CSCI 230 Data Structures and Algorithms
Homework - Graphs
Jonathan Limpus

Introduction

This assignment is based on material from the course primary textbook, "Data Structures and Algorithms in Java" by Michael Goodrich, chapters:

- Section 14.1 Graphs
- Section 14.2 Data Structures for Graphs

These problems may require access to supplied source code which is available on the course GitHub under libraries.

Assignment

Problem 1. Write an external static function `containsCycle(AdjacencyMapGraph<V,E> G)` where G is a directed graph, that returns `true` if G contains a cycle; otherwise, returns `false`. **Note**: If you need to change any class implementations (e.g., make a member `public` vs `protected`), provide documentation for those changes which includes your rationale for the change (e.g., it is otherwise impossible or is significantly slower, etc.).

Code: GraphOperations.java

```java
// This list will hold the visted vertices for both functions
  public LinkedList<Vertex<V>> visted = new LinkedList<Vertex<V>>();
  private boolean cycle;

  /**
   * helper: this will allow us to pass in vertex parameters for easier access, the
   * real 'meat' of the algorithm
   * @param g
   * @param v
   * @return true if there is a cycle
   */
  public boolean helper(AdjacencyMapGraph<V,E> g, Vertex<V> v) {
    visted.add(v);
    for (Edge<E> e : g.outgoingEdges(v)) {
      Vertex<V> connected[] = g.endVertices(e);
      for (int j = 0; j < visted.size(); j++) {
        if(connected[1].equals(visted.get(j)))
          return cycle = true;
      }
      helper(g,connected[1]);
    }
    visted.remove(visted.size() - 1);
    return cycle;
  }

  /**
   * containsCycle - essentially a loop for a our helper function
   * @param g
   * @return true if there is a cycle
   */
  public boolean containsCycle(AdjacencyMapGraph<V, E> g) {

    Iterable<Vertex<V>> vertex = g.vertices();
    for(Vertex<V> v : vertex) {
      return helper(g, v);
    }
    return cycle;
  }
```

Problem 2. Describe an algorithm for counting all possible paths between two arbitrary vertices in a directed acyclic graph.

Solution. This algorithm assumes usage of methods in AdjacencyMapGraph.java from the textbook source code.

Algorithm 1 countPaths(graph, startVertex, endVertex)

```
for edge i : graph.outgoingEdges do
vertex ← graph.opposite(startVertex ,i)
    if vertex ≠ endVertex then
return countPaths(graph, vertex, endVertex)
    else
return totalPaths ← totalPaths + 1
```

□