[Node can be visited more than Once]

Can visit a cell twice due to RED & BLUE colours.

1129. Shortest Path with Alternating Colors

Medium 3.1K 164

https://leetcode.com/problems/shortest-path-with-alternating-colors/

Can visit a cell K times due to how many obstacles removed/remaining.

1293. Shortest Path in a Grid with Obstacles Elimination

Hard 4.1K 72

https://leetcode.com/problems/shortest-path-in-a-grid-with-obstacles-elimination/

Here also a cell can be visited multiple times due to keys.

864. Shortest Path to Get All Keys

Hard 964 41

https://leetcode.com/problems/shortest-path-to-get-all-keys/

1293. Shortest Path in a Grid with Obstacles Elimination

Hint

Hard 4.1K 72

Companies

4 directional movement

You are given an `m x n` integer matrix `grid` where each cell is either `0` (empty) or `1` (obstacle). You can move up, down, left, or right from and to an empty cell in **one step**.

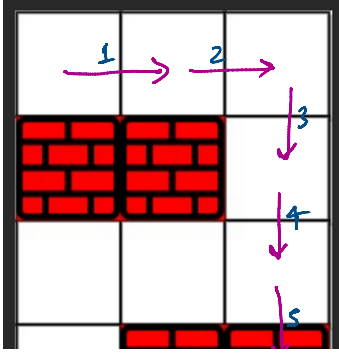
Return the *minimum number of* ***steps*** *to walk from the upper left corner* `(0, 0)` *to the lower right corner* `(m - 1, n - 1)` *given that you can eliminate* ***at most*** `k` *obstacles*. If it is not possible to find such walk return `-1`.

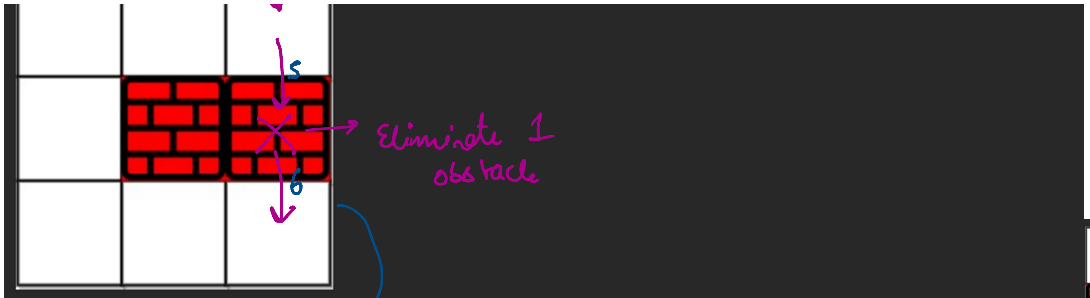
Shortest distance to reach from (0,0) to (m-1, n-1)

"special condition".

BFS

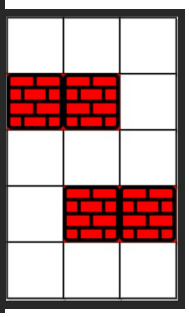
Example 1:



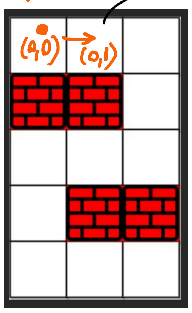


Input: grid = [[0,0,0],[1,1,0],[0,0,0],[0,1,1],[0,0,0]], k = 1
Output: 6
Explanation:
The shortest path without eliminating any obstacle is 10.
The shortest path with one obstacle elimination at position (3,2) is 6. Such path is (0,0) -> (0,1) -> (0,2) -> (1,2) -> (2,2) -> **(3,2)** -> (4,2).

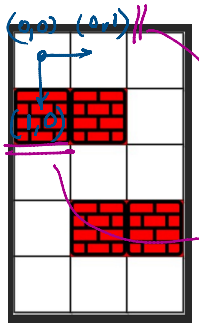
6 are the minimum number of steps needed !!



⇒ Let's try to do a simple BFS



My queue will have standard (cell) → {i, j} as the input.

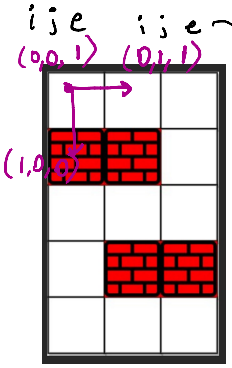


but now as, I have power to elimate obstacle thus I can go through obstacles also.

Are these both same kind of Movements ?? — No Right ?

One has used the eliminatig power

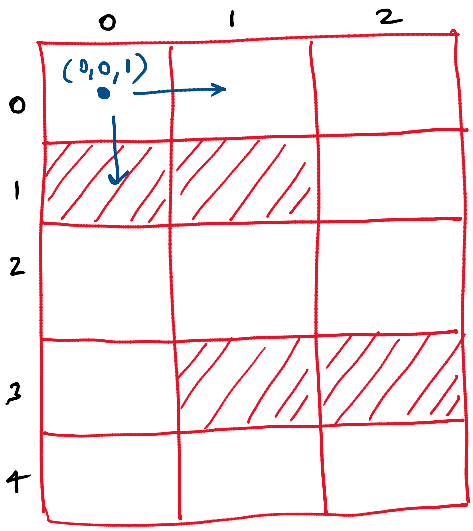
One has eliminatig power left.

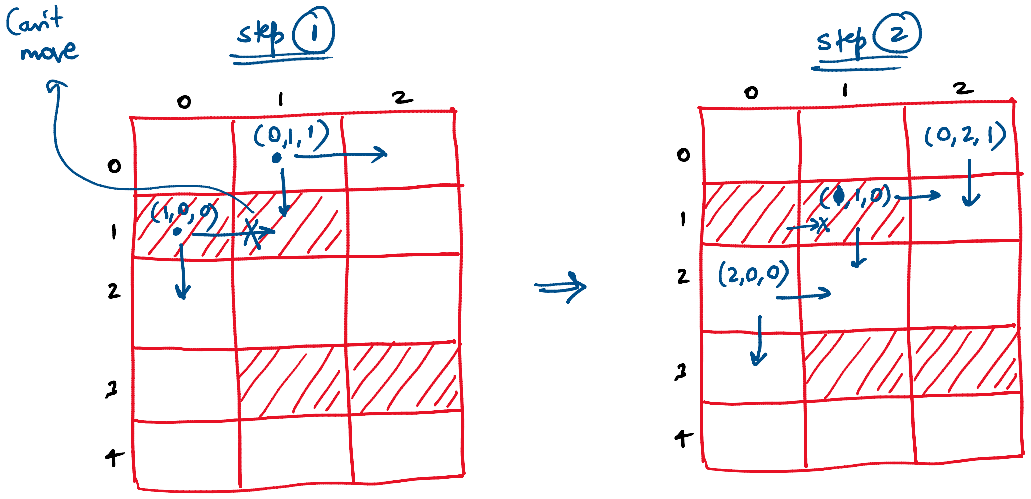


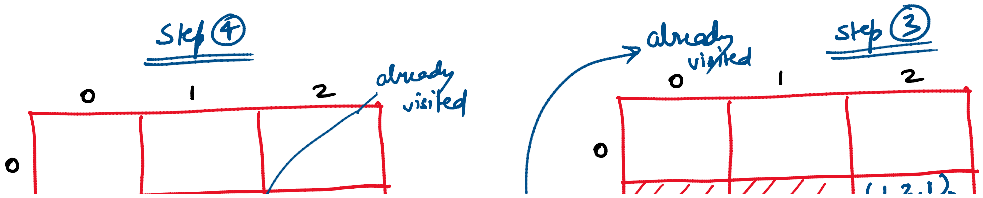
Thus, we will also have Eliminatig Power left as a variable ⇐

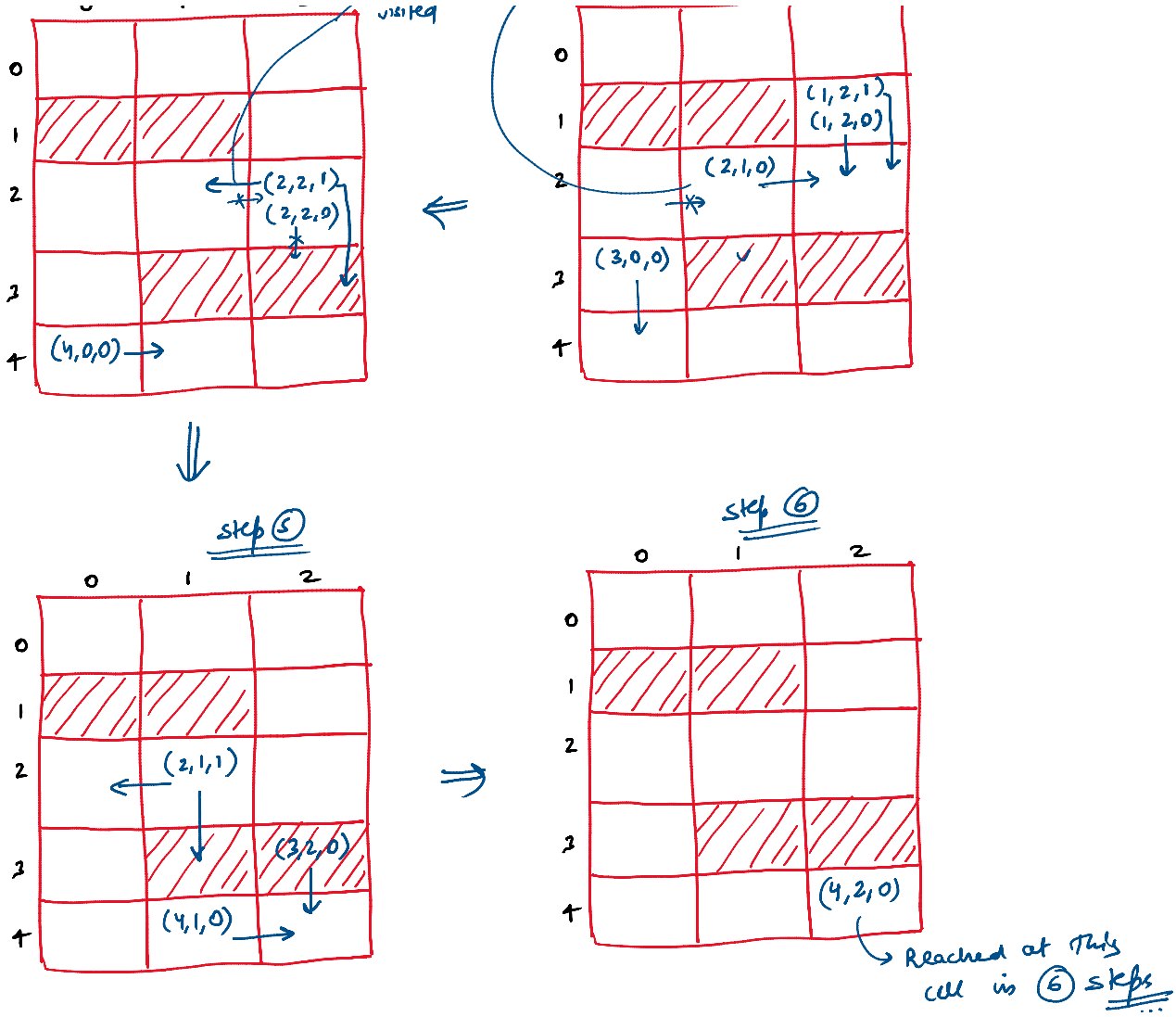
⇒ Now we do BFS & every time we encounter a cell that we need to go to (neighbour cell) → we check if that is not visited at this combination {i, j, Eliminaby Power}.

We can only visit a blocked cell when we have eliminaby power left with me !!








visited
(2,2,1)
(2,2,0)
(4,0,0)
(1,2,1)
(1,2,0)
(2,1,0)
(3,0,0)
step ⑤
(2,1,1)
(3,2,0)
(4,1,0)
step ⑥
(4,2,0)
Reached at this cell in ⑥ steps

```cpp
class Solution {
public:
    int shortestPath(vector<vector<int>>& grid, int k) {
        int m = grid.size(), n = grid[0].size();
        vector<vector<vector<bool>>> v(m, vector<vector<bool>>(n, vector<bool>(k + 1)));
        queue<vector<int>> q;
        int steps = 0;
        q.push({0, 0, k});
        vector<vector<int>> dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

        while (!q.empty()) {
            int size = q.size();
            while (size-- > 0) {
                vector<int> curr = q.front();
                q.pop();
                if (curr[0] == m - 1 && curr[1] == n - 1) return steps;

                for (vector<int>& d : dirs) {
                    int i = curr[0] + d[0];
                    int j = curr[1] + d[1];
                    int obs = curr[2];

                    if (i >= 0 && i < m && j >= 0 && j < n) {
                        if (grid[i][j] == 0 && !v[i][j][obs]) {
                            q.push({i, j, obs});
                            v[i][j][obs] = true;
                        } else if (grid[i][j] == 1 && obs > 0 && !v[i][j][obs-1]) {
                            q.push({i, j, obs - 1});
                            v[i][j][obs - 1] = true;
                        }
                    }
                }
            }
            ++steps;
        }
        return -1;
    }
};
```

To visit the state [i, j, e]
queue will store [i, j, e]
e: Eliminating Power left
i: cell i
j: cell j
4 directionally connected
Standard BFS
As soon as we reach last cell.
Going on to the neighbours
If it a valid location (Not out of bounds)
We can move easily
It is an obstacle so, we should have eliminating power left.
Increment steps every time
If we can't reach cell (m-1)(n-1)

time: $O(m * n * k)$
$m * n$: doing BFS on every cell of the matrix
k: Every cell can be visited at max k times.

Space: $O(m * n * k)$

Space: $O(m * n * k)$

↳ for visited vector & queue too.

C++ Auto

```cpp
class Solution {
public:
    int shortestPath(vector<vector<int>>& grid, int k) {
        int m = grid.size(), n = grid[0].size();
        vector<vector<vector<bool>>> v(m, vector<vector<bool>>(n, vector<bool>(k + 1)));
        queue<vector<int>> q;
        int steps = 0;
        q.push({0, 0, k});
        vector<vector<int>> dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

        while (!q.empty()) {
            int size = q.size();
            while (size-- > 0) {
                vector<int> curr = q.front();
                q.pop();
                if (curr[0] == m - 1 && curr[1] == n - 1) return steps;

                for (vector<int>& d : dirs) {
                    int i = curr[0] + d[0];
                    int j = curr[1] + d[1];
                    int obs = curr[2];

                    if (i >= 0 && i < m && j >= 0 && j < n) {
                        if (grid[i][j] == 0 && !v[i][j][obs]) {
                            q.push({i, j, obs});
                            v[i][j][obs] = true;
                        } else if (grid[i][j] == 1 && obs > 0 && !v[i][j][obs-1]) {
                            q.push({i, j, obs - 1});
                            v[i][j][obs - 1] = true;
                        }
                    }
                }
            }
            ++steps;
        }
        return -1;
    }
};
```


```cpp
class Solution {
public:
    int shortestPath(vector<vector<int>>& grid, int k) {
        int m = grid.size(), n = grid[0].size();
        vector<vector<vector<bool>>> v(m, vector<vector<bool>>(n, vector<bool>(k + 1)));
        queue<vector<int>> q;
        int steps = 0;
        q.push({0, 0, k});
        vector<vector<int>> dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

        while (!q.empty()) {
            int size = q.size();
            while (size-- > 0) {
                vector<int> curr = q.front();
                q.pop();
                if (curr[0] == m - 1 && curr[1] == n - 1) return steps;

                for (vector<int>& d : dirs) {
                    int i = curr[0] + d[0];
```

```
                    int j = curr[1] + d[1];
                    int obs = curr[2];

                    if (i >= 0 && i < m && j >= 0 && j < n) {
                        if (grid[i][j] == 0 && !v[i][j][obs]) {
                            q.push({i, j, obs});
                            v[i][j][obs] = true;
                        } else if (grid[i][j] == 1 && obs > 0 && !v[i][j][obs-1]) {
                            q.push({i, j, obs - 1});
                            v[i][j][obs - 1] = true;
                        }
                    }
                }
            }
            ++steps;
        }
        return -1;
    }
};
```

Java Auto

```java
class Solution {
    public int shortestPath(int[][] grid, int k) {

        int m = grid.length, n = grid[0].length;
        int[][] DIR = {{0,1}, {1,0}, {0,-1}, {-1,0}};
        boolean[][][] v = new boolean[m][n][k+1];
        Queue<int[]> q = new LinkedList<>();
        int steps = 0;
        q.offer(new int[]{0,0,k});
        while(!q.isEmpty()) {
            int size = q.size();

            while(size-- > 0) {
                int[] curr = q.poll();
                if(curr[0] == m-1 && curr[1] == n-1) return steps;
                for(int[] d : DIR) {
                    int i = curr[0]+d[0];
                    int j = curr[1]+d[1];
                    int obs = curr[2];

                    if(i >= 0 && i < m && j >= 0 && j < n) {
                        if(grid[i][j] == 0 && !v[i][j][obs]) {
                            q.offer(new int[]{i,j,obs});
                            v[i][j][obs] = true;
                        }
                        else if(grid[i][j] == 1 && obs > 0 && !v[i][j][obs-1]) {
                            q.offer(new int[]{i,j,obs-1});
                            v[i][j][obs-1] = true;
                        }
                    }
                }
            }
            ++steps;
        }
        return -1;
    }
}
```

```java
class Solution {
    public int shortestPath(int[][] grid, int k) {

        int m = grid.length, n = grid[0].length;
        int[][] DIR = {{0,1}, {1,0}, {0,-1}, {-1,0}};
        boolean[][][] v = new boolean[m][n][k+1];
        Queue<int[]> q = new LinkedList<>();
        int steps = 0;
        q.offer(new int[]{0,0,k});
        while(!q.isEmpty()) {
            int size = q.size();

            while(size-- > 0) {
```

```
                int[] curr = q.poll();
                if(curr[0] == m-1 && curr[1] == n-1) return steps;
                for(int[] d : DIR) {
                    int i = curr[0]+d[0];
                    int j = curr[1]+d[1];
                    int obs = curr[2];

                    if(i >= 0 && i < m && j >= 0 && j < n) {
                        if(grid[i][j] == 0 && !v[i][j][obs]) {
                            q.offer(new int[]{i,j,obs});
                            v[i][j][obs] = true;
                        }
                        else if(grid[i][j] == 1 && obs > 0 && !v[i][j][obs-1]) {
                            q.offer(new int[]{i,j,obs-1});
                            v[i][j][obs-1] = true;
                        }
                    }
                }
            }
            ++steps;
        }
        return -1;
    }
}
```