HW 5

Pete Newman

Instructions for running code

1. in the terminal run `make` to compile

2. in the terminal run `./hw5` to execute

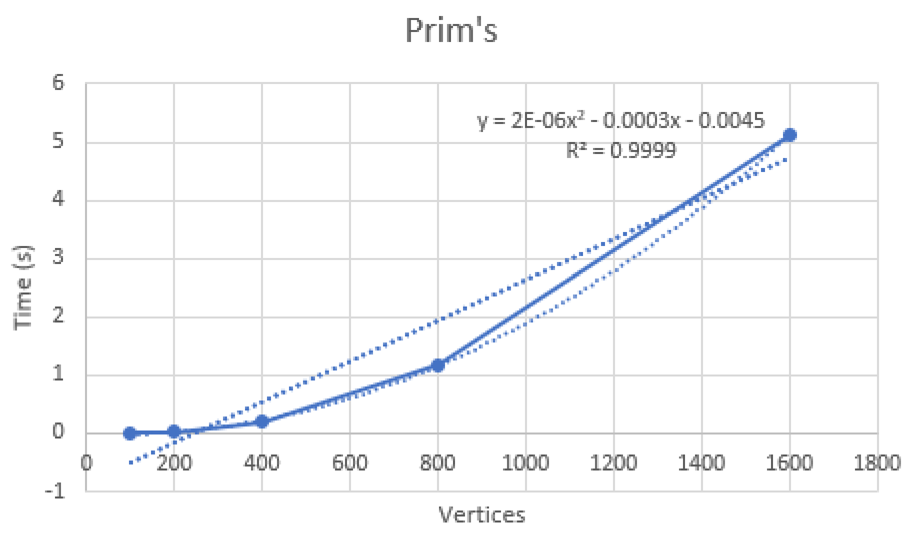
`make clean` will clean up the folders

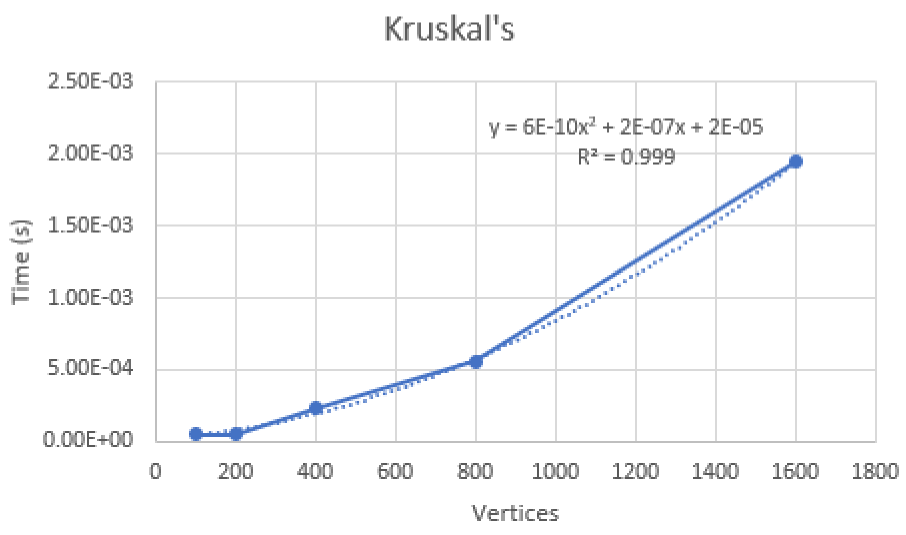
How each algorithm was implemented

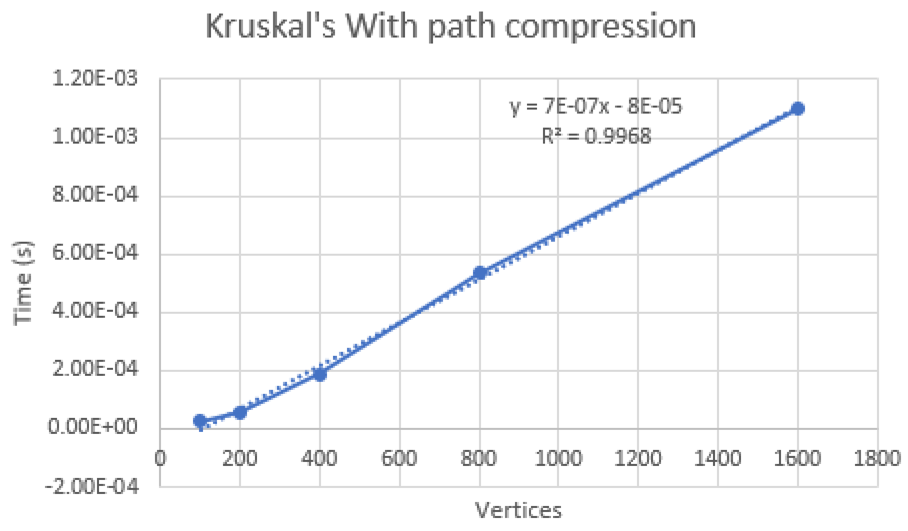
To implement Prim's algorithm, I first initialized the MST and the variables I chose to use to track the visited and unvisited vertexes. For this I chose to use the set type. This is because it is perfect for when you only want one of each thing recorded in it as we do in the case of the vertexes. After this was complete, I entered a loop that continued until the visited set was equal to the unvisited set. Inside that loop, there is another loop that iterates over all the edges. If that edge connects a visited vertex with an unvisited vertex and if the weight of the edge is less than the current minimum weight, it updates the next edge and minimum weight. Once this inner loop is completed it adds the next edge to the MST. Finally, once the outer loop is completed it returns the completed MST.

For the two implementations of Kruskal's algorithm, the only real difference is the way that the disjoint sets operate. I created 2 different sets of find and union functions. One set did not use path compression and the other did. Path compression makes every node point to the root node instead of the previous making the find function much faster. I only had to create a different union function because it needed to use the find function that used path compression. Once I had these created, I just implemented Kruskal's algorithm in the same way 2 times with different find and union functions. I started by sorting the edges by weight, then I created the disjoint sets. After this I was able to enter a loop that iterated over all the edges and checks to see if the vertices are in different sets. If they are, the vertex is added to the MST and the sets are merged. Once this loop is completed, the function returns the complete MST.

Charts


Prim's
y = 2E-06x² - 0.0003x - 0.0045
R² = 0.9999
Time (s)
Vertices
6
5
4
3
2
1
0
-1
0
200
400
600
800
1000
1200
1400
1600
1800


Kruskal's
y = 6E-10x² + 2E-07x + 2E-05
R² = 0.999
Time (s)
Vertices
2.50E-03
2.00E-03
1.50E-03
1.00E-03
5.00E-04
0.00E+00
0
200
400
600
800
1000
1200
1400
1600
1800



These algorithms performed well. They all behaved similarly to how I thought they would given their Implementations and time complexities. Prim's algorithm did seem to be a bit high but this could be due to an unoptimized loop somewhere. The Kruskal's algorithms seemed to do a little better than the time complexity. Kruskal's algorithm did seem to benefit from path compression. The times are about half of what they were without the path compression. This actually led it to having the best times out of all three algorithms.