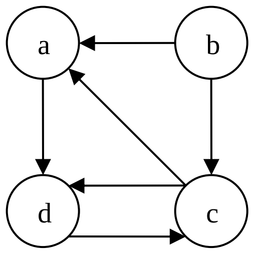
Warshall's Algorithm

Transitive closure method of finding path between nodes is quite inefficient because it processes the adjacency matrix lots of times. An efficient method for calculating transitive closure is by Warshall's algorithm whose time complexity is only $O(n^3)$.



We first find the matrices d_0, d_1, d_2, d_3, and d_4. The matrix d_4 is the transitive closure.

$$d_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad d_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1^* \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

d_1 has 1 as its (i,j)th entry if there is a path from vi to vj that has v1 = *a* as an interior vertex.

$$d_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad d_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1^* & 0 & 1 & 1^* \end{bmatrix}$$

d_2 has 1 as its (i,j)th entry if there is a path from v_i to v_j that has v_1 and v_2 as interior vertex. d_3 has 1 as its (i,j)th entry if there is a path from v_i to v_j that has v_1, v_2 and v_3 as interior vertex.

$$d_4 = \begin{bmatrix} 1^* & 0 & 1^* & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1^* & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Finally, d_4 has 1 as its (i,j)th entry if there is a path from v_i to v_j that has v_1, v_2, v_3 and v_4 as interior vertex.

The matrix d_4 is the transitive closure.

Algorithm

```
procedure warshall ($M_R$ : n x n zero one matrix)
{
        D = $M_R$
        for k = 1 to n
        begin
        for i = 1 to n
        begin
            for j = 1 to n
                $d_{ij}^{k} = d_{ij}^{k-1} \vee (d_{ik}^{k-1} \wedge d_{kj}^{k-1})$
        end
}
```

D = $[d_{ij}]$ is the transitive closure.

Topological Sorting

A topological sort is an ordering of vertices in a directed acyclic (graph with no cycle) graph (dag), such that if there is a path from v_i to v_j, then vi appears before v_j in the ordering. A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

Applications

Consider the course available at university as the vertices of a directed graph; where there is an edge from one course to another if the first is a prerequisite for the second.

A topological ordering is then a listing of all the courses such that all prerequisite for a course appear before it does.

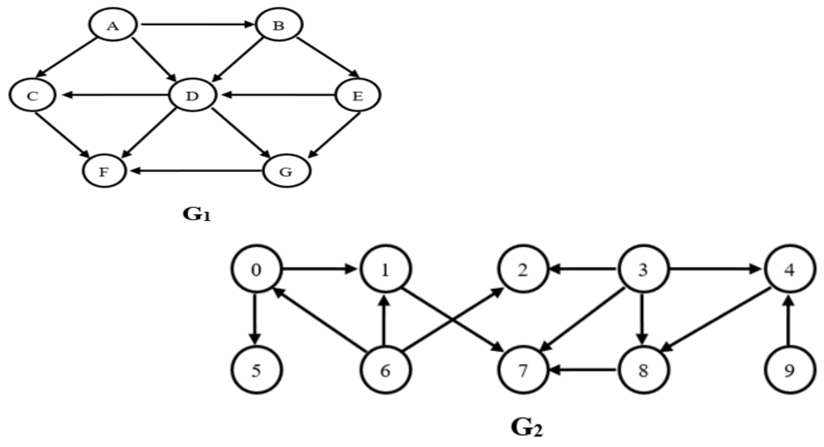


Figure 10.4: Directed graph with no directed cycle

Algorithms to find out topological ordering:

1. Breadth First Ordering
2. Depth First Ordering

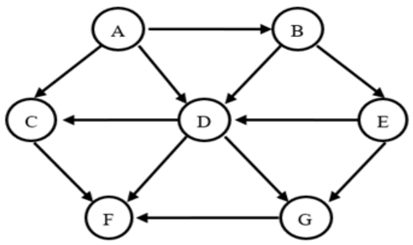
Breadth First Ordering

In breadth first topological ordering of a directed graph with no cycles, we start by finding the vertices that should be first in the topological order and then apply the fact that every vertex must come before its successors in the topological order. The vertices that come first are that are not successors of any other vertex.

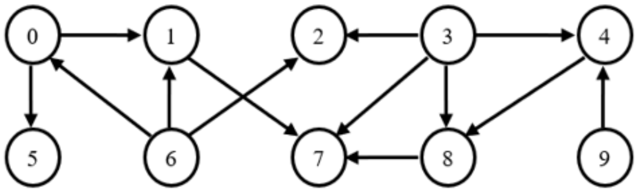
Determining the breadth first topological ordering:

1. Fill the information from the graph into the table (vertices, predecessors and predecessor count).
2. Find the predecessor count of each vertex v of graph G.
3. Insert all vertices with zero predecessor count in a queue.
4. Repeat steps 5 and 6 while queue is not empty.
5. Remove front vertex v from the queue and add it to breadth first topological order from left to right.
6. For each neighbour w of the vertex v,
    a. Decrease the predecessor count of w by 1.
    b. If predecessor count of w is zero, then add w to the rear of the queue.

Example: Determine the breadth first topological ordering for the following graphs G_1 and G_2.



G_1



G_2

Solution:

From the graph G_1 we get the following information:

Vertex	**Adjacent predecessor**	**Predecessor count**
A	-	~~0~~
B	A	~~1~~ 0
C	A, D	~~2~~ ~~1~~ 0
D	A, B, E	~~3~~ ~~2~~ ~~1~~ 0
E	B	~~1~~ 0
F	C, D, G	~~3~~ ~~2~~ ~~1~~ 0
G	D, E	~~2~~ ~~1~~ 0

The breadth first topological ordering gives:

A	B	E	D	C	G	F

From the graph G_2 we get following information:

Vertex	**Adjacent predecessor**	**Predecessor count**
0	6	~~1~~ 0
1	0, 6	~~2~~ ~~1~~ 0
2	3,6	~~2~~ ~~1~~ 0
3	-	0
4	3, 9	~~2~~ ~~1~~ 0
5	0	~~1~~ 0
6	-	0
7	1, 3, 8	~~3~~ ~~2~~ ~~1~~ 0
8	3, 4	~~2~~ ~~1~~ 0
9	-	0

The breadth first topological ordering gives:

3	6	9	0	2	4	1	5	8	7

Depth First Ordering

Start by finding a vertex that has no successors and place it last in the order.

After, we have, by recursion placed all the successors of a vertex into the topological order, then place the vertex itself in the order (before any of its successors).

Determining the depth first topological ordering:

1. Fill the information from the graph into the table (vertices, successors and successor count).
2. Find the successor count of each vertex v of graph G.
3. Push all vertices with zero successor count on a stack.
4. Repeat steps 5 and 6 while stack is not empty.
5. Pop the top vertex v from the stack and add it to depth first topological order from right to left.
6. For each neighbour w of the vertex v,
    a. Decrease the successor count of w by 1.
    b. If successor count of w is zero, then push w on top of the stack.

Example: Determine the depth first topological ordering for the above graphs G_1 and G_2.

Solution:

From the graph G_1, we get the following information:

Vertex	**Adjacent successor**	**Successor**
A	B, C, D	~~3~~ ~~2~~ ~~1~~ 0
B	D, E	~~2~~ ~~1~~ 0
C	F	~~1~~ 0
D	C,F, G	~~3~~ ~~2~~ ~~1~~ 0
E	D, G	~~2~~ ~~1~~ 0
F	-	0
G	F	~~1~~ 0

The depth first topological ordering gives:

A	B	E	D	C	G	**F**

From the graph G_2, we get the following information:

Vertex	**Adjacent successor vertices**	**Successor count**
0	1, 5	~~2~~ ~~1~~ 0
1	7	~~1~~ 0
2	-	0
3	2, 4, 7, 8	~~4~~ ~~3~~ ~~2~~ ~~1~~ 0
4	8	~~1~~ 0
5	-	0
6	0, 1, 2	~~3~~ ~~2~~ ~~1~~ 0
7	-	~~0~~
8	7	~~1~~ 0
9	4	~~1~~ 0

The depth first topological ordering gives:

3	6	2	0	5	1	9	4	8	**7**