MEDIUM

Distance of the nearest cell having 1 | 0/1 matrix

Intuition

Given is grid :

$$\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{array}$$

1 - 0 distance as its nearest to itself
0 - nearest 1 is at [row difference + column difference]

Nearest 1 is

$$\begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 2 \text{ [row difference + column difference]} \end{array}$$

Eg.

$$\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{array}$$

Outputs :

$$\begin{array}{ccc} 2 & 1 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{array}$$

Approach to solve :

BFS will be applied here because we have to go level wise traversal because first we want to cover the adjacent sides and then perform the minimum updation possible

$$\begin{array}{cccc} & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{array}$$

2 1   0   1

Consider the above to be the given matrix

First we will create a visited array having all initially 0 and a result matrix

Original Array [**1** are the initial sources]

Queue = [[[1,1],0],[[2,1],0],[[2,2],0]] //initial configuration

[[2,1],0],[[2,2],0],[[0,1],1],.....] continue for all four directions1

// While inserting [1,0],2 will be inserted

0	0	0
0	**1**	0
1	0	**1**

Visited

0	0	0
0	1	0
1	0	1

Result

2	1	2
1	0	1
0	1	0

We will perform the multisource BFS here on the points that are already 1

Algorithm :

- Declare :
    - Empty result vector of size n*m

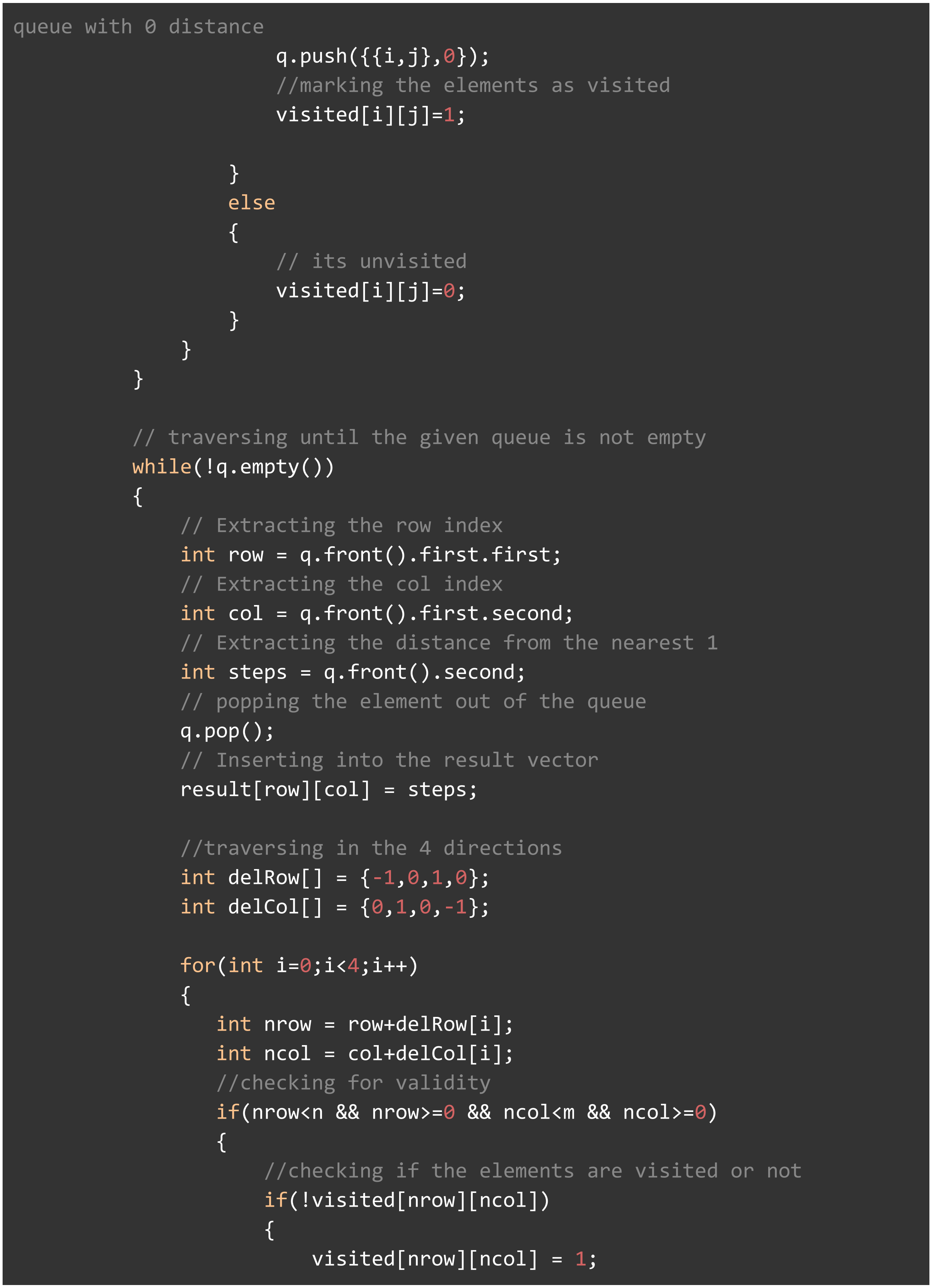
    - Visited array having all elements initially 0 of size n*m
    - Empty queue containing <<indexRow, indexCol>,distance>
- Traverse the array and insert the row, col and distance=0 in queue at all places where the grid contains 1
- Traverse until the queue becomes empty :
    - Compute the values of :
        - Row
        - Column
        - Distance from original node
        - Update result as result[row][col] = steps
        - Pop the node from the queue
        - Traverse the adjacent nodes [up, down, left, right] :
            - If not visited then : visit it and add to queue with distance = steps+1
- Return result

Function Code :

```
//Function to find distance of nearest 1 in the grid for each cell.
      vector<vector<int>>nearest(vector<vector<int>>grid)
      {
          // creating variables containing the dimensions of the grid
          int n = grid.size();
          int m = grid[0].size();
          //Declaring a visited array having 0 of size n*m
          //Creating a resultant array having dimensions of n*m
         vector<vector<int>> visited(n, vector<int>(m, 0));
          vector<vector<int>> result(n,vector<int> (m,0));

          //Declaring a queue that will contain the coordinates of the
point and distance
          queue<pair<pair<int,int>,int>> q;


          // traversing to check where one is present
          for(int i=0;i<n;i++)
          {
              for(int j=0;j<m;j++)
              {
                  if(grid[i][j]==1)
                  {
                      //inserting all the elements that have 1 into the
```



```
queue with 0 distance
                      q.push({{i,j},0});
                      //marking the elements as visited
                      visited[i][j]=1;

                  }
                  else
                  {
                      // its unvisited
                      visited[i][j]=0;
                  }
              }
          }

          // traversing until the given queue is not empty
          while(!q.empty())
          {
              // Extracting the row index
              int row = q.front().first.first;
              // Extracting the col index
              int col = q.front().first.second;
              // Extracting the distance from the nearest 1
              int steps = q.front().second;
              // popping the element out of the queue
              q.pop();
              // Inserting into the result vector
              result[row][col] = steps;

              //traversing in the 4 directions
              int delRow[] = {-1,0,1,0};
              int delCol[] = {0,1,0,-1};

              for(int i=0;i<4;i++)
              {
                 int nrow = row+delRow[i];
                 int ncol = col+delCol[i];
                 //checking for validity
                 if(nrow<n && nrow>=0 && ncol<m && ncol>=0)
                 {
                     //checking if the elements are visited or not
                     if(!visited[nrow][ncol])
                     {
                         visited[nrow][ncol] = 1;
```

```
                       q.push({{nrow,ncol},steps+1});
                   }
               }


            }

        }
        return result;
```

Time Complexity

O(n*m)