# Execution Report

Homework 4

By: Aaron Zachariah

## Contents

Experiment Overview .......... 3
Experiment Measurements .......... 3
Experiment Plots .......... 4
Running Time Analysis .......... 5

## Experiment Overview

The experiment conducted involves running the multithreaded application using various graph sizes, with certain numbers of threads. For the experiment, three different graph sizes were used. A small graph of 1000 nodes, a medium graph with 10,000 nodes, and a large graph with 100,000 nodes. Edges would be generated for each node such that the expected value of edges for a node would be 5. For example, the small graph of 1000 nodes would have an edge probability of p = 0.005

$$E[edges] = \# \, of \; nodes * p = 1000 * 0.005 = 5$$

Therefore, to get the p value used for generating a graph with n nodes,

$$p = \frac{5}{n}, where \; n = \# \, of \; nodes$$

For each graph size, the simulation was run multiple times. Each time uses a different number of threads to concurrently run the simulation. The thread counts used in this experiment are – 1, 2, 4, and 8.

## Experiment Measurements

Below are tables that contain the measurements for each of the tests. Each table corresponds to a different graph size. The tables contain the number of threads used and its respective execution time. Also note the size of the graph and the number of tick run for the test.

| Number of Threads | Time (in ms) |
|---:|---:|
| 1 | 712 |
| 2 | 563 |
| 4 | 334 |
| 8 | 252 |

Figure 1: Data with graph size of 1000 and 3 ticks

| Number of Threads | Time (in ms) |
|---:|---:|
| 1 | 5448 |
| 2 | 3616 |
| 4 | 2579 |
| 8 | 1578 |

Figure 2: Data with graph size 10,000 and 3 ticks

| Number of Threads | Time (in ms) |
|---:|---:|
| 1 | 137291 |
| 2 | 70229 |
| 4 | 39517 |
| 8 | 22631 |

Figure 3: Data with graph size 100,000 and 1 tick

## Experiment Plots

Below are plots of the experiment measurements, where the number of threads is the x axis and the execution time is the y axis. Each plot corresponds to a different graph size.

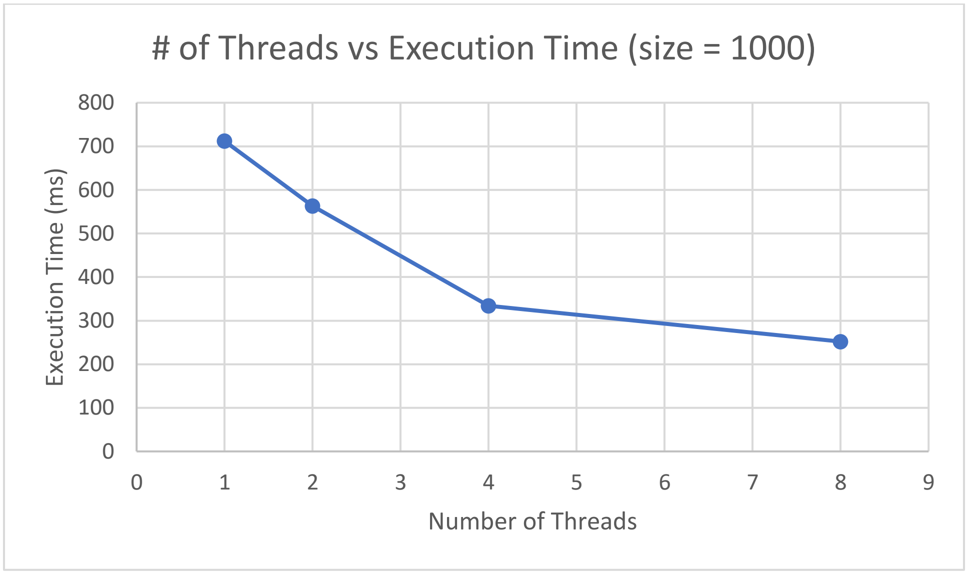

Figure 4: Plot with graph size of 1000 and 3 ticks

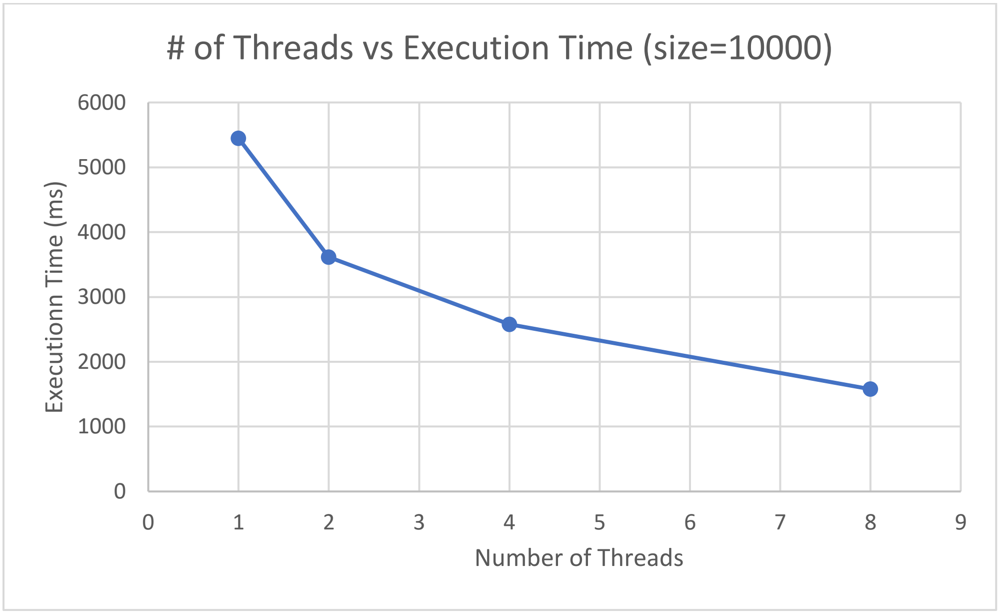

Figure 5: Plot with graph size 10,000 and 3 ticks

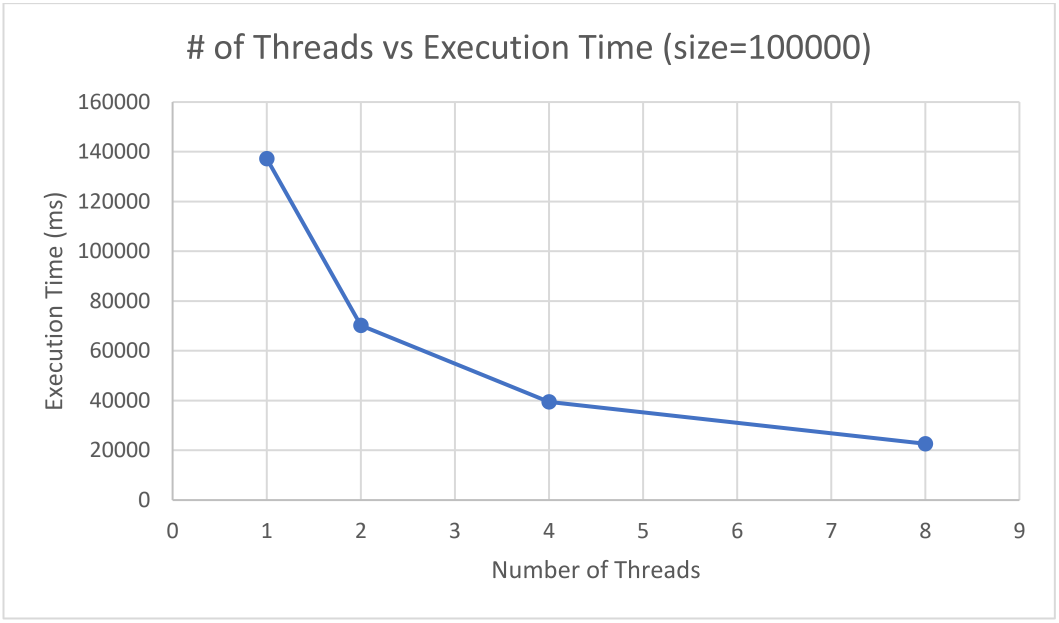

Figure 6: Plot with graph size 100,000 and 1 tick

Despite increasing the graph sizes, the general trend seems to be consistent, which may not be a surprise. The graph seems to indicate diminishing returns when adding more and more threads. Another interesting observation was the relatively large jump in execution time between the 10K and 100K graph sizes. Each graph increases the graph size by 10X but the execution time grows faster, which can provide insight on the running time of the simulation.

## Running Time Analysis

The Thread object that is responsible for performing computations to update the simulation is mostly responsible for the execution time. It contains a few methods that perform computations, but the method with the greatest effect on performance is the method responsible for simulating the infection spread.

Each thread is given an equal partition of the container holding all infected nodes at some time t. The thread loops through the partition, which is an O(# infected nodes / # threads) operation. In each loop iteration, the method must also obtain the number of susceptible neighbors, which is a O(E* N) operation, where E is the number of edges and N is the number of nodes.

If we let each thread's partition (# infected nodes / # threads) be represented as P, then the total execution time is O(P * E * N).

Considering how when the magnitude of N grows the magnitude of P grows as well, the growth in execution time certainly seems plausible.

No other method in any class of the simulation has a worse run time that the process explained above, so it is safe to assume that O(P * E * N) is a reasonable representation of the running time of the simulation. In reality it is longer, since there are other expensive loops on other methods.

The biggest challenge was that I was not use the HashMap O(1) get() method to access a certain key. I could only get the key's corresponding value. This is probably due to HashMap's having strange behavior when the keys are modified, but the Node class I created hashes nodes only by their "name" field, which is unique and immutable. The other fields were not included because they were intended to be modified. If I could use some variation of a HashMap, which would allow access to a key reference, then it would greatly improve the performance of the simulation.