CSE 274: Data Abstraction and Data Structures
Project #5: Graph Algorithms

Outcomes:

- Create solutions to these graph algorithms:
  - Breadth-first traversal
  - Depth-first traversal
  - Topological Sort
  - Dijkstra's Shortest Path
- Use Java's classes as needed to implement these algorithms

General requirements:

- Follow good programming practices.
  - Format your code so that it is readable using generally accepted guidelines for formatting source code.
  - Don't make code more complicated than it needs to be. If you find yourself repeating code, cutting and pasting code, etc., write a method to perform that task.

Specific requirements:

- Down load the Program#5 zip file that contains:
  - Following Java files: Graph.java, GraphInterface.java, Vertex.java, and GraphAlgorithms.java
  - Several csv files containing data to describe the vertices and edges that can be used in testing your algorithms
- Modify only the GraphAlgorithms.java file.
- In GraphAlgorithms.java, you will complete four methods:
  - breadthFirstTraversal
  - depthFirstTraversal
  - topologicalSort
  - findShortestPath
- When completed, turn in just the GraphAlgorithms.java file.

Output Format

Generally, paths should be printed out with vertex name separated by a single space with not spaces before or after the path. Example:

```
CVG CMH ATL ORD STL MSY DFW DEN LAX
```

For the Topological Sort solution, the values should appear as follows. Note that, when there is a choice, the vertices should appear in alphabetical order (when reading the result from left to right.) This will happen automatically since the *adjacentVertices* data member of a Vertex is implemented as a *TreeSet<String>*:

```
watch underpants shirt tie socks pants shoes belt jacket
```

For Dijkstra's Shortest Path Algorithm, the path and total weight should be return as shown here:

```
Shortest Path CVG CMH ATL MSY DFW LAX
Total weight: 2250
```

For any cases for which there is no solution (for example, if the client asks for a starting or ending Vertex that doesn't exist, or if those two vertices are not connected), your program should return the string "path not found".

Graph Algorithms pseudo-code:

Breadth-first search

```
add starting vertex to queue
while queue is not empty {
    currentVertex = dequeue
    add currentVertex to result
    add currentVertex to visited set
    add all of the nodes adjacent to currentVertex to queue
        (but only if that node has not been visited yet.)
        (we will add these in alphabetical order)
}
```

Depth-first search

```
push starting vertex to stack
add starting vertex to result
add starting vertex to visited set
while stack is not empty {
    currentVertex = peek. //. Leave it on the stack
if (currentVertex has neighbors that are unvisited {
        push one of the neighboring nodes adjacent to stack
            (Let's call this node neighbor)
            (we will add these in alphabetical order)
        add neighbor to visited set
        append neighbor to the result
        push neighbor onto the stack
    } else pop stack
}
```

Topological Sort

```
Input: acyclic graph
Create a Stack called stk
N = numVertices
visited = { }
for i=1 to N {
    next = unvisited node w/o unvisited successors
   add next to set visited
   stk.push(next)
}
return stk
Result is the stack in reverse
```

Dijkstra's Algorithm

```
Input: graph, starting vertex, ending vertex

priorityQueue = priority queue of State
priorityQueue.add( new State(startingVertex, 0, startingVertex) )
visited = { }

while (!priorityQueue.isEmpty())
    nextEntry = priorityQueue.remove()
    if (nextEntry.vertex has not been visited) {
        add nextEntry.vertex to visited
        if (nextEntry.vertex == endingVertex)
            return nextEntry
        else {
            currVertex = nextVertex.vertex
            currCost = nextVertex.cost
            currPath = nextVertex.path
            for every unvisited neighbor to currVertex, V {
                nextCost = currCost + edge cost of currVertex → V
                nextPath = currPath with V appended
                nextState = new State(V, nextCost, nextPath)
                priorityQueue.add(nextState)
            }
    }
}
```

Submit:

Submit a single file: GraphAlgorithms.java file