For my implementation, I used an adjacency matrix to represent edge connections between nodes. I accomplished this by using a 2d ArrayList of ArrayList<String>, where the first index was the parent node, and the second index was the child node. If the index had an empty ArrayList of Strings, then there were no edges between the parent and child nodes. If there was a non-empty ArrayList of Strings at the index, there were label(s) of existing edges between the parent and the child nodes. While this is functionally a 3d ArrayList of Strings, it is easier to imagine it as a 2d ArrayList of ArrayList<String>.

The advantage of an adjacency matrix is its efficiency in acquiring the edge labels, which is constant time, even when the matrix is dense.

The advantage of a collection of edges is the ability to store all the edges directly, so it's simple to implement and memory-efficient for sparse graphs.

The advantage of an adjacency list only stores relevant connections to the nodes, so it is space efficient for sparse graphs and allows quick neighbor lookup.

I chose to use an adjacency matrix because it would be more efficient to acquire edges for dense graphs. The constant access time of labels is extremely optimal for situations where the user constantly accesses the ArrayList of labels.