Illinois Institute of Technology
Department of Computer Science

Homework 9 Solutions

CS 430 Introduction to Algorithms
Spring Semester, 2014

1. **Problem 1** Problem 22.4-2 (Pg 614) CLRS(3rd Edition): Give a linear-time algorithm that takes as input a directed acyclic graph $G = (V, E)$ and two vertices s and t, and returns the number of simple paths from s to t in G. For example, the directed acyclic graph of Figure 22.8 contains exactly four simple paths from vertex p to vertex v: *pov* , *poryv* , *posryv* , and *psryv* . (Your algorithm needs only to count the simple paths, not list them.) (20 pts)
Solution: For each vertex v from $s \to t \in V$ sorted topologically, count the vertices adjacent to v and add them to get the number of simple paths. Let $P[v]$ be the number of paths from v to t, so the recurrence is $P[v] = \sum_{u \in adj(v)} P[u]$ where $adj(u)$ is all the vertices adjacent to v s.t $(u, v) \in E$. So the time taken is $O(V + E)$.

2. **Problem 2** Problem 22-1 (Pg 621) CLRS(3rd Edition):A depth-first forest classifies the edges of a graph into tree, back, forward, and cross edges. A breadth-first tree can also be used to classify the edges reachable from the source of the search into the same four categories.

   (a) Prove that in a breadth-first search of an undirected graph, the following properties hold:

      i. There are no back edges and no forward edges.
      Solution: Suppose (u, v) is a back edge or a forward edge in a BFS of an undirected graph. Then one of u and v, say u, is a proper ancestor of the other (v) in the breadth-first tree. Since we explore all edges of u before exploring any edges of any of us descendants, we must explore the edge (u, v) at the time we explore u. But then (u, v) must be a tree edge.

      ii. For each tree edge (u, v) , we have $v.d = u.d + 1$.
      Solution: In BFS, an edge (u, v) is a tree edge when we set $v.\pi = u$. But we only do so when we set $v.d = u.d + 1$. Since neither $u.d$ nor $v.d$ ever changes thereafter, we have$v.d = u.d + 1$ when BFS completes.

      iii. For each cross edge (u, v), we have $v.d = u.d$ or $v.d = u.d + 1$.
      Solution: Consider a cross edge (u, v) where, without loss of generality, u is visited before v. At the time we visit u, vertex v must already be on the queue, for otherwise (u, v) would be a tree edge. Because v is on the queue, we have $v.d \leq u.d + 1$ by Lemma 22.3. By Corollary 22.4, we have $v.d \geq u.d$. Thus, either $v.d = u.d$ or $v.d = u.d + 1$.

   (b) Prove that in a breadth-first search of a directed graph, the following properties hold:

      i. There are no forward edges.
      Solution: Suppose (u, v) is a forward edge. Then we would have explored it while visiting u, and it would have been a tree edge.

      ii. For each tree edge (u, v) , we have $v.d = u.d + 1$.
      Solution: Same as for undirected graphs.

      iii. For each cross edge (u, v), we have $v.d \leq u.d + 1$.
      Solution: For any edge (u, v), whether or not its a cross edge, we cannot have $v.d > u.d + 1$, since we visit v at the latest when we explore edge (u, v). Thus,$v.d \leq u.d + 1$

iv. For each back edge (u, v), we have $0 \leq v.d \leq u.d$
Solution: Clearly, $v.d \geq 0$ for all vertices v. For a back edge (u, v), v is an ancestor of u in the breadth-first tree, which means that $v.d \leq u.d$.

(20 pts)

3. **Problem 3** Problem 22-4 (Pg 623) CLRS(3rd Edition): Let $G = (u, v)$ be a directed graph in which each vertex $u \in V$ is labeled with a unique integer $L(u)$ from the set $\{1, 2, ..., |V|\}$. For each vertex $u \in V$, let $R(u) = \{v \in V : u \to v\}$ be the set of vertices that are reachable from u. Define $min(u)$ to be the vertex in $R(u)$ whose label is minimum, i.e., $min(u)$ is the vertex v such that $L(v) = min\{L(w) : w \in R(u)\}$. Give an $O(V + E)$-time algorithm that computes $min(u)$ for all vertices $u \in V$. (20 pts)
Solution: Compute G^T in the usual way, so that G^T is G with its edges reversed. Then do a depth-first search on G^T , but in the main loop of DFS, consider the vertices in order of increasing values of $L(v)$. If vertex u is in the depth-first tree with root v, then $min(u) = v$. Clearly, this algorithm takes $O(V + E)$ time.

To show correctness, first note that if u is in the depth-first tree rooted at v in G^T ,then there is a path $v \to u$ in G^T , and so there is a path $v \to u$ in G. Thus, the minimum vertex label of all vertices reachable from u is at most $L(v)$, or in other words,$L(v) \geq \min L(w) : w \in R(u)$.

Now suppose that $L(v) > \min L(w) : w \in R(u)$, so that there is a vertex $w \in R(u)$ such that $L(w) < L(v)$. At the time $v.d$ that we started the depth-first search from v, we would have already discovered w, so that $w.d < v.d$. By the parenthesis theorem, either the intervals $[v.d, v.f]$, and $[w.d, w.f]$ are disjoint and neither v nor w is a descendant of the other, or we have the ordering$w.d < v.d < v.f < w.f$ and v is a descendant of w. The latter case cannot occur, since v is a root in the depth-first forest (which means that v cannot be a descendant of any other vertex). In the former case, since $w.d < v.d$, we must have $w.d < v.d < v.f < w.f$. In this case, since u is reachable from w in G^T , we would have discovered u by the time $w.f$, so that $u.d < w.f$. Since we discovered u during a search that started at v, we have $v.d \leq u.d$. Thus, $v.d \leq u.d < w.f < v.d$, which is a contradiction. We conclude that no such vertex w can exist.

4. **Problem 4** Problem 23.2-5 (Pg 637) CLRS(3rd Edition):Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Prims algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W ? (20 pts)
Solution: The time taken by Prims algorithm is determined by the speed of the queue operations. With the queue implemented as a Fibonacci heap, it takes $O(E + VlgV)$ time. Since the keys in the priority queue are edge weights, it might be possible to implement the queue even more efficiently when there are restrictions on the possible edge weights.

We can improve the running time of Prims algorithm if W is a constant by implementing the queue as an array $Q[0, , W + 1]$ (using the $W + 1$ slot for key=∞),where each slot holds a doubly linked list of vertices with that weight as their key. Then EXTRACT-MIN takes only $O(W) = O(1)$ time (just scan for the first nonempty slot), and DECREASE-KEY takes only $O(1)$ time (just remove the vertex from the list its in and insert it at the front of the list indexed by the new key).This gives a total running time of $O(E)$, which is the best possible asymptotic time(since $\omega(E)$ edges must be processed).

However, if the range of edge weights is 1 to $|V|$, then EXTRACT-MIN takes $\theta(V)$ time with this data structure. So the total time spent doing EXTRACT-MIN is $\theta(V^2)$, slowing the algorithm to $\theta(E + V^2) = \theta(V^2)$. In this case, it is better to keep the Fibonacci-heap priority queue, which gave the $\theta(E + VlgV)$ time.