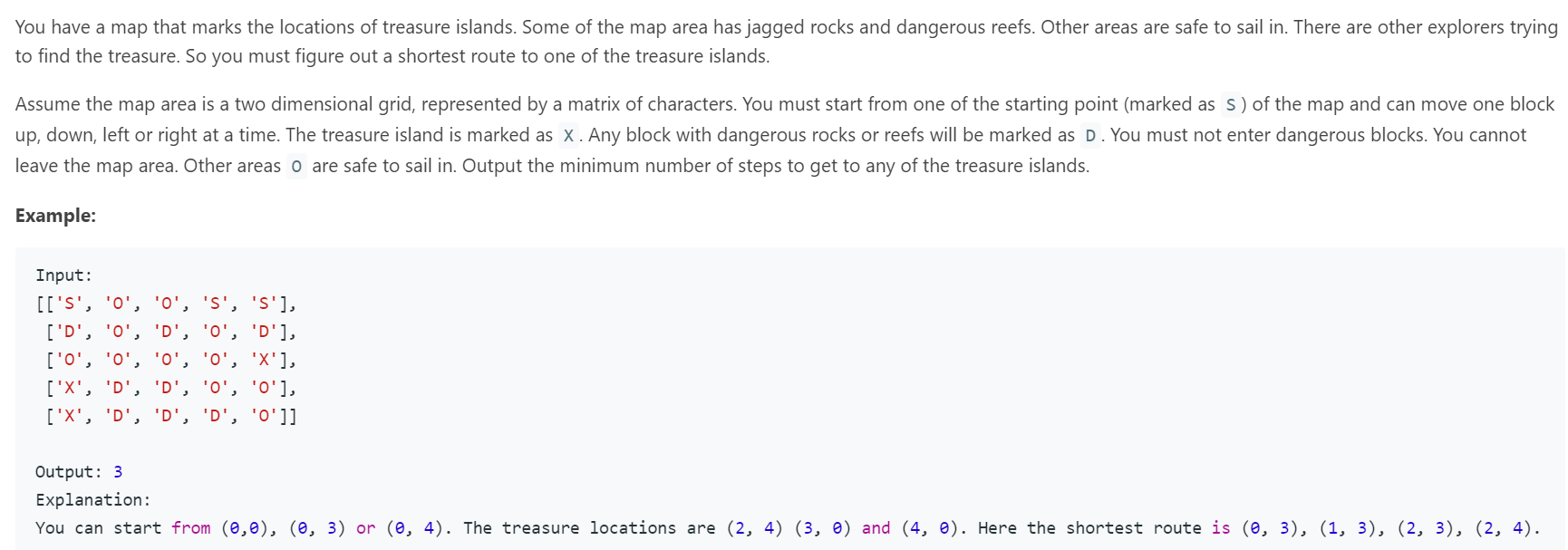
Amazon | OA 2019 | Treasure Island II

<https://leetcode.com/discuss/interview-question/356150>



// Amazon | OA 2019 | Shortest Path From Multiple Sources

// https://leetcode.com/discuss/interview-question/356150

public class Main {

final int[][] dirs = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};

public int shortestPath(char[][] islands){

if(islands.length == 0 || islands[0].length == 0) return -1;

int R = islands.length, C = islands[0].length;

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

int steps = 1;

// add all sources to queue and set 'visited'.

for(int i = 0; i < R; ++i){

for(int j = 0; j < C; ++j){

if(islands[i][j] == 'S'){

queue.add(new int[]{i, j}); islands[i][j] = 'D';

}

}

}

while(!queue.isEmpty()){

int size = queue.size();

for(int i = 0; i < size; ++i){

int[] pos = queue.poll();

for(int[] dir: dirs){

int x = pos[0] + dir[0], y = pos[1] + dir[1];

if(x < 0 || x >= R || y < 0 || y >= C || islands[x][y] == 'D') continue;

if(islands[x][y] == 'E') return steps;

queue.add(new int[]{x, y});

islands[x][y] = 'D';

}

}

++steps;

}

return -1;

}

public static void main(String[] args) {

Main main = new Main();

char[][] testcase = { {'S', 'O', 'O', 'S'},

{'D', 'O', 'D', 'D'},

{'O', 'O', 'O', 'E'},

{'E', 'D', 'D', 'O'}};

System.out.println(main.shortestPath(testcase));

}

}

ALTERNATE :

public class Main {

public static void main(String[] args) {

Main main=new Main();

char[][] grid1=new char[][]{{'S', 'O', 'O', 'S', 'S'},

{'D', 'O', 'D', 'O', 'D'},

{'O', 'O', 'O', 'O', 'X'},

{'X', 'D', 'D', 'O', 'O'},

{'X', 'D', 'D', 'D', 'O'}};

char[][] grid2=new char[][]{{'S', 'O', 'O', 'S', 'S'},

{'D', 'O', 'D', 'O', 'O'},

{'O', 'O', 'O', 'O', 'X'},

{'X', 'D', 'D', 'O', 'O'},

{'X', 'D', 'D', 'D', 'O'}};

int tc1 = main.shortestPathToTreasureIsland(grid1);

int tc2 = main.shortestPathToTreasureIsland(grid2);

if(tc1==3 && tc2==2) {

System.out.println("All Test Case Passes!");

} else {

System.out.println("There are test failures!");

}

}

private int shortestPathToTreasureIsland(char[][] grid) {

if(grid==null || grid.length==0) return 0;

int row=grid.length, col=grid[0].length, minStep=Integer.MAX\_VALUE;

for(int i=0; i<row; i++) {

for(int j=0; j<col; j++) {

if(grid[i][j]=='S') {

minStep=Math.min(minStep, this.bfs(grid, i, j));

}

}

}

return minStep;

}

private int bfs(char[][] grid, int row, int col) {

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

queue.offer(new int[]{row, col});

int steps=0;

boolean[][] visited=new boolean[grid.length][grid[0].length];

int[][] directions=new int[][]{{0,1},{0,-1},{1,0},{-1,0}};

while(!queue.isEmpty()) {

int size=queue.size();

while(size>0) {

int[] coor=queue.poll();

int x=coor[0], y=coor[1];

if(grid[x][y]=='X') return steps;

visited[x][y]=true;

for(int i=0; i<directions.length; i++) {

int dx=x+directions[i][0];

int dy=y+directions[i][1];

if(dx>=0 && dx<grid.length && dy>=0 && dy<grid[0].length && !visited[dx][dy] && grid[dx][dy]!='D') {

queue.offer(new int[]{dx,dy});

}

}

size--;

}

steps++;

}

return -1;

}

}