Nicholas Meeks

DS 210

Prof. Kontothanassis

GitHub Social Network Dataset Analysis

I chose to use the GitHub Social Network Dataset for my project, and with this data set I chose to do the 6 Degrees of Separation Problem. With this, I wanted to find the average minimum distance between two nodes in the graph through implementing the Breadth-First Search and Dijkstra's algorithms, and comparing the performance of the two. The GitHub Social Network Dataset contains data on over 37,000 software developers with accounts on the website as well as information about other developers that each follow. The dataset encodes the "follows" relationship between two developers in a set of edges, and the developers with a set of 10 or more repositories published are the vertices.

Performance Analysis of BFS vs Dijkstra's Algorithm

Breadth-first search (BFS) is an algorithm used for searching or exhaustively exploring an unweighted graph (directed or undirected). The standard BFS algorithm begins at a given start node and explores all neighbors of said node before moving beyond that frontier. It does this by using a queue data structure to store the list of nodes from which to explore in upcoming iterations. Similarly, Dijkstra's algorithm is a shortest-path algorithm. Normally, it is used to process weighted graphs (graphs wherein each edge has an associated weight or cost to traverse). The GitHub network dataset doesn't contain weights (more precisely, each weight is 1). So, this dataset provides an interesting opportunity to compare the run-time performances of using BFS to calculate average distance between a set of nodes and Dijkstra's to do the same task. I explore the relative performance of these two algorithms by timing the algorithms running on 10 different sizes of random vertex selection. Each size is timed 5 times.

Implementation Overview

Since this was my first time implementing any type of graph algorithm, I chose to generate a simple graph of 4 nodes and 6 edges and confirm that the generation of the edge list and subsequent adjacency list were properly structured. Then, I implemented the breadth-first search algorithm. It is

coupled with a helper method that takes as input the graph adjacency list and a requested random sample size. To generate the random sample, a copy of the graph's vertices is made and then shuffled. Using the requested sample size *n*, I keep the first *n* elements from the list. Using this subset of vertices, I calculate the shortest distance between each pair using the BFS algorithm. The number of pairs is equal to the number of combinations of size 2 from the subset. I follow a similar approach for using Dijkstra's algorithm for calculating the shortest distance between two vertices from randomly generated sets. For a random sample of 50 nodes, the average min distance (degrees of separation) between all pairs of nodes is consistently between 3.0 and 4.0. This can be interpreted as each developer being on average separated by approximately 4 other developers.

For comparing the run time performance of BFS and Dijkstras, I use the above functionality but wrapped it in timers. I ran each algorithm against random sample size data sets of 10 to 100 at intervals of 10. Each interval was run 5 times. The timing of each run for an interval was averaged. From Figure 1, as the random sample size increased the amount of time needed for BFS increased at a faster rate than for Dijkstra's. Said another way, the slope of the BFS performance timing was steeper than for Dijkstra's.

Running the Project Code

The project is currently broken into a main driver file (main.rs) and two modules: 1) *graph_reader* which contains the functions to read graph data from an input text file of edges, and 2) *graph_algos* which contains the implementation of BFS and Dijkstra's as well as the helper functions for running a single random test of a specified sample size. The main driver file contains the functions needed to produce the data for the timing performance.

Output

The output for the Degrees of Separation task is printed to the screen directly. An example output for a random sample size of 10 nodes is below in Listing 1.

```
Welcome to the GitHub Degrees of Separation
Graph Statistics:
  Number of edges: 289003
  Number of vertices: 37700
--> BFS Algorithm Implementation
Total pairs: 45
Mean Distance: 3.49
Standard Deviation: 0.687
```

```
Elapsed Time: 3.208617416s
--> Dijkstra's Algorithm Implementation
Total pairs: 45
Mean Distance: 3.09
Standard Deviation: 0.694
Elapsed Time: 2.182845791s
```

Listing 1 – Output of Degrees of Separation task for a 10-vertex random sample.

Of note from the sample output is that the mean distances and standard deviations are different between the two algorithms. This can be explained by the fact that each algorithm is operating on a different random sample of nodes of the same size. I also write all minimum calculated distances to output files (BFS.txt and Dijkstras.txt) for each algorithm for later inspection.

For the performance timing output follows the above output and is similar to Listing 2. For each random sample size interval (10, 20, 30, …, 100), I output the average for the five iterations of that interval. Each run of an interval is operating on a new set of random vertices.

```
BFS Timings
    Processing size 10
       Avg time for 5 iterations: 3.2596998412
    Processing size 20
       Avg time for 5 iterations: 6.5125759830000005
    ...
  Timings:
    Size = 20; avg time = 6.5125759830000005
    Size = 10; avg time = 3.2596998412
Dijkstra's Timings
    Processing size 10
       Avg time for 5 iterations: 2.2345651662000003
    Processing size 20
       Avg time for 5 iterations: 4.439498208
    ...
  Timings:
  size = 10; avg time = 2.2345651662000003
  size = 20; avg time = 4.439498208
  ...
```

Listing 2 – Output of the runtime performance analysis of BFS vs. Dijkstra's algorithms.

Conclusion

From this, I found that Dijkstra's algorithm consistently performed quicker than the Breadth-First-Search Algorithm, and also computed a smaller mean distance between two nodes than the Breadth-First-Search did. The time difference may be due to the time complexity difference- the Breadth-First-Search algorithm has a time complexity of O(V+E), while Dijkstra's algorithm has a time complexity of O(V + E(logV)), where V is the number of vertices and E is the number of edges in the

graph. Depending on the size, this can account for the difference between the two algorithms. Breadth-First-Search also visits nodes level by level based on how close they are to the starting point, while Dijkstra's visits the node with the lowest cost, which is another reason why Dijakstra's algorithm may produce a smaller mean distance in lesser time