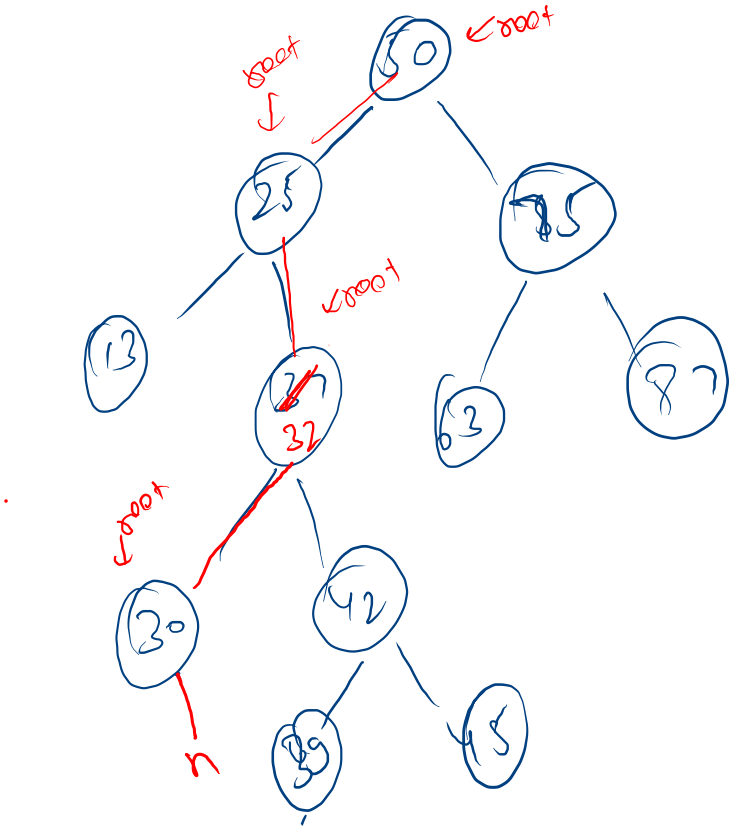
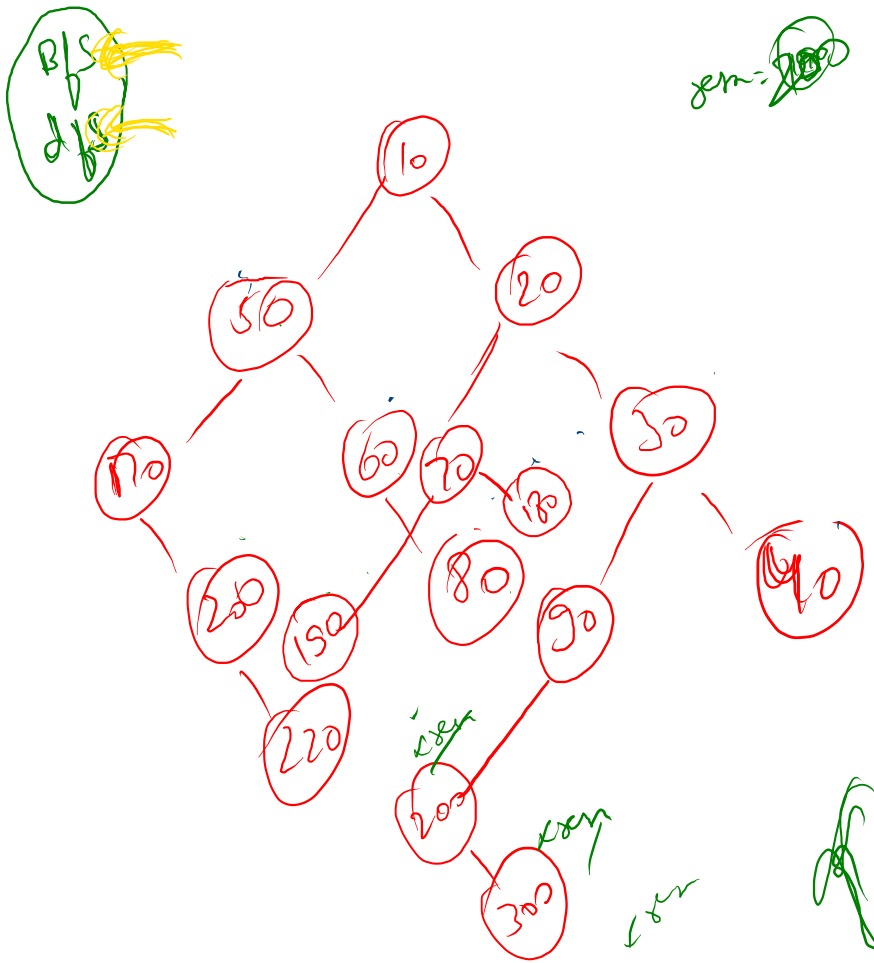
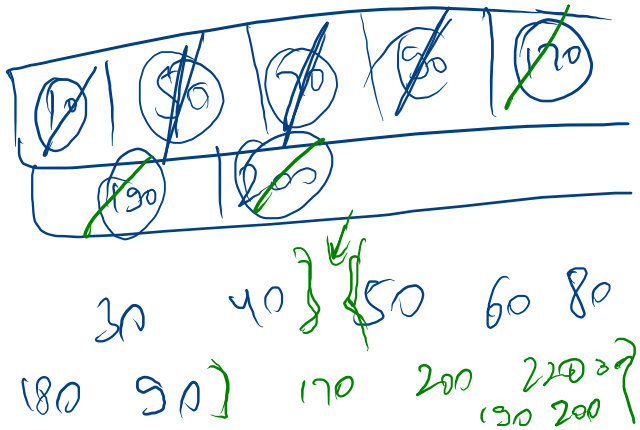



```java
public static Node deleteNode(Node root, int key) {

    if (root.data < key) {
        root.right = deleteNode(root.right, key);
    } else if (root.data > key) {
        root.left = deleteNode(root.left, key);
    } else {
        if (root.left == null && root.right == null) {
            return null;
        } else if (root.left == null) {
            return root.right;
        } else if (root.right == null) {
            return root.left;
        } else {

            Node rootp1 = root.left;
            while (rootp1.right != null) {
                rootp1 = rootp1.right;
            }

            root.data = rootp1.data;
            root.left = deleteNode(root.left, rootp1.data);

        }

    }

    root.height = Math.max(height(root.left), height(root.right)) + 1;

    int diff = difference(root);

    if (diff > 1 && difference(root.left) >= 0) { // LL
        return rightRotate(root);
    } else if (diff > 1 && difference(root.left) < 0) { // LR
        root.left = leftRotate(root.left);
        return rightRotate(root);
    } else if (diff < -1 && difference(root.right) <= 0) { // RR
        return leftRotate(root);
    } else if (diff < -1 && difference(root.right) > 0) { // RL
        root.right = rightRotate(root.right);
        return leftRotate(root);
    }

    return root;

}
```



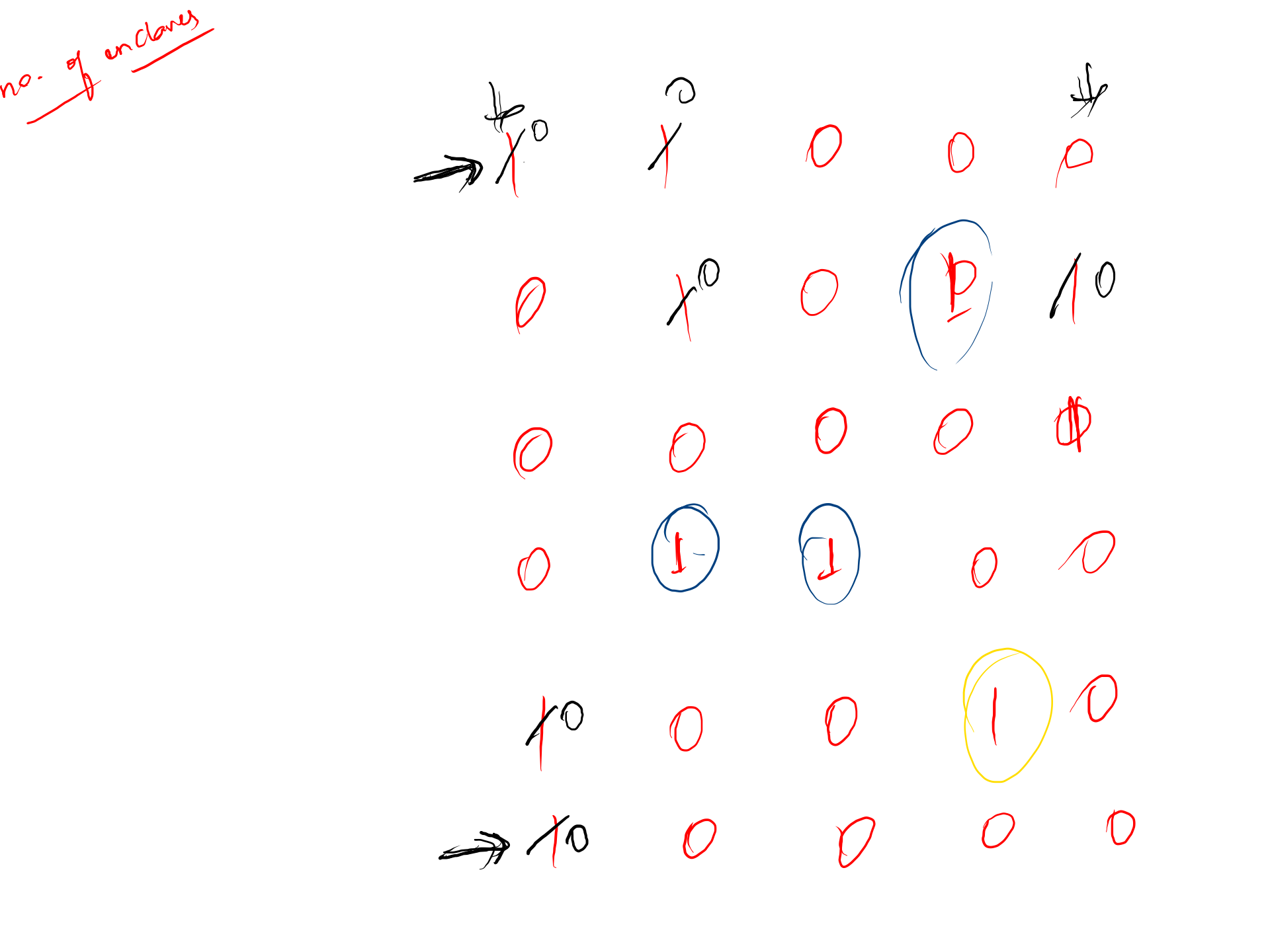

```java
public ArrayList<Integer> diagonal(TreeNode root) {
    // add your code here.
    ArrayList<Integer> ans = new ArrayList<>();
    LinkedList<TreeNode> queue = new LinkedList<>();

    queue.addLast(root);

    while (queue.size() > 0) {
        TreeNode rem = queue.removeFirst();

        while (rem != null) {
            ans.add(rem.data);

            if (rem.left != null) {
                queue.addLast(rem.left);
            }

            rem = rem.right;
        }
    }
```

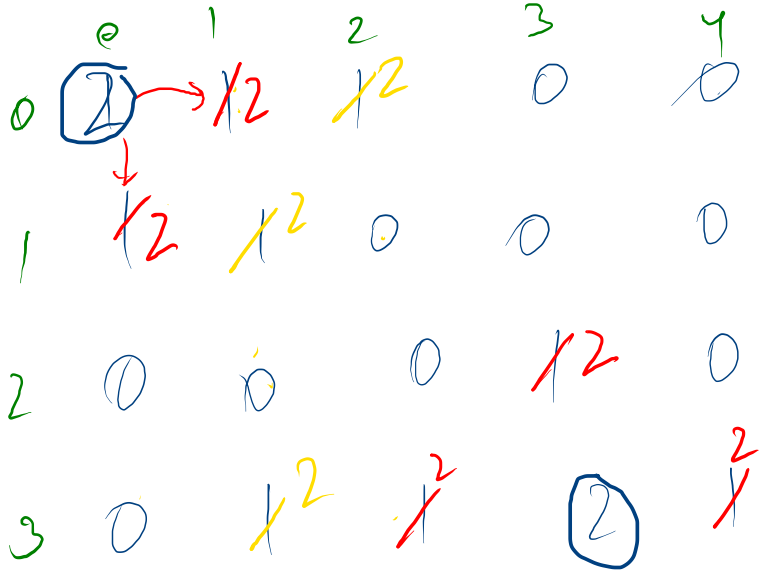


St: XRD222

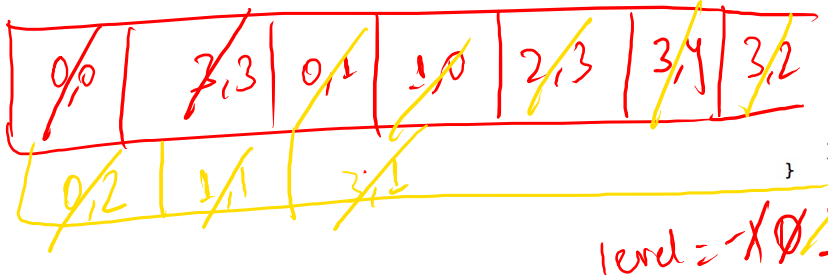
XR2D22

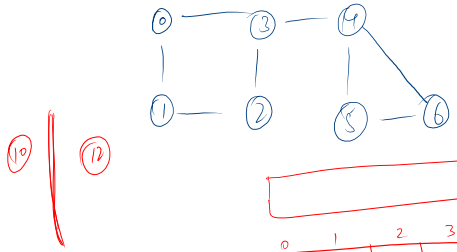
	0	1	2	3
0	10	10	0	0
1	10	0	0	0
2	0	0	10	10
3	0	0	0	10
4	1	1	0	0

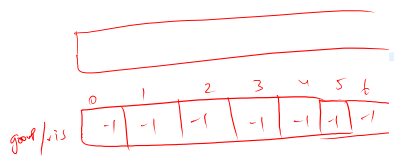

no. of enclaves



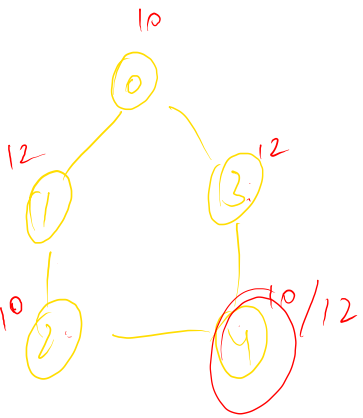
```
public int orangesRotting(int[][] grid) {
    LinkedList<Pair> queue = new LinkedList<>();

    int fresh = 0;
    for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[0].length; j++) {
            if (grid[i][j] == 2) {
                queue.addLast(new Pair(i, j));
            } else if (grid[i][j] == 1) {
                fresh++;
            }
        }
    }

    if (fresh == 0) {
        return 0;
    }

int[][] dirs = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };
int level = -1;
while (queue.size() > 0) {
    level++;
    int size = queue.size();

    while (size-- > 0) {
        Pair rem = queue.removeFirst();

        for (int i = 0; i < dirs.length; i++) {
            int rowdash = rem.row + dirs[i][0];
            int coldash = rem.col + dirs[i][1];

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

            queue.addLast(new Pair(rowdash, coldash));
            fresh--;
            grid[rowdash][coldash] = 2;
        }
    }
}
```

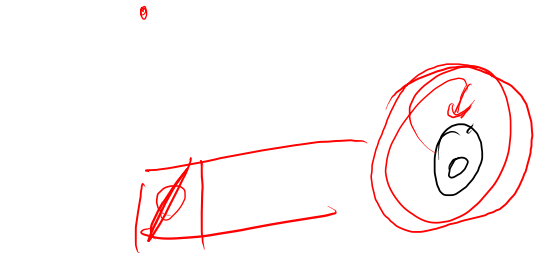


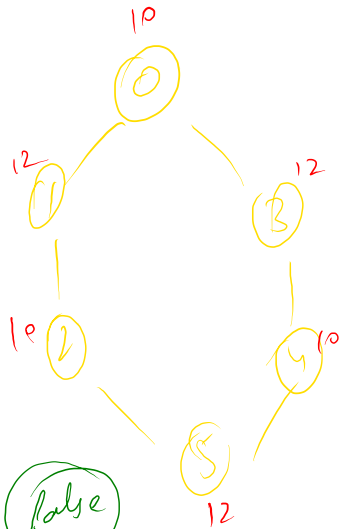


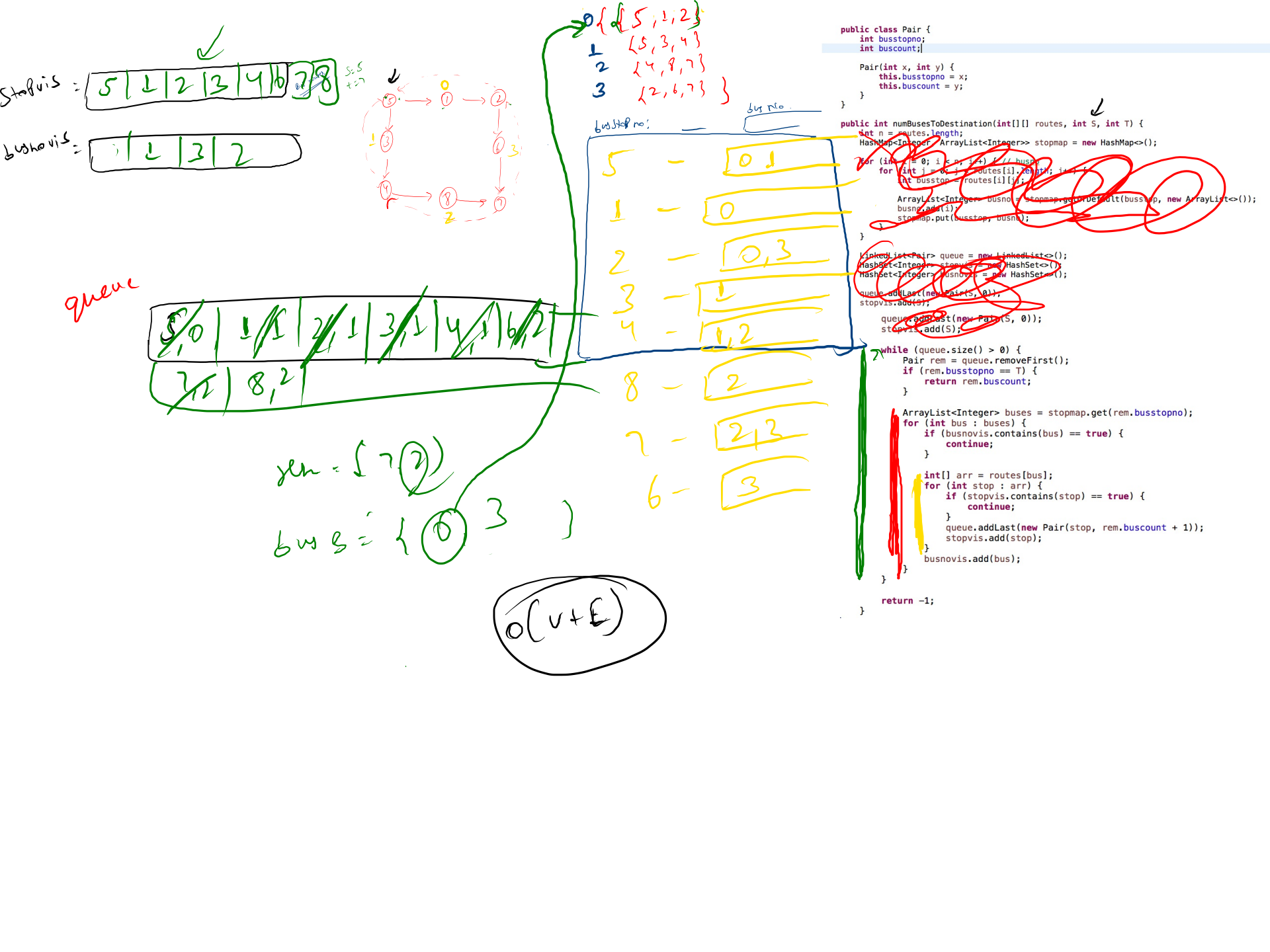



```java
public boolean isBipartite(int[][] graph) {
    int n = graph.length;

    for (int i = 0; i < n; i++) {
        LinkedList<Integer> queue = new LinkedList<>();

        int[] vis = new int[n];
        Arrays.fill(vis, -1);

        queue.addLast(i);
        vis[i] = 10;

        while (queue.size() > 0) {
            int rem = queue.removeFirst();

            int[] nbrs = graph[rem];
            for (int nbr : nbrs) {
                if (vis[nbr] == -1) { // unvisited
                    queue.addLast(nbr);
                    if (vis[rem] == 10) {
                        vis[nbr] = 12;
                    } else { // vis[rem] = 12
                        vis[nbr] = 10;
                    }

                } else {
                    if (vis[rem] == vis[nbr]) {
                        return false;
                    }
                }
            }
        }
    }
```



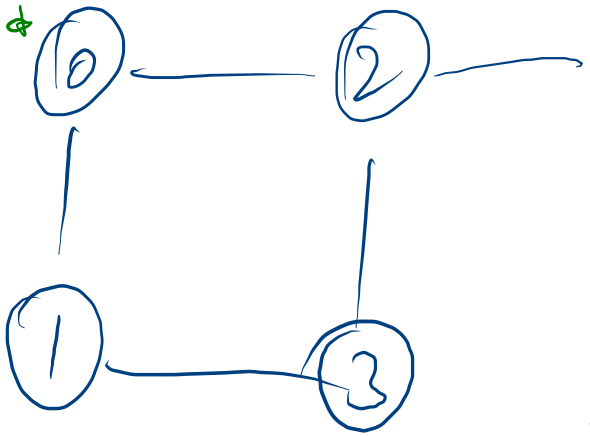


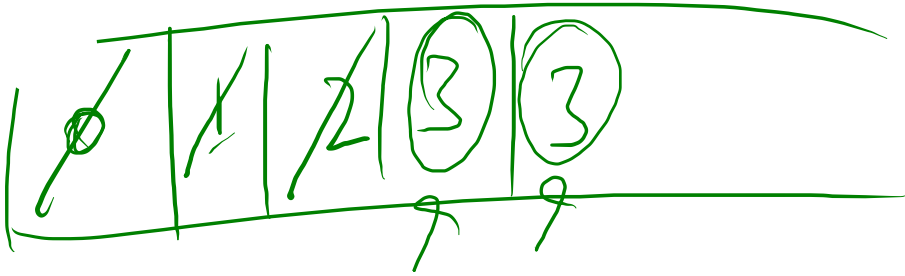





```java
public class Pair {
    int busstopno;
    int buscount;

    Pair(int x, int y) {
        this.busstopno = x;
        this.buscount = y;
    }
}

public int numBusesToDestination(int[][] routes, int S, int T) {
    int n = routes.length;
    HashMap<Integer, ArrayList<Integer>> stopmap = new HashMap<>();
    for (int i = 0; i < n; i++) { // bus
        for (int j = 0; j < routes[i].length; j++) {
            int busstop = routes[i][j];
            ArrayList<Integer> busno = stopmap.getOrDefault(busstop, new ArrayList<>());
            busno.add(i);
            stopmap.put(busstop, busno);
        }
    }

    LinkedList<Pair> queue = new LinkedList<>();
    HashSet<Integer> stopvis = new HashSet<>();
    HashSet<Integer> busnovis = new HashSet<>();

    queue.addLast(new Pair(S, 0));
    stopvis.add(S);

    queue.addLast(new Pair(S, 0));
    stopvis.add(S);

    while (queue.size() > 0) {
        Pair rem = queue.removeFirst();
        if (rem.busstopno == T) {
            return rem.buscount;
        }

        ArrayList<Integer> buses = stopmap.get(rem.busstopno);
        for (int bus : buses) {
            if (busnovis.contains(bus) == true) {
                continue;
            }

            int[] arr = routes[bus];
            for (int stop : arr) {
                if (stopvis.contains(stop) == true) {
                    continue;
                }
                queue.addLast(new Pair(stop, rem.buscount + 1));
                stopvis.add(stop);
            }
            busnovis.add(bus);
        }
    }

    return -1;
}
```


10^5


0
2
1
3


0
1
2
3
3


0
1
2