HW 5 Project Report

CS4040

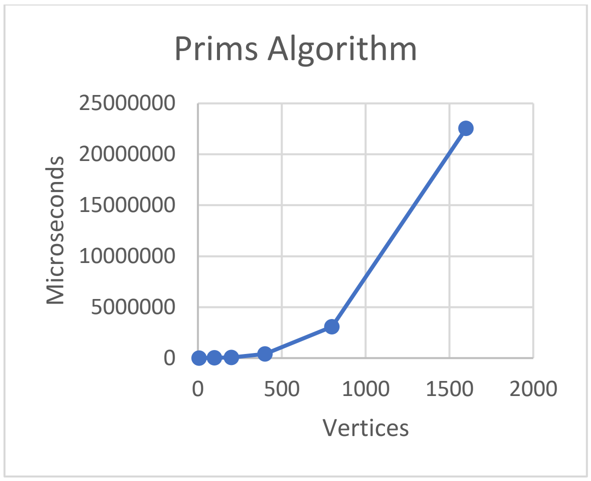
Drew VanAtta

To use the makefile I provided simply use make run. This creates and runs an output file called "greedy.out". My program creates a bunch of output .txt files into the tests folder, so running make clean clears that. When running the executable, it loops through the test graphs, creating multiple output files. The output files are named after their algorithm with the # of the loop at the end signifying which graph it used (1 is the first graph, 2 the second, etc.). The first line of the output files are the weight of the MST, and the rest of the lines are the edges. The terminal shows the loop number, followed by the input graph it used for the algorithms, and the file name it outputted it to.

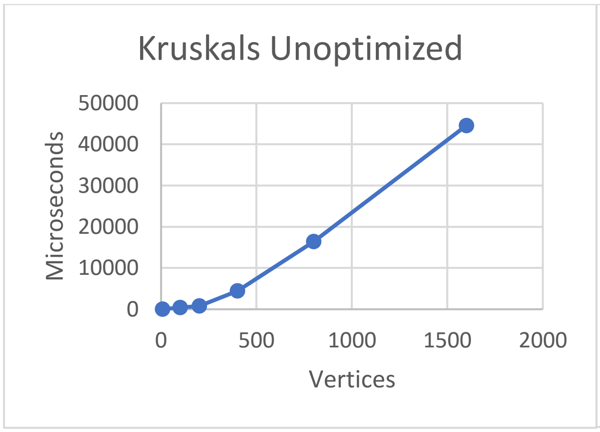
First to implement prim's algorithm I used an adjacency matrix, which turned out to be really slow. I used this because I remembered doing it from data structures as a way to represent a graph. Basically what I did was I created a vector of bools to represent if the vertice was in the MST or not. Then I used a while loop that ran as long as the amount of edges in the MST was less than the vertices -1, since that's how many edges will always be in the MST. I probably could have implemented it better because I used two for loops and it definitely showed in the timing.

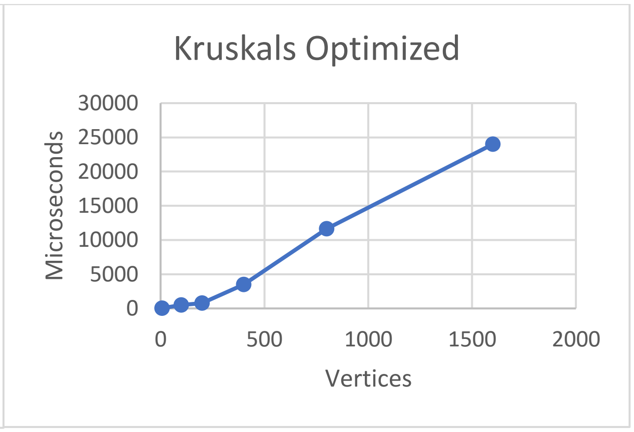
For implementing Kruskals I used a vector of the list of edges in the form of (v1, v2, weight). I probably should've just done this for prim's as well, but it works and its too late to change it. First for kruskals I sorted the vectors by weight so that we could chose the lowest edge. Using the disjoint array and union+find I was able to compare the parents of each disjoint set. I checked if the parents were the same, and if they were, it did nothing. If they were different, then there wasn't a loop and it could be added to the MST. To make the implementation faster, in the find function when returning the root parent, the optimal find sets the parent.

Next page has my graphs:



Prims	
Vertices	Time (Microseconds)
7	60
100	20745
200	64193
400	409453
800	3061985
1600	22540737





Kruskals Unoptimized		Kruskals Optimized	
Vertices	Time (Microseconds)	Vertices	Time (Microseconds)
7	29	7	38
100	414	100	475
200	791	200	782
400	4418	400	3476
800	16421	800	11628
1600	44575	1600	23994

These graphs are kind of what I expected. Prim’s is way slower than I expected, and I think it’s because I implemented it so sloppy. The two versions of kruskals behaved as I expected. The time for the lower vertices was a bit higher for the unoptimized version, but as it continued to get bigger and bigger, the optimized version became significantly faster. Both graphs curve’s are pretty similar. It was easier to compare them in the excel sheet where I grabbed them from, but I made them fit onto one page on here to look nicer. The type of graph could affect it to, as prims and kruskals do different things. Kruskals would be better for graphs that are more spread out since it sorts by shortest edges, while prims would be better for nodes that are connected to more nodes in big groups/clusters.