Lab 10 – Graphs

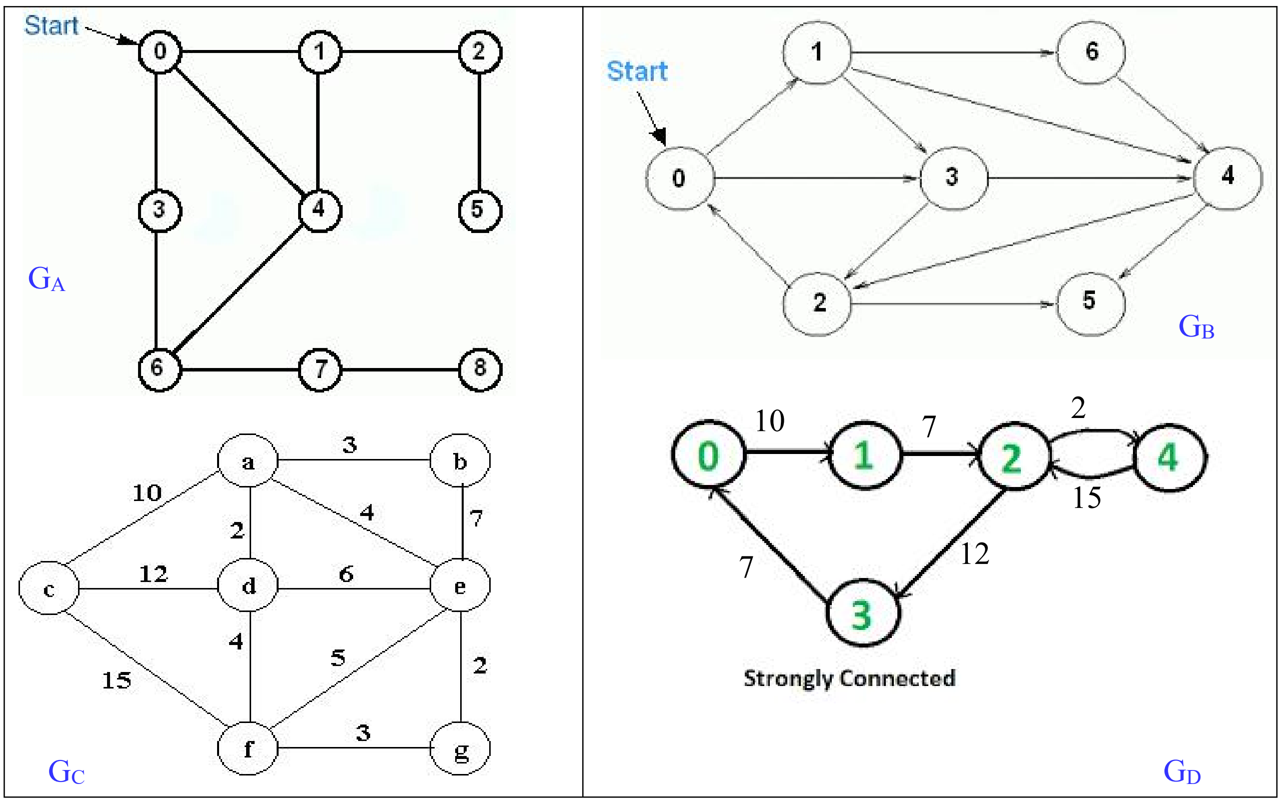
Objectives:

Learn How to Implement Graph using Linked Lists
Learn How to Implement some basic graph Algorithms

Examples:

1- Class **GraphAsLists** shows how graph may be implemented using array of linked lists. Note that **Vertex** and **Edge** are implemented as inner classes inside the GraphAsLists class. Study these classes carefully to make sure you fully understand their implementation.

2- Class **TestGraph** shows how GraphAsLists may be used. You can use the following example graphs for which the data files are given as, GraphDataA.txt, GraphDataB.txt GraphDataC.txt and GraphDataD.txt for graphs G_A, G_B, G_C and G_D respectively.



Study carefully the implementation of the ***initGraph()*** method inside the TestGraph class. This method can be used to read data files representing both Directed and Undirected Graphs to instantiate the graph object. It assumes that the first line in the data file contains the size of the graph (i.e. number of vertices) followed by 'D' for directed graph or 'U' for undirected graph followed optionally by the labels of the vertices (A, B, C, etc). The rest of the lines contain two integers each, representing the edges of the graph.

Tasks:

1. Manually find the depth-first and breadth-first traversals of the above graphs and then run the TestGraph example to confirm your results.

2. (a) Add the following Methods to the GraphAsLists class:

public ArrayList<Edge> getEdges():	Returns all the edges in the graph as a single ArrayList
public boolean isEdge(int v0, int v1)	Check if an Edge linking v0 and v1 exists
public boolean isConnected():	Check for Connectedness of the graph, assuming the graph is undirected. **Hint:** Similar to Breadth-First traversal, only that instead of visiting a vertex you should increment a counter. At the end, you should compare the counter with number of vertices. If not all vertices are visited, then the graph is not connected.

(b) Update the TestGraph class to test your methods.

3. (a) Add the following Methods to the GraphAsLists class:

public boolean isStronglyConnected()	Checks if the current directed graph is strongly connected. Example G_C is strongly connected.
public GraphAsLists getUndirected()	Returns an undirected version of the current graph, assuming the graph is directed
public boolean isWeaklyConnected()	Checks if the current directed graph is weakly connected. Example G_B is weakly connected

(b) Update the TestGraph class to add an option that test and print the type of connectedness of the graph. For Undirected graph, the output should be either "Connected" or "Not connected". For Directed graph, the output should be one of "Strongly Connected", "Weakly Connected" or "Not Connected".