**CS 2302 Data Structures**

**Fall 2019**

**Lab Report #6**

Due: November 19, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab, I implemented different types of graphs and search algorithms in order to come up with a solution for a puzzle. The main purpose of this lab is to find six different solutions to the fox, chicken, and sack of grain problem. The different types of graphs were used and implemented with two search algorithms, breadth-first search and depth-first search. The search algorithms were then used to find valid paths to solve the problem.

**Proposed Solution Design and Implementation**

**Part 1:** For part 1, I created insertion, deletion, and display functions for each type of graph. I made sure to cover all conditions for all types of graphs within these functions. I did this by checking if the graph was directed, or weighted and inserting and deleting accordingly based off of these features of the graph. The insertion, deletion, and display functions were given for the adjacency lists. For the adjacency matrix, I made the insertion to insert a 1 in the matrix if the graph was unweighted, or the actual value of the weight if the graph was weighted. For the deletion function I changed the value to -1. For insertion using an edge list, I added the source, destination, and weight to an object then appended it to the edge list. For the deletion, I deleted by popping the element that needs to be deleted from the list. The display functions were similar for all 3 lists, I simply iterated through the graphs and printed each of the attributes of the objects within the graphs. For the draw functions, the code was given, however I changed the iteration depending on the type of graph.

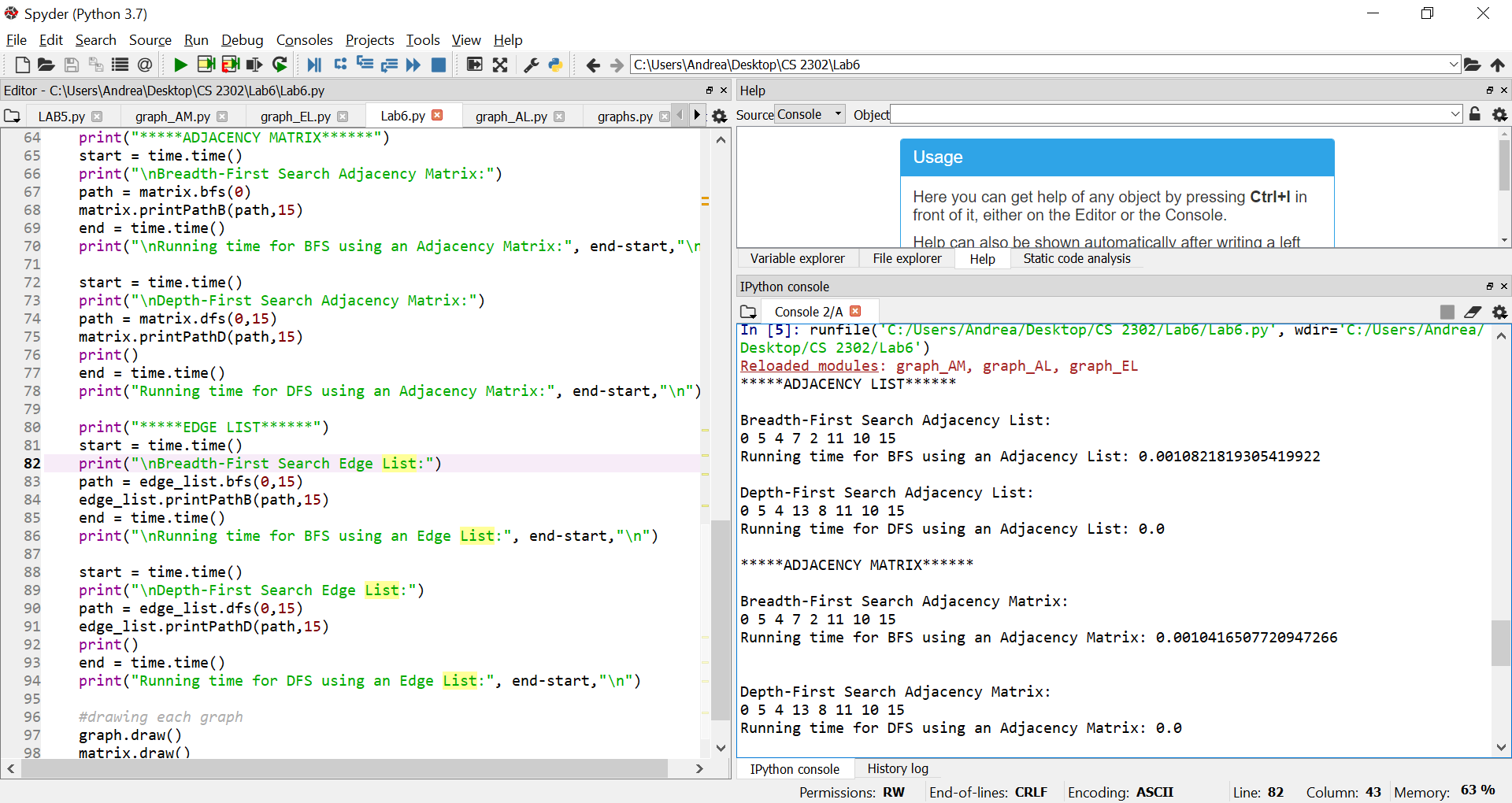
**Part 2:** For part 2, I solved the problem first by drawing it out and figuring out the correct solutions that goes by every rule. I then implemented each type of graph along with the search algorithms to search for the correct or shortest path. I did this by first finding all the legal states and adding them to each type of graph. I then created a separate breadth first search function for every type of graph and also a separate depth first search function for each type of graph.

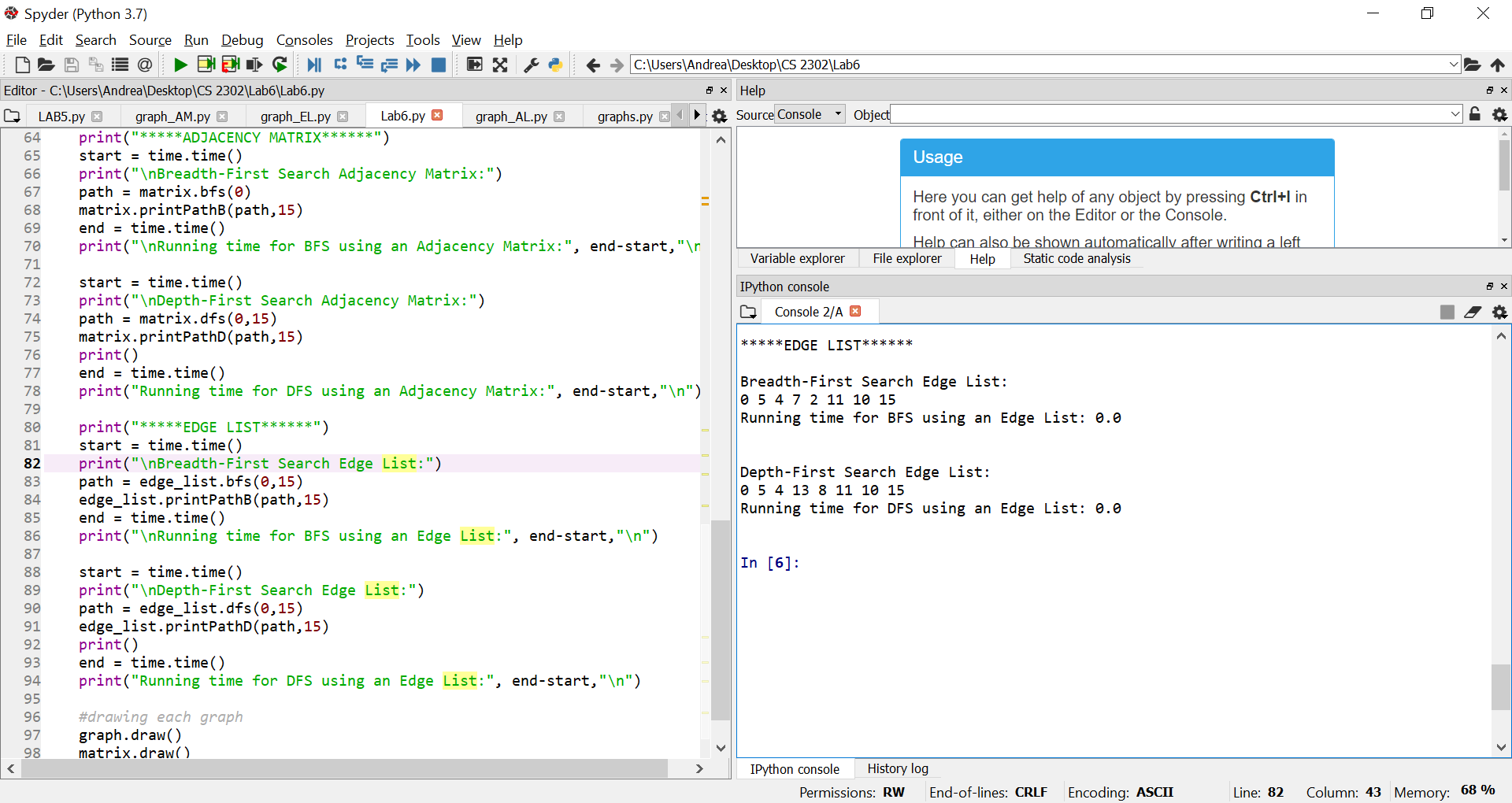
**Experimental Results**

To test my program, I timed the execution of each graph with the two different search algorithms. From the running times, I concluded that the Depth-First search algorithm was the most effective in terms of running time. The Breadth-First search was a bit slower however still efficient.

|  |  |  |
| --- | --- | --- |
| **Graph Type** | **Avg running time**  **Breadth-First Search** | **Avg running time**  **Depth-First Search** |
| Adjacency List | 0.00108 | 0.0 |
| Adjacency Matrix | 0.00104 | 0.0 |
| Edge List | 0.0 | 0.0 |

When it came to finding the shortest path, breadth first search and depth first search found different paths however they both consisted of legal paths. Both paths worked out in the riddle, and none of the transitions are illegal.



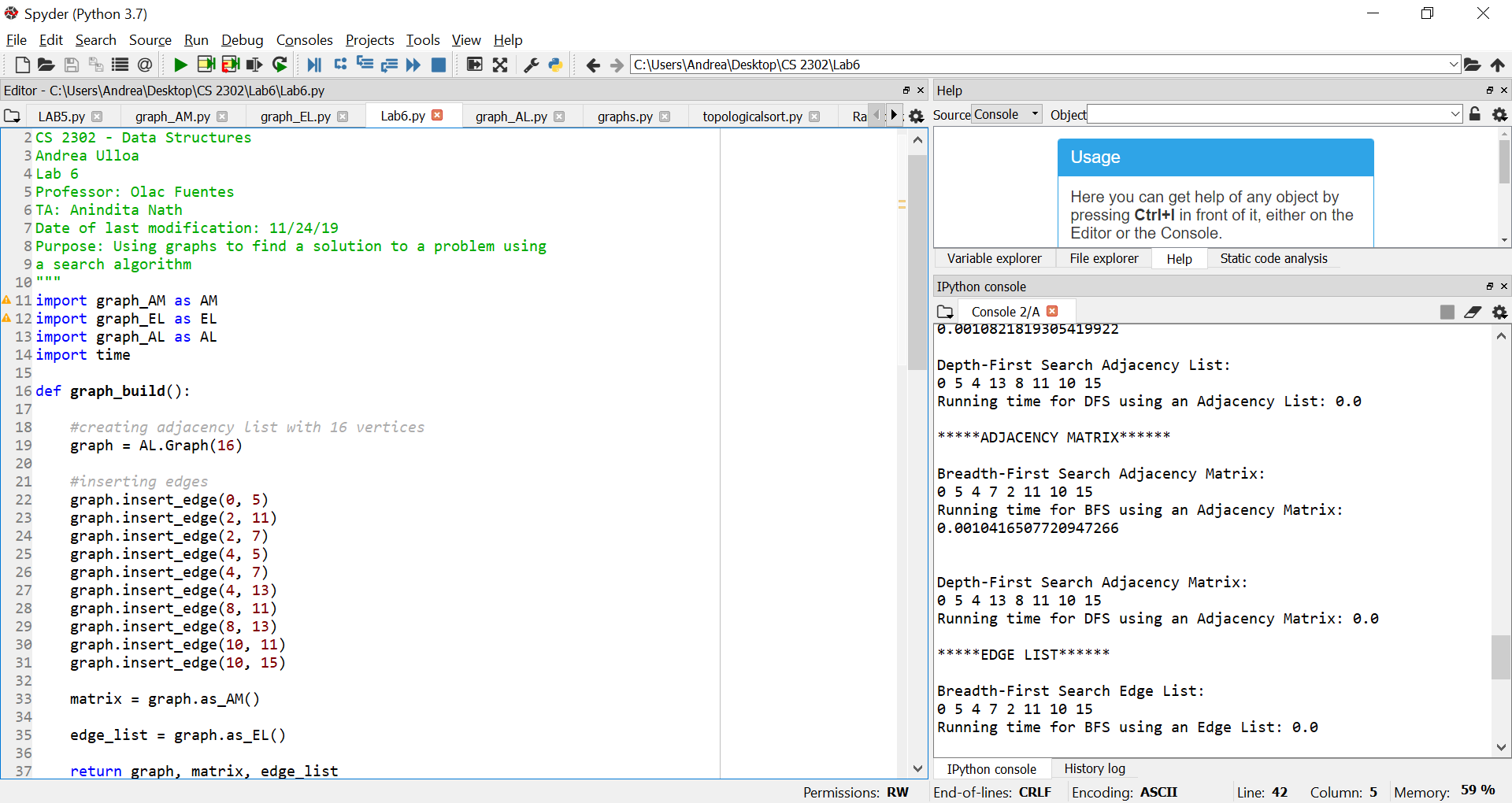


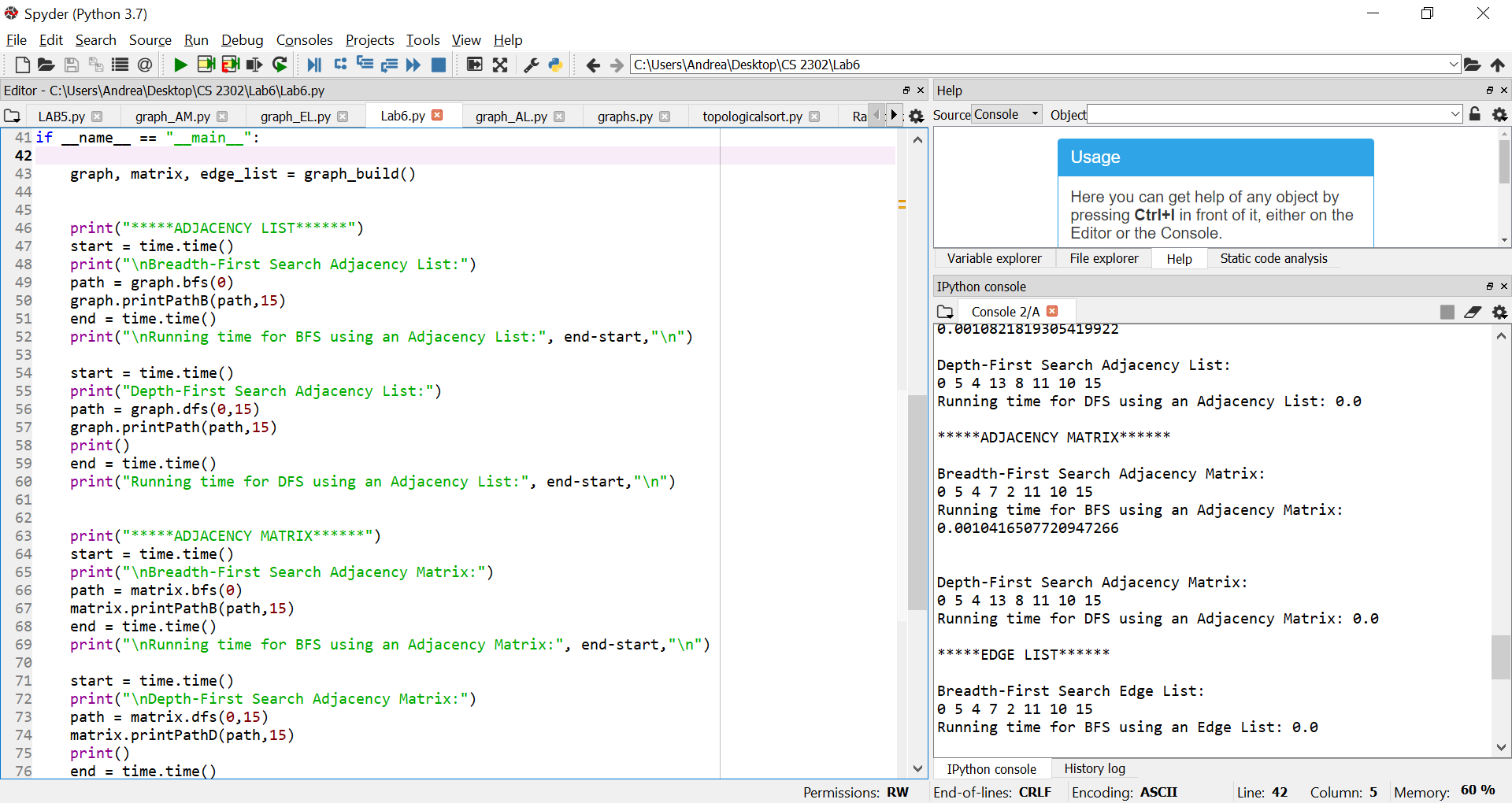
**Conclusion**

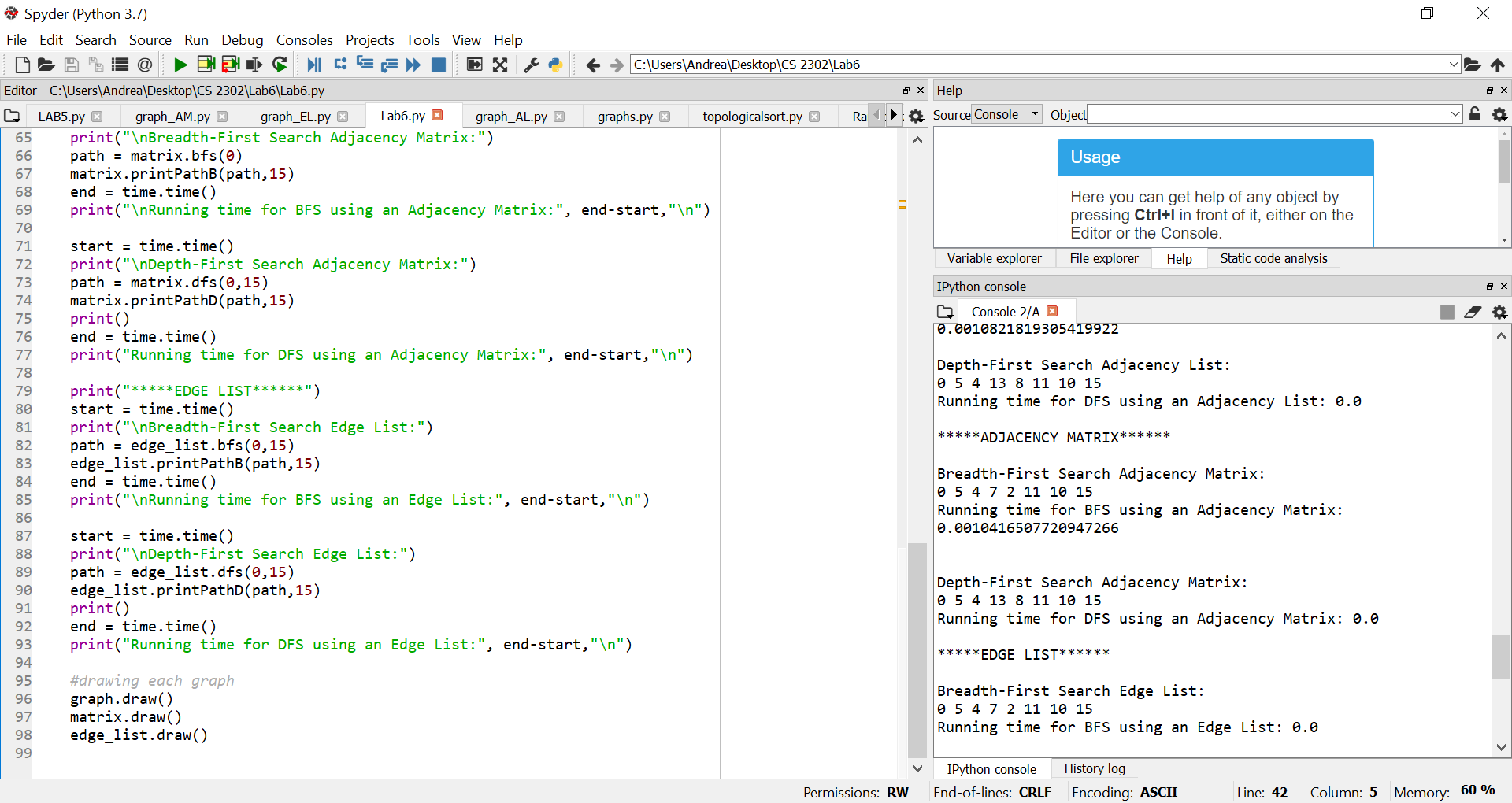
In this lab I learned how to use graphs in order to solve a riddle. I also learned how to use different search algorithms for each type of graph to solve the same problem. I learned how to modify the breadth first search and depth first search in order to apply to the different types of graph. I also learned about which search algorithm is better in terms of running time. I concluded that the depth-First search was the faster search algorithm, however breadth-first search was still pretty quick, and at times had the same running time as depth-first search.

**Appendix**

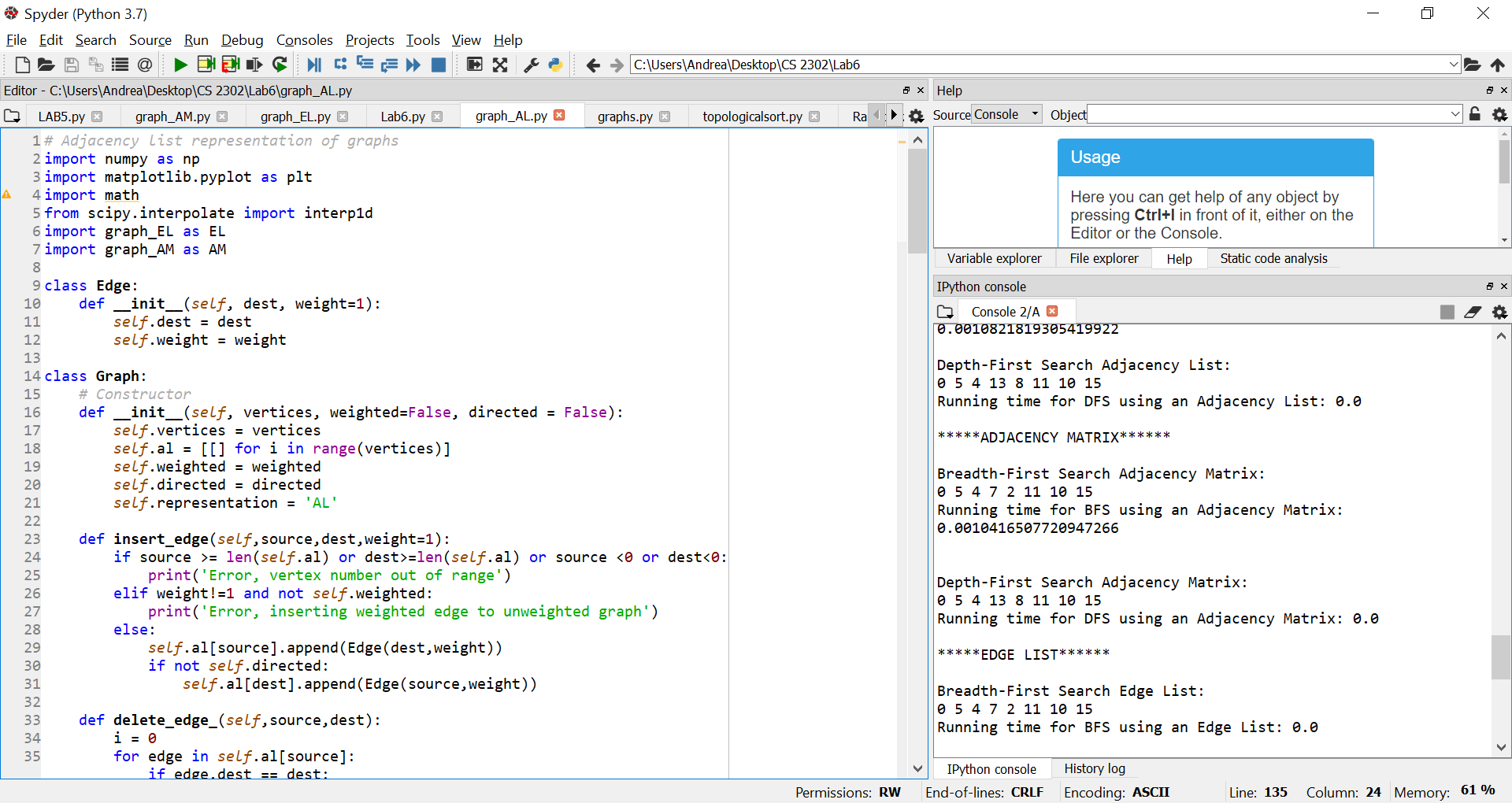
Lab6.py

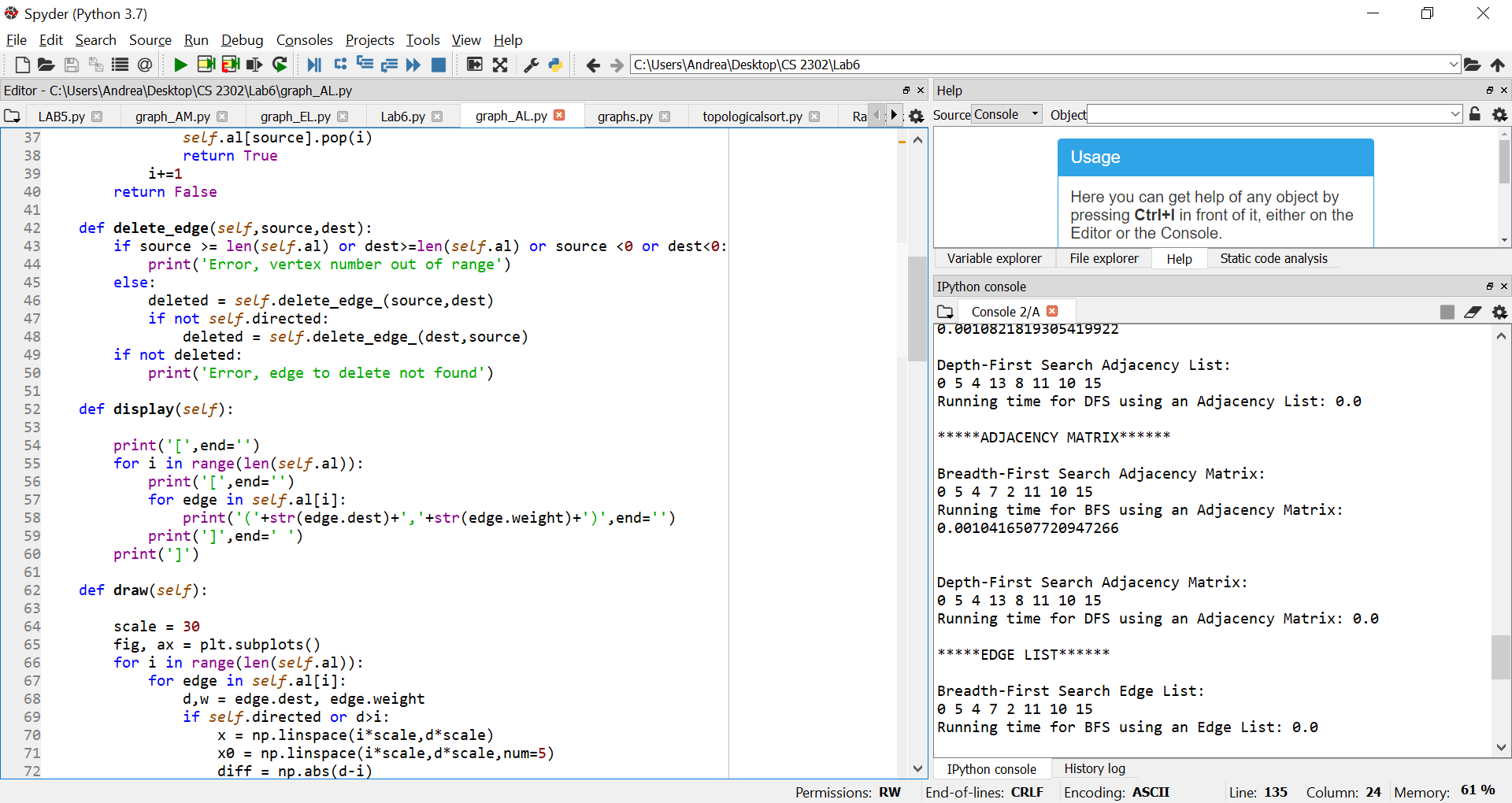


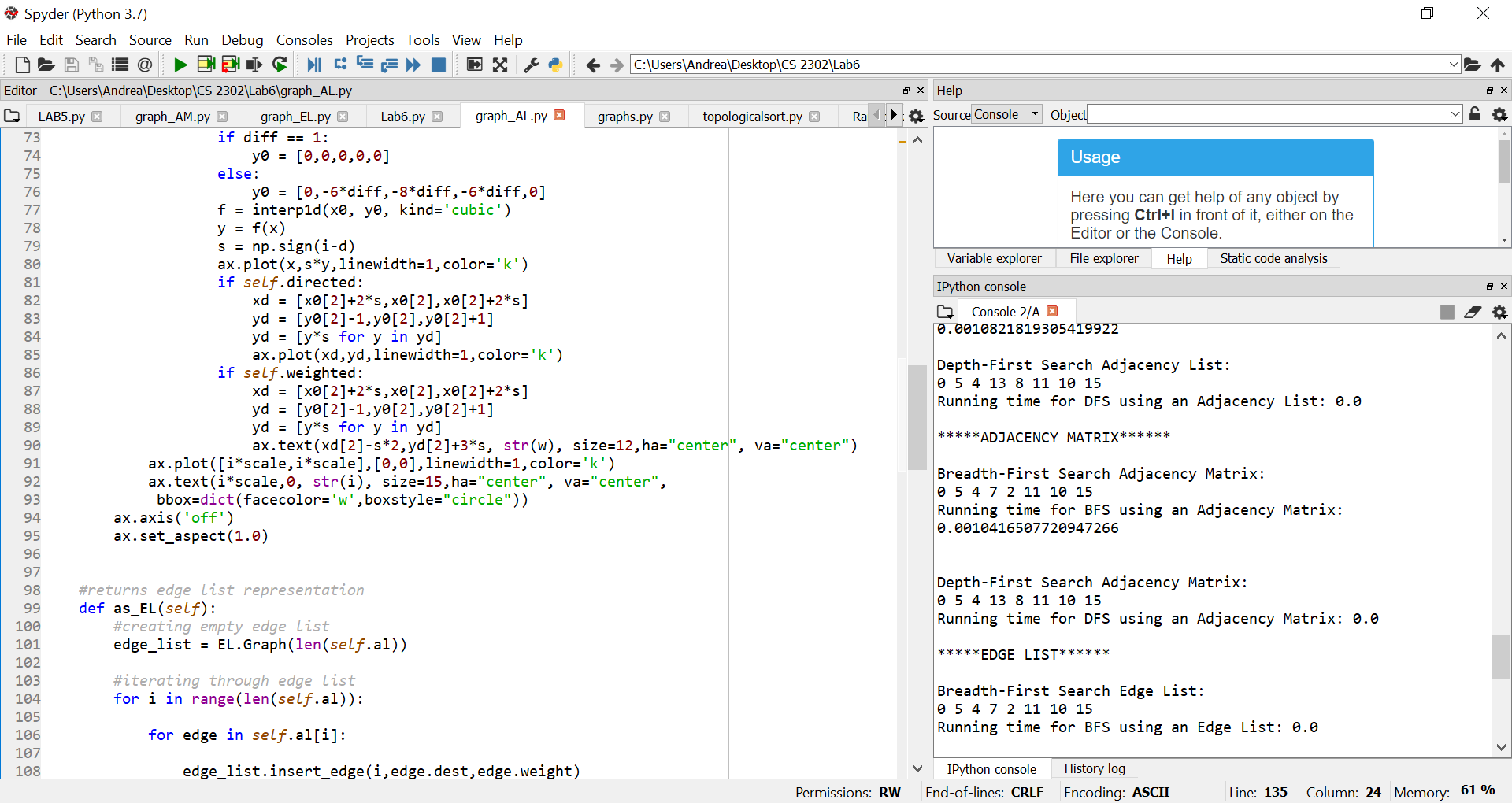


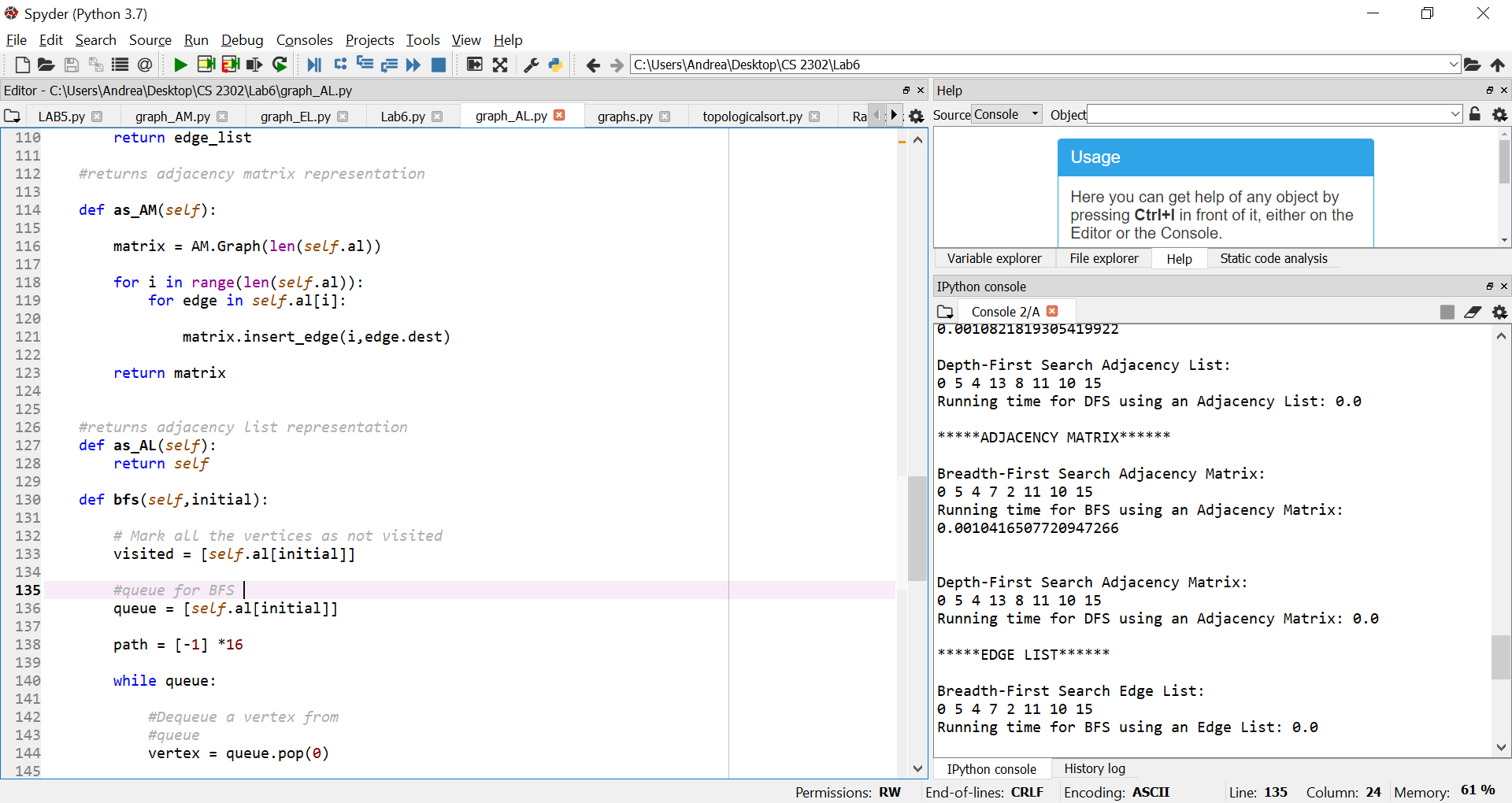


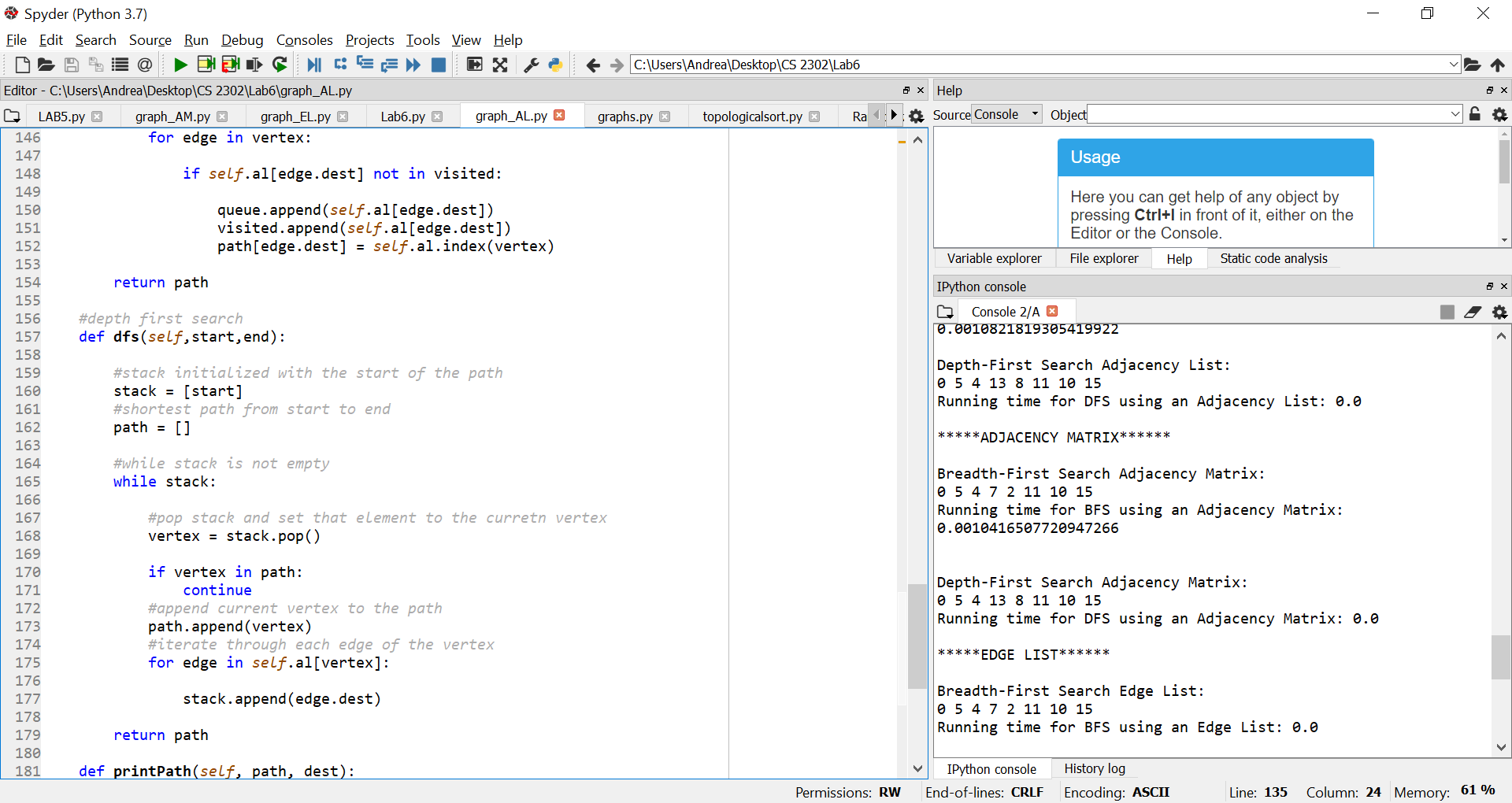
graph\_AL.py

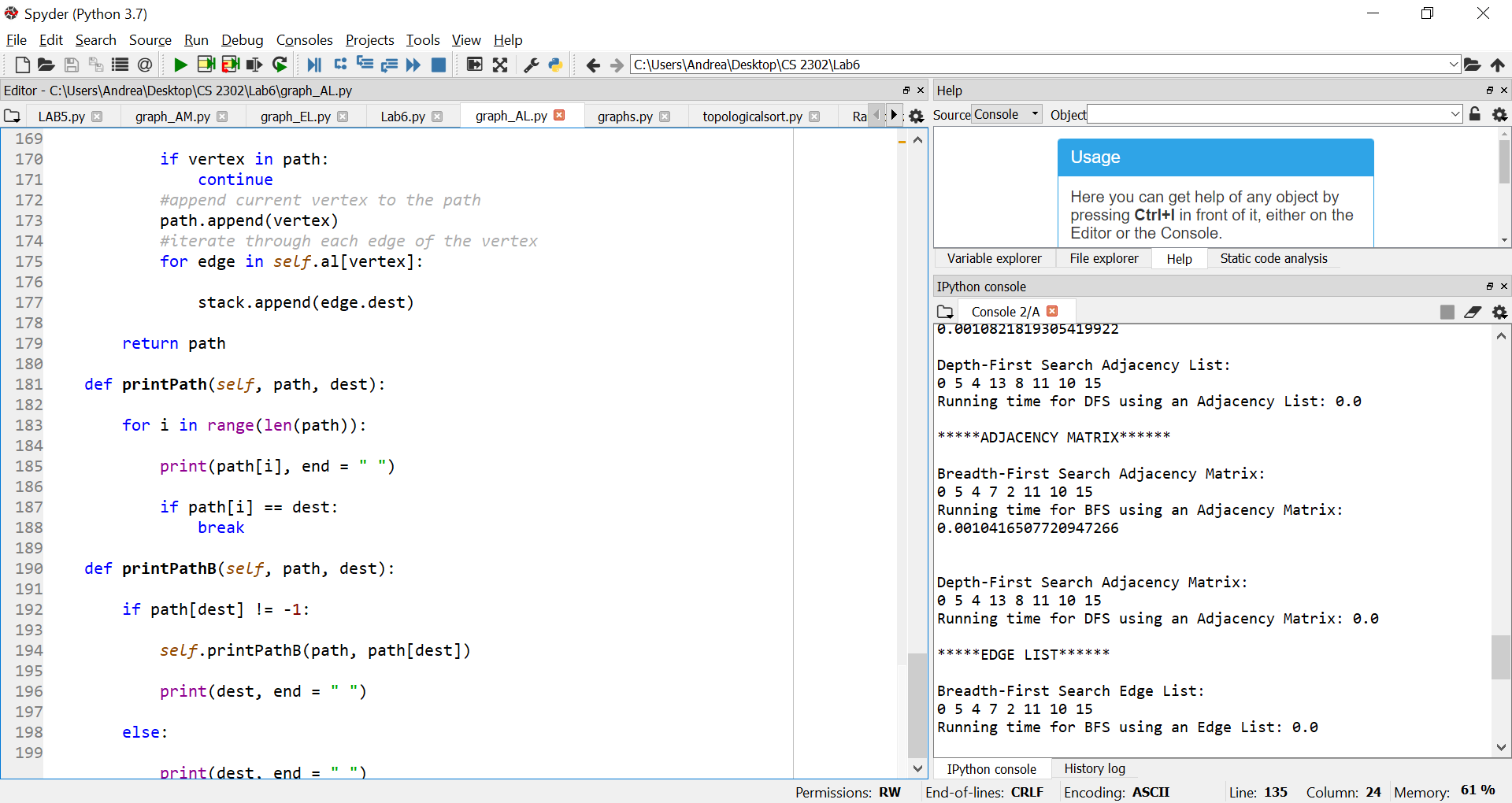




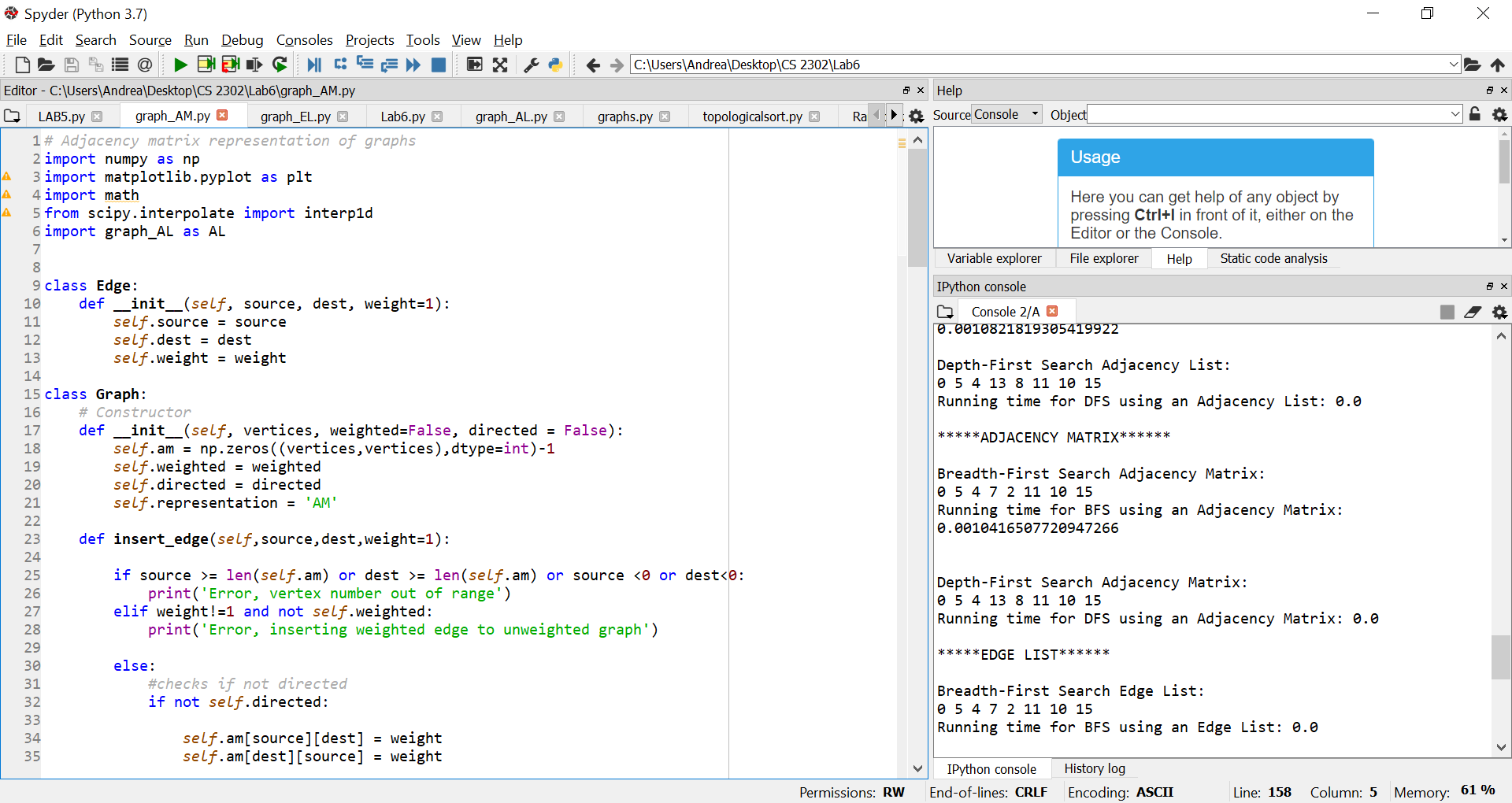


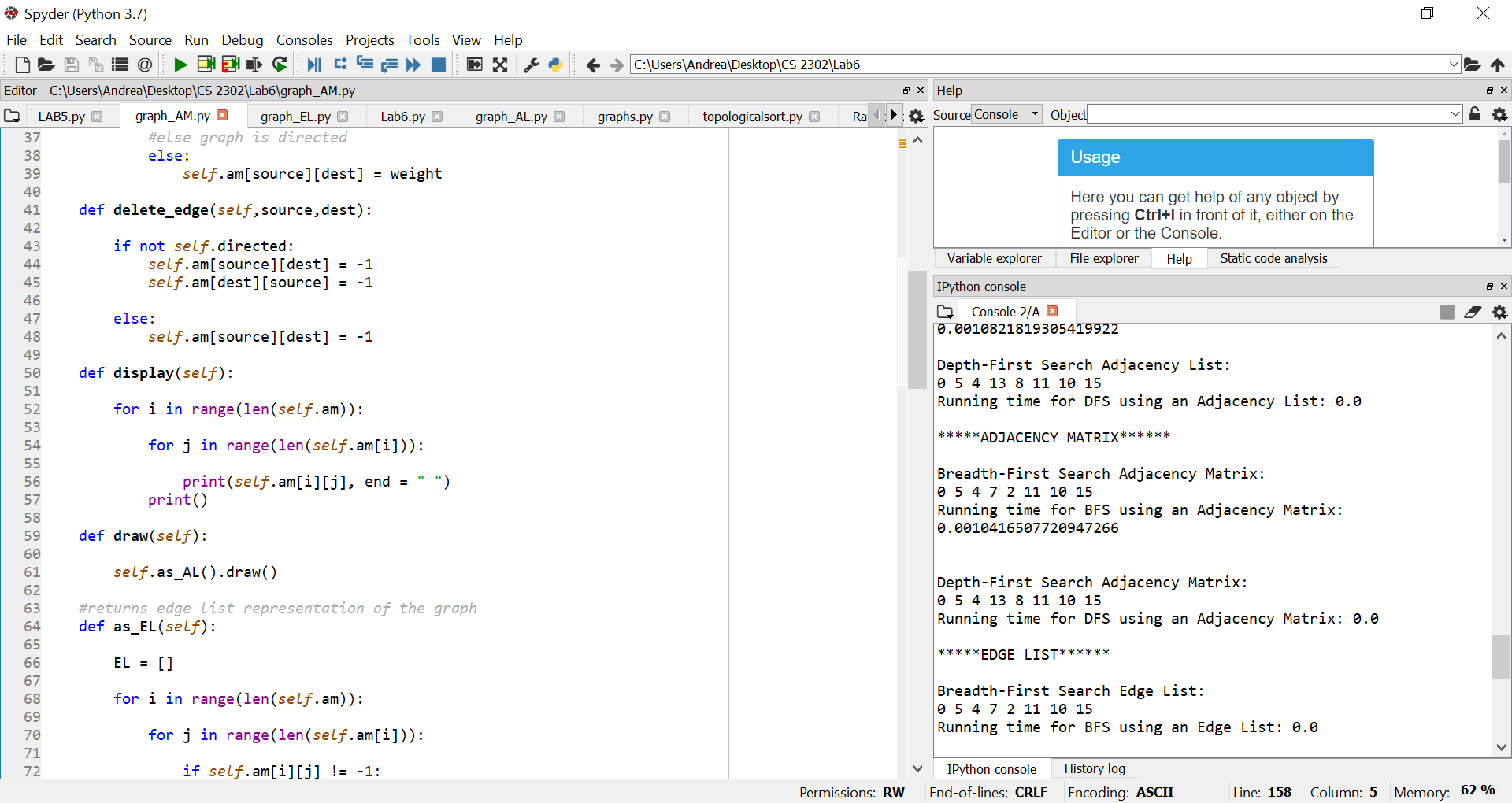


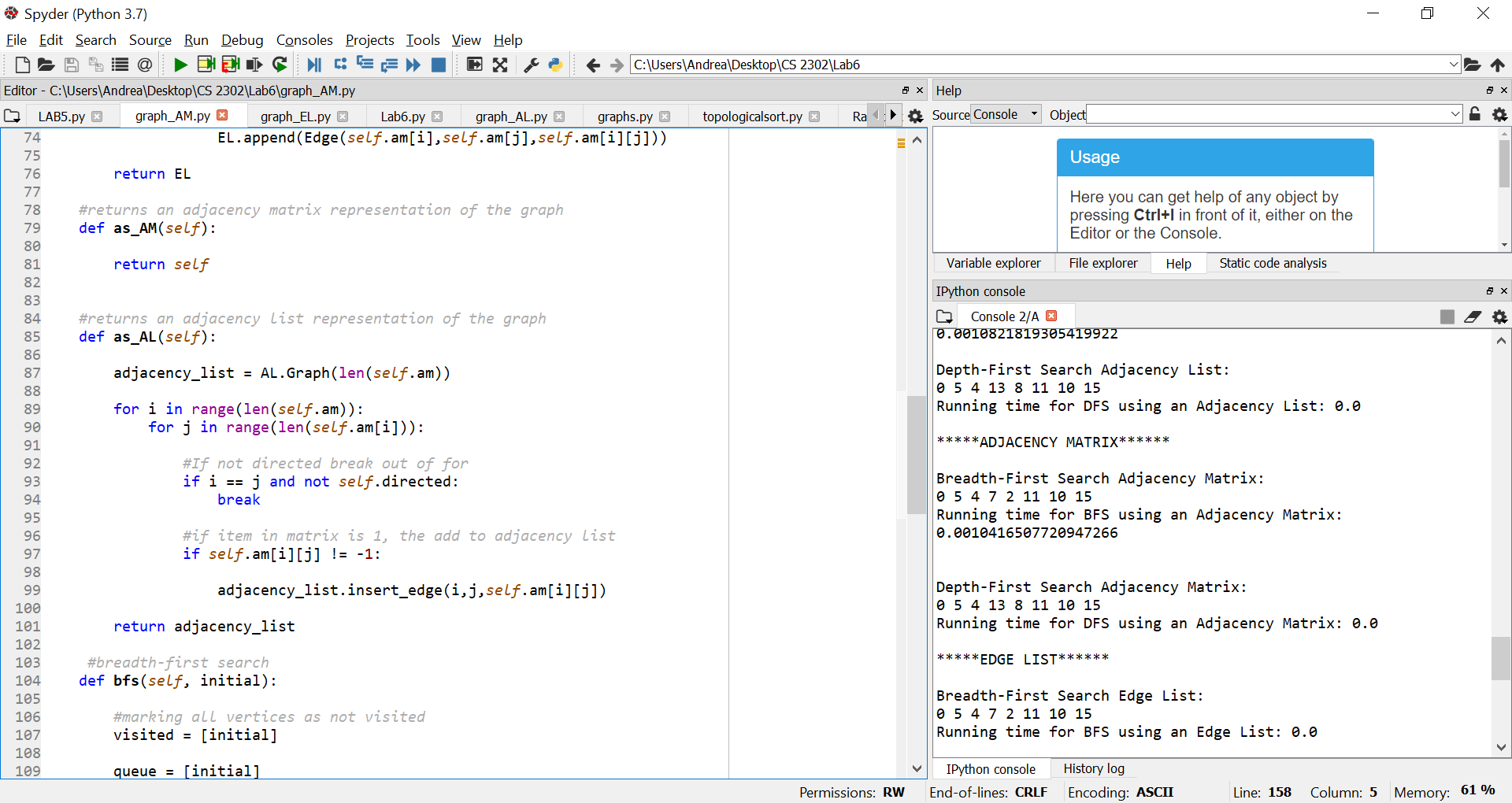


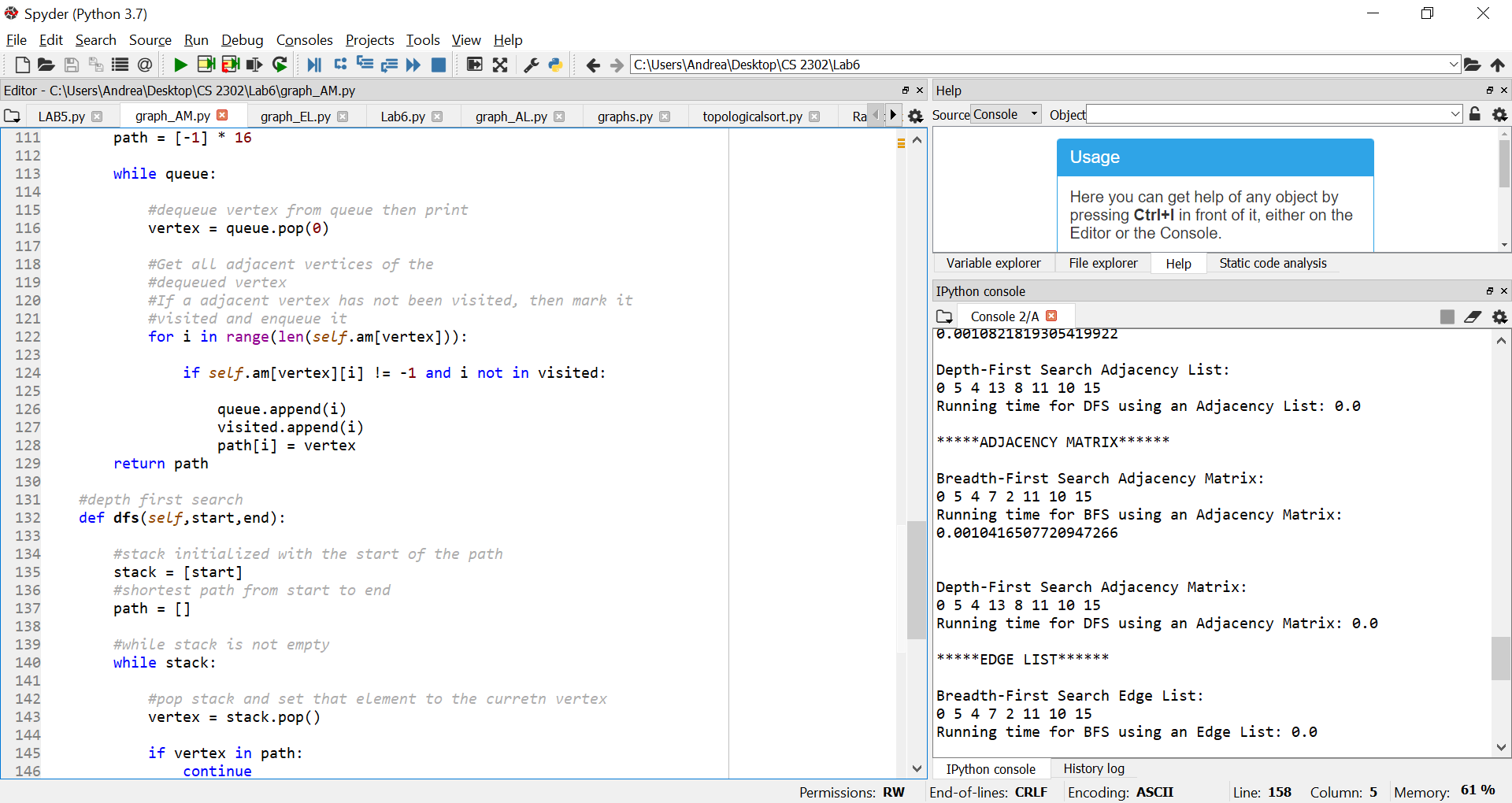


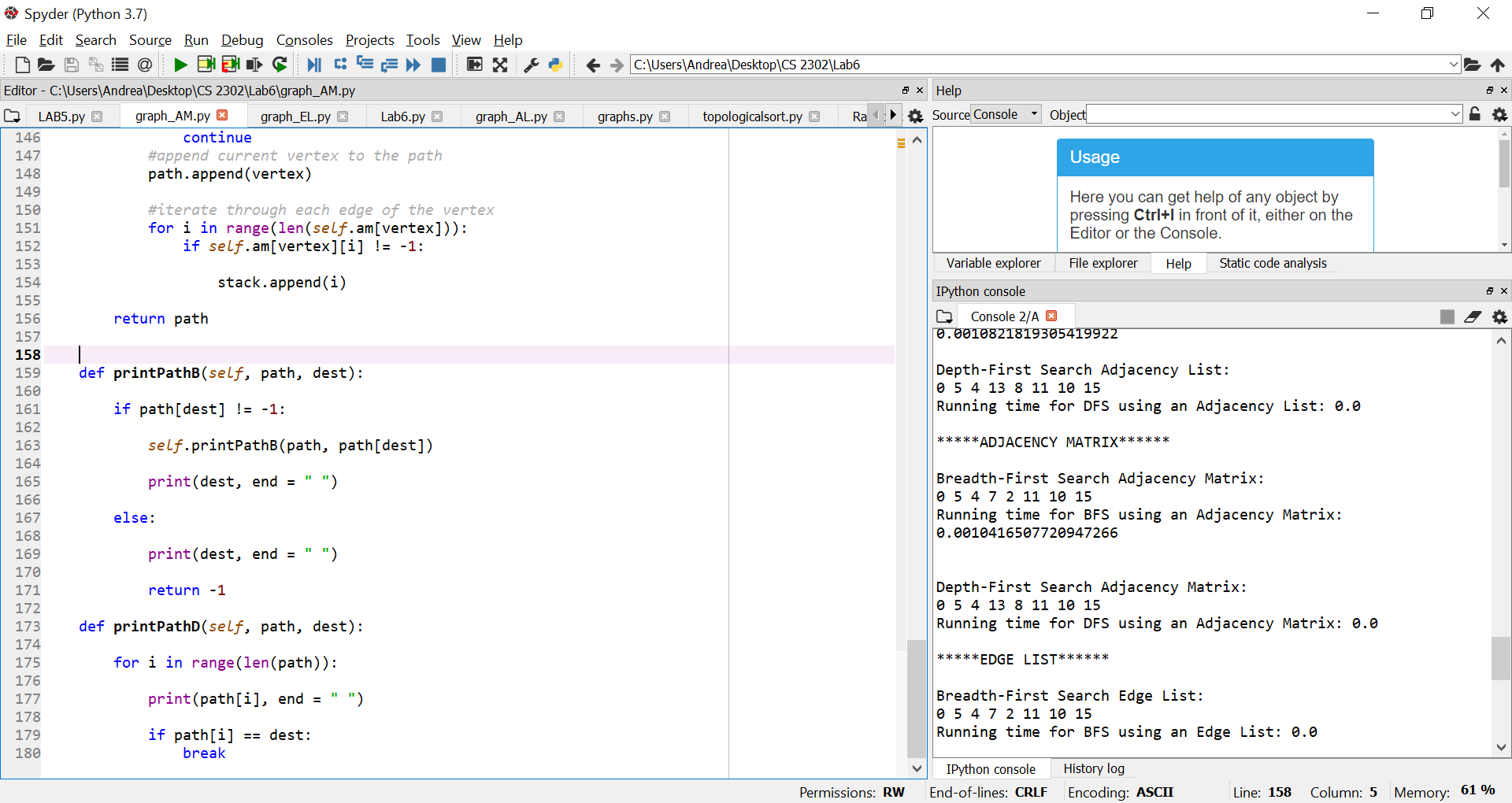
graph\_AM.py



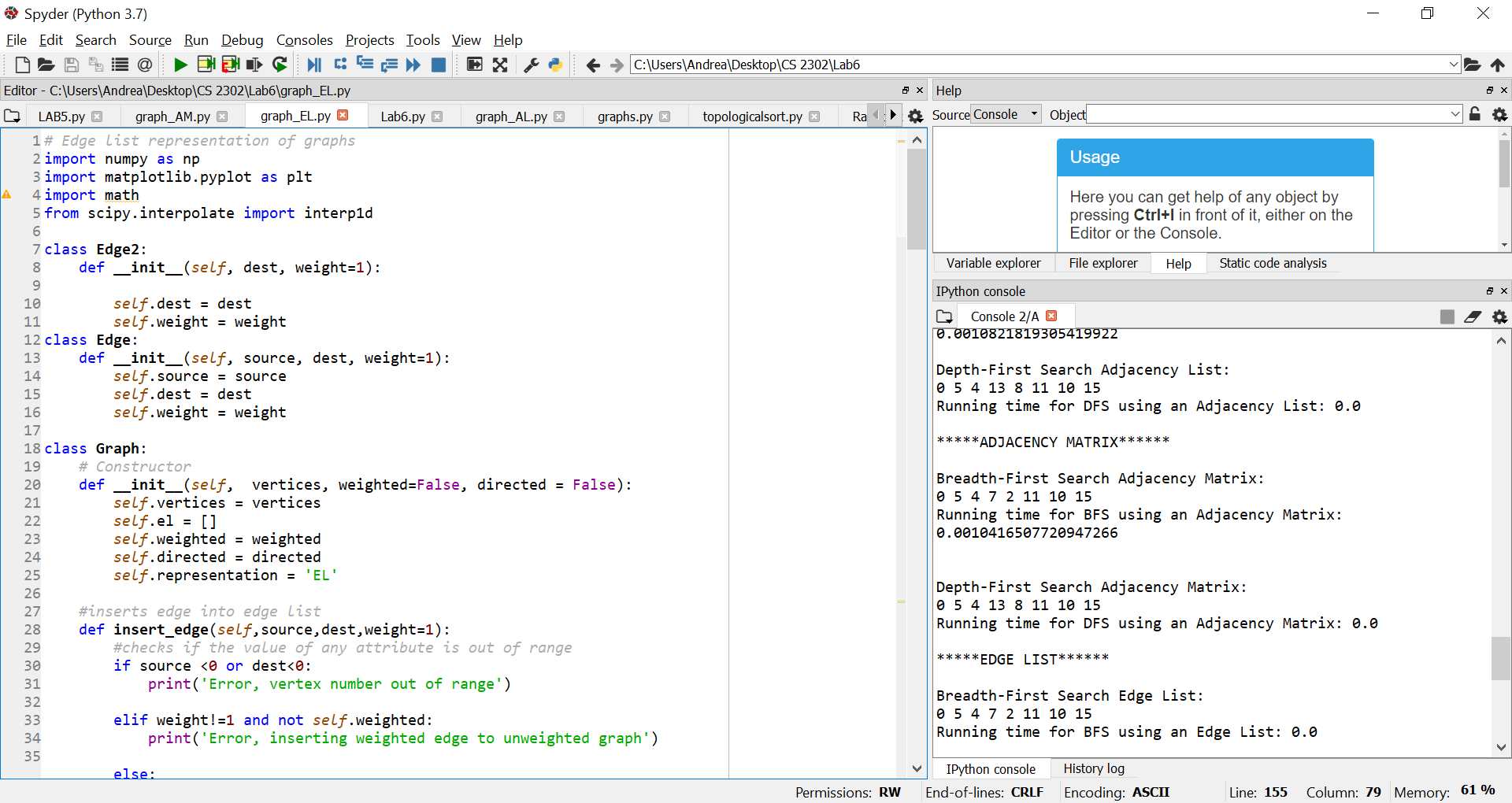


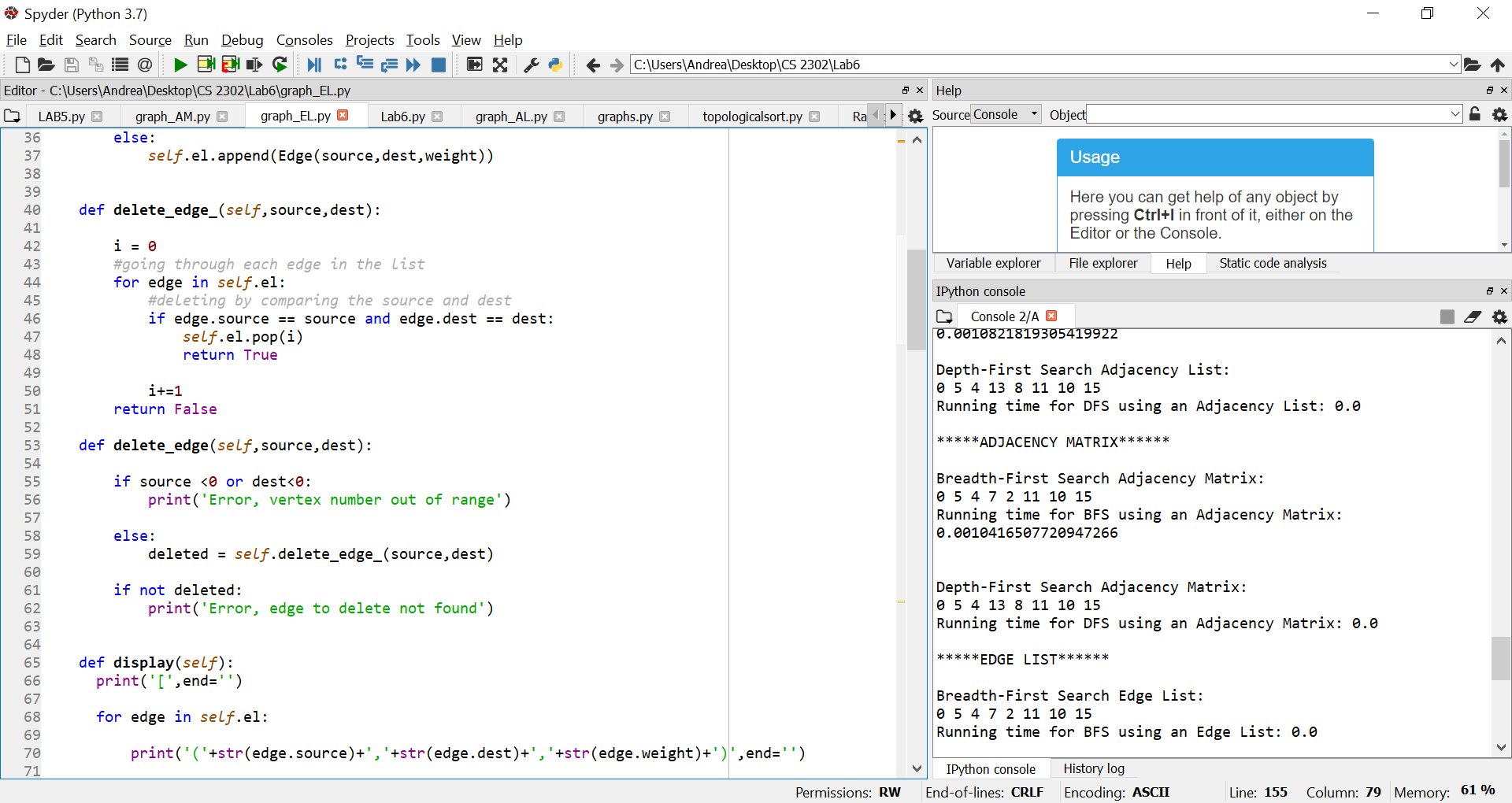


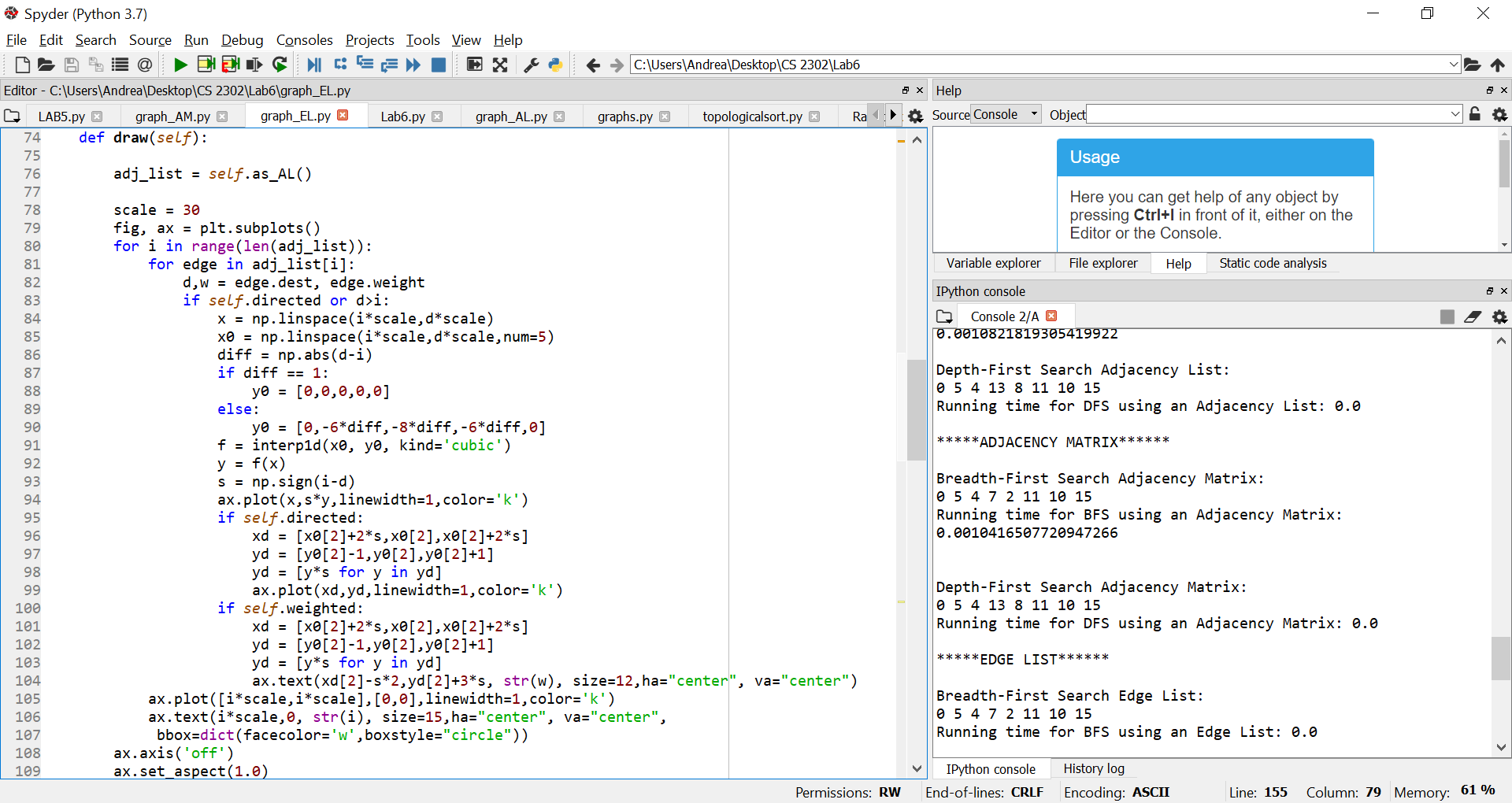


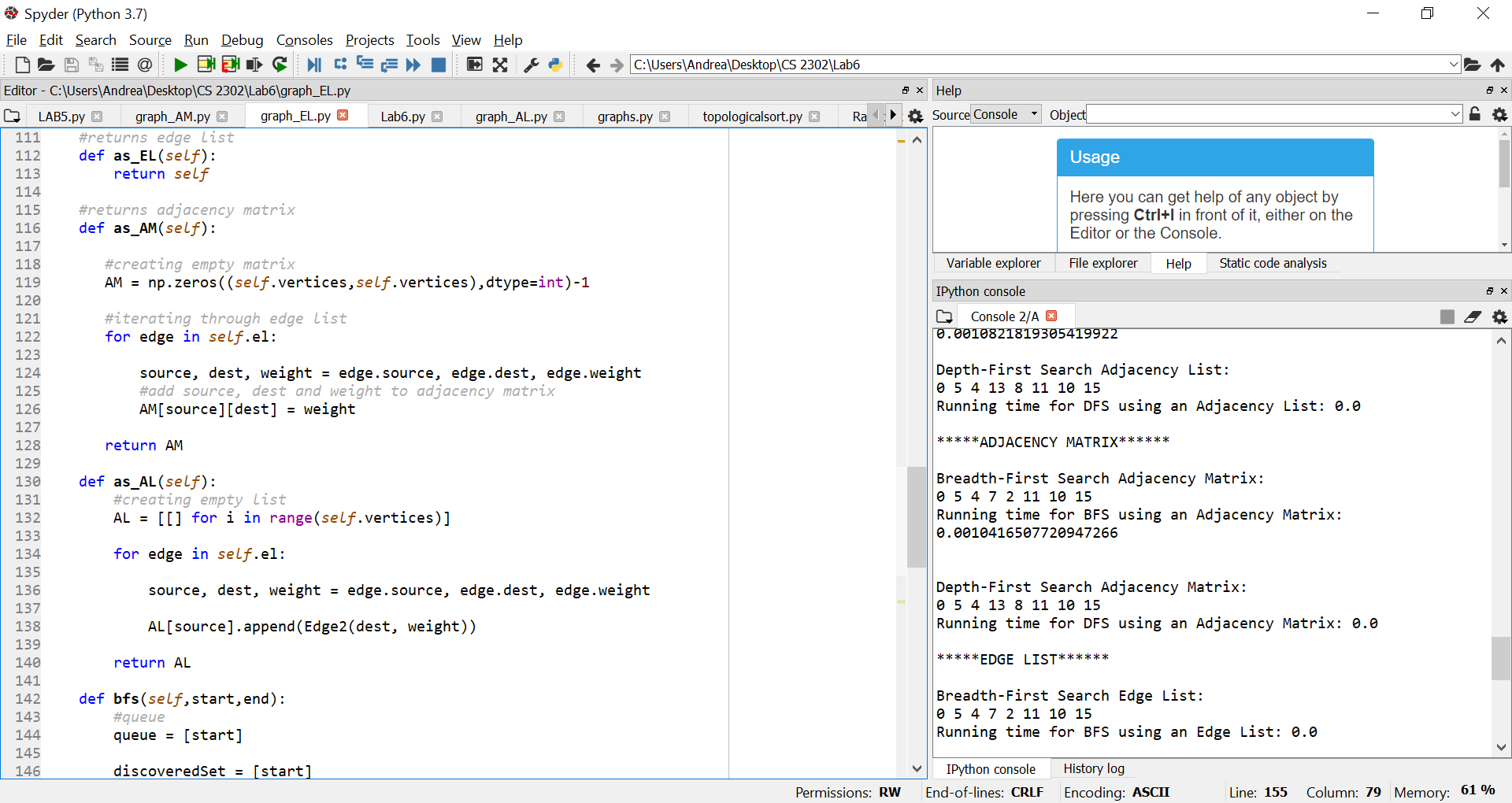


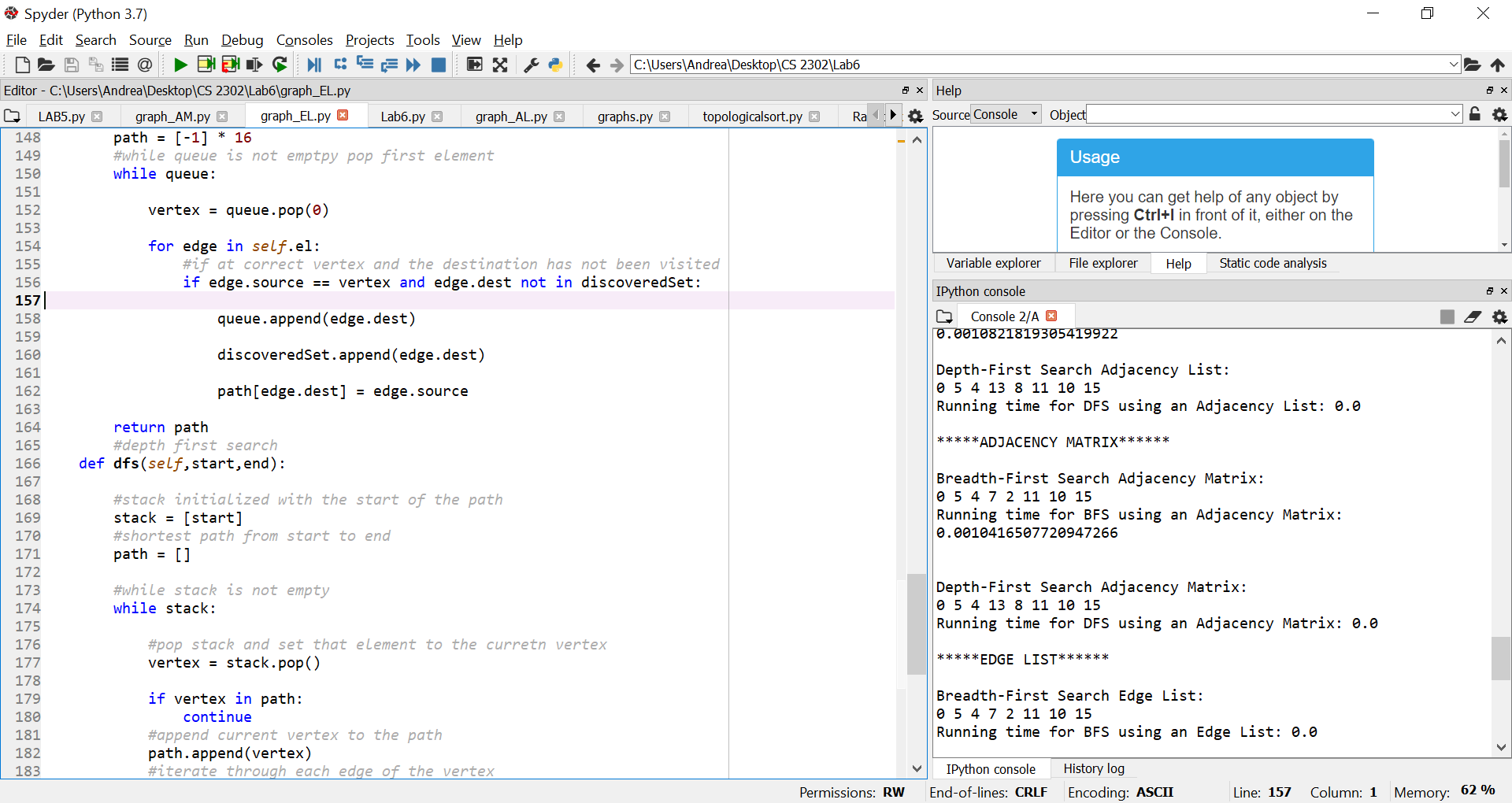
graph\_EL.py

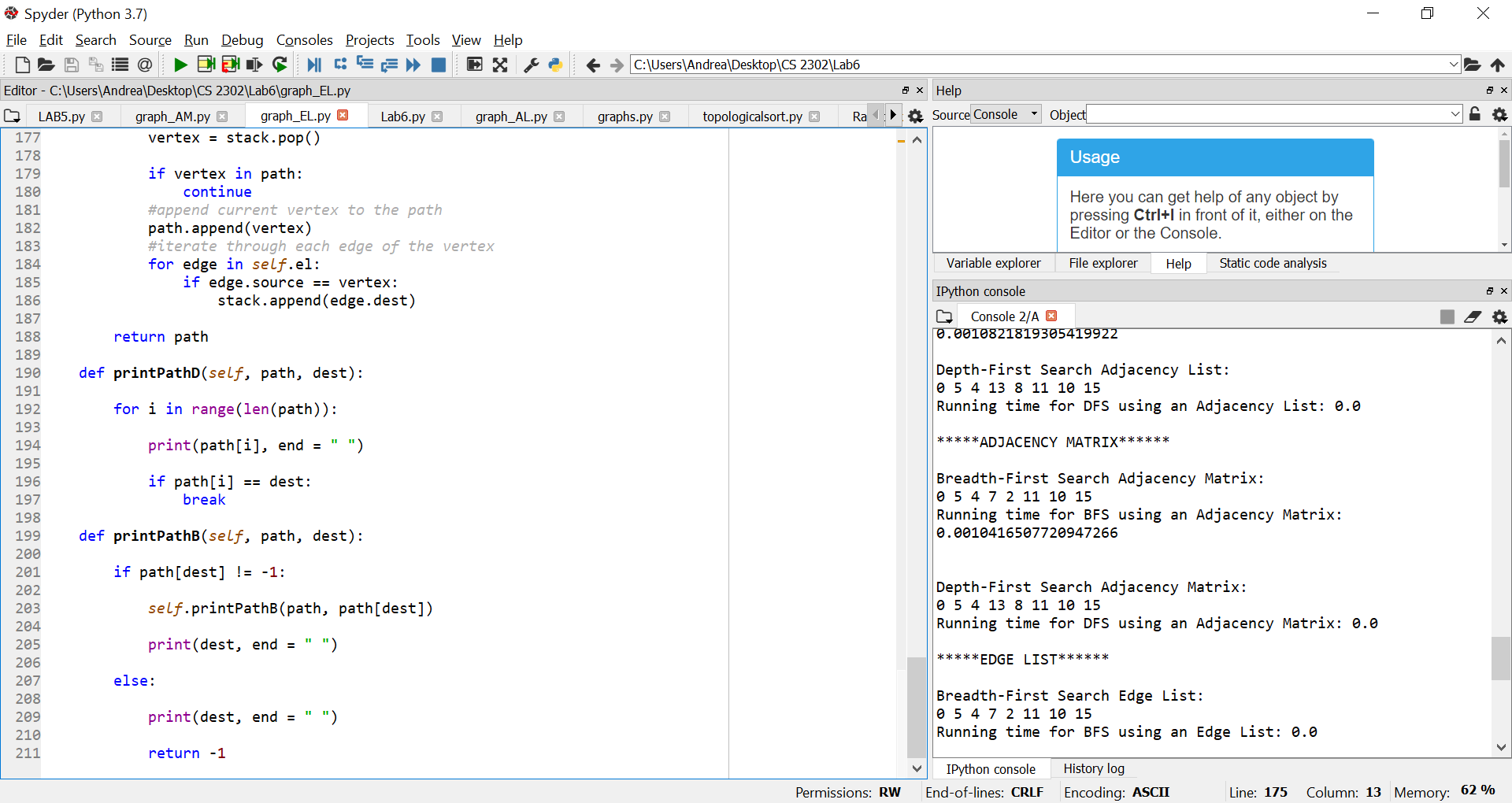












I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class