Trust Network Project

By: Vishesh Prasad, Arnav Mehta, Anirudh Bachiraju, Rohit Uruma

Overview and Goals

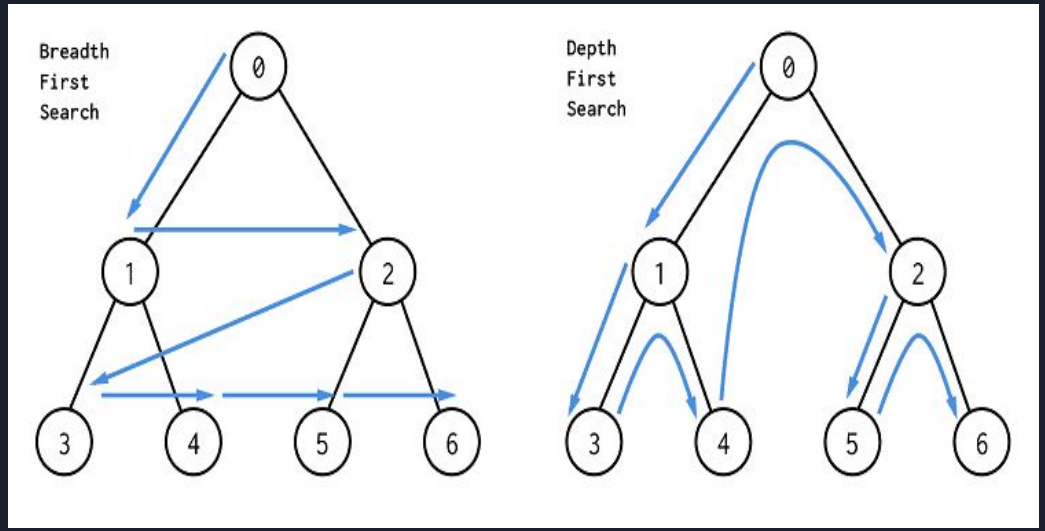
- Analyze a trust network
- Set up project with MakeFile and necessary structure to build project upon
- Build in a data parsing pipeline to read in a directed, unweighted graph with 75879 nodes and 508837 edges
- Store data in a custom graph data structure
- Implement and run the Breadth First Search(BFS) algorithm on the graph
- Implement and run the PageRank algorithm on the graph
- Implement and run the Betweenness Centrality algorithm on the graph
- Leading Question: Given this dataset of trust relationships for Epinions.com used to determine which reviews are shown to the user, which users are the most influential/important in this social network and why are they significant?

The Development: Graph Implementation

- Store data read in from the data parsing pipeline into a 2 dimensional vector of integers
  - We specified a max node so that we can decrease size of data set relatively easily if we wished to
- The data structure represents the adjacency list of the graph by storing the edges between nodes
  - Since we are storing a directed graph, we have an Adjacency To list and and an Adjacency From list
    - The Adjacency To list stores the adjacency list where for each node, we store the nodes that it points to
    - The Adjacency From list stores the adjacency list where for each node, we store the nodes that points to it
- Storing two forms of the adjacency list allows for easier implementation of our algorithms since each different types of algorithm requires different types of graph data structures

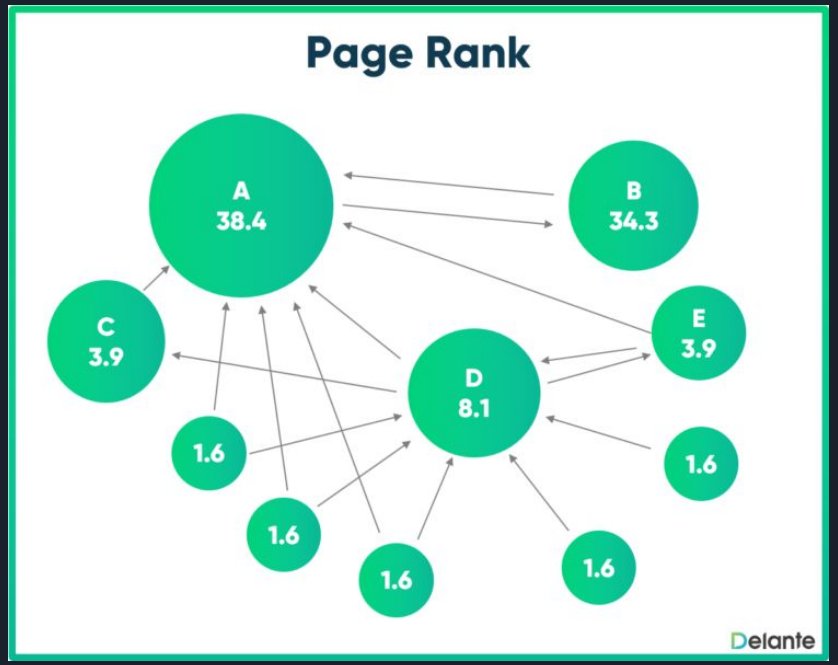
The Development: Breadth First Search(BFS)

- Overview
  - Input: graph object that holds necessary adjacency list
  - Output: vector of integers that stores the nodes along path taken
  - Utilize a queue in its implementation
- Testing
  - Check output path against custom data set
  - Print path of first 10 in the trust network
- Results
  - Runtime of O(n+m) where there are n nodes and m edges for each node
  - Relatively faster compared to worst case since we are working with a sparse graph where the amount of edges are much less than it could be for worst case



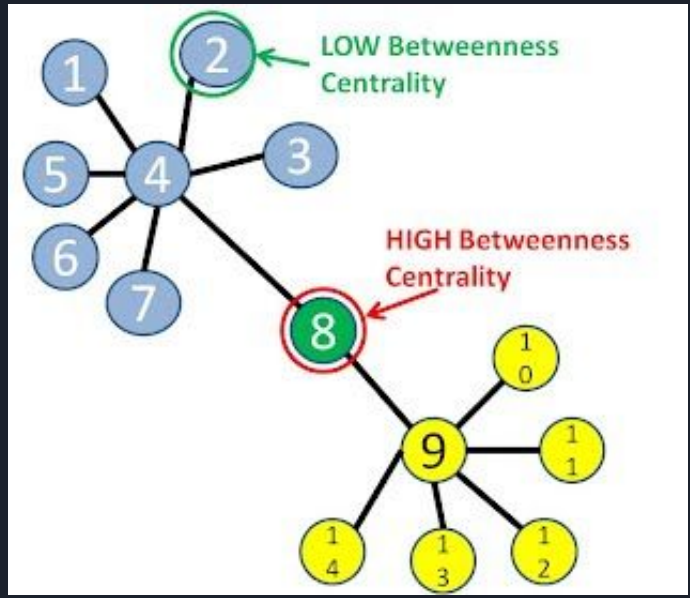
The Development: PageRank

- Overview
  - Input: graph object that holds necessary adjacency list
  - Output: the calculated rank of each node in the data set
  - Convert imputed adjacency list into an adjacency matrix and compute the starting rank vector
  - Do matrix multiplication for each power iteration, taking into consideration the damping factor
- Testing
  - Check output rank against calculated rank of custom data set
  - Print ranks of nodes in large data set
- Results
  - Runtime of O(p*n^2) where n is the number of nodes and p is the number of power iterations we specify to run on PageRank
  - Runtime can get out of hand with large data set and increased power iterations but is relatively quick compared to worst time when run on small graph and with a decent amount of power iterations



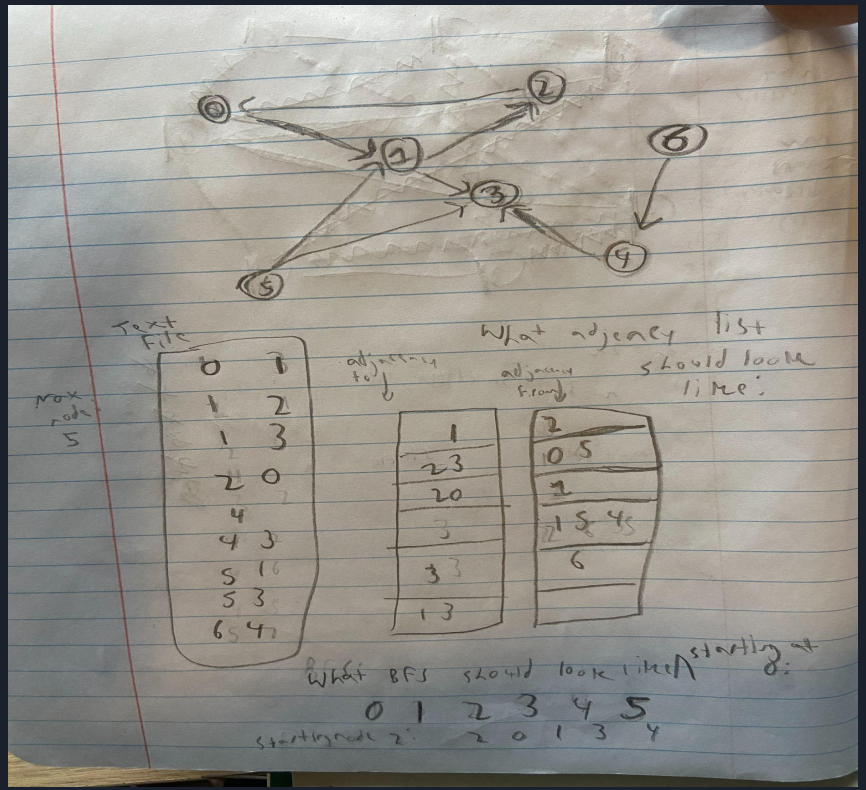
The Development: Betweenness Centrality

- Overview
  - Input: Graph representing the adjacency list
  - Output: The node in the graph with the most shortest paths running through it
  - For each pair of nodes in the network:
    - Find the shortest path between these two nodes using a BFS traversal
    - For each node in this shortest path, increase its “score”
- Testing
  - Check output against expected results from custom data set
  - Print out this expected node
  - Another test includes testing the shortest path algorithm between two nodes on the custom data set
- Results
  - Runtime: O(v^2 * (v + e)) in the worst case, where v is the number of nodes and e is the number of edges. We running BFS between two nodes for every pair of nodes - the worst case for BFS to determine shortest path is O(v + e)



The Development: Smaller Test Data

- Custom data set we developed is shown here
- Number of nodes in data set is controlled by max node set when data parsing
- Allows us to test the correctness of our algorithms with shorter runtime and easier calculations

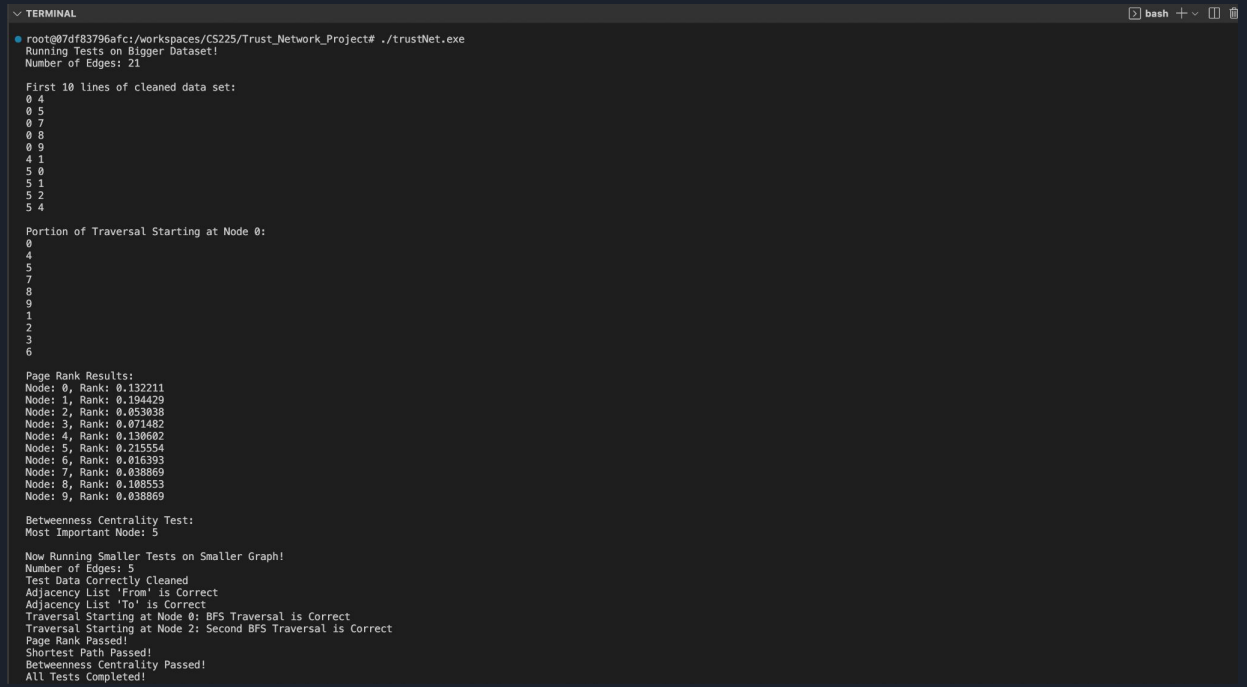


Testing on the Larger Dataset: Results

Most Influential/Important Node

- PageRank: 5
- Betweenness Centrality: 5

*Note: Output shown here is in the terminal but it has been changed such that output is now in output.txt



The Conclusion and Final Thoughts

- We were successfully able to implement and run the algorithms that we set out to accomplish at the beginning of the project and learnt a lot with how to build a project from scratch and reinforced what we learned throughout the semester about graphs, tests, and good coding
- We were able to answer our leading question by using our algorithm to find which node is the most integral to connect different nodes, and which node is the most important to allow for the fewest degrees of trust between nodes.
- Our next steps would be to implement an interactive visual tool to observe the output of Breadth First Search, PageRank, and Betweenness Centrality and how the output changes when we change the max nodes of the data set and, specifically for PageRank, what happens when we change the damping factor and the number of power iterations
- We decided to not run our algorithms on the full scale data set in order to make it easier to output our results and keep the runtime low, however it would be interesting to observe how the runtime would change if we used the whole data sets, and other, larger, data sets