CS 3510 Algorithms Last updated: September 25, 2023

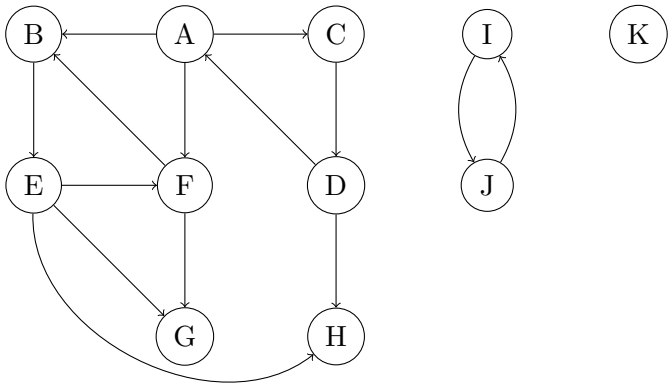
Lecture 7: Topological Sorting

Lecturer: Sahil Singla *Scribe(s): Joseph Gulian*

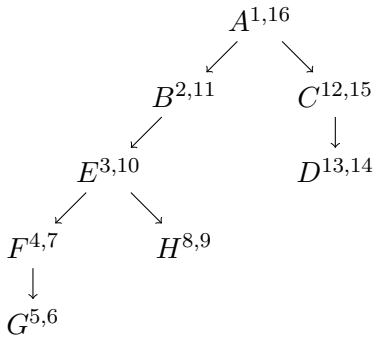
As we discussed last time a graph G is represented by the tuple (V, E) where V is the set of vertices and E is the set of edges. With directed graphs, each edge (u, v) implies there is an edge from vertex u to vertex v, but it does not imply that there is a vertex from v to u.

1 Topological Sorting

Now imagine you have a set of tasks that depend on other tasks, and you'd like to sort the graph by precedence. Formally this means that you will find integers for which $f(v)$ for $v \in V$ s.t. $(u, v) \in E \Rightarrow f(u) < f(v)$. This is called topological sorting. Now, a topological sorting may not always exist, for you might have a cycle in which case the equation above breaks down because you can not sort the cycle.



Say you want to find a directed cycle in the graph we discussed last time. We'll run DFS like we discussed last time, and you'll notice that the edges in the tree correspond to edges in the graph. However there are certain edges missing. For instance we don't have a graph from E to G or the edge from F to B; these are called forward edges and back edges respectively. Cross edges go across the DFS tree, for instance the edge from D to G would be a cross edge. The important part of this is that back-edges are cycles. This is because if an ancestor of a node is able to be reached by that node, that means there is a path from that ancestor to the node (because it is an ancestor) and a path from the node to the ancestor (because the back-edge at end is that path). So to check for cycles all you have to do is run DFS and check for back-edges.



As you've probably seen, we have numbers above the each vertex in the DFS tree. Imagine for each pre and post in the explore algorithm we incremented a clock, and we saved that to the pre and post variables for that vertex. So, we reach pre for A, set it's pre label to 1 and then increment. We go down to G (and keep incriminating the clock as we go down) where we have nowhere to traverse to, so we go back up but we go through post for G and F, so we increment the clock from 6 and from 7.

Now when you have an edge to another vertex, you check to see if it's start and end time are a subset of the ancestor which you can easily identify using the pre and post labels. So for instance if you were looking at the edge from D to H you would see that although the pre label is smaller, the post label is also smaller, so it is not a cycle.

However by doing this method, you also have some valid topological ordering by considering $f(v) = -post[v]$, i.e., the order given by decreasing post values. This comes from an observation on back edges and will be left as an exercise.

2 Connectivity

If two vertices u, v are strongly connected, there is a path from u to v and from v to u. A component of a graph is strongly connected if every node in that component is strongly connected to every other node in the graph. The challenge here is how to find strongly connected components given a graph. There's this nice property that if you replace each strongly connected component with one vertex, you create a DAG (directed acyclic graph) [1].

We can check if the entire graph is one strongly connected component by running DFS from some node. If some node is not reachable from your starting node, it is not a strongly connected graph. Then you must check that there is a path from each node to the root. This can also be done by depth first search by reversing the edges of the graph and run from the starting node.

[1] Know this acronym