

Longest Line of Consecutive One in Matrix

Given a 01 matrix **M**, find the longest line of consecutive one in the matrix. The line could be horizontal, vertical, diagonal or anti-diagonal.

Example:

Input:

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

Output: 3

Hint: The number of elements in the given matrix will not exceed 10,000.

Solution 1

```
public int longestLine(int[][] M) {
    int n = M.length, max = 0;
    if (n == 0) return max;
    int m = M[0].length;
    int[][][] dp = new int[n][m][4];
    for (int i=0; i<n; i++)
        for (int j=0; j<m; j++) {
            if (M[i][j] == 0) continue;
            for (int k=0; k<4; k++) dp[i][j][k] = 1;
            if (j > 0) dp[i][j][0] += dp[i][j-1][0]; // horizontal line
            if (j > 0 && i > 0) dp[i][j][1] += dp[i-1][j-1][1]; // anti-diagonal line
            if (i > 0) dp[i][j][2] += dp[i-1][j][2]; // vertical line
            if (j < m-1 && i > 0) dp[i][j][3] += dp[i-1][j+1][3]; // diagonal line
            max = Math.max(max, Math.max(dp[i][j][0], dp[i][j][1]));
            max = Math.max(max, Math.max(dp[i][j][2], dp[i][j][3]));
        }
    return max;
}
```

Note that each cell of the DP table only depends on the current row or previous row so you can easily optimize the above algorithm to use only $O(m)$ space.

written by [compton_scatter](#) original link [here](#)

Solution 2

For each **unvisited** direction of each **1**, we search length of adjacent **1**s and mark those **1**s as **visited** in that direction. And we only need to search 4 directions: **right, down, down-right, down-left**. We only access each cell at max 4 times, so time complexity is $O(mn)$. m = number of rows, n = number of columns.

```
public class Solution {
    public int longestLine(int[][] M) {
        int m = M.length;
        if (m <= 0) return 0;
        int n = M[0].length;
        if (n <= 0) return 0;

        Set<String> horizontal = new HashSet<>();
        Set<String> vertical = new HashSet<>();
        Set<String> diagonal = new HashSet<>();
        Set<String> antidiagonal = new HashSet<>();
        int max = 0;

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (M[i][j] == 0) continue;
                String pos = i + "," + j;
                if (!horizontal.contains(pos)) {
                    int count = 0;
                    for (int k = j; k < n; k++) {
                        if (M[i][k] == 1) {
                            count++;
                            horizontal.add(i + "," + k);
                        }
                    }
                    else break;
                }
                max = Math.max(max, count);
            }
            if (!vertical.contains(pos)) {
                int count = 0;
                for (int k = i; k < m; k++) {
                    if (M[k][j] == 1) {
                        count++;
                        vertical.add(k + "," + j);
                    }
                }
                else break;
            }
            max = Math.max(max, count);
        }
        if (!diagonal.contains(pos)) {
            int count = 0;
            for (int k = i, l = j; k < m && l < n; k++, l++) {
                if (M[k][l] == 1) {
                    count++;
                    diagonal.add(k + "," + l);
                }
            }
            else break;
        }
    }
}
```

```

        max = Math.max(max, count);
    }
    if (!antidiagonal.contains(pos)) {
        int count = 0;
        for (int k = i, l = j; k < m && l >= 0; k++, l--) {
            if (M[k][l] == 1) {
                count++;
                antidiagonal.add(k + "," + l);
            }
            else break;
        }
        max = Math.max(max, count);
    }
}

return max;
}
}

```

A more concise version.

```

public class Solution {
    public int longestLine(int[][] M) {
        int m = M.length;
        if (m <= 0) return 0;
        int n = M[0].length;
        if (n <= 0) return 0;

        int max = 0;
        int[][] dirs = {{0, 1}, {1, 0}, {1, 1}, {1, -1}};
        List<Set<String>> memo = new ArrayList<>();
        for (int i = 0; i < 4; i++) {
            memo.add(new HashSet<String>());
        }

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (M[i][j] == 0) continue;
                String pos = i + "," + j;
                for (int k = 0; k < 4; k++) {
                    if (!memo.get(k).contains(pos)) {
                        int count = 0;
                        for (int r = i, c = j; r < m && r >= 0 && c < n && c >= 0;
                             r += dirs[k][0], c += dirs[k][1]) {
                            if (M[r][c] == 1) {
                                count++;
                                memo.get(k).add(r + "," + c);
                            }
                            else break;
                        }
                        max = Math.max(max, count);
                    }
                }
            }
        }

        return max;
    }
}

```

written by [shawngao](#) original link [here](#)

Solution 3

```
public int longestLine(int[][] M) {
    if (M.length == 0 || M[0].length == 0) {
        return 0;
    }
    int max = 0;
    int[] col = new int[M[0].length];
    int[] diag = new int[M.length + M[0].length];
    int[] antiD = new int[M.length + M[0].length];
    for (int i = 0; i < M.length; i++) {
        int row = 0;
        for (int j = 0; j < M[0].length; j++) {
            if (M[i][j] == 1) {
                row++;
                col[j]++;
                diag[j + i]++;
                antiD[j - i + M.length]++;
                max = Math.max(max, row);
                max = Math.max(max, col[j]);
                max = Math.max(max, diag[j + i]);
                max = Math.max(max, antiD[j - i + M.length]);
            } else {
                row = 0;
                col[j] = 0;
                diag[j + i] = 0;
                antiD[j - i + M.length] = 0;
            }
        }
    }
    return max;
}
```

written by [ihaveayaya](#) original link [here](#)

From [LeetCoder](#).