EECS 114: Hw2 Solution

November 29, 2012

1 Solution to Exercise 22.3-4

The DFS and DFS-VISIT procedures care only whether a vertex is white or not. By coloring vertex \( u \) gray when it is first visited, in line 3 of DFS-VISIT, we ensure that \( u \) will not be visited again. Once we have changed a vertex’s color to non-white, we do not need to change it again.

2 Solution to Exercise 22.4-3

An undirected graph is acyclic (i.e., a forest) if and only if a DFS yields no back edges.

- If there’s a back edge, there’s a cycle.
- If there’s no back edge, then by Theorem 22.10, there are only tree edges. Hence, the graph is acyclic.

Thus, we can run DFS: if we find a back edge, there’s a cycle.

- Time: \( O(V) \). (Not \( O(V + E)! \))

  If we ever see |\( V \)| distinct edges, we must have seen a back edge because (by Theorem B.2 on page 1174) in an acyclic (undirected) forest, |\( E \)| ≤ |\( V \)| − 1.

3 Solution to Exercise 23.2-4

We know that Kruskals algorithm takes \( O(V) \) time for initialization, \( O(E \lg E) \) time to sort the edges, and \( O(E \alpha(V)) \) time for the disjoint-set operations, for a total running time of \( O(V + E \lg E + E \alpha(V)) = O(E \lg E) \).

If we knew that all of the edge weights in the graph were integers in the range from 1 to |\( V \)|, then we could sort the edges in \( O(V + E) \) time using counting sort. Since the graph is connected, \( V = O(E) \), and so the sorting time is reduced to \( O(E) \). This would yield a total running time of \( O(V + E + E \alpha(V)) = O(E \alpha(V)) \), again since \( V = O(E) \), and since \( E = O(E \alpha(V)) \). The time to process the edges, not the time to sort them, is now the dominant term. Knowledge about the weights won't help speed up any other part of the algorithm, since nothing besides the sort uses the weight values.

If the edge weights were integers in the range from 1 to \( W \) for some constant \( W \), then we could again use counting sort to sort the edges more quickly. This time, sorting would take \( O(E + W) = O(E) \) time, since \( W \) is a constant. As in the first part, we get a total running time of \( O(E \alpha(V)) \).

4 Solution to Exercise 24.1-3

If the greatest number of edges on any shortest path from the source is \( m \), then the path-relaxation property tells us that after \( m \) iterations of BELLMAN-FORD, every vertex \( v \) has achieved its shortest-path weight in \( v.d \). By the upper-bound property, after \( m \) iterations, no \( d \) values will ever change. Therefore, no \( d \) values will change in the \((m + 1)\)st iteration. Because we do not know \( m \) in advance, we cannot make the algorithm iterate exactly \( m \) times and then terminate. But if we just make the algorithm stop when nothing changes any more, it will stop after \( m + 1 \) iterations.
BELLMAN-FORD-(M+1) (G,w,s)  
INITIALIZE-SINGLE-SOURCE(G,s)  
changes = TRUE  
while changes == TRUE  
  changes = FALSE  
  for each edge (u,v) in G.E  
    RELAX-M(u,v,w)  

RELAX-M(u,v,w)  
  if v.d > u.d + w(u,v)  
    v.d = u.d + w(u,v)  
    v.pi = u  
    changes = TRUE

The test for a negative-weight cycle (based on there being a \(d\) value that would change if another relaxation step was done) has been removed above, because this version of the algorithm will never get out of the while loop unless all \(d\) values stop changing.

5 Solution to Exercise 24.3-3

Yes, the algorithm still works. Let \(u\) be the leftover vertex that does not get extracted from the priority queue \(Q\). If \(u\) is not reachable from \(s\), then \(u.d = \delta(s,u) = \infty\). If \(u\) is reachable from \(s\), then there is a shortest path \(p = s \leadsto x \rightarrow u\). When the vertex \(x\) was extracted, \(x.d = \delta(s,x)\) and then the edge \((x,u)\) was relaxed; thus, \(u.d = \delta(s,u)\).