INDIAN STATISTICAL INSTITUTE

Assignment 12

Data and File Structures Laboratory, M. Tech (CS) - I Year, 2014-2015 (Semester - I)

Total marks : 30 + 20 + 10 + 10 + 40 + 30 + 20 + 20 + 20 + 50 + 50 + 50 + 50 + 50 + 50 + 50 + 100 + 50 (Good prog. habit) = 700

Uploading date: 24.11.2014 Clarification deadline: 01.12.2014

Submission deadline: 07.12.2014

(DiGraph Class): Create a directed weighted graph class in Python, with suitable methods as described next, for a directed graph G:

(i) (initialize) Design a method __init that creates G from various data input; an empty graph (0 nodes) is created if there is no data input:
– a dictionary having the list of adjacent vertices for each vertex;
– a list of edges given as ordered tuples (vertex1, vertex2);
– adjacency matrix of the graph; The weight of the edges also needs to be taken as input. For an unweighted graph, you can take all the edge weights to be 1. [10+10+10=30]

(ii) (presence of vertex/edge) Create methods has_vertex and has_edge that return True if the input vertex/edge is present in G. [10+10=20]

(iii) (addition of vertex:) Create methods add_vertex and add_vertices that add a (few) new vertex (vertices) to G, without any specified edges. [10]

(iv) (addition of edge:) Create methods add_edge and add_edges that add a (few) new edge(s) to G, between existing vertices. The vertices, between which edge(s) will be added, and the weight of the edge need to be specified. [10]

(v) (deletion of vertex/edge:) Create methods delete_vertex – that deletes a vertex and all edges incident to it, delete_edge – that deletes an edge between two vertices, delete_vertices – that deletes a set of vertices and all edges incident to it and delete_edges – that deletes a set of edges; [10+10+10+10=40]

(vi) (induced subgraph:) Create a method induced_subgraph that finds the induced subgraph of a set of vertices for a given graph G. [30]

(vii) (degree:) Create methods indeg_vertex and outdeg_vertex that returns the indegree and outdegree, respectively of a specific input vertex. [10+10=20]

(viii) (adjacency matrix:) Create a method adjacency_matrix that returns the adjacency matrix of the graph as list of lists. [20]

(ix) (neighbors:) Create a method neighbor that returns two lists for a vertex v– one has all the vertices from which there are incoming edges to v; and the other list has all the vertices to which there are outgoing edges from v. [10+10=20]

(x) (breadth first search:) Create a method breadth_first_search that takes a vertex v of G as input and returns a list of vertices in G in breadth-first ordering from v. [50]

(xi) (depth first search:) Create a method depth_first_search that takes a vertex v of G as input and returns a list of vertices in G in depth-first ordering from v. [50]

(xii) (report paths:) Create a method all_paths that returns all paths between input vertices (u, v) in G. [50]

(xiii) (shortest path:) Create a method shortest_path that returns the shortest path between input vertices (u, v) in G. [50]

(xiv) (minimum spanning tree:) Create a method mst that finds the minimum spanning tree of G. [50]

(xv) (strongly connected component:) Create a method connected_component that takes a vertex as input and returns the strongly connected component of G containing that vertex. [50]

(xvi) (connected graph:) Create a method is_connected that returns True if the graph is connected. [50]

(xvii) (undirected graph:) How can you use the above methods for an undirected graph? [100]

(xviii) (graph display:) Can you use a graph visualization package to display the graph G and the shortest paths, MST, BFS, DFS, strongly connected component? [200 (extra credit)]