Exercise 6: SCCs, DAGs, and topological sorting

[updated Feb 3]

Questions

1. Consider a graph G and the corresponding reversed graph G^T (defined by reversing the directions of each edge of G) Prove that the strongly connected components of these two graphs are identical.

2. [solution updated Jan 31] Suppose you have a directed graph G. Define a forest of trees by doing a depth first search DFS(G), as defined in class. Prove that each strongly connected component of G lies entirely in one of the trees of this forest.

3. Show that every DAG must have at least one vertex with no outgoing edges.

4. What is the maximum number of edges m that can exist in a DAG with n vertices ?

5. The definition of a topological ordering of a graph G=(V,E) is an ordering of the vertices from 1 to n such that if (i,j) is in E then i < j. Give an example of a graph with a topological ordering such that the converse statement "if i < j then edge (i,j) is in E" is false.

6. Describe a data structure that you could use to efficiently implement the topological ordering algorithm discussed in class, and what role this data structure plays. (You may assume you have a graph data structure – don't provide details on that.) With this data structure, what is the running time of the algorithm?

7. In class I sketched out a proof that a DAG has a topological ordering. Write out a formal proof by induction.

8. Under what condition can there be more than one topological ordering for a given DAG?

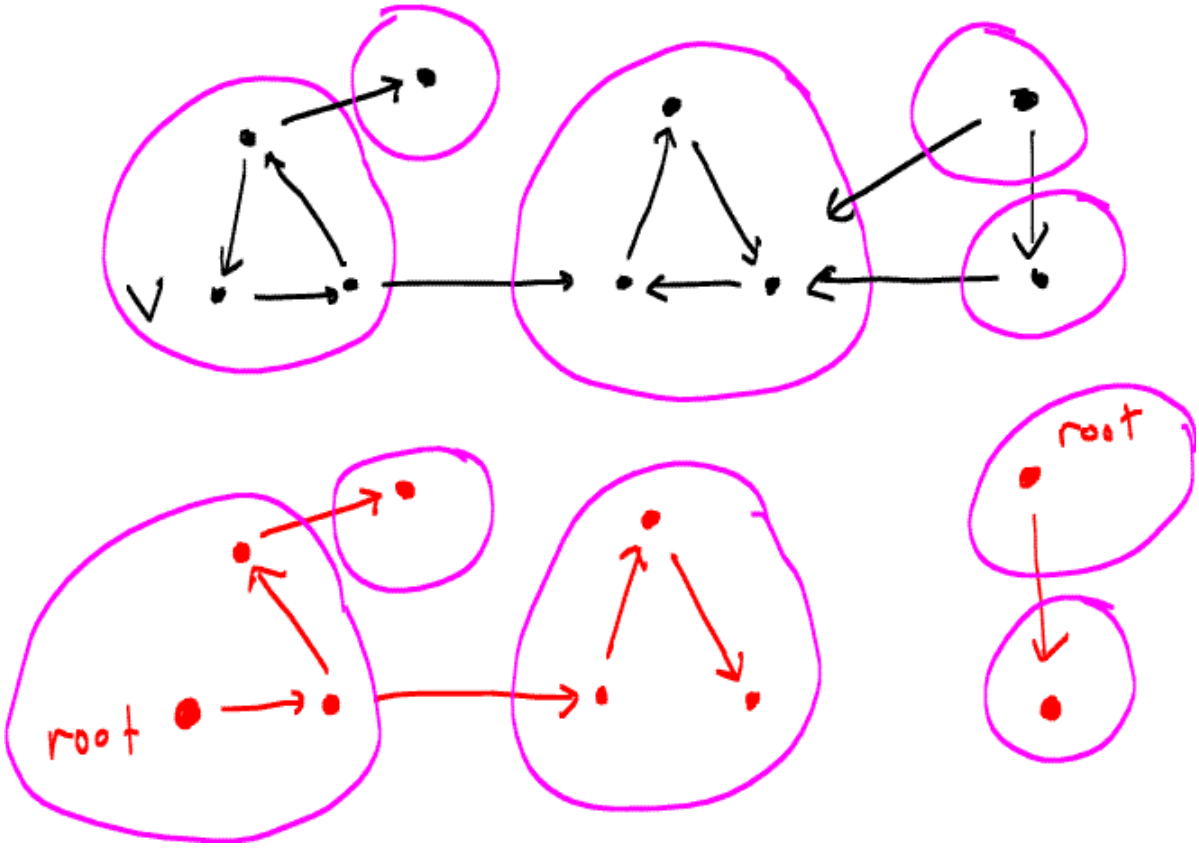
9. Consider a directed graph G. Suppose we define a "collapsed graph" C as follows: the vertices of C are the SCCs of G and there is an edge between two vertices (v_SCC1 and v_SCC2) if there is an edge in the original graph from one of the vertices in SCC1 to one of the vertices in SCC2. (We allow at most one edge between vertices in the collapsed graph.) Can the collapsed graph have a cycle?

10. Let G be a DAG containing edge (v,w). Suppose that during an execution of DFS(G) there is a call DFS(G,v). Is it possible that DFS(G,w) has already been called ? If so, can we say whether DFS(G,w) has already returned i.e. exited?

Answers

1. Two vertices v and w belong to the same SCC of G if there is a path from v to w and there is a path from w to v. If we reversing all the edges of the graph to get G^T, then we would still have a path from v to w and a path from w to v, namely the same paths as in G.

2. To understand the question, consider an example. The graph is shown in black. The DFS trees are shown in red. The strongly connected components are shown in purple. The claim is that the vertices in each SCC are contained in one of the DFS trees. You can see that from this example. But why is it true in general?
   Consider any strongly connected component. Let v be the first vertex in this SCC that was reached by the DFS(G) algorithm. When DFS(G) reaches v, it calls DFS(G,v) and that will cause to all vertices that are reachable from v (and that haven't yet been visited) to be in the subtree that is rooted at v. But the set of vertices reachable from v includes all vertices in the SCC containing v.



3. If every vertex has at least one outgoing edge, then starting at an arbitrary vertex and following any sequence of outgoing edges would eventually cause you to revisit a vertex. But this means that the graph has a cycle (which contradicts the given fact that the graph is acyclic).

Another way to prove it is to take the result proven in class, namely that every DAG has at least one vertex with no incoming edges, and apply that result to the reversed graph. (If G is a DAG, then the reversed graph G^T is also a DAG. That should be obvious.) Since the G^T has at least one vertex with in-degree 0, it must be that G has at least one vertex with out-degree 0.

4. Given a DAG, there must exist at least one topological ordering, so choose one topological ordering and "fix it" (i.e. don't change it). Vertex v_1 in this topological ordering has no incoming edges (its in-degree is 0). It may have up to n-1 outgoing edges. Take v_2 in the topological ordering. It can have at most n-2 outgoing edges, that is, to vertices v_3, … v_n. Following this pattern, v_k can have at most n-k outgoing edges. In particular, v_n can have no outgoing edges. Thus the total number of edges is at most (n-1) + (n-2) + (n-3) + …. + 1 + 0 = n(n-1)/2. Note that this is much less than n^2 which is the maximum number of edges in a directed graph with n vertices.

5. Take a graph with three vertices and two edges (1,2) and (2,3). The converse statement given in the question is that "if i < j then (i,j) is an edge", but that's not true since (1,3) is not an edge.

6. At each iteration of the algorithm we need to find a vertex v in G' such that v has no incoming edges. For this, we maintain an array of "incoming-edge counts": for each vertex, keep track of the number of incoming edges to each vertex. We can initialize this array as follows: start by setting the incoming-edge-count of each vertex to 0; then iterate through all the edges: for each edge (u,v), increment the count of incoming edges to v. After we have initialized the incoming-edge-count array, take one pass through the array, and make a list of vertices that have no incoming edges, i.e. an incoming edge count of 0.

   Now recall how the algorithm works. At each iteration, you remove a v from the list of vertices that have no incoming edges. You also need to remove edges of the form (v, w), i.e. edges in the adjacency list of vertex v. For each edge (v,w) in v's adjacency list, reduce the incoming edge count of w. When the incoming edge count of any w reaches 0, add w to the list of vertices that have no incoming edges.

   The algorithm of computing a topological sort is O(n + m). Why? You need to iterate over all vertices so it takes at least O(n). You also need to check all edges in the graph. But you only need to check each edge once, and the work for each edge is (bounded above by a) constant.

7. The base case is n=1 which is trivial. The induction hypothesis is that for any DAG with k vertices, there is a topological ordering. The induction step is to show that the claim therefore must hold for a graph with k+1 vertices. So consider a DAG with n=k+1 vertices. As argued in class, at least one of these k+1 vertices that has no incoming edges. So take one such vertex (there may be many) and call it v1. Define G' = (V', E') where V' = V - {v_1} and E' = E with the edges of the form (v_1, w) removed. But G' is obviously a DAG too, and it has k vertices. So

by the induction hypothesis, it has a topological ordering, namely (v_2, … v_n). This gives a topological ordering for the original graph G, namely (v_1, v_2, … v_n).

8. Another way of stating the question is to ask, under what conditions is the topological ordering of a DAG unique? To answer that question, think of how the algorithm for building a topological ordering works. At iteration i, you need to choose a vertex that has no incoming edges. There is a unique topological ordering only if there is only one vertex available at every iteration i.

   When I first posted the solutions, I wrote "This can happen in only one way, namely that the entire graph is one singly linked list." However, that proved to be false. The example in my lecture 6 page 5 (ABCD) is a counter example.

   At this moment, I believe that the correct answer is that there exists only one vertex with no incoming edges (v1), and only one vertex with no outgoing edges (vn), and there exists only one path in the graph from v1 to vn. But I am not 100% sure. Anyone who wants to fill in the proof (or find a counterexample), please do.

9. No, the collapsed graph cannot have a cycle. (That is, the collapsed graph is a DAG.) Why? Suppose the cycle contained two (distinct) vertices corresponding to (distinct) SCC1 and SCC2. That would mean there is a path from every vertex in SCC1 to every vertex in SCC2 (since the vertices in SCC1 are all reachable from each other, and the same for SCC2) and it would mean there is a path from every vertex in SCC2 to every vertex in SCC1. But then the vertices in SCC1 and SCC2 must be in the same strongly connected component in the original graph, which would mean that SCC1 and SCC2 are the same (not distinct).

10. Yes it possible that DFS(G,w) has already been called e.g. the DFS(G) could have started with DFS(G,w). However, if DFS(G,w) has already been called, then it must have already returned. Otherwise, there would be a path from w to v on the tree, namely the path defined by the (allegedly) currently running DFS(G,w). But if there is a path from w to v then the graph cannot be a DAG (since there is an edge (v,w)).