Code Output:

```
crc-dot1x-nat-10-239-168-137:degree_project aryasuresh$ cargo run --release
   Compiling degree_project v0.1.0 (/Users/aryasuresh/degree_project)
    Finished `release` profile [optimized] target(s) in 0.84s
     Running `target/release/degree_project`

graph loaded with 4039 nodes and 88234 edges

top 10 nodes by degree:
degree     node
----------------------
1045       107
792        351
755        352
547        1821
347        0
294        1490
291        2154
254        1373
245        1285
235        1149

bottom 10 nodes by degree:
degree     node
----------------------
1          4035
1          4025
1          4023
1          4016
1          4012
1          4010
1          3987
1          3978
1          3943
1          3868

degree histogram (top 10):
degree     count
----------------------
1045       1
792        1
755        1
547        1
347        1
294        1
291        1
254        1
245        1
235        1
```

```
degree statistics:
---------------------
average degree: 43.69
minimum degree: 1
maximum degree: 1045

top 5 closeness centrality scores:
node NodeIndex(107): 0.4597
node NodeIndex(58): 0.3974
node NodeIndex(350): 0.3948
node NodeIndex(371): 0.3939
node NodeIndex(351): 0.3936

bottom 5 closeness centrality scores:
node NodeIndex(1837): 0.1783
node NodeIndex(1946): 0.1783
node NodeIndex(1986): 0.1783
node NodeIndex(1889): 0.1783
node NodeIndex(1844): 0.1783

crc-dot1x-nat-10-239-168-137:degree_project aryasuresh$ cargo test
   Compiling degree_project v0.1.0 (/Users/aryasuresh/degree_project)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.30s
     Running unittests src/main.rs (target/debug/deps/degree_project-04bcbd707c585d4f)
running 3 tests
test tests::test_empty_graph ... ok
test tests::test_degree_distribution ... ok
test tests::test_graph_loading ... ok
test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Code Writeup:

For my final project, I wanted to analyze the structure of a social network graph dataset, specifically the facebook_combined.txt dataset. This dataset represents user connections as an undirected graph with 4039 nodes and 88234 edges. Each node represents a user, and each edge represents a friendship or connection between two users. My analysis focuses on identifying key nodes based on connectivity/degree and centrality, which helps reveal certain popular users or critical connections in the network.

For this project, I used the petgraph library, which specifically focuses on understanding the structure and properties of the graph. Overall, this project provides functionality to load a graph from a file, calculate the degree distribution of nodes, compute closeness centrality, and present the results in a clear and meaningful way.

I separated this project into three primary modules: graph_loader, degree_analysis, and centrality, which are each responsible for a different part of the analysis process.

The graph_loader module handles the loading of graph data from a file. It reads an edge list from a file where each line contains two integers representing a connection between two nodes. The function load_graph constructs an undirected graph using the petgraph library. Furthermore, load_graph(filename: &str) -> io::Result<UnGraph<u32, ()>>: reads a graph from the given filename and returns a UnGraph (an undirected graph). It also handles errors if the file cannot be read.

The degree_analysis module provides functions for calculating and analyzing the degree distribution of nodes, which is the number of edges connected to each node. It also includes functions for displaying the top and bottom nodes by degree, a histogram of degree counts, and statistics like average, minimum, and maximum degrees. The calculate_degree_distribution(graph: &UnGraph<u32, ()>) -> Vec<(usize, usize)>: calculates the degree distribution and returns a sorted list of node degrees. The display_top_bottom(degree_distribution: &Vec<(usize, usize)>, top_n: usize): displays the top and bottom top_n nodes by degree. Additionally, the display_degree_histogram(degree_distribution: &Vec<(usize, usize)>): displays a histogram of the degree counts for the top 10 degrees. And calculate_statistics(degree_distribution: &Vec<(usize, usize)>), calculates and displays some statistics, including average, minimum, and maximum degrees.

The centrality module focuses on calculating the closeness centrality of nodes in the graph. calculate_closeness_centrality(graph: &UnGraph<u32, ()>) -> HashMap<NodeIndex, f64>: calculates the closeness centrality for each node, which measures how quickly a node can reach all other nodes in the graph, and display_top_bottom_closeness(centrality: &HashMap<NodeIndex, f64>, top_n: usize): displays the top and bottom top_n nodes by closeness centrality, similar to the degree_analysis module.

The main function loads the graph from the specified file, facebook_combined.txt, calculates the degree distribution, displays the top and bottom nodes, shows a histogram of degrees, and computes degree statistics. It also calculates closeness centrality and displays the results. The graph is loaded using graph_loader::load_graph(filename), degree analysis is computed with calculate_degree_distribution and then displayed, as well as the histogram. Then, the statistics are calculated with calculate_statistics. And lastly, closeness centrality computes and displays the top and bottom 5 nodes by closeness centrality using calculate_closeness_centrality.

To run the project, I used cargo run –release for a quick output time.

Furthermore, I used multiple tests to ensure functionality. The tests I used include graph loading, which ensures the graph is loaded correctly from a file, degree distribution, which verifies the degree distribution calculation for a small graph, and empty graph, which checks behavior when an empty graph is loaded.

To run these tests, I used cargo test.