Zingfront智线 | 技术笔试题@ 2018 B

- 笔试题，我承诺:

1. 保证答题的独立完成性， 向任何笔试无关人员泄露笔试内容;

2. 如需”链表/图/树/字典/矩阵”等复杂结构，自行设计，尽量不引用java/python等语言里的成熟库;

3. 独立完成2~3道笔试题 ， 并完成我功能测试;

4. 编码之前，请先给出基本算法描述，代码需要适当注释; 写出自己设计的算法和数据结构对于时间复杂度和空间复杂度的分析。

5. 在当天4时内完成笔试，并反馈结果。

备注：笔试题要求里的2.4两条需要重点完成。

- 题 1:

给定一组数字， 一组有9个数字，将这9个数字填写到3*3的九宫格内;使得横，竖，斜对角一条线上的三个数字之和相等;如果无解则打印无解;

- 题 2:

给定形如下 的矩阵，

1 1 1 1 1 1

1 1 0 0 0 1

1 0 0 0 1 0

1 1 0 1 1 1

0 1 0 1 0 0

1 1 1 1 1 1

上面矩阵的中的1代表海岸线，0代表小岛。求第二岛的面积(即被1中包围的0的个数，如果只有一个小岛，输出最大岛的面积)。注意:

1. 仅求这样的0，该0所在行中被两个1包围，该0所在列中被两个1包围;

2. 输入矩阵中包含的小岛K >= 1;

样例输入 :

1 1 1 1 1 1

1 1 0 0 0 1

1 0 0 0 1 0

1 1 0 1 1 1

0 1 0 1 0 0

1 1 1 1 1 1

样例输出:

8

- 题 3:

设计一个股票模拟交易系统。假设我们有一个很牛叉的AI系统，已经预测到未来一段时间内给定股票的价格，以数组来表示，它的第i个元素是一支给定的股票在第i天的价格。

假设:

1. 如果你最多只允许完成两次交易(一次交易是指:买入和卖出);

2. 你有本金K单位(K >= 1)，一单位本金可以购买1股票;这意味着你寻找的是K单位本金条件下最大利润。

提示:K = 1的时候最简单，可以先考虑。

设计一个算法来找出最大利润。

解答:

题1

```
import java.util.Arrays;
/**
 *题目 1:
 给定一组数字，一组有 9 个数字，将这 9 个数字填写到 3*3 的九宫格内；使得横，竖,斜对角条线一上的三个数字之和相等；如果无解则打印无解;
 算法解释:
 如下将 9 宫格视为一个二维数组，先将 9 个数字依次填入格中，然后查看是否满足横、竖、斜一条线之和相等。满足就输出，不满足则无解。
 [0][0] [0][1] [0][2]
 [1][0] [1][1] [1][2]
 [2][0] [2][1] [2][2]
 */
public class Test1 {

    public static void main(String[] args){

        int[] n={3,4,1,2,5,6,7,8,9};//例
        suduku(n);
```

```
    }

    public static void suduku(int[] n){
        Arrays.sort(n);//把数组从小到大排序
        int x=0,y=1;
        int[][] m = new int[3][3];//九宫格视为一个数组
        m[x][y]=n[0];//把数组第一个数放在九宫格第一行中间
        int i,a,b;
        for(i=1;i<9;i++){
            a=x-1;
            b=y+1;//依次斜填
            if(a<0){//上出框往下填
                a=2;
            }
            if(b>2){//右出框往左填
                b=0;
            }
            if (m[a][b]!=0){//排重在下格填，右上排重一样
                a=x+1;
                b=y;
            }
            m[a][b]=n[i];//赋值
            x=a;
            y=b;
        }
        //计算九宫格中每一行，列，斜对角线上的值
        int row1=m[0][0]+m[0][1]+m[0][2];
        int row2=m[1][0]+m[1][1]+m[1][2];
        int row3=m[2][0]+m[2][1]+m[2][2];
        int col1=m[0][0]+m[1][0]+m[2][0];
        int col2=m[0][1]+m[1][1]+m[2][1];
        int col3=m[0][2]+m[1][2]+m[2][2];
        int dig1=m[0][0]+m[1][1]+m[2][2];
        int dig2=m[2][0]+m[1][1]+m[0][2];
        //比较横竖斜方向上的的值是否相等
        if(row1==row2 && row1==row3&& row1==col1&&row1==col2&&row1==col3&&row1==dig1&&row1==dig2){
            for (i=0;i<3;i++)
                System.out.println ( Arrays.toString (m[i]));
        }else{
            System.out.println("无解");
        }

    }
}
```

时间复杂度0(n)
空间复杂度0(1)

题2：

```
/**题目 2:
给定形如下面的矩阵，
111111
110001
100010
110111
010100
111111
上面矩阵的中的 1 代表海岸线，0 代表小岛。求第二大小岛的面积(即被 1 中包围的 0 的
个数，如果只有一个小岛，输出最大小岛的面积)
注意
1.仅求这样的 0,该 0 所在行中被两个 1 包围，该 0 所在列中被两个 1 包围;
2.输入矩阵中包含的小岛 K >= 1;
```

```
样例输入:
111111
110001
100010
110111
010100
111111
样例输出:
8
解题思路:
 遍历所有的行和列，记录该行或列，最左面和最右面和最上面和最下面 1 的坐标，然后当遇到 0，
 判断是否处于记录的值的中间，是，则是内陆，面积加 1，否则不加。
*/
public class Test2 {

    public static void main(String[] args){

        int[] n1={1,1,1,1,1,1};
        int[] n2={1,1,0,0,0,1};
        int[] n3={1,0,0,0,1,0};
        int[] n4={1,1,0,1,1,1};
        int[] n5={0,1,0,1,0,0};
        int[] n6={1,1,1,1,1,1};

        int[][] area={n1,n2,n3,n4,n5,n6};

        index(area);

    }

    public static void index(int[][] area){
        int i,j,t,r,c,sum=0;
        int rowNum=area.length;//确定区域长
        int colNum=area[0].length;//确定区域宽
        for(i=1;i<rowNum;i++){//i=0 或 j=0 肯定不满足，所以从 1 开始
            for(j=1;j<colNum;j++){
                if(area[i][j]==0){
                    t=0;
                    for(r=i+1;r<rowNum;r++){//下方有 1
                        if(area[r][j]==1){
                            t++;
                            break;
                        }
                    }
                    for(r=i-1;r>=0;r--){//上方有 1
                        if(area[r][j]==1){
                            t++;
                            break;
                        }
                    }
                    for(c=j+1;c<colNum;c++){//右方有 1
                        if(area[i][c]==1){
                            t++;
                            break;
                        }
                    }
                    for(c=j-1;c>=0;c--){//左方有 1
                        if(area[i][c]==1){
                            t++;
                            break;
                        }
                    }
                    if(t==4){//如果满足以上四个条件
                        sum++;
                    }
                }
            }
```

```
        }

        System.out.println("小岛面积为"+sum);
    }
}
```

时间复杂度O(n^3)
空间复杂度O(1)

题3

```
/**题目 3:
设计- 一个股票模拟交易系统。假设我们有一一个很牛叉的 AI 系统，已经预测到未来一段时间内给定股票的价格，以数组来表
示，它的第 i 个元素是一支给定的股票在第 i 天的价格。
假设:
1.如果你最多只允许完成两次交易(一次交易是指：买入和卖出);
2.你有本金 K 单位(K>=1)，一单位本金可以购买 1 股票；这意味着你寻找的是 K 单位本金条件下最大利润。
提示：K=1 的时候最简单，可以先考虑。
设计一个算法来找出最大利润。
解体思路：
对数组进行遍历，求出某点之前的最大利益和某点之后的最大利益，当二者和最大时即为利润最大
*/

public class Test3 {
    public static void main(String[] args){
        int[] n= {1,2,3,7,6,1,77};

        System.out.println("最大利润为"+maxProfit(n));

    }

    public static int maxProfit(int[] n){
        if(n.length==0){
            return 0;
        }
        int i;
        int[] post=n;
        int[] pre=n;
        int minPrice=n[0];
        for (i=1;i<n.length;i++){
            minPrice=min(minPrice,n[i]);
            pre[i]=max(pre[i-1],n[i]-minPrice);
        }

        //计算第 i 点之后的最大利润 post
        int maxPrice=n[n.length-1];
        for (i=n.length-2;i>=0;i--){
            maxPrice=max(maxPrice,n[i]);
            post[i]=max(post[i+1],maxPrice-n[i]);
        }
        int maxProfit=0;
        //计算第 i 点的，pre[i]与 post[i]之和的最大值，赋值给 maxProfit
        for (i=0;i<n.length;i++){
            maxProfit=max(maxProfit,pre[i]+post[i]);
        }
        return maxProfit;

    }
    public static int min(int a,int b){
        if(a>b)
            return b;
        if(a<b)
            return a;
        if(a==b)
            return a;
```

```
        return 0;
    }
    public static int max(int a,int b){
        if(a>b)
            return a;
        if(a<b)
            return b;
        if(a==b)
            return a;
        return 0;
    }
}
```

时间复杂度O(n)
空间复杂度O(1)