CSC 226 A2 SOL

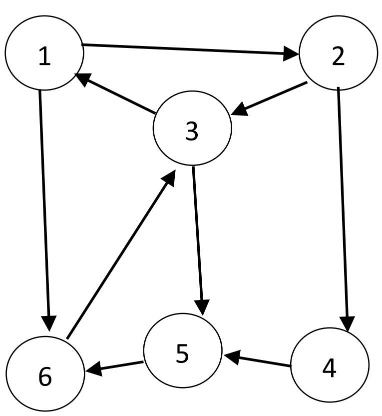
1. Unique topological ordering algorithm based on the iterative approach.

Algorithm uniqueTopSort(G)
Input: Digraph G with n vertices
Output: Unique topological ordering of G or an indication that no unique ordering exists

```
S ← Empty stack
for all u ∈ G.vertices() do
        incounter(u) ← indeg(u)
        if incounter(u) = 0 then
                S.push(u)
                if S.size() > 1 then
                        return "Ordering not unique."
i ← 1
while ¬S.isEmpty() do
        u ← S.pop()
        number u as vertex v_i
        i ← i + 1
for all  e ∈ G.outIncidentEdges(u)
        w ← G.opposite(u,e)
        incounter(w) ← incounter(w) - 1
        if incounter(w) = 0 then
                S.push(w)
                if S.size() > 1 then
                        return "Ordering not unique."
if i > n then
        return v_1, v_2, ..., v_n
return "G has a directed cycle"
```

2. Below are initial and resulting adjacency matrices of running Floyd-Warshall on the given graph G.

(a) G



G_0 Note: I have left out the 0 entries for readability

	1	2	3	4	5	6
1		1				1
2			1	1		
3	1				1	
4					1	
5						1
6			1			

(b) Here, I will indicate the new edges in red for each iteration.

G_1 - 2 new edges

	1	2	3	4	5	6
1		1				1
2			1	1		
3	1	1			1	1
4					1	
5						1
6			1			

G_2 - 3 new edges

	1	2	3	4	5	6
1		1	1	1		1
2			1	1		
3	1	1		1	1	1
4					1	
5						1
6			1			

G_3 - 8 new edges

	1	2	3	4	5	6
1		1	1	1	1	1
2	1		1	1	1	1
3	1	1		1	1	1
4					1	
5						1
6	1	1	1	1	1	

G_4 - 0 new edges (potential new edges already exist)

	1	2	3	4	5	6
1		1	1	1	1	1
2	1		1	1	1	1
3	1	1		1	1	1
4					1	
5						1
6	1	1	1	1	1	

G_5 - 1 new edges

	1	2	3	4	5	6
1		1	1	1	1	1
2	1		1	1	1	1
3	1	1		1	1	1
4					1	1
5						1
6	1	1	1	1	1	

G_6 - 7 new edges

	1	2	3	4	5	6
1		1	1	1	1	1
2	1		1	1	1	1
3	1	1		1	1	1
4	1	1	1		1	1
5	1	1	1	1		1
6	1	1	1	1	1	

3. Suppose we are given an unweighted, directed graph G with n vertices (labelled 1 to n), and let M be the $n \times n$ adjacency matrix for G (that is, $M(i, j) = 1$ if directed edge (i, j) is in G and 0 otherwise).

If $M^2(i, j) = 1$, then there exists some k such that $M(i, k) \cdot M(k, j) = 1$ which implies that $(i, k), (k, j) \in G$, meaning that there exists at least one path of length 2 from i to j in G.

If $M^2(i, j) = 0$, then for all k, $M(i, k) \cdot M(k, j) = 0$ which implies there is no path of length 2 from i to j in G.

In general, if $M^k(i, j) = 1$, then there is a path of length k from i to j and if $M^k(i, j) = 0$, there is not. If you add each M^k together, for $k = 1, \dots, n$, you then have all the paths in the graph G, i.e. the transitive closure.

4.

Algorithm 1 IterativeDFS(V, E, u)

```
Input: V = set of vertices, E = edges, u = start vertex
Output: discoveryEdges, backEdges
for each v ∈ V do
    visited[v] ← false,    parent[v] ← NIL
end for
discoveryEdges ← {},    backEdges ← {}
Create an empty stack S
S.push((u, NIL, 0))            ▷ Push frame (startVertex, parent, nextNeighborIndex=0)
while S is not empty do
    (v, p, i) ← S.top()        ▷ Peek stack without popping
    if ¬visited[v] then
        visited[v] ← true,    parent[v] ← p
    end if
    foundUnvisitedNeighbor ← false
    while i < length(Adj[v]) and ¬foundUnvisitedNeighbor do
        w ← Adj[v][i],    i ← i + 1        ▷ Move to next neighbor
        if ¬visited[w] then
            discoveryEdges.add((v, w))     ▷ Discovery edge
            S.pop(),    S.push((v, p, i))  ▷ Update stack frame
            S.push((w, v, 0))              ▷ Explore deeper
            foundUnvisitedNeighbor ← true
        else if w ≠ p then
            backEdges.add((v, w))          ▷ Back edge in undirected graph
        end if
    end while
    if ¬foundUnvisitedNeighbor then
        S.pop()                            ▷ Return from v's DFS call
    end if
end while
return (discoveryEdges, backEdges)
```