CS41 Lab 5

October 5 2020

In typical labs this semester, you'll be working on a number of problems in groups of 3-4 students. You will not be handing in solutions; the primary purpose of these labs is to have a low-pressure space to discuss algorithm design. However, it will be common to have some overlap between lab exercises and homework sets.

The main focus of this lab is on directed graphs without cycles.

- A *tree* is an undirected graph without cycles.
- A directed graph without cycles is called a *directed acyclic graph* or *DAG*.[1]

As mentioned at the end of class today, it's important for cycles in dependency networks to be avoided. Otherwise, deadlock occurs—a series of processes each wait for another process to finish, but that process itself is waiting for another process in the group to finish. Given their importance in dependency networks, it's worth considering if we can efficiently detect that a directed graph is without cycles, and if so, how to schedule processes so all get the resources they need.

1. **Detecting Cycles in Directed Graphs.** Sketch out an $O(n + m)$ time algorithm that detects cycles in directed graphs.

2. **Directed Acyclic Graphs.** Recall the *in-degree* of a vertex v is the number of edges coming into v. In other words, it is the number of u such that (u, v) is an edge.

   Show that if $G = (V, E)$ is a directed acyclic graph, then there exists a vertex $v \in V$ with in-degree zero.

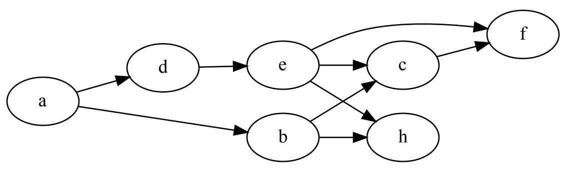
3. **Topological Sorting**. For applications like the deadlock problem, once we have a DAG, we'd like to know which processes should be served in which order. This is known as a *topological ordering.* A *topological ordering* of a DAG $G = (V, E)$ is an ordering of vertices $v_1, \ldots, v_n$ such that $i < j$ for all $(v_i, v_j) \in E$. The TOPOLIGICAL-SORTING (TOPSORT) problem takes a DAG G as input and outputs a topological ordering of G.

   Give an efficient algorithm for TOPSORT. What is the runtime of your topsort algorithm?

   Hint: use Problem 2 to drive your intuition.

   Note: Solving problem 2 is not necessary to solve this problem. Just assume the claim is true and use it to design an algorithm!

4. Consider the graph $G = (V, E)$ below.



[1] I guess they ran out of creative names at this point.

How many *different* topological orderings does G have?