Meschach Loblack
DS 210 Project

Project Report

Overview:
In this project, I analyzed a graph representing a scientific collaboration network extracted from papers submitted to the General Relativity and Quantum Cosmology category on arXiv. The graph consists of nodes representing authors and undirected edges indicating co-authorship relationships between authors.

Graph Reading (read_graph):
Reads the graph data from a text file and constructs an adjacency list representation.
Distance Calculations:
Breadth-First Search Distance (bfs_distance): Calculates the shortest distance between two nodes using BFS.
Average Distance (average_distance): Computes the average distance between randomly selected pairs of nodes in the graph.
Maximum Distance (max_distance): Determines the maximum distance between any pair of nodes in the graph using the Floyd-Warshall algorithm.
Median Distance (median_distance): Finds the median distance between all pairs of nodes in the graph.
Degree Distribution (degree_distribution):
Computes the distribution of node degrees in the graph.
Testing:
Includes unit tests for all the distance calculation functions and random graph generation.

Code Implementation:

The code follows a modular structure with functions split into separate modules for readability and maintainability. It utilizes standard libraries such as std::fs, std::collections, and std::time for file I/O, data structures, and time measurements. The CA-GrQ text file contains 5242 nodes and 14496 edges. The read graph function is responsible for parsing the data from a text file and constructing the graph representation. It reads the graph data from the specified text file, where each line represents an edge between two nodes. The function initializes an empty hashmap to store the graph data,

with node identifiers as keys and lists of neighboring nodes as values. It then opens the file and iterates over each line, parsing it to extract the two nodes connected by the edge.

The Breadth-First Search Distance function is implemented to calculate the shortest distance between two nodes within the graph. It employs a queue-based approach to traverse the graph. It starts with the start_node and explores its neighbors layer by layer, incrementing the distance as it traverses. The function maintains a set of visited nodes to avoid revisiting them and then terminates when it reaches the target_node. The distance between the two nodes is then returned. If the target node is unreachable, the function returns usize::max_value() to signify an infinite distance.

The Average Distance calculates the average distance between randomly selected pairs of nodes within the graph. It begins by randomly selecting pairs of nodes and computing the shortest distance between each pair using the bfs_distance() function. The distances obtained are accumulated, and the total number of pairs for which distances are calculated is tracked. Finally, the average distance is computed by dividing the total accumulated distance by the number of pairs calculated. The use of random sampling ensures a representative estimation of the average distance across the entire graph.

The Maximum Distance function is implemented to find the longest shortest path between any two nodes within the graph. It utilizes the Floyd-Warshall algorithm, a dynamic programming approach, to compute the shortest distances between all pairs of nodes in the graph. The function first initializes a matrix representing the distances between nodes, setting initial values to indicate infinite distance between nodes. Then, it iterates over all pairs of nodes and updates the distance matrix based on the shortest path found. After repeatedly considering intermediate nodes, the algorithm computes the shortest distances between all pairs of nodes. Once the distance matrix is computed, the function identifies the maximum distance among all pairs of nodes, which represents the longest shortest path within the graph.

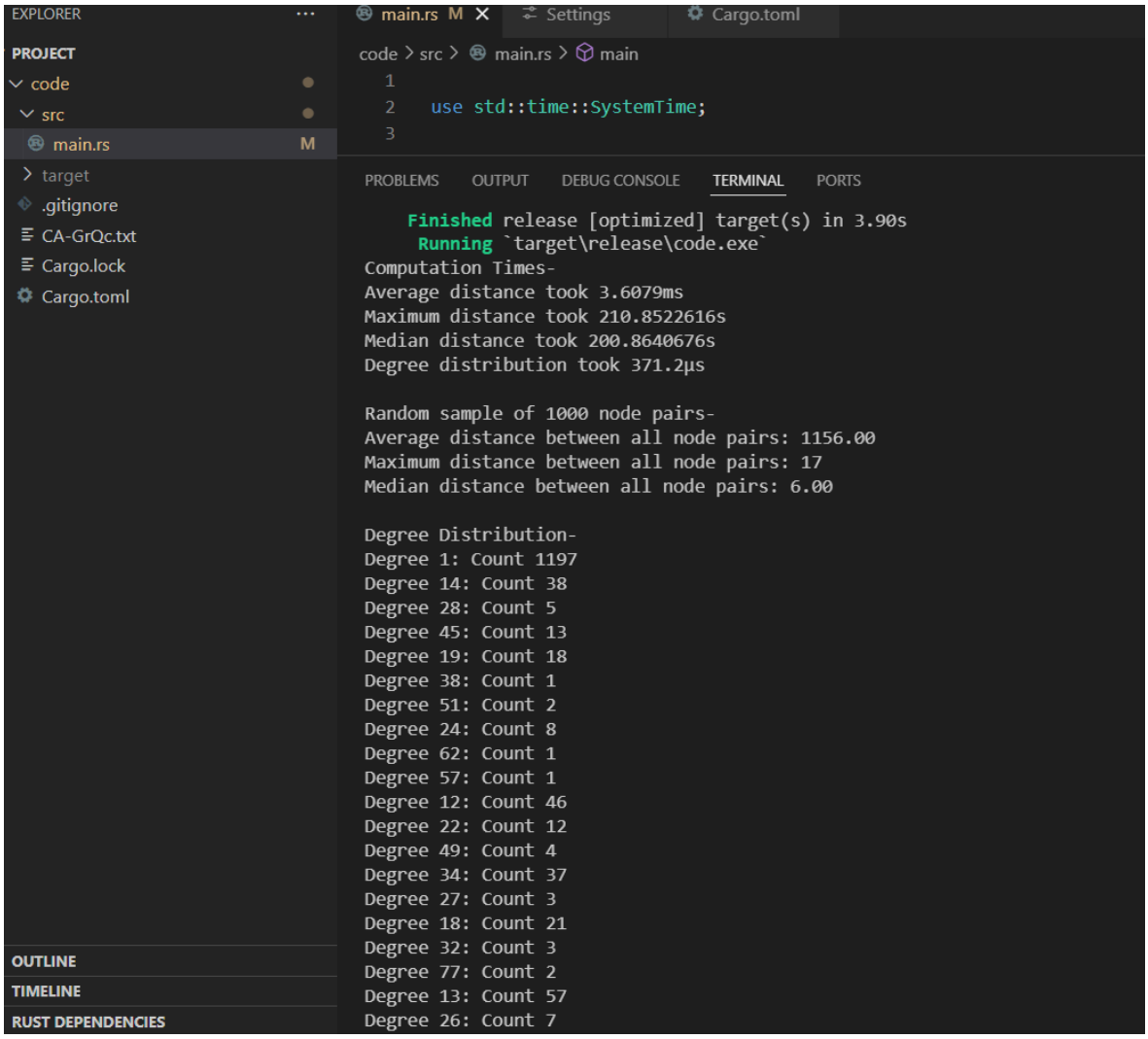
The Median Distance function calculates the median distance between all pairs of nodes in the graph. It first constructs a matrix to store the shortest distances between every pair of nodes using the Floyd-Warshall algorithm, similar to the approach used in the maximum distance function. After computing the distance matrix, the function collects all the unique shortest distances between pairs of nodes and sorts them in ascending order. With the distances sorted, it then calculates the median distance based on the number of distances collected. If the number of distances is odd, the function selects the middle value as the median. If the number of distances is even, it calculates the average of the two middle values.

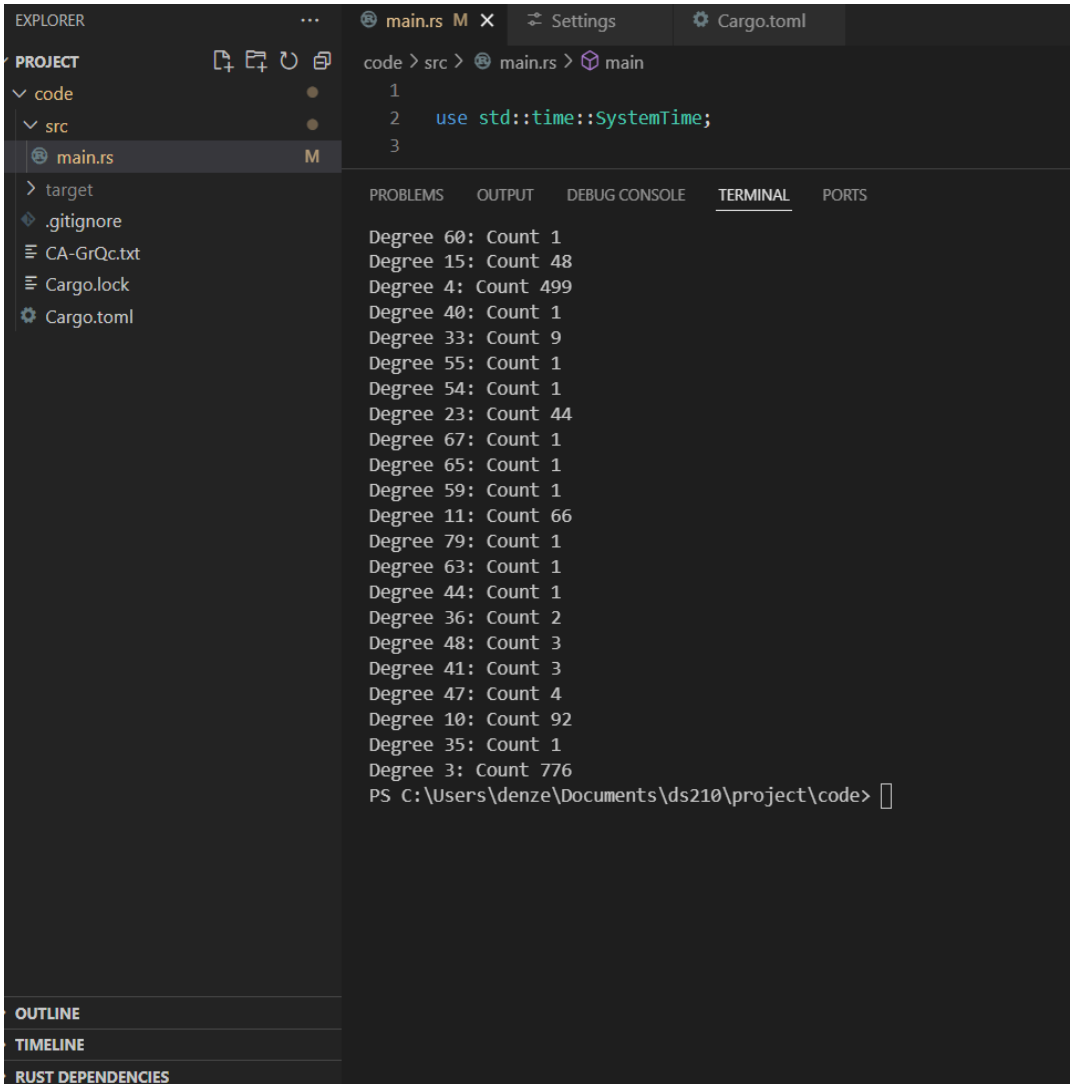
The Degree Distribution function analyzes the distribution of node degrees within the graph. It iterates over each node in the graph and calculates its degree, which is the number of edges connected to the node. For each node, the function retrieves its neighbors

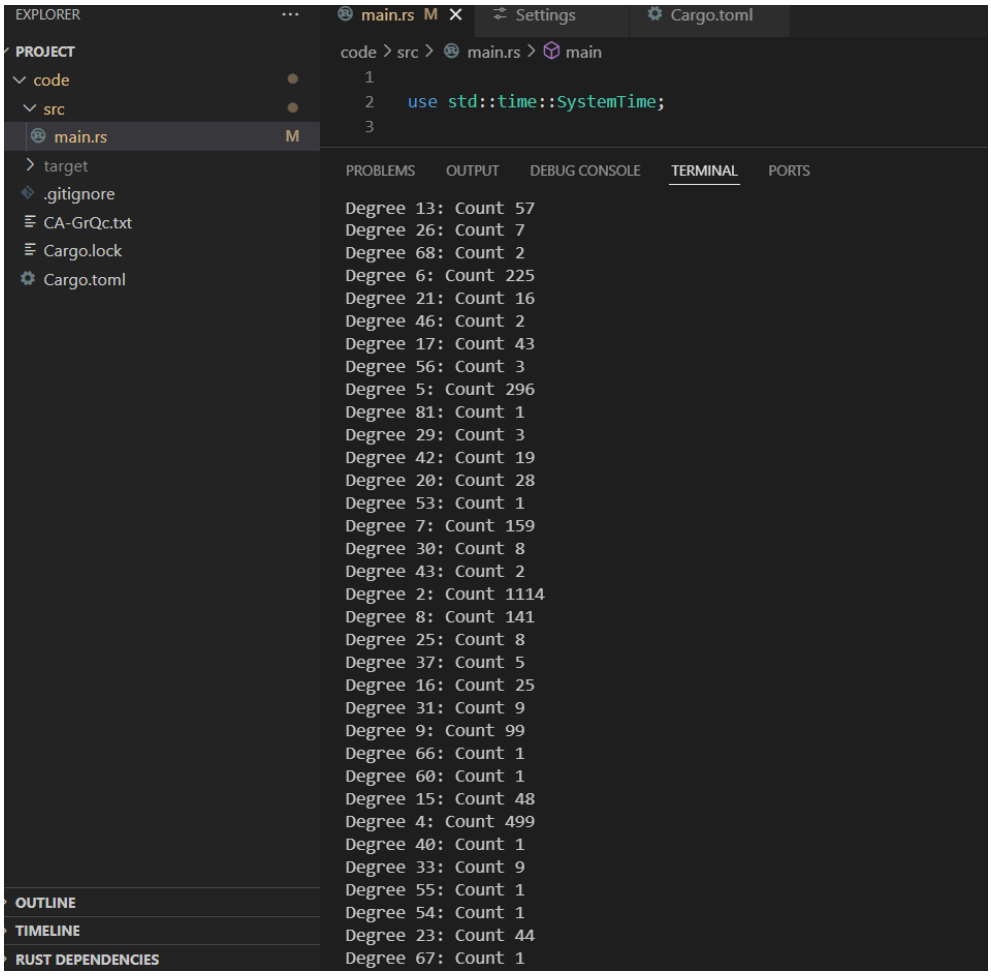
and counts the number of adjacent edges, representing the node's degree. It then updates a hashmap, where the keys represent the degrees and the values represent the count of nodes with that degree. This process results in a mapping of degree values to the number of nodes having each degree. This helps identify common node degrees and reveals whether the graph follows a particular distribution pattern.

The project includes several unit tests to check the correctness of the implemented functions. The test_bfs_distance_simple verifies the correctness of the BFS distance calculation function by creating a simple graph with known distances between nodes. It then asserts that the calculated distance matches the expected value. The test_bfs_distance_not_connected case ensures that the BFS distance function correctly handles scenarios where nodes are not connected. It constructs a graph with disjoint components and asserts that the distance between nodes in different components is reported as infinite. The test_average_distance evaluates the accuracy of the average distance calculation function. It generates a random graph and compares the computed average distance against expected bounds, making sure it falls within a reasonable range. The test_max_distance verifies the correctness of the maximum distance calculation function by generating a random graph and checking that the computed maximum distance falls within expected bounds. The test_median_distance, similar to test_max_distance assesses the accuracy of the median distance calculation function. It generates a random graph and asserts that the computed median distance falls within an acceptable range.

Code Execution and Output:




EXPLORER
PROJECT
code
src
main.rs
M
target
.gitignore
CA-GrQc.txt
Cargo.lock
Cargo.toml
OUTLINE
TIMELINE
RUST DEPENDENCIES
main.rs M
Settings
Cargo.toml
code > src > main.rs > main
1
2 use std::time::SystemTime;
3
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Degree 60: Count 1
Degree 15: Count 48
Degree 4: Count 499
Degree 40: Count 1
Degree 33: Count 9
Degree 55: Count 1
Degree 54: Count 1
Degree 23: Count 44
Degree 67: Count 1
Degree 65: Count 1
Degree 59: Count 1
Degree 11: Count 66
Degree 79: Count 1
Degree 63: Count 1
Degree 44: Count 1
Degree 36: Count 2
Degree 48: Count 3
Degree 41: Count 3
Degree 47: Count 4
Degree 10: Count 92
Degree 35: Count 1
Degree 3: Count 776
PS C:\Users\denze\Documents\ds210\project\code>


EXPLORER
main.rs M
Settings
Cargo.toml
PROJECT
code
src
main.rs M
target
.gitignore
CA-GrQc.txt
Cargo.lock
Cargo.toml
code > src > main.rs > main
1
2 use std::time::SystemTime;
3
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Degree 13: Count 57
Degree 26: Count 7
Degree 68: Count 2
Degree 6: Count 225
Degree 21: Count 16
Degree 46: Count 2
Degree 17: Count 43
Degree 56: Count 3
Degree 5: Count 296
Degree 81: Count 1
Degree 29: Count 3
Degree 42: Count 19
Degree 20: Count 28
Degree 53: Count 1
Degree 7: Count 159
Degree 30: Count 8
Degree 43: Count 2
Degree 2: Count 1114
Degree 8: Count 141
Degree 25: Count 8
Degree 37: Count 5
Degree 16: Count 25
Degree 31: Count 9
Degree 9: Count 99
Degree 66: Count 1
Degree 60: Count 1
Degree 15: Count 48
Degree 4: Count 499
Degree 40: Count 1
Degree 33: Count 9
Degree 55: Count 1
Degree 54: Count 1
Degree 23: Count 44
Degree 67: Count 1
OUTLINE
TIMELINE
RUST DEPENDENCIES