**Report**

**DSA Assignment 2019 – Social Simulation**

**By Jonathan Wright**

**Contents**

**1 – Coverpage  
2 – Contents  
3 – Abstract  
4 – Background  
5 – Methodology  
6 – Results  
7 - Conclusion**

**Abstract**

The purpose of this report is to investigate whether the order of my simulation of a Social network ( O(N) ) and memory overhead are ‘good enough’ to simulate a real world case of a Social Network such as Facebook or Twitter. I first investigated the O(N) of my algorithms to make predictions of how I believe my program would perform, and then I wrote code to test the speed of the program and ran htop to see the memory usage of the program. I recorded averages of memory overhead (in GB) and time taken (in ms) across data sets of three, one hundred and seven thousand using the loading, time stepping and saving operations of my program.

I concluded from my results that the time stepping and saving operations had a much better speed than I thought they would, however, loading was extremely slow for large data sets. This led me to be able to state that my model would in fact be able to model a real world Social Media site like Facebook or Twitter, so long as they were not loading in large datasets or they would be waiting a long time. I also discussed in my conclusion that it may indeed be better to put the users into a hash table with lists rather than a graph, and the messages in a hash table rather than a list, this removes the idea of a graph being the best idea for a ADT. I am unable to investigate this however, in a future investigation I am quite confident that a graph is may indeed not the most efficient ADT to be storing a Social Network in and could be overthrown by a different ADT.

**Background**

**Approach To Developing The Simulation:**  
First I looked into how different Social Media sites operated, how they presented data to their users and thought about different ways they could be stored.  
I came to the conclusion that its likely they most likely do not actually use graphs but rather find a way to use a efficient ADT perhaps a edited graph or a binary search tree modification (Black Red Trees, etc). For my implementation, I decided to use a graph, however, I did have the thought of using a hash map as I believe it would be much faster in terms of access and write operation speed (better O(N)). I then decided to make generic classes for a social media site that could be used outside of my specific simulation, while someone may be able to use my Social Simulation program, it’s not very helpful if they can’t use the exact same logic in their own programs. Therefore, I made Network, which is a class designed with the intent of being a Social Media network, modelled after Facebook & Twitter. After finishing the construction of my generic classes (and testing them), I linked it together in my Social Simulation program.  
  
**What My Program Will Be Investigating:**  
I will also be investigating the O(N) and memory overhead of my simulation. I would also like to look at better, more efficient ways of writing a social media site and what you would ideally like to prioritize with a social media site, I also look into if my program would be effective at modelling a real world Social Network like Facebook or Twitter.

**Methodology**

**How I’ve chosen to profile and compare multiple runs of my simulation.**I attempted to profile memory using a terminal screen running my java program of various sizes and using Linux’s htop. I then take the MEM% of each instance (Small Set, Medium Set, Large Set) and multiply it by the total memory to get how much memory it is using. To profile the speed of my simulation for each set I used System.currentTimeMillis to get start and finish by sub currTime – start. Here are the results.

**Why I am profiling:** I am profiling to test speed and efficiency of my simulation, and seeing whether it would be useful in a real world case, I will be testing time step, loading and saving speed and memory overhead and this is likely what in real world would be done (timeStep being, the post propagating one level out).

**Quick Observations:** Time was very stochastic, (in one case the small set took 20000ms on the lab pc, I believe this was partly because I had the Large Set loading in background.).  
With the Large Set, as time went on the program seem to get slower and slower at reading in the lines. Also trying to serialize a large set causes a crash.

**My Expectations:** I expected my program to be able to read in small sets quite quickly and large sets slow but not much slower than small sets as I expect the data isn’t that big in terms of memory. I expect time stepping to be quite slow with all datasets as it uses multiple linked list operations which have O(N) so for multiple linked lists to be searched is very slow (for searching as an average). Saving will most likely be about the same speeds and memory overheads as loading in a network as it uses the same operations so same order.

**What The Results Conclude:**   
**LOADING:** While small sets of n < 10 and medium sets of n < 1000 read in relatively quickly, large sets n > 5000 will take a while, from the results I’ve displayed in that table I can conclude that my expectations were wrong, as a large set was a massive 333% increase in memory usage and a 19,489.94% increase in processing time.  
**TIMESTEPPING:** Memory overhead did not change from load, the speeds did increase but by not a lot, so I can conclude that my time Stepping isn’t as inefficient as I first thought it would be, perhaps O(N) isn’t as bad as it is visualised as?  
**SAVING:** Memory overhead did not change from load, the speeds did increase but likewise with time Stepping, not by much, I can conclude that my Saving does in fact not use as inefficient algorithms as my loading does, perhaps this is because saving simply has to iterate through the list O(N).

**How To Replicate My Testing:** I have the required datasets I used attached, for the Medium Set it is called testData.csv, it contains 100 lines of data. For the small set simply load in doReMe.csv, it contains a small set, for a large set simply load in RandomNames7000.csv. To test memory you need htop and screen to run the program in background. To test speed simply put a timer at the start and end of NetworkIO readGraph and then subtract end – start to get the time taken.  
(For Saving I used a different set for medium then as testData seemed to contain invalid characters to save to a file?)

**Results**

**LOADING IN A NETWORK**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Small Set (3 Items) | Medium Set (100 Items | Large Set (7000 Items) |
| MEM% (HTOP) | 0.3 | 0.3 | 1.3 |
| (memDecVal) \* (Memory) = (GB) | 0.0459 | 0.0459 | 0.1989 |
| TIME TAKEN (ms) (average over 3 runs) | 3420.33 | 4209.33 | 747323 |

**TIME STEPPING**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Small Set (3 Items) | Medium Set (100 Items) | Large Set (7000 Items) |
| MEM% (HTOP) | 0.3 | 0.3 | 1.3 |
| (memDecVal) \* (Memory) = (GB) | 0.0459 | 0.0459 | 0.1989 |
| TIME TAKEN (ms) (average over 3 runs) | 16 | 20.39 | 35.25 |

**SAVING**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Small Set (3 Items) | Medium Set (100 Items) | Large Set (7000 Items) |
| MEM% (HTOP) | 0.3 | 0.3 | 1.3 |
| (memDecVal) \* (Memory) = (GB) | 0.0459 | 0.0459 | 0.1989 |
| TIME TAKEN (ms) (average over 3 runs) | 35 | 42 | 120 |

**Conclusion**

**Do the results meet my expectations**: No, I thought that time Stepping would be slow for all data sets and saving would be about the same speed as loading a network. This was proven wrong by my results, time stepping was fast for all tested data sets and saving was much faster than loading a network.  
  
From this investigation I can answer my initial questions, my model would in fact be able to model a real world Social Network like Twitter or Facebook, so long as you are not loading the data sets a lot as this takes a long time. There are possibly many more efficient ways of modelling a social network that my implementation did not do, that mainly due to time constraints, I was unable to test.  
In future I would investigate trying to put the users into a hash table with lists rather than a graph, and the messages in a hash table rather than a list, this would give much greater average big O notations for searching (Hash Table O(1) vs Linked Lists O(N)).  
Overall despite being told we **should** use graphs, I don’t believe that the base graph is a elegant solution, and there most likely is a better ADT for a far more efficient Social Network.