Zach Gentile - DS 210 Final Report

Dataset: http://konect.cc/networks/subelj_euroroad/

I picked an undirected dataset representing E-roads in Europe. E-roads are major highways that connect countries together, much like US interstate highways such as I-95. The nodes represent cities, and the edges represent cities that the nodes are connected to by E-roads. Initially I was planning on using a US road network database, but over one million nodes, it was not feasible to perform the operations that I wanted to. However, since this dataset is quite tame in size (1174 nodes), the runtime is only around 40 seconds with cargo run –release.

My goal with this project is to analyze the distances and degrees of separation between nodes, or in this context, the minimum amount of E-roads it takes to travel from one city to another. I created various algorithms that helped perform this task or make this task more efficient.

In the middle of doing the project, I noticed that using my implementation of shortest distance calculations would be more efficient if I reduced the sparsity of my graph. So, I created the data_processing module. The data_processing module contains three functions that work in succession of each other to reduce the sparsity of a graph. The first function is merge_and_sort_columns, which combines all unique node and edge values, and then sorts them from least to greatest. The second function is reassign_indexes, which takes every node and edge value and reassigns it to the rank of its value compared to all unique values in the graph. The smallest number will become zero, the second smallest value becomes one, etc. Lastly, the function combine_vectors simply recombines the vectors of nodes and edges so graph analysis can be carried out. This is all done to make sure there are no gaps between node values, which makes my later functions run faster and print more readable output.

For graph analysis of separation, I made a module called degrees_separation. This module contains four functions. The first function is adj_list, which creates an undirected adjacency list out of given columns of nodes and edges. The second function is find_distance, which calculates the shortest distance between two given nodes. It uses a vecDeque and a vector of nodes visited to determine where each node visits and the optimal path (or amount of visits needed to reach a destination) as an implementation of breadth first search. If the function determines that a given node is not connected to another, it will return None. The third function is calculate_all_distances, which calculates the distance between every possible unordered combination of nodes, if there is one. It also organizes these distances by placing every pair into a vector corresponding to their separation from each other. Additionally, it returns the number of connections and the number of invalid connections. I am considering invalid connections to be self connections and pairs that are not connected in any way. The final function in this module is separation_distribution, which calculates the percentage of the total connections that have a certain degree of separation. In my code, I use this to print the entire distribution of separation, and print the percentage of pairs that can reach each other within six degrees of separation, and also those that can reach each other through a maximum of 20 degrees of separation.

The third and final module, statistics, serves to calculate the mean and standard deviation of the distribution of separation. An interesting observation is that it takes a significant amount of time to calculate the standard deviation as compared to the mean. This is likely because you have to map to every point its squared difference from the mean, being much more computationally expensive than just finding the mean itself. In addition, using standard rust functions, my code outputs the maximum degrees of separation between two nodes as well as the number of valid and invalid connections in the graph.

Here is a selection of the important output of my code: (note: more things such as the adjacency list and all connections can be printed, but it is an extremely large amount of data to

print in the terminal. The print statements are there in my code and can be commented out before running.)

```
    Finished release [optimized] target(s) in 0.57s
     Running `target\release\finalstep.exe`
The connections with 1 degrees of separation are 0.2615543% of the valid connections.
The connections with 2 degrees of separation are 0.49078372% of the valid connections.
The connections with 3 degrees of separation are 0.85208327% of the valid connections.
The connections with 4 degrees of separation are 1.3146058% of the valid connections.
The connections with 5 degrees of separation are 1.8122239% of the valid connections.
The connections with 6 degrees of separation are 2.3211095% of the valid connections.
The connections with 7 degrees of separation are 2.8482819% of the valid connections.
The connections with 8 degrees of separation are 3.3449764% of the valid connections.
The connections with 9 degrees of separation are 3.8490596% of the valid connections.
The connections with 10 degrees of separation are 4.266512% of the valid connections.
The connections with 11 degrees of separation are 4.5110726% of the valid connections.
The connections with 12 degrees of separation are 4.6368628% of the valid connections.
The connections with 13 degrees of separation are 4.6233783% of the valid connections.
The connections with 14 degrees of separation are 4.536748% of the valid connections.
The connections with 15 degrees of separation are 4.4146523% of the valid connections.
The connections with 16 degrees of separation are 4.2798114% of the valid connections.
The connections with 17 degrees of separation are 4.131302% of the valid connections.
The connections with 18 degrees of separation are 3.9903653% of the valid connections.
The connections with 19 degrees of separation are 3.841117% of the valid connections.
The connections with 20 degrees of separation are 3.6789384% of the valid connections.
The connections with 21 degrees of separation are 3.4862823% of the valid connections.
The connections with 22 degrees of separation are 3.2561293% of the valid connections.
The connections with 23 degrees of separation are 3.001594% of the valid connections.
The connections with 24 degrees of separation are 2.7284029% of the valid connections.
The connections with 25 degrees of separation are 2.4360013% of the valid connections.
The connections with 26 degrees of separation are 2.1701987% of the valid connections.
The connections with 27 degrees of separation are 1.9448483% of the valid connections.
The connections with 28 degrees of separation are 1.7492367% of the valid connections.
The connections with 29 degrees of separation are 1.6066378% of the valid connections.
The connections with 30 degrees of separation are 1.4991347% of the valid connections.
The connections with 31 degrees of separation are 1.4062238% of the valid connections.
The connections with 32 degrees of separation are 1.3158989% of the valid connections.
The connections with 33 degrees of separation are 1.2264975% of the valid connections.
The connections with 34 degrees of separation are 1.1252745% of the valid connections.
The connections with 35 degrees of separation are 1.0098286% of the valid connections.
The connections with 36 degrees of separation are 0.88754827% of the valid connections.
The connections with 37 degrees of separation are 0.7824463% of the valid connections.
The connections with 38 degrees of separation are 0.6889813% of the valid connections.
The connections with 39 degrees of separation are 0.5921914% of the valid connections.
The connections with 40 degrees of separation are 0.5064844% of the valid connections.
The connections with 41 degrees of separation are 0.44091108% of the valid connections.
The connections with 42 degrees of separation are 0.3768155% of the valid connections.
The connections with 43 degrees of separation are 0.3151212% of the valid connections.
The connections with 44 degrees of separation are 0.2685734% of the valid connections.
The connections with 45 degrees of separation are 0.22571987% of the valid connections.
The connections with 46 degrees of separation are 0.18064979% of the valid connections.
The connections with 47 degrees of separation are 0.14056696% of the valid connections.
The connections with 48 degrees of separation are 0.10879624% of the valid connections.
The connections with 49 degrees of separation are 0.0821975% of the valid connections.
The connections with 50 degrees of separation are 0.06206373% of the valid connections.
The connections with 51 degrees of separation are 0.044700664% of the valid connections.
The connections with 52 degrees of separation are 0.032324858% of the valid connections.
The connections with 53 degrees of separation are 0.02438218% of the valid connections.
The connections with 54 degrees of separation are 0.017917208% of the valid connections.
The connections with 55 degrees of separation are 0.013299371% of the valid connections.
The connections with 56 degrees of separation are 0.0092356745% of the valid connections.
The connections with 57 degrees of separation are 0.0059108315% of the valid connections.
The connections with 58 degrees of separation are 0.0038789832% of the valid connections.
The connections with 59 degrees of separation are 0.0024012753% of the valid connections.
The connections with 60 degrees of separation are 0.0014777079% of the valid connections.
The connections with 61 degrees of separation are 0.00073885394% of the valid connections.
The connections with 62 degrees of separation are 0.00018471348% of the valid connections.
-----------------
Invalid - (no connection or self connection): 147173
Valid connections: 541378
Mean separation: 18.372121694541885
Max distance: 62
Standard deviation of separation: 13.4663353841851
The percentage of connected nodes that can be reached within six degrees of separation is 7.0523605%
The percentage of connected nodes that can be reached within twenty degrees of separation is 64.00544%
-----------------
```

Only a small portion, 7.05%, of the combinations of cities can reach each other through 6 different E-roads or less. A larger portion, 64%, of the combinations of cities can reach each other through 20 different E-roads or less.

The main interesting thing that I discovered while doing this project is that the average and maximum degrees of separation for a road network like one tends to be much larger than another type of undirected graph such as a social media network between friends. I hypothesize that this is because it is much easier for roads to have a large amount of connections (i.e. connect via 50 different roads) than it is for someone to have a long chain of 50 mutual friends to finally connect with someone else. Note that each level of separation from 6 to 26 contain at least 2% of the total connections. The distribution of separation is skewed right, with each degree of separation from 36 to 62 having less than 1% of the pairs of cities.

Resources: Lecture notes, homework, homework solutions, CAs, Rust official documentation

Chatgpt use:

is there a built in binary search in rust

Yes, Rust provides a built-in binary search method called `binary_search` for slices, which is implemented using the binary search algorithm. You can use this method instead of implementing your own binary search algorithm. Here's an example of how to use it:

rust Copy code

```rust
fn main() {
    let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
    let search_value = 6;

    // Perform binary search on the sorted array
    let result = arr.binary_search(&search_value);

    match result {
        Ok(index) => {
            println!("Found {} at index {}", search_value, index);
        }
        Err(_) => {
            println!("{} not found in the array", search_value);
        }
    }
```

Regenerate response

Used to help implement binary search in reassign_indexes function

when to use a hashset in rust

A `HashSet` in Rust is a data structure that stores a collection of unique elements in an unordered manner. You would typically use a `HashSet` when you need to efficiently store and query a set of unique values, and you don't require any specific ordering of the elements.

Here are a few scenarios where a `HashSet` can be useful:

1. Removing duplicates: If you have a collection of elements and you want to eliminate duplicates, you can use a `HashSet` to store the unique values. You can insert each element into the `HashSet`, and it will automatically handle deduplication.

Used to remove duplicates to find unique values in a vector in merge_and_sort_columns function