Josh Balzer

Discrete Structures

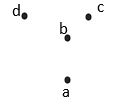
April 30, 2020

Travelling Salesman Report

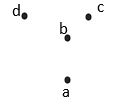
The travelling salesman problem is a problem that definitive solutions can be found for a small number of cities but becomes impossible to solve as the number of cities increases. This is because as the number of cities increases, the number of possible solutions increases exponentially. Therefore, the only way to answer the travelling salesman problem is by making an estimate of the fastest route.

In my program I have arrays that contain each capitals’ latitude and longitude coordinates. My program uses the coordinates to create a complete graph connecting every capital with edges weighted by the miles between the cities. I used an algorithm based on Prim’s Minimum Spanning Tree algorithm to find a minimum spanning tree from the graph connecting all of the state capitals. At this point, for each pair of capitals that still have an edge connecting them another edge is created to connect them. This allows for a simple circuit to be created from the minimum spanning tree. As the circuit is being created, a list is created that logs each city that is visited in the order they were visited. After the circuit has completed, the program iterates through the list and creates “shortcuts” by removing duplicate listings of any capitals that have been visited multiple times (excluding the ending city which is also the starting city) and recreates the edge that existed between the listed city before and the listed city after the duplicate listings. This is done because the circuit must sometimes return to parent nodes in the tree during traversal and the straight distance between these two capitals must be less than the distance of revisiting the “root” capital in between them. By this point, the minimum spanning tree has been transformed into a new graph and a simple circuit has been created connecting all the capitals. The graph that remains is the result that my program estimates is the shortest path between the listed cities.

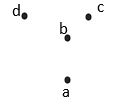
I will use this smaller scale example to help illustrate what is going on in the background of my program’s execution. Rather than starting with the latitude and longitude coordinates of the cities, in this example my program would start with the x and y coordinates of the points a, b, c, and d.



It would first create the complete graph with weighted edges.

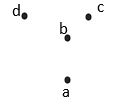


Using the weighted edges, it would find the minimum spanning tree.

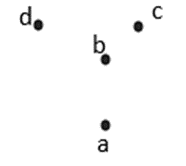


Each edge would be duplicated, and the circuit would be found visiting the nodes in the order,

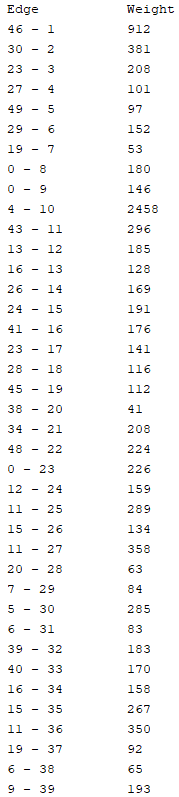
a, b, c, b, d, b, a.

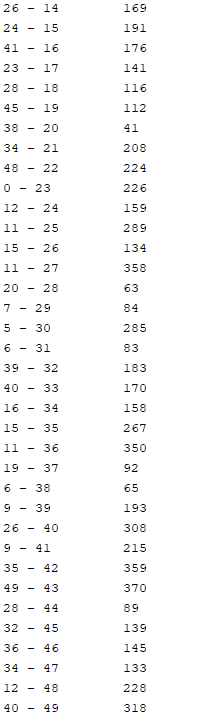


The duplicate traversals through b would be removed and the shortcuts from c to d and d to a would be created. The following graph would be the result of my program’s final calculation. The program would finish by printing the ordered list of capitals visited, in this case the order of traversal would be a, b, c, d, a.



The program is written as a console application. After downloading the project from the GitHub, the easiest way to compile and run the program is to use an IDE with Java support. Using the IDE, click on “open project”, locate where the project has been installed, open the “TravellingSalesmanSolution-master” folder and the “TravellingSalesmanSolution-master” folder inside it. From there open the “TravellingSalesmanSolution” project and use the IDE’s build and run function. Upon running the function, the following will be printed out representing the minimum spanning tree rooted at 0(representing Montgomery, Alabama as it if first alphabetically and the numbering continues by alphabetical order). Under the edge list, the left number is the parent node of the node on the right.





The program then prints the circuit before shortcutting.



















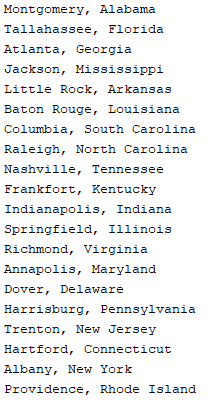








Finally, the program prints the order of the tour after shortcutting and prints the name of the capital and the state rather than the number.





This is the end of execution.