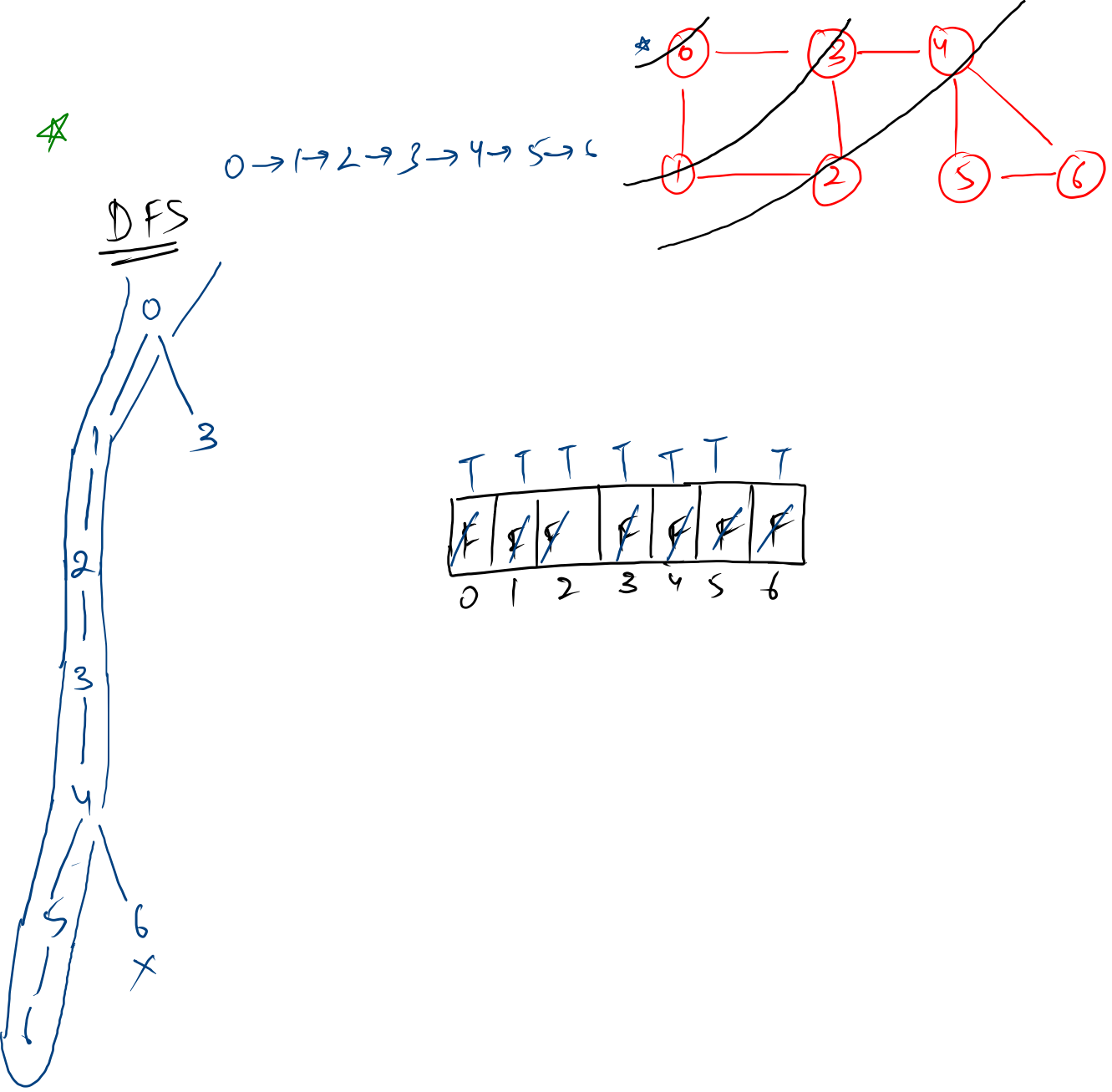

0 → 1 → 2 → 3 → 4 → 5 → 6
DFS
0
1
2
3
4
5
6
X
T T T T T T T
F F F F F F F
0 1 2 3 4 5 6
BFS ↔ Level order
0 1 3 2 4 5 6 6
T T T T T T T
F F F F F F F
0 1 2 3 4 5 6

Num of Islands

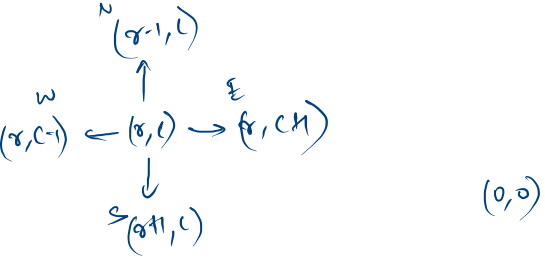
1 ⇒ land

0 ⇒ water

2 ⇒ visited land

Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]

Output: 1




```
static int dir[][] = {{-1,0},{0,+1},{+1,0},{0,-1}};
```

d=0, r'=-1, c'=0

d=1, r'=0, c'=1

d=2, r'=1, c'=0

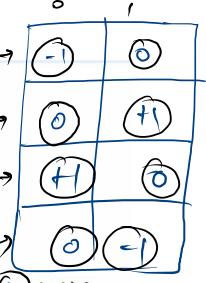
d=3, r'=0, c'=-1

r=0 c=0

(-1,0) ← (0,0) → (0,1)

(1,0)

N 0→ ⇒ S 2→ E 1→ W 3→



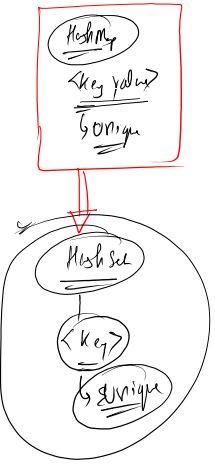

```
public void helper(char[][] grid,int r,int c){
    grid[r][c] = '2'; // mark

    for(int d = 0 ; d < 4 ; d++){
        int rowdash = r + dir[d][0];
        int coldash = c + dir[d][1];

        if(rowdash < 0 || coldash < 0 || rowdash >= grid.length || coldash >= grid[0].length || grid[rowdash][coldash] != '1'){
            continue;
        }

        helper(grid,rowdash,coldash);
    }
}
```

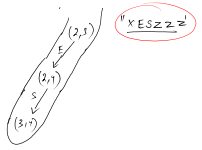
out of array

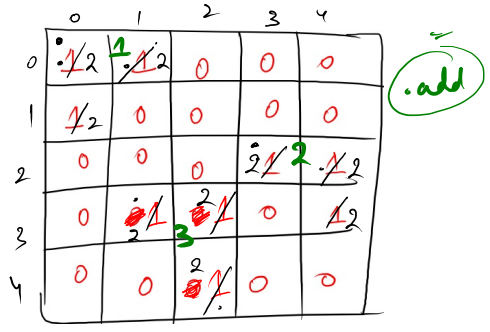
unvisited land

```
public int numIslands(char[][] grid) {
    int count = 0;

    for(int r = 0 ; r < grid.length ; r++){
        for(int c = 0 ; c < grid[0].length ; c++){
            if(grid[r][c] == '1'){ //  found an unvisited land i.e. an island
                count++;
                helper(grid,r,c); // visit & mark all the connected area(unvisited land)
            }
        }
    }

    return count;
}
```

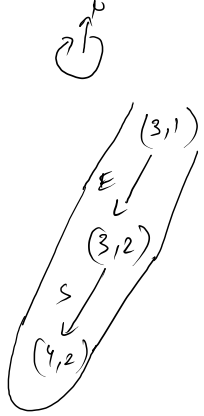
	0	1	2	3	4
0	1	1	1	1	0
1	1	1	0	1	0
2	1	1	0	0	0
3	0	0	0	0	0







.add



"XESZZZ"

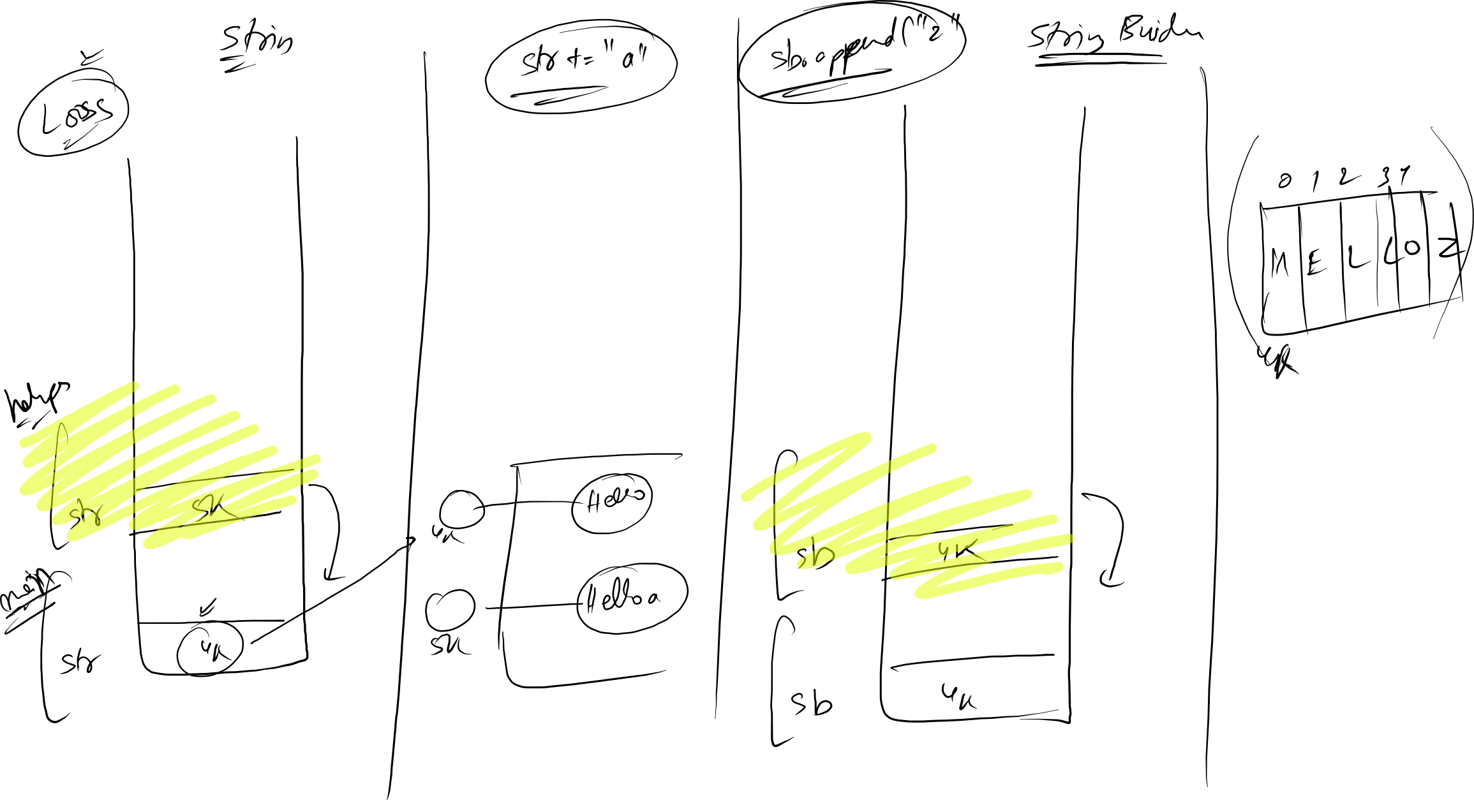
= HashSet<String> set = new HashSet<>();

island → traverse → String → Structure

String → Hash set

String unique value

no. of distinct islands

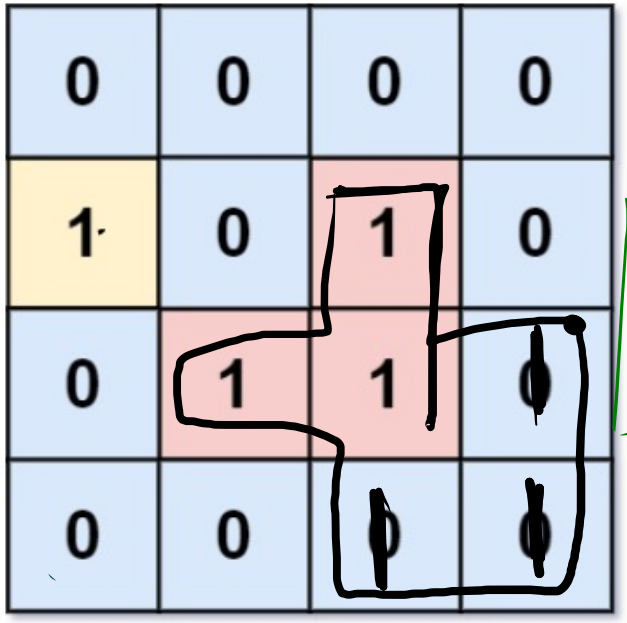

String
Loss
str += "a"
sb.append("2")
StringBuilder
0 1 2 3 4
H E L L O
helper
str
sb
yk
main
str
yk
Hello
yk
Hello a
sb
yk
sb
yk

No. of Areas = 5

+

Not touching boundary

Example 1:



Count = 3

Island mark

land cells / size

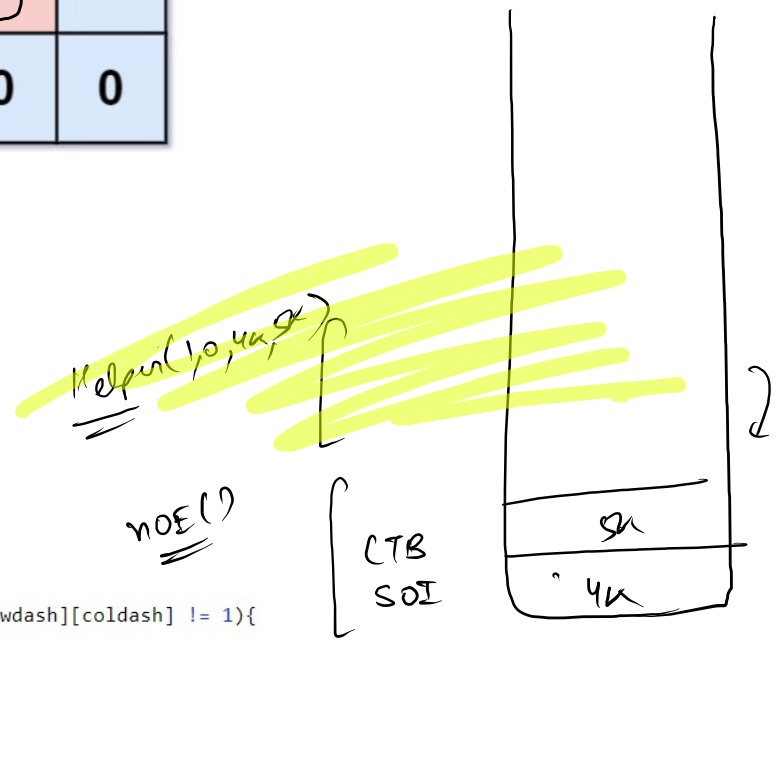
touching boundary → Y/N

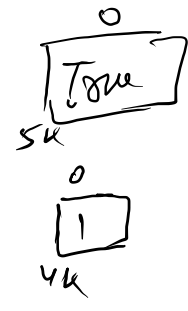
```java
public int numEnclaves(int[][] grid) {
    int count = 0;
    int[] sizeOfIsland;
    boolean[] connectedToBoundary;
    for(int r = 0 ; r < grid.length ; r++){
        for(int c = 0 ; c < grid[0].length ; c++){
            if(grid[r][c] == 1){

                sizeOfIsland = new int[]{0};
                connectedToBoundary = new boolean[]{false};

                helper(grid,r,c,sizeOfIsland,connectedToBoundary);

                if(!connectedToBoundary[0]){
                    count += sizeOfIsland[0];
                }
            }
        }
    }
    return count;
}
```

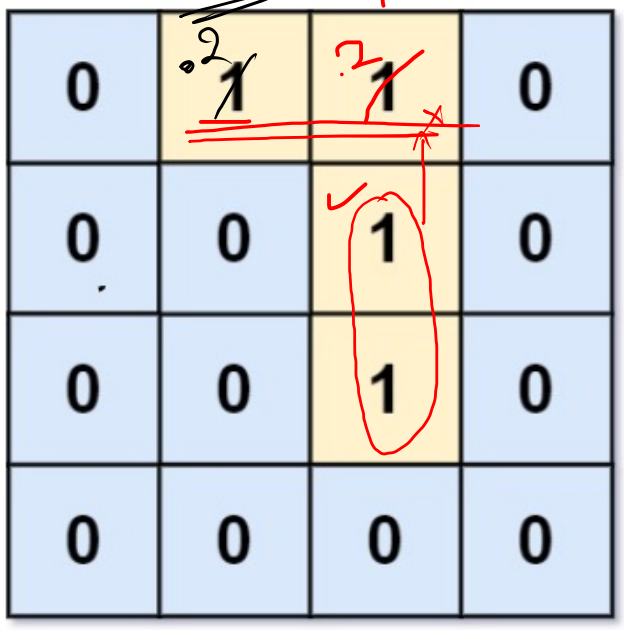
Example 1:

	0	1	2	3
0	0	0	0	0
1	1	0	1	0
2	0	1	1	0
3	0	0	0	0

```java
 static int dir[][] = {{-1,0},{0,+1},{+1,0},{0,-1}};

 public void helper(int[][] grid,int r,int c,int []sizeOfIsland,boolean[] connectedToBoundary){
     if(r == 0 || c == 0 || r == grid.length-1 || c == grid[0].length-1){
         connectedToBoundary[0] = true;
     }

     grid[r][c] = 2;
     sizeOfIsland[0]++;

     for(int d = 0 ; d < 4; d++){
         int rowdash = r + dir[d][0];
         int coldash = c + dir[d][1];

         if(rowdash < 0 || coldash < 0 || rowdash >= grid.length || coldash >= grid[0].length || grid[rowdash][coldash] != 1){
             continue;
         }

         helper(grid,rowdash,coldash,sizeOfIsland,connectedToBoundary);
     }
 }
```





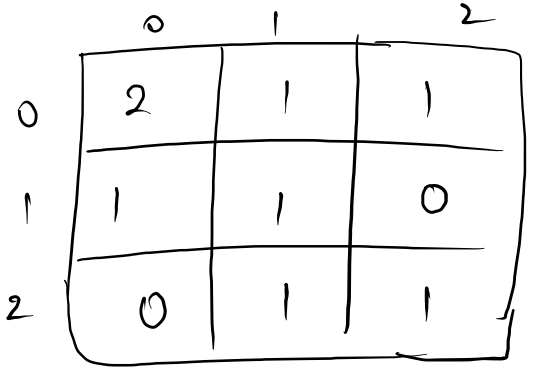


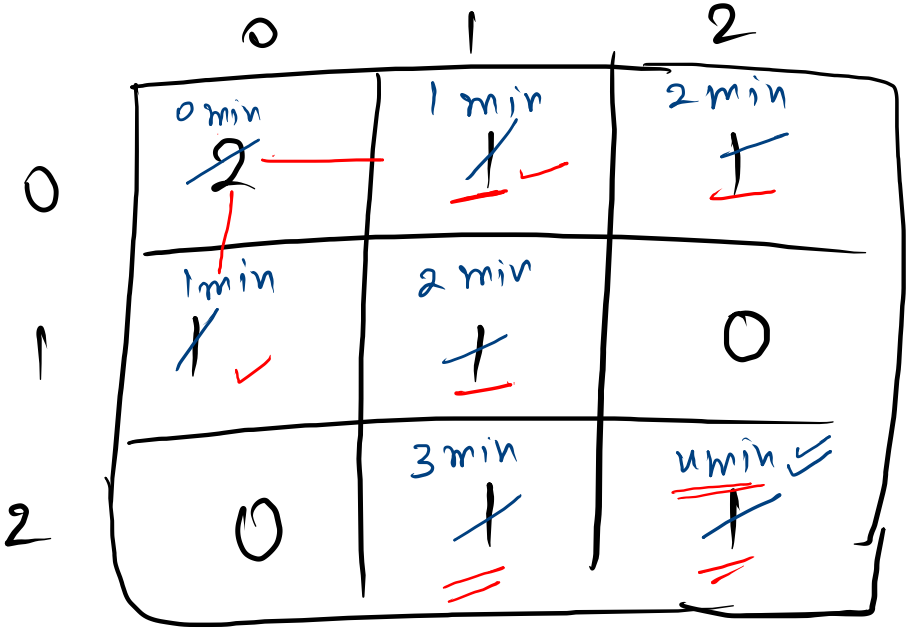
Return *the number of land cells in* `grid` *for which we cannot walk off the boundary of the grid in any number of* ***moves***.

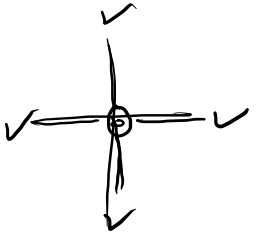
```
Input: grid = [[2,1,1],[1,1,0],[0,1,1]]
Output: 4
```

2 → rotten orange 1 → fresh orange

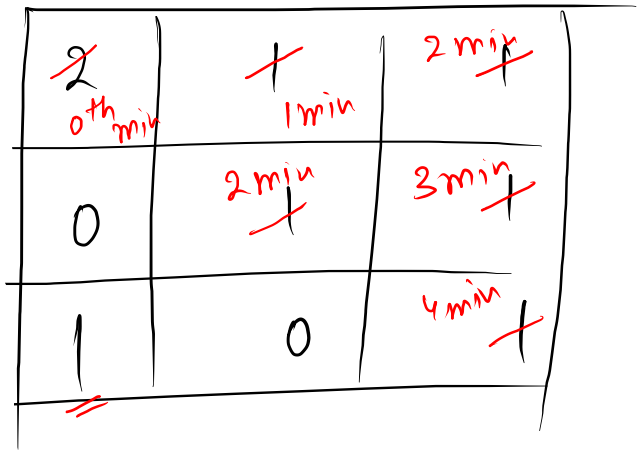
0 → Empty Cell



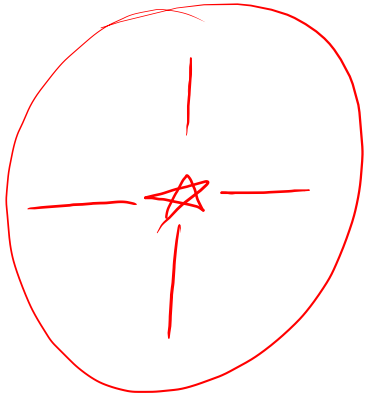




```
Input: grid = [[2,1,1],[0,1,1],[1,0,1]]
Output: -1
```

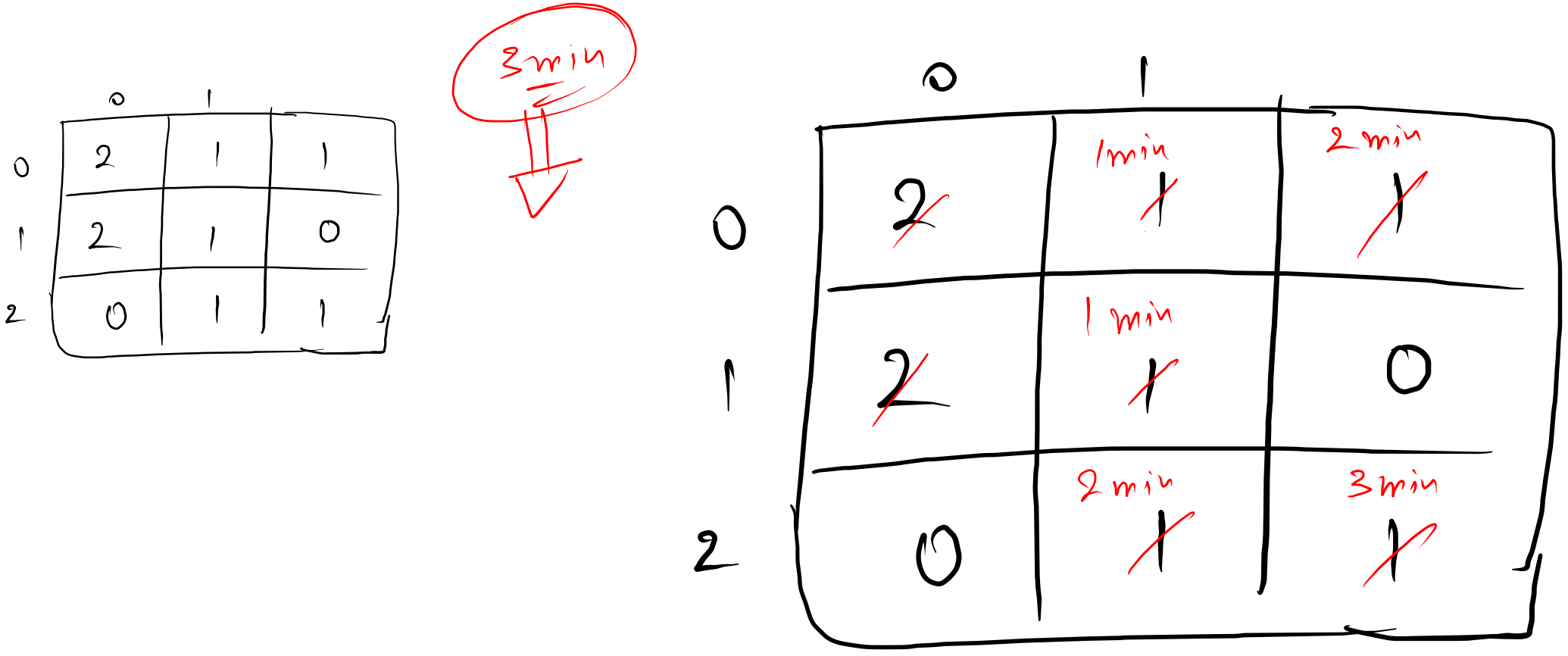


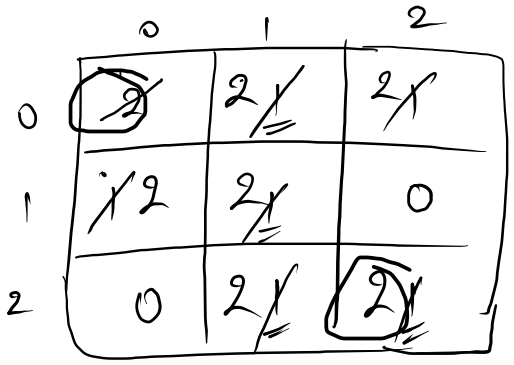
	0	1	2
0	2	1	1
1	1	1	0
2	0	1	2

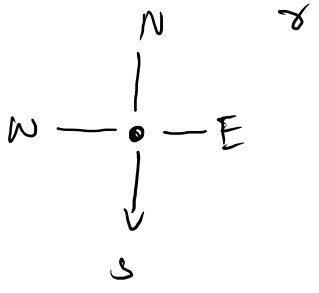


2-min

	0	1	2
0	~~2~~	1min ~~1~~	2min ~~1~~
1	~~1~~ 1min	2min ~~1~~	0
2	0	~~1~~ 1min	~~2~~


3min
0 1
0 2 1 1
1 2 1 0
2 0 1 1
0 1
0 2 1min 1 2min 1
1 2 1min 1 0
2 0 2min 1 3min 1





Omin

LevelSize : 1 1 1 0

fresh = 6 5 4 3 2 1 0

0 1 2 3 4

(0,0) (0,1) (1,0) (0,2) (1,1) (1,1) (2,1) (2,1) (2,2) (2,2)

(0,0) => 0

(0,1) (1,0) => 1

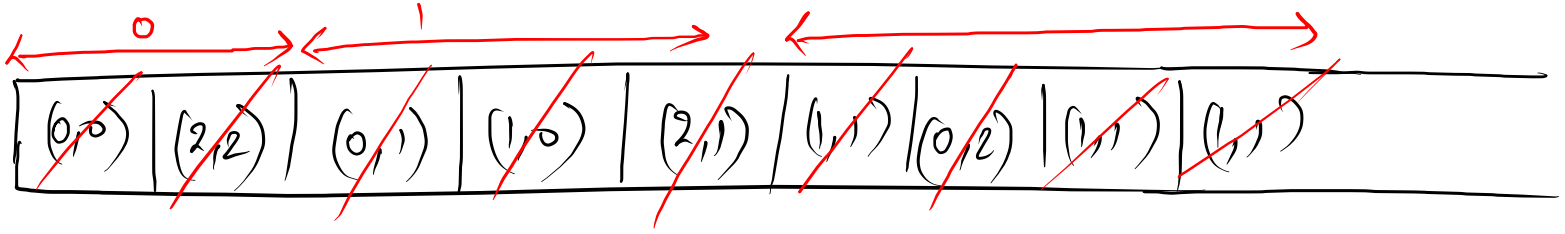
(0,2) (1,1) (1,1) => 2

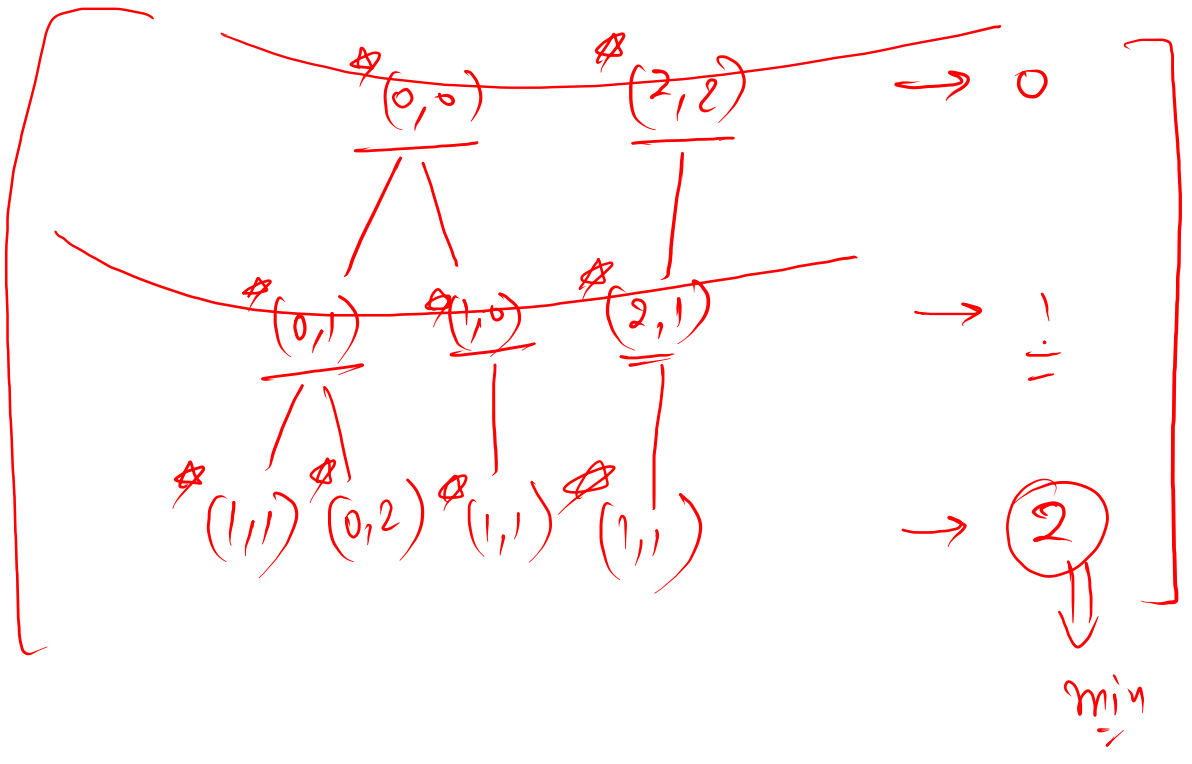
(2,1) (2,1) => 3

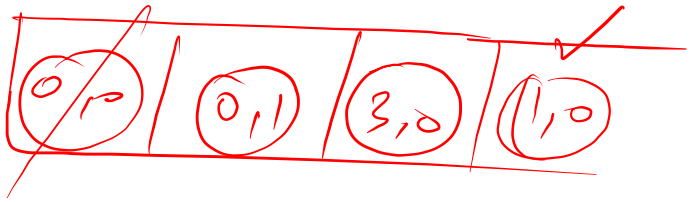
(2,2) (2,2) => 4

	0	1	2
0	2 (0 min)	2	2
1	2	2	0
2	0	2	2 (0 min)

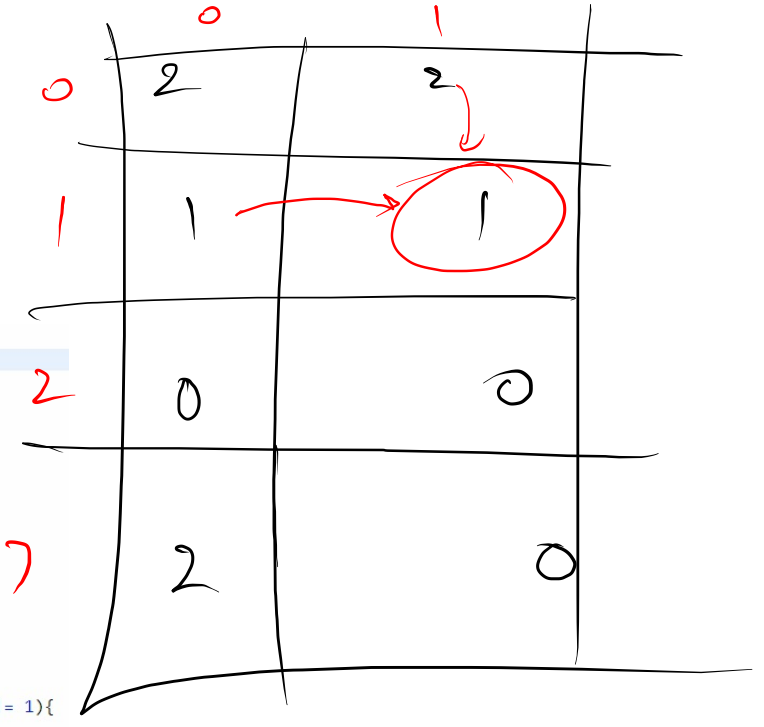
Fresh Oranges → ~~5~~ ~~4~~ ~~3~~ ~~2~~ 1 0 Level Size → 4





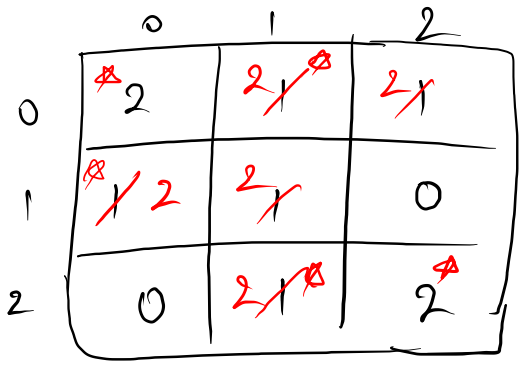


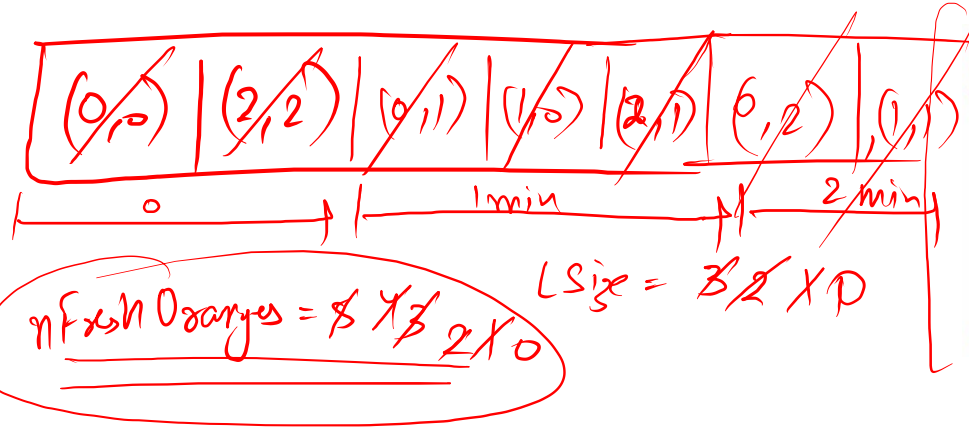
#2



Level = 0

```
int dir[][] = {{-1,0},{0,1},{1,0},{0,-1}};
int level = 0;
while(queue.size() > 0){
    int lSize = queue.size();

    while(lSize-- > 0){
        Pair tmp = queue.remove();

        if(grid[tmp.row][tmp.col] == 1){
            nFreshOranges--;
            grid[tmp.row][tmp.col] = 2;
        }

        for(int i = 0 ; i < 4 ; i++){
            int rdash = tmp.row + dir[i][0];
            int cdash = tmp.col + dir[i][1];

            if(rdash < 0 || cdash < 0 || rdash >= grid.length || cdash >= grid[0].length || grid[rdash][cdash] != 1){
                continue;
            }

            queue.add(new Pair(rdash,cdash));
        }
    }

    if(queue.size() != 0){
        level++;
    }
}

if(nFreshOranges == 0){
    return level;
}else{
    return -1;
}
```



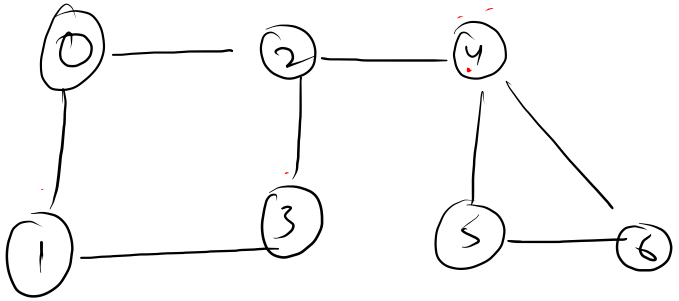


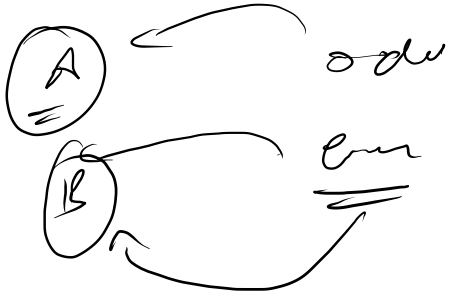

```
Queue<Pair> queue = new ArrayDeque<>();
int nFreshOranges = 0;
for(int r = 0 ; r < grid.length ; r++){
    for(int c = 0 ; c < grid[0].length ; c++){
        if(grid[r][c] == 1){
            nFreshOranges++;
        }else if(grid[r][c] == 2){
            queue.add(new Pair(r,c));
        }
    }
}
```

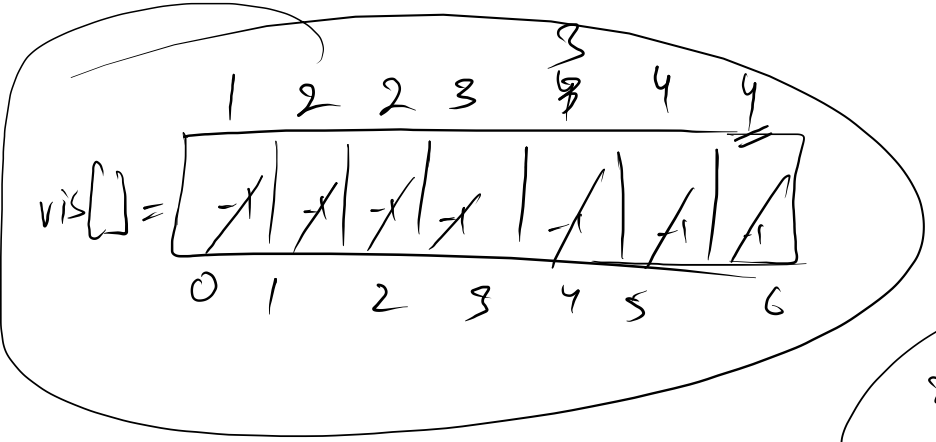
(0,0) (2,2) ← 0

(0,1) ← 1

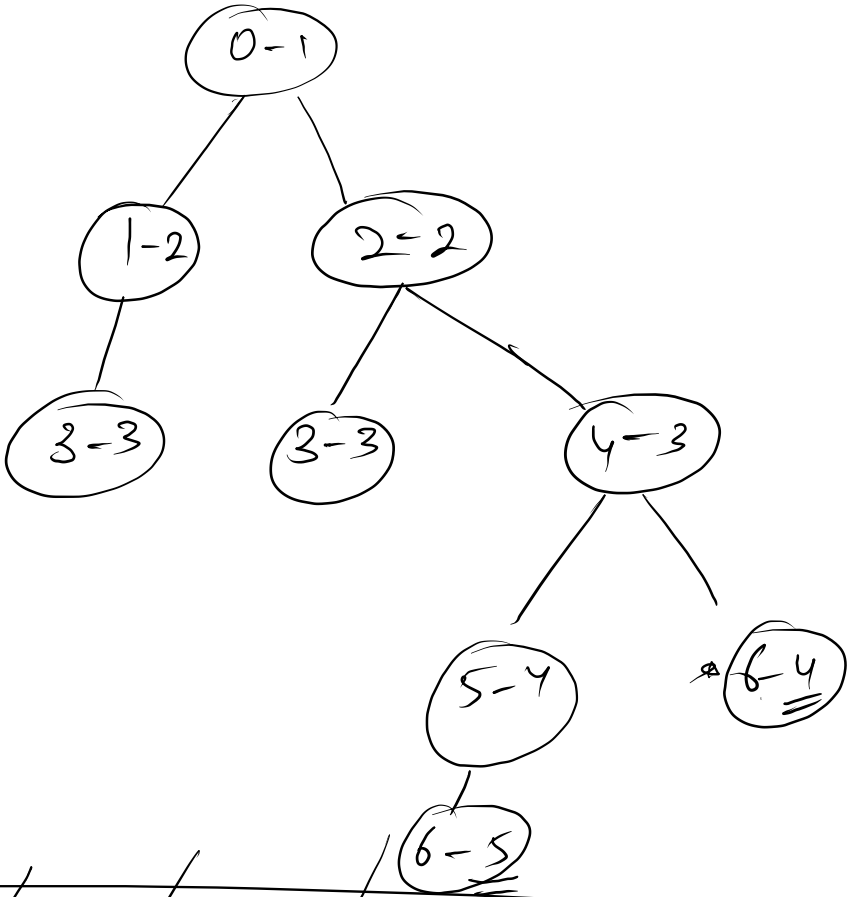

```
int dir[][] = {{-1,0},{0,1},{1,0},{0,-1}};
int level = 0;
while(queue.size() > 0){
    int lSize = queue.size();

    while(lSize-- > 0){
        Pair tmp = queue.remove();
        for(int i = 0 ; i < 4 ; i++){
            int rdash = tmp.row + dir[i][0];
            int cdash = tmp.col + dir[i][1];

            if(rdash < 0 || cdash < 0 || rdash >= grid.length || cdash >= grid[0].length || grid[rdash][cdash] != 1){
                continue;
            }
            nFreshOranges--;
            grid[rdash][cdash] = 2;
            queue.add(new Pair(rdash,cdash));
        }
    }

    if(queue.size() != 0){
        level++;
    }
}

if(nFreshOranges == 0){
    return level;
}else{
    return -1;
}
```

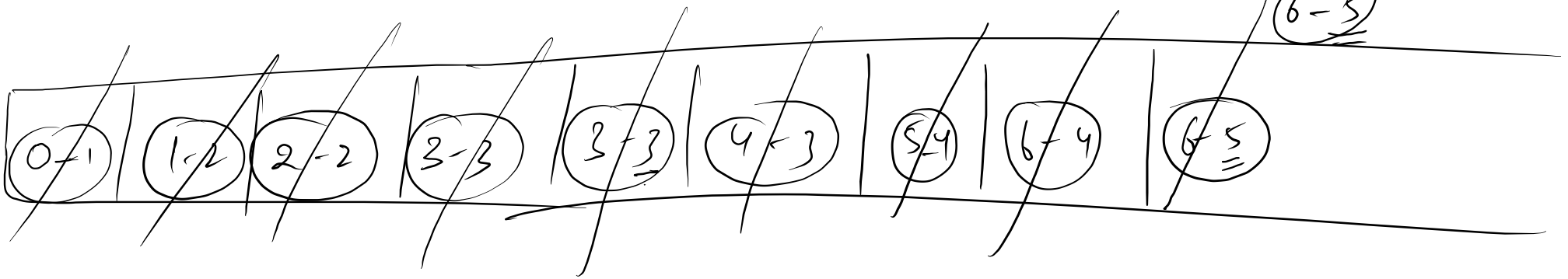

0
2
4
1
3
5
6


A
B


1 2 2 3 3 4 4
vis[] =
0 1 2 3 4 5 6


schon fahr


0-1
1-2
2-2
3-3
3-3
4-3
5-4
6-4
6-5


0-1
1-2
2-2
3-3
3-3
4-3
5-4
6-4
6-5