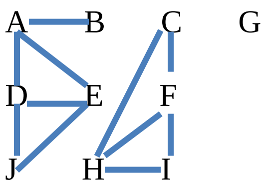
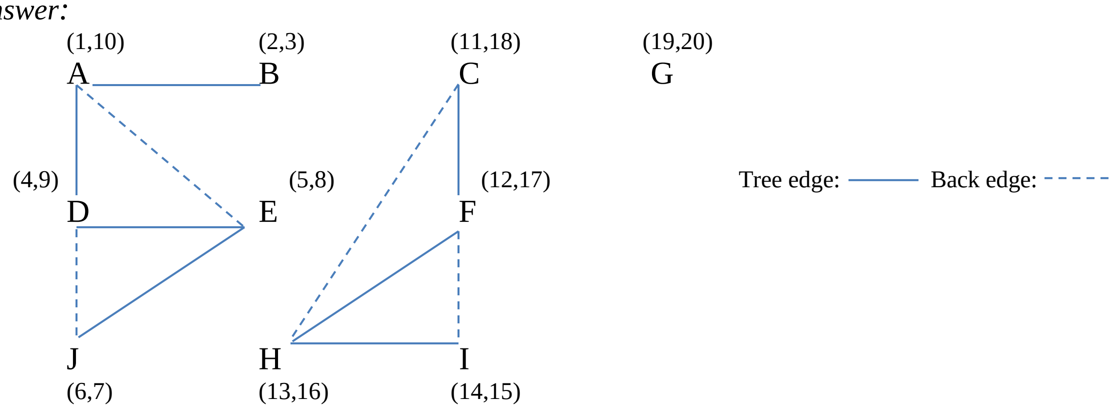
CS372 Spring 2019 Assignment #4 solutions.

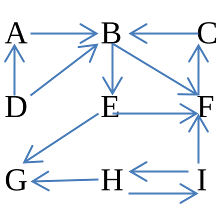
1. Perform a depth-first search on the following graph; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge or back edge, and give the `pre` and `post` number of each vertex.



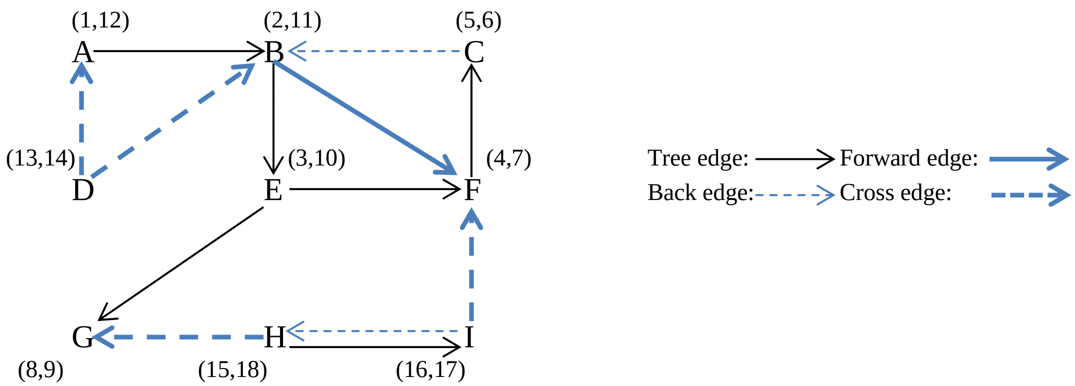
Answer:



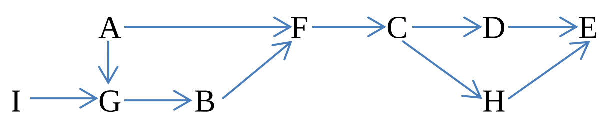
2. Perform depth-first search on the following graph; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, forward edge, back edge, or cross edge, and give the `pre` and `post` number of each vertex.



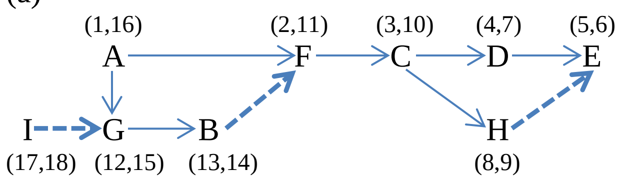
Answer:



3. Run the DFS-based topological ordering algorithm on the following graph. Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.
(a) Indicate the `pre` and `post` numbers of the nodes.
(b) What are the sources and sinks of the graph?
(c) What topological ordering is found by the algorithm?
(d) How many topological orderings does this graph have? List all of them.



Answer:
(a)



(b)
The `sources` of the graph: A, I.
The `sinks` of the graph: E.

(c)
The topological ordering found by the algorithm is: I, A, G, B, F, C, H, D, E.

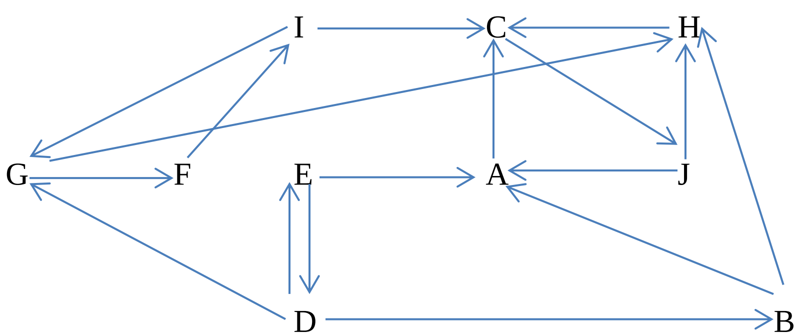
(d)
The graph has 4 topological orderings:
I, A, G, B, F, C, H, D, E
I, A, G, B, F, C, D, H, E
A, I, G, B, F, C, H, D, E
A, I, G, B, F, C, D, H, E

4. Directed graph G is represented by the adjacency matrix below. Draw the directed graph G and run the strongly connected components algorithm on it (use the algorithm from p.94 of the textbook). When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first. Answer the following questions.
(a) In what order are the strongly connected components (SCCs) found?
(b) Which are source SCCs and which are sink SCCs?
(c) Draw the "metagraph" (each meta-node is an SCC of G).
(d) What is the minimum number of edges you must add to this graph to make it strongly connected?

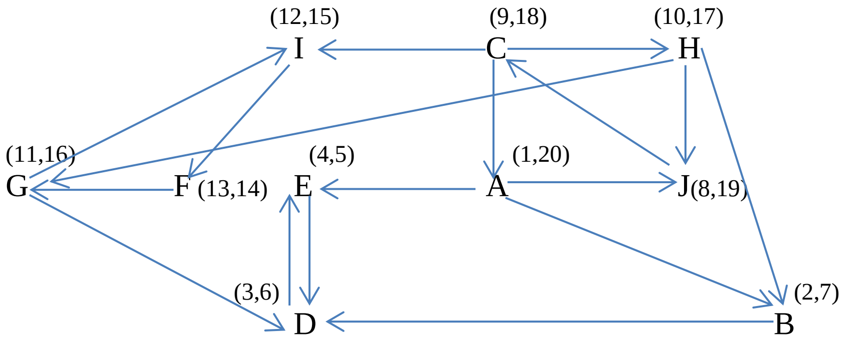
	A	B	C	D	E	F	G	H	I	J
A	0	0	1	0	0	0	0	0	0	0
B	1	0	0	0	0	0	0	1	0	0
C	0	0	0	0	0	0	0	0	0	1
D	0	1	0	0	1	0	1	0	0	0
E	1	0	0	1	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	1	0
G	0	0	0	0	0	1	0	1	0	0
H	0	0	1	0	0	0	0	0	0	0
I	0	0	1	0	0	0	1	0	0	0
J	1	0	0	0	0	0	0	1	0	0

(Recall that 1 in row i and column j means that there is an edge from vertex i to vertex j. For instance, there is an edge from A to C.)

Graph G:



Graph G^R:

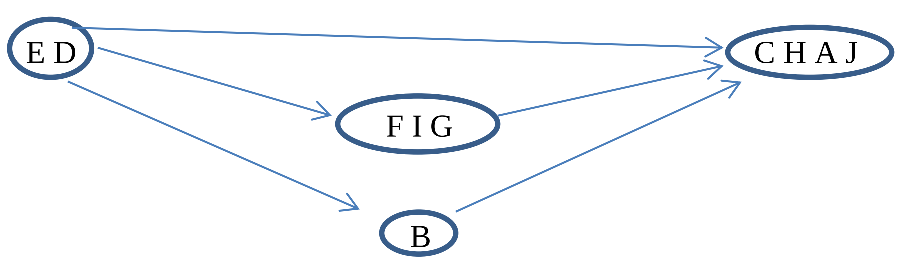


The decreasing `post` numbers in the depth-first search of G^R is: A, J, C, H, G, I, F, B, D, E.

(a) The strongly connected components (SCCs) are found in the following order:
{A, C, J, H}, {G, F, I}, {B}, {D, E}

(b) {E, D} is the source SCC; {C, H, A, J} is the sink SCC.

(c) The "metagraph" (each meta-node is an SCC of G):



(d)
The minimum number of edges that we must add to this graph to make it strongly connected is one edge. The edge should be from any vertex in the sink component to any vertex in the source component. Adding such edge makes the metagraph strongly connected and hence the given graph G also becomes strongly connected.
Therefore, adding edge, say, (A,E) makes G strongly connected.

5. Exercise 3.15. Note: "Formulate this problem graph-theoretically" means that you need to explain what the vertices in the graph are and what the edges are (when exactly two vertices are connected by an edge).

Answer:
(a) We view the intersections as vertices of a graph with the streets being directed edges, since they are one-way. Then the claim is equivalent to saying that this graph is strongly connected.

This is true iff the graph has only one strongly connected component, which can be checked in linear time.
(b) The claim says that starting from the town hall, one cannot get to any other SCC in the graph. This is equivalent to saying that the SCC containing the vertex corresponding to the town hall is a sink component. This can be easily done in linear time by first finding the components, and then running another DFS from the vertex corresponding to the town hall, to check if any edges go out of the component.
Note: In fact, it can even be done while decomposing the graph into SCCs by noting that the algorithm for decomposing progressively removes sinks from the graph at every stage. A component found by the algorithm is a sink if and only if there are no edges going out of the component into any component found before it.

6. Exercise 3.24. Note: You need to explain the algorithm and analyze its running time.

Answer:
Start by linearizing the DAG. Since the edges can only go in the increasing direction in the linearized order, and the required path must touch all the vertices, we simply check if the DAG has an edge (i, i+1) for every pair of consecutive vertices labelled i and i + 1 in the linearized order. Both, linearization and checking outgoing edges from every vertex, take linear time and hence the total running time is linear.