LeetCode 417

https://leetcode.com/problems/pacific-atlantic-water-flow/description/

Yifeng Zeng

Description

417. Pacific Atlantic Water Flow

Given an m x n matrix of non-negative integers representing the height of each unit cell in a continent, the "Pacific ocean" touches the left and top edges of the matrix and the "Atlantic ocean" touches the right and bottom edges.

Water can only flow in four directions (up, down, left, or right) from a cell to another one with height equal or lower.

Find the list of grid coordinates where water can flow to both the Pacific and Atlantic ocean.

Note:
The order of returned grid coordinates does not matter.
Both m and n are less than 150.

Example:

Given the following 5x5 matrix:

```
Pacific ~   ~   ~   ~   ~
     ~  1   2   2   3  (5) *
     ~  3   2   3  (4) (4) *
     ~  2   4  (5)  3   1  *
     ~ (6) (7)  1   4   5  *
     ~ (5)  1   1   2   4  *
        *   *   *   *   * Atlantic
```

Return:

[[0, 4], [1, 3], [1, 4], [2, 2], [3, 0], [3, 1], [4, 0]] (positions with parentheses in above matrix).

Idea Report

This problem can be represented as a search problem in an undirected graph. Each cell is a node (vertex), and we search from each node to see if it can go to top/left or bottom/right with the flow in four directions. My primitive idea is to seach from each node in the matrix, see if it can go to top or left edge of the matrix and see if it can also go to bottom or right edge of the matrix, if both are true, we can just put it in the result array list.

Code:

```
class Solution {
    // TLE
    public List<int[]> pacificAtlantic(int[][] matrix) {
        List<int[]> res = new ArrayList<>();
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (helper(matrix, i, j)) {
                    res.add(new int[]{i, j});
                }
            }
        }

        return res;
    }

    private boolean helper(int[][] matrix, int row, int col) {
        int[] dx = {0, 0, 1, -1};
        int[] dy = {1, -1, 0, 0};
        int[][] visited = new int[matrix.length][matrix[0].length];
        Deque<int[]> q = new LinkedList<>();
        q.offer(new int[]{row, col});
        visited[row][col] = 1;
        int[] count = new int[2];

        while (!q.isEmpty()) {
            if (count[0] > 0 && count[1] > 0) {
                return true;
            }
            int[] cur = q.poll();
            int r = cur[0];
            int c = cur[1];
            if (r == 0 || c == 0) {
                count[0]++;
            }
            if (r == matrix.length - 1 || c == matrix[0].length - 1) {
                count[1]++;
            }
            if (count[0] > 0 && count[1] > 0) {
                return true;
            }

            for (int i = 0; i < dx.length; i++) {
                r = cur[0] + dx[i];
                c = cur[1] + dy[i];

                if (!isValid(matrix, r, c)) {
                    continue;
                }

                if (visited[r][c] == 0
                    && matrix[r][c] <= matrix[cur[0]][cur[1]]) {
                    visited[r][c] = 1;
                    q.offer(new int[]{r, c});
                }
            }
        }

        return count[0] > 0 && count[1] > 0;
    }

    private boolean isValid(int[][] matrix, int r, int c) {
        if (0 <= r && r < matrix.length && 0 <= c && c < matrix[0].length) {
            return true;
        }
        return false;
    }
}
```

The primitive idea method has a lot of duplications (TLE). Since we are looking for a node to the edge and it is undrected, so we can also search from edge to each node. So we can have two visited matrices, one is to store the nodes can be visited from top or left edge, another matrix is to store the nodes can be visited from bottm or right edge. We add the nodes appears in both matrices into the result array list.

Code:

```
class Solution {
    // BFS AC
    public List<int[]> pacificAtlantic(int[][] matrix) {
        List<int[]> res = new ArrayList<>();
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return res;
        }

        int[][] visitedPacific = new int[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
            helper(matrix, i, 0, visitedPacific);
        }

        for (int i = 0; i < matrix[0].length; i++) {
            helper(matrix, 0, i, visitedPacific);
        }

        int[][] visitedAtlantic = new int[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
            helper(matrix, i, matrix[0].length - 1, visitedAtlantic);
        }

        for (int i = 0; i < matrix[0].length; i++) {
            helper(matrix, matrix.length - 1, i, visitedAtlantic);
        }

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (visitedPacific[i][j] == 1 && visitedAtlantic[i][j] == 1) {
                    res.add(new int[]{i, j});
                }
            }
        }

        return res;
    }

    // BFS
    private void helper(int[][] matrix, int row, int col, int[][] visited) {
        int[] dx = {0, 0, 1, -1};
        int[] dy = {1, -1, 0, 0};
        Deque<int[]> q = new LinkedList<>();
        q.offer(new int[]{row, col});
        visited[row][col] = 1;

        while (!q.isEmpty()) {
            int[] cur = q.poll();
            for (int i = 0; i < dx.length; i++) {
                int r = cur[0] + dx[i];
                int c = cur[1] + dy[i];
                if (isValid(visited, r, c)
                    && matrix[cur[0]][cur[1]] <= matrix[r][c]) {
                    visited[r][c] = 1;
                    q.offer(new int[]{r, c});
                }
            }
        }
    }

    private boolean isValid(int[][] visited, int r, int c) {
        if (0 <= r && r < visited.length && 0 <= c && c < visited[0].length) {
            return visited[r][c] == 0;
        }
        return false;
    }
}
```

Summary

- Node to node path in a matrix can be represented as a search search problem is an undirected graph.
- Consider to search from destination to source, it may be more efficient.
- Follow up, this may also be solved by using DFS.

Follow up

Same as the second solution above, but using DFS.

Code:

```
class Solution {
    // DFS AC
    public List<int[]> pacificAtlantic(int[][] matrix) {
        List<int[]> res = new ArrayList<>();
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return res;
        }

        int[][] visitedPacific = new int[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
            helper(matrix, i, 0, visitedPacific);
        }

        for (int i = 0; i < matrix[0].length; i++) {
            helper(matrix, 0, i, visitedPacific);
        }
```

```java
        int[][] visitedAtlantic = new int[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
            helper(matrix, i, matrix[0].length - 1, visitedAtlantic);
        }

        for (int i = 0; i < matrix[0].length; i++) {
            helper(matrix, matrix.length - 1, i, visitedAtlantic);
        }

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (visitedPacific[i][j] == 1 && visitedAtlantic[i][j] == 1) {
                    res.add(new int[]{i, j});
                }
            }
        }

        return res;
    }

    private void helper(int[][] matrix, int row, int col, int[][] visited) {
        visited[row][col] = 1;
        int[] dx = {0, 0, 1, -1};
        int[] dy = {1, -1, 0, 0};
        for (int i = 0; i < dx.length; i++) {
            int r = row + dx[i];
            int c = col + dy[i];
            if (isValid(visited, r, c) && matrix[row][col] <= matrix[r][c]) {
                helper(matrix, r, c, visited);
            }
        }
    }

    private boolean isValid(int[][] visited, int r, int c) {
        if (0 <= r && r < visited.length && 0 <= c && c < visited[0].length) {
            return visited[r][c] == 0;
        }
        return false;
    }
}
```