Final Project Writeup: Amazon co-purchasing dataset

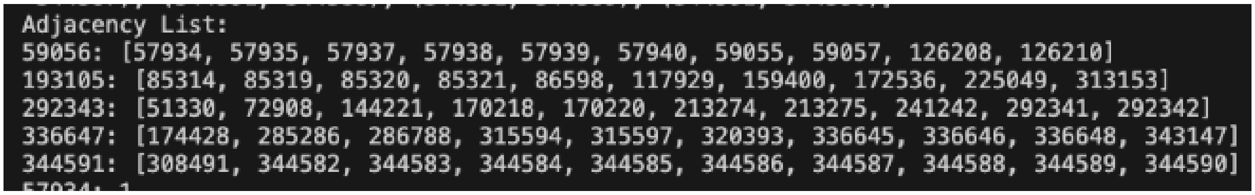
For my project, I used an Amazon co-buying dataset. In this dataset the nodes are the different items within the Amazon shop and their connections are other nodes/items that people have bought after they bought the first item. What I've decided to do is to take a random sample of five items represented by nodes, find their connections, evaluate their distance using BFS and adjacency lists, and then find the average distance between the 5 random nodes and their neighbors to gauge the popularity of these items in the Amazon shop.

First, I made a read function located in read.rs, which basically just reads in the Amazon.txt file and separates the nodes and their connections by tabs.

Next, I used the samples function to get a random sample of 5 nodes from Amazon.txt. Using these 5 nodes, I put them into a vector, called nodes_connections_formatted.

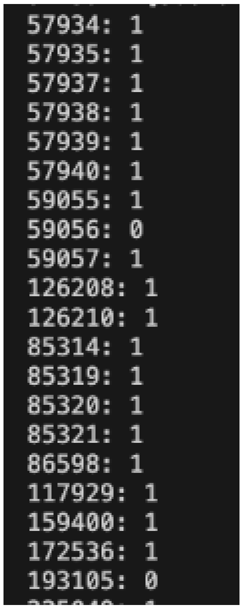
Then I used the get_source_connections function located in module sample_connect.rs to get the connections of the 5 nodes I randomly sampled. This is then stored inside a vector called node_conenctions_formatted.

After that, I created a graph function and a compute_and_print_distance_bfs function within the module graph_distances.rs module. This establishes a struct with the vertex labels and an adjacency list, which is used to implement a graph of directed nodes. This makes an adjacency list that looks like this:



```
Adjacency List:
59056: [57934, 57935, 57937, 57938, 57939, 57940, 59055, 59057, 126208, 126210]
193105: [85314, 85319, 85320, 85321, 86598, 117929, 159400, 172536, 225049, 313153]
292343: [51330, 72908, 144221, 170218, 170220, 213274, 213275, 241242, 292341, 292342]
336647: [174428, 285286, 286788, 315594, 315597, 320393, 336645, 336646, 336648, 343147]
344591: [308491, 344582, 344583, 344584, 344585, 344586, 344587, 344588, 344589, 344590]
```

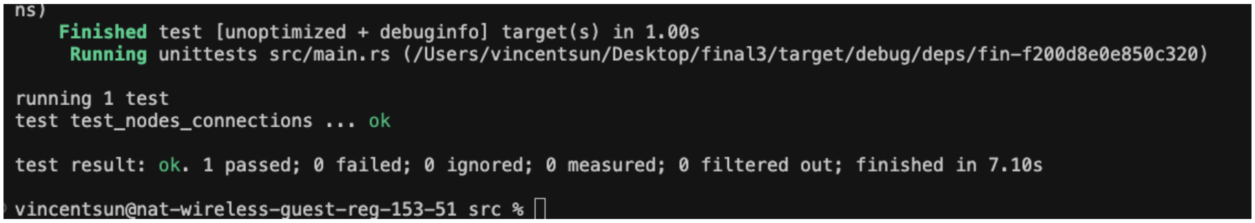
The adjacency list lists the five nodes that were selected at random and their connections. Next, the compute_and_print_distance_bfs prints the distance of these connections from the nodes selected, we know that the distance for all the connections is one except for the node itself which is zero, this just helps us visualize that and also shows that the code works. →



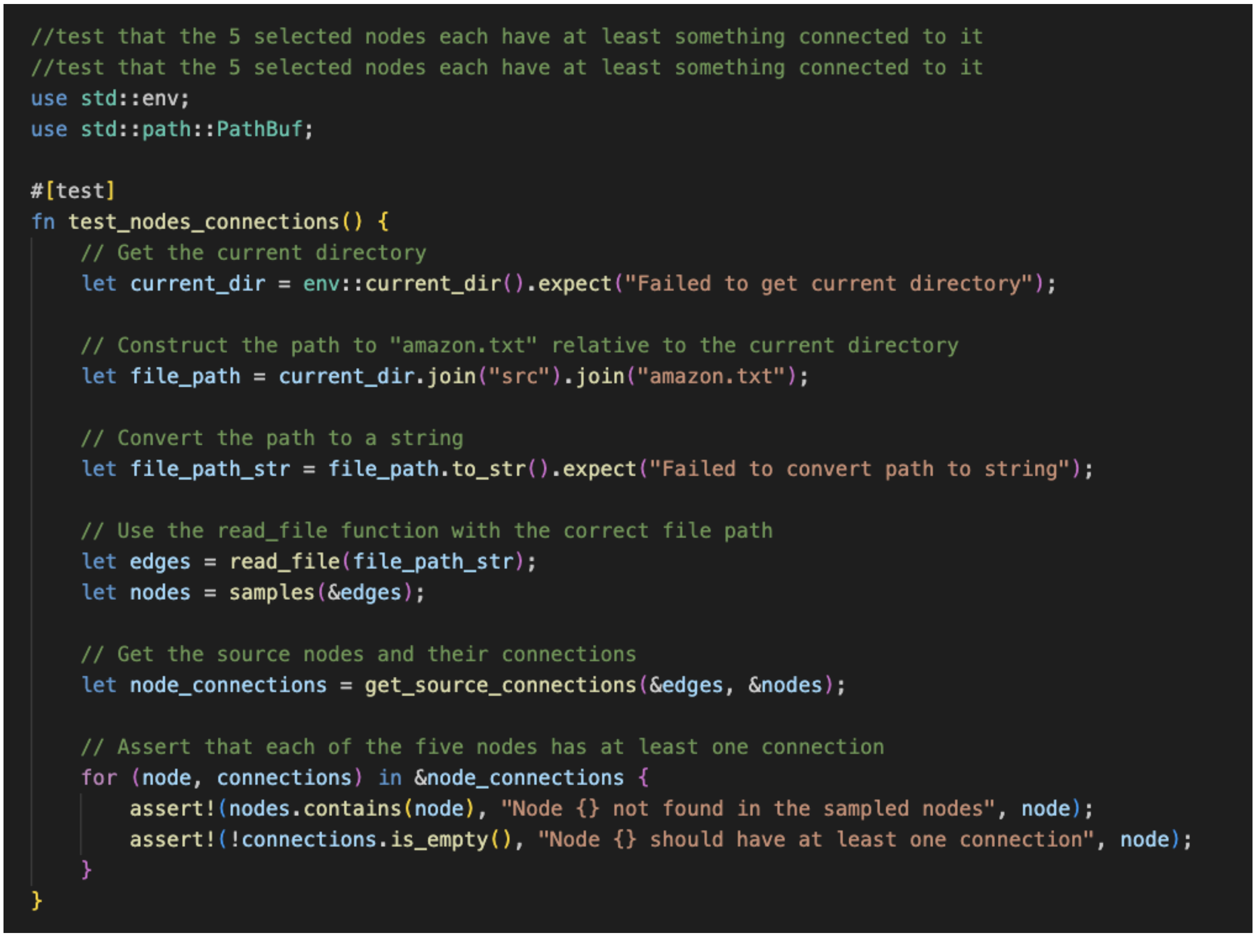
```
57934: 1
57935: 1
57937: 1
57938: 1
57939: 1
57940: 1
59055: 1
59056: 0
59057: 1
126208: 1
126210: 1
85314: 1
85319: 1
85320: 1
85321: 1
86598: 1
117929: 1
159400: 1
172536: 1
193105: 0
```

Last but not least, I just summed up the distances and divided by the amount of nodes connected to find the average distance of the group of 5 nodes randomly selected. I used this data to compare the distances of another 5 sets of randomly selected nodes to see which group of items is more popular overall. This is determined by the average value, if the average value is higher, this means that the group of 5 items is less popular because it has more distance and not more nodes connected, the average is determined by the total distance (sums of 1s) divided by the total nodes connected. I attached examples below which shows the code being run twice and comparing the average distance values of .909 and .883, we can say that the .883 group of items is more popular and has more co-buyers than the .909 group because if the denominator which is the total number of connected nodes goes up the average values grows smaller.

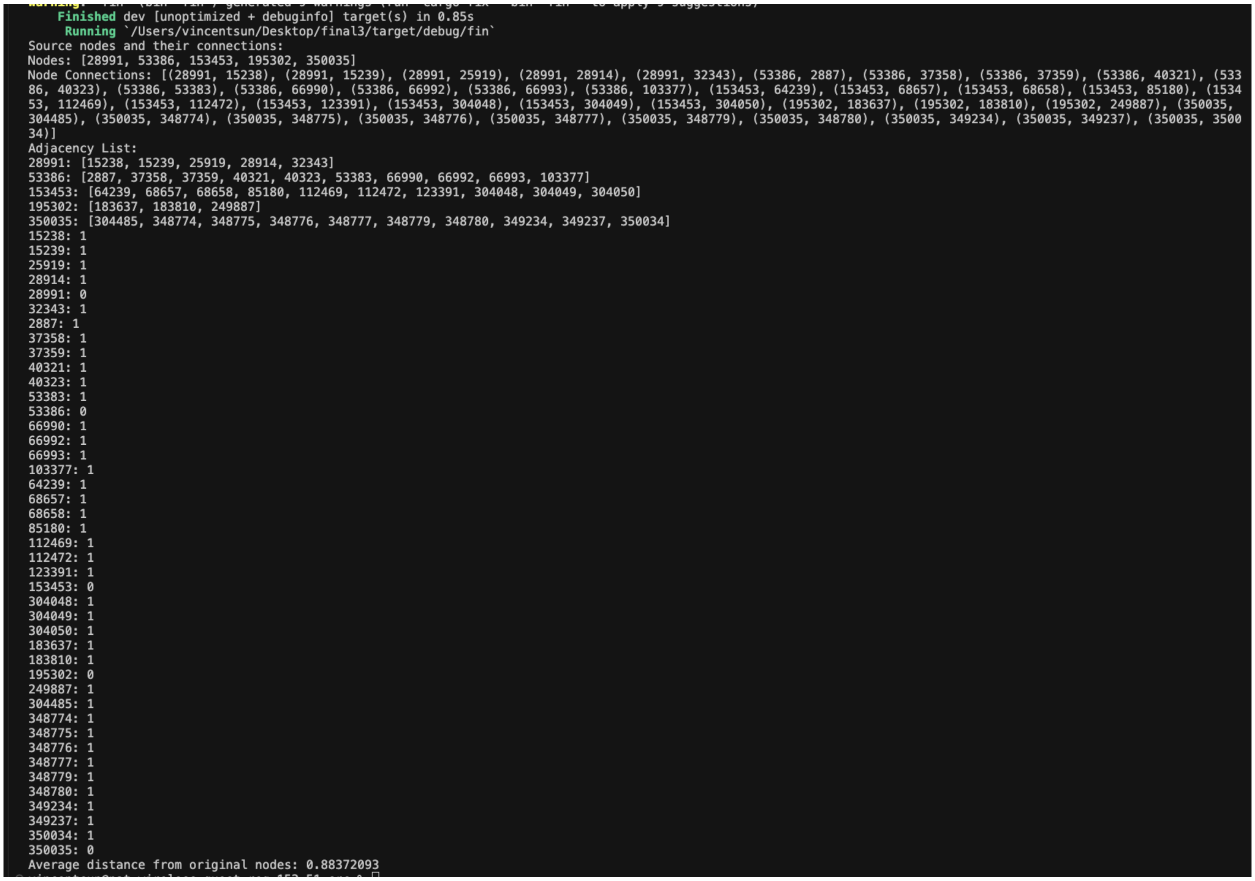
I also wrote a test function that tests whether or not the 5 nodes chosen at random from the amazon dataset even has connections to it, which if it doesn't then it will throw an error, but if it doe shave at least one connection to it, then it would pass the test.



```
ns)
    Finished test [unoptimized + debuginfo] target(s) in 1.00s
     Running unittests src/main.rs (/Users/vincentsun/Desktop/final3/target/debug/deps/fin-f200d8e0e850c320)

running 1 test
test test_nodes_connections ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 7.10s

vincentsun@nat-wireless-guest-reg-153-51 src %
```

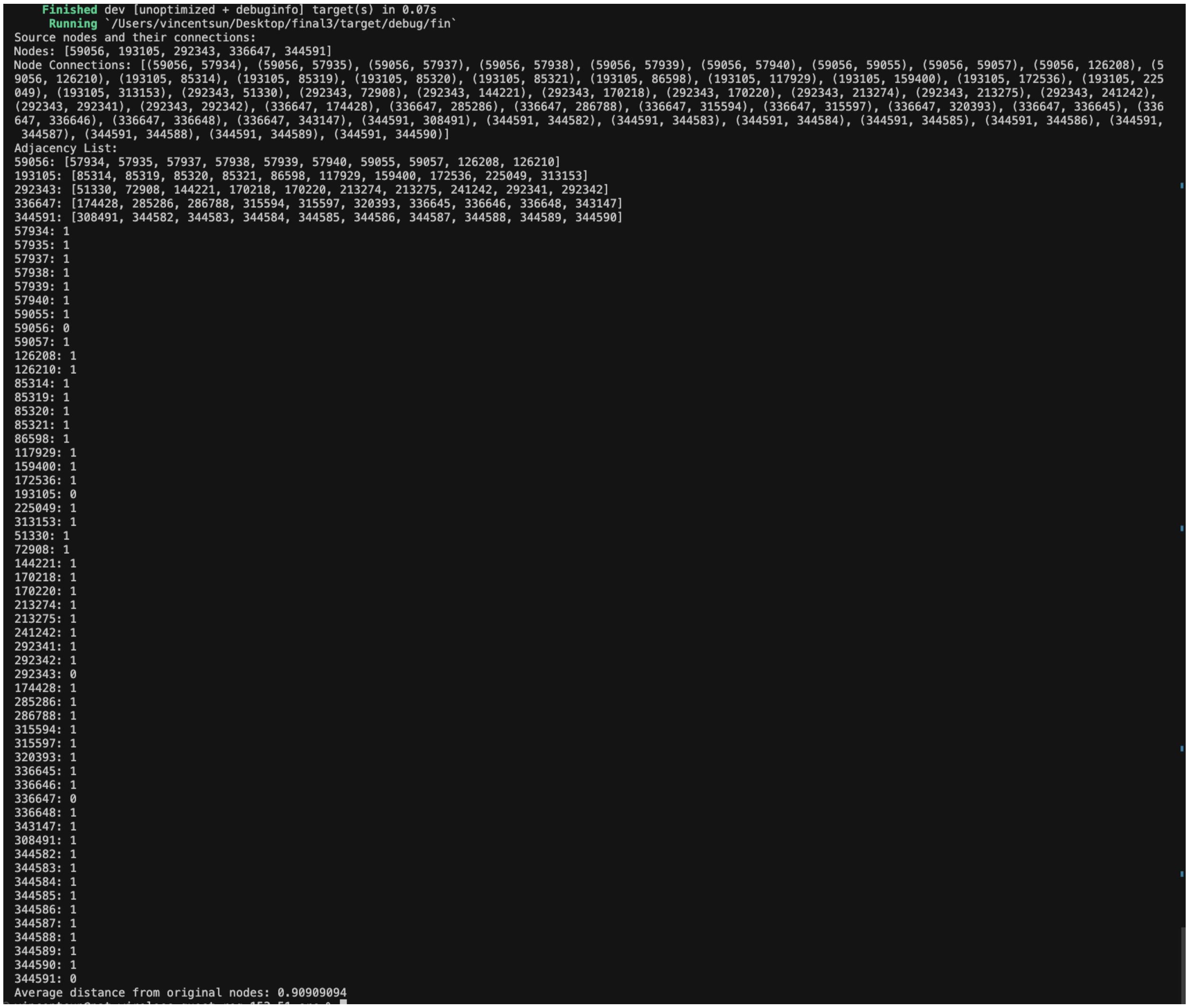


```rust
//test that the 5 selected nodes each have at least something connected to it
//test that the 5 selected nodes each have at least something connected to it
use std::env;
use std::path::PathBuf;

#[test]
fn test_nodes_connections() {
    // Get the current directory
    let current_dir = env::current_dir().expect("Failed to get current directory");

    // Construct the path to "amazon.txt" relative to the current directory
    let file_path = current_dir.join("src").join("amazon.txt");

    // Convert the path to a string
    let file_path_str = file_path.to_str().expect("Failed to convert path to string");

    // Use the read_file function with the correct file path
    let edges = read_file(file_path_str);
    let nodes = samples(&edges);

    // Get the source nodes and their connections
    let node_connections = get_source_connections(&edges, &nodes);

    // Assert that each of the five nodes has at least one connection
    for (node, connections) in &node_connections {
        assert!(nodes.contains(node), "Node {} not found in the sampled nodes", node);
        assert!(!connections.is_empty(), "Node {} should have at least one connection", node);
    }
}
```



```
   warning: `fin` (bin "fin") generated 3 warnings (run `cargo fix --bin "fin"` to apply 3 suggestions)
       Finished dev [unoptimized + debuginfo] target(s) in 0.85s
        Running `/Users/vincentsun/Desktop/final3/target/debug/fin`
   Source nodes and their connections:
   Nodes: [28991, 53386, 153453, 195302, 350035]
   Node Connections: [(28991, 15238), (28991, 15239), (28991, 25919), (28991, 28914), (28991, 32343), (53386, 2887), (53386, 37358), (53386, 37359), (53386, 40321), (533
   86, 40323), (53386, 53383), (53386, 66990), (53386, 66992), (53386, 66993), (53386, 103377), (153453, 64239), (153453, 68657), (153453, 68658), (153453, 85180), (1534
   53, 112469), (153453, 112472), (153453, 123391), (153453, 304048), (153453, 304049), (153453, 304050), (195302, 183637), (195302, 183810), (195302, 249887), (350035,
   304485), (350035, 348774), (350035, 348775), (350035, 348776), (350035, 348777), (350035, 348779), (350035, 348780), (350035, 349234), (350035, 349237), (350035, 3500
   34)]
   Adjacency List:
   28991: [15238, 15239, 25919, 28914, 32343]
   53386: [2887, 37358, 37359, 40321, 40323, 53383, 66990, 66992, 66993, 103377]
   153453: [64239, 68657, 68658, 85180, 112469, 112472, 123391, 304048, 304049, 304050]
   195302: [183637, 183810, 249887]
   350035: [304485, 348774, 348775, 348776, 348777, 348779, 348780, 349234, 349237, 350034]
   15238: 1
   15239: 1
   25919: 1
   28914: 1
   28991: 0
   32343: 1
   2887: 1
   37358: 1
   37359: 1
   40321: 1
   40323: 1
   53383: 1
   53386: 0
   66990: 1
   66992: 1
   66993: 1
   103377: 1
   64239: 1
   68657: 1
   68658: 1
   85180: 1
   112469: 1
   112472: 1
   123391: 1
   153453: 0
   304048: 1
   304049: 1
   304050: 1
   183637: 1
   183810: 1
   195302: 0
   249887: 1
   304485: 1
   348774: 1
   348775: 1
   348776: 1
   348777: 1
   348779: 1
   348780: 1
   349234: 1
   349237: 1
   350034: 1
   350035: 0
   Average distance from original nodes: 0.88372093
```

jb



```
     Finished dev [unoptimized + debuginfo] target(s) in 0.07s
      Running `/Users/vincentsun/Desktop/final3/target/debug/fin`
Source nodes and their connections:
Nodes: [59056, 193105, 292343, 336647, 344591]
Node Connections: [(59056, 57934), (59056, 57935), (59056, 57937), (59056, 57938), (59056, 57939), (59056, 57940), (59056, 59055), (59056, 59057), (59056, 126208), (5
9056, 126210), (193105, 85314), (193105, 85319), (193105, 85320), (193105, 85321), (193105, 86598), (193105, 117929), (193105, 159400), (193105, 172536), (193105, 225
049), (193105, 313153), (292343, 51330), (292343, 72908), (292343, 144221), (292343, 170218), (292343, 170220), (292343, 213274), (292343, 213275), (292343, 241242),
(292343, 292341), (292343, 292342), (336647, 174428), (336647, 285286), (336647, 286788), (336647, 315594), (336647, 315597), (336647, 320393), (336647, 336645), (336
647, 336646), (336647, 336648), (336647, 343147), (344591, 308491), (344591, 344582), (344591, 344583), (344591, 344584), (344591, 344585), (344591, 344586), (344591,
 344587), (344591, 344588), (344591, 344589), (344591, 344590)]
Adjacency List:
59056: [57934, 57935, 57937, 57938, 57939, 57940, 59055, 59057, 126208, 126210]
193105: [85314, 85319, 85320, 85321, 86598, 117929, 159400, 172536, 225049, 313153]
292343: [51330, 72908, 144221, 170218, 170220, 213274, 213275, 241242, 292341, 292342]
336647: [174428, 285286, 286788, 315594, 315597, 320393, 336645, 336646, 336648, 343147]
344591: [308491, 344582, 344583, 344584, 344585, 344586, 344587, 344588, 344589, 344590]
57934: 1
57935: 1
57937: 1
57938: 1
57939: 1
57940: 1
59055: 1
59056: 0
59057: 1
126208: 1
126210: 1
85314: 1
85319: 1
85320: 1
85321: 1
86598: 1
117929: 1
159400: 1
172536: 1
193105: 0
225049: 1
313153: 1
51330: 1
72908: 1
144221: 1
170218: 1
170220: 1
213274: 1
213275: 1
241242: 1
292341: 1
292342: 1
292343: 0
174428: 1
285286: 1
286788: 1
315594: 1
315597: 1
320393: 1
336645: 1
336646: 1
336647: 0
336648: 1
343147: 1
308491: 1
344582: 1
344583: 1
344584: 1
344585: 1
344586: 1
344587: 1
344588: 1
344589: 1
344590: 1
344591: 0
Average distance from original nodes: 0.90909094
```