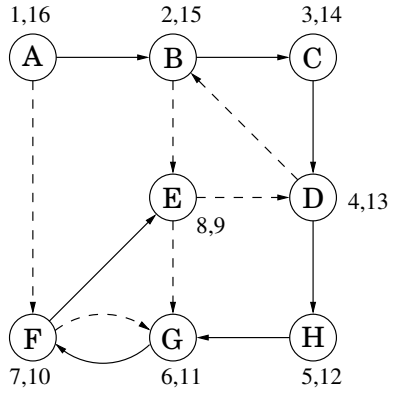
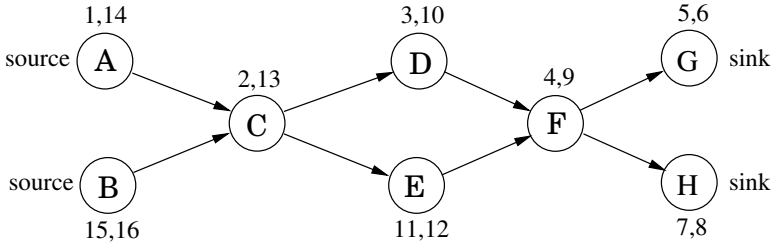
Solutions to Homework Three

CSE 101

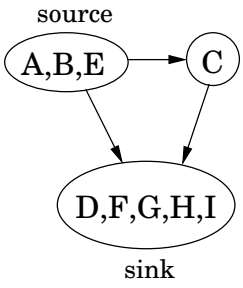
1. *Textbook problem 3.2(a).* Solid edges are tree edges. (D, B), (E, D), (E, G), and (F, G) are back edges.



2. *Textbook problem 3.3.* Here the algorithm finds the ordering: B, A, C, E, D, F, H, G. There are 8 possible orderings.



3. *Textbook problem 3.4(ii).*



The SCCs are found in the order: $\{D, F, G, H, I\}, \{C\}, \{A, B, E\}$. The additional of a single edge (such as $D \to A$) will make the graph strongly connected.

4. *Textbook problem 3.14: another algorithm for topological sorting.*

We will keep an array $\texttt{in}[u]$ 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 nodes u: $\texttt{in}[u] \leftarrow 0$
for all edges $(u, w) \in E$: $\texttt{in}[w] \leftarrow \texttt{in}[w] + 1$

(Check for sources.)
$L \leftarrow$ empty linked list
for all nodes u:
    if $\texttt{in}[u]$ 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 out-edge $(u, w) \in E$:
        $\texttt{in}[w] \leftarrow \texttt{in}[w] - 1$
        if $\texttt{in}[w]$ 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).

5. *Textbook problem 3.15: Computopia.*

   (a) Construct a directed graph G with a node for each intersection and a directed edge for each (one-way) street. Then there is a way to drive from any intersection to any other if and only if G is strongly connected.

   This can be determined in linear time by running the strongly connected components algorithm from class and checking whether there is exactly one SCC.

   (b) Suppose the townhall lies in strongly connected component C of the graph G described above. The intersections reachable from the townhall are precisely the nodes in C and all SCCs reachable from it.

   If there is an SCC $C' \neq C$ which can be reached from C, then there cannot be a path from C' to C (since the SCCs form a dag), and thus the desired property is violated.

   In short:

   property holds $\Leftrightarrow$ the only SCC reachable from C is C itself $\Leftrightarrow$ townhall lies in a sink SCC

   This can be checked easily: run the SCC algorithm to label each node with its SCC; now run `explore` starting at the townhall and check that the only nodes reached are in the same SCC. The running time of either step is linear, and is thus linear overall.

6. *Textbook problem 3.22.* Let G be a directed graph.

   Method 1. There is a vertex from which all other vertices are reachable if and only if G has exactly one source strongly connected component. This can be determined in linear time: run the SCC algorithm, construct the metagraph, use the source-finding algorithm from an earlier homework to return all sources, and check if there is only one source.

   The only step that needs further clarification is the construction of the metagraph. The SCC algorithm returns an array $\texttt{scc}[\cdot]$: for each node u,

   $$\texttt{scc}[u] = \text{the number of the SCC to which } u \text{ belongs.}$$

   We can scan this array to determine the number of SCCs, call it k. We then create a new graph G^M (where M is for "meta") with k nodes $V^M = \{1, 2, \ldots, k\}$. We fill in its edge-set E^M as follows:

   ```
   for each edge (u, v) ∈ E:
      if scc[u] ≠ scc[v]:
         add edge (scc[u], scc[v]) to E^M
   ```

This takes linear time. The resulting G^M might have duplicate edges, but that is not a concern for this problem.

Method 2. In analyzing the SCC algorithm, we discovered the following property: in DFS, the node with highest `post` number lies in a source SCC. This suggests a very simple linear-time algorithm:

```
Run DFS on G
Let u be the node with highest POST number
Run EXPLORE from u
If EXPLORE visits all nodes, return TRUE, else return FALSE
```

7. *Textbook problem 3.25.*

   (a) Suppose u has edges to nodes $w_1, \ldots, w_k$. Then the nodes reachable from u are precisely: u itself, and the nodes reachable from all the w_i. This gives us a simple recursive formula for `cost` values:

   $$\text{cost}[u] = \min\{p_u, \min_{(u,w)\in E} \text{cost}[w]\}.$$

   To make it iterative, it would help if we could make sure to compute all the cost$[w]$ values before we get to cost$[u]$. Well, for a dag this is easy: just handle vertices in reverse topological order! Here's the algorithm, which is linear time because topological sorting is linear time.

   Topologically sort the dag.
   For each node $u \in V$, in reverse topological order:
   cost$[u] = \min\{p_u, \min_{(u,w)\in E} \text{cost}[w]\}$.

   (b) In a general directed graph, if two nodes u, v are in the same SCC, then the nodes reachable from u are identical to those reachable from v, so cost$[u]$ = cost$[v]$. Therefore we do not need to distinguish between nodes in the same SCC, and we can pretty much work with the metagraph, which is a dag!

   • Find the strongly connected components of G.
   • For each SCC C: let the (meta-)price p^*_C for component C be the smallest price of its nodes, that is, $p^*_C = \min_{u\in C} p_u$.
   • Run the dag version of the algorithm (from part (a)) on the metagraph, using these metaprices p^*_C. This returns component metacosts cost$^*[C]$.
   • For each SCC C, and each node $u \in C$: cost$[u]$ = cost$^*[C]$.

   This takes linear time because the algorithm for strongly connected components and the algorithm from part (a) are both linear time.