**Project 2 – BFS and DFS**

Zachary Shumate

[CECS]

Speed School of Engineering

University of Louisville, USA

znshum01@louisville.edu

1. **Introduction** (What did you do in this project and why?)

This project’s goal was to find the shortest path, Hamiltonian Cycle, from the first to last city in a given list. The cities this time where only connected to cities that were specified. To find the best path I needed to use a Breadth First Search and a Depth First Search approach. The Breadth First Search approach finds the fewest transitions to each node; however, the Depth First Search approach calculated the shortest distance to the goal node.

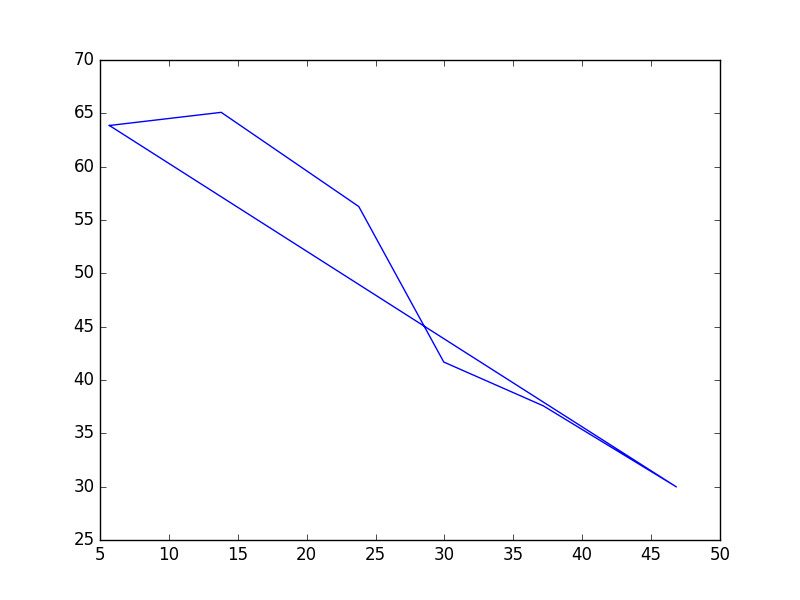
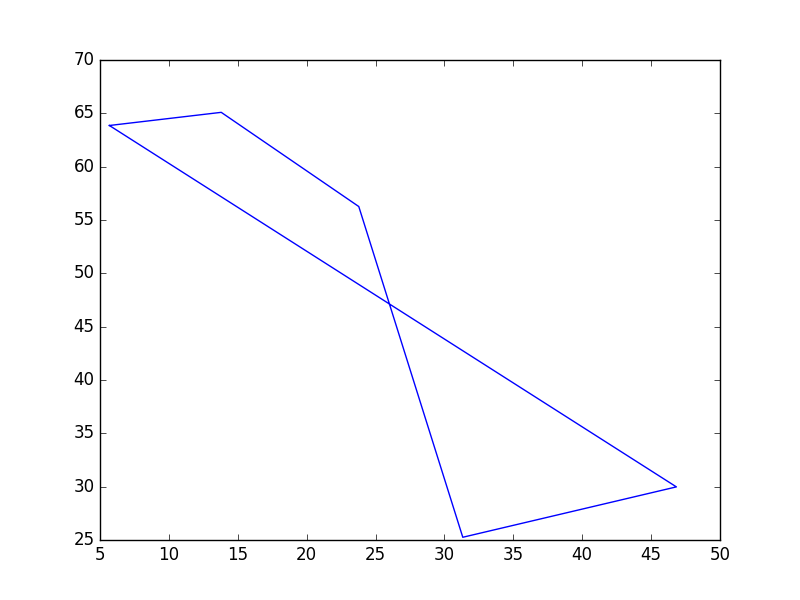
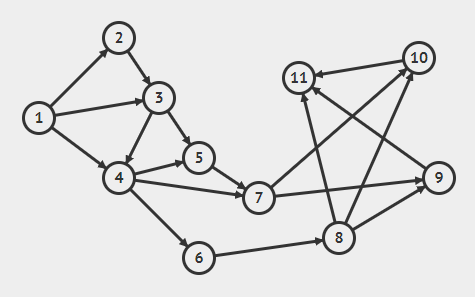
1. **Approach** (Describe algorithm you are using for this project)

My Python script starts by accepting the files with the cities and coordinates that are put into a list. From there I added the connected cities to each node manually. In the Breadth First Search, the algorithm, starting with the first city, followed each available path from each node and recorded the number of transitions to said node and the previous node to accomplish this, skipping nodes that already have these details recorded. From there all I needed to do was backtrack from the final city to recover the best path for fewest transitions. In the Depth First Search, the algorithm entered a recursive function that called itself in order to follow each path to the goal; when a better route than was previously discovered is found, the new distance and previous node are overwritten with the more optimal data. The recorded path is then retraced to find the optimal path for shortest Euclidean distance.

1. **Results** (How well did the algorithm perform?)
   1. **Data** (Describe the data you used.)

The data used for this project consisted of the provided eleven city file. The x and y coordinates for each city where unique.

* 1. **Results** (Numerical results and any figures or tables.)

****

**Breadth First Search**

**Depth First Search**

**Breadth First Search**: **'path'**: [{'prev': '0', 'next': [2, 3, 4], 'num': '1', 'y': '63.860370', 'x': '5.681818', 'transitions': 0}, {'prev': '1', 'next': [4, 5], 'num': '3', 'y': '65.092402', 'x': '13.798701', 'transitions': 1}, {'prev': '3', 'next': [7, 8], 'num': '5', 'y': '56.262834', 'x': '23.782468', 'transitions': 2}, {'prev': '5', 'next': [9, 10, 11], 'num': '8', 'y': '25.256674', 'x': '31.331169', 'transitions': 3}, {'prev': '8', 'next': [], 'num': '11', 'y': '29.979466', 'x': '46.834416', 'transitions': 4}], **'transitions'**: 4

**Depth First Search**: **'path'**: [{'distance': 0, 'next': [2, 3, 4], 'num': '1', 'y': '63.860370', 'x': '5.681818', 'prev': '0'}, {'distance': 8.209853377784102, 'next': [4, 5], 'num': '3', 'y': '65.092402', 'x': '13.798701', 'prev': '1'}, {'distance': 21.53790179369681, 'next': [7, 8], 'num': '5', 'y': '56.262834', 'x': '23.782468', 'prev': '3'}, {'distance': 37.36835819134047, 'next': [9, 10], 'num': '7', 'y': '41.683778', 'x': '29.951299', 'prev': '5'}, {'distance': 45.67812117008132, 'next': [11], 'num': '9', 'y': '37.577002', 'x': '37.175325', 'prev': '7'}, {'distance': 57.96716475441191, 'next': [], 'num': '11', 'y': '29.979466', 'x': '46.834416', 'prev': '9'}], **'distance'**: 57.96716475441191

1. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

The algorithms that I used were able to quickly find the paths for the 11 cities; breadth first search took 0.000648975372314 seconds and depth first search took 0.000304937362671 seconds. The breadth first search achieved the fewest transitions, 4, taking the path 1,3,5,8,11 while the depth first search found the shortest distance, 57.96716475441191, in 5 transitions following the path 1,3,5,7,9,11.

1. **References** (If you used any sources in addition to lectures please include them here.)

I did not use any additional resources. All of the code was written by me.