

// My code (TLE)

class Solution:

def wallsAndGates(self, rooms: List[List[int]]) -> None:

"""

Do not return anything, modify rooms in-place instead.

"""

from collections import deque as queue

M, N = len(rooms), len(rooms[0])

dir\_r = [-1, 0, 1, 0]

dir\_c = [0, 1, 0, -1]

for r in range(M):

for c in range(N):

if rooms[r][c] == 2147483647:

q = queue()

visited = [[False for i in range(N)] for i in range(M)]

step = 0

q.append([r,c])

visited[r][c] = True

while len(q) > 0:

point = q.popleft()

x, y = point[0], point[1]

if rooms[x][y] == 0:

rooms[r][c] = abs(x-r) + abs(y-c)

break

for i in range(4):

adj\_x, adj\_y = x + dir\_r[i], y + dir\_c[i]

if adj\_x < 0 or adj\_y < 0 or adj\_x >= M or adj\_y >= N or rooms[adj\_x][adj\_y] == -1: continue

if visited[adj\_x][adj\_y]: continue

q.append([adj\_x, adj\_y])

visited[adj\_x][adj\_y] = True

step += 1

return rooms

Instead of searching from an empty room to the gates, how about searching the other way round? In other words, we initiate breadth-first search (BFS) from all gates at the same time. Since BFS guarantees that we search all rooms of distance d before searching rooms of distance d + 1, the distance to an empty room must be the shortest.

// Inspired, Brilliant solution

// start searching from all gates together

// most of points become visited quickly

// code

class Solution:

def wallsAndGates(self, rooms: List[List[int]]) -> None:

"""

Do not return anything, modify rooms in-place instead.

"""

from collections import deque as queue

M, N = len(rooms), len(rooms[0])

if M == 0 or N == 0: return

dir\_r = [-1, 0, 1, 0]

dir\_c = [0, 1, 0, -1]

q = queue()

for r in range(M):

for c in range(N):

if rooms[r][c] == 0: q.append([r,c])

while len(q) > 0:

point = q.popleft()

x, y = point[0], point[1]

for i in range(4):

rx, cy = x + dir\_r[i], y + dir\_c[i]

if rx<0 or cy<0 or rx>=M or cy>=N or rooms[rx][cy]!=2147483647: continue

rooms[rx][cy] = rooms[x][y] + 1

q.append([rx,cy])

return rooms

