CSCE 411-200: Honors Design and Analysis of Algorithms
Assignment Cover Page
Fall 2016

Name:	Carsten Hood
Email:	carstenhood@tamu.edu
Assignment:	HW P
Grade (filled in by grader):	

Please list below all sources (people, books, webpages, etc) consulted regarding this assignment (use the back if necessary):

CSCE 411 Students	Other People	Printed Material	Web Material (give URL)	Other Sources
1.	1.	1.	1. (URLS listed at bottom)	1.
2.	2.	2.	2.	2.
3.	3.	3.	3.	3.
4.	4.	4.	4.	4.
5.	5.	5.	5.	5.

Recall that TAMU Student Rules define academic misconduct to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. *Disciplinary actions range from grade penalty to expulsion*.

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment."

Signature:	[illegible]
Date:	Dec 5, 2016

http://thecodersportal.com/mst-krushkal/
http://scanftree.com/Data_Structure/prim's-algorithm
https://en.wikipedia.org/wiki/Kruskal's_algorithm
http://www.sanfoundry.com/cpp-program-implement-binary-heap/
http://stackoverflow.com/questions/17009056/how-to-implement-ologn-decrease-key-operation-for-min-heap-based-priority-queu
cplusplus.com

Comparing Minimum Spanning Tree Algorithms
Kruskal's Algorithm and Prim's Algorithm

Overview (Updated Proposal)

This report analyzes and compares the runtimes of two algorithms that address the minimum spanning tree (MST) problem. The MST problem purposes to find a subset of edges in a connected weighted undirected graph that minimizes the total edge weight while including all vertices and avoiding cycles. The two MST algorithms discussed are *Kruskal's MST algorithm* and *Prim's MST algorithm*.

These algorithms are implemented as C++ programs. They are measured and compared by their empirical runtimes, calculated in nanoseconds. Multiple iterations of each algorithm across varied inputs are assessed in order to obtain more accurate results.

Implementation

Both Kruskal's algorithm and Prim's algorithm are implemented as simple C++ functions, in the files *kruskals.h* and *prims.h*, and are modeled after their primary descriptions in Lecture 8's notes on greedy algorithms. These functions accept a reference to a graph object as their sole input parameter and then output a sequence of edges constituting their calculated MSTs. Each graph object, as defined in *graph.h*, is represented in memory by a vector of edges, a vector of vertices, and a corresponding vertex matrix comprising edge weights. These parallel underlying data structures facilitate graph operations.

Several other data structures are implemented as C++ classes to underpin the MST algorithms. Prim's algorithm is based on a priority queue implementation, which in turn utilizes a binary heap. Notably, the binary heap implementation entails a vector that maps vertex identifiers to their associated weights' indices in the heap structure, which facilitates looking up heap elements in the course of the necessary *decrease-key* operation. Similarly, Kruskal's algorithm is based on a disjoint sets data structure that features path compression and a weighted union function. In addition, Kruskal's utilizes a middle-pivot quick-sort function, which sorts edge objects according to their weights. Hence, altogether we have the secondary header files *priorityqueue.h*, *binaryheap.h*, *disjointsets.h*, and *quicksort.h*.

The two algorithm implementations are tested in *main.cpp* and executed on my local machine. Their runtimes are measured using the C++ library functions in <*sys/time.h*>. Clock values are recorded before and after each algorithm process and then evaluated to produce the process's overall runtime. Each algorithm is tested with both a completely connected graph and a minimally connected linked-list-style graph at each of four input sizes, these being 10, 100, 1000, and 10000 vertices. Larger input values produced impractically large runtimes.

Figures

Figure 1

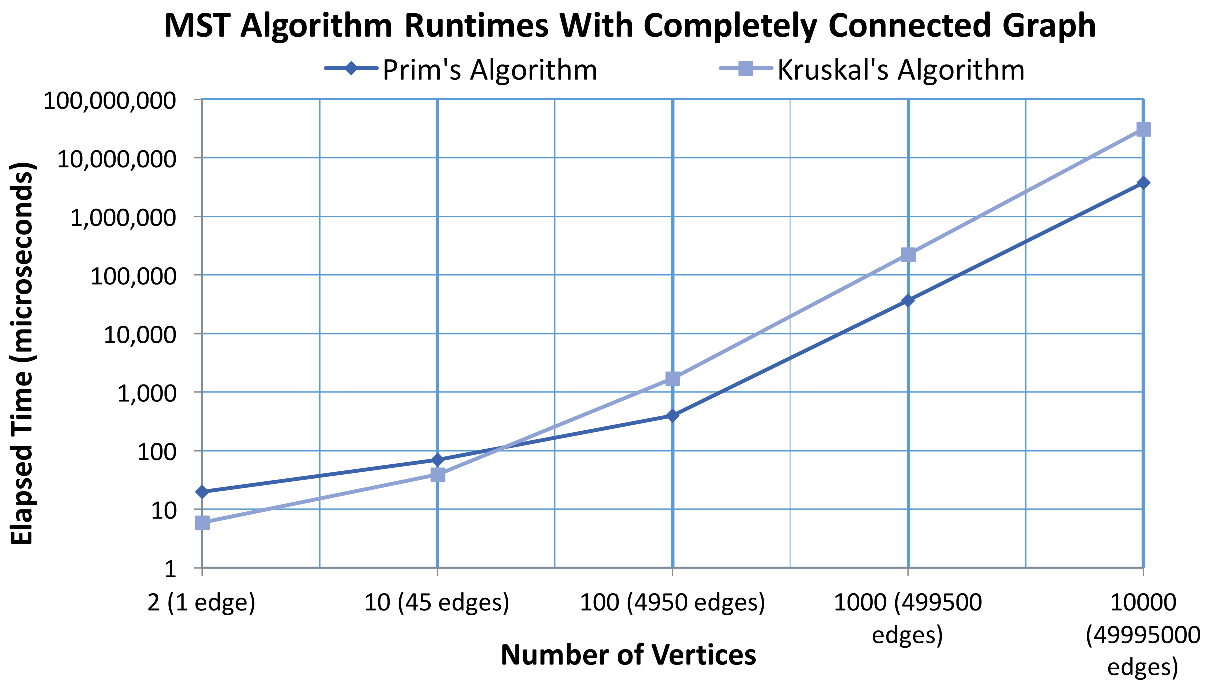
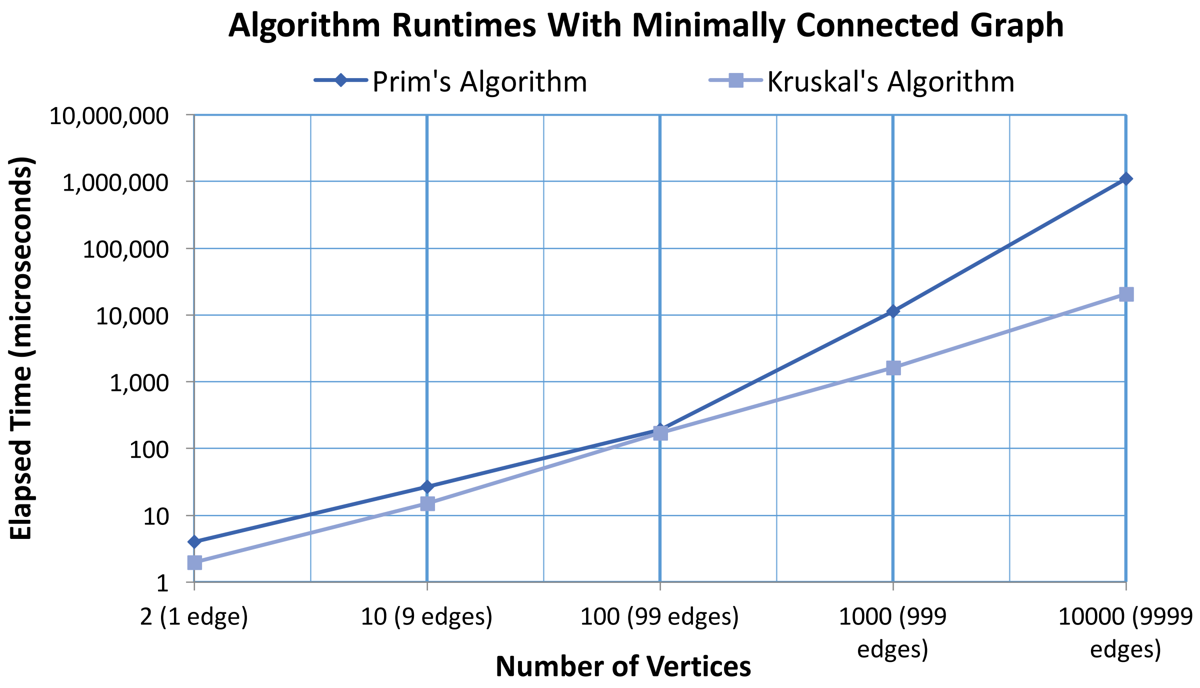


Figure 2



Theoretical Complexity

Both MST algorithms are implemented with underlying data structures such that they should exhibit $O(E \log V)$ complexities.

Sorting edges via quicksort at the beginning of Kruskal's algorithm takes $O(E \log E)$. Its remaining operations can be evaluated using amortized analysis of the disjoint-sets-related processes. There is one make-set operation for each vertex, or V total, of $O(E)$ total disjoint set operations; hence we have $O(E \log V)$. This efficiency is achievable due to the disjoint set implementation's path compression and weighted union features. Therefore, Kruskal's runtime is $O(E \log E)$ for the quicksort and $O(E \log V)$ for the disjoint set operations, which together simplify to $O(E \log V)$.

The complexity of Prim's algorithm depends on its underlying priority queue implementation. The binary-heap-based priority queue furnishes insert, extract-min, and decrease-key operations that each take $O(\log V)$. Since there are $O(E)$ decrease-key operations and $O(V)$ insertion and extraction operations, we have $O(E \log V + V \log V)$, or $O(E \log V)$.

Results

The above plots depict the runtimes of Prim's and Kruskal's algorithms across varied graph sizes and types. In Figure 1 the test graphs are completely connected, with edges between every pair of vertices, or $V(V + 1)/2$ edges total. In Figure 2 the test graphs are formed in a chain (or linked list) such that there are $V - 1$ edges. Edge weights are randomized integers between 1 and 100. The two figures' contrasting extremes of possible ratios between edge and vertex counts are intended to span all possible graph inputs, allowing for more nuanced comparison of the two algorithms.

Figure 1 exemplifies both MST algorithms' characteristic $O(E \log V)$ runtimes. The plots scale slightly upwards against the logarithmic axes, indicating the logarithmic element of their omplexity functions. However, while they scale similarly on the plot, the implementations' empirical runtimes clearly differ. The Prim's implementation processes a graph of 100 vertices (almost 5000 edges) in around 400 microseconds. Given the same input graph, Kruskal's implementation will take around 1700 microseconds – over four times as long – to produce the same output. This differential persists through all large data sizes. I suspect that this owes to Kruskal's high preparatory cost of sorting these fully connected graphs' large edge sequences, whereas the outer loop of Prim's algorithm's iterates through a much smaller vertices sequence.

The runtimes of the minimally connected linked-list-style graphs of Figure 2 support this idea. In these tests, the input graphs' vertex and edge counts are nearly identical. Here Kruskal's actually scales significantly more efficiently than Prim's for large input sizes. The Prim's implementation's relative inefficiency may be partly due to it considering all potential neighbors of a vertex when seeking a new shortest edge. This is one limitation of using a matrix to look up a vertices' edges and might be improved by augmenting the graph data structure, e.g., by mapping vertices to lists of their edges.

Conclusion

Overall, the implementations demonstrated comparable performances and adhered to their theoretical complexities. One clear takeaway from this report is that neither Kruskal's algorithm nor Prim's algorithm is generally superior. Whether one is preferable depends on the application, particularly on the properties of the graphs they will deal with. Less connected graphs may perform better using Kruskal's algorithm; wsell-connected graphs may perform better with Prim's algorithm.