Report (Project One)

Student Name: Danita Anubhuti Prakash
Student ID: s4745511
Student Email: d.prakash@uq.edu.au

Introduction

This project aims to implement a number of efficient algorithms to compute centrality measures for user nodes such as PageRank and Node Betweenness by working with publicly available Facebook social network data. I used Python Language to code by importing libraries like numpy, pandas, matplotlib and networkx. This data has been anonymized by replacing the Facebook-internal ids for each user with a new value. The data contains 4039 nodes, and 88234 edges in total. Each line of the data represents an undirected link starting from one node to another. In order to convert the given txt file into an undirected and unweighted graph. I first imported the file with the help of pandas and converted it to a dataframe. Then I used the function **'construct_graph'** which takes the input as the dataframe and returns an undirected and unweighted graph which can be used to calculate the centralities. The tasks given insists to output the first ten nodes with highest centralities. For this, I wrote the function **'top_n'** which takes the dictionary obtained as output (from the centrality functions) and sorts it to return the top 10 nodes.

Task 1

Node betweenness centrality is a measure of a node's importance in a network based on the number of shortest paths that pass through it. Brades Algorithm is used to calculate score for betweeness centrality[1]. The algorithm works by works computing the shortest path iteratively between all pairs of nodes in a graph and then counting the number times each node appears on a shortest path between two other nodes[2].

To compute the shortest path for an undirected and unweighted graph, breadth-first search algorithm is used. The time the BFS gets a node during a traversal that distance from the source is it's shortest path[3]. The function **'shortestpath_bfs'** used here inputs the graph and the node from which the traversal should start. The function initializes different data structures to keep track of the visited nodes, shortest path and the parent nodes. In each iteration, the function deques the next node and adds it to the list of visited nodes. It checks the adjacent nodes of the current node and adds it to the queue if it is visited. For each of the adjacent nodes the values of shortest path and number of paths are updates. Once all adjacent nodes are examined the loop continues with the next node in the queue. The function returns four different variabes that is the list of visited nodes, the list of parent nodes for each node, number of shortest path and shortest distance from starting node to each node.

The dependancy of the node is a measure of it's importance, this parameter is useful to calculate betweeness. The dependency of the node is direclty reated to the number of shortest paths that pass through it. The function **'accumulate_dependencies'** helps to calculate the dependecies of each node based on it's position in the BFS tree. It takes as input the graph, the list of visited nodes, list of parent nodes and no. of shortest paths to each node. The function initialises delta (dictionary) which keeps track of dependencies. The function updates this delta for each parent node, accumulating dependencies from all it's children in the tree. At the end, it returns delta to the calling function.

Lastly, I make use of the above mentioned function to find the centrality inside the **'betweeness_centrality'** function for each node in the graph. The input is an undirected and unweighted graph constructed from given dataset. At first, a dictionary is initialized with vertex as keys to store the centrality and call the bfs function. Next, the dependencies of each node are found with the help of the information obtained from the bfs tree. The function then iterates over each node in the dependency dictionary to calculate the centrality value. The line centrality[w] += delta[w]/2 calculates the centrality of node w by adding half of dependency score. Here, division by 2 is done to ensures that each path is counted only once. This function returns the centrality values of each node in the form of dictionary. By applying **'top_n'** to this output we get the top 10 nodes with highest cetrality.

The output obtained is:

```
[107, 1684, 3437, 1912, 1085, 0, 698, 567, 58, 428]
```

Task 2

PageRank algorithm helps to find the importance of nodes in a graph based on the number of incoming relationships and the importance of the corresponding source node[4]. The power iteration method is a technique to evaluate the pagerank centrality[5]. This method involves iterativly multipying the adjacency matrix by a vector of initial scores and normalizing the results till convergance[6].

For evaluating the page rank of the undirected and unweighted graph, we need to construct a set of matrices. This helps to compute the page rank score for each node of the graph. The function **'create _matrices'** creates an adjacency matrix and inverse degree matrix by taking the graph as input. Here the adjacency matrix is a matrix representation of the graph where each entry (i, j): is 1 if there is a directed edge from the node or 0 otherwise. Likewise, the degree matrix is a diagnol matrix where i^{th} diagnol element is the sum of the i^{th} row of the adjacency matrix. The inverse of degree matrix is obtained by taking the sum of each row of adjacency matrix and then inversing each of the sums. If the row sum is zero we invert set the value to a small value '1e-8' to avoid division by zero.

Next, the **'pagerank_centrality'** function computes the centrality for page rank for each node by utilizing the matrices obtained from function mentioned above. Here a number of paramteres are used to efficiently caluculate the page rank like:

1. alpha: It is a factor used to determine the importance of incoming edges to a node. (From the task stated the alpha value is 0.85)
2. beta : It is a paramter that determines the probability of jumping to a random node instead of following incoming edges of the node. (From the task stated the beta value is 0.15)
3. eps: A small value of 1e-8 which is used as convergence threshold for the pagerank algorithm.

The page rank vector is calculated using a loop until the difference between previous page rank vector and current one is smaller than eps(1e-8). During each iteration, the function updates PR using the formula:

alpha * adj_matrix.T @ D_inv @ page_rank + (1 - alpha) * beta * np.ones(len_graph) + (1 - alpha) * (1 - beta) * np.sum(page_rank) / len_graph

alpha * adj_matrix.T @ D_inv @ page_rank calculates sum of the scores of nodes that link to the current node, based on adjacency matrix and inverse degree matrix .
(1 - alpha) * beta * np.ones(len_graph) adds the contribution of jumping, where beta is the probability of jumping and np.ones(len_graph) is an arry of ones which tells about uniform distribution for all nodes.
(1 - alpha) * (1 - beta) * np.sum(page_rank) / len_graph is used to handle graphs with outgoing edges and it does not assume that the inverse of the degree matrix exists.

The obtained output is:
```
[107, 3437, 0, 1684, 1912, 348, 414, 3980, 686, 698]
```

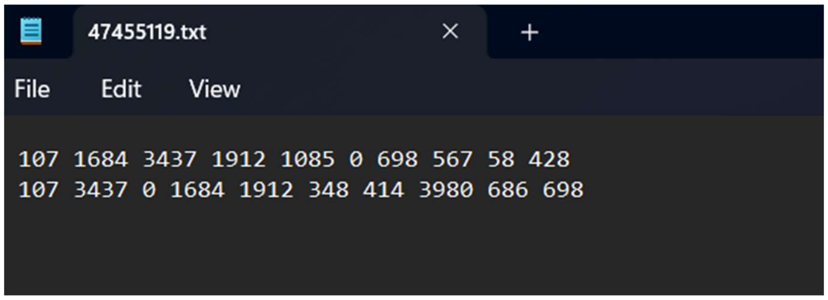
Summary

The tasks of finding centralities of node betweeness and page rank helped me learn different concepts. The first step was to understand the conversion of the dataset given in the form of a textfile and convert this into a undirected and unweighted grap. For finding the betweeness, I used the brandes algorithm. Which involved using shortest path implemented through breadth first search mechanism and finding dependencies for each node. On the other hand, for page rank I used the power iteration method. This involved making the adjacency matrix and inverse degree matrix and iterativly multiplying the matrices and normalizing it to get the results. After calculating the centralitlies for each of the tasks, in order to verify my asnwers I computed the centralities using the networkx inbuilt functions, which is written in the function **'construct_graph'** for betweenss and page rank. These outputs are generated are:

Output obtained by using networkx pakage for calculating node betweeness:
```
[107, 1684, 3437, 1912, 1085, 0, 698, 567, 58, 428]
```
Output obtained by using networkx pakage for calculating pagerank :
```
[3437, 107, 1684, 0, 1912, 348, 686, 3980, 414, 698]
```

In total I used the following functions:
1. **construct_graph(datafram)**
2. **top_n(dic_centrality, n)**
3. **shortestpath_bfs(graph, start_node)**
4. **accumulate_dependencies(graph, nodes_visited, parent_nodes, sigma)**
5. **create_matrices(graph, eps, length_graph)**
6. **centrality_networkx(graph)**
7. **output_textfile(betweeness, pagerank)**
8. **betweenness_centrality(graph)**
9. **pagerank_centrality(graph, alpha=0.85, beta=0.15)**
10. **main()**

Next, I saved the result of my code in a text file called '47455119.txt' using file operations. This code is written in the **'output_textfile'** and called in the **main function**. This can be shown as:



```
107 1684 3437 1912 1085 0 698 567 58 428
107 3437 0 1684 1912 348 414 3980 686 698
```

Hence, I would like to say that this project provided me a good opportunity to understand different types of graph, centralities, it's implementation and coding. I would like to try this code on different datasets and graphs. Further, I would also like to explore different centrality measures.

Reference

[1] *Betweenness_centrality#*. betweenness_centrality - NetworkX 3.1 documentation. (n.d.). Retrieved April 19, 2023, from https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html

[2] *Brandes algorithm*. Department of Computer Science and Technology. (n.d.). Retrieved April 19, 2023, from https://www.cl.cam.ac.uk/teaching/1617/MLRD/handbook/brandes.html

[3] freeCodeCamp.org. (2018, June 28). *Finding shortest paths using breadth first search*. freeCodeCamp.org. Retrieved April 19, 2023, from https://www.freecodecamp.org/news/exploring-the-applications-and-limits-of-breadth-first-search-to-the-shortest-paths-in-a-weighted-1e7b28b3307/#:~:text=We%20say%20that%20BFS%20is,said%20for%20a%20weighted%20graph.

[4] Infovis cyberinfrastructure- betweenness centrality. (n.d.). Retrieved April 19, 2023, from https://iv.cns.iu.edu/sw/bc.html

[5] *PageRank - Neo4j Graph Data Science*. Neo4j Graph Data Platform. (n.d.). Retrieved April 19, 2023, from https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/

[6] *Pagerank#*. pagerank - NetworkX 3.1 documentation. (n.d.). Retrieved April 19, 2023, from https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html