**COMP 360 Algorithm Design Techniques**

# Assignment 3: Solutions

(1) *Strongly Connected Components*

We use the convention that adjacency lists are given in lexicographic order, and that the vertex list for the entire graph is also given in lexicographic order. For the directed graph $G$ shown in Figure 1, we compute the strongly connected components using the algorithm given in the text and in class.

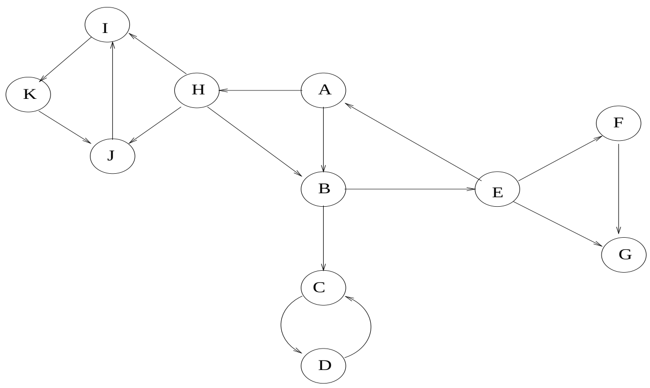


FIGURE 1. Graph $G$.

First, we DFS the directed graph $G$, recording the finish order (or alternatively, the finish time) of each vertex. See Figure 2.

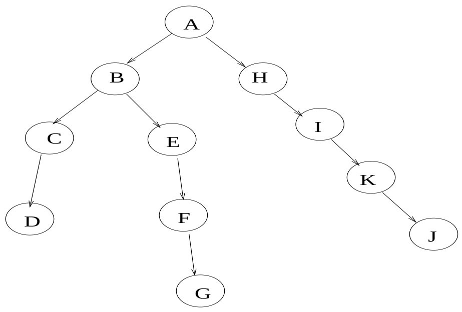


FIGURE 2. The tree produced by the first DFS.

Then we reverse the edge (arc) directions to get a graph $G'$. See Figure 3.

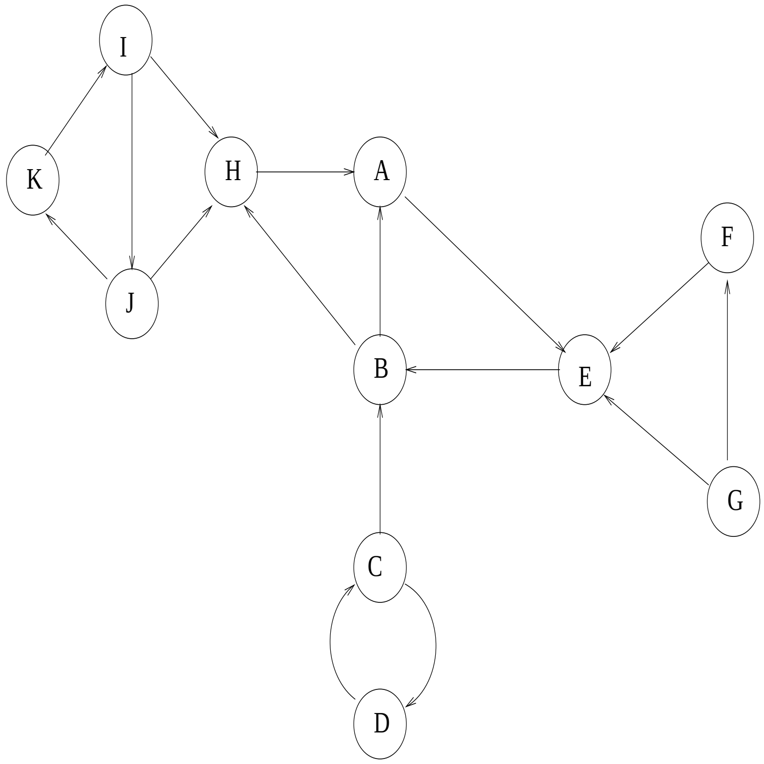


FIGURE 3. Graph $G' = G$ with its arcs reversed.

Finally, we do a DFS on $G'$. As usual, DFS will be making use of an ordered list of all the vertices of $G'$. This time, we *don't* list the vertices in lexicographic order. Instead, we list them according to their finish times in the first DFS, beginning the list with the vertex that finished DFS *last*, and ending the list with the vertex that finished DFS *first*. This list is therefore: $V = A, H, I, K, J, B, E, F, G, C, D$.

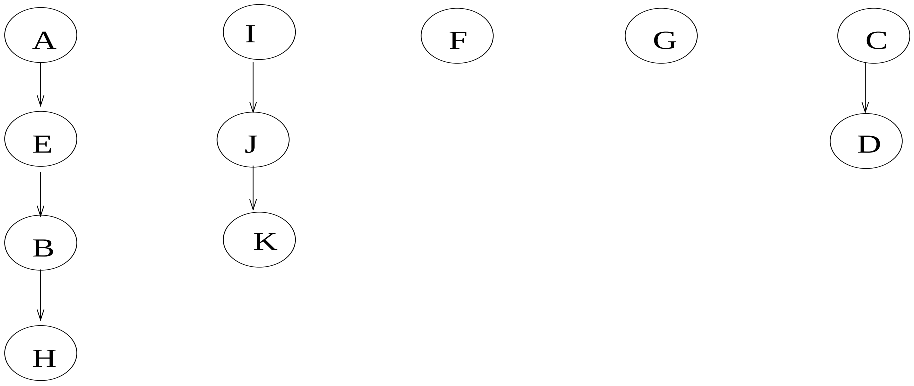


FIGURE 4. The forest found by the DFS of $G'$.

The trees in the forest produced by the second DFS give the vertex sets of the strongly connected components of $G$. (Note that these vertex sets are the same as the ones for the strongly connected components of $G'$.) To get one entire strongly connected component, not just the set of its vertices, take a tree in the second DFS forest, reverse its edges back to their original directions, and add in the edges of $G$ that connect pairs of vertices inside that tree.

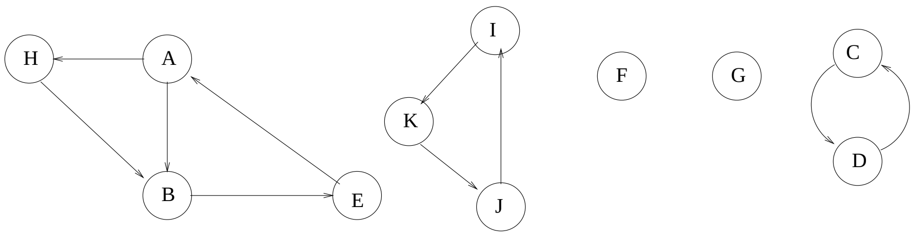


FIGURE 5. The strongly connected components of $G$.

This gives a strongly connected component of $G$. In this way, the trees of the second DFS forest produce *all* the strongly connected components of $G$. Because there are 5 trees in the second DFS forest, there are 5 strongly connected components of $G$. Figure 5 gives the forest of 5 trees produced by the second DFS (using the special ordering of $V$), and Figure 5 gives the 5 corresponding strongly connected components of $G$.

(2) *Dijkstra's Algorithm*

You should obtain the following shortest path tree. The vertices are permanently labelled in the order $1, 3, 7, 4, 2, 6, 9, 8, 5$, and the arcs selected in the order $(1, 3), (3, 7), (1, 4), (1, 2), (4, 6), (6, 9), (4, 8), (2, 5)$.

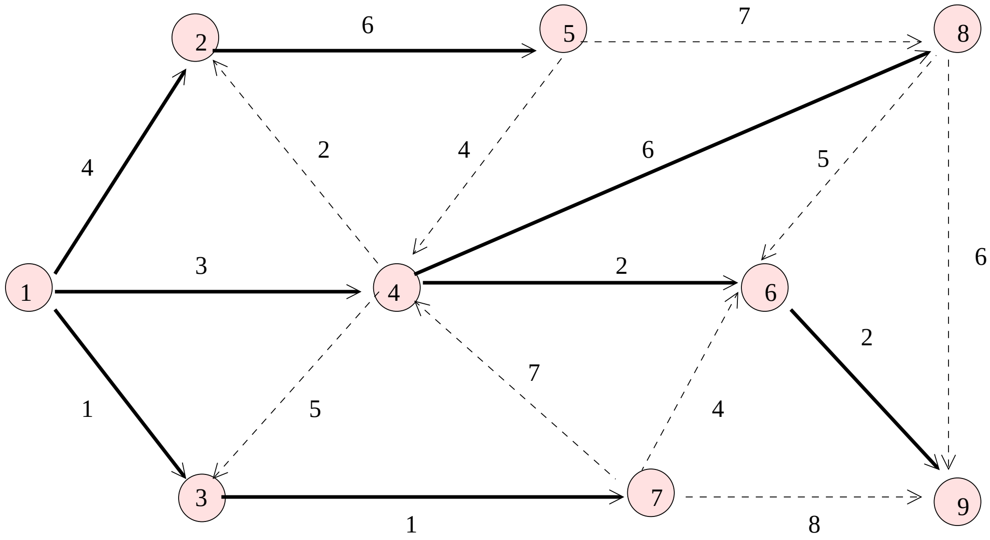


FIGURE 6.

(3) *Coin Changing.*

(a) Label the vertices $0, 1, 2, 3, \ldots$ and from each vertex $i$ we add $k$ directed out-arcs: $(i, i + c_1), (i, i + c_2), \ldots, (i, i + c_k)$. This gives the desired acyclic graph. It is easy to check that we can change $t$ cents exactly if and only if there is a directed path from the vertex 0 to the vertex $t$.

(b) Take our acyclic graph and give each arc a length 1. A shortest path on the from the vertex 0 to the vertex $t$ then corresponds to using the minimum possible number of coins. Thus, running a shortest path algorithm (or applying breadth first search) will find an optimal solution if a solution exists.

(4) *True or False*

(a) False: the example in Figure 7i) gives two different shortest path trees.

(b) False: take the graph shown in Figure 8) and decrease each arc length by 2 units. This gives a different shortest path from $s$ to $t$. The shortest path distance from $s$ to $t$ therefore fall by 3 units, which is not a multiple of 2.

(c) True: We prove this by contradiction. Suppose that the shortest path from $s$ to some vertex $v$ changes from $P$ to $P'$. Observe that both $P$ and

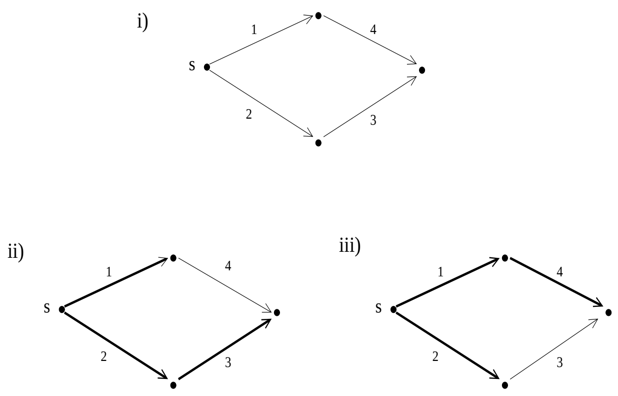


FIGURE 7.

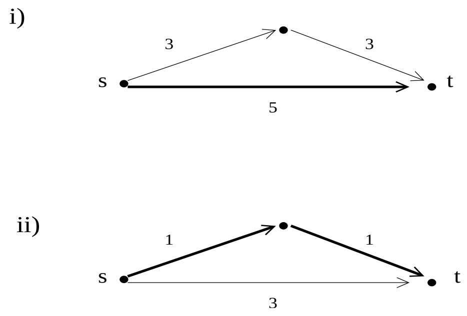


FIGURE 8.

$P'$ use exactly one arc emanating from $s$. This implies that the length of $P$ and $P'$ increase by exactly the same amount when we increase by $k$ units the length of each arc out of $s$. But the length of $P'$ was at least the length of $P$ before the increases and thus it remains so after the increases. Therefore we obtain our contradiction.

(d) False: take the graph shown in Figure 8i) and increase the length of each arc out of $v$ (in this case there is only one such arc) by 2 units. The shortest path tree changes.

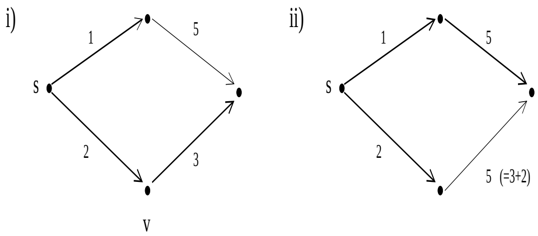


FIGURE 9.

(5) *Example for Dijkstra's Algorithm*

Here is one example showing that Dijkstra's algorithm does not always give the correct shortest path tree on a graph that contains negative weight arcs, even if the graph has no negative weight cycles. Let $G = (V, E)$, where $V = \{s, a, b\}$ and $E$ consists of the directed edges $(s, a)$ of weight 1, $(s, b)$ of weight 2, and $(b, a)$ of weight -100. Dijkstra's algorithm first moves $s$ into the shortest path tree, then $a$, then $b$. At the time $b$ is moved to the shortest path tree, it has no outgoing edges to vertices that are not already in the tree. In particular the edge $(b, a)$ is not relaxed, and the distance to $a$ is *not* updated from 1 to -98.

(6) *Floyd-Warshall and Negative Weight Cycles*

Suppose there is a negative weight cycle. Let $i$ be the smallest index of any vertex along this cycle, and let $k$ be the maximum index. By definition of negative weight cycles, we must have $d_{ik} + d_{ki} < 0$. At the $k$th iteration of the Floyd-Warshall algorithm, we will set $d_{ii}^{(k)} \leftarrow d_{ik}^{(k-1)} + d_{ki}^{(k-1)}$, which will cause $d_{ii}^{k}$ to be negative. The $ii$th entry in the table can only decrease during subsequent iterations, so after the final iteration we will have $d_{ii}^{(n)} < 0$. On the other hand, if there are no negative cycles then we must have $d_{jj}^{(n)} = 0$ for all $j$.

This gives us a simple test for negative weight cycles. If any of the elements along the diagonals of $D^{(n)}$ are negative, then the input graph contains a negative weight cycle. Otherwise, it does not.