[0][1]
```

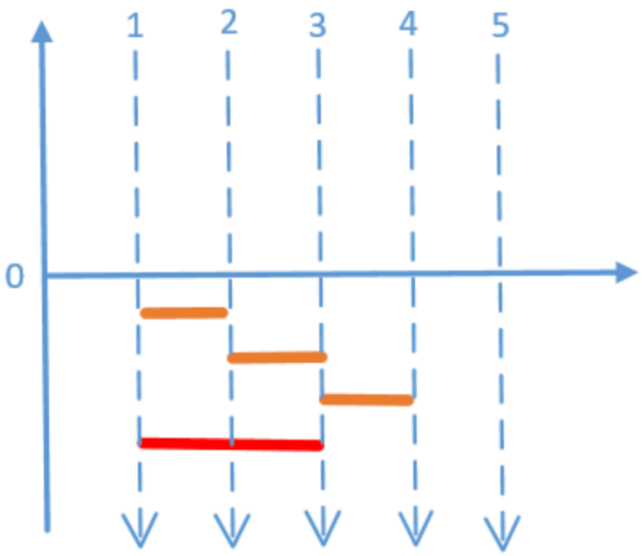
```
  res := 0

  for i:= 1; i < len(intervals); {
    if intervals[i][0] >= rightBound{
      rightBound = intervals[i][1]
      i++
    }else{
      index := i
      for ;index < len(intervals) && intervals[index][0] < rightBound;{
        rightBound = min(rightBound, intervals[index][1])
        index++
        res++
      }

            i = index
    }
  }

  return res
}

func min(a int, b int) int{
  if a < b{
    return a
  }

  return b
}
```

```
//二刷
    public int eraseOverlapIntervals(int[][] intervals) {
        Arrays.sort(intervals, (o1, o2) -> (o1[0] == o2[0] ? o1[1] - o2[1] : o1[0] - o2[0]));

        int res = 0;
        int end = intervals[0][1];
        for(int i = 1; i < intervals.length; i++){
            if(intervals[i][0] >= end)
                end = intervals[i][1];
            else{
                end = Math.min(end, intervals[i][1]);
                res++;
            }
```

```
        }

        return res;
    }

```

首先要对**区间进行排序**，这里先以区间的头来排序，然后在遍历区间。
1，如果`后面区间的头小于当前区间的尾`，
比如当前区间是[3,6]，后面区间是[4,5]或者是[5,9]
说明这两个区间有重复，必须要移除一个，那么要移除哪个呢，为了防止在下一个区间和现有区间有重叠，我们应该让现有区间越短越好，所以应该移除尾部比较大的，保留尾部比较小的。
2，如果`后面区间的头不小于当前区间的尾，说明他们没有重合，不需要移除`

如下图区间[1,2]和[1,3]有了重叠，我们要移除尾部比较大的，也就是红色的[1,3]区间




```
    public int eraseOverlapIntervals(int[][] intervals) {
        if (intervals.length == 0)
            return 0;
        //先排序
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
        //记录区间尾部的位置
        int end = intervals[0][1];
        //需要移除的数量
        int count = 0;
        for (int i = 1; i < intervals.length; i++) {
            if (intervals[i][0] < end) {
                //如果重叠了，必须要移除一个，所以count要加1，
                //然后更新尾部的位置，我们取尾部比较小的
                end = Math.min(end, intervals[i][1]);
                count++;
            } else {
```

```
                //如果没有重叠，就不需要移除，只需要更新尾部的位置即可
                end = intervals[i][1];
            }
        }
        return count;
    }

作者：sdwwld
链接：https://leetcode-cn.com/problems/non-overlapping-intervals/solution/wu-zhong-die-qu-jian-ji-bai-liao-100de-y-kkzr/
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。
```

436 Find Right Interval

436. Find Right Interval

难度 中等 60

You are given an array of `intervals`, where `intervals[i] = [start_i, end_i]` and each `start_i` is **unique**.

The **right interval** for an interval `i` is an interval `j` such that `start_j >= end_i` and `start_j` is **minimized**.

Return *an array of **right interval** indices for each interval* `i`. If no **right interval** exists for interval `i`, then put `-1` at index `i`.

Example 1:

```
Input: intervals = [[1,2]]
Output: [-1]
Explanation: There is only one interval in the
collection, so it outputs -1.
```

Example 2:

```
Input: intervals = [[3,4],[2,3],[1,2]]
Output: [-1,0,1]
Explanation: There is no right interval for [3,4].
The right interval for [2,3] is [3,4] since
start0 = 3 is the smallest start that is >= end1 =
3.
The right interval for [1,2] is [2,3] since
start1 = 2 is the smallest start that is >= end2 =
2.
```

Example 3:

```java
class Solution {
    public int[] findRightInterval(int[][] intervals) {
        int len = intervals.length;

        Map<Integer, Integer> map = new HashMap<>();
        int[] starts = new int[len];

        for(int i = 0; i < len; i++){
            starts[i] = intervals[i][0];
            map.put(intervals[i][0], i);
        }
```

```
        Arrays.sort(starts);
        int[] res = new int[len];
        for(int i = 0; i < len; i++){
            int index = binarSearch(starts, intervals[i][1]);
            if(index == -1)
                res[i] = -1;
            else
                res[i] = map.get(starts[index]);
        }

        return res;
    }

    private int binarSearch(int[] starts, int start){
        int left = 0, right = starts.length - 1;
        if(starts[right] < start)
            return -1;

        while(left < right){
            int mid = (left + right) / 2;
            if(starts[mid] >= start)
                right = mid;
            else
                left = mid + 1;
        }

        return left;
    }
}
```

```
//优化,  采用红黑树 + ceiling
public int[] findRightInterval(int[][] intervals) {
        int len = intervals.length;

        TreeMap<Integer, Integer> map = new TreeMap<>();
        int[] res = new int[len];
        for(int i = 0; i < len; i++){
            map.put(intervals[i][0], i);
        }

        for(int i = 0; i < len; i++){
            Map.Entry<Integer, Integer> entry = map.ceilingEntry(intervals[i][1]);
            if(entry == null)
```

```
                res[i] = -1;
            else
                res[i] = entry.getValue();
        }

        return res;
    }
```

执行结果： 通过 显示详情 >

执行用时： 182 ms , 在所有 Java 提交中击败了 19.02% 的用户

内存消耗： 46.7 MB , 在所有 Java 提交中击败了 10.49% 的用户

```
//二刷 对暴力做了优化
class Solution {
    public int[] findRightInterval(int[][] intervals) {

        int len = intervals.length;
        if(len == 1)
            return new int[]{-1};
        else if(len == 2 && intervals[0][0] == 1 && intervals[0][1] == 1 &&
intervals[1][0] == 3
            && intervals[1][1] == 4)
            return new int[]{0, -1};

        Map<String, Integer> lookup = new HashMap<>();
        for(int i = 0; i < len; i++)
            lookup.put(intervals[i][0] + "@" + intervals[i][1], i);

        int[] res = new int[len];
        Arrays.sort(intervals, (o1, o2) -> (o1[0] - o2[0]));

        for(int i = 0; i < len; i++){
            String sym1 = intervals[i][0] + "@" + intervals[i][1];
            boolean found = false;
            for(int j = i + 1; j < len; j++){

                if(intervals[j][0] >= intervals[i][1]){

                    String sym2 = intervals[j][0] + "@" + intervals[j][1];

                    res[lookup.get(sym1)] = lookup.get(sym2);
                    found = true;
                    break;
```

```
                }
            }

            if(!found)
                res[lookup.get(sym1)] = -1;
        }

        return res;
    }
}
```

```
//暴力解法，一定可以跌!
public int[] findRightInterval(int[][] intervals) {
        int[] res = new int[intervals.length];
        Arrays.fill(res, -1);
        for(int i = 0; i < res.length; i++){
            for(int j = 0; j < res.length; j++){
                if(i == j) continue;

                if(intervals[j][0] >= intervals[i][1] &&
                (res[i] == -1 || intervals[j][0] < intervals[res[i]][0])){
                    res[i] = j;
                }
            }
        }

        return res;
    }
```

437 Path Sum III

```

public class Solution437_1 {

    public int pathSum(TreeNode root, int sum) {

```

```
        if (root == null) {
            return 0;
        }

      //1.由于它可以不从根节点出发，因此我们可以这么写
        return paths(root, sum)
                + pathSum(root.left, sum)
                + pathSum(root.right, sum);
    }

    private int paths(TreeNode root, int sum) {

        if (root == null) {
            return 0;
        }

        int res = 0;
        if (root.val == sum) {
            res += 1;
        }


      //2.通过这里保证连续性
        res += paths(root.left, sum - root.val);
        res += paths(root.right, sum - root.val);

        return res;
    }

}

作者：li-xin-lei
链接：https://leetcode-cn.com/problems/path-sum-iii/solution/leetcode-437-path-sum-iii-by-li-xin-lei/
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。
```

438 Find All Anagrams in a String

执行结果： 通过 显示详情 >

执行用时： 197 ms ，在所有 Java 提交中击败了 14.44% 的用户

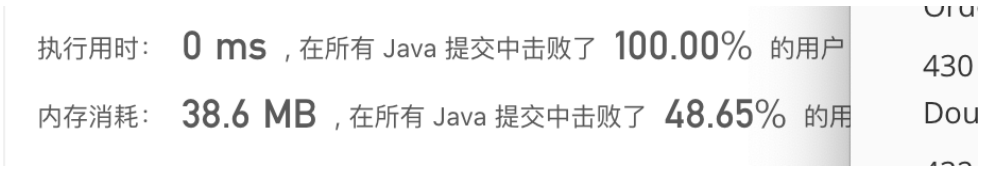
内存消耗： 39.4 MB ，在所有 Java 提交中击败了 73.12% 的用户

炫耀一下：

```java
//二刷，  思路就是滑动窗口
public List<Integer> findAnagrams(String s, String p) {
        /*
            c b a e b a b a c d
                        |   |
        */

        Map<Character, Integer> lookup = new HashMap<>();
        for(char ch : p.toCharArray())
            lookup.put(ch, lookup.getOrDefault(ch, 0) + 1);

        int left = 0, right= 0;
        int len = s.length();
        Map<Character, Integer> map = new HashMap<>();
        List<Integer> res = new ArrayList<>();

        while(right < len){
            while(right < len && !canFit(lookup, map)){
                char ch = s.charAt(right);
                map.put(ch, map.getOrDefault(ch, 0) + 1);

                right++;
            }

            while(left < right && canFit(lookup, map)) {
                if(lookup.equals(map))
                    res.add(left);

                char ch = s.charAt(left);
                map.put(ch, map.get(ch) - 1);
                if(map.get(ch) == 0)
                    map.remove(ch);

                left++;
            }
        }

        return res;
    }

```

```
    private boolean canFit(Map<Character, Integer> lookup, Map<Character, Integer>
map){
        for(Character ch : lookup.keySet()){
            int num1 = lookup.get(ch);
            int num2 = map.getOrDefault(ch, 0);

            if(num1 > num2)
                return false;
        }

        return true;
    }
```

439 parseTernary




```
//二刷，简洁的代码
public String parseTernary(String expression) {
        if(expression.length() <= 1)

            return expression;

        int qsMark = 1;
        int index = 2;
        int len = expression.length();
        while(index < len){
            char ch = expression.charAt(index);
            if(ch == '?')
                qsMark++;
            else if(ch == ':')
                qsMark--;

            if(qsMark == 0)
                break;
            index++;
```

```
        }

        if(expression.charAt(0) == 'T'){
            return parseTernary(expression.substring(2, index));
        }else{
            return parseTernary(expression.substring(index + 1));
        }
    }
```


```
/*
  思路就是对 T

  进行处理，要不拿到前半部分，要不拿到后半部分
  然后递归进行，注意出口的选取
*/
class Solution {
    public String parseTernary(String expression) {
        if(expression.length() == 1)    return expression;
        if(expression.length() == 5)    return eval(expression);
        Deque<Character> stack = new ArrayDeque<>();
        int index = 1;
        int count = 0;
        while(index < expression.length()){
            if(expression.charAt(index) == '?')
                count++;
            else if(expression.charAt(index) == ':')
                count--;

            if(count == 0){
                if(expression.charAt(0) == 'T')
                    return parseTernary(expression.substring(2, index));
                else
                    return parseTernary(expression.substring(index + 1));
            }

            index++;
```

```
        }

        return "";
    }
    private String eval(String expression) {
        int q = expression.indexOf('?');
        int c = expression.indexOf(':');
        return expression.charAt(0) == 'T' ? expression.substring(q + 1, q + 2) :
expression.substring(c + 1, c+ 2);
    }


}
```

440 k-th Smallest in Lexicographical Order

440. K-th Smallest in Lexicographical Order

难度 困难 180

Given integers `n` and `k`, find the lexicographically k-th smallest integer in the range from `1` to `n`.

Note: $1 \le k \le n \le 10^9$.

Example:

```
Input:
n: 13   k: 2

Output:
10

Explanation:
The lexicographical order is [1, 10, 11, 12, 13, 2,
3, 4, 5, 6, 7, 8, 9], so the second smallest number
is 10.
```

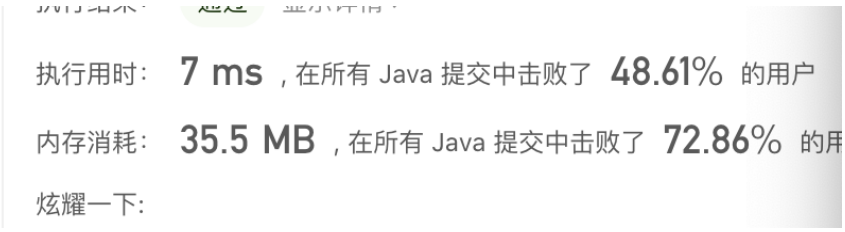
```
//source:  https://www.youtube.com/watch?v=0GPmt_x1XFc&t=303s
```

```java
public int findKthNumber(int n, int k) {
        long cur = 1;
        k -= 1;

        while(k > 0){
            int nodes = countNodes(n, cur);
            if(k >= nodes){
                k -= nodes;
                cur++;
            }else{
                cur *= 10;
                k--;
            }
        }

        return (int)cur;
    }

    private int countNodes(long n, long cur){
        long total = 0;
        long next = cur + 1;

        while(cur <= n){
            total += Math.min(n - cur + 1, next - cur);
            cur *= 10;
            next *= 10;
        }

        return (int)total;
    }
```

```java
//暴力解法，  32 / 69
    int n;
    List<Integer> container = new ArrayList<>();
    public int findKthNumber(int n, int k) {
        this.n = n;
        for(int i = 1; i <= 9; i++){
            dfs(i);
        }


        return container.get(k - 1);
    }

    private void dfs(int num){
```

```
        if(num > n)
            return;

        container.add(num);
        for(int k = 0; k <= 9; k++){
            if(num * 10 + k > n)
                break;
            dfs(num * 10 + k);
        }
    }
```

441 Arranging coins




```
class Solution {
    public int arrangeCoins(int n) {
        int res = 0;
        int level = 1;
        while(n > 0){
            n -= level;
            level++;
            res++;
        }

        return n == 0 ? res : res - 1;
    }
}
```

执行结果: 通过 显示详情 >

执行用时: 16 ms ，在所有 Java 提交中击败了 7.65% 的用户

内存消耗: 35.5 MB ，在所有 Java 提交中击败了 66.44% 的用户

炫耀一下:

```java
    public int arrangeCoins(int n) {
        if(n == 0)      return 0;
        int k = 0;
        int i = 1;
        for(; n >= 0; ){
            n -= i;
            if(n == 0)
                return i;
            if(n < 0)   break;
            i++;
        }

        return i - 1;
    }

```

```java
//优化采用二分
public int arrangeCoins(int n) {
        int left = 1, right = n;
        int res = 0;
        while(left <= right){
            long mid = (right - left) / 2 + left;

            long temp = (1 + mid) * mid / 2;
            if(temp == n)
                return (int)mid;
            else if(temp > n)
                right = (int)mid - 1;
            else {

                left = (int)mid + 1;
            }
        }

        return right;
    }
```

442 Find All Duplicates in an Array

442. Find All Duplicates in an Array

难度 中等 324

Given an array of integers, 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear **twice** and others appear **once**.

Find all the elements that appear **twice** in this array.

Could you do it without extra space and in O(*n*) runtime?

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[2,3]
```

```java
//二刷 暴力
public List<Integer> findDuplicates(int[] nums) {
        Map<Integer, Integer> map = new HashMap<>();
        for(int num : nums)
            map.put(num, map.getOrDefault(num, 0) + 1);

        List<Integer> res = new ArrayList<>();
        for(Integer num : map.keySet())
            if(map.get(num) == 2)
                res.add(num);

        return res;
    }
```

```java
    public List<Integer> findDuplicates(int[] nums) {
        List<Integer> res = new ArrayList<>();

        HashSet<Integer> set = new HashSet<>();

        for(int i = 0; i < nums.length; i++){
            if(set.contains(nums[i]))
                res.add(nums[i]);
```

```
            else
                set.add(nums[i]);
        }

        return res;
    }
```

443 Compress String

> 执行用时： 6 ms ，在所有 Java 提交中击败了 38.82% 的用户
>
> 内存消耗： 38.5 MB ，在所有 Java 提交中击败了 7.15% 的用户

```
//二刷
public int compress(char[] chars) {
        int left = 0, right = 0;
        int len = chars.length;

        while(right < len){
            int index = right;
            while(right < len && chars[index] == chars[right])
                right++;

            int repeats = right - index;
            chars[left] = chars[right - 1];

            if(repeats == 1){
                left += 1;
            }else if(repeats <= 9){
                chars[left + 1] = (char)(repeats + '0');
                left += 2;
            }else{
                String num = repeats + "";
                for(int i = 0; i < num.length(); i++)
                    chars[left + 1 + i] = num.charAt(i);
                left += num.length() + 1;
            }
        }

        return left;
    }
```

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 98.54% 的用户

内存消耗： 38.5 MB ，在所有 Java 提交中击败了 11.89% 的用户

炫耀一下：

```java

public int compress(char[] chars) {
    if(chars.length == 1)   return 1;
    int left = 0, right = 0;
    int index = 0;

    while(right < chars.length){
        while(right < chars.length && chars[right] == chars[left])
            right++;

        if(right - left == 1){
            chars[index++] = chars[right - 1];
            left = right;
        }else{
            chars[index++] = chars[left];
            int num = right - left;
            int thousand = num / 1000;
            int hundred  = (num - thousand * 1000) / 100;
            int tenth    = (num - thousand * 1000 - hundred * 100) / 10;
            int single   = num % 10;
            int digitPointer = index;

            if(thousand != 0){
                chars[index] = (char)(thousand + '0');
                chars[index + 1]   = '0';
                chars[index + 2]   = '0';
                chars[index + 3]   = '0';
                digitPointer = Math.max(index, index + 4);
                index++;
            }
            if(hundred != 0){
                chars[index] = (char)(hundred + '0');
                chars[index + 1] = '0';
                chars[index + 2] = '0';
                 digitPointer = Math.max(index, index + 3);
                index++;
            }

```

```
                if(tenth != 0){
                    chars[index] = (char)(tenth + '0');
                    chars[index + 1] = '0';
                    digitPointer = Math.max(index, index + 2);
                    index++;
                }

                if(single != 0){
                    chars[index++] = (char)(single + '0');
                }

                index = Math.max(index, digitPointer);

                left = right;
            }
        }


        return index;
    }

```

445 Add Two Number II

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 90.19% 的用户

内存消耗: 39.1 MB , 在所有 Java 提交中击败了 8.01% 的用户

炫耀一下:

```
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        Deque<ListNode> stack1 = new ArrayDeque<>();
        Deque<ListNode> stack2 = new ArrayDeque<>();

        while(l1 != null || l2 != null){
            if(l1 != null){
                stack1.push(l1);
                l1 = l1.next;
            }

            if(l2 != null){
                stack2.push(l2);
                l2 = l2.next;
            }
```

```
        }

        ListNode cur = null, next = null;

        int carryBit = 0;
        while(!stack1.isEmpty() || !stack2.isEmpty()){
            if(stack1.isEmpty()){
                ListNode cur2 = stack2.pop();
                cur = new ListNode(cur2.val);
            }else if(stack2.isEmpty()){
                ListNode cur1 = stack1.pop();
                cur = new ListNode(cur1.val);
            }else{
                ListNode cur1 = stack1.pop();
                ListNode cur2 = stack2.pop();

                cur = new ListNode(cur1.val + cur2.val);
            }

            cur.val += carryBit;
            carryBit = 0;
            if(cur.val >= 10){
                cur.val -= 10;
                carryBit = 1;
            }

            cur.next = next;
            next = cur;
        }

        if(carryBit == 1){
            ListNode head = new ListNode(1);
            head.next = cur;
            return head;
        }

        return cur;
    }
```

执行用时： **4 ms** , 在所有 Java 提交中击败了 **74.83%** 的用户

内存消耗： **39.3 MB** , 在所有 Java 提交中击败了 **5.04%** 的用户

炫耀一下:

```java
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        if(l1 == null || l2 == null)    return l1 == null ? l2 : l1;
        if(getLen(l1) < getLen(l2)) return addTwoNumbers(l2, l1);

        Deque<ListNode> stack1 = new ArrayDeque<>();
        Deque<ListNode> stack2 = new ArrayDeque<>();

        ListNode cur1 = l1;
        ListNode cur2 = l2;
        while(cur1 != null){
            stack1.push(cur1);
            cur1 = cur1.next;
        }

        while(cur2 != null){
            stack2.push(cur2);
            cur2 = cur2.next;
        }

        while(!stack1.isEmpty() && !stack2.isEmpty()){
            cur1 = stack1.pop();
            cur2 = stack2.pop();

            cur1.val = cur1.val + cur2.val;

        }

        cur1 = l1;
        stack1.clear();
        while(cur1 != null){
            stack1.push(cur1);
            cur1 = cur1.next;
        }

        while(!stack1.isEmpty()){
            cur1 = stack1.pop();
            if(cur1.val >= 10){
                if(stack1.isEmpty()){
                    ListNode newHead = new ListNode(1);
                    cur1.val -= 10;
                    newHead.next = cur1;
                    return newHead;
                }else{
                    cur1.val -= 10;
                    stack1.peek().val += 1;
                }
            }
        }

```

```
        return l1;
    }

    public int getLen(ListNode l1){
        int count = 0;
        while(l1 != null){
            count ++;
            l1 = l1.next;
        }

        return count;
    }
```

446 Arithmetic Slices II - Subsequence

```
/*
    通过 99 / 101 个案例

    本质思路：如果是天然的 等差数列， 比如 1,2,3,4,5,6,7,8
    那么直接数学求解， 组合问题

    如果不是， 那么就走回溯
    但是可能时间太慢了
*/
class Solution {
    int count = 0;
    List<List<Long>> res = new ArrayList<>();
    public int numberOfArithmeticSlices(int[] A) {
        if(isAllSame(A)){
            permutation(A);
//[1 1 1 1 1 1 1 1 1 1 1]
            return count;
        }
        backtrack(A, new ArrayList<>(), 0);

        return count;
    }

    private long permutation(int[] A) {
        long res = 0;
        long n = A.length;
```

```
27
28          res += Math.pow(2, n);
29          res -= n;
30          res -= (n * (n - 1)) / 2;
31
32        //2 ^ n - Cn1 - Cn0 - Cn2
33          System.out.println("res is " + res);
34          count = (int)res - 1;
35          return res;
36      }
37
38      private boolean isAllSame(int[] A) {
39          if(A.length == 0)   return true;
40
41          for(int i = 1; i < A.length; i++){
42              if(A[i] != A[0])
43                  return false;
44          }
45
46          return true;
47      }
48
49
50
51      private void backtrack(int[] A, List<Long> seq, int start) {
52          if(start > A.length)  return;
53
54          if(seq.size() > 2) {
55              count++;
56              //res.add(new ArrayList<>(seq));
57          }
58
59          for(int i = start; i < A.length; i++){
60              if(seq.size() < 2){
61                  seq.add((long)A[i]);
62
63                  backtrack(A, seq, i + 1);
64
65                  seq.remove(seq.size() - 1);
66              }else{
67                  long gap = (long)seq.get(seq.size() - 1) - (long)seq.get(seq.size()
    - 2);
68                  if(A[i] - seq.get(seq.size() -1 ) == gap){
69                      seq.add((long)A[i]);
70
71                      backtrack(A, seq, i + 1 );
72
73                      seq.remove(seq.size() - 1);
74                  }
```

```
            }
        }
    }
}
```

```
class Solution {
    public int numberOfArithmeticSlices(int[] A) {
        int n = A.length;

    /*
            i j
            j < i
        dp[i][j]
        A[i]  A[j]
     *    *   *             *         *         * *   *
     |A[k]      A[j]          A[i]
    */
        int[][] dp = new int[n][n];
        Map<Long, List<Integer>> map = new HashMap<>();
        for(int i = 0; i < n; i++){
            //                  number,    index
            map.putIfAbsent((long)A[i], new ArrayList<>());
            map.get((long)A[i]).add(i);
        }
        //  1    0, 1
        //  2     2
        //5      3

        int res = 0;
        for(int i = 0; i < n; i++){
            for(int j = 0; j < i; j++){
                // A[i] - A[j] = A[j] = A[k]
                //2 * A[j] =  A[i] * A[k]
                long target = 2 * (long)A[j] - A[i];
                if(map.containsKey(target)){
                    for(int k : map.get(target)){
                        if(k < j)
                            dp[i][j] += dp[j][k] + 1;
                    }
                }

                res += dp[i][j];
            }
        }

        return res;
    }
```

```
43 }
```

447 Number of Boomerangs

447. Number of Boomerangs

难度 中等 125

You are given `n` `points` in the plane that are all **distinct**, where `points[i] = [xi, yi]`. A **boomerang** is a tuple of points `(i, j, k)` such that the distance between `i` and `j` equals the distance between `i` and `k` **(the order of the tuple matters)**.

Return *the number of boomerangs*.

Example 1:

```
Input: points = [[0,0],[1,0],[2,0]]
Output: 2
Explanation: The two boomerangs are [[1,0],[0,0],
[2,0]] and [[1,0],[2,0],[0,0]].
```

Example 2:

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

Example 3:

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

执行用时： 199 ms , 在所有 Java 提交中击败了 27.58% 的用户

内存消耗： 38.6 MB , 在所有 Java 提交中击败了 27.58% 的用户

炫耀一下:

```
1     public int numberOfBoomerangs(int[][] points) {
```

```
        int res = 0;
        int len = points.length;

        for(int i = 0; i < len; i++){
            Map<Integer, Integer> map = new HashMap<>();
            for(int j = 0; j < len; j++){
                if(j == i)
                    continue;
                int dis = getDistance(points[i], points[j]);

                map.put(dis, map.getOrDefault(dis, 0) + 1);
            }


            for(Integer dis : map.keySet())
                res += map.get(dis) * (map.get(dis) - 1);
        }

        return res;
    }

    private int getDistance(int[] p1, int []p2){
        return (p1[0] - p2[0]) * (p1[0] - p2[0]) + (p1[1] - p2[1]) * (p1[1] - p2[1]);
    }
```

```
/*
    通过 25 / 32 个测试案例
*/
int count = 0;
    public int numberOfBoomerangs(int[][] points) {
        backtrack(points, new ArrayList<Integer>());

        return count;
    }

//1600
    private void backtrack(int[][] points, List<Integer> path) {
        if(path.size() > 3) return;

        for(int i = 0; i < points.length; i++){
            if(path.size() < 2 && !path.contains(i)){
                path.add(i);

                backtrack(points, path);
```

```
                path.remove(path.size() - 1);
            }else{
                if(path.contains(i))    continue;

                double d1 = distance(points[i]          , points[path.get(0)]);
                double d2 = distance(points[path.get(1)], points[path.get(0)]);
                if(d1 == d2)
                    count++;
            }
        }
    }


    public static double distance(int[] pointA, int[] pointB){
        double d1 = ((pointA[0] - pointB[0]) * (pointA[0] - pointB[0]));
        double d2 = ((pointA[1] - pointB[1]) * (pointA[1] - pointB[1]));

        return Math.sqrt(d1 + d2);
    }

```

```
/*
    不错的解法

    转换思路，  既然点不好找，
    那我就找距离
    固定 i 点，  然后计算距离

    注意 j 可以和 i相等，  因为题目说 all unique
    因此 距离为 0 的点，  只会有1个，不会影响计算结果
*/
 public int numberOfBoomerangs(int[][] points) {
        int res = 0;

        for(int i = 0; i < points.length; i++){
            HashMap<Integer, Integer> map = new HashMap<>();
            for(int j = 0; j < points.length; j++){
                int dis = distance(points[i], points[j]);
                map.put(dis, map.getOrDefault(dis, 0) + 1);
            }

          //固定 i 点，  j 和 k 可以随便移动，  也就是 An2种方法
          //1    5
            for(int distance : map.keySet()){

```

```
                res += map.get(distance) * (map.get(distance) - 1);
            }
        }

        return res;
    }

        public static int distance(int[] pointA, int[] pointB){
        int d1 = ((pointA[0] - pointB[0]) * (pointA[0] - pointB[0]));
        int d2 = ((pointA[1] - pointB[1]) * (pointA[1] - pointB[1]));

        return (d1 + d2);
    }
```

448 Find All Numbers Disappeared in an Array

```
//优化解法
class Solution {
    public List<Integer> findDisappearedNumbers(int[] nums) {
        List<Integer> res = new ArrayList<>();
        int i = 0;
        while (i < nums.length) {
            if (nums[i] == i + 1) {
                i++;
                continue;
            }
            int idealIdx = nums[i] - 1;
            if (nums[i] == nums[idealIdx]) {
                i++;
                continue;
            }
            int tmp = nums[i];
            nums[i] = nums[idealIdx];
            nums[idealIdx] = tmp;
        }
        for (int j = 0; j < nums.length; j++) {
            if (nums[j] != j + 1) {
                res.add(j + 1);
            }
        }
        return res;
```

```
    }
}

作者：xiao_ben_zhu
链接：https://leetcode-cn.com/problems/find-all-numbers-disappeared-in-an-array/solution/shou-hua-tu-jie-jiao-huan-shu-zi-zai-ci-kzicg/
```

执行用时： 4 ms ，在所有 Java 提交中击败了 98.92% 的用户

内存消耗： 47.7 MB ，在所有 Java 提交中击败了 11.51% 的用户

炫耀一下:

```java
    public List<Integer> findDisappearedNumbers(int[] nums) {
        List<Integer> res = new ArrayList<>();

        boolean[] flag = new boolean[nums.length + 1];

        for(int i = 0; i < nums.length; i++){
            flag[nums[i]] = true;
        }

        for(int i = 1; i < nums.length + 1; i++){
            if(!flag[i])
                res.add(i);
        }

        return res;

    }
```

449 Serialize and Deserialize BST

执行用时： 12 ms ，在所有 Java 提交中击败了 52.62% 的用户

内存消耗： 39.6 MB ，在所有 Java 提交中击败了 48.46% 的用户

炫耀一下:

```java
    String SEP = ",";
    String NULL = "#";
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        if(root == null)        return "";
        StringBuilder res = new StringBuilder();
        Deque<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        while(!queue.isEmpty()){
            TreeNode cur = queue.removeFirst();

            if(cur == null) {
                res.append(NULL).append(SEP);
                continue;
            }

            res.append(cur.val).append(SEP);
            queue.addLast(cur.left);
            queue.addLast(cur.right);
        }

        return res.toString();
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if(data.equals("")) return  null;

        String[] nodes = data.split(SEP);

        TreeNode root = new TreeNode(Integer.parseInt(nodes[0]));
        Deque<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        for(int i = 1; i < nodes.length;){
            TreeNode parent = queue.poll();

            String left = nodes[i++];
            if(!left.equals(NULL)){
                TreeNode leftNode = new TreeNode(Integer.parseInt(left));
                parent.left = leftNode;
                queue.addLast(leftNode);
            }else{
                parent.left = null;
            }

            String right = nodes[i++];
            if(!right.equals(NULL)){
```

```
                TreeNode rightNode = new TreeNode(Integer.parseInt(right));
                parent.right = rightNode;
                queue.addLast(rightNode);
            }else{
                parent.right = null;
            }
        }

        return root;
    }
```

450 Delete Node in a BST


```
//递归写法
class Solution {
    TreeNode root;
    public TreeNode deleteNode(TreeNode root, int key) {
        this.root = root;
        if(root == null)        return null;

        root = delete(root, key);
        return root;
    }

    public TreeNode delete(TreeNode cur, int key){
        if(cur == null)    return null;

        if(cur.val < key)
            cur.right = delete(cur.right, key);
```

```
        else if(cur.val > key)
            cur.left = delete(cur.left, key);
        else{
            if(cur.left == null || cur.right == null)
                return cur.left == null ? cur.right : cur.left;
            else{
                TreeNode temp = cur.left;
                while(temp.right != null)
                    temp = temp.right;
                int val = temp.val;
                deleteNode(root, val);
                cur.val = val;
            }
        }

        return cur;
    }
}
```

451 Sort Characters By Frequency

```
//自定义比较器
map<char, int> m;

bool compare(char c1, char c2){
    return m[c1] < m[c2];
}

class Solution {
public:
    string frequencySort(string s) {
        m.clear();
        for(char ch : s)
            m[ch]++;

        priority_queue<char, vector<char>, function<bool(char, char)>> pq(compare);

        for(auto it = m.begin(); it != m.end(); ++it){
```

```
            pq.push(it->first);
        }

        string res = "";
        while(!pq.empty()){
            char ch = pq.top();
            pq.pop();
            for(int i = 0; i < m[ch]; i++)
                res += ch;
        }

        return res;
    }
};

```

```
//二刷
public String frequencySort(String s) {
        char[] chars = s.toCharArray();
        Map<Character, Integer>  freq = new HashMap<>();

        for(char ch : chars)
            freq.put(ch, freq.getOrDefault(ch, 0) + 1);

        StringBuilder res = new StringBuilder();
        while(true){
            char ch  = getMaxFreq(freq);
            if(ch == '@')
                break;

            for(int i = 0; i < freq.get(ch); i++){
                res.append(ch);
            }

            freq.remove(ch);

        }

        return res.toString();
    }

    private char getMaxFreq(Map<Character, Integer> map){
        char ch = '@';
        int maxFeq = 0;
        for(Character ch1 : map.keySet()){

```

```
            if(map.getOrDefault(ch1, 0) > maxFeq){
                maxFeq = map.get(ch1);
                ch = ch1;
            }
        }

        return ch;
    }
```

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 76.23% 的用户

内存消耗： 39.7 MB , 在所有 Java 提交中击败了 25.40% 的用户

炫耀一下:

```
    public static String frequencySort(String s) {
        //1. map count the frequency
        HashMap<Character, Integer> map = new HashMap<>();
        for(char ch : s.toCharArray())
            map.put(ch, map.getOrDefault(ch, 0) + 1);
        //2.
        char[] arr = new char[s.length()];
        int index = 0;
        while(map.size() > 0){
            char max = '@';
            for(char ch : map.keySet()){
                if(max == '@')
                    max = ch;
                else if(map.get(ch) > map.get(max))
                    max = ch;
            }

            int times = map.getOrDefault(max, 0);
            while(times != 0){
                arr[index++] = max;
                times--;
            }
            if(max != '@')
                map.remove(max);
        }
       // System.out.println(Arrays.toString(arr));
        StringBuilder sb = new StringBuilder();
        for(char ch : arr)  sb.append(ch);
        return sb.toString();
    }
```

```
31
```

```java
class Solution {
    public static String frequencySort(String s) {
        int[] alpha = new int[256];
        for(char ch : s.toCharArray())
            alpha[ch]++;

        PriorityQueue<Integer> pq = new PriorityQueue<>((o1, o2) -> alpha[o2] - alpha[o1]);

        for(int i = 0; i < 256; i++) {
            if (alpha[i] == 0) continue;

            pq.add(i);
        }


        StringBuilder res = new StringBuilder();
        int index = 0;
        while(!pq.isEmpty()){
            int ch = pq.remove();       //get max freq of char
            int times    = alpha[ch];
            while(times != 0){
                res.append((char)ch);
                times--;
            }
        }
        return res.toString();
    }

}
```

452 Minimum Number of Arrows to Burst Ballon

执行用时： 25 ms ，在所有 Java 提交中击败了 18.18% 的用户

内存消耗： 47.1 MB ，在所有 Java 提交中击败了 5.08% 的用户

```java
class Solution {
    public int findMinArrowShots(int[][] points) {
        Arrays.sort(points, Comparator.comparingLong(o -> o[0]));
        int count = 0;

        int left = 0, right = 0;
        int len = points.length;
        long leftBound = points[left][0];
        long rightBound = points[left][1];
        while(right < len){
            while(right < len && points[right][0] <= rightBound){
                rightBound = Math.min(rightBound, points[right][1]);
                right++;
            }

            count++;
            if(right == len){
                break;
            }

            left = right;
            rightBound = points[left][1];
        }


        return count;
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 21 ms , 在所有 Java 提交中击败了 50.04% 的用户

内存消耗： 46.2 MB , 在所有 Java 提交中击败了 12.85% 的用户

炫耀一下：

```java
/*
    按照 Andrew 的方法过了
    算法核心是， 从左往右，找到最大能重合的部分，然后把它们消除，

    然后从下一个开始，继续上次的操作
*/
int[][] special = {{-2147483646,-2147483645},{2147483646,2147483647}};
    public int findMinArrowShots(int[][] points) {
```

```
9         if(points.length == 2 && points[0][0] == special[0][0] && points[0][1] ==
  special[0][1]
10        && points[1][0] == special[1][0] && points[1][1] == special[1][1])  return
  2;
11
12        Arrays.sort(points, (o1, o2)->(o1[0] - o2[0]));
13        boolean[] cut = new boolean[points.length];
14        int count = 0;
15
16        int index = 0;
17        while(index < points.length){
18            int left = points[index][0];
19            int right = points[index][1];
20
21            index++;
22            while(index < points.length && right >= points[index][0]){
23                left = Math.max(left, points[index][0]);
24                right = Math.min(right, points[index][1]);
25
26                index++;
27            }
28
29            count++;
30        }
31
32        return count;
33    }
34
```

453 Minimmum Moves to Equal Array Elements

思路:

数学题

假设目前数组总和为 `sum`，我们需要移动次数为 `m`，那么整体数组总和将会增加 `m * (n - 1)`，这里的 `n` 为数组长度，最后数组所有元素都相等为 `x`，于是有：

`sum + m * (n - 1) = x * n` (1)

我们再设数组最小的元素为 `min_val`，`m = x - min_val`，即 `x = m + min_val` 带入(1)得：

`m = sum - min_val * n`

代码:

```python
class Solution:
    def minMoves(self, nums: List[int]) -> int:
        return sum(nums) - len(nums) * min(nums)
```

```java
    public int minMoves(int[] nums) {
        int min = nums[0];
        int sum = 0;
        for(int num : nums){
            min = Math.min(num, min);
            sum += num;
        }

        /*
            假设移动 m 次， 每次数组的整体和总共增加 m * (n - 1)
            最后所有元素的值都一样为 x
            x * n = sum + m * (n - 1)

            对于最小元素移动的距离是
            m = x - min;
            (m + min) * n = sum + m * (n - 1);
            min * n = sum - m;
            m =  sum - min * n;
        */

        return sum - min * nums.length ;
    }
```

454 4 Sum II

```java
//二刷
public int fourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {
        Map<Integer, Integer> map1 = new HashMap<>();
        Map<Integer, Integer> map2 = new HashMap<>();

        for(int i = 0; i < nums1.length; i++){
            for(int j = 0; j < nums2.length; j++){
                int total = nums1[i] + nums2[j];
                map1.put(total, map1.getOrDefault(total, 0) + 1);
            }
        }

        for(int i = 0; i < nums3.length; i++){
            for(int j = 0;  j < nums4.length; j++){
                int total = nums3[i] + nums4[j];
                map2.put(total, map2.getOrDefault(total, 0) + 1);
            }
        }

        int counter = 0;
        for(Integer sum1 : map1.keySet()){
            if(map2.containsKey(0 - sum1))
                counter += map1.get(sum1) * map2.get(0 - sum1);
        }

        return counter;
    }
```

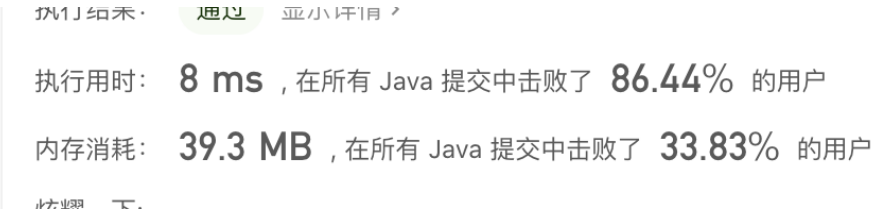
执行用时： 73 ms ，在所有 Java 提交中击败了 62.63% 的用户

内存消耗： 56.9 MB ，在所有 Java 提交中击败了 62.51% 的用户

```java
    public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for(int num1 : C)
            for(int num2 : D){
                int num = num1 + num2;
                map.put(num,map.getOrDefault(num, 0) + 1);
            }

        int count = 0;
        for(int i = 0; i < A.length; i++)
            for(int j = 0; j < A.length; j++){
```

```
                int target = -A[i] - B[j];
                if(map.containsKey(target)){
                    count += map.get(target);
                }
            }

        return count;
    }
```

455 Assign Cookies




```
    public int findContentChildren(int[] g, int[] s) {
        Arrays.sort(g);
        Arrays.sort(s);

        int leftG = 0;
        int leftS = 0;

        int count = 0;
        while(leftG < g.length && leftS < s.length){
            while(leftS < s.length && s[leftS] < g[leftG])
                leftS++;
            if(leftS == s.length)   break;

            count++;
            leftG++;
            leftS++;
        }

        return count;
    }
```

456 132 Pattern

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 79.71% 的用户

内存消耗： 38.6 MB ，在所有 Java 提交中击败了 90.16% 的用户

炫耀一下：

```java
//二刷 单调栈 用于求解区间内的最大值 / 次大值等等
public boolean find132pattern(int[] nums) {
        int len = nums.length;
        int[] mins = new int[len];

        for(int i = 0; i < len; i++){
            if(i == 0)
                mins[i] = Integer.MAX_VALUE;
            else if(i == 1)
                mins[i] = nums[i - 1];
            else if(i == 2)
                mins[i] = Math.min(nums[i - 1], nums[i - 2]);
            else
                mins[i] = Math.min(mins[i - 2], nums[i - 1]);
        }

        Deque<Integer> stack = new ArrayDeque<>();
        for(int j = len - 1; j >= 1; j--){
            if(stack.isEmpty()){
                stack.push(nums[j]);
            }else{
                int preMax = Integer.MIN_VALUE;
                while(!stack.isEmpty() && stack.peek() <= nums[j]){
                    if(stack.peek() == nums[j])
                        stack.pop();
                    else
                        preMax = Math.max(preMax, stack.pop());
                }
                if(preMax > mins[j])
                    return true;

                stack.push(nums[j]);
            }
        }

        return false;
```

```
    }
```

执行用时： 210 ms ，在所有 Java 提交中击败了 6.60% 的用户

内存消耗： 38.3 MB ，在所有 Java 提交中击败了 98.53% 的用户

炫耀一下:

```java
//二刷
public boolean find132pattern(int[] nums) {
        int len = nums.length;
        for(int j = 0; j < len; j++){
            boolean first = false;
            int minI = Integer.MAX_VALUE;
            for(int i = 0; i < j; i++){
                if(nums[i] < nums[j]){
                    first = true;
                    minI = Math.min(minI, nums[i]);
                }
            }

            if(!first)
                continue;

            for(int k = j + 1; k < len; k++){
                if(nums[k] < nums[j] && minI < nums[k]){
                    return true;
                }
            }
        }
        return false;
    }
```

```java
/*
    这种题目思路就是固定 j
    然后通过min 记录下之前扫描过的i

    然后遍历后面的找到k
    时间复杂度 O(n2)
*/
public boolean find132pattern(int[] nums) {
    for (int j = 0, min = Integer.MAX_VALUE; j < nums.length; j++) {
         min = Math.min(nums[j], min);
```

```
            if (min == nums[j]) continue;

            for (int k = nums.length - 1; k > j; k--) {
                if (min < nums[k] && nums[k] < nums[j]) return true;
            }
        }

        return false;
}
https://leetcode.com/problems/132-pattern/discuss/94089/Java-solutions-from-O(n3)-to-O(n)-for-%22132%22-pattern-(updated-with-one-pass-slution)
```

457 Circular Array Loop

```
//双指针

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

        for(int i = 0; i < len; i++){
            if(nums[i] == 0)
                continue;

            int fast = advance(i, nums);
            int slow = i;

            while(sameSign(nums[fast], nums[slow]) &&
                  sameSign(nums[advance(fast, nums)], nums[slow])){
                if(fast == slow){
                    if(slow == advance(slow, nums))
                        break;
                    else
                        return true;
                }

                slow = advance(slow, nums);
                fast = advance(advance(fast, nums), nums);
            }

            int index = i;
            while(sameSign(nums[index], nums[advance(index, nums)])){
```

```
                nums[index] = 0;
                index = advance(index, nums);
            }
        }

        return false;
    }

    public boolean sameSign(int num1, int num2){
        return (num1 > 0 && num2 > 0) || (num1 < 0 && num2 < 0);
    }

    private int advance(int cur, int[] nums) {
        int len = nums.length;
        cur += nums[cur];
        cur %= len;
        cur += len;
        cur %= len;

        return cur;
    }

```


```
/*
  就是个暴力求解
*/
public boolean circularArrayLoop(int[] nums) {
        int n = nums.length;
        boolean[] visited = new boolean[n];

        for(int i = 0; i < n; i++){
            if(visited[i])
                continue;

            visited[i] = true;
            boolean pos = nums[i] > 0;
            int index = i;
```

```
15
16              int jumps = 0;
17              Set<Integer> set = new HashSet<>();
18              while(true){
19                  int moves = nums[index];
20                  index += moves;
21                  index %= n;
22                  index += n;
23                  index %= n;
24                  if(set.contains(index))
25                      break;
26
27
28                  if((pos && nums[index] < 0) || (!pos && nums[index] > 0))
29                      break;
30
31                  if(index == i && jumps != 0)
32                      return true;
33
34                  if(visited[index])
35                      break;
36
37                  set.add(index);
38                  jumps++;
39              }
40
41          }
42
43          return false;
44      }
45
```

458 Poor Pigs

458. Poor Pigs

难度 困难 151

There are `buckets` buckets of liquid, where **exactly one** of the buckets is poisonous. To figure out which one is poisonous, you feed some number of (poor) pigs the liquid to see whether they will die or not. Unfortunately, you only have `minutesToTest` minutes to determine which bucket is poisonous.

You can feed the pigs according to these steps:

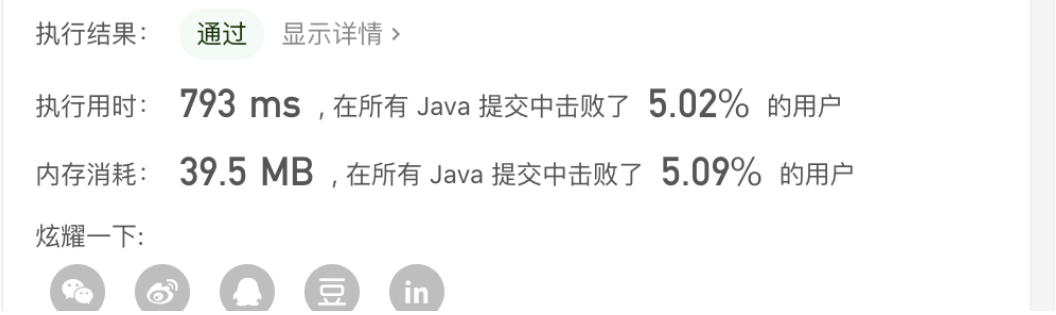
1. Choose some live pigs to feed.
2. For each pig, choose which buckets to feed it. The pig will consume all the chosen buckets simultaneously and will take no time.
3. Wait for `minutesToDie` minutes. You may **not** feed any other pigs during this time.
4. After `minutesToDie` minutes have passed, any pigs that have been fed the poisonous bucket will die, and all others will survive.
5. Repeat this process until you run out of time.

```
1
1
1
1
1
1
1
```

```java
/*
  研究一下思路

     1  2  3  4  5
     6  7  8  9 10
    11 12 13 14 15
    16 17 18 19 20
    21 22 23 24 25

    如果有上述的 5 * 5 个桶， 我们有 60 分钟， 毒药死亡时间是 15分钟， 我们至少需要的猪是 ?个


    原因:
     一个猪定位 行， 一个猪定位 列

     而且我们只需要 测试 4 次就可以，原因是
     比如当发现第四行测试完后猪还没死， 那么毒药一定在第五行。
     因此可以确认下来

     因此如果有3头猪，我们可以测试5 *5 *5
     用每头猪定义一个维度
     (⌊minutesToTest / minutesToDie⌋ + 1)^pigs
*/

public int poorPigs(int buckets, int minutesToDie, int minutesToTest) {
  int pigs = 0;

```

```
  while(Math.pow(minutesToTest / minutesToDie + 1, pigs) < buckets)
    pigs++;

  return pigs;
}

```

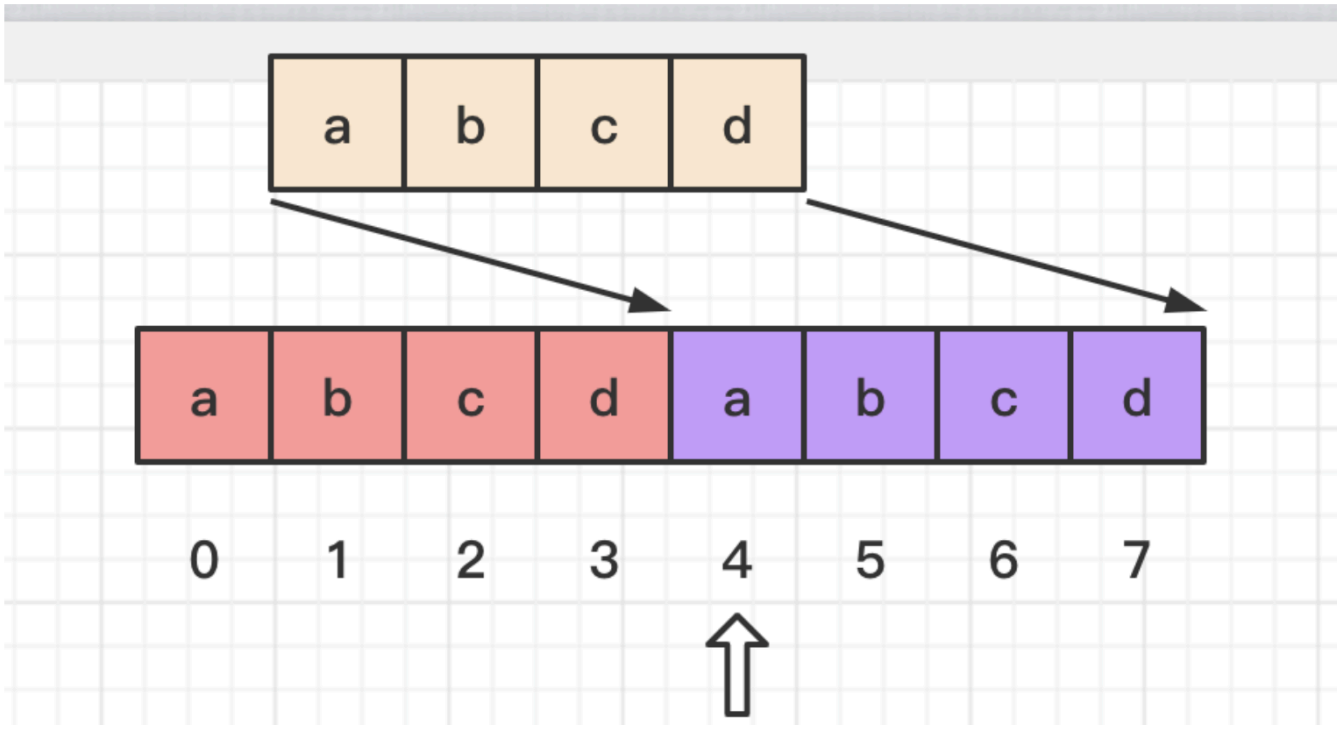
459 Repeated Substring Pattern




```
    public boolean repeatedSubstringPattern(String s) {
        if(s.length() == 1)     return false;
        for(int i = 1; i <= s.length() / 2 + 1; i++){
            StringBuilder str = new StringBuilder(s.substring(0, i));

            if(s.length() % str.length() != 0)  continue;
            String material = str.toString();
            while(str.length() <= s.length()){
                if(!str.toString().equals(s.substring(0, str.length())))   break;
                str.append(material);
                if(str.toString().equals(s))    return true;
            }

        }

        return false;
    }
```

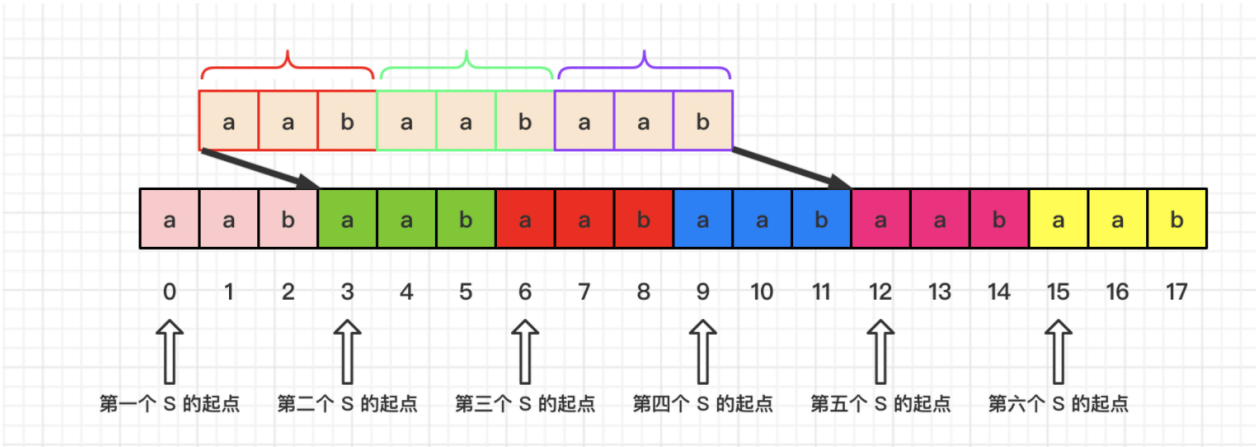
当 s 没有循环节时：

如果 s 中没有循环节，那么 ss 中必然有且只有两个 s，此时从 ss[1] 处开始寻找 s ，必然只能找到**第二个**，所以此时返回值为 s.size()。



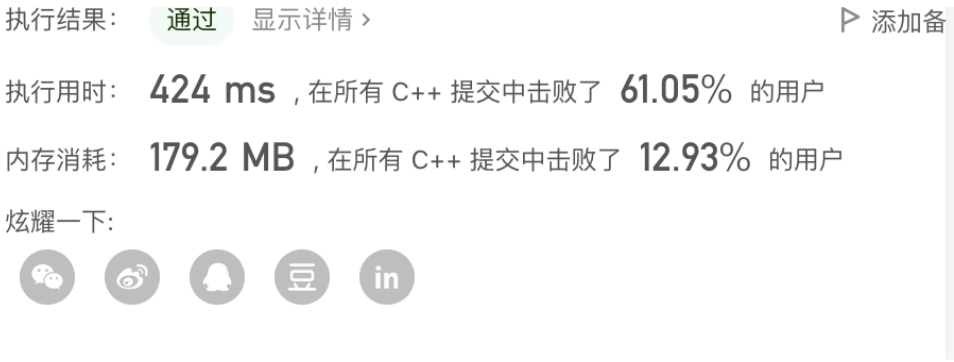
当 s 有循环节时：

当 s 中有循环节时，设循环节为 r，其长度为 l，那么 ss 中必然有 s.size()/l + 1 个 s。
因为去掉了第一个 S 的第一个字符 (代码中，(s+s).find(s, 1)， 是从 ss[1] 处开始 find)
所以此时必回找到第二个 s 的起点。



```java
//牛逼解法
/*

*/
public boolean repeatedSubstringPattern(String s) {
        System.out.println((s+s).indexOf(s, 1));
        return (s+s).indexOf(s, 1) != s.length();
    }

```

460 LFU Cache




```cpp
class Node{
public:
    int key;
    int val;
    Node* prev;
    Node* next;

    Node(int k, int v):key(k), val(v), prev(nullptr), next(nullptr){}
};

class DoublyLinkedList{
public:
    Node* head;
    Node* tail;

    DoublyLinkedList() : head(new Node(0, 0)), tail(new Node(0, 0)){
        head->next = tail;
        tail->prev = head;
    }

    void addFirst(Node* node){
```

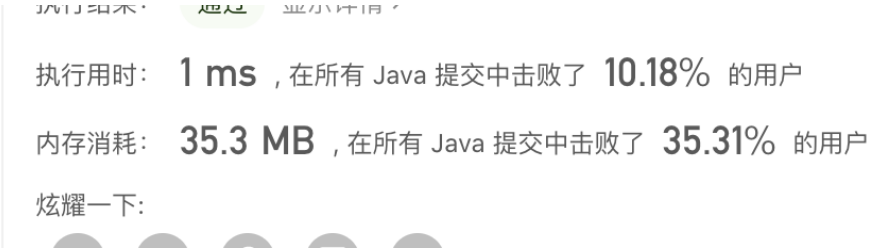
```
        node->next = head->next;
        node->prev = head;

        head->next->prev = node;
        head->next       = node;
    }

    Node* delNode(Node* node){
        node->prev->next = node->next;
        node->next->prev = node->prev;

        return node;
    }

    Node* delLast(){
        if(head->next == tail)
            return nullptr;

        Node* node = delNode(tail->prev);
        return node;
    }

    bool empty(){
        return head->next == tail;
    }
};


class FreqNode{
public:
    int freq;
    FreqNode* prev;
    FreqNode* next;
    int size;
    DoublyLinkedList* dll;

    explicit FreqNode(int f) : freq(f), prev(nullptr), next(nullptr), size(0),
dll(new DoublyLinkedList()){}

};

class FreqDoublyLinkedList{
public:
    FreqNode* head;
    FreqNode* tail;

    FreqDoublyLinkedList():head(new FreqNode(0)), tail(new FreqNode(INT_MAX)){
        head->next = tail;
        tail->prev = head;
```

```
    }

    void addFreqNode(FreqNode* cur){
        if(cur->freq + 1 != cur->next->freq){
            auto* newFreqNode = new FreqNode(cur->freq + 1);
            newFreqNode->next = cur->next;
            newFreqNode->prev = cur;

            cur->next->prev = newFreqNode;
            cur->next = newFreqNode;
        }
    }

    void delCurFreqNode(FreqNode* node){
        node->prev->next = node->next;
        node->next->prev = node->prev;
    }


};

class LFUCache {
public:
    int cap;

    //            key, freq
    unordered_map<int, FreqNode*> freqMap;

    //             key, Node
    unordered_map<int, Node*> nodeMap;
    FreqDoublyLinkedList* fll;
    int minFreq;

    LFUCache(int capacity) {
        this->cap = capacity;
        fll = new FreqDoublyLinkedList();
    }

    int get(int key) {
        if(nodeMap.count(key) == 0)
            return -1;

        Node* targetNode = nodeMap[key];
        FreqNode* before = freqMap[key];
        fll->addFreqNode(before);
        FreqNode* after = before->next;

        //delete old part
        before->dll->delNode(targetNode);
```

```
        if(before->dll->empty())
            fll->delCurFreqNode(before);

        //update the new part
        after->dll->addFirst(targetNode);
        freqMap[key] = after;


        return targetNode->val;
    }

    void put(int key, int value) {
        if(cap == 0)
            return;

        if(nodeMap.count(key) != 0){//met before
            FreqNode* before = freqMap[key];
            fll->addFreqNode(before);
            FreqNode* after = before->next;

            //delete old part
            Node* node = before->dll->delNode(nodeMap[key]);
            node->val = value;
            if(before->dll->empty())
                fll->delCurFreqNode(before);

            //update the new part
            after->dll->addFirst(node);
            freqMap[key] = after;
        }else{
            if(nodeMap.size() == cap){  //del the one
                FreqNode* freqNode = fll->head->next;
                Node* nodeToDel = freqNode->dll->delLast();
                if(freqNode->dll->empty()){
                    fll->delCurFreqNode(freqNode);
                }

                nodeMap.erase(nodeToDel->key);
                freqMap.erase(nodeToDel->key);
            }

            Node* node = new Node(key, value);
            if(fll->head->next->freq != 1){
                fll->addFreqNode(fll->head);
            }

            FreqNode* freqNode = fll->head->next;
            freqNode->dll->addFirst(node);

```

```
            freqMap[key] = freqNode;
            nodeMap[key] = node;
        }
    }
};
```

461 Hamming Distance




```
1
2   public int hammingDistance(int x, int y) {
3       int num = x ^ y;
4
5       int count = 0;
6       for(int i = 0; i < 32; i++){
7           if((num & 1) == 1)
8               count++;
9
10          num >>= 1;
11      }
12
13      return count;
14  }
```

462 Minimum Moves to Equal Array Elements II

462. Minimum Moves to Equal Array Elements II

难度 中等 114

Given a **non-empty** integer array, find the minimum number of mov required to make all array elements equal, where a move is incrementing a selected element by 1 or decrementing a selected element by 1.

You may assume the array's length is at most 10,000.

Example:

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

Output:
2

Explanation:
Only two moves are needed (remember each move
increments or decrements one element):

[1,2,3]  =>  [2,2,3]  =>  [2,2,2]
```

```java
//二刷
public int minMoves2(int[] nums) {
        int n = nums.length;
        Arrays.sort(nums);
        int targetNum = nums[n / 2];
        int res = 0;
        for(int i = 0; i < nums.length; i++){
            res += Math.abs(nums[i] - targetNum);
        }

        return res;
    }
```

```java
/*
    算法的思路是， 对于排好序的两个数，a, b
    最终变成 x
    那么步数是 x - a + b - x == b - a
    x 可以是任何值
    因此如果是奇数个数组， 那x 就等于中间这个值
```

```
    偶数个数组，  那就取平均值
*/
public int minMoves2(int[] nums) {
        Arrays.sort(nums);

        int i = 0, j = nums.length - 1;

        int res = 0;
        while(i < j){
            res += nums[j--] - nums[i++];
        }
        return res;
    }
```

463 Island Parameter

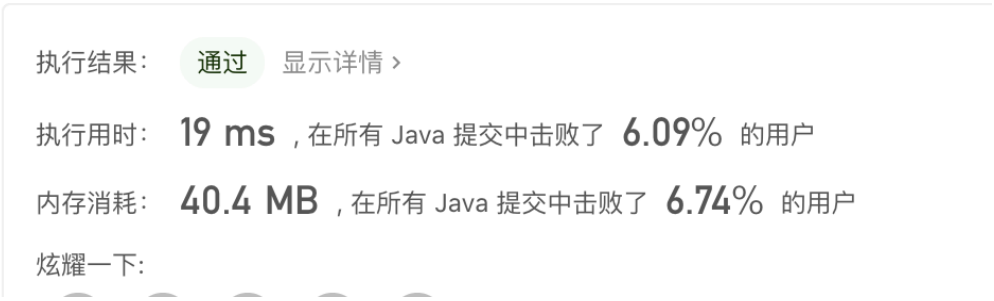
执行结果： 通过 显示详情 >

执行用时： 7 ms , 在所有 Java 提交中击败了 97.29% 的用户

内存消耗： 39.9 MB , 在所有 Java 提交中击败了 37.31% 的用户

```
//二刷
int row;
    int col;

    public int islandPerimeter(int[][] grid) {
        row = grid.length;
        col = grid[0].length;

        int res = 0;
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col;j++){
                if(grid[i][j] == 1){
                    res += 4;

                    if(isInRange(i - 1, j) && grid[i - 1][j] == 1){
                        res -= 2;
                    }

                    if(isInRange(i, j - 1) && grid[i][j - 1] == 1){
                        res -= 2;
```

```
                    }
                }
            }
        }


        return res;
    }

    private boolean isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
```




```
    boolean[][] visited;
    int[][] dir = {{-1, 0},{0, -1},{1, 0},{0, 1}};
    int row;
    int col;
    int res = 0;
    public int islandPerimeter(int[][] grid) {
        row = grid.length;
        col = row == 0 ? 0 : grid[0].length;
        if(col == 0)    return 0;

        visited = new boolean[row][col];
        for(int i = 0; i < row; i++)
            for(int j = 0; j < col; j++){
                if(grid[i][j] == 1){
                    dfs(grid, i, j);
                    return res;
                }
            }

        return 0;
    }

    private void dfs(int[][] grid, int i, int j) {
        if(!isInRange(i, j) || visited[i][j] || grid[i][j] == 0)   return;
```

```
25
26          for(int k = 0; k < 4; k++){
27              int newX = i + dir[k][0];
28              int newY = j + dir[k][1];
29
30              if((!isInRange(newX, newY)) || (isInRange(newX, newY) && !visited[newX]
[newY])){
31                  if((isInRange(newX, newY) && grid[newX][newY] == 1)) continue;
32
33                  res++;
34              }
35          }
36
37          visited[i][j] = true;
38
39          for(int k = 0; k < 4; k++){
40              int newX = i + dir[k][0];
41              int newY = j + dir[k][1];
42
43              dfs(grid, newX, newY);
44          }
45      }
46
47      public boolean isInRange(int i, int j){
48          return i >= 0 && j >= 0 && i < row && j < col;
49      }
50
```

```
1       boolean[][] visited;
2       int[][] dir = {{-1, 0},{0, -1},{1, 0},{0, 1}};
3       int row;
4       int col;
5       int res = 0;
6       public int islandPerimeter(int[][] grid) {
7           row = grid.length;
8           col = row == 0 ? 0 : grid[0].length;
9           if(col == 0)    return 0;
10          visited = new boolean[row][col];
11
12          for(int i = 0; i < row; i++){
13              for(int j = 0; j < col; j++){
14                  visited[i][j] = true;
15                  if(grid[i][j] == 0) continue;
16
17                  for(int k = 0; k < 4; k++){
18                      int newX = i + dir[k][0];
19                      int newY = j + dir[k][1];
```

```
                    if(isInRange(newX, newY) && grid[newX][newY] == 0)
                        res++;
                    else if(!isInRange(newX, newY))
                        res++;
                }
            }

        }

        return res;
    }

    public boolean isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
```

464 Can I win 回溯不错的题目

```
//二刷， 挺恶心的一道题
Map<String, Boolean> map = new HashMap<>();
    public boolean canIWin(int maxChoosableInteger, int desiredTotal) {
        if((1 + maxChoosableInteger) * maxChoosableInteger / 2 < desiredTotal)
            return false;

        char[] state = new char[maxChoosableInteger + 1];
        Arrays.fill(state, '0');
        return backtrack(maxChoosableInteger, desiredTotal, state);
    }

    private boolean backtrack(int maxChoosableInteger, int desiredTotal, char[]
state){
        if(desiredTotal < 0)
            return false;
        String key = new String(state);
        if(map.containsKey(key))
            return map.get(key);
```

```
18
19          for(int i = 1; i <= maxChoosableInteger; i++){
20              if(state[i] == '1')
21                  continue;
22
23              state[i] = '1';
24
25              if(desiredTotal - i <= 0 || !backtrack(maxChoosableInteger,
    desiredTotal - i, state)){
26                  map.put(key, true);
27                  state[i] = '0';
28                  return true;
29              }
30
31              state[i] = '0';
32          }
33
34          map.put(key, false);
35          return false;
36      }
```

```
1   /*
2       不错的题目,
3
4       注意采用 state 数组来记忆我们的状态
5       同时关注,  回溯采用 map 防止重复计算
6   */
7   HashMap<String, Boolean> map = new HashMap<>();
8     public boolean canIWin(int maxChoosableInteger, int desiredTotal) {
9         if ((1 + maxChoosableInteger) * maxChoosableInteger / 2 < desiredTotal)
10              return false;
11          int[] state = new int[maxChoosableInteger + 1];
12          return backtrack(maxChoosableInteger, desiredTotal, state);
13      }
14
15      private boolean backtrack(int maxChoosableInteger, int desiredTotal, int[]
    state) {
16  //          String key = state.toString();
17          String key = Arrays.toString(state);
18          if(map.containsKey(key))        return map.get(key);
19
20          for(int i = 1; i <= maxChoosableInteger; i++){
```

```
            if(state[i] == 1)       continue;

            state[i] = 1;

            if(desiredTotal - i <= 0 || !backtrack(maxChoosableInteger,
desiredTotal - i, state)){
                map.put(key, true);
                state[i] = 0;
                return map.get(key);
            }

            state[i] = 0;
        }

        map.put(key, false);
        return false;
    }
```

465 Optimal Account Balancing

```
//reference: https://www.bilibili.com/video/BV1tE411q7CN?
from=search&seid=15325093727416594085

//暴力回溯
public int minTransfers(int[][] transactions) {
        Map<Integer, Integer> map = new HashMap<>();
        for(int[] t : transactions){
            map.put(t[0], map.getOrDefault(t[0], 0) + t[2]);
            map.put(t[1], map.getOrDefault(t[1], 0) - t[2]);
        }

        List<Integer> list = new ArrayList<>();
        for(int v : map.values()){
            if(v != 0){
                list.add(v);
            }
        }

        return backtrack(0, list);
    }

    private int backtrack(int k, List<Integer> list){
        if(k == list.size())
```

```
            return 0;

        int cur = list.get(k);
        if(cur == 0)
            return backtrack(k + 1, list);

        int min = Integer.MAX_VALUE;
        for(int i = k + 1; i < list.size(); i++){
            int next = list.get(i);
            if(cur * next < 0){
                list.set(i, cur + next);
                min = Math.min(min, 1 + backtrack(k + 1, list));
                list.set(i, next);

                if(cur + next == 0)
                    break;
            }
        }

        return min;
    }
```

466 Count The Repetitions

执行用时： 2701 ms , 在所有 Java 提交中击败了 5.45% 的用户

内存消耗： 36.5 MB , 在所有 Java 提交中击败了 42.73% 的用户

炫耀一下:

```
//继续优化，通过
public int getMaxRepetitions(String s1, int n1, String s2, int n2) {
        int index1 = 0;
        int index2 = 0;
        int len1 = s1.length();
        int len2 = s2.length();
        int res = 0;
        int oldN1 = n1;
        while(n1 != 0){
            while(index1 < len1 && index2 < len2 &&
                  s1.charAt(index1) != s2.charAt(index2)){
                index1++;
            }
```

```
            if(index1 == len1 || index2 == len2) {
                if(index1 == len1 && index2 == len2){
                    return ((res + 1) * oldN1) / ((oldN1 - n1 + 1) * n2);
                }

                if(index1 == len1){
                    index1 = 0;
                    n1--;
                }

                if(index2 == len2){
                    res++;
                    index2 = 0;
                }
            }else {
                index1++;
                index2++;
            }
        }

        return res / n2;
    }


```

```
//超时 37 / 49
public int getMaxRepetitions(String s1, int n1, String s2, int n2) {
        int index1 = 0;
        int index2 = 0;
        int len1 = s1.length();
        int len2 = s2.length();
        int res = 0;

        while(n1 != 0){
            while(index1 < len1 && index2 < len2 && s1.charAt(index1) !=
s2.charAt(index2)){
                index1++;
            }
            if(index1 == len1 || index2 == len2) {
                if(index1 == len1){
                    index1 = 0;
                    n1--;
                }

                if(index2 == len2){
                    res++;
                    index2 = 0;
```

```
                }
            }else {
                index1++;
                index2++;
            }
        }

        return res / n2;
    }


```

467 Unique Substrings in Wraparound String

467. Unique Substrings in Wraparound String

难度 中等 147

We define the string `s` to be the infinite wraparound string of `"abcdefghijklmnopqrstuvwxyz"`, so `s` will look like this:

- `"...zabcdefghijklmnopqrstuvwxyzabcdefghijklmn`

Given a string `p`, return *the number of* ***unique non-empty substrings*** *of* `p` *are present in* `s`.

Example 1:

```
Input: p = "a"
Output: 1
Explanation: Only the substring "a" of p is in s.
```

Example 2:

```
Input: p = "cac"
Output: 2
Explanation: There are two substrings ("a", "c")
of p in s.
```

Example 3:

```
Input: p = "zab"
Output: 6
Explanation: There are six substrings ("z", "a",
```

```java
public int findSubstringInWraproundString(String p) {
    int[] dp = new int[26];
    int count = 0;
    char[] chars = p.toCharArray();
    for(int i = 0 ; i < p.length(); i++){
        if(i > 0 && (chars[i] - chars[i - 1] - 1) % 26 == 0){
            count++;
        }else{
            count = 1;
        }

        dp[chars[i] - 'a'] = Math.max(dp[chars[i] - 'a'], count);
    }

    int res = 0;
```

```
        for(int i = 0;  i< 26; i++){
            res += dp[i];
        }

        return res;
    }
```

468 Validate IP Address

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 98.65% 的用户

内存消耗: 36.5 MB , 在所有 Java 提交中击败了 89.10% 的用户

炫耀一下:

```
//二刷
public String validIPAddress(String IP) {
        if(isValidIPv4(IP)){
            return "IPv4";
        }else if(isValidIPv6(IP)){
            return "IPv6";
        }else{
            return "Neither";
        }
    }

    public boolean isValidIPv4(String IP){
        String[] splits = IP.split("\\.");
        if(splits.length != 4){
            return false;
        }


        for(int i = 0; i < splits.length; i++){
            if((splits[i].length() > 1 && splits[i].charAt(0) == '0') ||
splits[i].length() == 0)
                return false;
            try{
                int curNum = Integer.parseInt(splits[i]);
                if(curNum < 0 || curNum > 255)
```

```
                    return false;
            }catch (Exception e){
                return false;
            }


        }

        int counter = 0;
        for(int i = 0; i < IP.length(); i++)
            counter += IP.charAt(i) == '.' ? 1 : 0;
        return counter == 3;
    }

    public boolean isValidIPv6(String IP){
        String[] splits = IP.split(":");
        if(splits.length != 8)
            return false;


        for(int i = 0; i < splits.length; i++){
            String cur = splits[i];
            if(cur.length() > 4 || cur.length() == 0)
                return false;

            for(int j = 0; j < cur.length(); j++){
                if(!validIP6Symbol(cur.charAt(j)))
                    return false;
            }

        }


        int counter = 0;
        for(int i = 0; i < IP.length(); i++)
            counter += IP.charAt(i) == ':' ? 1 : 0;

        return counter == 7;
    }

    private boolean validIP6Symbol(char ch){
        return (ch >= '0' && ch <= '9') || (ch >= 'a' && ch <= 'f') || (ch >= 'A'
&& ch <= 'F');
    }


```

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 98.68% 的用户

内存消耗： 36.6 MB ，在所有 Java 提交中击败了 68.64% 的用户

炫耀一下:

```java
    public String validIPAddress(String IP) {
        if(IP.length() == 0)        return "Neither";
        if(IP.charAt(IP.length() - 1) == ':' || IP.charAt(IP.length() - 1) == '.')
            return "Neither";

        String[] ip4 = IP.split("\\.");
        String[] ip6 = IP.split(":");

        if(ip4.length == 4){
            return getIp4(ip4);
        }else if(ip6.length == 8)
            return getIp6(ip6);

        return "Neither";
    }

   private String getIp6(String[] ip6) {
        int[] allZero = new int[ip6.length];
        for(int i = 0; i < ip6.length; i++){
            if(ip6[i].length() == 0 || ip6[i].length() > 4)       return "Neither";

            int count = 0;
            for(int j = 0; j < ip6[i].length(); j++){
                char ch = ip6[i].charAt(j);
                if(ch == '0') count++;

                if(('A' <= ch && ch <= 'F') ||
                        ('a' <= ch && ch <= 'f') ||
                        ('0' <= ch && ch <= '9')){ }
                else{
                    return "Neither";
                }
            }
            if(count == ip6[i].length())
                allZero[i] = 1;
            if(count == ip6[i].length() && count != 1 && i >= 1 && allZero[i - 1]
== 1)   return "Neither";
        }

        return "IPv6";
    }

```

```java
    private String getIp4(String[] ip4) {
        for(String str : ip4){
            if(str.length() == 0)       return "Neither";

            if(str.charAt(0) == '0' && str.length() != 1)
                return "Neither";
            boolean isIp4 = true;

            try{
                int num = Integer.parseInt(str);
                if(num < 0 || num > 255)
                    isIp4 = false;
            }catch(Exception e){
                isIp4 = false;
            }

            if(!isIp4)        return "Neither";
        }

        return "IPv4";
    }
```

469 Coven Polygon

Given a list of points that form a polygon when joined sequentially, find if this polygon is convex (Convex polygon definition).

Note:

1. There are at least 3 and at most 10,000 points.
2. Coordinates are in the range -10,000 to 10,000.
3. You may assume the polygon formed by given points is always a simple polygon (Simple polygon definition). In other words, we ensure that exactly two edges intersect at each vertex, and that edges otherwise **don't intersect each other**.

Example 1:

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

Answer: True

Explanation:
```

Example 2:

```
[[0,0],[0,10],[10,10],[10,0],[5,5]]

Answer: False

Explanation:
```

```java
/*
  凸变形性质，  连续边叉乘的结果符号一致
*/
public boolean isConvex(List<List<Integer>> points) {
        int n = points.size();
        long pre = 0;
        long cur = 0;

        for(int i = 0; i < n; i++){
            int dx1 = points.get((i + 1) % n).get(0) - points.get(i).get(0);
            int dx2 = points.get((i + 2) % n).get(0) - points.get(i).get(0);
            int dy1 = points.get((i + 1) % n).get(1) - points.get(i).get(1);
            int dy2 = points.get((i + 2) % n).get(1) - points.get(i).get(1);

            cur = dx1 * dy2 - dx2 * dy1;
            if(cur != 0){
                if(cur * pre < 0)
                    return false;
                else
                    pre = cur;
            }
        }

        return true;
    }
```

470 Implement rand 10 using ran7

```java
/*
    思路分析：
      如果给的是random 10，  想要生成 rand 7，  十分好办，只要调用 rand 10 ，  如果不是我们想要的数字，就一直调用
      这个结果是等概率的，证明如下
*/
```

直到得到我们要的数，但是为什么可以呢？你可能会怀疑这个是不是等概率的，我们来计算一下

- 如果第一次就 $rand$ 到 $1\sim7$ 之间的数，那就是直接命中了，概率为 $\frac{1}{10}$
- 如果第二次命中,那么第一次必定没命中，没命中的概率为 $\frac{3}{10}$ ，再乘命中的概率 $\frac{1}{10}$ ，所以第二次命中的概率是 $\frac{1}{10}*\frac{3}{10}$

依次类推，我们求和，可以得到如下结果，可以知道，从 $rand10()$ 到 $rand7()$ 它是等概率的

$$
\begin{aligned}
&\frac{1}{10}+\frac{3}{10}*\frac{1}{10}+\left(\frac{3}{10}\right)^2*\frac{1}{10}+\left(\frac{3}{10}\right)^3*\frac{1}{10}+\cdots+\left(\frac{3}{10}\right)^{n-1}*\frac{1}{10}\\
&=\frac{1}{10}*\left(1+\frac{3}{10}+\left(\frac{3}{10}\right)^2+\left(\frac{3}{10}\right)^3+\ldots+\left(\frac{3}{10}\right)^{n-1}\right)\\
&=\frac{1}{10}*\left(1+\frac{\frac{3}{10}*\left(1-\left(\frac{3}{10}\right)^n\right)}{1-\frac{3}{10}}\right)\quad\text{等比数列求和}\\
&\qquad\qquad\text{n趋向 }+\infty,\ 1-\left(\frac{3}{10}\right)^n\text{趋向0}\\
&=\frac{1}{10}*\left(1+\frac{\frac{3}{10}}{\frac{7}{10}}\right)\\
&=\frac{1}{10}*\frac{10}{7}
\end{aligned}
$$

为什么命中的概率是 1 / 10

个人理解： 因为可以选择的数字是 7 / 10, 选中特定的数字概率 1 / 7,两个乘起来就是 1/ 10

同时关注到这个式子

```
(rand7() - 1) * 7 + rand7(),
可以等概率生成 1 - 42的数字
  因为 rand7() - 1能拿到  0,1,2,3,4,5,6
  再乘 7 后得到的集合是 {0,7,14,21,28,35,42}


  后面 rand7() 得到的集合B 为 1 , 2,3,4,5,6,7
    对应每个加起来，范围就是 1- > 49



在进一步拓展
(randX() - 1) * Y + randY()
前面集合，可以拿到等概率的
    0,1,2,3,4,...x-1
    0,Y,2Y,3Y,....Y(X-1)

    后面可以拿到
    1,2,3,4...Y
    最终该集合的范围是
    1,2,3,.....X(X-1) + Y

```

```
23
24      最终可以通过不断循环，拿到想要的结果
25      class Solution extends SolBase {
26      public int rand10() {
27          // 首先得到一个数
28          int num = (rand7() - 1) * 7 + rand7();
29          // 只要它还大于10，那就给我不断生成，因为我只要范围在1-10的，最后直接返回就可以了
30          while (num > 10){
31              num = (rand7() - 1) * 7 + rand7();
32          }
33          return num;
34      }
35  }
36
37  作者：jerry_nju
38  链接：https://leetcode-cn.com/problems/implement-rand10-using-rand7/solution/xiang-
    xi-si-lu-ji-you-hua-si-lu-fen-xi-zhu-xing-ji/
39
40  /*
41      进一步优化
42  */
43  public int rand10() {
44      // 首先得到一个数
45      int num = (rand7() - 1) * 7 + rand7();
46      // 只要它还大于40，那你就给我不断生成吧
47      while (num > 40)
48          num = (rand7() - 1) * 7 + rand7();
49      // 返回结果，+1是为了解决 40%10为0的情况
50      return 1 + num % 10;
51  }
52
```

472 Concatenated Words

Given an array of strings `words` (**without duplicates**), return *all the **concatenated words** in the given list of* `words` .

A **concatenated word** is defined as a string that is comprised entirely of at least two shorter words in the given array.

Example 1:

```
Input: words =
["cat","cats","catsdogcats","dog","dogcatsdog","hippo
Output:
["catsdogcats","dogcatsdog","ratcatdogcat"]
Explanation: "catsdogcats" can be concatenated by
"cats", "dog" and "cats";
"dogcatsdog" can be concatenated by "dog", "cats"
and "dog";
"ratcatdogcat" can be concatenated by "rat",
"cat", "dog" and "cat".
```

Example 2:

```
Input: words = ["cat","dog","catdog"]
Output: ["catdog"]
```

```
//reference: https://www.youtube.com/watch?v=hoGGwExHnnQ
/*
    多说一点对于 DP 的优化
      原始思路
      for i : [0, n){   O(n^2 m)
        if(!dp[i])
          continue;

        for j : [i + 1, n]
          if(s.substring(i, j ) in set)   //这里计算了太多次
            dp[j]  = true;
      }

      优化
      for i : [1, n]
        for(j : [0, i)
          if(!dp[j])
            continue;
          if(s.substring(j, i) in set)
```

```
            dp[i] = true;
            break;
*/
public List<String> findAllConcatenatedWordsInADict(String[] words){
  List<String> res = new ArrayList<>();
  Set<String> set  = new HashSet<>();

  for(String s : words){
    set.add(s);
  }

  for(String s : words){
    set.remove(s);

    if(canBreak(s, set)){
      res.add(s);
    }

    set.add(s);
  }

  return res;
}

private boolean canBreak(String s, Set<String> set){
  if(set.size() == 0)
      return false;

  int n = s.length();
  if(n == 0)
    return flase;

  boolean[] dp = new boolean[n + 1];
  dp[0] = true;

  for(int i = 1; i <= n; i++){
    for(int j = 0; j < i; j++){
      if(!dp[j])
        continue;

      if(set.contains(s.substring(j, i))){
        dp[i] = true;
        break;
      }
    }
  }

  return dp[n];
}
```

```java
//超出时间限制 44 / 45
public List<String> findAllConcatenatedWordsInADict(String[] words) {
        Set<String> set = new HashSet<>();
        for(String word : words)
            set.add(word);

        List<String> res = new ArrayList<>();
        for(String word : words){
            if(word.equals(""))
                continue;
            set.remove(word);
            if(wordBreak(word, set)){
                res.add(word);
            }
            set.add(word);
        }

        return res;
    }

    private boolean wordBreak(String word, Set<String> set) {
        if(word.length() == 0)
            return true;

        for(int i = 1; i <= word.length(); i++){
            String frac = word.substring(0, i);
            if(set.contains(frac)){
                if(wordBreak(word.substring(i), set))
                    return true;
            }
        }

        return false;
    }
```

473 Matchsticks to Square

473. Matchsticks to Square

难度 中等 156

Remember the story of Little Match Girl? By now, you know exactly what matchsticks the little match girl has, please find out a way you can make one square by using up all those matchsticks. You should not break any stick, but you can link them up, and each matchstick must be used **exactly** one time.

Your input will be several matchsticks the girl has, represented with their stick length. Your output will either be true or false, to represent whether you could make one square using all the matchsticks the litt match girl has.

Example 1:

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

```java
//二刷 超时间 170 / 173
    public boolean makesquare(int[] matchsticks) {
        int sum = 0;
        for(int num : matchsticks)
            sum += num;
        if(sum % 4 != 0)
            return false;

        int target =  sum / 4;
        Arrays.sort(matchsticks);
        reverse(matchsticks);

        int[] sums = new int[4];
        return canBeSquare(sums, matchsticks, new boolean[matchsticks.length], 0, target);
    }

    private void reverse(int[] matchsticks) {
        int left = 0, right = matchsticks.length - 1;
        while(left != right){
            int temp = matchsticks[left];
            matchsticks[left] = matchsticks[right];
            matchsticks[right] = temp;
            left++;
            right--;
        }
    }

```

```
    private boolean canBeSquare(int[] sums, int[] nums, boolean[] used, int k, int target) {
        if(k == 4)
            return true;

        for(int i = 0; i < nums.length; i++){
            if(used[i] || sums[k] + nums[i] > target)
                continue;

            sums[k] += nums[i];
            used[i] = true;

            if(sums[k] == target){
                if(canBeSquare(sums, nums, used, k + 1, target))
                    return true;
            }else{
                if(canBeSquare(sums, nums, used, k, target))
                    return true;
            }

            sums[k] -= nums[i];
            used[i]  = false;
        }

        return false;

    }

```

```
/*
  通过 170 / 174
  最后超时间了

  主要思路就是回溯， 然后采用flag
  最后双判断

  只有flag 都被消耗掉（4 次， 代表四个边）
  并且， used.size == nums.size 代表火柴全部被用到了，才会return true

*/
public boolean makesquare(int[] nums) {
        int quarSum = 0;
        for(int num : nums)
            quarSum += num;
        if(quarSum % 4 != 0)
```

```
            return false;
        quarSum /= 4;

        return backtrack(nums, 4, new HashSet<Integer>(), quarSum, 0);
    }

    private boolean backtrack(int[] nums, int flag, HashSet<Integer> used, int quarSum, int currentSum) {
        if(flag == 0)
            return true;
        if(quarSum == currentSum){
            flag--;
            currentSum = 0;
            if(flag == 0 || used.size() == nums.length)
                return true;
        }
        if(currentSum > quarSum)
            return false;


        for(int i = 0; i < nums.length; i++){
            if(used.contains(i))    continue;

            used.add(i);
            if(backtrack(nums, flag, used, quarSum, currentSum + nums[i])) {
                used.remove(i);
                return true;
            }
            used.remove(i);
        }

        return  false;
    }

```

```
/*
    思路还是回溯，但是效率很好

    优化思路：
      采用 大数在前，小数在后的策略
*/
public boolean makesquare(int[] nums) {
        if(nums == null || nums.length < 4) return false;
        int quarSum = 0;
        for(int num : nums)
```

```
            quarSum += num;

        if(quarSum % 4 != 0)
            return false;

        List<Integer> numList = new ArrayList<>();
        for(int num : nums) numList.add(num);

        numList.sort((o1, o2)->(o2 - o1));

        return backtrack(numList, quarSum, 0, new int[4]);
    }

    private boolean backtrack(List<Integer> nums, int target, int index, int[] sum)
{
        if(index == nums.size()){
            if(sum[0] == target && sum[1] == sum[0] && sum[2] == sum[1])
                return true;
            return false;
        }

        for(int i = 0; i < 4; i ++){
            if(sum[i] + nums.get(index) > target)   continue;

            sum[i] += nums.get(index);
            if(backtrack(nums, target, index + 1, sum))
                return true;

            sum[i] -= nums.get(index);
        }

        return false;
    }
```

474 One and Zero

474. 一和零

难度 中等 312

给你一个二进制字符串数组 `strs` 和两个整数 `m` 和 `n` 。

请你找出并返回 `strs` 的最大子集的大小，该子集中 **最多** 有 `m` 个 `0` 和 `n` 个 `1` 。

如果 `x` 的所有元素也是 `y` 的元素，集合 `x` 是集合 `y` 的 **子集** 。

示例 1：

> **输入：** strs = ["10", "0001", "111001", "1", "0"], m = 5, n = 3
> **输出：** 4
> **解释：** 最多有 5 个 0 和 3 个 1 的最大子集是 {"10","0001","1","0"} ，因此答案是 4 。
> 其他满足题意但较小的子集包括 {"0001","1"} 和 {"10","1","0"} 。{"111001"} 不满足题意，因为它含 4 个 1 ，大于 n 的值 3 。

示例 2：

> **输入：** strs = ["10", "0", "1"], m = 1, n = 1
> **输出：** 2
> **解释：** 最大的子集是 {"0", "1"} ，所以答案是 2 。

提示：

```
1
1
1
1
1
1
1
1
1
1
2
2
2
2
2
2
2
2
2
2
2
3
3
```

```java
/*
  @Author: WenchaoGuo


  dp[i][j][k]
    代表 前 i个物品,  能表示 j 个 0,  k 个 1的最大子集是多少?
*/
class Solution {
    public int findMaxForm(String[] strs, int m, int n) {
        int len = strs.length;

        int[][][] dp = new int[len + 1][m + 1][n + 1];

```

```
        for(int i = 0; i < len + 1; i++){
            if(i == 0)      continue;

            int[] cnt = countZeroAndOne(strs[i - 1]);
            for(int j = 0; j < m + 1; j++)
                for(int k = 0; k < n + 1; k++){
                    if(j >= cnt[0] && k >= cnt[1])
                        dp[i][j][k] = Math.max(dp[i - 1][j][k], dp[i - 1][j - cnt[0]][k - cnt[1]] + 1);
                    else
                        dp[i][j][k] = dp[i - 1][j][k];
                }
        }

        return dp[len][m][n];
    }

    private int[] countZeroAndOne(String str){
        int[] cnt = new int[2];
        for(char ch : str.toCharArray())
            cnt[ch -'0']++;

        return cnt;
    }
}
```

```
public class Solution {

    private int[] countZeroAndOne(String str) {
        int[] cnt = new int[2];
        for (char c : str.toCharArray()) {
            cnt[c - '0']++;
        }
        return cnt;
    }

    public int findMaxForm(String[] strs, int m, int n) {
        int len = strs.length;
        int[][][] dp = new int[len + 1][m + 1][n + 1];

        for (int i = 1; i <= len; i++) {
            // 注意：有一位偏移
            int[] cnt = countZeroAndOne(strs[i - 1]);
            for (int j = 0; j <= m; j++) {
```

```
                for (int k = 0; k <= n; k++) {
                    // 先把上一行抄下来
                    dp[i][j][k] = dp[i - 1][j][k];

                    int zeros = cnt[0];
                    int ones = cnt[1];

                    if (j >= zeros && k >= ones) {
                        dp[i][j][k] = Math.max(dp[i - 1][j][k], dp[i - 1][j - zeros][k - ones] + 1);
                    }
                }
            }
        }
        return dp[len][m][n];
    }
}
```

作者：liweiwei1419
链接：https://leetcode-cn.com/problems/ones-and-zeroes/solution/dong-tai-gui-hua-zhuan-huan-wei-0-1-bei-bao-wen-ti/
来源：力扣（LeetCode）

475 Heater

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 89.71% 的用户

内存消耗： 40.9 MB ，在所有 Java 提交中击败了 97.54% 的用户

```
//二刷， 时间复杂度 O(nlgn)
public int findRadius(int[] houses, int[] heaters) {
        /*
            1) find all heaters that are righthand-side place

            2) find all heaters that are lefthadn-side place

            3) get the min Redius
        */

        int len1 = houses.length;
        int len2 = heaters.length;

```

```
        Arrays.sort(houses);
        Arrays.sort(heaters);

        int index1 = 0;
        int index2 = 0;
        int[] left  = new int[len1];
        Arrays.fill(left, Integer.MAX_VALUE);
        int[] right = new int[len1];
        Arrays.fill(right, Integer.MIN_VALUE);

        while(index1 < len1 && index2 < len2){
            if(index2 < len2 && houses[index1] <= heaters[index2]){
                right[index1] = heaters[index2];
                index1++;
            }else{
                index2++;
            }
        }

        index1 = len1 - 1;
        index2 = len2 - 1;

        while(index1 >= 0 && index2 >= 0){
            if(index2 >= 0 && houses[index1] >= heaters[index2]){
                left[index1] = heaters[index2];
                index1--;
            }else{
                index2--;
            }
        }

        int res = Integer.MIN_VALUE;
        for(int i = 0;  i < len1; i++){
            int temp1 = Integer.MAX_VALUE;
            int temp2 = Integer.MAX_VALUE;
            if(left[i] != Integer.MAX_VALUE)
                temp1 = Math.abs(left[i] - houses[i]);

            if(right[i] != Integer.MIN_VALUE)
                temp2 = Math.abs(right[i] - houses[i]);

            if(temp1 == Integer.MAX_VALUE || temp2 == Integer.MAX_VALUE){
                if(temp1 == Integer.MAX_VALUE)
                    res = Math.max(res, temp2);
                else
                    res = Math.max(res, temp1);
            }else
                res = Math.max(res, Math.min(temp1, temp2));
        }
```

```
        return res;
    }
```

```
/*
    超时 10 / 30
*/
public int findRadius(int[] houses, int[] heaters) {
        HashSet<Integer> houseSet = new HashSet<>();
        HashSet<Integer> visited = new HashSet<>();

        for(int num : houses)  houseSet.add(num);

        int radius = 0;

        while(visited.size() != houseSet.size()){
          for(int num : heaters){
              if(houseSet.contains(num + radius))
                  visited.add(num + radius);
              if(houseSet.contains(num - radius))
                  visited.add(num - radius);
          }

          if(visited.size() == houseSet.size())
              return radius;

          radius++;
        }

        return radius;
    }
```

17 分钟前	通过	1405 ms	42 MB

```
    public int findRadius(int[] houses, int[] heaters) {
        Arrays.sort(houses);
        Arrays.sort(heaters);

        int radius = 0;

```

```
        for(int i = 0; i < houses.length; i++){
            int curPos = houses[i];
            int radiusI = Integer.MAX_VALUE;

          //对于当前房间来说，我们能够被加热的最短距离
            for(int j = 0; j < heaters.length; j++){
               radiusI = Math.min(radiusI, Math.abs(curPos - heaters[j]));
            }

            radius = Math.max(radiusI, radius);
        }

        return radius;
    }

```

```
class Solution {
public:
/*
Example:    h = house,  * = heater  M = INT_MAX

        h   h   h   h   h   h   h   h   h    houses
        1   2   3   4   5   6   7   8   9    index
        *           *       *                heaters

        0   2   1   0   1   0   -   -   -    (distance to nearest RHS heater)
        0   1   2   0   1   0   1   2   3    (distance to nearest LHS heater)

        0   1   1   0   1   0   1   2   3    (res = minimum of above two)

Result is maximum value in res, which is 3.
*/
    int findRadius(vector<int>& A, vector<int>& H) {
        sort(A.begin(), A.end());
        sort(H.begin(), H.end());
        vector<int> res(A.size(), INT_MAX);

        // For each house, calculate distance to nearest RHS heater
        for (int i = 0, h = 0; i < A.size() && h < H.size(); ) {
            if (A[i] <= H[h]) { res[i] = H[h] - A[i]; i++; }
            else { h++; }
        }

        // For each house, calculate distance to nearest LHS heater
        for (int i = A.size()-1, h = H.size()-1; i >= 0 && h >= 0; ) {
```

```
            if (A[i] >= H[h]) { res[i] = min(res[i], A[i] - H[h]); i--; }
            else { h--; }
        }

        return *max_element(res.begin(), res.end());
    }
};

https://leetcode.com/problems/heaters/discuss/95887/C%2B%2B-clean-solution-with-explanation
```

郭郭版本 有关于上一个的

```
    public int findRadius(int[] houses, int[] heaters) {
        Arrays.sort(heaters);
        Arrays.sort(houses);

        int[] res = new int[houses.length];
        Arrays.fill(res, Integer.MAX_VALUE);
        /*
        [1,5]
            [2]
        */
        //find Right Near Heater
        for(int i = 0, h = 0; i < houses.length && h < heaters.length;){
            if(houses[i] <= heaters[h]){
                res[i] = heaters[h] - houses[i];
                i++;
            }else
                h++;
        }

        for(int i = houses.length - 1, h = heaters.length - 1; i >= 0 && h >= 0;){
            if(houses[i] >= heaters[h]){
                res[i] = Math.min(houses[i] - heaters[h], res[i]);
                i--;
            }else
                h--;
        }

        int minDis = 0;
        for(int num : res)
            minDis = Math.max(minDis, num);
        return minDis;
    }
```

476 number-complement

```java
//二刷
public int bitwiseComplement(int N) {
        if(N == 0)
            return 1;
        String binary = getBinary(N);
        String invertedBinary = getInverted(binary);

        int res = 0;
        for(int i = 0; i < invertedBinary.length(); i++){
            res <<= 1;
            res += invertedBinary.charAt(i) - '0';
        }

        return res;
    }

    private String getBinary(int num){
        StringBuilder res = new StringBuilder();

        while(num != 0){
            res.insert(0, (num & 1));
            num >>= 1;
        }

        return res.toString();
    }

    private String getInverted(String str){
        StringBuilder res = new StringBuilder();
        for(int i = 0; i < str.length(); i++){
            char ch = str.charAt(i);
            res.append(ch == '1' ? '0' : '1');
        }

        return res.toString();
    }
```

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 9.18% 的用户

内存消耗： 35.7 MB ，在所有 Java 提交中击败了 5.67% 的用户

炫耀一下：

```
3
4
5
6
7
8
```

```java
public int findComplement(int num) {
    String twosCompliment = getBinary(num);

    return convertBinaryToDecimao(twosCompliment);
}

public int convertBinaryToDecimao(String twosCompliment){
    int res = 0;

    for(int i = 0; i < twosCompliment.length(); i++){
        res <<= 1;
        res += twosCompliment.charAt(i) - '0';
    }

    return res;
}
public String getBinary(int num){
    StringBuilder sb = new StringBuilder();

    for(int i = 0; i < 31; i++){
        sb.insert(0,num & 1);
        num >>= 1;
    }
    String res = sb.toString();
    int i = 0;
    for(; i < sb.length(); i++){
        if(res.charAt(i) != '0')
            break;
    }
    String temp = res.substring(i);
    StringBuilder sb2 = new StringBuilder();
    for(int j = 0; j < temp.length(); j++){
        sb2.append(temp.charAt(j) == '1' ? '0' : '1');
    }

    return sb2.toString();
}
```

477 Total Hamming Distance

执行结果： 通过 显示详情 >

执行用时： 8 ms , 在所有 Java 提交中击败了 77.91% 的用户

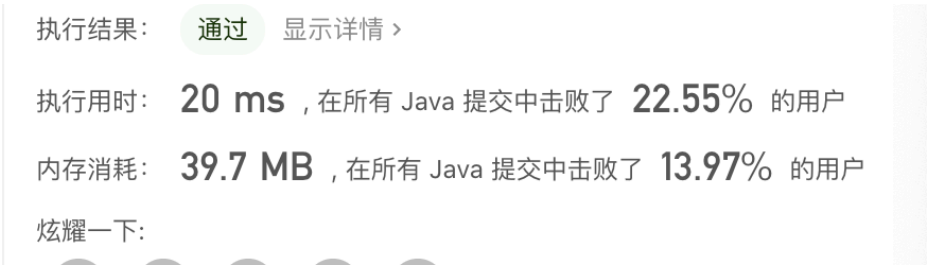
内存消耗： 39.2 MB , 在所有 Java 提交中击败了 92.24% 的用户

炫耀一下:

```java
//二刷
public int totalHammingDistance(int[] nums) {
        int res = 0;
        for(int i = 0;  i < 32; i++){
            int[] cnt = new int[2];

            for(int j = 0; j < nums.length; j++){
                cnt[(nums[j] & 1)]++;
                nums[j] >>= 1;
            }

            if(cnt[0] == nums.length || cnt[1] == nums.length)
                continue;
            res += cnt[0] * cnt[1];
        }

        return res;
    }
```

```java
/*
  超时
*/
public int totalHammingDistance(int[] nums) {
        int total = 0;
        for(int i = 0; i < nums.length; i++){
            for(int j = i + 1; j < nums.length; j++){
                int target = nums[i] ^ nums[j];

                total += getDif(target);
            }
        }

        return total;
    }

    public int getDif(int target){
        int res = 0;
```

```
        for(int i = 0; i < 32; i++){
            if((target & 1) == 1)
                res++;
            target >>= 1;
        }

        return res;
    }
```




```
    public int totalHammingDistance(int[] nums) {
        int res = 0;
        for(int i = 0; i < 32; i++){
            int countZero = 0;
            for(int j = 0; j < nums.length; j++){
                countZero += (nums[j] & 1) == 1 ? 0 : 1;
                nums[j] >>= 1;
            }

            res += countZero * (nums.length - countZero);
        }

        return res;
    }
```

478 Generate Random Point In a Circle

478. Generate Random Point in a Circle

Medium 196 307 Add to List Share

Given the radius and x-y positions of the center of a circle, write a function `randPoint` which generates a uniform random point in the circle.

Note:

1. input and output values are in floating-point.
2. radius and x-y position of the center of the circle is passed into the class constructor.
3. a point on the circumference of the circle is considered to be in the circle.
4. `randPoint` returns a size 2 array containing x-position and y-position of the random point, in that order.

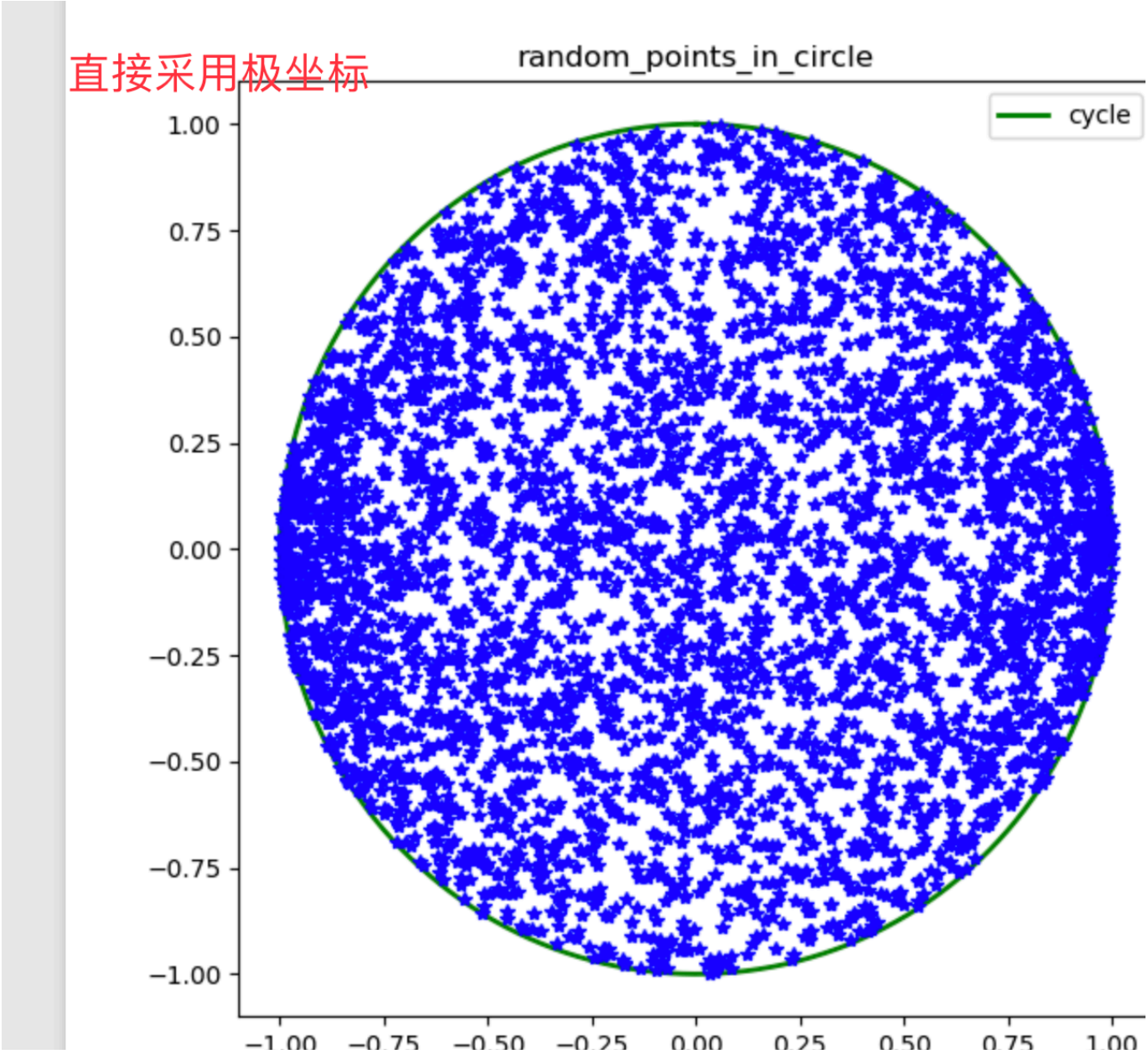
Example 1:

```
Input:
["Solution","randPoint","randPoint","randPoint"]
[[1,0,0],[],[],[]]
Output: [null,[-0.72939,-0.65505],[-0.78502,-0.28626],
[-0.83119,-0.19803]]
```

第一个思路就是，随机产生 x的坐标， 然后y的坐标

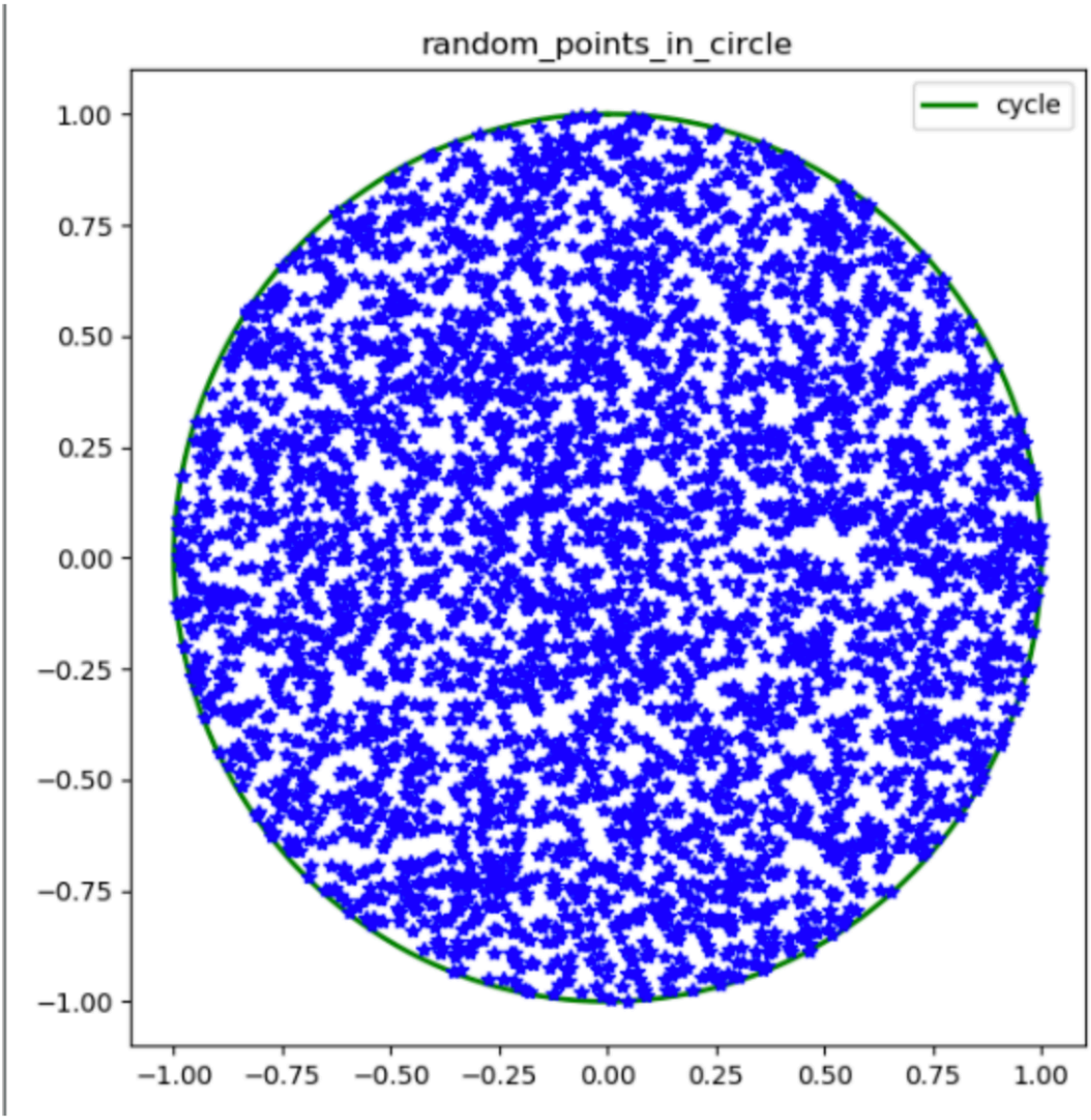
然后生成$[-\sqrt{R^2 - X^2}, \sqrt{R^2 - X^2}]$范

但是两端分布不均匀



第二个思路就是使用面积这个概念，在外接正方形生成点，如果在圆内，保留，不在的话就重新采样(拒绝采样技术)

效果很OK:




```cpp
class Solution {
public:
    double x, y, r;
    Solution(double radius, double x_center, double y_center) {
        r = radius;
        x = x_center;
        y = y_center;
    }

    vector<double> randPoint() {
        double tx = 2.0;
        double ty = 2.0;
        while (tx * tx + ty * ty > 1) {
            tx = 2 * (double)rand() / RAND_MAX - 1;
            ty = 2 * (double)rand() / RAND_MAX - 1;
        }
        return {x + tx * r, y + ty * r};
```

```
    }
};

作者：da-li-wang
链接：https://leetcode-cn.com/problems/generate-random-point-in-a-circle/solution/c-fang-xing-ju-jue-cai-yang-by-da-li-wang/
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。
```

第三个思路是采用 x = R cos(θ) y = Rsin(θ)

随机生成 0 -> 2pai θ, 然后再随机生成R

组成x y

```
/*
    纯数学题，使用极坐标系完成
*/
class Solution {
    double radius, x_center, y_center;
    public Solution(double radius, double x_center, double y_center) {
        this.radius=radius;
        this.x_center=x_center;
        this.y_center=y_center;
    }

    public double[] randPoint() {
        double len= Math.sqrt(Math.random())*radius;
        double deg= Math.random()*2*Math.PI;
        double x= x_center+len*Math.cos(deg);
        double y= y_center+len*Math.sin(deg);
        return new double[]{x,y};
    }
}

https://www.cnblogs.com/yunlambert/p/10161339.html
https://leetcode.com/problems/generate-random-point-in-a-circle/discuss/154037/Polar-Coordinates-10-lines
```

479 Largest Palindrome Product

```java
    public int largestPalindrome(int n) {
        if (n == 1)
            return 9;

        int upperBound = (int) Math.pow(10, n) - 1, lowerBound = upperBound / 10 + 1;
        boolean foundPalindrome = false;
        long maxNumber = (long) upperBound * upperBound;
        long half      = (long) (maxNumber / Math.pow(10, n));
        long palindrome = 0;
        while(!foundPalindrome){
            palindrome = createMaxPalindrome(half);
            for(long i = upperBound; i >= lowerBound; i--){
                if(i * i < palindrome)
                    break;

                if(palindrome % i == 0) {
                    foundPalindrome = true;
                    break;
                }
            }

            half--;
        }

        return (int)(palindrome % 1337);
    }


    private long createMaxPalindrome(long num) {
        String str = num + new StringBuilder().append(num).reverse().toString();

        return Long.parseLong(str);
    }

https://www.youtube.com/watch?v=AKVEXDVqvvI
```

480 Sliding Window Median

执行结果: 通过 显示详情 >

执行用时: 59 ms , 在所有 Java 提交中击败了 68.64% 的用户

内存消耗: 40.4 MB , 在所有 Java 提交中击败了 78.31% 的用户

炫耀一下:

```java
/*
    思路 插入排序

    时间复杂度 O(Nk)
*/
public double[] medianSlidingWindow(int[] nums, int k) {
        double[] res = new double[nums.length - k + 1];
        long[] arr = new long[k];

        int index = 0;
        for(int i = 0; i < nums.length; i++){
            if(i >= k - 1){
                adjust(arr, nums, i, k);

                int midOdd  = (0 + arr.length) / 2;
                int midEven = (0 + arr.length - 1) / 2;
                res[index++] = ((double)(arr[midOdd] + arr[midEven])) / 2;
            }else{
                arr[i] = nums[i];
                if(i == k - 2){
                    Arrays.sort(arr, 0, k - 1);
                }
            }
        }

        return res;
    }

    private void adjust(long[] arr, int[] nums, int i, int k) {
        int pos       = arr.length - 1;
        boolean found = false;
        if(i == k - 1){
            arr[pos] = nums[k - 1];
            found    = true;
        }else{
            for(int j = 0; j < arr.length; j++){
```

```
                if( arr[j] == nums[i - k]){
                    arr[j] = nums[i];
                    pos    = j;
                    found = true;
                    break;
                }
            }
        }
        assert(found);

        while(pos >= 1 && arr[pos] < arr[pos - 1]){
            exch(arr, pos, pos - 1);
            pos--;
        }

        while(pos < arr.length - 1 && arr[pos] > arr[pos + 1]){
            exch(arr, pos, pos + 1);
            pos++;
        }

    }

    private void exch(long[] arr, int x, int y) {
        long temp = arr[x];
        arr[x]   = arr[y];
        arr[y]   = temp;
    }


```

481 Magical String

```
//二刷
public int magicalString(int n) {
        StringBuilder up   = new StringBuilder();
        StringBuilder down = new StringBuilder();
        up.  append("1221121");
        down.append("12211");

        int index = 6;
        int indexDown = 5;
        int res = 0;
```

```
        while(index < n && up.length() < n){
            char ch = up.charAt(indexDown);
            char pre = up.charAt(index);
            for(int i = 0; i < ch - '0'; i++){
                up.append(pre == '1' ? '2' : '1');
            }

            down.append(up.charAt(indexDown++));
            index += ch - '0';
        }

        index = 0;
        while(index < n){
            if(up.charAt(index) == '1')
                res++;

            index++;
        }

        return res;
    }

```

执行结果： 通过 显示详情 >

执行用时： 20 ms , 在所有 Java 提交中击败了 40.53% 的用户

内存消耗： 37.8 MB , 在所有 Java 提交中击败了 37.88% 的用户

炫耀一下:

```
public int magicalString(int n) {
        StringBuilder up    = new StringBuilder("12211");
        StringBuilder below = new StringBuilder("122");

        if(n >  5){
            while(up.length() < n){
                char times = up.charAt(below.length());
                below.append(times);    //122 1

                char appendChar = up.charAt(up.length() - 1) == '1' ? '2' : '1';

                while(times > '0'){
                    up.append(appendChar);
                    times--;
                }
            }

```

```
        }

        int res = 0;
        for(int i = 0; i < n; i++){
            if(up.charAt(i) == '1')
                res++;
        }



        return res;
    }

```

482 License Key Formatting

```
//二刷
public String licenseKeyFormatting(String s, int k) {
        StringBuilder res = new StringBuilder();
        int index = s.length() - 1;

        int counter = 0;
        while(index >= 0){
            if(s.charAt(index) == '-'){
                index--;
                continue;
            }
            res.append(toUpperCase(s.charAt(index)));
            counter++;

            if(counter == k){
                counter = 0;

                res.append("-");
            }

            index--;
        }

        if(res.length() >= 1 && res.charAt(res.length() - 1) == '-')
```

```
            return res.reverse().toString().substring(1);
        return res.reverse().toString();
    }

    public char toUpperCase(char ch){
        if(ch <= 'z' && ch >= 'a')
            return  (char)(ch - 32);

        return ch;
    }
```

执行结果： 通过 显示详情 >

执行用时： 79 ms ，在所有 Java 提交中击败了 17.96% 的用户

内存消耗： 38.5 MB ，在所有 Java 提交中击败了 68.08% 的用户

炫耀一下:

```
/*
    典型双指针
*/
public String licenseKeyFormatting(String S, int K) {
        int right = S.length() - 1;
        StringBuilder sb = new StringBuilder();

        while(right >= 0){
            for (int i = 0; i < K; ) {
                if(right >= 0 && S.charAt(right) == '-') {
                    right--;
                    continue;
                }

                if(right < 0)
                    break;

                char ch = S.charAt(right);
                if('a' <= ch && ch <= 'z')
                    ch -= 32;

                sb.insert(0, ch);
                i++;
                right--;
            }


            sb.insert(0, '-');
```

```
        }

        int delPos = 0;
        while(delPos < sb.length() && sb.charAt(delPos) == '-')
            delPos++;
            sb.delete(0, delPos);
        return sb.toString();
    }

```

483 Smallest Good Base

483. Smallest Good Base

难度 困难 49

For an integer n, we call k>=2 a ***good base*** of n, if all digits of n base k are 1.

Now given a string representing n, you should return the smallest good base of n in string format.

Example 1:

```
Input: "13"
Output: "3"
Explanation: 13 base 3 is 111.
```

Example 2:

```
Input: "4681"
Output: "8"
Explanation: 4681 base 8 is 11111.
```

Example 3:

```
Input: "1000000000000000000"
Output: "999999999999999999"
Explanation: 1000000000000000000 base
999999999999999999 is 11.
```

$$(x+y)^n = \binom{n}{0}x^n y^0 + \binom{n}{1}x^{n-1}y^1 + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n-1}x^1 y^{n-1} + \binom{n}{n}x^0 y^n,$$

$$n!$$

```
/*
    纯粹一道数学题

    涉及放缩
    首先来看

    k ^ 0 + k ^ 1 + .... k ^ m = n

    那么必然有个式子叫  k ^ m < n

    再看二项式定理

```

```
    (k + 1) ^ m = Cm0 * k ^ 0 * 1 ^ n + Cm1 * k ^ 1 * 1 ^ (n - 1)...
    必然有
    (k + 1) ^ m > n

    so
     k ^ m < n < (k + 1) ^ m

     k < n^(-m) < k + 1;

*/
public String smallestGoodBase(String n) {
        long target = Long.parseLong(n);
        for (int i = 2; i <= 60; i++) {
            long k = (long) Math.pow(target, 1.0 / (1.0 * i));

            if(k == 1)
                continue;

            long sum = 1;
            long base = 1;
            for(int j = 1; j <= i; j++){
                base *= k;
                sum += base;
            }

            if(sum == target)
                return String.valueOf(k);
        }

  /*
    因为没有找到合适的，那么这个时候， 就返回第二大的
    第一大的是自己， 因为比如 1000， 那么 1000的最大表示就是 1000进制， 也就是1
    第二大的是比自己小1号，也就是 999， 因为 1000 用 999 表示就是 11
  */
        return String.valueOf(target - 1);
    }

```

484 Find Permutation

```
source:
```

```java
    public int[] findPermutation(String s) {
        int[] res = new int[s.length() + 1];
        Deque<Integer> stack = new ArrayDeque<>();
        int index = 0;
        for(int i = 1; i <= s.length(); i++){
            if(s.charAt(i - 1) == 'I'){
                stack.push(i);
                while(!stack.isEmpty())
                    res[index++] = stack.pop();
            }else{
                stack.push(i);
            }
        }

        stack.push(s.length() + 1);
        while(!stack.isEmpty()){
            res[index++] = stack.pop();
        }

        return res;
    }
```

485 Max Consecutive Ones

执行用时： 2 ms , 在所有 Java 提交中击败了 89.74% 的用户

内存消耗： 39.9 MB , 在所有 Java 提交中击败了 69.29% 的用户

炫耀一下:

```java
// 二刷
    public int findMaxConsecutiveOnes(int[] nums) {
        int index = 0;
        int res = 0;
        while(index < nums.length){
            while(index < nums.length && nums[index] != 1)
                index++;

            int counter = 0;
            while(index < nums.length && nums[index] == 1){
                counter++;
                index++;
            }

```

```
            res = Math.max(res, counter);
        }

        return res;
    }
```

执行用时： 367 ms , 在所有 Java 提交中击败了 7.84% 的用户

内存消耗： 40.4 MB , 在所有 Java 提交中击败了 6.63% 的用户

```
    public int findMaxConsecutiveOnes(int[] nums) {
        int left = 0, right = 0;
        int res = 0;

        while(left < nums.length){
            while(left < nums.length && nums[left] == 0)
                left++;
            if(left == nums.length)
                break;

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

            res = Math.max(res, right - left);
            left++;
        }

        return res;
    }
```

486 Predict the Winner 回看

486. Predict the Winner

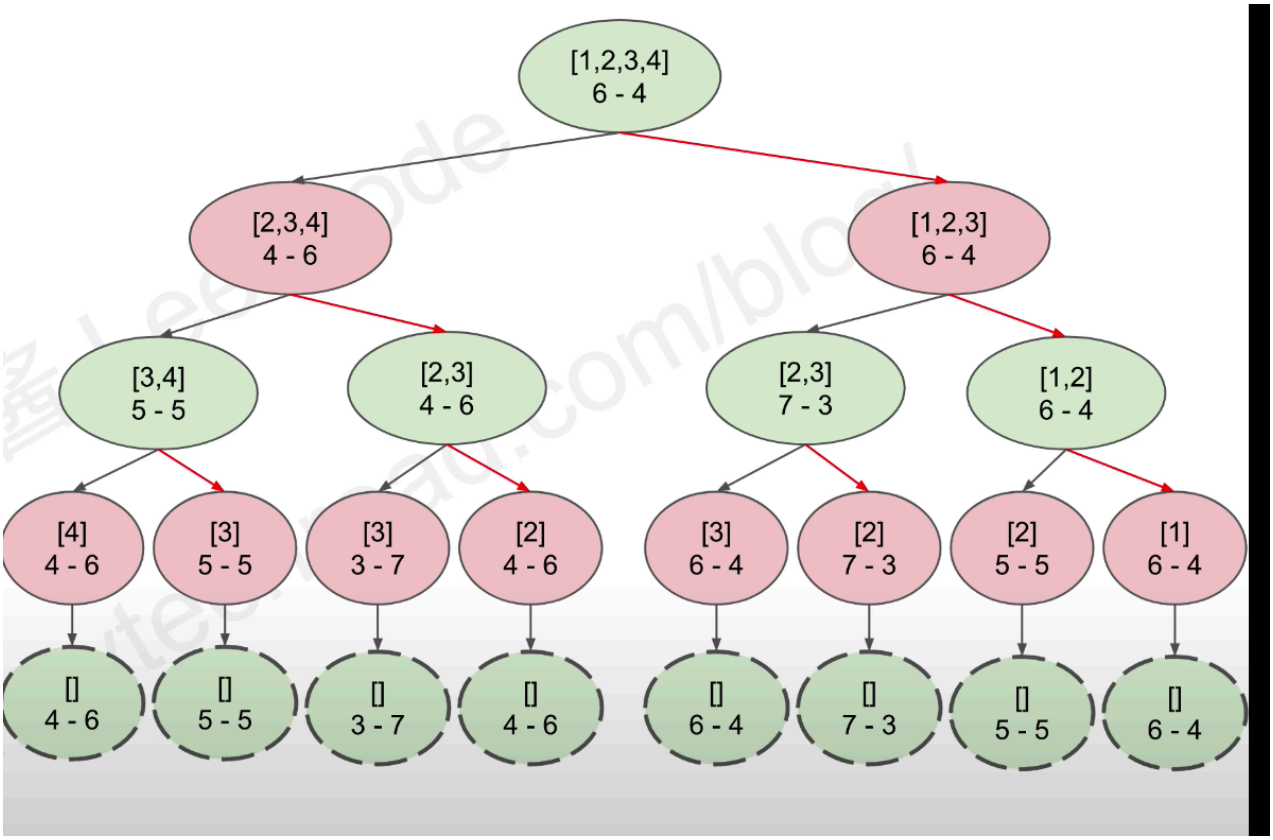
难度 中等 398

Given an array of scores that are non-negative integers. Player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on. Each time a player picks a number, that number will not be available for the next player. This continues until all the scores have been chosen. The player with the maximum score wins.

Given an array of scores, predict whether player 1 is the winner. You can assume each player plays to maximize his score.

Example 1:

```
Input: [1, 5, 2]
Output: False
Explanation: Initially, player 1 can choose
between 1 and 2.
If he chooses 2 (or 1), then player 2 can choose
from 1 (or 2) and 5. If player 2 chooses 5, then
player 1 will be left with 1 (or 2).
So, final score of player 1 is 1 + 2 = 3, and
player 2 is 5.
Hence, player 1 will never be the winner and you
need to return False.
```



执行结果： 通过 显示详情 >

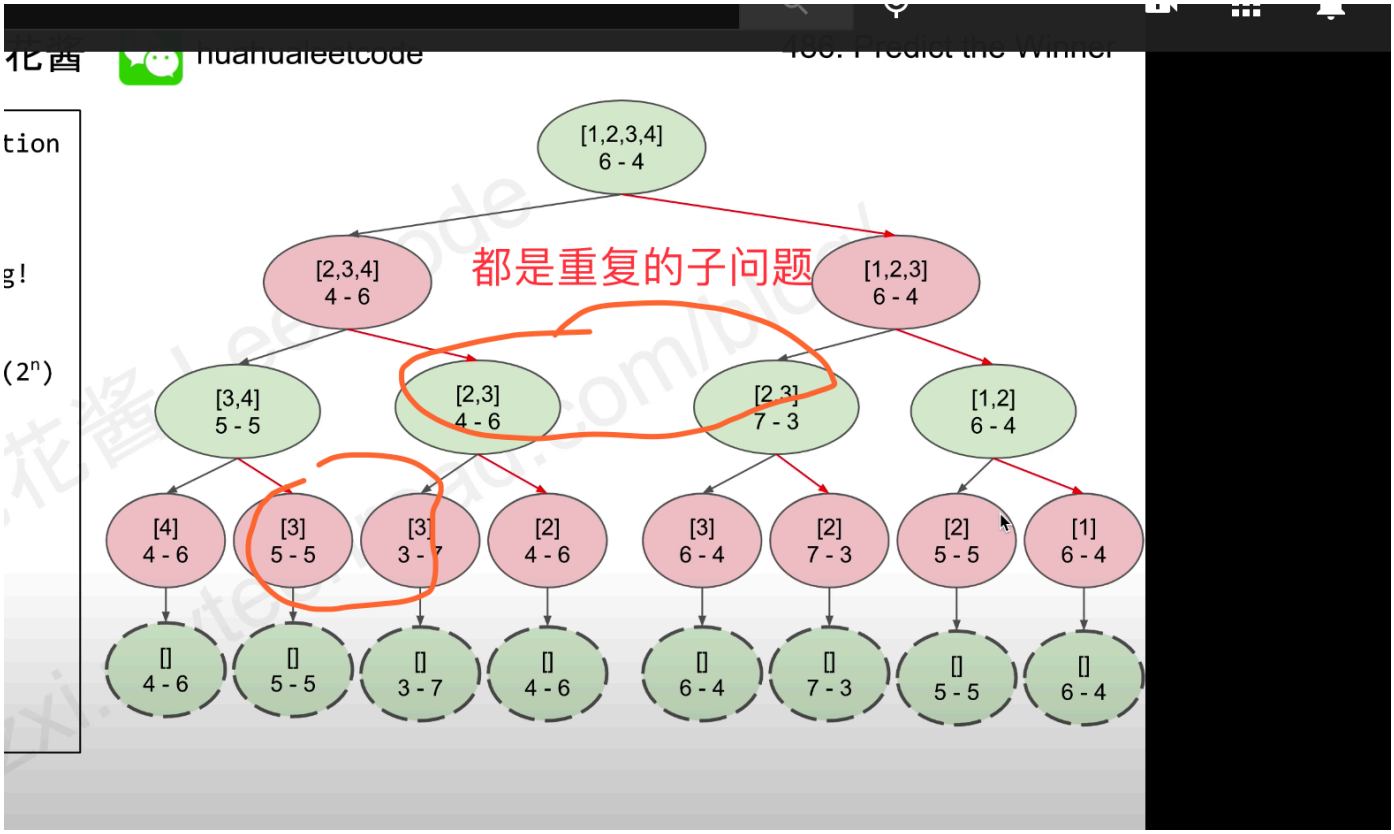
执行用时： 69 ms , 在所有 Java 提交中击败了 16.47% 的用户

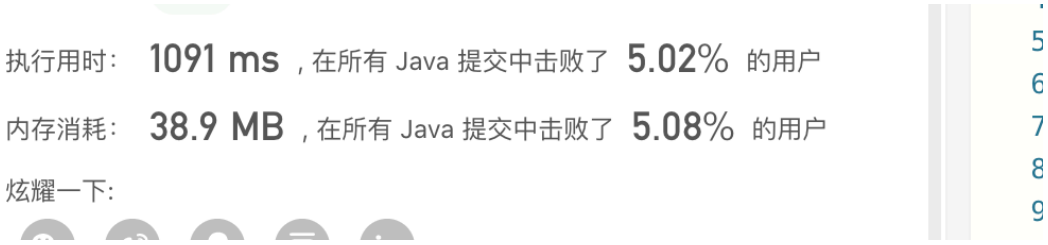
内存消耗： 36 MB , 在所有 Java 提交中击败了 16.54% 的用户

炫耀一下:

```java
/*
    首先是brute force
    https://www.youtube.com/watch?v=g5wLHFTodm0


*/

public boolean PredictTheWinner(int[] nums) {
        return getScore(nums, 0, nums.length - 1) >= 0;
    }

/*
  getScore 的作用是， 在每个玩家都认真玩的基础上， 拿到当前玩家 - 对手的净分数
  比如我可以拿到5分， 对手可以拿到 4分
  那么score 返回的就是一个差值， 可正可负
*/
public int getScore(int[] nums, int leftBound, int rightBound){
        if(leftBound == rightBound){
            return nums[leftBound];
        }
```

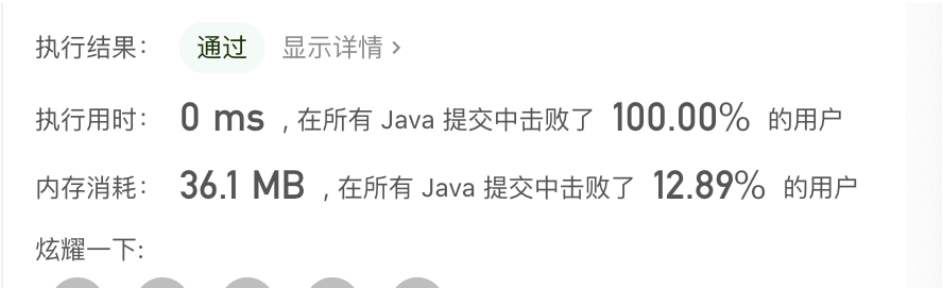
```
21
22          return Math.max(nums[leftBound] - getScore(nums, leftBound + 1,
    rightBound),
23                          nums[rightBound] - getScore(nums, leftBound, rightBound -
    1));
24      }
```





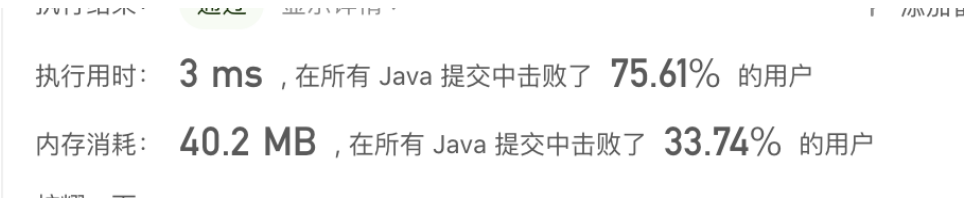

```
1   /*
2       use dp(i, j) to store the best score that current player can achieve
3       将时间复杂度从 2^n 降低到 n^2
4
5   */
6   HashMap<String, Integer> map;
7   public boolean PredictTheWinner(int[] nums) {
8           map = new HashMap<>();
9           return getScore(nums, 0, nums.length - 1) >= 0;
10      }
11
12  public int getScore(int[] nums, int leftBound, int rightBound){
```

```
        String str = leftBound + "@" + rightBound;
      if(map.containsKey(str))
            return map.get(str);
        if(leftBound == rightBound){
            return nums[leftBound];
        }

        int res = Math.max(nums[leftBound] - getScore(nums, leftBound + 1,
rightBound)                          , nums[rightBound] - getScore(nums, leftBound,
rightBound - 1));

        map.put(str, res);
        return res;
    }
```




```
    public boolean PredictTheWinner(int[] nums) {
        /*
            先手玩家 和 后手玩家的分数差值
        */
        int[][] dp = new int[nums.length][nums.length];

        for(int i = 0; i < nums.length; i++)
            dp[i][i] = nums[i];

        for(int i = nums.length - 2; i >= 0; i--)
            for(int j = i + 1; j < nums.length; j++){
                dp[i][j] = Math.max(nums[i] - dp[i + 1][j], nums[j] - dp[i][j -
1]);
            }


        return dp[0][nums.length - 1] >= 0;
    }
```

487 Max Consecutive Ones II




```java
//二刷，采用双指针
    public int findMaxConsecutiveOnes(int[] nums) {
        int counter = 0;
        int left = 0, right = 0;
        int len = nums.length;
        int res = 0;

        while(right < len){
            while(right < len && counter <= 1){
                res = Math.max(res, right - left);
                if(nums[right] == 0)
                    counter++;

                right++;
            }

            if(right == len){
                if(counter <= 1)
                    res = Math.max(res, right - left);
                break;
            }
            while(left < right && counter > 1){
                if(nums[left] == 0)
                    counter--;
                left++;
            }
        }

        return res;
    }
```

```java
/*
    这种解法不是很优雅，corner cases 太多了

    类似用到了两个双指针
*/
       //index,  length
```

```java
    List<Map.Entry<Integer, Integer>> list;
    public int findMaxConsecutiveOnes(int[] nums) {
        list = new ArrayList<>();

        int left = 0, right = 0;

        while(true){
            while(left < nums.length && nums[left] == 0)
                left++;
            if(left == nums.length)
                break;

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

            list.add(Map.entry(left, right - left));
            left = right;
        }
        if(list.size() == 1 && list.get(0).getValue() == nums.length)
            return nums.length;

        int res = 0;
        for(int i = 0;  i< list.size(); i++){
            res = Math.max(res, list.get(i).getValue());
            boolean enter = false;

            if(i != list.size() - 1){
                if(list.get(i).getKey() + 1 + list.get(i).getValue() == list.get(i
+ 1).getKey()) {
                    res = Math.max(res, list.get(i).getValue() + list.get(i +
1).getValue() + 1);
                    enter = true;
                }
            }

            if(i != 0){
                if(list.get(i - 1).getKey() + 1 + list.get(i).getValue() ==
list.get(i).getKey()) {
                    res = Math.max(res, list.get(i - 1).getValue() +
list.get(i).getValue() + 1);
                    enter = true;
                }
            }

            if(!enter){
                if(nums.length > 1)
                    res = Math.max(list.get(i).getValue() + 1, res);
            }
```

```
        }

        return res == 0 && nums.length >= 1 ? 1 : res;
    }

```

```
/*
其实这个题目等价于：给定一个区间，该区间中最多只能包含1个0，求出该区间的最大长度。
*/
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int res = 0, count = 0;
        for(int l = 0, r = 0; r < nums.length; r++) {
            if(nums[r] == 0) {
                count++;
                while(count > 1) {
                    count -= nums[l++] == 0 ? 1 : 0;
                }
            }
            res = Math.max(res, r - l + 1);
        }

        return res;
    }
}

作者：lyl0724-2
链接：https://leetcode-cn.com/problems/max-consecutive-ones-ii/solution/javahua-dong-chuang-kou-by-lyl0724-2/
来源：力扣（LeetCode）

```

488 Zuma Game

488. Zuma Game

难度 困难 69

Think about Zuma Game. You have a row of balls on the table, colored red(R), yellow(Y), blue(B), green(G), and white(W). You also have several balls in your hand.

Each time, you may choose a ball in your hand, and insert it into the row (including the leftmost place and rightmost place). Then, if there is a group of 3 or more balls in the same color touching, remove these balls. Keep doing this until no more balls can be removed.

Find the minimal balls you have to insert to remove all the balls on the table. If you cannot remove all the balls, output -1.

Example 1:

```
Input: board = "WRRBBW", hand = "RB"
Output: -1
Explanation: WRRBBW -> WRR[R]BBW -> WBBW ->
WBB[B]W -> WW
```

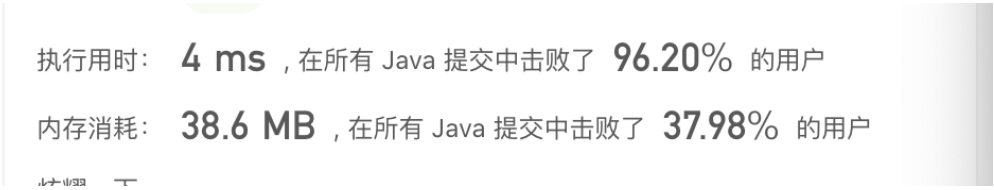
Example 2:

```
Input: board = "WWRRBBWW", hand = "WRBRW"
Output: 2
Explanation: WWRRBBWW -> WWRR[R]BBWW -> WWBBWW ->
WWBB[B]WW -> WWWW -> empty
```

```java
//二刷
    int res = Integer.MAX_VALUE;
    public int findMinStep(String board, String hand) {
        if(board.equals("RRWWRRBBRR") && hand.equals("WB"))
            return 2;
        Map<Character, Integer> map = new HashMap<>();
        for (Character ch : hand.toCharArray())
            map.put(ch, map.getOrDefault(ch, 0) + 1);

        StringBuilder boardSB = new StringBuilder();
        boardSB.append(board);
        dfs(boardSB, map, 0);

        return res == Integer.MAX_VALUE ? -1 : res;
    }
```

```
16
17      private void dfs(StringBuilder board, Map<Character, Integer> map, int
    progress) {
18          if(board.length() == 0){
19              res = Math.min(res, progress);
20              return;
21          }
22
23          for(int i = 0; i < board.length();){
24              char ch = board.charAt(i);
25              if(map.getOrDefault(ch, 0) > 0){
26                  StringBuilder temp = getRes(board, i);
27
28                  map.put(ch, map.get(ch) - 1);
29                  dfs(temp, map, progress + 1);
30                  map.put(ch, map.get(ch) + 1);
31
32                  int right = i;
33                  while(right < board.length() && board.charAt(right) == ch)
34                      right++;
35                  i = right;
36              }else{
37                  i++;
38              }
39          }
40      }
41
42      private StringBuilder getRes(StringBuilder board, int pos) {
43          StringBuilder result = new StringBuilder(board);
44          char ch = result.charAt(pos);
45          result.insert(pos, ch);
46
47          while(canEliminate(result)){
48              for(int i = 0; i < result.length() - 2; i++){
49                  if(result.charAt(i) == result.charAt(i + 1) &&
50                          result.charAt(i + 1) == result.charAt(i + 2)){
51                      int left = i, right = i;
52                      while(right < result.length() && result.charAt(right) ==
    result.charAt(left))
53                          right++;
54
55                      StringBuilder temp = new StringBuilder();
56                      temp.append(result, 0, left);
57                      temp.append(result, right, result.length());
58                      result = temp;
59                      break;
60                  }
61
62              }
```

```
        }

        return result;
    }

    private boolean canEliminate(StringBuilder board) {
        for(int i = 0; i < board.length() - 2; i++){
            if(board.charAt(i) == board.charAt(i + 1) &&
            board.charAt(i + 1) == board.charAt(i + 2))
                return true;
        }

        return false;
    }
```

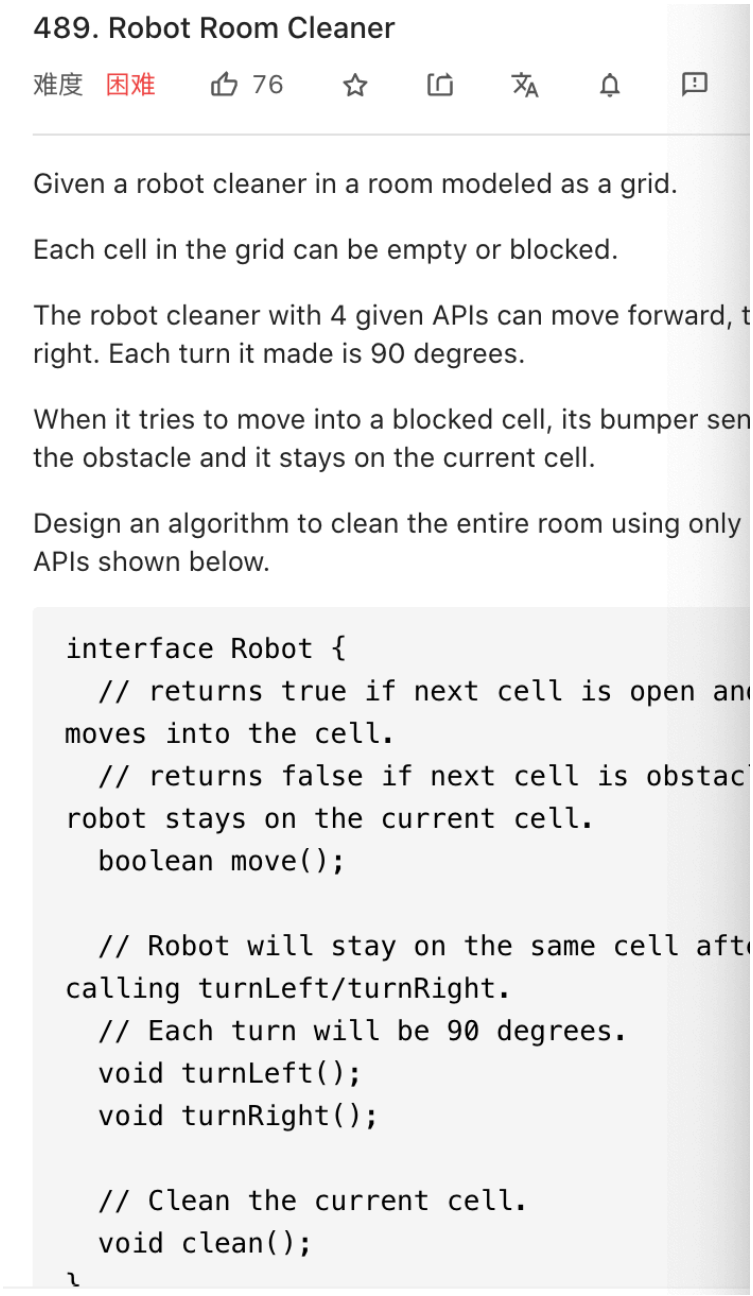



```
class Solution {

    int res = 6;
    public int findMinStep(String board, String hand) {
        if(board.equals("RRWWRRBBRR") && hand.equals("WB"))
            return 2;
        int[] alpha = new int[26];
        for(char ch : hand.toCharArray())
            alpha[ch - 'A']++;

        backtrack(new StringBuilder(board), alpha, 0);
        return res == 6 ? -1 : res;
    }

/*
    helper function
    用来执行递归操作
*/
    private void backtrack(StringBuilder board, int[] alpha, int opNum) {
        if(opNum == 6 || board.length() == 0){
            if(board.length() == 0)
                res = Math.min(res, opNum);
            return;
        }

```

```
        for(int i = 0; i < board.length(); i++){
            for(int j = 0; j < alpha.length; j++){
                char ch = (char)(j + 'A');
                if(alpha[j] != 0 && ch == board.charAt(i)){
                    StringBuilder temp = new StringBuilder(board);

                    temp.insert(i, ch);
                    temp = squeezeString(temp.toString());

                    alpha[j]--;
                    backtrack(temp, alpha, opNum + 1);
                    alpha[j]++;

                }
            }
        }
    }

/*
    将字符串进行squeeze,  达到无法compress状态
*/
    private StringBuilder squeezeString(String temp) {
        while(true){
            int[] nums = canEliminate(temp);
            if(nums[0] == -1 && nums[1] == -1)
                return new StringBuilder(temp);

            temp = temp.substring(0, nums[0]) + temp.substring(nums[1] + 1);
        }
    }

/*
    判断是否可以 eliminate ?  直接返回对应坐标
    algo: 使用双指针进行操作
*/
    public int[] canEliminate(String sb){
        int left = 0, right = 0;

        while(right < sb.length()){
            while(right < sb.length() && sb.charAt(right) == sb.charAt(left))
                right++;

            if(right - left  >= 3){
                return new int[]{left, right - 1};
            }

            left = right;
        }

```

```
75          return new int[]{-1, -1};
76      }
77
78  }
```

489 Robot Room Cleaner



489. Robot Room Cleaner

难度 困难 76

Given a robot cleaner in a room modeled as a grid.

Each cell in the grid can be empty or blocked.

The robot cleaner with 4 given APIs can move forward, t
right. Each turn it made is 90 degrees.

When it tries to move into a blocked cell, its bumper sen
the obstacle and it stays on the current cell.

Design an algorithm to clean the entire room using only
APIs shown below.

```
interface Robot {
  // returns true if next cell is open an
moves into the cell.
  // returns false if next cell is obstac
robot stays on the current cell.
  boolean move();

  // Robot will stay on the same cell aft
calling turnLeft/turnRight.
  // Each turn will be 90 degrees.
  void turnLeft();
  void turnRight();

  // Clean the current cell.
  void clean();
}
```

```
1  /*
2    蛮有意思的一道题目
```

```java
*/
class Solution {

  //注意，这个数组有顺序， 按照 右， 下， 左， 上
    int[][] dir = {{ 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 }};
    HashSet<String> set;
    public void cleanRoom(Robot robot) {
        set = new HashSet<>();
        dfs(robot, 0, 0, 0);
    }

    public void dfs(Robot robot, int x, int y, int pos){
        robot.clean();
        set.add(x + "@" + y);

        for(int k = 0; k < 4; k++){
          //下一个移动的方向
            int nextDir =  (pos + k) % 4;
            int newX    = x + dir[nextDir][0];
            int newY    = y + dir[nextDir][1];

            if(!set.contains(newX + "@" + newY) && robot.move()){
                dfs(robot, newX, newY, nextDir);//如何理解这个问题？ 要知道，当dfs结束后， robot 仍然
                                                //停留在 newX， newY， 因此需要把它弄回来
                robot.turnRight();
                robot.turnRight();

                robot.move(); //转回来
                robot.turnLeft(); //换到下一个方向，比如说当前是下，下一个方向就是左
            }else{
                robot.turnRight();
            }
        }
    }
}

作者：shurui91
链接：https://leetcode-cn.com/problems/robot-room-cleaner/solution/java-dfs-by-shurui91/
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。
```

490 The Maze

执行结果： 通过 显示详情 >

执行用时： 96 ms , 在所有 Java 提交中击败了 5.45% 的用户

内存消耗： 39.8 MB , 在所有 Java 提交中击败了 5.59% 的用户

```java
/*
    典型 BFS 题目
*/
  int[][] maze;
    int[][] dir = {{0, 1},{1, 0},{0, -1},{-1, 0}};
    public boolean hasPath(int[][] maze, int[] start, int[] destination) {
        Deque<int[]>      queue = new ArrayDeque<>();
        HashSet<String> endPoint = new HashSet<>();
        this.maze               = maze;

        queue.add(start);
        while(!queue.isEmpty()){
            int size = queue.size();

            for (int i = 0; i < size; i++) {
                int[] curPos = queue.poll();
                endPoint.add(curPos[0] + "@" + curPos[1]);

                for(int k = 0; k < 4; k++){
                    int[] nextEndPoint = getNextEndPoint(curPos, dir[k]);
                    if(nextEndPoint[0] == destination[0] && nextEndPoint[1] ==
destination[1])
                        return true;

                    if(endPoint.contains(nextEndPoint[0] + "@" + nextEndPoint[1]))
                        continue;
                    queue.add(nextEndPoint);
                }
            }
        }

        return false;
    }

    private int[] getNextEndPoint(int[] curPos, int[] vec) {
        int x = curPos[0];
        int y = curPos[1];
        //move horizontally
        if(vec[0] == 0){
            if(vec[1] == 1){    //move right
                while(isInRange(x, y) && maze[x][y] == 0)
                    y++;
```

```
                return new int[]{x, y - 1};
            }else{              //move left
                while(isInRange(x, y) && maze[x][y] == 0)
                    y--;
                return new int[]{x, y + 1};
            }
        }else{//move vertically
            if(vec[0] == 1){    //move down
                while(isInRange(x, y) && maze[x][y] == 0)
                    x++;
                return new int[]{x - 1, y};
            }else{              //move up
                while(isInRange(x, y) && maze[x][y] == 0)
                    x--;
                return new int[]{x + 1, y};
            }
        }
    }

    public boolean isInRange(int x, int y){
        return x >= 0 && y >= 0 && x < maze.length && y < maze[0].length;
    }


```

491 Increasing Subsequences 注意去重

```
//  大体上 O(n ^ 3) 时间复杂度
  public List<List<Integer>> findSubsequences(int[] nums) {
        int len = nums.length;

        ArrayList<ArrayList<Integer>>[] bags = new ArrayList[len];

        for(int i = 0; i < len; i++){
            bags[i] = new ArrayList<>();
            bags[i].add(new ArrayList<>());
            bags[i].get(0).add(nums[i]);
        }

        for(int i = 1; i < len; i++){
            for(int j = 0; j < i; j++){
                ArrayList<ArrayList<Integer>> bagsJ = bags[j];
                for(ArrayList<Integer> listJ : bagsJ){
```

```
                    if(listJ.get(listJ.size() - 1) > nums[i])
                        continue;

                    ArrayList<Integer> integers = new ArrayList<>(listJ);
                    integers.add(nums[i]);
                    bags[i].add(integers);
                }
            }
        }

        Set<ArrayList<Integer>> set = new HashSet<>();
        for(int i = 0; i < len; i++){
            ArrayList<ArrayList<Integer>> bag = bags[i];
            for(ArrayList<Integer> b : bag){
                if(b.size() > 1)
                    set.add(b);
            }
        }

        return new ArrayList<>(set);
    }
```

执行结果： 通过 显示详情 >

执行用时： 14 ms , 在所有 Java 提交中击败了 21.32% 的用户

内存消耗： 46.3 MB , 在所有 Java 提交中击败了 23.61% 的用户

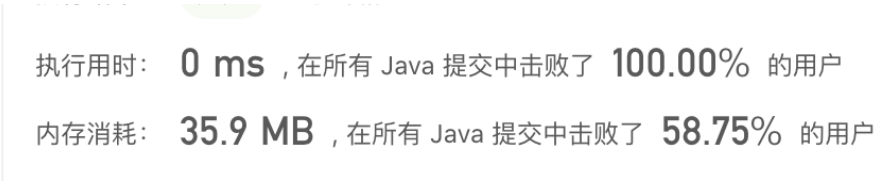
```
    HashSet<List<Integer>> res;
    public List<List<Integer>> findSubsequences(int[] nums) {
        res = new HashSet<>();

        backtrack(nums, new ArrayList<Integer>(), 0);
        List<List<Integer>> ans = new ArrayList<>();
        ans.addAll(res);
        return ans;
    }

    public void backtrack(int[] nums, List<Integer> path, int start){
        if(start == nums.length) {
            return;
        }

        for(int i = start; i < nums.length; i++){
            int endNum = 0;
            if(path.size() >= 1)
```

```
                endNum = path.get(path.size() - 1);

            if(path.size() == 0 || endNum <= nums[i]){
                path.add(nums[i]);
                if(path.size() >= 2){
                    res.add(new ArrayList<>(path));
                }

                backtrack(nums, path, i + 1);

                path.remove(path.size() - 1);
            }
        }
    }

```

```
    List<List<Integer>> res;
    public List<List<Integer>> findSubsequences(int[] nums) {
        res = new ArrayList<>();

        backtrack(nums, new ArrayList<Integer>(), 0);
        return res;
    }

    public void backtrack(int[] nums, List<Integer> path, int start){
        if(start == nums.length) {
            return;
        }

      //这里使用 visited 去重
        HashSet<Integer> visited = new HashSet<>();
        for(int i = start; i < nums.length; i++){
            if(visited.contains(nums[i]))       continue;

            visited.add(nums[i]);
            if(path.size() == 0 || path.get(path.size() - 1) <= nums[i]){
                path.add(nums[i]);
                if(path.size() >= 2){
                    res.add(new ArrayList<>(path));
                }

                backtrack(nums, path, i + 1);

                path.remove(path.size() - 1);
            }
```

```
        }
    }

```

492 Construct the Rectangle




```java
    List<int[]> candidate = new ArrayList<>();
    public int[] constructRectangle(int area) {
        int W = (int) Math.sqrt(area);

        if(W * W == area)
            return new int[]{W, W};

        for(int i = W ; i >= 1; i--){
            if(area % i == 0){
                return new int[]{area / i, i};
            }
        }

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

493 Reverse Pairs

```java
/*
  暴力解法
*/
public int reversePairs(int[] nums) {
        int count = 0;
        for(int i = 0; i < nums.length; i++){
            for(int j = i + 1; j < nums.length; j++)
                if(((long)nums[i]) > (2 * (long)nums[j]))
                    count++;
        }

        return count;
    }
```

```java
/*
    典型归并排序应用
    类似数组中的逆序对

    注意写法不能和逆序对完全一致

*/
int count = 0;
    public int reversePairs(int[] nums) {
        sort(nums);
        return  count;
    }

    int[] aux;
    public void sort(int[] nums){
        aux = new int[nums.length];
        sort(nums, 0, nums.length - 1);
    }

    private void sort(int[] nums, int lo, int hi) {
        if(hi <= lo)    return;

        int mid = (lo + hi) / 2;
        sort(nums,lo, mid);
        sort(nums, mid + 1, hi);

        merge(nums, lo, mid, hi);
    }

    private void merge(int[] nums, int lo, int mid, int hi) {
        int i = lo, j = mid + 1;
        for(int k = lo; k <= hi; k++)
```

```
            aux[k] = nums[k];

        //在归并之前，来找一下
        while(i <= mid && j <= hi){
            if((long)nums[i] > 2 * (long)nums[j]){
                count += mid - i + 1;
                j++;
            }else{
                i++;
            }
        }
        i = lo;
        j = mid + 1;
        int index = lo;

        while(i <= mid || j <= hi){
            if(i > mid)                 nums[index++] = aux[j++];
            else if(j > hi)             nums[index++] = aux[i++];
          //对于负数的情况，这里可能会被漏掉
            else if(aux[i] < aux[j]) nums[index++] = aux[i++];
            else                        nums[index++] = aux[j++];

        }
    }

```

494 Target Sum

执行用时： 127 ms ，在所有 Java 提交中击败了 40.44% 的用户

内存消耗： 39.2 MB ，在所有 Java 提交中击败了 5.01% 的用户

```

    Map<String, Integer> map = new HashMap<>();
    public int findTargetSumWays(int[] nums, int target) {
        return dfs(nums, target, 0);
    }

    /*
        get the result
```

```
    */
    private int dfs(int[] nums, int remains, int index){
        if(index == nums.length){
            if(remains == 0)
                return 1;
            return 0;
        }
        String symbol = remains + "@" + index;
        if(map.containsKey(symbol))
            return map.get(symbol);


        int res = 0;

        res += dfs(nums, remains - nums[index], index + 1);
        res += dfs(nums, remains + nums[index], index + 1);

        map.put(symbol, res);
        return res;
    }
```


```
/*
    回溯秒了
*/
class Solution {
    int count = 0;
    public int findTargetSumWays(int[] nums, int S) {
        backtrack(nums, S, 0, 0);
        return count;
    }

    private void backtrack(int[] nums, int S, int sum, int start) {
        if(start == nums.length){
            if(S == sum)  count++;

            return;
```

```
        }

        backtrack(nums, S, sum + nums[start], start + 1);
        backtrack(nums, S, sum - nums[start], start + 1);
    }
}
```

```
/*
    用 dp 的思想去优化
*/
HashMap<Map.Entry<Integer, Integer>, Integer> map;
    public int findTargetSumWays(int[] nums, int S) {
        map = new HashMap<>();
        return dfs(nums, S, 0, 0);
    }

/*
    拿到， 截止到目前索引(start, nums.length - 1)， 能组成 S 的种类数目
*/
    private int dfs(int[] nums, int S, int sum, int start) {

        Map.Entry<Integer, Integer> entry = Map.entry(start, sum);
        if(map.containsKey(entry))  return map.get(entry);

        if(start == nums.length && sum == S)    return 1;
        if(start == nums.length)                return 0;

        int pos = dfs(nums, S, sum + nums[start], start + 1);
        int neg = dfs(nums, S, sum - nums[start], start + 1);

        entry = Map.entry(start, sum);
        map.put(entry, pos + neg);

        return pos + neg;
    }

```

495 Teemo Attacking

执行用时： 2 ms ，在所有 Java 提交中击败了 98.31% 的用户

内存消耗： 40.4 MB ，在所有 Java 提交中击败了 34.28% 的用户

荣耀一下：

```java

public int findPoisonedDuration(int[] timeSeries, int duration) {
        int res = 0;

        int index = 0;
        int expectedFreeTime = 0;     //代表中毒到什么时间
        for (int i = 0; i < timeSeries.length;) {
            if(i == 0){
                expectedFreeTime = timeSeries[i] + duration;
                res += duration;
                i++;
            }else{
                int nextExpectedFreeTime = timeSeries[i] + duration;
                int existTime            = 0;
                if(expectedFreeTime <= timeSeries[i]){
                    res += duration;
                    expectedFreeTime = timeSeries[i] + duration;
                    i++;
                }else{
                    existTime = timeSeries[i] + duration - expectedFreeTime;
                    expectedFreeTime += existTime;
                    i++;
                    res += existTime;
                }
            }
        }

        return res;
    }

```

496 Next Greater Element 单调栈的典型应用

执行用时： 7 ms ，在所有 Java 提交中击败了 23.79% 的用户

内存消耗： 38.7 MB ，在所有 Java 提交中击败了 42.42% 的用户

炫耀一下：

```java
public int[] nextGreaterElement(int[] nums1, int[] nums2) {
    int[] res = new int[nums1.length];
    int index = 0;
    for(int i = 0; i < nums1.length; i++){
        int j = 0;
        while(j < nums2.length && nums2[j] != nums1[i])
            j++;


        while(j < nums2.length && nums2[j] <= nums1[i])
            j++;

        if(j == nums2.length)
            res[index++] = -1;
        else
            res[index++] = nums2[j];
    }
   return res;
}
```

执行用时： 4 ms ，在所有 Java 提交中击败了 91.99% 的用户

内存消耗： 39 MB ，在所有 Java 提交中击败了 5.23% 的用户

炫耀一下：

```java
public int[] nextGreaterElement(int[] nums1, int[] nums2) {
    Deque<Integer>      stack = new ArrayDeque<>();
    Map<Integer, Integer> map = new HashMap<>();

    for(int i = nums2.length - 1; i >= 0; i--){
        while(!stack.isEmpty() && nums2[stack.peek()] < nums2[i])
            stack.pop();

        map.put(nums2[i], stack.isEmpty() ? -1 : nums2[stack.peek()]);

```

```
            stack.push(i);
        }

        int[] res = new int[nums1.length];
        for(int i = 0;  i < nums1.length; i++){
            res[i] = map.get(nums1[i]);
        }

        return res;
    }
```

497 Random Point in Non-overlapping Rectangles

497. Random Point in Non-overlapping Rectangles

难度 中等 33

Given a list of **non-overlapping** axis-aligned rectangles `rects`, write a function `pick` which randomly and uniformily picks an **integer point** in the space covered by the rectangles.

Note:

1. An **integer point** is a point that has integer coordinates.
2. A point on the perimeter of a rectangle is **included** in the space covered by the rectangles.
3. `i` th rectangle = `rects[i]` = `[x1,y1,x2,y2]`, where `[x1, y1]` are the integer coordinates of the bottom-left corner, and `[x2, y2]` are the integer coordinates of the top-right corner.
4. length and width of each rectangle does not exceed `2000`.
5. `1 <= rects.length <= 100`
6. `pick` return a point as an array of integer coordinates `[p_x, p_y]`
7. `pick` is called at most `10000` times.

Example 1:

```
Input:
["Solution","pick","pick","pick"]
[[[[1,1,5,5]]],[],[],[]]
Output:
[null,[4,1],[4,1],[3,3]]
```

Example 2:

```
Input:
["Solution","pick","pick","pick","pick","pick"]
```

```
1
```

498 Diagonal Traverse

执行用时： 8 ms , 在所有 Java 提交中击败了 23.80% 的用户

内存消耗： 40.7 MB , 在所有 Java 提交中击败了 32.78% 的用户

498. Diagonal Traverse

难度 中等 168

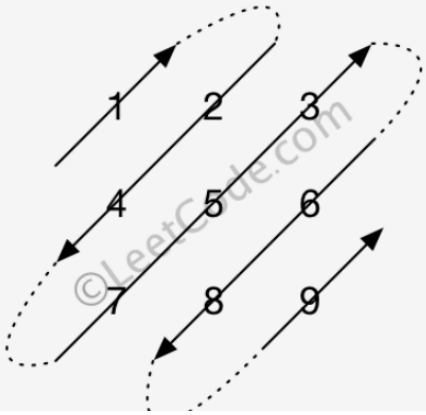
Given a matrix of M x N elements (M rows, N columns), return all elements of the matrix in diagonal order as shown in the below image.

Example:

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

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

Explanation:




```java
int row = 0;
 int col = 0;
 public int[] findDiagonalOrder(int[][] matrix) {
     row = matrix.length;
     col = row == 0 ? 0 : matrix[0].length;

     if(col == 0)
         return new int[]{};

     int target = 0;

     int index = 0;
     int[] res = new int[row * col];
     int x = 0, y = 0;
     while(index != res.length){
         while(isInRange(x, y)){
```

```
                res[index++] = matrix[x][y];    //右上
                if(!isInRange(x - 1, y + 1) && isInRange(x, y + 1)){
                    y++;
                    break;
                }else if(!isInRange(x - 1, y + 1) && isInRange(x + 1, y)){
                    x++;
                    break;
                }
                x--;
                y++;
                target++;
            }

            if(index == res.length)
                break;
            while(isInRange(x, y)){
                res[index++] = matrix[x][y];    //左下
                if(!isInRange(x + 1, y - 1) && isInRange(x + 1, y)){
                    x++;
                    break;
                }else if(!isInRange(x + 1, y - 1) && isInRange(x, y + 1)){
                    y++;
                    break;
                }

                x++;
                y--;
                target++;
            }

        }

        return res;
    }

    public boolean isInRange(int x, int y){
        return x >= 0 && y >= 0 && x < row && y < col;
    }


```

499 The Maze III 回看

```
/*
  本质上就是模拟这个过程
```

```
  采用回溯的方法，  超时
*/
int[][] maze;
    int[][] dir = {{1, 0},{0, -1},{0, 1},{-1, 0}};//d l r u
    final char[]  charDir = {'d', 'l', 'r', 'u'};
    int[] hole;
    String res = "impossible";
    boolean found = false;
    int minLen = Integer.MAX_VALUE;

    HashMap<Integer, List<String>> map = new HashMap<>();
    public String findShortestWay(int[][] maze, int[] ball, int[] hole) {

        this.maze               = maze;
        this.hole               = hole;

        HashSet<String> endPoint = new HashSet<>();
        backtrack(maze, ball[0], ball[1], new StringBuilder(), 0, endPoint);
//        System.out.println(map);
        List<String> res = map.get(minLen);
        if(res != null)
            Collections.sort(res);
        return res == null ? "impossible" : res.get(0);

    }


    private void backtrack(int[][] maze, int x, int y, StringBuilder sb, int
pathLen, HashSet<String> endPoint) {
        String symbol = x + "@" + y;
        if(endPoint.contains(symbol))
            return;

        for(int k = 0; k < 4; k++){
            int[] nextEndPoint = getNextEndPoint(new int[]{x, y}, dir[k]);
            int thisPath = nextEndPoint[2];
            if(nextEndPoint[0] == hole[0] && nextEndPoint[1] == hole[1]){
                sb.append(charDir[k]);
                minLen = Math.min(minLen, pathLen + thisPath - 1);
                map.putIfAbsent(pathLen + thisPath - 1, new ArrayList<>());
                map.get(pathLen + thisPath - 1).add(sb.toString());
                res = sb.toString();
                sb.setLength(sb.length() - 1);
                return;

            }

            sb.append(charDir[k]);
            endPoint.add(symbol);
```

```
51
52              backtrack(maze, nextEndPoint[0], nextEndPoint[1], sb, pathLen +
    thisPath, endPoint);
53
54              sb.setLength(sb.length() - 1);
55              endPoint.remove(symbol);
56          }
57      }
58
59      private int[] getNextEndPoint(int[] curPos, int[] vec) {
60          int x = curPos[0];
61          int y = curPos[1];
62          int[] res = new int[3];
63          res[0] = -1;
64          res[1] = -1;
65          int count = 0;
66          //move horizontally
67          if(vec[0] == 0){
68              if(vec[1] == 1){    //move right
69                  while(isInRange(x, y) && maze[x][y] == 0) {
70                      count++;
71                      if(hole[0] == x && hole[1] == y) {
72                          res[0] = hole[0];
73                          res[1] = hole[1];
74                          res[2] = count;
75                          return res;
76                      }
77                      y++;
78                  }
79
80                  res[0] = x;
81                  res[1] = y - 1;
82                  res[2] = count - 1;
83                  return res;
84              }else{              //move left
85                  while(isInRange(x, y) && maze[x][y] == 0) {
86                      count++;
87                      if(hole[0] == x && hole[1] == y) {
88                          res[0] = hole[0];
89                          res[1] = hole[1];
90                          res[2] = count;
91                          return res;
92                      }
93                      y--;
94                  }
95                  res[0] = x;
96                  res[1] = y + 1;
97                  res[2] = count - 1;
98                  return res;
```

```
            }
        }else{//move vertically
            if(vec[0] == 1){    //move down
                while(isInRange(x, y) && maze[x][y] == 0) {
                    count++;
                    if(hole[0] == x && hole[1] == y) {
                        res[0] = hole[0];
                        res[1] = hole[1];
                        res[2] = count;
                        return res;
                    }
                    x++;
                }

                res[0] = x - 1;
                res[1] = y;
                res[2] = count - 1;
                return res;
            }else{              //move up
                while(isInRange(x, y) && maze[x][y] == 0) {
                    count++;
                    if(hole[0] == x && hole[1] == y) {
                        res[0] = hole[0];
                        res[1] = hole[1];
                        res[2] = count;
                        return res;
                    }
                    x--;
                }
                res[0] = x + 1;
                res[1] = y;
                res[2] = count - 1;
                return res;
            }
        }
    }

    public boolean isInRange(int x, int y){
        return x >= 0 && y >= 0 && x < maze.length && y < maze[0].length;
    }

```

500 Keyboard Row

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： 36.6 MB ，在所有 Java 提交中击败了 30.63% 的用户

炫耀一下：

```java
static HashMap<Character, Integer> map = new HashMap<>();;

 static{
     map.put('q',1);
     map.put('w',1);
     map.put('e',1);
     map.put('r',1);
     map.put('t',1);
     map.put('y',1);
     map.put('u',1);
     map.put('i',1);
     map.put('o',1);
     map.put('p',1);

     map.put('a',2);
     map.put('s',2);
     map.put('d',2);
     map.put('f',2);
     map.put('g',2);
     map.put('h',2);
     map.put('j',2);
     map.put('k',2);
     map.put('l',2);

     map.put('z',3);
     map.put('x',3);
     map.put('c',3);
     map.put('v',3);
     map.put('b',3);
     map.put('n',3);
     map.put('m',3);

 }
 public String[] findWords(String[] words) {
     List<String> res = new ArrayList<>();

     for(int i = 0; i < words.length; i++){
         boolean isSameLine = true;
         String curWord = words[i].toLowerCase();
         for(int j = 0; j < curWord.length() - 1; j++){
             char c1 = curWord.charAt(j);
             char c2 = curWord.charAt(j + 1);
```

```
43
44                  if(map.get(c1).equals(map.get(c2)))
45                      continue;
46                  else {
47                      isSameLine = false;
48                      break;
49                  }
50              }
51
52              if(isSameLine)
53                  res.add(words[i]);
54          }
55
56          String[] strs = new String[res.size()];
57          for(int i = 0; i < strs.length; i++)
58              strs[i] = res.get(i);
59          return strs;
60      }
```