Nico Lehtinen

WRITTEN REPORT:

Project Overview: The question that I wanted to answer was, based on the thousands of teams present in professional European football (soccer) and the thousands of matches played between a variety of combinations of teams, which two teams would be the most isolated from each other? In other words, have the least indirect interaction through matches played with other teams.

Dataset: Off of Kaggle, the link is European Soccer Data

DATA PREPROCESSING: I loaded the data into rust by uploading the file in csv format to the working src directory, and then using the csv crate in order to read it.

CLEANING AND PREPROCESSING: There was definitely cleaning involved. Out of all the columns in the data set, I took specific note of two (Opponent and Team) in order to create a HashMap to acknowledge the match. Then, the data was used in accordance with HashMap to assign unique IDs to each team, and to assign weights based on the frequency of same matchups occurring.

CODE STRUCTURE:

Module 1: matchtree.rs: The purpose of matchtree.rs is to hold the logic that creates the undirected weighted graph from the data points held in the European Soccer csv file, as well as read the file. It is separated because the logic is purely to do with the creation of the undirected weighted graph out of the data in the csv file.

Module 2: isolatedteams.rs: The purpose of isolatedteams.rs is to hold the logic for the dfs, for finding the farthest teams and to provide a comparison at the end that allows to see the actual match distribution per team. This module holds the logic functions that involve the output from the undirected weighted graph created in matchtree.rs, so the separation splits the two stages.

Module 3: main.rs: The purpose of main.rs is simple: its where every function is called and where the tests are held.

The functions are as follow:

Pub fn add_undirected_edges() simply have edges created between the adjacency lists u and v.

Pub fn sort_graph_lists() iterates through every adjacency list in self.outedges based on ID in ascending order

Pub fn create_undirected() combines the two previous functions' utilizations in order to create an undirected graph

Pub fn data_read() is a csv reader that reads the csv file. Using Vec and HashMap. Each row is iterated through and the 3 and 6th index (4th and 7th columns that correspond with Team and Opponent names) are taken, where the team names are then assigned IDs that help identify them and assign weights based duplicate matches with other IDd team names. The resulting HashMap creates the edges, which are then used in create_undirected() to create the undirected graph

Next, in isolatedteams.rs

dfs_weighted() performs a depth first search on the undirected graph, where the weights are inverted by 1/w in order to have the relationship of playing more matches with a team would mean you are more closely related with that team, hence the distance is shorter

Pub fn final_dfs() does two dfs searches (similar to homework), where the farthest point is found by comparing values using partial cmp. At the end, we iterate through distance 2 in order to find the maximum distance, and the second iteration collects the indices, which are used to identity the actual teams that are the most isolated by name (so they are identified by team name not index)

Pub fn match_num_comparisons() iterates through each teams adjacency list to find which team has the least and most matches played, as well as the average matches per team.

main(): simple calls to functions in logical progression.

WORKFLOW: To phrase it simply, the matchtree.rs starts through the data_read function, where the logic brings it to the create_undirected() function that actually creates the graph and utilizes the remainder of the functions in the module to achieve this. The undirected graph is used in the isolatedteams.rs module to be used in the DFS functions and for team comparisons, where the final functions are defined and can then be called in the main.rs.

Tests:

```
    Running unittests src/main.rs
(target/debug/deps/DS_210_Final_Project-0cba5967f21e6e76)

running 2 tests
test test_graph_creation ... ok
test test_path ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

The first test checks for the fact that the graphs are being initialized correctly, where the number of vertices need to match expected size based on teams count.Also confirms that match frequencies are correct, which are crucial for doing the calculations in the isolatedteams.rs module.

The second test makes sure that the longest distance from the DFS functions is actually running correctly. It verifies an isolated team is detected, and that it has a positive maximum path distance.

RESULTS:

```
Reading CSV: EuropeanData.csv
1966 teams.
Longest Infrequent Path (inverted weight): 96.6260384926393
Most isolated teams from each other:
US OBERSCHAEFFOLHEIM
ROSSELANGE VITRY
------Team Match Comparison-------
Team w/ fewest matches: 'NOVELDA' (1 matches)
Team w/ most matches: 'CHELSEA' (118 matches)
Average # matches per team: 11.54
```

We can see that the teams that are at the greatest distance from one another (as in the most unique matches in between them in European football) are US Oberschaeffolheim and Rosselange Vitry. Which is something that I find both interesting (because they are both teams in France so them having the least indirect connection is very rare), but also unsurprising considering they are both very small clubs. It might raise some questions about the lack of friendly matches

for smaller tier clubs in France, because most countries hold friendlies against higher tier clubs that would likely boost their connection to other clubs significantly despite very low stature.

Usage Instructions:

Building and running the code would follow the creation of the modules and functions that I already described. There are some key points that need to be hit. 1. We need a function that can read and clean the data to only contain the team and opponent names, because we want to have teams as nodes and matches as edges between them. 2. Utilize DFS and inverted weights in order to not only find the clubs with the least connection between them (# unique club matches between them), but also that are based on frequency of duplicate matches (which makes the whole more accurate).

Command Line: Other than cargo run and cargo test, the terminal is not used.

The expected Run Time is < 1 second, it is very quick.