Noah French (njf5cu)
Lab 11
12/1/17
Filename: postlab11.pdf
File Description: Graphs Post-lab Report

Complexity Analysis

My topological sorting algorithm is theta(V + E), where V is the number of vertices and E is the number of edges. First, I loop through vertices to find the vertex with zero edges in. I print out the node, set its in-value to be a NULL value (-1). The runtime of this step is linearly dependent on V, the number of vertices. Then, I loop through the nodes that the zero-in node projected to and decrement each of their in-values by one. The runtime of this step is linearly dependent on E, on the number of edges. Thus the total big-theta runtime is V + E. My algorithm allocates two temporary variables, the node to print out, and a vector of the nodes it projects to. Both of these are automatically deallocated at the end of my algorithm. The graph itself, however, stays in memory after the topological sort is completed, so the space used is v + e, where v is the space taken up by all of the nodes, and e is the space taken up by all of the edges.

My algorithm for finding the shortest possible route was, as instructed brute force. Therefore, the runtime is theta(n!). I sort the destinations, set the shortestDistance float to be the total distance of the first permutation of the destination order, then calculate the distance of every other permutation, updating shortestDistance and a vector that holds the route whenever a shorter path is found. There are n! routes to check, which is why the big-theta runtime is n!. My algorithm does not allocate any new memory aside from a float of the shortest distance and a vector of the names (in order) of the shortest route. Thus, the space used linearly increases with the number of cities to travel to, as each one will be stored as a string in the algorithm.

Acceleration Techniques

Researchers have developed a number of ecologically-inspired acceleration techniques for solving the traveling salesman problem. One group of computer scientists at Universiti Malaysia Pahang derived a “metaheuristic algorithm that is derived from careful observation of the African buffalos, a species of wild cows, in the African forests and savannahs” (https://www.hindawi.com/journals/cin/2016/1510256/). They modeled the buffalo’s navigational patterns searching for food through the African landscape to create a “very competetive” algorithm that can quickly find the best case route over a number of different test cases. The algorithm has a better big-theta runtime than a brute-force solution. It is exponential, rather than n!, so it would be able beat my solution in speed for certain test cases when the number of nodes gets large.

The Christofides algorithm is a technique for approximating the shortest possible route in the traveling salesman problem. With the algorithm, the approximated route has been proven to be within a factor of 1.5 of the shortest possible route, which is currently he best known “approximation ratio” for general cases of the traveling salesman problem (https://en.wikipedia.org/wiki/Christofides_algorithm). First, the algorithm finds what is known as the shortest “spanning tree,” which is a graph with the shortest total edge length that still connects all the cities. Then, it finds a perfect matching (a graph where no edges share common vertices) of all the vertices with an odd number of edges connected to them. The edges of the spanning tree and perfect matching are then combined, and the resulting graph is modified so every edge is visited just once. Finally, all repeated vertices are skipped (https://xlinux.nist.gov/dads/HTML/christofides.html). The runtime would be closer to linear. Finding the shortest spanning tree just involves looping through all the possible edges and finding the shortest path between two, then adding onto that. It would therefore drastically increase my code’s runtime, which is currently theta(n!).

Researchers at Stanford and McGill University recently developed an approximation technique that generates a path within 1.4 times the length of the shortest path. Fractions of the graph are solved separately, and are rounded off into a spanning tree (not necessarily the shortest spanning tree). This spanning tree then undergoes all the same steps as the shortest panning tree in the Chrisofides algorithm (https://web.stanford.edu/~saberi/tsp.pdf). This technique too would drastically speed up my code, but less so than the pure Christofides algorithm. This newer algorithm introduces randomized components to the creation of the spanning tree, adding more time to the Christofides algorithm. Obviously, as a trade off, the generated path is guaranteed to be less than 1.4 times the actual shortest path, whereas the Christofides algorithm only guarantees that the path will be within 1.5 times the actual shortest.