Geir Matti Järvelä (geirmj11@ru.is)
Þorgeir Auðunn Karlsson (thorgeirk11@ru.is)

Programming Assignment 4: Properties of Social Networks

For the assignment we created three classes, explained further below. All of the classes implement an intelligent lazy solution to each problem. Each method checks first if a solution exists, and if so simply returns a previously calculated solution. This method costs a little bit more in memory, since previous solutions need to be kept, but makes up for it with constant calculating times when duplicating previous calculations, or when two functions use the same results.

Part I - The Centrality class

Degree: degree() counts the number of edges to a given Vertex, and returns that number.
Popularity: popularVertex() calculates the degree for all vertices and returns the highest one.
Eccentricity: ecc() creates a BreathFirstPath for the given node, finds the highest distance to every other node, and returns the highest distance.
Center: center() goes through the ecc[] array, calls ecc() for every node, and returns the lowest value.
Closeness: closeness() creates a BreathFirstPath for a given node, goes through all paths in the resulting collection and adds their distances together. It then divides one with that result, and returns the result.
Closest: closest() calculates the closeness of all vertices, and returns the vertex with the highest closeness.
Effective Eccentricity: effEcc() generates an arrayList and creates a breathFirstPath for the given node. It then adds all distances from the BreadthFirstPath to the list, sorts it, and returns the value in the 90% position in the sorted list.
Effective Center: The effCenter() method goes through the list of effective eccentricities and generates the missing values, and returns the node with the lowest value.

The numbers in the table on the right show the complexity of each method, for the first time each method is called. However some of the methods call other methods for every point, for example the closest() method calls the closeness() function for every point in the graph, and since the results are stored, they can be accessed directly thereafter.

Method	Initial Complexity	Subsequent
Degree	$\sim V$	constant
Eccentricity	$\sim 2V+E$	constant
Effective Eccentricity	$\sim V(\log V)$	constant
Closeness	$\sim 2V+E$	constant
Most Popular Vertex	$\sim V^2$	constant
Center	$\sim V$	constant
Effective Center	$\sim V(V(\log V))$	constant
Closest Vertex	$\sim V(V+E)$	constant

Complexity table: V: Number of Vertices
E: Number of Edges.

Part II - The SymbolCentrality class

The SymbolCentrality class is an extention of Centrality. It creates a Symbolgraph class instance, and uses it to create indexed integer keys for each actor. This allows the class to use the methods of the Centrality class to calculate all necessary values.
Since each function calls the sg.index() method, this adds log(n) complexity to each operation of the Centrality class. In addition more memory is needed because of the SymbolGraph that stores the indices.

Part III - The ExtendedBreadthFirstPaths class

The generate() method creates a queue and initializes the arrays numOfPaths[] and levels[] with a value of -1. Each field represents a single vertex in the graph. The base path is initialized to one, since the base node has one path to itself, and given the level zero. It is then added to a queue. After that a loop goes through all points in the queue, takes the number of paths of each node connected to the current node, and adds the sum of paths from those nodes already initialized, that have a lower value than the current node to the current nodes total paths. It then adds all uninitialized nodes to the queue, and gives them a level of one higher than the current node. At the same time it keeps track of the vertex with the highest number of paths, and once the method has gone through all vertices, we return that value.
The complexity of this method is $\sim 2V+E$ where V is the number of vertices and E is the number of edges.