CS360 Assignment 7

22.1-1 Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degrees?

To compute the *out-degree* of every vertex will simply entail determining the number of vertices in each list since these are all the vertices for which there is an edge from the given vertex. Thus we can create an array $A[]$ of length $|V|$ to store the in-degree and initialize all the entries to $0 \Rightarrow O(|V|)$. Then we simply determine the length of each list such that for each edge $(u, v) \in G.E$ we increment $A[u]$ which gives $\sum_{u \in G.V} |Adj(u)| = O(|E|)$ since each edge is represented *once* in the adjacency list. Hence the total time to compute the in-degree of all the vertices is $O(|V|) + O(|E|) = O(|V| + |E|)$.

For the *in-degree* we perform a similar iteration over each list in the adjacency lists but for each edge $(u, v) \in G.E$ rather than incrementing the counters for $A[u]$ we instead increment the counters for $A[v]$. Thus we will still require $O(|V|)$ initialization and $O(|E|)$ for the iteration over the edges again giving a total time for the in-degree of $O(|V| + |E|)$.

22.2-7 Separating professional wrestlers into "babyfaces" and "heels".

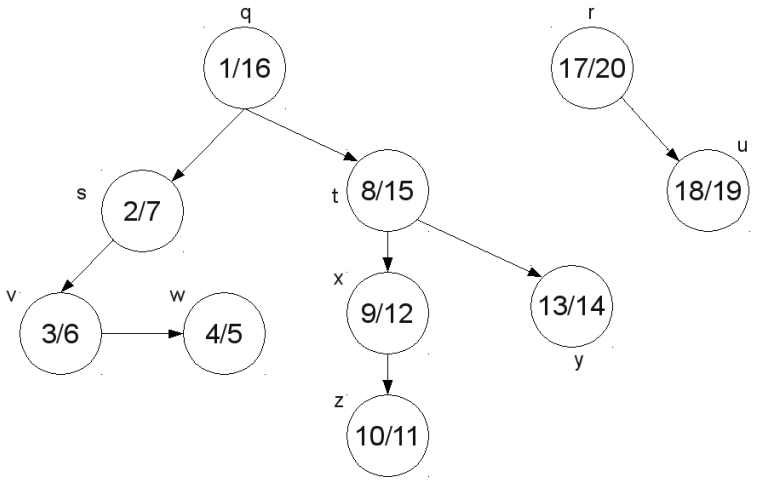
Create a graph with n vertices, one vertex per wrestler. Then add r edges such that there is an edge between vertices for each rivalry. Run BFS on this graph starting at any vertex and repeat as needed to have each vertex appear in one of the resulting breadth-first trees which takes $O(n + r)$. Label each vertex as "babyface" or "heel" depending on whether the distance from the source (i.e. # of edges) is even or odd which takes $O(n)$. Clearly the assignment of sources does not affect the results since they exist in separate trees, i.e. there are by definition no edges between the trees. Finally check every edge (including ones not in the trees) to see if it is between vertices in different categories (returning that no separation is possible if any edge is between vertices in the same category) which takes $O(r)$. Thus the total run time is $O(n + r) + O(n) + O(r) = O(n + r)$.

22.4-3 Determine if an undirected graph contains a cycle in $O(V)$.

By Theorem 22.10 we know that DFS on an undirected graph produces only *tree* or *back* edges. Furthermore, by Theorem 22.11, a graph is acyclic if and only if a DFS produces no *back* edges. Therefore if we run DFS on an undirected graph and get no *back* edges, then the graph contains no cycles since every edge must then be a *tree* edge. However, DFS runs in $O(V + E)$ not $O(V)$. If we observe that by Theorem B.2.5, an undirected graph is acyclic if and only if $|E| = |V| - 1$ therefore if during DFS we explore more than $|V|$ edges at least one must be a *back* edge (if none were previously found). Therefore the algorithm can be terminated after only checking $|V|$ edges and hence runs in $O(V)$ time (although

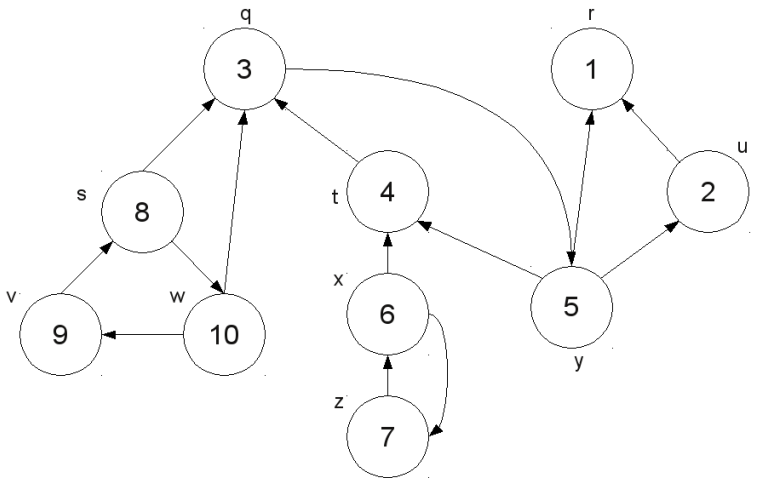
not necessarily to completion).

22.5-2 Running STRONGLY-CONNECTED-COMPONENTS() on figure 22.6 gives the initial DFS:

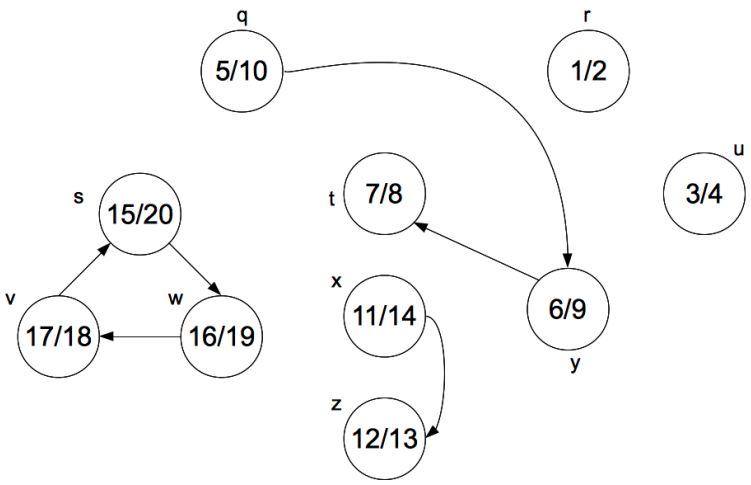


Thus sorting the vertices by decreasing finishing times gives $\{r, u, q, t, y, x, z, s, v, w\}$.

Computing G^T produces the following graph where the new ordering is indicated inside each vertex.



Running DFS on the previous graph gives



It is clear from the above graph that there are 5 strongly connected components - $\{r\}, \{u\}, \{q, y, t\}, \{x, z\}, \{s, w, v\}$.