Homework 2

Handed out Friday, July 16. Due at the start of class Friday, July 23. Late homeworks will not be accepted.

Problem 1. Given a directed acyclic graph $G = (V, E)$ and two vertices $x, y \in V$ we want to study the number of distinct paths joining them.

(a) Present an algorithm for counting the number of paths between $x$ and $y$, notice that you are not asked to actually find or output the paths. Your algorithm should run in linear time.

(b) Suppose $G$ has $n$ vertices. Give an upper bound on the number of paths between any two vertices as a function of $n$, show an example that attains your bound.

Problem 2. In the strongly connected components algorithm presented in class it seems like we are making our existence harder that it should be. Do we really need to use $G^T$? What about the following “more intuitive” algorithm:

i call DFS($G$) to compute finishing times $f[u]$ for every vertex $u$
ii call DFS($G$) but process the vertices in the main loop in increasing order of $f[u]$.
iii output the vertices of each tree in the depth-search forest as a separate strongly connected component.

Prove or disprove the correctness of the above algorithm.

Problem 3. A cut edge or bridge of a connected graph $G$ is an edge whose removal disconnects $G$. Design an algorithm with running time $O(V + E)$ to find all the cut edges of an input graph.

Problem 4. Let $G$ be a directed graph. The vertices of $G$ have been numbered $1 \ldots n$ (where $n$ is the number of vertices in $G$). Let $small(i) = \min\{j \mid j$ is reachable from $i\}$. In other words, for a vertex numbered $i$, $small(i)$ is the smallest numbered vertex reachable from it. Design an $O(V + E)$ algorithm to compute $small(i)$ for all vertices in the graph.

Problem 5. An alternative algorithm for topologically sort a directed acyclic graph is to iteratively find a vertex with in-degree 0, output the vertex and remove it from the graph. Implement the algorithm so it runs in $O(V + E)$ time and prove that if run on a DAG, it will output the vertices in topological order. What happens if the input graph has a cycle?

Problem 6. A directed graph is said to be half-connected if, for any two vertices $u, v \in V$ we have that $u$ can reach $v$ or $v$ can reach $u$. Give an efficient algorithm to determine whether or not $G$ is half-connected. Prove that your algorithm is correct and analyse its running time.