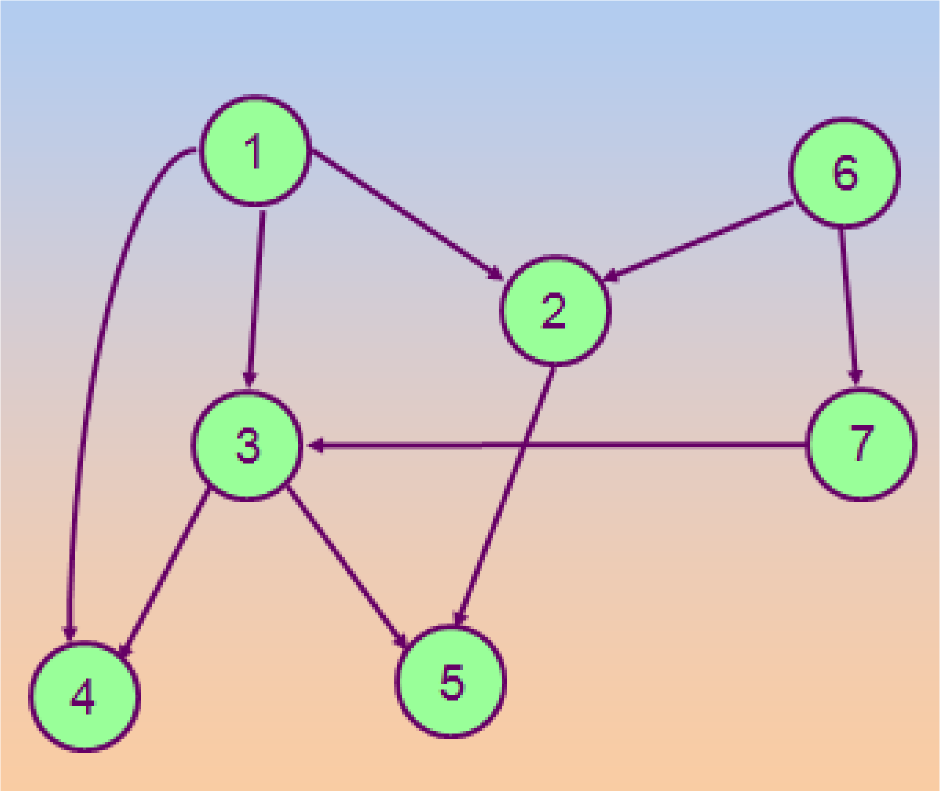
Topological Sort Algorithm for Directed Acyclic Graph

WHAT IS DIRECTED ACYCLIC GRAPH (DAG)?

A Graph:



TOPOLOGICAL SORT

Topological sort of a DAG:

- Linear ordering of all vertices in graph G such that vertex *u* comes before vertex *v* if there is an edge $(u, v) \in G$.

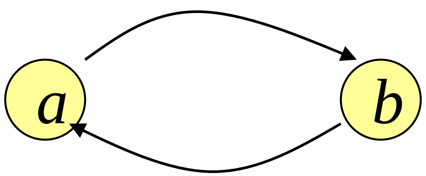
It is important to note that if the graph is not acyclic, then no linear ordering is possible. That is, we must not have circularities in the directed graph. For example, in order to get a job you need to have work experience, but in order to get work experience you need to have a job.

EXISTENCE OF TOPOLOGICAL SORT

Lemma
A directed graph *G* is acyclic iff a DFS of G yields no back edges.

Lemma
G can be topologically sorted iff it has no cycle, that is, iff it is a ***dag*** (directed acyclic graph).

Proof $\Rightarrow$ If *G* has a cycle, then it cannot be topologically sorted.



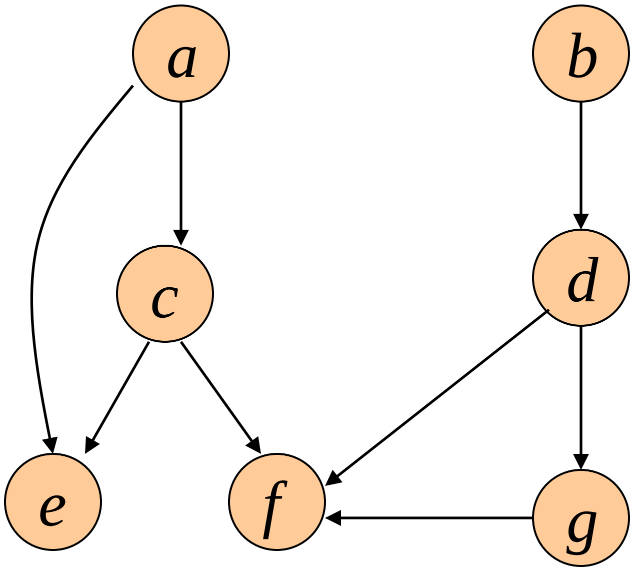
$\Leftarrow$ If *G* has no cycle, then it can be topologically sorted.

TOPOLOGICAL SORT

- Each node represents an activity; e.g., taking a class.
- $(u, v) \in E(G)$ implies activity u must be scheduled before activity v.
- Topological sort schedules all activities.
- More than one schedule may exist.

TOPOLOGICAL SORT OF DIGRAPHS

Different orderings in Topological Sort Algorithm are as follows



Some topological sorts:

1. *a, c, e, b, d, g, f*
2. *a, b, c, d, g, f, e*
3. *b, d, g, a, c, f, e*

TOPOLOGICAL SORT - APPLICATIONS

- Scheduling a dependent graph.
- Find a feasible course plan for university studies with Course prerequisites.
- Job scheduling (car manufacturing etc.)
- Etc…

TOPOLOGICAL SORT - ALGORITHM

Performed on a DAG.

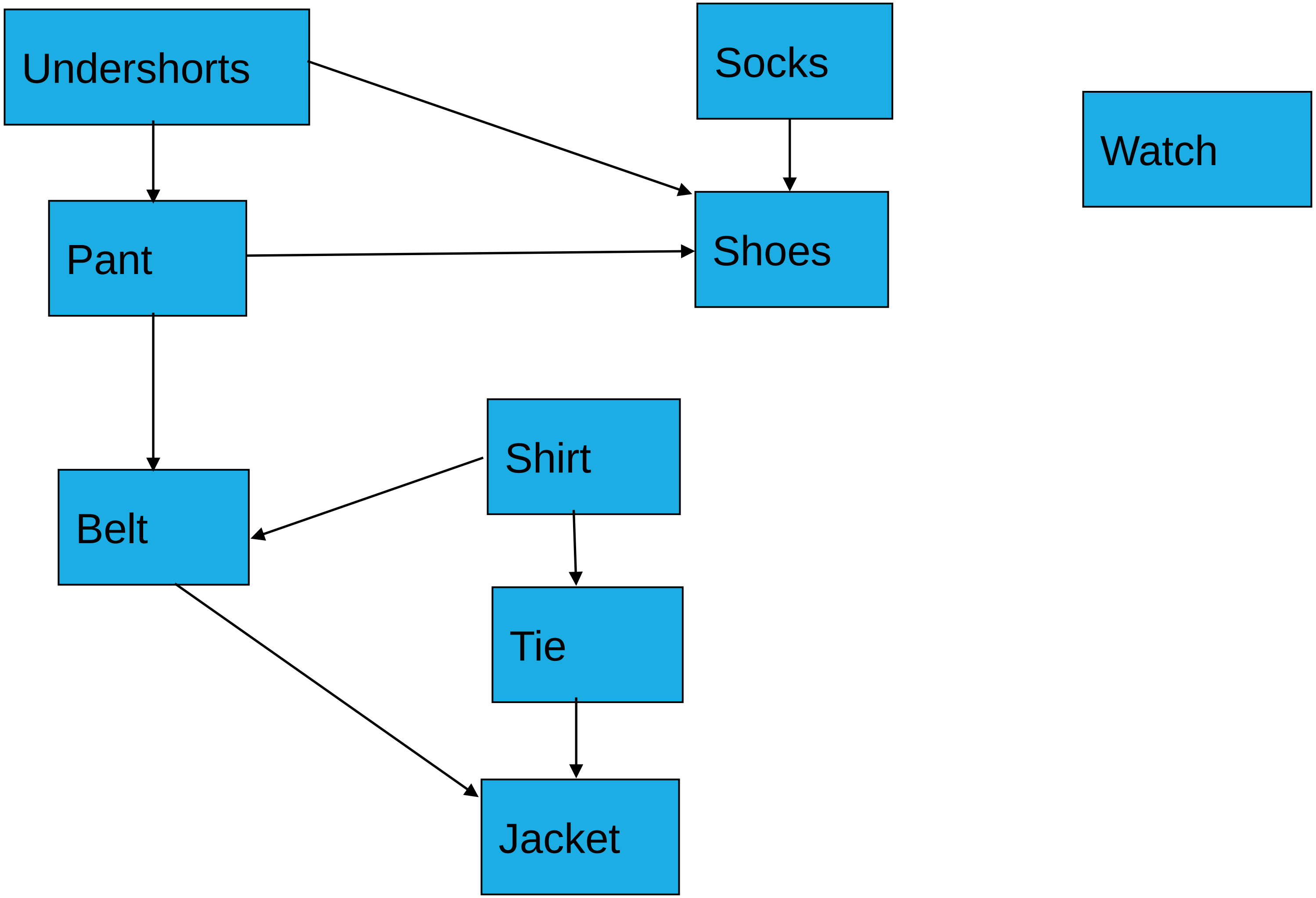
Linear ordering of the vertices of G such that if

$(u, v) \in E$, then u appears somewhere before v.

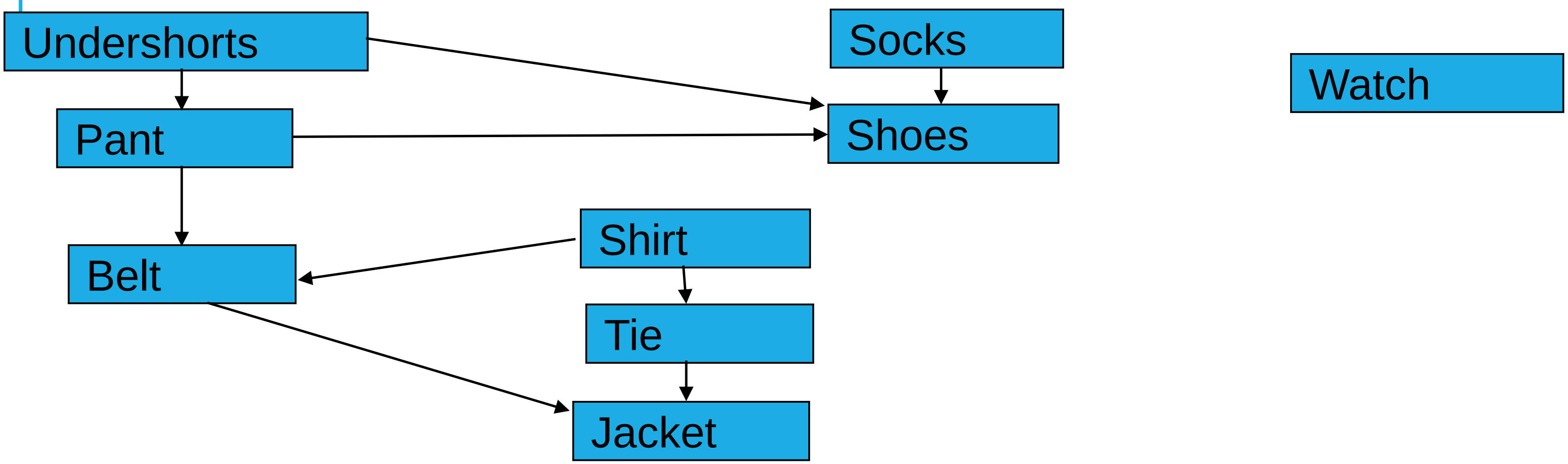
Topological-Sort (G)

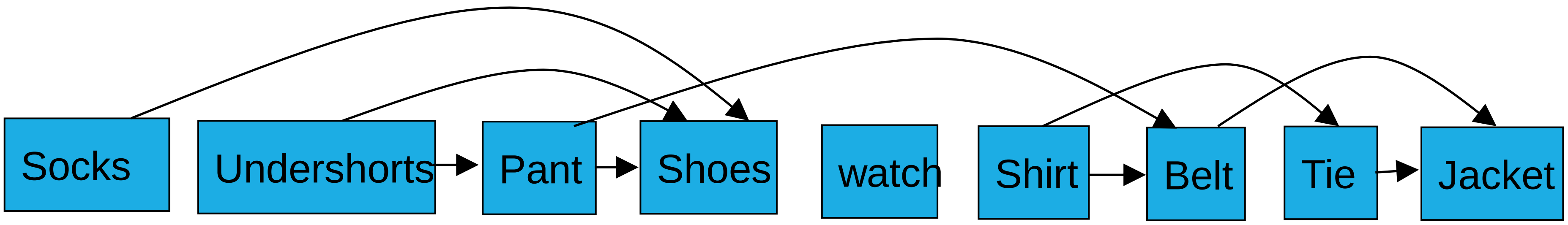
1. call DFS(G) to compute finishing times $f[v]$ for all $v \in V$
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices

EXAMPLE 1 – GETTING DRESSED FOR OFFICE

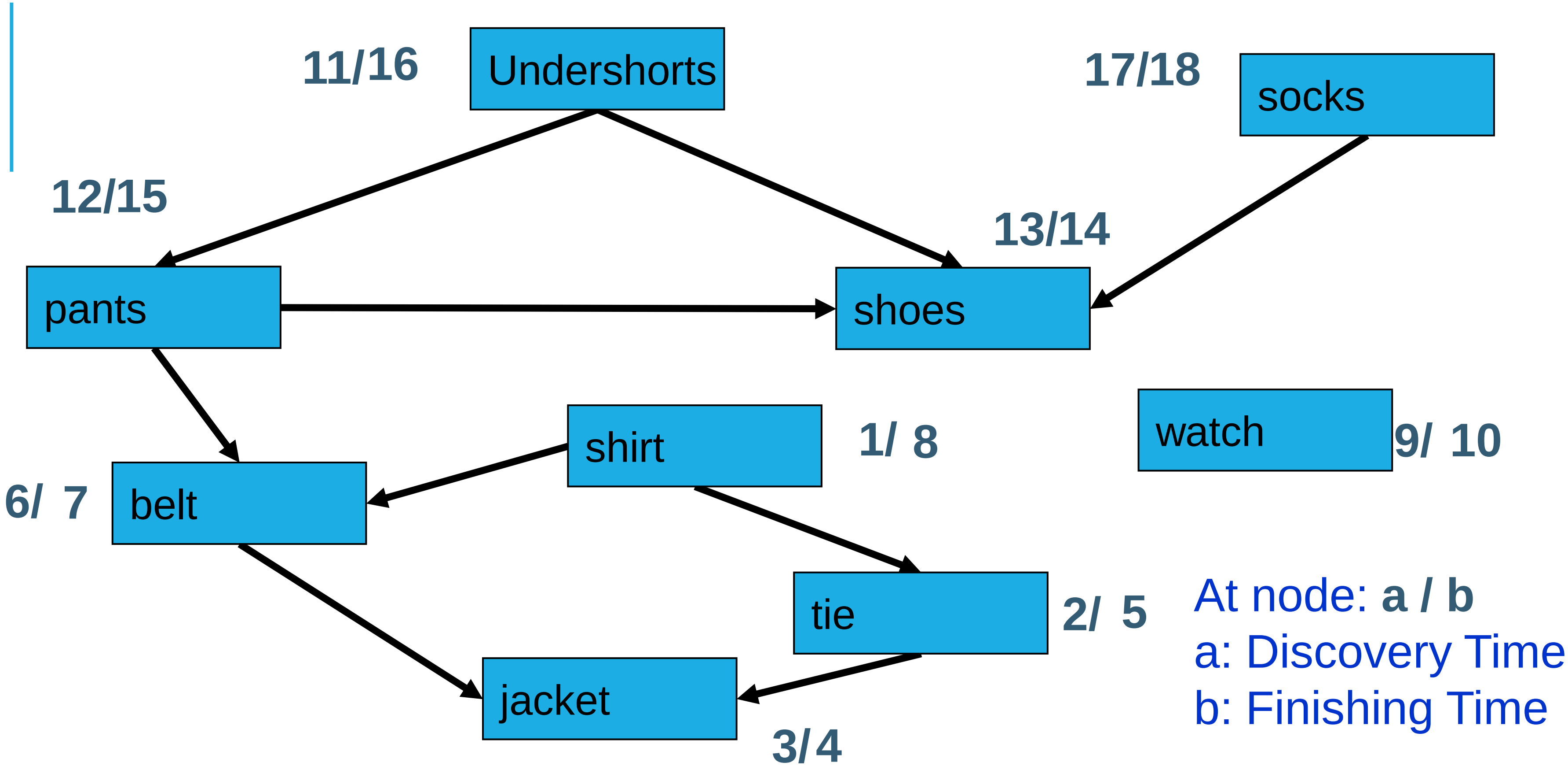


EXAMPLE (CONT.)

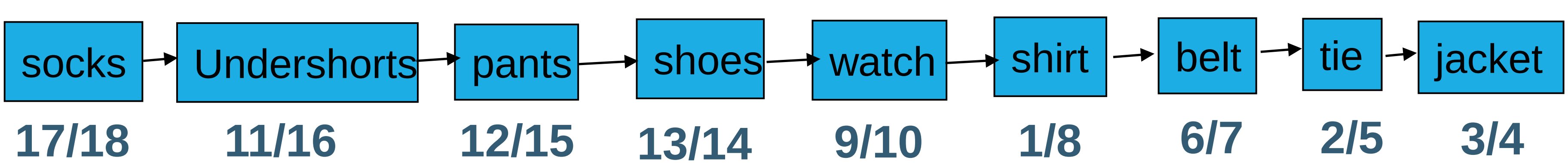




Example – Getting Dressed for Office



Linked List :-

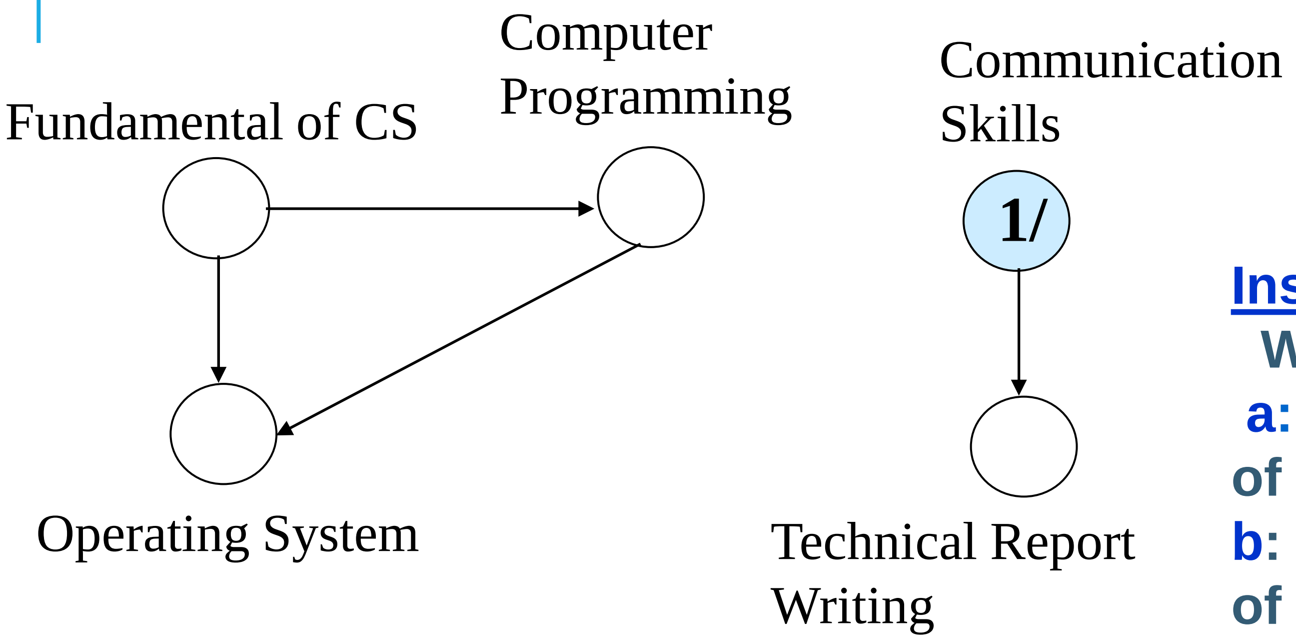


ANOTHER EXAMPLE

The CS dept course prerequisites can be represented as a directed acyclic graph (DAG).

- It must be directed because one course is the prerequisite for another (and not vice versa).
- There can't be any cycles because then it would be impossible to meet all the prerequisites.

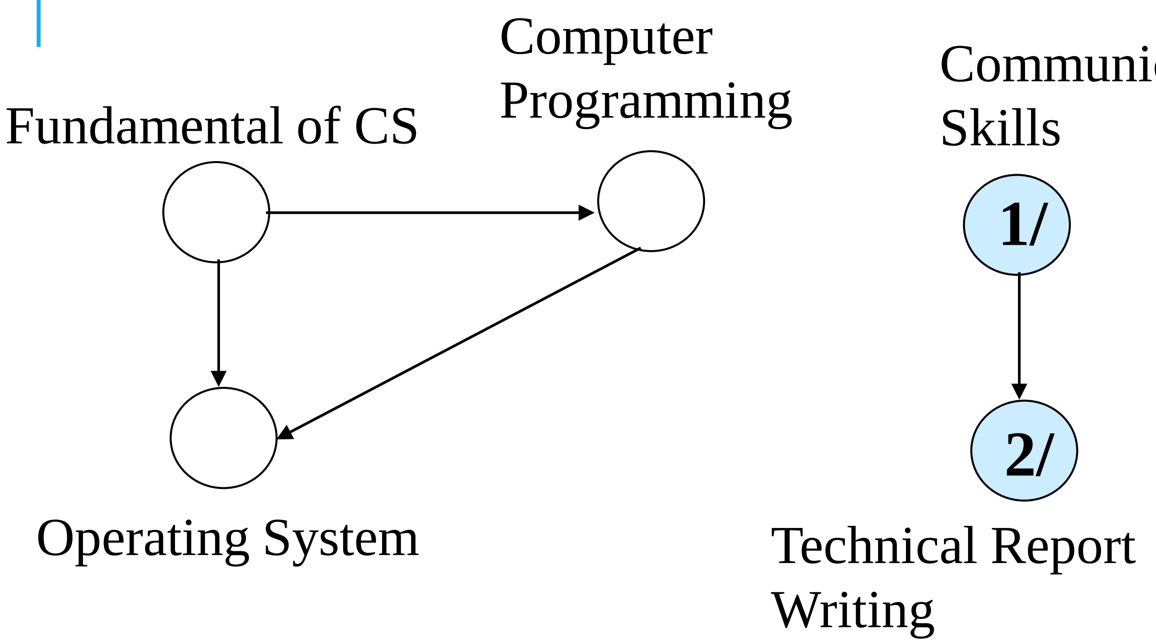
EXAMPLE 2- COURSE PREREQUISITE PLAN AT UNIVERSITY



Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

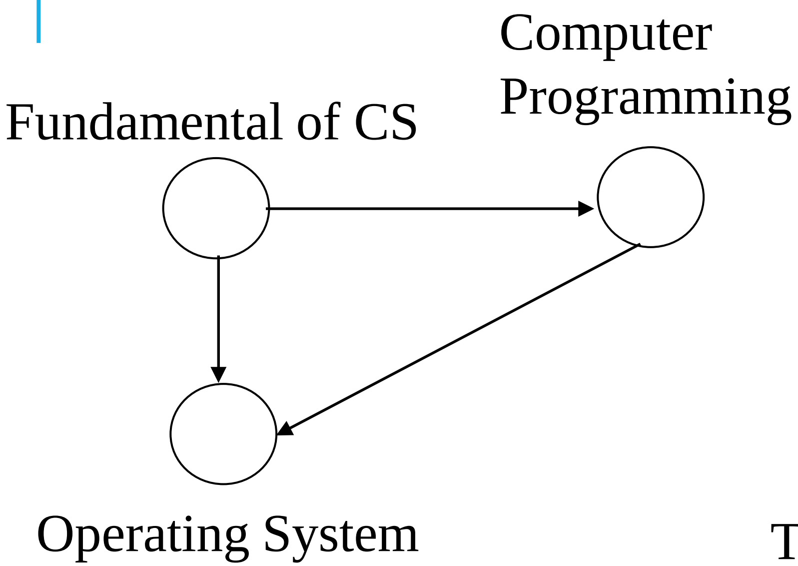
EXAMPLE - 2

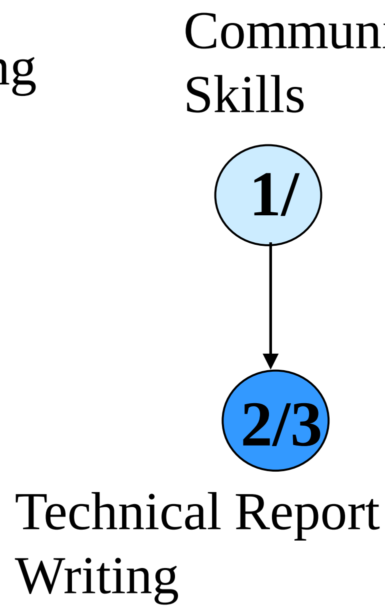


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

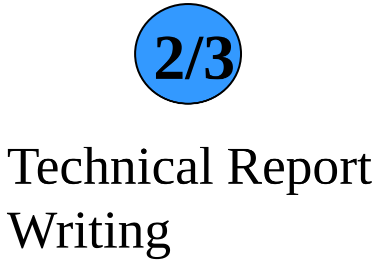
EXAMPLE - 2



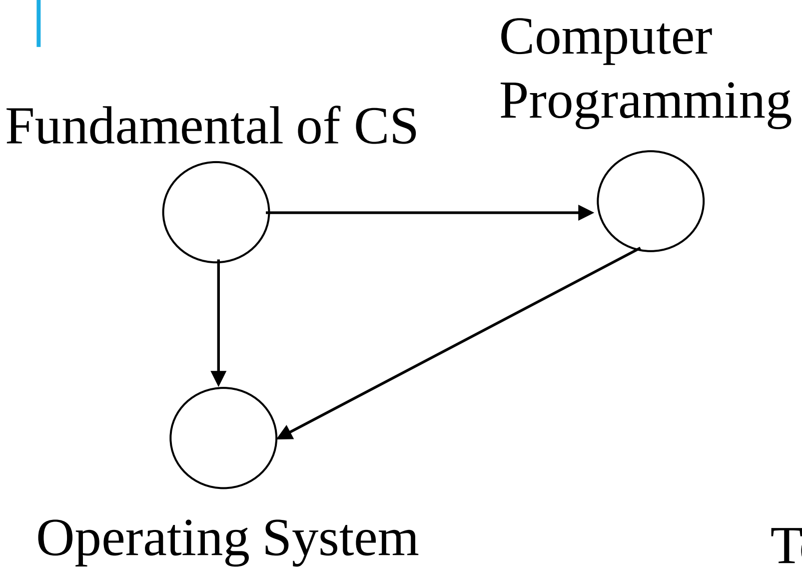


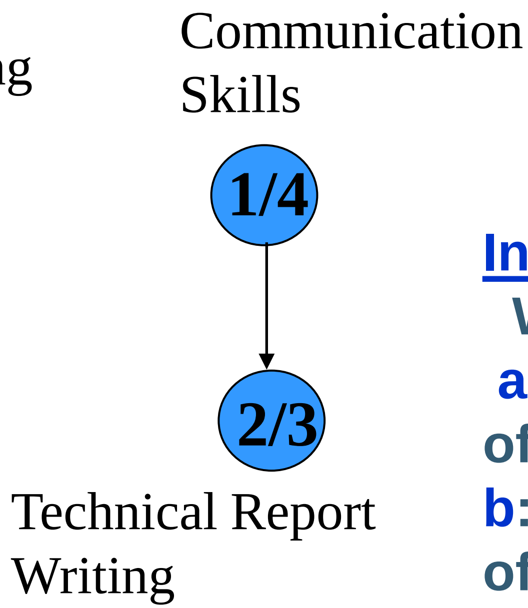
Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:



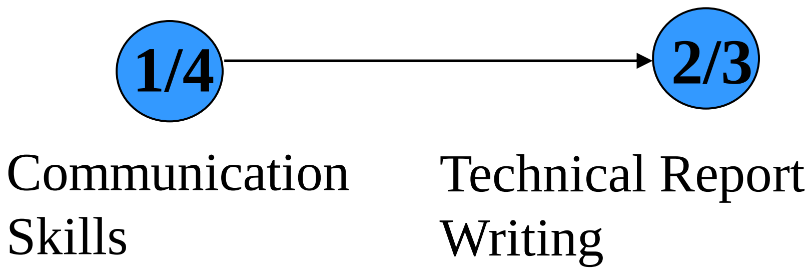
EXAMPLE - 2



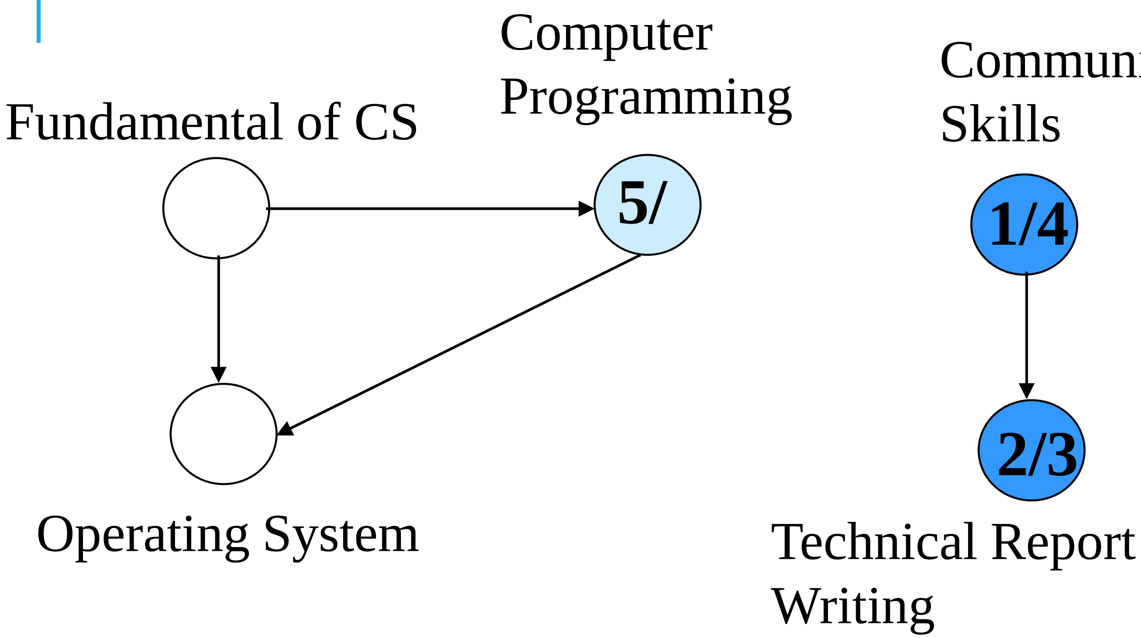


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

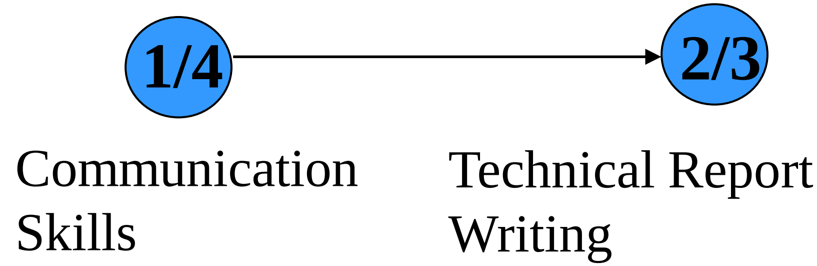


Example - 2

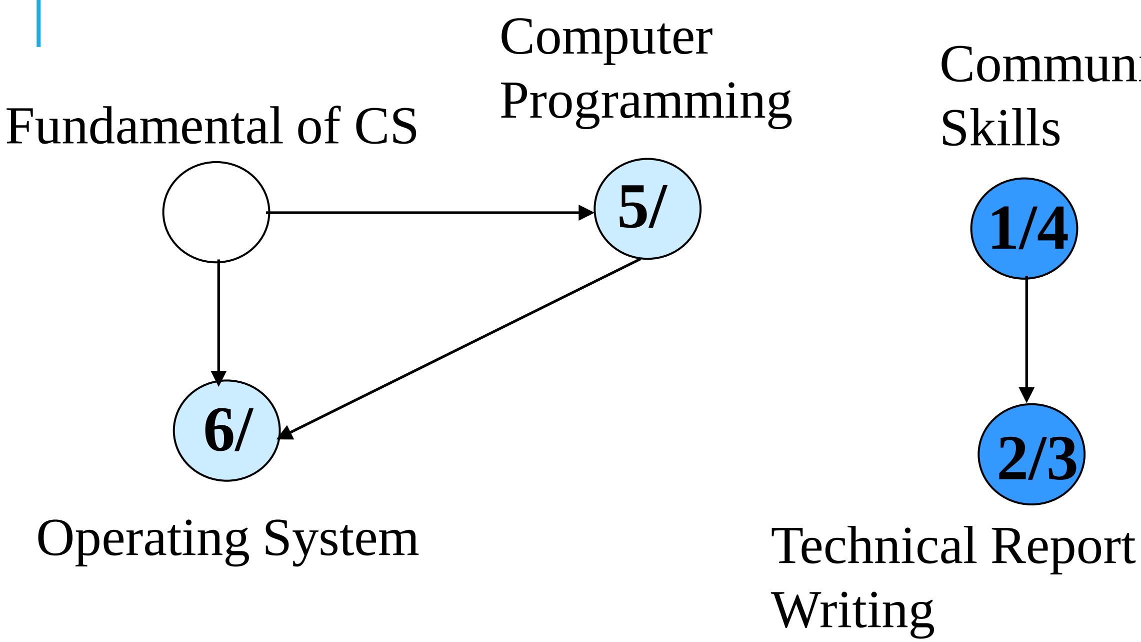


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

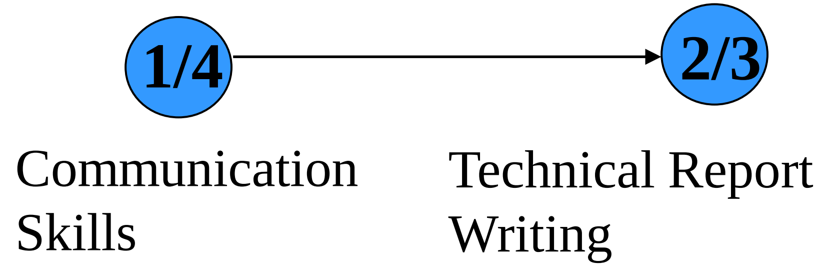


Example - 2

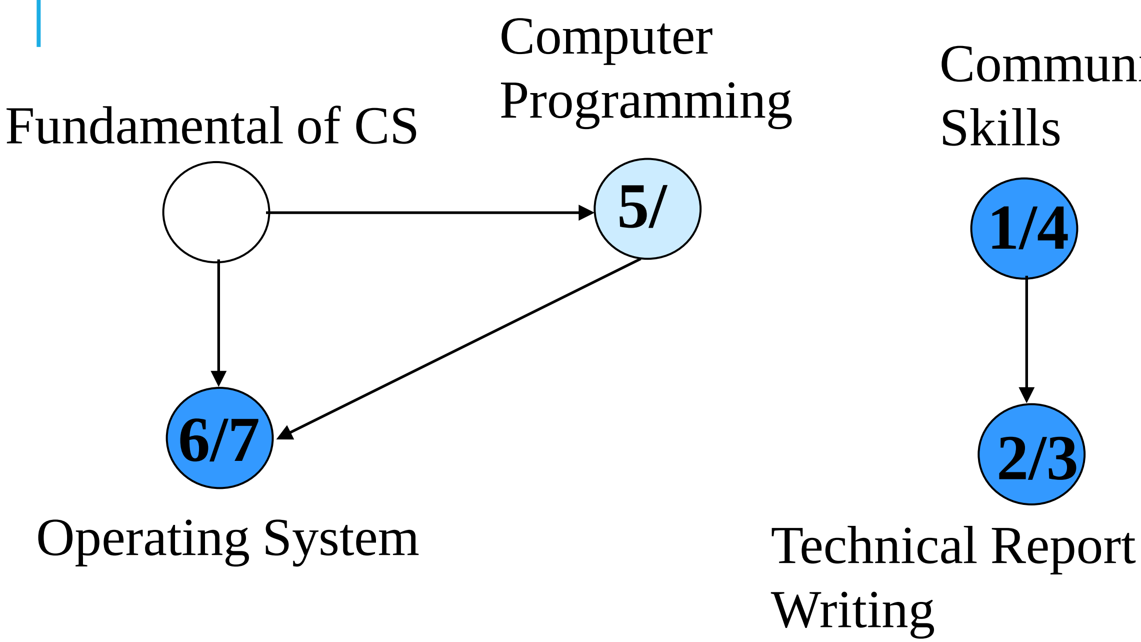


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

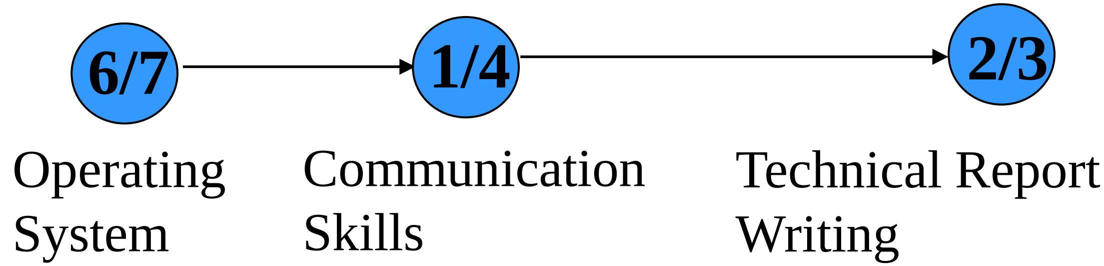


Example - 2

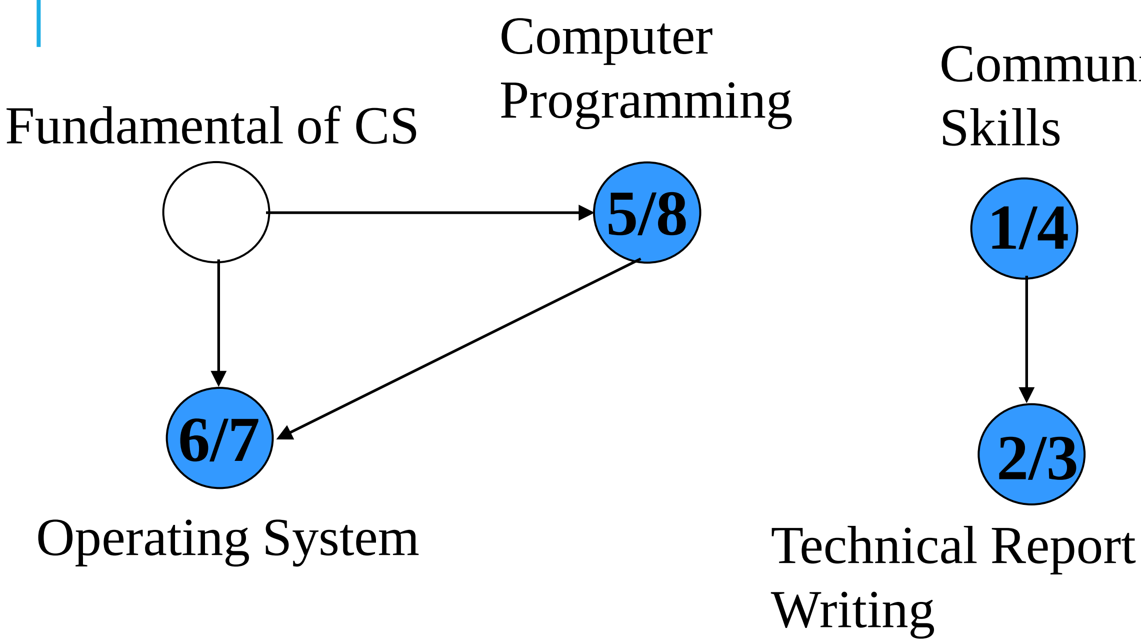


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

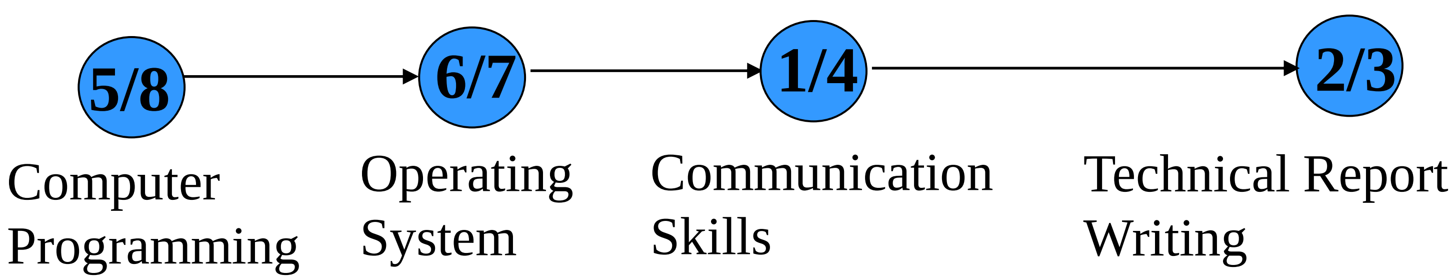


EXAMPLE - 2

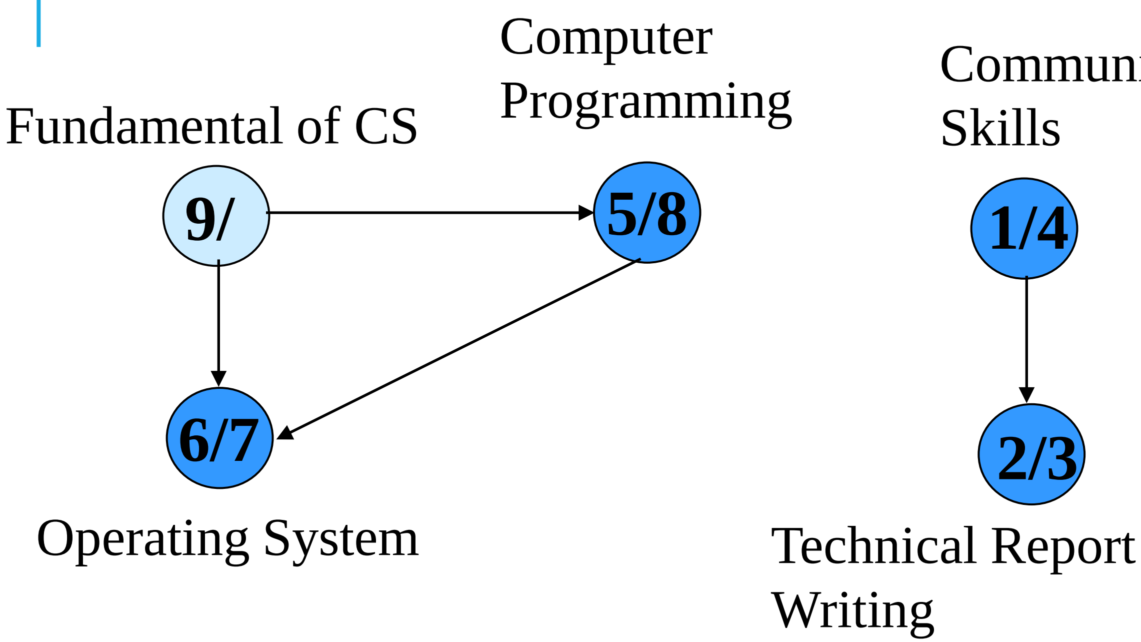


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

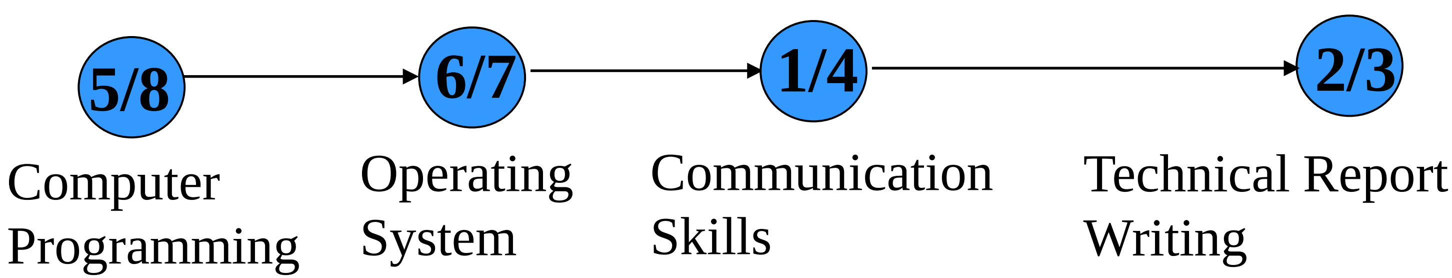


Example - 2

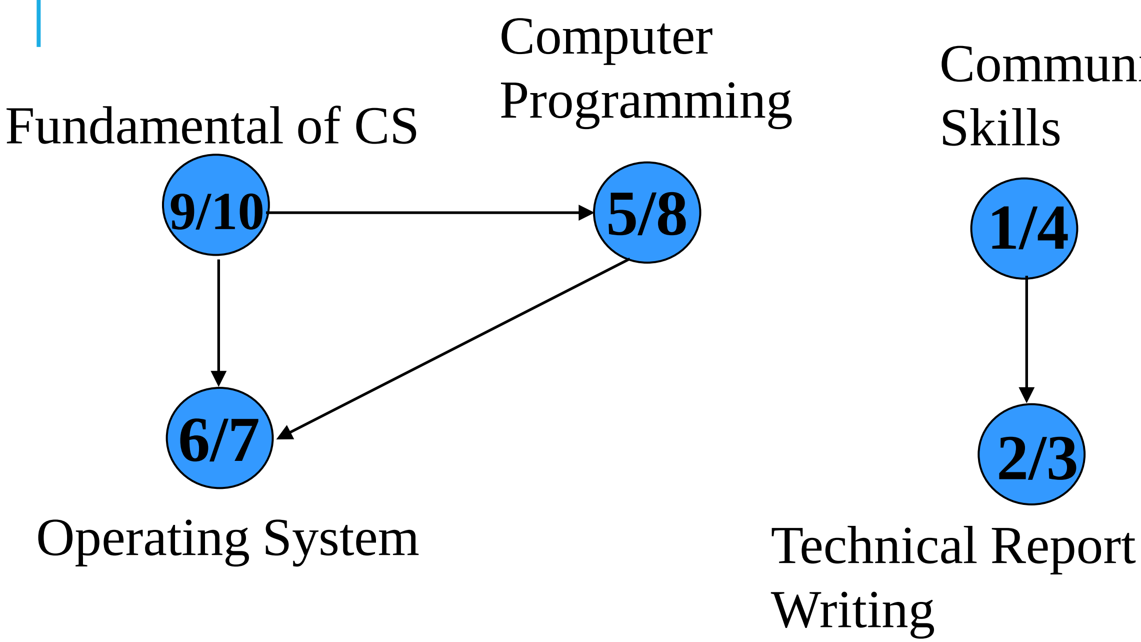


Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:

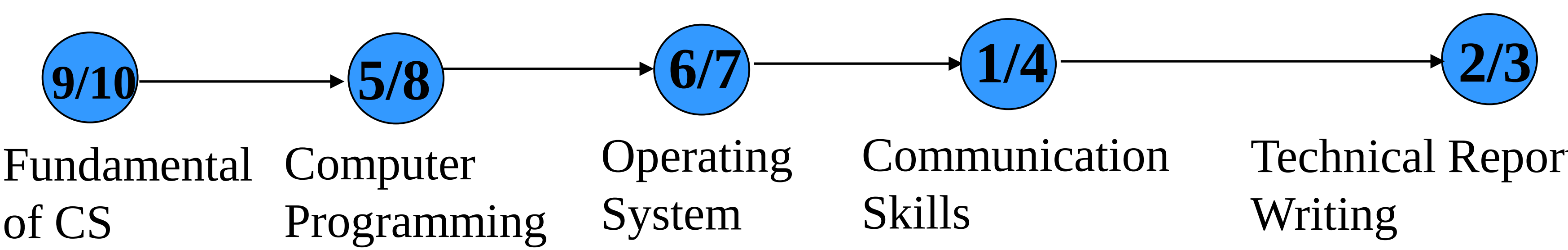


Example - 2



Inside Node: a/b
Where
a: Discovery Time of Node and
b: Finishing Time of Node

Linked List:



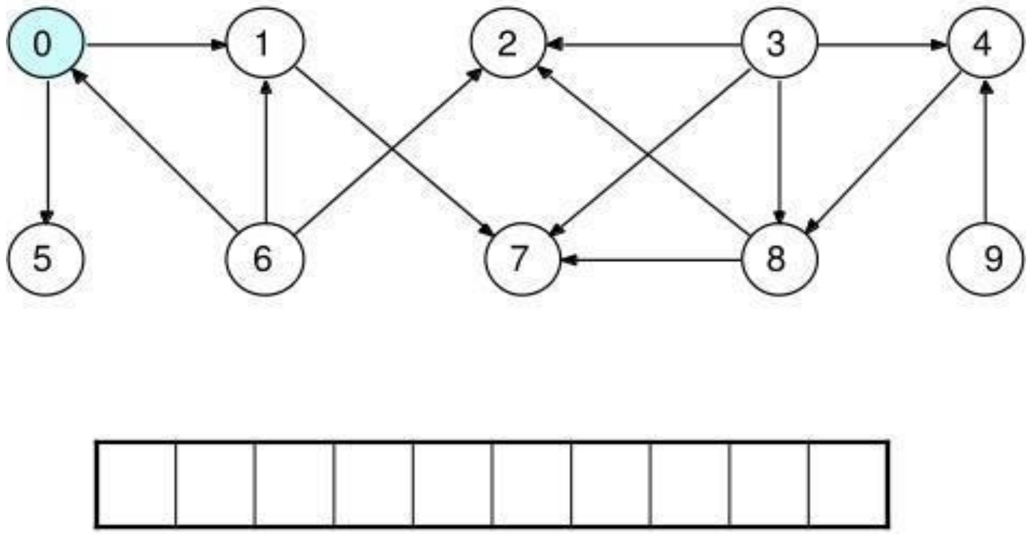
TIME COMPLEXITY

It takes O(1) time to insert each of the |V| vertices onto the front of the linked list.

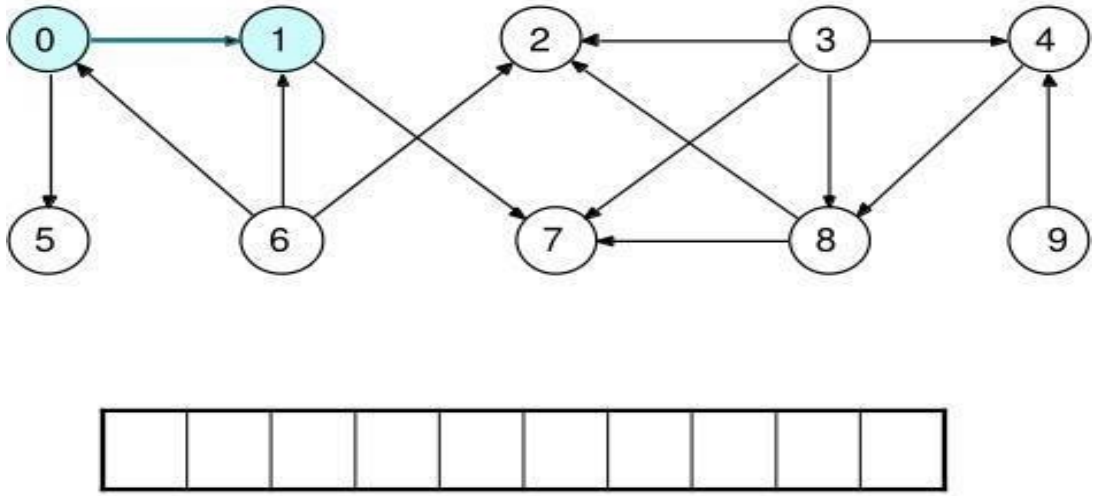
Total running time of topological sort is θ(V+E) . Since DFS(G) search takes θ(V+E) time

Correctness: need to prove that (*u*,*v*) in G → f[u]>f[v]

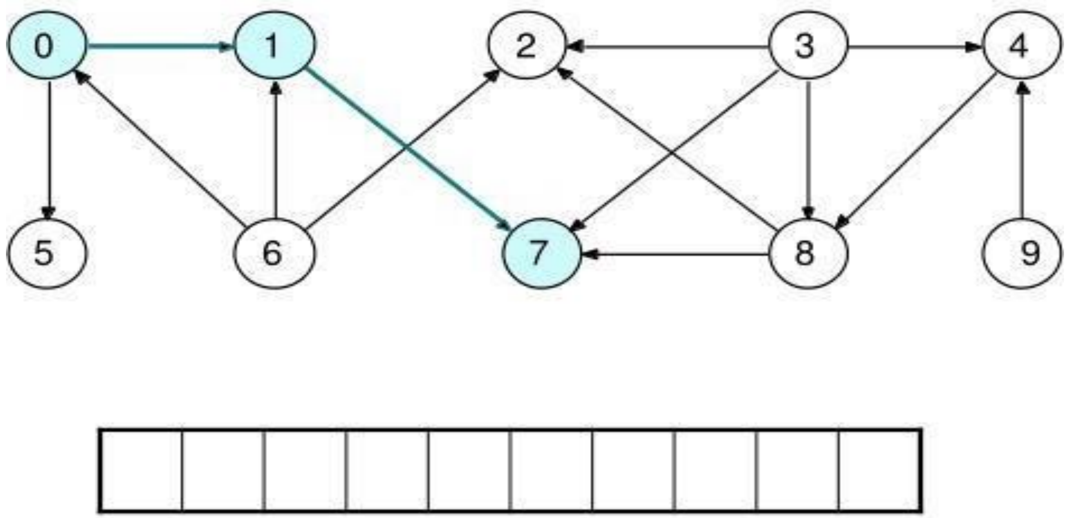
TOPOLOGICAL SORT ALGORITHM: EXAMPLE



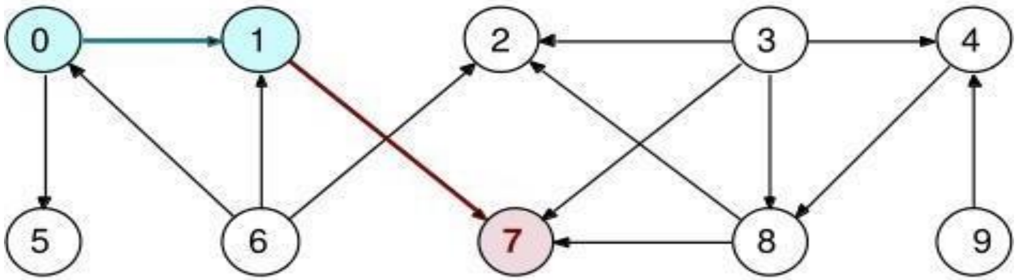
TOPOLOGICAL SORT ALGORITHM: EXAMPLE

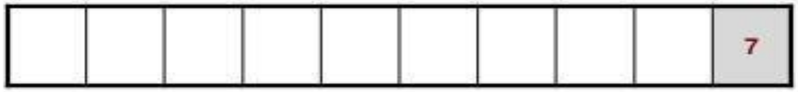


TOPOLOGICAL SORT ALGORITHM: EXAMPLE

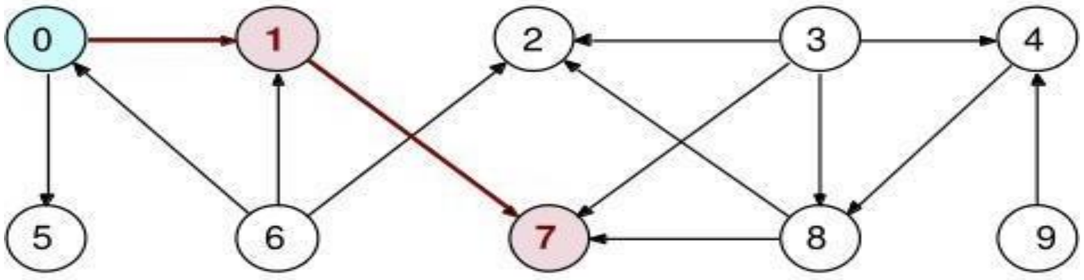


TOPOLOGICAL SORT ALGORITHM: EXAMPLE



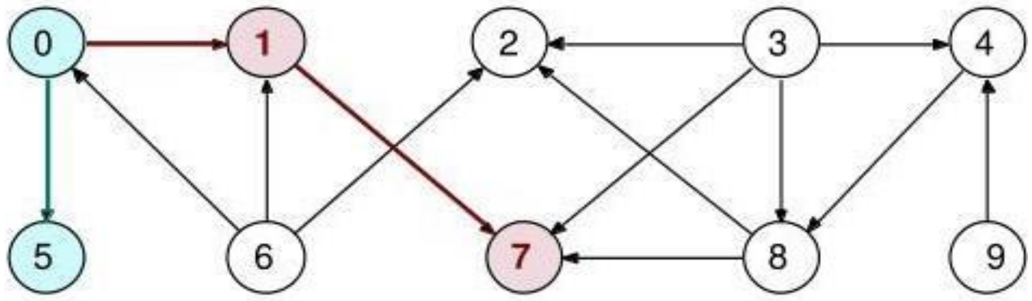


TOPOLOGICAL SORT ALGORITHM: EXAMPLE



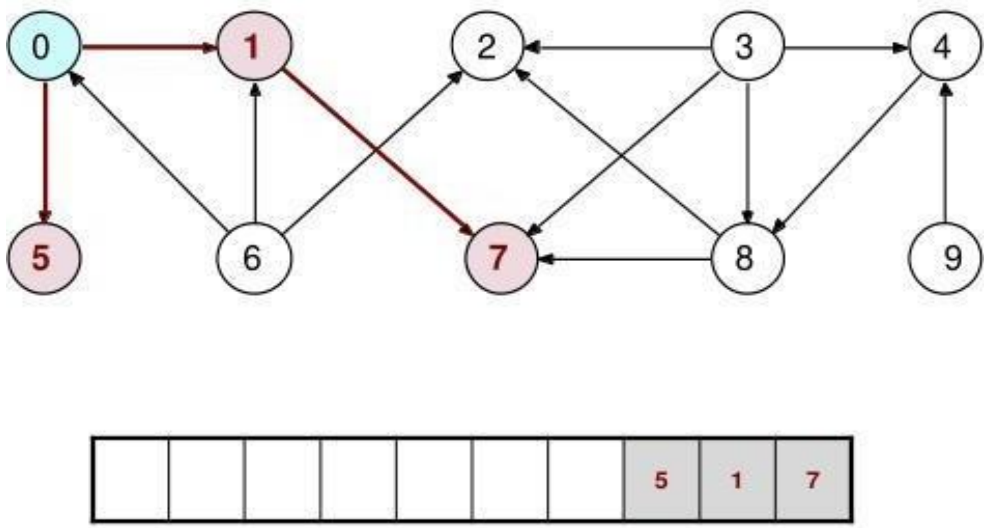
								1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE

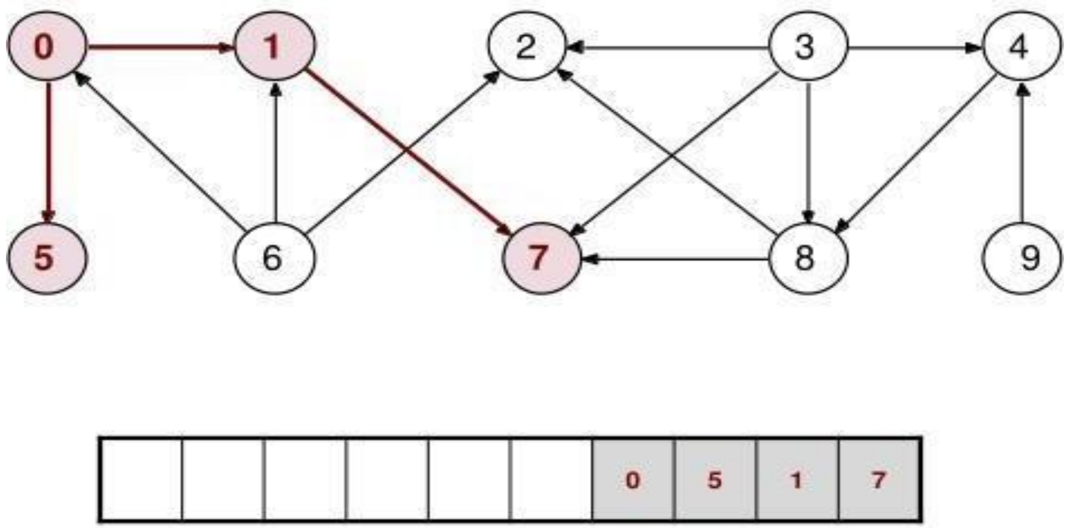


								1	7

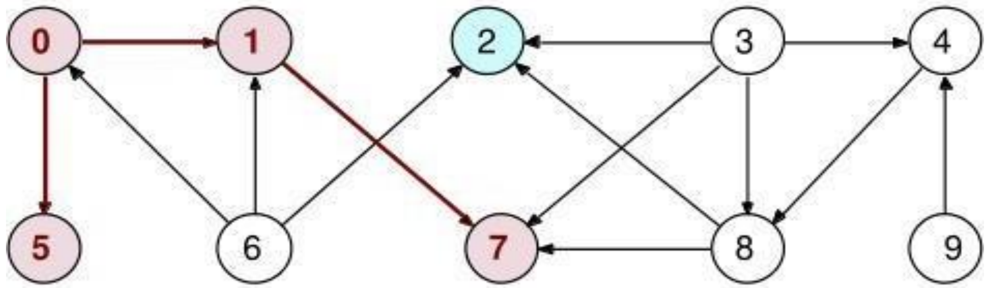
TOPOLOGICAL SORT ALGORITHM: EXAMPLE



TOPOLOGICAL SORT ALGORITHM: EXAMPLE

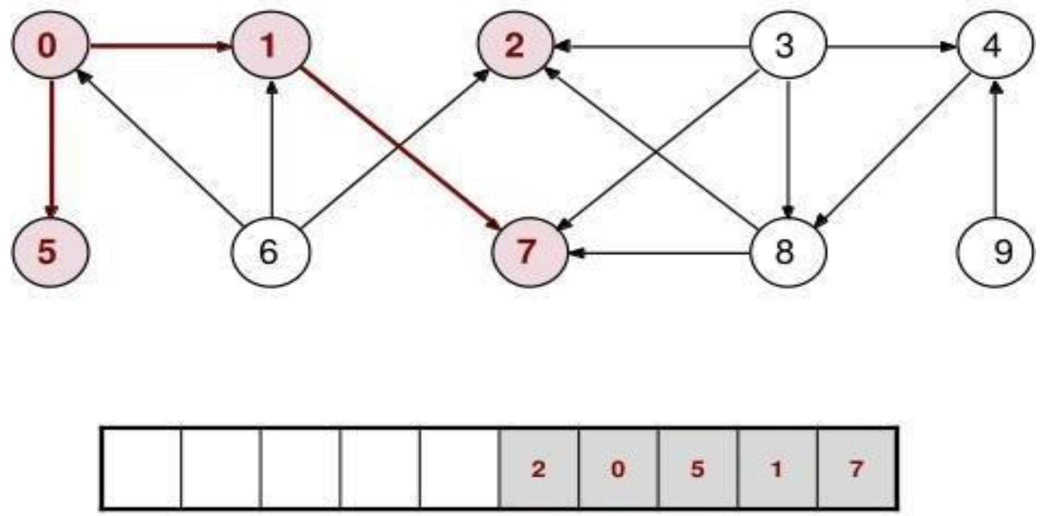


TOPOLOGICAL SORT ALGORITHM: EXAMPLE

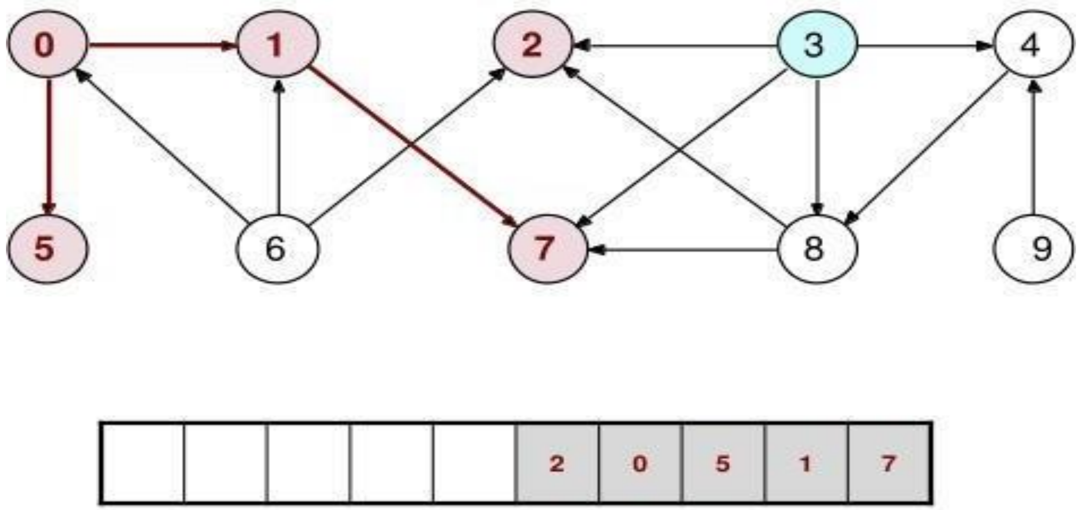


						0	5	1	7

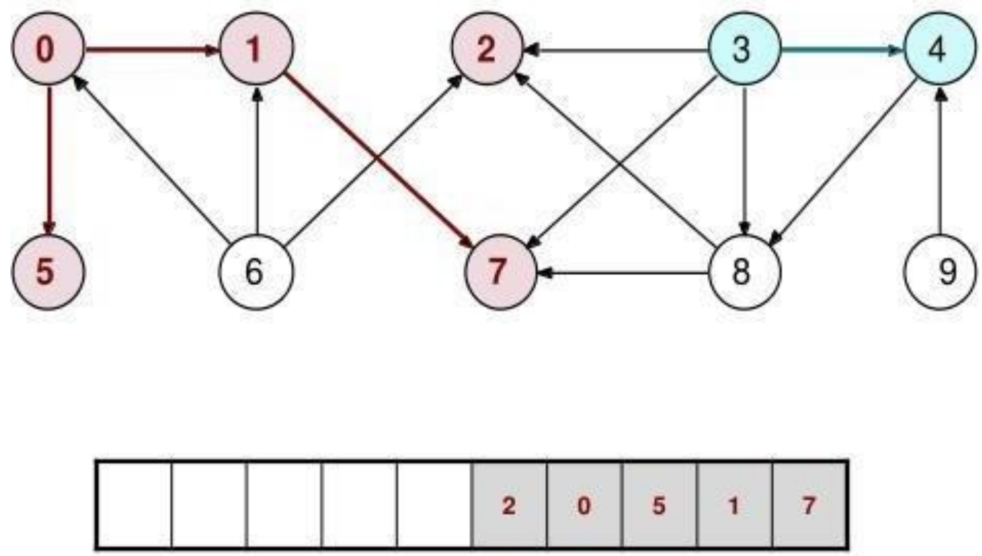
TOPOLOGICAL SORT ALGORITHM: EXAMPLE



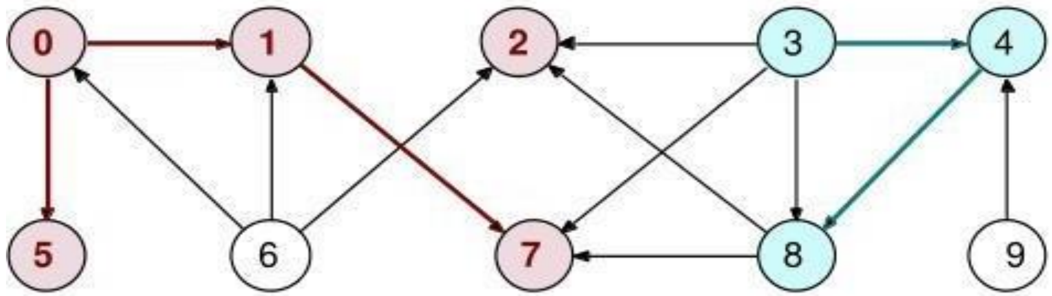
TOPOLOGICAL SORT ALGORITHM: EXAMPLE



TOPOLOGICAL SORT ALGORITHM: EXAMPLE

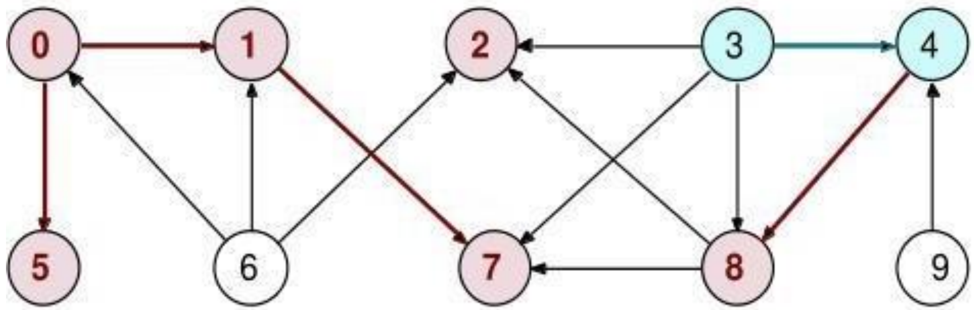


TOPOLOGICAL SORT ALGORITHM: EXAMPLE



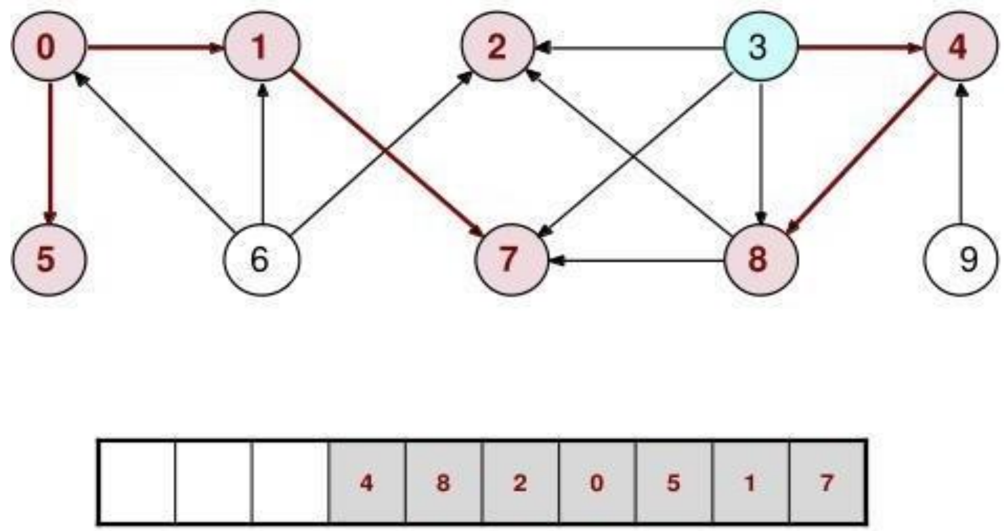
					2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE

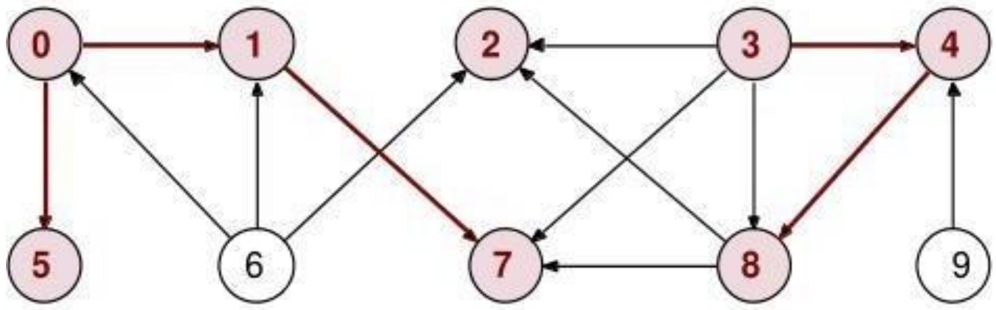


				8	2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE

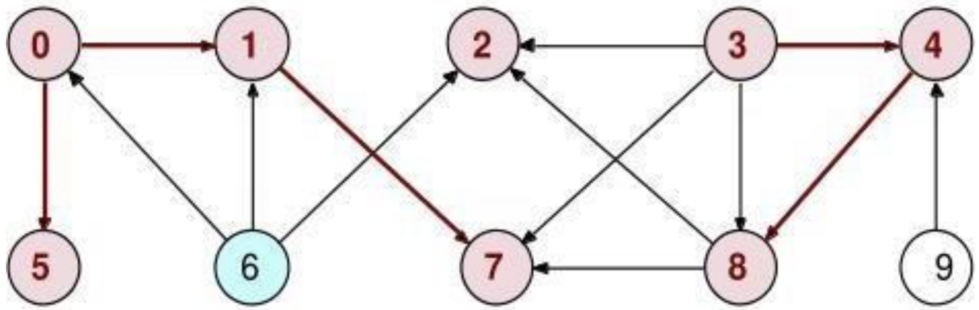


TOPOLOGICAL SORT ALGORITHM: EXAMPLE



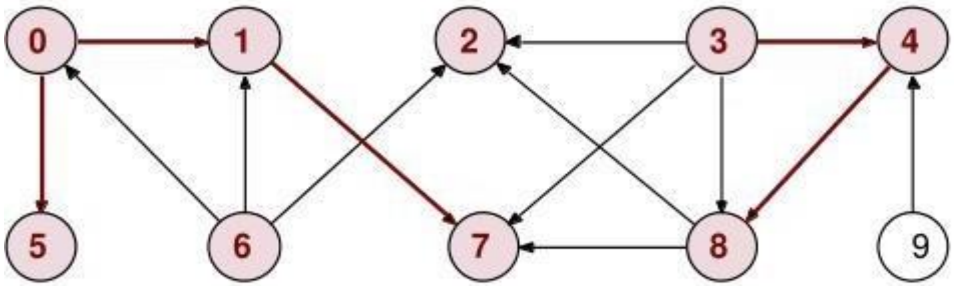
		3	4	8	2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE



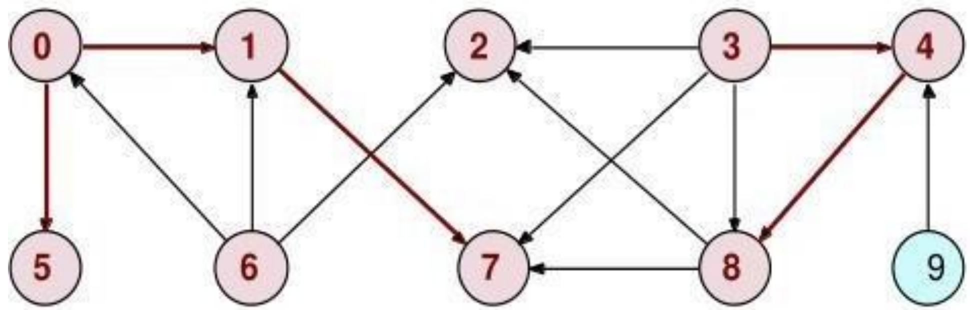
		3	4	8	2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE



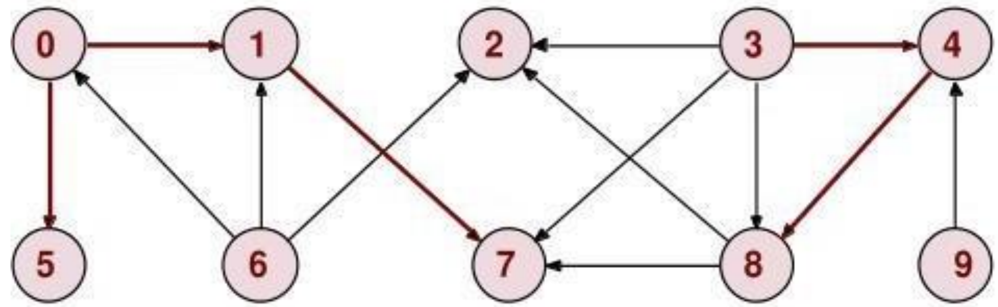
	6	3	4	8	2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE



	6	3	4	8	2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE



9	6	3	4	8	2	0	5	1	7

TOPOLOGICAL SORT ALGORITHM: EXAMPLE