Kevin Villeda De Leon

Professor Kontothanassis

DS210

3 May, 2024

Connectivity of Amazon Network Items

My project is centered around an amazon co-purchasing network found on SNAP. The graph contains directed edges and revolves around the concept of "Customers who bought this item also bought this". If a customer often buys a product with another then the pair of nodes contains an edge. I was interested in this specific data set because I always wondered how amazon recommended items after you buy something. Additionally, I would always find myself clicking through recommendations until I could get to another. For example how long would it take for me to get from a simple t-shirt to a bed. While the graph doesn't associate its nodes with items it still was interesting to see how many clicks it would take to go from node to node. In this project I aimed at exploring the dataset to calculate things of statistical significance which could potentially be used in a real world scenario. For example, finding the shortest path from one node to another using breadth first search, calculating what percentage of nodes could be reached in 10 steps, and what the average distance is between every node. In other words, what is the average number of clicks it would take me to go from node a to every other node, then moving onto the next, and the one following that.

The first thing I tackled was calculating the shortest path from node A to node B. For this I used a breadth-first search method that initializes a queue and a vector to store the distances from node A to any node B. Afterwards I checked the distances from Node A to its neighbors and then from its neighbors to its neighbors until finally the final destination node was reached. I stored all these distances in a vector and it would then return the shortest distance in that vector. Initially I was struggling with just setting these new distances as it would often just repeat itself, to fix this I added one to the distance so it would update every time a node was reached. In my algo.rs module this function is called shortest_path. I then tested this function in the main.rs module, which it passed. Testing it on a much smaller data set that I

hardcoded in the create_test_graph function. This function uses the first 55 pairs of edges. After that I calculated what the shortest distance would be to go from node 1 to the 50,000th thousand node finding that the shortest path from 1 to 50,000 is 17 steps. This is applicable when making networks because this would mean that going from a lamp to an xbox would take around 17 clicks. The closer an item is the more related they are but since a lamp and an xbox are pretty unrelated it would take a lot longer to reach.

After this I aimed at tackling the question of what percentage of nodes were reachable in 10 steps from the first node to any node in the next 50,000. (50,000 chosen because of the time it would take to run this function). This function is called reachable_within_steps in my algo.rs module. This function much like the shortest_path function initializes a queue which then goes through and checks to see what the shortest distance is from point A to point B. he function goes through checking the neighbors and afterwards pushing them back into the queue so their neighbors can be explored as well. I then update the counter which updates every time a node is reached within 10 steps or less. After running a test function I calculated what percentage of nodes were reachable within 10 steps in a sample size of 50,000. I found out that in this sample size 17.9%. This means you would be able to reach almost 20 percent of items in a sample of 50,000 within 10 clicks.

Finally, the function that takes the longest time to run and is the sole reason why I limited the sample size - the average degrees of separation from any node A to any node B. This function goes through checking the average distance of every node from every node. After it adds these averages and calculates the average path length each node is from each node. The function implemented a helper function called bfs_average_path_length which found the average path length of one node to every node using breadth first search. Afterwards the average_degrees_of_seperation function goes through and implements this function for every node calculating the average. I tested this function in my main.rs module with the graph I hard coded earlier. The function passed the test which I ran using cargo test. Afterwards I ran this function on 50,000 nodes and found the average shortest path from each node to each node is 13.55. This means in a couple thousand items it would take you an average of 13.57 clicks to go from item to item following the recommended items section.

In order to run the code make sure rust and cargo are installed on your device. Additionally, you must add a dependency on your cargo.toml which is shown in the git repository but in case there is trouble accessing it, add rand = “0.0.8” and rayon = “1.5”. Additionally, if you would like to run this code on your own directed graph feel free to replace the name of the txt.file on line 33 with one of your own choosing. Just make sure the file is large enough or else the sample size numbers must be adjusted.

In order to ensure my code works I ran three tests, one for each of the functions.

```
warning: `part1` (bin "part1" test) generated 1 warning (run `cargo fix --bin "part1" --tests` to apply 1 suggestion)
    Finished test [unoptimized + debuginfo] target(s) in 0.13s
     Running unittests src/main.rs (target/debug/deps/part1-3f7fe88b651371a6)

running 3 tests
test tests::test_reachable_within_steps ... ok
test tests::test_shortest_path ... ok
test tests::test_average_degrees_of_separation ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Since the test worked I then ran my code on the main function to find specifics. Here are the results of running these functions on the amazon txt.file.

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.65s
     Running `target/debug/part1`
Shortest path from node 1 to node 50,000 is 17 steps.
Average percentage of nodes reachable within 10 steps (sampled): 17.90%

Starting at node 54767
Starting at node 352413
Starting at node 275132
Estimated average degrees of separation: 13.55
(base) kevinvilleda@MacBook-Air-2 part1 %
```

The average_degrees_of_seperation outputs where it is currently at to track progress. On 50,000 nodes it would take around 30 minutes to run.

From this project one can gain a deeper understanding of the connectivity of amazon item connections and how accessible one item is from another. For the future it would be interesting to calculate how efficient this network actually is and ways in which it could be upgraded.