CS 405: Algorithm Analysis II  
Homework 3: Distinct paths

Given a directed acyclic graph $G = (V, E)$ and two vertices $u, v \in V$, construct a linear time algorithm, $O(V + E)$, that returns the number of distinct simple paths from $u$ to $v$. A path is a sequence of vertices $(u = w_1, w_2, \ldots, w_k = v)$ such that $(w_{i-1}, w_i) \in E$ for $1 \leq i \leq k$. Two paths are distinct if their sequences have different length or if their sequences have the same length but differ in some component. Submit the pseudo-code for your algorithm and explain why it is linear. You need not write an executable Java program.

Solution.

The topological sort algorithm runs DFS and stacks the vertices as they finish. Popping this stack recovers vertices in order of decreasing finish time, and this order constitutes a topological sort in which all edges point forward in the listing. Therefore if $u.f < v.f$, then $u$ follows $v$ in the listing and there can be no paths from $u$ to $v$. On the other hand, if $u.f > v.f$, then $v$ follows $u$ in the listing and any nodes that participate in a path from $u$ to $v$ must lie between $u$ and $v$ in the listing.

The plan for our algorithm is to run DFS as in topological sort, adding each vertex to the beginning of a linked list as it finishes. If $u$ finishes before $v$, we report zero for the number of distinct path from $u$ to $v$. If $v$ finishes first, then we place a count attribute in each vertex, say cnt. We initialize $v.cnt$ to 1, acknowledging that there exists exactly one path from $v$ to $v$, namely the path with zero links. We initialize $w.cnt$ to zero for all other vertices. Now, when the next node finishes, say $x$, it may have paths to $v$. If so, nodes on those paths have already finished, because all edges point toward earlier finish times. In particular, the number of distinct paths from $x$ to $v$ will be

$$x.cnt = \sum_{y \in x.adj} y.cnt.$$  

In the summation, any nodes $y \in x.adj$ that have finish times less than $v.f$ will contribute zero to the sum. They correspond to neighbors of $x$ that are downstream from $v$ and therefore can never continue on to $v$. By induction, we can assume that nodes $y \in x.adj$ that have nonzero cnt attributes are such that the nonzero cnt is the number of distinct paths from $y$ to $v$. The summation therefore gives the number of distinct paths from $x$ to $v$. We can continue the algorithm until it terminates, or we can interrupt it when $u$ finishes. In either case, $u.cnt$ contains the number of distinct paths from $u$ to $v$. With some variation for efficiency, the code is as follows. Note that each node visited may have its adjacency list scanned twice instead of once as in the topological sort algorithm. The complexity remains $\Theta(V + E)$.

```java
PathCount(G, u, v) { // G is directed acyclic
    for w in G.V { // u, v in G.V
        w.cnt = 0;
        w.color = white;
        w.pi = null;
        list = null; // initialize finish-time stack
    }
    time = 0;
    for w in G.V {
        if (w.color == white)
            Visit(G, u, v, w);
    }
}

Visit(G, u, v, w) {
    time = time + 1;
    w.d = time;
    w.color = gray;
    for x in w.adj
        if (x.color == white)
            x.pi = w;
        Visit(G, u, v, x);
    w.color = black;
    time = time + 1;
    w.f = time;
    if (w == v)
        v.cnt = 1;
    else if (v.cnt == 1)
        for x in w.adj
            w.cnt = w.cnt + x.cnt;
}
```