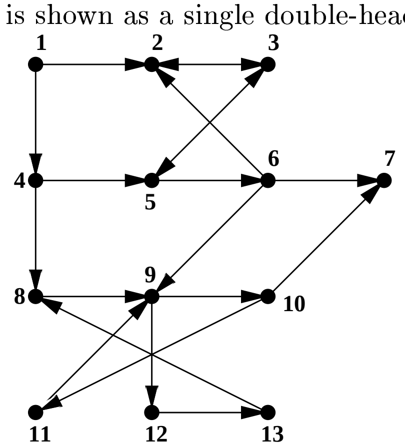
1. Run the strongly connected components algorithm on the following graph. For each of the two DFS's, label each edge as a tree, forward, back or cross edge. Label each vertex with its strongly connected component number. Wherever there is an option to visit more than one vertex, visit the lowest numbered vertex first. For ease of drawing, a pair of edges $(u, v)$ and $(v, u)$ is shown as a single double-headed arrow connecting $u$ and $v$.




2. Suppose you add a new single edge to a directed graph. What is the largest change in the number of strongly connected components that can result from this addition? Prove your answer, and give an example.

3. Give an efficient algorithm that takes as input a DAG G(V, E), and two vertices s, t and outputs the number of paths from s to t in G(V, E). Make your algorithm be as fast as you can. Show that it is correct, and analyze its complexity. Is the number of paths necessarily bounded by a polynomial function of the number of vertices and number of edges? Give either a proof or a counterexample.

4. CLR 23-2.

   Note: There is a bug in part b. It should read:

   > Let $v$ be a nonroot vertex in $G_\pi$. Prove that $v$ is an articulation point of $G$ if and only if there is a child $v'$ of $v$, such that there is no backedge from a descendant of $v'$ (including $v'$ itself) to a proper ancestor of $v$.

5. CLR 23.5-3.

6. CLR 23.5-7.

7. Devise an efficient algorithm to find the *transitive closure* of a directed graph $G$ represented with adjacency lists. The transitive closure of a directed graph $G$ is a graph $G*$ with the same vertices as $G$, in which vertices $u$ and $v$ are adjacent exactly when there is a directed path from $u$ to $v$ in $G$. (The result of your algorithm should also be represented with adjacency lists.) Make your algorithm be as fast as you can. Give a running time analysis of your algorithm, along with a justification of the estimate and the algorithm's correctness.