CS 5720 Design and Analysis of Algorithms

Project #4

Submission requirements:

- A .zip file containing your source code. You may use any language you would like.
- A PDF (submitted separately to the Canvas assignment) containing each item below that is listed as a **Deliverable.** For each item contained in your PDF, clearly mark which deliverable it is associated with. Plots should be clearly labeled and have descriptive captions.

Implementing an MST algorithm. In class we discussed two algorithms (Prim's and Kruskal's) for computing the Minimum Spanning Tree of a weighted undirected graph. In the recent homework assignment, you studied the time complexities of these algorithms as a function of two things: first, the *representation* used for the graph, and second the *density* of the underlying graph. In this project, you will implement **one** of these algorithms (along with some auxiliary functions) and test your implementation on example graphs that are provided in the **graphs.zip** file in Canvas.

Graphs provided in graphs.zip. For this project, I have provided 30 graphs in the **graphs.zip** file. These graphs are divided into three types (called "type 1," "type 2," and "type 3"); there are 10 graphs of each type. Each CSV file in **graphs.zip** contains a **weight matrix** representation of an undirected graph. Specifically:

- If cell (i, j) contains the value -1, that means there is no edge between nodes i and j.
- Every cell on the diagonal contains -1; this means the graphs have no self-loops.
- If an edge exists between nodes i and j, its weight (a positive number) is in cell (i, j).
- The graph is undirected: the value in cell (i, j) is always equal to the value in cell (j, i).

1. **Load the example graphs and programmatically count their nodes and edges.** Write code which reads the CSV files and loads the graphs into memory. You may use any graph representation you wish. For each graph, your code should determine both the number of nodes and the number of edges.

   Deliverable 1: A table which reports the exact number of nodes and edges for each of the 30 graphs. You will be graded in part on the correctness of these values; please clearly state which graph yields which values. Note: the values in this table must be obtained using code, not by counting by hand. If I can't verify that your code works, I may provide you with additional graph examples and ask you to demonstrate that your code is computing the values.

2. **Determine whether the graphs are sparse or dense.** The three "types" of graphs are different in terms of their density; it's your job to determine which is which. This difference has an impact on which MST algorithm is best for them. **Note:** it's often not useful to talk about the density or sparsity of a single graph in isolation without additional context. In this project, think about each of the *types* of graphs as having a characteristic sparsity or density, rather than looking at any graph in isolation. Is the Type-x "family" sparse or dense? That's the question you need to answer here, rather than answering for individual graphs.

   Deliverable 2: For each of the 3 types, provide your conjecture about whether the type is sparse (that is, its graphs satisfy $|E| \in O(|V|)$) or dense (that is, its graphs satisfy $|E| \in \Theta(|V|^2)$). Note: there are multiple (sometimes conflicting) definitions of graph density. Please use only the definition I'm giving you here. Support your conjecture using plots or some other means!

3. **Implement 2 distinct algorithms to determine the Minimum Spanning Tree of a graph.** Select two MST algorithms from the following list and implement that algorithm in code:

   (a) Prim's using weight matrix with an unordered array-based priority queue,

   (b) Prim's using adjacency list with a heap-based priority queue,

   (c) Kruskal's using adjacency list.

   A complete and correct implementation of any 2 of these will receive full credit; however, selecting all 3 algorithms may make you eligible for bonus points.

   Deliverable 3: The code for your implementation, and a table showing the total weight of an MST for each of the 30 graphs. Note: your submitted report does not need to contain the MSTs themselves, you only need to report their weights.

4. **Estimate the time complexity of your implementations.** Perform an empirical timing analysis of your implemented algorithms. You may use any means you wish to do this (plots and an empirical "limit test" are reasonable options), but please verify that your algorithm achieves or exceeds the time complexity that you answered on Homework 8. If your implementation improves upon the bound your proved in Homework 8, try to explain why.

   Deliverable 4: An estimate of the time complexity of your implementations for each of the 3 types of graph. Support your estimate using plots or some other means. Make sure your conclusions are directly and clearly supported by the results of your experiments!