COP4531 Assignment 4 Jared Bennett
Spring 2017 Discussion jab13f

Shortest Flight Distance Between Airports and Known Flight Routes

1. The Application

Vertices – All Airports world-wide (at least as far as my data source knows). Names correspond with airport ID's from the data source.

Edges – Known flight routes between airports. These are directional as specified in the data source.

Edge Weights – Distance in km between the airports, calculated from their respective GPS coordinates

- If I had a better data source this could have been something more interesting, such as flight cost or time. I did not find any useable sources that were free.

2. Graph Size

After parsing the data files and formatting as the right input, the graph came to 7184 vertices and 66067 edges. Many of the vertices had no edges connect to them at all due to lack of data. Technically it is many graphs as there is no guarantee that any two given airports are connected. Though, the data source is not complete so the end results are not as accurate as they could be.

Visual representation from data source: http://openflights.org/demo/openflights-routedb-2048.png.

3. Data Source

All the airport and flight data came from http://openflights.org/data.html. It is a free website that supplies various data related to traveling. It also has a few online tools to view it. It may not be as complete as other, paid, sources, but it was hard enough just to find something in a readable, text-based format. I combined the airport data from here with the route data from here.

4. Optimizations

There are no necessary optimizations needed for the algorithm for this data source, as the data was not that large. There are relatively few airports and flight paths comparted to other large data sets. Processing is almost instantaneous with an average of 62ms. However, I did have to implement an error check in case the destination was not actually connected to the start airport, as many of the airports were missing route data.

5. Summary

Although it would have been more interesting with more data, using the Dijkstra algorithm has proven to be very effective at finding the shortest flight distance among known flight roots. Given more data, I could see the weight being altered to include price, delay, and flight time information to help decided which flights to purchase. Since the N is small for airports and routes, it does not cost too much time or money to calculate. This allows companies, such as airlines, to provide these calculations as services to their customers through an easy to use website.

General Discussion

Optimization

I spent a bit of time optimizing the algorithm and came up with a way to 'remove' an item from the standard priority queue without that much cost (I did not fully evaluate it, but it was much faster). The general idea is that the actual items in the queue are proxy pointers to pointers that in turn point to each vertex. This allows an item to be 'removed' by swapping the proxy's pointer with a fake pointer with the same comparative value and then re-inserting the pointer so that it gets properly placed in the queue. The original implementation of this was cumbersome but I slimmed it down and it seems to work quite well.

Testing

I created a separate program to generate large, random graph inputs to test the algorithm. This is largely testing environment specific, but I thought I'd at least include a few times since the real-world usage was so small. In each case the names were a random alpha-numeric string of 1-10 characters and the edges were slightly randomized for their maximum amount per-vertex. The larger the input file the more time it takes just to read it from disk (My VM was running out of space so I didn't go super high).

Vertex Count	**Edge Count**	**Input File Size (MB)**	**Time (s)**
100000	549342 (10 max/v)	17	1.692
100000	5058185 (100 max/v)	150	9.954
1000000	50515229 (100 max/v)	1500	137.734
1000000	5498132 (10 max/v)	169	27.761

All tests were run on an Ubuntu 16.04 VMware Virtual Machine with 2 processors @ 3.4GHz and 8GB RAM. I believe a large portion of the time is spent parsing the data from the file, as they must be loaded from disk. There is a chance multithreading the initial disk load would make it faster, though.