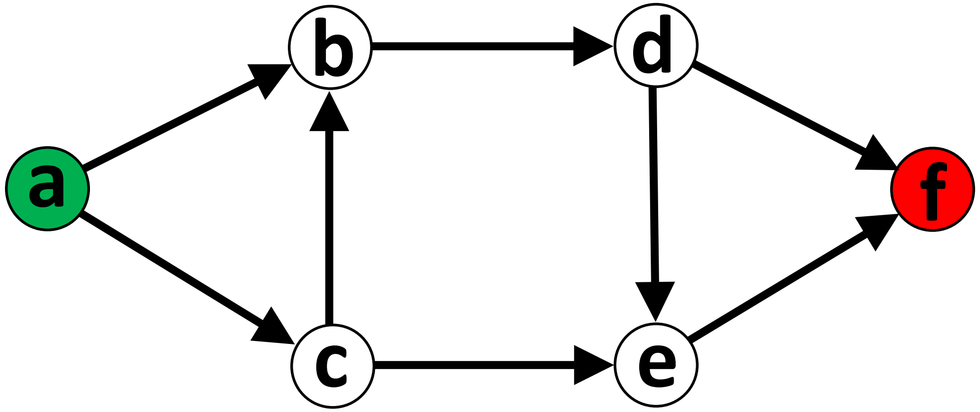
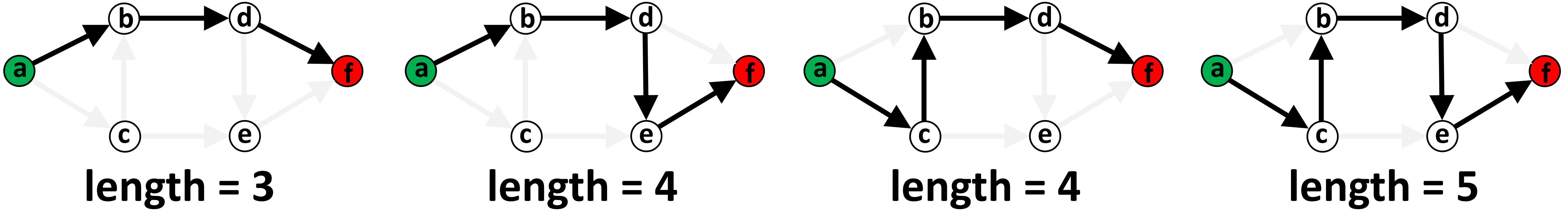
Dynamic Programming
CSCI 532

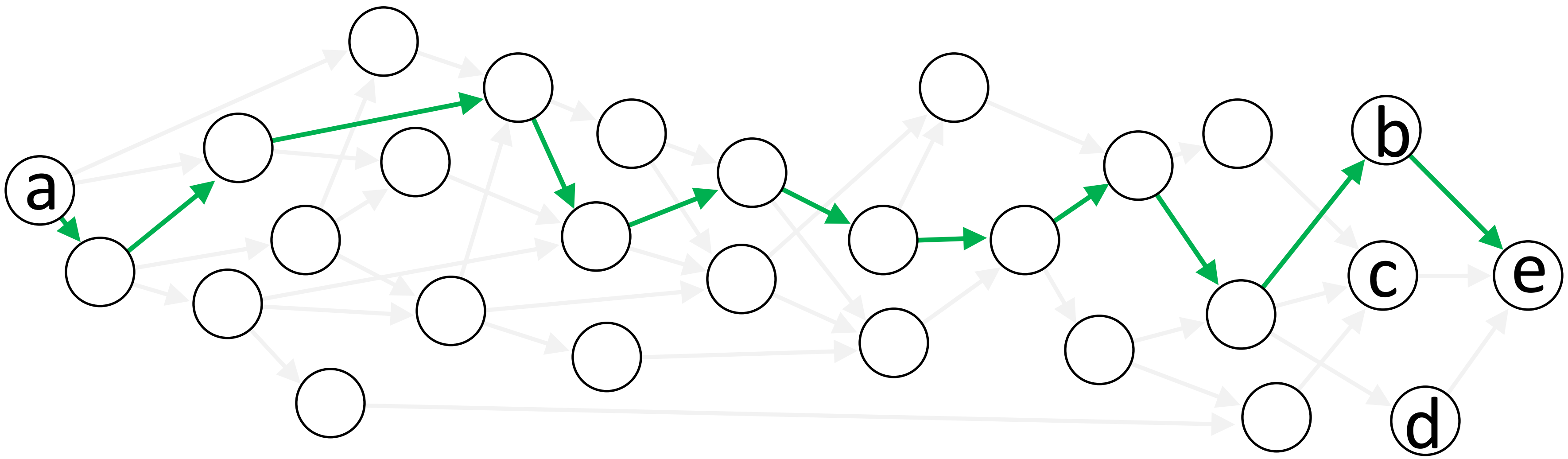
Longest Path in a DAG

Given a DAG, find the longest path between any two vertices in the graph.





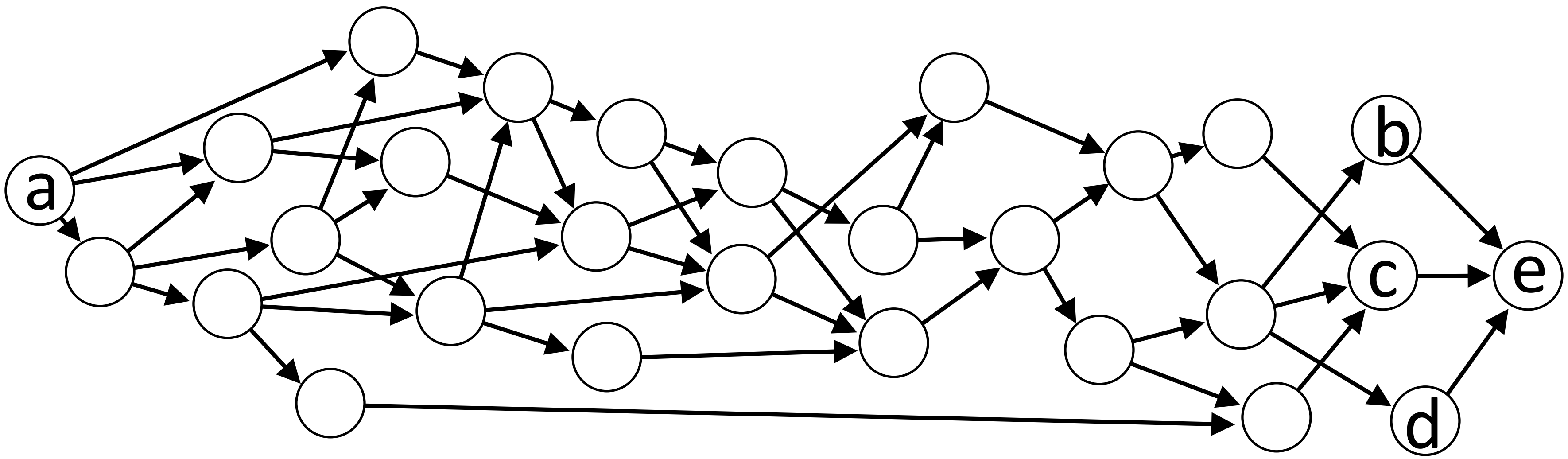
Find the Longest Path in a DAG



Interesting observations?

If the longest path goes from **a** to **e** and passes through **b**, that must be the longest path that ends at **b**. If not, then we could make a longer path.

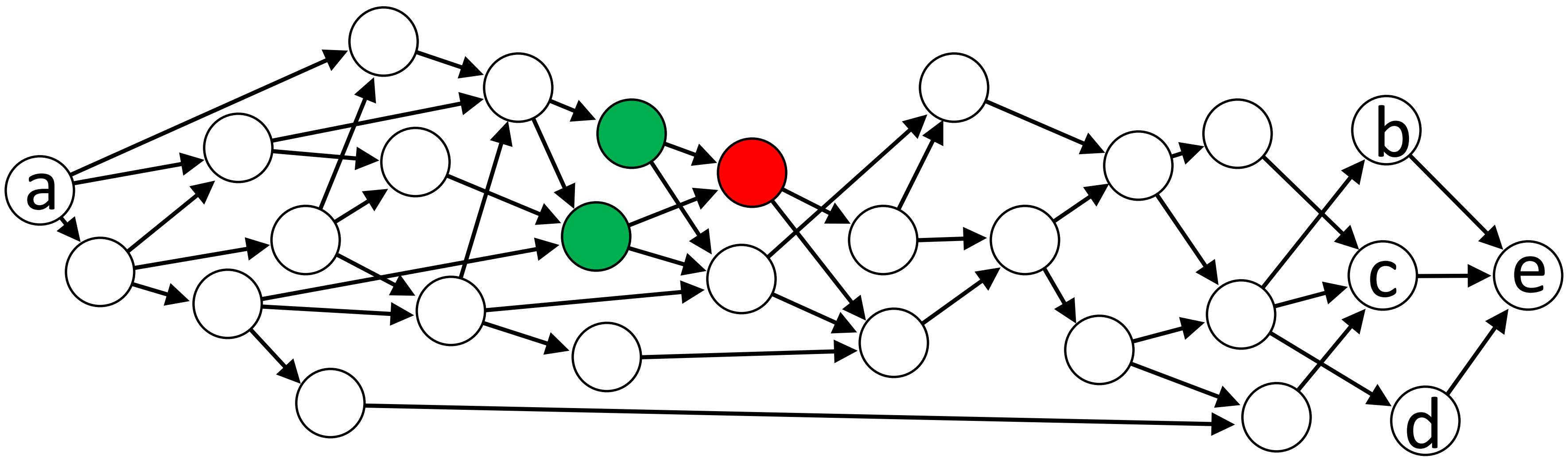
Find the Longest Path in a DAG



Interesting observations?

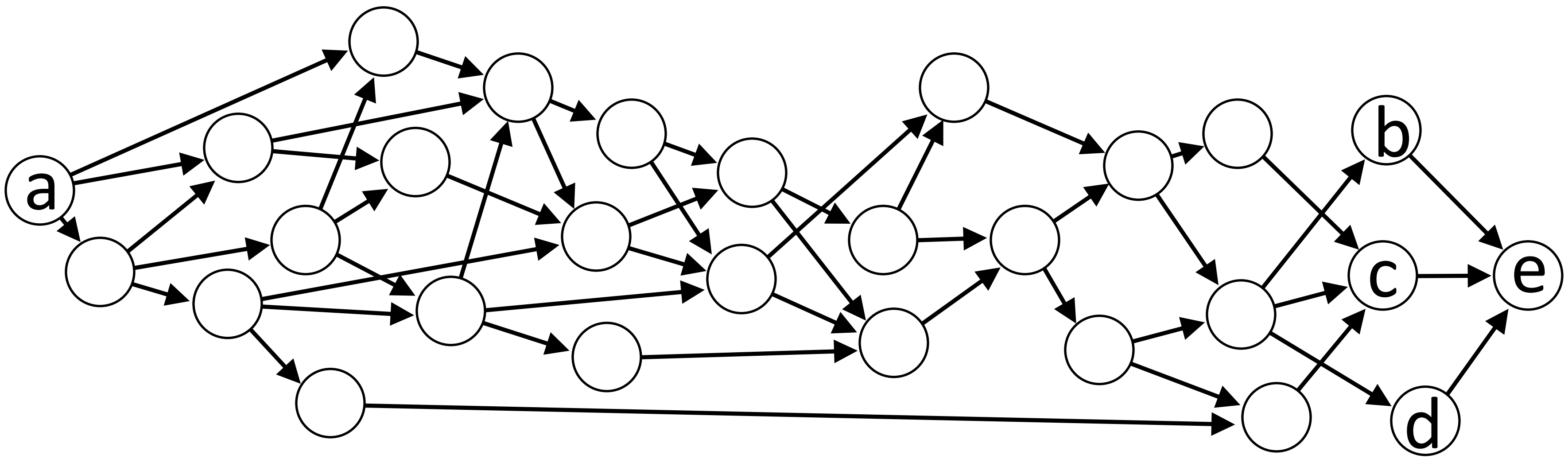
$$\text{The longest path to e} = \max\begin{pmatrix}\text{longest path to b}\\\text{longest path to c}\\\text{longest path to d}\end{pmatrix} + 1$$

Find the Longest Path in a DAG



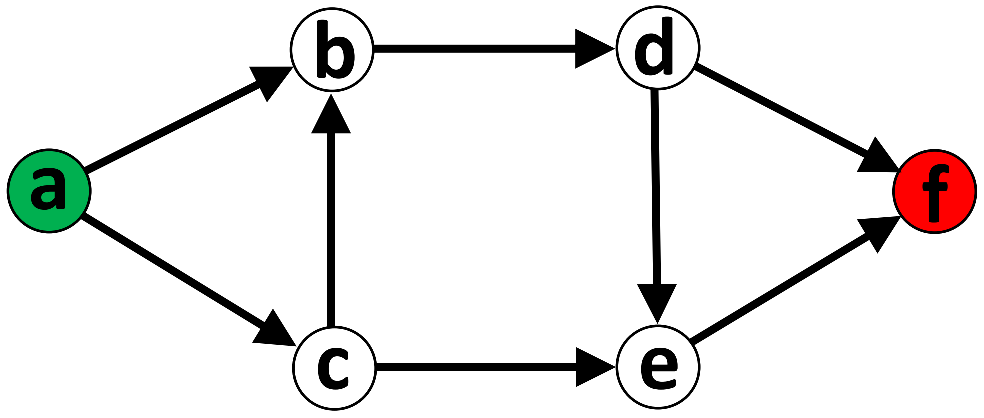
In general: We are ready to calculate the longest path to a vertex if we know the longest path for all incoming neighbors.

Find the Longest Path in a DAG



Topological Ordering of a graph: ordering of its vertices such that for every directed edge (u, v), vertex u comes before vertex v in the ordering.

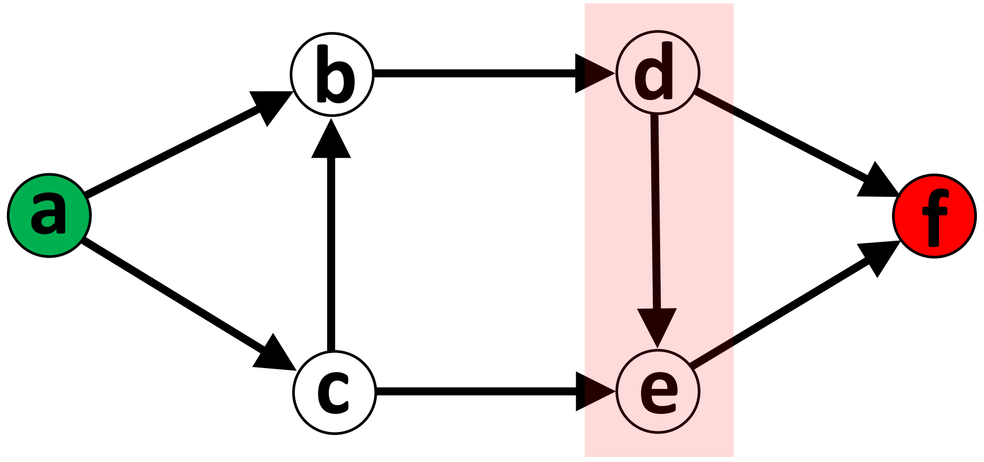
Topological Ordering



Topologically Ordered:

{a, c, b, d, e, f} ✓

Topological Ordering

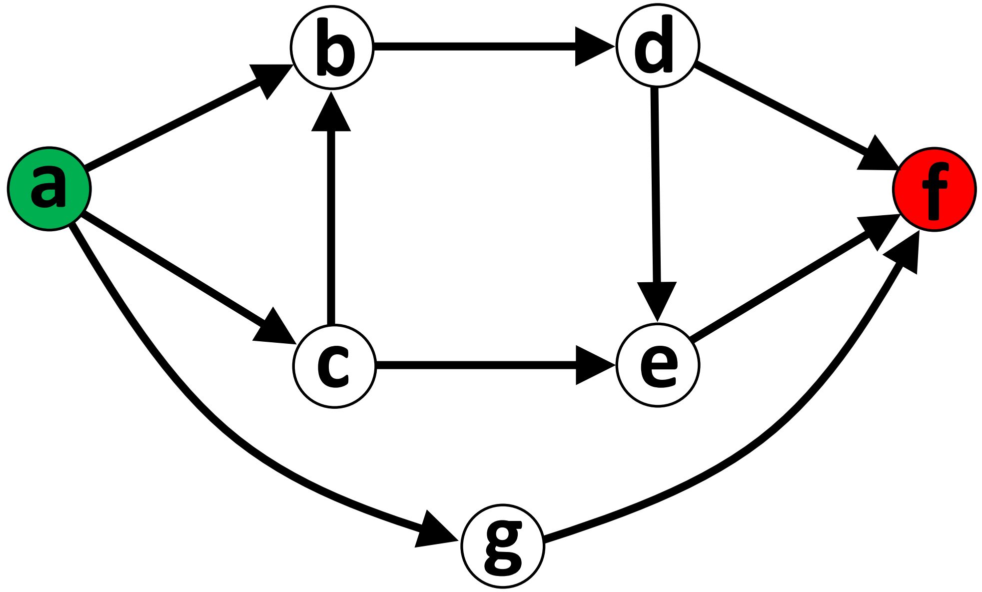


Topologically Ordered:

{a, c, b, d, e, f} ✓

{a, c, b, e, d, f} ✗

Topological Ordering

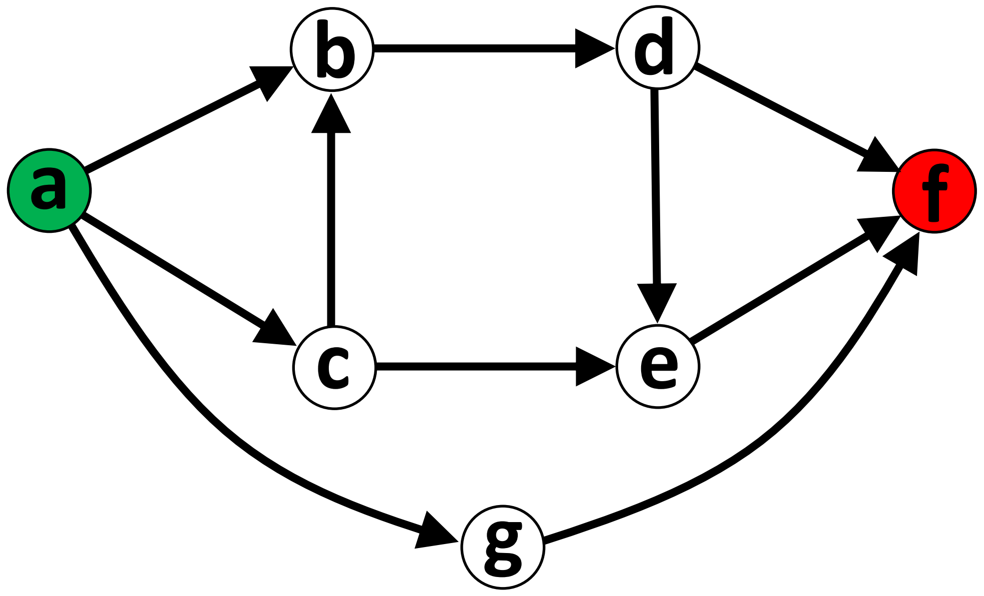


Topologically Ordered:

{a, c, g, b, d, e, f} ✓

{a, c, b, d, e, g, f} ✓

Topological Ordering

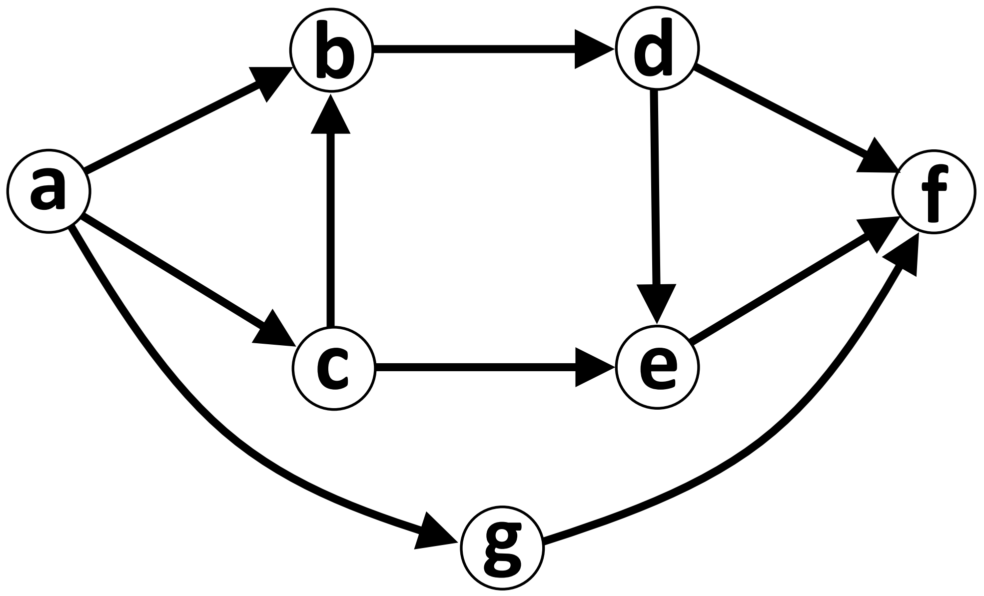


- There are various algorithms to find topological orderings
- Standard running time $= O(|V| + |E|)$.

Find the Longest Path in a DAG

Plan:

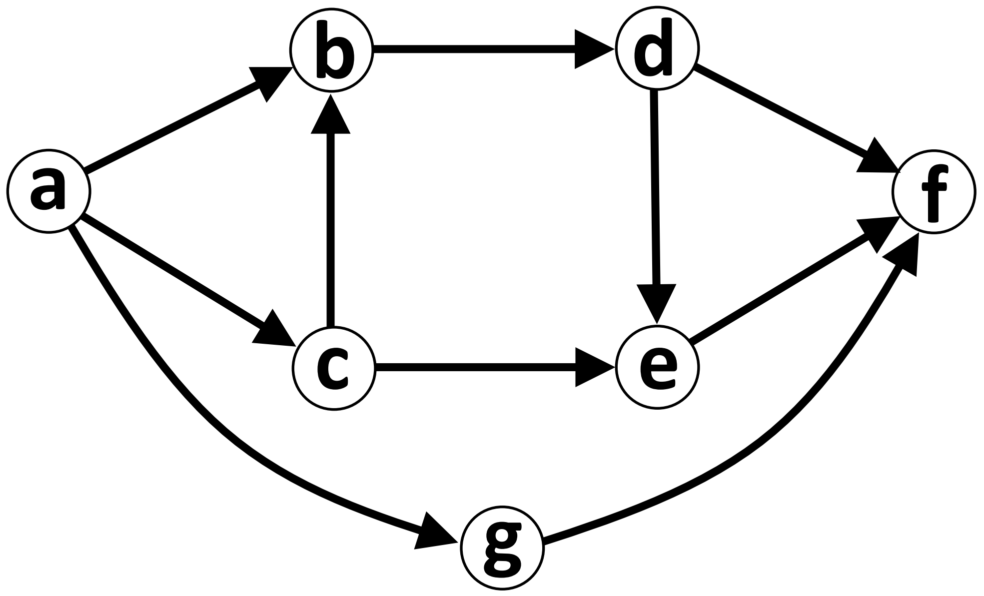
- ??



Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.

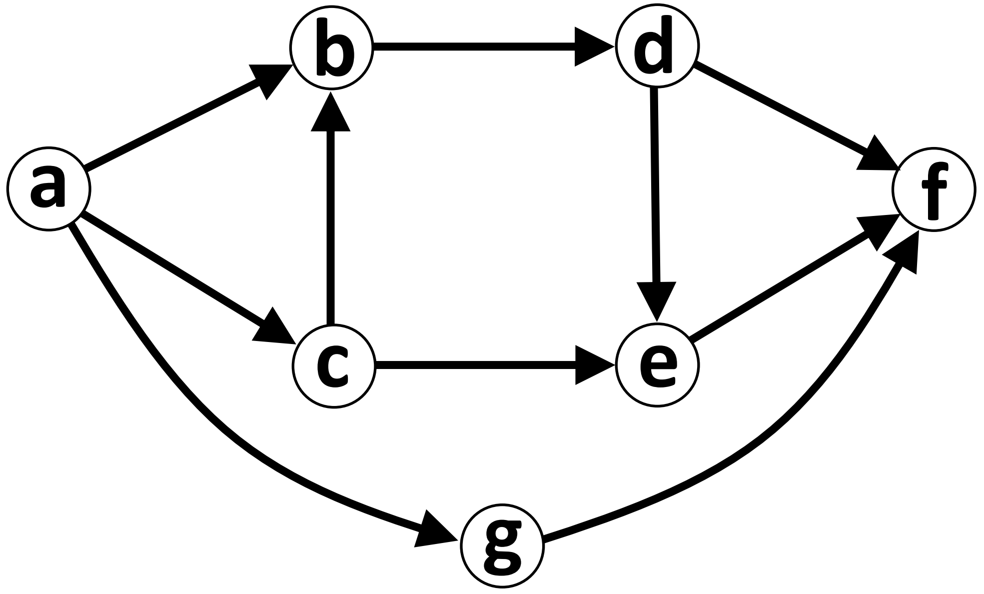


{a, c, g, b, d, e, f}

Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	0	0	0	0	0

Length of longest path that ends at c.

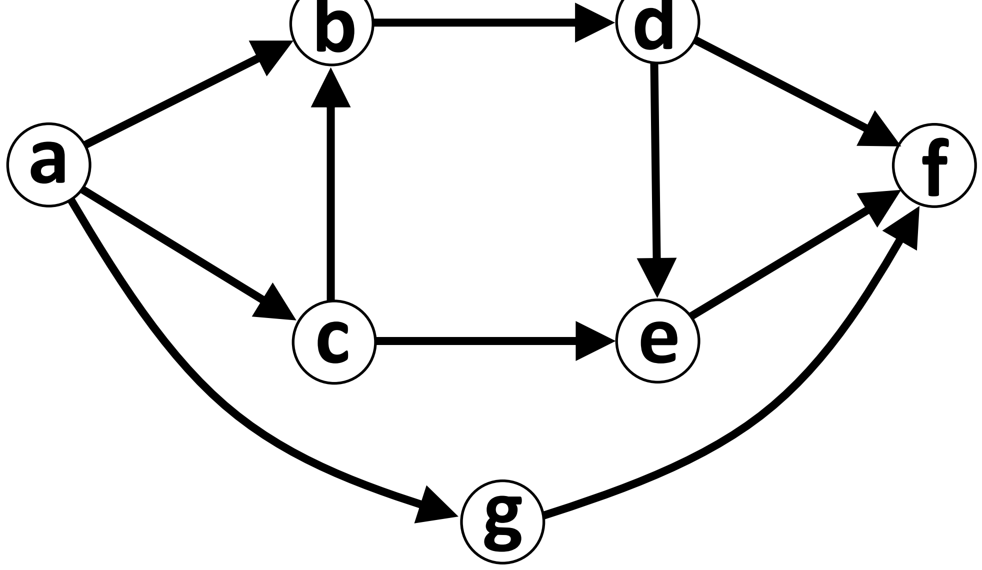
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n(\text{longest path to } n) + 1$,
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	0	0	0	0	0

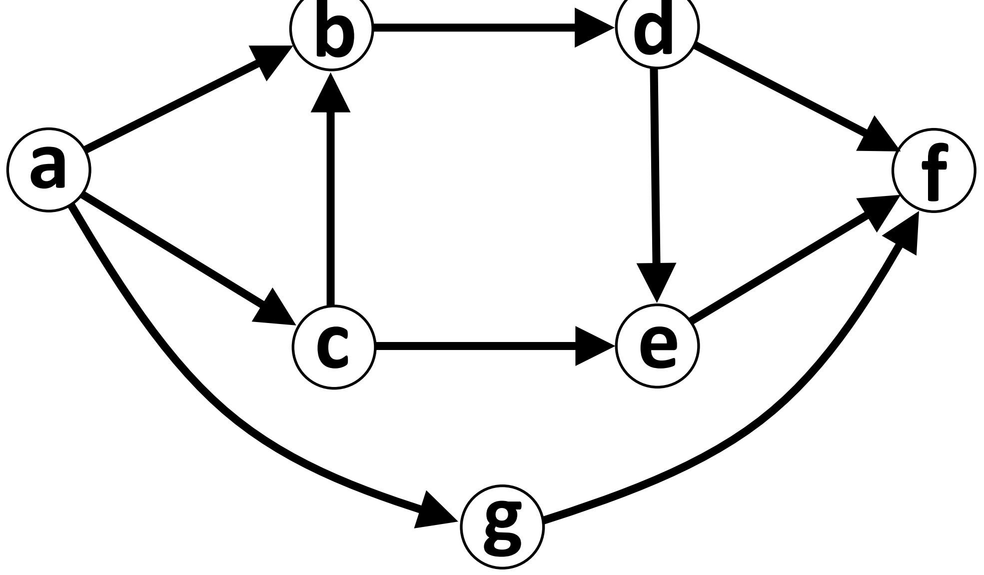
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n(\text{longest path to } n) + 1$,
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	0	0	0	0	0

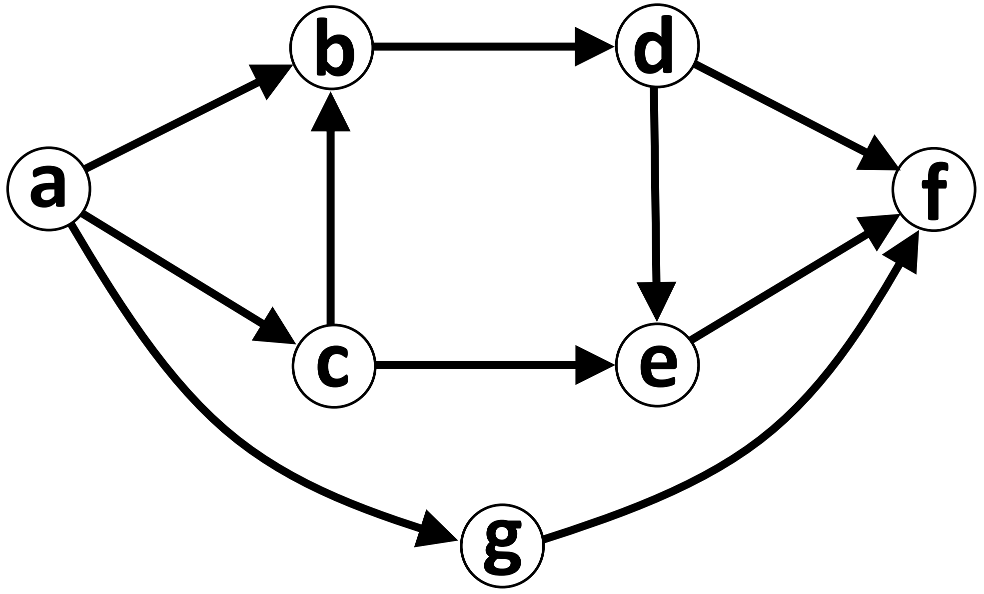
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n$(longest path to n) $+ 1$,
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	0	0	0	0	0

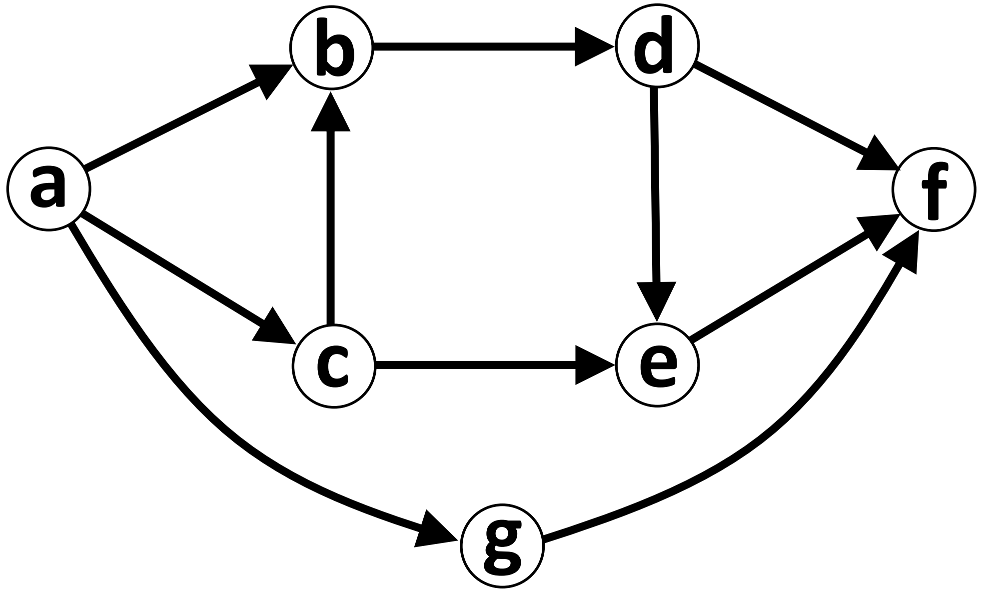
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n(\text{longest path to } n) + 1,$
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	1	0	0	0	0

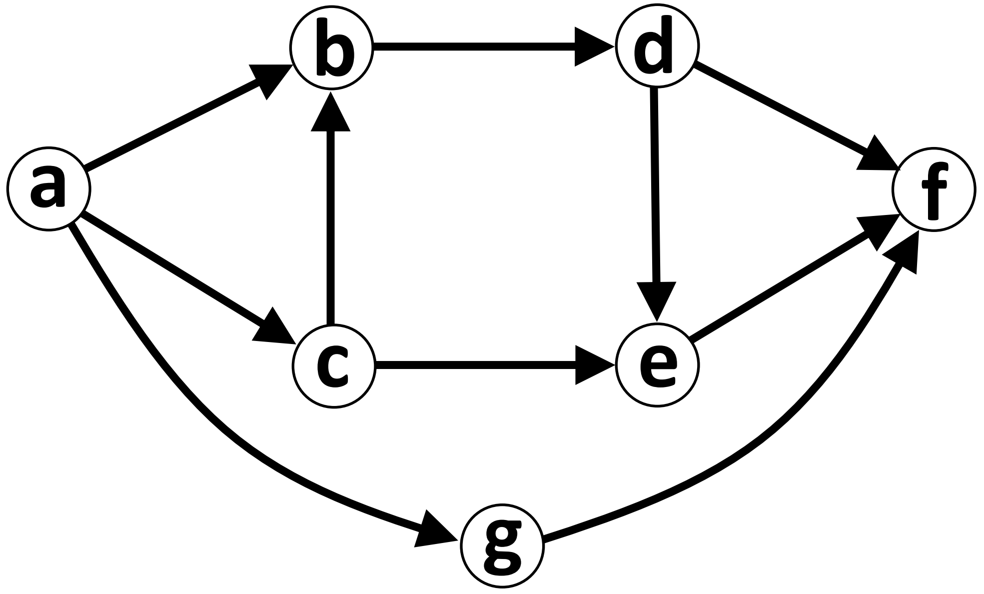
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n$(longest path to n) $+ 1$,
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	1	0	0	0	0

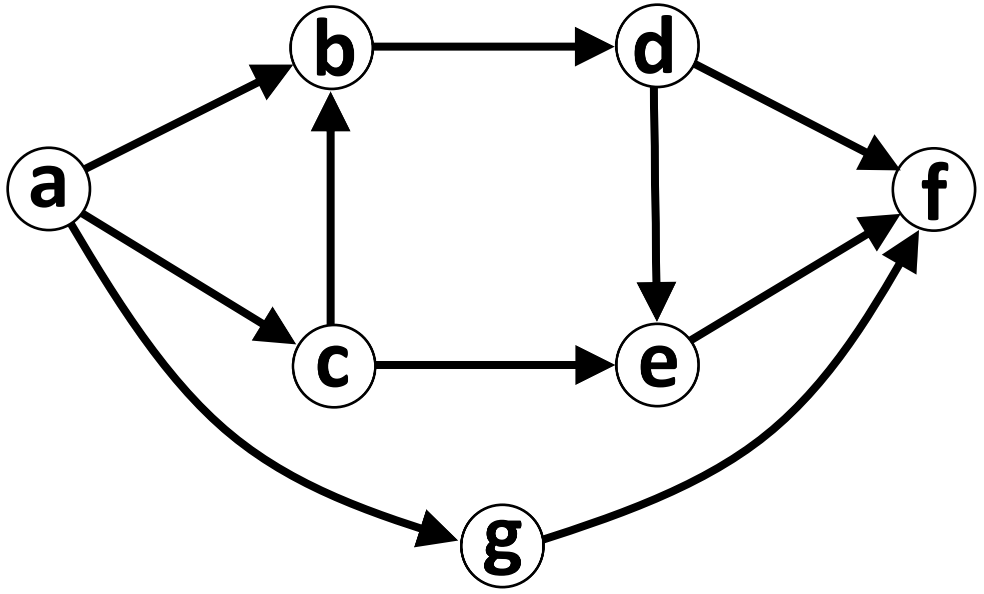
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n$(longest path to n) $+ 1$,
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	1	0	0	0	1

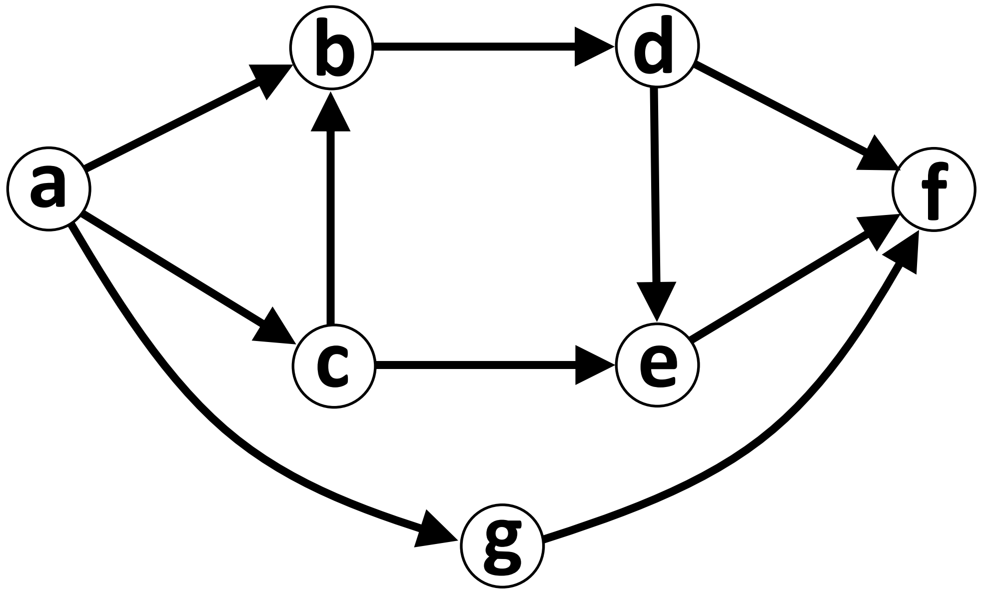
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n(\text{longest path to } n) + 1,$
  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	1	0	0	0	1

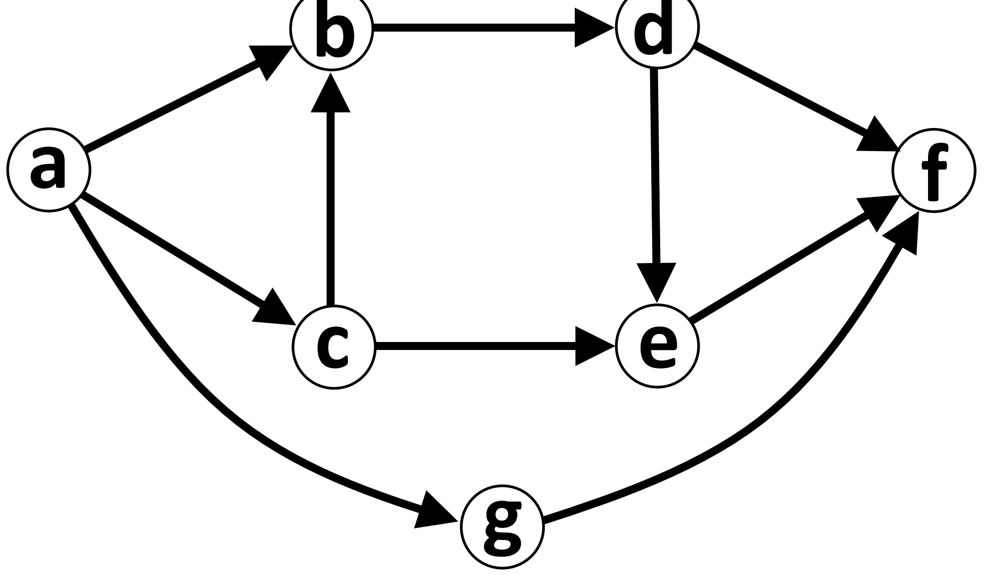
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

$\max_n$(longest path to n) $+ 1$,
for all incoming neighbors n.

(Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	0	0	0	1

Find the Longest Path in a DAG

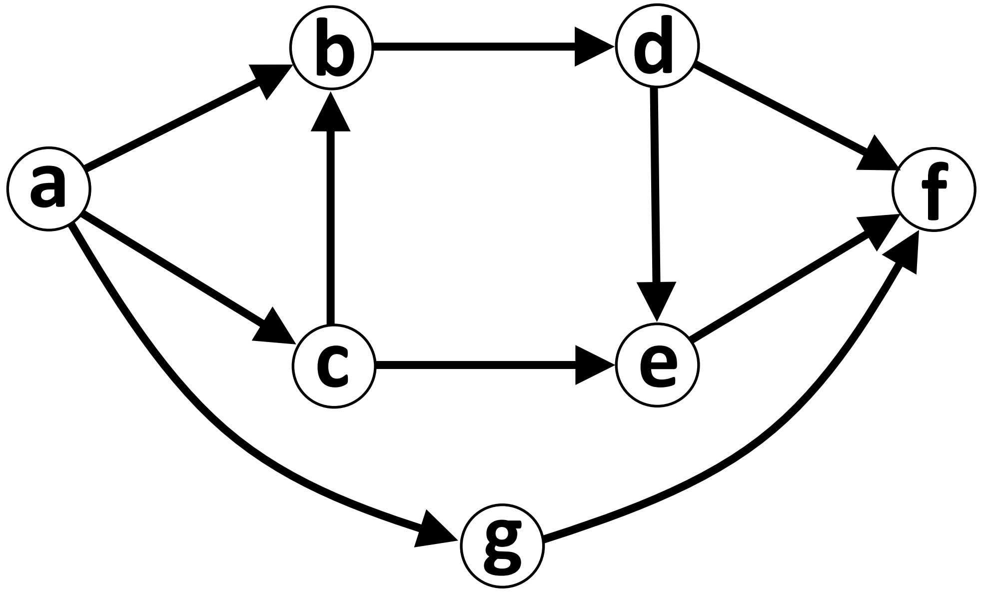
Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n(\text{longest path to } n) + 1,$

  for all incoming neighbors n.

  (Or 0 if there are no incoming neighbors)



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1

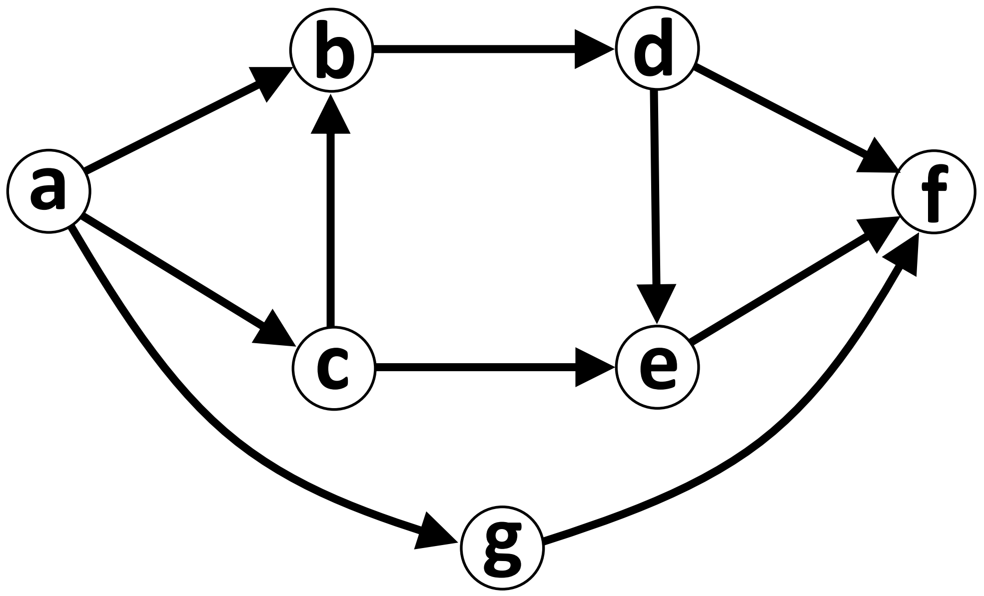
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:

  $\max_n(\text{longest path to } n) + 1,$

  for all incoming neighbors n.
- Largest value in array = Longest path.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1

Find the Longest Path in a DAG

```
longest_path(G=(V,E)):
    pathLengths = [0,...,0]
    Let Vsort be topologically sort vertices
    for each vertex v in Vsort:
        for each incoming neighbor n of v:
            if pathLengths[n] + 1 > pathLengths[v]:
                pathLengths[v] = pathLengths[n] + 1
    return maxValue(pathLengths)
```

Running time: ?

Find the Longest Path in a DAG

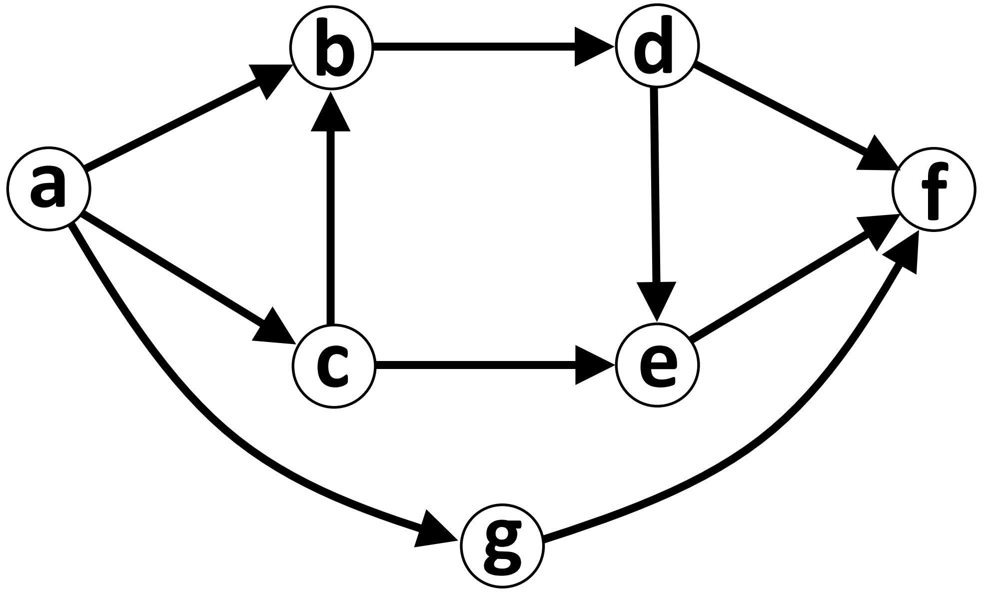
```
longest_path(G=(V,E)):
    pathLengths = [0,...,0]
    Let V_sort be topologically sort vertices
    for each vertex v in V_sort:
        for each incoming neighbor n of v:
            if pathLengths[n] + 1 > pathLengths[v]:
                pathLengths[v] = pathLengths[n] + 1
    return maxValue(pathLengths)
```

Running time: $O(\text{Topological Sort} + |V|^2) \in O(|V|^2)$

Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store length of longest path that ends at each vertex.
- For each vertex in order, calculate longest path as:
  $\max_n$(longest path to n) $+ 1$,
  for all incoming neighbors n.
- Largest value in array = Longest path.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1

Find the Longest Path in a DAG

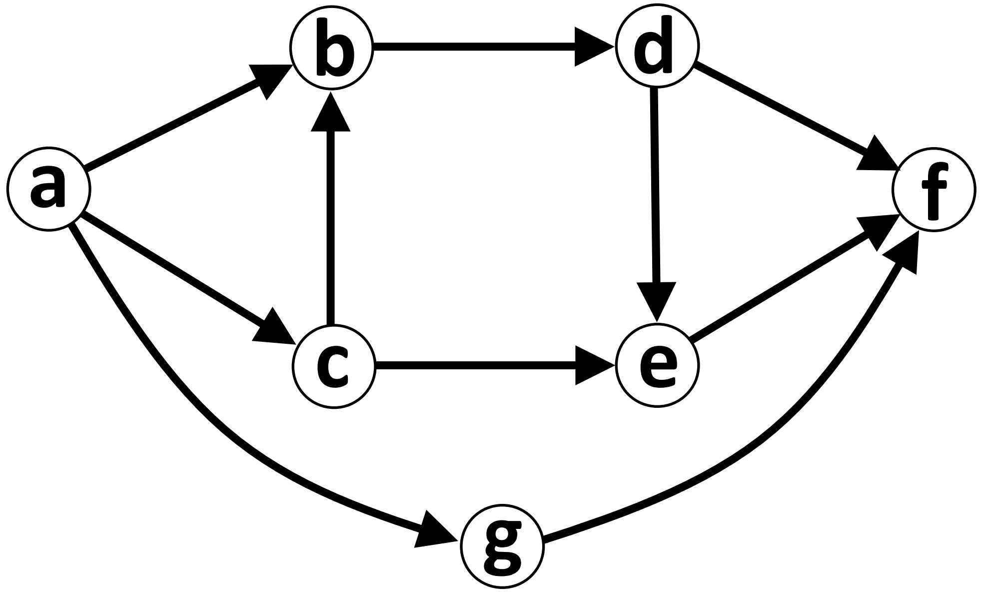
Plan:

- Topologically sort vertices.
- Make array to store le ongest path that ends at e tex.
- For each verte der, calculate longest pa

  ma gest path to n) $+ 1$,

  ncoming neighbors n.
- Larg st value in array = Longest path.

So, what's the path??



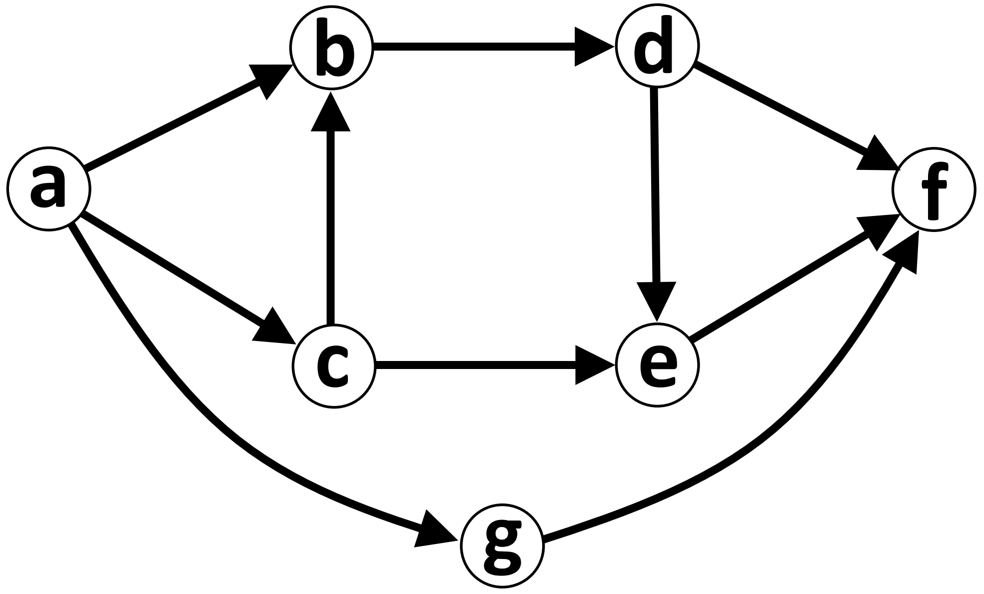
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.



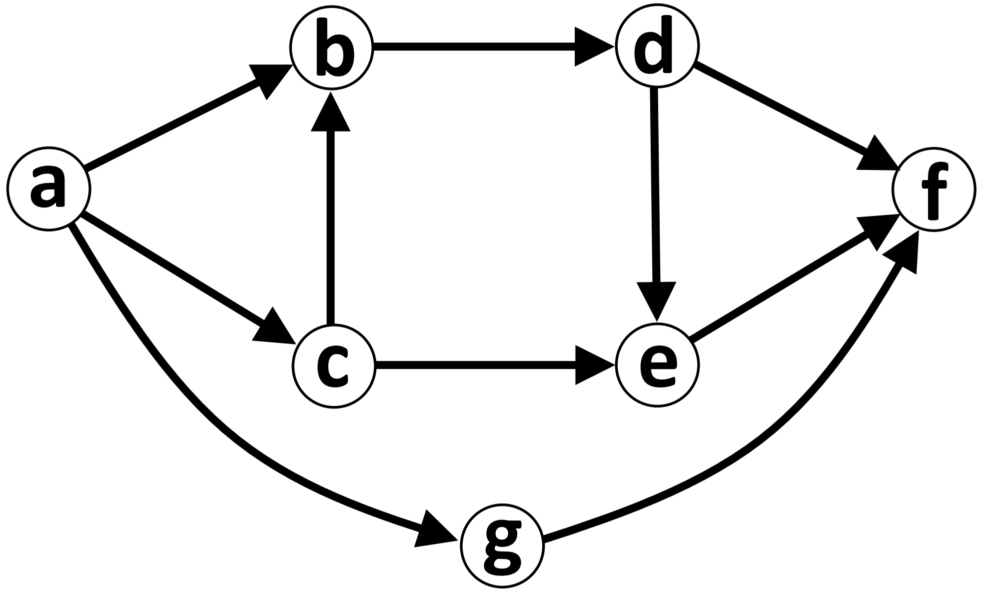
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	-	-	-	-	-	-

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.



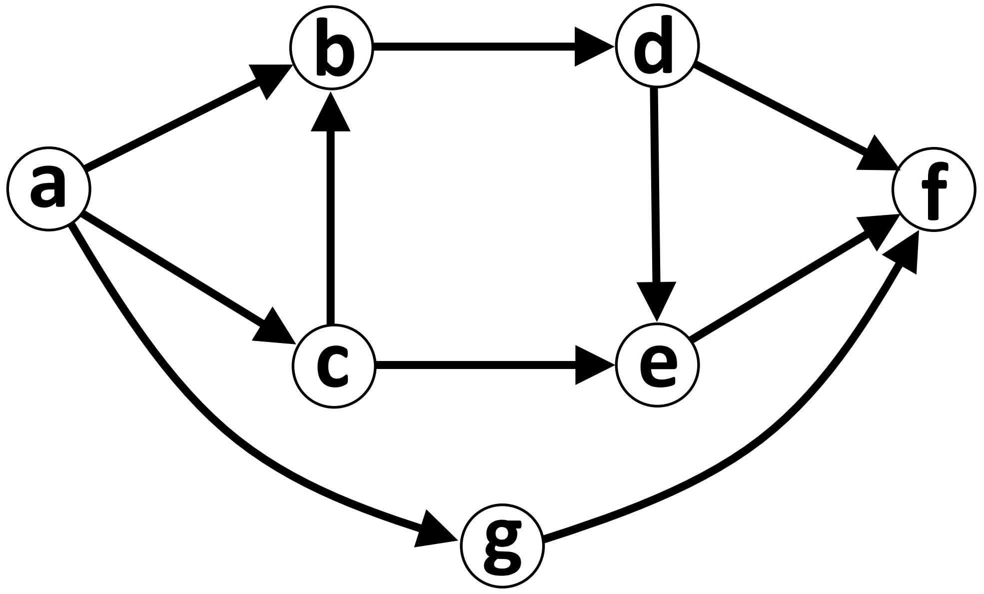
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	-	-	-	-	-	-

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.



{a, c, g, b, d, e, f}

For each vertex in order, calculate longest path as:

$\max_n(\text{longest path to } n) + 1$,

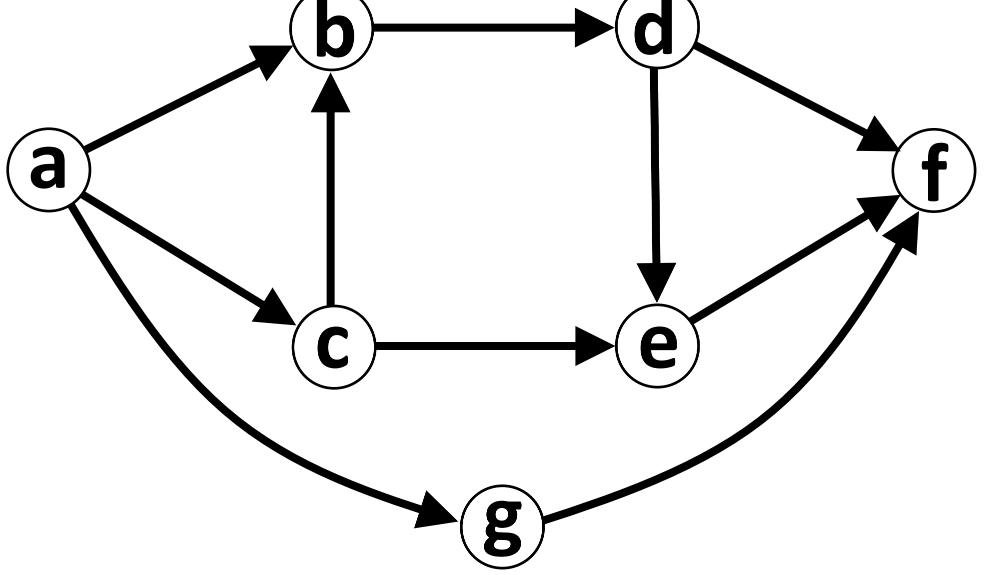
for all incoming neighbors n.

a	b	c	d	e	f	g
0	0	0	0	0	0	0
-	-	-	-	-	-	-

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.



{a, c, g, b, d, e, f}

For each vertex in order, calculate longest path as:

$\max_n(\text{longest path to } n) + 1,$

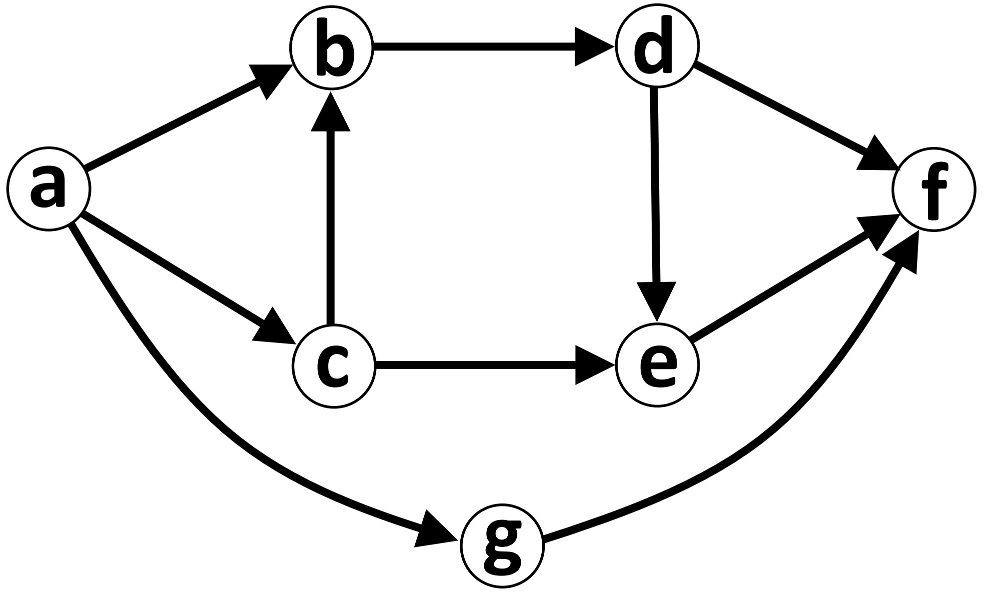
for all incoming neighbors n.

a	b	c	d	e	f	g
0	0	0	0	0	0	0
-	-	-	-	-	-	-

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	0	0	0	0	0
-	-	-	-	-	-	-

For each vertex in order, calculate longest path as:

$\max_n$(longest path to n) $+ 1$,

for all incoming neighbors n.

Find the Longest Path in a DAG

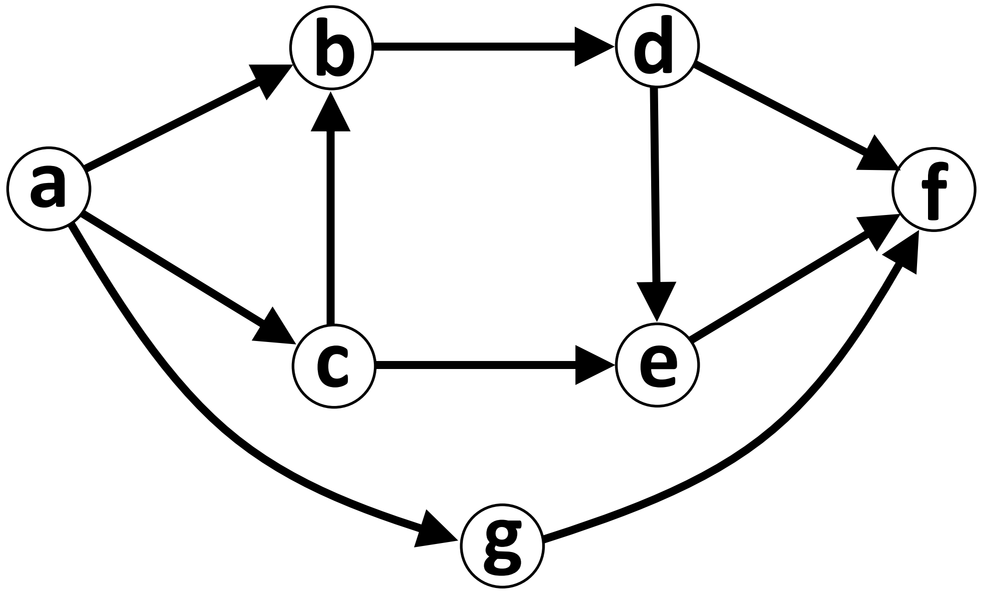
Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.

For each vertex in order, calculate longest path as:

$\max_n$(longest path to n) $+ 1$,

for all incoming neighbors n.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	1	0	0	0	0
-	-	-	-	-	-	-

Find the Longest Path in a DAG

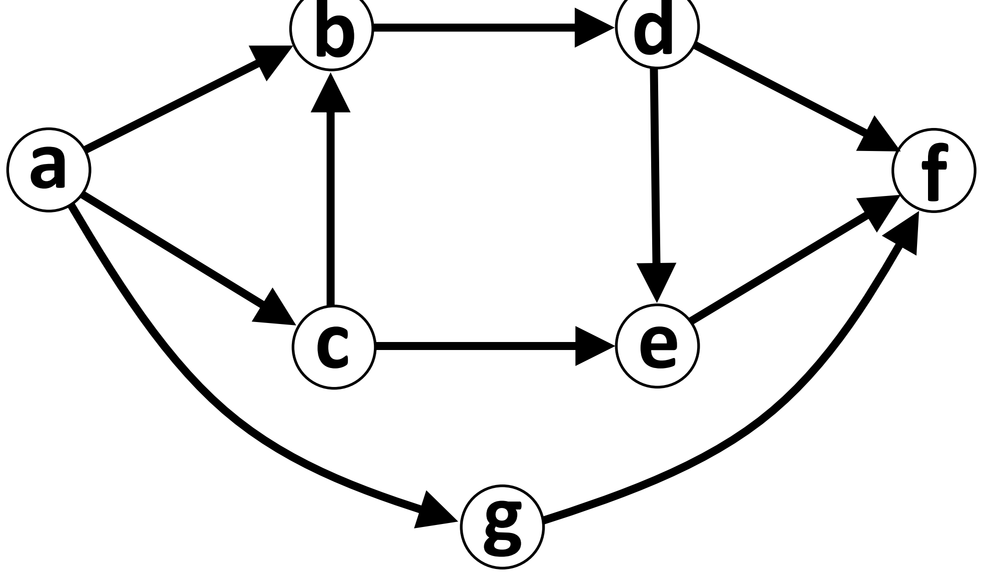
Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.

For each vertex in order, calculate longest path as:

$\max_n(\text{longest path to } n) + 1$,

for all incoming neighbors n.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	0	1	0	0	0	0
-	-	a	-	-	-	-

Find the Longest Path in a DAG

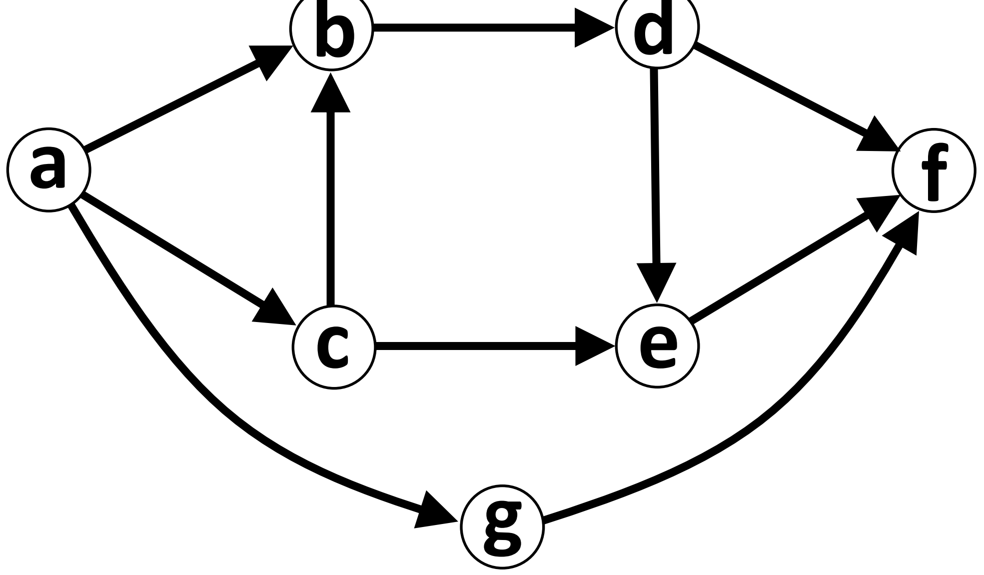
Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.

For each vertex in order, calculate longest path as:

$$\max_n(\text{longest path to } n) + 1,$$

for all incoming neighbors n.



{a, c, g, b, d, e, f}

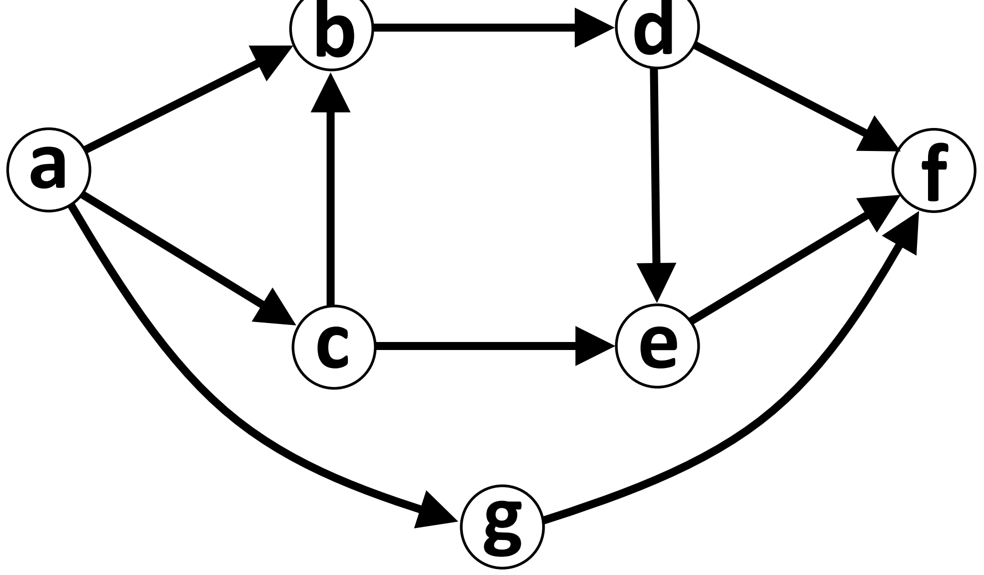
a	b	c	d	e	f	g
0	0	1	0	0	0	1
-	-	a	-	-	-	a

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.

For each vertex in order, calculate longest path as:
$\max_n(\text{longest path to } n) + 1$,
for all incoming neighbors n.



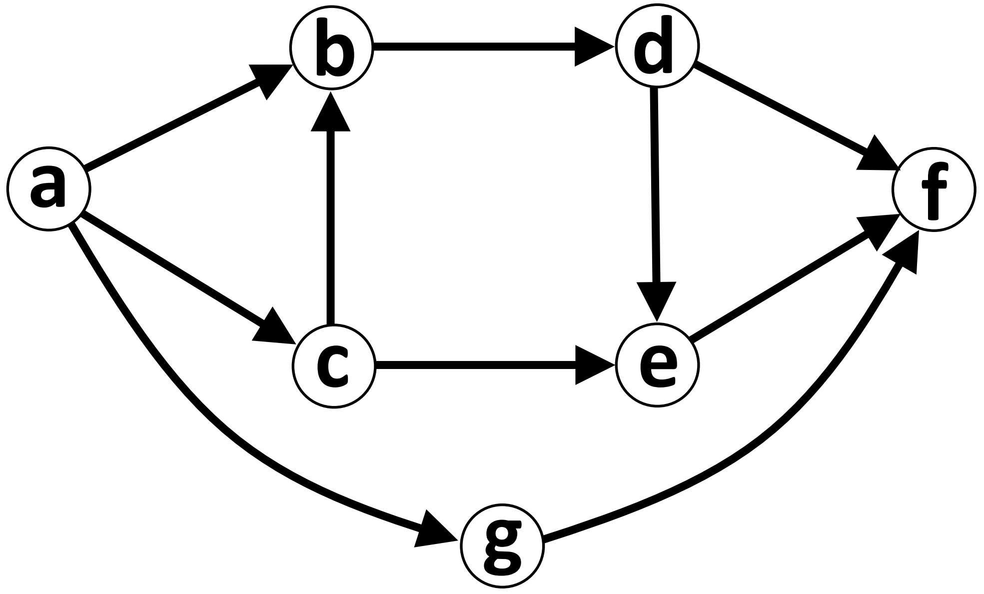
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	0	0	0	1
-	c	a	-	-	-	a

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.



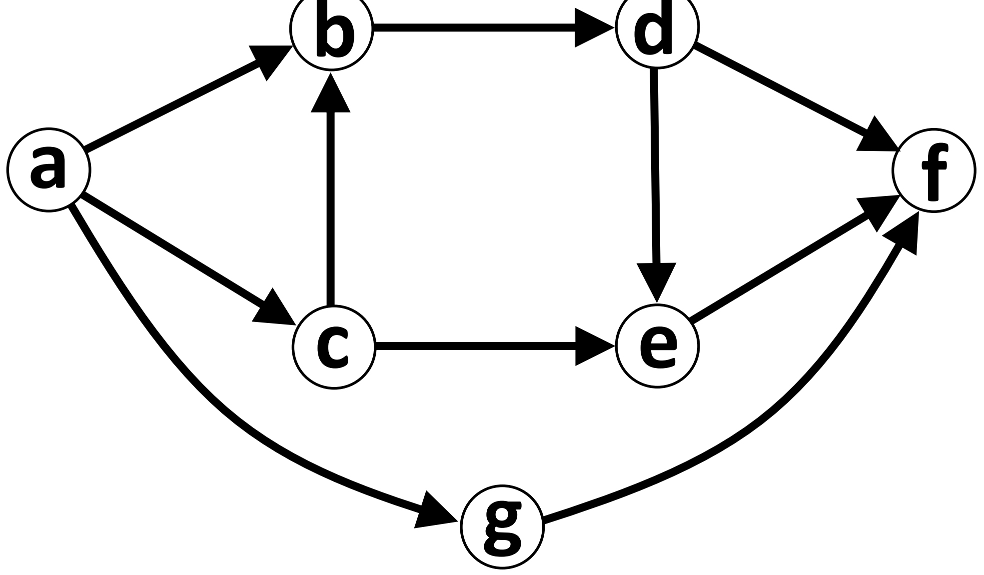
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.
- Backtrack through array to construct path.



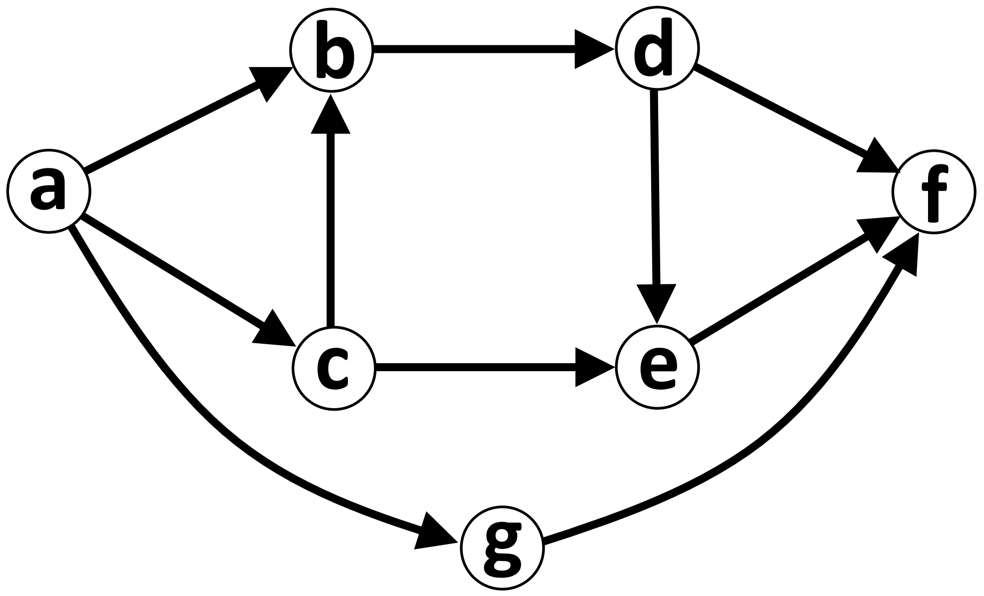
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.
- Backtrack through array to construct path.



{a, c, g, b, d, e, f}

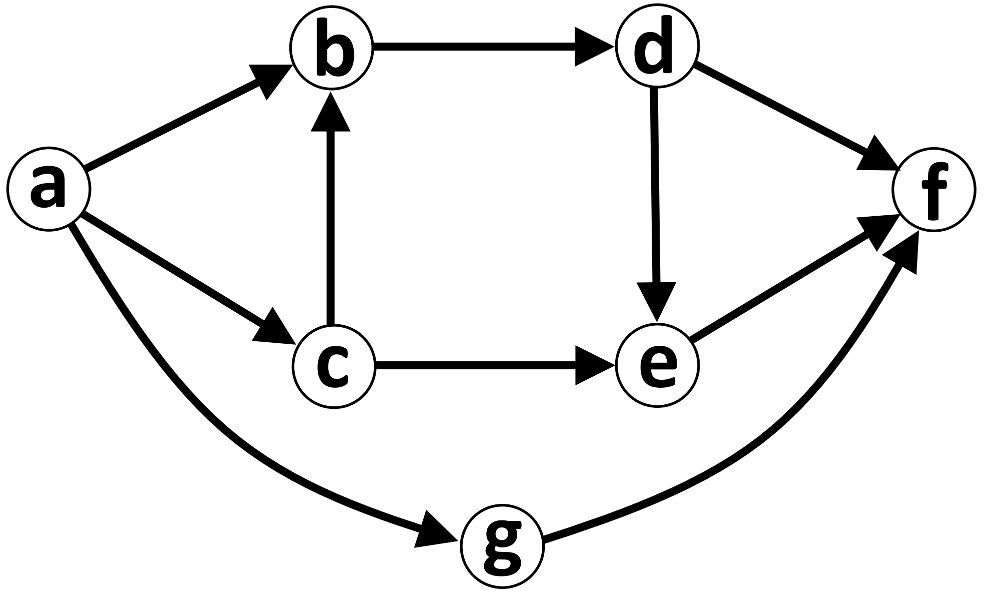
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

path: f

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.
- Backtrack through array to construct path.



{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

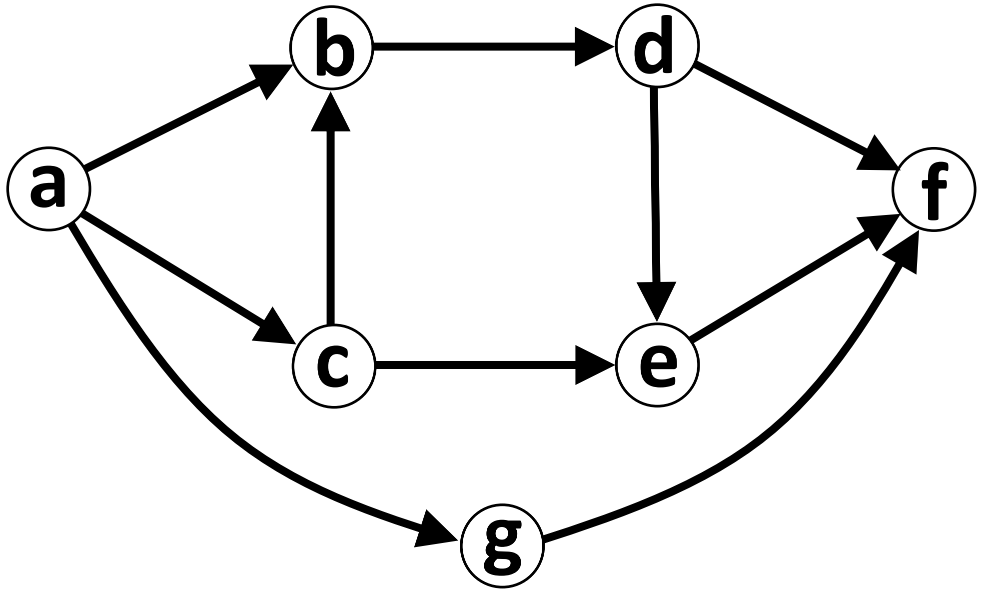
path: f <- e

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.
- Backtrack through array to construct path.

path: f <- e <- d



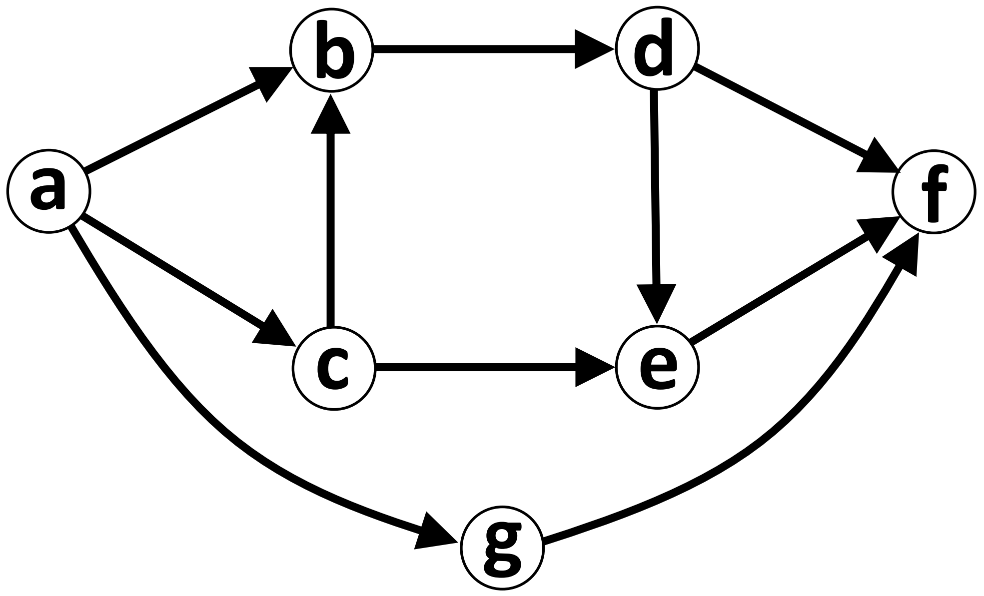
{a, c, g, b, d, e, f}

a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.
- Backtrack through array to construct path.



{a, c, g, b, d, e, f}

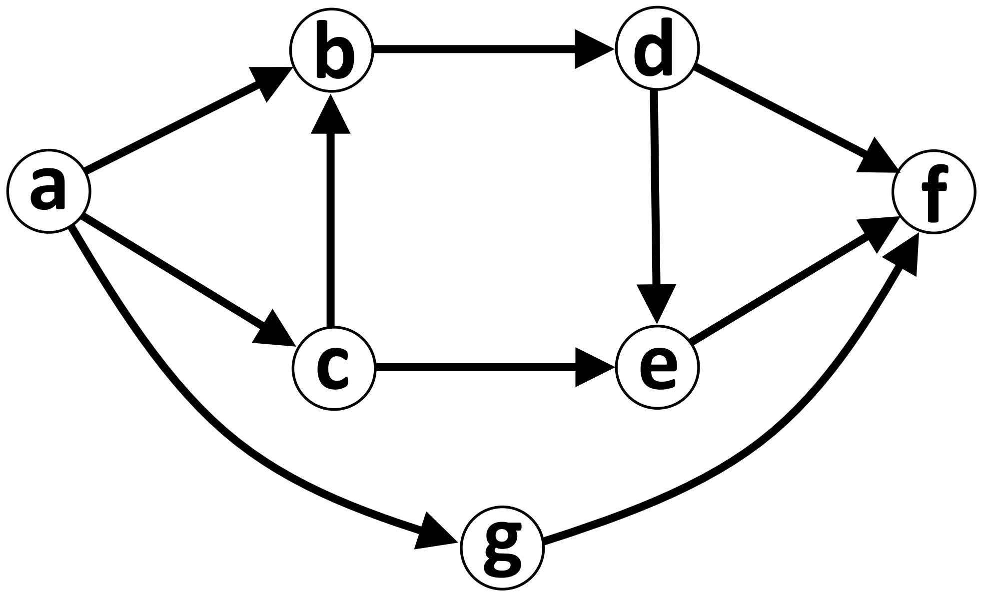
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

path: f <- e <- d <- b

Find the Longest Path in a DAG

Plan:

- Make second array that tracks where longest path came from.
- When neighbor with longest path is determined, save that neighbor.
- Backtrack through array to construct path.



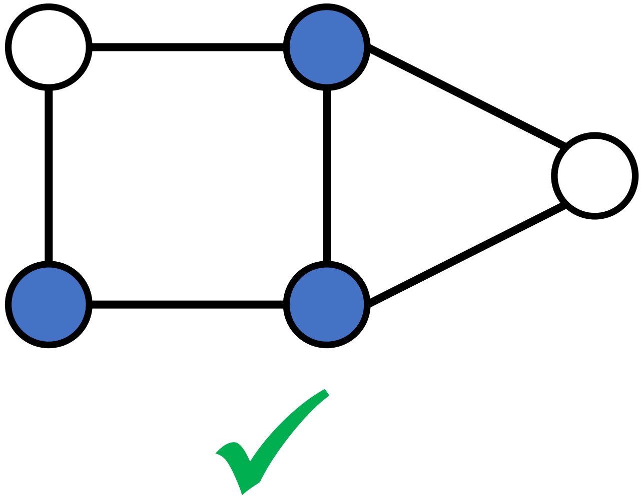
{a, c, g, b, d, e, f}

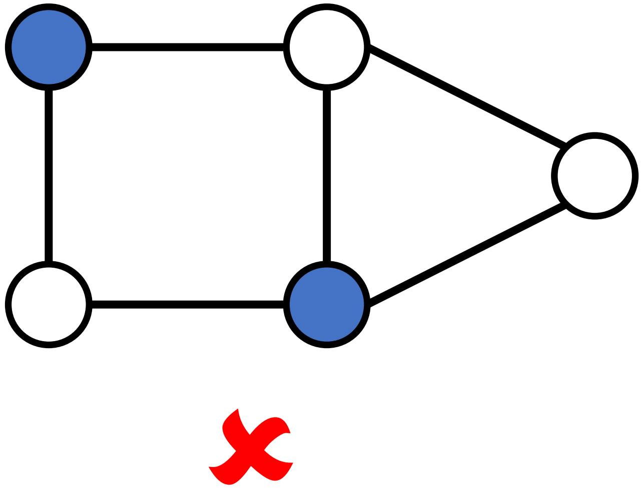
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

path: f <- e <- d <- b <- c <- a

Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.



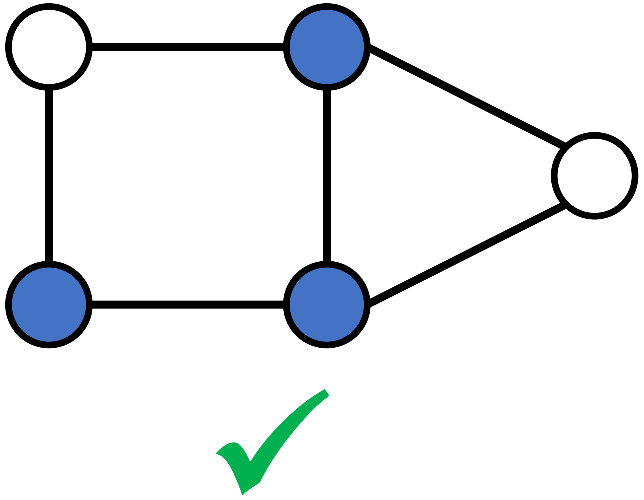


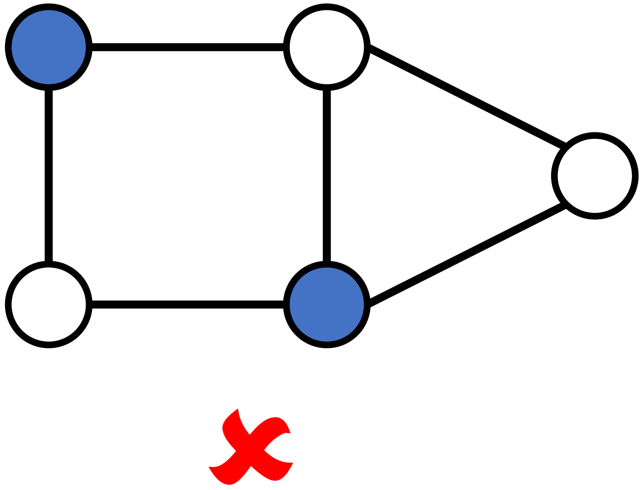
Vertex Cover

tree

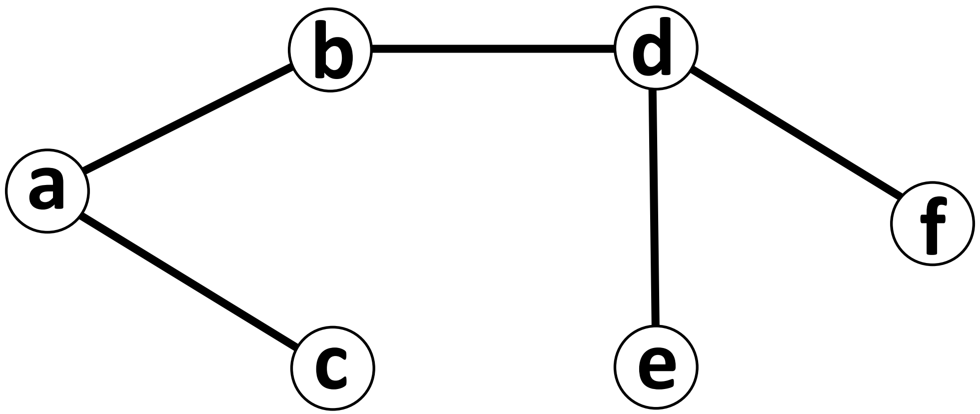
Vertex Cover: Given ~~graph~~, find the smallest subset of vertices such that every edge in the ~~graph~~ has at least one vertex in the subset.

tree





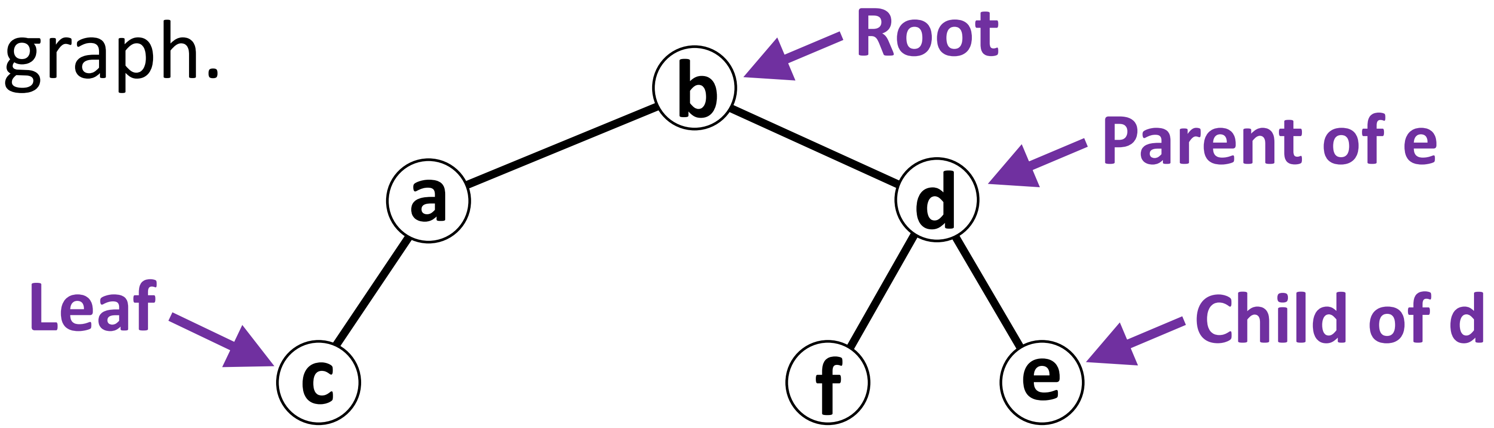
Special Graphs



Vertices (or Nodes)
Edges

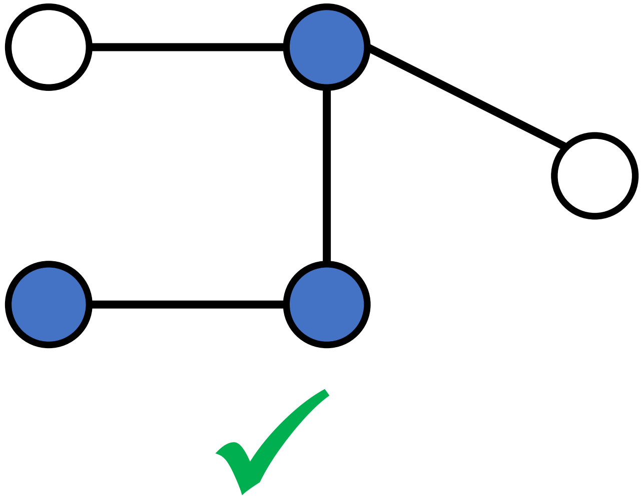
$$\boldsymbol{G = (V, E)}$$

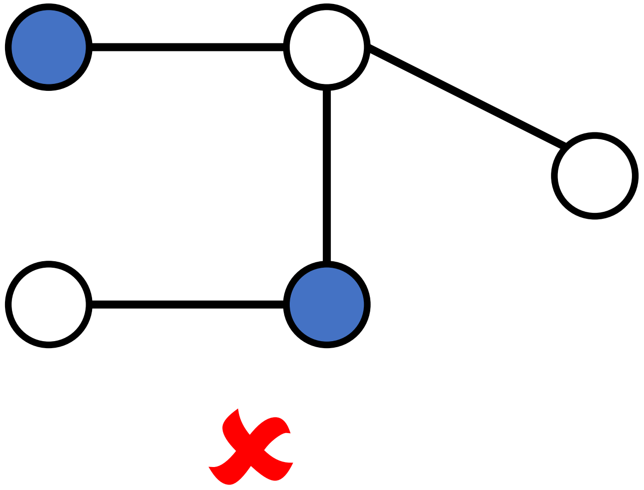
- Connected Graph = Graph that has a path between every vertex pair.
- Acyclic Graph = Graph with no cycles.
- Directed Acyclic Graph (DAG) = Directed graph with no cycles.
- Tree = Connected acyclic graph.



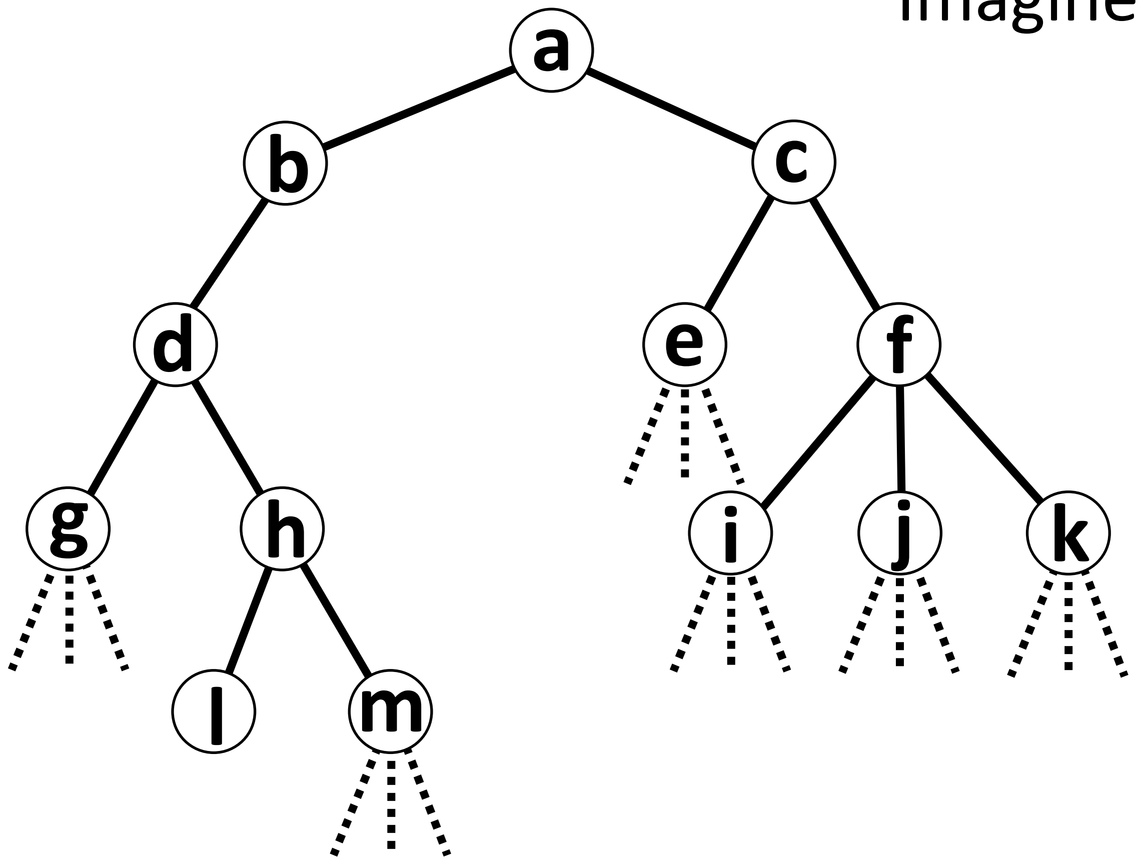
Vertex Cover

Vertex Cover: Given ~~graph~~ **tree**, find the smallest subset of vertices such that every edge in the ~~graph~~ **tree** has at least one vertex in the subset.



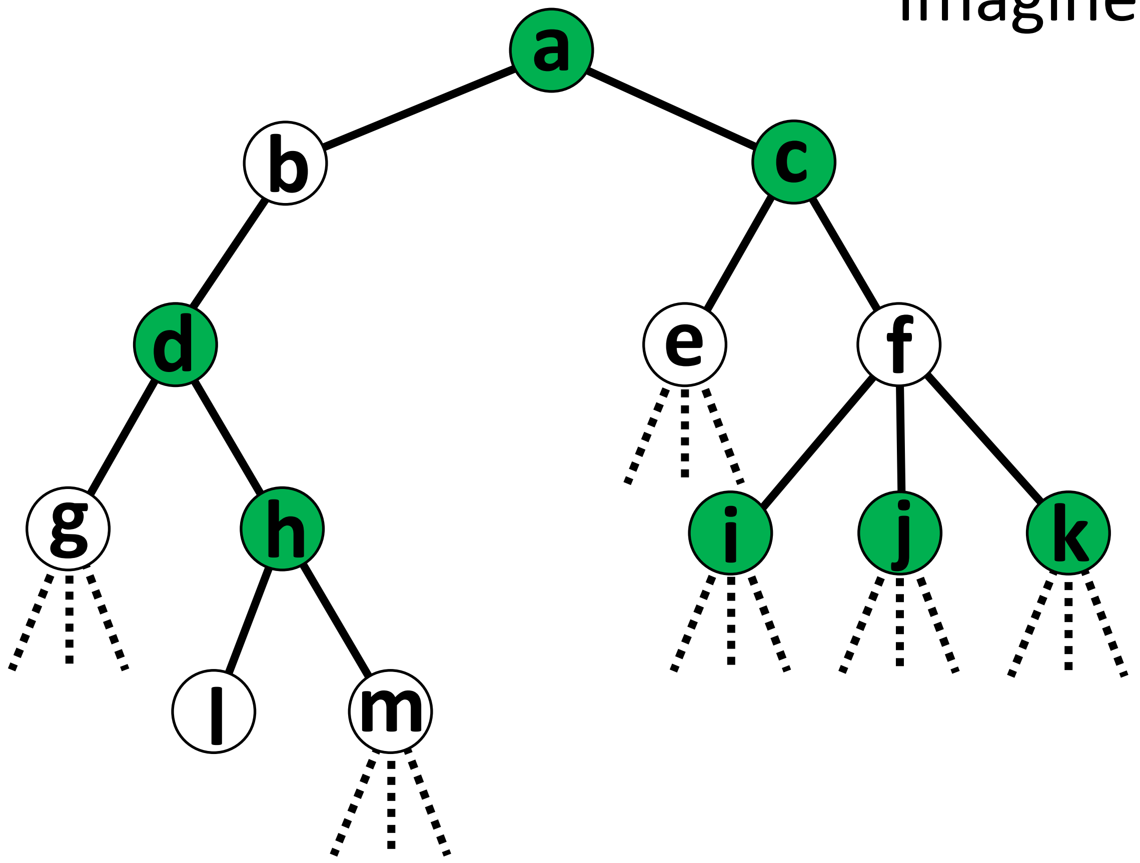


Vertex Cover in Trees



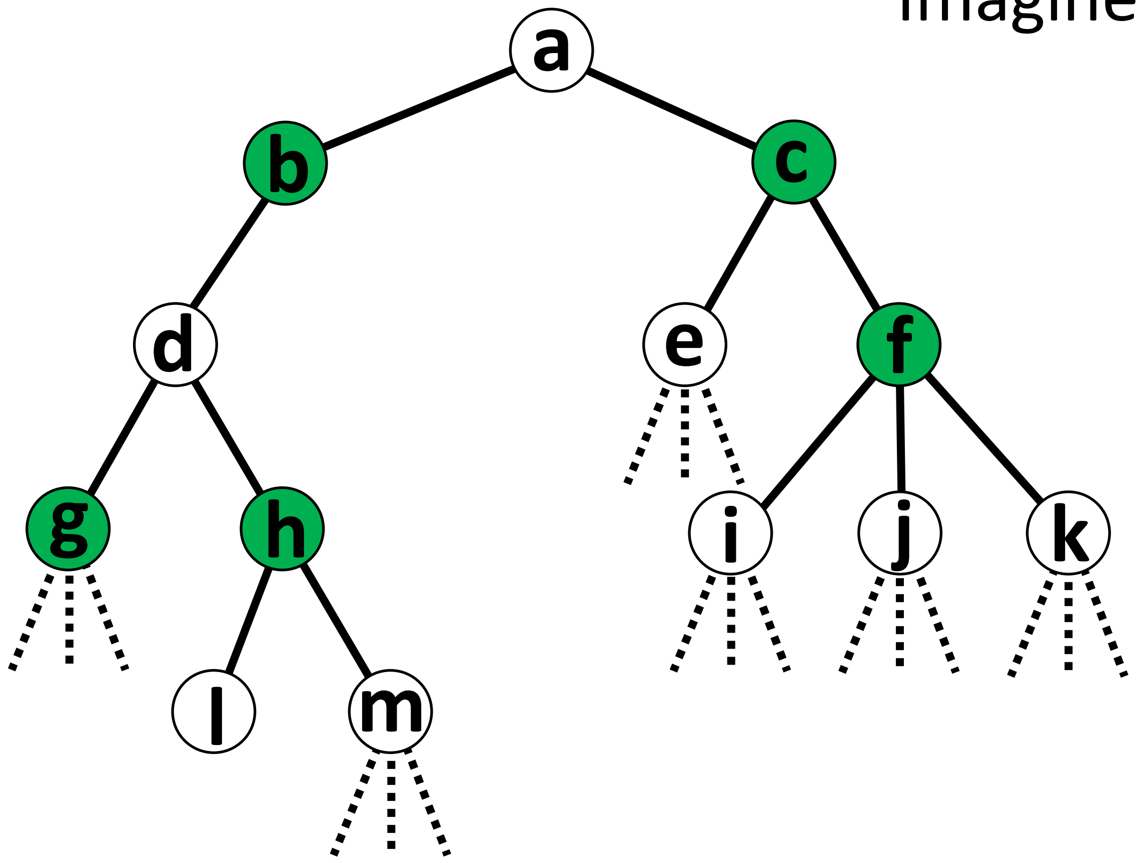
Imagine the minimum vertex cover.

Vertex Cover in Trees



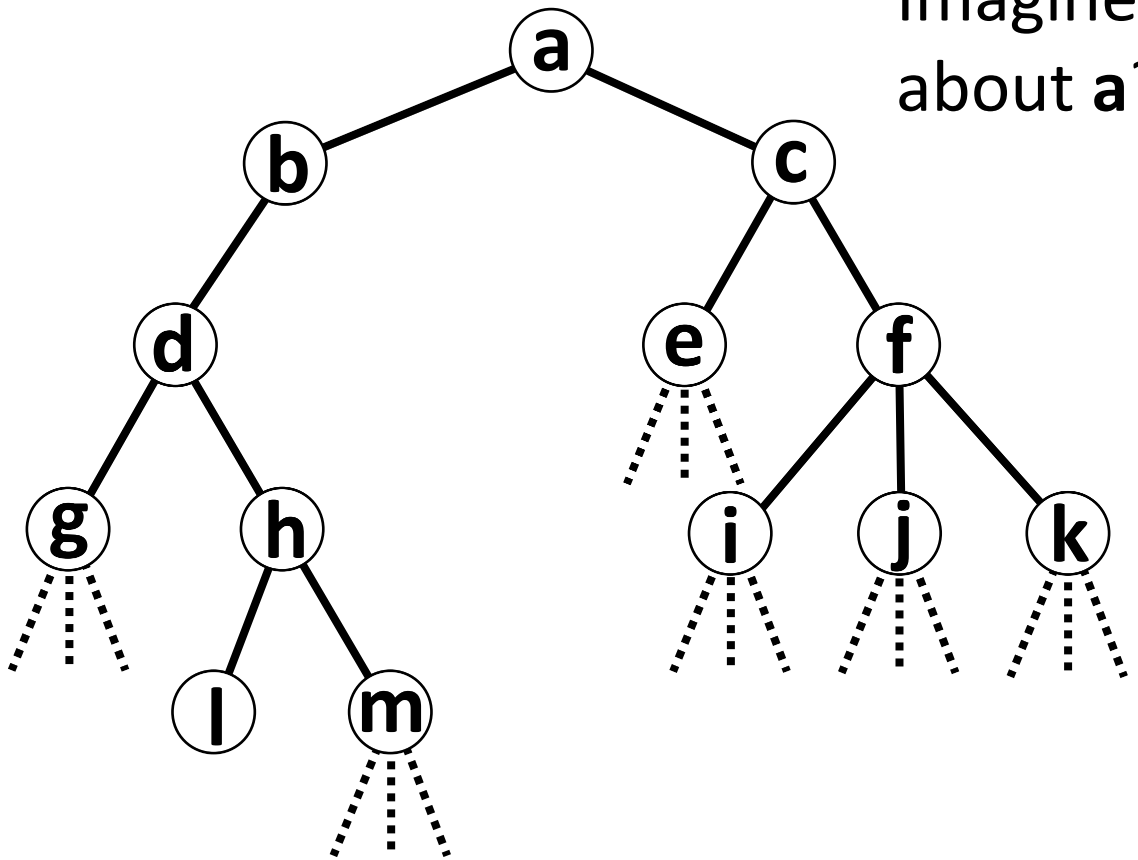
Imagine the minimum vertex cover.

Vertex Cover in Trees



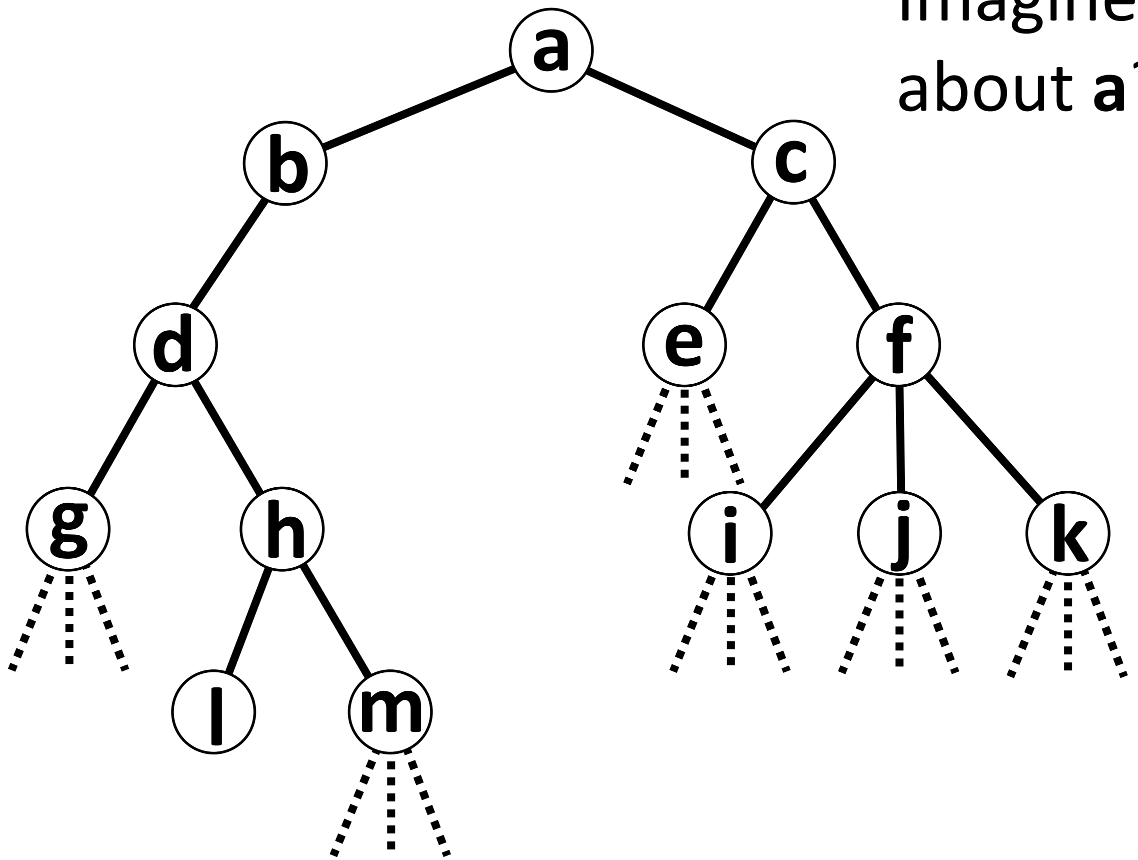
Imagine the minimum vertex cover.

Vertex Cover in Trees



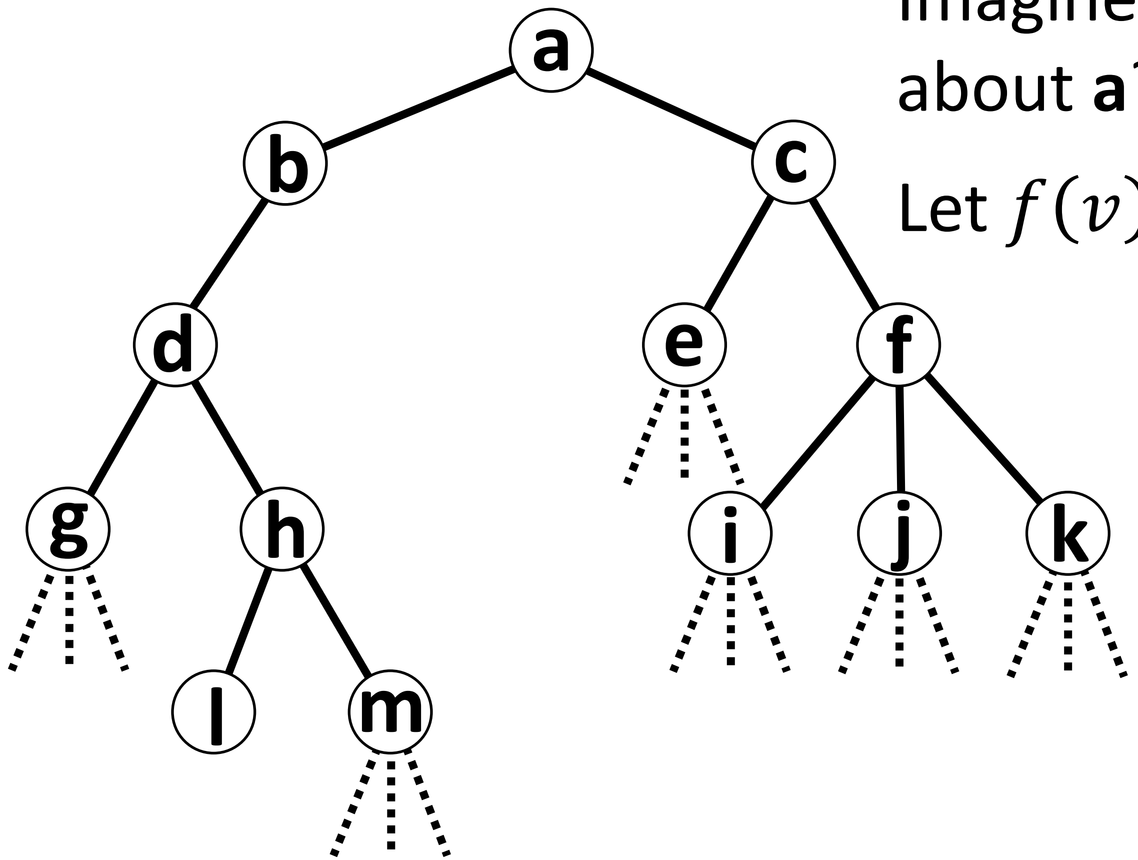
Imagine the minimum vertex cover. What can we say about **a**?

Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

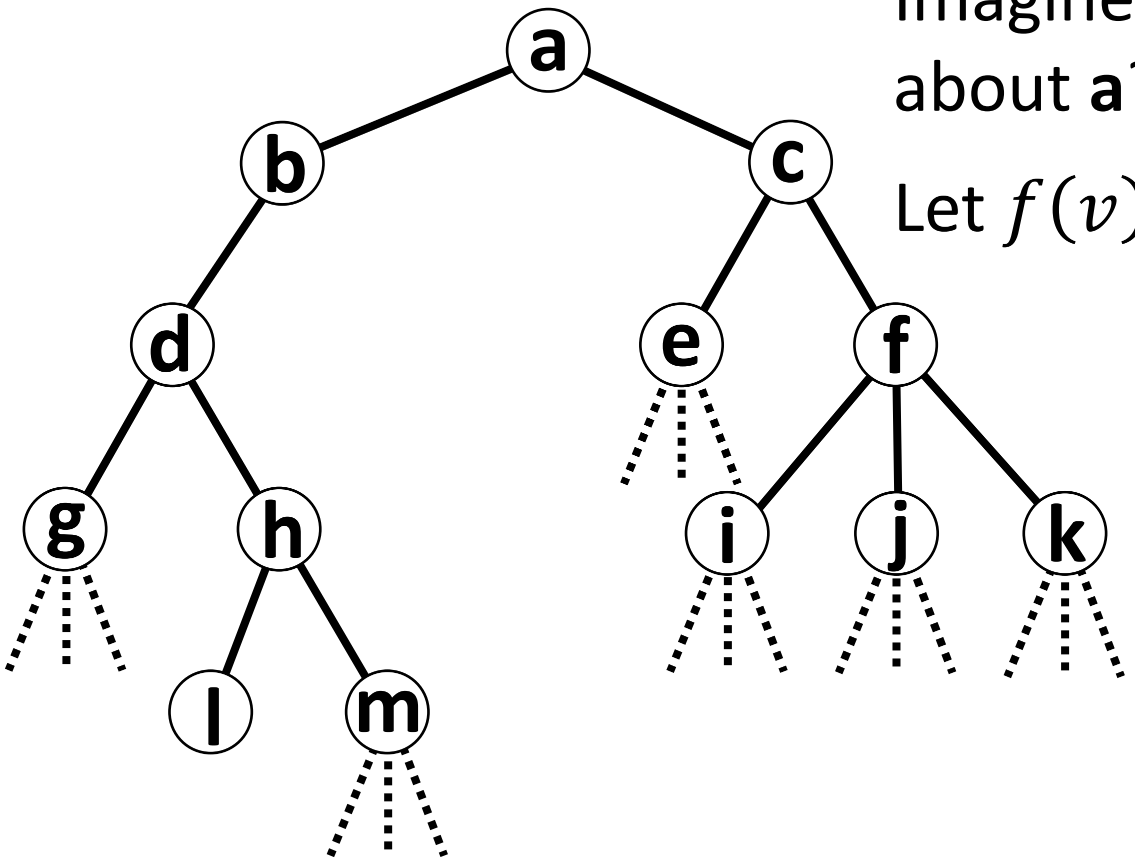
Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v) =$ Size of minimum vertex cover rooted at v.

Vertex Cover in Trees



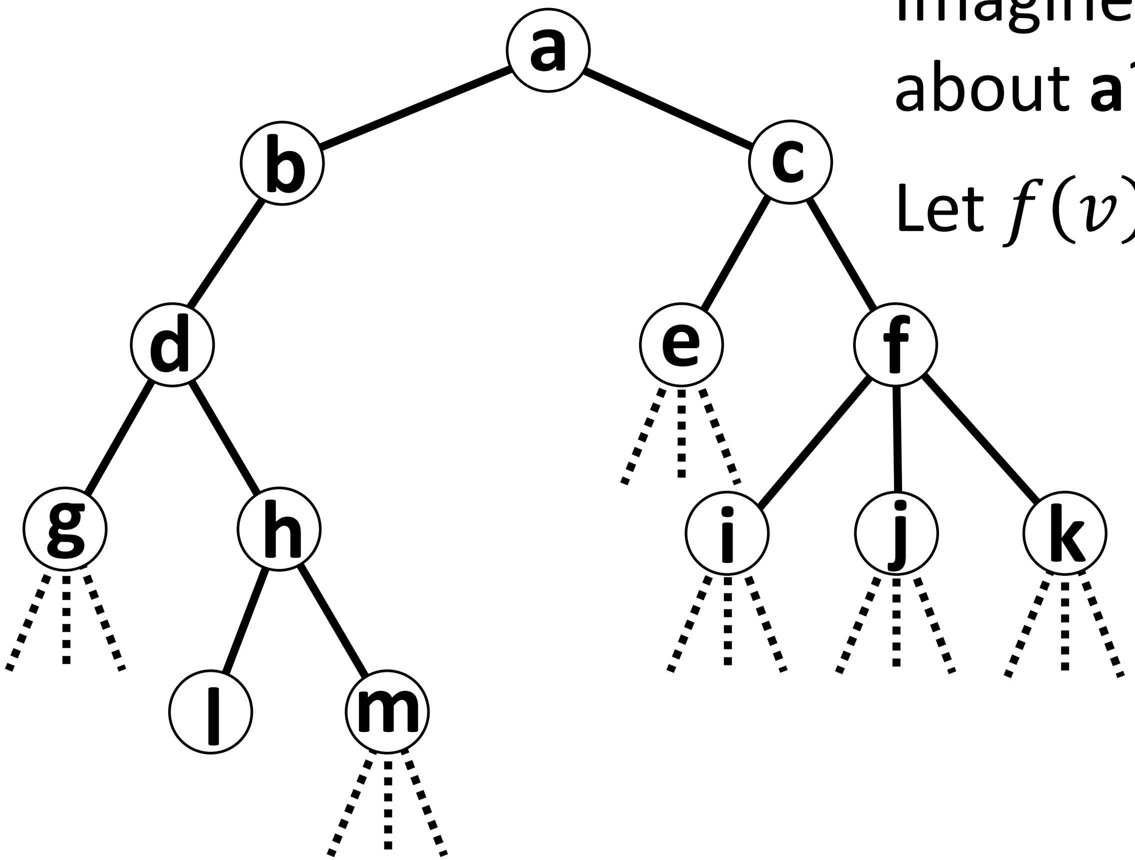
Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v) =$ Size of minimum vertex cover rooted at v.

If a <u>**is**</u> in a minimum VC

If a <u>**is not**</u> in a minimum VC

Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v)$ = Size of minimum vertex cover rooted at v.

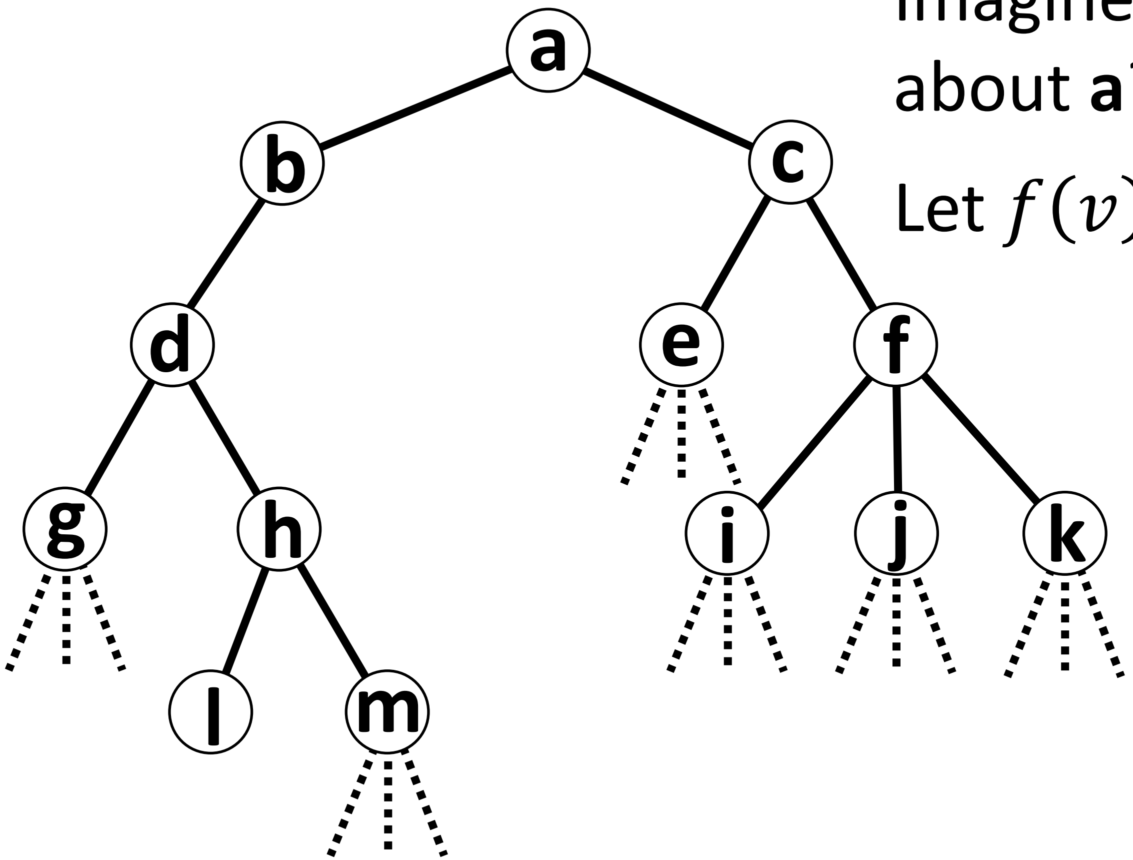
If a **<u>is</u>** in a minimum VC

$f(a) = ??$

If a **<u>is not</u>** in a minimum VC

$f(a) = ??$

Vertex Cover in Trees



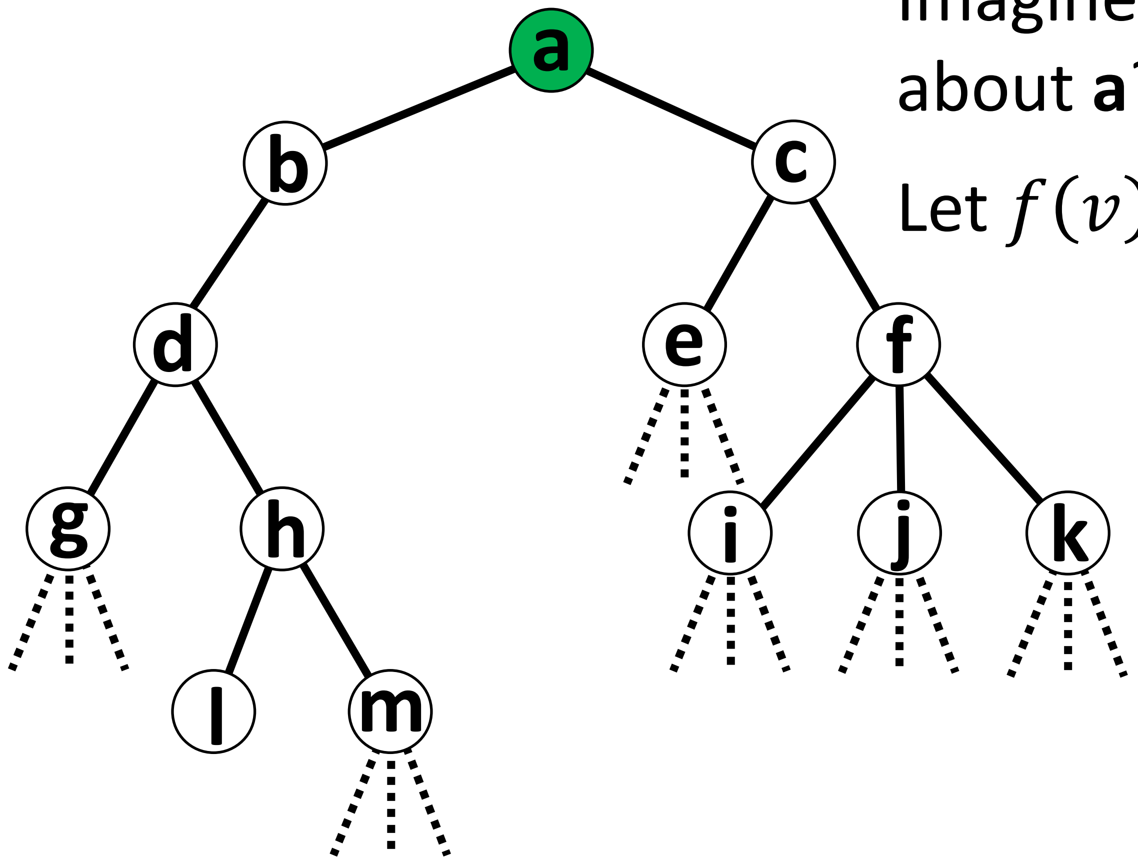
Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v) =$ Size of minimum vertex cover rooted at v.

If a <u>**is**</u> in a minimum VC

$f(a) = ??$

Vertex Cover in Trees



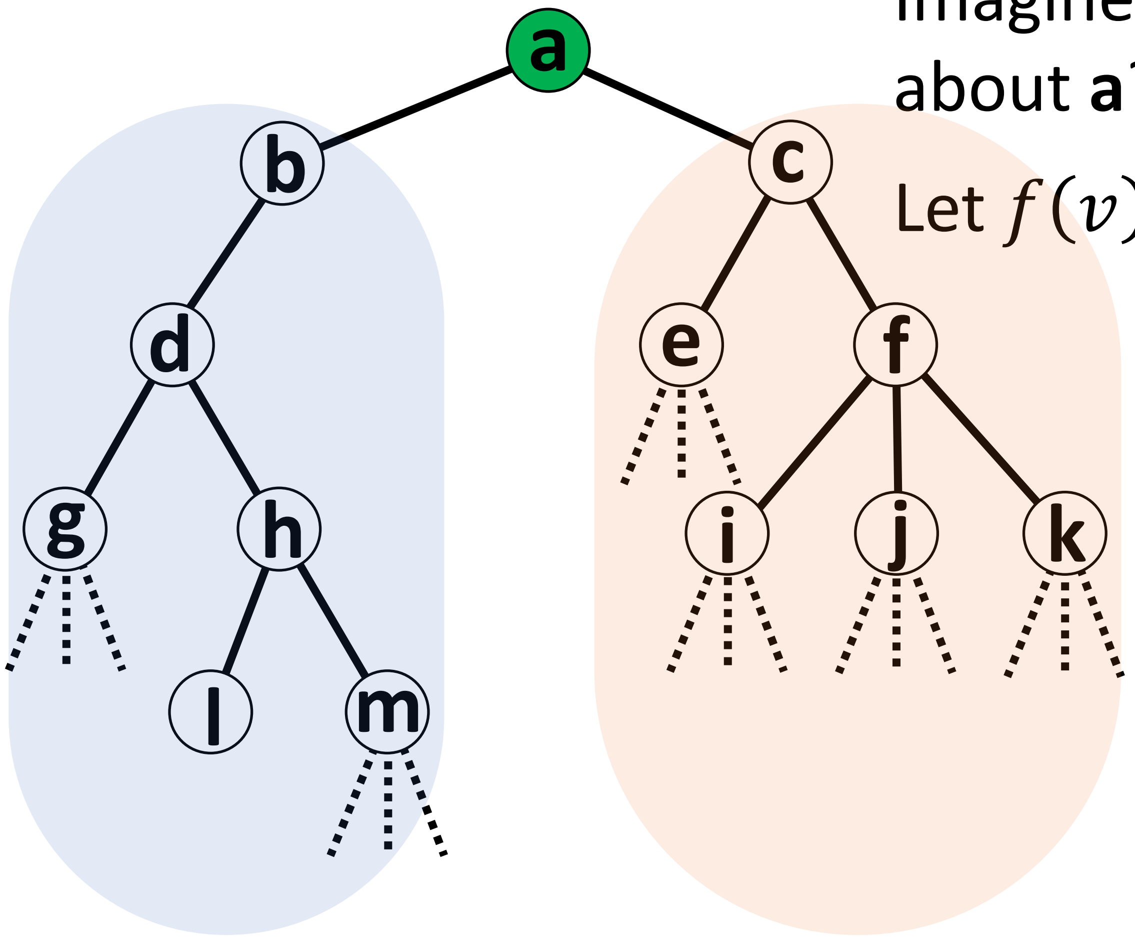
Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v)$ = Size of minimum vertex cover rooted at v.

If a <u>**is**</u> in a minimum VC

$f(a) = 1 + ??$

Vertex Cover in Trees



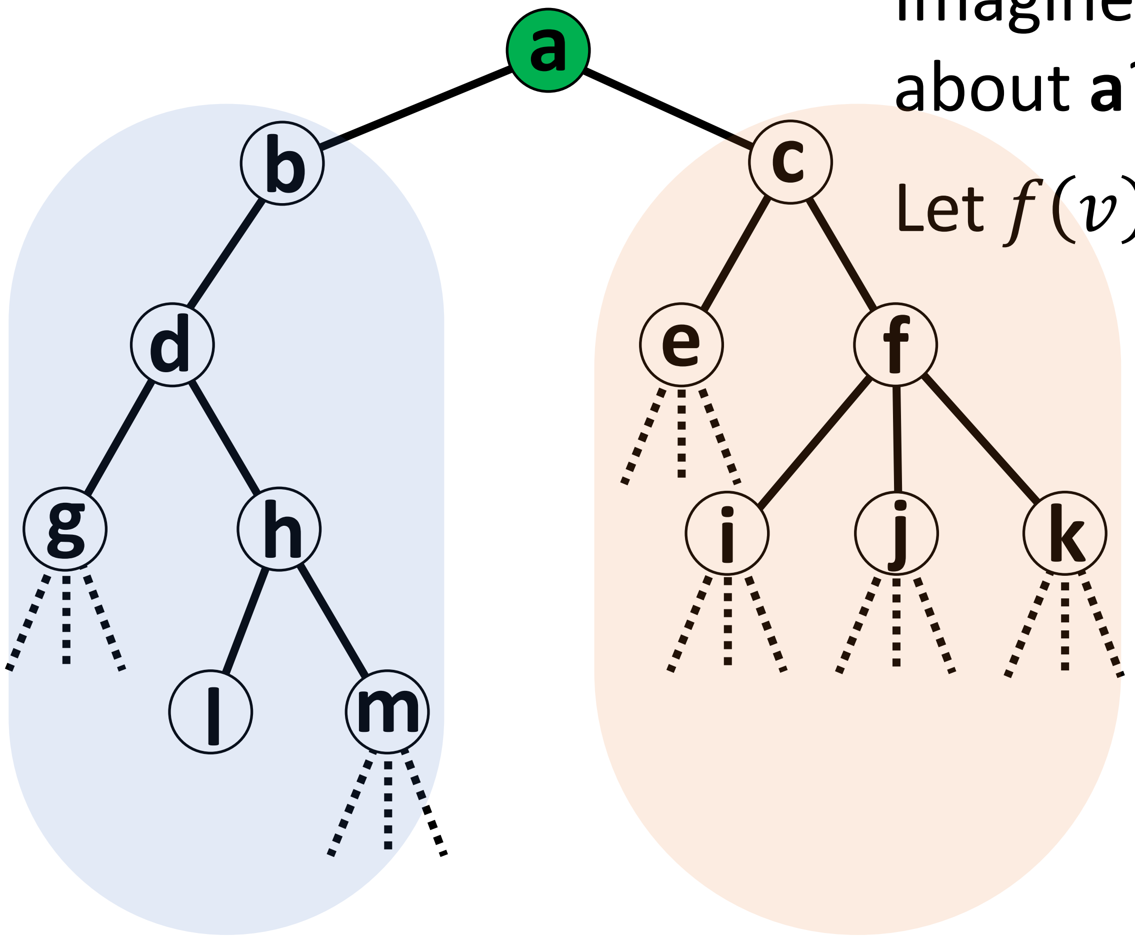
Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v) =$ Size of minimum vertex cover rooted at v.

If a **is** in a minimum VC

$f(a) = 1 + ??$

Vertex Cover in Trees



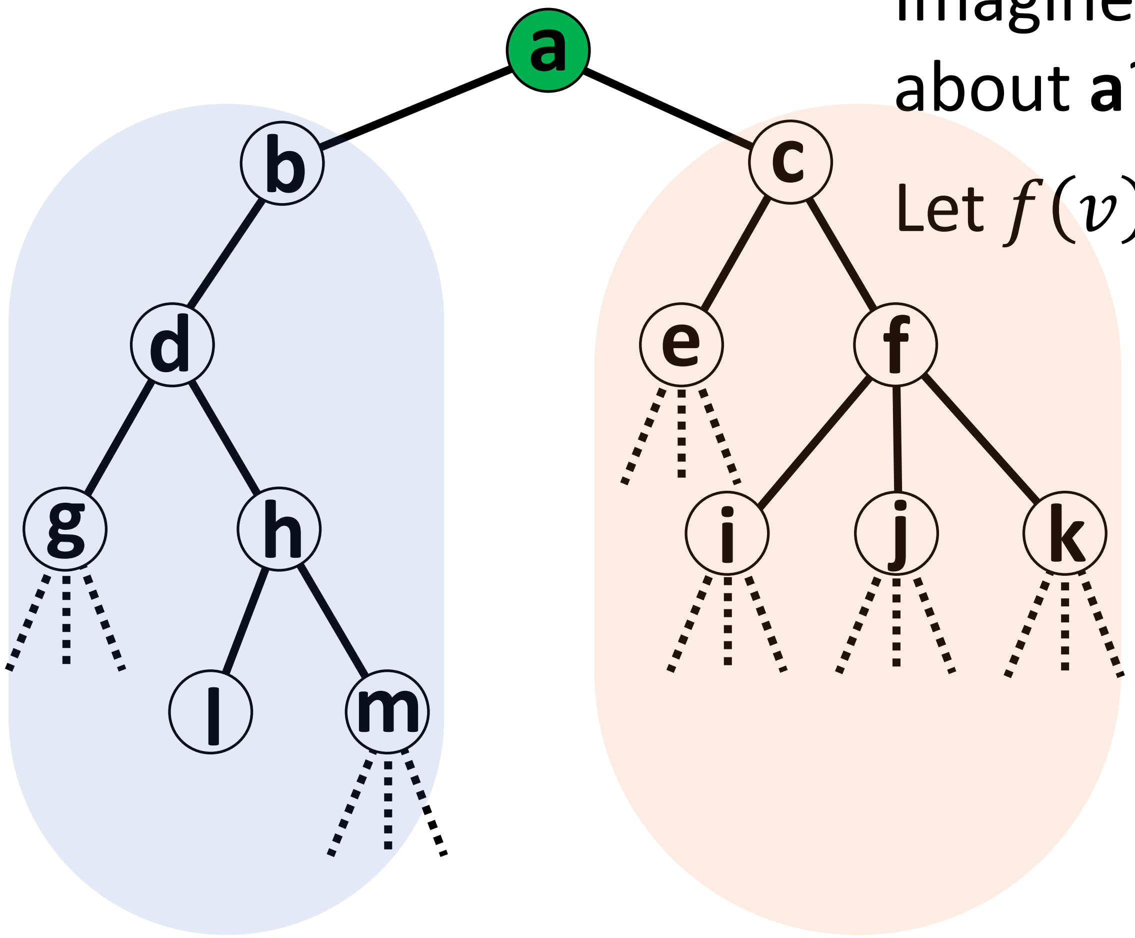
Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v)$ = Size of minimum vertex cover rooted at v.

If a **<u>is</u>** in a minimum VC

$f(a) = 1 + f(b) + f(c)$

Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

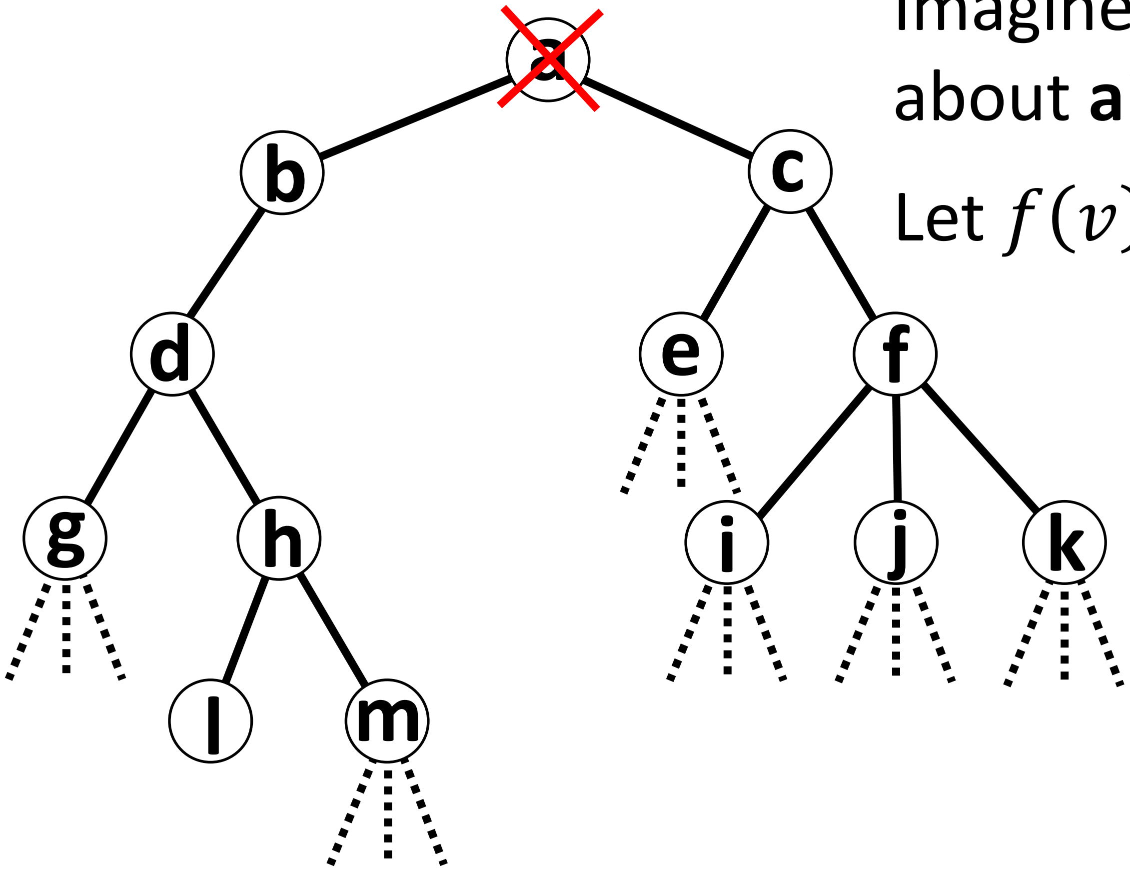
Let $f(v)$ = Size of minimum vertex cover rooted at v.

If a **is** in a minimum VC

$f(a) = 1 + f(b) + f(c)$

*"If there was a smaller VC rooted at **b**, it would give us a smaller VC rooted at **a**."*

Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v)$ = Size of minimum vertex cover rooted at v.

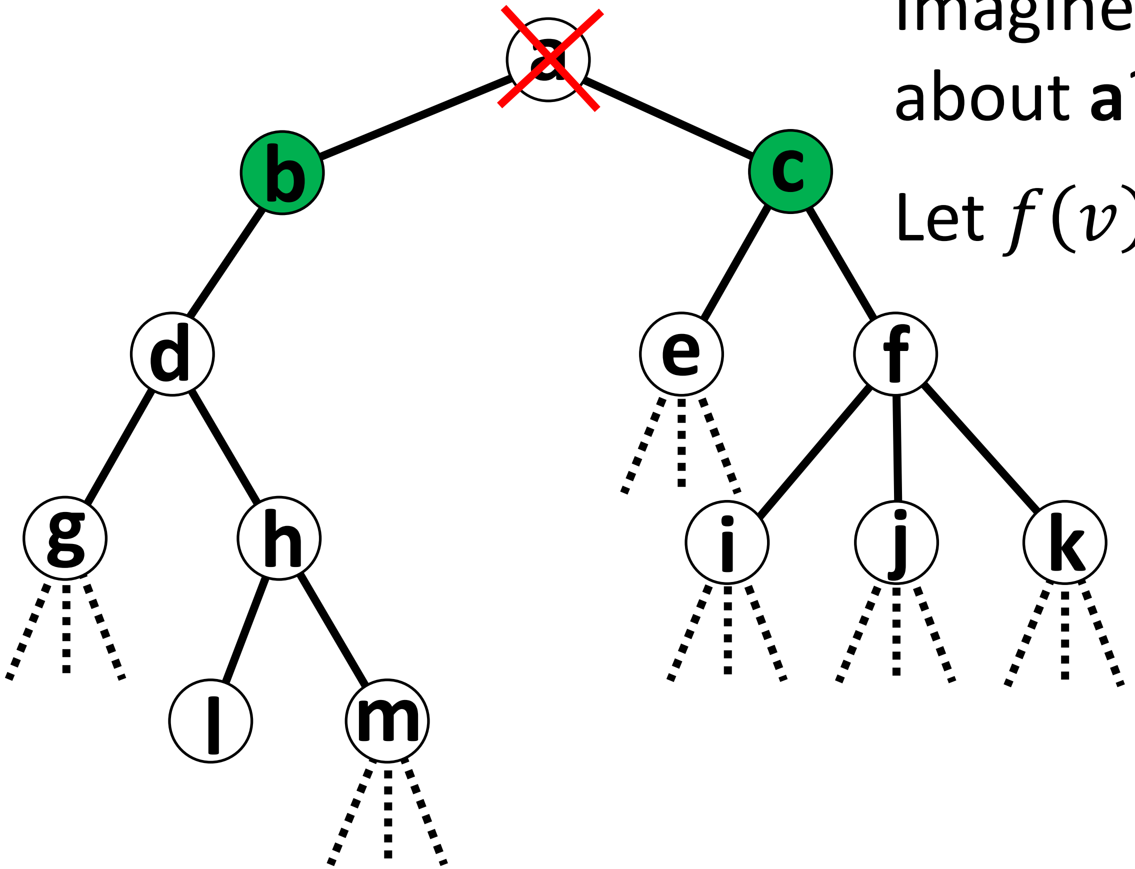
If a **is** in a minimum VC

$$f(a) = 1 + f(b) + f(c)$$

If a **is not** in a minimum VC

$$f(a) = ??$$

Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v)$ = Size of minimum vertex cover rooted at v.

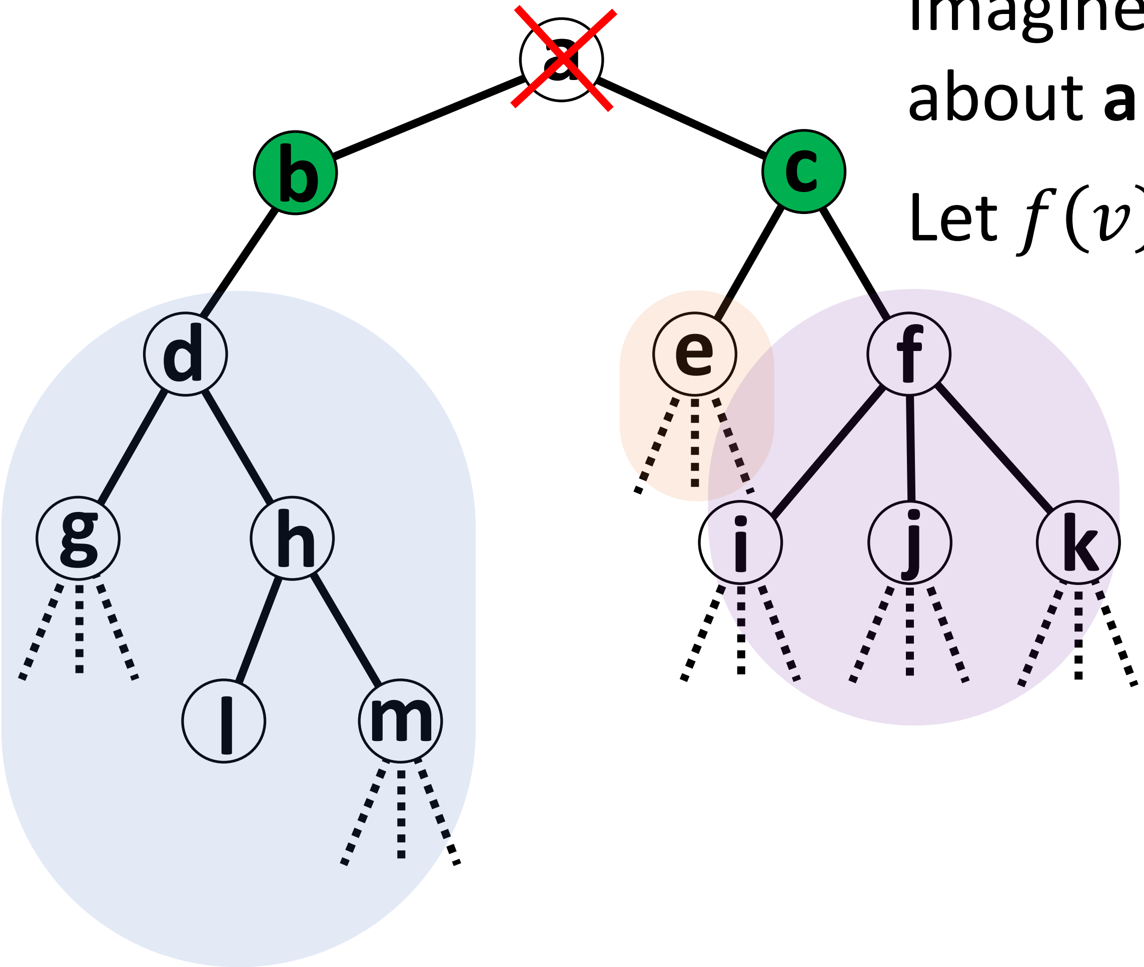
If a **<u>is</u>** in a minimum VC

$f(a) = 1 + f(b) + f(c)$

If a **<u>is not</u>** in a minimum VC

$f(a) = 2 + ??$

Vertex Cover in Trees



Imagine the minimum vertex cover. What can we say about **a**? **a** is in a minimum vertex cover, or it's not.

Let $f(v)$ = Size of minimum vertex cover rooted at v.

If a **<u>is</u>** in a minimum VC

$f(a) = 1 + f(b) + f(c)$

If a **<u>is not</u>** in a minimum VC

$f(a) = 2 + f(d) + f(e) + f(f)$