Project Documentation and Report

Project Overview

The goal of this project is to analyze NCAA March Madness team performance by building a similarity-based graph and identifying structural insights using centrality and subgraph analysis. The aim is to determine clusters of statistically similar teams and highlight key teams based on graph metrics such as closeness centrality and densest subgraph inclusion. The dataset used is a CSV file titled "DEV _ March Madness.csv," which contains college basketball team statistics across multiple seasons and can be found at https://www.kaggle.com/datasets/jonathanpilafas/2024-march-madness-statistical-analysis.

Data Processing

The dataset was loaded into Rust using the csv crate. Each record in the file corresponds to a team-season combination and includes numeric performance statistics. We filtered the dataset to exclude non-numeric or irrelevant fields. The data was structured into a custom team struct, and rows with missing names or all-zero stats were excluded. Additionally, teams that participated in the NIT or didn't make the playoffs were excluded, as we only wanted to focus on teams that participated in March Madness. Some columns had only one unique value or were incomplete; therefore, we eliminated them. This step ensured that we only analyzed meaningful records.

Code Structure

The codebase is modularized into five key modules: models, io_utils, similarity, graph_utils, and tests.

Models.rs

The models module defines the Team struct, which holds the name, season, and performance statistics for each team:name:

- String — the team's name and season.season:
- String — the season string (e.g., "2023").stats:
- Vec<f64> — numeric performance metrics.

io_utils.rs

Handles all input/output functionality. It reads the CSV file and returns a list of Team structs after cleaning. It also writes the final string output to a text file.
read_csv(path: &str) -> Vec<Team>:

Its purpose is to load and parse team data from a CSV file. It accepts the path to the CSV as its input, and outputs a vector of cleaned Team structs. It does this by using csv::Reader to iterate through records, filters out non-numeric fields, parses float values, and ensures the team has non-zero stats.
write_to_file(path: &str, content: &str):
Its purpose is to save analysis results. It accepts a File path and content as its input and outputs a file with the results.

similarity.rs

Defines the core similarity metric used to compare teams.
cosine_similarity(a: &[f64], b: &[f64]) -> f64:
Its purpose is to compute cosine similarity between two stat vectors. It accepts two slices of f64 as input and outputs the Similarity score in [0.0, 1.0]. It does this by computing the dot product and normalizing by the L2 norms of each vector.

graph_utils.rs

Builds and analyzes the graph of team similarities.
build_graph(teams: &[Team], threshold: f64) -> (UnGraph<String, f64>, HashMap<String, NodeIndex>)
Its purpose is to create a graph where nodes are teams and edges connect similar teams. It accepts a list of teams and a similarity threshold as inputs, and outputs an undirected graph and a mapping from team name to node index. It does this by iterating over all team pairs with tuple_combinations and adding edges when the cosine similarity exceeds the threshold.
compute_closeness_centrality(graph: &UnGraph<String, f64>) -> HashMap<NodeIndex, f64>
Its purpose is to compute closeness centrality for all nodes. It accepts a graph as input and outputs a mapping of node index to centrality score. It does this by using BFS-style traversal to calculate the sum of the shortest distances to all reachable nodes.
densest_subgraph(graph: &UnGraph<String, f64>) -> (UnGraph<String, f64>, f64)
Its purpose is to identify the subgraph with the highest edge-to-node ratio. It accepts a graph as input and outputs a Subgraph and its density. It does this by iteratively removing the lowest-degree node and tracking the subgraph with the maximum density.

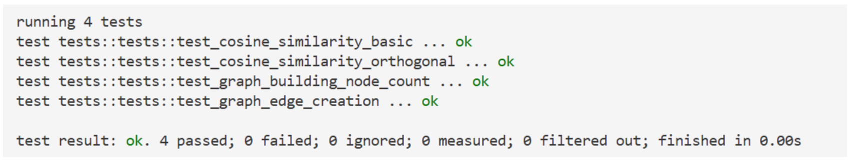
Main.rs

The main workflow starts in main.rs, where we call read_csv to load the data, followed by build_graph to construct a similarity graph with a threshold of 0.75. We then compute closeness centrality using compute_closeness_centrality and extract the densest subgraph using densest_subgraph. Finally, the output is formatted and saved using write_to_file. Modules are imported using mod and use, and each function is commented to provide clarity on its role and implementation.

Tests

Tests are located in tests.rs, behind the #[cfg(test)] annotation. We validate the behavior of cosine_similarity in simple and orthogonal vector cases to ensure correct implementation. Graph-building logic is also tested to verify that nodes and edges are correctly created based on the similarity threshold. These tests ensure core functionality behaves as expected and help catch future regressions. The test and their purpose can be found below.

- test_cosine_similarity_basic — checks perfect match.
- test_cosine_similarity_orthogonal — checks zero similarity.
- test_graph_building_node_count — ensures correct number of nodes.
- test_graph_edge_creation — verifies edges form when appropriate.



```
running 4 tests
test tests::tests::test_cosine_similarity_basic ... ok
test tests::tests::test_cosine_similarity_orthogonal ... ok
test tests::tests::test_graph_building_node_count ... ok
test tests::tests::test_graph_edge_creation ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Results

Program out:
Graph has 1534 nodes and 691885 edges
Average node degree: 902.07

Top 5 teams by closeness centrality:
Creighton Bluejays (2007): 0.001
Ohio State Buckeyes (2022): 0.001
Oregon Ducks (2013): 0.001
Louisiana Ragin' Cajuns (2014): 0.001
Michigan Wolverines (2022): 0.001

Densest subgraph: 1218 nodes, density = 497.823
Top 5 nodes in densest subgraph by degree:
Dayton Flyers (2014) (degree 1217)
Oregon Ducks (2024) (degree 1216)
Western Michigan Broncos (2004) (degree 1215)
Butler Bulldogs (2013) (degree 1215)
Oklahoma Sooners (2008) (degree 1215)

With a similarity threshold of 0.5, the generated graph is moderately dense, containing 1,534 nodes and 691,885 edges, with an average node degree of approximately 902. This indicates that teams are broadly similar to many others across the dataset, forming a tightly-knit graph. The top five teams by closeness centrality — including Creighton (2007), Ohio State (2022), and

Oregon (2013) — each have the highest average proximity to all other teams in terms of graph distance, although their centrality scores are still quite low (0.001), suggesting that relative differences are small due to the uniform density of the graph.

The densest subgraph extracted contains 1,218 nodes, with a density of approximately 497.82, meaning the average node in this subgraph connects to nearly 500 others. Notably, the top five teams in this subgraph by degree — such as Dayton (2014) and Oregon (2024) — each have over 1,215 connections, confirming their roles as highly similar to a wide variety of teams. This reinforces their strategic balance or statistical generality, making them strong representatives of mainstream or widely-shared team profiles across seasons.

Usage Instructions

To build and run the program, ensure you have Rust installed and navigate to the project folder. Then, use cargo run --release to execute the code. The input file must be placed in the root directory and named exactly as specified in the code (DEV _ March Madness.csv). No command-line arguments are required. The program completes execution in under a minute on most machines.