LC 695

class Solution:

def maxAreaOfIsland(self, grid: List[List[int]]) -> int:

m = len(grid)

n = len(grid[0])

res = 0

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

for i in range(m):

for j in range(n):

if grid[i][j] == 1:

stack = [(i,j)]

cur = 1

grid[i][j] = 0

#seen = set((i,j))

while stack:

p,q = stack.pop()

for h,v in directions:

if 0<=p+h and p+h<m and 0<=q+v and q+v<n and grid[p+h][q+v] == 1:

stack.append((p+h,q+v))

grid[p+h][q+v] = 0

cur += 1

res = max(res,cur)

return res

LC 542

class Solution:

def updateMatrix(self, mat: List[List[int]]) -> List[List[int]]:

m = len(mat)

n = len(mat[0])

for i in range(m):

if mat[i][0] == 1:

mat[i][0] = float('inf')

for j in range(1,n):

if mat[i][j] != 0:

mat[i][j] = mat[i][j-1] + 1

for j in range(n-2,-1,-1):

if mat[i][j] != 0:

mat[i][j] = min(mat[i][j], mat[i][j+1] + 1)

for i in range(1,m):

for j in range(n):

if mat[i][j] != 0:

mat[i][j] = min(mat[i][j], mat[i-1][j] + 1)

for i in range(m-2,-1,-1):

for j in range(n):

if mat[i][j] != 0:

mat[i][j] = min(mat[i][j], mat[i+1][j] + 1)

return mat

LC 773

class Solution:

def slidingPuzzle(self, board: List[List[int]]) -> int:

target = (1,2,3,4,5,0)

tup = tuple([board[i//3][i%3] for i in range(6)])

queue = [(tup,0)]

seen = set()

seen.add(tup)

while queue:

q = queue.pop(0)

b = q[0]

res = q[1]

if b == target:

return res

for i in range(2):

for j in range(3):

if b[i\*3+j] == 0:

temp = list(b)

temp[j], temp[3+j] = temp[3+j], temp[j]

tup = tuple(temp)

if tup not in seen:

seen.add(tup)

queue.append((tup,res+1))

if j>0:

temp = list(b)

temp[i\*3+j], temp[i\*3+3-j] = temp[i\*3+3-j], temp[i\*3+j]

tup = tuple(temp)

if tup not in seen:

seen.add(tup)

queue.append((tup,res+1))

if j<2:

temp = list(b)

temp[i\*3+j], temp[i\*3+1-j] = temp[i\*3+1-j], temp[i\*3+j]

tup = tuple(temp)

if tup not in seen:

seen.add(tup)

queue.append((tup,res+1))

return -1

LC 499

class Solution:

def findShortestWay(self, maze: List[List[int]], ball: List[int], hole: List[int]) -> str:

m = len(maze)

n = len(maze[0])

visited = dict()

visited[(ball[0],ball[1])] = (0,"")

h = [(0,"",ball[0],ball[1])]

heapq.heapify(h)

directions = ((1,0,"d"),(0,-1,"l"),(0,1,"r"),(-1,0,"u"))

while h and ((hole[0],hole[1]) not in visited or visited[(hole[0],hole[1])][0] >= h[0][0]):

cur = heapq.heappop(h)

#print(cur)

for hor,ver,letter in directions:

a,b = cur[2],cur[3]

while 0 <= a+hor and a+hor < m \

and 0 <= b+ver and b+ver < n \

and maze[a+hor][b+ver] == 0:

a += hor

b += ver

if [a,b] == hole:

step = cur[0]+abs(a-cur[2])+abs(b-cur[3])

if (a,b) not in visited or visited[(a,b)][0] > step or (visited[(a,b)][0] == step and cur[1]+letter < visited[(a,b)][1]):

visited[(a,b)] = (step, cur[1]+letter)

break

step = cur[0]+abs(a-cur[2])+abs(b-cur[3])

if (a,b) not in visited or (visited[(a,b)][0] > step or (visited[(a,b)][0] == step and cur[1]+letter < visited[(a,b)][1])):

visited[(a,b)] = (step,cur[1]+letter)

if a != cur[2] or b != cur[3]:

heapq.heappush(h,(step,cur[1]+letter,a,b))

return "impossible" if (hole[0],hole[1]) not in visited else visited[(hole[0],hole[1])][1]

LC 230

class Solution:

def kthSmallest(self, root: TreeNode, k: int) -> int:

p = root

count = 0

stack = []

while p or stack:

while p:

stack.append(p)

p = p.left

cur = stack.pop()

count += 1

if count == k:

return cur.val

p = cur.right

LC 305

My solution:

class Solution:

def numIslands2(self, m: int, n: int, positions: List[List[int]]) -> List[int]:

res = [1]

seen = set()

d = dict()

seen.add((positions[0][0],positions[0][1]))

d[0] = set()

d[0].add((positions[0][0],positions[0][1]))

directions = ((-1,0),(0,-1),(1,0),(0,1))

keys = [0]

for p in positions[1:]:

a = p[0]

b = p[1]

if (a,b) in seen:

res.append(res[-1])

continue

seen.add((a,b))

lst = set()

for hor,ver in directions:

if (a+hor,b+ver) in seen:

for k in d:

if (a+hor,b+ver) in d[k]:

lst.add(k)

d[k].add((a,b))

if not lst:

d[keys[-1]+1] = set()

d[keys[-1]+1].add((a,b))

keys.append(keys[-1]+1)

if len(lst) > 1:

lst = list(lst)

lst.sort()

for l in lst[1:]:

d[lst[0]].update(d[l])

for l in lst[1:]:

del d[l]

#d.pop(l,None)

#if l in keys:

keys.remove(l)

res.append(len(d))

return res

Referenced solution:

class Solution:

def numIslands2(self, m: int, n: int, positions: List[List[int]]) -> List[int]:

ans = []

islands = Union()

for p in map(tuple, positions):

if p in islands.id:

ans += [islands.count]

else:

islands.add(p)

for dp in (0, 1), (0, -1), (1, 0), (-1, 0):

q = (p[0] + dp[0], p[1] + dp[1])

if q in islands.id:

islands.unite(p, q)

ans += [islands.count]

return ans

class Union(object):

def \_\_init\_\_(self):

self.id = {}

self.sz = {}

self.count = 0

def add(self, p):

self.id[p] = p

self.sz[p] = 1

self.count += 1

def root(self, i):

while i != self.id[i]:

self.id[i] = self.id[self.id[i]]

i = self.id[i]

return i

def unite(self, p, q):

i, j = self.root(p), self.root(q)

if i == j:

return

if self.sz[i] > self.sz[j]:

i, j = j, i

self.id[i] = j

self.sz[j] += self.sz[i]

self.count -= 1