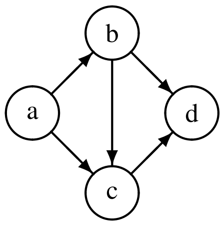
COMP4500/7500 Advanced Algorithms & Data Structures Sample Solution to Tutorial Exercise 5 (2014/2)[*]

School of Information Technology and Electrical Engineering, University of Queensland

September 2, 2014

1. The following is a small example of a directed acyclic graph (dag).



In this dag there is only one topological ordering: *abcd*. There are three paths from *a* to *d*: *abd, abcd,* and *acd*.

A directed acyclic graph (dag) may have many distinct topological orderings. Give a dag with n nodes which has the minimum possible number of topological orderings? Which dag has the maximum number?

Sample solution. If we number the nodes from 1 to n, then the graph with edges of the form $(i, i+1)$ for $1 \leq i < n$, has only one topological order – the nodes must be arranged in increasing order from 1 to n.

The graph with no edges allows any ordering of the nodes to be a topological order. There are $n!$ possible topological orderings of such a graph because there are n choices for the first vertex, $n-1$ choices for the second vertex, and so on.

2. Design an abstract algorithm to list all the topological orderings of a given directed acyclic graph. You may use the abstract traversals of graphs, and make use of abstract set and sequence operations, e.g., set union and subtraction, and sequence concatenation. You may also assume the procedure to calculate the in-degrees of a graph (from Revision 4) is available. What are the worst-case time and space complexities of your algorithm?

Sample solution. The algorithm first calculates the in-degrees of all the vertices of the graph. All the vertices that have in-degree 0 may appear as the first vertex in a topological ordering of the vertices. For each such vertex, v, all topological orderings are determined. This is done by a recursive call on LIST-TOPOLOGICAL-ORDERS with the sequence extended with the vertex and the induced subgraph obtained by deleting the vertex as arguments.

We assume operations to write out the current value of a sequence, append a vertex to a sequence, delete a vertex (and any associated edges) from a graph, and determine the vertices of a graph with in-degree 0. Note that APPEND creates a new sequence without changing S and the new sequence is passed to the recursive call. Similarly, DELETE creates a new graph without changing G.

```
LIST-TOPOLOGICAL-ORDERS($S, G$)
1  if $Empty(G)$
2      WRITE-SEQUENCE($S$)
3  else
4      IN-DEGREE($G$)
5      for each vertex $v \in G.V$
6          if $v.indegree$ == 0
7              LIST-TOPOLOGICAL-ORDERS(APPEND($S, v$), DELETE($G, v$))
```

The initial call on the above procedure should be of the form

[*]Copyright © reserved September 2, 2014

```
ListTopologicalOrders($EmptySequence, G$)
```

The function Delete(G, v) returns a copy of the graph G with the vertex v and all edges leading out of v removed.

The worst case time complexity satisfies the recurrence $T(n) = nT(n-1)+\Theta(n^2)$, which means $T(n) = O(n!)$, where $n = |V|$.

The worst case space complexity is found by determining the maximum stack depth, which is n. At each level we use $\Theta(n^2)$ space, which means the space complexity is $O(n^3)$. Yes, it is easy to write meaningful programs which use only polynomial space but take exponential time. (Note: this does not count the length of the output, which may be exponential. The output is not all stored simultaneously.)

3. Find a recurrence equation which computes the number of paths between two vertices of a directed acyclic graph.

   Sample solution. Because an acyclic graph cannot have a self-loop, there are no paths from a vertex a to itself, i.e.,
$$N(a, a) = 0.$$
   A path from a to v may pass through any predecessor of v before reaching v, so
$$N(a, v) = \sum_{w \in P(v)} N(a, w),$$
   where $P(v)$ is the set of predecessors of v.
$$P(v) = \{w : G.V \mid (w, v) \in G.E\}$$
   Note that, if there are no predecessors of a node then the sum is zero.

4. (CLRS Exercise 23.1-7; CLR Exercise 24.1-7)

   Argue that if all edge weights of a graph are positive, then any subset of edges that connects all vertices and has minimum total weight must be a tree. Give an example to show that the same conclusion does not hold if we allow some weights to be nonpositive.

   Sample solution. Assume we have a non-tree subset. Then it contains a back-edge. Remove the back-edge and we obtain a smaller subset with lower weight (since all edge weights are positive) which spans the same set of vertices. A simple example which shows this does not hold when we allow nonpositive weights is to have a back-edge with zero weight.

5. The Bellman-Ford algorithm solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ as long as there are no negative weight cycles. The algorithm returns TRUE if and only if the graph contains no negative weight cycles that are reachable from the source. In that case the final shortest-path weight from the source s to a vertex v is stored in $v.d$ at the termination of the algorithm. For each vertex v, on termination $v.\pi$ stores the predecessor on the path from s to v. The following pseudo-code describes an implementation of the Bellman-Ford algorithm.These procedures are taken from CLRS p648–651 [3rd].

```
Bellman-Ford($G, w, s$)
// $G$ is the graph, $w$ the weight function, $s$ the source vertex
Init-Single-Source($G, s$)
for $i = 1$ to $|G.V| - 1$
    for each edge $(u, v) \in G.E$
        Relax($u, v, w$)
for each edge $(u, v) \in G.E$
    if $v.d > u.d + w(u, v)$
        return FALSE
return TRUE
```

```
INIT-SINGLE-SOURCE($G, s$)
1  for each vertex $v \in G.V$
2      $v.d = \infty$
3      $v.\pi = $ NIL
4  $s.d = 0$
```

```
RELAX($u, v, w$)
1  if $v.d > u.d + w(u, v)$
2      $v.d = u.d + w(u, v)$
3      $v.\pi = u$
```

Consider the graph on vertices v_1, v_2, v_3, v_4, v_5 described by the following adjacency matrix w:

	v_1	v_2	v_3	v_4	v_5
v_1	∞	5	8	-4	∞
v_2	-2	∞	∞	∞	∞
v_3	∞	-3	∞	9	∞
v_4	∞	7	∞	∞	2
v_5	6	∞	7	∞	∞

For vertices u and v, an entry x in row u and column v of the adjacency matrix means that the distance from u to v on the directed edge (u, v) is x, that is, $w(u, v) = x$. If the distance is shown as ∞ then u is not connected to v in that direction.

(a) Draw a pictorial representation of the graph, showing vertices, edges and distances.

(b) Show how the Bellman-Ford algorithm runs on this directed graph, using vertex v_5 as the source. Make the following assumptions about the implementation. An adjacency list representation is used in such a way that edges are made available in lexicographic order in the loops:
"**for** each edge $(u, v) \in G.E$".
This means that (v_i, v_j) precedes (v_k, v_ℓ) if $i < k$ or if $i = k$ and $j < \ell$.
Give your answer by showing the values of $v.d$ and $v.\pi$ (as used in the pseudocode) for each vertex v after each iteration of the first **for** loop in the BELLMAN-FORD procedure.

(c) What is the return value of the BELLMAN-FORD procedure in this case (TRUE or FALSE)?

(d) Analyse the complexity of the Bellman-Ford algorithm considering the most appropriate graph representation. Give your answer using O-notation in terms of $|V|$ and $|E|$, where these represent the numbers of vertices and edges, respectively. Make your bound as tight as possible. Provide a justification of your analysis.

Sample solution. The Bellman-Ford algorithm ... It is answered (pictorially) in CLRS 24.1 p652 [3rd] and CLR 25.3.

6. (CLR Exercise 25.1-4)
Let $G = (V, E)$ be a weighted, directed graph with source vertex s, and let G be initialized by the above procedure INIT-SINGLE-SOURCE(G, s). Prove that if a sequence of relaxation steps sets $s.\pi$ to a non-NIL value, then G contains a negative-weight cycle.

Sample solution. Whenever RELAX sets $v.\pi$ for a vertex v, it also reduces $v.d$. Thus, if $s.\pi$ gets set to a non-NIL then $s.d$ is reduced from its initial value of 0 to a negative number. But $s.d$ is the weight of some path from s to s, which is a cycle including s, i.e., a negative weight cycle.

7. Design an algorithm to compute single-source shortest paths in a directed acyclic graph in $O(|V| + |E|)$. (Hint: start by doing a topological sort.)

Sample solution. This is covered in CLRS 24.2; CLR 25.4:

```
DAG-SHORTEST-PATHS$(G, w, s)$
// $G$ is the graph, $w$ the weight function, $s$ the source vertex
$T$ = TOPOLOGICAL-SORT$(G)$
INITIALIZE-SINGLE-SOURCE$(G, s)$
for each vertex $u$ in $T$ // that is, taken in topologically sorted order
    for each vertex $v \in G.Adj(u)$
        RELAX$(u, v, w)$
```

8. (CLRS Exercise 24.1-3; CLR Exercise 25.3-3)

   Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all pairs of vertices $u, v \in V$ of the minimum number of edges in a shortest path from u to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple modification to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes.

   Sample solution. For every vertex v, $v.d$ has reached its final value if the number of iterations is greater than or equal to the length of any shortest weight path to v. So after m passes the algorithm can terminate. Since we do not know m in advance we make the algorithm stop when nothing changes in an iteration, i.e., after one extra pass.

```
BELLMAN-FORD2$(G, w, s)$
// $G$ is the graph, $w$ the weight function, $s$ the source vertex
INIT-SINGLE-SOURCE$(G, s)$
$changes$ = TRUE
while $changes$
    $changes$ = FALSE
    for each edge $(u, v) \in G.E$
        $changes$ = $changes$ or RELAX2$(u, v, w)$
```

```
RELAX2$(u, v, w)$
if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
    return TRUE
else
    return FALSE
```

Note that the test for a negative-weight cycle has been removed because this version of the algorithm will loop forever if there is one. Can you fix that?