CS/SE 2XC3 Lab 8 Report

Glotov, Oleg
L03, 400174037
glotovo@mcmaster.ca

Willson, Emma
L02, 400309856
willsone@mcmaster.ca

March 21, 2022

This report includes the main observations that we found in this week's lab, along with the analysis of our results.

1 Prim's Algorithm

In this section, we discuss Prim's algorithm for finding the minimum spanning tree.

1.1 Prim's Algorithm Version 1

In this implementation of Prim's algorithm, two sets are used: One set contains a list of vertices already included in MST, the other set contains vertices not yet included. With an adjacency list representation, all graph vertices can be traversed in O(V+E) time using BFS. Here, the vertices not yet included in the MST will be stored in a list that has to be sorted each time. This approach is inefficient and will is improved in the following implementation.

1.2 List vs. Min Heap

The most expensive functions in the implementation of Prim's algorithm are finding and updating the weight of the minimum edge. Our first implementation uses a list of edges that are sorted by weight. The algorithm re-sorts this list for every edge visited. The Python `sort()` function has a time complexity in $O(nlogn)$. Our second implementation uses a heap of nodes that are sorted by edge weight. The heap property is maintained using `extract_min` and `decrease_key`, so there is no need to sort. Both of these functions are in $O(logn)$ The difference between these implementations is that the first one visits each edge, while the second visits each node. As the graph becomes more densely connected, visiting nodes (v2) becomes less efficient than visiting edges (v1). However in a sparse graph with a large number of nodes, using a min heap of nodes is faster. That's why we expect v2 to perform significantly better on large, sparse graphs.

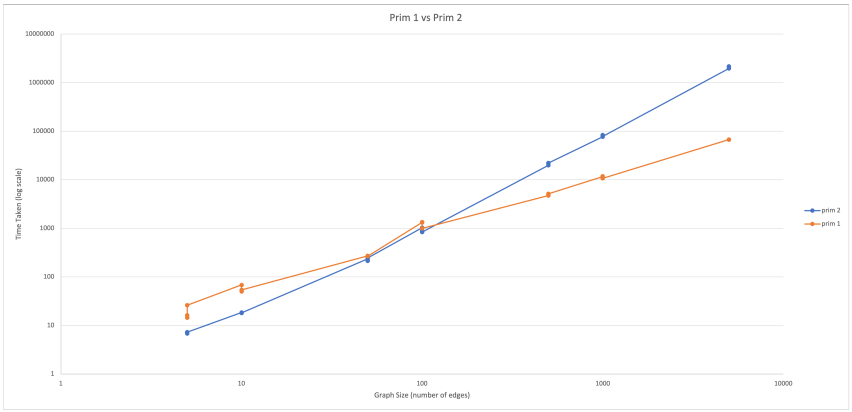

Figure 1: time complexity of prim v1 vs. prim v2

The graph above shows that the second implementation outperforms the first after graph sizes of around 100. Sorting vast arrays becomes significantly slower at larger array sizes than rearranging a Min heap which was explained earlier. This effect is clearly seen in the graph.

The testing method used was consistent with earlier labs. A function to generate a random graph given a number of edges and vertices was used. Given the graphs' randomness, the assumption that input graphs are connected had to be relaxed to avoid re-computing. For each graph size, the algorithm was timed three times, ranging from a size of 5 to 5000 vertices.