- 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 82 points.

---

1. A tree is defined as an undirected graph that does not contain any cycles. An undirected graph is said to be connected iff for every pair of vertices, there is a path between them. For this question, you have to show the following statement:

   Any connected undirected graph with n vertices and $(n-1)$ edges is a tree.

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

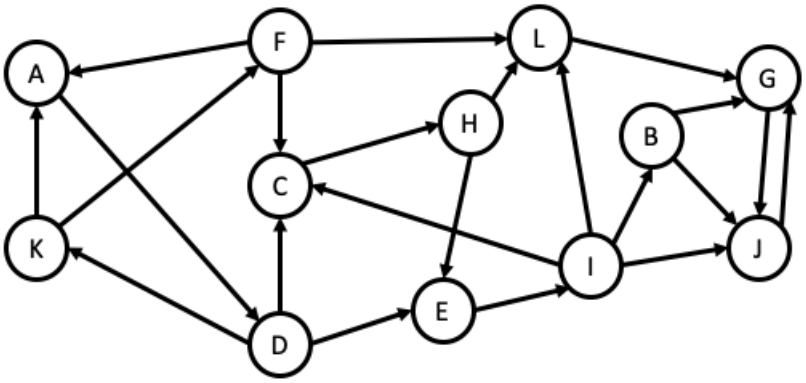
   $P(n)$: Any connected undirected graph with n vertices and $(n-1)$ edges is a tree.

   The base case is simple. $P(1)$ holds since any graph with 1 node and 0 edges is indeed a tree. 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:

   (a) (3 points) Show that G has a node v with degree 1.

   (b) (2 points) 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 is a tree.

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



   (a) (½ point) Is the graph a DAG?

   (b) (1 point) How many SCCs does this graph have?

   (c) (½ point) How many source SCCs does this graph have?

(d) (1 point) Suppose we run the DFS algorithm on the graph exploring nodes in alphabetical order. Given this, what is the pre-number of vertex F?

(e) (1 point) Suppose we run the DFS algorithm on the graph exploring nodes in alphabetical order. Given this, what is the post-number of vertex J?

(f) (1 point) Is it possible to add a single edge to this graph so that the graph becomes a strongly connected graph? If so, which edge would you add?

3. (18 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 on 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 possible to collect starting from s.

   (a) (9 points) Give a linear time algorithm that works for DAGs.

   (b) (9 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) (9 points) Give a linear time algorithm that works for DAGs.

   (b) (9 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. (18 points) Given a directed acyclic graph $G = (V, E)$ and a vertex u, design an algorithm that outputs all vertices $S \subseteq V$ such that for all $v \in S$, there is an even length simple path from u to v in G.
   (A simple path is a path with all distinct vertices.)

   Give running time analysis and proof of correctness.