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:

1. The order of returned grid coordinates does not matter.
2. 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 parenthese
s in above matrix).
```

Solution 1

1. Two Queue and add all the Pacific border to one queue; Atlantic border to another queue.
2. Keep a visited matrix for each queue. In the end, add the cell visited by two queue to the result.
   BFS: Water flood from ocean to the cell. Since water can only flow from high/equal cell to low cell, add the neighboor cell with height larger or equal to current cell to the queue and mark as visited.

```
public class Solution {
    int[][]dir = new int[][]{{1,0},{-1,0},{0,1},{0,-1}};
    public List<int[]> pacificAtlantic(int[][] matrix) {
        List<int[]> res = new LinkedList<>();
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return res;
        }
        int n = matrix.length, m = matrix[0].length;
        //One visited map for each ocean
        boolean[][] pacific = new boolean[n][m];
        boolean[][] atlantic = new boolean[n][m];
        Queue<int[]> pQueue = new LinkedList<>();
        Queue<int[]> aQueue = new LinkedList<>();
        for(int i=0; i<n; i++){ //Vertical border
            pQueue.offer(new int[]{i, 0});
            aQueue.offer(new int[]{i, m-1});
            pacific[i][0] = true;
            atlantic[i][m-1] = true;
        }
        for(int i=0; i<m; i++){ //Horizontal border
            pQueue.offer(new int[]{0, i});
            aQueue.offer(new int[]{n-1, i});
            pacific[0][i] = true;
            atlantic[n-1][i] = true;
        }
        bfs(matrix, pQueue, pacific);
        bfs(matrix, aQueue, atlantic);
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                if(pacific[i][j] && atlantic[i][j])
                    res.add(new int[]{i,j});
            }
        }
        return res;
    }
    public void bfs(int[][]matrix, Queue<int[]> queue, boolean[][]visited){
        int n = matrix.length, m = matrix[0].length;
        while(!queue.isEmpty()){
            int[] cur = queue.poll();
            for(int[] d:dir){
                int x = cur[0]+d[0];
                int y = cur[1]+d[1];
                if(x<0 || x>=n || y<0 || y>=m || visited[x][y] || matrix[x][y] <
matrix[cur[0]][cur[1]]){
                    continue;
                }
                visited[x][y] = true;
                queue.offer(new int[]{x, y});
            }
        }
    }
}
```

DFS version:

```java
public class Solution {
    public List<int[]> pacificAtlantic(int[][] matrix) {
        List<int[]> res = new LinkedList<>();
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return res;
        }
        int n = matrix.length, m = matrix[0].length;
        boolean[][]pacific = new boolean[n][m];
        boolean[][]atlantic = new boolean[n][m];
        for(int i=0; i<n; i++){
            dfs(matrix, pacific, Integer.MIN_VALUE, i, 0);
            dfs(matrix, atlantic, Integer.MIN_VALUE, i, m-1);
        }
        for(int i=0; i<m; i++){
            dfs(matrix, pacific, Integer.MIN_VALUE, 0, i);
            dfs(matrix, atlantic, Integer.MIN_VALUE, n-1, i);
        }
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (pacific[i][j] && atlantic[i][j])
                    res.add(new int[] {i, j});
        return res;
    }

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

    public void dfs(int[][]matrix, boolean[][]visited, int height, int x, int y){
        int n = matrix.length, m = matrix[0].length;
        if(x<0 || x>=n || y<0 || y>=m || visited[x][y] || matrix[x][y] < height)
            return;
        visited[x][y] = true;
        for(int[]d:dir){
            dfs(matrix, visited, matrix[x][y], x+d[0], y+d[1]);
        }
    }
}
```

written by star1993 original link here

Solution 2

```
class Solution {
public:
    vector<pair<int, int>> res;
    vector<vector<int>> visited;
    void dfs(vector<vector<int>>& matrix, int x, int y, int pre, int preval){
        if (x < 0 || x >= matrix.size() || y < 0 || y >= matrix[0].size()
                || matrix[x][y] < pre || (visited[x][y] & preval) == preval)
            return;
        visited[x][y] |= preval;
        if (visited[x][y] == 3) res.push_back({x, y});
        dfs(matrix, x + 1, y, matrix[x][y], visited[x][y]); dfs(matrix, x - 1, y,
matrix[x][y], visited[x][y]);
        dfs(matrix, x, y + 1, matrix[x][y], visited[x][y]); dfs(matrix, x, y - 1,
matrix[x][y], visited[x][y]);
    }

    vector<pair<int, int>> pacificAtlantic(vector<vector<int>>& matrix) {
        if (matrix.empty()) return res;
        int m = matrix.size(), n = matrix[0].size();
        visited.resize(m, vector<int>(n, 0));
        for (int i = 0; i < m; i++) {
            dfs(matrix, i, 0, INT_MIN, 1);
            dfs(matrix, i, n - 1, INT_MIN, 2);
        }
        for (int i = 0; i < n; i++) {
            dfs(matrix, 0, i, INT_MIN, 1);
            dfs(matrix, m - 1, i, INT_MIN, 2);
        }
        return res;
    }
};
```

written by zyoppy008 original link here

Solution 3

```
/*
1.Naive solution:
    Standard dfs, which means for each point, we check if it can reach both pacifi
c and atlantic,
    for each point, we can possibly check all the rest of points, O(m*n * m*n)

2.A little improvement:
    What about we 4 hash tables, they keep track of all the points we know so far
that
        can reach atlantic
        cannot reach atlantic
        can reach pacific
        cannot reach pacific
    It's doable, still hit TLE, although I didn't hit TLE when not submitting the
code, but running it using the provided testing environment

3.On the other hand, we can consider the flip side
    We can let the pacific and atlantic ocean "flow into" the matrix as much as po
ssible,
    using 2 boolean arrays, one for each ocean.
    The result are the points that are true in both boolean table
*/


public class Solution {
    public List<int[]> pacificAtlantic(int[][] matrix) {
        List<int[]> result = new ArrayList<int[]>();
        if(matrix.length == 0 || matrix[0].length == 0) return result;
        boolean[][] pacific = new boolean[matrix.length][matrix[0].length];  // t
he pacific boolean table
        boolean[][] atlantic = new boolean[matrix.length][matrix[0].length]; // t
he atlantic booean table
        //initially, all the top and left cells are flooded with pacific water
        //and all the right and bottom cells are flooded with atlantic water
        for(int i = 0; i < matrix.length; i++){
            pacific[i][0] = true;
            atlantic[i][matrix[0].length-1] = true;
        }
        for(int i = 0; i < matrix[0].length; i++){
            pacific[0][i] = true;
            atlantic[matrix.length-1][i] = true;
        }
        //we go around the matrix and try to flood the matrix from 4 side.
        for(int i = 0; i < matrix.length; i++){
            boolean[][] pacificVisited = new boolean[matrix.length][matrix[0].len
gth];
            boolean[][] atlanticVisited = new boolean[matrix.length][matrix[0].le
ngth];
            water(pacific, pacificVisited, matrix, i,0);
            water(atlantic, atlanticVisited, matrix, i, matrix[0].length - 1);

        }
        for(int i = 0; i < matrix[0].length; i++){
            boolean[][] pacificVisited = new boolean[matrix.length][matrix[0].len
gth];
            boolean[][] atlanticVisited = new boolean[matrix.length][matrix[0].le
ngth];
            water(pacific, pacificVisited, matrix, 0,i);
            water(atlantic, atlanticVisited, matrix, matrix.length - 1, i);

        }
        //check the shared points among 2 tables
        for(int i = 0; i < matrix.length; i++){
            for(int j = 0; j < matrix[0].length; j++){
                if(pacific[i][j] && atlantic[i][j]){
                    int[] element = {i,j};
                    result.add(element);
                }
            }
        }
        return result;
    }
    //the flood function
    private void water(boolean[][] wet, boolean[][] visited, int[][] matrix, int
i , int j){
        wet[i][j] = true;
        visited[i][j] = true;
        int[] x = {0,0,1,-1};
        int[] y = {1,-1,0,0};
        for(int k = 0; k < 4; k++){
            if(i+y[k] >= 0 && i+y[k] < matrix.length && j+x[k] >= 0 && j+x[k] < m
atrix[0].length
                && !visited[i+y[k]][j+x[k]] && matrix[i+y[k]][j+x[k]] >= matrix[i
][j]){
                water(wet, visited, matrix, i+y[k], j+x[k]);
            }
        }
    }
}````

P.S Sometimes you choose an option just because the alternative is just worse.....
```

written by DonaldTrump original link here