1. 题判断给的array是不是逐一递增或递减的rotated array
比如说 4 5 6 7 1 2 3 -> True

4 5 6 8 1 2 3 -> False

4 3 2 1 7 6 5 -> True

```
public boolean check(int[] A) {
 int k = 0, n = A.length;
 for (int i = 0; i < n; ++i) {
    if (A[i] > A[(i + 1) % n]) {
       k++;
    }
    if (k > 1) {
       return false;
    }
 }
 return true;
}
```

1. 给一个 int[][] array, 一个菱形的 radius。返回所有菱形 sum 最大的前三个数

2. 给一个字符串形式的二进制数，给一个query list，query有两种，'+'是把那个二进制数加一，'?'是在返回list中加入二进制字符串中'1'的个数

比如说，input='1110'， query=[?,+,?,+,+,?]

return [3，4，2]

3. 交错拼接字符串

```
public class mergeString {

    // Function for alternatively merging two strings
    public static String merge(String s1, String s2)
    {
        // To store the final string
        StringBuilder result = new StringBuilder();

        // For every index in the strings
        for (int i = 0; i < s1.length() || i < s2.length(); i++) {

            // First choose the ith character of the
            // first string if it exists
            if (i < s1.length())
                result.append(s1.charAt(i));
```

```
            // Then choose the ith character of the
            // second string if it exists
            if (i < s2.length())
                result.append(s2.charAt(i));
        }

        return result.toString();
    }

    // Driver code
    public static void main(String[] args)
    {
        String s1 = "geeks";
        String s2 = "forgeeks";
        System.out.println(merge(s1, s2));
    }
}
```

4. Substring divisible

给定num = 264 和 k = 1

问把num转换为string后 长度为k的substring中有多少个可以拿来整除num

这道题就有2,6,4都可以

```
import java.util.HashSet;
import java.util.Set;

   class SubStringDivisible {

    static int countSubStr(int n, int k) {
        String str = String.valueOf(n);
        int count = 0;
        long strNum = Long.parseLong(str);
        Set<String> set = new HashSet<>();
        for (int i = 0; i < str.length(); i++) {
            for (int j = i + 1; j <= str.length(); j++) {
                String sub = str.substring(i, j);
                if(sub.length() == k) {
                    long num = Long.parseLong(sub);
                    if (num != 0 && strNum % num == 0 && !set.contains(sub)) {
                        count++;
                        set.add(sub);
```

```
                }
            }
        }
    }
    return count;
}

// Driver code
public static void main(String[] args) {
    int str = 120;
    int k = 2;
    System.out.println(countSubStr(str, k));

    str = 555;
    k = 1;
    System.out.println(countSubStr(str, k));

    str= 5341;
    k = 2;
    System.out.println(countSubStr(str, k));
}
}
```

5. 装货

source = [. * .
* . .]

source 中 *代表obstacle，.代表空地

figure = [* *
* .]

figure 中 * 代表 people，.代表空地

现在允许移动source中k个obstacle，问能不能在source中装下people，obstacle的地方不能放people。

如果有多个地方能装下，请问最小的index[i, j]组合

例如题中例子

k = 2, source 可以把两个* 往右边推

这样figure就能在[0,0] 为top left的起始位置装下

6.

update and add给int[] a, int[] b, in[][] operations

两种operation:

1) [0, i, x]: update b += x
2) [1, x]: 找出a + b[j] = x的pair数量
返回int[] res, pair的数量

举例: a = [2,3], b = [1,2], operations = [[1,3], [0,1,1],[1,5]], res = [1, 0]
operation [1,3]: a = [2,3], b = [1,2], a + b[j] = 3的pair只有a[0]=2,b[0]=1, return 1
operation [0,1,1]: a = [2,3], b = [1,3] (b[1] +=1)
operation [1,5]: a = [2,3], b = [1,3], a + b[j] = 5的pair不存在, return 0

7. 给一个int array，每个value是三角形的一条边，判断每连续三个value能不能构成一个三角形。
三角形判断方法是任意两边的和大于第三边

```
import java.io.*;
import java.util.*;

/*
 * To execute Java, please define "static void main" on a class
 * named Solution.
 *
 * If you need more classes, simply define them inline.
 */

class Solution {
  public static void main(String[] args) {
    int[] A = {1, 2, 2, 5, 5, 4};
    int[] B = validTriangle(A);
    for (int i : B) {
      System.out.println(i);
    }
  }

  private static int[] validTriangle(int[] A) {
    if (A == null || A.length < 3) return new int[]{};
    int[] res = new int[A.length - 2];

    int i = 0;
    while (i < A.length - 2) {
      int a = A[i];
      int b = A[i + 1];
      int c = A[i + 2];

      if (a + b > c && b + c > a && a + c > b) {
```

```
                res[i] = 1;
            } else {
                res[i] = 0;
            }
            i ++;
        }
        return res;
    }
}
```

8. 假设你有一个键盘，上面有很多字母键是坏的。 给一个letter string，表示键盘上还能用的字母键。其他按键（数字，符号，shift, caps lock)都是好的。再给一个sentence string，问这个sentence里有几个单词可以被打印出来。大意就是先split sentence，然后每个word里每个char走一遍，注意大小写
   https://leetcode.com/discuss/interview-question/1014660/samsara-oa-codesignal-summer-internship

```
private static int brokenKeyboard(String words, char[] letters) {
    if (noLetters(words)) {
        return words.length();
    }
    int result = -1;
    String[] brokenSentence = words
            .replaceAll("\\W", " ")
            .toLowerCase()
            .split(" ");
    for (String s : brokenSentence) {
        if (doesContainLetters(s, Arrays.toString(letters))) {
            result++;
        }
    }
    return result;
}

private static boolean noLetters(String words) {
    return words.matches("[0-9\\-/@#$%^&_+=()]+");
}

private static boolean doesContainLetters(String a, String b) {
    Set<Character> chars = new HashSet<>();
    for(int i = 0; i < a.length(); i++) {
        chars.add(a.charAt(i));
    }
```

```
    for (Character character : chars) {
        if (!b.contains(character.toString())) {
            return false;
        }
    }
    return true;
}
```

相似的例题：

https://github.com/emotionless/Google_interview_question_solution/blob/master/faulty_keyboard_google.cpp

```
public class Main {

    public static void generateStrings(char [] str, int index,Set<String> dict,List<String> result){
        if(index==str.length )
        {
            String words[]= new String(str).split("\\s+");
            for(String w:words)
                if(!dict.contains(w))
                    return;
            result.add(new String(str));
            return;
        }
        generateStrings(str,index+1,dict,result);
        if(str[index]==' '){
            str[index]='e';
            generateStrings(str,index+1,dict,result);
            str[index]=' ';
        }

    }
    public static void main(String[] args) {
        String input="I lik   to   xplor   univ rs  ";
        Set<String> dict=  new HashSet<>(Arrays.asList("I","like","to", "explore", "xplore", "toe", "universe",
      "rse"));
        List<String> result= new ArrayList<>();
        generateStrings(input.toCharArray(), 0,dict,result);
        System.out.println("Result="+ result.toString());
    }
}
```

9.

具体描述不记得了，大意是 input是一个matrix和一个radius，这个radius表示一个element到中心element的euclidean distance。一个元素和自己的距离是1.然后对matrix里每一个可能的center，求关于这个center距离为radius的元素的和。大概是这样:

1,2,3,4,5 radius = 3，那么唯一一个valid的center是正中间的那个3，因为其他的元素 作为center的话，距离为3会超出边界。结果就是3+4+5+4+3+2+1+2=24

1,2,3,4,5

1,2,3,4,5

1,2,3,4,5

1,2,3,4,5

这个题弯弯绕挺多的，我用print debug法debug了很久才过了所有test

10. 给一个array和一个int diff，求以这个diff为差的最长等差数列，要求这个等差数列每一个元素都在array里。O(n^2)会超时，要做到On

参考1218. Longest Arithmetic Subsequence of Given Difference

```
class Solution {
    public int longestSubsequence(int[] arr, int diff) {
        Map<Integer, Integer> map = new HashMap<>();
        int res = 1;
        for (int num : arr) {
            int prev = map.getOrDefault(num - diff, 0);
            map.put(num, prev + 1);
            res = Math.max(map.get(num), res);
        }
        return res;
    }
}
```

11. 操作数字，把数字的每一位相乘减去数字的每一位相加

```
import java.io.*;
import java.util.*;

/*
 * To execute Java, please define "static void main" on a class
 * named Solution.
 *
 * If you need more classes, simply define them inline.
 */

class Solution {
  public static void main(String[] args) {
        int test0 = 0;
```

```
        int test1 = 5;
        int test2 = 123;
        int test3 = 12345678;

        System.out.println(helper(test0));
        System.out.println(helper(test1));
        System.out.println(helper(test2));
        System.out.println(helper(test3));

    }
    private static int helper(int test) {
        if (test == 0) return 0;
        List<Integer> list = new ArrayList<>();
        while (test > 0) {
            list.add(test % 10);
            test /= 10;
        }
        if (list.size() == 1) {
            return 0;
        }
        int times = list.get(0);
        int adds = list.get(0);

        for (int i = 1; i < list.size(); i ++) {
            times *= list.get(i);
            adds += list.get(i);
        }
        return times - adds;
    }
}
```

12.

前缀字符串：

给一个字符串数组a，正确前缀的定义是字符串数组a里面的字符串按0123...的顺序拼接在一起，比如[a, b, c]那么合理的前缀有[a, ab, abc]，再给定一个b字符串数组，问b里面所有的字符串是否符合a构成的合理前缀。

输入 a: [one, two, three] b: [onetwo, one]

输出 True

输出 a: [one, two, three] b: [onetwo, two]

输出 False

```
public static boolean isPossibleWordBreak(String s, Set<String> wordDict) {
```

```
        boolean[] dp = new boolean[s.length()+1];
        dp[0] = true;

        for(int i=1; i <= s.length(); i++){
            for(int j=i-1; j >=0 ; j--){
                dp[i] = dp[j] && wordDict.contains(s.substring(j,i));
                if(dp[i]) break;
            }
        }
        return dp[s.length()];
    }

    public static void main(String[] args) {
        String[] A = {"one", "two", "three"};
        String[] B = {"onetwo", ""};

        Set<String> wordDict = new HashSet<>();
        for(String str : A){
            wordDict.add(str);
        }

        for(String bStr : B){
            if(!isPossibleWordBreak(bStr,wordDict)) {
                System.out.println(false);
                System.exit(0);
            }
        }
        System.out.println(true);
    }
```

13. 拼积木

给定一个n*m的矩阵和五个俄罗斯方块的形状，不互相重复的往里放俄罗斯方块，每个放好的方块填满index+1，比如第一个方块就fill 1，第二个fill 2，放了最多五个方块后，返回矩阵

```
public class AlmostTetris {
    public static void main(String[] args) {
                int n = 4;
                int m = 4;
                int[] figures = {4, 2, 1, 3};
                AlmostTetris almostTetris = new AlmostTetris();
                UtilityHelper.printMatrix(almostTetris.almostTetris(n, m, figures));
    }
```

```
    int[][][] figureDimension = {{{0, 0}}, {{0, 0}, {0, 1}, {0, 2}}, {{0, 0}, {0, 1}, {1, 0}, {1, 1}}, {{0, 0}, {1, 0}, {2,
    0}, {1, 1}}, {{0, 1}, {1, 0}, {1, 1}, {1, 2}}};

    public int[][] almostTetris(int n, int m, int[] figures) {
            int[][] matrix = new int[n][m];
            int code = 1;
            for (int figure : figures) {
                    boolean figurePlaced = false;
                    for (int i = 0; i < matrix.length; i++) {
                            for (int j = 0; j < matrix[i].length; j++) {
                                    if (isPossibleAtThisPoint(matrix, figureDimension[figure - 1], i, j,
    code)) {
                                            figurePlaced = true;
                                            code++;
                                            break;
                                    }
                            }
                            if (figurePlaced) {
                                    break;
                            }
                    }
            }
            return matrix;
    }

    private boolean isPossibleAtThisPoint(int[][] matrix, int[][] fd, int x, int y, int code) {
            for (int i = 0; i < fd.length; i++) {
                    int next_x = fd[i][0] + x;
                    int next_y = fd[i][1] + y;
                    if (next_x >= 0 && next_x < matrix.length && next_y >= 0 && next_y <
    matrix[0].length) {
                            if (matrix[next_x][next_y] != 0) {
                                    return false;
                            }
                    } else {
                            return false;
                    }
            }
            for (int i = 0; i < fd.length; i++) {
                    int next_x = fd[i][0] + x;
                    int next_y = fd[i][1] + y;
                    matrix[next_x][next_y] = code;
```

```
            }
            return true;
        }
    }
```

14. 重组数组
    https://leetcode.com/discuss/interview-question/950096/uber-codesignal-oa-construct-an-array-by-concatenating-given-smaller-arrays
给定一个大数组， 和一群小数组，问大array能否通过小array拼接得到

```
class Solution {
    public boolean canFormArray(int[] arr, int[][] pieces) {
        int n = arr.length;
        // initialize hashmap
        Map<Integer, int[]> mapping = new HashMap<>();
        for (int[] p : pieces) {
            mapping.put(p[0], p);
        }

        int i = 0;
        while (i < n) {
            // find target piece
            if (!mapping.containsKey(arr[i])) {
                return false;
            }
            // check target piece
            int[] targetPiece = mapping.get(arr[i]);
            for (int x : targetPiece) {
                if (x != arr[i]) {
                    return false;
                }
                i++;
            }
        }
        return true;
    }
}
```

15. input: {"apple","pen","car","class"} ,输出首尾字母连接就行，最后一个连第一个：
    output:{"ap","pc","cc","ca"}

```
class Solution {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("apple");
        list.add("pen");
        list.add("car");
        list.add("class");
        // list.add("");
        // list.add("aa");

        List<String> res = helper(list);
        System.out.println(res);

    }

    private static List<String> helper(List<String> list) {
        List<String> res = new ArrayList<>();
        if (list == null | list.size() == 0) return res;
        if (list.size() == 1) {
            char[] array = list.get(0).toCharArray();
            res.add(String.valueOf(array[0]));
            return res;
        }
        for (int i = 0; i < list.size() - 1; i ++) {
            StringBuilder sb = new StringBuilder();
            if (list.get(i) == null || list.get(i).length() == 0) {

                sb.append("");
            } else {
                char[] first = list.get(i).toCharArray();
                sb.append(first[0]);
            }
            if (list.get(i + 1) == null || list.get(i + 1).length() == 0) {

                sb.append("");
            } else {
                char[] second = list.get(i + 1).toCharArray();
                sb.append(second[0]);
            }
            res.add(sb.toString());
```

```
        }
        StringBuilder sb = new StringBuilder();
        if (list.get(list.size() - 1) == null || list.get(list.size() - 1).length() == 0) {

            sb.append("");
        } else {
            char[] second = list.get(list.size() - 1).toCharArray();
            sb.append(second[0]);
        }

        if (list.get(0) == null || list.get(0).length() == 0) {
            sb.append("");
        } else {
            char[] first = list.get(0).toCharArray();
            sb.append(first[0]);
        }

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

16. input: abababababac , {ab, abab, abc}问对应的String在大字符里出现了几次，不考虑重叠情况， output {4, 2, 0}

```
class Solution {
    public static void main(String[] args) {
        String text = "abababababac";
        List<String> array = new ArrayList<>();
        array.add("ab");
        array.add("abab");
        array.add("abc");

        List<Integer> res = helper(text, array);
        System.out.println(res);
    }

    private static List<Integer> helper(String text, List<String> list) {
        List<Integer> res = new ArrayList<>();
        if (text == null || text.length() == 0 ||list == null || list.size() == 0) return res;
```

```
            for (String s : list) {
                int count = countMatches(text, s);
                res.add(count);
            }
            return res;
        }

        private static int countMatches(String text, String s) {
            if (s == null || s.length() == 0) return 0;
            return text.split(s, -1).length - 1;
        }
    }
```

17. 得二维矩阵带传送门的题，问你能不能从起始点走到终点，思路比较直接，把传送点坐标都存在map里，障碍物坐标也存下来，每走一步判断一下是否遇到障碍物或者开始传送，再把visited的坐标也记下来就行
    https://leetcode.com/discuss/interview-question/1147715/uber-coding-questions-oa
18.

也是地里出现过好几次的题，input: nums {1,3,5,7,2}, ops = {{2},{0,1},{1,2},{2}}。因为只有操作2有输出值，所有output就是2的结果数列

{0, x}：操作0，将数组中的每个元素加上x
{1, x}：操作1，append x to 数组
{2}: 操作2, 返回当前数组中的最小值

19. count occurance 数从0到n中，digit 0，2，4 一共出现了多少次。例如n=22， 0 出现了（0，10，20）3次，2出现了（2，12，20，21，22）6次，4出现了（4，14）2次，所以一共是11次
    https://leetcode.com/discuss/interview-question/1113153/Uber-OA-2021-CodeSignal/880331

```
    import java.io.*;
    import java.util.*;
    class Solution {
        public static int countAllOcurrences(int n) {
            int result = 0;

            while(n>0) {
                int reminder = n % 10;
```

```
                switch(reminder) {
                    case 0:result++;
                    break;
                    case 2:result++;
                    break;
                    case 4:result++;
                    break;
                    default:
                }
                n = n/10;
            }
            return result;
        }
        public static int numberOfsinRange(int n)
        {
            int count = 1;
            for(int i = 0; i <= n; i++) {
                count += countAllOcurrences(i);
            }
            return count;
        }

        public static void main(String[] args) {
            int total = numberOfsinRange(0);
            System.out.println(total);

        }
    }
```

20. moving diagonally。一个 n*m的矩阵，start from（x1，y1）一开始以step（dx=1，dy=1）移动，如果x方向移动出了矩阵，step变成（dx=-dx，dy=dy）并且回到移动出矩阵前的位置以新的step方向继续移动。如果y方向移动出了矩阵，step变成（dx=dx，dy=-dy）并且回到移动出矩阵前的位置以新的step方向继续移动。如果x，y方向同时移动出了矩阵，step变成（dx=-dx，dy=-dy）并且回到移动出矩阵前的位置以新的step方向继续移动。要求输出以这种移动方式需要多少step可以移动到（x2，y2），如果无法到达（x2，y2），输出-1（感觉是其实最短路径的变种，是不是这样？）用一个4维数组记录是否已经visit 某个位置以及移动

```
public class Solution {
    /**
     * @param grid: a chessboard included 0 (false) and 1 (true)
     * @param source: a point
```

```
     * @param destination: a point
     * @return: the shortest path
     */
    public int shortestPath(boolean[][] grid, Point source, Point destination) {
        // write your code here
        if (grid == null || grid.length == 0 || grid[0].length == 0) return -1;
        if (source.x == destination.x && source.y == destination.y) return 0;

        int col = grid.length;
        int row = grid[0].length;
        int res = 0;
        Queue<Point> queue = new ArrayDeque<>();
        boolean[][] visited = new boolean[col][row];

        queue.offer(source);
        visited[source.x][source.y] = true;

        int[] dx = {1, 1, -1, -1, 1, 1, -1, -1};
        int[] dy = {1, -1, 1, -1, 1, -1, 1, -1};

        while (!queue.isEmpty()) {
            int size = queue.size();
            res ++;
            for (int level = 0; level < size; level ++) {
                Point current = queue.poll();
                for (int i = 0; i < 8; i ++) {
                    int nx = current.x + dx[i];
                    int ny = current.y + dy[i];
                    if (isValid(nx, ny, grid) && !visited[nx][ny]) {
                        if (nx == destination.x && ny == destination.y) return res;
                        visited[nx][ny] = true;
                        queue.offer(new Point(nx, ny));
                    }
                }
            }
        }
        return -1;
    }
    private boolean isValid(int x, int y, boolean[][] grid) {
        return 0 <= x && x < grid.length && 0 <= y && y < grid[0].length && !grid[x][y];
```

```
    }
}
```

21. unique on segment。给一个数组和一个整数k，例如arr= [1,2,3,4,2], k=3. 求有多少的subarr可以满足subarr中的数字都是unique的且至少有k个。[1,2,3], [1,2,3,4], [2,3,4], [3,4,2], 一共有4个subarray

https://www.lintcode.com/problem/substring-with-at-least-k-distinct-characters/

https://leetcode.com/problems/subarrays-with-k-different-integers/

22. timestamp hit limit

    给一个滑窗, maxCount=5, limit=3

    输入为timestamp数组

    返回每个timestamp为true/false

    如果某个timestamp在过去的limit秒内 出现的次数超过maxCount 返回false, 否则true

    用的hit limit的 queue做的

    while(q不空且q.peek()-timestamp>limit)q.poll

    结果一直不对 7个case过了4个

    http://blog.gssxgss.me/not-a-simple-problem-rate-limiting/

```
    import java.util.HashMap;
    import java.util.LinkedList;
    import java.util.Queue;

    public class RateLimiter {
        public int REQUEST_LIMIT = 100;
        public Long TIME_LIMIT = 1000L;
        public class HitCounter {
            public Queue<Long> queue;
            public HitCounter() {
                queue = new LinkedList<>();
            }
            public boolean hit(long timestamp) {
                /* when a timestamp hit, we should poll all the timestamp before
    TIME_LIMIT*/
                while (!queue.isEmpty() && queue.peek() - timestamp >= TIME_LIMIT)
    queue.poll();
                if (queue.size() < 500) {
```

```
                queue.add(timestamp); return true;
            }
            return false;
        }
    }
    public HashMap<String, HitCounter> clientTimeStampMap = new HashMap<>();
    public boolean isAllow(String clientId) {
        long currTime = System.currentTimeMillis();
        if (!clientTimeStampMap.containsKey(clientId)) {
            HitCounter h = new HitCounter();
            h.hit(currTime); return true;
        } else {
            HitCounter h = clientTimeStampMap.get(clientId);
            return h.hit(currTime);
        }
    }

}
```

23.

给一个array 需要查看 三个条件 1) a ^ 2 + a[i + 1] ^2 = a[i + 2] ^ 2

2）a ^ 2 + a[i + 2] ^2 = a[i + 1] ^ 2

3）a[i+1] ^ 2 + a[i + 2] ^2 = a[i] ^ 2

如果满足上面三个条件就返回1 不然就是0， 返回类型是个 int array

example: [1,2, 5, 5, 0] 返回 [1, 0, 1]

24.

地里有过的面经， 就是给一个字符串string，里面只有 "W" "D" "L" 需要按照以下顺序排列

1）如果string 有“W”, 加到result 里，输入的string remove这个“W” (每次只能remove一个)

2）如果string 有“D”, 加到result 里，输入的string remove这个“D” (每次只能remove一个)

3）如果string 有“L”, 加到result 里，输入的string remove这个“L” (每次只能remove一个)

4） 如果输入字符串没有 "W" "D" "L" ， return result 不然从step 1重复

example: "LDWDL" -> 输出 "WDLDL"

25.

输入一个2d string array 里面每个 string array都有 文字， 再个一个 width 需要输出一个 center align的String array（解释比较复杂，看example）类似lc 68
输出[["Hello", "World"], ["You"], ["Are", "beautiful", "U", "canDoThis"]] width = 16
输出　　[["******************"],
　　["* Hello World   *"].
　　["*　　You　 *"],
　　["*Are beautiful U *"],
　　["*　 canDoThis *"],
　　["******************"]]
每个string array都算一行，如果里面的单词length 加上空格多于width就得新启一行 “*”不计算在width内
https://leetcode.com/problems/text-justification/submissions/

26.
输入 int array a， int array b, 和一个 2d int array　querry， 在querry里会有两种情况，
1）[1, x] 操作是求 a + b[j] = x 有几种可能性
2）[0, i , x] 操作时 a = x 替换array a的数值
返回一个int array 包含在querry里所有求sum的解（解释不清，看例子）

input a[1, 2, 2] b[2, 3] querry[[1, 4], [0, 0 , 3], [1, 5]]
output [3, 3]

querry先[1, 4] a + b[j] = 4 有三种可能性，（1，3），（2，2），（2，2）

[0, 0, 3] 会把 a[0] = 3 然后 array a就变成了 a[3, 2, 2]

[1, 5]　 a + b[j] = 5　有三种可能性　（3，2），（2，3），（2，3）

27.
update and add给int[] a, int[] b, in[][] operations
[hide=168]
两种operation:
1) [0, i, x]: update b += x
2) [1, x]: 找出a + b[j] = x的pair数量
返回int[] res, pair的数量

举例: a = [2,3], b = [1,2], operations = [[1,3], [0,1,1],[1,5]], res = [1, 0]
operation [1,3]: a = [2,3], b = [1,2], a + b[j] = 3的pair只有a[0]=2,b[0]=1, return 1
operation [0,1,1]: a = [2,3], b = [1,3] (b[1] +=1)
operation [1,5]: a = [2,3], b = [1,3], a + b[j] = 5的pair不存在, return 0

C++

```
vector<bool> rectangleBoxes(vector<vector<int>>& vec) {

    vector<bool> ret;
    vector<vector<int>> zero, one;

    for(int i = 0; i < vec.size(); i++) {
        for (auto it = vec[i].begin(); ;) {
            if(*it == 0) {
                zero.push_back(vec[i]);
                break;
            }
            else
            {
                one.push_back(vec[i]);
                break;
            }
        }
    }

    if(!one.size())
        return ret;



    bool temp = false;


    for (size_t i = 0; i < one.size(); ++i) {

        if(!zero.size())
            temp = true;

        for (size_t j = 0; j < zero.size(); ++j) {
            if ((zero[j][1] <= one[i][1] && zero[j][2] <= one[i][2]) ||
                (zero[j][1] <= one[i][2] && zero[j][2] <= one[i][1])) {
                    temp = true;
            } else {
                    temp = false;
                    break;
            }
        }
        ret.push_back(temp);
    }
```

```
    return ret;
}
int main()
{
    cout<<"Hello World \n";
    vector<vector<int>> vec{{0, 1, 3}, {0, 4, 2}, {1, 3, 4}, {1, 3, 2}};
    vector<bool> ret;
    ret = rectangleBoxes(vec);
    for (size_t i = 0; i < ret.size(); ++i) {
        cout << ret[i] << " ";
    }
    cout << endl << ret.size();

    vector<vector<int>> vec1{{1,1,1}};
   // vector<bool> ret;
    ret = rectangleBoxes(vec1);
    for (size_t i = 0; i < ret.size(); ++i) {
        cout << ret[i] << " ";
    }
    cout << endl << ret.size();



    return 0;
}
```

28. 盖房子，输出最大连续房子的个数。并查集，不需要压缩路径也能过。
就是leetcode 128原题

Longest Consecutive Sequence

```
class Solution {
    public int longestConsecutive(int[] nums) {
        Set<Integer> num_set = new HashSet<Integer>();
        for (int num : nums) {
            num_set.add(num);
        }

        int longestStreak = 0;

        for (int num : num_set) {
            if (!num_set.contains(num-1)) {
```

```
                int currentNum = num;
                int currentStreak = 1;

                while (num_set.contains(currentNum+1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }

                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }

        return longestStreak;
    }
}
```

29. memory allocator

给一个表示memory状态的数组，里面只有0和1，0表示free，1表示used
有两个操作，
{0, x}，0表示分配内存，x是长度，要在当前memory中找到第一个满足长度x的free memory，将这块memory设置为1，并返回该内存块的起始位置（free时要用到这个位置）
{1, index}：1表示释放内存，index就是之前alloc时记录下来的“起始位置”，并把该内存块设置为0
比如初始memory状态的数组：{0, 0, 0, 1, 1, 0, 0, 0, 0}
操作序列：{0, 2}, {0, 3}, {1,0}
{0,2}表示分配长度为2的内存
{0,3}表示分配长度为3的内存
{1,0}表示释放之前分配的"起始位置"为0的内存

30. n-1个array 的neighbor pair，要求reconstruct 出原本的array
31. 给一个矩形，对每一圈进行排序。类似利口的旋转矩阵

```
class Solution {
    public static void main(String[] args) {
```

```
    int[][] test = { { 1, 2, 3, 4, 0 },
                     { 1, 1, 1, 1, 2 },
                     { 1, 2, 2, 2, 4 },
                     { 1, 9, 3, 1, 7 },
                     { 1, 8, 4, 2, 9 }};
    List<List<Integer>> res = helper(test);//按层遍历放到list里
    int n = test.length;
    int m = test[0].length;

    int[][] array = new int[n][m];//建一个1d数组
    addBackToArray(array, res);//数组重新填回矩阵
    for (int i = 0; i < array.length; i ++) {
        for (int j = 0; j < array[0].length; j ++) {
            System.out.print(array[i][j] + " ");
        }
        System.out.println("");
    }
}
private static List<List<Integer>> helper(int[][] matrix) {



    List<List<Integer>> res = new ArrayList<>();
    if (matrix.length == 0) return res;

    int r1 = 0, r2 = matrix.length - 1;
    int c1 = 0, c2 = matrix[0].length - 1;

    while (r1 <= r2 && c1 <= c2) {
        List<Integer> ans = new ArrayList<>();
        for (int c = c1; c <= c2; c++) {
            ans.add(matrix[r1][c]);

        }

        for (int r = r1 + 1; r <= r2; r++){
            ans.add(matrix[r][c2]);

        }
```

```
            if (r1 < r2 && c1 < c2) {
                for (int c = c2 - 1; c > c1; c--) {
                    ans.add(matrix[r2][c]);

                }
                for (int r = r2; r > r1; r--) {
                    ans.add(matrix[r][c1]);

                }

            }
            res.add(new ArrayList<>(ans));
            r1++;
            r2--;
            c1++;
            c2--;

        }
        return res;
    }

    private static void addBackToArray(int[][] mat, List<List<Integer>> lists) {
        int[] arr = new int[mat.length * mat[0].length];

        int M = mat.length;
        int N = mat[0].length;
        List<Integer> temp = new ArrayList<>();
        for (List<Integer> list : lists) {
            Collections.sort(list);
        }
        for (List<Integer> list : lists) {
            for (int i : list) {
                temp.add(i);
            }
        }
        for (int i = 0; i < temp.size(); i ++) {
            arr[i] = temp.get(i);
        }

        int top = 0, bottom = M - 1;
```

```
int left = 0, right = N - 1;

int index = 0;

while (true)
{
    if (left > right) {
        break;
    }

    // print top row
    for (int i = left; i <= right; i++) {
        mat[top][i] = arr[index++];
    }
    top++;

    if (top > bottom) {
        break;
    }

    // print right column
    for (int i = top; i <= bottom; i++) {
        mat[i][right] = arr[index++];
    }
    right--;

    if (left > right) {
        break;
    }

    // print bottom row
    for (int i = right; i >= left; i--) {
        mat[bottom][i] = arr[index++];
    }
    bottom--;

    if (top > bottom) {
        break;
    }
```

```
            // print left column
            for (int i = bottom; i >= top; i--) {
                mat[i][left] = arr[index++];
            }
            left++;
        }

    }

}

class Solution {
    public List < Integer > spiralOrder(int[][] matrix) {
        List ans = new ArrayList();
        if (matrix.length == 0)
            return ans;
        int r1 = 0, r2 = matrix.length - 1;
        int c1 = 0, c2 = matrix[0].length - 1;
        while (r1 <= r2 && c1 <= c2) {
            for (int c = c1; c <= c2; c++) ans.add(matrix[r1][c]);
            for (int r = r1 + 1; r <= r2; r++) ans.add(matrix[r][c2]);
            if (r1 < r2 && c1 < c2) {
                for (int c = c2 - 1; c > c1; c--) ans.add(matrix[r2][c]);
                for (int r = r2; r > r1; r--) ans.add(matrix[r][c1]);
            }
            r1++;
            r2--;
            c1++;
            c2--;
        }
        return ans;
    }
}
```

重新填回去https://www.techiedelight.com/create-spiral-matrix-given-array/

32.

字符串处理，模拟textx, 有TYPE, Select, move_cursor, undo操作

例子:(右边表示结果，| 表示cursor)

TYPE SIGNAL -> SIGNAL|
TYPE CODE -> SIGNALCODE|
MOVE_CURSOR -3 -> SIGNALC|ODE
SELECT 5 8 -> SIGNALCO|DE
TYPE IN -> SIGNAIN|DE
UNDO -> SIGNALCO|DE

https://leetcode.com/discuss/interview-question/860501/text-editor-implementation
https://leetcode.com/discuss/interview-question/304048/google-onsite-interview-design-text-editor
这两个需要结合起来，用stack 把光标标记位置

33. 给一个矩形，对每一圈进行排序。类似利口的旋转矩阵
34. 按照峰顶删除数字，给出删除数字的顺序

```
public int[] minpeaks(int[] arr) {
    int n = arr.length;
    int[] result = new int[n];
    List<Integer> list=new ArrayList<>();
    for(int i:arr)
        list.add(i);

    for(int i=0;i<n;i++){                          //O(N)
        int min=Integer.MAX_VALUE;
        int index=-1;
        int size=list.size();
        for(int j=0;j<size;j++){                   //O(N)
            if(j==0 && j+1<size) {
                if (list.get(j) > list.get(j + 1) && min > list.get(j)) {
                    min = list.get(j);
                    index = j;
                }
            }else if(j==size-1 && j-1>=0) {
                if (list.get(j) > list.get(j - 1) && min > list.get(j)) {
                    min = list.get(j);
                    index = j;
                }
            }else if(size==1){
                min=list.get(j);
                index=j;
            }else if(list.get(j)>list.get(j-1) && list.get(j)>list.get(j+1) && min>list.get(j)) {
                min=list.get(j);
                index=j;
            }
```

```
            }
            list.remove(index);                                 //O(N)
            result[i]=min;
        }



        return result;
    }


    private static List<Integer> minPeaks(List<Integer> list) {
        ArrayList<Integer> result = new ArrayList<>();
        if (list == null || list.size() == 0) return result;
        int n = list.size();
        for (int i = 0; i < n; i++) {
             int min = Integer.MAX_VALUE;
            int index = -1;
            int size = list.size();

            for (int j = 0; j < size; j++) {
                if (j == 0 && j + 1 < size) {
                    if (list.get(j) > list.get(j + 1)
                        && min > list.get(j)) {
                        min = list.get(j);
                        index = j;
                    }
                } else if (j == size - 1 && j - 1 >= 0) {
                    if (list.get(j) > list.get(j - 1) && min > list.get(j)) {
                        min = list.get(j);
                        index = j;
                    }
                } else if (size == 1) {

                    min = list.get(j);
                    index = j;
                } else if (list.get(j) > list.get(j - 1)
                        && list.get(j) > list.get(j + 1)
                        && min > list.get(j)) {
```

```
                min = list.get(j);
                index = j;
            }
        }

        list.remove(index);
        result.add(min);
    }
    return result;
  }
class Num{
    int num;
    int idx;
    int left;
    int right;
    public Num(int num, int idx, int left, int right){
        this.num = num;
        this.idx = idx;
        this.left = left;
        this.right = right;
    }
  }
  public ArrayList<Integer> getMinPeakElements(int[] arr){
    HashMap<Integer, Num> map = new HashMap<>();
    PriorityQueue<Num> minHeap = this.populateHeap(arr, map);
    ArrayList<Integer> res = new ArrayList<>();
    while(!minHeap.isEmpty()){
        Num n = minHeap.poll();
        res.add(n.num);
        Num left = map.get(n.left);
        Num right = map.get(n.right);
        if(left!=null) left.right = n.right;
        if(right!= null) right.left = n.left;
        this.addIfPeak(left, arr, minHeap);
        this.addIfPeak(right, arr, minHeap);
    }
    return res;
  }
  private void addIfPeak(Num n, int[] arr, PriorityQueue<Num> minHeap){
    if(n==null) return;
    int leftVal = (n.left>=0)? arr[n.left]:Integer.MIN_VALUE;
    int rightVal = (n.right<arr.length)?arr[n.right]:Integer.MIN_VALUE;
    if(leftVal<n.num && rightVal<n.num) minHeap.add(n);
```

```
    }
    private PriorityQueue<Num> populateHeap(int[] arr, HashMap<Integer, Num> map){
        PriorityQueue<Num> minHeap = new PriorityQueue<>(new Comparator<Num>(){
            public int compare(Num a, Num b){ return a.num - b.num; }
        });
        for(int i=0; i<arr.length; i++){
            Num n = new Num(arr[i], i, i-1, i+1);
            map.put(i, n);
            this.addIfPeak(n, arr, minHeap);
        }
        return minHeap;
    }
```

35. 移动矩阵，给定一个大的方形矩阵，先按照size分割，每一个小的方形矩阵找到里面第一个missing positive，然后再按照missing positive的顺序移动矩阵得到一个新的矩阵。这个题目完全就是在考你是不是能细心的处理好矩阵的位置和坐标，处理好之后排序什么的其实不难。找missing positive其实不需要用刷题网那个方法，就直接数一遍就好

```
class Solution {
    public int[][] beautyOfMatrix(int[][] matrix, int k) {
        Map<Integer, int[]> cells = new HashMap<>();

        //store grids into arrays keyed by their grid id
        TreeMap<Integer, List<Integer>> map = new TreeMap<>();
        int n = matrix.length;
        for(int i =0; i<n; i++){
            for(int j =0; j<n ; j++){
                int r = k * (i/k);
                int c = k * (j/k);
                int key = r*n + c;
                cells.put(key, new int[]{r,c});
                if(!map.containsKey(key)) map.put(key, new ArrayList<>());
                map.get(key).add(matrix[i][j]);
            }
        }


        //calculate beauty of each grid
        TreeMap<Integer, List<Integer>> beauty = new TreeMap<>();
```

```
    for(Integer kS : map.keySet()){
        int b = getPositive(map.get(kS));
        if(!beauty.containsKey(b)) beauty.put(b, new ArrayList<>());
        beauty.get(b).add(kS);
    }

    //sorting beauty to access them in order
    for(List kS : map.values()){
      Collections.sort(kS);
    }

    int[][] res = new int[n][n]; //resultant array to return

     int r1 = 0; int c1 = 0;
     for(List<Integer> l : beauty.values()){
         for(int idx : l){
             int row = cells.get(idx)[0];
             int col = cells.get(idx)[1];
             for(int i = row,  rk= 0; i<row+k; i++, rk++){
                for(int j = col,  ck = 0; j<col+k; j++, ck++){
                    res[k*(r1/k) + rk][k*(c1/k) + ck] = matrix[i][j];
                }
             }
            if(c1 + k < n)
             {
                c1+=k;
             }
             else{
                r1+=k;
                c1=0;
             }
         }
    }

    return res;
}

/* function to get first positive */
private int getPositive(List<Integer> list){
    int num = 1;
```

```
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for(int i: list){
            pq.offer(i);
        }
Σ
        while(!pq.isEmpty()){
            int val = pq.poll();
            if (val<=0) continue;
            if(val > num) break;
            else num++;
        }

        return num;
    }
}
```

36.

给定一个不重复数字的数组，找出数组里面所有组合等于2的幂（1，2，4，8，...）自己可以和自己组合。

这个题目我就先排个序然后找到最大的数字*2得到上限，再简单的用了hashmap把所有数字放进去了，然后从start开始乘以二开始找可能的组合，超过上限就停止

hashmap竟然可以把所有数字都放进去，看来测试用例给的数字数量也不是很多。

37.

Add up two strings:

input: String a, String b

output: 从右向左，a 与b中每一个相对应的pair相加，得到的结果转化成string，concat 起来。注意这里不需要进位，单纯concat在一起就好 (output不太好描述，下面举例说明)

Example: a="1234", b="56" => Output: "12810"。

```
class Solution {
    public static void main(String[] args) {
        String a = "56";
        String b = "1234";

        String res = addStrings(a, b);
        System.out.println(res);
    }

    private static String addStrings(String a, String b) {
```

```
    if (a == null && b == null || a.length() == 0 && b.length() == 0) return "";
    if (a == null || a.length() == 0) return b;
    if (b == null || b.length() == 0) return a;

    char[] arrayA = a.toCharArray();
    char[] arrayB = b.toCharArray();
    List<Integer> list = new ArrayList<>();

    reverseArray(arrayA);
    reverseArray(arrayB);

    StringBuilder sb = new StringBuilder();

    if (arrayA.length >= arrayB.length) {
        addNumber(arrayA, arrayB, list);
        addEnd(arrayA, arrayA.length - arrayB.length, list);
    } else {
        addNumber(arrayB, arrayA, list);
        addEnd(arrayB, arrayB.length - arrayA.length, list);
    }


    Collections.reverse(list);
    for (int number : list) {
        sb.append(number + "");
    }
    return sb.toString();
}

private static void addNumber(char[] a, char[] b, List<Integer> list) {
    int sum = 0;
    for (int i = 0; i < b.length; i ++) {
        int number1 = Integer.parseInt(String.valueOf(a[i]));
        int number2 = Integer.parseInt(String.valueOf(b[i]));

        sum = number1 + number2;
        list.add(sum);
    }
}
```

```
    private static void addEnd(char[] a, int diffLen, List<Integer> list) {
        for (int i = a.length - diffLen; i < a.length; i ++) {

            list.add(Integer.parseInt(String.valueOf(a[i])));
        }
    }

    private static void reverseArray(char[] array) {
        for (int i = 0; i < array.length / 2; i ++) {
            char temp = array[i];
            array[i] = array[array.length - 1 - i];
            array[array.length - 1 - i] = temp;
        }
    }
}
```

38.

Matrix manipulation

input: 2-D array with shape 4x(4*n). 每个4*4，non-overlapping 的为一个block。每个block里会有一个'?' . 每个block出现的所有数（除了问号）set(1 ~ 16), missing one number to fill in '?'.

Goal: find the missing element and fill in '?', and re-organize the blocks so that it's sorted by this missing value. If two blocks have the same missing value, then keep them in their original relevant order.

Output: Still a 4x(4*n) 2-D Array.

Example input:

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

Output:

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

39.

data structure

input: a: List of int ,

b: List of int, query: list of operations.

there are two types of operations: [0, i, x] and [1, x].

Type 0: add x to a.
Type 1: return number of combinations such that a + b[j] = x.
output: a list of int, each item is the result of each Type 1 operation.
example: input [1,2,2], [3,2], [[1,4], [0,0,1], [1,5]]. => [3, 3]

前两道单纯的string manipulation，没有任何data structure 或algorithm
第三道考array manipulation，以及要怎么sort
第四题考一点点data structure。要思考怎么储存a and b 来让Type 1 operation 更简单。

我做了50min吧。中间有internet connection issue，换了个电脑重新login 花了点时间。（之前的部分代码还丢了 哭哭）
整体做下来不算难。很多地理原题。

40. 判断两个字符串是否相似 相似的定义是可以
    1. 任意swap character
    2. 任意把一种character的所有appearance和另外一个character的appearance呼唤

```
class Solution {

    public boolean closeStrings(String word1, String word2) {
        if (word1.length() != word2.length()) {
            return false;
        }
        int word1Map[] = new int[26];
        int word2Map[] = new int[26];
        for (char c : word1.toCharArray()) {
            word1Map[c - 'a']++;
        }
        for (char c : word2.toCharArray()) {
            word2Map[c - 'a']++;
        }
        for (int i = 0; i < 26; i++) {
            if ((word1Map[i] == 0 && word2Map[i] > 0) ||
                (word2Map[i] == 0 && word1Map[i] > 0)) {
                return false;
            }
        }
        Arrays.sort(word1Map);
        Arrays.sort(word2Map);
        return Arrays.equals(word1Map, word2Map);
```

```
    }
  }
```

41. 两个长度一样的数组 求满足a[i] - b[j] = a[j] - b[i] 的pair的个数 只能o(n)才能过所有的testcase

    然后从0到n-1扫一遍 获得每个i的两个array的和 看每个和有多少个frequency就能得出一共有多少个pair了

42. 输入为一个长度为n的字符串数组words，输出为一个长度为n-1的布尔数组，如果words和words[i+1]的开始和结束字母相同，输出的第i位为真。
43.

You are given a matrix of characters representing a big board. Each cell of the matrix contains one of three characters:
'.', which means that the cell is empty;
'*', which means that the cell contains an obstacle;
'#', which means that the cell contains a small box.

first step: 把 “#”box全部推到最右边
second step: 把 “#”box全部推到最下边
返回最终的状态

example1:

```
board= [['#', '#', '.', '.', '.', '.', '.'],
        ['#', '#', '#', '.', '.', '.', '.'],
        ['#', '#', '#', '.', '.', '#', '.']]

output board = [['.', '.', '.', '.', '.', '#', '#'],
                ['.', '.', '.', '.', '#', '#', '#'],
                ['.', '.', '.', '#', '#', '#', '#']]
```

example2:

```
board= [['#', '.', '.', '.', '*', '.', '.'],
        ['.', '#', '.', '*', '.', '#', '.'],
```

['#', '*', '#', '.', '.', '#', '.']]

output board = [['.', '.', '.', '#', '*', '.', '.'],
['.', '.', '.', '*', '.', '.', '#'],
['#', '*', '#', '.', '.', '#', '#']]

我分了两步，第一步是处理往右推：每一行从右往左处理，如果当前是箱子而且能往右推，就推到最右，然后当前位改成'.'。第二部是处理往下推，和第一步类似。

44. 给一个最多十万长度的二进制数，有两种操作，一种是减一，一种是求当前的数里面1的个数，比如101就是2 。需要支持最多十万个操作

```
public int hammingWeight(int n) {
    int sum = 0;
    while (n != 0) {
        sum++;
        n &= (n - 1);
    }
    return sum;
}
```

45. 给一个int，让你交叉算和与差然后返回值。比如12345 -> 1-2+3-4+5 = 3 返回3。5243 -> 5-2+4-3 = 4 返回4。从左到右一定是奇数位算加法偶数位算减法

```
private static int helper(int num) {
    if (num == 0) return 0;
    List<Integer> list = new ArrayList<>();
    while (num > 0) {
        list.add(num % 10);
        num /= 10;
    }
    Collections.reverse(list);
    int sum = 0;
    int diff = 0;
    for (int i = 0; i < list.size(); i ++) {
        if (i % 2 == 0) {
            sum += list.get(i);
        } else {
            diff += list.get(i);
        }
    }
```

```
        return sum - diff;
    }
```

46. 给你一个int和k值，让你把这个int的位数分成k份，分别求值。不足k的位数直接放在后面。一直到你这个值位数小于等于k。比如 1111122222，k=3，那你要111 112 222 2 -> 3462 -> 346 2 -> 132 返回132

```
class TestIntK:
    def test(self):
        n = 10000000001
        k = 3
        self.run(n,k)

    def run(self, n, k):
        a = []
        while n > 0:
            a.append(n%10)
            n //= 10
        a.reverse()

        while len(a) > k:
            print(a)
            index = 0
            b = []
            while index < len(a):
                if index + k > len(a):
                    b += a[index:]
                    break
                else:
                    count = 0
                    s = 0
                    while count < k:
                        s += a[index]
                        index += 1
                        count += 1
                    if s <= 9:
                        b.append(s)
                    else:
                        temp = []
                        while s > 0:
```

```
                temp.append(s%10)
                s //= 10
            b += reversed(temp)
        a = b

    result = 0
    for x in a:
        result = result*10 + x

    print(result)
```

47.
提供一个一维坐标objects ex. [1,2,3,5,7,9], 和一个路灯能照到的radius， 返回一个坐标使得radius范围内照射到的objects最多。
这题我先用画出从最小值到最大值的数组，使用sliding window， hidden case超时。 然后用binary search全过。

48.
反转数字求和，末尾0 保留，
a=[123,210,106]
return 321+120+601=1042

```
int num = 1234567, reversed = 0;

  for(;num != 0; num /= 10) {
    int digit = num % 10;
    reversed = reversed * 10 + digit;
  }
```

49. 第三题是老题， 题干很长， 大意是给一个matrix, 从左下开始斜着print， 每组print的string要填充至matrix的长度。 最后找到每组string排序后的index, return

```
[ ['a','b'],
  ['c', d']]
-> [['c'], ['a','b'], ['b']]
-> ['cc', 'ab', 'bb'] # 如果sort, ['ab', 'bb', 'cc']
-> [3, 1, 2]
```

50. 给一个 int[] array, 每次可以给一个数加 1，经过若干次操作后保证每个数只出现一次，返回至少需要多少次操作
leetcode 945

```
class Solution {
```

```
    public int minIncrementForUnique(int[] A) {
        Arrays.sort(A);
        int ans = 0, taken = 0;

        for (int i = 1; i < A.length; ++i) {
            if (A[i-1] == A[i]) {
                taken++;
                ans -= A[i];
            } else {
                int give = Math.min(taken, A[i] - A[i-1] - 1);
                ans += give * (give + 1) / 2 + give * A[i-1];
                taken -= give;
            }
        }

        if (A.length > 0)
            ans += taken * (taken + 1) / 2 + taken * A[A.length - 1];

        return ans;
    }
}
```

51. 给一个 int[][] array, 一个菱形的 radius。返回所有菱形 sum 最大的前三个数

```
  1
1 1 1    radius = 2
  1


   1
  1 1 1
1 1 1 1 1     radius = 3
  1 1 1
    1
```

52. 给你两个字符串，string binaryString，string request 长度可到10^5。 binaryString中只有'0'和'1'，request中只有 '-' 和 '?'。现在要求遍历request, 如果发现当前字符是'-'，则binaryString所表示的正整数要减去1, 如果当前字符是'?' 求当前binaryString所表示的正整数的二进制位为1的个数. 最后返回一个数组, 按request中出现？的顺序返回当前binaryString所表示的正整数中二进制位为1的个数。

```
vector<int> solve(string &binaryString, string &request)
{
    vector<vector<int>> ones_interval;
```

```
    int n = binaryString.size();
    int total_ones = 0;

    /*初始化 O(N), ones_interval存的是‘1’的{start index, count}
     for example: 10011
     ones_interval = {{0,1}, {3,2}}
     */
    for (int i = 0; i < n; i++) {
        if (binaryString[i] == '1') {
            if (i == 0 || binaryString[i-1] == '0')
                ones_interval.push_back({i, 0});
            ones_interval.back()[1]++;

            total_ones++;
        }
    }

    vector<int> result;
    for (char c : request) {
        if (c == '?') //'?' 操作 O(1)
            result.push_back(total_ones);
        else { //'-' 操作 O(1)
            if (ones_interval.empty()) { //如果binaryString全是0了，那'-'操作将binaryString
全置为1
                ones_interval.push_back({0, n});
                total_ones = n;
                continue;
            }
            auto &itv = ones_interval.back();
            int idx = itv[0] + itv[1];
            int new_ones = n - idx;
            if (--itv[1] == 0)
                ones_interval.pop_back();
            if (new_ones > 0)
                ones_interval.push_back({idx, new_ones});
            total_ones = total_ones - 1 + new_ones;
        }
    }

    return result;
```

```
    }
```

53. sortChessSubsquares

这题input是个二维数组，这个二维数组在图上面展示出来类似于黑白棋，最左上角的点是黑色的，与其相邻的点是白色，以此类推黑色和白色相互交错

然后又给了数组类似于[[0,1,2],[0,2,3]]， 意思是其中[0,1,2] 表示最左上角以（0，1）为起点的2*2的矩阵，矩阵中的黑白棋子按照各自的颜色分组进行排序，

进行了两次排序后输出得到的数组

```
    void sortChessSubsquares(vector<vector<int>> &matrix, vector<int> &sub)
{
    int rows = matrix.size(), cols = matrix[0].size();
    int x1 = sub[0], y1 = sub[1], k = sub[2];
    int x2 = x1 + k - 1, y2 = y1 + k - 1;
    x2 = min(x2, rows-1);
    y2 = min(y2, cols-1);

    vector<int> black, white;
    for (int x = x1; x <= x2; x++) {
        for (int y = y1; y <= y2; y++) {
            if ((x % 2 == 0 && y % 2 == 0) || ((x % 2) && (y % 2)))
                black.push_back(matrix[x][y]);
            else
                white.push_back(matrix[x][y]);
        }
    }

    sort(black.begin(), black.end());
    sort(white.begin(), white.end());
    int b = 0, w = 0;
    for (int x = x1; x <= x2; x++) {
        for (int y = y1; y <= y2; y++) {
            if ((x % 2 == 0 && y % 2 == 0) || ((x % 2) && (y % 2)))
                matrix[x][y] = black[b++];
            else
                matrix[x][y] = white[w++];
        }
    }
}
```

```
void print(vector<vector<int>> &matrix)
{
    for (auto &row : matrix) {
        for (auto num : row)
            cout << num << " ";
        cout << endl;
    }
}

int main()
{
    vector<vector<int>> matrix = {{1,4,3,2}, {8,4,7,1}, {1,5,2,1}};
    vector<int> sub = {0, 1, 3};
    print(matrix);
    cout << endl;
    sortChessSubsquares(matrix, sub);
    print(matrix);
    return 0;
}
```

54. matrixQueries。一个matrix, matrix[x][y] = (x+1) * (y+1)。现在有两个操作，抹除行或者列，返回matrix最小值。求设计这两个API。比如 removeCol(0), removeRow(3), getMin() -> 2
    1 2 3 4
    2 4 6 8
    3 6 9 12
    4 8 12 16
    之前做了sample questions发现第四题一定要用优化的数据结构和算法，不然过不了全部的case。这个题卖个关子，看大家怎么想。

```
class Solution {
public:
    Solution(int rows, int cols) {
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                int val = (r+1) * (c+1);
                num_list.push_back(val);
                it_map[val] = prev(num_list.end());
            }
        }
```

```
        this->rows = rows;
        this->cols = cols;
    }

    void removeRow(int r) {
        for (int c = 0; c < cols; c++) {
            int val = (r+1) * (c+1);
            if (it_map.count(val)) {
                auto it = it_map.find(val);
                num_list.erase(it->second);
                it_map.erase(it);
            }
        }
    }

    void removeCol(int c) {
        for (int r = 0; r < rows; r++) {
            int val = (r+1) * (c+1);
            if (it_map.count(val)) {
                auto it = it_map.find(val);
                num_list.erase(it->second);
                it_map.erase(it);
            }
        }
    }

    int getMin() {
        if (it_map.size() == 0) return -1;

        return num_list.front();
    }

private:
    int rows, cols;
    list<int> num_list;
    unordered_map<int, list<int>::iterator> it_map;
};

int main()
{
```

```
    int rows = 4, cols = 4;
    Solution s(rows, cols);

    s.removeCol(0);
    s.removeRow(3);
    cout << s.getMin() << endl;
    return 0;
}
```