CSE 332: Data Structures and Parallelism

Section 9: Graphs & Connectivity

In lecture, we solved the **CONN** problem. That is,

CONN	
Input(s):	Graph G
Output:	`true` iff G is connected

We used a `WorkList` algorithm:

```
isConnected(G) {
   V, E = G
   worklist = first(V);
   seen = {v};
   while (worklist.hasWork()) {
      v = worklist.next();
      for (w : v.neighbors()) {
         if (w ∉ seen) {
            worklist.add(w);
            seen.add(w);
         }
      }
   }
   return seen == V;
}
```

This algorithm has several distinct names based on which type of WorkList we use. If we use a `FIFOQueue`, it's called Breadth-First Search (BFS), and if we use a `Stack`, it's called Depth-First Search (DFS).

0. Recursively, Now!

Although we've implemented it here and in lecture iteratively, we could implement **DFS** recursively as well. Write Pseudo-code for a DFS that uses recursion.

1. Two-Coloring

A graph G is two-colorable if and only if we can make a function $f : V \to \{\text{red}, \text{black}\}$ such that $\{u, v\} \in G \to f(u) \neq f(v)$. That is "adjacent vertices have different colors". Write an algorithm to solve the **2-COLOR** problem.

2. A Social Networking Event

Suppose you have social network data for some people (including yourself and a famous person). Explain how to use a graph algorithm to find the answers to the following questions:

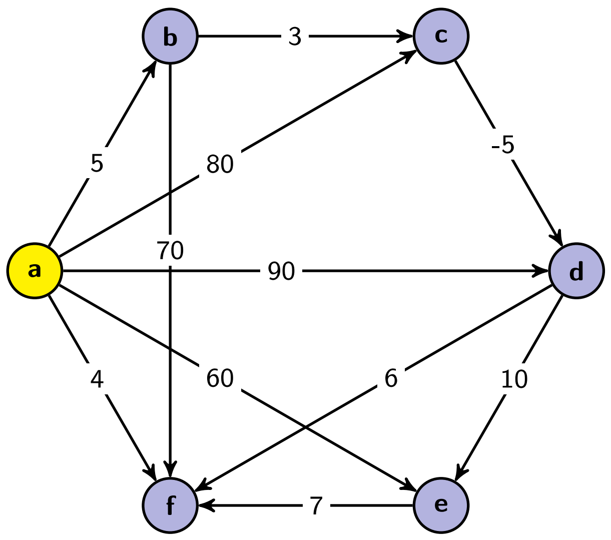
(a) Find the person with the *most friends* in the data?

(b) Find *the length of the shortest path* from yourself to the famous person.

(c) Find *the number of people* who do not know anyone you know.

3. Better Find the Shortest Path Before It Catches You!

Consider the following graph:



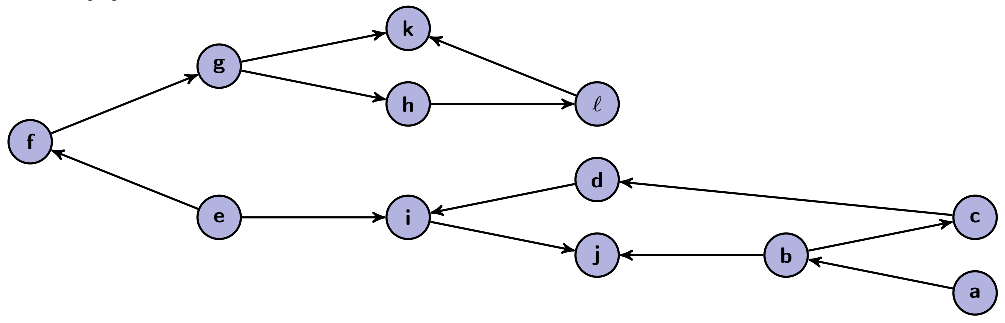
(a) Use Dijkstra's Algorithm to find the **lengths** of the shortest paths from **a** to each of the other vertices. For full credit, you must show the worklist at every step, but how you show it is up to you.

(b) Are any of the lengths you computed using Dijkstra's Algorithm in part (a) incorrect? Why or why not?

(c) Explain how you would use Dijkstra's Algorithm to recover the actual paths (rather than just the lengths).

4. It Rhymes with Flopological Sort

Consider the following graph:



(a) Does this graph have a topological sort? Explain why or why not. If you answered that it does not, remove the **MINIMUM** number of edges from the graph necessary for there to be a topological sort and carefully mark the edge(s) you are removing. Otherwise, just move on to the next part.

For the remaining parts, work with this (potentially) new version of the graph.

(b) Find a topological sort of the graph. Do not bother showing intermediary work.

(c) If this graph represented various tasks in a `ForkJoin` algorithm, what would the work of the algorithm be assuming each individual task is $\Theta(1)$.

Hint: There are 12 tasks...