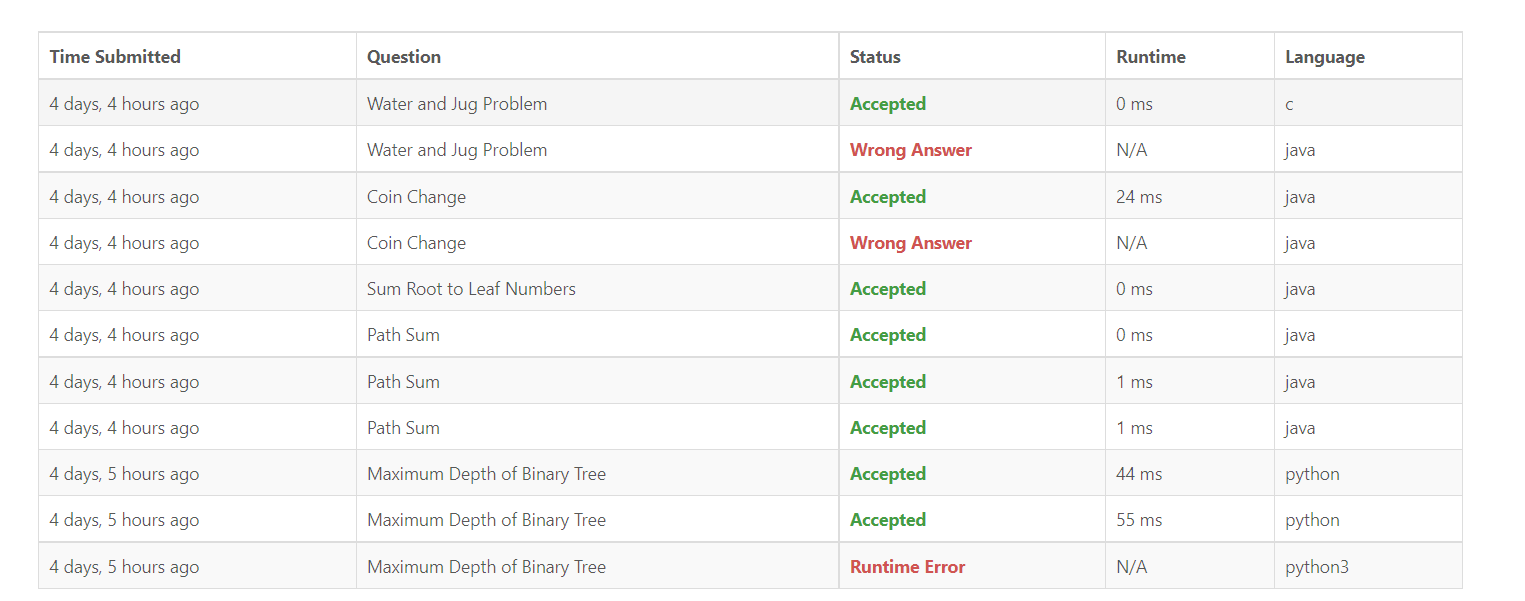
2100032489\_G SURYA SIVA TEJA11111

LEETCODE



Water and jug problem:

class Solution(object):

def maxDepth(self, root):

return 1 + max(self.maxDepth(root.left), self.maxDepth(root.right)) if root else 0

coin change:

class Solution {

public int coinChange(int[] coins, int amount) {

int max = amount + 1;

int[] dp = new int[amount + 1];

Arrays.fill(dp, max);

dp[0] = 0;

for (int i = 1; i <= amount ; i++) {

for (int j = 0; j < coins.length; j++) {

if (coins[j] <= i) {

dp[i] = Math.min(dp[i], 1 + dp[i - coins[j]]);

}

}

}

return dp[amount] > amount ? -1 : dp[amount];

}

}

Sum root to leaf numbers:

class Solution {

private int res;

public int sumNumbers(TreeNode root) {

help(root, 0);

return res;

}

private void help(TreeNode node, int num) {

if (node == null) {

return;

}

num = num \* 10 + node.val;

if (node.left == null && node.right == null) {

res += num;

}

help(node.left, num);

help(node.right, num);

}

}

Path sum:

class Solution {

public boolean hasPathSum(TreeNode root, int sum) {

if(root == null) return false;

if(root.left == null && root.right == null){

return root.val == sum;

}

return hasPathSum(root.left, sum-root.val) || hasPathSum(root.right, sum-root.val);

}

}

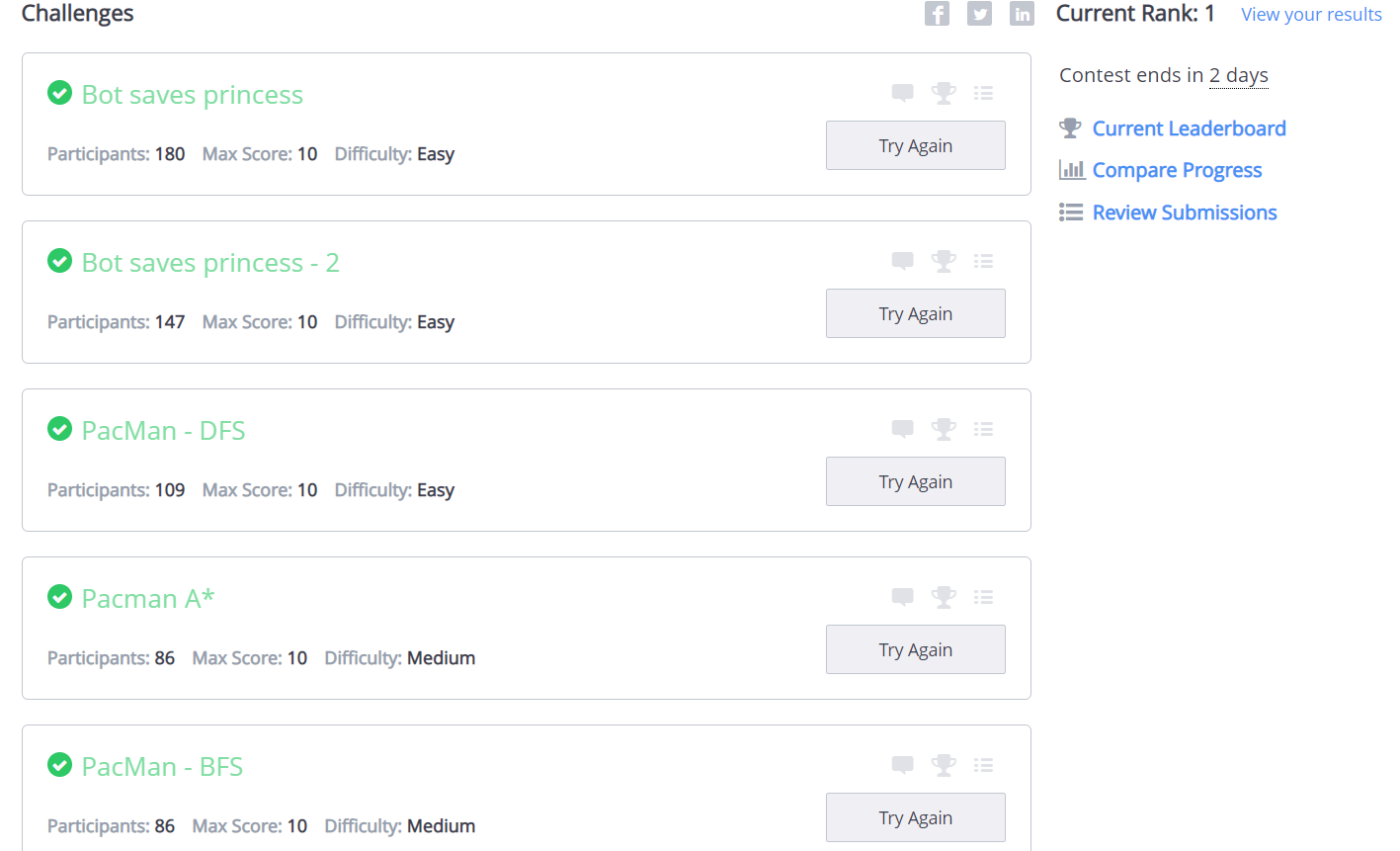
Maximum depth of binary tree:

class Solution(object):

def maxDepth(self, root):

return 1 + max(self.maxDepth(root.left), self.maxDepth(root.right)) if root else 0

HACKERRANK



Bot saves princess:

#include <stdio.h>

#include <string.h>

#include <math.h>

void displayPathtoPrincess(int n, char grid[n][n]){

int i, j, up, left;

short prince[2], princess[2];

for(i = 0; i < n; ++i) {

for(j = 0; j < n; ++j) {

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

prince[0] = i;

prince[1] = j;

}

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

princess[0] = i;

princess[1] = j;

}

}

}

if ((up = princess[0] - prince[0]) < 0) {

for(; up < 0; ++up)

printf("UP\n");

} else {

for(; up > 0; --up)

printf("DOWN\n");

}

if ((left = princess[1] - prince[1]) < 0) {

for(; left < 0; ++left)

printf("LEFT\n");

} else {

for(; left > 0; --left)

printf("RIGHT\n");

}

}

int main() {

int m;

scanf("%d", &m);

char grid[m][m];

char line[m];

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

scanf("%s", line);

strncpy(grid[i], line, m);

}

displayPathtoPrincess(m, grid);

return 0;

}

Bot saves princess-2:

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) {

int [][] a = new int[2][2];

Scanner sc = new Scanner(System.in);

int n;

n = sc.nextInt();

a[0][0]=sc.nextInt();

a[0][1]=sc.nextInt();

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

String S = sc.next();

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

int c = (int)S.charAt(j);

if(c==109)

{

a[0][0]=i;

a[0][1]=j;

}

if(c==112)

{

a[1][0]=i;

a[1][1]=j;

}

}

}

int mI = a[0][0];

int mJ = a[0][1];

int pI = a[1][0];

int pJ = a[1][1];

int dif =(mI -pI);

boolean printS= true;

if(dif>0 && printS)

{

System.out.println("UP");

printS= false;

}

if(dif<0&& printS)

{

System.out.println("DOWN");

printS= false;

}

dif =(mJ -pJ);

if(dif>0&& printS){

System.out.println("LEFT");

printS= false;

}

if(dif<0&& printS){

System.out.println("RIGHT");

}

}

}

PacMan-DFS:

import copy

pacman\_x, pacman\_y = list(map(int, input().split()))

food\_x, food\_y = list(map(int, input().split()))

n, m = list(map(int, input().split()))

grid = []

node\_expanded = []

stack = []

answer\_routes = None

for i in range(0, n):

grid.append(list(map(str, input())))

directions = [[-1, 0], [0, -1], [0, 1], [1, 0]]

stack.append([pacman\_x, pacman\_y, []])

while len(stack) > 0:

x, y, r = stack.pop()

routes = copy.deepcopy(r)

routes.append([x, y])

node\_expanded.append([x, y])

if x == food\_x and y == food\_y:

if answer\_routes == None:

answer\_routes = routes

break

for direction in directions:

next\_x, next\_y = x + direction[0], y + direction[1]

if next\_x < 0 or next\_x >= n or next\_y < 0 and next\_y >= n:

continue

if grid[next\_x][next\_y] == "-" or grid[next\_x][next\_y] == ".":

grid[next\_x][next\_y] = '='

stack.append([next\_x, next\_y, routes])

print(str(len(node\_expanded)))

for point in node\_expanded:

print(str(point[0]) + " " + str(point[1]))

print(str(len(answer\_routes) - 1))

for point in answer\_routes:

print(str(point[0]) + " " + str(point[1]))

PacMan A\*:

import copy

pacman\_x, pacman\_y = list(map(int, input().split()))

food\_x, food\_y = list(map(int, input().split()))

n, m = list(map(int, input().split()))

grid = []

queue = []

answer\_routes = None

for i in range(0, n):

grid.append(list(map(str, input())))

directions = [[-1, 0], [0, -1], [0, 1], [1, 0]]

queue.append([pacman\_x, pacman\_y, [], 0])

while len(queue) > 0:

x, y, r, score = queue.pop(0)

routes = copy.deepcopy(r)

routes.append([x, y])

if x == food\_x and y == food\_y:

if answer\_routes == None:

answer\_routes = routes

break

possible\_moves = []

for direction in directions:

next\_x, next\_y = x + direction[0], y + direction[1]

if next\_x < 0 or next\_x >= n or next\_y < 0 and next\_y >= n:

continue

if grid[next\_x][next\_y] == "-" or grid[next\_x][next\_y] == ".":

grid[next\_x][next\_y] = '='

possible\_moves.append([next\_x, next\_y, score + abs(food\_x - next\_x) + abs(food\_y - next\_y)])

possible\_moves.sort(key = lambda x: x[2])

for move in possible\_moves:

queue.append([move[0], move[1], routes, score])

print(str(len(answer\_routes) - 1))

for point in answer\_routes:

print(str(point[0]) + " " + str(point[1]))

PacMan-BFS:

import copy

pacman\_x, pacman\_y = list(map(int, input().split()))

food\_x, food\_y = list(map(int, input().split()))

n, m = list(map(int, input().split()))

grid = []

node\_expanded = []

queue = []

answer\_routes = None

for i in range(0, n):

grid.append(list(map(str, input())))

directions = [[-1, 0], [0, -1], [0, 1], [1, 0]]

queue.append([pacman\_x, pacman\_y, []])

while len(queue) > 0:

x, y, r = queue.pop(0)

routes = copy.deepcopy(r)

routes.append([x, y])

node\_expanded.append([x, y])

if x == food\_x and y == food\_y:

if answer\_routes == None:

answer\_routes = routes

break

for direction in directions:

next\_x, next\_y = x + direction[0], y + direction[1]

if next\_x < 0 or next\_x >= n or next\_y < 0 and next\_y >= n:

continue

if grid[next\_x][next\_y] == "-" or grid[next\_x][next\_y] == ".":

grid[next\_x][next\_y] = '='

queue.append([next\_x, next\_y, routes])

print(str(len(node\_expanded)))

for point in node\_expanded:

print(str(point[0]) + " " + str(point[1]))

print(str(len(answer\_routes) - 1))

for point in answer\_routes:

print(str(point[0]) + " " + str(point[1]))

HACKERRANK LEADERBOARD

