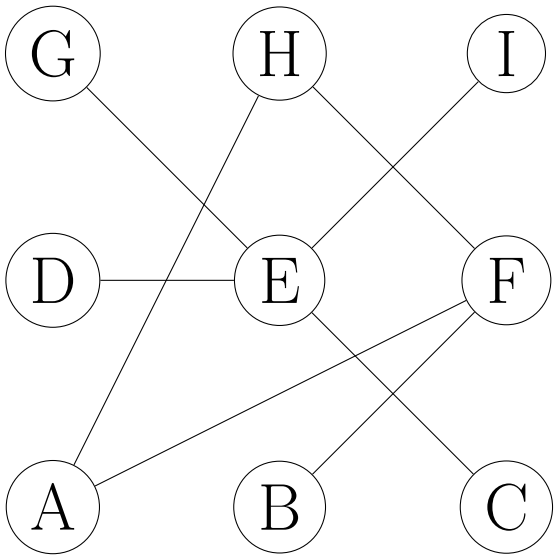
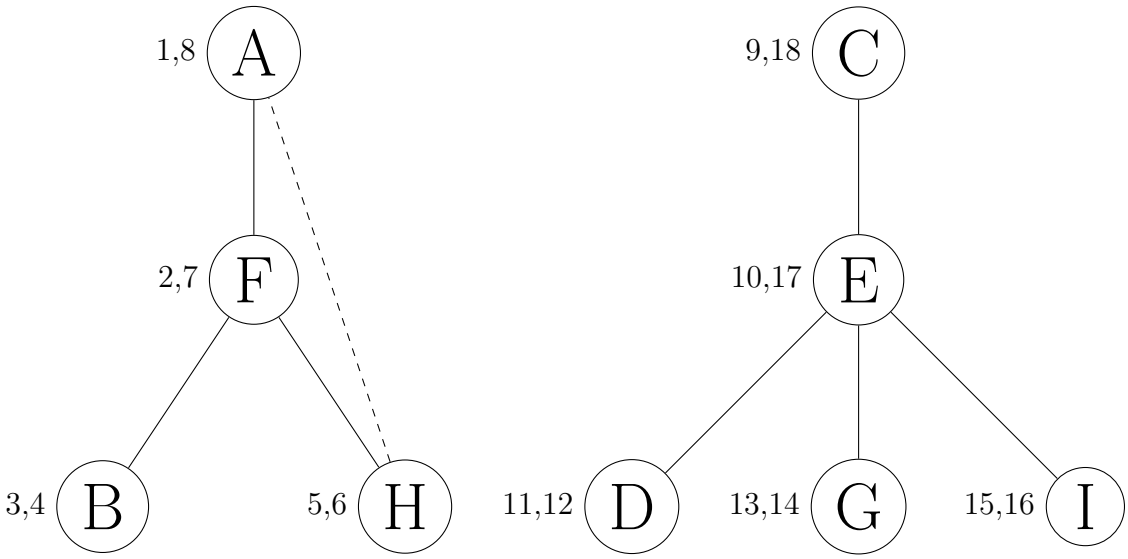
CS-UY 2413: Design&Analysis of Algorithms—Spring 2017 Prof. Boris Aronov
Department of Computer Science and Engineering Prof. Lisa Hellerstein

Homework 6 Solutions

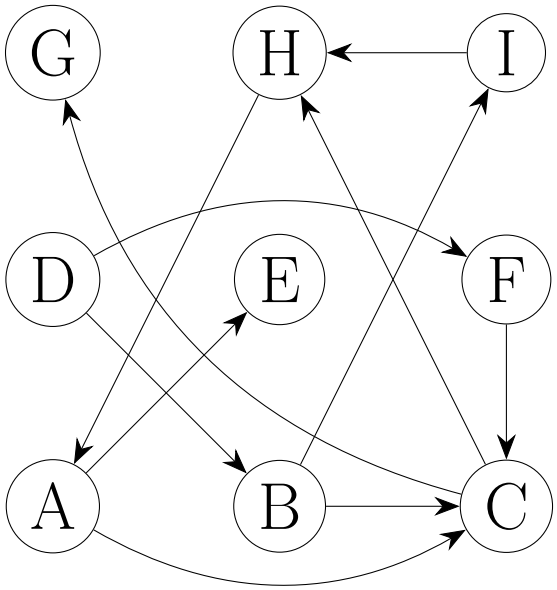
1. Run a DFS on the given *undirected* graph, starting with vertex A. Draw the DFS tree/forest and label every vertex with entry/exit times (pre/post times, in textbook terminology), as shown in class; when there is choice in visiting a vertex's neighbors, visit them in alphabetical order. If the first top-level call to DFS does not visit all the vertices, repeatedly invoke it on the alphabetically next unvisited vertex. Only show the final tree/forest, no need to show the stack contents, or other details of running the algorithm.



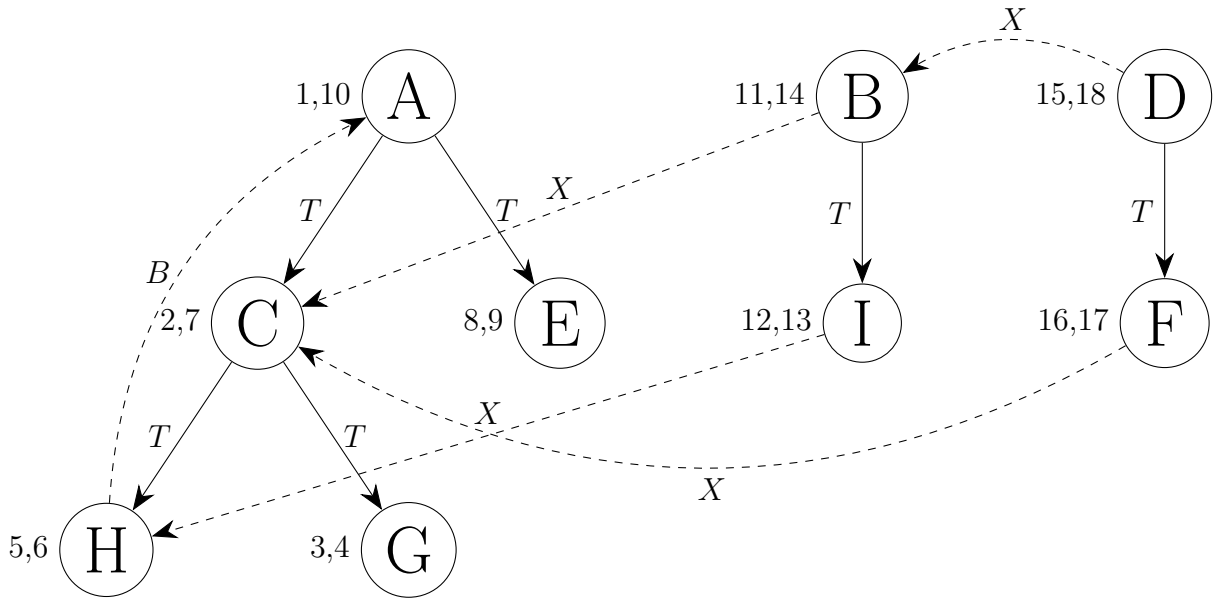
Solution:



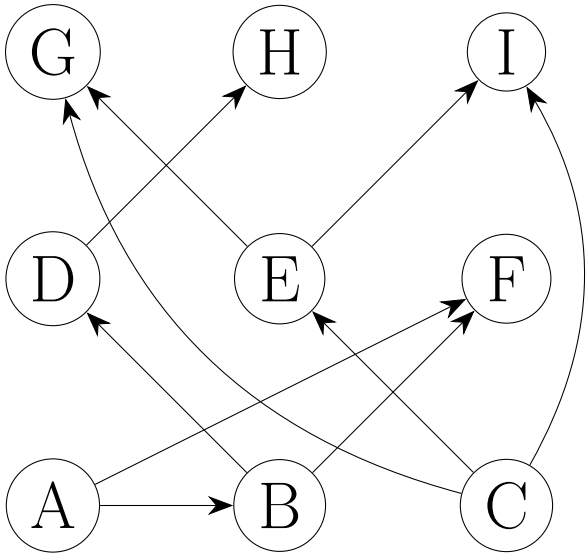
2. Repeat for the given *directed* graph. Draw separately the DFS tree/forest. Label each graph edge as a *tree (T)/forward (F)/backward (B)/cross (X)* edge.



Solution:

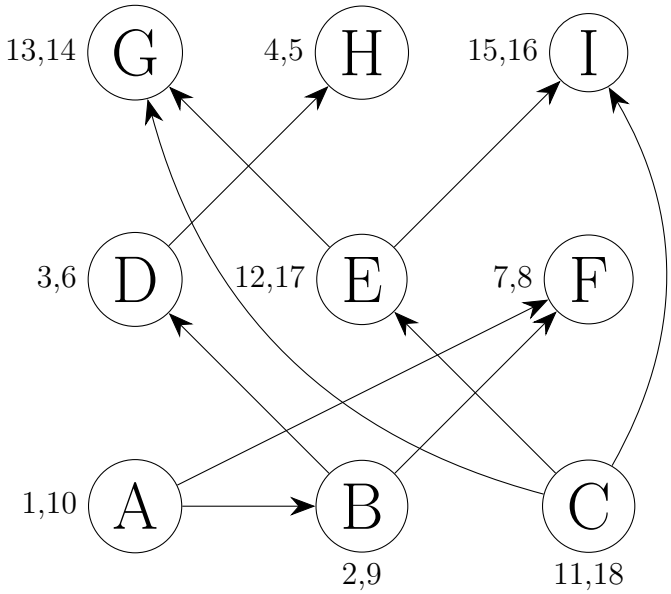


3. Consider the given directed graph. Demonstrate that it is acyclic by constructing a topological sort OR show that it is not by finding a cycle. Use a DFS-based algorithm.

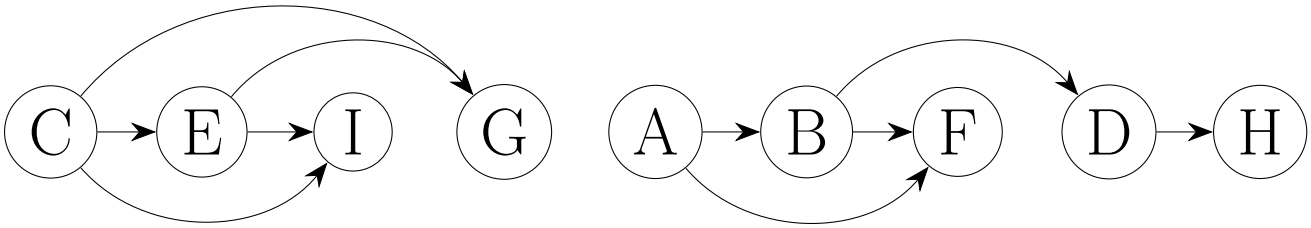


Solution:

If a directed graph can be topologically sorted, then it is acyclic. We will first use DFS to find the pre/post times for each vertex:

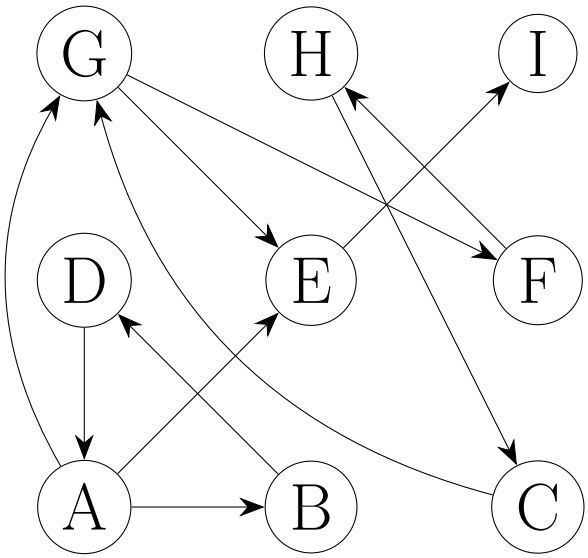


Now, we will list the vertices in descending order of post time.

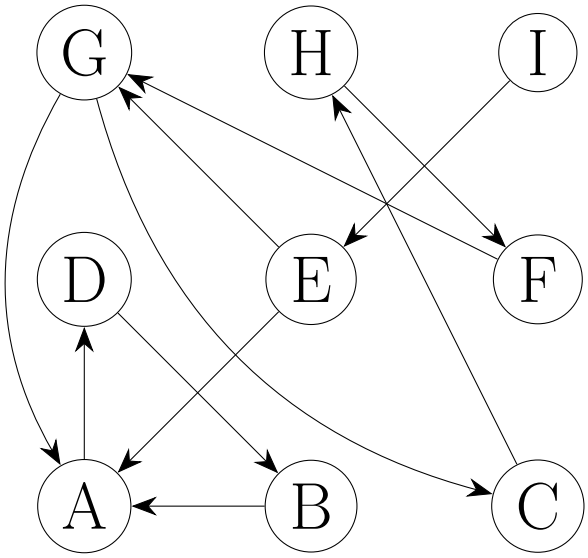


Since there are no back edges, this graph is acyclic.

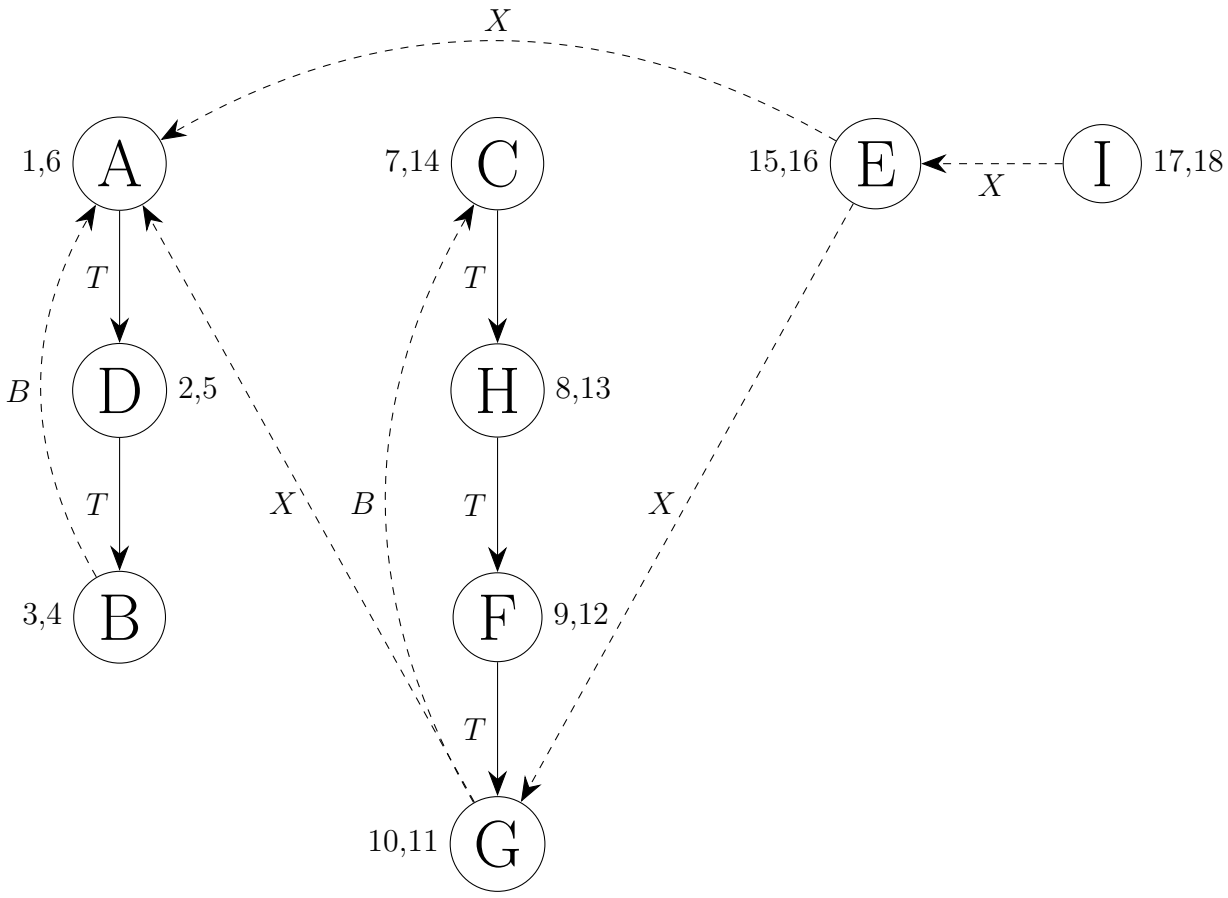
4. On the given directed graph, run the strongly connected components (SCC) algorithm described in the textbook. Show the first DFS forest, the ordering used by the final stage and the resulting strongly connected components.



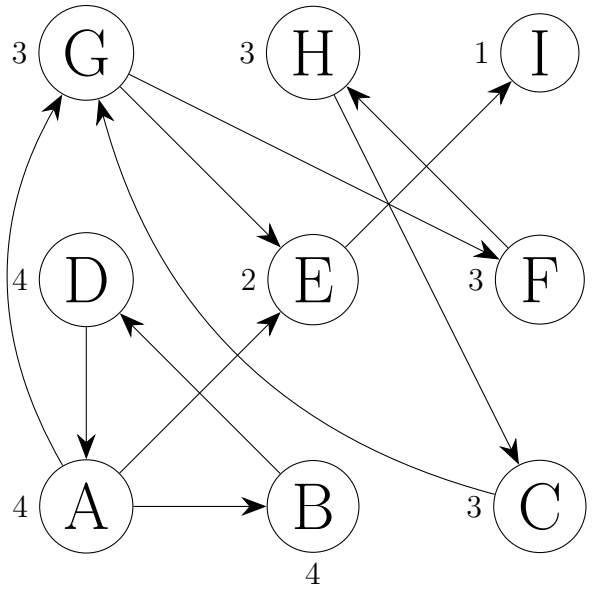
Solution:
Reversed Graph:



First DFS forest (DFS forest of the reversed graph):



The order used by the final stage (vertices in decreasing post number of the first DFS tree): I, E, C, H, F, G, A, D, B. We will run our connected component variant of DFS on our original graph, but this time, process the vertices in the order we obtained:



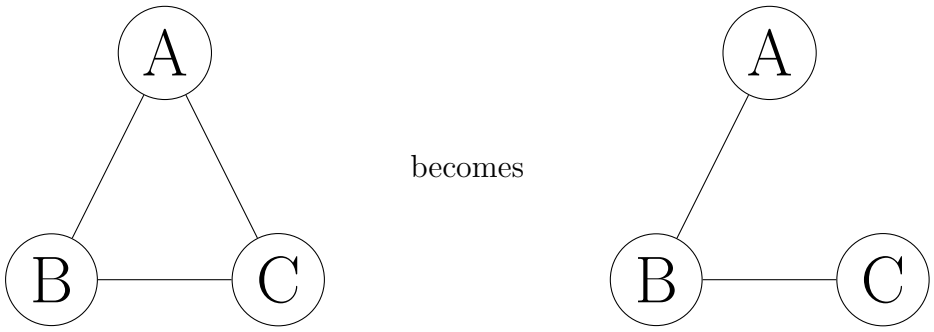
Thus, the strongly connected components are:

- A, B, D
- C, H, F, G
- E
- I

5. For each of the following situations, state whether it is possible. If not, explain why not. If yes, give a (small) example when it happens:

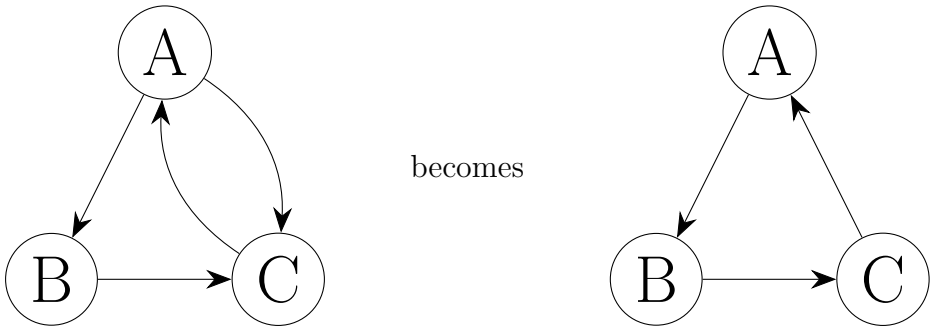
   (a) In an undirected graph, you remove an edge, and the number of connected components decreases.
   Solution: This is not possible. If you think of each connected component as a node, then you would have n separate nodes, where n is the number of connected components. The only way to decrease the number of connected components would be to add an edge between two nodes. If you were to take an edge away, the edge would have to be taken from within a connected component. This could either produce another connected component or have no effect on the total number of connected components. Thus, removing an edge can either have no effect on the total number of connected components or increase them.

   (b) In an undirected graph, you remove an edge, and the number of connected components stays the same.
   Solution: This is possible. Observe the following:



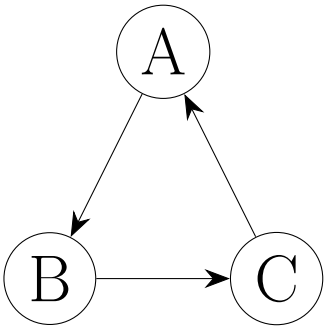
   The number of connected components remains as one.

   (c) In a directed graph, you remove an edge, and the number of strongly connected components stays the same.
   Solution: This is possible. Observe the following:



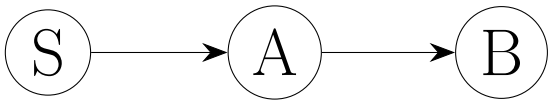
   The number of strongly connected components remains as one.

(d) In a directed graph, you remove an edge, and the number of strongly connected components increases by 2 or more.
Solution: This is possible. Observe the following:

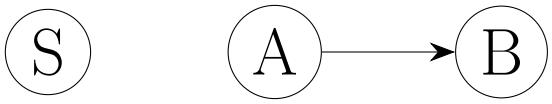


If we remove any edge from the graph, we transition from one strongly connected components to three strongly connected components.

(e) In a directed graph, with every vertex reachable from the vertex s, you remove an edge, and the number of vertices reachable from s (other than s itself) goes down to 0
Solution: This is possible. Observe the following:



Removing the edge from s to a gives us the following:



Thus, the number number of vertices reachable from s decreases from 2 to 0.

6. You are working with investigators on a legal case involving text messages. There is a collection of n text messages, numbered 1 through n, all sent at different times. Noone knows when the messages were written. However, for some pairs of messages, the investigators have evidence that one was written before the other. Based on this evidence, they have created a file with information about these pairs. Each line of the file has the form (message $i <$ message j), which means that message number i was written before message number j.

   The investigators want you to use the data in the file to determine the answer to the following question, which we will call Question A.

   Question A: Could all the information in the file be correct? That is, is there a potential ordering of the messages, from earliest to latest, that is consistent with all the information given in the file?

This question can be answered by using the information in the file to build a graph, and then running an appropriate graph algorithm. The following problems ask you to explain, at a high level, how this could be done. (Your answers to all of the following questions should be short, and should NOT include information about low-level programming issues such as reading in the data file.)

(a) What are the vertices of the graph and what are the edges of the graph? Explain the relationship between the information in the file and the vertices and edges of the graph. Also specify whether the graph is directed or undirected.

Solution: The vertices of the graph are the messages in the file and the edges indicate which message was written before the other. This will be a directed graph, where the arrow starts from messages that were sent earlier and points towards messages sent later.

(b) Which graph algorithm should you run to answer Question A? Just give the name, or a very short description, of the graph algorithm. You do NOT have to give pseudocode.

Then describe how would you use the output of the graph algorithm to answer Question A.

Solution: The question is essentially asking if the graph is linearizable. Therefore, running DFS on the graph would be sufficient. To determine if there is an ordering of the messages and is consistent with all the information, we only have to check if there are no back edges in the resulting DFS forest. In the event that there is a back edge, a cycle exists within the graph which leads us to believe there is incorrect information within the file. This process checks whether or not a topological sort can be constructed. A topological sort can only be constructed if and only if the graph is a DAG. By showing that a topological ordering of the vertices is possible, we also show that there are no cycles in the graph.

(c) Suppose all the information in the file is correct. Consider the following new question.

Question B: Given two messages, i and j, could message i have been written before message j? That is, is there a potential ordering of the messages, from earliest to latest, consistent with all the information in the file, where message i is listed somewhere before message j?

Describe how to use a standard graph algorithm to answer Question B. You should NOT give pseudocode for the standard graph algorithm or describe how it works, but you do need to be clear about how you are *using* that algorithm to get the answer to Question B.

Solution:

Note: There is more than one correct answer to this question. Here is one of them.

Algorithm:
Run Explore from vertex j to check whether vertex i is reachable from j. If i is reachable from j, output "message i could not have been written before message j". Otherwise, output "message i could have been written before message j".

We now prove that this algorithm works correctly.

Exactly one of the three following cases must hold for the graph:

- There is a path from i to j
- There is a path from j to i
- There is no path from i to j or from j to i

We show that in each of these cases, the algorithm outputs the correct answer.

In Case 1, there is a directed path from i to j. Since the graph has no directed cycles, there can't also be a directed path from j to i, so i is not reachable from j. Therefore, Explore will output the answer "message i could have been written before message j". To see that this is the correct answer for this case, consider the path from i to j. Since each message in this path must have been written before the next message in the path, any valid ordering of the messages places i sometime before j.

In Case 2, Explore will detect that i is reachable from j and output the answer "message i could not have been written before message j". We show that this answer is correct for this case. Consider the directed path from j to i. Since each message in this path must have been written before the next message in the path, j must have been written before i, and therefore i could not have been written before j.

Finally, we consider Case 3. In this case, the algorithm will output "message i could have been written before message j". To see that this is the correct answer for this case, suppose we add edge (i,j) to the graph. Call the new graph G'. The original graph had no directed cycle. If G' had a directed cycle, the cycle would have to contain the new edge (i,j). But this is impossible, because such a cycle would have to include a path in the original graph from j to i, and we are in Case 3, so no such path exists. Therefore, G' does not have a directed cycle and hence G' can be linearized. The linearization must place i before j, because G' has edge (i,j). This linearization is also valid for the original graph, and so message i could have been written before message j.