Topological Sorting

Arnab Ganguly, Assistant Professor
Department of Computer Science, University of Wisconsin – Whitewater
Data Structures (CS 223)

Simulation Weblink

https://www.cs.usfca.edu/~galles/visualization/TopoSortIndegree.html

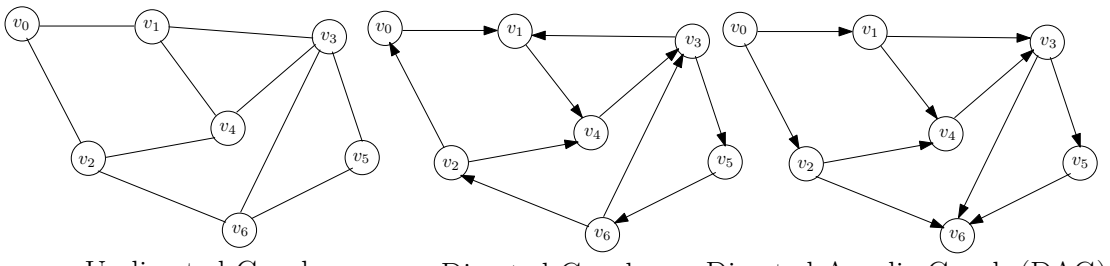
1 Graph Types

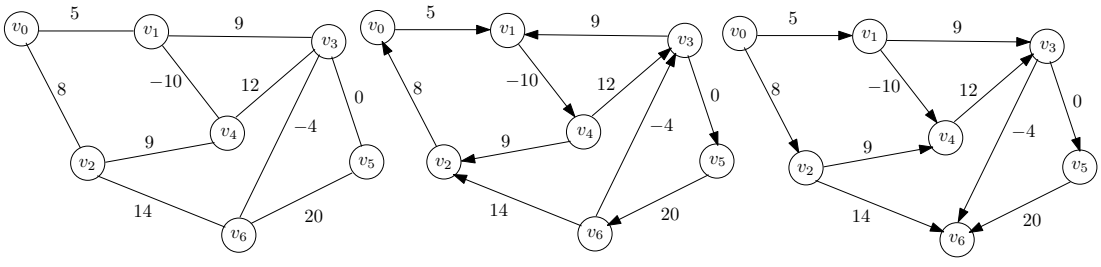
A graph $\mathcal{G}$ is a pair (V, E), where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. Thus, an edge connects exactly two vertices. We will always assume – (i) the two endpoints of an edge are unique, and (ii) there is a unique edge between two vertices.

A path in a graph is a sequence of vertices $v_1, v_2, \ldots, v_k$, where each consecutive pair of vertices v_i and v_{i+1} have an edge between them. **A cycle in a graph** is a sequence of vertices $v_1, v_2, \ldots, v_k$, where each consecutive pair of vertices v_i and v_{i+1} have an edge between them and $v_1 = v_k$. We will assume that a path is simple, i.e., it does not contain a cycle.

So far we have defined an **undirected graph**. In a **directed graph**, each edge has a unique direction. A **(directed) path** in a directed graph is a sequence of vertices $v_1, v_2, \cdots, v_k$, such that there is a directed edge from v_i to v_{i+1}. Accordingly, we can define a directed cycle. A **directed acyclic graph** (DAG) is a directed graph that does not contain any directed cycle. Lastly, in **weighted graphs**, each edge is associated with a real number called weight.

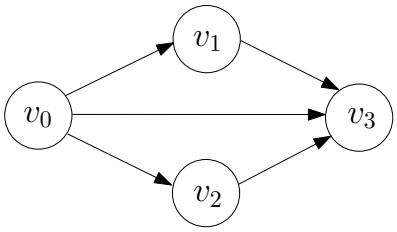
See figure below for illustration.





2 Topological Order of a DAG

Given a DAG with n vertices $v_1, v_2, \ldots, v_n$, let $P = v'_1, v'_2, \ldots, v'_n$ be an ordering of the vertices, i.e., $v_i = v'_j$ for some $i, j \in [1, n]$. Then, P is a topological ordering if and only if for two vertices v'_i and v'_j, $i < j$, there does not exist a (directed) edge from v'_j to v'_i.



Thus for the graph in the adjoining figure, both $\langle v_0, v_1, v_2, v_3 \rangle$ and $\langle v_0, v_2, v_1, v_3 \rangle$ are valid topological orders. However, $\langle v_0, v_1, v_3, v_2 \rangle$ is not a valid topological order because there is a directed edge from v_2 to v_3.

An important thing to note is that topological order is **not unique**, i.e., a DAG can have multiple topological orders. Next, our objective is to find a topological ordering of a graph.

3 Topological Sorting Algorithm

The main intuition is that since v_0 in the figure above does not have any incoming edge, it has to be the first vertex in topological order. Now, remove v_0 and all its outgoing edges, following which we will get another graph with some vertices having no incoming edge. Pick any such vertex and repeat the process, until there are no more vertices to process.

Deletion of vertices and their edges, however, is not straightforward. So we are going to simulate the deletion process, using an array, called *inDegree*. The following is a formal description.

Algorithm

- Create an array $inDegree[\,]$ of length the number of vertices in the graph
- Compute the *inDegree* of every vertex in the DAG. The indegree of a node is the number of incoming edges onto the node.
- Start by enqueueing all vertices v with $inDegree[v] = 0$ into a queue.
- Repeat the following steps until the queue is empty:
  - let v be the vertex obtained by dequeueing the queue
  - output v as the next vertex in topological order
  - for every outgoing edge e of v, do the following:
    - let w be the destination of e, i.e., w is the vertex to which e points at
    - decrement $inDegree[w]$ by 1
    - if $inDegree[w] = 0$, then enqueue w

Remark: The algorithm can also be implemented using a stack, i.e., the queue replaced with a stack. The rest remaining the same, we may get a different topological order.

3.1 Simulation

We apply the above algorithm on the following DAG. See table on next page for simulation.

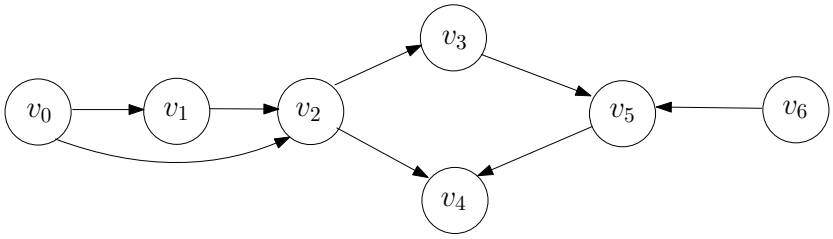


Table 1: Illustration of Topological Sorting

	inDegree array							**queue**	**Topological Order**
	v_0	v_1	v_2	v_3	v_4	v_5	v_6		
At Start:	0	1	2	1	2	2	0	$[v_0, v_6]$	
Dequeue (Relax v_0)	0	0	1	1	2	2	0	$[v_6, v_1]$	v_0
Dequeue (Relax v_6)	0	0	1	1	2	1	0	$[v_1]$	v_0, v_6
Dequeue (Relax v_1)	0	0	0	1	2	1	0	$[v_2]$	v_0, v_6, v_1
Dequeue (Relax v_2)	0	0	0	0	1	1	0	$[v_3]$	v_0, v_6, v_1, v_2
Dequeue (Relax v_3)	0	0	0	0	1	0	0	$[v_5]$	v_0, v_6, v_1, v_2, v_3
Dequeue (Relax v_5)	0	0	0	0	0	0	0	$[v_4]$	$v_0, v_6, v_1, v_2, v_3, v_5$
Dequeue (Relax v_4)	0	0	0	0	0	0	0	[ ]	$v_0, v_6, v_1, v_2, v_3, v_5, v_4$

3.2 Complexity

Note that an edge or a vertex is processed only a constant number of times. Hence the complexity on a graph with N vertices and M edges is $O(N + M)$.

4 Testing Acyclicity

Topological sorting can be used to test whether a graph $\mathcal{G}$ is acyclic or not, as follows:

Testing Acyclicity

- We execute the topological sorting algorithm on $\mathcal{G}$.
- If at any point, the **queue is empty**, yet **there exists an unrelaxed edge** (i.e., all vertices have not been outputted in topo-order), we know that the graph contains a cycle. In other words, **if we have a vertex with a non-zero entry in the** *inDegree* **array at the end**, then the graph contains a cycle.
- If you get a complete topological order, then of course the graph is acyclic.