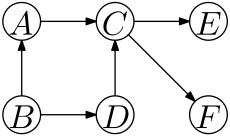
Directed Graphs

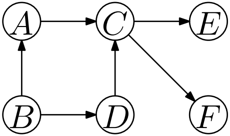
Directed Graphs

Assume that $G = (V, E)$ is directed **and acyclic**.



Directed Graphs

Assume that $G = (V, E)$ is directed **and acyclic**.

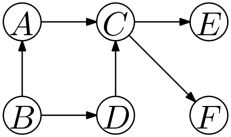


Topological Sorting (or *topological ordering*): number the vertices $1, 2, \ldots, n$ such that for each edge (u, v),

$$\#(u) < \#(v).$$

Directed Graphs

Assume that $G = (V, E)$ is directed **and acyclic**.



Topological Sorting (or *topological ordering*): number the vertices $1, 2, \ldots, n$ such that for each edge (u, v),

$$\#(u) < \#(v).$$

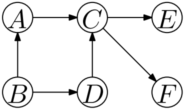
If G is cyclic, this is not possible. Do you see why?
How to compute such a numbering.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while

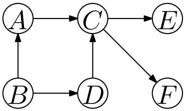


Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



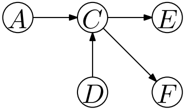
B gets number 1.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```



B gets number 1.

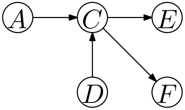
Remove B from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



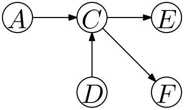
We can pick A or D.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



We can pick A or D.

Let us choose A.

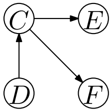
A gets number 2.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```



We can pick A or D.

Let us choose A.

A gets number 2.

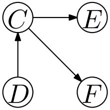
Remove A from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```



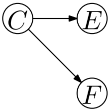
D gets number 3.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while



D gets number 3.

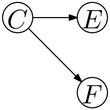
Remove D from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
  Choose a vertex $u \in V$ with indegree 0.
  Give u the number k.
  $k = k + 1$
  Remove u from G.
end while



C gets number 4.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
    Choose a vertex $u \in V$ with indegree 0.
    Give u the number k.
    $k = k + 1$
    Remove u from G.
end while

C gets number 4.

Remove C from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

We can pick E or F.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

$k = 1$
while $V \neq \{\,\}$ **do**
  Choose a vertex $u \in V$ with indegree 0.
  Give u the number k.
  $k = k + 1$
  Remove u from G.
end while

We can pick E or F.

Let us choose E.

E gets number 5.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

We can pick E or F.

Let us choose E.

E gets number 5.

Remove E from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

F gets number 6.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```

F gets number 6.

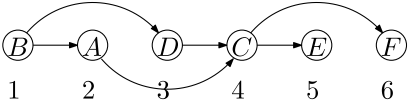
Remove F from G.

Algorithm *TopologicalOrdering*(G)

Input: A directed acyclic graph $G = (V, E)$

Output: A topological ordering of V

```
k = 1
while V ≠ { } do
    Choose a vertex u ∈ V with indegree 0.
    Give u the number k.
    k = k + 1
    Remove u from G.
end while
```



Prenumbers and Postnumbers

Let $G = (V, E)$ be a directed graph. For each vertex $v \in V$, we define the following two numbers with respect to Depth-First-Search.

$pre(v)$: the first time we visit v (the time at which $explore(v)$ is called)

$post(v)$: the time at which $explore(v)$ is finished

Prenumbers and Postnumbers

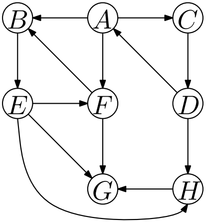
Let $G = (V, E)$ be a directed graph. For each vertex $v \in V$, we define the following two numbers with respect to Depth-First-Search.

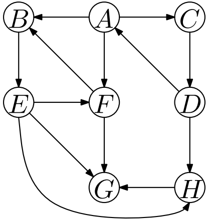
- $pre(v)$: the first time we visit v (the time at which $explore(v)$ is called)
- $post(v)$: the time at which $explore(v)$ is finished

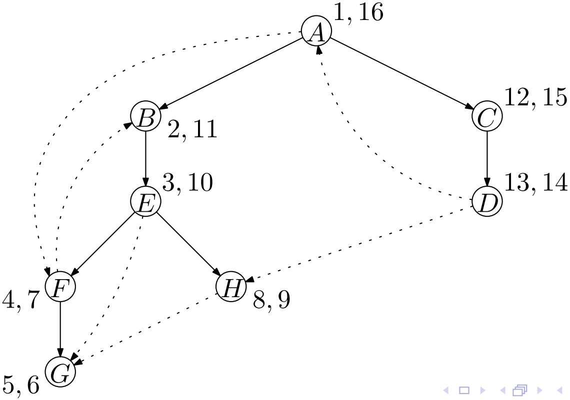
Use variable $clock$. At start, $clock = 1$.

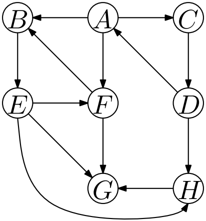
$$previsit(v) \equiv \quad \begin{array}{l} pre(v) = clock \\ clock = clock + 1 \end{array}$$

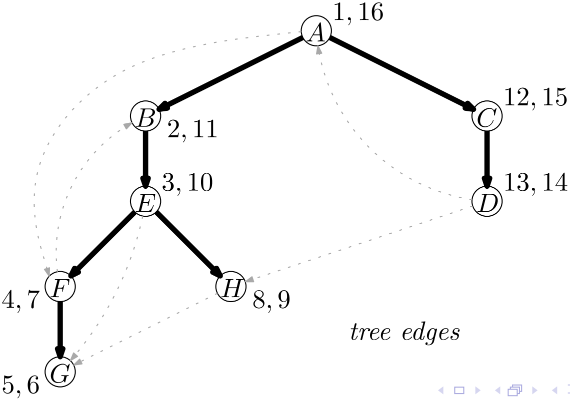
$$postvisit(v) \equiv \quad \begin{array}{l} post(v) = clock \\ clock = clock + 1 \end{array}$$


A
B
C
E
F
D
G
H

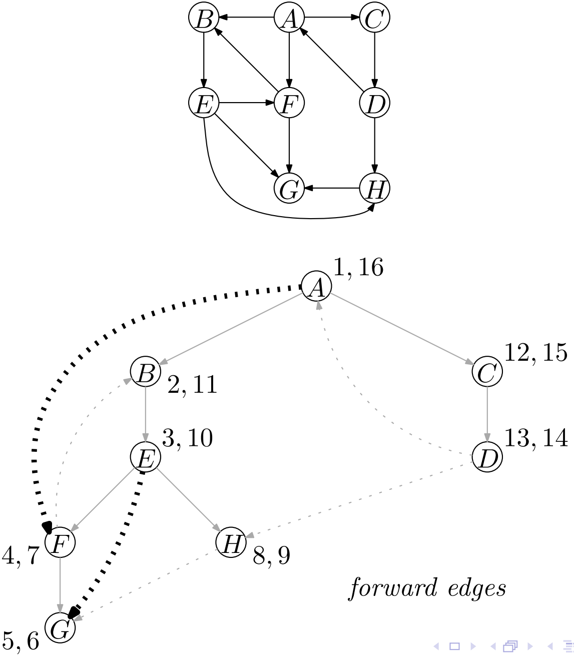

B
A
C
E
F
D
G
H


A
1, 16
B
2, 11
C
12, 15
E
3, 10
D
13, 14
F
4, 7
H
8, 9
G
5, 6

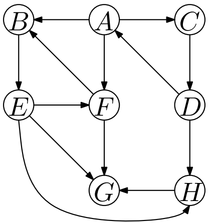


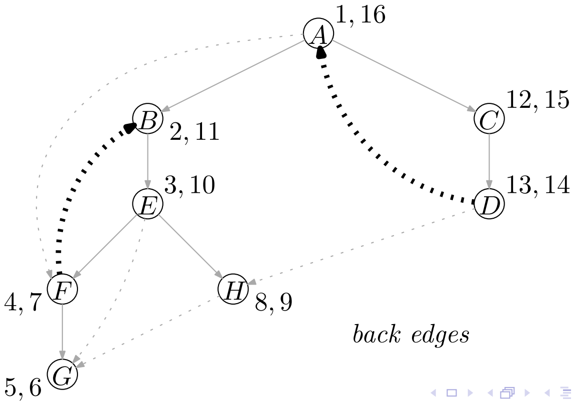


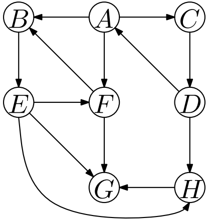
tree edges

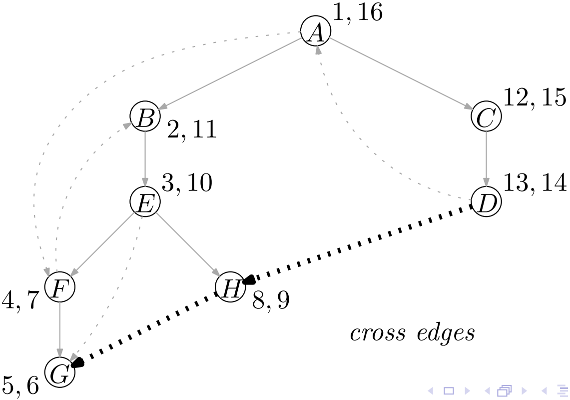


forward edges

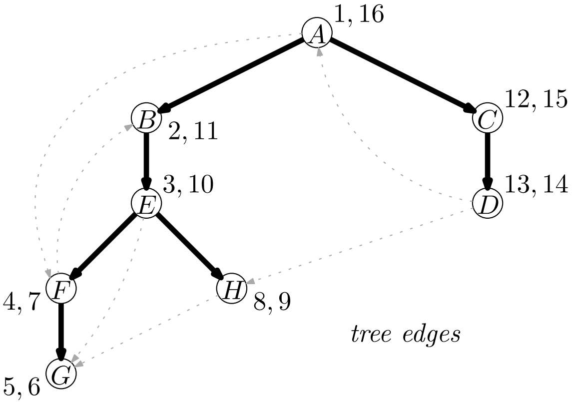

B
A
C
E
F
D
G
H


A 1, 16
B 2, 11
C 12, 15
E 3, 10
D 13, 14
F 4, 7
H 8, 9
G 5, 6
back edges


A
B
C
E
F
D
G
H


A 1, 16
B 2, 11
C 12, 15
E 3, 10
D 13, 14
F 4, 7
H 8, 9
G 5, 6
cross edges

4 Types of Edges



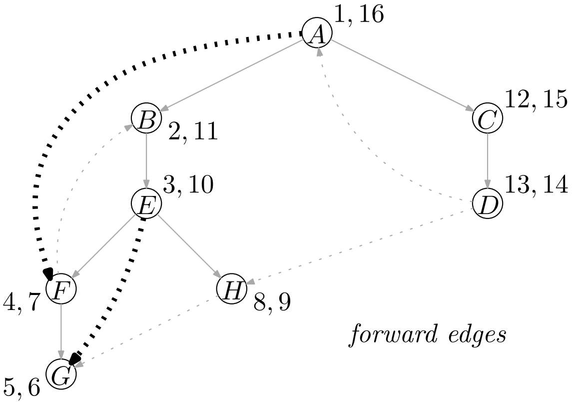
tree edges

Tree edge:

- edge $v \longrightarrow u$
- $explore(u)$ is called as a recursive call within $explore(v)$

Solid edges

4 Types of Edges



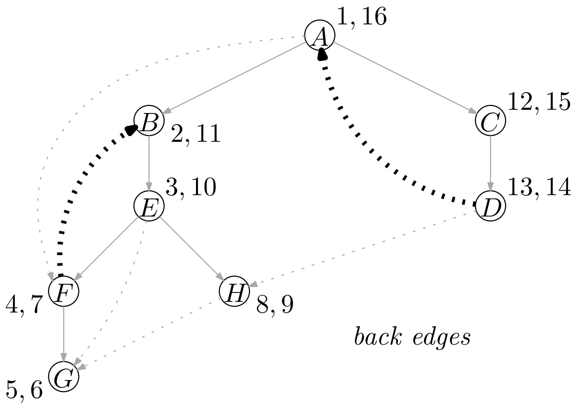
forward edges

Forward edge:

- edge $v \longrightarrow u$ where in the (solid) tree,
  - u is in subtree of v
  - u is not a child of v

(A, F), (E, G)

4 Types of Edges



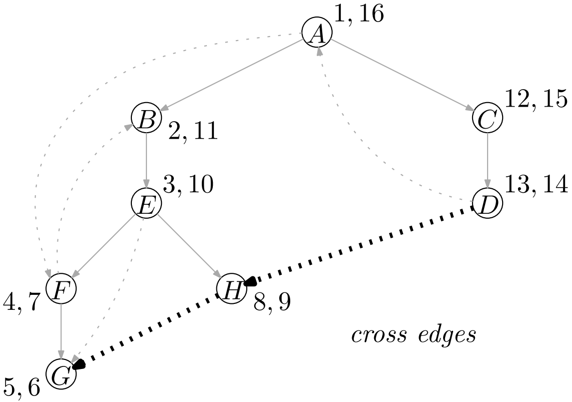
back edges

Back edge:

- edge $v \longrightarrow u$ where in the (solid) tree, $\cdot$ v is in subtree of u

(F, B), (D, A)

4 Types of Edges



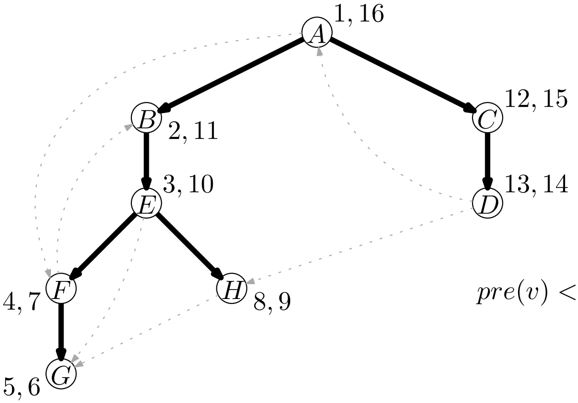
Cross edge:

- All other edges

(D, H), (H, G)

4 Types of Edges

How to decide the type of an edge?



Tree edge:

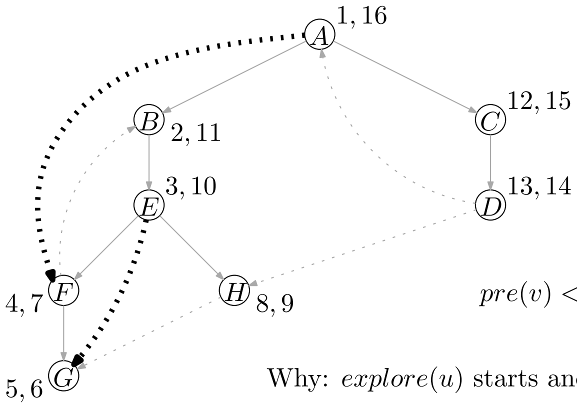
(v, u)

These are discovered during the execution of the algorithm.

$$pre(v) < pre(u) < post(u) < post(v)$$

4 Types of Edges

How to decide the type of an edge?



Forward edge:

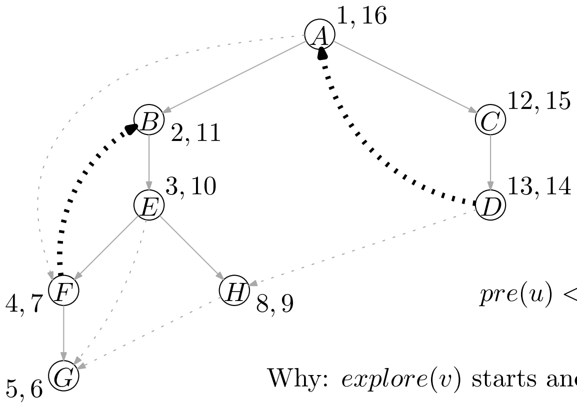
(v, u) not a tree edge

$$pre(v) < pre(u) < post(u) < post(v)$$

Why: $explore(u)$ starts and finishes within $explore(v)$.

4 Types of Edges

How to decide the type of an edge?



Back edge:

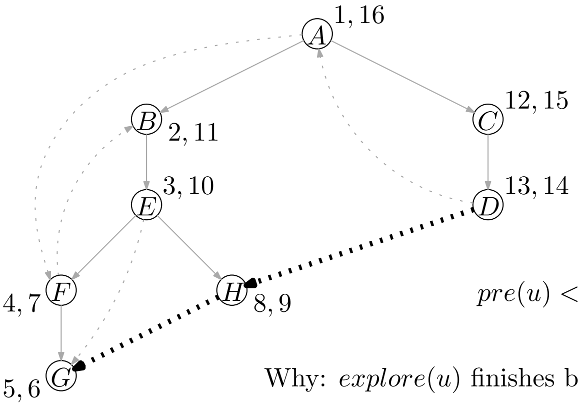
(v, u) where

$$pre(u) < pre(v) < post(v) < post(u)$$

Why: $explore(v)$ starts and finishes within $explore(u)$.

4 Types of Edges

How to decide the type of an edge?



Cross edge:

(v, u) where

$$pre(u) < post(u) < pre(v) < post(v)$$

Why: $explore(u)$ finishes before $explore(v)$ starts.

Acyclic vs Cyclic

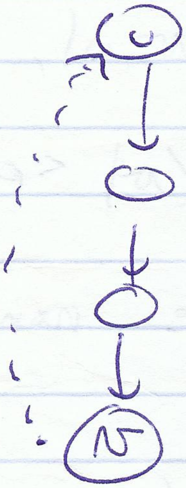
How to decide if a directed graph has a directed cycle?

Lemma

G has a directed cycle
if and only if
DFS-forest has a back-edge.

Proof:

[$\Longleftarrow$] Assume (v, u) is a back edge.



Then: the tree edges from u to v, plus edge (v, u) form a directed cycle.

[$\Longrightarrow$] Assume

$$v_0 \to v_1 \to v_2 \to \dots \to v_k \to v_0$$

is a directed cycle.

We may assume that v_0 has the smallest pre-number among the vertices on this cycle (otherwise, relabel the the vertices

Therefore, each of explore(v_1), explore(v_2), ..., explore(v_k) is called within explore(v_0)

Thus, each of $v_1, v_2, \dots, v_k$ is in the (solid) subtree of v_0

Hence, by definition of back edge, (v_k, v_0) is a back edge. $\square$

Cyclic Directed Graphs

How to test if a directed graph is cyclic?

Cyclic Directed Graphs

How to test if a directed graph is cyclic?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : For each non-tree edge (v, u), test if

$$pre(u) < pre(v) < post(v) < post(u).$$

- If "yes" for at least one non-tree edge, return "cyclic".
- If "no" for all non-tree edges, return "acyclic".

Cyclic Directed Graphs

How to test if a directed graph is cyclic?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : For each non-tree edge (v, u), test if

$$pre(u) < pre(v) < post(v) < post(u).$$

- If "yes" for at least one non-tree edge, return "cyclic".
- If "no" for all non-tree edges, return "acyclic".

Running time: $O(|V| + |E|)$.

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

How much time does it take for Bucket Sort?

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

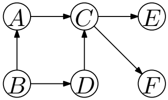
How much time does it take for Bucket Sort?

Total running time: $O(|V| + |E|)$.

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

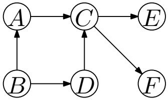
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

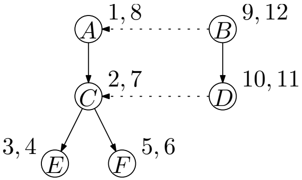


Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

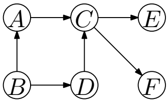


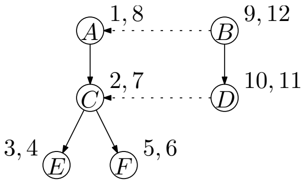


Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

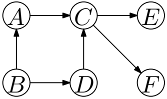
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

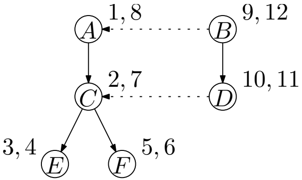




Sort by postnumber: E, F, C, A, D, B

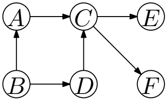
Step 1 : Run DFS (including pre/post-numbers)
Step 2 : Run Bucket Sort to sort the vertices by postnumber.
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

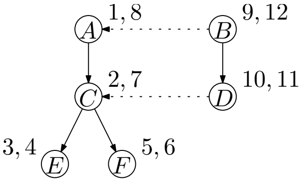




Sort by postnumber: E, F, C, A, D, B
Topologial ordering:
$B \quad D \quad A \quad C \quad F \quad E$

Step 1 : Run DFS (including pre/post-numbers)
Step 2 : Run Bucket Sort to sort the vertices by postnumber.
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.





Sort by postnumber: E, F, C, A, D, B
Topologial ordering:

	B	D	A	C	F	E
#	1	2	3	4	5	6

Correctness: Let (v, u) be any edge in G.

To show: in topological sorting: number of v < number of u

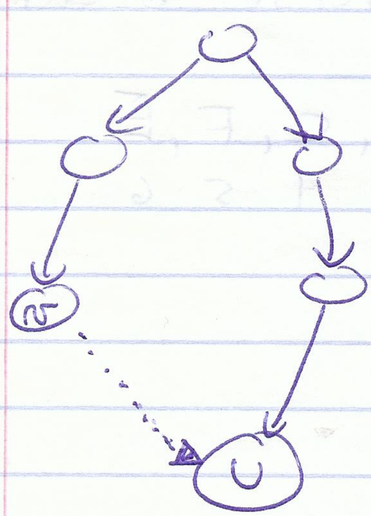
$$\text{post}(v) > \text{post}(u)$$

By contradiction. Assume $\text{post}(v) < \text{post}(u)$

Therefore, (v, u) is not a tree edge and not a forward edge (see page 73).

Since G is acyclic, (v, u) is not a back edge.

Hence, (v, u) is a cross edge.



But since $\text{post}(v) < \text{post}(u)$, (v, u) is not a cross edge, which is a contradiction.

$\square$