Sarah Kirby

Module 9 Assignment

Output:

Average joint neighbors of linked nodes: 2.8861031518624642

Average joint neighbors of unlinked nodes: 0.23764582673713436

Program:

package assignments;  
  
import graph.GraphUtils;  
import org.apache.tinkerpop.gremlin.structure.Direction;  
import org.apache.tinkerpop.gremlin.structure.Vertex;  
import org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerGraph;  
  
import java.util.HashMap;  
import java.util.Iterator;  
import java.util.Map;  
  
public class Module9 {  
  
 private String GRAPH\_INPUT = "GraphDatabases\\students.graphml";  
  
 public Module9(){  
 TinkerGraph graph = GraphUtils.*readGraphML*(GRAPH\_INPUT);  
  
 System.*out*.println("Average joint neighbors of linked nodes: " +  
 jointNeighborAvg\_Linked(graph));  
  
 System.*out*.println("Average joint neighbors of unlinked nodes: " +  
 jointNeighborAvg\_Unlinked(graph));  
 }  
  
 */\*\*  
 \* Counts the number of joint neighbors of two nodes  
 \** ***@param*** *v1  
 \** ***@param*** *v2  
 \** ***@return*** *\*/* public int jointNeighbors(Vertex v1, Vertex v2){  
 int jointNeighbors = 0;  
 Map<Integer, Vertex> v1\_neighbors = new HashMap<>();  
  
 Iterator<Vertex> neighbors = v1.vertices(Direction.*BOTH*);  
 while(neighbors.hasNext()){  
 Vertex neighbor = neighbors.next();  
 v1\_neighbors.put(neighbor.hashCode(), neighbor);  
 }  
  
 neighbors = v2.vertices(Direction.*BOTH*);  
 while(neighbors.hasNext()){  
 Vertex neighbor = neighbors.next();  
 if( !neighbor.equals(v1) && v1\_neighbors.containsValue(neighbor)){  
 jointNeighbors++;  
 }  
 }  
  
 return jointNeighbors;  
 }  
  
 */\*\*  
 \* Calculates the average number of joint neighbors for linked nodes  
 \** ***@param*** *graph The graph to operate on  
 \** ***@return*** *The joint neighbor average  
 \*/* public double jointNeighborAvg\_Linked(TinkerGraph graph){  
 int jointNeighborSum = 0;  
 int pairCount = 0;  
  
 //calculate joint neighbors for each pair of linked nodes  
 Iterator<Vertex> nodes = graph.vertices();  
 while(nodes.hasNext()){  
 Vertex node = nodes.next();  
 Iterator<Vertex> links = node.vertices(Direction.*OUT*);  
 while(links.hasNext()){  
 Vertex linked\_node = links.next();  
 jointNeighborSum += jointNeighbors(node,linked\_node);  
 pairCount++;  
 }  
 }  
  
 return jointNeighborSum / (double)pairCount;  
 }  
  
 */\*\*  
 \* Calculates the average number of joint neighbors for unlinked nodes  
 \** ***@param*** *graph The graph to operate on  
 \** ***@return*** *The joint neighbor average  
 \*/* public double jointNeighborAvg\_Unlinked(TinkerGraph graph){  
 int jointNeighborSum = 0;  
 int pairCount = 0;  
  
 //calculate joint neighbors for each pair of linked nodes  
 Iterator<Vertex> nodes = graph.vertices();  
 while(nodes.hasNext()){  
 Vertex node = nodes.next();  
 Iterator<Vertex> other\_nodes = graph.vertices();  
 while(other\_nodes.hasNext()){  
 Vertex other\_node = other\_nodes.next();  
 if(!other\_node.equals(node) &&  
 !MyGraphUtils.*hasLink*(node, other\_node)){  
 jointNeighborSum += jointNeighbors(node, other\_node);  
 pairCount++;  
 }  
 }  
 }  
  
 return jointNeighborSum / (double)pairCount;  
 }  
  
 public static void main(String[] args) {  
 new Module9();  
 }  
}