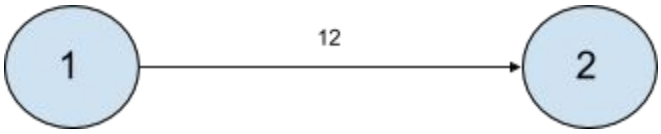
Minimum Spanning Tree in Parallel
Progress Report #1
CS4745
Andrew Atkinson
3229071

So far I've found the MST algorithm easy to implement sequentially, and the current C implementation using two-dimensional arrays. I've chosen this method over linked-lists as traversing and organizing the tree's vertices will be simpler. Dividing the tree into smaller segments for different processes would be simpler using a two-dimensional array as well, simply creating smaller subset arrays for each process. Before subprocesses can begin the algorithm, the array is divided into p-blocks, with p = process count.

eg. edges[1][2] = 12



After the grid is divided and the respective pieces handed to the worker processes, the workers begin Kruskal's Algorithm. The algorithm itself begins at the initial vertex of the grid and starts to create a new set using a two-dimensional array with one dimension equal to the edge count of the tree, and the second of length 2. The first dimension contains the vertices this edge connects while the second contains the edge weight. The index denotes the 'priority' of the edge.

The algorithm fills this array with the weights of edges sorted by non-decreasing weight. After the list is complete and the list contains all edges for this subset of the tree, the worker begins creating a final set of the MST for this portion of the graph. If the sorted list contains one vertex more than once, it will select the lowest available weight for that vertex for the sub-MST.

Once all the vertices from the sub-tree are found in the sub-MST the worker passes this sub-MST back to the master process. After the master has had all worker processes complete their work, it combines the results into a final complete MST. I have not yet determined how to implement the final combining of the results.

I do not believe implementing this algorithm through MPI should be extremely challenging. I will begin the MPI implementation of the program this week. I will likely create both static and dynamic implementations of the program. Both implementations will be tested with different values for P and their timing and efficiencies tested thoroughly.