01 Matrix

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Example 1:

Input:

```
0 0 0
0 1 0
0 0 0
```

Output:

```
0 0 0
0 1 0
0 0 0
```

Example 2:

Input:

```
0 0 0
0 1 0
1 1 1
```

Output:

```
0 0 0
0 1 0
1 2 1
```

Note:

1. The number of elements of the given matrix will not exceed 10,000.
2. There are at least one 0 in the given matrix.
3. The cells are adjacent in only four directions: up, down, left and right.

Solution 1

General idea is `BFS`. Some small tricks:

1. At beginning, set cell value to `Integer.MAX_VALUE` if it is not `0`.
2. If newly calculated distance `>=` current distance, then we don't need to explore that cell again.

```
public class Solution {
    public List<List<Integer>> updateMatrix(List<List<Integer>> matrix) {
        int m = matrix.size();
        int n = matrix.get(0).size();

        Queue<int[]> queue = new LinkedList<>();
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (matrix.get(i).get(j) == 0) {
                    queue.offer(new int[] {i, j});
                }
                else {
                    matrix.get(i).set(j, Integer.MAX_VALUE);
                }
            }
        }

        int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

        while (!queue.isEmpty()) {
            int[] cell = queue.poll();
            for (int[] d : dirs) {
                int r = cell[0] + d[0];
                int c = cell[1] + d[1];
                if (r < 0 || r >= m || c < 0 || c >= n ||
                    matrix.get(r).get(c) <= matrix.get(cell[0]).get(cell[1]) + 1)
continue;
                queue.add(new int[] {r, c});
                matrix.get(r).set(c, matrix.get(cell[0]).get(cell[1]) + 1);
            }
        }

        return matrix;
    }
}
```

written by shawngao original link here

Solution 2

In the first sweep, we visit each entry in natural order and answer[i][j] = min(Integer.MAX_VALUE, min(answer[i - 1][j], answer[i][j - 1]) + 1).
in the second sweep, we visit each entry in reverse order and answer[i][j] = min(answer[i][j], min(answer[i + 1][j], answer[i][j + 1]) + 1).

```
public List<List<Integer>> updateMatrix(List<List<Integer>> matrix) {
        List<List<Integer>> answer = new LinkedList();
  if(matrix.size() == 0) return answer;
  int[][] array = new int[matrix.size()][matrix.get(0).size()];
  int i = 0, j = 0;
  for(List<Integer> list : matrix) {
   for(Integer x : list) {
    if(x == 0) {
     array[i][j] = 0;
    }
    else {
     int left = Integer.MAX_VALUE - 1, top = Integer.MAX_VALUE - 1;
     if(i - 1 >= 0) top = array[i - 1][j];
     if(j - 1 >= 0) left = array[i][j - 1];
     array[i][j] = Math.min(Integer.MAX_VALUE - 1, Math.min(top, left) + 1);
    }
    j++;
   }
   j = 0;
   i++;
  }
  for(int k = array.length - 1; k >= 0; k--) {
   for(int m = array[0].length - 1; m >= 0; m--) {
    if(array[k][m] != 0 && array[k][m] != 1) {
     int down = Integer.MAX_VALUE - 1, right = Integer.MAX_VALUE - 1;
     if(k + 1 < array.length) down = array[k + 1][m];
     if(m + 1 < array[0].length) right = array[k][m + 1];
     array[k][m] = Math.min(array[k][m], Math.min(down, right) + 1);
    }
   }
  }
  for(int[] l : array) {
   List<Integer> tmp = new LinkedList();
   for(int n : l) {
    tmp.add(n);
   }
   answer.add(tmp);
  }
  return answer;
    }
```

written by qswawrq original link here

Solution 3

Simple DP, just check if the neigbours know how far are they to the nearest 0 when I don't know

```
class Solution {
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
        int h=matrix.size(), w=matrix[0].size();
        vector<vector<int>> dp(h,vector<int>(w,INT_MAX));
        for(int times=0;times<=1;times++) // two passes, first forward then backw
ard
            for(int i=times?h-1:0;times?i>=0:i<h;times?i--:i++)
                for(int j=times?w-1:0;times?j>=0:j<w;times?j--:j++)
                        if(matrix[i][j]==0)
                            dp[i][j]=0;
                        else if(dp[i][j]!=0) {
                            if(i&&dp[i-1][j]!=INT_MAX&&dp[i][j]>dp[i-1][j]+1) //
look up
                                dp[i][j]=dp[i-1][j]+1;
                            if(j&&dp[i][j-1]!=INT_MAX&&dp[i][j]>dp[i][j-1]+1) //
look left
                                dp[i][j]=dp[i][j-1]+1;
                            if(i<h-1&&dp[i+1][j]!=INT_MAX&&dp[i][j]>dp[i+1][j]+1)
// look down
                                dp[i][j]=dp[i+1][j]+1;
                            if(j<w-1&&dp[i][j+1]!=INT_MAX&&dp[i][j]>dp[i][j+1]+1)
// look right
                                dp[i][j]=dp[i][j+1]+1;
                        }
        return dp;
    }
};
```

written by luming89 original link here