LP2

G16: Ian Laurain & Vasudev Ravindran

CS6301: IDSA

Introduction:

This project required the students to implement graph algorithms for finding the minimum spanning tree of undirected, as well as directed, graphs. The project was divided into two levels. Level one focused on MST algorithms for undirected graphs (Prim & Kruskal), while level two focused on implementing an algorithm for finding the MST of a directed graph (a rough approximation of Edmond's algorithm without the emphasis on utilizing Priority Queues).

Methodology:

For level 1:

Level 1 of LP2 focused on implementing Prim and Kruskal's algorithms for finding MST's of undirected graphs. Prim's algorithms relies heavily on heap-based priority queues. Code from SP0-PQ was reused completely. The readme for this small project is attached for your consideration. The two versions of Prim's algorithm are similar in that they both utilize heap-based data structures. However, one version uses a regular BinaryHeap, while the other uses an IndexedHeap. The results of the IndexedHeap Prim algorithm far outperform the BinaryHeap version of Prim's algorithm for large input data sets. The slower performing version of Prim's algorithm utilizes a priority queue of edges, while the faster performing version utilizes a priority queue of vertexes, where the index can be accessed and changed at will, leading to faster performance.

Kruskal's algorithm is another algorithm for finding the MST of an undirected graph. It relies on Sets to achieve its desired behavior. Each vertex starts out as its own set, and sets are merged (with the set union operation) when the representative element of those sets are the same. It seems to be the slowest performing of all the undirected MST algorithms. This is most likely due to the merging of sets and finding of representative set elements being a bottleneck in the algorithm. However, Kruskal's algorithm is very simple, and clean to implement.

For level 2:

The level 2, directed MST algorithm was attempted, but significant issues were encountered, so it was not finished. Steps 1 through 4 of the algorithm outlined in class were, for the most part, implemented, but remain unfinished. The students will continue with the effort to finish it. The

level 2 algorithm is, conceptually, quite easy to understand, but a bit more difficult to implement. It relies on mutating the input graph (or its copy) to efficiently find the MST of a directed graph. Zero weight cycles are contracted until a zero-weight MST is found. Once this occurs, the resulting, shrunk graph is expanded (with knowledge of the original) to obtain the directed MST. The contraction phase of the algorithm (where a cycle of zero-weight nodes are shrunk into a super vertex) seemed to be the most challenging. Especially when updating the input graph. Making copies of the graph (zero-weight graph G0 and the graph with supervertexes) can slow the algorithm down quite a bit, since for each recursive step we need a new zero-weight edge Graph (G0) and a shrunken supervertex-containing subgraph. Implementation complexities were a challenge for the studenst, as deciding when to make nodes active or inactive, and maintaining knowledge of these changes, were a huge challenge. The students decided to resort to the simpler (from an abstraction point of view) implementation of making versions of the zero-weight edge graph and the mutated original graph, containing supervertexes. Despite these efforts, the studenst were unable to complete the level 2 task. Since the students put significant effort into both levels, and because of the availability of future ISDA long projects, the students have decided to submit what has been implemented as is.

Analysis:

For analysis of data and timing results, please view each readme in the submitted project directory. Thank you.