DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

DOMAIN WINTER WINNING CAMP2024

StudentName: Divya Sharma **UID:**22BCS14192
Branch:CSE **Section/Group:**22BCS_FL_IOT-603/B
Semester:5th

VERYEASY

Flood Fill-link

You are given an image represented by anmxngrido fintegers image,where image[i][j] represents the pixel value of the image.You are also given three integers sr,sc, and color.Your task is to perform a flood fill on the image starting from the pixel image[sr][sc].

To perform a flood fill:

Begin with the starting pixel and change its color to color.
Perform the same process for each pixel that is directly adjacent (pixels that share aside with the original pixel, either horizontally or vertically) and shares the same color as the starting pixel.
Keep repeating this process by checking neighboring pixels of the updated pixels and modifying their color if it matches the original color of the starting pixel.
The process stops when there are no more adjacent pixels of the original color to update.
Return the modified image after performing the flood fill.

Example1:

Input:image=[[1,1,1],[1,1,0],[1,0,1]],sr=1,sc=1, color=2
Output:[[2,2,2],[2,2,0],[2,0,1]]

Explanation:
From the center of the image with position(sr,sc)=(1,1)(i.e.,theredpixel),all pixels connected by a path of the same color as the starting pixel (i.e., the blue pixels) are colored with the new color.

Note the bottom corner is not colored 2, because it is not horizontally or vertically connected to the starting pixel.

Example2:

Input:image=[[0,0,0],[0,0,0]], sr=0, sc=0, color =0

Output:[[0,0,0],[0,0,0]]

Explanation:
The starting pixel is already colored with0,which is the same as the target color.Therefore,no changes are made to the image.

Constraints:

- m == image.length
- n==image[i].length
- 1<=m, n <=50
- 0<=image[i][j],color <2^16
- 0<=sr <m
- 0<=sc<n

CODE:

```
deffloodFill(image,sr,sc,color):
    rows,cols=len(image),len(image[0])
    original_color = image[sr][sc]
    iforiginal_color==color:
        return image

    defdfs(r,c):
        ifr<0orr>=rows orc<0orc>=colsorimage[r][c]!=original_color: return
        image[r][c]=color
        dfs(r + 1, c)
        dfs(r - 1, c)
        dfs(r,c+1)
        dfs(r, c - 1)

    dfs(sr, sc)
    returnimage

image1=[[1, 1,1], [1,1,0], [1,0, 1]]
sr1, sc1, color1 = 1, 1, 2
print(floodFill(image1,sr1,sc1,color1))
```

```
Output

[[2, 2, 2], [2, 2, 0], [2, 0, 1]]

=== Code Execution Successful ===
```

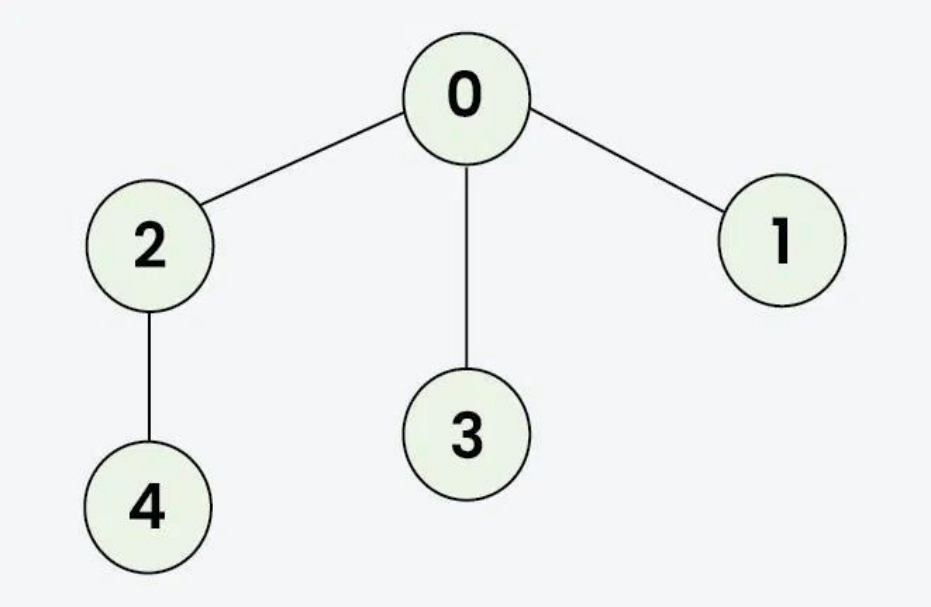
EASY

DFSofGraph

Givenaconnectedundirectedgraphrepresentedbyanadjacencylistadj,whichisavectorofvectors whereeachadj[i]representsthelistofverticesconnectedtovertexi.PerformaDepthFirstTraversal (DFS)startingfromvertex0,visitingverticesfromlefttorightaspertheadjacencylist,andreturna list containing the DFS traversal of the graph.

Note:Dotraverseinthe sameorderastheyareintheadjacency list.

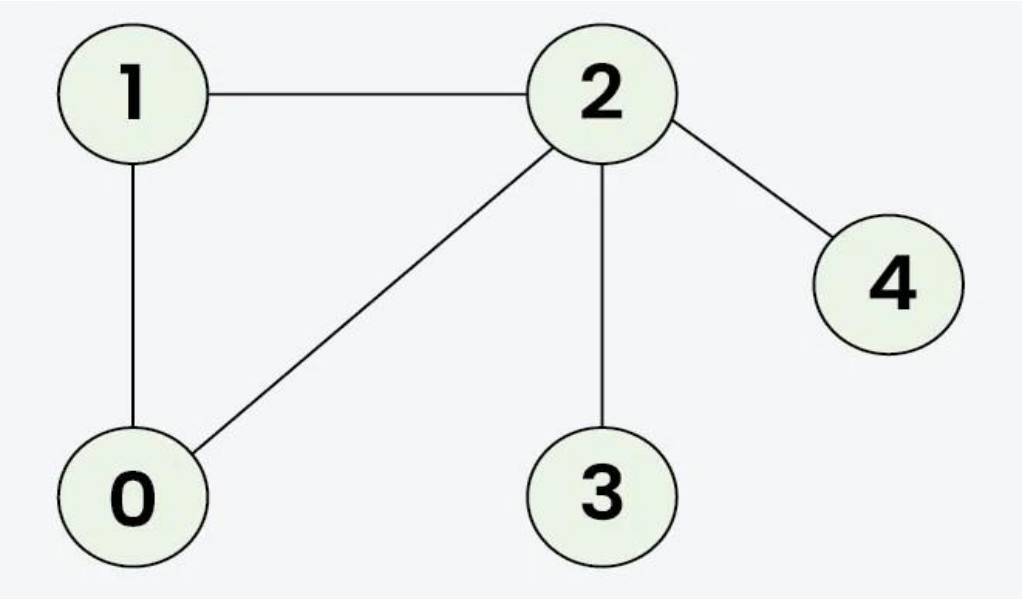
Example1:



Input:adj=[[2,3,1], [0], [0,4],[0], [2]]
Output:[0, 2, 4, 3, 1]
Explanation:Startingfrom0,theDFStraversalproceedsasfollows: Visit 0 → Output: 0
Visit2 (thefirst neighborof 0)→ Output: 0, 2
Visit4 (thefirst neighborof 2)→ Output: 0, 2, 4
Backtrackto 2,then backtrackto 0,and visit3 →Output: 0,2, 4, 3

Finally,backtrack to 0and visit 1 →Final Output:0,2, 4, 3, 1

Example2:



Input:adj= [[1,2], [0,2],[0,1,3,4],[2],[2]]
Output:[0, 1, 2, 3, 4]
Explanation:Startingfrom0,theDFStraversalproceedsasfollows: Visit 0
→ Output: 0
Visit1 (thefirst neighborof 0)→ Output: 0, 1
Visit2 (thefirst neighborof 1)→ Output: 0, 1, 2
Visit3 (thefirst neighborof 2)→ Output: 0, 1, 2, 3
Backtrackto2andvisit 4→FinalOutput: 0,1,2,3, 4

Constraints:

- 1≤ adj.size()≤ 1e4
- 1≤adj[i][j]≤ 1e4

CODE:

```
defdfs_traversal(adj):
    defdfs(node,visited,result):

        visited[node]=True
        result.append(node)

        for neighbor in adj[node]:
            if not visited[neighbor]:
                dfs(neighbor,visited,result)

    visited=[False]*len(adj) result
    = []
```

```
    dfs(0,visited,result)

    return result
adj= [[2,3,1],[0], [0,4], [0], [2]]
output=dfs_traversal(adj)
print(output)
```

```
Output
[0, 2, 4, 3, 1]

=== Code Execution Successful ===
```

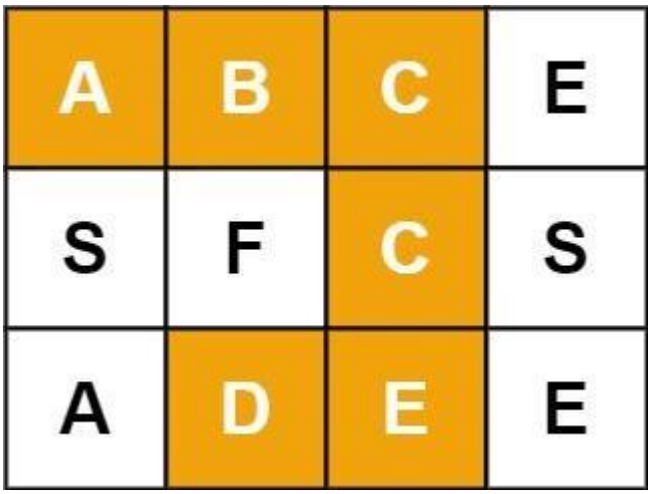
Medium

WordSearch

Givenan mx ngrid ofcharacters boardand astringword, returntrueifword existsin thegrid.

Thewordcanbeconstructedfromlettersofsequentiallyadjacentcells,whereadjacentcellsare horizontally or vertically neighboring. The same letter cell may not be used more than once.

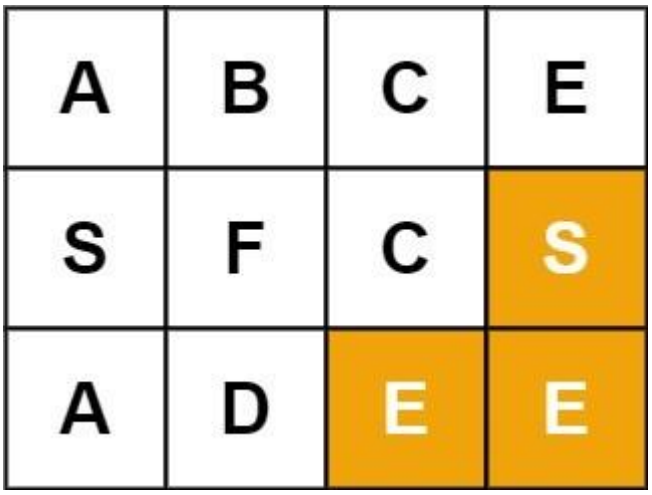
Example1:



Input:board=[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],word="ABCCED"

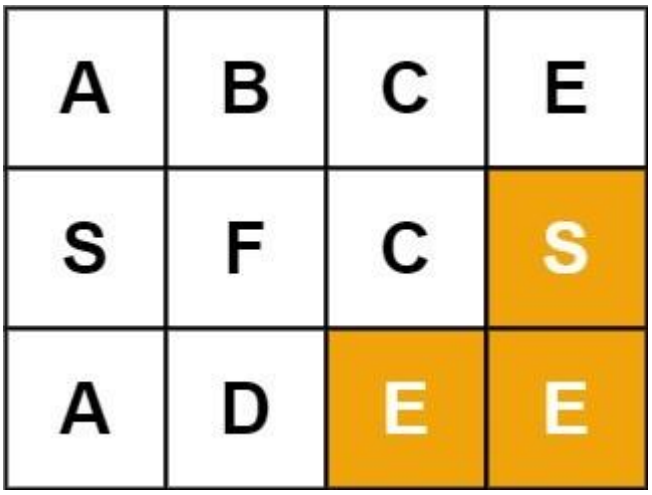
Output:true

Example2:



Input:board=[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],word="SEE"
Output:true

Example3:



Input:board=[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],word="ABCB"
Output:false

Constraints:

- m == board.length
- n =board[i].length
- 1<=m, n <=6
- 1<=word.length<=15
- boardandwordconsistsofonlylowercaseanduppercaseEnglish letters.

Followup:Could youusesearchpruningto makeyoursolutionfasterwith alarger board?

CODE:

```
defexist(board,word):
    rows,cols=len(board),len(board[0])

    defdfs(r,c, index):
        ifindex==len(word):
            return True
        ifr<0orr>=rowsorc<0orc>=colsorboard[r][c]!=word[index]: return False
        temp,board[r][c]=board[r][c],'#'

        found=(dfs(r+1,c,index+1)or dfs(r
                - 1, c, index + 1) or dfs(r, c +
                1, index + 1) or dfs(r, c - 1,
                index + 1))
        board[r][c]=temp
        return found
    for row in range(rows):
        forcolinrange(cols):
            ifboard[row][col]==word[0]anddfs(row,col,0): return
                True

    returnFalse

board1=[["A","B","C","E"],
        ["S","F","C","S"],
        ["A","D","E","E"]]
word1 = "ABCCED"
print(exist(board1,word1))#Output:True
```

```
Output

True

=== Code Execution Successful ===
```

Hard

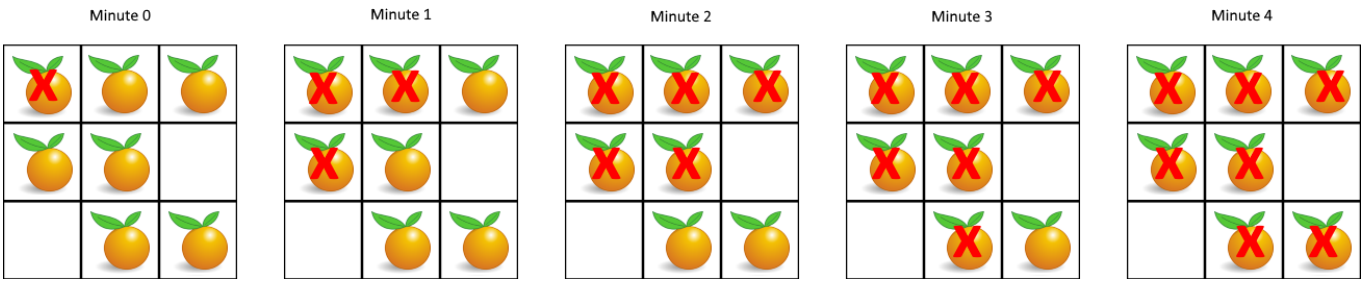
RottingOranges

Youaregiven an m xn grid whereeach cellcan haveoneof threevalues:

0 representinganemptycell,
1 representingafreshorange,or
2 representingarottenorange.
Everyminute,anyfreshorangethatis 4-directionallyadjacenttoa rottenorangebecomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this isimpossible, return -1.

Example1:



Input: grid =[[2,1,1],[1,1,0],[0,1,1]]
Output:4

Example2:

Input: grid =[[2,1,1],[0,1,1],[1,0,1]]
Output: -1
Explanation:Theorangeinthebottomleftcorner(row2,column0)isneverrotten,becauserotting only happens 4-directionally.

Example3:

Input: grid =[[0,2]]
Output:0

Explanation:Sincethereare already nofreshoranges atminute0,theanswer isjust 0.

Constraints:

- m == grid.length
- n==grid[i].length
- 1<=m, n <=10
- grid[i][j]is 0,1, or 2.

CODE:

```
fromcollectionsimportdeque

deforangesRotting(grid):
    rows,cols=len(grid),len(grid[0])
    queue = deque()
    fresh_count=0
    for r in range(rows):
        forcinrange(cols):
            ifgrid[r][c]==2:
                queue.append((r,c))
            elif grid[r][c] == 1:
                fresh_count+=1 if
    fresh_count == 0:
        return0
    minutes_passed=0
    directions=[(1,0),(-1,0),(0,1),(0, -1)]
    whilequeue:
        for_inrange(len(queue)):
            x, y = queue.popleft()
            for dx, dyin directions:
                nx,ny= x+dx, y+ dy
                if0<=nx<rowsand0<=ny<colsandgrid[nx][ny]==1: grid[nx][ny] = 2
                    queue.append((nx,ny))
                    fresh_count -= 1
        minutes_passed+=1
    returnminutes_passed-1iffresh_count==0else -1

grid1= [[2, 1,1],[1,1,0],[0, 1, 1]]
print(orangesRotting(grid1))#Output:4
```

```
Output

4

=== Code Execution Successful ===
```

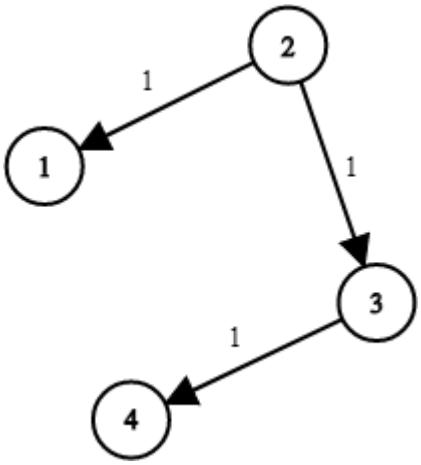
Very Hard

NetworkDelay Time

You are given a network of n nodes, labeled from 1 to n. You are also given times, a list of travel times as directed edges times[i] = (ui, vi, wi), where ui is the source node, vi is the target node, and wi is the time it takes for a signal to travel from source to target.

We will send a signal from a given node k. Return the minimum time it takes for all the n nodes to receive the signal. If it is impossible for all the n nodes to receive the signal, return -1.

Example1:



Input:times=[[2,1,1],[2,3,1],[3,4,1]],n =4,k=2
Output:2

Example2:

Input:times=[[1,2,1]], n=2,k=1
Output:1

Example3:

Input:times=[[1,2,1]], n=2,k=2
Output: -1

Constraints:

- 1<=k <=n <=100
- 1<=times.length<=6000
- times[i].length==3
- 1<=ui,vi <=n
- ui!=vi
- 0 <=wi <=100
- Allthepairs (ui,vi) areunique.(i.e., no multipleedges.)

CODE:

```
importheapq
from collections import defaultdict
defnetworkDelayTime(times,n,k):
    graph=defaultdict(list)
    for u, v, w in times:
        graph[u].append((v,w))
    min_heap = [(0, k)]
    shortest_times = {}
    whilemin_heap:
        current_time,node=heapq.heappop(min_heap) if
        node in shortest_times:
            continue
          shortest_times[node]=current_time
            forneighbor,weightingraph[node]:
              ifneighbornotin shortest_times:
                heapq.heappush(min_heap,(current_time+weight,neighbor))
    returnmax(shortest_times.values())iflen(shortest_times)==nelse-1
times1=[[2,1,1],[2,3,1],[3,4, 1]]
n1,k1=4, 2
print(networkDelayTime(times1,n1,k1))#Output:2
```

Output

```
2

=== Code Execution Successful ===
```