CSE 332: Data Abstractions

Section 8: Graphs & Connectivity Solutions

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:

```
1  isConnected(G) {
2     V, E = G
3     worklist = first(V);
4     seen = {v};
5     while (worklist.hasWork()) {
6        v = worklist.next();
7        for (w : v.neighbors()) {
8           if (w ∉ seen) {
9              worklist.add(w);
10             seen.add(w);
11          }
12       }
13    }
14    return seen == V;
15 }
```

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 Psuedo-code for a DFS that uses recursion.

Solution:

```
1  dfs(G) {
2     dfs(first(G), {})
3  }
4
5  dfs(v, seen) {
6     if (v ∈ seen) {
7        return;
8     }
9     for (w : v.neighbors()) {
10       seen = seen ∪ {w}
11       dfs(w, seen)
12    }
13 }
```

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.

Solution:

Instead of storing the vertices in a "seen" set, store them in a dictionary from $V \to \{\text{red}, \text{black}\}$. Then, attempt to make a coloring (start with, say black) using a DFS (starting from any vertex). If we try to color a vertex two different colors, there is no two-coloring. If we finish, it works.

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?

Solution:

Instead of storing the vertices in a "seen" set, store them in a dictionary from $V \to \texttt{int}$. Then, do a BFS/DFS (starting from any vertex), where we store the number of neighbors when we remove a vertex. Additionally, as we continue, also store the largest number of neighbors we've seen so far (and the vertex who had those neighbors).

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

Solution:

Do a BFS starting at yourself. In addition to the normal stopping condition (we've seen this vertex before), stop when we hit the famous person. In the case where we fail to find the famous person, return ∞. Otherwise, return the number of vertices we've seen so far.

Notice that a DFS *does not work here*! A DFS will return *some path*, but not necessarily the shortest one.

(c) Find *the number of people* who you are not friends with.

Solution:

Do a BFS/DFS starting at yourself. The answer is $|V| - |seen|$.