CS 6301.002 Short Project 2 Group 10

**Problems:**

A – TopoPrint.java - Topological ordering of a DAG.

B – TreeDiameter.java - Diameter of a tree.  
C – StronglyConnectedComponents.java - Strongly connected components of a directed graph.  
D – OddLengthCycle.java - Finding an odd-length cycle in a non-bipartite graph.  
E – TestEulerian.java - Is a given graph Eulerian?

**Report:**

**a. Topological ordering of a DAG.**

Algorithm 1:

This is a straightforward algorithm in which vertices with 0 incoming edges are removed along with their outgoing edges. This is repeated until all the vertices of the graph have been removed (or visited)

In the implementation TopoPrint.java, the actual data structure itself is modified so this algorithm must be run AFTER the 2nd algorithm is.

Algorithm 2:

Run DFS on g and push nodes to a stack in the order in which they finish. This is a recursive implementation.

**b. Diameter of a tree**

**Approach:**

Run BFS on the graph, update distance of each vertex from the source node.

Select Node Z with maximum distance from the first BFS.

With Node Z as source, reset the graph and run second BFS.

The diameter of the tree is the maximum distance of any node from Z in second BFS.

**Efficiency:**

Runtime of the Algorithm is same as runtime of BFS which is O(|E|)

Since BFS needs a queue for iterating through nodes, we need an extra O(|V|) space.

d. **Finding an odd-length cycle in a non-bipartite graph.**

Given a graph, find an odd-length cycle and return it.

If the graph is bipartite, return null.

Algorithm: run BFS. If no edge of G connects two nodes at the same

level, then the graph is bipartite and has no odd-length cycle.

If two nodes u and v at the same level are connected by edge (u,v),

then an odd-length cycle can be found by combining the edge (u,v)

with the paths from u and v to their least common ancestor in the BFS tree.

If g is not connected, this is repeated in each component.

List<Vertex> oddLengthCycle(Graph g) { ... }

e. **Is a given graph Eulerian?**

**Approach:**

Run DFS algorithm to determine the connected components(cn) of the graph. For each vertex that is visited, check for vertices with odd edge count and add them to oddDegreeVertices (elist) list. At the end of DFS, check for oddDegreeVertices (elist) size and connected components count.

We can draw following conclusions:

* cn is more than 1: the graph is not connected
* elist is empty then allvertices has even edge count so it has a Euler tour or the graph isEulerian
* elist has 2 item then graph has 2 vertices with odd edge count so it has Eulerian path starting from one odd edge count vertex to the other
* elist has 1 item or more than 2 items, then the graph is not Eulerian

**Efficiency:**

Runtime of the algorithm is same as runtime of DFSVisit which is O(|E|)