DS210 Project: The Examination of Links Between Soccer Players

Johnathan Finizio

Introduction

Soccer players are interconnected with one another based on who they play with on both the club and international level. This connection is translated in the video game FIFA based on the chemistry component of creating a team in the mode Ultimate Team. Players link together based on similarities with their team, nation or league. Oftentimes, two very talented players do not have chemistry with one another, so other players must be acquired in order to establish a well connected team. The goal of this program is to find the shortest connections between two players and provide the user with the computed path. The data used to run this program contains player information from the EA Sports game FIFA 23, meaning all player information is from one year ago.

The Program

The program computes the distance between two user inputted players, and outputs the players in the path.. Players are modeled by a struct, "Player", which contains their name, position, team, nationality, and ratings from the video game FIFA 23. The module "players.rs" contains all of the associated functions and code for the struct. The "Attributes" trait for a player is able to get their nationality or team. A player is able to be created using the function "create" associated with the player struct. While this function is not directly used, it is included in case a player was missing in the data set. Lastly in the module, there is a function called "find_player". This function examines the user's input and finds the inputted player in the list of nodes. If the name is not found in the nodes list, Levenshtein distance is used to see if there is a player of similar name that the user may have meant to input (in case of typos or missed formatting). Levenshtein distance examines the number of changes to make two strings equal. A low Levenshtein distance will point towards the intended player by the user. Player's names in the dataset typically follow the structure "First Initial. Last Name" (Ex. L. Messi), unless the player only goes by one name (Ex. Marquinhos). So, if the Levenshtein distance is less than or equal to 3, the function will print a player that the user may have meant to input. Here is the crate I used to compute Levenshtein distance: strsim - 0.10.0 https://crates.io/crates/strsim/0.10.0

The next component of the program is reading the data, creating the graph, and computing a BFS, which is executed in the module "bfs.rs". First, the function "read_csv_file" takes in a csv file with columns that correspond to the parameters of the "Player" struct. The serde and csv crates are used to read the csv file, and each record is deserialized to be represented as an object of the "Player" struct. Next, the function "edges" computes the neighbors for each node. The function starts by initializing an empty vector of vectors called edges. The function iterates through all pairs of players, and tests if they either have the same team of the same nationality. If a similarity occurs, the index of the second player is pushed onto the vector in edges that corresponds with the first player. The function then returns all the

indexes of a player's neighbors in the vector indexed to that player. The function reconstruct_path is essential for showing the steps of the bfs algorithm. The function takes in the vector of nodes, the vector of all parent nodes for a BFS, and the indexes for the start and end of the path. The function initializes a vector called "path" which keeps track of all parent nodes, and a variable "current" that keeps track of what node is being examined. Then, if the current node has a parent, that parent is pushed onto the path vector. If the current node matches the start index, the loop will break. The vector "path" is then reversed to output the correct direction of the desired path. Lastly, the function bfs_with_path computes a breadth first search on the graph given a starting point. The function initializes an empty vector called distances to store all of the distances to the starting point, and an empty vector called parents to store all parent nodes. A queue is also initialized to keep track of the nodes to visit. For each neighbor (u) of the current node (v), the function checks if node (u) has been visited yet. If not, it sets the distance of node (u) to be 1 more than the distance to node (v). If the node (u) is the same as the end_index, the search will stop and reconstruc_path is called to return the path between the two players.

The third component of the graph are the two tests. The tests ensure that the paths are being computed correctly, by comparing the length of the path to the length it should be. One test is designed to check if intermediate steps of the path are calculated correctly, and the other ensures that neighbors are still calculated correctly. Both test cases are passed when the program is run.

The last component of the program is the main. The main function puts all the pieces together and allows the user to interact with the program. First, the program reads the csv file, creates the list of nodes and neighbors, and checks all of the tests. Then, the program will ask the user to input a player's name. The program then calls the find_player function to determine if the inputted name is in the list, close to a name in the list, or not in the scope of the list. If a correct player name is not inputted, the loop will not break. The same process is then executed for the second player. Once two players are inputted correctly, the program computes a BFS based on the first player, and finds the distance between the two players based on the distance at the index of the second player. Then, the program prints the result of the calculation.

Running the Program

I. Run the program with "cargo run fifa_data.csv"
II. The program will output "Please wait…" while it reads the graph. Once done, it will prompt the user with "Enter a player name:", from which the user can type in the player of their choice. If the player is found, the program will move on. If not, the user must enter a valid name. This process will be repeated for the second player as well.
III. Once two players are correctly inputted, the resulting path between the two players will be printed.