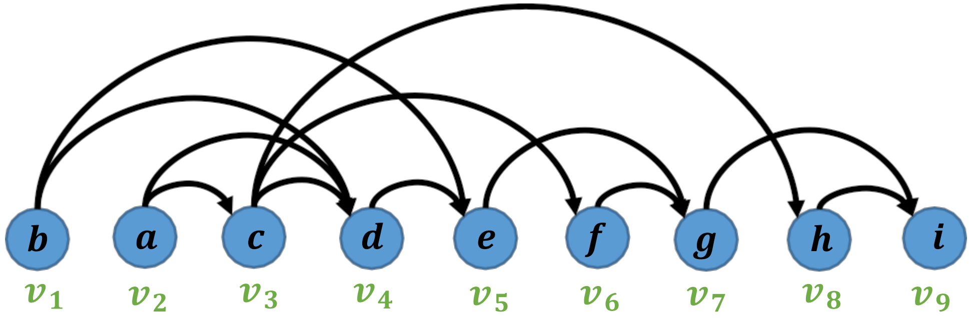


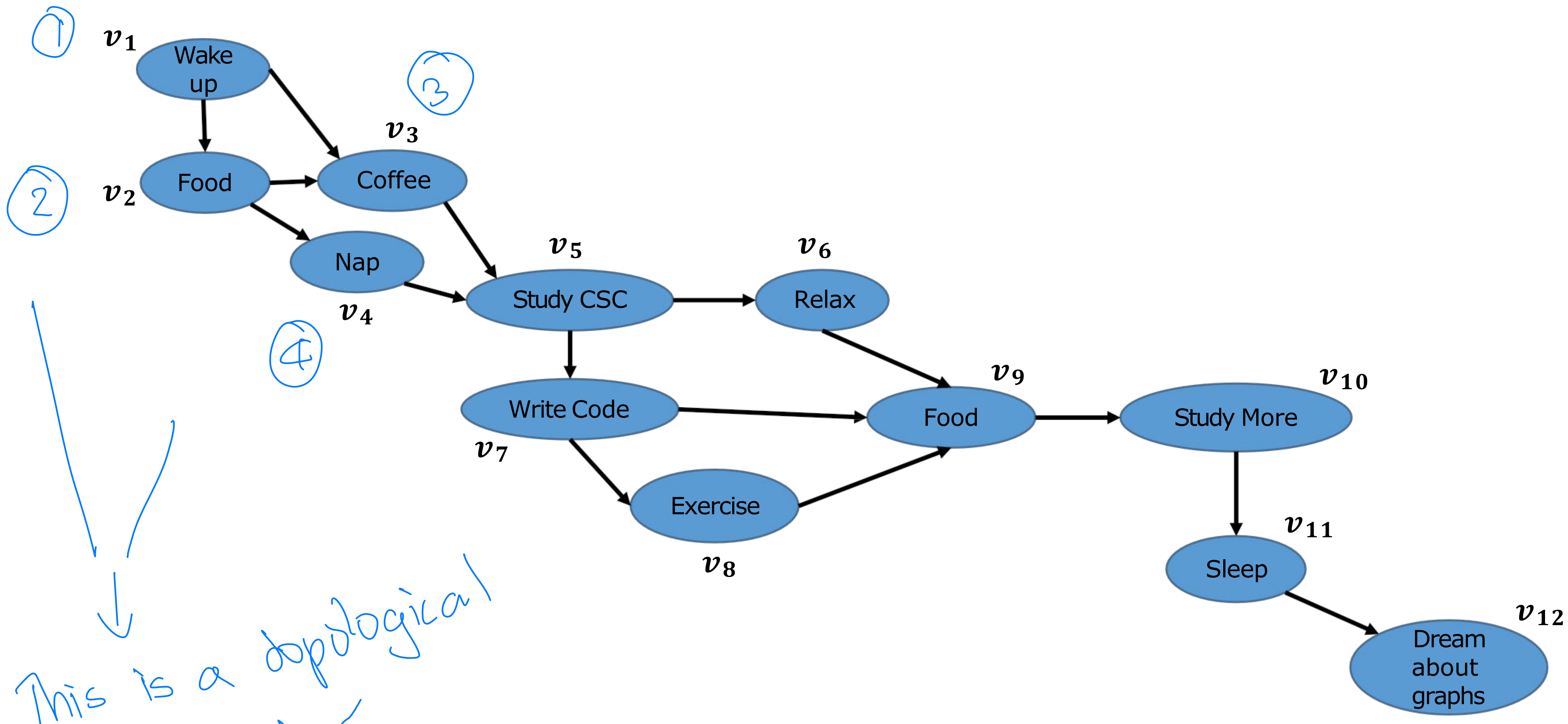
Lecture 4: Topological Sort

CSC 226: Algorithms and Data Structures II

University of Victoria

Topological Ordering

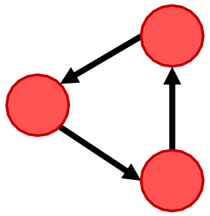
- Number vertices so that having edge (v_i, v_j) implies $i < j$ in numbering



This is a topological order

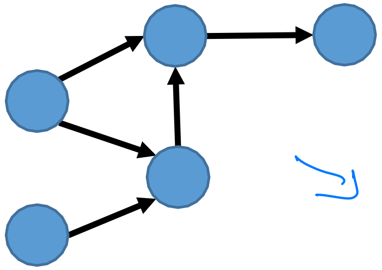
DAGs and Topological Ordering

- A **directed acyclic graph (DAG)** is a digraph that has no directed cycles



Directed cycle

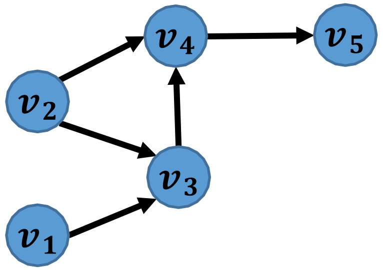
Directed acyclic graph G

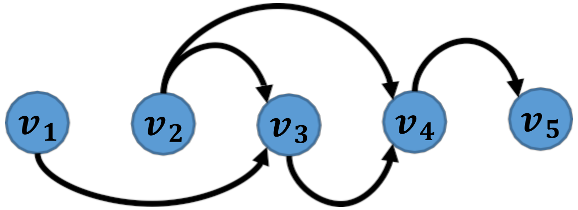


⇒ No directed cycles.

- A **topological ordering** of a digraph is a numbering $v_1, \ldots, v_n$ of the vertices such that for every directed edge (v_i, v_j), we have $i < j$ in the numbering.

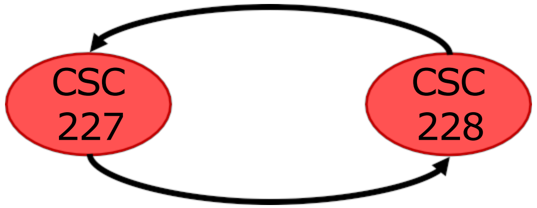
Topological ordering of G





- **Theorem:** A digraph admits a topological ordering if and only if it is a DAG

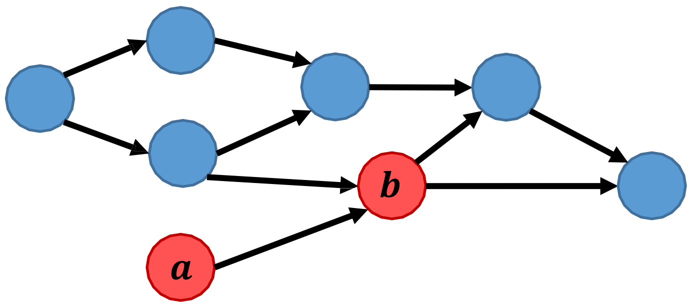
No topological ordering



If it has a cycle, you get a loop that you get stuck in.

Topological Sort

- A **directed acyclic graph** defines a **partial order** (way to compare vertices $<, =, >$)
- Hence, a graph can be partially sorted
- Applications:
    - **Nodes** are tasks or work assignments
    - **Edges** represent dependencies among tasks (precedence relationships)



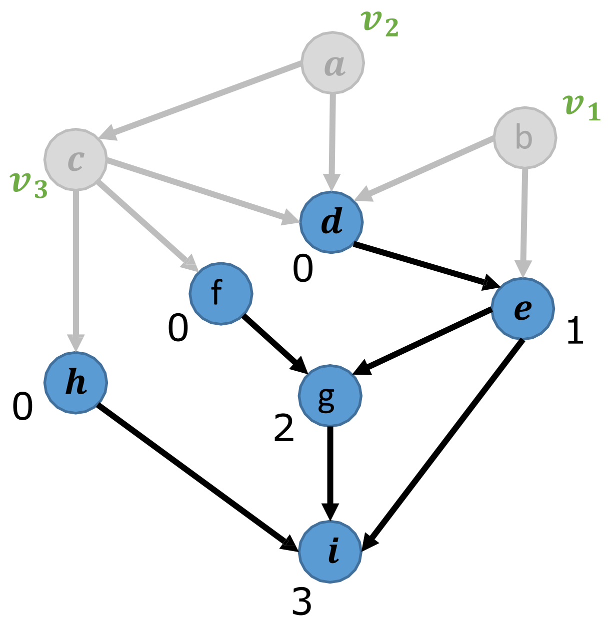
    - Task $\boldsymbol{b}$ cannot start until task $\boldsymbol{a}$ is completed
    - Course prerequisites
    - Inheritance between Java classes
    - Compilation dependency graph

Topological Sort Algorithm (Iterative)

- If a digraph is **acyclic**, then there must exist a node v_1 with $\mathbf{indeg}(v_1) = \mathbf{0}$

Or, a node that only has OUT connections, no IN connections
A base node.

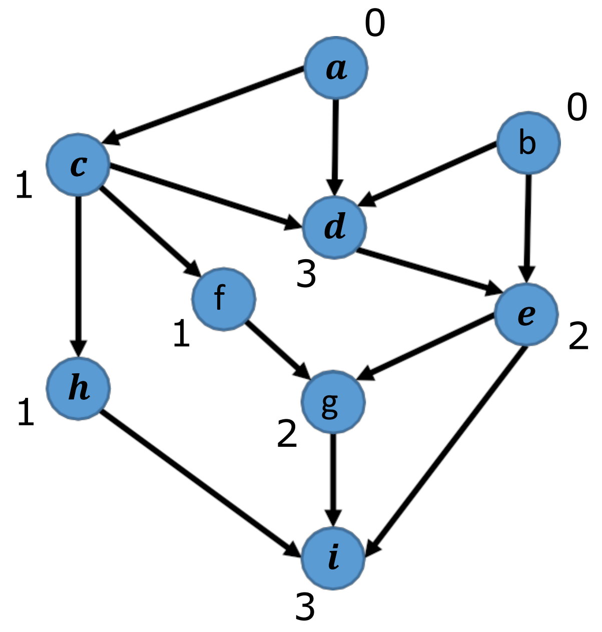
- Remove v_1 and all its outgoing edges
- The **resulting graph** must also be **acyclic**
  - Removing a vertex from an acyclic graph can't create a cycle
- Remove the next vertex v_2 with $\mathbf{indeg}(v_2) = \mathbf{0}$
- Repeat until all vertices are removed



Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort(G)}$:
Input: Digraph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices
Output: Topological ordering of $\boldsymbol{G}$ or an indication of a directed cycle

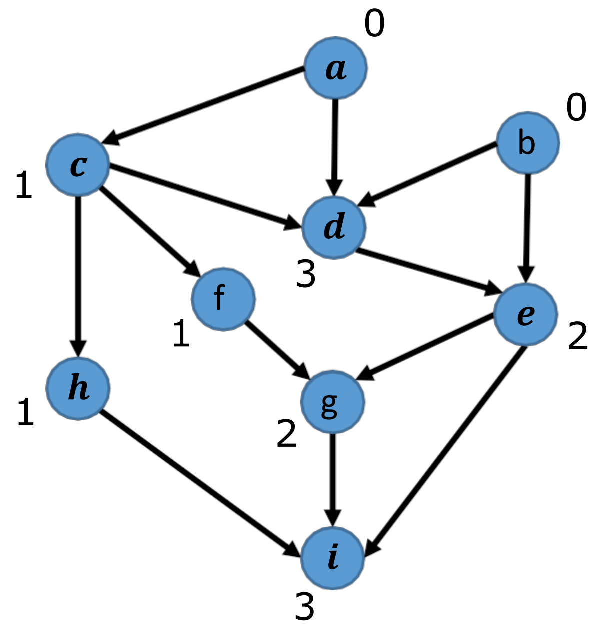
$\boldsymbol{S} \leftarrow$ empty stack
for each vertex $\boldsymbol{u}$ in $\boldsymbol{G}$ **do**
    if $\deg(\boldsymbol{u}) = 0$ **then**
        $\boldsymbol{S.push(u)}$
$\boldsymbol{i \leftarrow 1}$
while $\boldsymbol{S}$ is not empty **do**
    $\boldsymbol{u \leftarrow S.pop()}$
    Number $\boldsymbol{u}$ as vertex $\boldsymbol{v_i}$
    $\boldsymbol{i \leftarrow i + 1}$
    for each vertex $\boldsymbol{v}$ adjacent to $\boldsymbol{u}$ **do**
        $\deg(\boldsymbol{v}) \leftarrow \deg(\boldsymbol{v}) - \boldsymbol{1}$
        if $\deg(\boldsymbol{v}) = \boldsymbol{0}$ **then**
            $\boldsymbol{S.push(v)}$
if $\boldsymbol{i > n}$ **then**
    return $\boldsymbol{v_1, v_2, \ldots, v_n}$
return "$\boldsymbol{G}$ has a directed cycle"



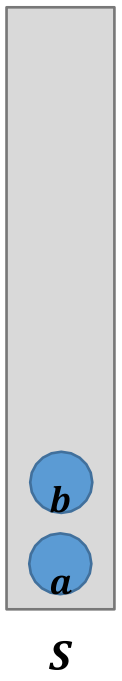
Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort(G)}$**:**
Input: Digraph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices
Output: Topological ordering of $\boldsymbol{G}$ or an indication of a directed cycle

$\boldsymbol{S} \leftarrow$ empty stack
for each vertex $\boldsymbol{u}$ in $\boldsymbol{G}$ **do**
  if $\deg(\boldsymbol{u}) = 0$ **then**
    $\boldsymbol{S.push(u)}$
$\boldsymbol{i \leftarrow 1}$
while $\boldsymbol{S}$ is not empty **do**
  $\boldsymbol{u \leftarrow S.pop()}$
  Number $\boldsymbol{u}$ as vertex $\boldsymbol{v_i}$
  $\boldsymbol{i \leftarrow i + 1}$
  for each vertex $\boldsymbol{v}$ adjacent to $\boldsymbol{u}$ **do**
    $\deg(\boldsymbol{v}) \leftarrow \deg(\boldsymbol{v}) - \mathbf{1}$
    if $\deg(\boldsymbol{v}) = \mathbf{0}$ **then**
      $\boldsymbol{S.push(v)}$
if $\boldsymbol{i > n}$ **then**
  return $\boldsymbol{v_1, v_2, \ldots, v_n}$
return "$\boldsymbol{G}$ has a directed cycle"



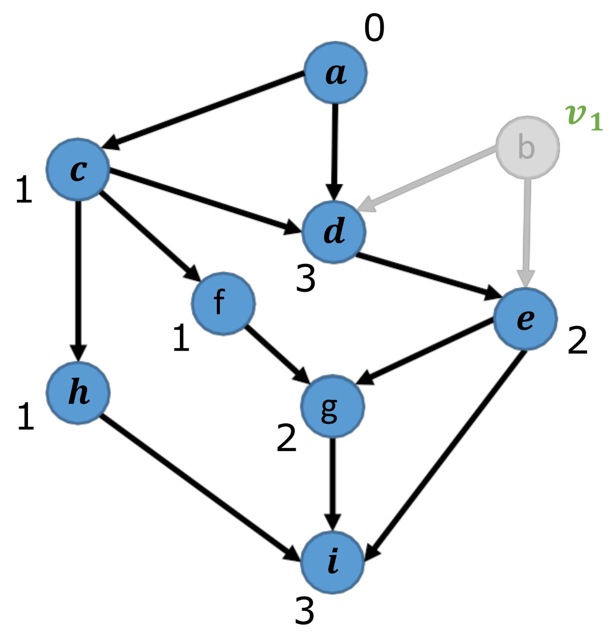
$\boldsymbol{i = 1}$



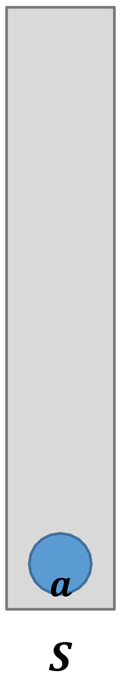
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
    if $\deg(u) = 0$ **then**
        $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
    $u \leftarrow S.pop()$
    Number u as vertex v_i
    $i \leftarrow i + 1$
    for each vertex v adjacent to u **do**
        $\deg(v) \leftarrow \deg(v) - 1$
        if $\deg(v) = 0$ **then**
            $S.push(v)$
if $i > n$ **then**
    return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



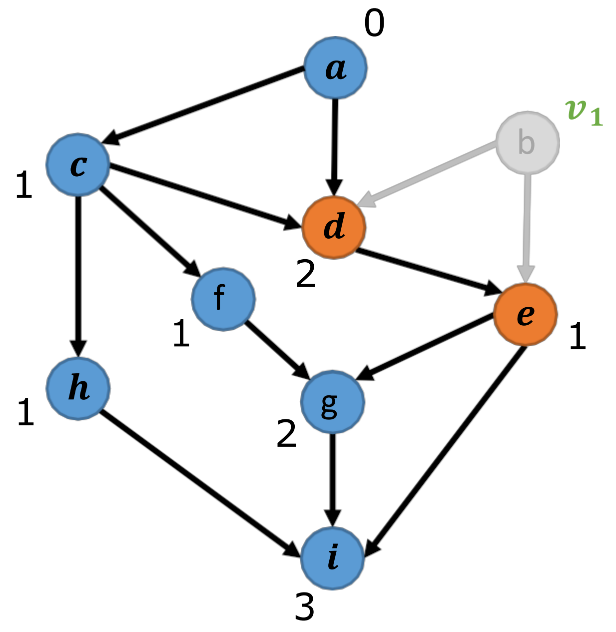
$i = 2$



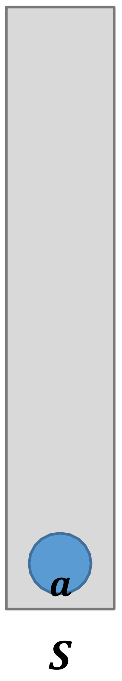
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
  if $\deg(u) = 0$ **then**
    $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
  $u \leftarrow S.pop()$
  Number u as vertex v_i
  $i \leftarrow i + 1$
  for each vertex v adjacent to u **do**
    $\deg(v) \leftarrow \deg(v) - 1$
    if $\deg(v) = 0$ **then**
      $S.push(v)$
if $i > n$ **then**
  return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



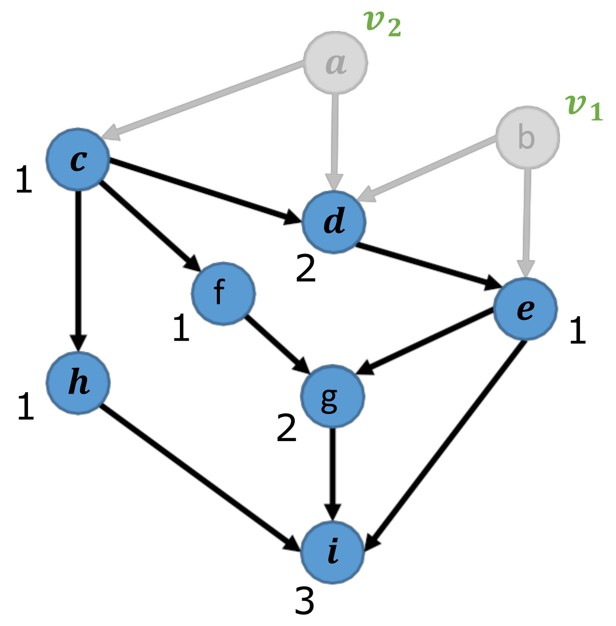
$i = 2$



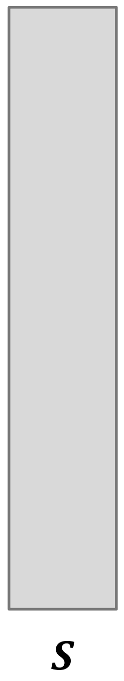
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$**:**
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
    if $\deg(u) = 0$ **then**
        $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
    $u \leftarrow S.pop()$
    Number u as vertex v_i
    $i \leftarrow i + 1$
    for each vertex v adjacent to u **do**
        $\deg(v) \leftarrow \deg(v) - 1$
        if $\deg(v) = 0$ **then**
            $S.push(v)$
if $i > n$ **then**
    return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



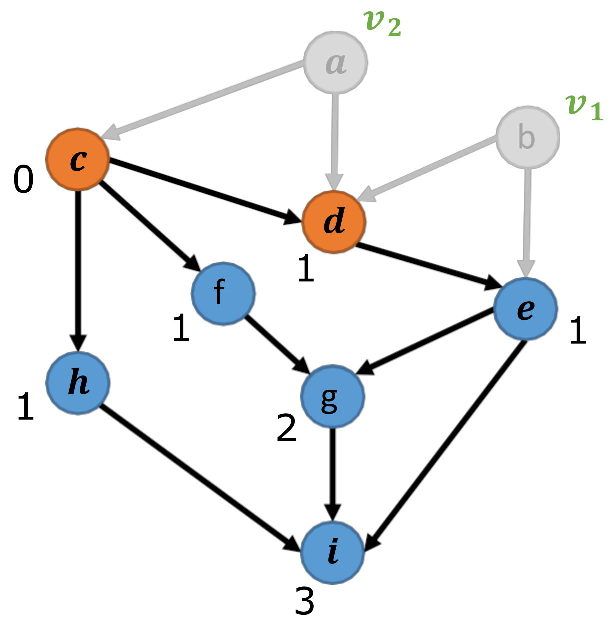
$i = 3$



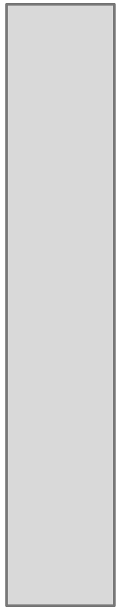
Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort(G)}$:
Input: Digraph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices
Output: Topological ordering of $\boldsymbol{G}$ or an indication of a directed cycle

$\boldsymbol{S} \leftarrow$ empty stack
for each vertex $\boldsymbol{u}$ in $\boldsymbol{G}$ **do**
    if deg$(\boldsymbol{u})$ = 0 **then**
        $\boldsymbol{S.push(u)}$
$\boldsymbol{i \leftarrow 1}$
while $\boldsymbol{S}$ is not empty **do**
    $\boldsymbol{u \leftarrow S.pop()}$
    Number $\boldsymbol{u}$ as vertex $\boldsymbol{v_i}$
    $\boldsymbol{i \leftarrow i+1}$
    for each vertex $\boldsymbol{v}$ adjacent to $\boldsymbol{u}$ **do**
        deg$(\boldsymbol{v}) \leftarrow$ deg$(\boldsymbol{v}) - \boldsymbol{1}$
        if deg$(\boldsymbol{v})$ = $\boldsymbol{0}$ **then**
            $\boldsymbol{S.push(v)}$
if $\boldsymbol{i > n}$ **then**
    return $\boldsymbol{v_1, v_2, \ldots, v_n}$
return "$\boldsymbol{G}$ has a directed cycle"



$\boldsymbol{i = 3}$

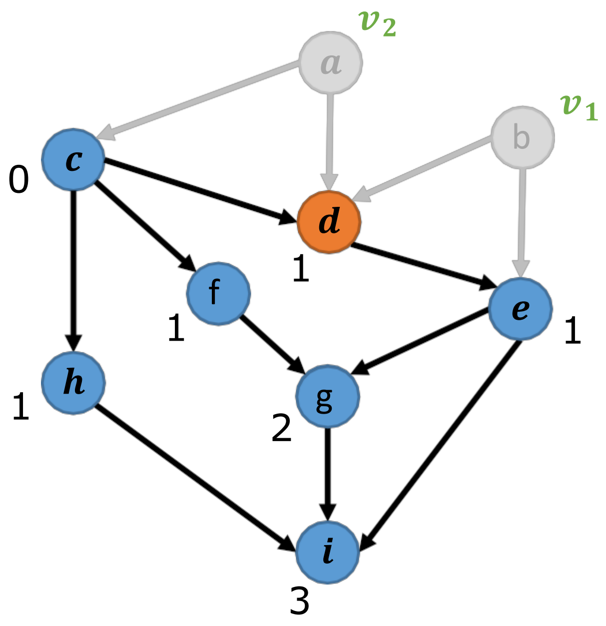


$\boldsymbol{S}$

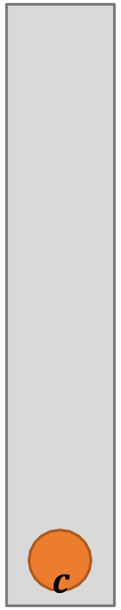
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
  if $\deg(u) = 0$ **then**
    $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
  $u \leftarrow S.pop()$
  Number u as vertex v_i
  $i \leftarrow i + 1$
  for each vertex v adjacent to u **do**
    $\deg(v) \leftarrow \deg(v) - 1$
    if $\deg(v) = 0$ **then**
      $S.push(v)$
if $i > n$ **then**
  return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



$i = 3$

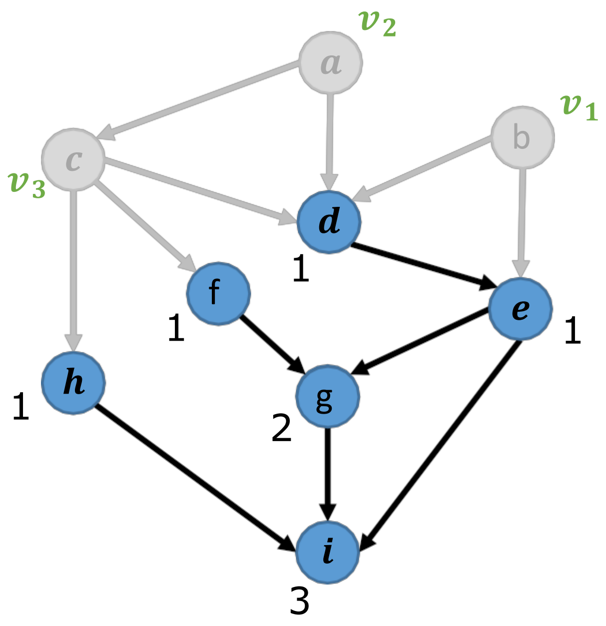


S

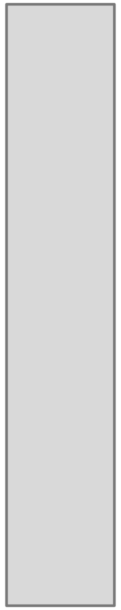
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
  if $\deg(u) = 0$ **then**
    $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
  $u \leftarrow S.pop()$
  Number u as vertex v_i
  $i \leftarrow i + 1$
  for each vertex v adjacent to u **do**
    $\deg(v) \leftarrow \deg(v) - 1$
    if $\deg(v) = 0$ **then**
      $S.push(v)$
if $i > n$ **then**
  return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



$i = 4$

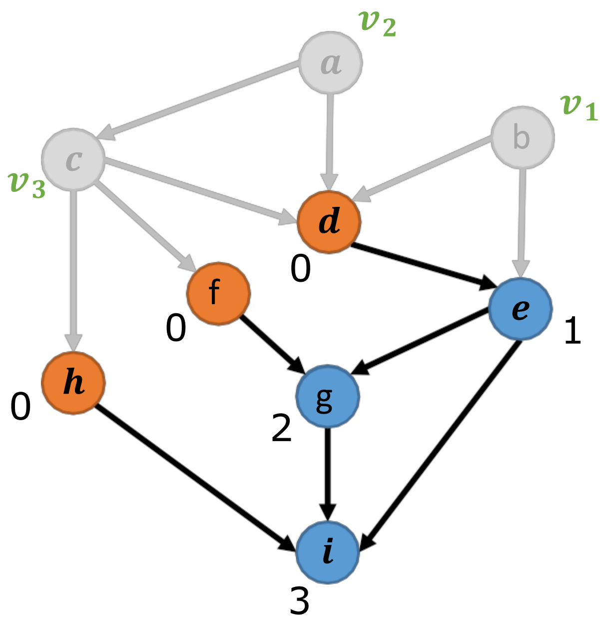


S

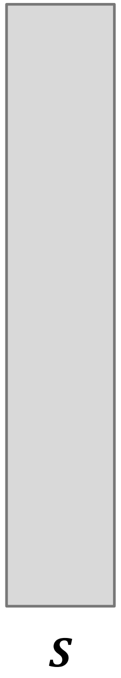
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$**:**
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
  if $\deg(u) = 0$ **then**
    $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
  $u \leftarrow S.pop()$
  Number u as vertex v_i
  $i \leftarrow i + 1$
  for each vertex v adjacent to u **do**
    $\deg(v) \leftarrow \deg(v) - 1$
    if $\deg(v) = 0$ **then**
      $S.push(v)$
if $i > n$ **then**
  return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



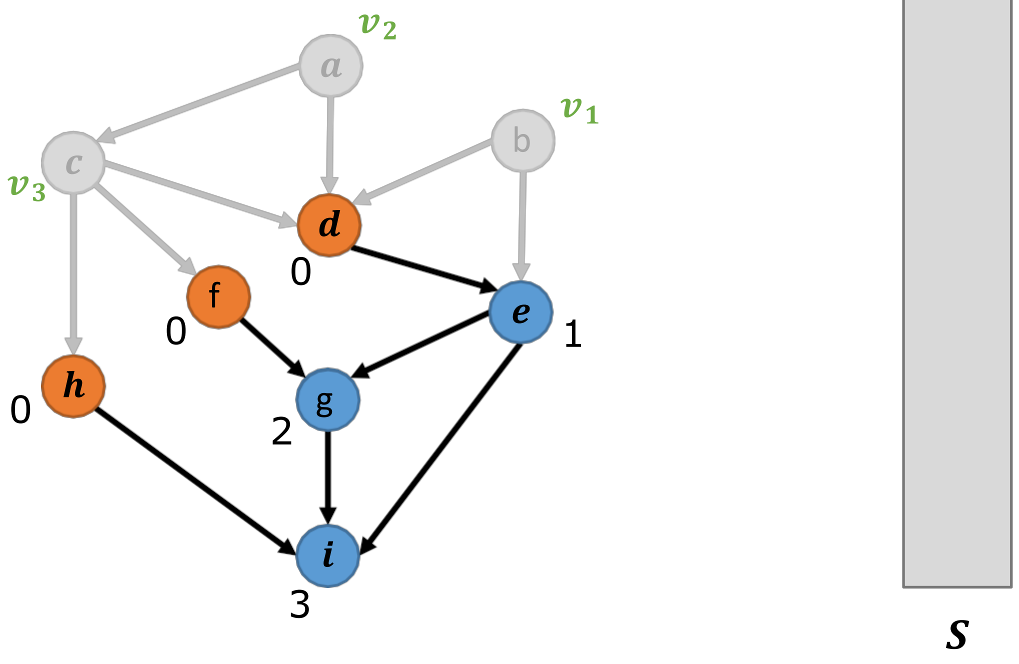
$i = 4$



Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
    if $\deg(u) = 0$ **then**
        $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
    $u \leftarrow S.pop()$
    Number u as vertex v_i
    $i \leftarrow i + 1$
    for each vertex v adjacent to u **do**
        $\deg(v) \leftarrow \deg(v) - 1$
        if $\deg(v) = 0$ **then**
            $S.push(v)$
if $i > n$ **then**
    return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"

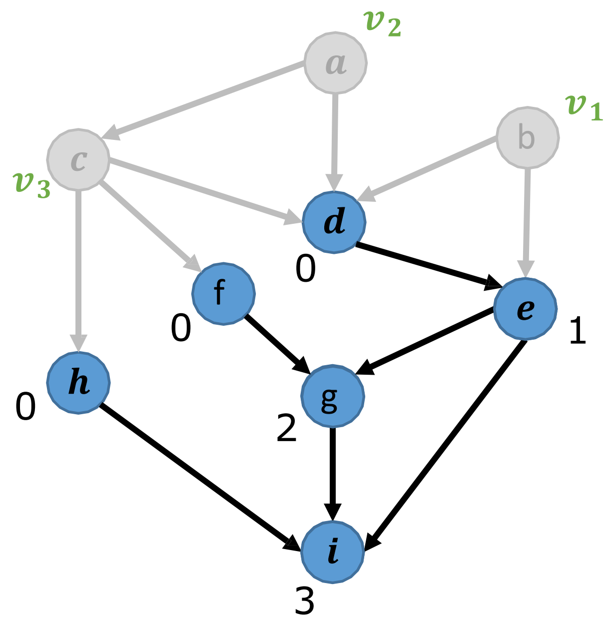


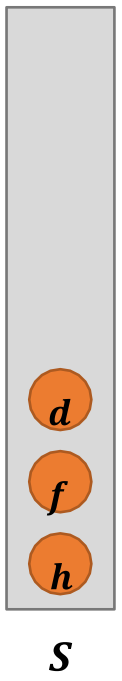
$i = 4$

Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort}(\boldsymbol{G})$**:**
Input: Digraph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices
Output: Topological ordering of $\boldsymbol{G}$ or an indication of a directed cycle

$\boldsymbol{S} \leftarrow$ empty stack
for each vertex $\boldsymbol{u}$ in $\boldsymbol{G}$ **do**
    if deg$(\boldsymbol{u})$ = 0 **then**
        $\boldsymbol{S.push(u)}$
$\boldsymbol{i} \leftarrow \boldsymbol{1}$
while $\boldsymbol{S}$ is not empty **do**
    $\boldsymbol{u} \leftarrow \boldsymbol{S.pop}()$
    Number $\boldsymbol{u}$ as vertex $\boldsymbol{v_i}$
    $\boldsymbol{i} \leftarrow \boldsymbol{i+1}$
    for each vertex $\boldsymbol{v}$ adjacent to $\boldsymbol{u}$ **do**
        deg$(\boldsymbol{v}) \leftarrow$ deg$(\boldsymbol{v}) - \boldsymbol{1}$
        if deg$(\boldsymbol{v})$ = $\boldsymbol{0}$ **then**
            $\boldsymbol{S.push(v)}$
if $\boldsymbol{i} > \boldsymbol{n}$ **then**
    return $\boldsymbol{v_1, v_2, \ldots, v_n}$
return "$\boldsymbol{G}$ has a directed cycle"



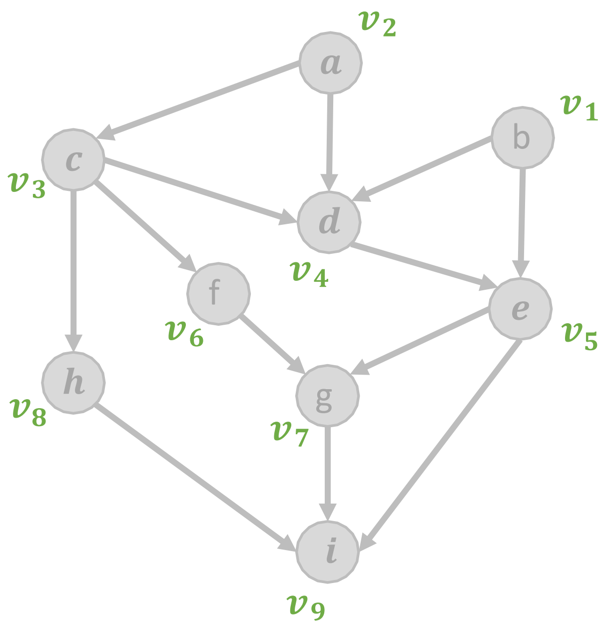


$\boldsymbol{i}$ = 4

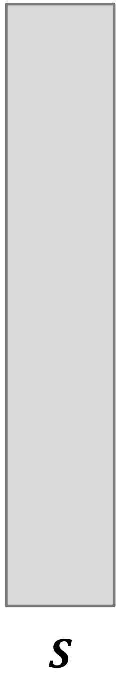
Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort(G)}$:
Input: Digraph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices
Output: Topological ordering of $\boldsymbol{G}$ or an indication of a directed cycle

$\boldsymbol{S} \leftarrow$ empty stack
for each vertex $\boldsymbol{u}$ in $\boldsymbol{G}$ **do**
  if $\deg(\boldsymbol{u}) = 0$ **then**
    $\boldsymbol{S.push(u)}$
$\boldsymbol{i \leftarrow 1}$
while $\boldsymbol{S}$ is not empty **do**
  $\boldsymbol{u \leftarrow S.pop()}$
  Number $\boldsymbol{u}$ as vertex $\boldsymbol{v_i}$
  $\boldsymbol{i \leftarrow i+1}$
  for each vertex $\boldsymbol{v}$ adjacent to $\boldsymbol{u}$ **do**
    $\deg(\boldsymbol{v}) \leftarrow \deg(\boldsymbol{v}) - \boldsymbol{1}$
    if $\deg(\boldsymbol{v}) = \boldsymbol{0}$ **then**
      $\boldsymbol{S.push(v)}$
if $\boldsymbol{i > n}$ **then**
  return $\boldsymbol{v_1, v_2, \ldots, v_n}$
return "$\boldsymbol{G}$ has a directed cycle"



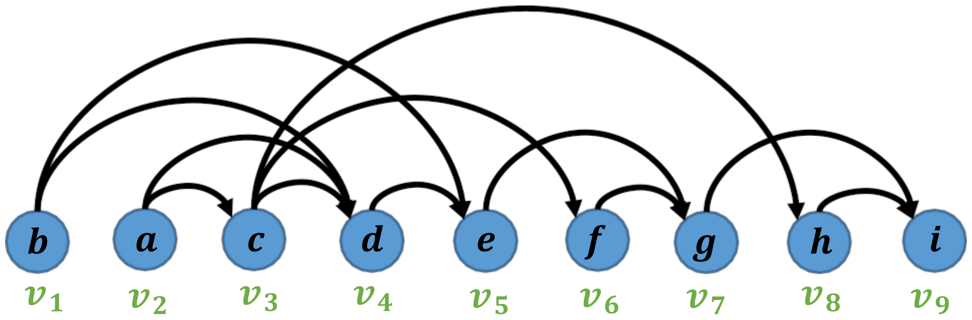
$\boldsymbol{i = 10}$



Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

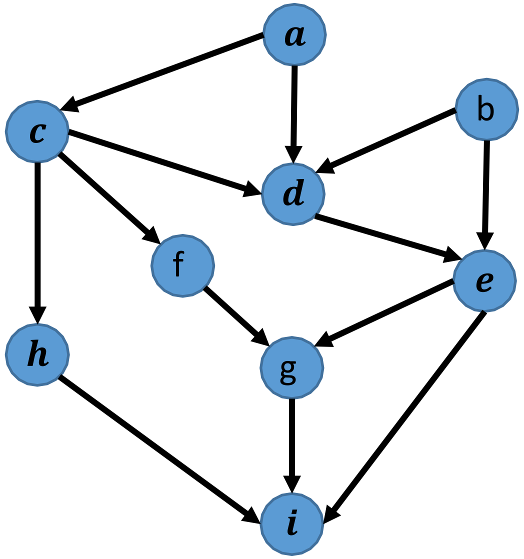
$S \leftarrow$ empty stack
for each vertex u in G **do**
    if $\deg(u) = 0$ **then**
        $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
    $u \leftarrow S.pop()$
    Number u as vertex v_i
    $i \leftarrow i + 1$
    for each vertex v adjacent to u **do**
        $\deg(v) \leftarrow \deg(v) - 1$
        if $\deg(v) = 0$ **then**
            $S.push(v)$
if $i > n$ **then**
    return $v_1, v_2, \dots, v_n$
return "G has a directed cycle"

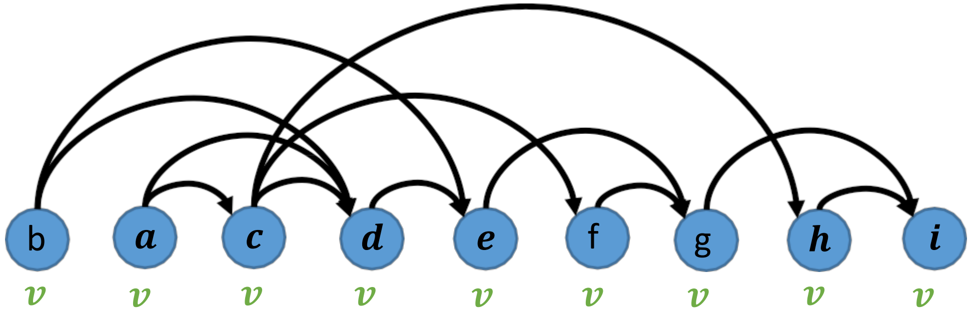


Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort}(\boldsymbol{G})$**:**
Input: Digraph $\boldsymbol{G}$ with $\boldsymbol{n}$ vertices
Output: Topological ordering of $\boldsymbol{G}$ or an indication of a directed cycle

$\boldsymbol{S} \leftarrow$ empty stack
for each vertex $\boldsymbol{u}$ in $\boldsymbol{G}$ **do**
    if $\deg(\boldsymbol{u}) = 0$ **then**
        $\boldsymbol{S.push}(\boldsymbol{u})$
$\boldsymbol{i} \leftarrow \mathbf{1}$
while $\boldsymbol{S}$ is not empty **do**
    $\boldsymbol{u} \leftarrow \boldsymbol{S.pop}()$
    Number $\boldsymbol{u}$ as vertex $\boldsymbol{v_i}$
    $\boldsymbol{i} \leftarrow \boldsymbol{i} + \mathbf{1}$
    for each vertex $\boldsymbol{v}$ adjacent to $\boldsymbol{u}$ **do**
        $\deg(\boldsymbol{v}) \leftarrow \deg(\boldsymbol{v}) - \mathbf{1}$
        if $\deg(\boldsymbol{v}) = \mathbf{0}$ **then**
            $\boldsymbol{S.push}(\boldsymbol{v})$
if $\boldsymbol{i} > \boldsymbol{n}$ **then**
    return $\boldsymbol{v_1, v_2, \ldots, v_n}$
return "$\boldsymbol{G}$ has a directed cycle"





Topological Sort Algorithm (Iterative) Running Time

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
    if $\deg(u) = 0$ **then**
        $S.push(u)$
$O(n)$
$i \leftarrow 1$
while S is not empty **do** $O(n)$
    $u \leftarrow S.pop()$
    Number u as vertex v_i
    $i \leftarrow i + 1$
    for each vertex v adjacent to u **do**
        $\deg(v) \leftarrow \deg(v) - 1$
        if $\deg(v) = 0$ **then**
            $S.push(v)$
$O(\deg(u))$
$O(m)$ since sum of all degrees is $O(m)$
$O(n + m)$
Total Run-time
if $i > n$ **then**
    return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"

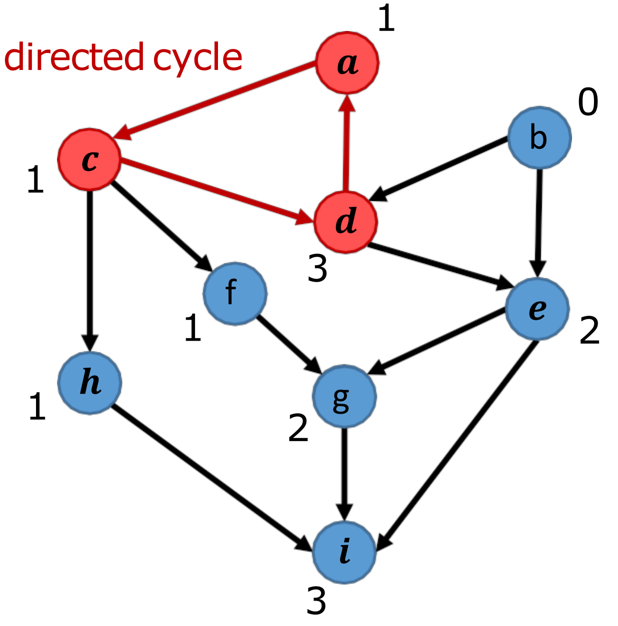
Topological Sort Algorithm (Iterative)

TopologicalSort(G):
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

Example if it is not a DAG. A cycle exists.

$S \leftarrow$ empty stack
for each vertex u in G **do**
    if $\deg(u) = 0$ **then**
        $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
    $u \leftarrow S.pop()$
    Number u as vertex v_i
    $i \leftarrow i + 1$
    for each vertex v adjacent to u **do**
        $\deg(v) \leftarrow \deg(v) - 1$
        if $\deg(v) = 0$ **then**
            $S.push(v)$
if $i > n$ **then**
    return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"

Consider if G has a directed cycle



Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort(G)}$:

Input: Digraph G with n vertices

Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack

for each vertex u in G **do**

  if $\deg(u) = 0$ **then**

    $S.push(u)$

$i \leftarrow 1$

while S is not empty **do**

  $u \leftarrow S.pop()$

  Number u as vertex v_i

  $i \leftarrow i + 1$

  for each vertex v adjacent to u **do**

    $\deg(v) \leftarrow \deg(v) - 1$

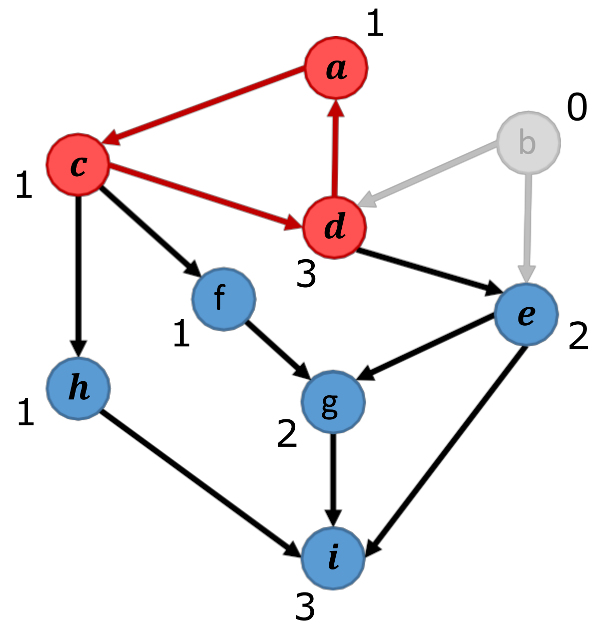
    if $\deg(v) = 0$ **then**

      $S.push(v)$

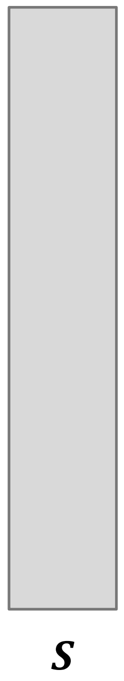
if $i > n$ **then**

  return $v_1, v_2, \ldots, v_n$

return "G has a directed cycle"



$i = 1$



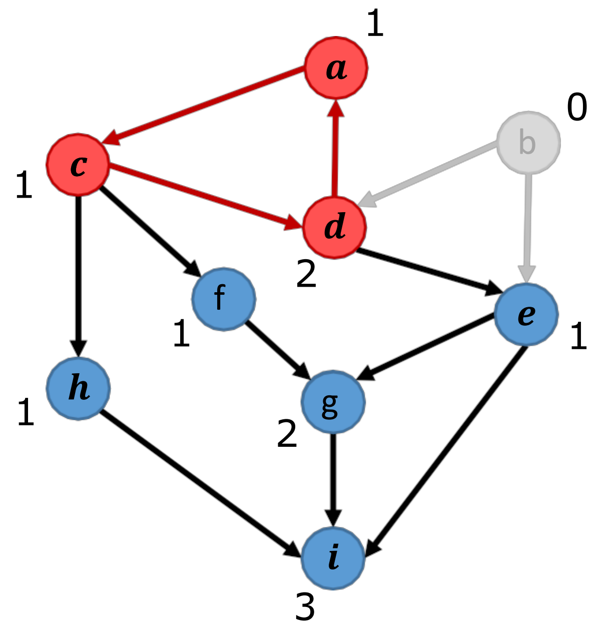
Topological Sort Algorithm (Iterative)

$\boldsymbol{TopologicalSort(G)}$:

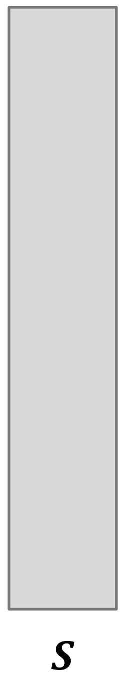
Input: Digraph G with n vertices

Output: Topological ordering of G or an indication of a directed cycle

```
S ← empty stack
for each vertex u in G do
    if deg(u) = 0 then
        S.push(u)
i ← 1
while S is not empty do
    u ← S.pop()
    Number u as vertex v_i
    i ← i + 1
    for each vertex v adjacent to u do
        deg(v) ← deg(v) − 1
        if deg(v) = 0 then
            S.push(v)
if i > n then
    return v_1, v_2, ..., v_n
return "G has a directed cycle"
```



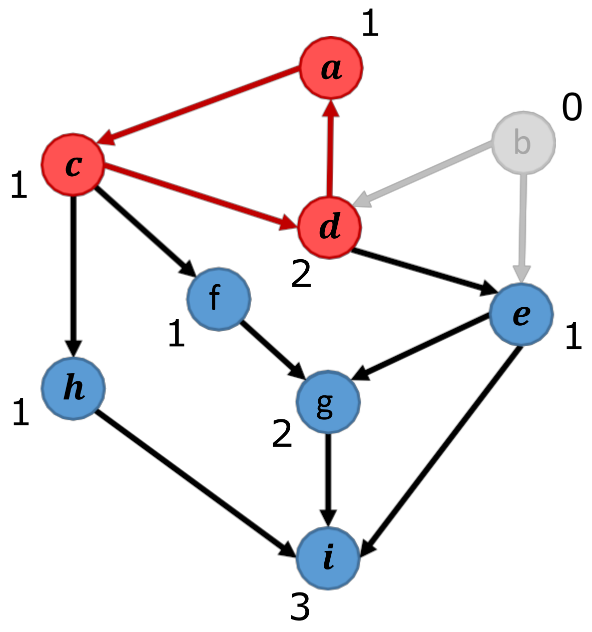
$i = 2$



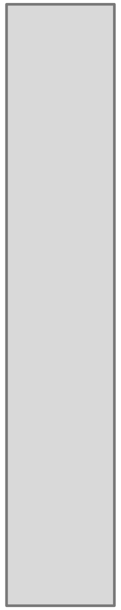
Topological Sort Algorithm (Iterative)

$TopologicalSort(G)$:
Input: Digraph G with n vertices
Output: Topological ordering of G or an indication of a directed cycle

$S \leftarrow$ empty stack
for each vertex u in G **do**
  if $\deg(u) = 0$ **then**
    $S.push(u)$
$i \leftarrow 1$
while S is not empty **do**
  $u \leftarrow S.pop()$
  Number u as vertex v_i
  $i \leftarrow i + 1$
  for each vertex v adjacent to u **do**
    $\deg(v) \leftarrow \deg(v) - 1$
    if $\deg(v) = 0$ **then**
      $S.push(v)$
if $i > n$ **then**
  return $v_1, v_2, \ldots, v_n$
return "G has a directed cycle"



$i = 2$

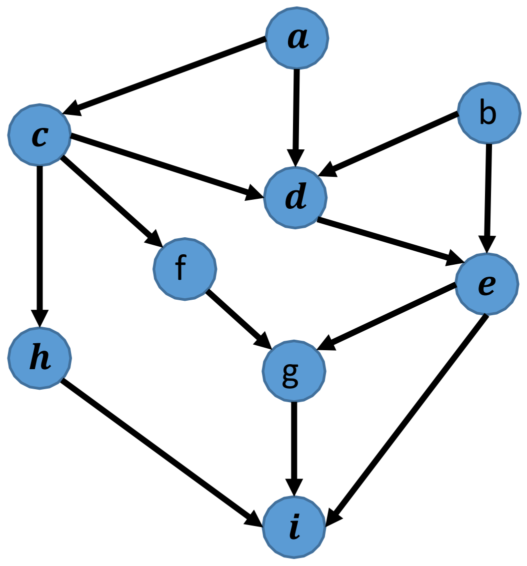


S

Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at n

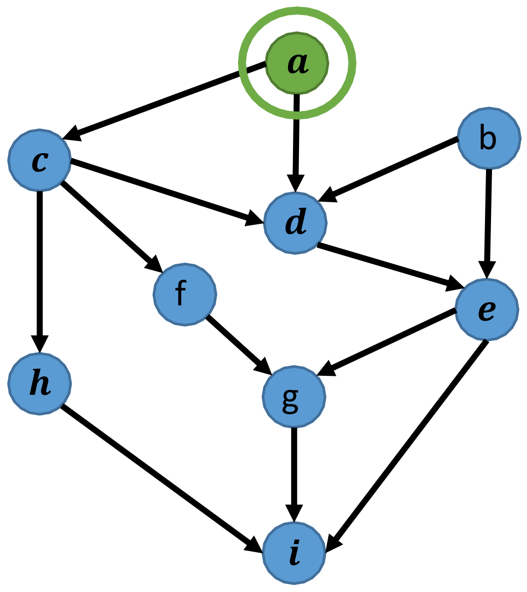
- The DFS reverse postorder numbering is a **topological order numbering**



Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at n

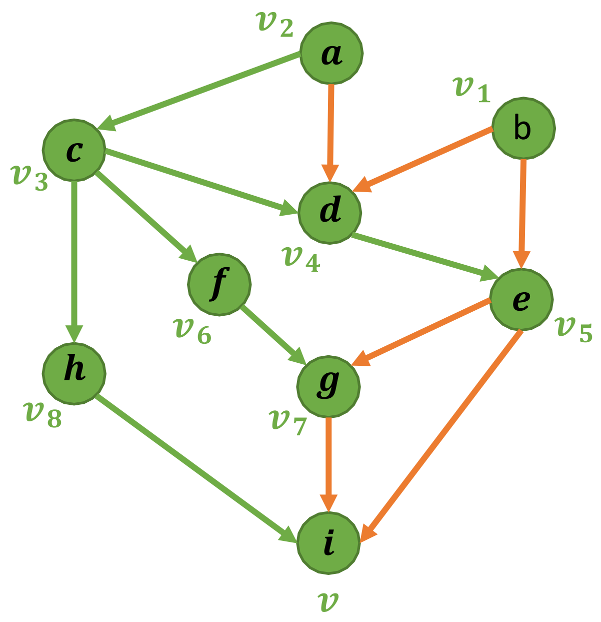
- The DFS reverse postorder numbering is a **topological order numbering**



Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at n

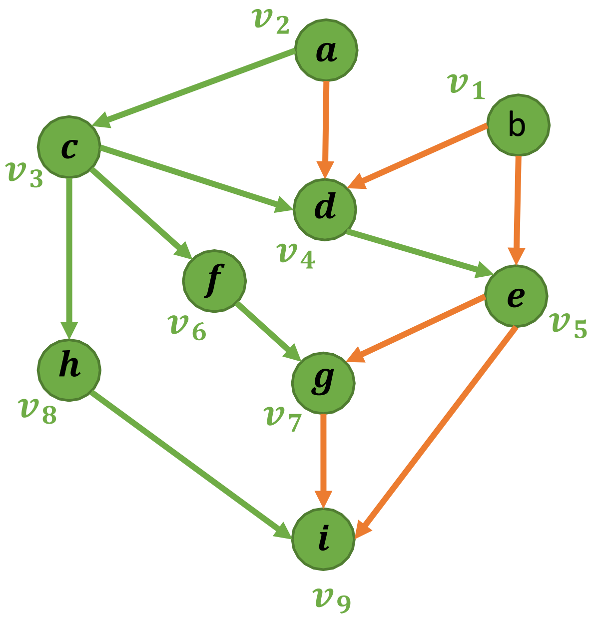
- The DFS reverse postorder numbering is a **topological order numbering**

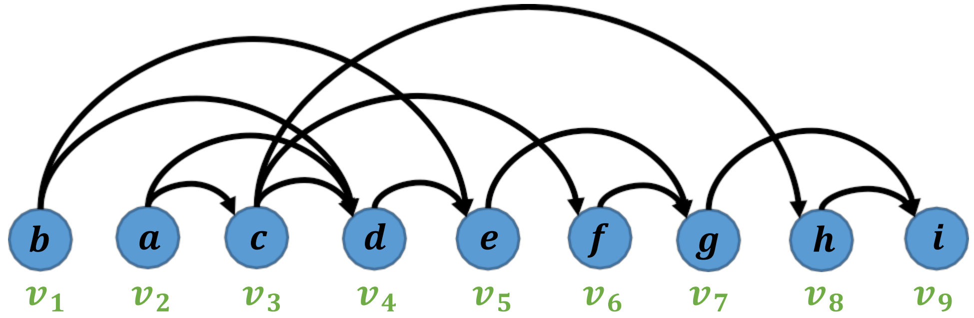


Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at n

- The DFS reverse postorder numbering is a **topological order numbering**





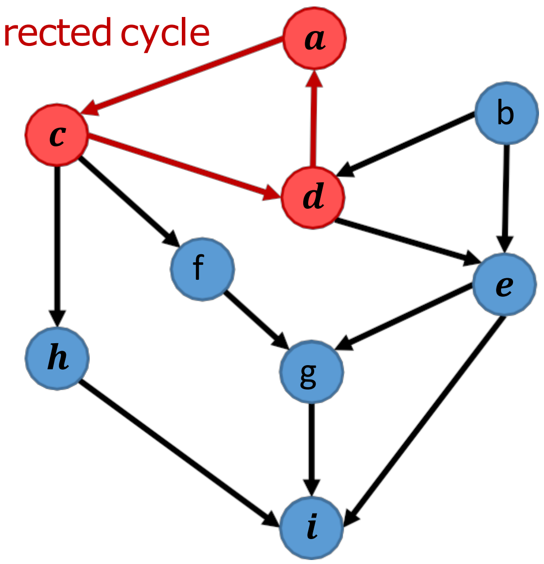
- Running time is $O(n+m)$

Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at n

- The DFS reverse postorder numbering is a **topological order numbering**

Consider if G has a directed cycle

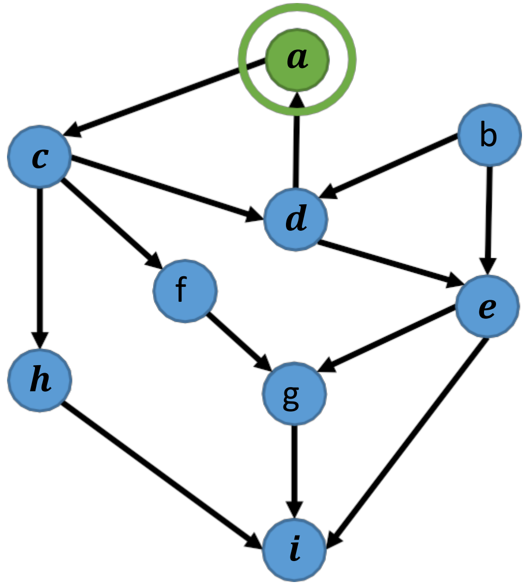


- Running time is $O(n + m)$

Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at $\boldsymbol{n}$

- The DFS reverse postorder numbering is a **topological order numbering**

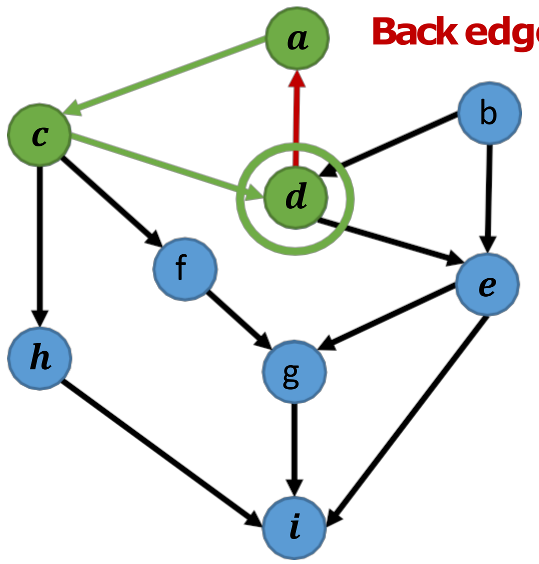


- Running time is $\boldsymbol{O}(\boldsymbol{n}+\boldsymbol{m})$

Topological Sort using DFS Reverse Postorder

- **DFS Postorder:** Assign a vertex numbering when it has no more unexplored outgoing edges
- **Reverse postorder** numbering is when numbering starts at n

- The DFS reverse postorder numbering is a **topological order numbering**

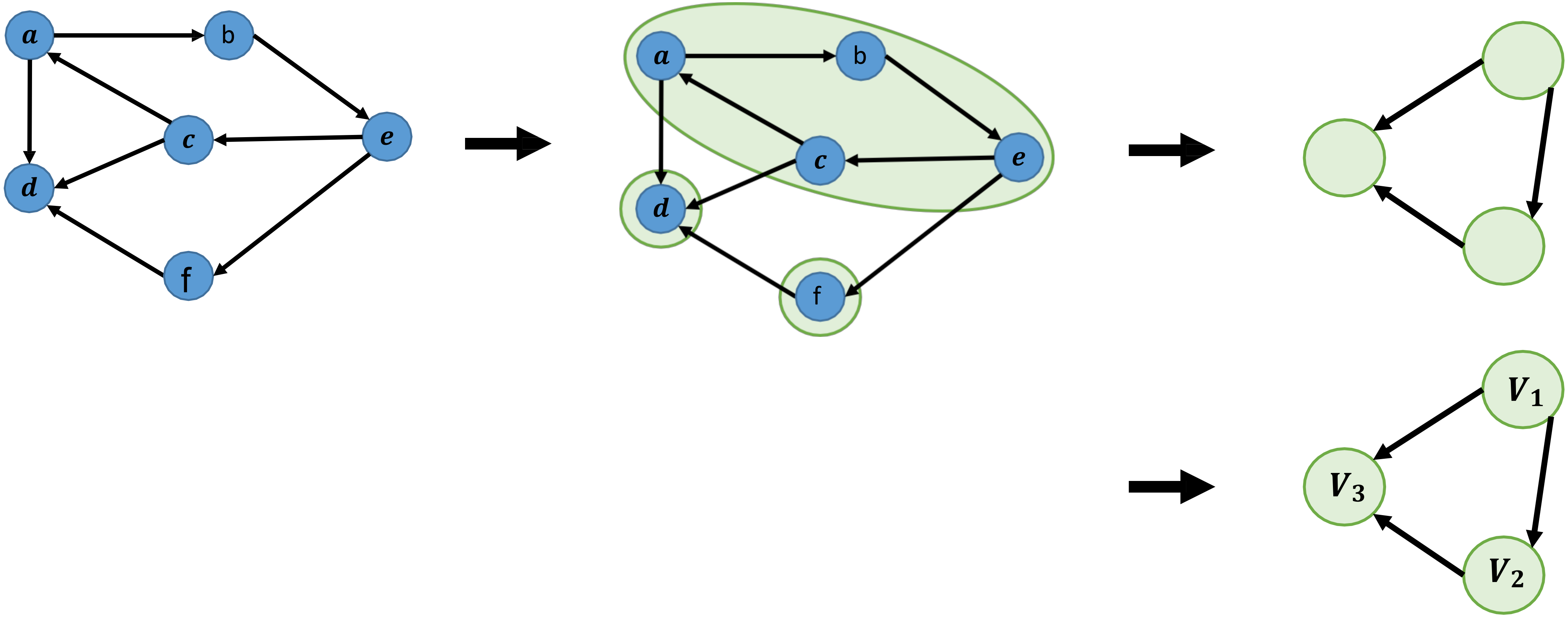


Back edge means a directed cycle was detected

- Running time is $O(n + m)$

Topological Sort and SCCs

- Compute the **strongly connected components** to produce a **reduced directed acyclic graph**
- Sort the directed acyclic graph using **topological sort**
- Note: both post order numberings and topological numberings may not be unique



Time Complexity of DFS Applications

Theorem: The time complexity of DFS traversal for a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ for

- Testing whether $\boldsymbol{G}$ is connected
- Computing a spanning forest of $\boldsymbol{G}$
- Computing a path between two vertices in $\boldsymbol{G}$ or reporting no path exists
- Computing a cycle in $\boldsymbol{G}$ or reporting that no cycles exist
- Identifying the strongly connected components of $\boldsymbol{G}$
- Computing a topological sort of $\boldsymbol{G}$

is $\boldsymbol{O}(\boldsymbol{n} + \boldsymbol{m})$ where $\boldsymbol{n} = |\boldsymbol{V}|$ and $\boldsymbol{m} = |\boldsymbol{E}|$.