CIS 575. Introduction to Algorithm Analysis
Material for April 19, 2024

Depth-First Search on Directed Graphs: An Application

The topic of this note is covered in *Cormen*'s Section 20.4.

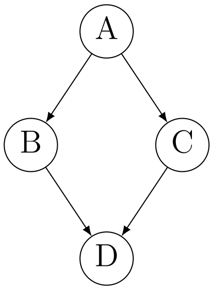
1 Topological Sort

For a **directed** graph (V, E), *topological* sort is the problem of listing the nodes in a order that "respects" the directed edges, in the sense that

> whenever there is an edge from u to w then u must be listed before w.

If the nodes represent tasks, and an edge from u to w represents that u is a prerequisite for doing w, a topological sort will show one possible linear schedule for the tasks.

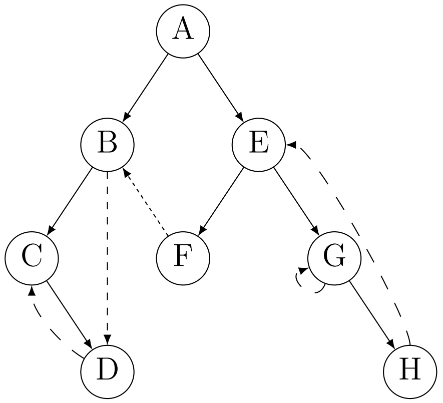
For example, consider the graph



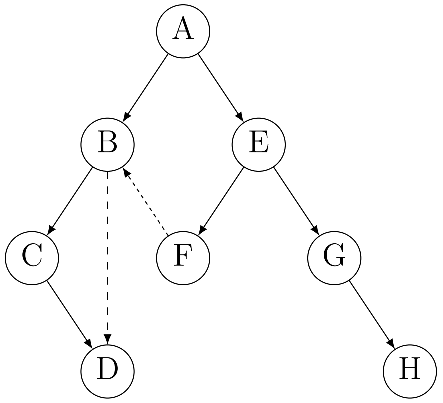
which allows *two* topological sorts: ABCD, and ACBD.

Obviously, if the graph has a cycle then no topological sort is possible. But we shall now show that for an **acyclic** graph it is *always* possible to construct a topological sort.

To do so, let us recall that the DFS algorithm from a directed graph constructs a tree, together with back edges, forward edges, and cross edges. We looked at the example



but observe that a back edge, going from a node to a tree ancestor, will *always* cause a cycle. Let us therefore remove the back edges, to arrive at



We shall use this example to derive a general recipe for how to find an order that respects each edge. With such an edge going from u to w, there are 3 cases to consider:

- For a *tree edge* (such as from E to F), where w is discovered only when u examines it, it will still be the case that (due to nodes being processed using a stack discipline) that w finishes before u does.
- For a *forward edge* (such as from B to D), even though w was discovered after u was discovered, w has finished before u gets to examine w.
- For a *cross edge* (such as from F to B), w finished even before u was discovered.

We see that in *all cases*, the node w was finished *before* u was finished. Hence, we can get a topological sort by listing the nodes in *decreasing* order of their finish time. Thus, if we print a node whenever it finishes, the output will be a topological sort in *reverse* order (a later processing can reverse it back).

Looking back at the generic DFS algorithm, this shows that the **PostNode** action should be "print the node". The other non-trivial action is **OtherEdge** which must report an error for a back edge, but ignore a forward edge or a cross edge. Thus the following instantiation of the DFS algorithm (which inherits its running time $\Theta(|V| + |E|)$) implements (reverse) topological sort:

```
TopologicalSort($u$)
        $color[u] \leftarrow$ gray
        foreach $(u, w) \in E$
            if $color[w] =$ white
                TopologicalSort($w$)
            else
                if $color[w] =$ gray
                        // back edge
                    Error: cyclic graph
        $color[u] \leftarrow$ black
        print $u$
```

For our example, the algorithm will output the list DCBFHGEA and the reverse list

AEGHFBCD

is indeed a topological sort (several others exist).