MICROSOFT PREPARATION

DAY : 02

Find Whether path exist :

Problem Link :
https://practice.geeksforgeeks.org/problems/find-whether-path-exist5238/1?page=1&company%5B%5D=Microsoft&category%5B%5D=Graph&sortBy=difficulty

Test Cases Passed : 10 / 10

Time Used : 06.30

Difficulty Level : MEDIUM

Approach Used :

is_Possible( )
- Declare visited vector of same size of grid initially assigned 0 to all elements
- Traverse through all components :
  - Check if the component is an unvisited source :
    - Make a dfs call to traverse the grid as dfs(row,col,visited,grid) and return it
- Outside loop returns false as in this case the traversal cannot reach the destination.

dfs( )
- Mark the given node as visited
- Traverse through all of the four directions where the transition is possible
  - Check if a node is unvisited and the node is a destination :
    - Return true
  - Check if its an unvisited empty space and not a wall :
    - Return the dfs call as dfs(adjRow,adjCol,visited,grid)
- Outside loop Return false // because in this case the dfs never reached the destination cell

Solution :

```
bool dfs(int row,int
col,vector<vector<int>>&visited,vector<vector<int>>&grid)
    {
        // calculating the dimensions of grid
        int n = grid.size();
        int m = grid[0].size();
        // marking the node as visited
        visited[row][col]=1;
        // traversing through the adjacent nodes and columns
        int delRow[] = {-1,0,1,0};
        int delCol[] = {0,1,0,-1};

        for(int i=0;i<4;i++)
        {
            // calculating the coordinates of neighboring rows and cols
            int nrow = row+delRow[i];
            int ncol = col+delCol[i];
            // checking if the coordinates are valid or not
            if(nrow<n && nrow>=0 && ncol<m && ncol>=0)
            {
                 // checking if node is unvisited and is a destination
                 if(!visited[nrow][ncol] && grid[nrow][ncol]==2)
                 {  // returning true if we reach destination
                    return true;
                 }
             // checking if node is unvisited and is not a wall
            if(!visited[nrow][ncol] && grid[nrow][ncol]!=0)
            {
                // checking if this node can lead to the destination
                if(dfs(nrow,ncol,visited,grid))
                {
                 // return true if grid can lead to destination
                    return true;
                }
            }
            }
        }
        // return false if the dfs cannot reach the destination
        return false;
    }
    bool is_Possible(vector<vector<int>>& grid)
    {
        // calculating the dimensions of grid
        int n = grid.size();
        int m = grid[0].size();
        // creating a visited vector of size n*m initially 0
        vector<vector<int>> visited(n,vector<int>(m,0));
        // traversing through all components
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
        // checking if the node is unvisited and grid element is source
        if(!visited[i][j] && grid[i][j]==1 )
                {
                 // returning if the dfs can reach to destination
                 return dfs(i,j,visited,grid);
                }
            }
        }

        // returning false because the dfs cannot reach the destination
        return false;
    }
```