Longest Path

CSCI 532

Dynamic Programing

Trick-or-treat planning.

Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack.

Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack. The blue house gives you 5 pounds of candy. What can you conclude?

Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack. The blue house gives you 5 pounds of candy. What can you conclude?

The red house and the green house are the fewest stops you need to get 20+ pounds of candy.

Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack. The blue house gives you 5 pounds of candy. What can you conclude?

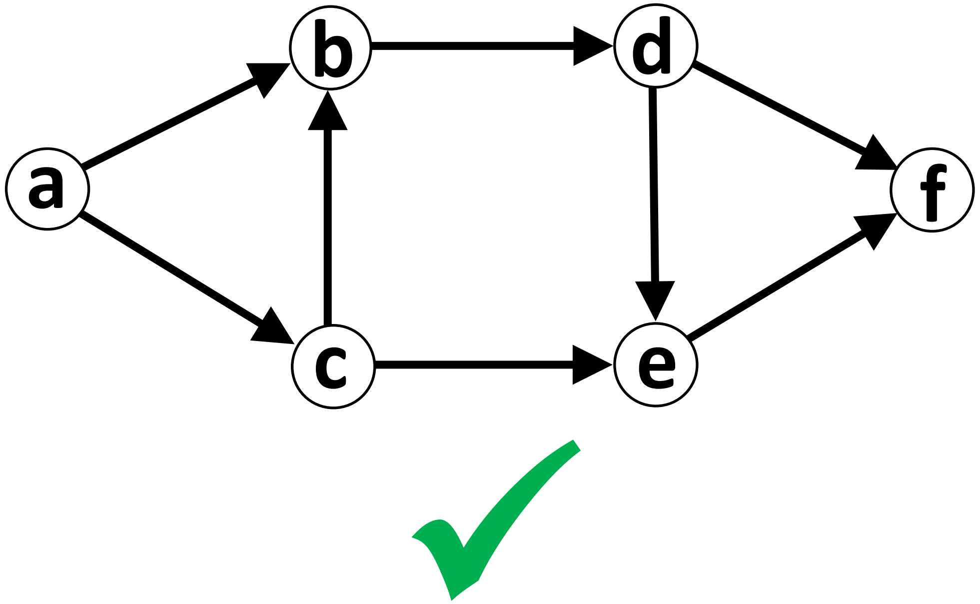
The red house and the green house are the fewest stops you need to get 20+ pounds of candy.

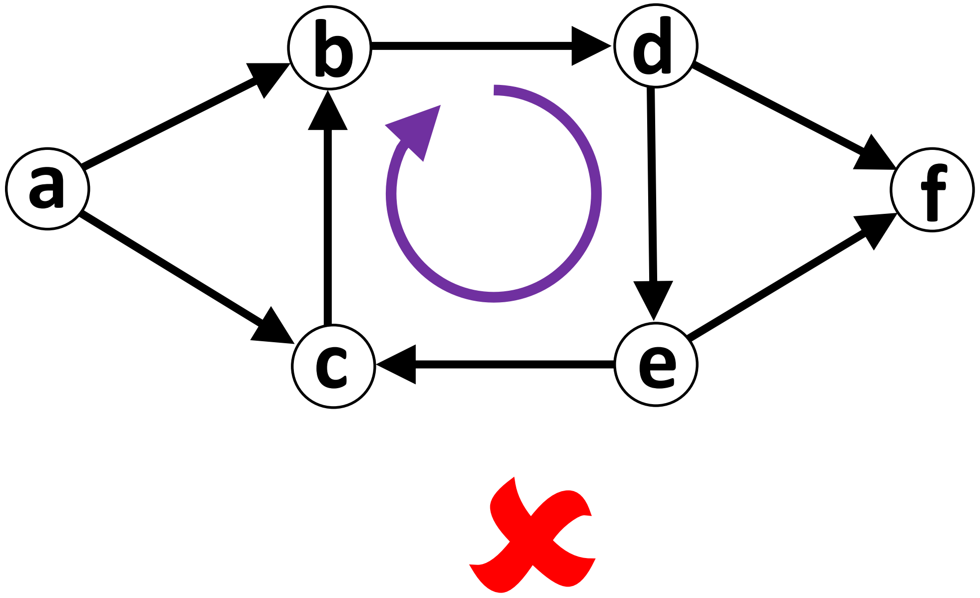
A problem exhibits optimal substructure if removing part of an optimal solution results in an optimal solution to a smaller problem.

Central tenant of Dynamic Programming: Leverage optimal sub-structure.

Directed Acyclic Graph (DAG)

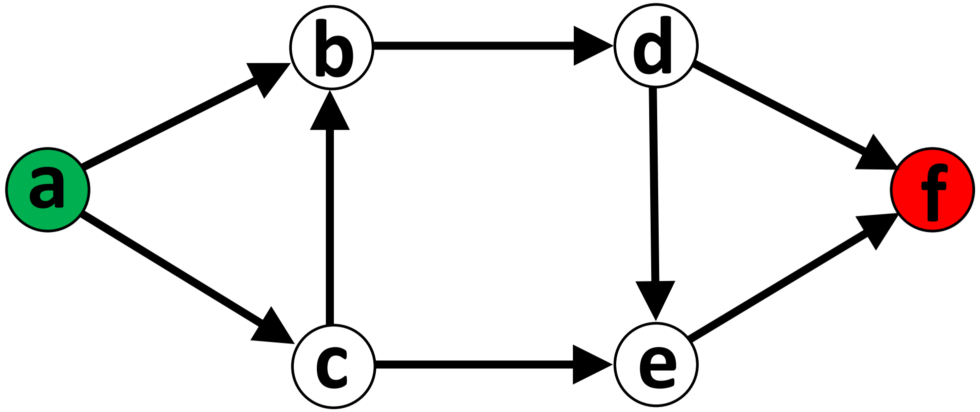
Directed Acyclic Graph (DAG) = Directed graph with no cycles.





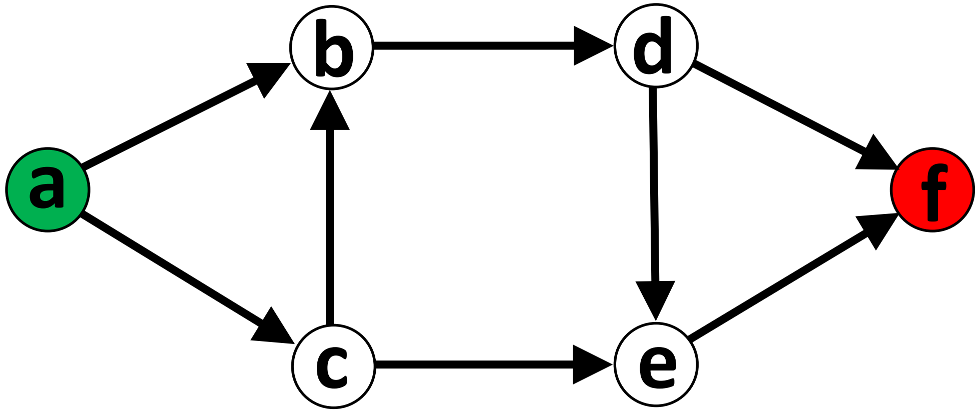
Longest Path in a DAG

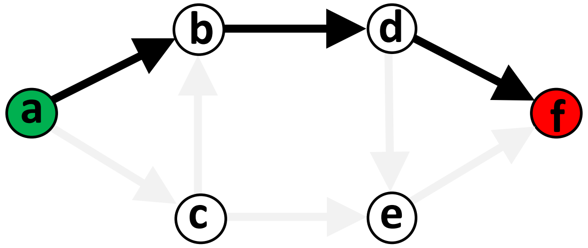
Given a DAG, find the longest path between any two vertices in the graph.



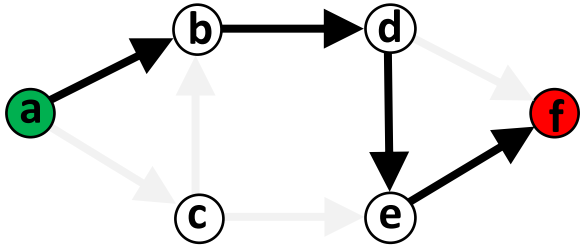
Longest Path in a DAG

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

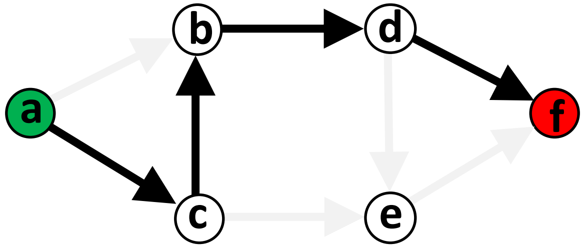




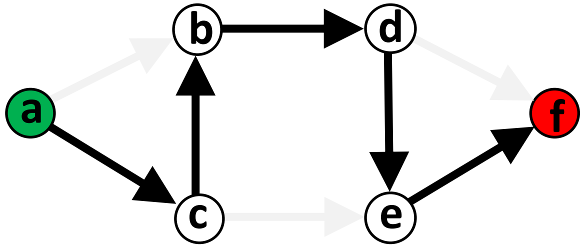
length = 3



length = 4



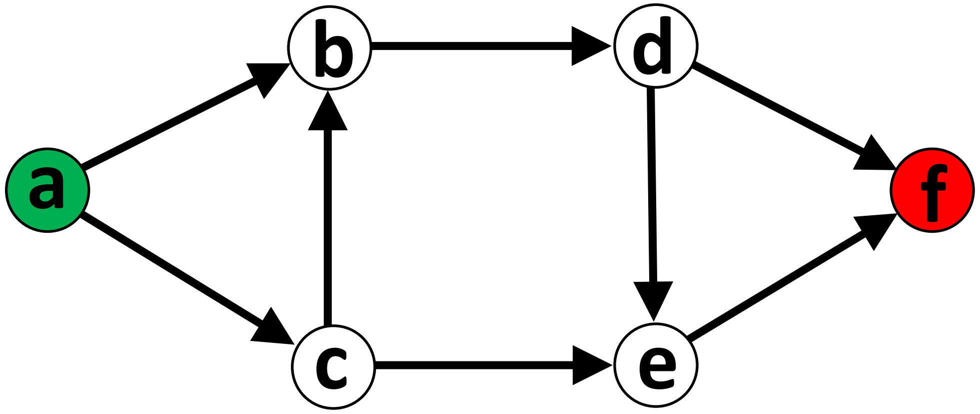
length = 4



length = 5

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day

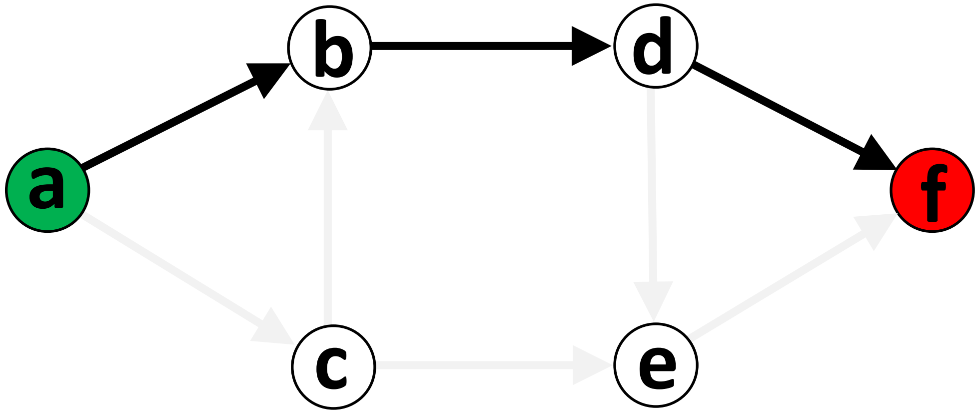


Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day

Length = 2 + 4 + 2 + 1 = 9 days

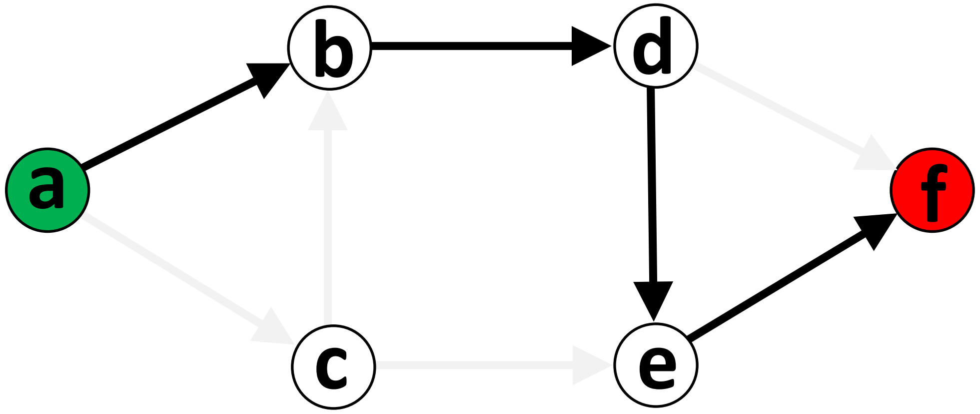


Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day

Length = 2 + 4 + 2 + 1 +1 = 10 days

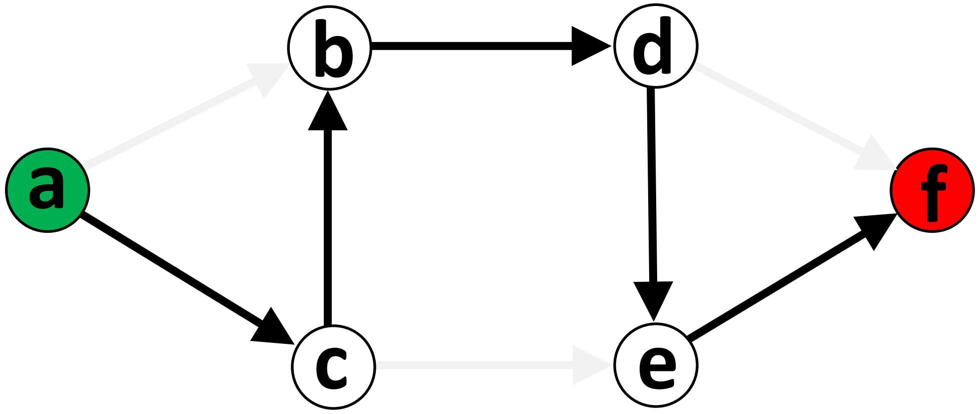


Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day

Length = 2 + 1 + 4 + 2 + 1 +1 = 11 days

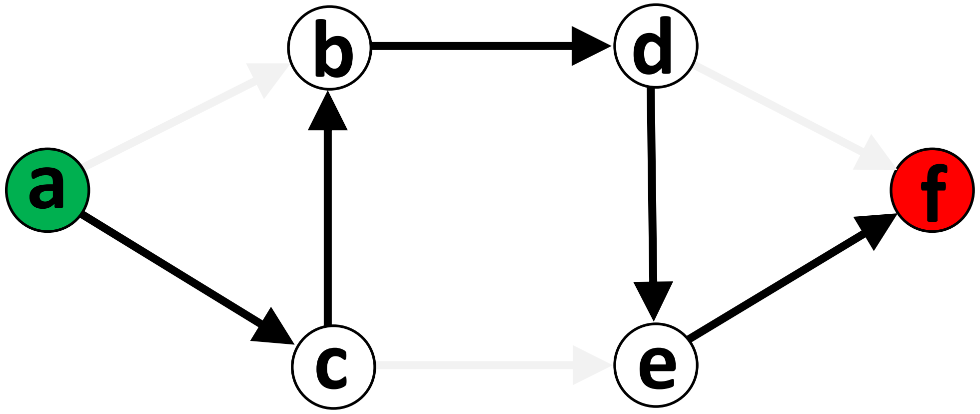


Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day

Length = 2 + 1 + 4 + 2 + 1 +1 = 11 days

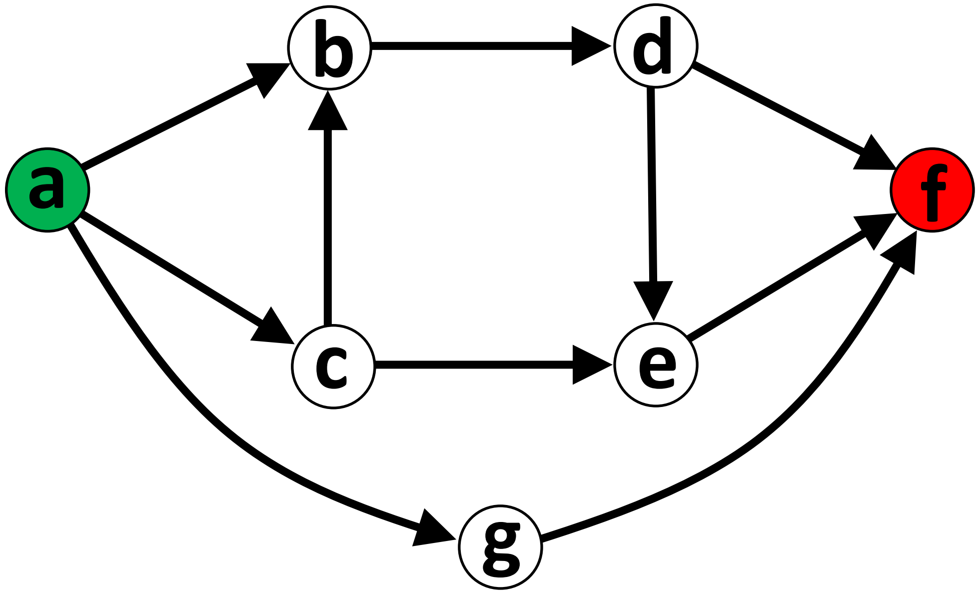


Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

If any task on critical path is delayed, project is delayed.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day
g	Photograph venue	1 day



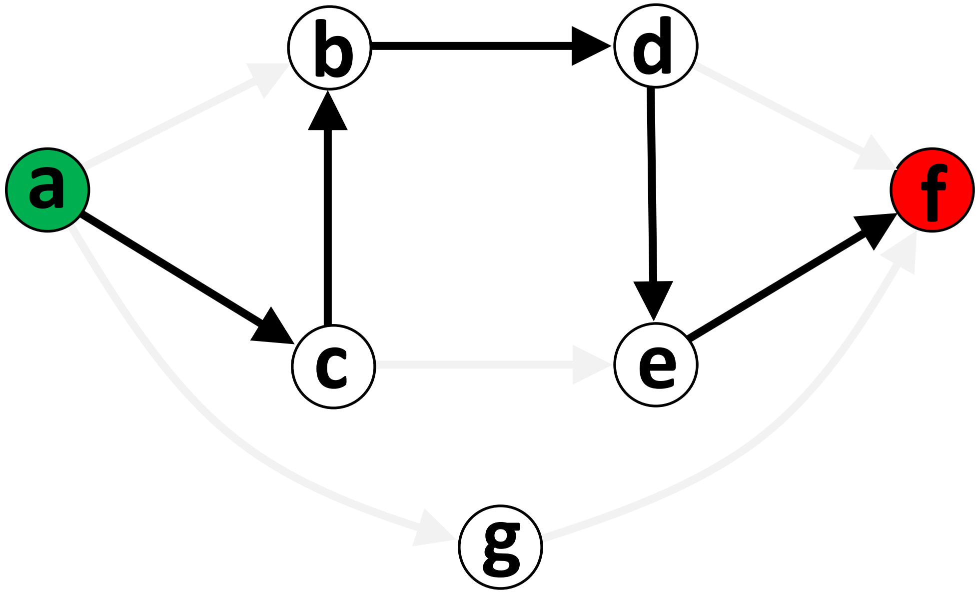
Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

If any task on critical path is delayed, project is delayed.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day
g	Photograph venue	1 day

Length = 2 + 1 + 4 + 2 + 1 +1 = 11 days



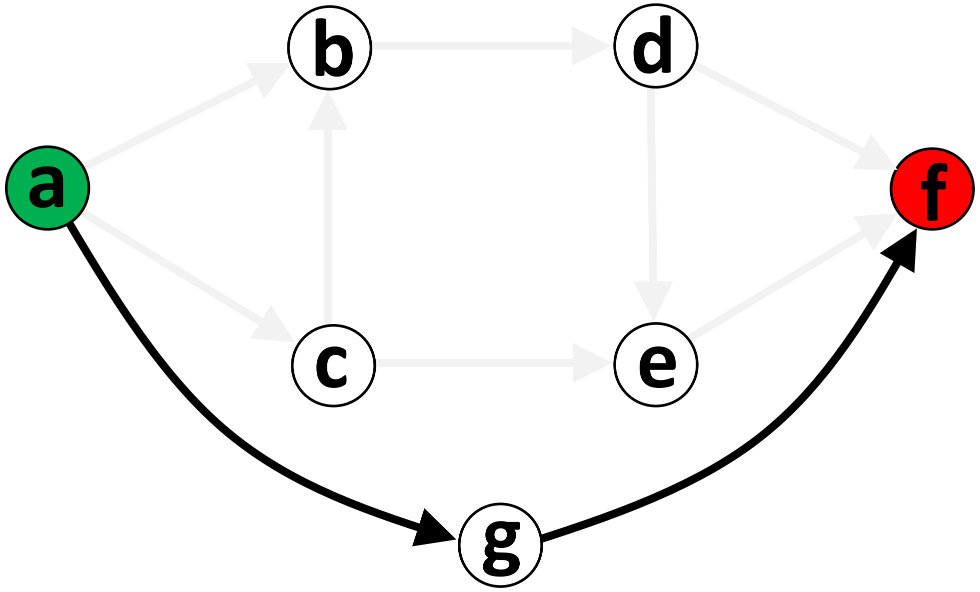
Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

If any task on critical path is delayed, project is delayed.

Find the Longest Path in a DAG

Task	Description	Duration
a	Select location	2 days
b	Get permits	4 days
c	Select date/time	1 day
d	Hire vendors	2 days
e	Make flyers	1 day
f	Market event	1 day
g	Photograph venue	**10 days**

Length = 2 + 10 +1 = 13 days



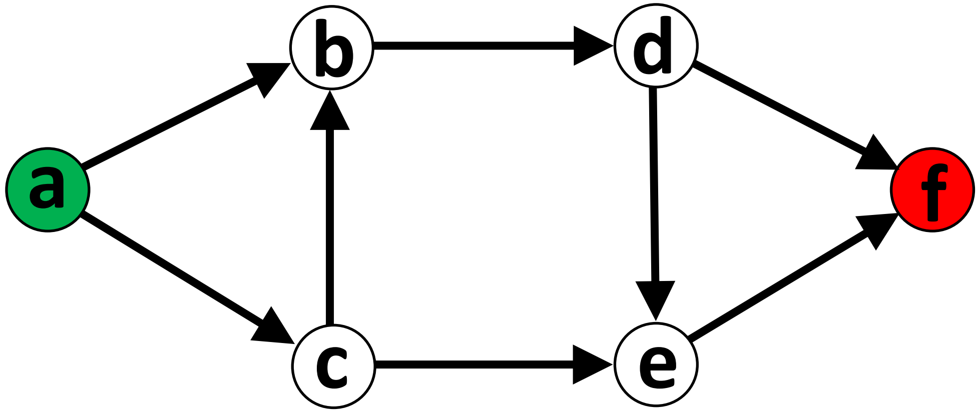
Critical Path: Sequence of dependent tasks that determines the minimum time to complete project.

If any task on critical path is delayed, project is delayed.

Find the Longest Path in a DAG

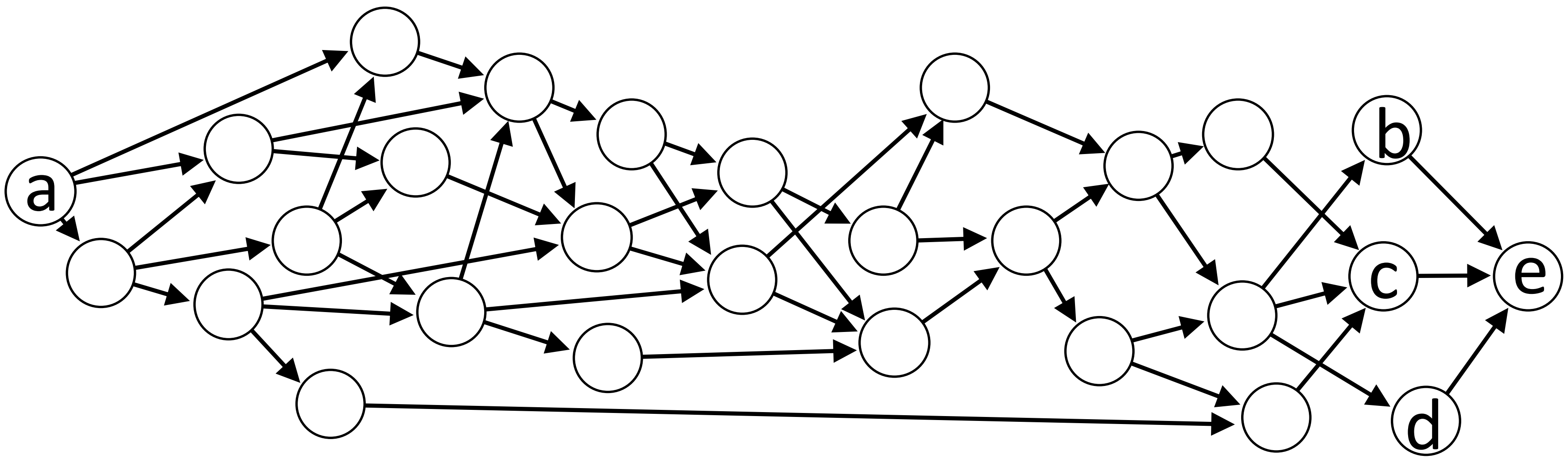
Number of edges.
No vertex weights.

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



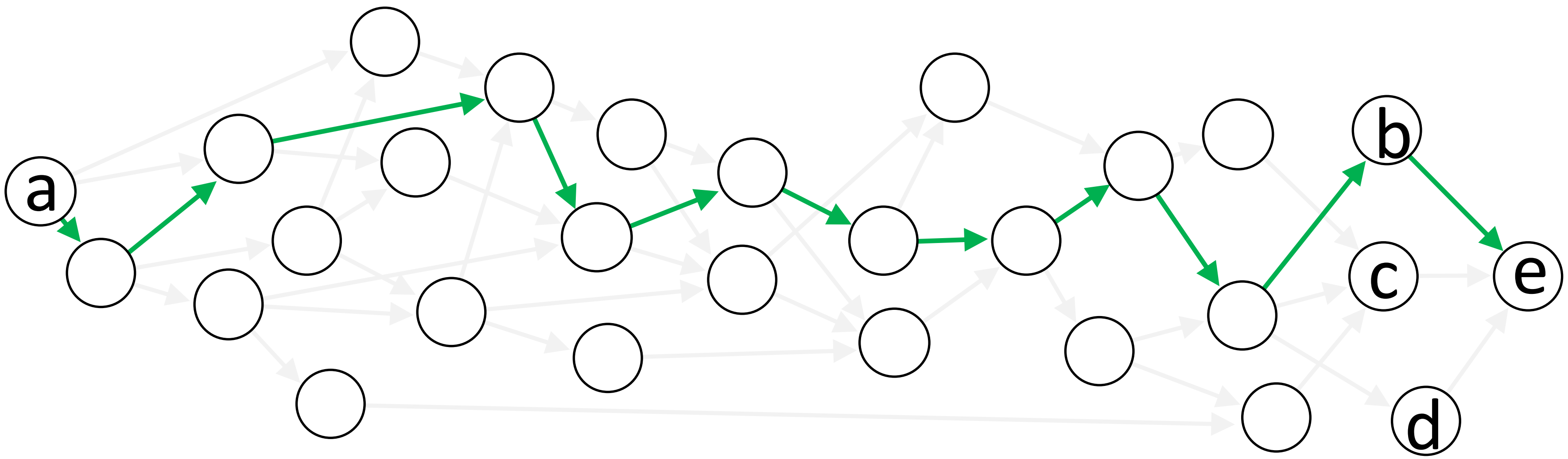
How do we do this?

Find the Longest Path in a DAG



Interesting observations?

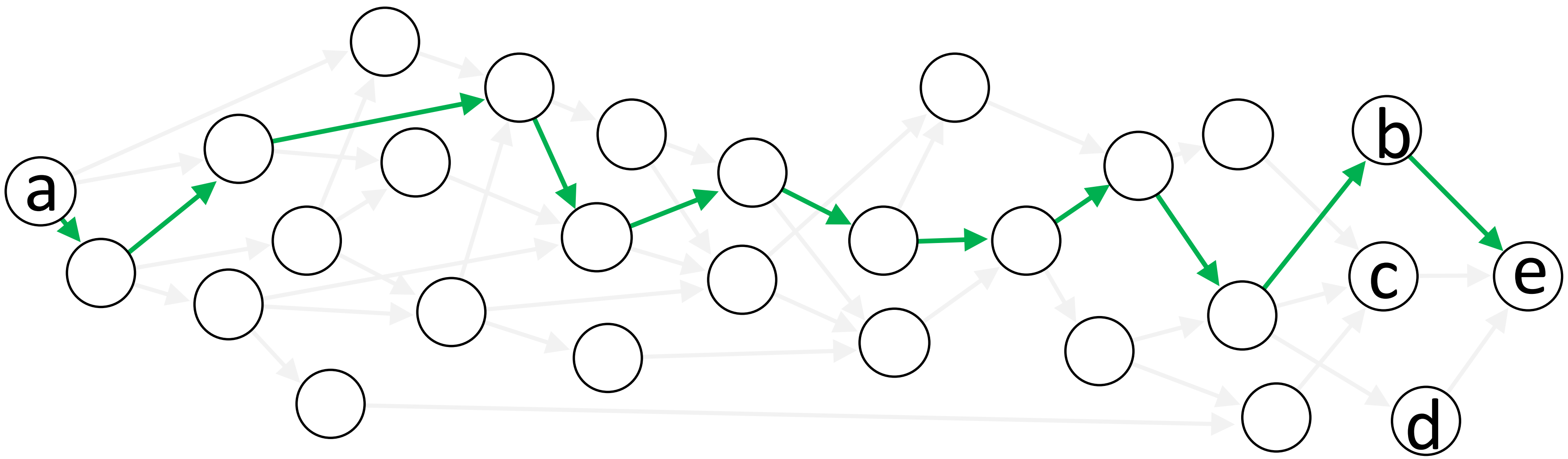
Find the Longest Path in a DAG



Interesting observations?

If the longest path goes from **a** to **e** and passes through **b**, what could we say about the part of that path to **b**?

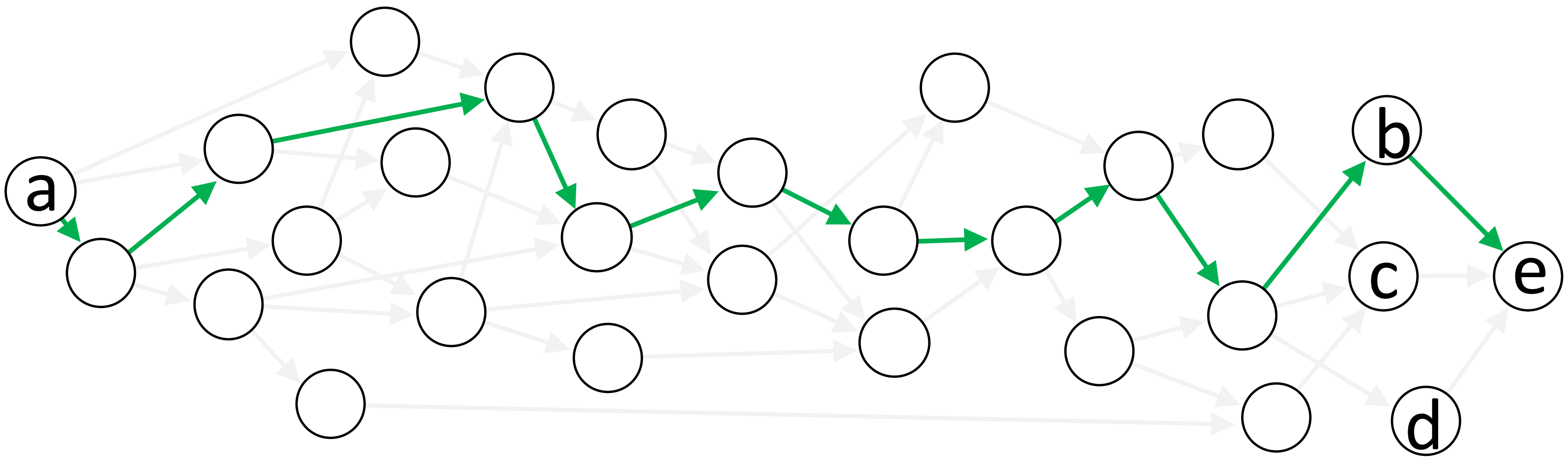
Find the Longest Path in a DAG



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**.

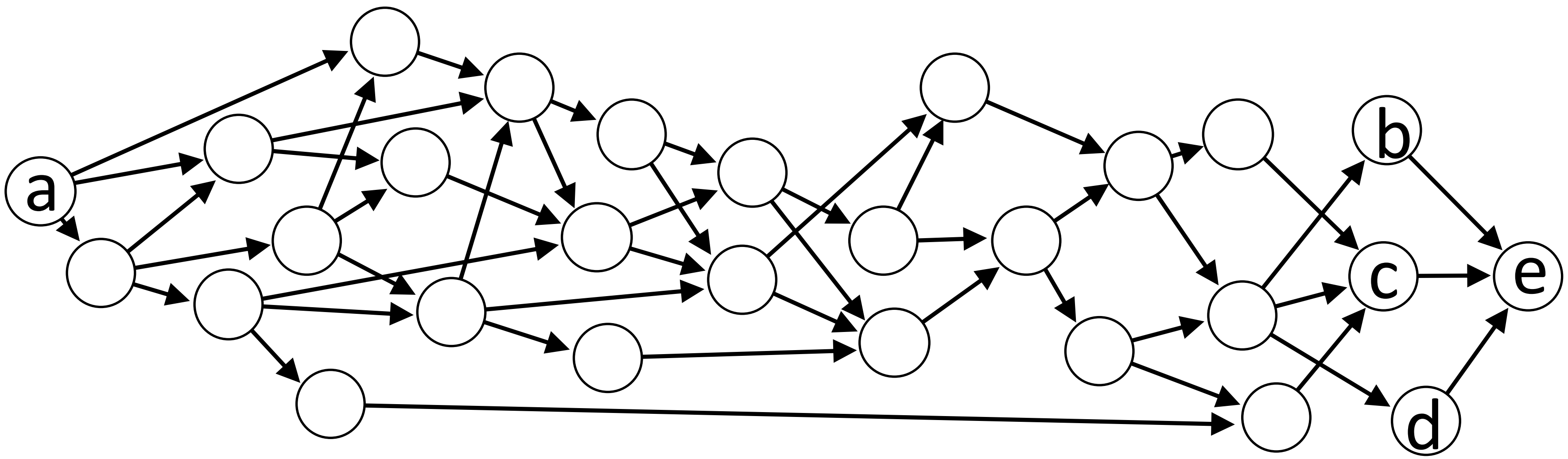
Find the Longest Path in a DAG



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.

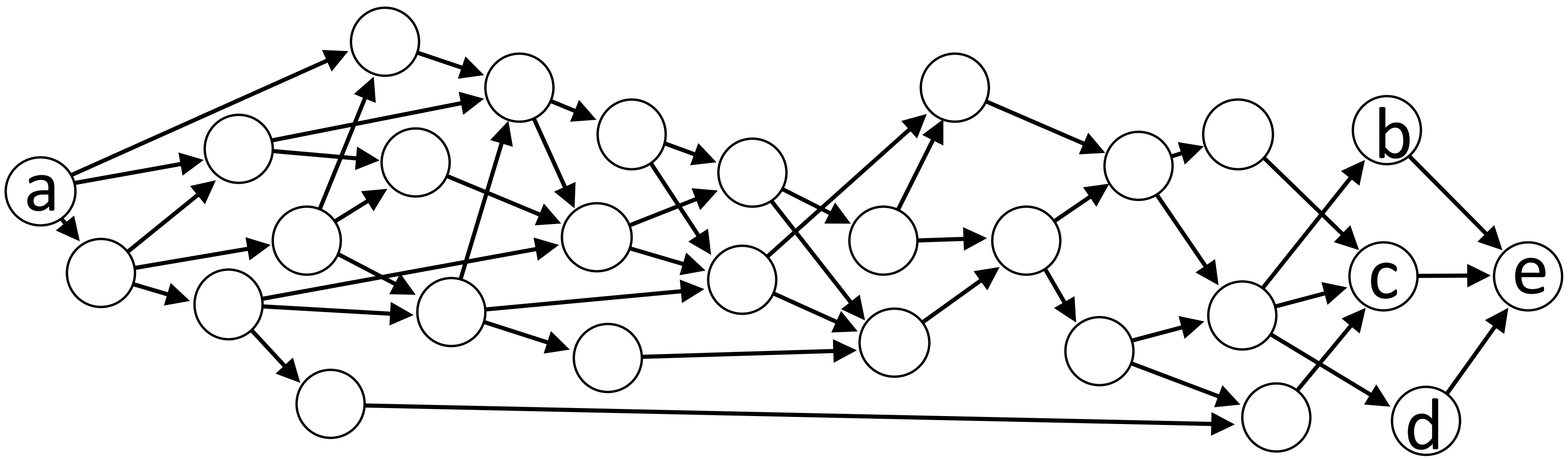
Find the Longest Path in a DAG



Interesting observations?

The longest path to e = ??

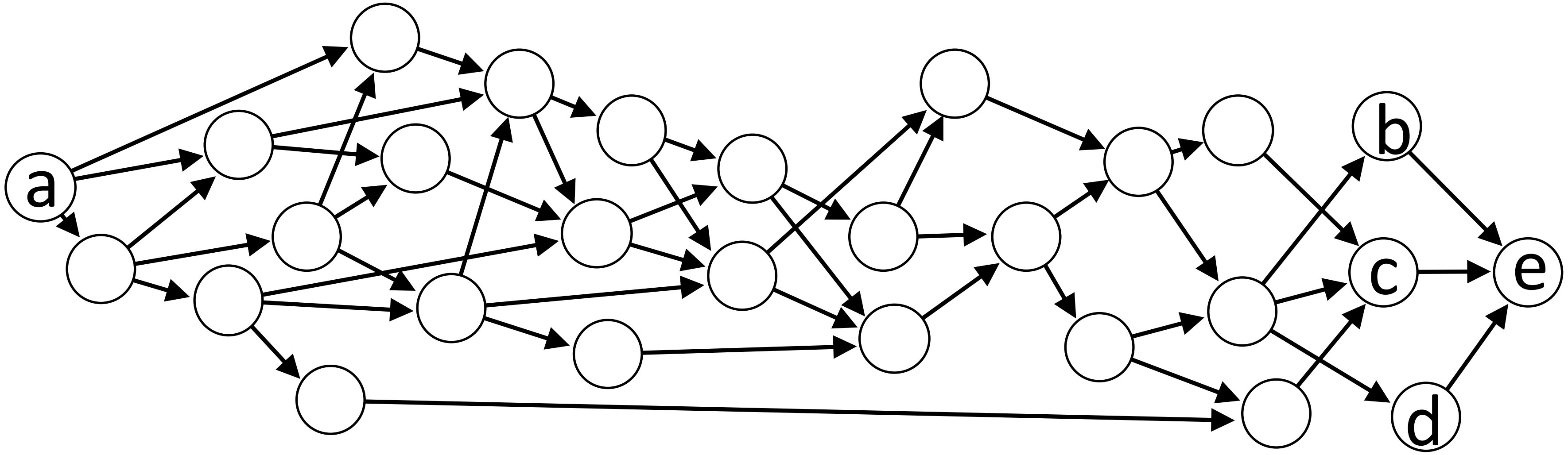
Find the Longest Path in a DAG



Interesting observations?

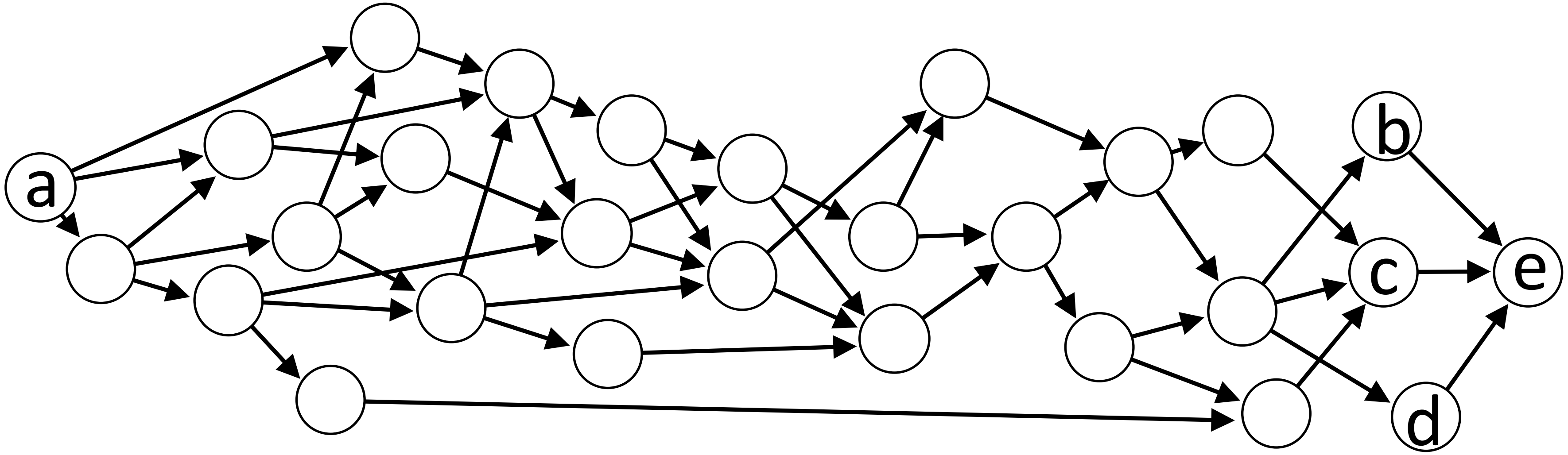
The longest path to e = $\max\begin{pmatrix}\text{longest path to b}\\ \text{longest path to c}\\ \text{longest path to d}\end{pmatrix} + 1$

Find the Longest Path in a DAG



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

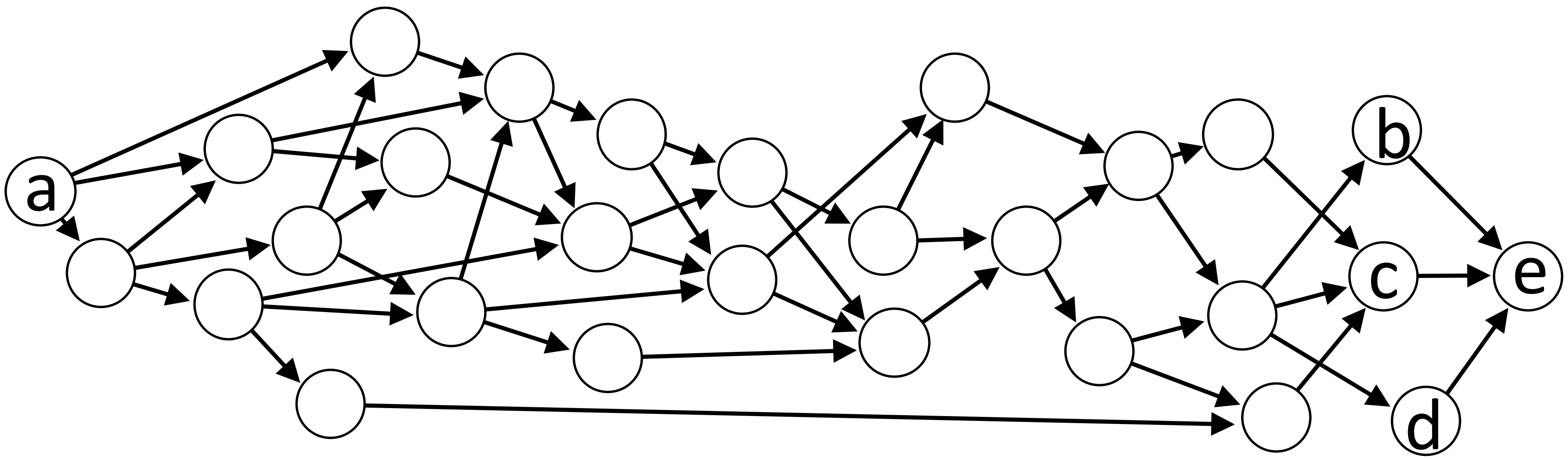
Find the Longest Path in a DAG



When are we ready to calculate the longest path to e?

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

Find the Longest Path in a DAG

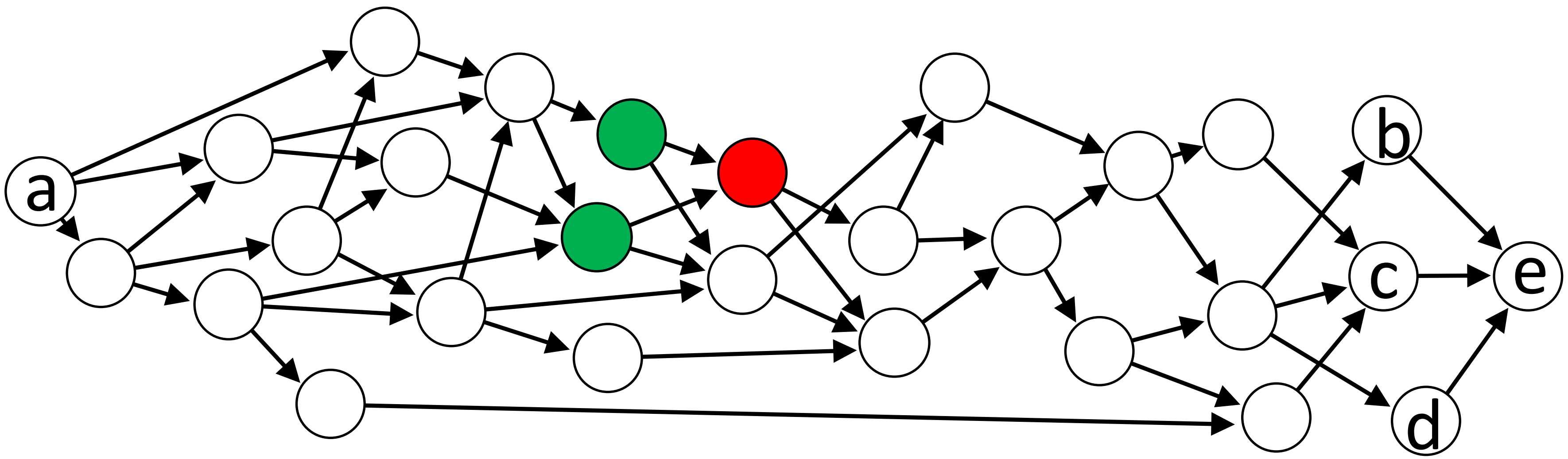


When are we ready to calculate the longest path to e?

When we have the longest paths to b, c, and d.

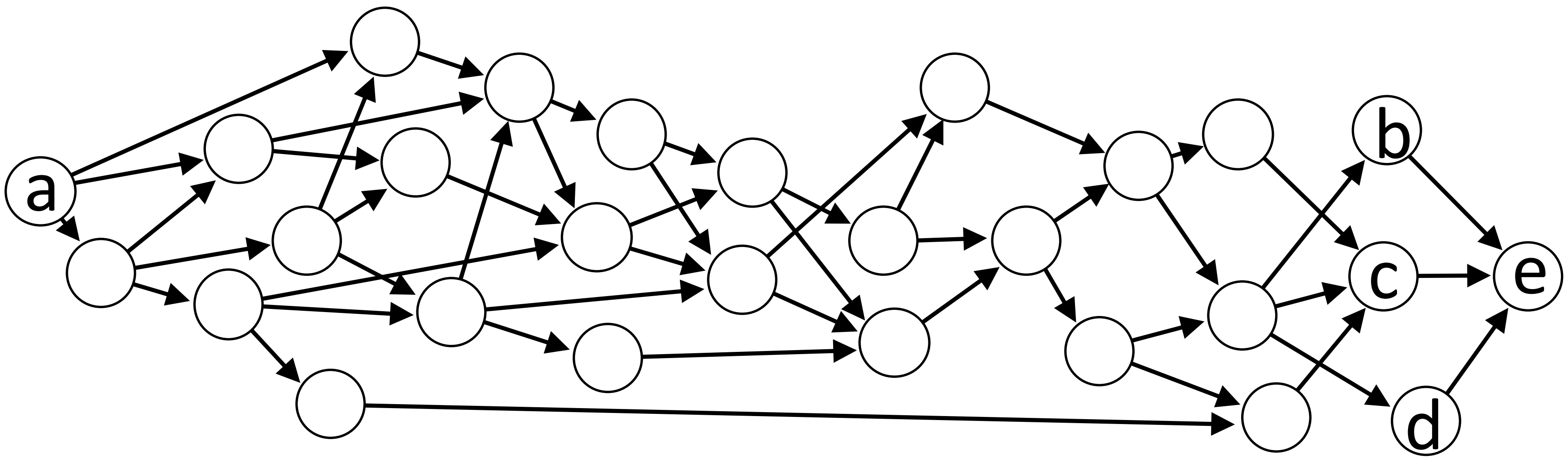
$$\text{longest path to e} = \max\begin{pmatrix}\text{longest path to b}\\ \text{longest path to c}\\ \text{longest path to d}\end{pmatrix} + 1$$

Find the Longest Path in a DAG



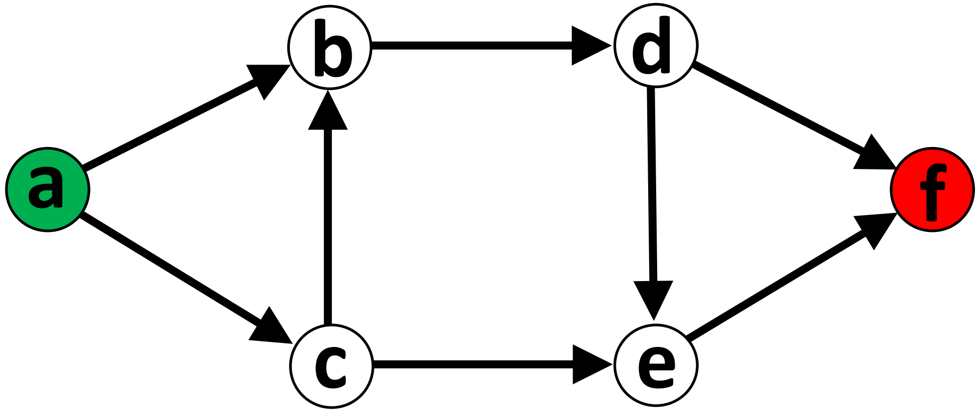
In general: We are ready to calculate the longest path to a vertex if we know the longest path for all incoming neighbors.

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.

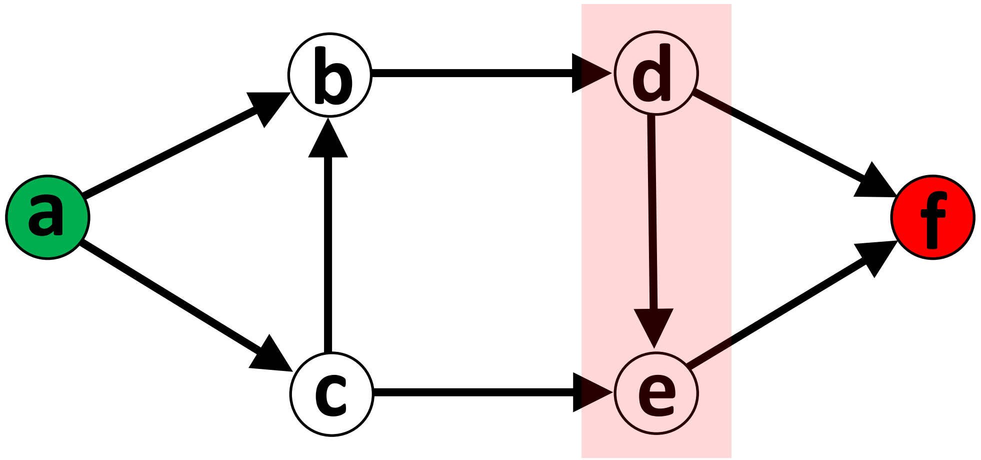
Topological Ordering



Topologically Ordered:

{a, c, b, d, e, f} ✓

Topological Ordering

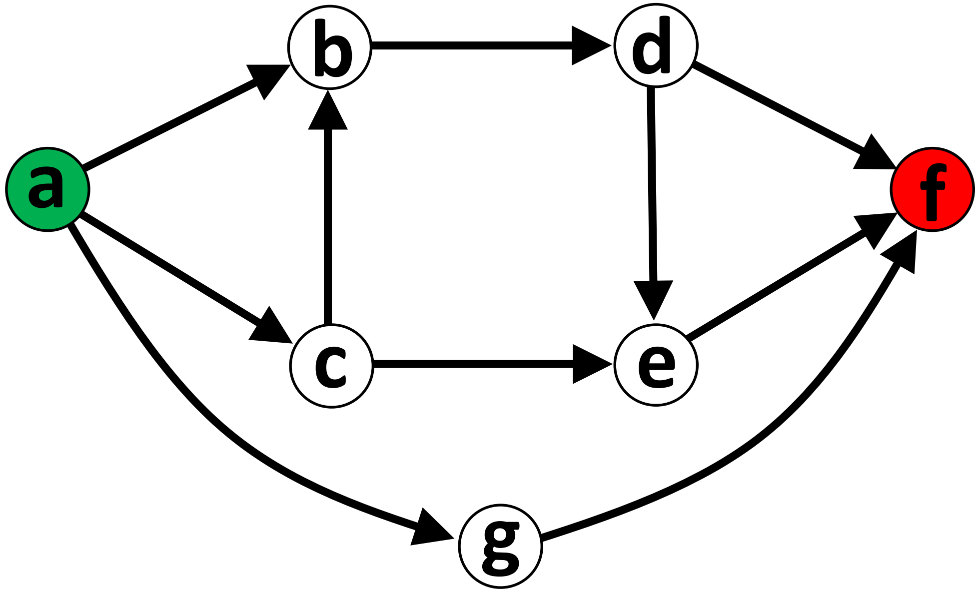


Topologically Ordered:

{a, c, b, d, e, f} ✓

{a, c, b, e, d, f} ✗

Topological Ordering

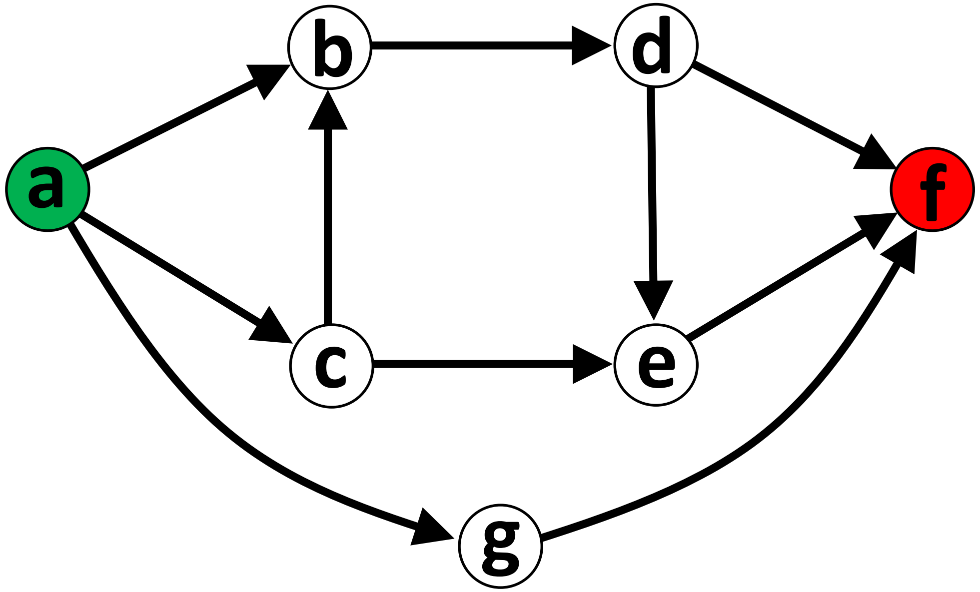


Topologically Ordered:

{a, c, g, b, d, e, f} ✓

{a, c, b, d, e, g, f} ✓

Topological Ordering

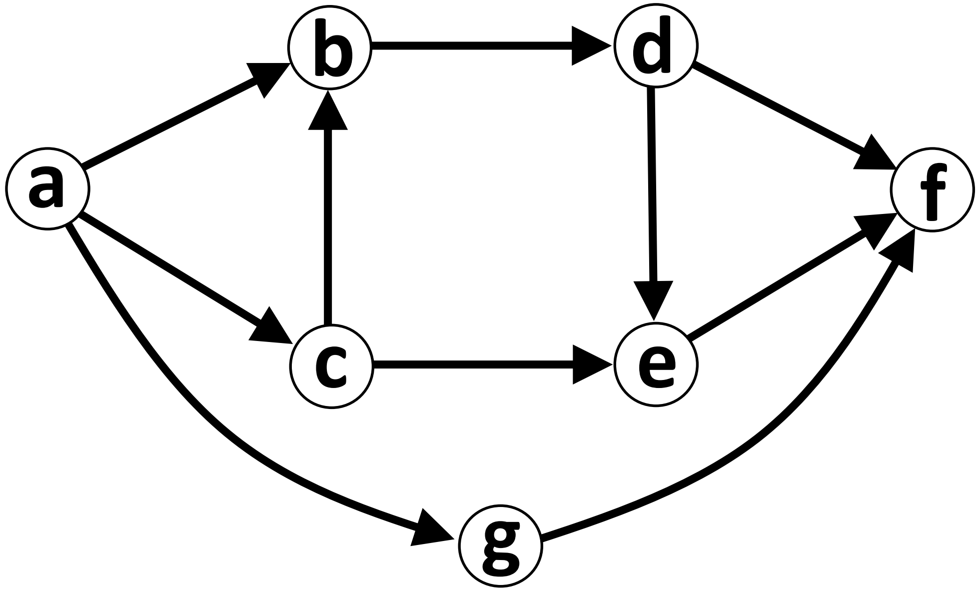


- There are various algorithms to find topological orderings
- Standard running time $= O(|V| + |E|)$.

Find the Longest Path in a DAG

Plan:

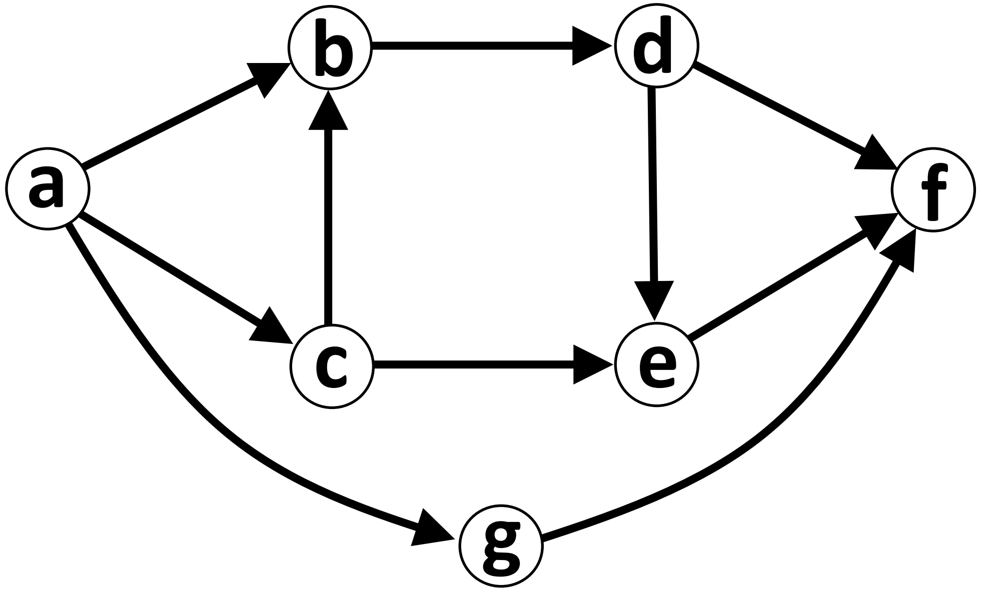
- ??



Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.

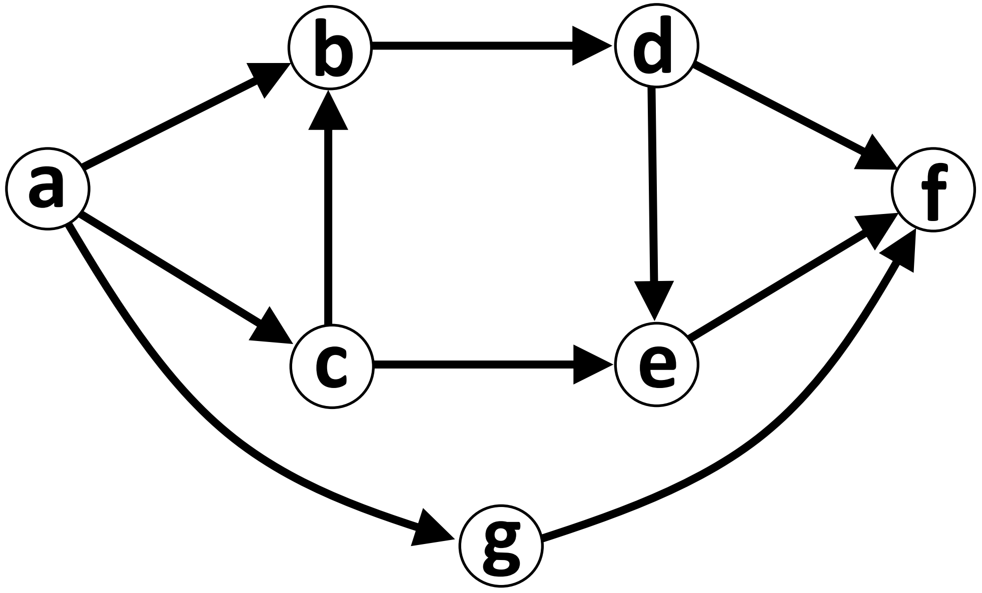


{a, c, g, b, d, e, f}

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

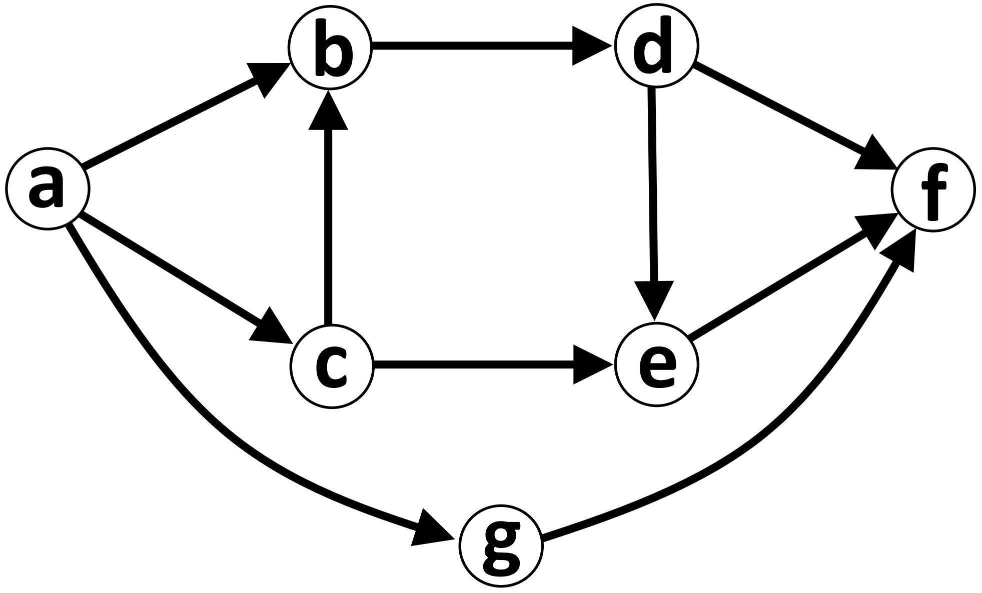
Plan:

- 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

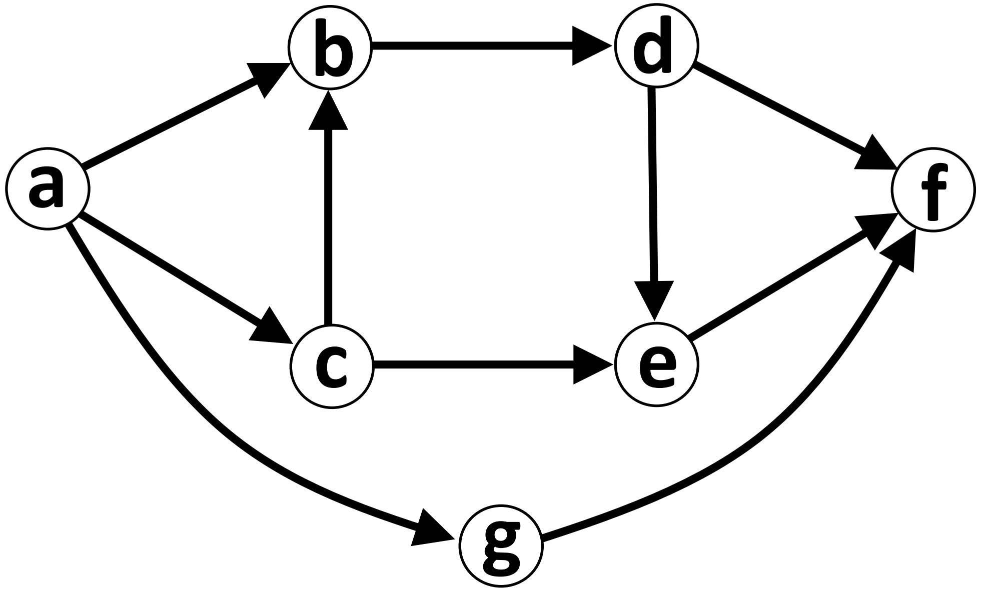
Find the Longest Path in a DAG

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

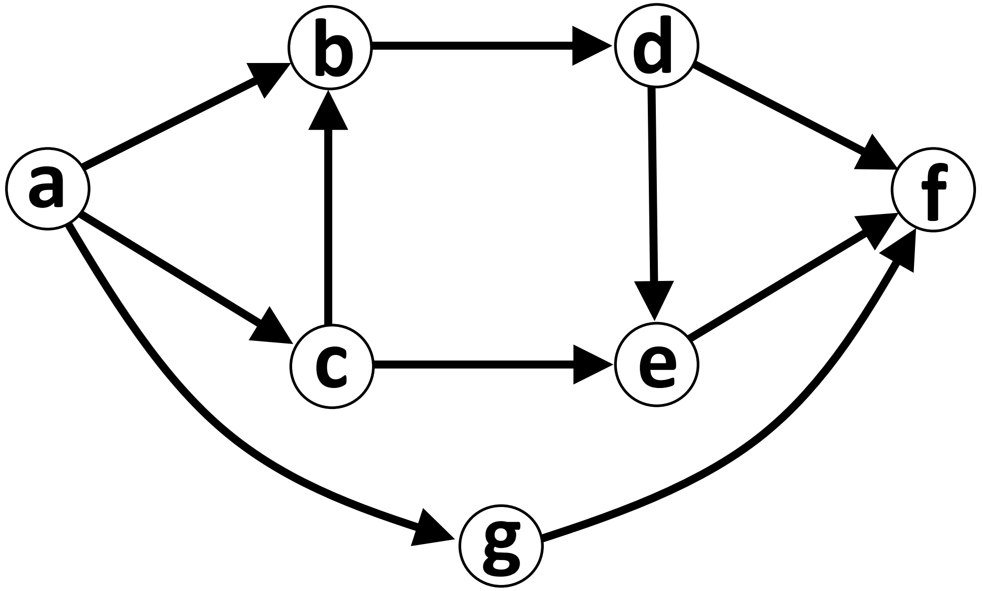
Find the Longest Path in a DAG

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

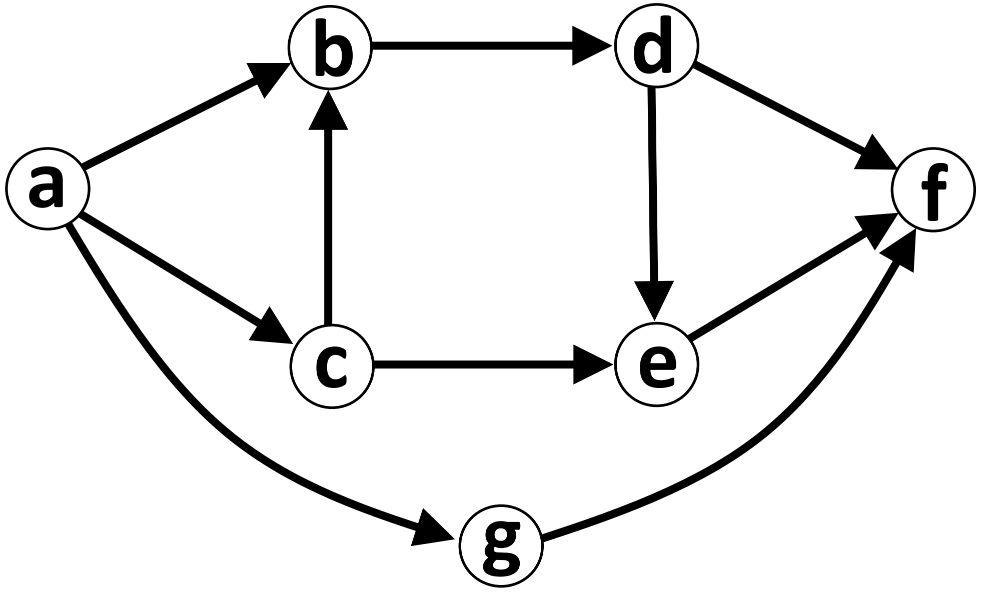
Find the Longest Path in a DAG

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

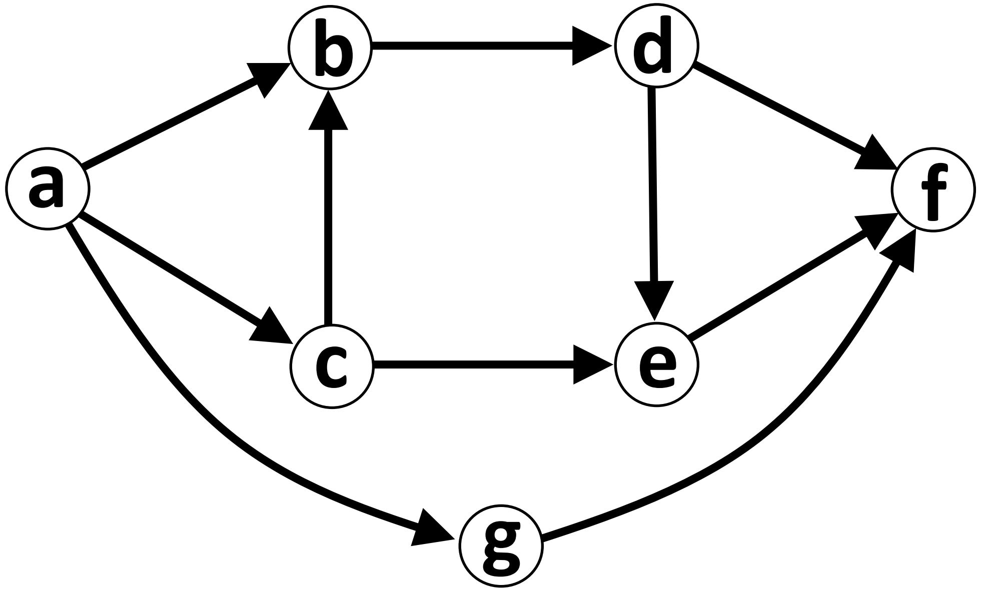
Find the Longest Path in a DAG

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:

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

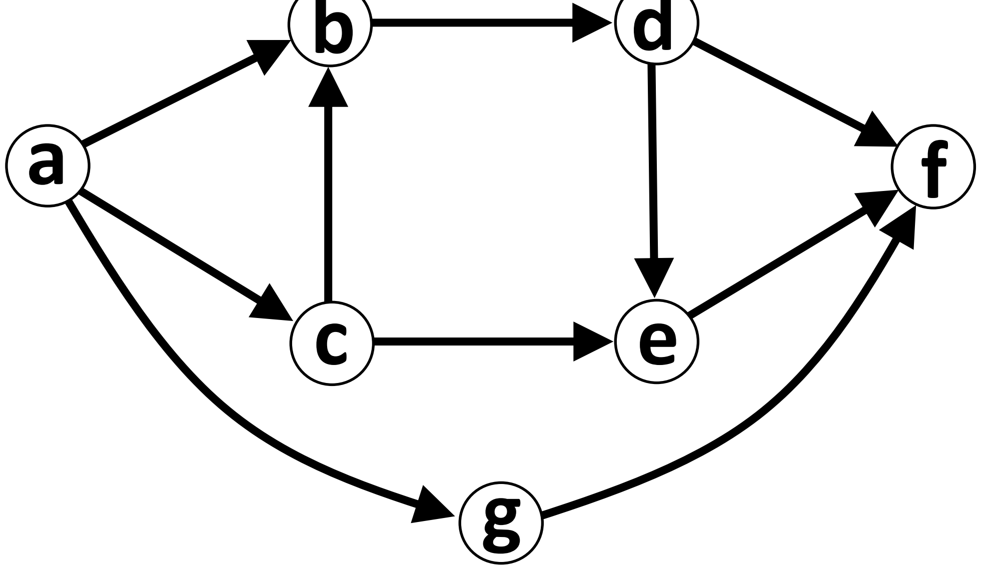
Find the Longest Path in a DAG

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:

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

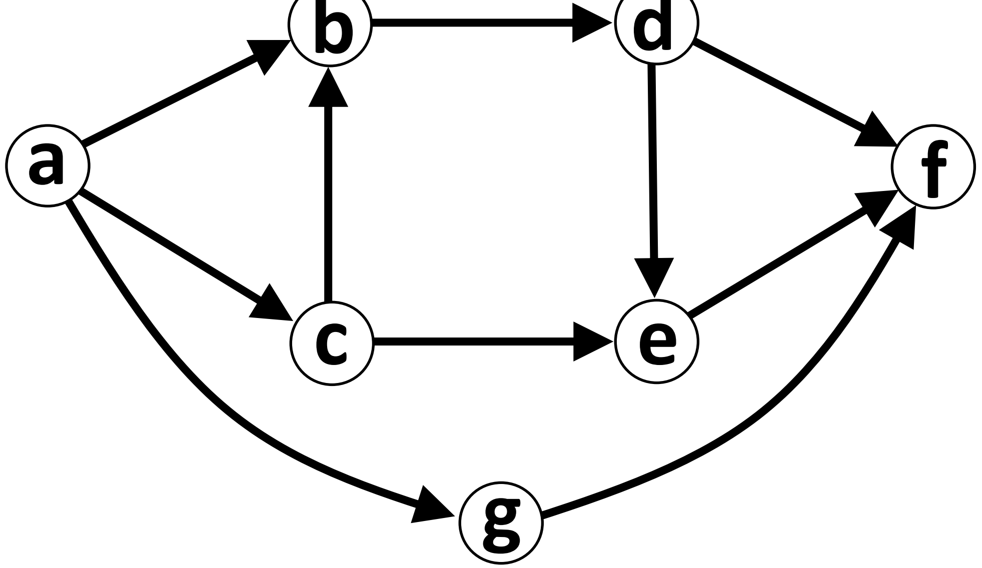
Find the Longest Path in a DAG

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:

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

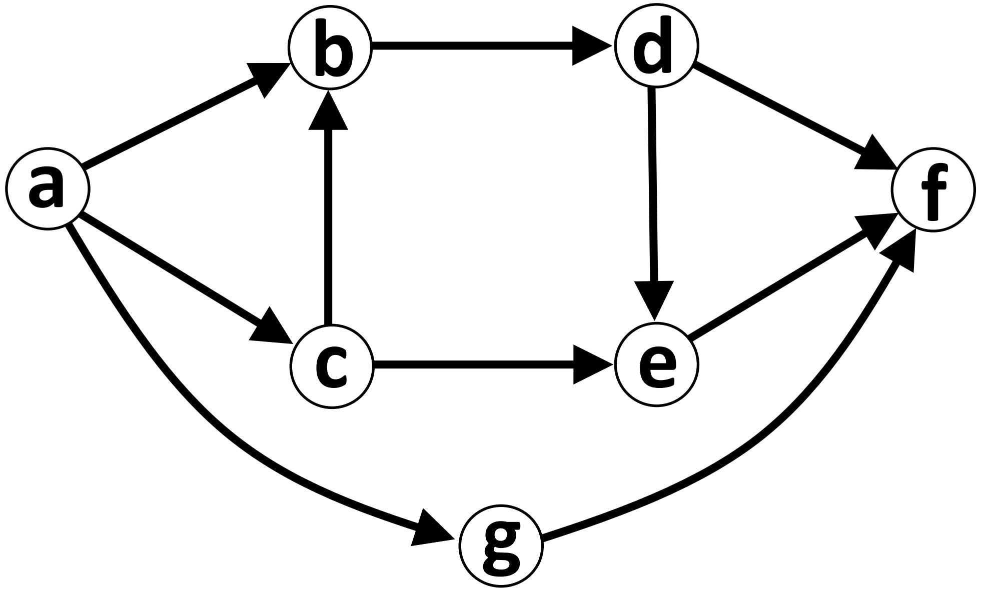
Find the Longest Path in a DAG

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:

  $\max_n(\text{longest path to } n) + 1,$
  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	0	0	0	1

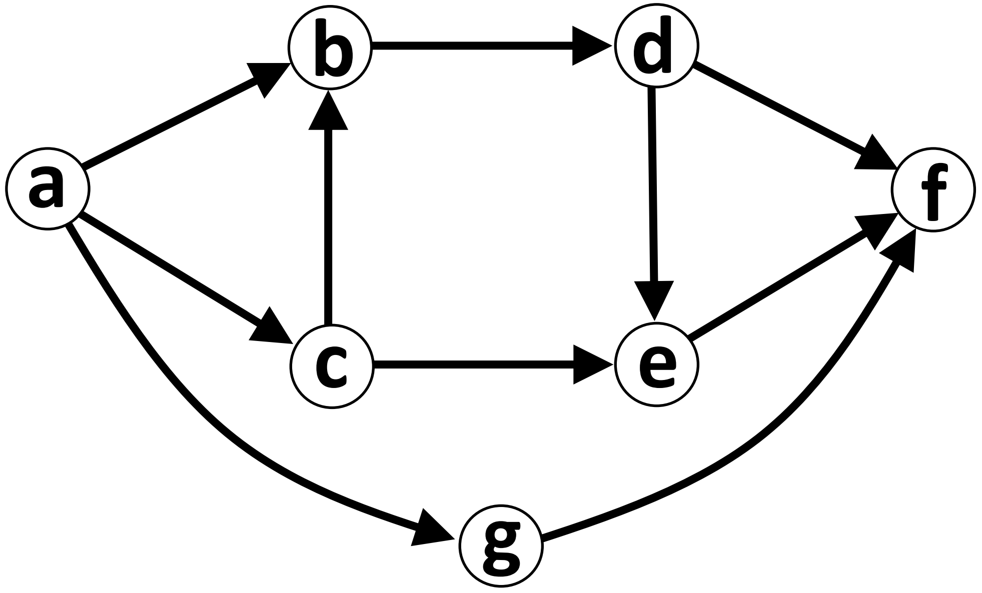
Find the Longest Path in a DAG

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

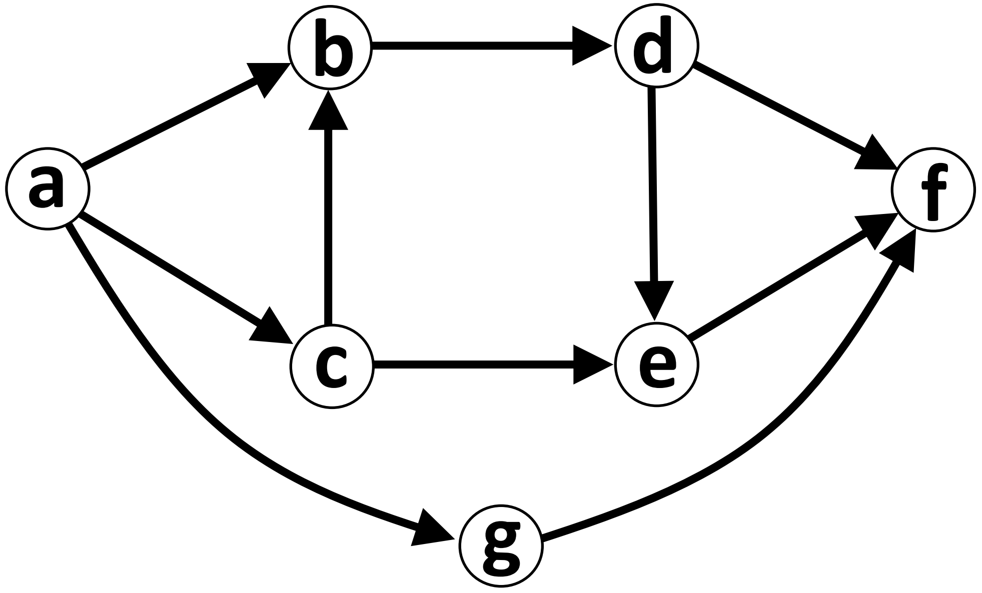
Find the Longest Path in a DAG

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 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: ?

Find the Longest Path in a DAG

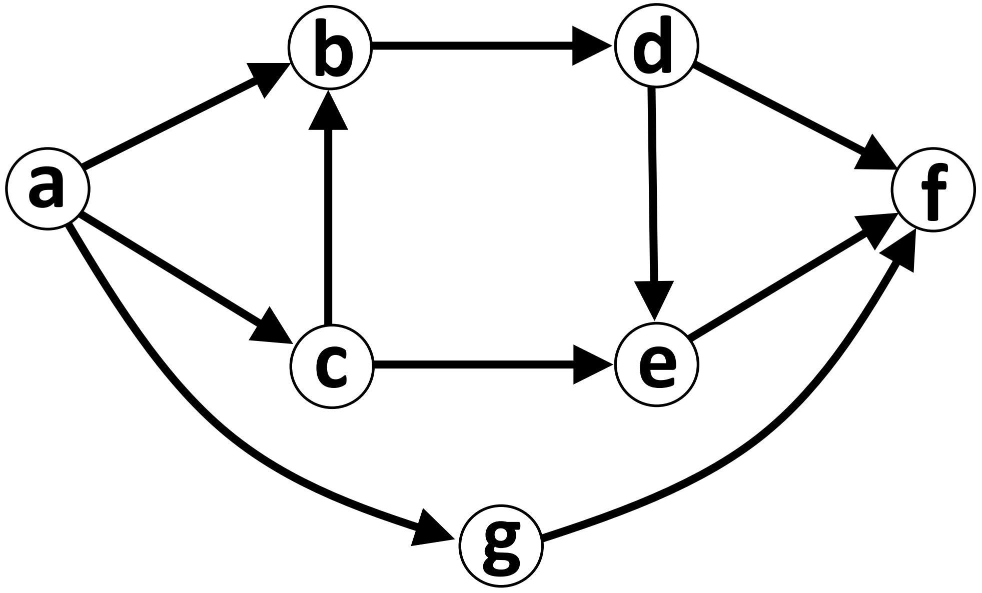
```
longest_path(G=(V,E)):
    pathLengths = [0,...,0]
    Let V_sort be topologically sort vertices
    for each vertex v in V_sort:
        for each incoming neighbor n of v:
            if pathLengths[n] + 1 > pathLengths[v]:
                pathLengths[v] = pathLengths[n] + 1
    return maxValue(pathLengths)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

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

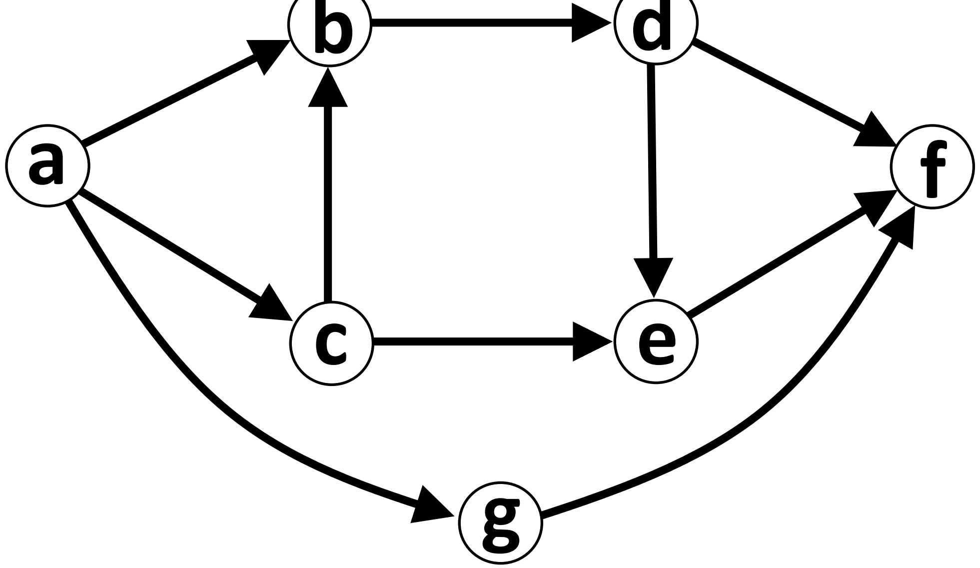
Find the Longest Path in a DAG

Plan:

- Topologically sort vertices.
- Make array to store leng ngest path that ends at ea x.
- For each vert er, calculate longest

  gest path to $n) + 1$, incoming neighbors n.
- Largest value in array = Longest path.

So, what's the path??



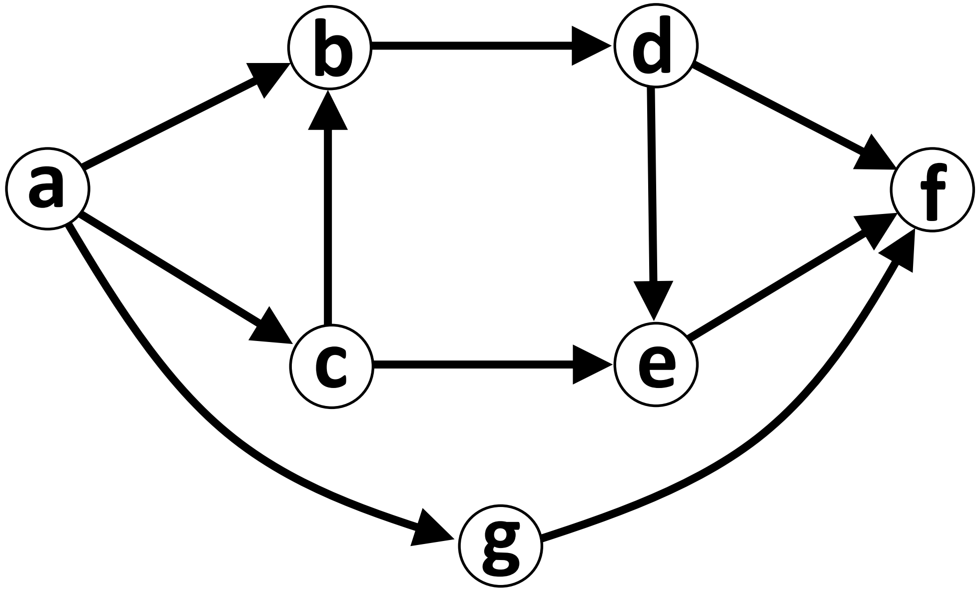
{a, c, g, b, d, e, f}

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.



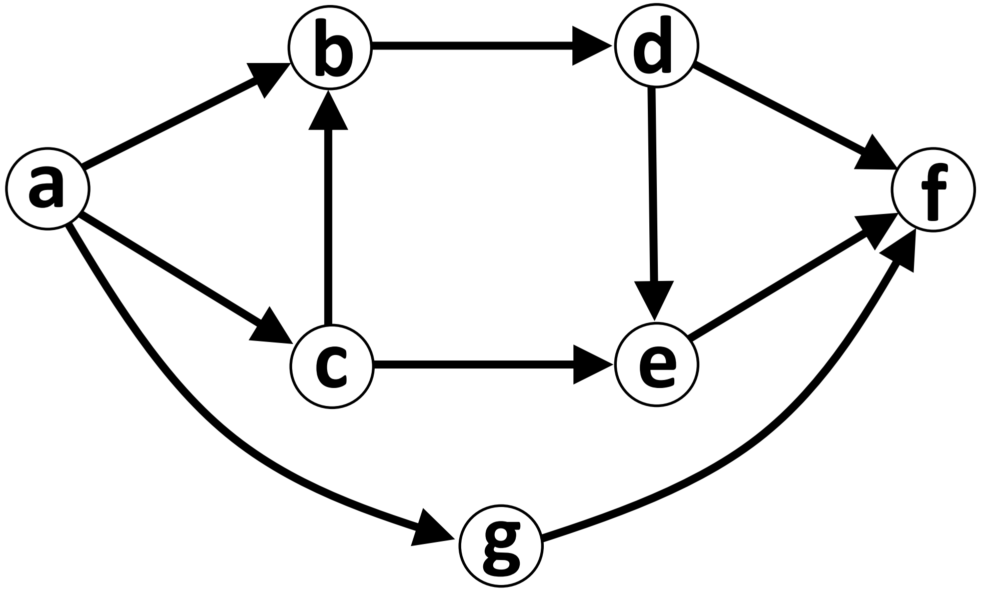
{a, c, g, b, d, e, f}

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.



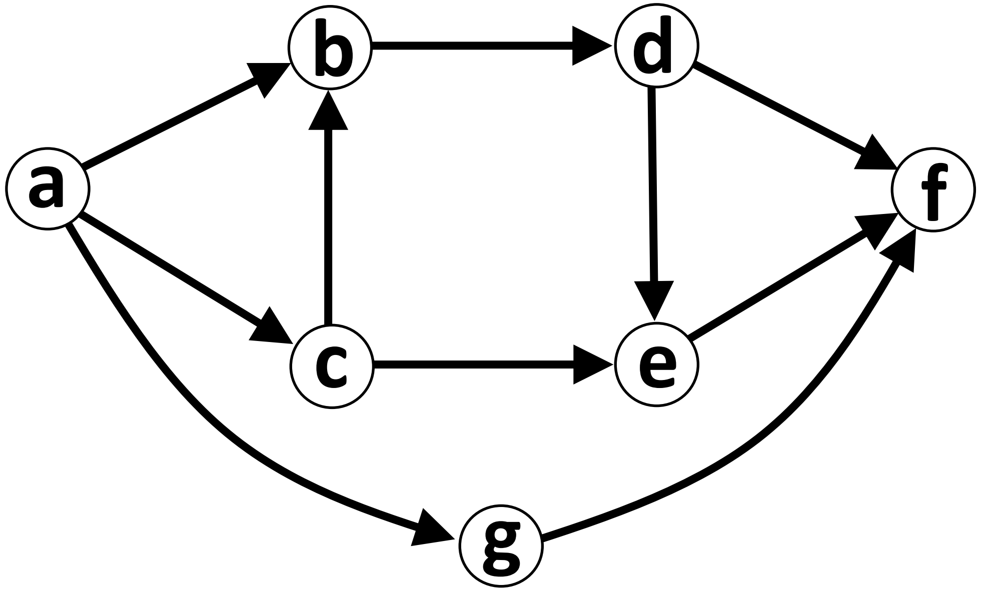
{a, c, g, b, d, e, f}

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.



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	0	0	0	0	0
-	-	-	-	-	-	-

Find the Longest Path in a DAG

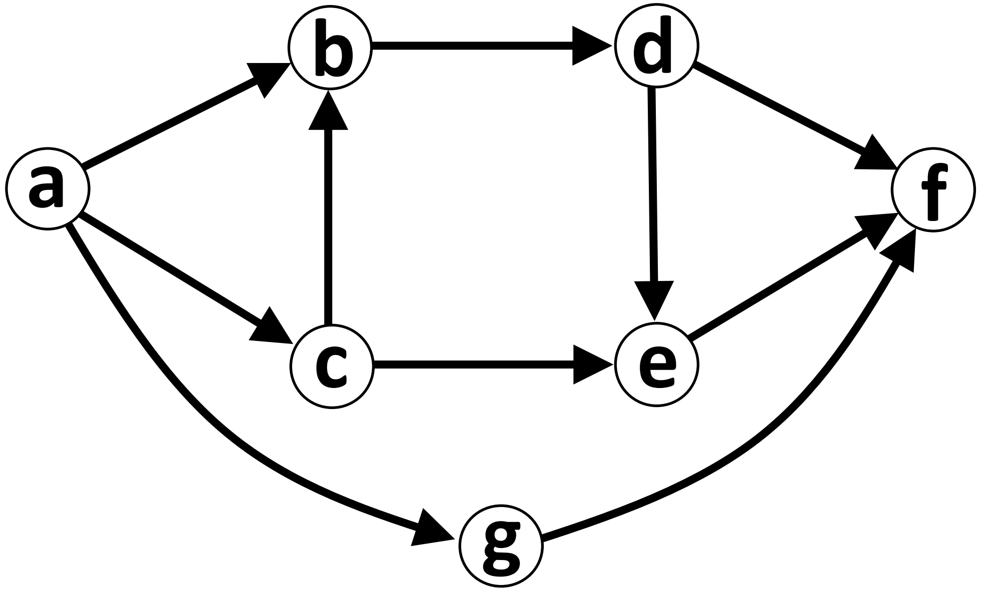
Plan:

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

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}

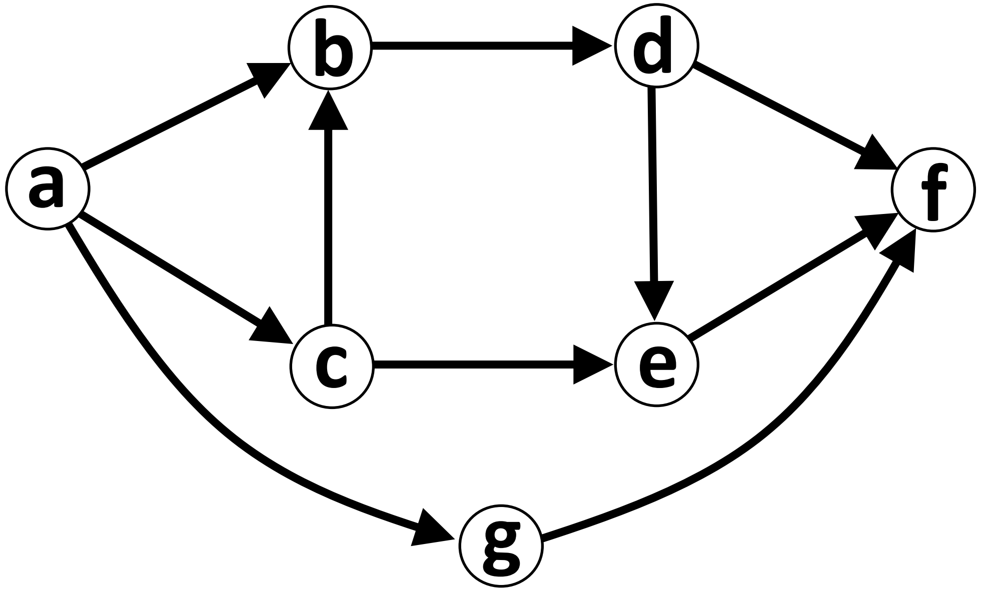
a	b	c	d	e	f	g
0	0	0	0	0	0	0
-	-	-	-	-	-	-

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.

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



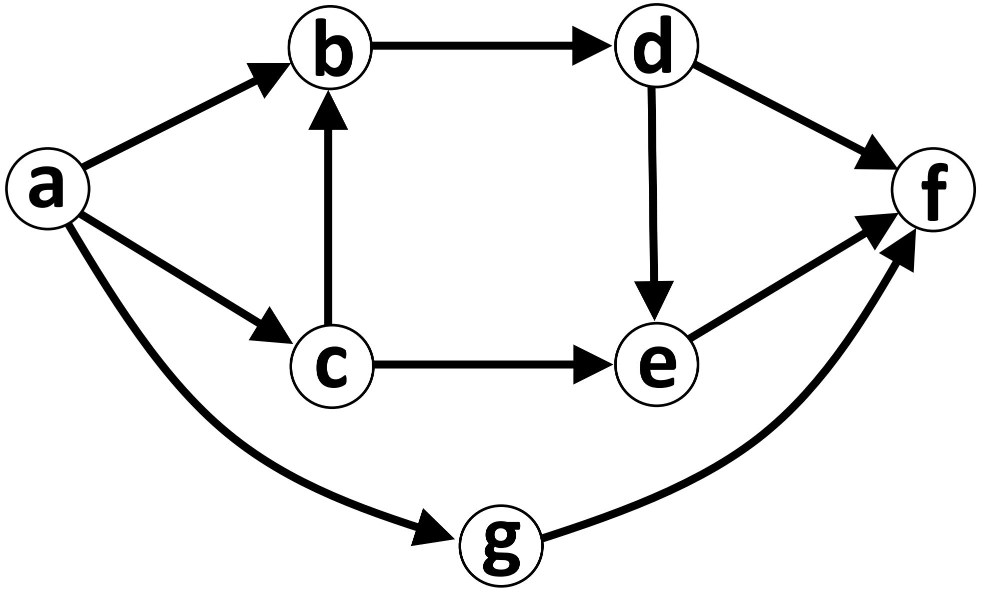
{a, c, g, b, d, e, f}

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

Find the Longest Path in a DAG

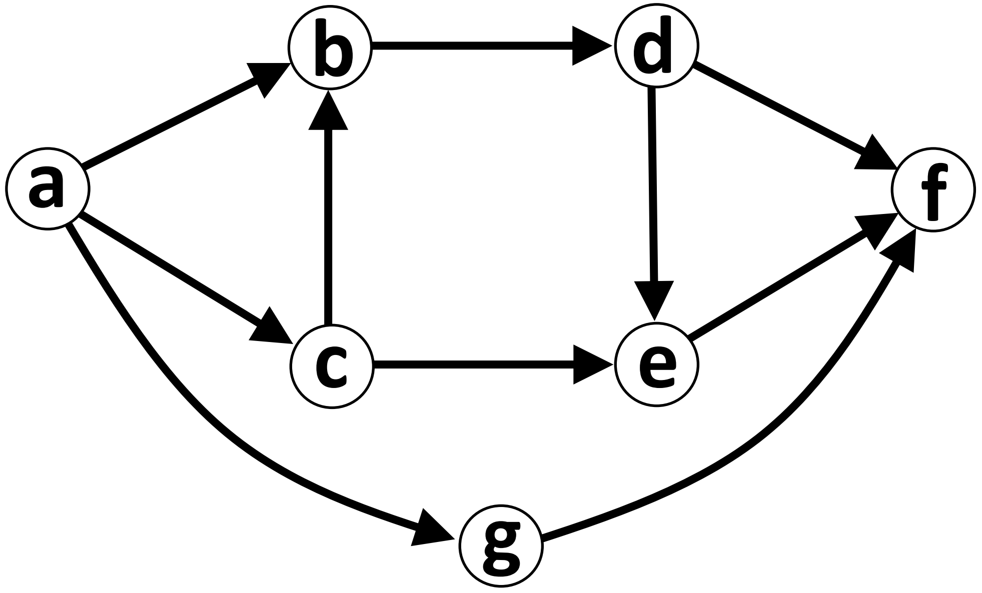
Plan:

- 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.



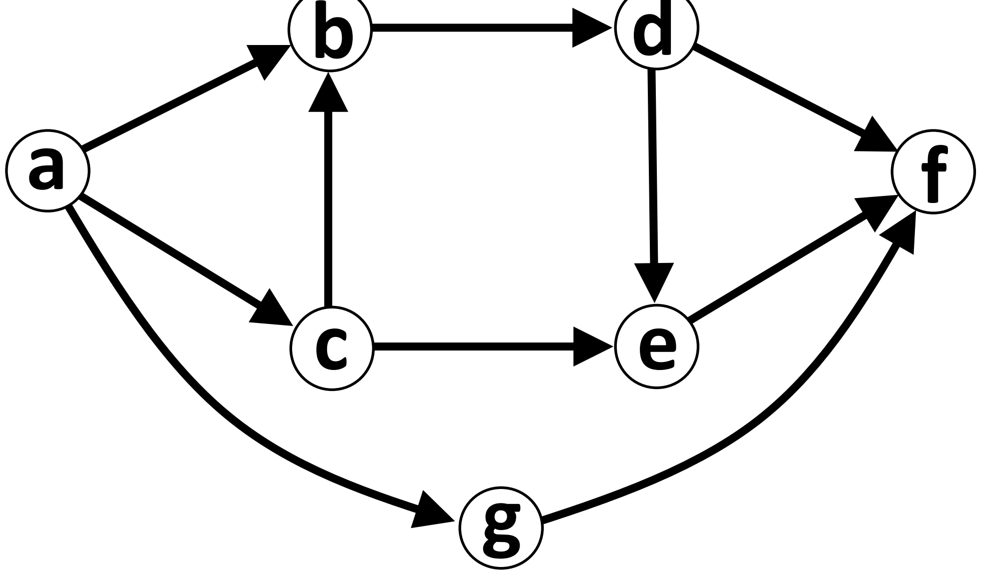
{a, c, g, b, d, e, f}

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.



{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	1	0	0	0	1
-	-	a	-	-	-	a

Find the Longest Path in a DAG

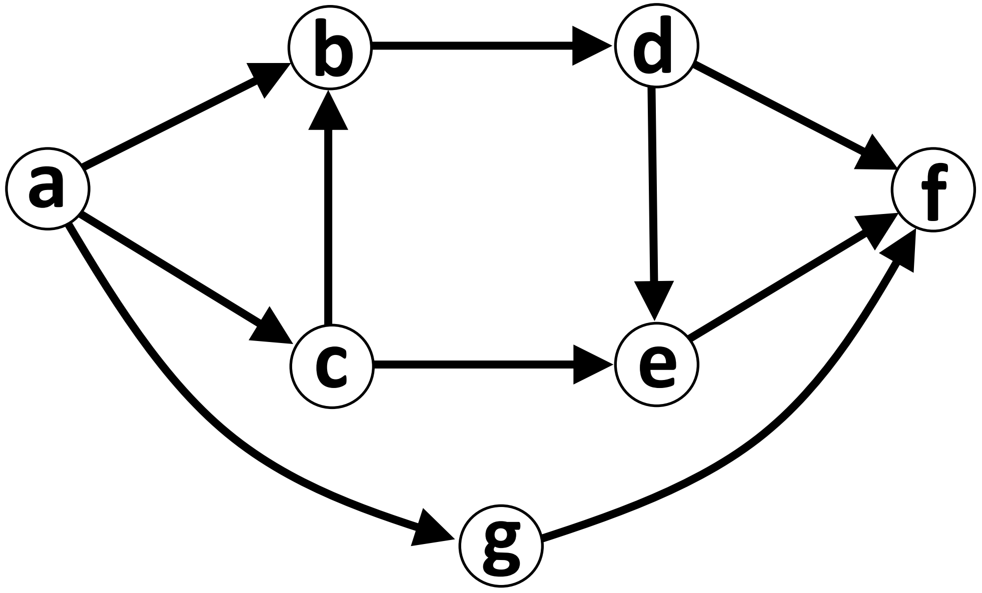
Plan:

- 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.



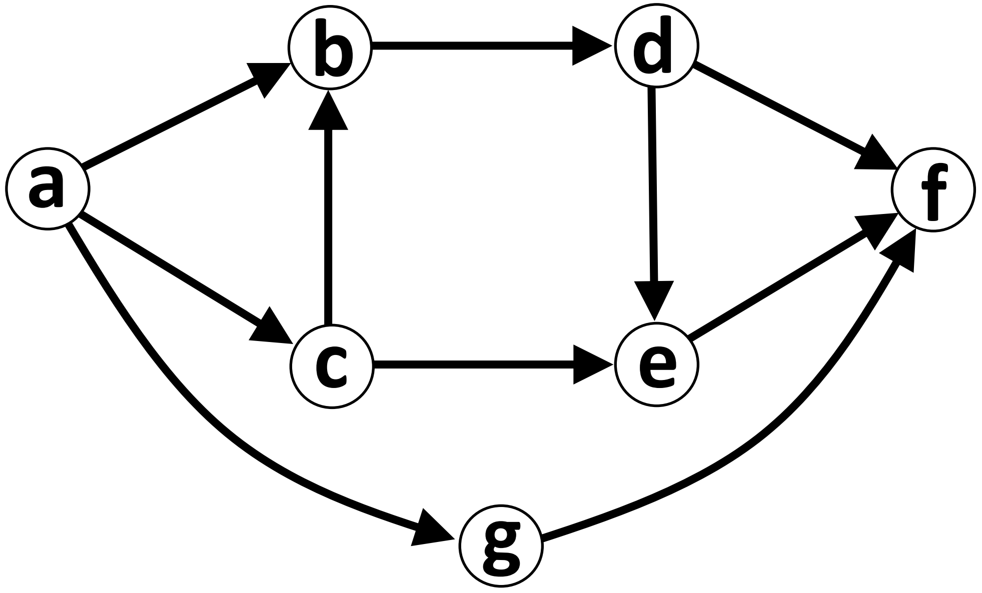
{a, c, g, b, d, e, f}

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.



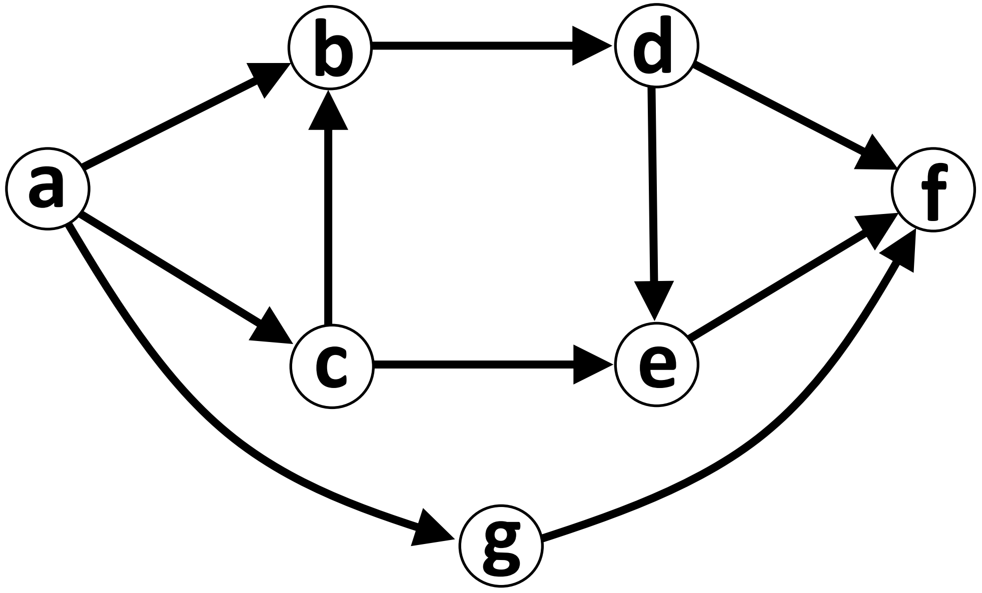
{a, c, g, b, d, e, f}

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}

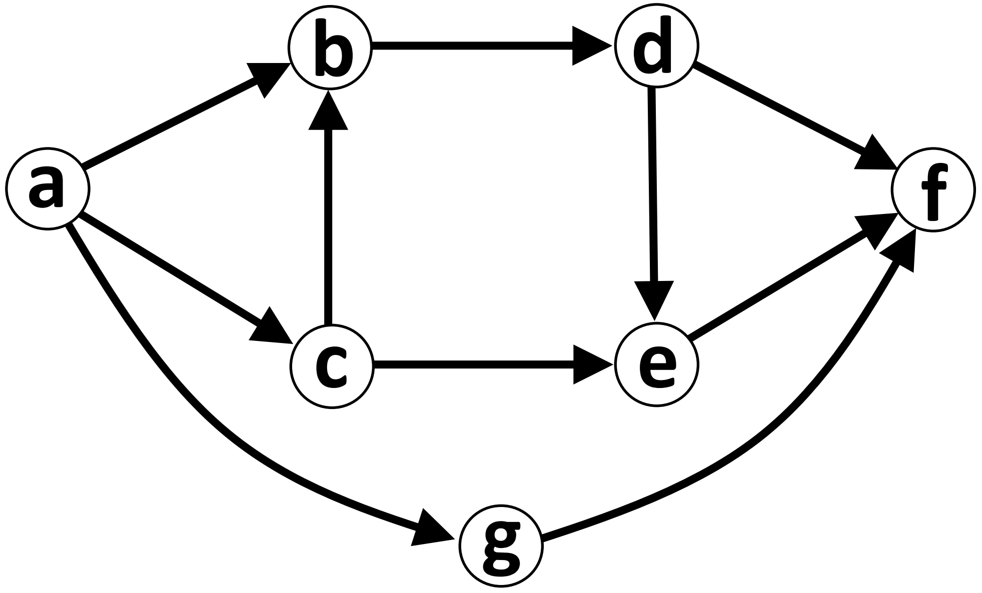
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

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.

path: f



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

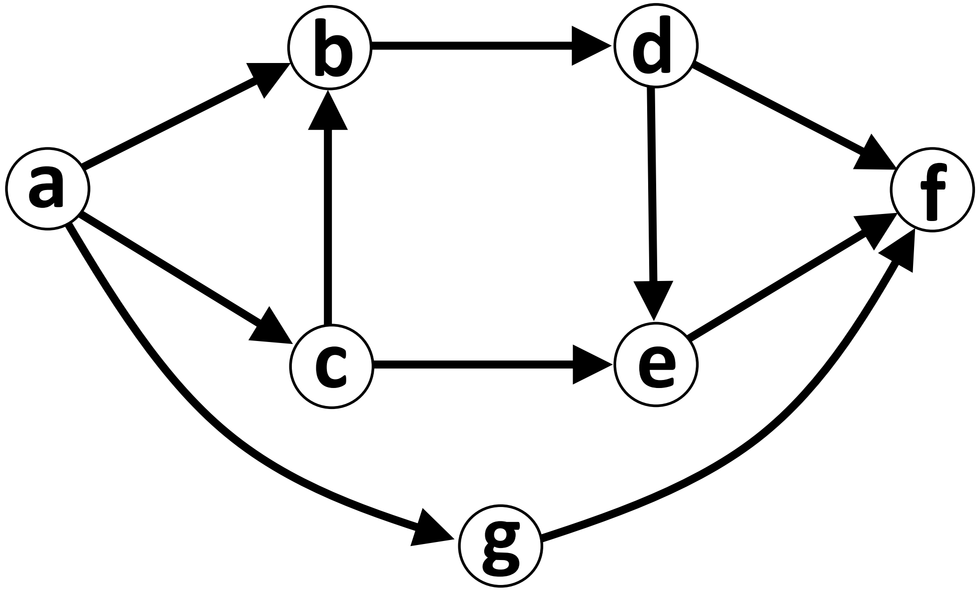
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

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.

path: f <- e



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

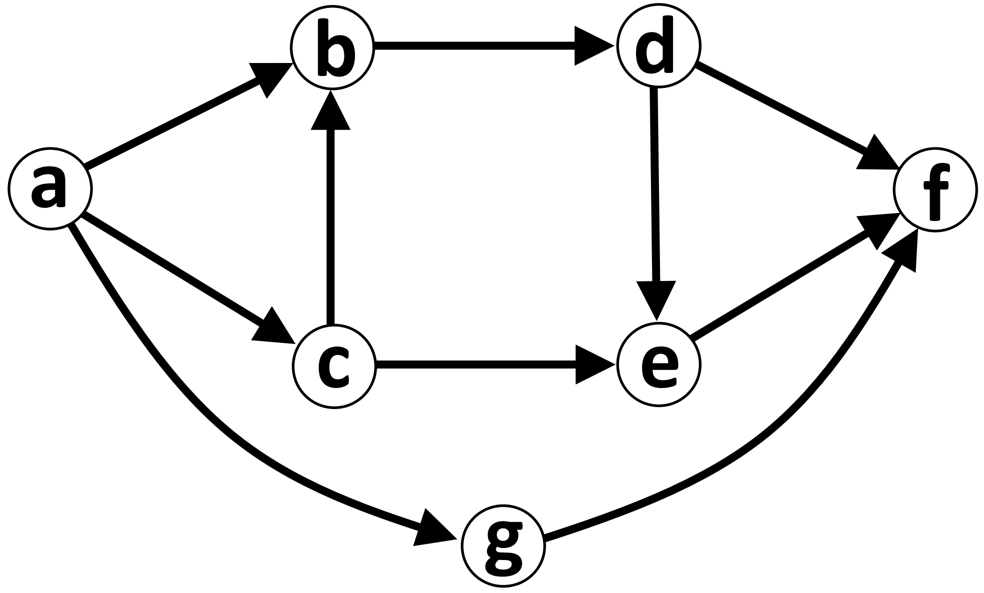
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

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.

path: f <- e <- d



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

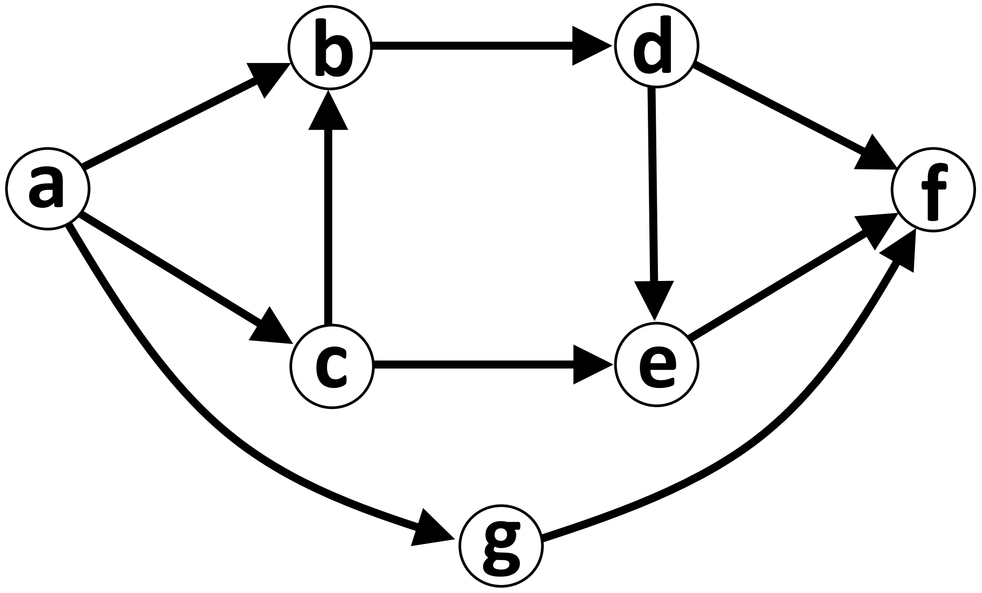
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

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.

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



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

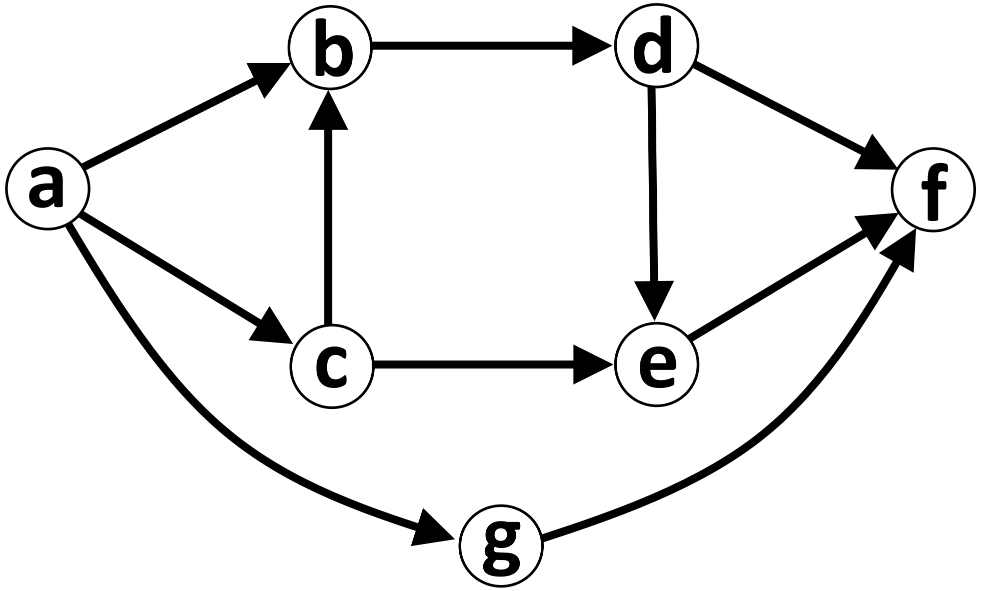
a	b	c	d	e	f	g
0	2	1	3	4	5	1
-	c	a	b	d	e	a

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.

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



{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