For lab 5 you wrote a program that created a maze where each cell was reachable from any other cell and there was a unique path from the start to the destination. Your program works by removing one wall at a time (making sure that the cells separated by that wall were not reachable from each other) until the graph representing the maze has exactly one connected component. If the maze has \( n \) cells, your program would remove exactly \( n - 1 \) walls to reach this situation.

If you removed less than \( n - 1 \) walls, the resulting maze would have more than one connected component and some cells would not be reachable from the start cell. If you removed more than \( n - 1 \) walls (notice that after removing \( n - 1 \) walls, all remaining walls separate cells that are reachable from each other), you could have multiple paths from the source to the destination.

Your task for this lab consists of the following:

1. Modify your maze-building program to allow for both cases mentioned above. Your program should display \( n \) the number of cells, and ask the user for \( m \), the number of walls to remove, then display a message indicating one of the following:
   
   (a) A path from source to destination is not guaranteed to exist (when \( m < n - 1 \))
   (b) There is a unique path from source to destination (when \( m = n - 1 \))
   (c) There is at least on path from source to destination (when \( m > n - 1 \))

2. Implement an algorithm that takes the maze and builds a graph representing the maze using an adjacency list. Represent each cell by a node, and for every neighboring cell, add an edge if there is no wall separating the two cells.

3. Implement an algorithm to find a path from the source to the destination, if it exists, using the stack-based implementation of depth-first search.

4. Implement an algorithm to find the shortest path from the source to the destination, if it exists, using the queue-based implementation of breadth-first search.

You may use the implementations of stacks and queues provided by Java or create your own. Compare the running times of your algorithms for different maze sizes. As usual, write a report describing your results.