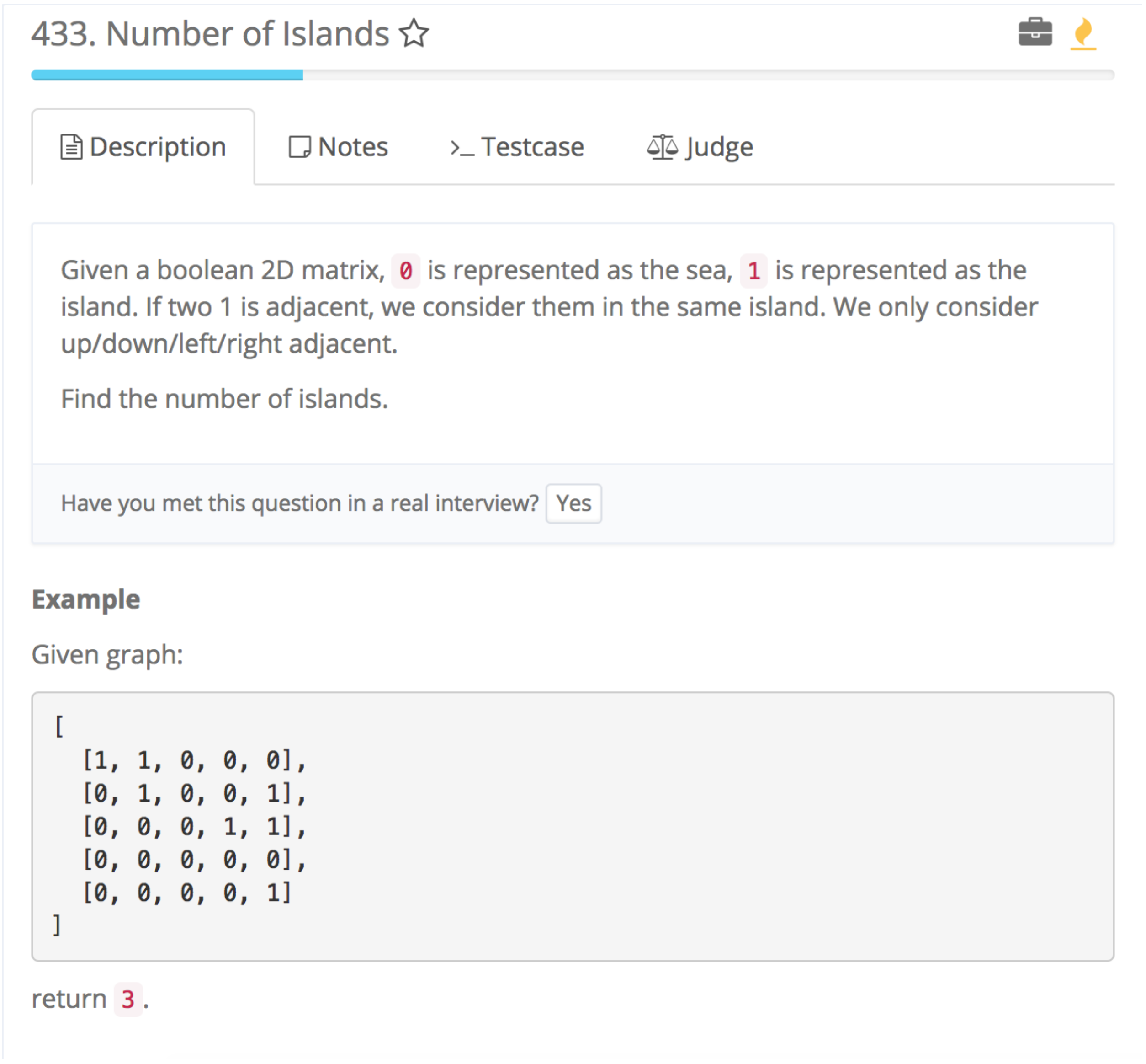


433. Number of Islands ☆

Description | Notes | >_ Testcase | Judge

Given a boolean 2D matrix, `0` is represented as the sea, `1` is represented as the island. If two 1 is adjacent, we consider them in the same island. We only consider up/down/left/right adjacent.

Find the number of islands.

Have you met this question in a real interview? Yes

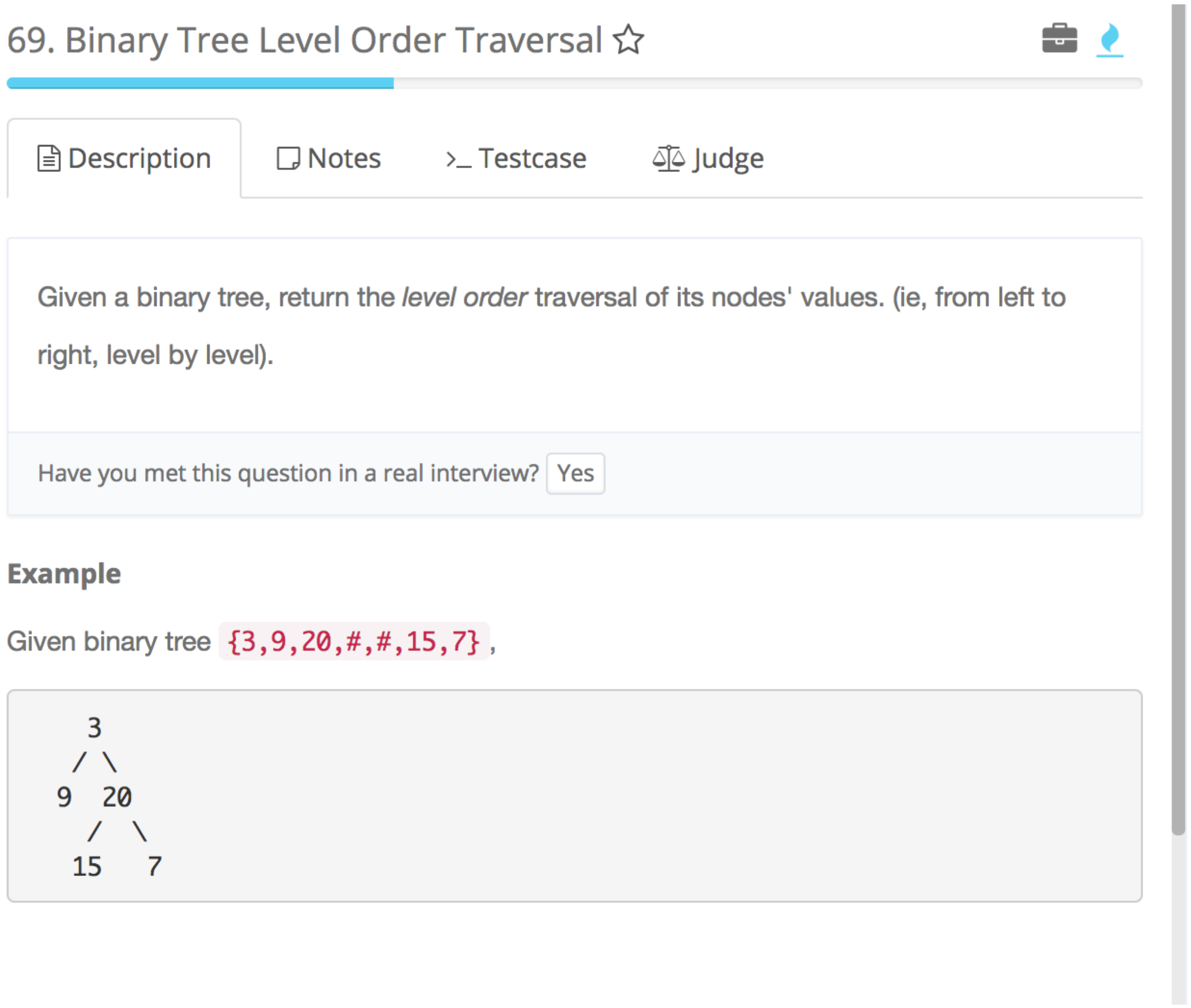
Example

Given graph:

```
[
  [1, 1, 0, 0, 0],
  [0, 1, 0, 0, 1],
  [0, 0, 0, 1, 1],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1]
]
```

return `3`.

```
class Coordinate {
    int x, y;
    public Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Solution {
    /**
     * @param grid a boolean 2D matrix
     * @return an integer
     */
    public int numIslands(boolean[][] grid) {
        if (grid == null || grid.length == 0 || grid[0].length == 0) {
            return 0;
        }

        int n = grid.length;
        int m = grid[0].length;
        int islands = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j]) {
                    markByBFS(grid, i, j);
                    islands++;
                }
            }
        }

        return islands;
    }

    private void markByBFS(boolean[][] grid, int x, int y) {
        // magic numbers!
        int[] directionX = {0, 1, -1, 0};
        int[] directionY = {1, 0, 0, -1};

        Queue<Coordinate> queue = new LinkedList<>();

        queue.offer(new Coordinate(x, y));
        grid[x][y] = false;

        while (!queue.isEmpty()) {
            Coordinate coor = queue.poll();
            for (int i = 0; i < 4; i++) {
                Coordinate adj = new Coordinate(
                    coor.x + directionX[i],
                    coor.y + directionY[i]
                );
                if (!inBound(adj, grid)) {
                    continue;
                }
                if (grid[adj.x][adj.y]) {
                    grid[adj.x][adj.y] = false;
                    queue.offer(adj);
                }
            }
        }
    }

    private boolean inBound(Coordinate coor, boolean[][] grid) {
        int n = grid.length;
        int m = grid[0].length;

        return coor.x >= 0 && coor.x < n && coor.y >= 0 && coor.y < m;
    }
}
```



69. Binary Tree Level Order Traversal

Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).

Have you met this question in a real interview?

Example

Given binary tree `{3,9,20,#,#,15,7}`,

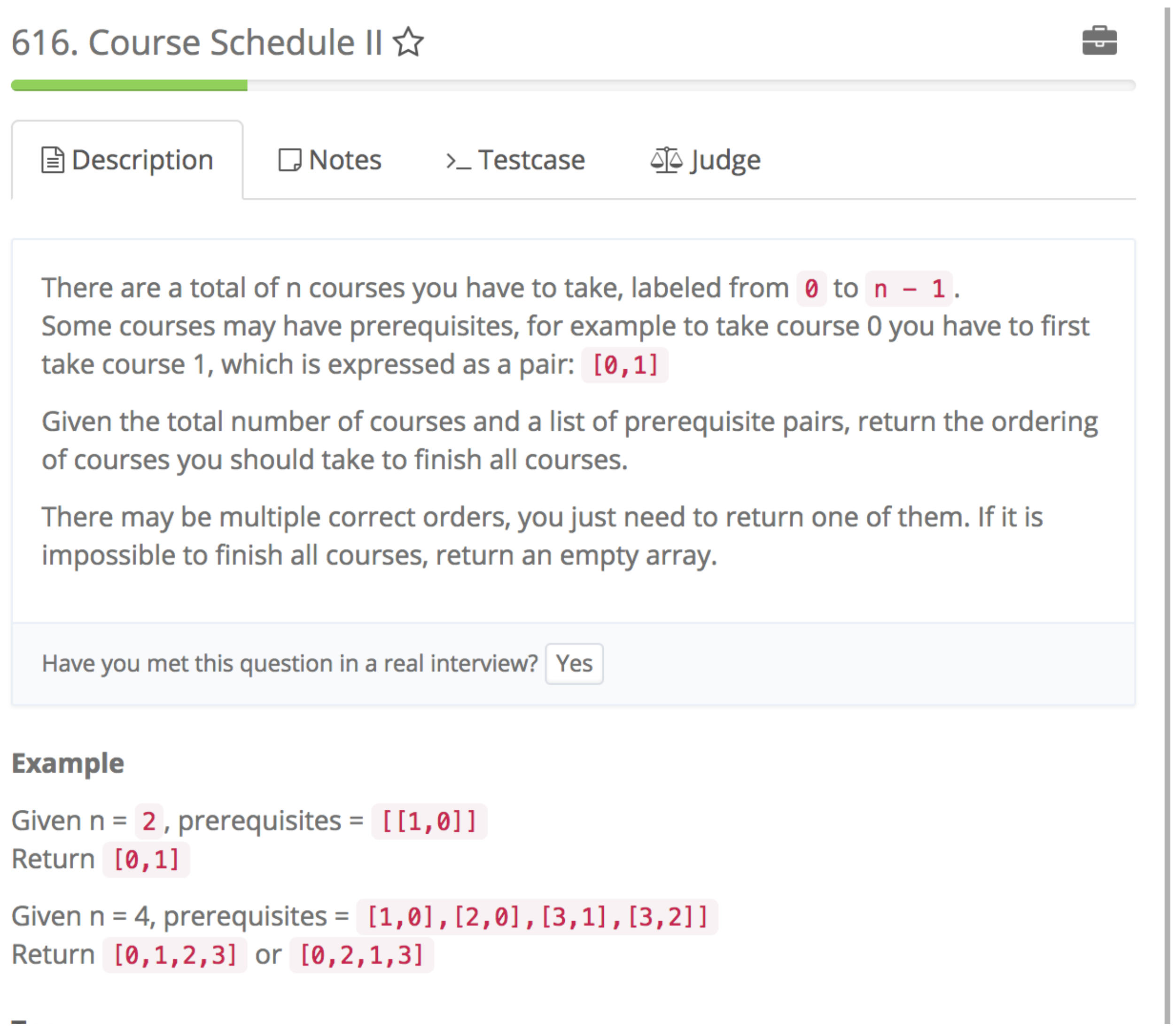
```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

// version 1: BFS
public class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List result = new ArrayList();

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

        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(root);

        while (!queue.isEmpty()) {
            ArrayList<Integer> level = new ArrayList<Integer>();
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode head = queue.poll();
                level.add(head.val);
                if (head.left != null) {
                    queue.offer(head.left);
                }
                if (head.right != null) {
                    queue.offer(head.right);
                }
            }
            result.add(level);
        }

        return result;
    }
}
```

```
// version 2:  DFS
public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: Level order a list of lists of integer
     */
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> results = new ArrayList<List<Integer>>();

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

        int maxLevel = 0;
        while (true) {
            List<Integer> level = new ArrayList<Integer>();
            dfs(root, level, 0, maxLevel);
            if (level.size() == 0) {
                break;
            }

            results.add(level);
            maxLevel++;
        }

        return results;
    }

    private void dfs(TreeNode root,
                     List<Integer> level,
                     int curtLevel,
                     int maxLevel) {
        if (root == null || curtLevel > maxLevel) {
            return;
        }

        if (curtLevel == maxLevel) {
            level.add(root.val);
            return;
        }

        dfs(root.left, level, curtLevel + 1, maxLevel);
        dfs(root.right, level, curtLevel + 1, maxLevel);
    }
}
```



616. Course Schedule II ☆

Description | Notes | >_ Testcase | Judge

There are a total of n courses you have to take, labeled from `0` to `n - 1`.
Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

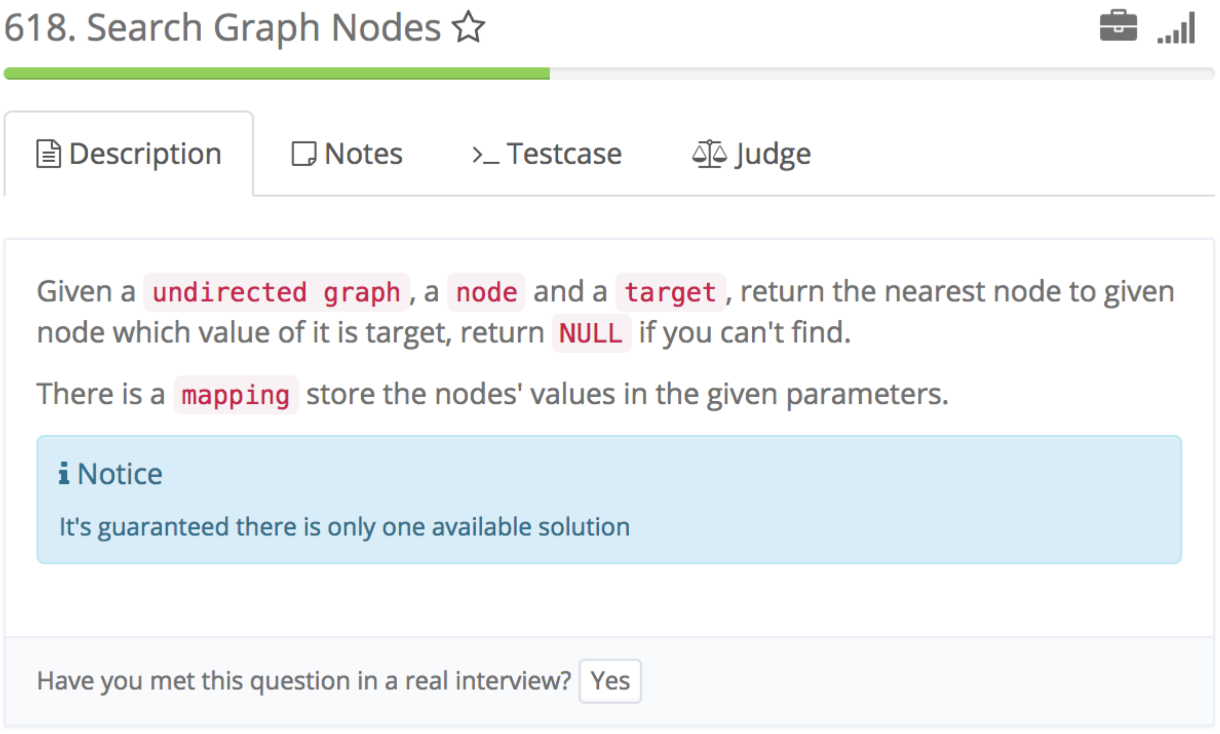
Have you met this question in a real interview? Yes

Example

Given n = `2`, prerequisites = `[[1,0]]`
Return `[0,1]`

Given n = 4, prerequisites = `[1,0],[2,0],[3,1],[3,2]]`
Return `[0,1,2,3]` or `[0,2,1,3]`

```
public class Solution {
    /**
     * @param numCourses a total of n courses
     * @param prerequisites a list of prerequisite pairs
     * @return the course order
     */
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        // Write your code here
        List[] edges = new ArrayList[numCourses];
        int[] degree = new int[numCourses];

        for (int i = 0;i < numCourses; i++)
            edges[i] = new ArrayList<Integer>();

        for (int i = 0; i < prerequisites.length; i++) {
            degree[prerequisites[i][0]] ++ ;
            edges[prerequisites[i][1]].add(prerequisites[i][0]);
        }

        Queue queue = new LinkedList();
        for(int i = 0; i < degree.length; i++){
            if (degree[i] == 0) {
                queue.add(i);
            }
        }

        int count = 0;
        int[] order = new int[numCourses];
        while(!queue.isEmpty()){
            int course = (int)queue.poll();
            order[count] = course;
            count ++;
            int n = edges[course].size();
            for(int i = n - 1; i >= 0 ; i--){
                int pointer = (int)edges[course].get(i);
                degree[pointer]--;
                if (degree[pointer] == 0) {
                    queue.add(pointer);
                }
            }
        }

        if (count == numCourses)
            return order;

        return new int[0];
    }
}
```



618. Search Graph Nodes ☆

Description Notes >_ Testcase Judge

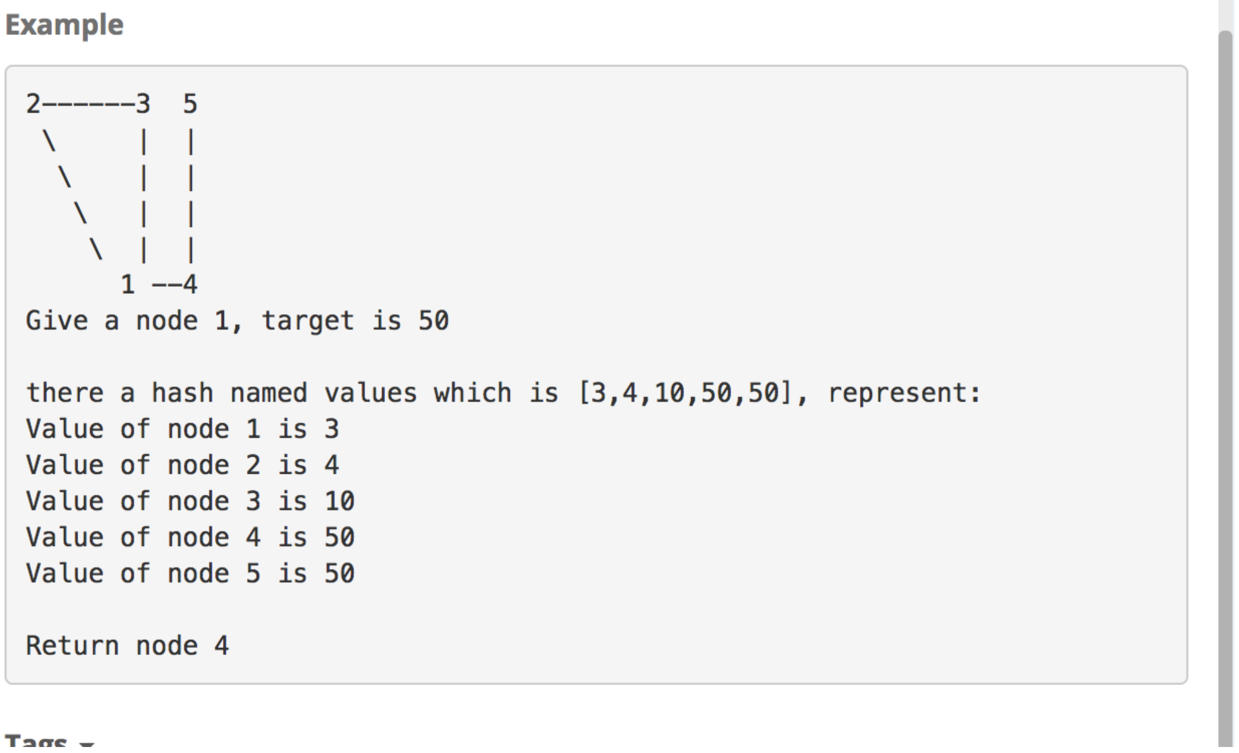
Given a `undirected graph`, a `node` and a `target`, return the nearest node to given node which value of it is target, return `NULL` if you can't find.

There is a `mapping` store the nodes' values in the given parameters.

Notice

It's guaranteed there is only one available solution

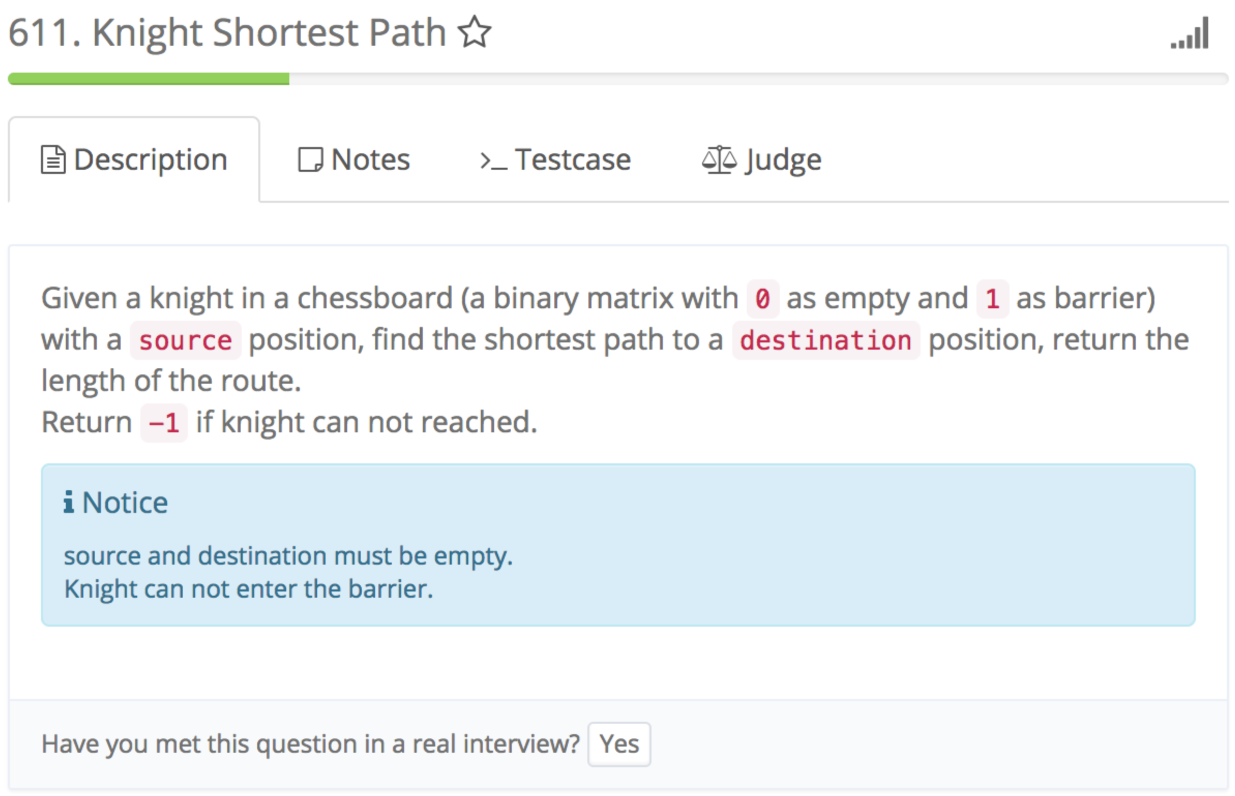
Have you met this question in a real interview? Yes



Example

```
2------3  5
 \     |  |
  \    |  |
   \   |  |
    \  |  |
      1 --4
Give a node 1, target is 50

there a hash named values which is [3,4,10,50,50], represent:
Value of node 1 is 3
Value of node 2 is 4
Value of node 3 is 10
Value of node 4 is 50
Value of node 5 is 50

Return node 4
```

```
/**
 * Definition for graph node.
 * class UndirectedGraphNode {
 *     int label;
 *     ArrayList<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) {
 *         label = x; neighbors = new ArrayList<UndirectedGraphNode>();
 *     }
 * };
 */

public class Solution {
    /**
     * @param graph a list of Undirected graph node
     * @param values a hash mapping, <UndirectedGraphNode, (int)value>
     * @param node an Undirected graph node
     * @param target an integer
     * @return the a node
     */
    public UndirectedGraphNode searchNode(ArrayList<UndirectedGraphNode> graph,
                                          Map<UndirectedGraphNode, Integer> values,
                                          UndirectedGraphNode node,
                                          int target) {
        // Write your code here
        Queue<UndirectedGraphNode> queue = new LinkedList<UndirectedGraphNode>();
        Set<UndirectedGraphNode> hash = new HashSet<UndirectedGraphNode>();

        queue.offer(node);
        hash.add(node);

        while (!queue.isEmpty()) {
            UndirectedGraphNode head = queue.poll();
            if (values.get(head) == target) {
                return head;
            }
            for (UndirectedGraphNode nei : head.neighbors) {
                if (!hash.contains(nei)){
                    queue.offer(nei);
                    hash.add(nei);
                }
            }
        }
        return null;
    }
}
```



611. Knight Shortest Path

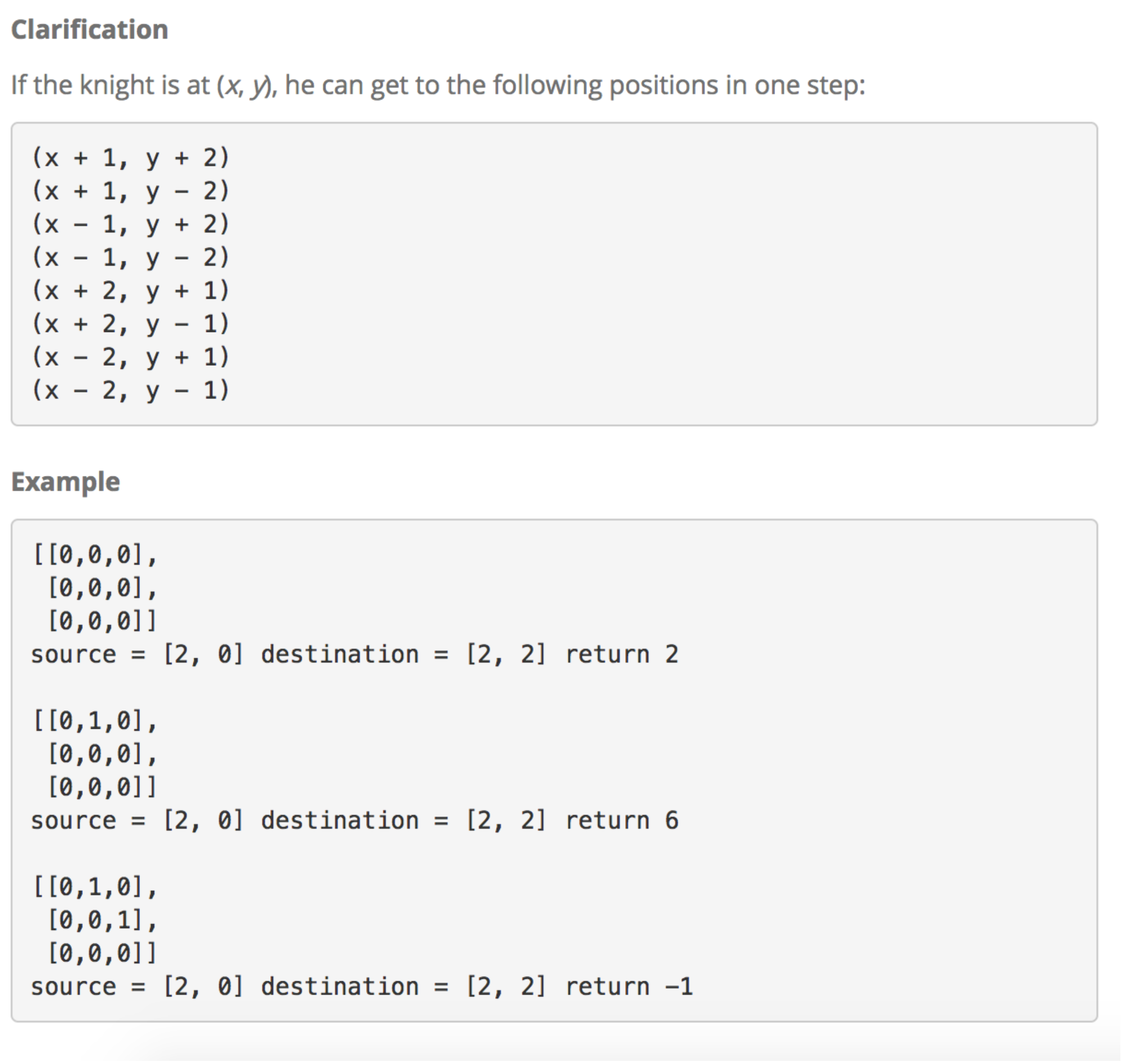
Description | Notes | >_ Testcase | Judge

Given a knight in a chessboard (a binary matrix with `0` as empty and `1` as barrier) with a `source` position, find the shortest path to a `destination` position, return the length of the route.
Return `-1` if knight can not reached.

Notice

source and destination must be empty.
Knight can not enter the barrier.

Have you met this question in a real interview? Yes



Clarification

If the knight is at (*x*, *y*), he can get to the following positions in one step:

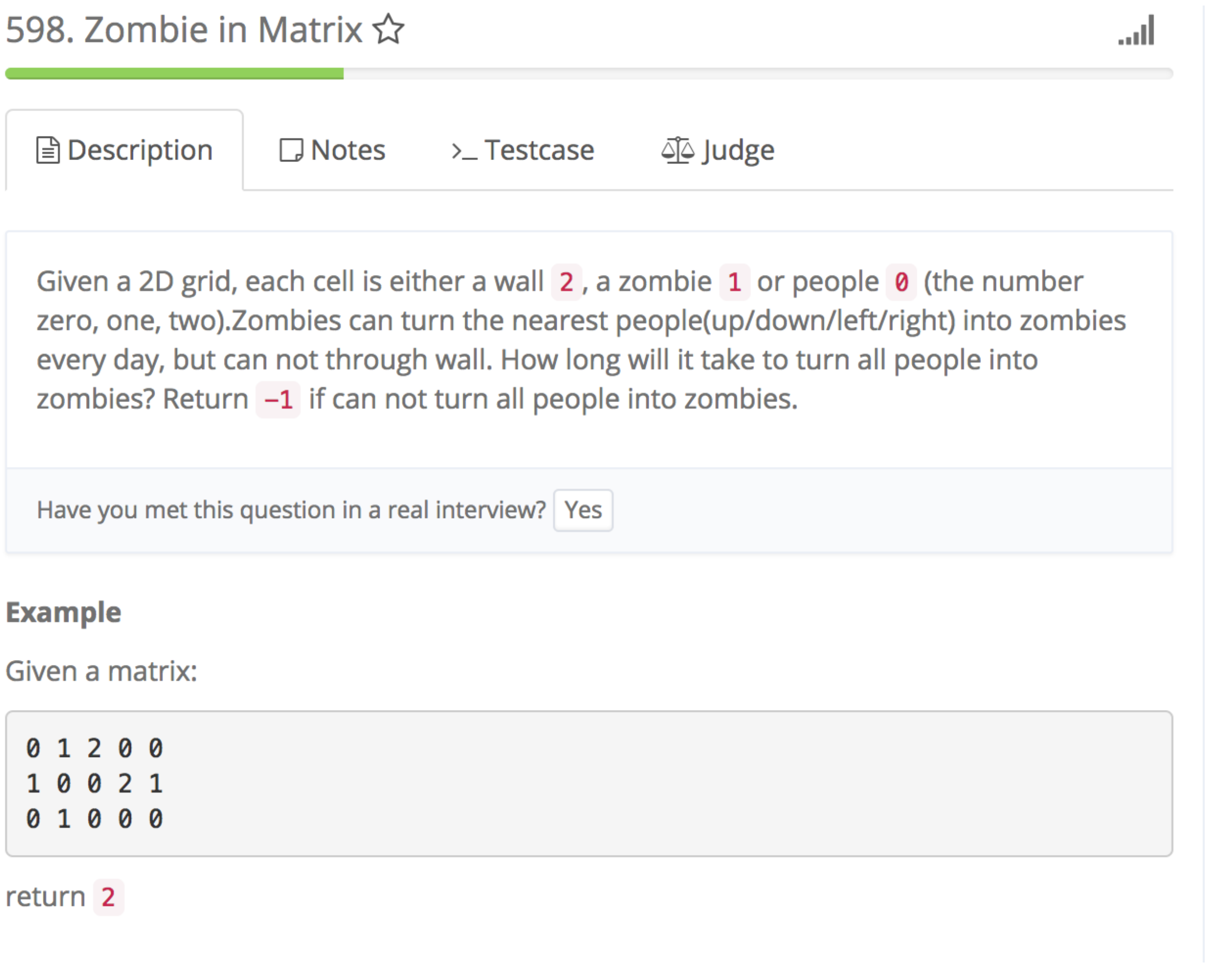
```
(x + 1, y + 2)
(x + 1, y - 2)
(x - 1, y + 2)
(x - 1, y - 2)
(x + 2, y + 1)
(x + 2, y - 1)
(x - 2, y + 1)
(x - 2, y - 1)
```

Example

```
[[0,0,0],
 [0,0,0],
 [0,0,0]]
source = [2, 0] destination = [2, 2] return 2

[[0,1,0],
 [0,0,0],
 [0,0,0]]
source = [2, 0] destination = [2, 2] return 6

[[0,1,0],
 [0,0,1],
 [0,0,0]]
source = [2, 0] destination = [2, 2] return -1
```

```
/**
 * Definition for a point.
 * public class Point {
 *     publoc int x, y;
 *     public Point() { x = 0; y = 0; }
 *     public Point(int a, int b) { x = a; y = b; }
 * }
 */


public class Solution {
    int n, m; // size of the chessboard
    int[] deltaX = {1, 1, 2, 2, -1, -1, -2, -2};
```

```
int[] deltaY = {2, -2, 1, -1, 2, -2, 1, -1};
/**
 * @param grid a chessboard included 0 (false) and 1 (true)
 * @param source, destination a point
 * @return the shortest path
 */
public int shortestPath(boolean[][] grid, Point source, Point destination) {
    if (grid == null || grid.length == 0 || grid[0].length == 0) {
        return -1;
    }

    n = grid.length;
    m = grid[0].length;

    Queue<Point> queue = new LinkedList<>();
    queue.offer(source);

    int steps = 0;
    while (!queue.isEmpty()) {
        int size = queue.size();
        for (int i = 0; i < size; i++) {
            Point point = queue.poll();
            if (point.x == destination.x && point.y == destination.y) {
                return steps;
            }

            for (int direction = 0; direction < 8; direction++) {
                Point nextPoint = new Point(
                    point.x + deltaX[direction],
                    point.y + deltaY[direction]
                );

                if (!inBound(nextPoint, grid)) {
                    continue;
                }

                queue.offer(nextPoint);
                // mark the point not accessible
                grid[nextPoint.x][nextPoint.y] = true;
            }
        }
        steps++;
    }

    return -1;
}

private boolean inBound(Point point, boolean[][] grid) {
    if (point.x < 0 || point.x >= n) {
```


598. Zombie in Matrix ☆

Description | Notes | >_ Testcase | Judge

Given a 2D grid, each cell is either a wall `2`, a zombie `1` or people `0` (the number zero, one, two).Zombies can turn the nearest people(up/down/left/right) into zombies every day, but can not through wall. How long will it take to turn all people into zombies? Return `-1` if can not turn all people into zombies.

Have you met this question in a real interview? Yes

Example

Given a matrix:

```
0 1 2 0 0
1 0 0 2 1
0 1 0 0 0
```

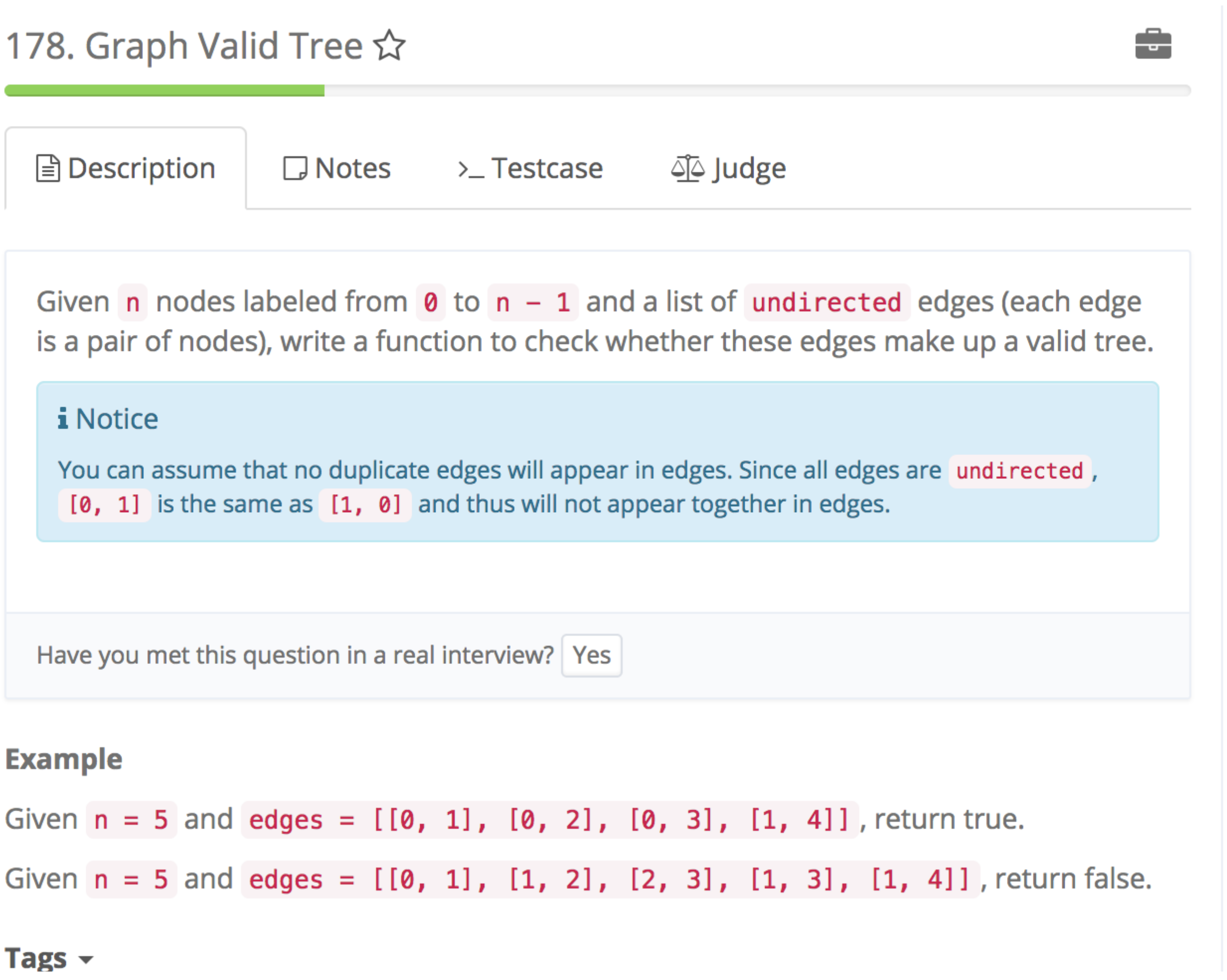
return `2`

```
            return false;
        }
        if (point.y < 0 || point.y >= m) {
            return false;
        }
        return (grid[point.x][point.y] == false);
    }
}
```

```
class Coordinate {
    int x, y;
    public Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Solution {
    public int PEOPLE = 0;
    public int ZOMBIE = 1;
    public int WALL = 2;

    public int[] deltaX = {1, 0, 0, -1};
    public int[] deltaY = {0, 1, -1, 0};
```

```
/**
 * @param grid a 2D integer grid
 * @return an integer
 */
public int zombie(int[][] grid) {
    if (grid == null || grid.length == 0 || grid[0].length == 0) {
        return 0;
    }

    int n = grid.length;
    int m = grid[0].length;

    // initialize the queue & count people
    int people = 0;
    Queue<Coordinate> queue = new LinkedList<>();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == PEOPLE) {
                people++;
            } else if (grid[i][j] == ZOMBIE) {
                queue.offer(new Coordinate(i, j));
            }
        }
    }

    // corner case
    if (people == 0) {
        return 0;
    }

    // bfs
    int days = 0;
    while (!queue.isEmpty()) {
        days++;
        int size = queue.size();
        for (int i = 0; i < size; i++) {
            Coordinate zb = queue.poll();
            for (int direction = 0; direction < 4; direction++) {
                Coordinate adj = new Coordinate(
                    zb.x + deltaX[direction],
                    zb.y + deltaY[direction]
                );

                if (!isPeople(adj, grid)) {
                    continue;
                }

                grid[adj.x][adj.y] = ZOMBIE;
```

```
                    people--;
                    if (people == 0) {
                        return days;
                    }
                    queue.offer(adj);
                }
            }
        }

        return -1;
    }

    private boolean isPeople(Coordinate coor, int[][] grid) {
        int n = grid.length;
        int m = grid[0].length;

        if (coor.x < 0 || coor.x >= n) {
            return false;
        }
        if (coor.y < 0 || coor.y >= m) {
            return false;
        }
        return (grid[coor.x][coor.y] == PEOPLE);
    }
}
```



178. Graph Valid Tree

Description | Notes | >_ Testcase | Judge

Given `n` nodes labeled from `0` to `n - 1` and a list of `undirected` edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

Notice

You can assume that no duplicate edges will appear in edges. Since all edges are `undirected`, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in edges.

Have you met this question in a real interview? Yes

Example

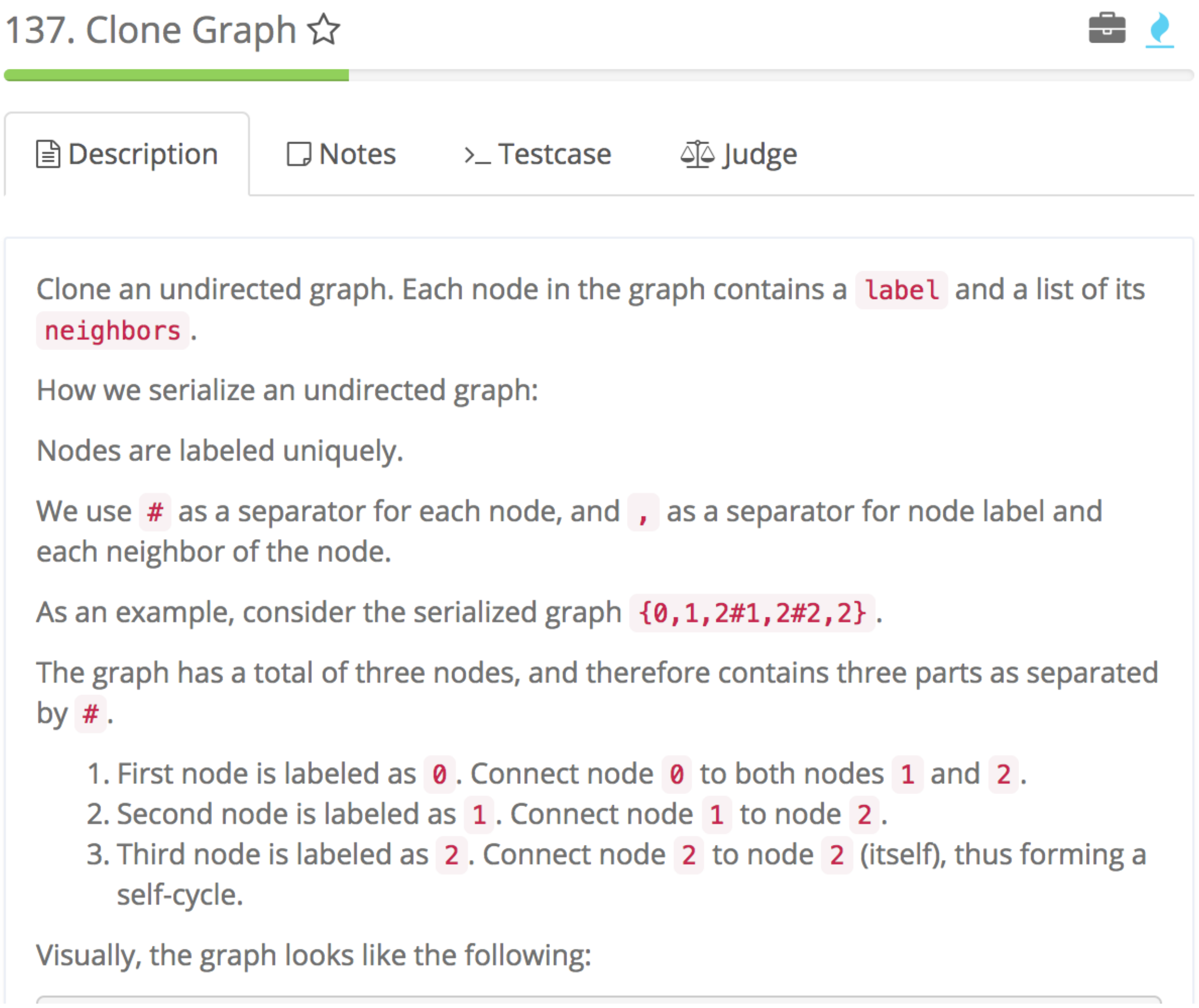
Given `n = 5` and `edges = [[0, 1], [0, 2], [0, 3], [1, 4]]`, return true.

Given `n = 5` and `edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]]`, return false.

Tags

```
public class Solution {
    /**
```

```
 * @param n an integer
 * @param edges a list of undirected edges
 * @return true if it's a valid tree, or false
 */
public boolean validTree(int n, int[][] edges) {
    if (n == 0) {
        return false;
    }

    if (edges.length != n - 1) {
        return false;
    }

    Map<Integer, Set<Integer>> graph = initializeGraph(n, edges);

    // bfs
    Queue<Integer> queue = new LinkedList<>();
    Set<Integer> hash = new HashSet<>();

    queue.offer(0);
    hash.add(0);
    while (!queue.isEmpty()) {
        int node = queue.poll();
        for (Integer neighbor : graph.get(node)) {
            if (hash.contains(neighbor)) {
                continue;
            }
            hash.add(neighbor);
            queue.offer(neighbor);
        }
    }

    return (hash.size() == n);
}

private Map<Integer, Set<Integer>> initializeGraph(int n, int[][] edges) {
    Map<Integer, Set<Integer>> graph = new HashMap<>();
    for (int i = 0; i < n; i++) {
        graph.put(i, new HashSet<Integer>());
    }

    for (int i = 0; i < edges.length; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        graph.get(u).add(v);
        graph.get(v).add(u);
    }

    return graph;
```



137. Clone Graph

Description Notes >_ Testcase Judge

Clone an undirected graph. Each node in the graph contains a `label` and a list of its `neighbors`.

How we serialize an undirected graph:

Nodes are labeled uniquely.

We use `#` as a separator for each node, and `,` as a separator for node label and each neighbor of the node.

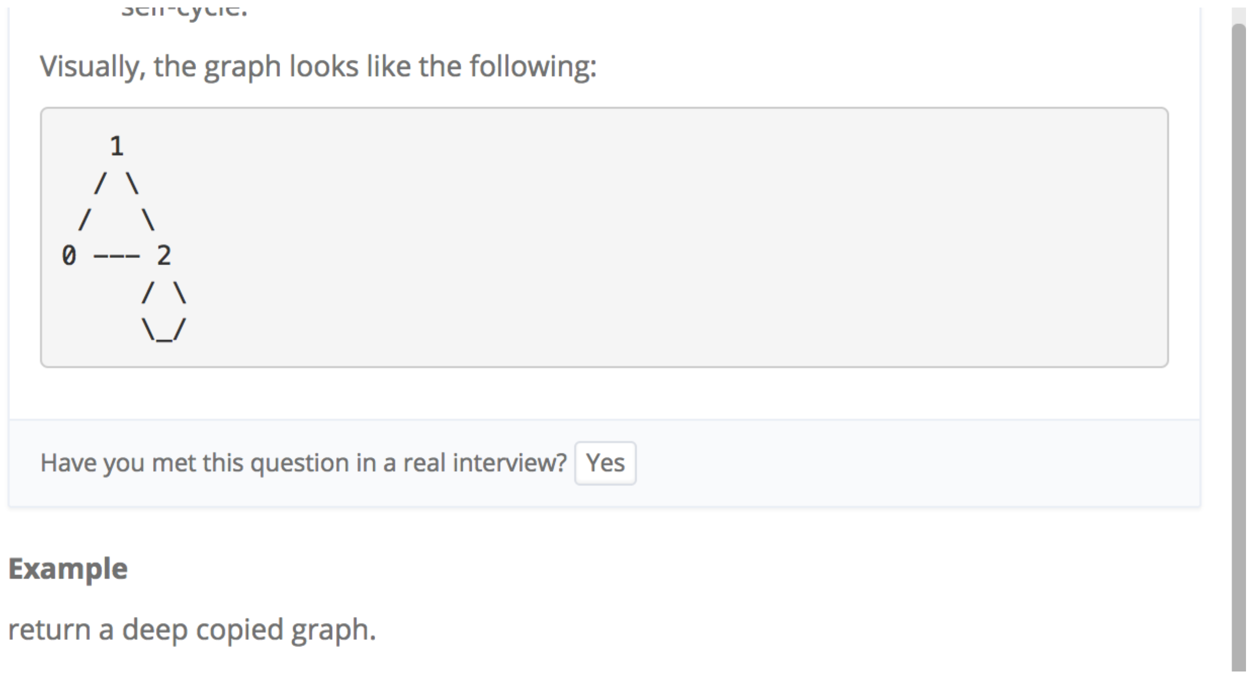
As an example, consider the serialized graph `{0,1,2#1,2#2,2}`.

The graph has a total of three nodes, and therefore contains three parts as separated by `#`.

1. First node is labeled as `0`. Connect node `0` to both nodes `1` and `2`.
2. Second node is labeled as `1`. Connect node `1` to node `2`.
3. Third node is labeled as `2`. Connect node `2` to node `2` (itself), thus forming a self-cycle.

Visually, the graph looks like the following:

```
    }
}
```



self-cycle.

Visually, the graph looks like the following:

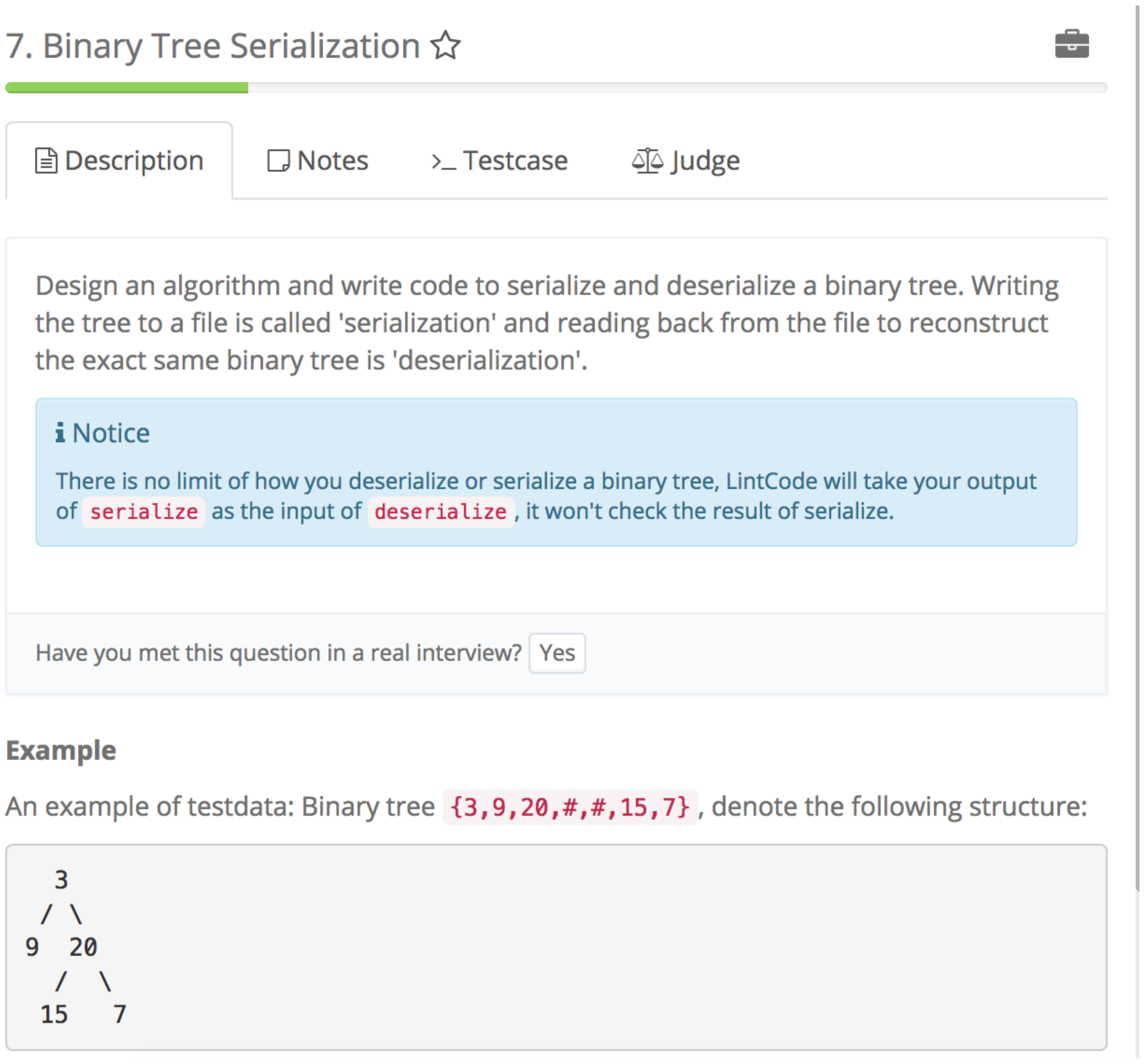
```
     1
    / \
   /   \
  0 --- 2
       / \
       \_/
```

Have you met this question in a real interview? Yes

Example

return a deep copied graph.

```
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     ArrayList<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new
ArrayList<UndirectedGraphNode>(); }
 * };
 */



public class Solution {
    /**
     * @param node: A undirected graph node
     * @return: A undirected graph node
     */
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null) {
            return node;
        }

        // use bfs algorithm to traverse the graph and get all nodes.
        ArrayList<UndirectedGraphNode> nodes = getNodes(node);

        // copy nodes, store the old->new mapping information in a hash map
        HashMap<UndirectedGraphNode, UndirectedGraphNode> mapping = new HashMap<>();
        for (UndirectedGraphNode n : nodes) {
            mapping.put(n, new UndirectedGraphNode(n.label));
        }

        // copy neighbors(edges)
        for (UndirectedGraphNode n : nodes) {
            UndirectedGraphNode newNode = mapping.get(n);
            for (UndirectedGraphNode neighbor : n.neighbors) {
                UndirectedGraphNode newNeighbor = mapping.get(neighbor);
                newNode.neighbors.add(newNeighbor);
            }
        }

        return mapping.get(node);
    }

    private ArrayList<UndirectedGraphNode> getNodes(UndirectedGraphNode node) {
        Queue<UndirectedGraphNode> queue = new LinkedList<UndirectedGraphNode>();
        HashSet<UndirectedGraphNode> set = new HashSet<>();

        queue.offer(node);
        set.add(node);
        while (!queue.isEmpty()) {
            UndirectedGraphNode head = queue.poll();
            for (UndirectedGraphNode neighbor : head.neighbors) {
                if(!set.contains(neighbor)){
                    set.add(neighbor);
                    queue.offer(neighbor);
                }
            }
        }

        return new ArrayList<UndirectedGraphNode>(set);
    }
}
```



7. Binary Tree Serialization

Design an algorithm and write code to serialize and deserialize a binary tree. Writing the tree to a file is called 'serialization' and reading back from the file to reconstruct the exact same binary tree is 'deserialization'.

Notice

There is no limit of how you deserialize or serialize a binary tree, LintCode will take your output of `serialize` as the input of `deserialize`, it won't check the result of serialize.

Have you met this question in a real interview? Yes

Example

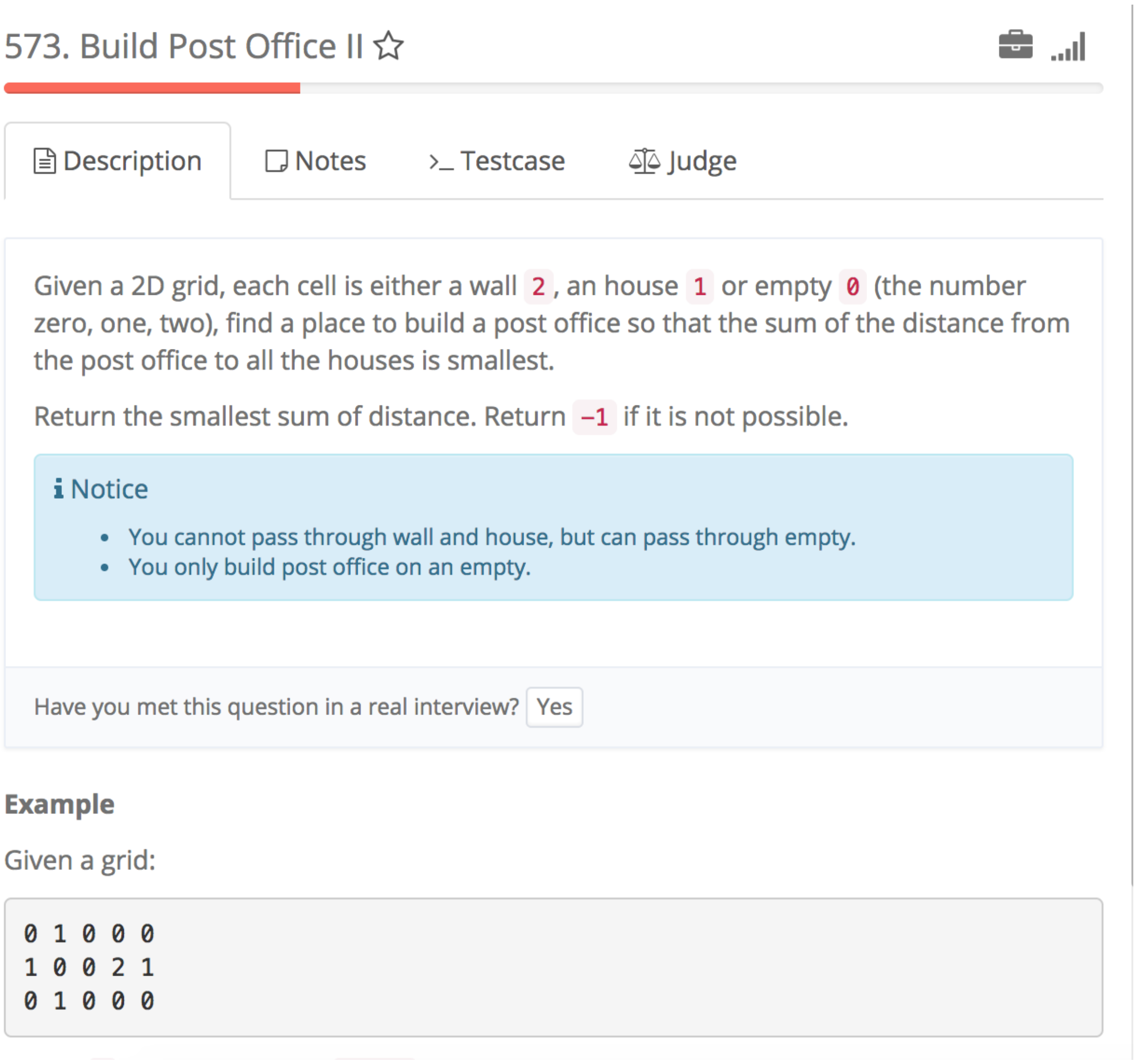
An example of testdata: Binary tree `{3,9,20,#,#,15,7}`, denote the following structure:

```
  3
 / \
9  20
  /  \
 15   7
```

Our data serialization use bfs traversal. This is just for when you got wrong answer and want to debug the input.

You can use other method to do serializaiton and deserialization.

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /**
     * This method will be invoked first, you should design your own algorithm
     * to serialize a binary tree which denote by a root node to a string which
     * can be easily deserialized by your own "deserialize" method later.
     */
    public String serialize(TreeNode root) {
        // write your code here
        if (root == null) {
            return "{}";
        }

        ArrayList<TreeNode> queue = new ArrayList<TreeNode>();
        queue.add(root);

        for (int i = 0; i < queue.size(); i++) {
            TreeNode node = queue.get(i);
            if (node == null) {
                continue;
            }
            queue.add(node.left);
            queue.add(node.right);
        }

        while (queue.get(queue.size() - 1) == null) {
            queue.remove(queue.size() - 1);
        }

        StringBuilder sb = new StringBuilder();
        sb.append("{");
        sb.append(queue.get(0).val);
        for (int i = 1; i < queue.size(); i++) {
            if (queue.get(i) == null) {
                sb.append(",#");
            } else {
                sb.append(",");
                sb.append(queue.get(i).val);
            }
        }
        sb.append("}");
        return sb.toString();
    }

    /**
     * This method will be invoked second, the argument data is what exactly
     * you serialized at method "serialize", that means the data is not given by
     * system, it's given by your own serialize method. So the format of data is
     * designed by yourself, and deserialize it here as you serialize it in
     * "serialize" method.
     */
    public TreeNode deserialize(String data) {
        // write your code here
                if (data.equals("{}")) {
            return null;
        }
        String[] vals = data.substring(1, data.length() - 1).split(",");
        ArrayList<TreeNode> queue = new ArrayList<TreeNode>();
        TreeNode root = new TreeNode(Integer.parseInt(vals[0]));
        queue.add(root);
        int index = 0;
        boolean isLeftChild = true;
        for (int i = 1; i < vals.length; i++) {
            if (!vals[i].equals("#")) {
                TreeNode node = new TreeNode(Integer.parseInt(vals[i]));
                if (isLeftChild) {
                    queue.get(index).left = node;
                } else {
                    queue.get(index).right = node;
                }
                queue.add(node);
            }
            if (!isLeftChild) {
                index++;
            }
            isLeftChild = !isLeftChild;
        }
        return root;
    }
}
```



573. Build Post Office II ☆

Description | Notes | >_ Testcase | Judge

Given a 2D grid, each cell is either a wall `2`, an house `1` or empty `0` (the number zero, one, two), find a place to build a post office so that the sum of the distance from the post office to all the houses is smallest.

Return the smallest sum of distance. Return `-1` if it is not possible.

Notice

- You cannot pass through wall and house, but can pass through empty.
- You only build post office on an empty.

Have you met this question in a real interview? Yes

Example

Given a grid:

```
0 1 0 0 0
1 0 0 2 1
0 1 0 0 0
```

return `8`, You can build at `(1,1)`. (Placing a post office at (1,1), the distance that post office to all the house sum is smallest.)

```
class Coordinate {
    int x, y;
    public Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Solution {
    public int EMPTY = 0;
    public int HOUSE = 1;
    public int WALL = 2;
    public int[][] grid;
    public int n, m;
    public int[] deltaX = {0, 1, -1, 0};
    public int[] deltaY = {1, 0, 0, -1};

    private List<Coordinate> getCoordinates(int type) {
        List<Coordinate> coordinates = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == type) {
                    coordinates.add(new Coordinate(i, j));
                }
            }
        }

        return coordinates;
    }

    private void setGrid(int[][] grid) {
        n = grid.length;
        m = grid[0].length;
        this.grid = grid;
    }

    private boolean inBound(Coordinate coor) {
        if (coor.x < 0 || coor.x >= n) {
            return false;
        }
        if (coor.y < 0 || coor.y >= m) {
            return false;
        }
    return grid[coor.x][coor.y] == EMPTY;
}

/**
 * @param grid a 2D grid
 * @return an integer
 */
public int shortestDistance(int[][] grid) {
    if (grid == null || grid.length == 0 || grid[0].length == 0) {
        return -1;
    }

    // set n, m, grid
    setGrid(grid);

    List<Coordinate> houses = getCoordinates(HOUSE);
    int[][] distanceSum = new int[n][m];;
    int[][] visitedTimes = new int[n][m];;
    for (Coordinate house : houses) {
        bfs(house, distanceSum, visitedTimes);
    }

    int shortest = Integer.MAX_VALUE;
    List<Coordinate> empties = getCoordinates(EMPTY);
    for (Coordinate empty : empties) {
        if (visitedTimes[empty.x][empty.y] != houses.size()) {
            continue;
        }

        shortest = Math.min(shortest, distanceSum[empty.x][empty.y]);
    }

    if (shortest == Integer.MAX_VALUE) {
        return -1;
    }
    return shortest;
}

private void bfs(Coordinate start,
             int[][] distanceSum,
             int[][] visitedTimes) {
    Queue<Coordinate> queue = new LinkedList<>();
    boolean[][] hash = new boolean[n][m];

    queue.offer(start);
    hash[start.x][start.y] = true;

    int steps = 0;
    while (!queue.isEmpty()) {
            steps++;
            int size = queue.size();
            for (int temp = 0; temp < size; temp++) {
                Coordinate coor = queue.poll();
                for (int i = 0; i < 4; i++) {
                    Coordinate adj = new Coordinate(
                        coor.x + deltaX[i],
                        coor.y + deltaY[i]
                    );
                    if (!inBound(adj)) {
                        continue;
                    }
                    if (hash[adj.x][adj.y]) {
                        continue;
                    }
                    queue.offer(adj);
                    hash[adj.x][adj.y] = true;
                    distanceSum[adj.x][adj.y] += steps;
                    visitedTimes[adj.x][adj.y]++;
                } // direction
            } // for temp
        } // while
    }
}
```