CS41 Lab 5

September 29, 2022

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, strongly-connected components, and directed graphs without cycles.

- Vertices u and v are *strongly connected* if there are $u \rightsquigarrow v$ and $v \rightsquigarrow u$ paths in G. A *strongly connected component* is a set of vertices $C \subseteq V$ such that u, v are strongly connected for all $u, v \in C$ (and no other vertices are strongly connected to a vertex $u \in C$.)
- A *tree* is an undirected graph without cycles.
- A directed graph without cycles is called a *directed acyclic graph* or *DAG*.

One major application of directed graphs are dependency networks. In a typical dependency network, vertices represent processes, and an edge (u, v) means that process u is waiting on a resource that process v uses. 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. **Strongly-Connected Components.**

   (a) Design an $O(n + m)$ time algorithm FindSCC that takes as input a directed graph $G = (V, E)$ and a vertex $s \in V$, and returns a List of vertices in the strongly-connected component containing s.

   (b) Sketch an algorithm FindAllSCCs that takes as input a directed graph $G = (V, E)$ and identifies all strongly-connected components of G. What is the runtime of your algorithm?

2. **Directed Acyclic Graphs.** 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**. 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. **Cycles in Directed Graphs.** Determine whether or not the graph below is *acyclic*. If the graph is cyclic, identify a cycle. If the graph is acyclic, give a topological ordering of its vertices.