CITS2200: Centralities Project Report

Design Structures:
Adjacency Matrix:
The program represents the edges of a graph in an adjacency matrix rather than an adjacency list since Floyd-Warshall works via a matrix and may be used in both Closeness and Katz Centralities. An adjacency list would reduce the amount of space required to store that graph but it would slow down both the Closeness and Katz centralities.

Sorted Array:
The SortedQueue class implements a sorted array with similarities to a queue as the first item added is inserted at the front of the array, but will be moved down if a higher valued element is inserted into the array.

Algorithms:
Brandes (Brandes, 2001):
Brandes is a very efficient algorithm for unweighted and sparse graphs, with time complexity of O(nm) (unweighted) and requires O(n+m) space. The complexity of Brandes is significantly better than the complexity of Floyd-Warshall $O(V^3)$ and is better suited to unweighted graphs like in this project, compared with Floyd-Warshall. However disadvantage of implementing Brandes is that is uses a number of data structures, (list, stack and queue) on top of using the sorted 'queue' (sorted array) that we implemented to sort of the highest values for each measure, and it is not as simple to implement as Floyd-Warshall.

Floyd-Warshall:
The Floyd-Warshall algorithm is used to calculate the Closeness and Katz Centralities, because it's implementation is simple and only runs on the adjacency matrix, and ends up being a relatively efficient algorithm long term despite its $O(V^3)$ complexity, due to no constants. We have saved computational time by running this algorithm once (because it is used in Katz and Closeness) and then storing the result.

Measures of Centrality Implemented: (assuming the graph is connected)

Degree Centrality:
Degree centrality of a vertex v is the sum of the edges directly connecting v to the other vertices in the graph (so the higher the number of vertices connected to v, the greater its degree centrality). In the program, two nested for loops are used to go through the adjacency matrix and count the number of vertices connected to each vertex.

Closeness Centrality:
Closeness Centrality is a measure of how close all other vertices are to a vertex v, by summing the shortest paths from vertex v to all other vertices (how 'far' away these vertices are) and then taking an inverse of this value (1/far) to find how 'close' v is to the other vertices. The higher the value for the closeness of a vertex, the more 'central' it is in the graph, as it is able to reach other vertices in a shorter distance.

Betweenness Centrality:
The Betweenness of a vertex v in the graph reflects the influence that v has over the flow of information through to the rest of the graph. It is calculated for every vertex by finding the shortest paths from all vertices (i) to all other vertices (j) which go through v. The higher this value the more influence the vertex will have over how information is transferred through the

graph. Implemented Brandes Algorithm for calculating Betweenness Centrality (see above discussion on the choice of algorithm).

Katz Centrality:
Katz Centrality can be seen as another measure of the degree of influence of vertex v. When given a factor k, the shortest path from v to vertex u is given a weight of k to the power of the distance between the vertices. The Katz centrality is found by summing all the different weights from v to every other vertex in the graph. The higher the Katz Centrality of a vertex, the greater the influence on the graph it has.

Output:
The output consists of five user ID's and the respective values for each measure (titled above each output), and the output ID's are ranked based on their corresponding values for the respective measure.
Also included a visualisation class which outputs in a window, all the connections between the vertices of the graph and colours them, based on the number of vertices connected to that vertex divided by the total number of vertices. (% = floor(degree(v)/numVertices * 10), multiply by ten as we use an index of 0-9) This functionality can be turned on via "true" or off using any other string, but needs to be in the 3rd position of the command line arguments. See input section.

Input:
Once the java source files have been compiled, add to the same folder the input text file (file of edges) and run the main class (Centrality), through - **java Centrality <input NAME> <katz-alpha value> <visualize-graph>** . The first argument is just the text file name for the file of edges. Do not include the extension (.txt). The second argument is the alpha value for Katz Centrality in a double e.g. 0.1, 0.5,0.9… The third argument is a "true" or some other string, which is used to visualise the graph as mentioned in the output section above.

Libraries:
java.awt.* for The GraphVisualiser class, needed awt.Color, Window and Canvas for this class.
java.io.* for reading the text file and for IOException catching.

Java Version:
The code for this was written with Java 8, but should be runnable with Java 7.

Brandes, U. (2001). A Faster Algorithm For Betweenness Centrality. *Journal of Mathematical Sociology,* 25(2),163-177. http://algo.uni-konstanz.de/publications/b-fabc-01.pdf