Shortest path in a Binary Maze

Given a MxN matrix where each element can either be 0 or 1. We need to find the shortest path between a given source cell to a destination cell. The path can only be created out of a cell if its value is 1.

Expected time complexity is O(MN).

For example –

```
Input:
mat[ROW][COL]  = {{1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
                  {1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },
                  {1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
                  {0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
                  {1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
                  {1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
                  {1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
                  {1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
                  {1, 1, 0, 0, 0, 0, 1, 0, 0, 1 }};
Source = {0, 0};
Destination = {3, 4};

Output:
Shortest Path is 11
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is inspired from Lee algorithm and uses BFS.

1. We start from the source cell and calls BFS procedure.
2. We maintain a queue to store the coordinates of the matrix and initialize it with the source cell.
3. We also maintain a Boolean array visited of same size as our input matrix and initialize all its elements to false.
    1. We LOOP till queue is not empty
    2. Dequeue front cell from the queue
    3. Return if the destination coordinates have reached.
    4. For each of its four adjacent cells, if the value is 1 and they are not visited yet, we enqueue it in the queue and also mark them as visited.

Below is C++ implementation of the idea –

```
// C++ program to find the shortest path between
// a given source cell to a destination cell.
#include <bits/stdc++.h>
using namespace std;
#define ROW 9
#define COL 10

//to store matrix cell cordinates
struct Point
{
    int x;
    int y;
};

// An Data Structure for queue used in BFS
struct queueNode
{
    Point pt;  // The cordinates of a cell
    int dist;  // cell's distance of from the source
};

// check whether given cell (row, col) is a valid
// cell or not.
```

```
bool isValid(int row, int col)
{
    // return true if row number and column number
    // is in range
    return (row >= 0) && (row < ROW) &&
           (col >= 0) && (col < COL);
}

// These arrays are used to get row and column
// numbers of 4 neighbours of a given cell
int rowNum[] = {-1, 0, 0, 1};
int colNum[] = {0, -1, 1, 0};

// function to find the shortest path between
// a given source cell to a destination cell.
int BFS(int mat[][COL], Point src, Point dest)
{
    // check source and destination cell
    // of the matrix have value 1
    if (!mat[src.x][src.y] || !mat[dest.x][dest.y])
        return INT_MAX;

    bool visited[ROW][COL];
    memset(visited, false, sizeof visited);

    // Mark the source cell as visited
    visited[src.x][src.y] = true;

    // Create a queue for BFS
    queue<queueNode> q;

    // distance of source cell is 0
    queueNode s = {src, 0};
    q.push(s);  // Enqueue source cell

    // Do a BFS starting from source cell
    while (!q.empty())
    {
        queueNode curr = q.front();
        Point pt = curr.pt;

        // If we have reached the destination cell,
        // we are done
        if (pt.x == dest.x && pt.y == dest.y)
            return curr.dist;

        // Otherwise dequeue the front cell in the queue
        // and enqueue its adjacent cells
        q.pop();

        for (int i = 0; i < 4; i++)
        {
            int row = pt.x + rowNum[i];
            int col = pt.y + colNum[i];

            // if adjacent cell is valid, has path and
            // not visited yet, enqueue it.
            if (isValid(row, col) && mat[row][col] &&
               !visited[row][col])
            {
                // mark cell as visited and enqueue it
                visited[row][col] = true;
                queueNode Adjcell = { {row, col},
                                      curr.dist + 1 };
                q.push(Adjcell);
            }
        }
    }

    //return -1 if destination cannot be reached
    return INT_MAX;
}
```

```
// Driver program to test above function
int main()
{
    int mat[ROW][COL] =
    {
        { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
        { 1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },
        { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
        { 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
        { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
        { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
        { 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
        { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
        { 1, 1, 0, 0, 0, 0, 1, 0, 0, 1 }
    };

    Point source = {0, 0};
    Point dest = {3, 4};

    int dist = BFS(mat, source, dest);

    if (dist != INT_MAX)
        cout << "Shortest Path is " << dist ;
    else
        cout << "Shortest Path doesn't exist";

    return 0;
}
```

Output :

```
Shortest Path is 11
```