Homework 5: Greedy Algorithms

CS 4040/5040

Due: Wednesday, December 6th at 5:00 pm

This assignment will include programming work as well as a project report. Make sure you include all files in your submission. Answer all questions from the prompt in your report.

In this project we will be experimenting with different algorithms for finding the Minimum Spanning Tree of different graphs. You will need to implement one version of Prim's Algorithm, and two different versions of Kruskal's Algorithm.

1. First, write the code that will read in the graphs from the provided files. The format of the graph text files is: the first line contains the number of vertices, all other lines contain an edge in vertex1, vertex2, weight format. Vertex 1 will always be a smaller number than vertex 2, and all weights are integers. You do not need to build the graph as an actual series of node objects with connecting edges, representing them as a list of edges will be easier and more time efficient.
2. Implement Prim's Algorithm to find the Minimum Spanning Tree and print that MST to a file in the same format as the graph files that were read as input.
3. Implement Kruskal's Algorithm to find the Minimum Spanning Tree of a graph and print that MST to a file in the same format as the graph files that were read as input. Use an array representation of Disjoint Sets (without any optimizations) to track when your MST is complete.
4. Implement Kruskal's Algorithm to find the Minimum Spanning Tree of a graph and print that MST to a file in the same format as the graph files that were read as input. Use an array representation of Disjoint Sets that uses the Path Compression optimization to track when your MST is complete. When using the Path Compression Optimization, every time we "find" a vertex in our array (go to the index for that vertex and trace back to find it's root in the disjoint set), at the end we move that vertex to be pointing directly at it's root, so later finds will be faster. It is acceptable to have one function in your code for Kruskal's that has a parameter for Path Compression optimization if you prefer to do so.
5. I have included 5 large graph files to use for the experiments. Test each algorithm on each graph, and time it's performance. Your timing should JUST include the time it takes to find the MST of the graph, not the time to read in the graph from the file or print the answer to a file. Record the time taken for the different algorithms. You should not need to re-run or recompile your code for different experiments.
6. In your project report:
    a. Include instructions on using your makefile to compile and run your code.
    b. Write a brief description on how you implemented each MST algorithm.
    c. Show a graph and table of the results of our timing experiments.
    d. Discuss the results of the experiments. Do they match your expectations from the time complexity of these algorithms? Why or Why not? Which algorithm performed best? Did using Path Compression on the Disjoint Sets representation impact the performance of Kruskal's algorithm?

Program Requirements:

- Written in C++ and can compile and run on the school Linux machines
- Makefile is provided to compile and run the code. At the beginning of the project report, include instructions on how to use yor makefile to run and compile your code
- TAs should be able to unzip/uncompress your submission and be able use your makefile to compile and run your code.
- Code should be well documented and follow best practices for readability (good variable names, comments, broken into meaningful functions)
- Output to the screen should be well organized and labeled to make it clear which results are being presented
- Do NOT have your code create the graphs for the report in C++. Just have it output the data and use excel, matplotlib, or your favorite data visualization program to create the graphs and charts. Doing so in C++ is unnecessary for the scope of this class.
- No changes should have to be made to the code to re-run the experiments. We should not be changing variables in the code to see different results. Running the code should run all experiments.

Report Requirements:

- Report should be a PDF to ensure it can be opened on the grader's computer.
- Report should be well organized and formatted and be free of major grammatical errors.
- All graphs/charts should be well labeled and easy to read