Please use this Google doc during your interview (your interviewer will see what you write here). To free your hands for typing, we recommend using a headset or speakerphone.

There is a plot of grid-like land with each spot having an elevation. Within this plot there are two villages that need water delivered to them. Which spot (or elevation) is the best place to build a water tower so that both villages get water. Water can flow to a spot with the same or lower elevation.
The input will be an NxM dimensional matrix and the coordinates or indices for the two villages return the best spot to build the water tower if there is one.
The best spot is the highest elevation where water can flow to both villages.

[4 **9** 7 6 5 |
| 2 6 5 4 **3** |
| 6 5 1 2 8 |
| 3 **4** 7 2 5]
Returns: 9

[2 **7** **7** **7** 5 |
| 6 5 6 **4** 1 |
| **3** 4 5 2 8]
Returns: 7

DATA
repeated elements continue
hier blocks
INPUT

OUTPUT
Just the level

EXAMPLE

APPROACH
```
Search from each cell  good solution, O((n*m)^2)
      Bfs take of visited
            if we visit the two villages count
```

```
        return max

Search from the villages O(n*m)
        bfs from each village
        save full path of water in reverse order
        //I have two full paths
        -traverse full matrix get the higher in common<-

        -save all cell intersec all cell and return the max

//Documentations package import class…

int towerSpot(int[][] matrix, int[] villageA, int[] villageB){
        //villageX[0] -> row, villageX[1] -> col


        int[][] visitedA=getPath(matrix,villageA);
        int[][] visitedB=getPath(matrix,villageB);





        int max=-1; //-1, in case of no tower spot
        for(int row=0;row<matrix.length;row++){
                for(int col=0;col<matrix[0].length;coll++){
                        if(visitedA[row][col]==1&&visitedB[row][col]==1){
                                max=Math.max(max,matrix[row][col]);
                        }
                }
        }
        return max;
}

int[][] getPath(int[][] matrix, int[] village){
//get all possible cell we can reach from a village to a tower spot
        int row=village[0];
        int col=village[1];

        int[][] visited=new int[matrix.length][matrix[0].length];

        Queue<int[]> bfs=new LinkedList();

        bfs.add(village);
        visited[row][col]=1;

        while(!bfs.isEmpty()){
                village=bfs.poll();
                row=village[0];
```

```
            col=village[1];

            if(visited[row][col]==1) continue;
            visited[row][col]=1;

            if(row-1>=0&&matrix[row-1][col]>=matrix[row][col])
            bfs.add(new int[]{row-1,col});
            if(row+1<matrix.length&&matrix[row+1][col]>=matrix[row][col])
            bfs.add(new int[]{row+1,col});
            if(col-1>=0&&matrix[row][col-1]>=matrix[row][col])
            ...
        }
        return visited:
}
```