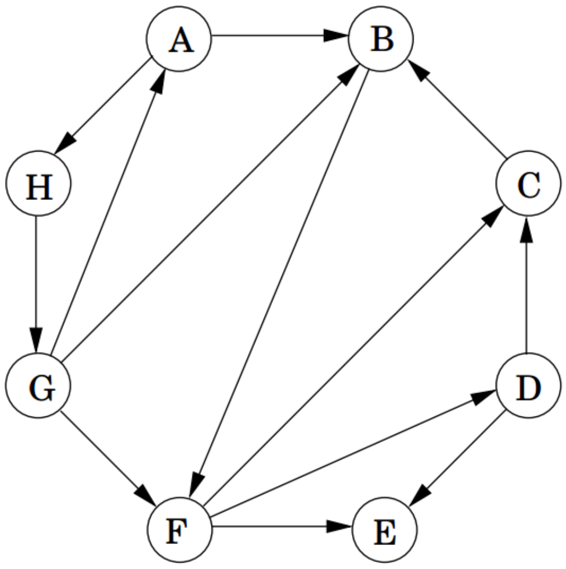
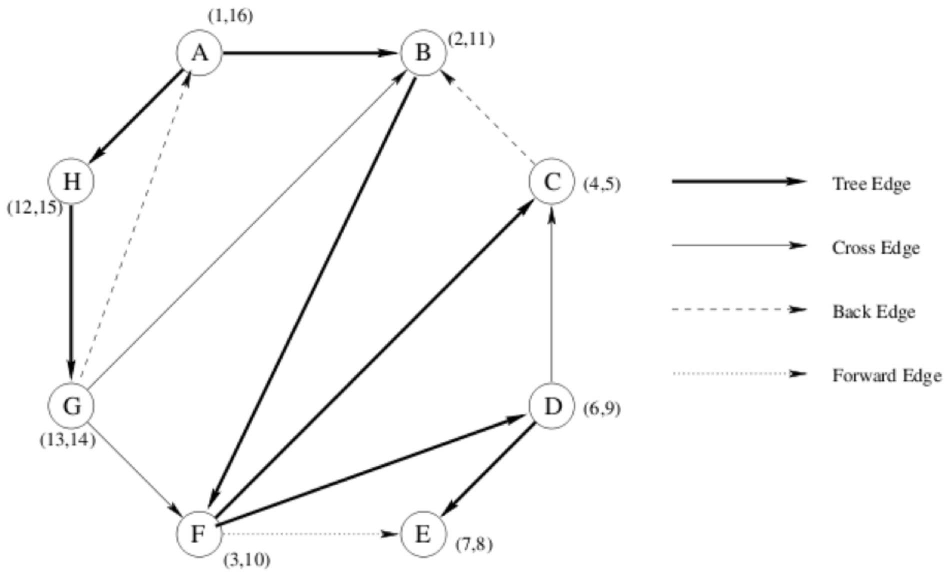
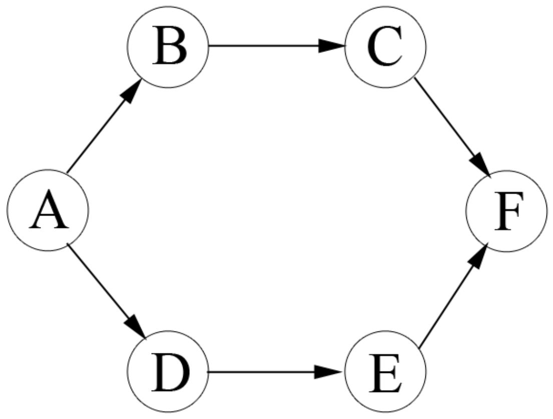
Problem 1 (10 Points). Perform a depth-first search on the following graph. Whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, forward edge, back edge, or cross edge, and give the `pre` and `post` number of each vertex (for this problem, you may scan handwritten drawings).



Solution:



Problem 2 (5 Points). Consider the following DAG:



How many topological orderings does it have? List them.

Solution: In any topological ordering, we must have A before B and D, B before C, D before E, and F after C and E. Therefore, A must be fist, and F last. The possible orderings are thus:

$$ABCDEF$$
$$ABDCEF$$
$$ABDECF$$
$$ADBCEF$$
$$ADBECF$$
$$ADEBCF$$

Thus there are 6 possible topological orderings

Problem 3 (20 Points). A *bipartite* graph is an undirected graph $G = (V, E)$ whose nodes can be partitioned into two sets V_1 and V_2 (i.e. $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that all edges have one endpoint in V_1 and the other endpoint in V_2.

(a) Give a linear-time algorithm to determine whether an undirected graph is bipartite. If the graph is bipartite, your algorithm should output V_1 and V_2.

(b) Prove that a graph is bipartite if and only if it has no cycles of odd length.

Solution:

(a) If we give nodes color attributes, say red or blue, then the graph is bipartite if and only if it admits a coloring of nodes such that every edge goes between nodes of different colors. Thus, checking if a graph is bipartite is equivalent to checking if it is *two-colorable*.

Since in a two-colorable graph, every edge must be between nodes of different colors, every adjacent nodes in the DFS tree will have different colors. Thus we can check for two-colorability by coloring the nodes in the DFS tree alternate colors, and checking that no non-tree edges go between nodes of different colors. The algorithm is as follow:

```
– Set visited(v) = false for all $v$
– For every node $v \in V$:
    ∗ If visited(v) = false: call explore(v, red).
– If we never reported that the graph is not bipartite, let $V_1$ be the set of red nodes and $V_2$
  be the set of blue nodes.
```

Where `explore(v, c)` is defined as:

```
– visited(v) = true
– color(v) = c
– Let $c'$ be be the opposite color of $c$.
– For each edge $(v, w) \in E$:
    ∗ If visited(w) = false, call explore(w, c′)
    ∗ Otherwise, check that color(v) ≠ color(w). If not, stop and output that the graph
      is not bipartite.
```

This algorithm is just a DFS with a constant number of extra operations per call to explore (and hence per node), so the running time is still $O(|V| + |E|)$

(b) If a graph has a cycle of odd length, the cycle itself is not two-colorable, so the entire graph definitely cannot be two-colorable.

Conversely, if a graph is not bipartite, it means that our DFS algorithm from part (a) failed. Let (u, w) an edge that has endpoints of the same color. It must be that u and w are both descendants of some red node v in the DFS tree (otherwise, they are in separate connected components, and the edge (u, w) does not exist). Consider the cycle formed by going from v to u in the DFS tree, crossing the edge (u, w), and going back up the DFS tree to v. How many edges are there in this cycle? If u and w are blue, then the paths from v to u and v to w have an odd number of edges. If u and w are red, then the paths have an even number of edges. In either case, the union of the two paths has an even number of nodes. Adding the edge (u, v) gives a cycle of odd length.

Problem 4 (10 Points). Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. Find an algorithm that works directly with this graph representation, and computes the minimum number of quarters necessary to complete the curriculum (assuming that a student can take any number of courses in one quarter). The running time of your algorithm should be linear. Prove its correctness and running time.

Solution: The prerequisite graph is a dag, so it makes sense to topologically order the graph first. Now, let the first term be term 0, second be term 1, etc. Let $q(i)$ be the length of the longest path ending at node i. We claim that if we take each course i in term $q(i)$, then all prerequisites will always be satisfied.

Indeed, suppose course i has a prerequisite j. Let p be the longest path ending at j. Then if we add the edge (j, i), we get a path to i of length $q(j) + 1$. Thus, $q(i) > q(j)$, so course j is taken before course i.

It should also be clear that we can never take course i before term $q(i)$. This is because there is a path p to i of length $q(i)$, and each previous node on p (there are $q(i)$ of them) must be taken in different semesters. Therefore, there must be at least $q(i)$ term completed before taking course i, so course i must be taken no earlier than term $q(i)$.

This means the minimum number of terms necessary is 1 greater than the length of the longest path. In a general graph, the longest path is ill-defined. However, in a dag, since there are no cycles, the longest path is well defined and efficiently computable. The longest path ending at node i must go through one of i's parents, so it is 1 plus the length of the longest path to that parent. Thus, $q(i)$ is equal to 1 plus the maximum of $q(j)$ over all parents j of i. This gives us the following algorithm for computing the $q(i)$:

- Topologically sort the nodes
- For each node i in topological order, set $q(i) = 0$ is i has indegree 0, and

$$q(i) = 1 + \max_{(j,i)\in E} p(j)$$

Topologically sorting takes linear time, and setting $q(i)$ takes time proportional to the indegree of i. Summing over all i, the time for the second step is $O(|V| + |E|)$. Therefore, the overall time is $O(|V| + |E|)$.

To find the number of terms necessary, find the largest $q(i)$ and add 1. This adds $O(|V|)$ time, so the time is still linear.

Problem 5 (10 Points). We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

(a) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

(b) What algorithm should be applied to solve the problem?

Solution:

(a) The graph is a directed graph where the nodes are labeled by triples of integers (x, y, z) where $0 \le x \le 10$, $0 \le y \le 7$, and $0 \le z \le 4$, where x, y, and z represent the amount of water in the 10-, 7-, and 4-pint containers, respectively. Since at any point, there will be 11 pints total, we only need to consider nodes where $x + y + z = 11$. There is an edge from (x, y, z) to (x', y', z') if there is a pouring from one container to another such that, after the pouring, the 10-pint container has x' pints, the 7-pint has y' pints, and the 4-pint has z' pints.

(b) We run `explore` on the node with $(x, y, z) = (0, 7, 4)$, and stop when we've reached a node (x', y', z') with either $y' = 2$ or $z' = 2$.

Problem 6 (10 Points). The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a linear-time algorithm.

(a) Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear time.

(b) Suppose it now turns out that the mayor's original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the town hall. Formulate this weaker property as a graph-theoretic problem, and carefully show how it too can be checked in linear time.

Solution:

(a) Create a node for every intersection, create an edge from node u to node v if one of the roads leading from intersection u goes to intersection v next. Then this problem is equivalent to whether the graph is strongly connected. This can indeed be solved in linear time by running the strongly connected components algorithm, and returning true if there is exactly one strongly connected component.

(b) This problem is equivalent to whether town hall is in a sink strongly connected component, which can be accomplished in linear time by running the strongly connected components algorithm, and checking that town hall is indeed in a sink.

Problem 7 (35 Points). When on your favorite social network, do you ever notice that your friends tend to have more friends than you do? Don't despair! There is a good reason that most people have fewer friends than there friends do, and it is known as the Friendship Paradox.

Let's model a social network as an undirected graph, with nodes being people, and edges being friendships. Then the degree $d(v)$ of a node v is the number of friends that person has. Let $f(v)$ be the average degree of the neighbors of v (i.e. the average number of friends v's friends have). That is

$$f(v) = \frac{1}{d(v)} \sum_{(v,u) \in E} d(u)$$

If a node v has no neighbors, we let $f(v) = 0$. Let D be the average of $d(v)$ over all v, and let F be the average of $f(v)$ over all v. We want to compare F to D.

(a) Show that $D = 2|E|/|V|$.

(b) Give an example of a family of graphs for $|V| = 2, 3, ...$ where $F = \Omega(D|V|)$.

(c) Give an example of a family of graphs for $|V| = 2, 3, ...$ where $F = D$.

(d) Show that

$$\frac{F}{D} = \frac{1}{2|E|} \sum_{(v,u)\in E} \left(\frac{d(u)}{d(v)} + \frac{d(v)}{d(u)} \right)$$

For positive x, $x + 1/x \geq 2$, with equality if and only if $x = 1$. (Proof: you can verify that $x + 1/x = 2 + \frac{(x-1)^2}{x} \geq 2$, with equality if and only if $(x-1)^2 = 0$, i.e. $x = 1$). Further, it is easy to see that $x + 1/x \leq \max(x, 1/x) + 1$.

(e) Use these facts to conclude that all graphs have $D \leq F \leq D|V|/2$.

(f) Precisely categorize which graphs have $F = D$ and which have $F > D$.

Solution:

(a)

$$D = \frac{1}{|V|} \sum_{v\in V} d(v) = \frac{1}{|V|} \sum_{v\in V} \sum_{(v,w)\in E} 1$$

We sum over all nodes v and all edges incident on v. Notice that each edge (v, w) is present in the sum exactly twice: once for v and once for w. Therefore, the sum evaluates to $2|E|$. Thus $D = 2|E|/|V|$.

(b) Consider the graph where one node v_0 is connected to every other node, and every other node is connected only to v_0. Thus $d(v_0) = |V| - 1$ and $d(v) = 1$ for $v \neq v_0$. Thus, the average degree is

$$D = 2\frac{|V| - 1}{|V|}$$

Since all the neighbors of v_0 have degree 1, $f(v_0) = 1$. For any $v \neq v_0$, v_0 is the only neighbor, so $f(v) = |V| - 1$ for $v \neq v_0$. This gives

$$F = \frac{(|V|-1)(|V|-1)+1}{|V|} > \frac{(|V|-1)^2}{|V|} = (|V|-1)D/2 \in \Omega(D|V|)$$

(c) In any graph where all nodes have the same degree, $F = D$. One example is the complete graph, where $F = D = |V| - 1$. Another example is the graph with no edges, where $F = D = 0$.

(d)

$$F = \frac{1}{|V|} \sum_{v\in V} f(v) = \frac{1}{|V|} \sum_{v\in V} \sum_{(v,w)\in E} \frac{d(w)}{d(v)}$$

There are two terms for every edge $(v, w) \in E$: $d(w)/d(v)$ and $d(v)/d(w)$. Therefore, the sum becomes

$$F = \frac{1}{|V|} \sum_{(v,w)\in E} \left(\frac{d(w)}{d(v)} + \frac{d(v)}{d(w)} \right)$$

Dividing by the expression for D from part a gives the desired result.

(e) Letting $x = d(w)/d(v)$, we see that each term in the sum for F/D is at least 2. Therefore,

$$\frac{F}{D} \geq \frac{1}{2|E|} \sum_{(v,w)\in E} 2 = 1$$

So $F \geq D$. On the flip side, for any edge (v, w), both v and w have degree at least 1 (since they are each other's neighbors), and all nodes have degree at most $|V| - 1$ (no one is a neighbor of themselves). Thus,

$$\frac{d(w)}{d(v)} + \frac{d(v)}{d(w)} \leq (|V| - 1) + 1 = |V|$$

Thus

$$\frac{F}{D} \leq \frac{1}{2|E|} \sum_{(v,w)\in E} |V| = \frac{|V|}{2}$$

This gives $F \leq D|V|/2$. We can also see this last inequality by noting that $f(v) \leq |E|$, so $F \leq |E| = D|V|/2$.

(f) Notice that we have $F = D$ if and only if $\frac{d(w)}{d(v)} + \frac{d(v)}{d(w)} = 2$ for all edges $(u, v) \in E$, which occurs if and only if $d(v) = d(w)$ for all edges $(u, v) \in E$. Thus, we have $F = D$ if all edges are between nodes of the same degree. This implies that for every connected component C, all nodes in C must have the same degree (or else there are two nodes u and v connected by a path such that $d(u) \neq d(v)$. But then there must be two neighboring nodes in the path that have different degree). Conversely, if every connected component has constant degree, then every edge must be between two nodes of the same degree.

Therefore, $F = D$ if and only if all connected components have constant degree, and $F >$ if there is some connected component that has nodes of different degrees.

Total points: 100