- Please note that one of the main goals of this course is to design efficient algorithms. So, there are points for efficiency even though we may not explicitly state this in the question.
- Unless otherwise mentioned, assume that graphs are given in adjacency list representation.
- In the lectures, we have discussed an $O(V + E)$ algorithm for finding all the SCCs of a directed graph. We can extend this algorithm to design an algorithm `CreateMetaGraph(`G`)` that outputs the meta-graph of a given directed graph in $O(V+E)$ time. For this homework, you may use `CreateMetaGraph(`G`)` as a sub-routine.
- The other instructions are the same as in Homework-0.

There are 6 questions for a total of 100 points.

1. An undirected graph is said to be *connected* iff every pair of vertices in the graph are reachable from one another. Prove the following statement:

   Any connected undirected graph with n nodes has at least $(n-1)$ edges.

   We will prove the statement using Mathematical Induction. The first step in such a proof is to define the propositional function. Fortunately for this problem, this is already given in the statement of the claim.

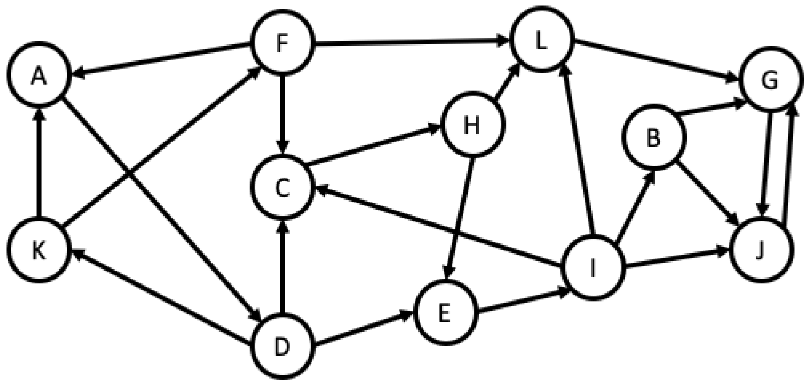
   $P(n)$: Any connected undirected graph with n nodes has at least $(n-1)$ edges.

   The base case is simple. $P(1)$ holds since any connected graph with 1 node having at least 0 edges, is indeed true. For the inductive step, we assume that $P(1), P(2), ..., P(k)$ holds for an arbitrary $k \geq 1$, and then we will show that $P(k+1)$ holds. Consider any connected graph G with $(k+1)$ nodes ~~and k edges~~. You are asked to complete the argument by doing the following case analysis:

   (a) (5 points) Show that if the degrees of all nodes in G is at least 2, then G has at least k edges.

   (b) (5 points) Consider the case where there exists a node v with degree 1 in G. In this case, consider the graph G' obtained from G by removing vertex v and its edge. Now use the induction assumption on G' to conclude that G has at least k edges.

2. Consider the following directed graph and answer the questions that follow:



   (a) (1 point) Is the graph a DAG?

   (b) (2 points) How many SCCs does this graph have?

(c) (1 point) How many source SCCs does this graph have?
(d) (2 points) What is the distance of node B from the A?
(e) (2 points) Suppose we run the DFS algorithm on the graph exploring nodes in alphabetical order. Given this, what is the pre-number of vertex F?
(f) (2 points) Suppose we run the DFS algorithm on the graph exploring nodes in alphabetical order. Given this, what is the post-number of vertex J?

3. (20 points) Suppose a degree program consists of n mandatory courses. The *prerequisite graph* G has a node for each course, and an edge from course u to course v if and only if u is a prerequisite for v. Design an algorithm that takes as input the adjacency list of the prerequisite graph G and outputs the minimum number of quarters necessary to complete the program. You may assume that there is no limit on the number of courses a student can take in one quarter. Analyse running time and give proof of correctness of your algorithm.

4. A particular video game involves walking along some path in a map that can be represented as a directed graph $G = (V, E)$. At every node in the graph, there is a single bag of coins that can be collected on visiting that node for the first time. The amount of money in the bag at node v is given by $c(v) > 0$. The goal is to find what is the maximum amount of money that you can collect if you start walking from a given node $s \in V$. The path along which you travel need not be a simple path.

   Design an algorithm for this problem. You are given a directed graph $G = (V, E)$ in adjacency list representation and a start node $s \in V$ as input. Also given as input is a matrix C, where $C[u] = c(u)$. Your algorithm should return the maximum amount of money that is possible to collect when starting from s.

   (a) (10 points) Give a linear time algorithm that works for DAG's.
   (b) (10 points) Extend this to a linear time algorithm that works for any directed graph. (*Hint: Consider making use of the meta-graph of the given graph.*)

   Give running time analysis and proof of correctness for both parts.

5. Given a directed graph $G = (V, E)$ that is not a strongly connected graph, you have to determine if there exists a pair of vertices $u, v \in V$ such that the graph $G' = (V, E \cup \{(u, v)\})$ is strongly connected. In other words, you have to determine whether there exists a pair of vertices $u, v \in V$ such that adding a directed edge from u to v in G converts it into a strongly connected graph. Design an algorithm for this problem. Your algorithm should output "yes" if such an edge exists and "no" otherwise.

   (a) (10 points) Give a linear time algorithm that works for DAG's.
   (b) (10 points) Extend this to a linear time algorithm that works for any directed graph. (*Hint: Consider making use of the meta-graph of the given graph.*)

   Give running time analysis and proof of correctness for both parts.

6. (20 points) Let us call any directed graph $G = (V, E)$ *one-way-connected* iff for all pair of vertices u and v at least one of the following holds:

   (a) there is a path from vertex u to v,
   (b) there is a path from vertex v to u.

   Design an algorithm to check if a given directed graph is one-way-connected. Give running time analysis and proof of correctness of your algorithm.