LeetCode 542

https://leetcode.com/problems/01-matrix/description/

Yifeng Zeng

Description

542. 01 Matrix

Idea Report

(This is a pretty standard graph coloring problem.) When we look for a '0' if its neighbor is not '0' we add value 1 to the distance and search from there. If we see a neighbor has a smaller value than the current distance, it means there is another path to that neighbor that has a shorter distance so we can skip this neighbor. Searching from '0' may not be a very effective way, so we can do a preprocess to see where is the boundary of the 0 and non-zeros, then we start from those 1s so that the search would be faster.

Code

```
class Solution {
  // BFS
  final int[] dx = {0, 1, 0, -1};
  final int[] dy = {1, 0, -1, 0};

  public int[][] updateMatrix(int[][] matrix) {
      int rows = matrix.length;
      int cols = matrix[0].length;
      Deque<int[]> q = new LinkedList<>();

      for (int r = 0; r < rows; r++) {
          for (int c = 0; c < cols; c++) {
              if (matrix[r][c] == 1) {
                  if (hasZeroNeiboughor(matrix, r, c)) {
                      q.offer(new int[]{r, c});
                  } else {
                      matrix[r][c] = Integer.MAX_VALUE;
```

```
                    }
                }
            }
        }

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

        return matrix;
    }

    private boolean hasZeroNeiboughor(int[][] matrix, int r0, int c0) {
        for (int i = 0; i < dx.length; i++) {
            int r = r0 + dx[i];
            int c = c0 + dy[i];
            if (isValid(matrix, r, c) && matrix[r][c] == 0) {
                return true;
            }
        }
        return false;
    }

    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;
    }
}
```

Code

```
class Solution {
    // DFS
    final int[] dx = {0, 1, 0, -1};
```

```java
    final int[] dy = {1, 0, -1, 0};

    public int[][] updateMatrix(int[][] matrix) {
        for (int r = 0; r < matrix.length; r++) {
            for (int c = 0; c < matrix[0].length; c++) {
                if (matrix[r][c] == 1) {
                    if (!hasZeroNeiboughor(matrix, r, c)) {
                        matrix[r][c] = Integer.MAX_VALUE;
                    }
                }
            }
        }

        for (int r = 0; r < matrix.length; r++) {
            for (int c = 0; c < matrix[0].length; c++) {
                if (matrix[r][c] == 1) {
                    dfsHelper(matrix, r, c);
                }
            }
        }

        return matrix;
    }

    private void dfsHelper(int[][] matrix, int r0, int c0) {
        for (int i = 0; i < dx.length; i++) {
            int r = r0 + dx[i];
            int c = c0 + dy[i];
            if (isValid(matrix, r, c) && matrix[r0][c0] + 1 < matrix[r][c]) {
                matrix[r][c] = matrix[r0][c0] + 1;
                dfsHelper(matrix, r, c);
            }
        }
    }

    private boolean hasZeroNeiboughor(int[][] matrix, int r0, int c0) {
        for (int i = 0; i < dx.length; i++) {
            int r = r0 + dx[i];
            int c = c0 + dy[i];
            if (isValid(matrix, r, c) && matrix[r][c] == 0) {
                return true;
            }
        }
        return false;
    }

    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;
    }
```

```
}
```

Summary

- Standard BFS, DFS problem.
- Modulize the functions.
- Use int[] dx, dy to simplify the moving up, down, left, right.
- Use isValid() to see if a move is inbound.