1091. Shortest Path in Binary Matrix

Medium 4.6K 181

https://leetcode.com/problems/shortest-path-in-binary-matrix/description/

994. Rotting Oranges

Medium 10.1K 340

https://leetcode.com/problems/rotting-oranges/

1091. Shortest Path in Binary Matrix Hint

Medium 4.6K 181

Companies

Given an `n x n` binary matrix `grid`, return *the length of the shortest* ***clear path*** *in the matrix.* If there is no clear path, return `-1`.

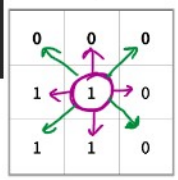
A **clear path** in a binary matrix is a path from the **top-left** cell (i.e., `(0, 0)`) to the **bottom-right** cell (i.e., `(n - 1, n - 1)`) such that:

- All the visited cells of the path are `0`.
- All the adjacent cells of the path are **8-directionally** connected (i.e., they are different and they share an edge or a corner).

The **length of a clear path** is the number of visited cells of this path.

→ Explain

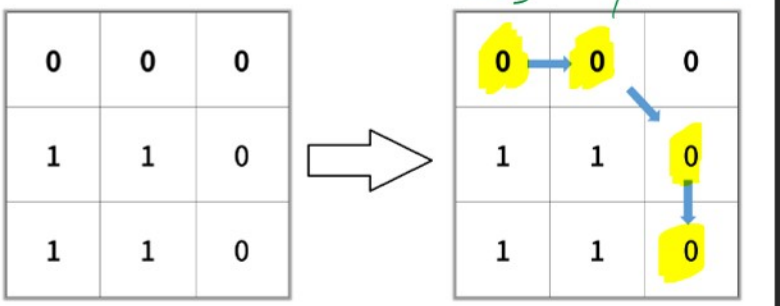
→ what Algo — BFS ↓ shortest path... — DFS — DP ↓ why not DP?



4 cells visited clear path is 4.

Example 2:

only 0's can be visited !!

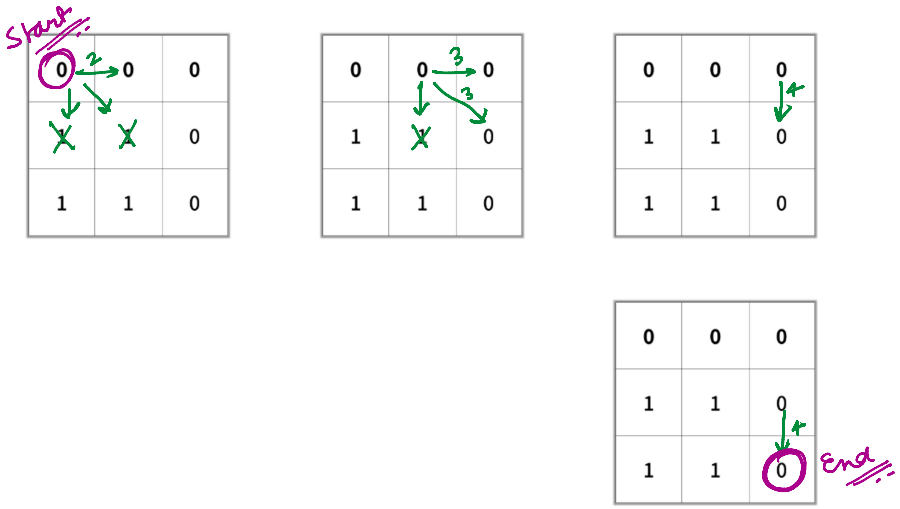


Input: grid = [[0,0,0],[1,1,0],[1,1,0]]
Output: 4

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

Example 3:

Input: grid = [[1,0,0],[1,1,0],[1,1,0]]
Output: -1



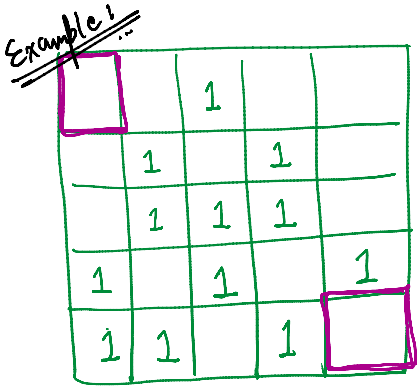
→ At every unit of time, I am trying to SPREAD & reach to final cell [n-1][n-1]

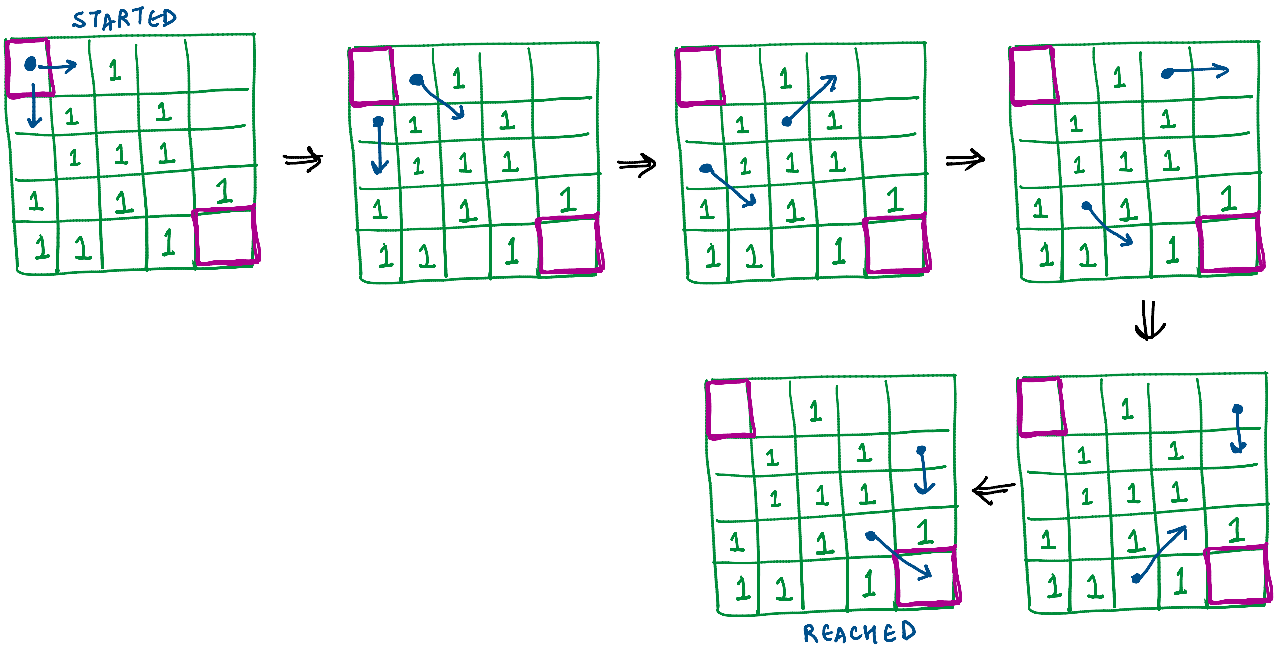
Why ??

spreading is nothing but going on breadth wise

↓

thus, we can use BFS.





" BUT BUT BUT "

Let's confuse you!!

You must have seen a same kind of Problem in DP also !!

Min Cost Path | DP-6

Difficulty Level : Easy • Last Updated : 17 Feb, 2023

Read Discuss(20+) Courses Practice Video

Given a cost matrix cost[][] and a position (M, N) in cost[][], write a function that returns cost of minimum cost path to reach (M, N) from (0, 0). Each cell of the matrix represents a cost to traverse through that cell. The total cost of a path to reach (M, N) is the sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j), cells (i+1, j), (i, j+1), and (i+1, j+1) can be traversed.

Only 1 directional i.e. Downwards-Right traversal.

Note: You may assume that all costs are positive integers.

Example:

thus DP will fail at these kind of Cases :↓

Example:

Input:

1	2	3

```
[0,1,0,0,0]
[0,1,0,1,0]
[0,1,0,1,0]
```

1	2	3
4	8	2
1	5	3

The path with minimum cost is highlighted in the following figure. The path is (> (2, 2). The cost of the path is 8 (1 + 2 + 2 + 3).

Output:

1	2	3
4	8	2
1	5	3

```
[0,1,0,1,0]
[0,1,0,1,0]
[0,1,0,1,0]
[0,0,0,1,0]
```

Acc to BFS

```
[0,1,0,0,0]
[0,1,0,1,0]
[0,1,0,1,0]
[0,1,0,1,0]
[0,0,0,1,0]
```

Ans = 16

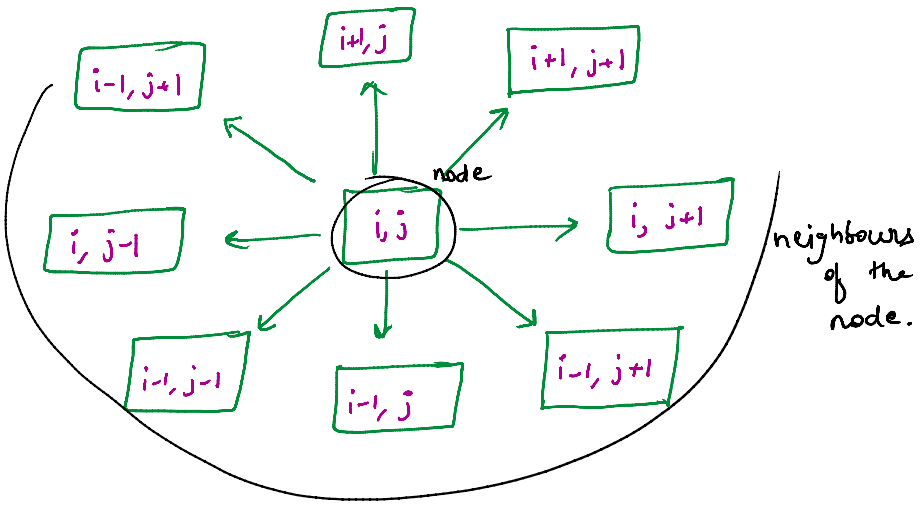
According to DP

```
[0,1,0,0,0]
[0,1,0,1,0]
[0,1,0,1,0]
[0,1,0,1,0]
[0,0,0,1,0]
```

Ans = -1

Thus, As we can traverse 8-directions, we can't use a DP, we have to use a BFS.

⇒ Now, we know, we traverse Breadth wise thus we are using BFS. But what is the next level, I have to go on.



```
int dir[][] = new int[][]{{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
```

↳ To go from node to neighbours we use this dir array

dirX
dirY
↓
2 1-D arrays of X's & Y's

dir
↓
1 2-D array

curr[0] = X
curr[1] = Y
} → Node

```
for (int k=0;k<8;k++) {
    int nextX = dir[k][0]+curr[0];
    int nextY = dir[k][1]+curr[1];
```

↳ Neighbours ..

```
class Solution {
public:
    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        if(grid[0][0]==1 || grid[m-1][n-1]==1) return -1;

        int dir[][2] = {{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
        vector<vector<bool>> visited(m, vector<bool>(n, false));
        visited[0][0] = true;
        queue<vector<int>> q;
        q.push({0,0});

        int ans=0;
        while (!q.empty()) {
            int size = q.size();
            for(int i=0;i<size;i++) {
                auto curr = q.front();
                q.pop();
                if(curr[0]==m-1 && curr[1]==n-1) {
                    return ans+1;
                }
                for (int k=0;k<8;k++) {
                    int nextX = dir[k][0]+curr[0];
                    int nextY = dir[k][1]+curr[1];

                    if(nextX>=0 && nextX<m && nextY>=0 && nextY<n && !visited[nextX][nextY] && grid[nextX][nextY]==0) {
                        q.push({nextX,nextY});
                        visited[nextX][nextY]=true;
                    }
                }
            }
            ans++;
        }
        return -1;
    }
};
```

Base case

dir array to get to the neighbours.

visited to actually visit a cell only once, as if it is visited then it would be the first & shortest path.

standard BFS

Analyze all the nodes at one level, as answer will increase by 1 for these nodes only

currently filled

neighbours coming

when reached end

going to neighbours

Check if that is not visited & a valid location

In bounds of matrix

Should be a value '0'.

Standard BFS

If still it can't reach

If still it can't reach cell [n-1][n-1] then return (-1).

time : $O(N^2)$ → BFS in worst case has to visit all the cells in grid.

space : $O(N^2)$

↳ visited Array / Queue

MODIFICATIONS TO MAKE CODE EFFICIENT :-

```
class Solution {
public:
    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        if(grid[0][0]==1 || grid[m-1][n-1]==1) return -1;

        int dir[][2] = {{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
        vector<vector<bool>> visited(m, vector<bool>(n, false));
        visited[0][0] = true;
        queue<vector<int>> q;
        q.push({0,0});

        int ans=0;
        while (!q.empty()) {
            int size = q.size();
            for(int i=0;i<size;i++) {
                auto curr = q.front();
                q.pop();
                if(curr[0]==m-1 && curr[1]==n-1) {
                    return ans+1;
                }
                for (int k=0;k<8;k++) {
                    int nextX = dir[k][0]+curr[0];
                    int nextY = dir[k][1]+curr[1];

                    if(nextX>=0 && nextX<m && nextY>=0 && nextY<n && !visited[nextX][nextY] && grid[nextX][nextY]==0) {
                        q.push({nextX,nextY});
                        visited[nextX][nextY]=true;
                    }
                }
            }
            ans++;
```

visited array not required, we can modify (grid) matrix itself.

↳ Not Recommended in Production Codes.

queue < pair<int, int>> q

↳ Pair is much faster than a vector

```
                }
            }
        }
        ans++;
    }
    return -1;
  }
};
```

C++ Auto

```cpp
class Solution {
public:
    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        if(grid[0][0]==1 || grid[m-1][n-1]==1) return -1;

        int dir[][2] = {{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
        vector<vector<bool>> visited(m, vector<bool>(n, false));
        visited[0][0] = true;
        queue<vector<int>> q;
        q.push({0,0});

        int ans=0;
        while (!q.empty()) {
            int size = q.size();
            for(int i=0;i<size;i++) {
                auto curr = q.front();
                q.pop();
                if(curr[0]==m-1 && curr[1]==n-1) {
                    return ans+1;
                }
                for (int k=0;k<8;k++) {
                    int nextX = dir[k][0]+curr[0];
                    int nextY = dir[k][1]+curr[1];

                    if(nextX>=0 && nextX<m && nextY>=0 && nextY<n && !visited[nextX][nextY] && grid[nextX][nextY]==0) {
                        q.push({nextX,nextY});
                        visited[nextX][nextY]=true;
                    }
                }
            }
            ans++;
        }
        return -1;
    }
};
```


```cpp
class Solution {
public:
    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        if(grid[0][0]==1 || grid[m-1][n-1]==1) return -1;
        int dir[][2] = {{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
        vector<vector<bool>> visited(m, vector<bool>(n, false));
        visited[0][0] = true;
        queue<vector<int>> q;
        q.push({0,0});
        int ans=0;
        while (!q.empty()) {
            int size = q.size();
            for(int i=0;i<size;i++) {
                auto curr = q.front();
                q.pop();
                if(curr[0]==m-1 && curr[1]==n-1) {
                    return ans+1;
```

```
                }
                for (int k=0;k<8;k++) {
                    int nextX = dir[k][0]+curr[0];
                    int nextY = dir[k][1]+curr[1];
                    if(nextX>=0 && nextX<m && nextY>=0 && nextY<n && !visited[nextX][nextY] && grid[nextX][nextY]==0) {
                        q.push({nextX,nextY});
                        visited[nextX][nextY]=true;
                    }
                }
            }
            ans++;
        }
        return -1;
    }
};
```

Java | Auto

```
class Solution {
    public int shortestPathBinaryMatrix(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        if(grid[0][0]==1 || grid[m-1][n-1]==1) return -1;

        int dir[][] = new int[][]{{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
        boolean[][] visited = new boolean[m][n];
        visited[0][0] = true;
        Queue<int[]> queue = new LinkedList<>();
        queue.add(new int[]{0,0});

        int ans=0;
        while (!queue.isEmpty()) {
            int size = queue.size();
            for(int i=0;i<size;i++) {
                int[] curr = queue.remove();
                if(curr[0]==m-1 && curr[1]==n-1) {
                    return ans+1;
                }
                for (int k=0;k<8;k++) {
                    int nextX = dir[k][0]+curr[0];
                    int nextY = dir[k][1]+curr[1];

                    if(nextX>=0 && nextX<m && nextY>=0 && nextY<n && !visited[nextX][nextY] && grid[nextX][nextY]==0) {
                        queue.add(new int[]{nextX,nextY});
                        visited[nextX][nextY]=true;
                    }
                }
            }
            ans++;
        }

        return -1;
    }
}
```

```
class Solution {
    public int shortestPathBinaryMatrix(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        if(grid[0][0]==1 || grid[m-1][n-1]==1) return -1;
        int dir[][] = new int[][]{{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
        boolean[][] visited = new boolean[m][n];
        visited[0][0] = true;
        Queue<int[]> queue = new LinkedList<>();
        queue.add(new int[]{0,0});
```

```
        int ans=0;
        while (!queue.isEmpty()) {
            int size = queue.size();
            for(int i=0;i<size;i++) {
                int[] curr = queue.remove();
                if(curr[0]==m-1 && curr[1]==n-1) {
                    return ans+1;
                }
                for (int k=0;k<8;k++) {
                    int nextX = dir[k][0]+curr[0];
                    int nextY = dir[k][1]+curr[1];
                    if(nextX>=0 && nextX<m && nextY>=0 && nextY<n && !visited[nextX][nextY] && grid[nextX][nextY]==0) {
                        queue.add(new int[]{nextX,nextY});
                        visited[nextX][nextY]=true;
                    }
                }
            }
            ans++;
        }
        return -1;
    }
}
```