Michael Warnock

CS 404

26 November 2016

Sneaky Path Project

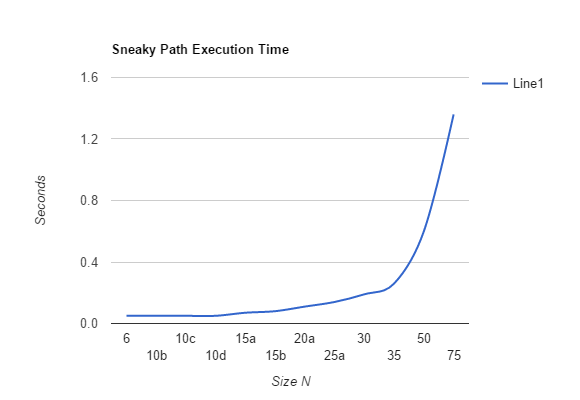
For this project there were two obvious Algorithms that could be used to complete the task, Floyd-Warshall and Dijkstra’s shortest path algorithms. At first I was leaning toward Dijkstra because it seemed like there was a lot less processing. Why calculate the shortest path for every source and destination when you could just do it for one? I also wanted to do adjacency lists first as well because it is more space efficient. In the end I saw that it was going to be a lot easier to code with the help of matrices which make indexing possible and iteration hassle-free. I found a Floyd-Warshall visualization at <https://www.cs.usfca.edu/~galles/visualization/Floyd.html>. This is what convinced me to choose Floyd-Warshall because I felt like I had a good understanding of it and it helped me realize the power of the Path Matrix which became a cornerstone in my solution.

First load the initial Edge (E) matrix and initial Flow (F) matrix into memory. With this information you can derive a path matrix from the E matrix. This shows all of the direct connections between nodes and the number that is represented in a cell of the path matrix is the node that it originated from. This makes an easily traceable matrix and also can be updated while running Floyd-Warshall on the Edge matrix which finds all shortest total distances from any A to B. The new path matrix is the all pairs shortest route matrix. Trace it and place each trace into a matrix and you have the hop by hop shortest path for all pairs. Now you know what edges are used for the initial Flow matrix. Iterate through every hop in the hop by hop shortest path matrix and accumulate the total traffic that is on each edge, this is matrix L. Once L is found, all that must be done is do Floyd-Warshall on it with the Initial Path Matrix and it will return the all pairs fewest cars matrix. Then that matrix, combined with the newly updated path matrix the hop by hop matrix can be derived. There was a lot of iteration but the math was light in my solution.

I know this algorithm is correct because the results are the same as they would be if you were to trace Floyd-Warshall by hand and that they match the example. However I think there is the case where there could be more than one possible sneaky path if they have the same cost. This algorithm only keeps track of one of them.

Floyd-Warshall being the main algorithm in this solution has a time complexity of O(n) = n^3. It also has the worst time complexity of any of the accompanying functions that helped me get to my solution. Creating and iterating through matrices are O(n) = n^2 for 2D matrices and O(n) = n^3 for the few 3D matrices. The overall performance is O(n) = n^3. My solution uses 9 2d arrays of size n and 2 3d arrays of size n. For a possible maximum of 2N^3 + 9N^2 bytes. I believe that this is an acceptable run time for this kind of problem, however n^3 could probably be improved on a little. According to Computational Complexity theory this solution is only one of an unknown number of possible answers in the deterministic computing space. The odds are likely that there is a faster solution.

The following graph is the runtime for each data file given us. On an Intel Core i7 4770k cpu:



This experiment shows that the actual time complexity coincides roughly with the theoretical time complexity that was expected; O(n) = n^3.

Looking back on this project I am content with most of the decisions I made. Many of the problems that appeared were due to implementation, not the ideas behind it. Although I did think I was getting the wrong answer a few times when It is actually right. In example the Matrices that can get to any node without seeing traffic. I wasted a lot of time trying to figure out what was wrong with my code because all of the other data sets seemed to be working fine. I think the hardest part of this project was getting the complete algorithm and in detailed steps. I wish I would have made my code more portable and involve some object oriented programming. This project took a lot of time and starting earlier would have certainly made my Thanksgiving break more enjoyable.