Network Analytics Assignment 1

Qian Zhang

C01939418

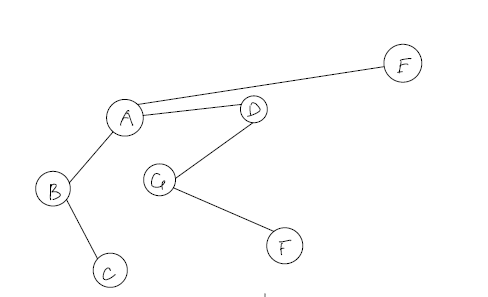
1. (10 points) A directed graph is strongly connected if there is a (directed) path from every node to every other node. Show that in a directed strongly connected graph containing more than one node, no node can have a zero indegree or a zero outdegree.

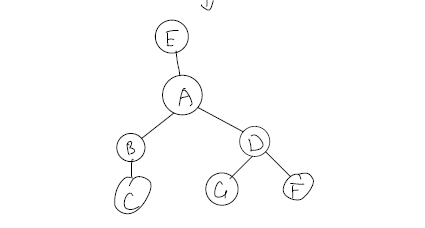
Indegree and outdegree are the two characteristics of nodes in directed graphs. Indegree of a node is the number of edges that are directed into this node, and outdegree of a node is the number of edges that are directed to another node from this node. For a strongly connected directed graph, there is a directed path from every node to every other node. Therefore, every node is reachable from every other node, and this property of directed graph cannot hold if, in a directed strongly connected graph, a single node has zero indegree (this node has no edges that are directed into this node from other node, so it is not reachable from other nodes), or a single node has zero outdegree (this node has no edges that are directed into other nodes from this node, so it could not reach any other nodes). Therefore, if a directed strongly connected graph contains more than one node, no node can have a zero indegree or a zero outdegree.

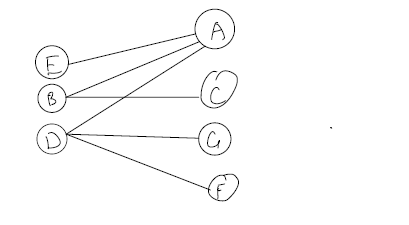
2. (10 points) Show that every tree is a bipartite graph.

For all bipartite graphs, the nodes could be divided into two sets U1 and U2, and every node in U1 could only connect to node in U2, and every node in U2 could only connect node in U1. There should be no edge that connects two nodes that are both in U1, and there should be no edge that connects two nodes that are both in U2. Therefore, for any graph G = {V,E}, where V is a set of nodes, if we could prove that all nodes in V could be divided into U1 and U2, and each node in the individual group could connect to nodes only in the other group there is not cycle in G, then we can prove that G is a bipartite graph.

For a graph to be a tree, the graph should be acyclic. A tree like graph 1 could be redrawn to be graph 2, which has multiple levels. Since the nodes in any levels could connect to only the level above it and the level below it, all levels could be divided to two sets, where every other level of nodes belongs into one set, and the node in one set is connect to the node only in the other set, like graph 3. Then, we could prove that all tree graphs are bipartite graph.

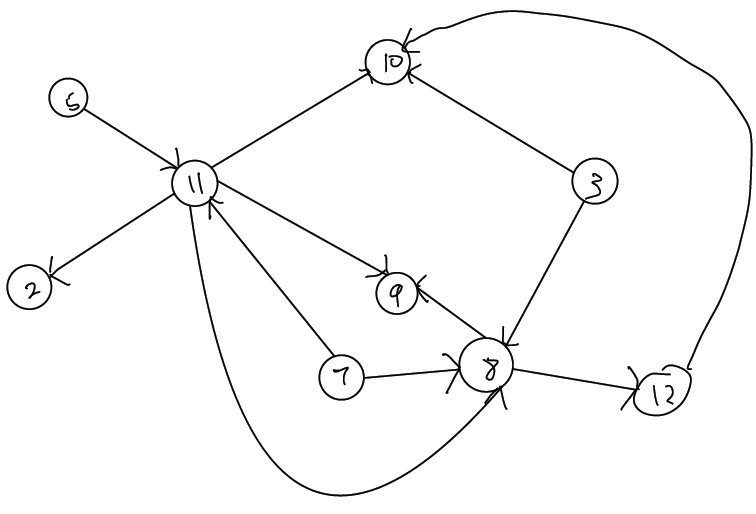
Graph 1

Graph 2

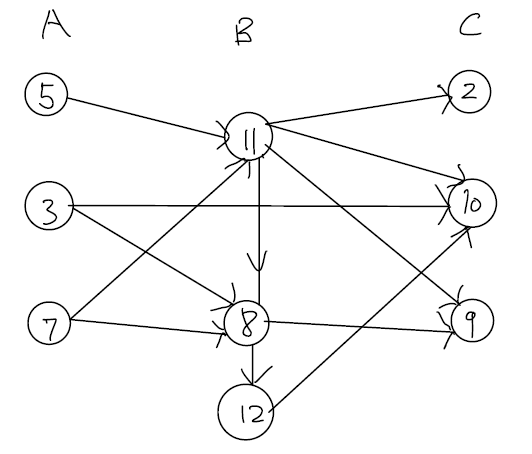
Grpah 3

3. (10 points) A directed acyclic graph (DAG) is a directed graph without cycles (the underlying graph may have cycles, so it is not a tree). Show that a DAG has a labeling of its nodes (that is a labeling *1, …, n* of its nodes) such that every arc goes from a lower-numbered node to a higher-numbered node. (You need to find a way to do it, and also show that is always possible)

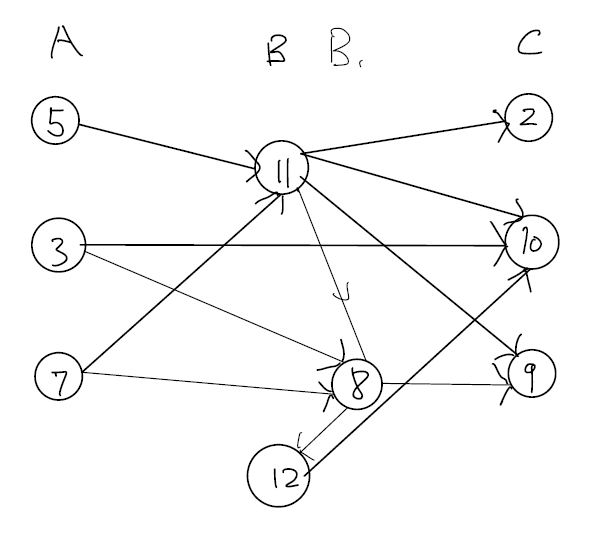
A DAG is a directed graph without cycles and one example could be like graph 1.

Graph 1

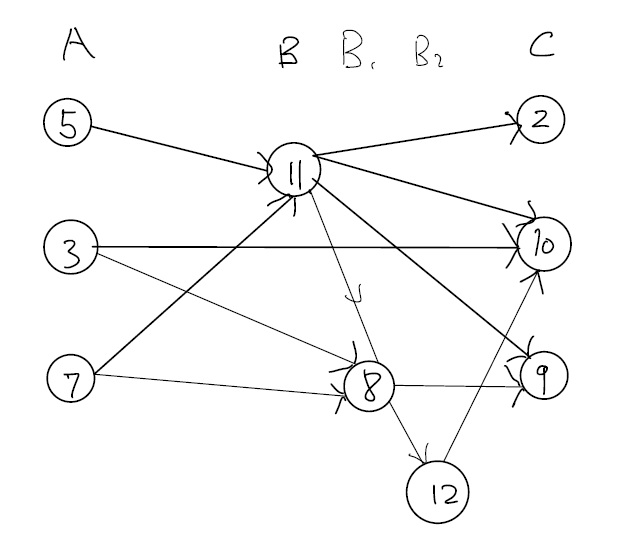
Since there is no cycle, there must be nodes that do not have outdegree (like nodes 5, 3, 7) and nodes that do not have indegree (like nodes 2,10,9). Therefore, we could always redraw a DAG in the layout of neural network, where all nodes that do not have outdegree are in the input layers, and all nodes that do not have indegree are in the output layers, and those nodes that have both non-zero outdegree and nonzero indegree are in the hidden layers, like graph 2. Since the input and output layers are defined in this way, it is guaranteed that no nodes in input layers or output layers are connected to the nodes in the same group.

Graph 2

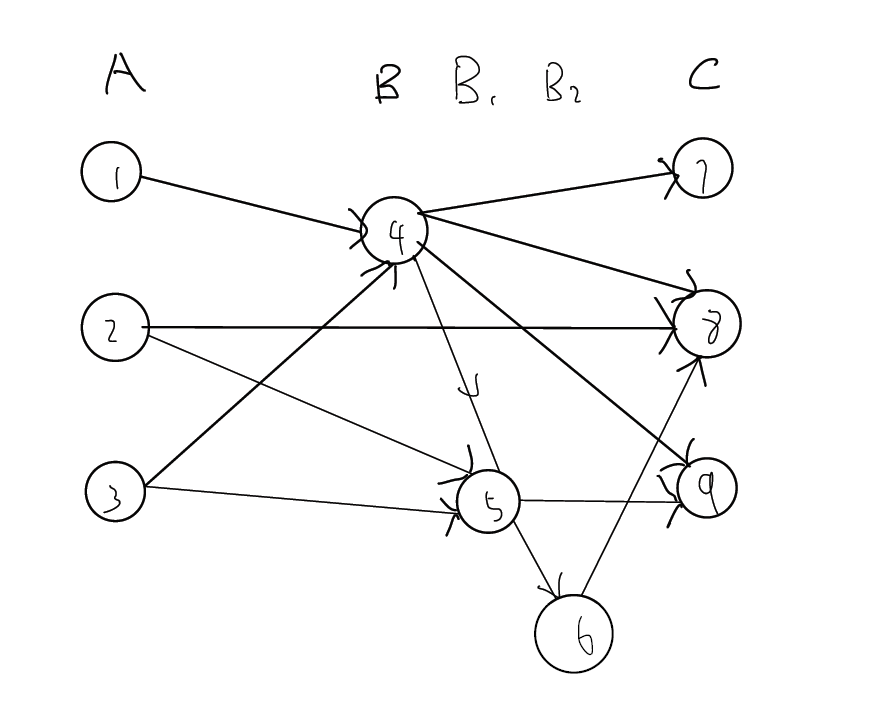
Next, we need to use an alrogithm to resolve the edges between nodes in hidden layer(layer B). To do so, the alrogithm should pick a random node in layer B first, and in this example it could start with node 11. If there are nodes that are directed from this node, the alrogithm will create a consectuive layer and push the these nodes into the layer, like graph 3.

Graph 3

Then, the algorithm jumps to the next node, which is node 8 in this example, and repeats the execution, like graph 4. While in this example, the algorithm should terminate now, in the worst scenario, runtime will be O(n^2) (n is the number of nodes in the hidden layer). However, since there is no cycle in DAG, it is guaranteed that this algorithm will terminate eventually.

Graph 4

Now we could relabel the nodes. The rank of number of labels of nodes in the same layer does not matter, since no nodes in the same layer are connected. Therefore, we just need to ensure the largest number of label in any layer (except the output layer) is smaller than the smallest number of label in the layer right to it, like graph 5.

Graph 5

As the algorism above shows, a DAG always have a labeling of its nodes such that every arc goes from a lower-numbered node to a higher-numbered node.