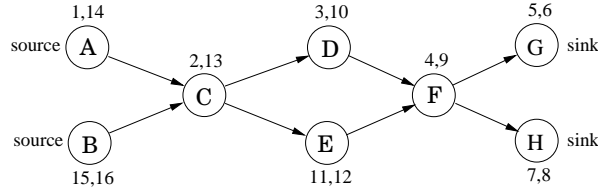


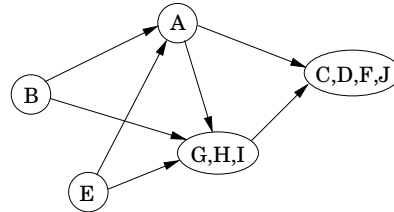
3. *Exercise 3.3.*

In the figure below, the algorithm finds the ordering: B, A, C, E, D, F, H, G . There are 8 possible orderings.



4. *Exercise 3.4(i).*

- A DFS on G^R (using alphabetical order) gives us the following ordering on the nodes (decreasing post numbers): $C, J, F, H, I, G, D, A, E, B$.
- Running DFS on G (using the above ordering) then reveals the SCCs in the following order: $\{C, D, F, J\}$ (sink), $\{G, H, I\}$, $\{A\}$, $\{E\}$ (source), $\{B\}$ (source).



- Meta-graph:
- To make the graph strongly connected, we need to add two edges, from a node in the unique sink SCC to nodes in each of the two source SCCs.

5. *Exercise 3.9.*

First, the degree of each node can be determined by going through the adjacency list. The array `twodegree` can then be set in a second pass through the adjacency list.

```

for all  $u \in V$ :
    degree[u]  $\leftarrow$  0
    twodegree[u]  $\leftarrow$  0
for all  $u \in V$ :
    for all  $\{u, w\} \in E$ :
        degree[u]  $\leftarrow$  degree[u] + 1
for all  $u \in V$ :
    for all  $\{u, w\} \in E$ :
        twodegree[u]  $\leftarrow$  twodegree[u] + degree[w]
    
```

This algorithm directly implements the definition of `degree` and `twodegree`. Its running time is $O(|V|)$ for the initializations, plus the time for the two loops; in each loop, a single pass is made through the adjacency list. Thus the total running time is $O(|V| + |E|)$, linear.

6. *Exercise 3.14.*

We will keep an array `in[u]` which holds the indegree (number of incoming edges) of each node. For a source, this value is zero. We will also keep a linked list of source nodes.

```

(Set the in array.)
for all  $u \in V$ : in[u]  $\leftarrow$  0
for all edges  $(u, w) \in E$ : in[w]  $\leftarrow$  in[w] + 1
    
```

(Check for sources.)
 $L \leftarrow$ empty linked list
for all $u \in V$:
 if $\text{in}[u]$ is 0: add u to L

for $i = 1$ to $|V|$:
 Let u be the first node on L , output it and remove it from L
 (Remove u , update indegrees.)
 for each edge $(u, w) \in E$:
 $\text{in}[w] \leftarrow \text{in}[w] - 1$
 if $\text{in}[w]$ is 0: add w to L

The total running time is $O(|V| + |E|)$ (in the innermost loop, over the course of the algorithm, each edge is examined exactly once).

7. The diameter of an undirected graph $G = (V, E)$ is the maximum distance between any pair of nodes.

- (a) Show that if each node has degree at most two, then the diameter of G must be at least $\log_2 n - 1$.
(In fact, using a careful argument, a much stronger result can be shown: that the diameter is $\Omega(n)$. This is worth trying out on your own.)

Suppose each node has degree at most two. Fix any node $u_0 \in V$. We will show by induction that the number of nodes at distance exactly k from u_0 is at most 2^k .

The base case, $k = 0$, holds since u_0 itself is the only node at distance 0. Suppose the hypothesis holds up to k ; we will show it for $k + 1$ as well. Any node at distance $k + 1$ from u_0 is adjacent to a node at distance k . Since nodes have at most two neighbors, the number of nodes at distance $k + 1$ is at most twice the number of nodes at distance k . By the inductive hypothesis, this is at most $2 \cdot 2^k = 2^{k+1}$. This completes the induction.

Therefore, the total number of nodes within distance k of u_0 (that is, at distance k or below) is at most $1 + 2 + \dots + 2^k < 2^{k+1}$. Let D be the diameter of the graph; so all nodes lie within distance D of u_0 . It follows that $|V| \leq 2^{D+1}$, so $D \geq \log_2 |V| - 1$.

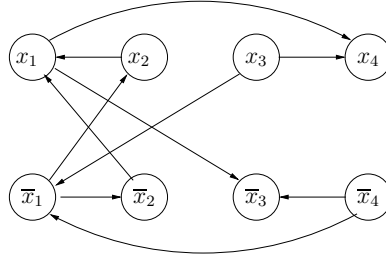
- (b) Generalize part (a) to the case where each node has degree at most Δ , where Δ is some number greater than one.

The same argument shows the number of nodes within distance k of u_0 is at most $1 + \Delta + \Delta^2 + \dots + \Delta^k = (\Delta^{k+1} - 1)/(\Delta - 1)$. Thus the diameter must be at least $\log_{\Delta}(|V|(\Delta - 1)) - 1$.

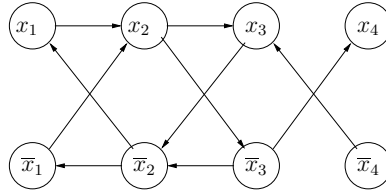
(Again, with a more careful argument, a stronger result can be shown, in which the base of the logarithm is reduced from Δ to $\Delta - 1$.)

8. Exercise 3.28.

- (a) The formula has two satisfying assignments, $(x_1, x_2, x_3, x_4) = (\text{true}, \text{false}, \text{false}, \text{true})$ and $(\text{true}, \text{true}, \text{false}, \text{true})$.
- (b) The formula $(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_2) \wedge (x_1 \vee x_2) \wedge (x_3 \vee x_4)$ has no satisfying assignments. In order to satisfy the first three clauses, we need $x_1 = \text{false}$ (try **true**, and you'll run into a contradiction). From the fourth clause, this means $x_2 = \text{true}$. From the second clause, we get $x_3 = \text{true}$. But then the third clause is violated.
- (c) Here's the graph G_I for the instance I presented in the problem.



Here's G_I for the instance I from part (b).



- (d) A path in the directed graph from x to y means that x implies y (that is, $x \Rightarrow y$; or in words, if x is **true**, then y must also be **true**). If x and \bar{x} are in the same connected component, then we get $x \Rightarrow \bar{x}$ and $\bar{x} \Rightarrow x$, a contradiction.
- (e) We will assign values to variables according to the following procedure:

- Pick a sink SCC of G_I and set all *literals* in it to **true** (so if the SCC contains \bar{z} , then z should be set to **false**).
- The negations of these literals form a source SCC (we will explain this below). Remove both the sink and source SCC from the graph. (Thus all variables whose values have been set are removed.)
- Repeat.

By the symmetry of edges in G_I , for each edge $\alpha \Rightarrow \beta$ in the sink SCC (or leading into this SCC), there is another edge $\bar{\beta} \Rightarrow \bar{\alpha}$ elsewhere. Thus the negations of the literals in the sink SCC form a source SCC.

Notice that an implication $\alpha \Rightarrow \beta$ is always satisfied if literal $\beta = \mathbf{true}$. Thus, setting the literals L in the sink SCC to **true** ensures that all implications within (or leading into) this SCC are satisfied. Likewise, $\alpha \Rightarrow \beta$ is always satisfied if $\alpha = \mathbf{false}$. Thus making the negations of the literals L **false** will satisfy all implications within (and leading out of) the source SCC. We recurse on the remaining graph.

- (f) We solve an instance I of 2SAT as follows:
- Build G_I ; this takes linear time.
 - Run the linear-time strongly connected components algorithm on it. If there are k SCCs, the procedure returns an array $scc[u]$ containing the number (1 to k) of each node's SCC.
 - For each variable x , check whether $scc[x]$ equals $scc[\bar{x}]$. If so, the formula is not satisfiable. Otherwise, it is satisfiable. This check takes linear time.