Project Report: Sorting Algorithms and Graph Implementation

Hasan Can İstekli

May 25, 2025

1 Introduction

In this project, I implemented multiple sorting algorithms and a graph class using the adjacency matrix structure. These implementations were tested in an example use case: a greedy graph coloring algorithm. The project is divided into three main parts:

- Implementation of sorting algorithms: `MyInsertSort`, `MySelectSort`, and `MyQuickSort`.
- Implementation of a graph class: `MatrixGraph`, which uses the adjacency matrix structure.
- Integration and testing of the implementations with the provided greedy graph coloring algorithm.

Additionally, bonus objectives such as implementing all methods of the `AdjacencyVect` class and automated unit testing were completed.

2 Environment

The project was developed and tested in the following environment:

- Operating System: Ubuntu 22.04 (via Docker container).
- Java Version: OpenJDK 11.
- Build Tool: `make`.
- IDE: Visual Studio Code.

3 Complexity Analysis

3.1 Sorting Algorithms

- `MyInsertSort`: $O(n^2)$ in the worst case, where n is the size of the array.
- `MySelectSort`: $O(n^2)$ in the worst case.
- `MyQuickSort`: $O(n \log n)$ on average, $O(n^2)$ in the worst case. The implementation uses random pivot selection to reduce the likelihood of the worst case.

3.2 Graph Implementation

- `MatrixGraph.reset`: $O(n^2)$, where n is the number of vertices.
- `MatrixGraph.setEdge` and `MatrixGraph.getEdge`: $O(1)$.
- `MatrixGraph.getNeighbors`: $O(n)$.

3.3 Greedy Graph Coloring Algorithm

The greedy graph coloring algorithm has a complexity of $O(n^2)$, where n is the number of vertices. Sorting the vertices by degree contributes $O(n \log n)$ to the complexity.

4 Testing

The project was tested using the `ProjectTests` class, which includes automated unit tests for all implemented classes. The following scenarios were tested:

- Correctness of sorting algorithms: Verified by sorting arrays and comparing the output with expected results.
- Correctness of graph implementation: Verified by constructing a graph from an input file and testing edge addition, neighbor queries, and adjacency matrix representation.
- Integration with the graph coloring algorithm: Verified by solving the graph coloring problem and ensuring the solution satisfies the constraints.

4.1 Automated Unit Testing

Automated unit tests were implemented in the `ProjectTests` class. These tests cover:

- `AdjacencyVect`: Adding, removing, and querying vertices.
- `MatrixGraph`: Edge addition, neighbor queries, and adjacency matrix correctness.
- Sorting algorithms: Correctness of `MyInsertSort`, `MySelectSort`, and `MyQuickSort`.

5 AI Usage

AI tools were used to:

- Assisted in structuring the initial implementation of sorting algorithms and graph classes.
- Provided insights for improving the efficiency and correctness of the `AdjacencyVect` class.
- Write unit tests for all implemented classes.
- Generate this report template and documentation structure.

6 Challenges

The main challenges faced during the project were:

- Ensuring compatibility with the provided graph coloring algorithm.
- Implementing the `AdjacencyVect` class to conform to the `Collection` interface while maintaining efficiency.
- Debugging edge cases in the sorting algorithms and graph implementation.
- Managing the complexity of integrating multiple components into a cohesive solution.

7 Conclusion

This project provided valuable experience in implementing and testing sorting algorithms and graph data structures. I learned how to integrate different components into a cohesive solution and gained a deeper understanding of algorithmic complexity and optimization. Additionally, I improved my skills in automated testing and debugging.