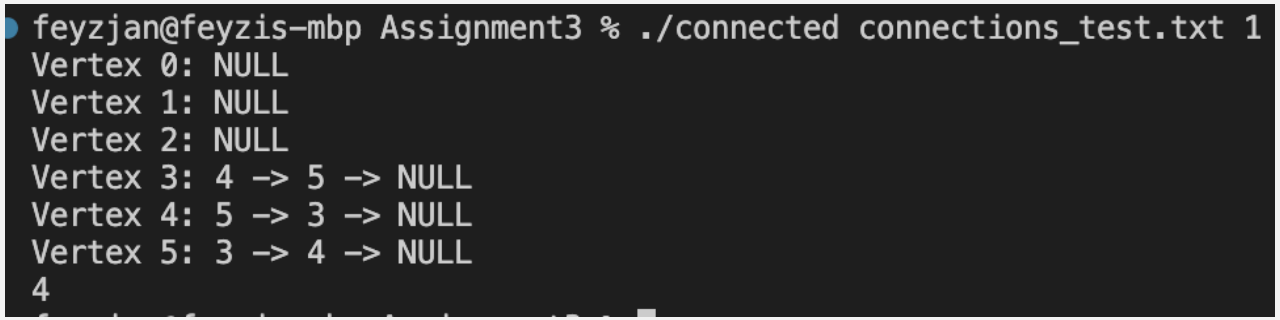
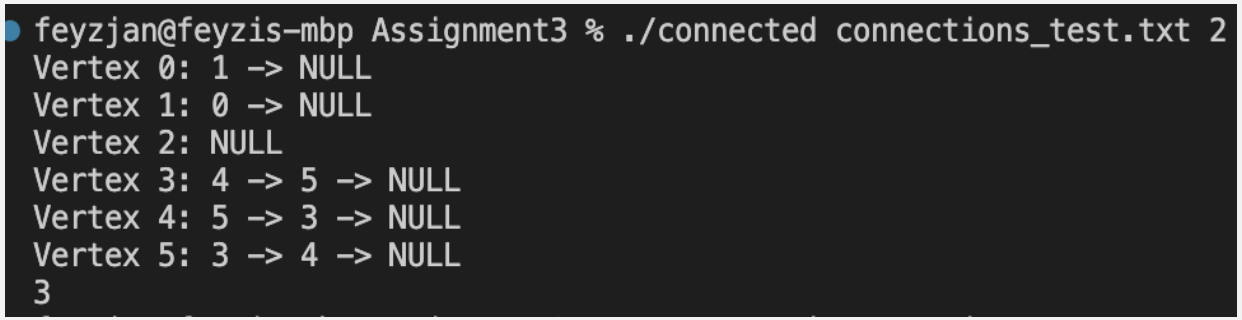
FEYZI CAN ESER – BRIEF EXPLANATION OF PROGRAM

- Deliverable: Program to output # connected components in a graph read from a text file while only considering edges of type 1 or 2
- I implemented all the methods suggested one by one, fixing implementation errors by testing via the code in main.c
- **Program Structure:**
  - First, read the file and store the graph in adj list format by adding all edges.
    - Command line arguments are read and fscanf is used to read file
  - Then run DFS and print the connected component counter
  - Finally, free all memory and exit
- **Key Elements Used:**
  - **Malloc**: Dynamic memory allocation used when creating new nodes, the graph, and adj. lists, which are freed at the end
  - **Node Struct:** Represents an element in the adjacency list and links to another element in the adj. list
  - **Graph Struct:** Holds an array of adjacency lists and the number of vertices
  - **While and for loops:** Used to read all edges from the file, initialize our visited and adj. lists, and to loop through nodes in DFS
  - **If statements:** Check the edge weight before adding the edge so that we conform to our mode specification, and check that a node has not been visited before running DFS on it.

WHY IT WORKS

- **Memory leaks:** Valgrind checked for memory leaks and passed Gradescope.
- **Edge cases:** Manual override to handle edge cases of graphs with 0 or 1 vertex.
- **Test Cases:** Successfully outputted 4 and 3 for the given test file under modes 1 and 2.
- **Correct Graph Storage:** Added a print_graph() function to print the graph and verify that it has been properly read and created





Graph created under mode 1 (left) and mode (2) right, the final line is the number of connected components.

Note: The print_graph function is commented out for the submission

- **Counting Connected Components:** The DFS function is a standard recursive implementation. The connected component counter is incremented by one whenever we explore all edges reachable from a node. The graph is an undirected graph, so if we can reach any node I from another node j, then there must be a path from node j to i, hence they are in a connected component.
  - Lone vertices with no edges are successfully counted as connected components