

DS210 Final Project Writeup

****I went to Office Hours to work on this project.**

In my project, I am using a directed graph of Gnutella's data (a peer to peer sharing network) from August 2002 with 36682 nodes and 88328 edges. A directed graph means that the two numbers on each line suggest that there is a connection from the first node to the other. For example, (3, 66) means that the graph edge is from $3 \rightarrow 66$. Let's say there is one on another line that is (66, 3), and then we'll know that these two nodes are bi-directional.

Within my code, I decided to perform my analysis on the data using functions like `main`, `read_file`, `make_rand_sample`, `unique_nodes`, `adjacency_list` (in `main.rs`), `bfs_dist` (in `bfs.rs`), and `mean`, `max`, and `mode` (in `stats_fc.rs`).

- **Main functions**

The main function reads the Gnutella data from a file, creates a random sample of 20,000 nodes and edges, and generates a unique set of nodes from the sampled data from the "p2p-Gnutella30.txt" file. It then constructs an adjacency list for the unique nodes and computes the shortest path distances between all pairs of nodes using breadth-first search, storing the results in a hashmap. Subsequently, it iterates over the hashmap to print the distances between nodes, excluding -1 and 0 distances. Finally, it calculates and prints the average, maximum, and mode of the distances between nodes using statistical functions from the `stats_fc` module. Additionally, there are utility functions for reading the file, creating a random sample, finding unique nodes, and constructing the adjacency list.

read_file: This function reads the file that I inputted (i.e. the main Gnutella file), and parses each line of the file as a tuple of usize values (representing edges between nodes). It ends by returning a vector containing the parsed edges.

make_rand_sample: This takes a random sample of n nodes and edges from the input graph, and uses the `choose_multiple` function from the `rand` crate to select random elements. It finally returns the sampled nodes and edges as a vector.

unique_nodes: The following function finds unique nodes from the given vector of edges, inserts each unique one into a `HashSet`, and returns the set.

adjacency_list: With the use of this function, my project constructs an adjacency list from the given edges and set of unique nodes. It maps each node to its index for efficient indexing, and iterates over the edges and populates the adjacency list accordingly. Finally, it prints out non-empty adjacency lists for each vertex, returning the constructed adjacency list.

- **Modules**

Breadth first search (BFS): The function in this module implements breadth-first search (BFS) to determine the distances between nodes in a graph represented as an adjacency list. Given a starting node index `start` and the graph, the function initializes a distance vector with -1 values indicating unreachable nodes. It also initializes an empty visited vector to keep track of visited nodes and a queue to store nodes to visit. Starting from the start node, the function iteratively explores neighboring nodes, updating their distances if they are reachable and not yet visited. If a shorter path to a node is found, or if the node was unreachable before, its distance is updated. The function continues until all reachable nodes are visited, updating their distances accordingly. Finally, it returns the distance vector, representing the shortest distances from the start node to all other nodes in the graph.

Statistical Functions:

- **mean:** This function calculates the mean (average) of a dataset represented as a slice of `f64` values. It first checks if the input data is empty, returning `None` if so. Otherwise, it computes the sum of all elements in the dataset and divides it by the total number of values to obtain the mean, returned as an `Option<f64>`.
- **max:** The max function finds the maximum number of connections node in a dataset represented as a `HashMap<usize, Vec<isize>>`. If the input data is empty, it returns `None`, else it iterates over the entries in the hashmap to find the node with the maximum number of connections. It returns the node index with the maximum number of connections as an `Option<usize>`.
- **mode:** The mode function calculates the mode (most common value) of a dataset represented as a slice of `f64` values. If the input data is empty, it returns `None`, else it creates a hashmap to store the frequency of each value in the dataset. It iterates over the values, updating the frequency map. Then, it identifies the value with the highest frequency and returns it as the mode value, wrapped in an `Option<f64>`.

- **Tests**

For my tests, I generated my own data file with nodes (11) and edges (24) called “node_test_file.txt,” which is either directly used to test the output of the functions as a file, or by hard coding the same data.

Unique nodes: This tests whether the `unique_nodes` function correctly finds the unique nodes from a test file. It compares the number of unique nodes found by the function to the expected value.

Max: Max tests whether the max function from the stats module correctly identifies the node with the most connections. It constructs a HashMap representing connections between nodes (hard-coded for testing), calls the max function to find the node with the highest number of connections, and compares the result to the expected value.

BFS: This test verifies the bfs_dist function's correctness by using a manually constructed graph (bfs_distance). It then calculates the shortest distance from node 0 to node 2 and asserts that it equals 2. If the assertion holds, it confirms the accurate computation of the shortest path distance.

- **Output**

** Please note that since I am randomly sampling my data of 20000 edges every time it is run, the outputs provided below won't be the same each time. Only the test functions have been hard-coded to test the functionality of certain functions such as unique nodes, the max statistical function, and BFS. Also, since the data I am sampling is very big, the terminal doesn't store all the data it prints for me to screenshot (and it's a lot), so I have put a portion of my results below.

- Up first is a screenshot of vertices with the nodes they're connected to listed as "Vertex (node itself): [the node/s it's connected to]."
- Next, is a screenshot of a portion of the results of the breadth-first search algorithm that was run on the adjacency list. This output is printed as "Node (itself) is (distance) away from Node (connected node)."
- The screenshot after includes the output of my statistical functions (again, will be different each time it's printed). First is the average number of connections that the sample of nodes has which is approximately 4.86 below. Then, I printed out the maximum connections node, as more connections mean that it is more popular (2835 below). And finally is the mode, which looks at the BFS output and finds the most common degree of connectivity between the nodes, which is 1.0. Generally this is a very intuitive idea, and using this function helps prove the code's testing validity.
- And finally, I have included a screenshot of the tests in my project (already detailed above), to show that they pass.

```

Vertex 18198: [12265, 268]
Vertex 18199: [3616, 5896]
Vertex 18200: [13588, 11666, 15915]
Vertex 18205: [6513, 6758, 15344, 4491]
Vertex 18206: [18124, 12227]
Vertex 18210: [10478, 3870]
Vertex 18215: [1568]
Vertex 18222: [10102, 15278, 891]
Vertex 18223: [14703]
Vertex 18226: [5204]
Vertex 18229: [4160, 7460, 2285, 16643]
Vertex 18231: [3838, 3922, 11926]
Vertex 18232: [18280, 3723]
Vertex 18233: [13635, 6281]
Vertex 18236: [13972]
Vertex 18237: [2958, 3871, 12433, 6069, 12947]
Vertex 18238: [11445]
Vertex 18239: [16032, 14714, 16297]
Vertex 18240: [14574, 11052, 17122]
Vertex 18241: [2008, 13448, 1984]
Vertex 18243: [10018, 8073]
Vertex 18244: [18265, 1759, 6373, 10895]
Vertex 18245: [6028]
Vertex 18247: [13186, 3103, 18228]
Vertex 18251: [16215, 17880, 1221, 2461, 1706, 4605]
Vertex 18252: [11390, 4636, 16774]
Vertex 18253: [2676, 16549]
Vertex 18254: [18305, 9008, 7165]
Vertex 18255: [12620, 4738, 12219]
Vertex 18261: [11015]
Vertex 18264: [10934, 15767, 16180, 473]
Vertex 18266: [4635, 5889]
Vertex 18267: [4390, 3864, 12598]
Vertex 18268: [7693, 18203, 1517, 10393, 3716]
Vertex 18269: [1457, 6902]
Vertex 18271: [849, 2698, 710]
Vertex 18273: [9905]
Vertex 18276: [12057, 12848, 4798, 5089, 7127]
Vertex 18277: [4690, 3255]

```

```

Node 2763 is 3 away from Node 15328
Node 2763 is 10 away from Node 15652
Node 2763 is 2 away from Node 15688
Node 2763 is 8 away from Node 16054
Node 2763 is 3 away from Node 16661
Node 2763 is 3 away from Node 16888
Node 2763 is 6 away from Node 17254
Node 627:
Node 627 is 1 away from Node 6404
Node 627 is 1 away from Node 14012
Node 627 is 1 away from Node 15629
Node 627 is 1 away from Node 16918
Node 8722:
Node 8722 is 8 away from Node 41
Node 8722 is 3 away from Node 119
Node 8722 is 9 away from Node 805
Node 8722 is 9 away from Node 879
Node 8722 is 2 away from Node 998
Node 8722 is 1 away from Node 1117
Node 8722 is 2 away from Node 1189
Node 8722 is 2 away from Node 1794
Node 8722 is 3 away from Node 1905
Node 8722 is 3 away from Node 1946
Node 8722 is 3 away from Node 2664
Node 8722 is 10 away from Node 2719
Node 8722 is 6 away from Node 3827
Node 8722 is 8 away from Node 4126
Node 8722 is 3 away from Node 4185
Node 8722 is 8 away from Node 4460
Node 8722 is 9 away from Node 5363
Node 8722 is 8 away from Node 7276
Node 8722 is 3 away from Node 7636
Node 8722 is 6 away from Node 8669
Node 8722 is 2 away from Node 9024
Node 8722 is 5 away from Node 9112
Node 8722 is 7 away from Node 9444
Node 8722 is 1 away from Node 10324
Node 8722 is 7 away from Node 11116
Node 8722 is 7 away from Node 11654

```

The average value of node connections is: 4.86163097962297

The maximum number of node connections (showing which ones are most popular) are: 2835

The most common node connections distance value is: 1.0

Finished test [unoptimized + debuginfo] target(s) in 1.34s

Running unittests src\main.rs (target\debug\deps\Project-e60b83151945242d.exe)

running 3 tests

test test_bfs_dist ... **ok**

test test_unique_nodes ... **ok**

test test_max ... **ok**

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

• Citation

Since I was having difficulties disabling the unreachable (-1) and self connection (0) nodes from being in my printed output in the BFS function, I used chat-gpt to help me as cited below:

ChatGPT

Sure, to exclude printing the distances with value -1 in the `bfs_distance_map`, you can modify the printing loop as follows:

```
rust Copy code

for (node, distances) in &bfs_distance_map {
    println!("Node {}: ", node);
    for (other_node, distance) in distances.iter().enumerate() {
        if *distance != -1 && *distance != 0 { // Exclude -1 and 0 distances
            println!("    Node {} is {} away from Node {}", node, distance, other_node);
        }
    }
}
```

This modification ensures that only distances greater than 0 are printed for each node. If the distance is -1 (indicating unreachable), it will not be printed. Let me know if you need further assistance!