COMPSCI 230: Discrete Mathematics for Computer Science 4/9/2020

Recitation 12: Graph Theory (SCC, induction)

Created By: David Fischer; adapted from Erin Taylor, Kevin Sun, Alex Steiger

1. Dangers of "build-up" induction. I've had a lot of students in OH talk about doing induction by assuming the inductive hypothesis for arbitrary k, and building off of what is given from that to show the property for $k+1$. Think critically about the following.

   - Consider an inductive proof for the following claim: if every node in a graph has degree at least one, then the graph is connected. By induction on the number of vertices. For the base case, consider a graph with a single vertex. The antecedent is false, so the claim holds for the base case. Assume the claim holds for an arbitrary k node graph. Then, add a vertex v to the graph, to obtain an arbitrary G' with $k+1$ vertices. To satisfy the antecedent, add at least one edge from v to node(s) in G' (that were also in G). By the inductive hypothesis G is connected, and we added a vertex and connected it, G' must also be connected, and so by induction the claim is proved.
   - *Discussion:* Let students think for about 5 minutes. Does this proof hold? If not, why not? It is not a correct proof; counterexamples are abundant (eg two connected components and two vertices with an edge between them in each). In fact it is incorrect to say "By the inductive hypothesis G is connected...", since it assumes that all possible graphs can be constructed by taking an arbitrary connected graph and adding a vertex with an edge to it. In some cases the range of the operation you perform will in fact be all possible examples of the object you want to prove the property for, but as was just demonstrated, this is not guaranteed. This is why it is far better to start with an arbitrary object of the type you want to prove the property for, then remove a vertex or otherwise alter it to get an object you can apply the inductive hypothesis to, and then build it back up to the original object. As an exercise, what happens if you try that on the claim given above?

2. To give a bit of a hint on the structure of a homework proof, we will prove a familiar result in a novel manner: Prove that the number of edges in a connected graph is greater than or equal to $n-1$.

   - For one vertex, 0=0, so the claim holds. Assume the property is true for all k vertex graphs. Consider an arbitrary $k+1$ vertex graph and m edges. We want to show that $m \geq k$ Remove some vertex v with degree $d(v)$, and all incident edges to create G'. Note that G' has $m - d(v)$ edges. As well, note that we *cannot apply the inductive hypothesis to* G', since G' is not necessarily connected. So, suppose there are p connected components, and label each connected component C_i, $i \in [1, p]$. Note that $p \leq d(v)$. Add an edge between an arbitrary vertex in each C_i to an arbitrary vertex in C_{i+1}. We have added $p - 1 \leq d(v) - 1$ edges, and the result is a graph G'' with a single connected component. Another way to think of this if the fewest number of edges to make G' connected are added, less than $p - 1 \leq d(v) - 1$ edges will be added since any cycle has the same number of edges as vertices, so if we added p edges we would have a cycle (and so could remove an edge without disconnecting the graph). We denote the number of edges in G'' as m'', so $m'' \leq m' + p - 1 \leq m - d(v) + p - 1 \leq m - d(v) + d(v) - 1 \leq m - 1$. We can also now apply the inductive

hypothesis to G'': G'' is connected and has k vertices (since we've only removed the one vertex from G), and so must have at least $k-1$ edges, which means $m'' \geq k-1 \rightarrow m-1 \geq m'' \geq k-1 \rightarrow m \geq k$ as desired. So the theorem is proved by induction on the number of vertices.

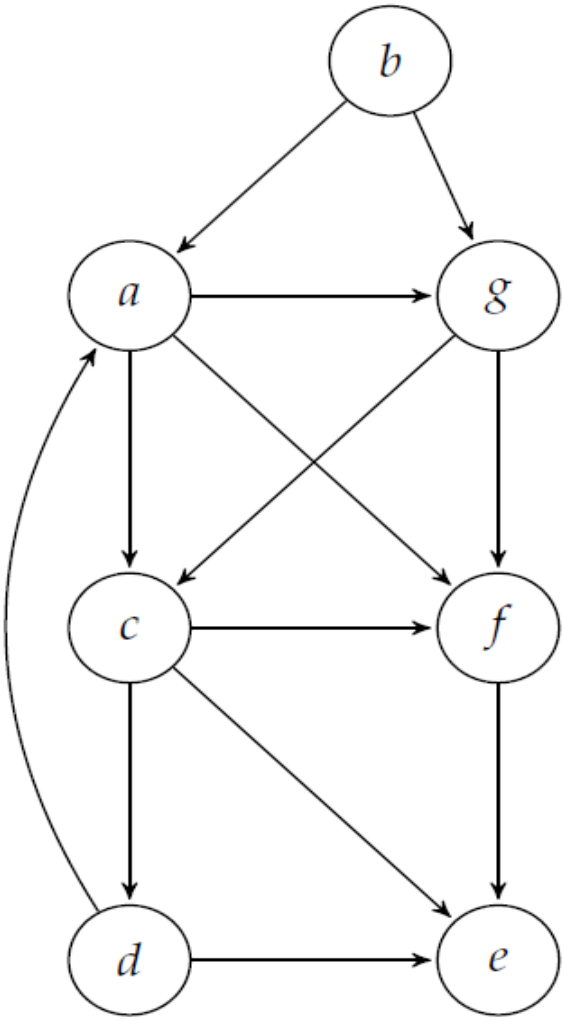
3. Give an example of a directed graph with one strongly connected component that is not a simply circuit.

   - Really anything can do here. One way to construct a nice complicated one is to draw some complicated flow graph with nonzero flow on every edge and then connect the sink to the source.

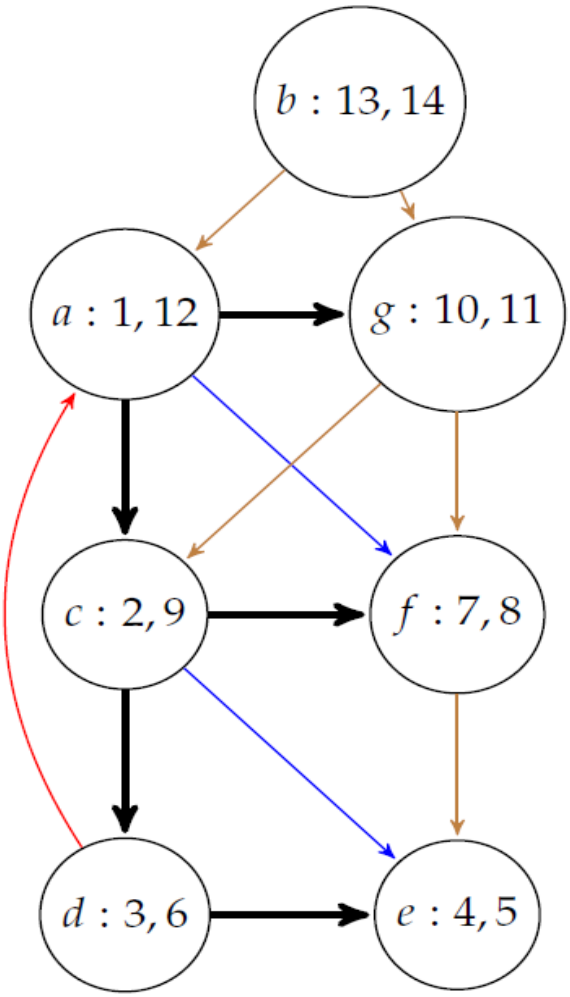
4. Give an example of a directed graph with on the order of n^2 edges, but each vertex is its own strongly connected component.

   - Again, not hard to construct one. One method is to draw some number of vertices in a circle, and start at some arbitrary vertex and in clockwise order draw directed edges only to vertices farther in the clockwise direction than the current vertex, but which are before the initial vertex.

5. Consider the following directed graph. Perform DFS starting from vertex a such that when choosing between potential vertices to explore, the one that is earlier in the alphabet is explored first. What are the tree, forward, backward, and cross edges with respect to the resulting DFS tree or forest?



- Tree edges are bolded, back edges are red and cross edges are brown. The notation $v : a, b$ denotes that $pre(v) = a$ and $post(v) = b$ for vertex v



6. In the above example, classify the different edge types according to their pre and post values. For example, which, if any, edge types have the property that $post(u) < pre(v) < post(v) < pre(u)$?

Type of edge (u, v)	Pre/post relationship of u, v
Tree	
Forward	
Back	
Cross	

(a) Tree edges (u, v) have that v was visited from u; ie v is a child of u

(b) Back edges have that v was visited before u; ie u was a non child descendent of v

(c) Forward edges have that u was visited before v; ie u is a non parent ancestor of v

(d) Cross edges have neither u was visited before v, nor v was visited before u, so u is neither an ancestor nor a descendent of v

We now go through the logic for filling in the table. Recall that for all vertices v, $pre(v) < post(v)$.

(a) Tree edges and forward edges must have that v is a child of u. So we can say $pre(u) < pre(v)$. As well, DFS must "return to u" through v, so it must be that $post(v) < post(u)$. Together with $pre(v) < post(v)$ and the same for u we have $pre(u) < pre(v) < post(v) < post(u)$.

(b) Back edges have that v was visited before u, so $pre(v) < pre(u)$. Further, DFS must "return to v" through u, so $post(u) < post(v)$. Together this gives $pre(v) < pre(u) < post(u) < post(v)$

(c) Cross edges have that neither v was visited before u nor was u visited before v, with similar logic for the order at which each node will have been fully explored. Since (u, v) is an edge in the graph, however, we can say that v was visited before u, and since DFS fully explored v before visiting u, we must have that $pre(v) < post(v) < pre(u) < post(u)$.

Type of edge (u, v)	Pre/post relationship of u, v
Tree	$pre(u) < pre(v) < post(v) < post(u)$
Forward	$pre(u) < pre(v) < post(v) < post(u)$
Back	$pre(v) < pre(u) < post(u) < post(v)$
Cross	$pre(v) < post(v) < pre(u) < post(u)$