1. A bipartite graph is a graph whose vertices can be partitioned into two sets such that there are no edges between vertices in the same set. Prove that an undirected graph is bipartite if and only if it contains no cycles of odd length.
   (**10 points**)
   Solution:
   Perform a DFS on the graph and color vertices alternately using two colors, red and blue. The graph is bipartite if and only if there is no monochromatic edge.

   ($\Longrightarrow$) If an undirected graph is bipartite, then it contains no cycles of odd length.
   If the graph is bipartite, there should be no monochromatic edge. Any two vertices of the same color would thus have paths with an even number of edges connecting them. Thus, there cannot be odd length cycles.

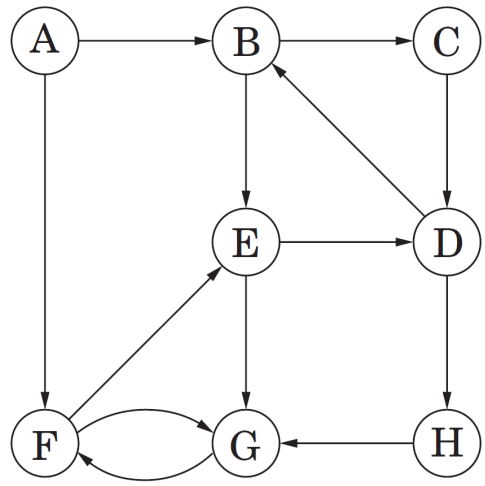
   ($\Longleftarrow$) If an undirected graph contains no cycles of odd length, then it is bipartite.
   If an undirected graph has no odd-length cycles, then it means it could be a tree or may have one or more even length cycles. For an even-length cycle, the coloring algorithm would assign half the vertices the red color and the other half a blue color. This means there can be no monochromatic edge and so the graph is bipartite.

   Grading scheme: 5 points for each part, 3 or 6 points for partial credit.

2. Your job is to prepare a lineup of n awardees at an award ceremony. You are given a list of m constraints of the form "i wants to receive an award before j." If you violate a constraint, it might upset the affected award recipient (i) and then i may leave the ceremony. Give an algorithm that prepares such a lineup, (or says that it is not possible) in $O(m+n)$ time.
(**8 points**)
Solution:
We will formulate this as a graph problem. Create a directed graph G with each awardee denoting a vertex. For every "i wants to receive an award before j" constraint, add an edge $\langle i, j\rangle$. *Preparing the lineup* would mean ordering the vertices. If there is an edge $\langle i, j\rangle$ in G, then i should appear before j in the order. Now, let us consider when an ordering is not possible. One such case is when there are edges $\langle i, j\rangle$ and $\langle j, i\rangle$. In this case, there is clearly no ordering. Similarly, when there is an edge triple $\langle i, j\rangle$, $\langle j, k\rangle$, and $\langle k, i\rangle$, an ordering of vertices is not possible. Thus, we want G to be acyclic or a DAG. If G has a cycle, then no ordering is possible. We can perform a DFS to check if there is a back edge (and a cycle). If we don't find a back edge, then the graph is a DAG. One possible ordering of vertices in a DAG is through a topological sort (specifically, decreasing order of post identifiers when performing a DFS). We can place the vertex appearing first in the topological sort at the head of the line, the second vertex at the second position, and so on. Since both DFS and topological sort are linear time, the overall approach takes $\mathrm{O}(m+n)$ time.

Grading scheme: 2 points for modeling + 3 points for explaining why there cannot be cycles + 2 points for mentioning topological sort + 1 point for comment on running time.

3. Run the strongly connected components algorithm on the following directed graph. Whenever there is a choice of vertices, pick the one that is alphabetically first.



(**8 points**)
Solution: The SCCs are {A} and {B, C, D, E, F, G, H}.

Grading scheme: 3 points for drawing G^R and showing pre/post numbers, 2 points for giving postorder vertex ordering, 3 points for identifying the two components correctly. 2/4 points for partial or incorrect responses.

4. Prove that the following statement is true, or give a counterexample to disprove it: when we perform depth-first search on a directed graph G, for any nodes u and v, the two intervals $[\text{pre}(u), \text{post}(u)]$ and $[\text{pre}(v), \text{post}(v)]$ are either disjoint or one is contained within the other.
(**7 points**)
Solution:
The statement is true.
Proof: Given two vertices u and v, assuming u is visited before v ($\text{pre}(u) < \text{pre}(v)$), after performing DFS, we can have one of three cases:

(a) $\text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u)$ (v's interval contained in u's interval)
(b) $\text{pre}(u) < \text{post}(u) < \text{post}(v) < \text{post}(u)$ (disjoint)
(c) $\text{pre}(u) < \text{pre}(v) < \text{post}(u) < \text{post}(v)$ (interleaved)

To show that the statement is true, we need to show that case (c) above cannot happen for any graph. We know that vertices visited in a DFS are stored in a stack, and the stack pop ordering is LIFO. If v is a descendant of u, then because of LIFO ordering, we must have $\text{post}(v) < \text{post}(u)$ (case (a)). If v is not a descendant of u, then because of LIFO ordering, we have case (b). Thus case (c) can never happen.

If v is visited before u, we can use a similar argument to show that the interleaved interval case does not happen.

Grading scheme: 2 points for attempt + 1/3/5 points depending on clarity of solution.

5. Design a linear-time algorithm which, given an undirected graph G and a particular edge e in it, determines whether G has a cycle containing e.
(**7 points**)
Solution:
Let the edge $e = \langle u, v \rangle$. If G has a cycle containing e, then there must exist a path from u to v in $G - \{e\}$. Delete e from G, invoke the explore routine from u, and check if v is visited. This takes linear time because at most m edges are visited.

Grading scheme: 3 points for observation + 3 points for mentioning edge deletion and explore (only 1 point if the solution says perform DFS from u) + 1 point for running time comment.