Daniel Murga

Organization

For each value of n starting at 1000, a for loop is used based on n until 8,000, incrementing n by n = n * 2. Within the loop, a nested for loop from 1 to 5 sets the seed for each n, using srand() with values incremented from 1 to 20. For each location in the matrix, a vertex with random cost is assigned or no vertex with value 0. The program passes the adjacency matrix pointer to the minimum spanning tree class, starts the timer, and runs each algorithm on the graph. The time to complete these operations is then output to the terminal and the test calculates the average with the stored durations for each algorithm.

Data Generation

To generate the data for the test, a for loop from 1 to 5 sets the seed in srand() for each n based on the value of variable srandx incremented from 1 to 20. The inner double nested loop iterates from 0 to n and populates the adjacency matrix called random with vertices with a random cost depending on the value of int x or 0 if there is no vertex. Each iteration doubles the size of n until 8000.

Summary of Results

From the results using a priority queue, the average time to perform the graph algorithms on the set of data, Prim's algorithm is significantly faster than Kruskal's. Since Prim's algorithm is better suited for a densely filled graph and the adjacency matrix for each graph was significantly large with many edges, Prim's algorithm was more suited and performed faster. For Prim's results, each time n was doubled, the time to complete seemed to be a geometric growth. Going from 5.17 to 42.23, 333.25, and 2664.42 seconds for n. While the timing for Prim's algorithm may be inflated due to the use of unnecessary loop iterations during implementation or inefficient techniques, the times Kruskal's were much shorter when implemented using quicksort, starting at 0.073, 0.362, 1.914, and 9.739.

Observation and Conclusion

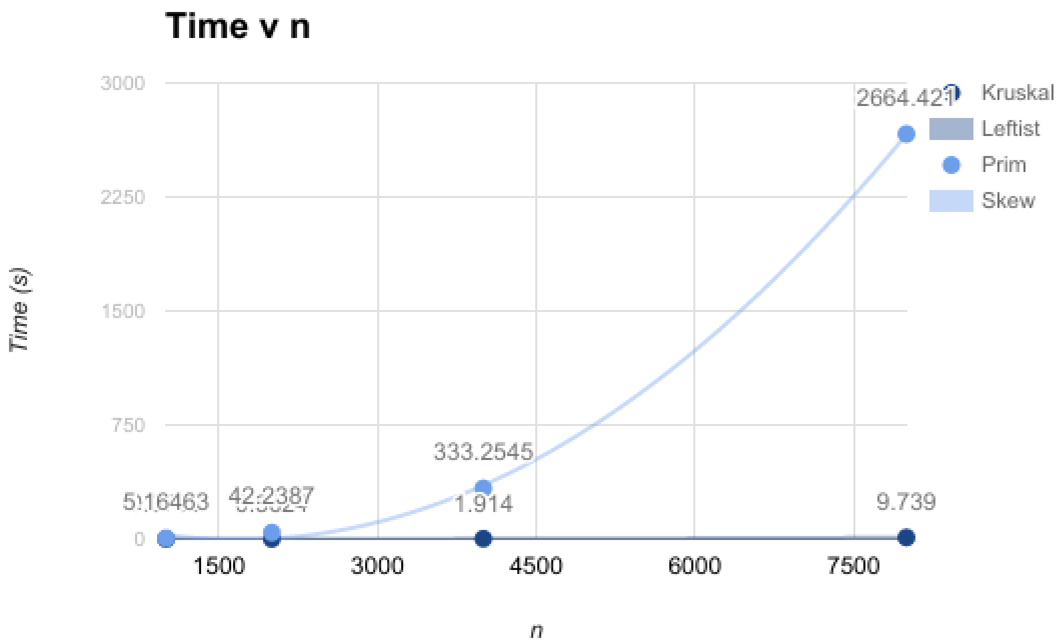
When performing the algorithms on the same set of data, Prim's algorithm performs much faster than the equivalent Kruskal's implementation when using a priority queue. Kruskal must iterate through the entire upper triangular section of the adjacency matrix and use a priority queue to keep track of the minimum cost edge. Since Kruskal's algorithm must iterate through both the matrix and queue both with complexity of O(n), it takes longer to complete than Prim's implementation which does not add an edge if either vertex is already part of the accepted set and does not check a vertex unless it has already been accepted. With the modification to use quicksort instead of a priority queue, the results are completely different and Kruskal's is faster, but this may only happen if Prim's implementation is not as efficient as could be achieved.

Daniel Murga

BUILD	Kruskal	Prim
1000	0.0724	5.1648
	0.0731	5.1626
	0.0741	5.1659
	0.0728	5.1608
	0.0738	5.1687
2000	0.3713	42.2314
	0.3788	42.4512
	0.381	43.1689
	0.3684	41.9785
	0.3624	42.3612
4000	1.953	333.3456
	1.965	335.1248
	1.99	332.5132
	1.919	331.2448
	1.913	333.0148
8000	9.4992	2664.789
	9.6142	2668.9541
	9.7391	2662.5483
	9.6934	2664.4543
	9.7392	2661.4651

AVG BUILD	Kruskal	Prim
1000	0.0732	5.16463
2000	0.3624	42.2387
4000	1.914	333.2545
8000	9.739	2664.421

Daniel Murga


Time v n
Kruskal
Leftist
Prim
Skew
Time (s)
n