Corbin Charpentier

Lab 13 Report

**Introduction:**

The following report contains efficiency analysis of Kruskal and Prim algorithms for finding the minimal spanning tree of a graph. The report starts with implementation, a description of how the experiment was quantified. After that, the actual experimental results and brief analysis are displayed. Finally, the report concludes with a deeper analysis of the results and a few comments .

**Implementation:**

*Timing* – timing was implemented using the provided Timer class. A global instance of Timer was used record the time in seconds needed to execute Kruskal and Prim algorithms. More technically, the timer began (Timer::start()) ticking when each algorithm was called and stopped (Timer::stop()) when they returned. The elapsed time, returned by Timer::stop(), was the total time required for each algorithm.

*Comparison Counting* – A global variable, int \_count, was allocated in main.cpp to track the number of comparisons in Prim’s or Kruskal’s algorithms. Also, a public member, int count, was added to the DisjointSet class and added to the comparisons in the Kruskal function. At the beginning of each of the two functions, \_count was set to 0, and after each comparison, \_count was incremented by one.

*Random Data* – Random full graphs were generated by generating a random integer between 1 and 100 using rand(). The random number generated was seeded with the computer time to ensure non-duplicate graphs. The half connect graph was generated in the following way: for each i,j on the matrix, generate a 1 or 0 randomly using rand. If the number is 1, give M[I,j] a weight, else leave it at infinity.

The time took longest for Kruskal because there was not early out if the first weight in the Edge list was INF, so the algorithm was applied on every edge in the edge list (they were all INF). A simple check INF before the while loop will fix this. For comparisons, again, a check before the while loop will eliminate all comparisons for Kruskal’s algorithm when given a non-connected graph.

Time seems a bit erratic for Prim’s algorithm. This can probably be attributed to inconsistent load on the cycle servers. The test was run multiple times with varying results for time. The comparison count yielded more consistent results. It’s important to note that comparisons for sorting were not counted in this experiment. Hence, the slower run time and fewer comparisons. The growth rate for both algorithms is obviously greater than n.

Again we see inconclusive results for time and a greater-than-linear growth rate.

**Conclusion:**

While the experiments don’t obviously demonstrate the actually complexities of Prim’s and Kruskal’s algorithms, they did show they are no bound by O(n). Prim’s is bounded by O(n^2) and Kruskal’s O(nlogn), where n is the number of vertices. The algorithm run times were inconclusive at best, attributing the pseudo-random times to EECS cycle server activity. Kruskal’s algorithm appeared slower than Prim’s while actually having the smaller complexity. Kruskal did, however, have the burden of sorting the edge array each time called. Sorting the array beforehand or inserting the edges into a priority queue would solve this problem (by moving it somewhere else).