Project Report: Graphs

Shizza Fatima Shafqat

April 25, 2025

Individual Contribution

This project was completed individually. I was responsible for implementing the entire graph structure using an adjacency list and writing all the functions and ensured that they worked correctly. I also wrote all the inline comments, comment headers, student test cases and the this report. (one woman* army)

Reflection

I found it really interesting how something that feels so easy and intuitive when you're drawing or working with graphs by hand becomes more complex in code. Physically, adding an edge is as simple as drawing a line—but in code, you need to represent the graph as data structures like lists, and then locate the right vertices before you can add the edge. This process of breaking the graph into lists and carefully managing those structures made me realize how different (and detailed) graph operations can be when you're programming them. Overall, I found this project enjoyable and relatively approachable, especially since we had already implemented BFS and DFS in class. Those earlier experiences made it easier to understand the traversal logic and adapt it for this project. It was also very interesting to see how the book implemented the DFS and BFS with colors while we used time stamps.

Challenges Encountered

That said, I encountered a few challenges:

- **Implementing `removeEdge`:** Initially, I had some difficulty designing the logic to safely remove an edge from a vertex's neighbor list. After some trial and error, I implemented a simple solution using a loop and manual index tracking.
- **Calling `readFromSTDIN()`:** Since `readFromSTDIN()` is a non-static member function, I couldn't call it directly without an instance of the `Graph` class. However, I didn't know how many vertices the graph would have until after reading the input. To work around this, I created a dummy instance:

```
Graph A(0);
Graph g = A.readFromSTDIN();
```

This allowed me to call the method and construct the graph from input without modifying the header file. However, My function kept reading in random vertices like 3475 causing problems when I used to test it. Now I commented out the test cases from the studentTests.cpp

Known Issues

There are no known issues with my implementation. All functions have been tested and behave as expected for valid input. stdin implementation is commented out.

Further Considerations

- **Hardest Part:** Debugging input from `STDIN` and figuring out how to call `readFromSTDIN()` without knowing the graph size ahead of time. Implementing `removeEdge` was also somewhat tricky.
- **Easiest Part:** Writing BFS and DFS felt intuitive thanks to our previous classwork. Helper functions like `vertexIn` and `edgeIn` were also quick to implement.
- **What I Learned About Graphs:** I learned how graph traversal algorithms can be adapted in different ways—for example, using timestamps instead of colors for DFS. I also saw how to gather useful metadata like parent, distance, discovery/finish times, and topological order. (I also talked about what I learnt in the reflection part)
- **What I Learned About C++:** I got more comfortable with features like `vector` and `queue`as well as language-specific tools like unions and `static_cast`. Each CS 271 project introduces new language features, and I look forward to applying them in future assignments. I also learned how to implement graph structures using lists in C++, which helped me understand how data structures are represented and manipulated in code.
- **Adapting to Weighted Graphs:** To support weighted graphs, I would modify the adjacency list to store pairs in the form of (neighbor, weight) instead of just neighbors. Since BFS does not account for edge weights and is intended for unweighted graphs, I would replace it with Dijkstra's algorithm for shortest path calculations. DFS, on the other hand, would generally remain the same unless the traversal logic is changed to prioritize edge weights.