CSC 225 Fall 2016
Algorithms and Data Structures I
Assignment 4—Programming Part

Due Date
Friday, December 2, 2016

Objectives
- Understand undirected, simple graph algorithms and data structures
- Implement adjacency matrix data structure to represent graphs
- Implement DFS (Depth First Search) graph traversal algorithms
- Implement connected component algorithm
- Implement cycle detection algorithm
- Understand and implement preorder and post order graph traversal numberings

Instructions
You are to implement, using the template the given template but otherwise from scratch, a java class Graph225 to practice designing, implementing and testing algorithms and data structures for undirected, simple graphs.

Note: You are to use the attached template Graph225.java. **Additional** helper methods may be implemented. However: **do not** change the method signatures of the original methods or class names/package. **No marks** will be given if file/classes/method signatures are changed.

Part 1: Adjacency matrix data structure class
Design and implement a java class **Graph** with includes the following methods **generate, read** and **write** to initialize an adjacency matrix. **Graph** provides square, symmetric adjacency matrices, which represent simple, undirected graphs. An adjacency matrix M represents a graph G=(V,E) where V is a set of n vertices and E is a set of m edges. The size of the matrix is n where n is in the range 4 to 15 only. Thus, the rows and columns of the matrix are in the range of 0 to n-1 representing vertices. The elements of the matrix are 1 if the edge exists in the graph and 0 otherwise. Since the graph is undirected, the matrix is symmetric and contains 2m 1's.
Method **read(fn)** updates the graph's adjacency matrix by reading data from a file named fn (cf. for the file structure please refer to the sample input file `testadjmat.txt`). Method **write(fn)** writes the graph's adjacency matrix representation into a file with name fn.
Method **generate(n, density)** populates a graph G of size n, with random data. The value of the density parameter is 1, 2 or 3 specifying the density of G—the number of edges m of the graph where m is in the range n to $\frac{n(n-1)}{2}$. Density 1 means $m = n + \frac{2n}{5} = \frac{7n}{5}$; density 2 means $m = \frac{n^2}{4}$; and density 3 means $m = \frac{n^2}{2} - \frac{n^2}{10} = \frac{2n^2}{5}$. **generate** populates the adjacency matrix by randomly by generating m pairs of random numbers (row, col) where row and col are in the range of 0 to n-1 (i.e., edge from vertex row to vertex col or vice versa. You may use your random number generator from the previous assignment or java library functions to generate random numbers.

Part 2: Reachability

Design and implement a java method **reach(G, v)** from scratch to traverse graph G represented, starting at vertex v, using the depth first search graph traversal algorithm. **reach** returns a vector R of n elements where R[j] is 1 if vertex j can be reached from v and 0 otherwise. Adjacent vertices must be visited in strictly increasing order (for automated marking). Use a local vector variable to represent visited nodes.

Part 3: Connected Components

Design and implement a java method **connectedComponents(G)** that computes the number connected components of a graph G. **connectedComponents** returns the number of components of G and 1 if graph G is connected. Adjacent vertices must be visited in strictly increasing order (for automated marking). Use a vector to represent visited nodes.

Part 4: Cycles

Design and implement a java method **hasCycle(G)** that determines whether graph G contains at least one cycle. **hasCycle** returns true if G has a cycle and false otherwise. Adjacent vertices must be visited in strictly increasing order (for automated marking). Use a vector to represent visited nodes.

Part 5: Preorder and postorder numbers

Design and implement java methods **preOrder(G)** and **postOrder(G)** that compute the preorder and postorder numbers of the vertices in G represented. These methods return the preorder and postorder numbers for each vertex in G in a vector of n elements. Adjacent vertices must be visited in strictly increasing order (for automated marking). Use a vector to represent visited nodes.

Part 6: Testing

Develop a method **test** to test and exercise the algorithms and data structures developed for the first five parts of this assignment extensively. The output generated by this method must convince the marker that the algorithms and data structures are implemented as specified. For example:

- Generate graphs of different sizes and densities
- Test the algorithms for different graphs
- Test your algorithms using the sample input file `testadjmat.txt`