CSE 105:
Data Structures and Algorithms-I
(Part 2)

Instructor

Dr Md Monirul Islam

Graph Searching

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

$\Theta(V)$

$\Theta(V)$ EXCLUDING the time required for DFS-VISIT().

Review

How many times DFS-Visit() is called?

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

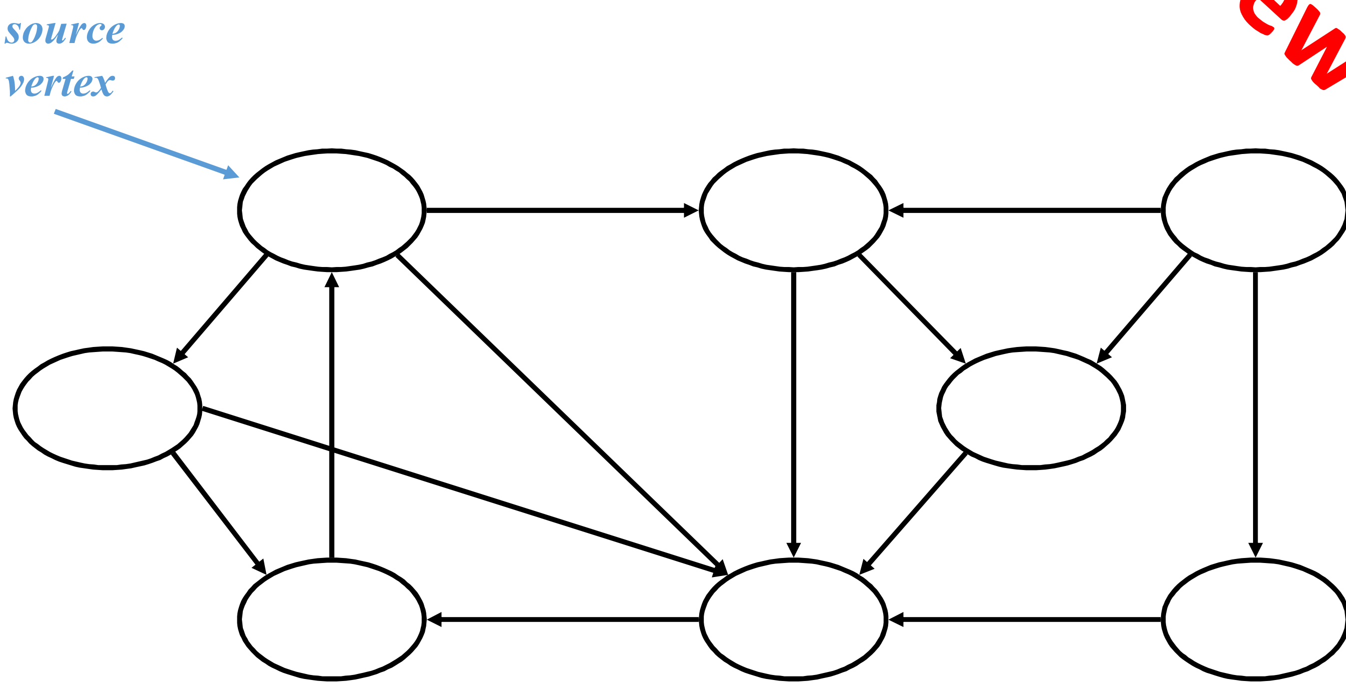
$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

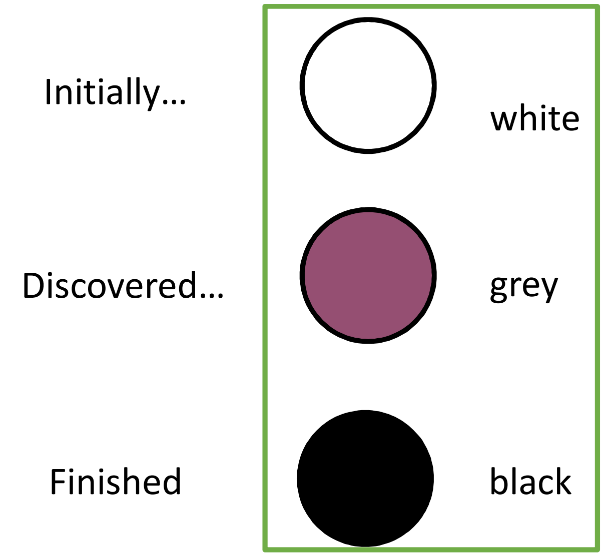
$\Theta(V + E)$

- The procedure DFS-VISIT is called exactly once for each vertex since:
  - the vertex u on which DFS-VISIT() is invoked must be white
  - the first thing DFS-VISIT does is paint vertex u gray

DFS Example

Review

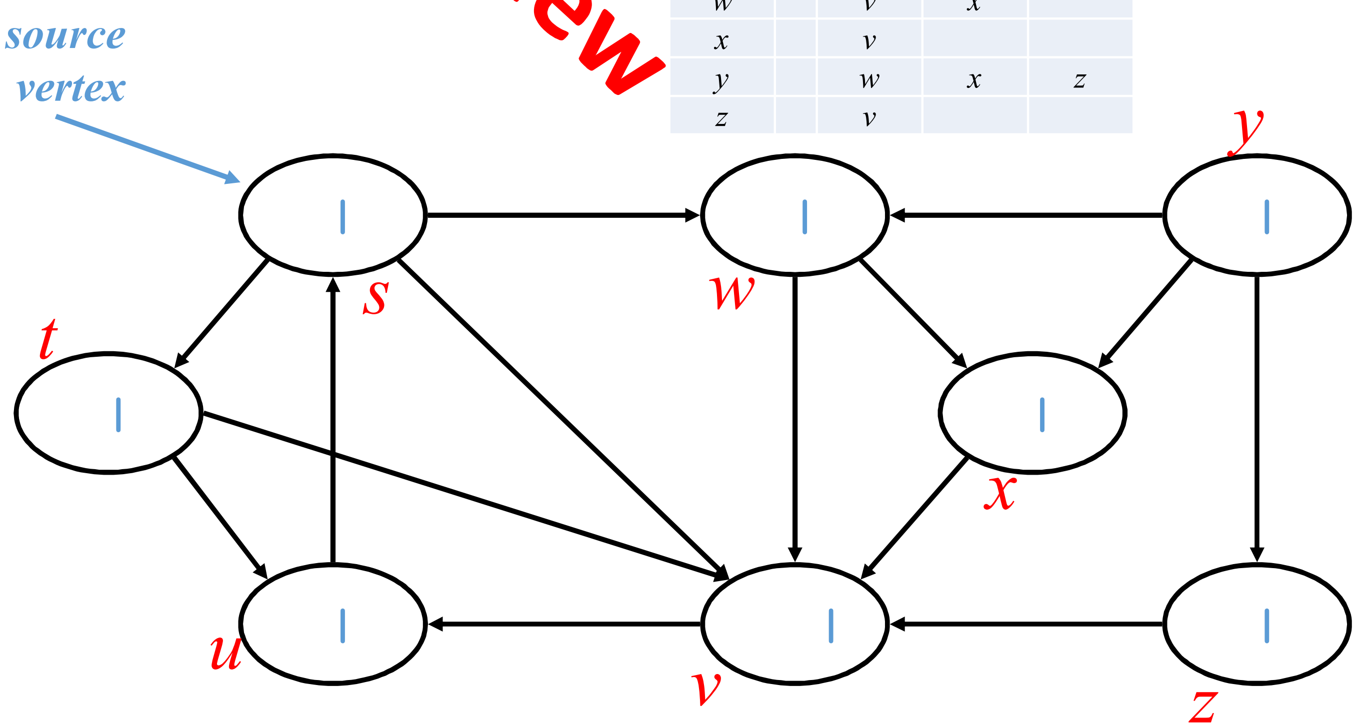


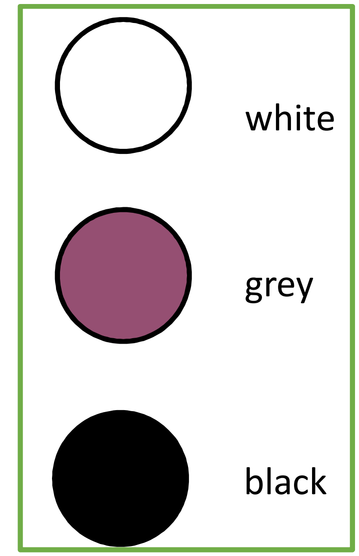


DFS Example

Review

Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		

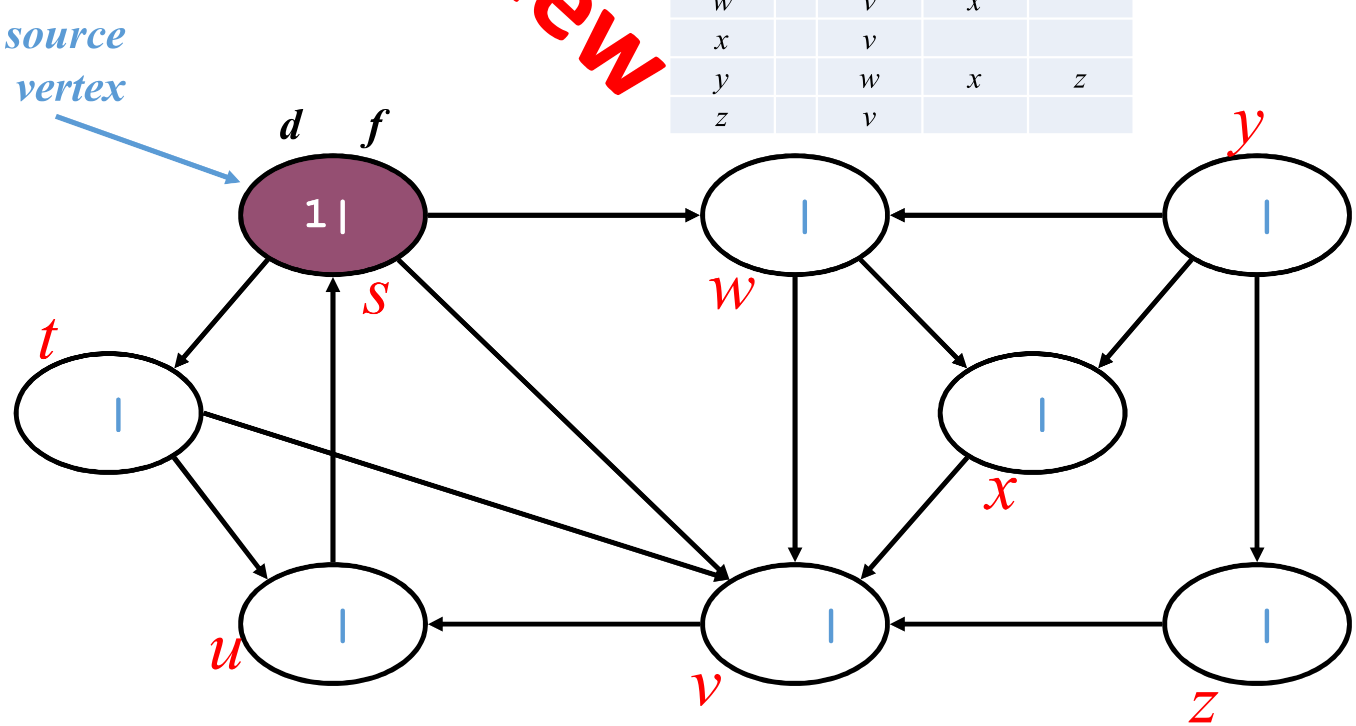


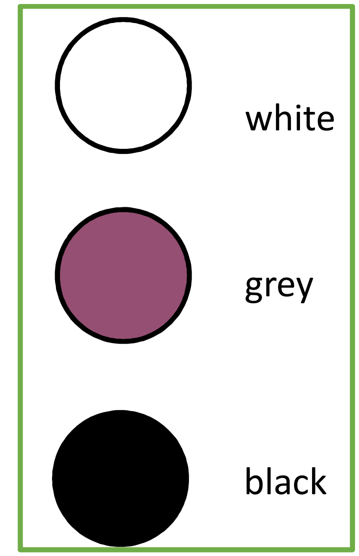


DFS Example

Review

Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		

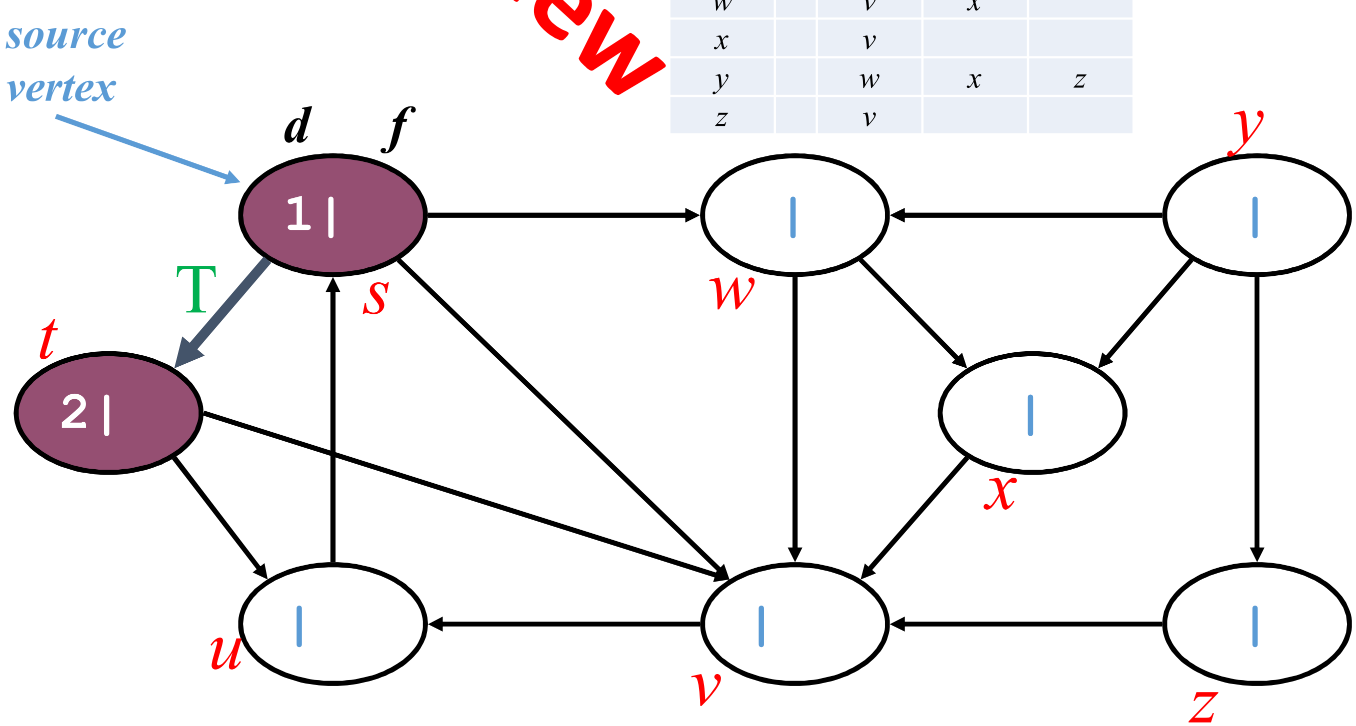


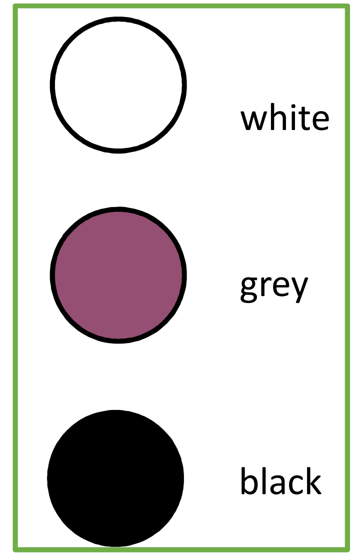


DFS Example

Review

Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		

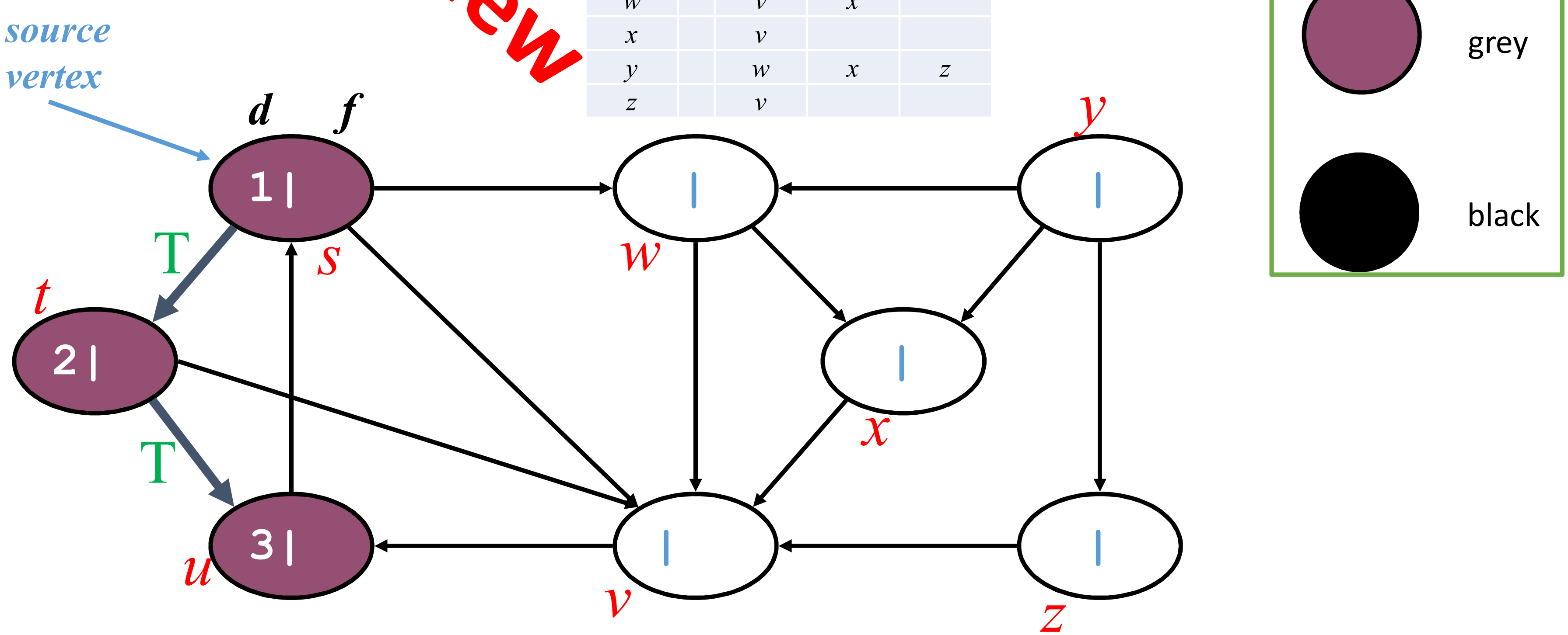




DFS Example

Review

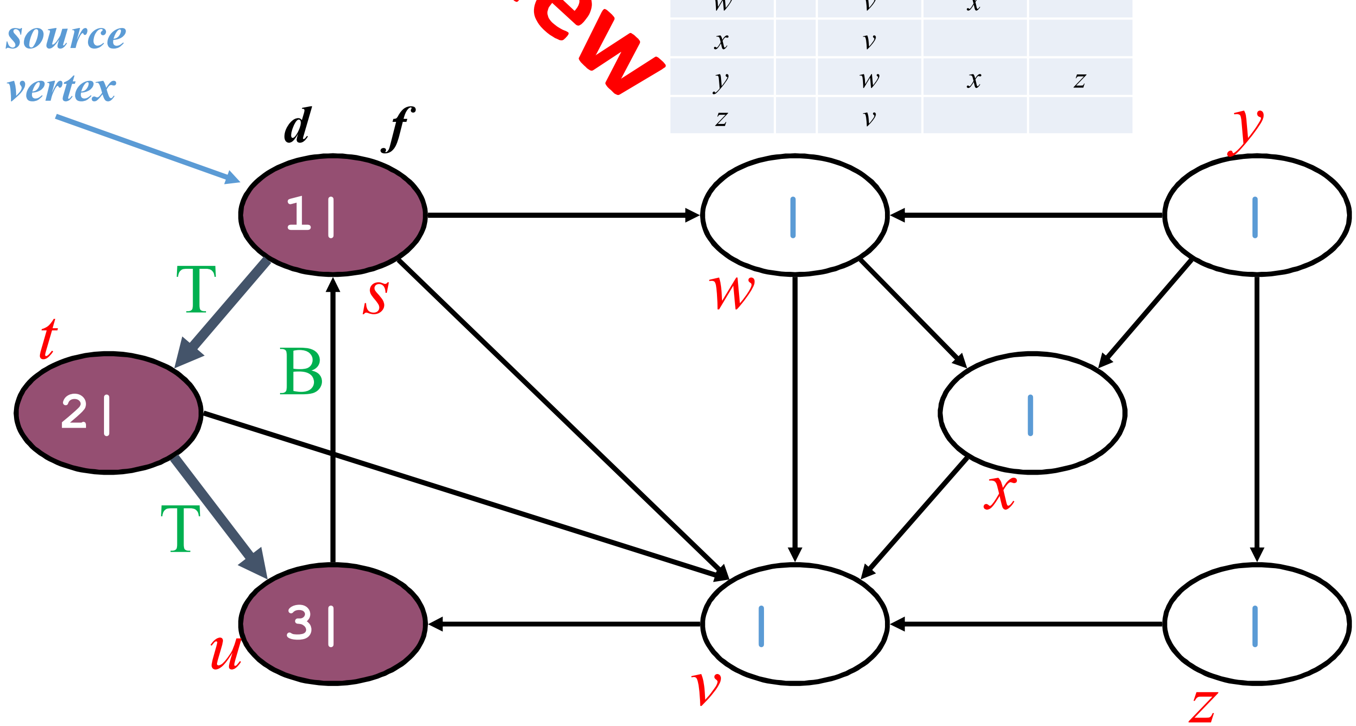
Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		

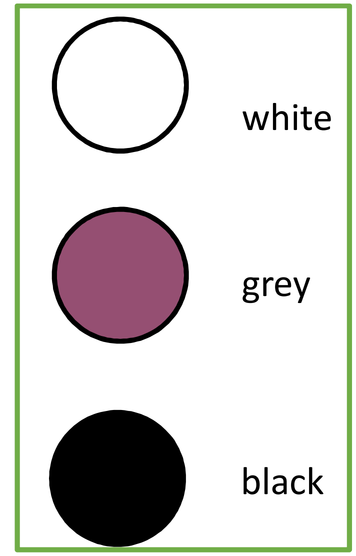


DFS Example

Review

Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		

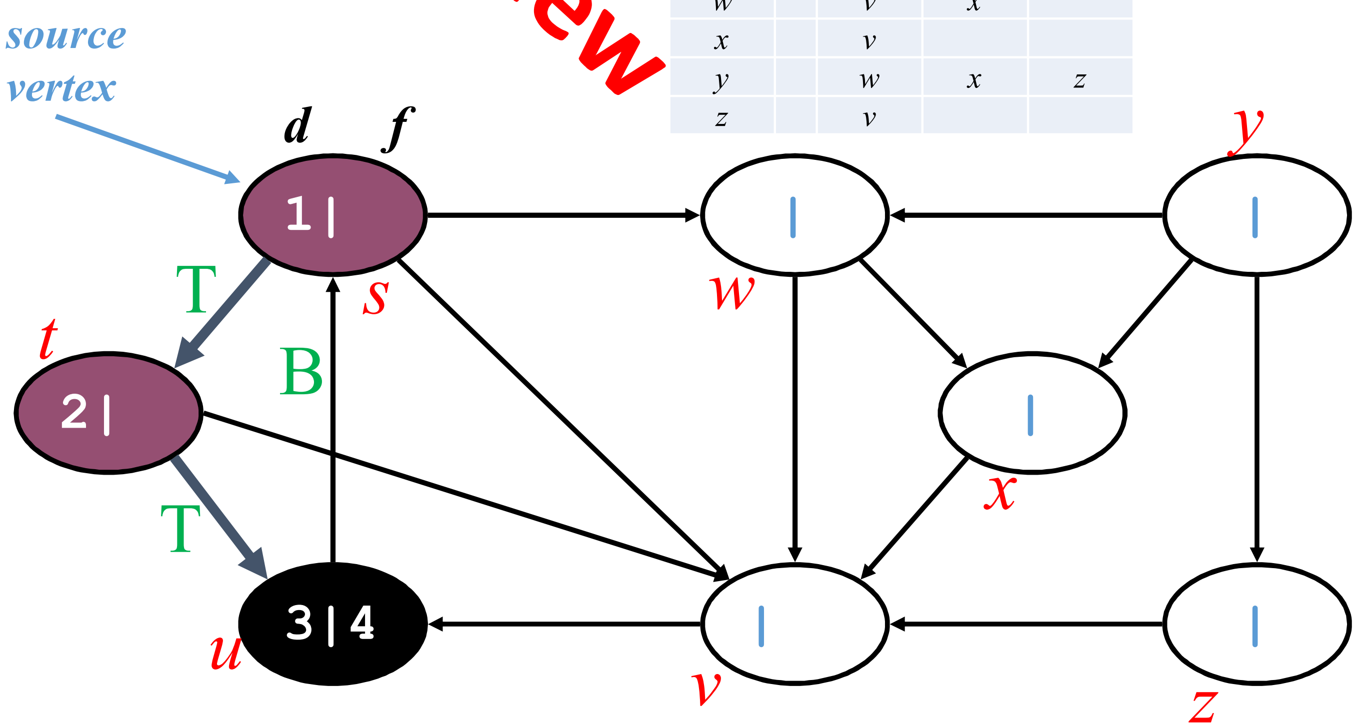


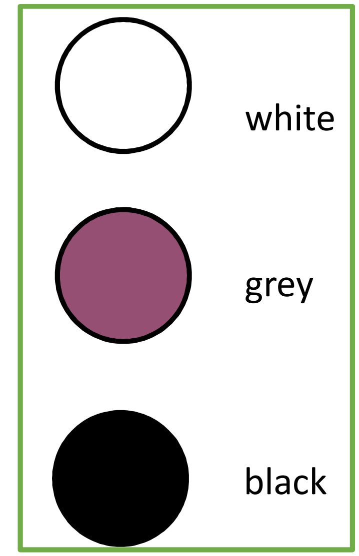


DFS Example

Review

Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		

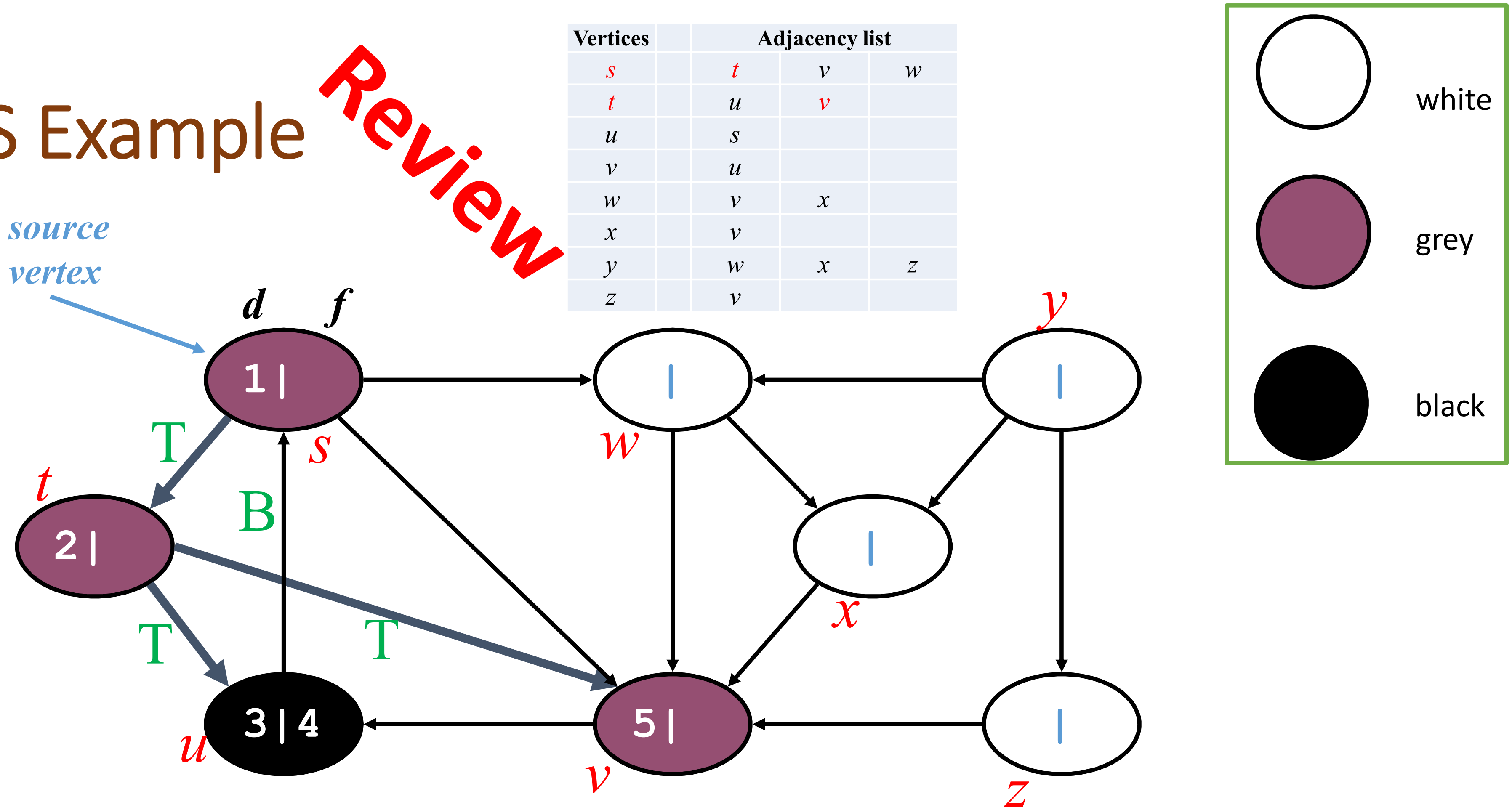




DFS Example

Review

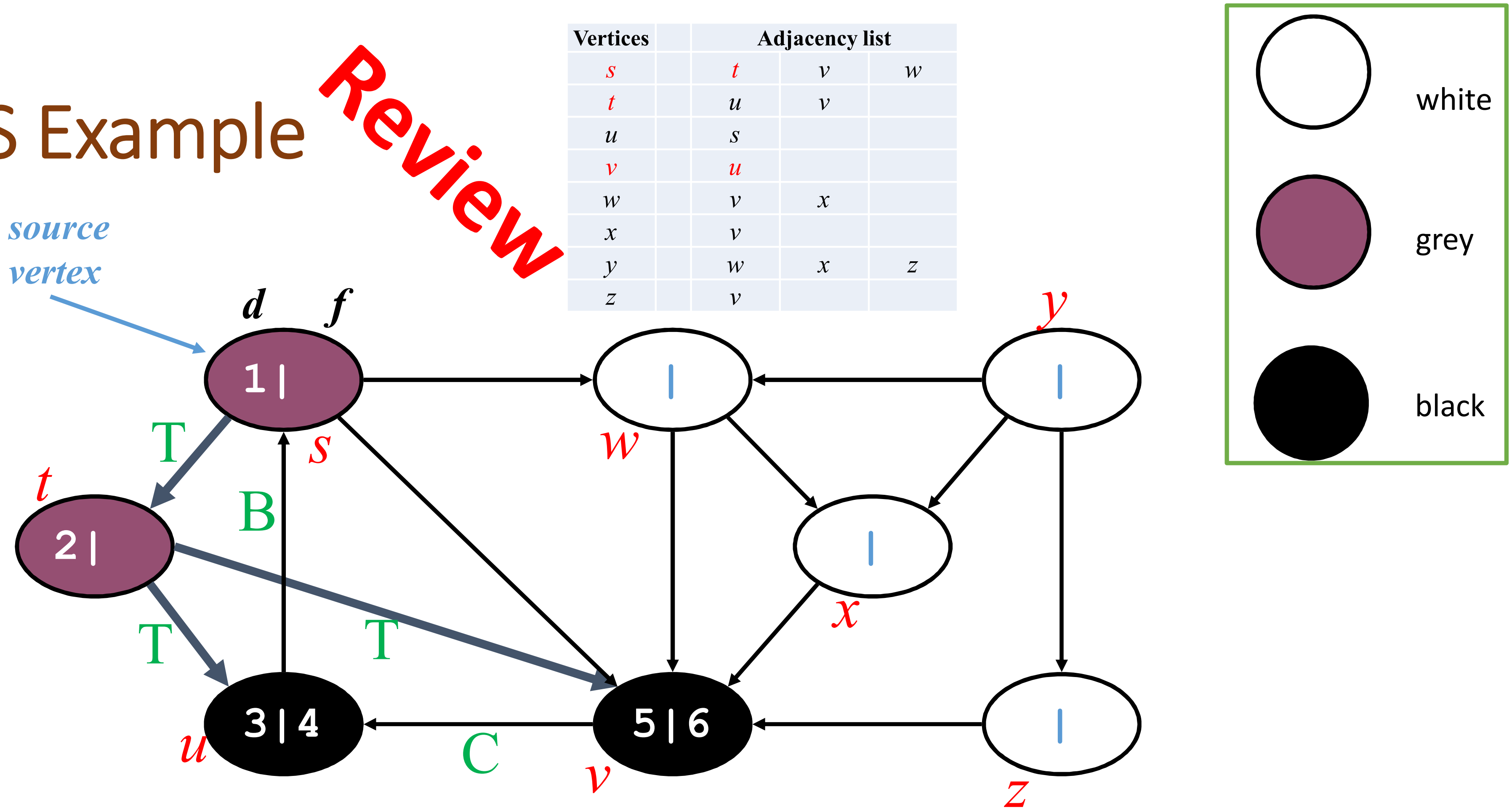
Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		



DFS Example

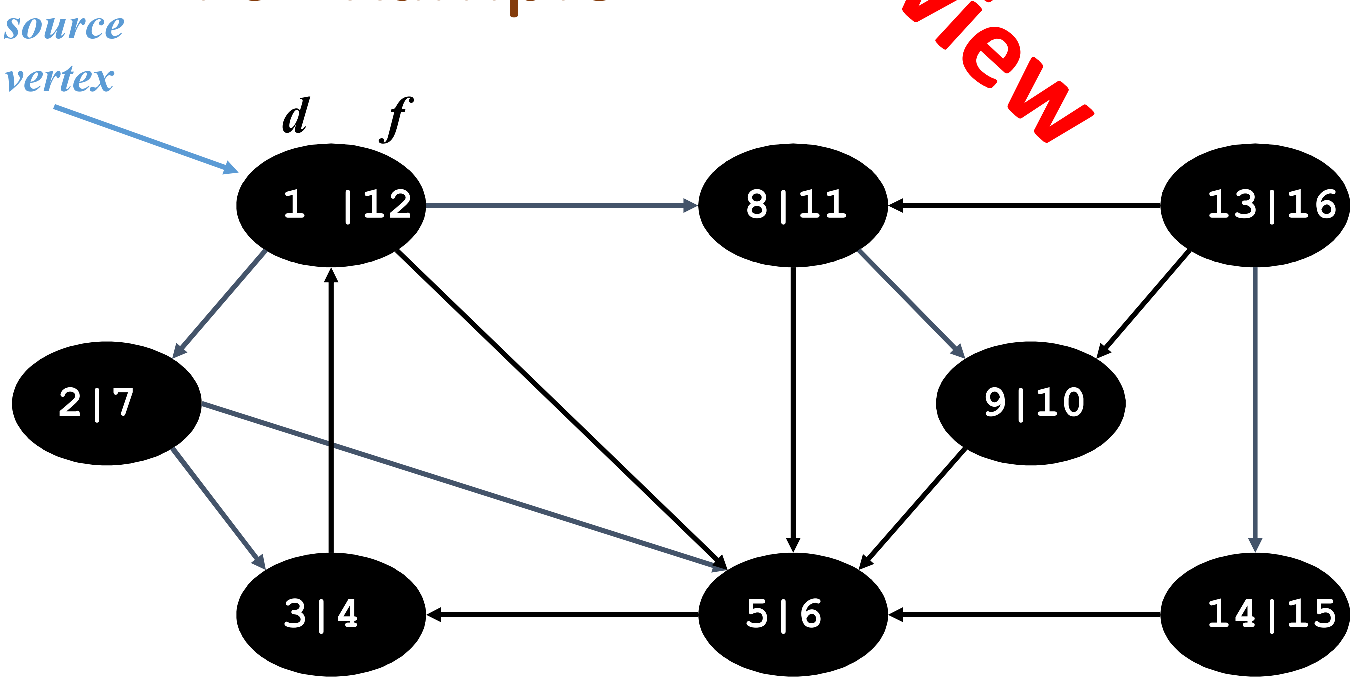
Review

Vertices		Adjacency list		
s		*t*	*v*	*w*
t		*u*	*v*	
u		*s*		
v		*u*		
w		*v*	*x*	
x		*v*		
y		*w*	*x*	*z*
z		*v*		



DFS Example

Review



For every vertex u, we have: $u.d < u.f$ ---(22.2)

Interesting Facts

- $u.d$ records when vertex u is discovered
- $u.f$ *records* when the processing of vertex u is finished.
- These timestamps are integers between 1 and $2 \times |V|$.
  - Since there is one discovery event and one finishing event for each of the $|V|$ vertices

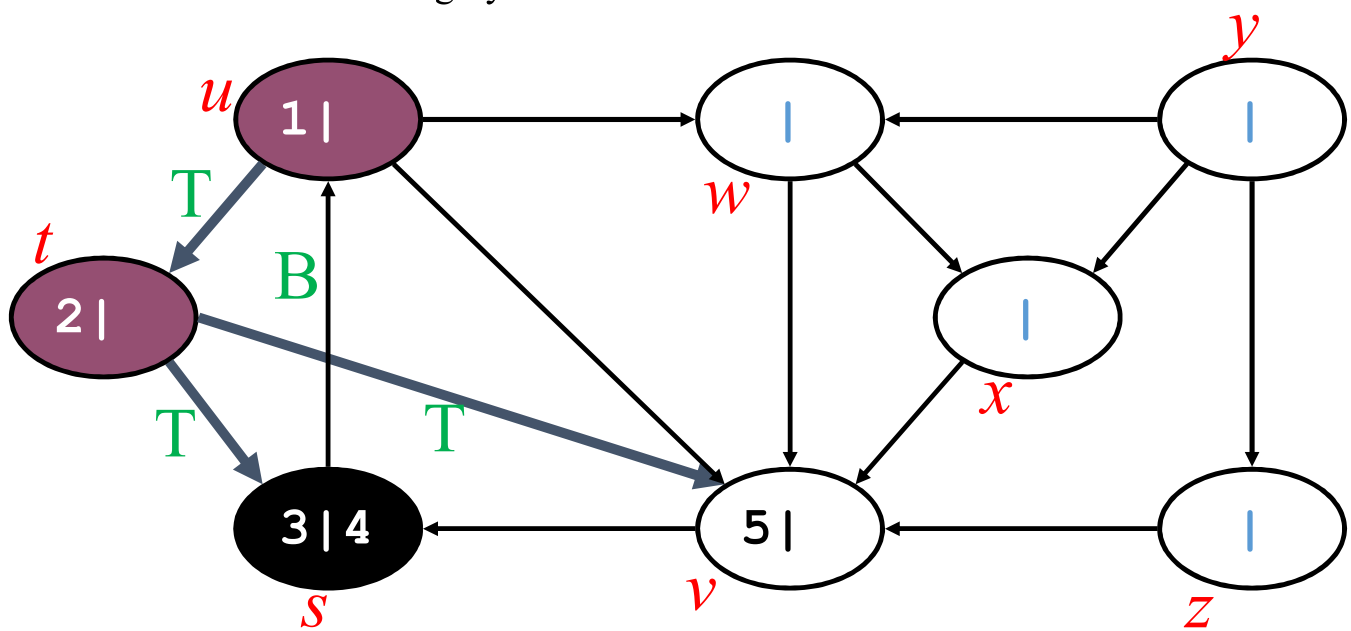
DFS: Properties

Review

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

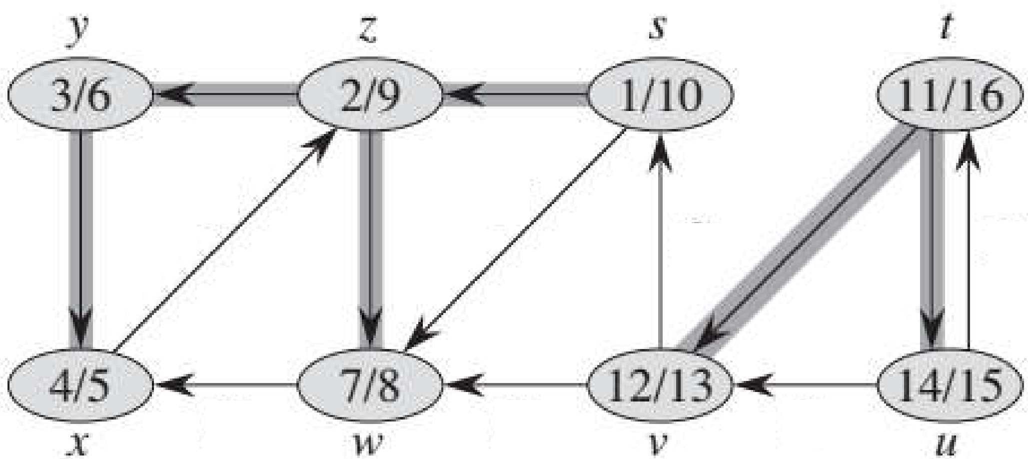
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

- $u = v.\pi$ if and only if DFS-VISIT (G, v) is called while searching u's adjacency list and v is white
- v is a descendent of u iff v is discovered WHITE while u is still grey

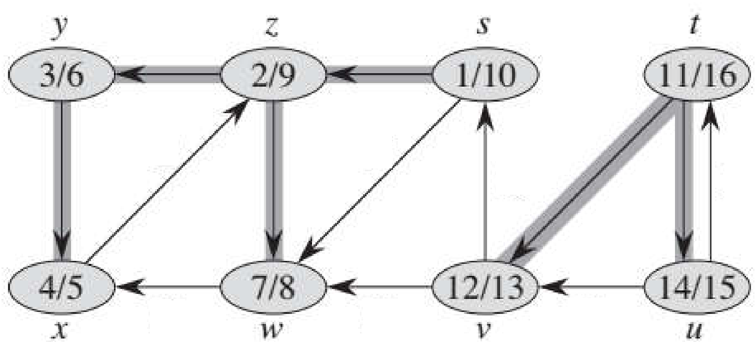


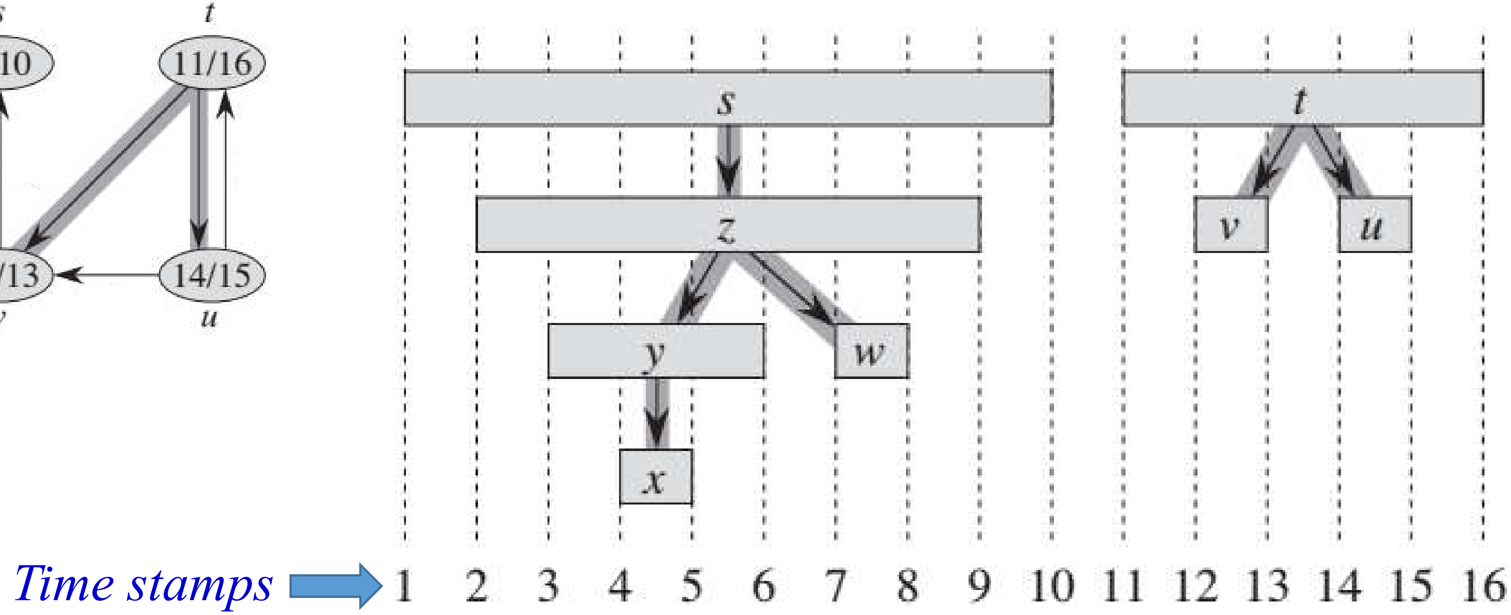
DFS: Properties

- *Parenthesis structure*

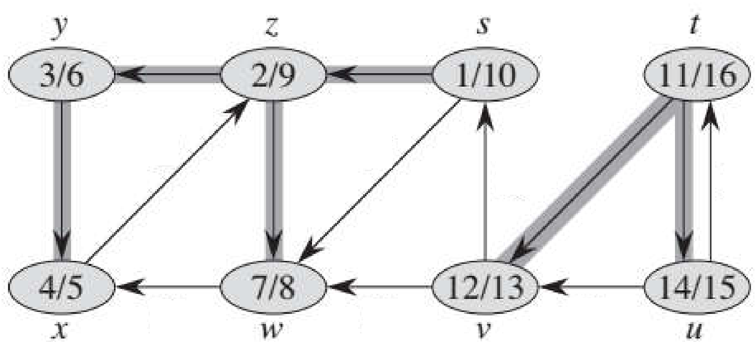


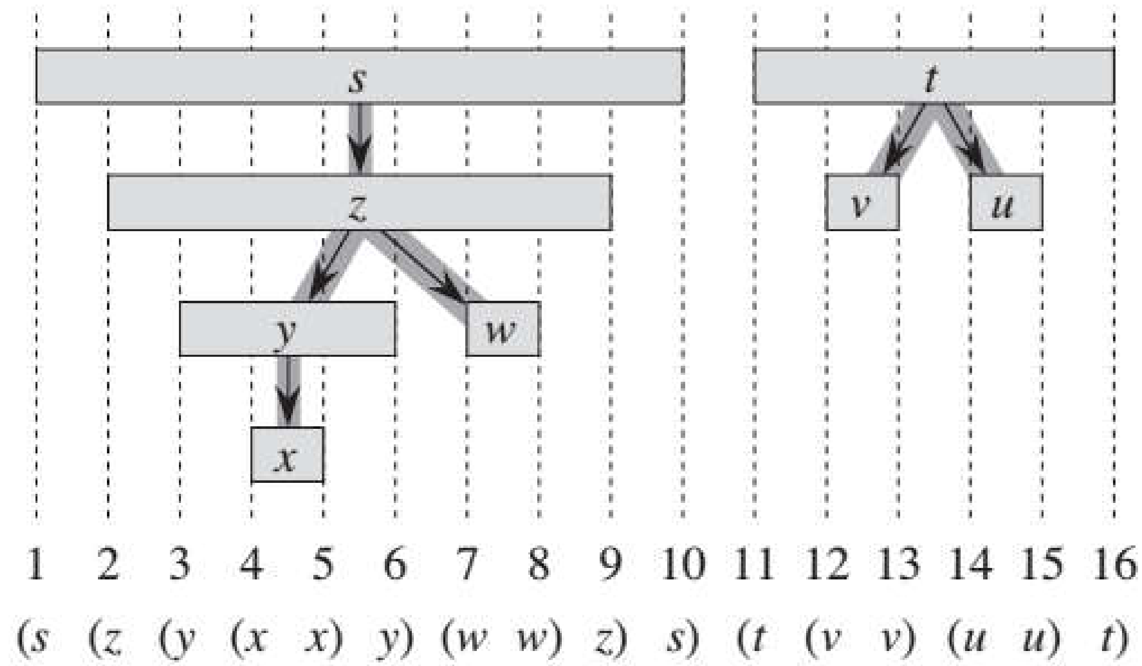
DFS: Parenthesis Structure



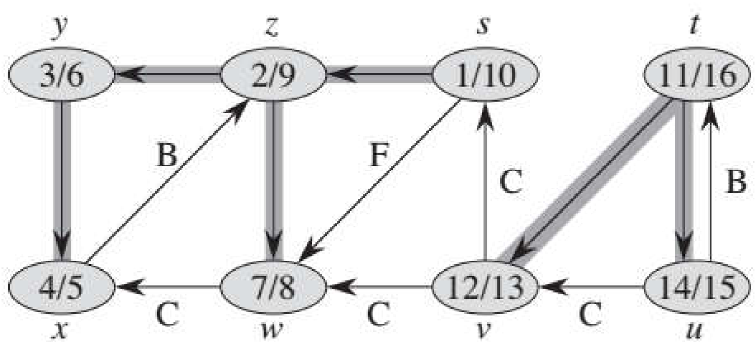


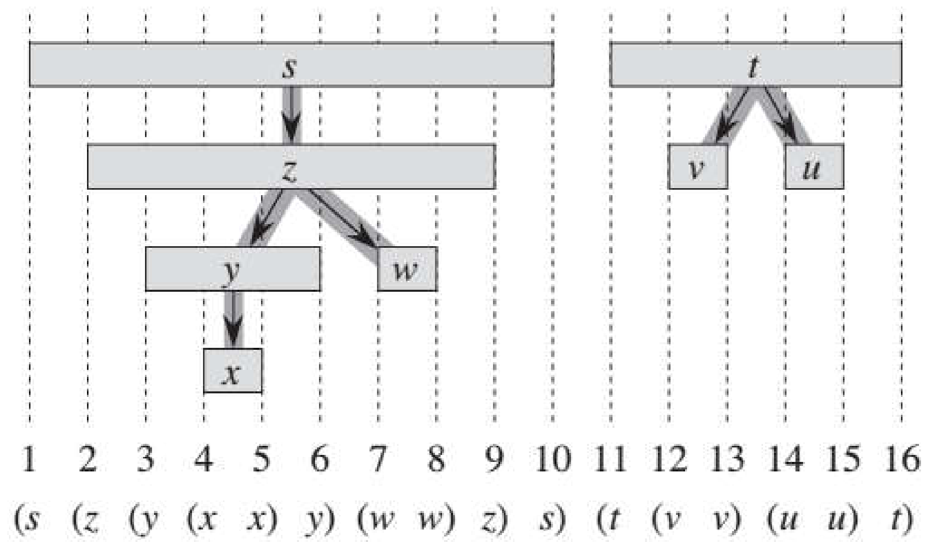
DFS: Parenthesis Structure





DFS: Parenthesis Structure





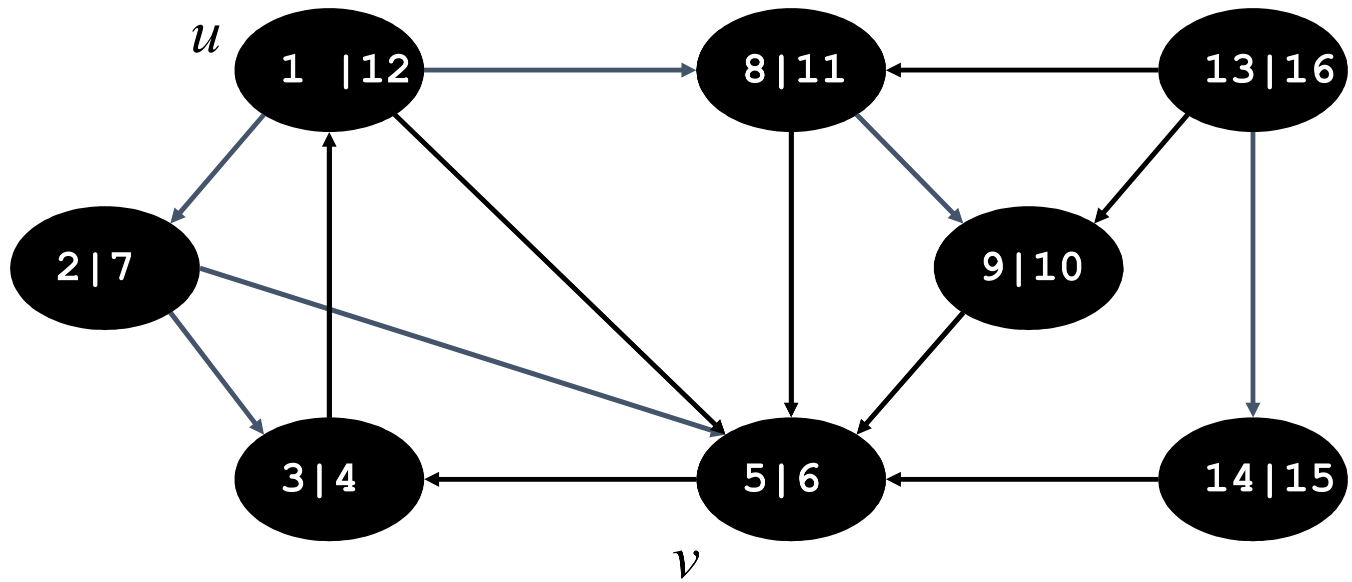
- ***parenthesis structure***
  - Represent discovery of vertex u with "(u"
  - represent finishing of vertex u with "u)"
  - Then the history of discoveries and finishings makes a well-formed expression in the sense that the parentheses are properly nested.

Theorem 22.7: Parenthesis Theorem

In any depth-first search of a (directed or undirected) graph G = (V, E), for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,
- the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a depth-first tree, or
- the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u in a depth-first tree

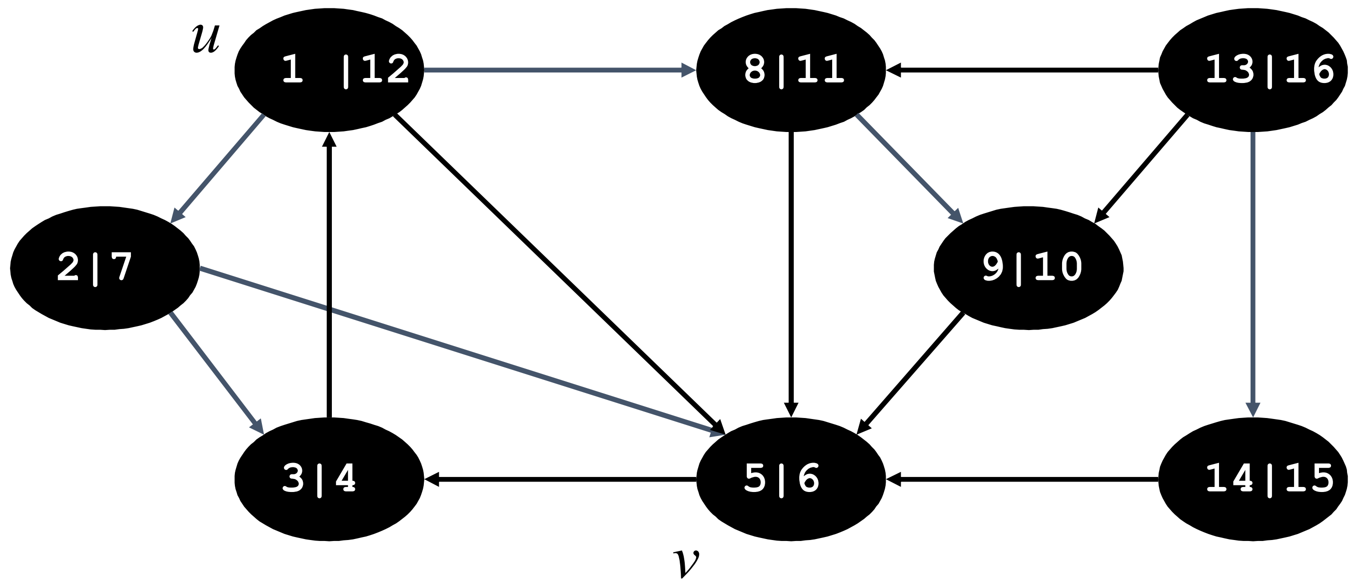
Case 1: $u.d < v.d$



Case 1: $u.d < v.d$

Sub-case 1A: $v.d < u.f$

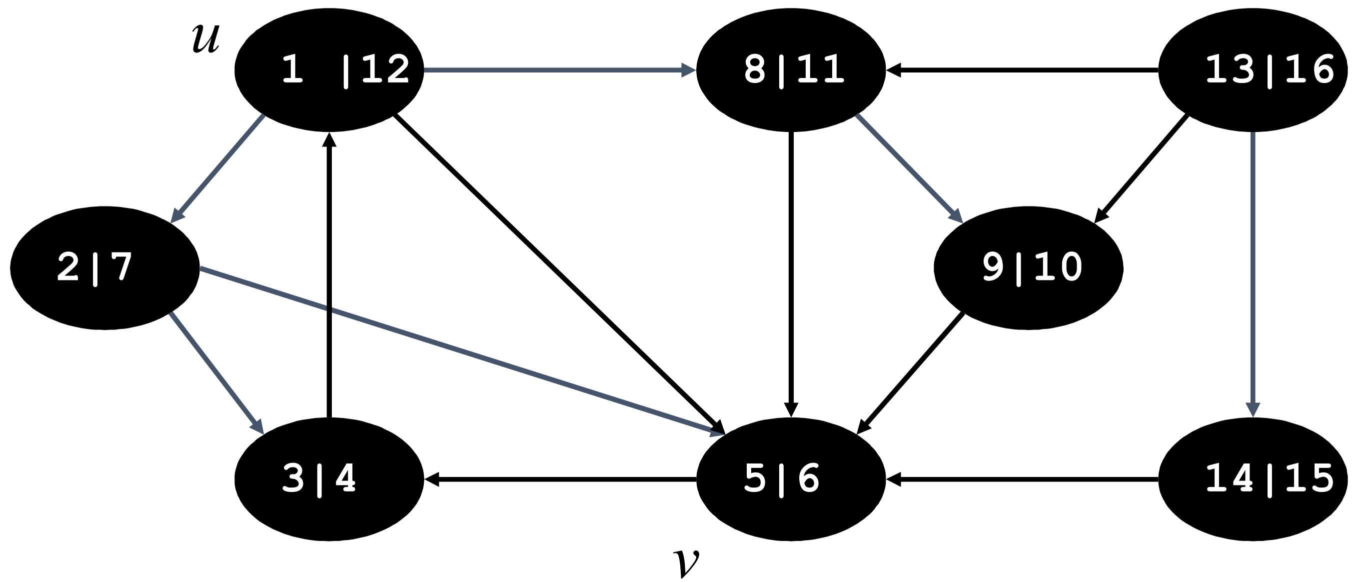
Sub-case 1B: $v.d > u.f$



Case 1: $u.d < v.d$

Sub-case 1A: $v.d < u.f$

$\Rightarrow u.d < v.d < u.f$



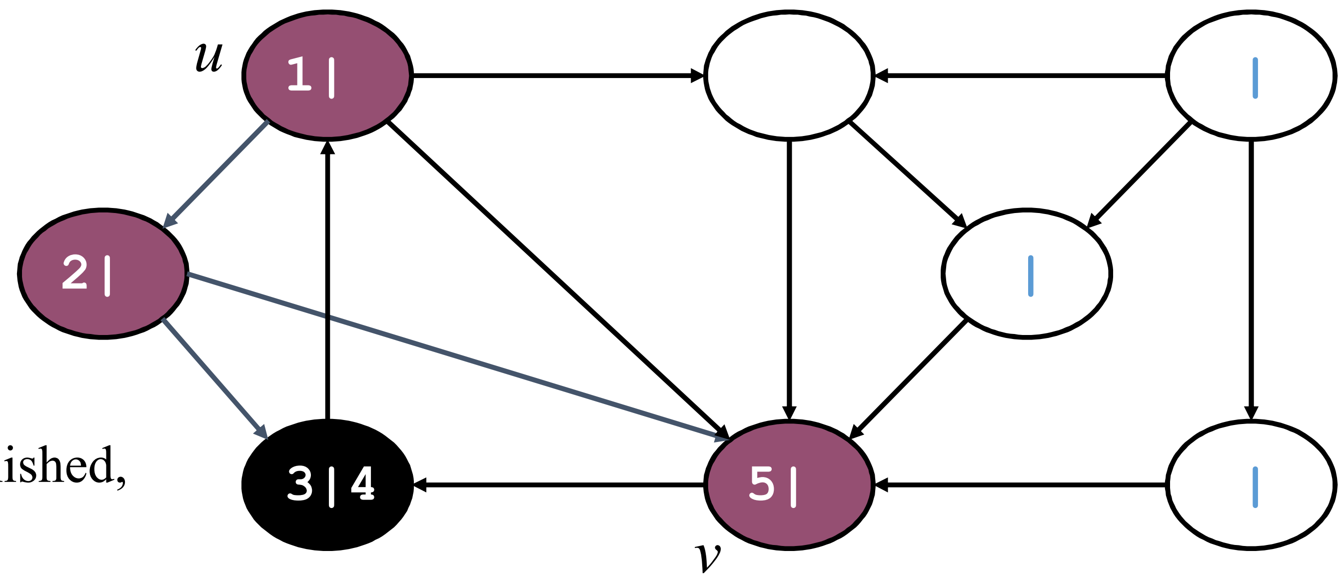
Case 1: $u.d < v.d$

Sub-case 1A: $v.d < u.f$

$\Rightarrow u.d < v.d < u.f$

$\Rightarrow$ v is discovered before u is finished,

$\Rightarrow$ i.e., u is gray.



Case 1: $u.d < v.d$

Sub-case 1A: $v.d < u.f$

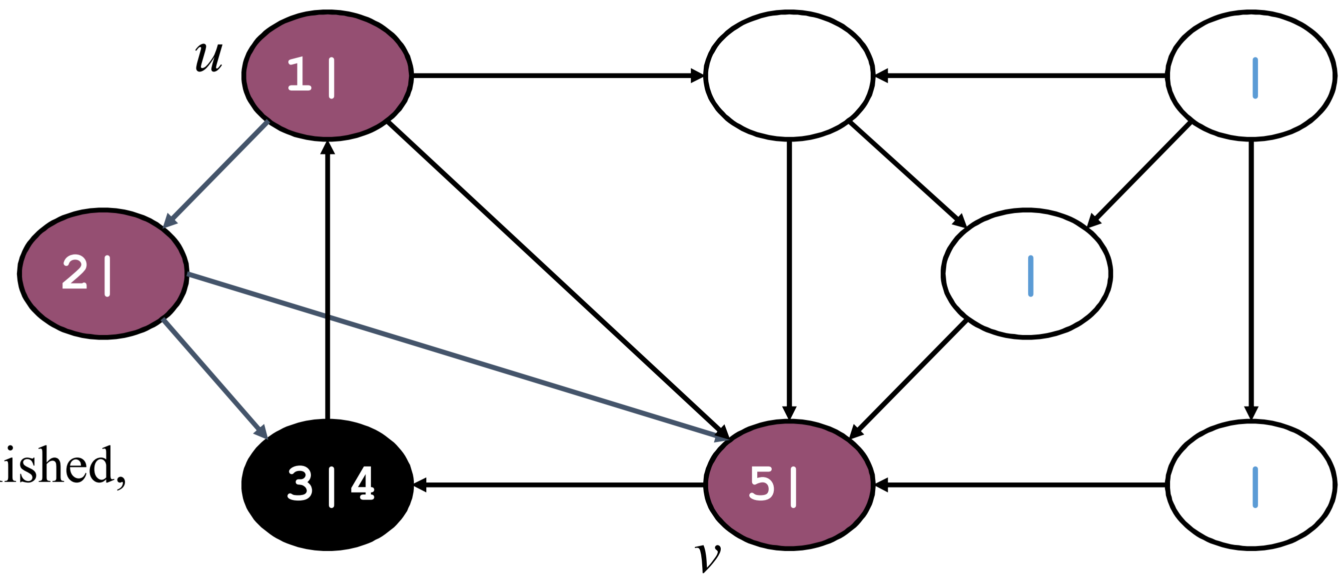
$\Rightarrow u.d < v.d < u.f$

$\Rightarrow$ v is discovered before u is finished,

$\Rightarrow$ i.e., u is gray.

v is a descendent of u.

v is discovered more recently than u



Case 1: $u.d < v.d$

Sub-case 1A: $v.d < u.f$

$\Rightarrow u.d < v.d < u.f$

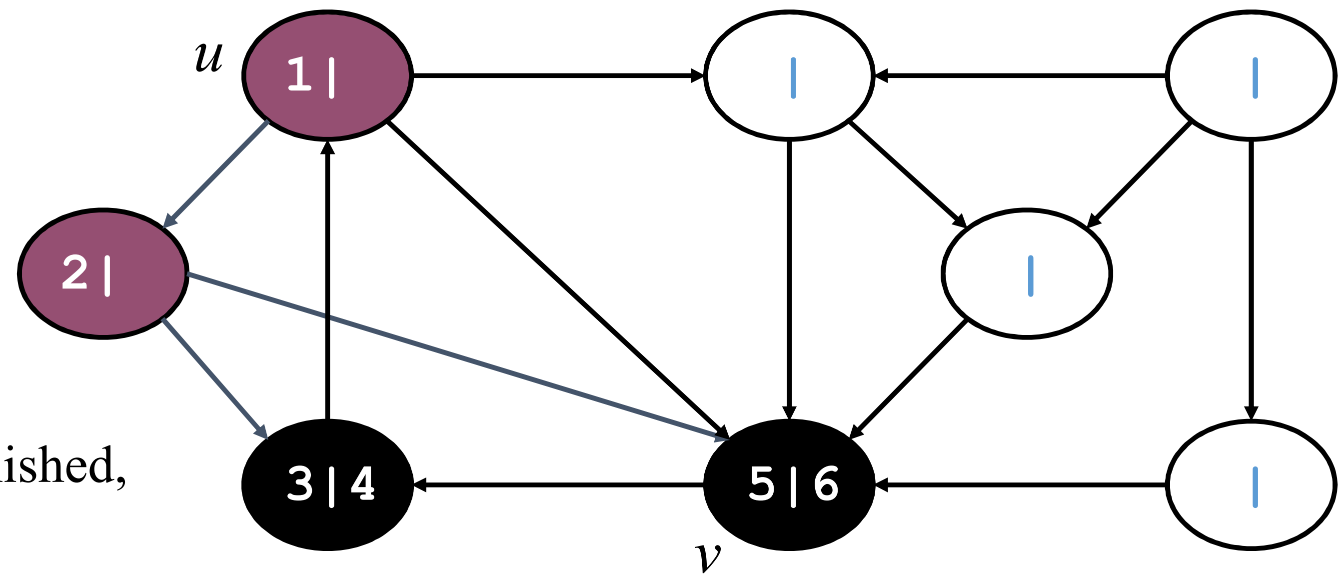
$\Rightarrow$ v is discovered before u is finished,

$\Rightarrow$ i.e., u is gray.

v is a descendent of u.

v is discovered more recently than u

=> v is finished before search returns to u.



Case 1: $u.d < v.d$

Sub-case 1A: $v.d < u.f$

$\Rightarrow u.d < v.d < u.f$

$\Rightarrow$ v is discovered before u is finished,

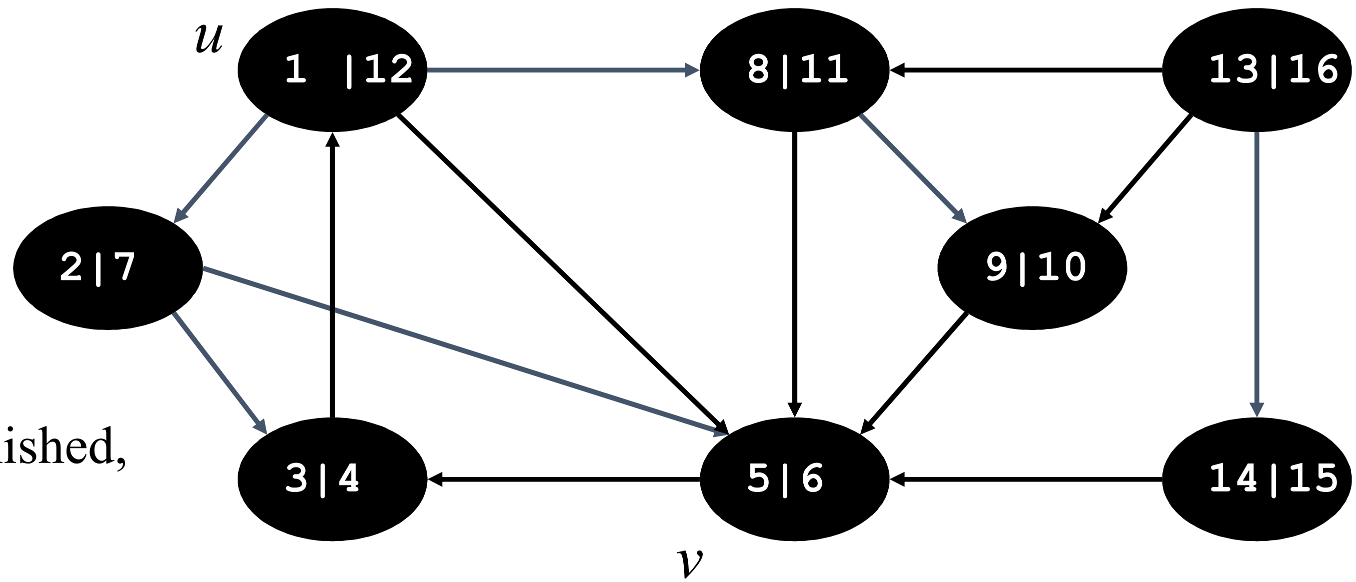
$\Rightarrow$ i.e., u is gray.

v is a descendent of u.

v is discovered more recently than u

=> v is finished before search returns to u.

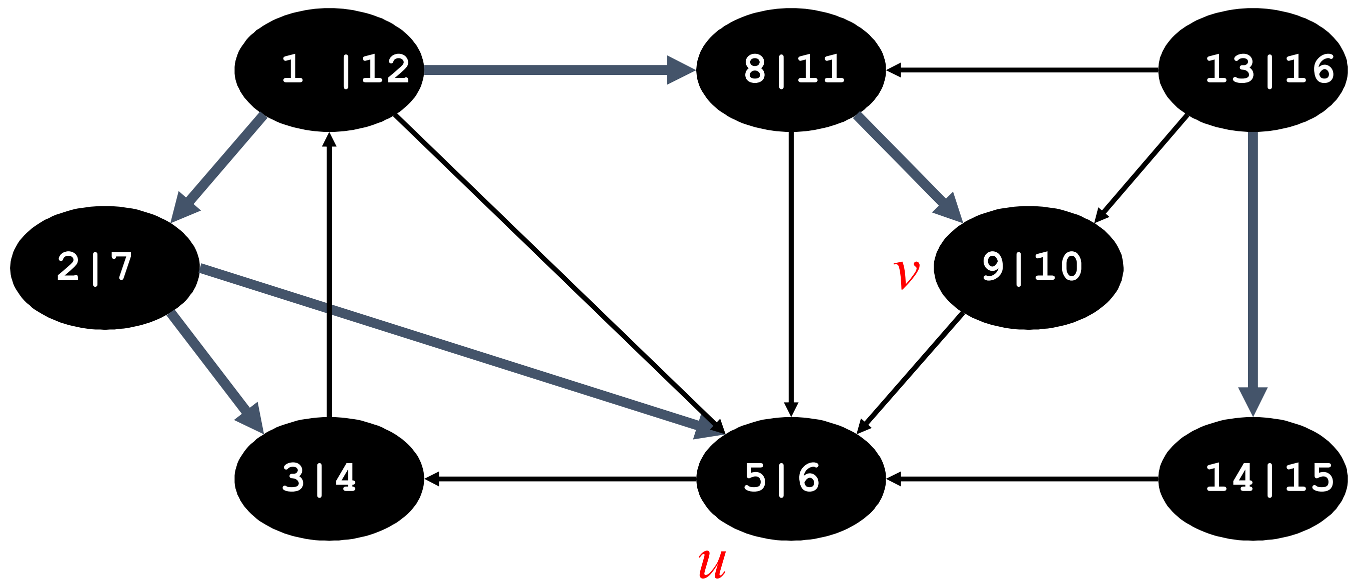
So, $[v.d, v.f]$ in $[u.d, u.f]$



Case 1: $u.d < v.d$

Sub-case 1B: $v.d > u.f$

$u.f < v.d \Rightarrow u.d < u.f < v.d$

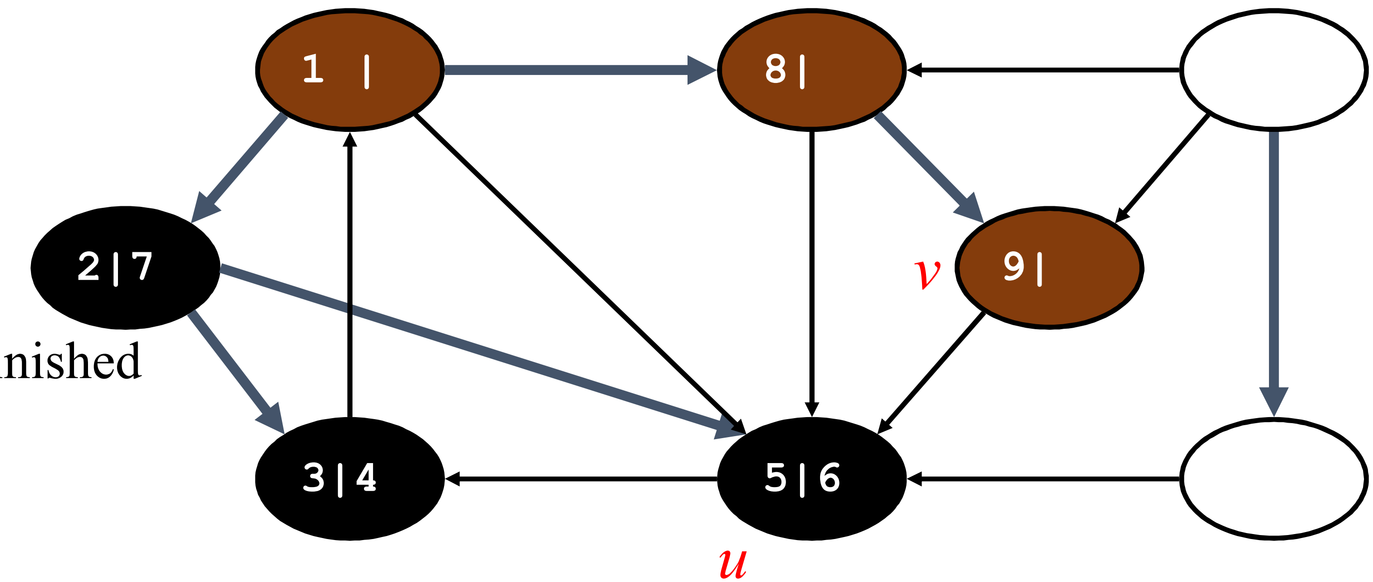


Case 1: $u.d < v.d$

Sub-case 1B: $v.d > u.f$

$u.f < v.d \Rightarrow u.d < u.f < v.d$

$\Rightarrow v$ is discovered AFTER u is finished
i.e., u is Black



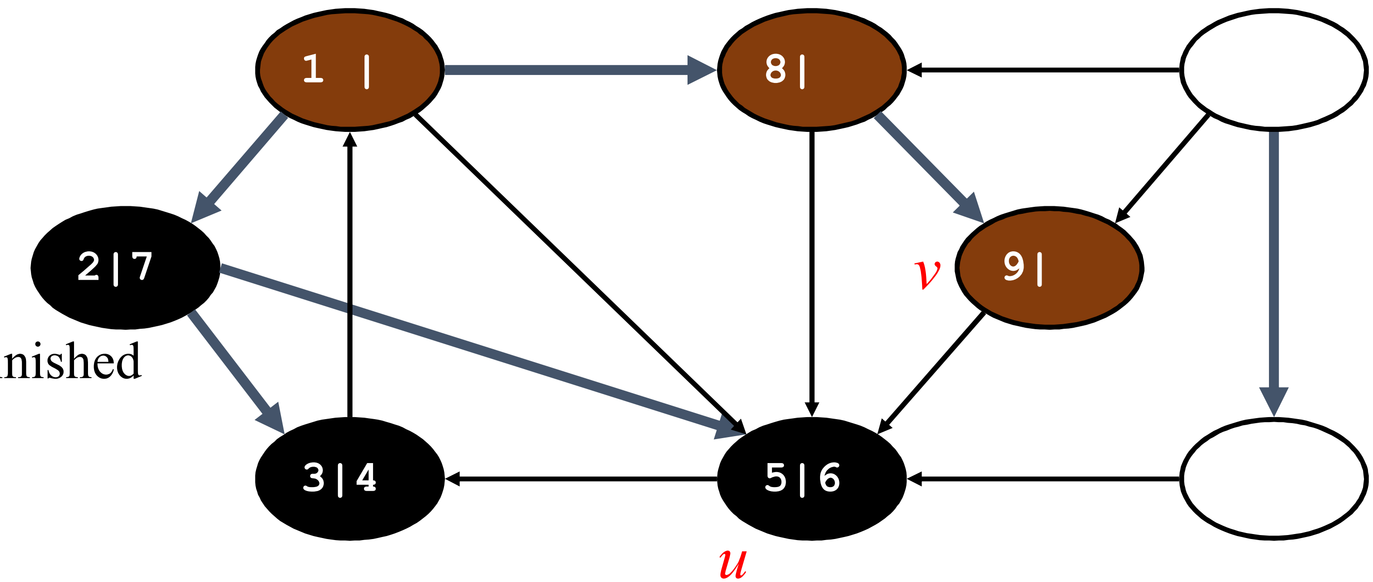
Case 1: $u.d < v.d$

Sub-case 1B: $v.d > u.f$

$u.f < v.d \Rightarrow u.d < u.f < v.d$

$\Rightarrow$ v is discovered AFTER u is finished
i.e., u is Black

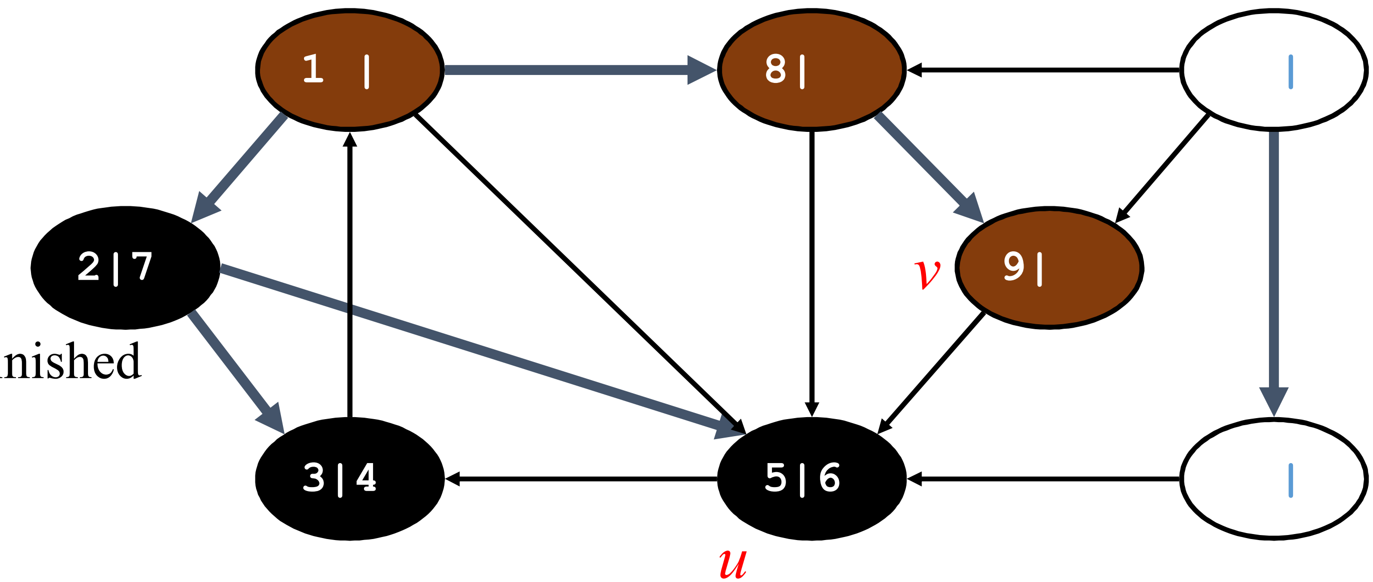
By Eq. (22.2) => $u.d < u.f < v.d < v.f$
=> [$u.d$, $u.f$] and [$v.d$, $v.f$] are disjoint.



Case 1: $u.d < v.d$

Sub-case 1B: $v.d > u.f$

$u.f < v.d \Rightarrow u.d < u.f < v.d$



$\Rightarrow$ v is discovered AFTER u is finished
i.e., u is Black

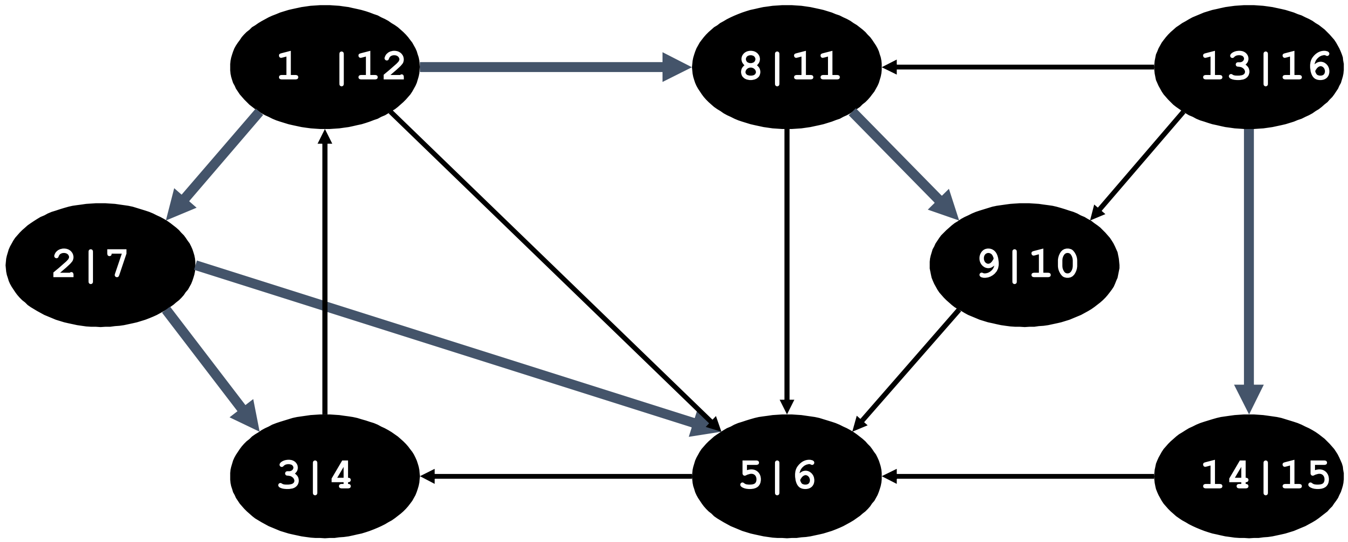
By Eq. (22.2) => $u.d < u.f < v.d < v.f$
$\Rightarrow$ $[u.d, u.f]$ and $[v.d, v.f]$ are disjoint.

$\Rightarrow$ neither vertex was discovered when the other was gray
$\Rightarrow$ neither is a descendent of the other.

Case 1: $u.d < v.d$

Case 2: $v.d < u.d$

Exactly similar argument, with the roles of u and v reversed



Corollary 22.8 (Nesting of descendants' intervals)
Vertex v is a proper descendant of vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $u.d < v.d < v.f < u.f$.

Corollary 22.8 (Nesting of descendants' intervals)
Vertex v is a proper descendant of vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $u.d < v.d < v.f < u.f$.

Can be proved from Theorem 22.7

Theorem 22.7 (Parenthesis theorem)
In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,
- the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a depth-first tree, or
- the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u in a depth-first tree.

P If, and only if Q

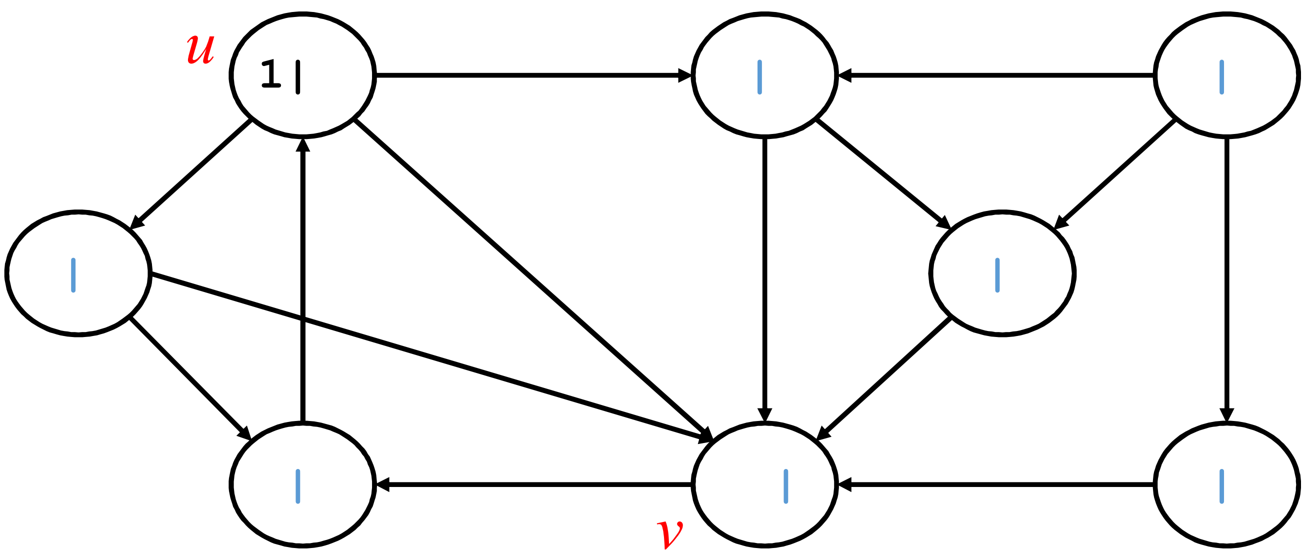
Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



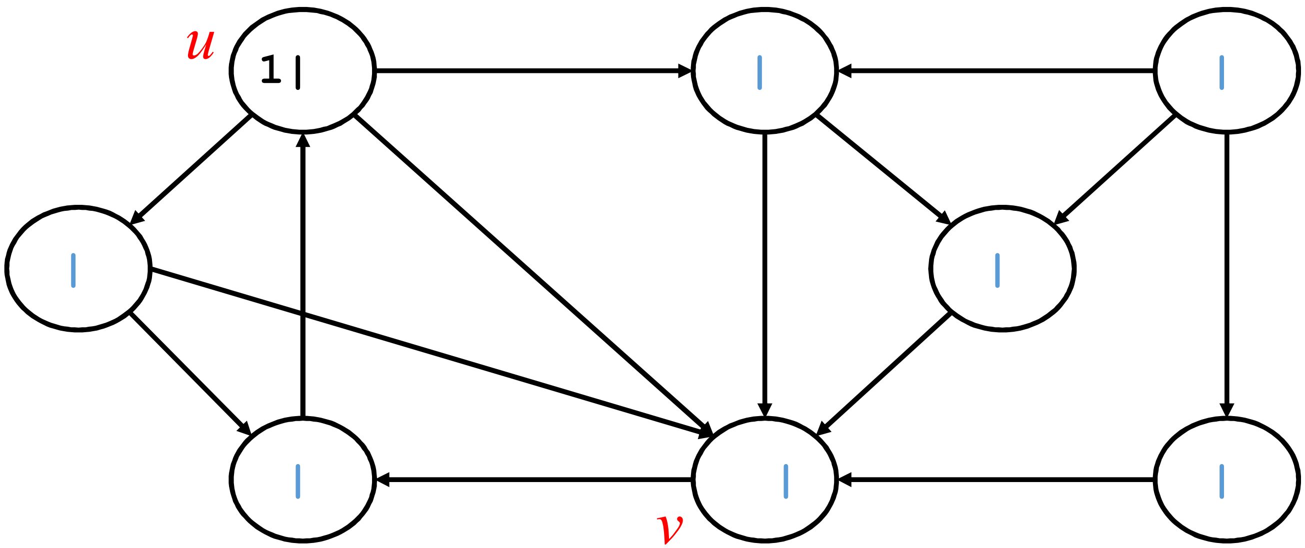
u is still white when u.d is set.

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

P If, and only if Q

Theorem 22.9 (White-path theorem)
In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



A. <u>If P then Q</u>

u is still white when $u.d$ is set.

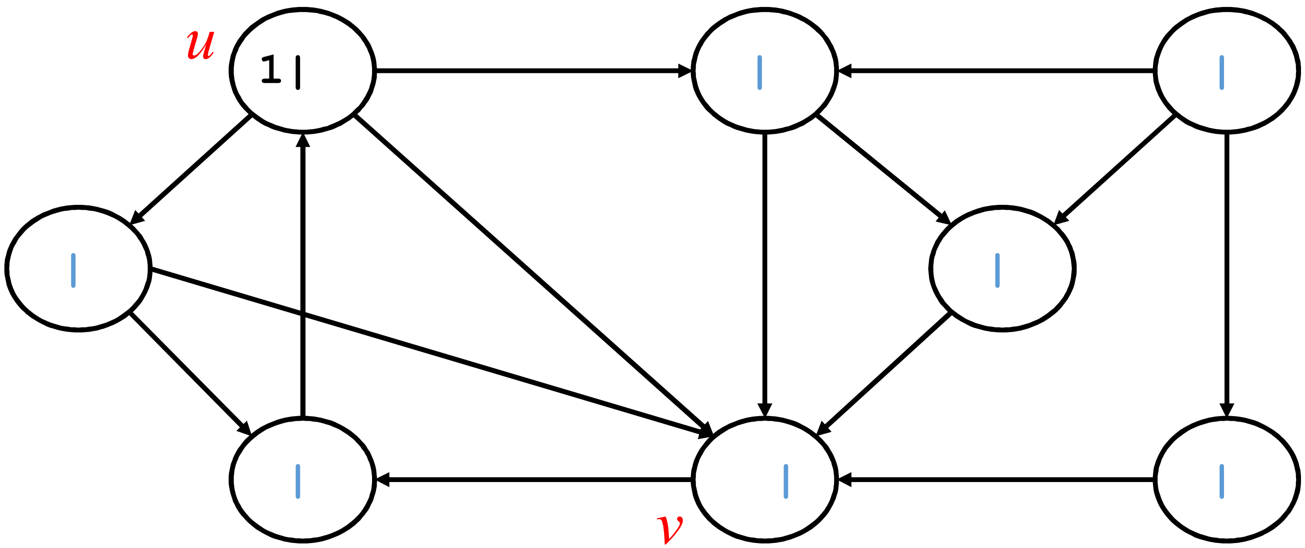
Let v is a descendant of u

If $v = u$, we are done as both are white

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



A. <u>If P then Q</u>

u is still white when $u.d$ is set.

If v is a proper descendant of u,

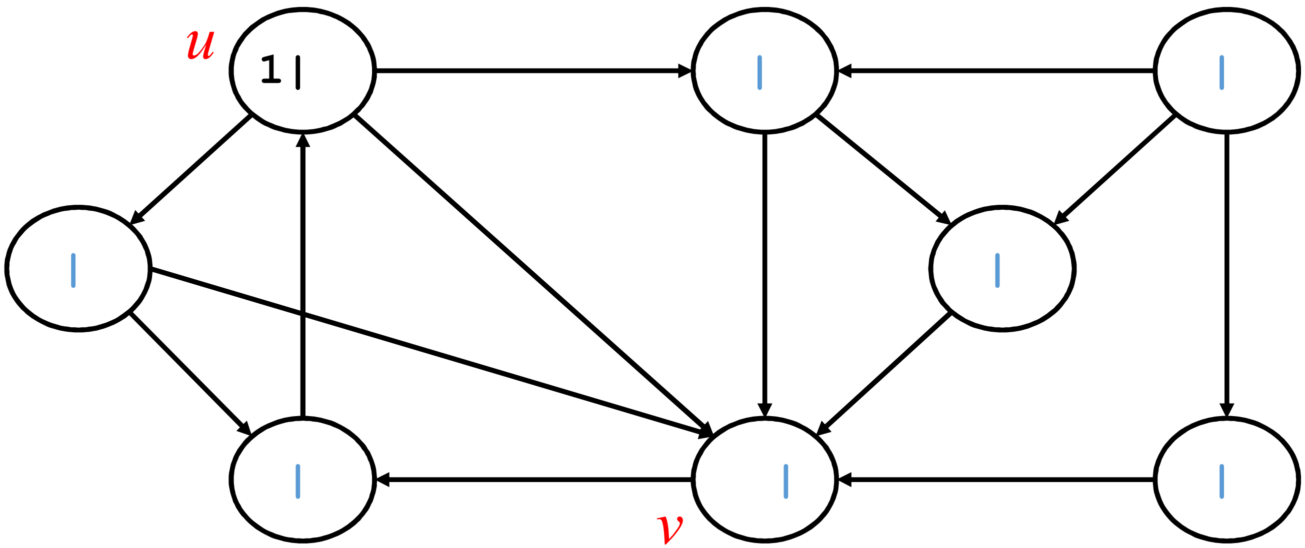
$u.d < v.d$ [by Corollary 22.8]

=> v must be WHITE at time $u.d$

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



A. <u>If P then Q</u>

u is still white when $u.d$ is set.

If v is a proper descendant of u,

$u.d < v.d$ [by Corollary 22.8]

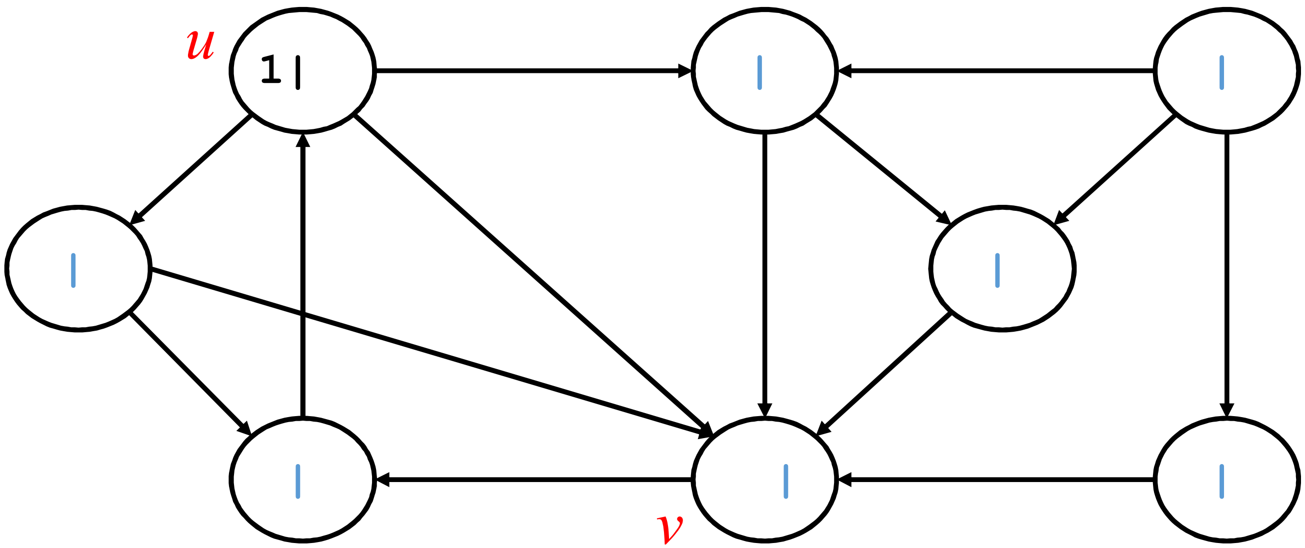
$\Rightarrow$ v must be WHITE at time $u.d$

$\Rightarrow$ Other vertices in the path to v must be WHITE too.

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

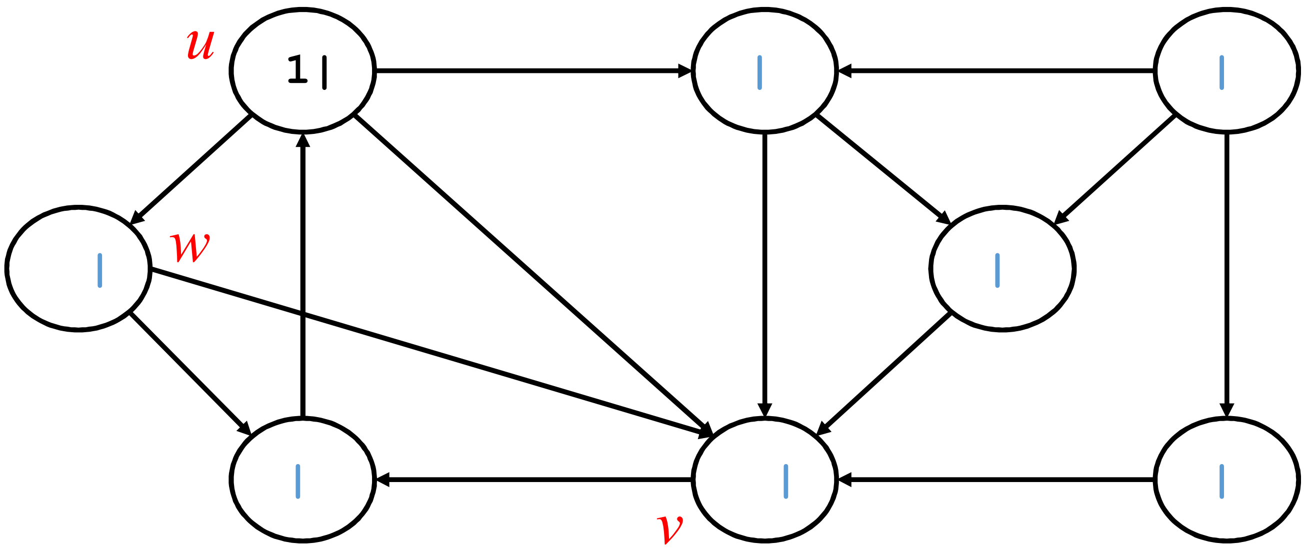


B. <u>If Q then P</u>

Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is not a descendant of u in DFT.

P If, and only if Q

Theorem 22.9 (White-path theorem)
In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

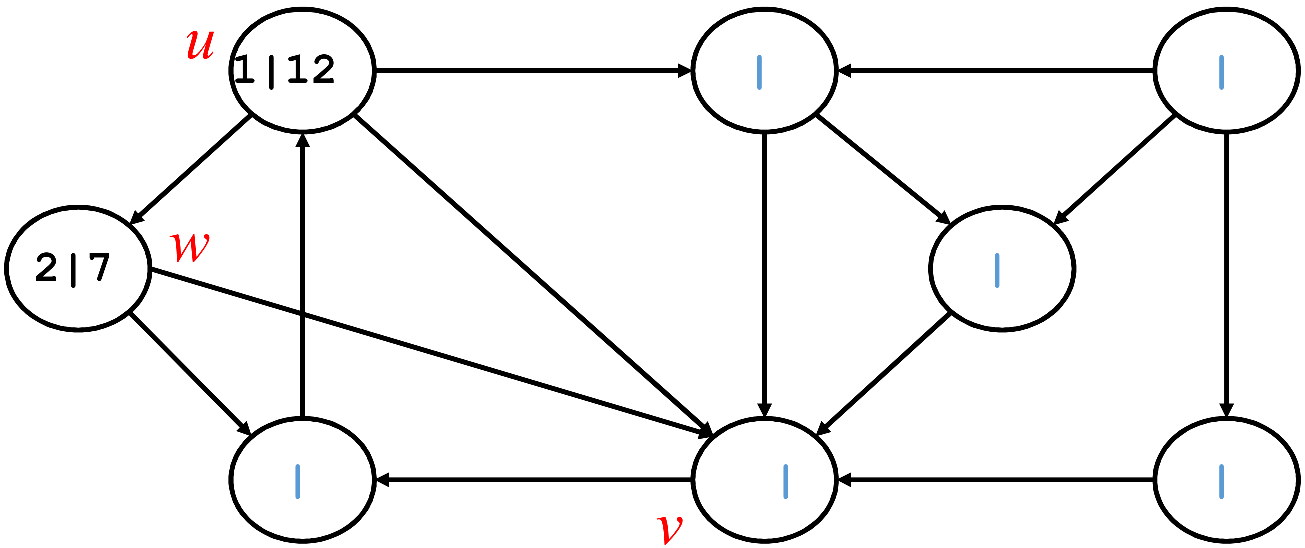


B. <u>If Q then P</u>
Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is not a descendant of u in DFT.
Let w be a predecessor of v (in Z)

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



B. <u>If Q then P</u>

Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is not a descendant of u in DFT.

Let w be a predecessor of v (in Z)

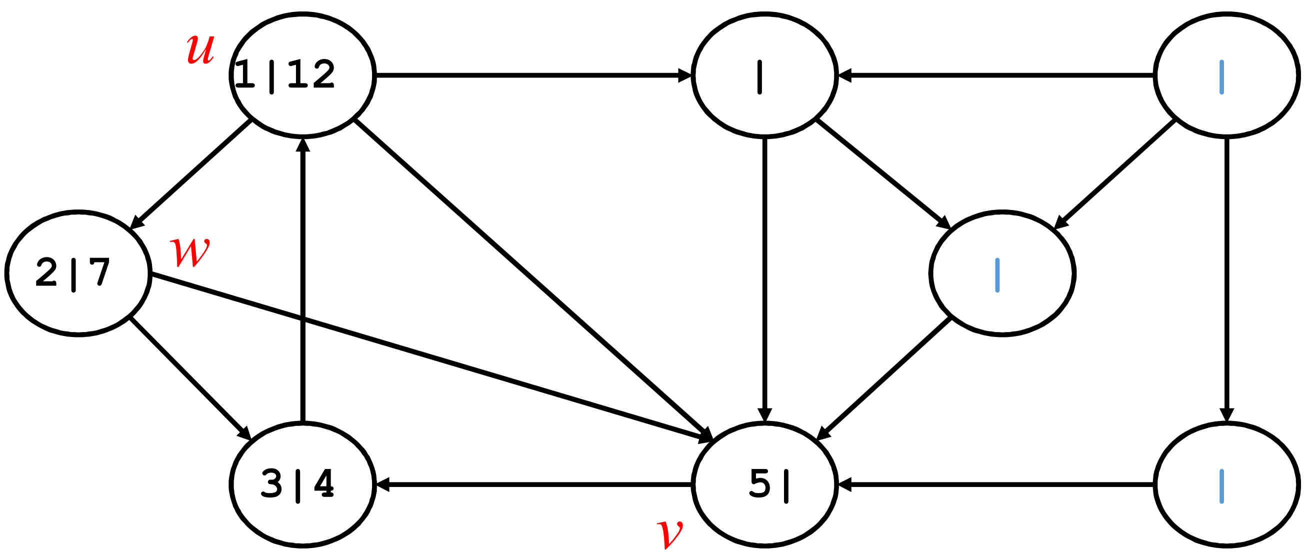
w is a descendant of u.

$\Rightarrow w.f <= u.f$

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



B. <u>If Q then P</u>

Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is not a descendant of u in DFT.

Let w be a predecessor of v (in Z) w is a descendant of u.

$\Rightarrow w.f <= u.f$

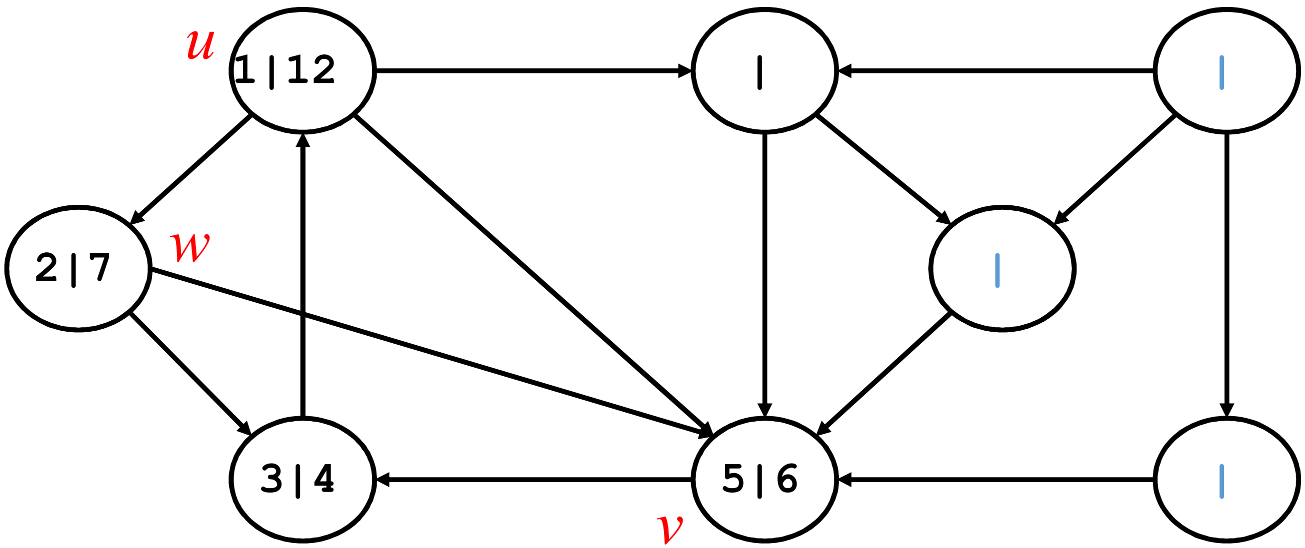
v must be discovered after u is discovered but before w is finished

$=> u.d < v.d < w.f$

P If, and only if Q

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



B. <u>If Q then P</u>

Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is not a descendant of u in DFT.

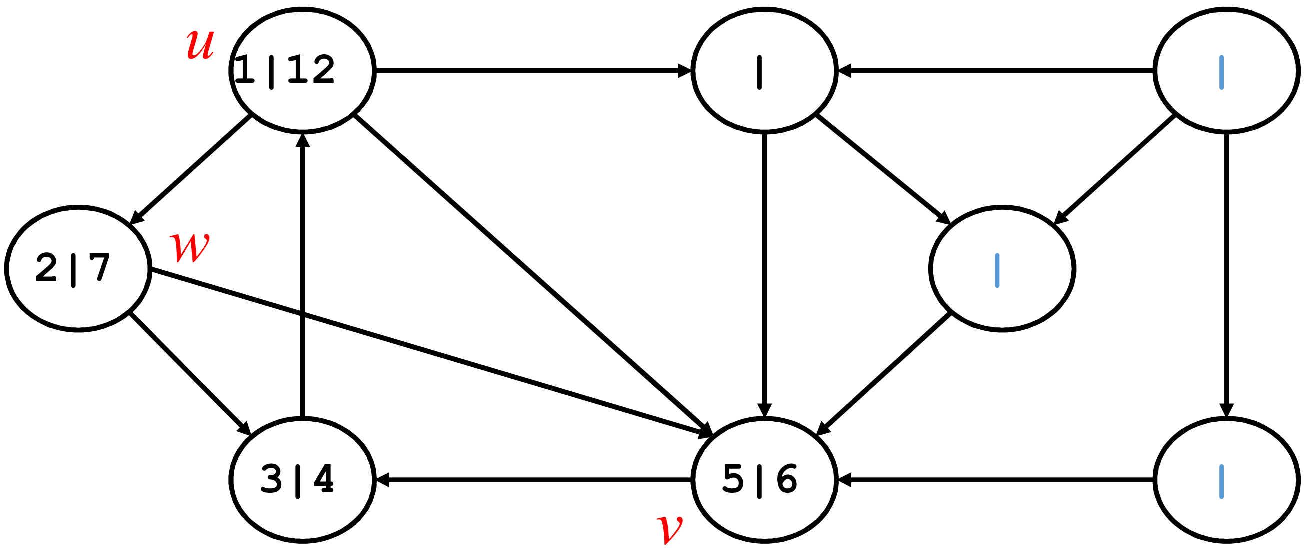
$\Rightarrow w.f <= u.f$

$\Rightarrow u.d < v.d < w.f$

$\Rightarrow u.d < v.d < w.f <= u.f$

P If, and only if Q

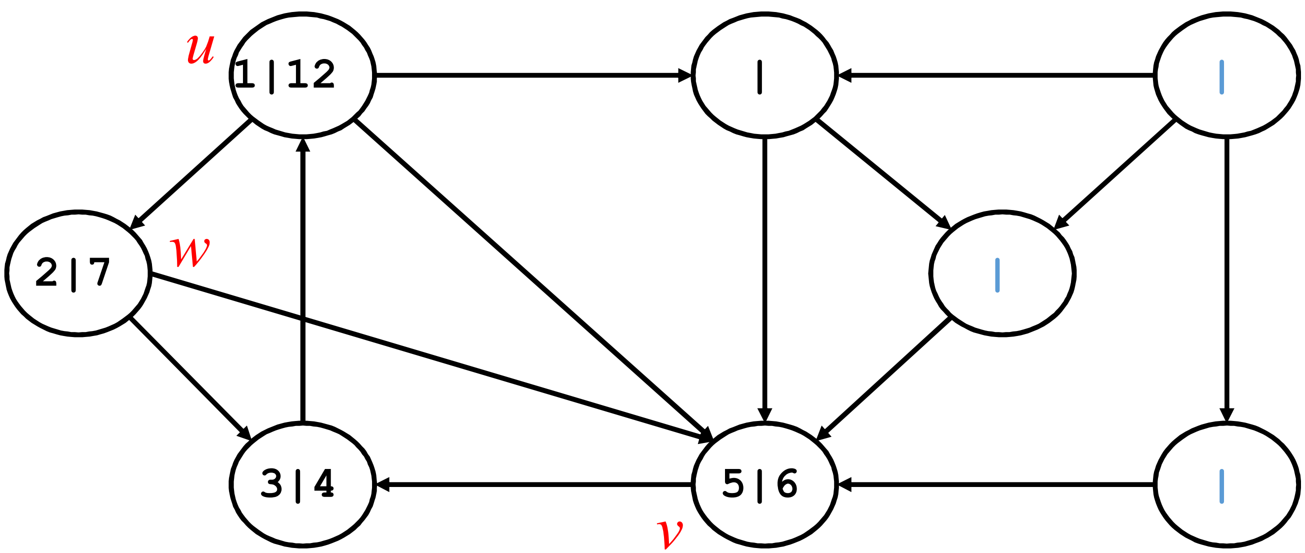
Theorem 22.9 (White-path theorem)
In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



B. <u>If Q then P</u>
Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is not a descendant of u in DFT.
$\Rightarrow w.f <= u.f$
$\Rightarrow u.d < v.d < w.f$
$\Rightarrow u.d < v.d < w.f <= u.f$
$\Rightarrow$ Th. 22.7, $[v.d,\ v.f]$ must be contained within $[u.d,\ u.f]$

P If, and only if Q

Theorem 22.9 (White-path theorem)
In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.



B. <u>If Q then P</u>

Let there is a path (Z) of white vertices from u to v at time $u.d$, but v is NOT a descendant of u in DFT.

$\Rightarrow w.f <= u.f$

$\Rightarrow u.d < v.d < w.f$

$\Rightarrow u.d < v.d < w.f <= u.f$

$\Rightarrow$ Th. 22.7, $[v.d,\ v.f]$ must be contained within $[u.d,\ u.f]$

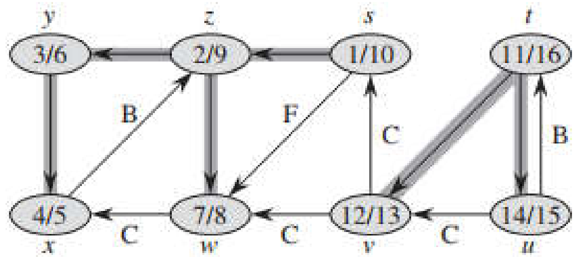
$\Rightarrow$ v MUST BE a descendant of u in DFT

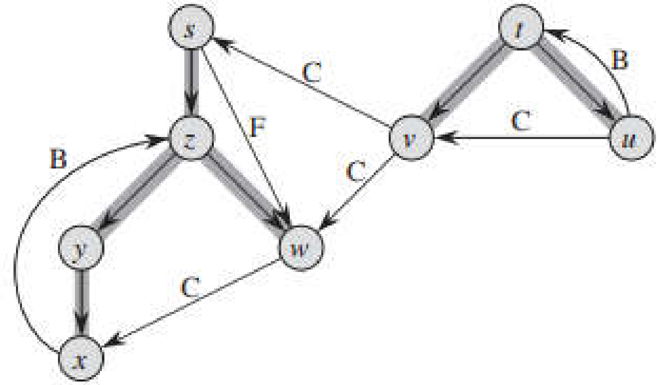
Depth-first forest

- The procedure DFS builds a depth-first forest comprising several depth-first trees as it searches the graph
- The forest/trees corresponds to the π attributes.
- More formally, for a graph $G = (V, E)$, we define the ***predecessor subgraph*** of G as $G_\pi = (V, E_\pi)$, where $E_\pi = \{(v.\pi, v) : v \in V \text{ and } v.\pi \neq \text{NIL}\}$
- The edges in E_π are called tree edges.

DFS: Types of Edges

- Tree Edges
- Forward Edges
- Back Edges
- Cross Edges





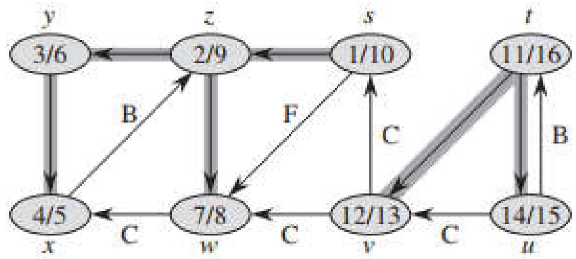
DFS: Types of Edges

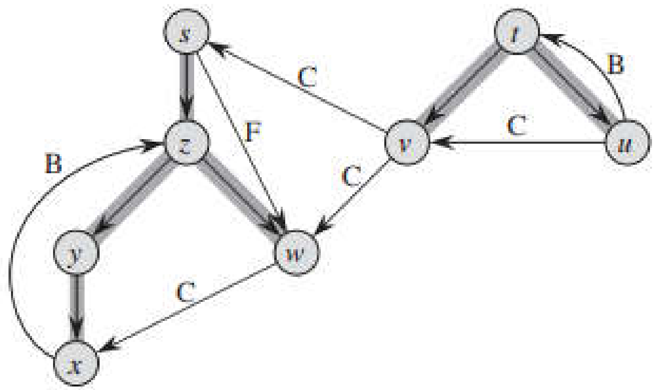
When we first explore (u, v), u is gray.
The color of v determines the edged type.

- Tree Edges are edges in the depth-first forest G_π. Edge (u, v) is a tree edge if it is first discovered by exploring edge (u, v)

encounters a WHITE vertex v

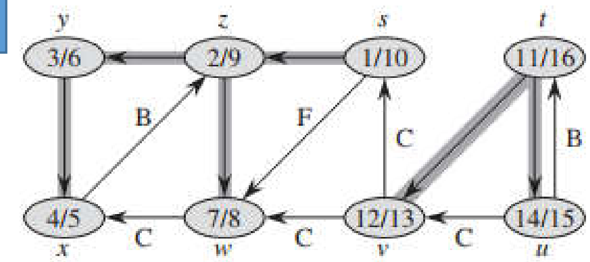
- Forward Edges
- Back Edges
- Cross Edges





When we first explore (u, v), u is gray.
The color of v determines the edged type.

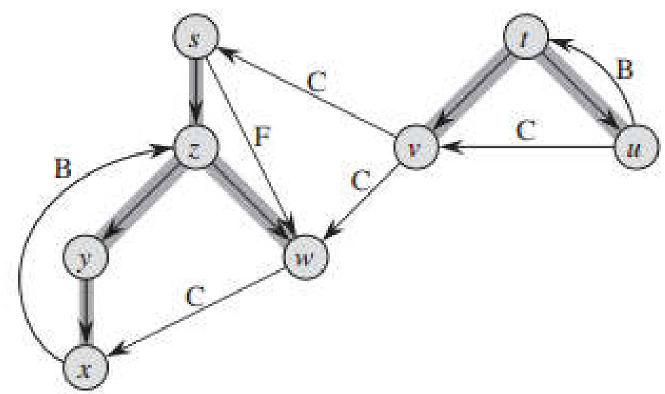
DFS: Types of Edges



- Tree Edges
- Forward Edges are those edges (u, v) connecting a vertex u to an descendant v in a depth-first tree.

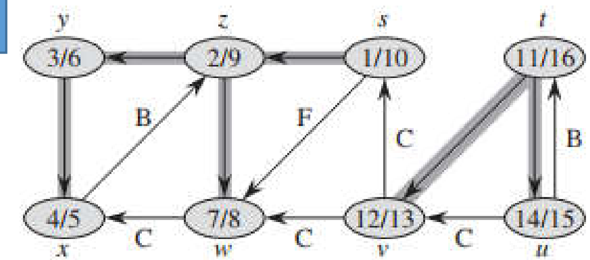
Encounters a BLACK vertex, v

- Back Edges
- Cross Edges



When we first explore (u, v), u is gray.
The color of v determines the edged type.

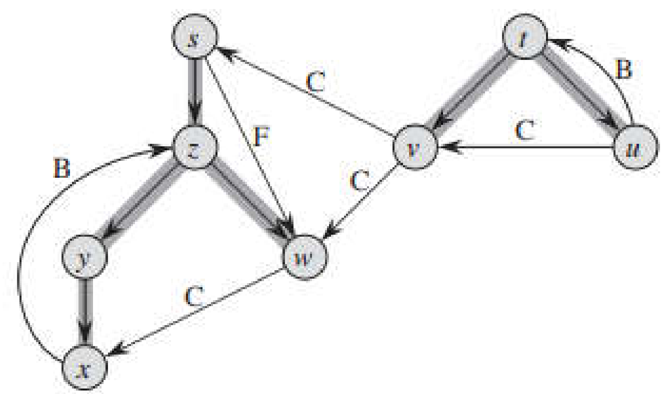
DFS: Types of Edges



- Tree Edges
- Forward Edges

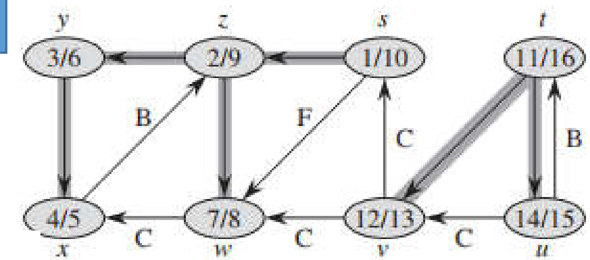
encounters a GRAY vertex, v

- Back Edges are those non-tree edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges.
- Cross Edges

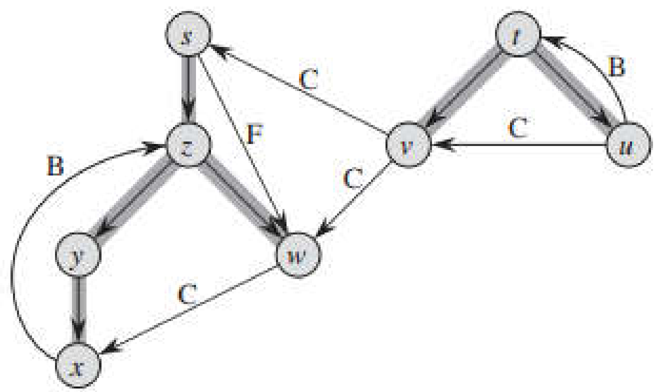


When we first explore (u, v), u is gray.
The color of v determines the edged type.

DFS: Types of Edges



- Tree Edges

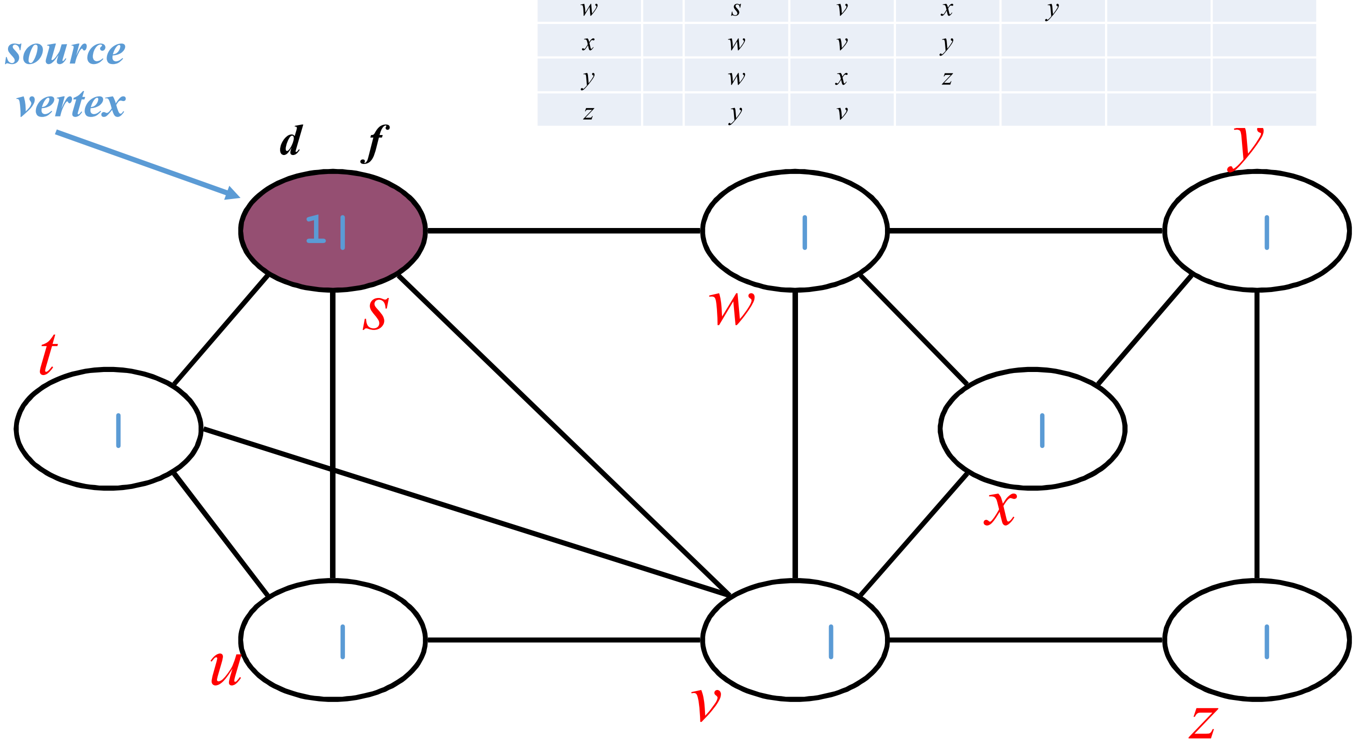


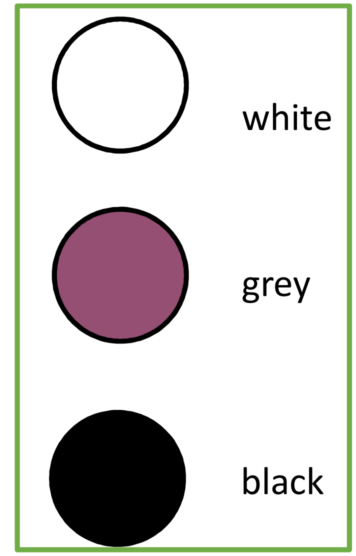
- Forward Edges
- Back Edges
- Cross Edges are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

encounters a BLACK vertex, v

Undirected graph and edges

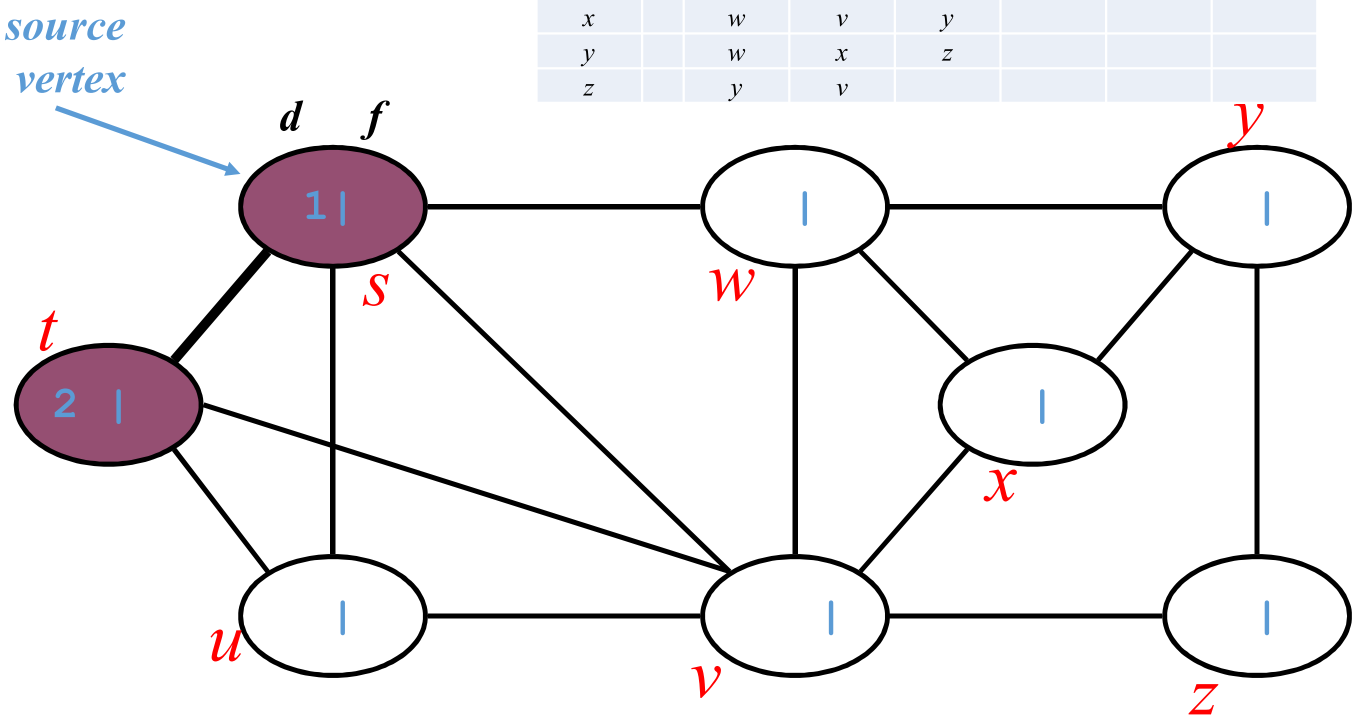
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

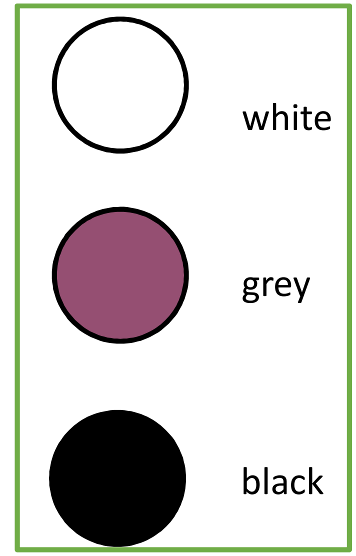




Undirected graph and edges

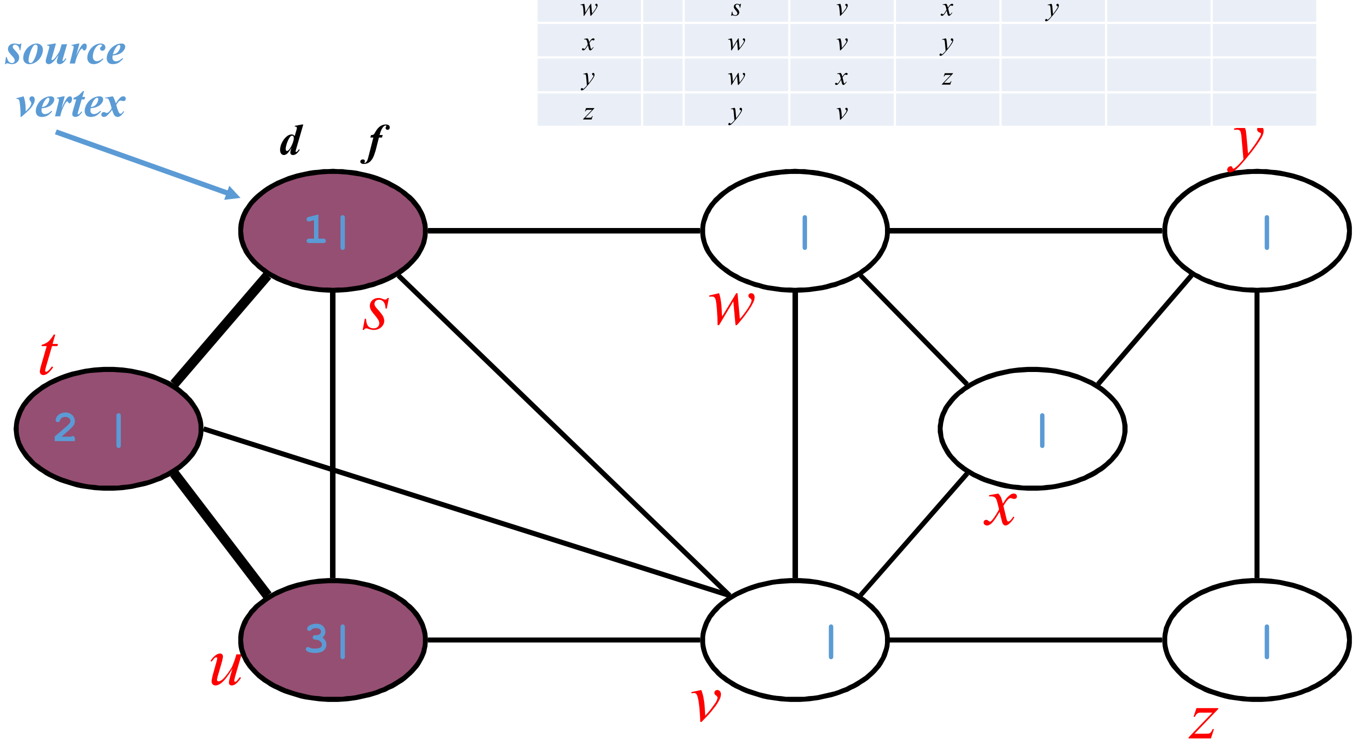
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

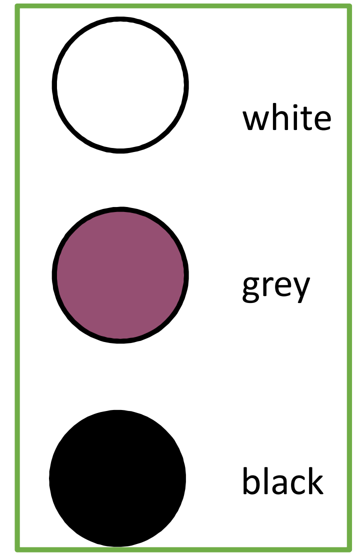




Undirected graph and edges

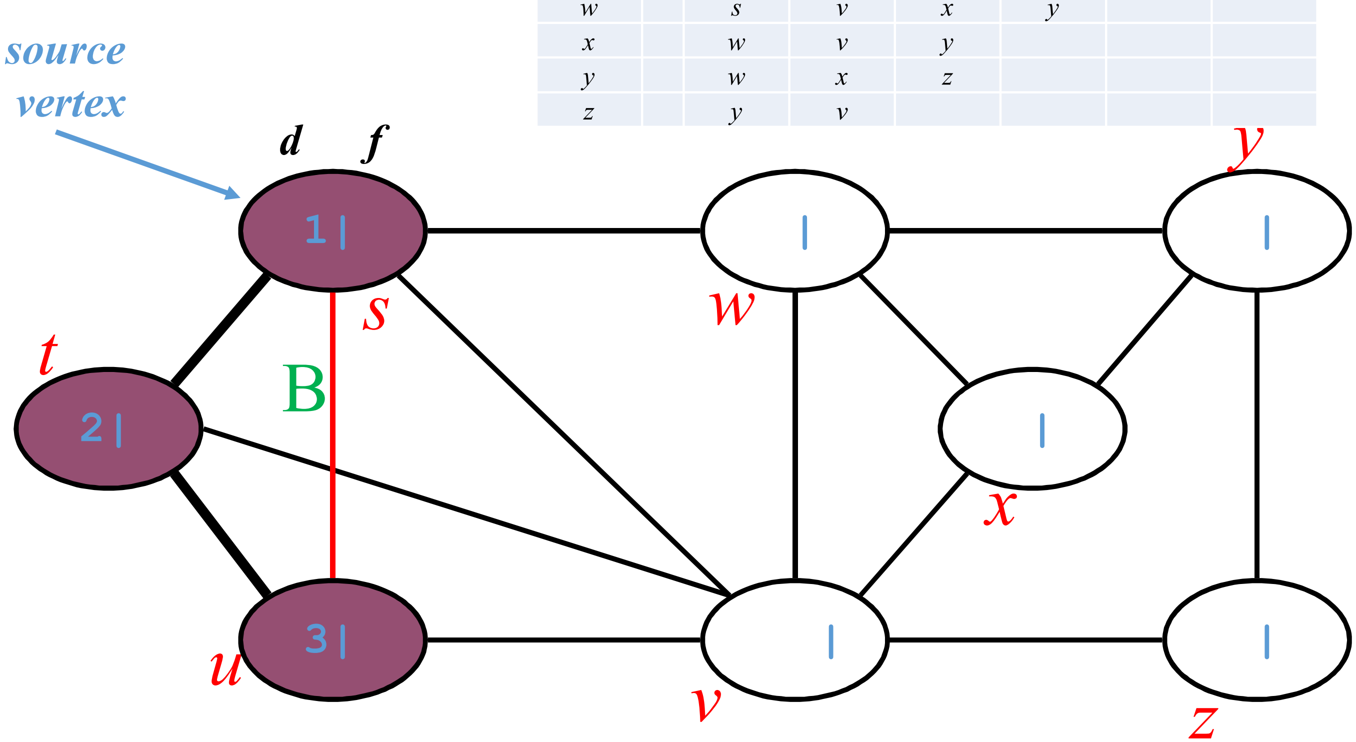
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

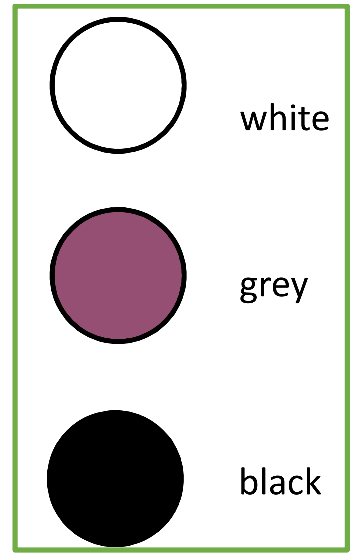




Undirected graph and edges

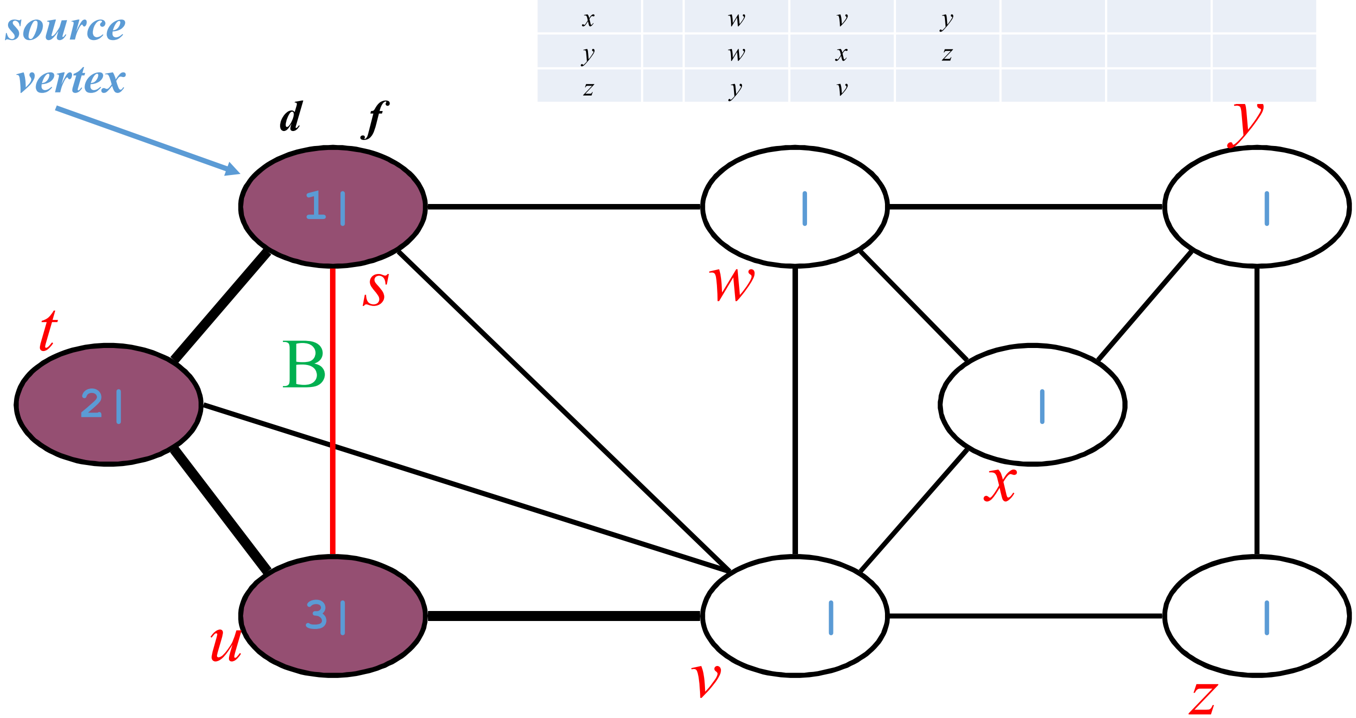
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

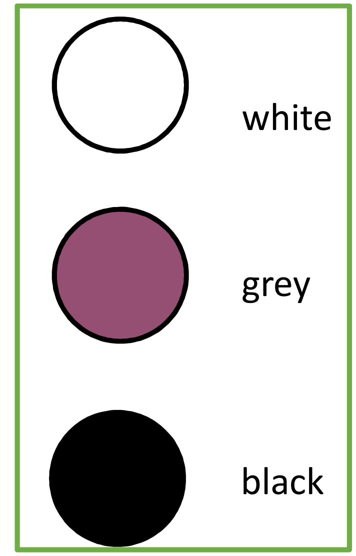




Undirected graph and edges

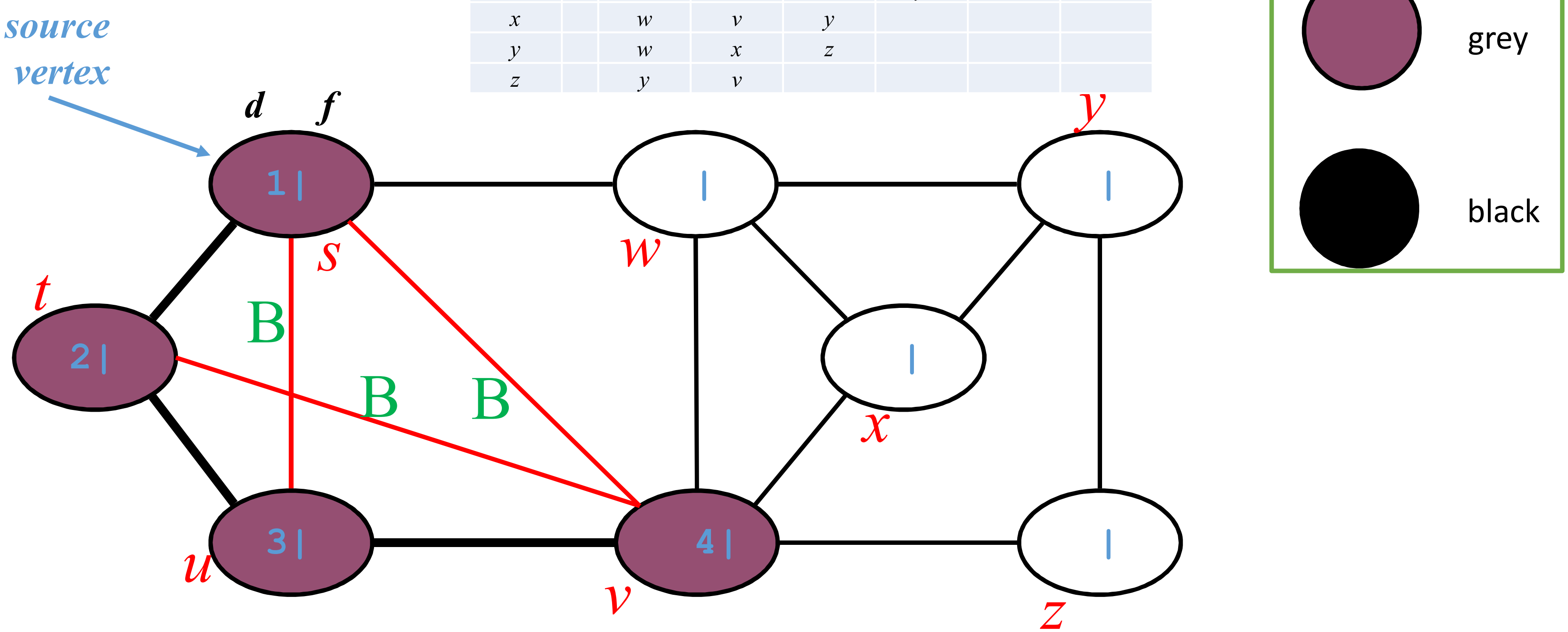
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				





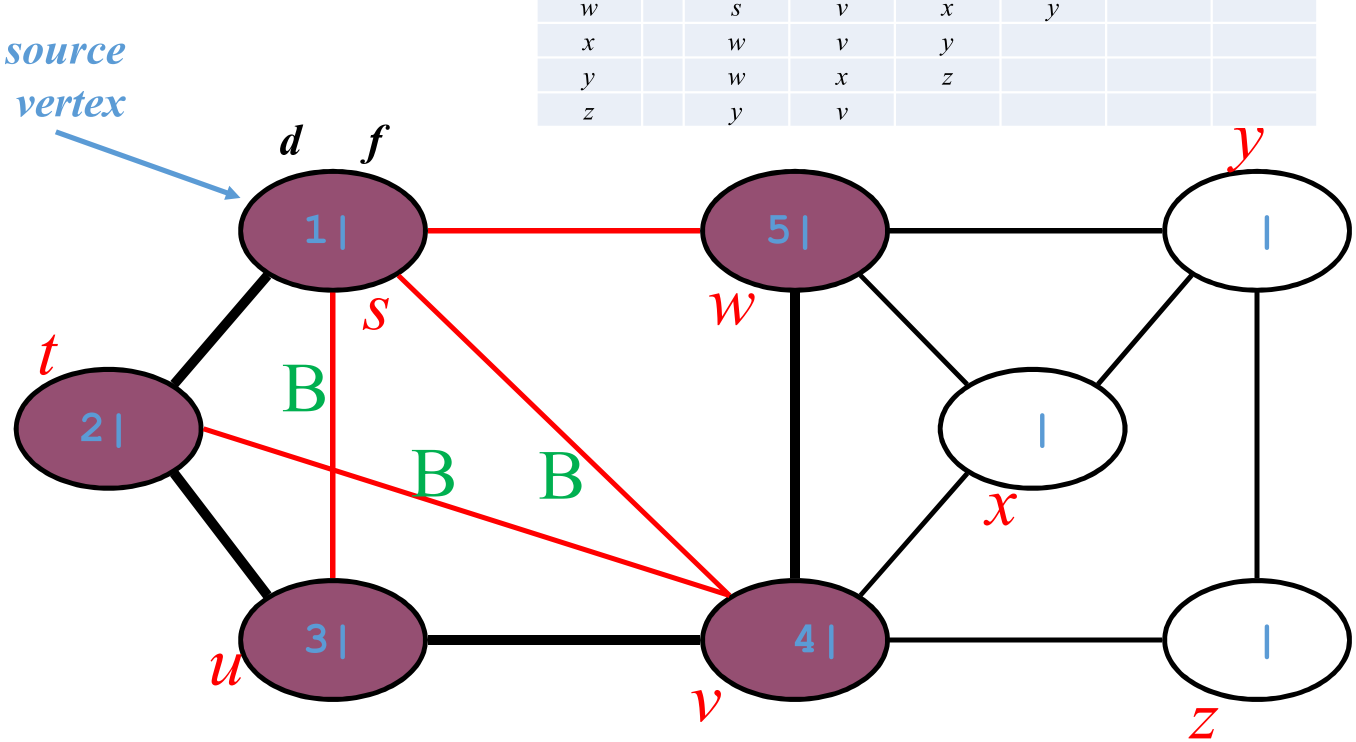
Undirected graph and edges

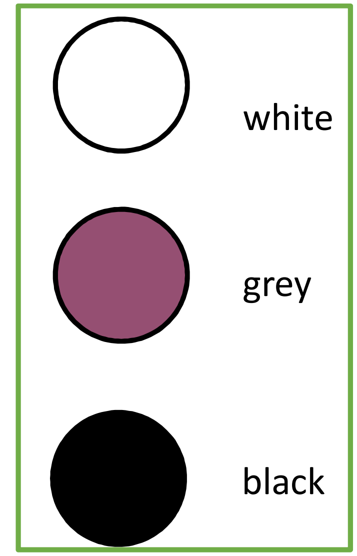
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				



Undirected graph and edges

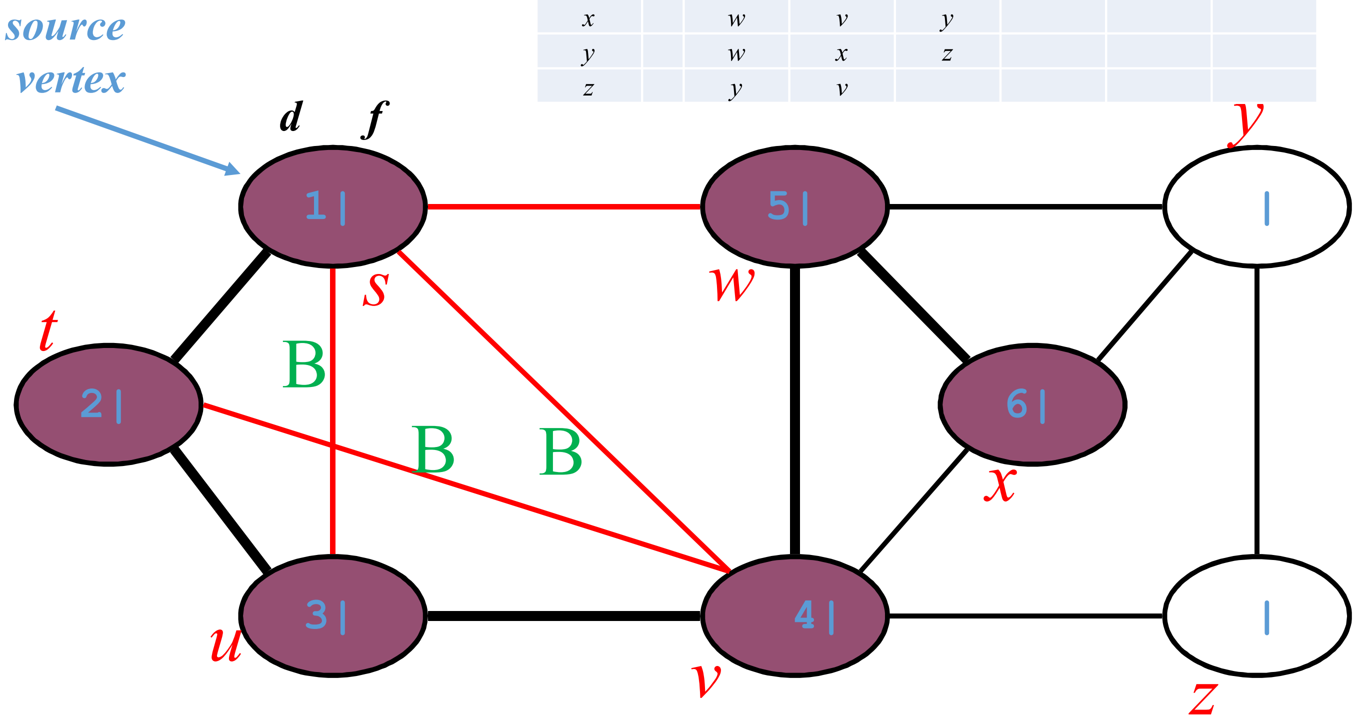
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

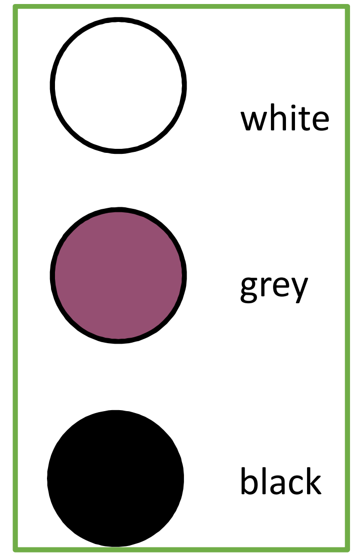




Undirected graph and edges

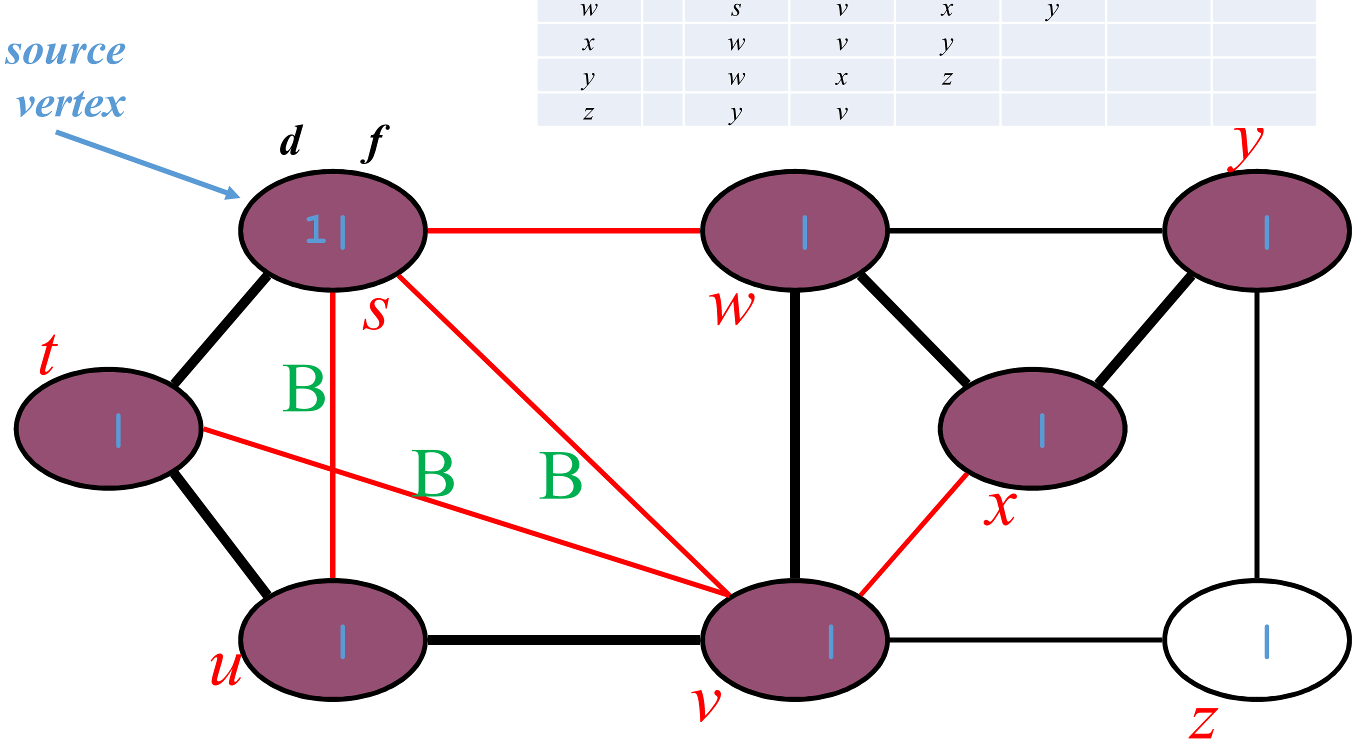
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

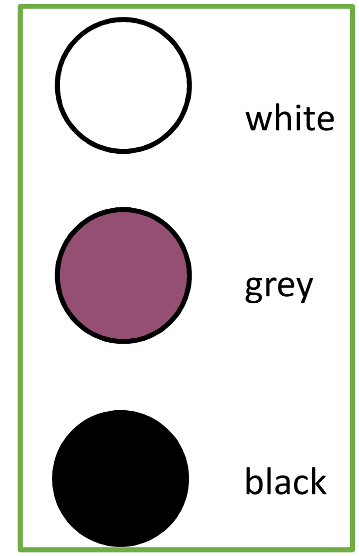




Undirected graph and edges

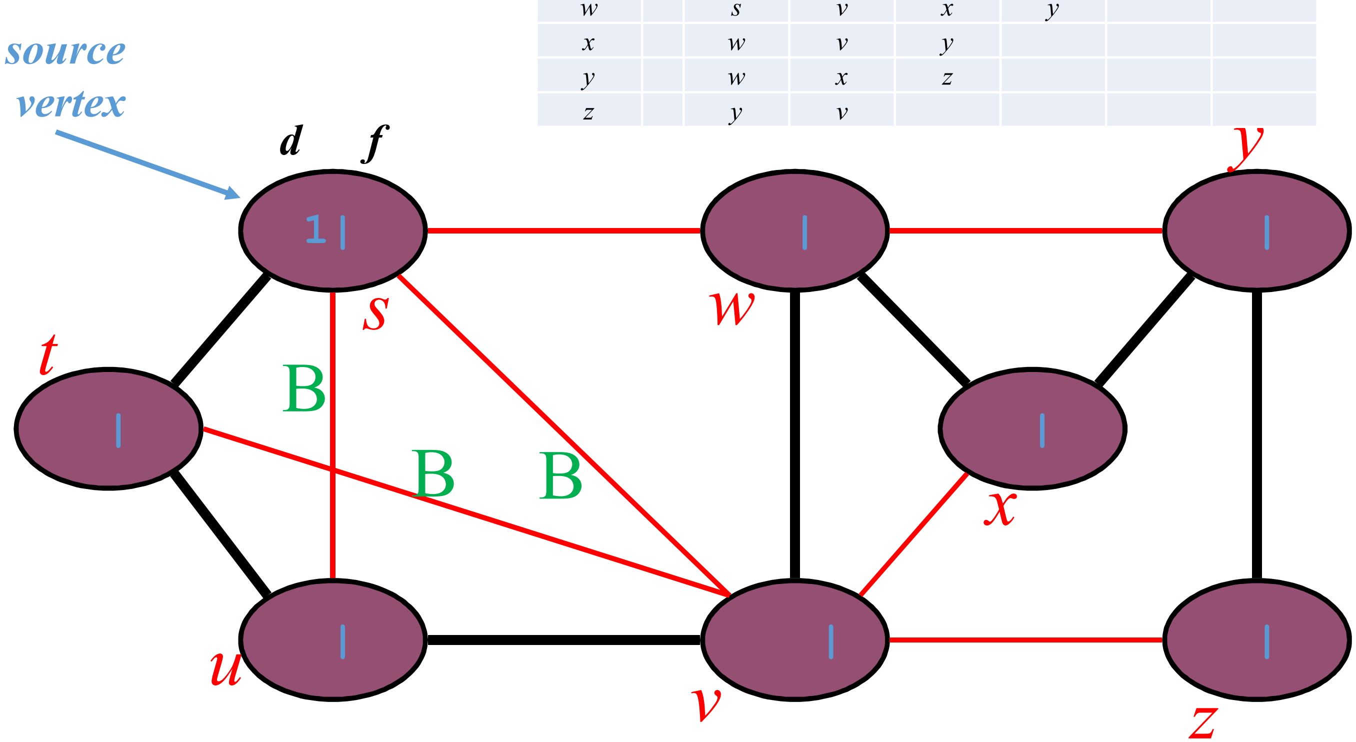
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				

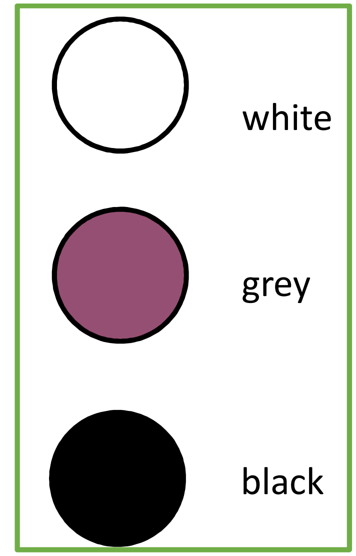




Undirected graph and edges

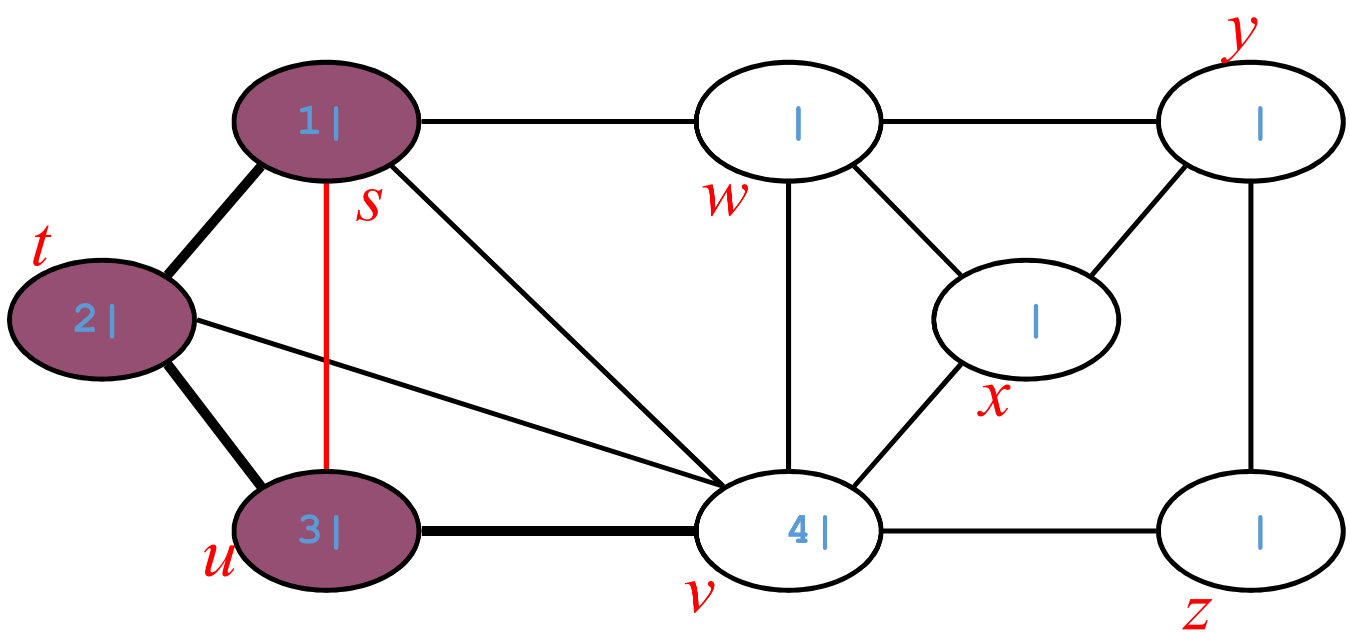
Nodes		Adjacency list					
s		*t*	*u*	*v*	*w*		
t		*s*	*u*	*v*			
u		*s*	*t*	*v*			
v		*s*	*t*	*u*	*w*	*x*	*z*
w		*s*	*v*	*x*	*y*		
x		*w*	*v*	*y*			
y		*w*	*x*	*z*			
z		*y*	*v*				





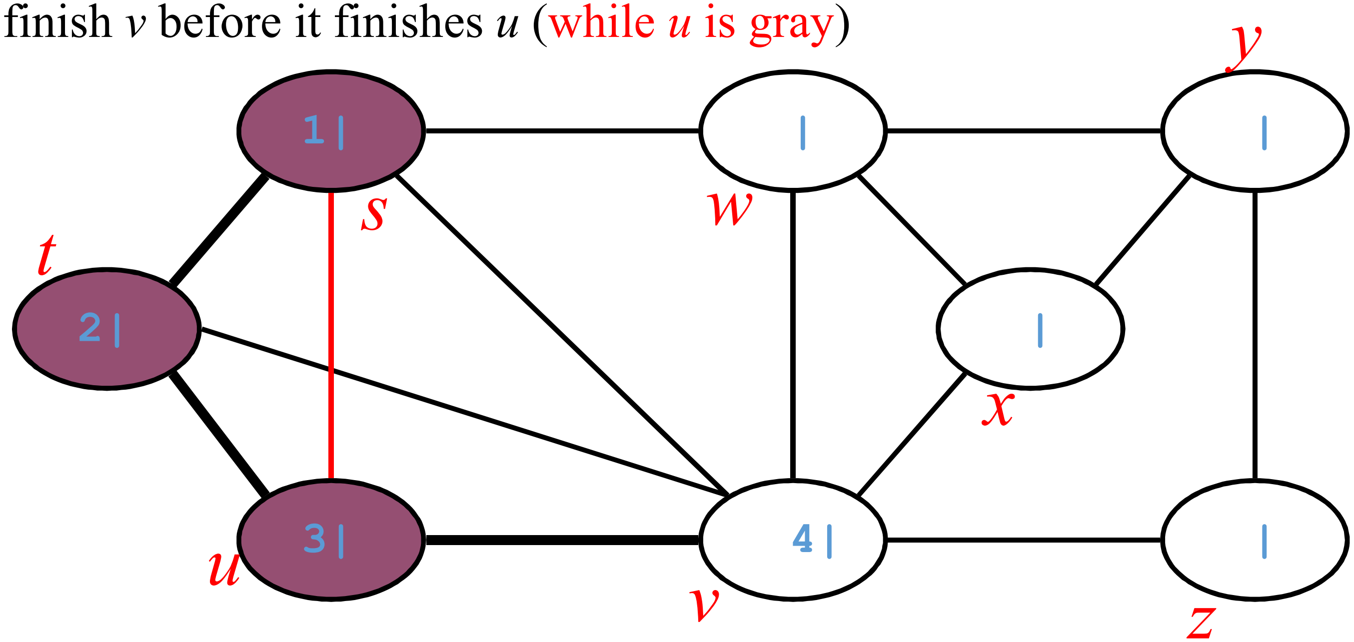
In a DFS of an undirected graph $\boldsymbol{G}$, every edge of $\boldsymbol{G}$ is either a tree edge or a back edge.

- Let (u, v) be an arbitrary edge of G, and suppose without loss of generality that $u.d < v.d$.



In a DFS of an undirected graph *G*, every edge of *G* is either a tree edge or a back edge.

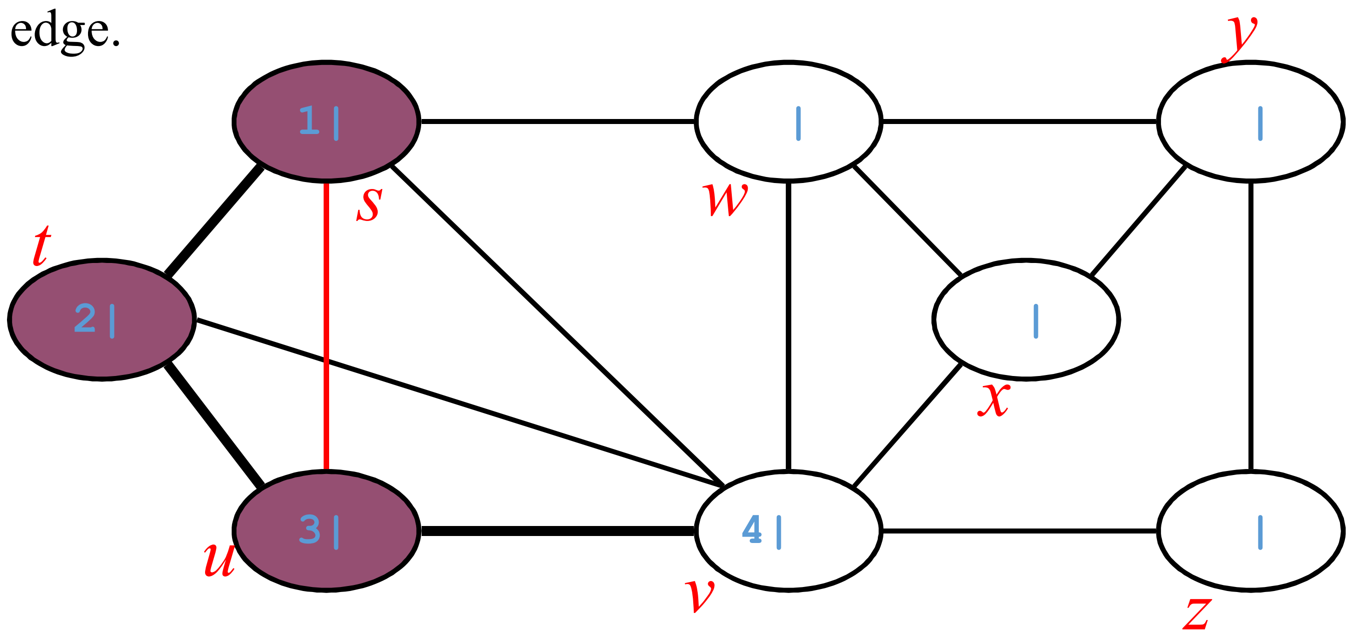
- Let (u, v) be an arbitrary edge of G, and suppose without loss of generality that $u.d < v.d$.
  - v is on u's adjacency list.
  - the search must discover and finish v before it finishes u (while u is gray)



In a DFS of an undirected graph G, every edge of G is either a tree edge or a back edge.

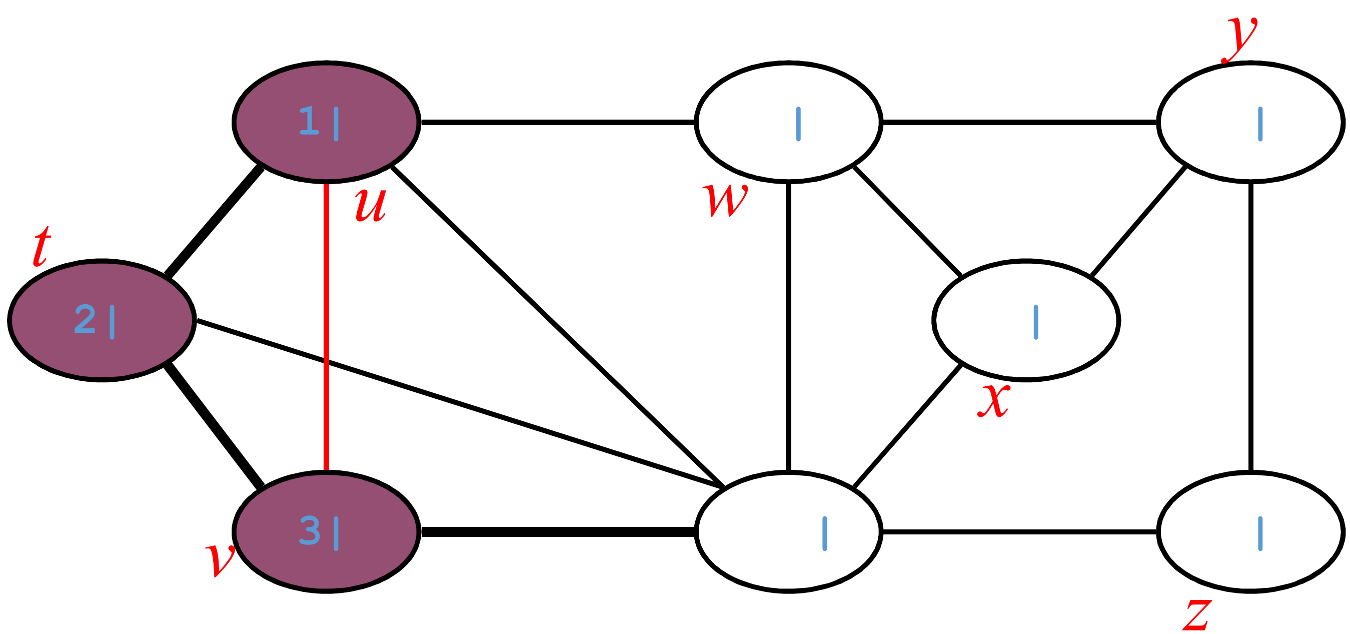
- Case A: The search explores edge (u, v) first in the direction from u to v:
    - then v is undiscovered (white) until that time ($u.d$)
    - Thus, (u, v) becomes a tree edge.

$$u.d < v.d$$



In a DFS of an undirected graph $\boldsymbol{G}$, every edge of $\boldsymbol{G}$ is either a tree edge or a back edge.

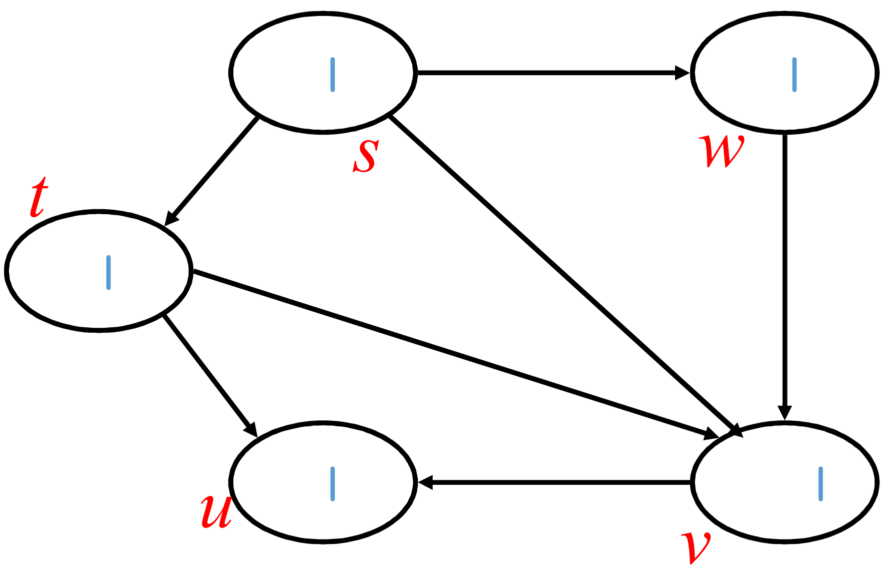
- Case B: The search explores (u, v) first in the direction from v to u:
  - u is still gray at the time the edge is first explored
  - then (u, v) is a back edge.



$u.d < v.d$

Topological sort

- Done on *directed acyclic graph* (*DAG*), $G = (V, E)$
  - makes a linear ordering of vertices: u appears before v if there is an edge (u, v)

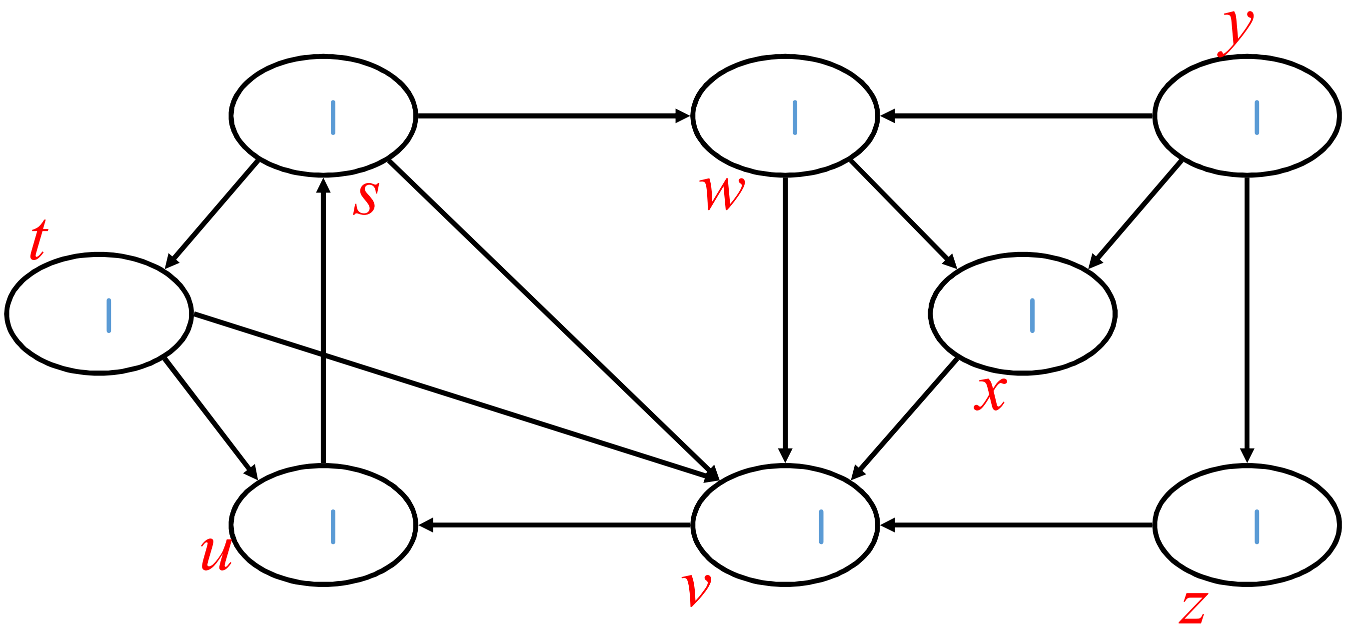


DAG

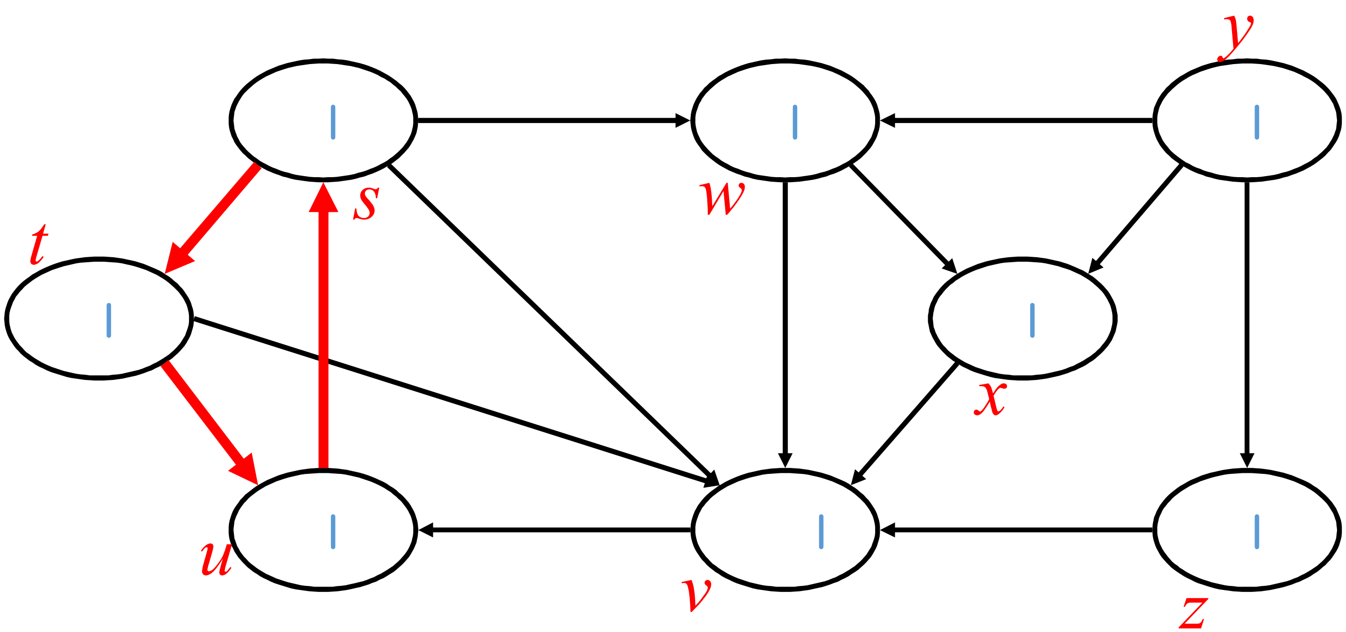
Linear ordering

s, t, w, v, u

IS it a DAG?

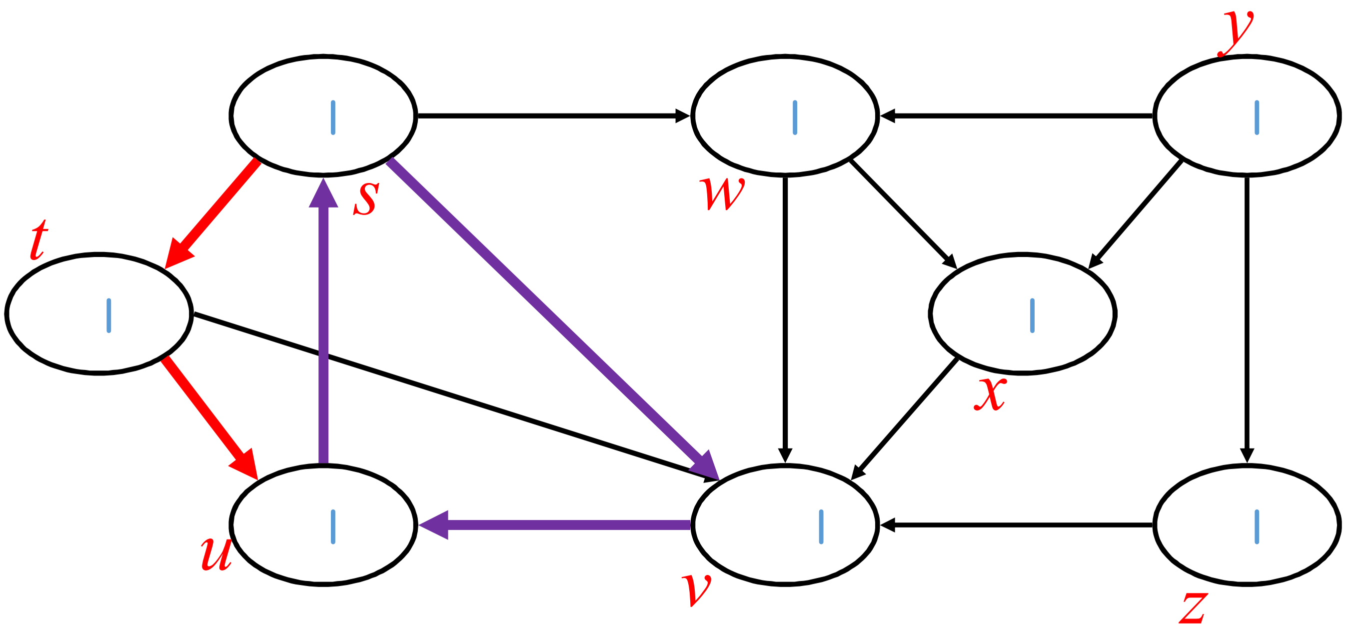


IS it a DAG?



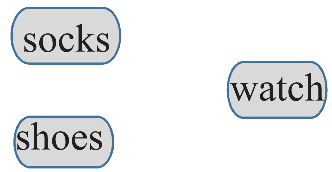
Cycle: s -> t ->, u ->s

IS it a DAG?

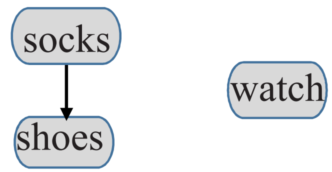


Cycle: s -> v ->, u ->s

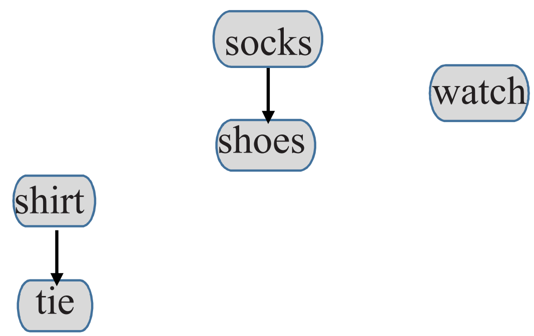
Topological sort Example: dressing of a person



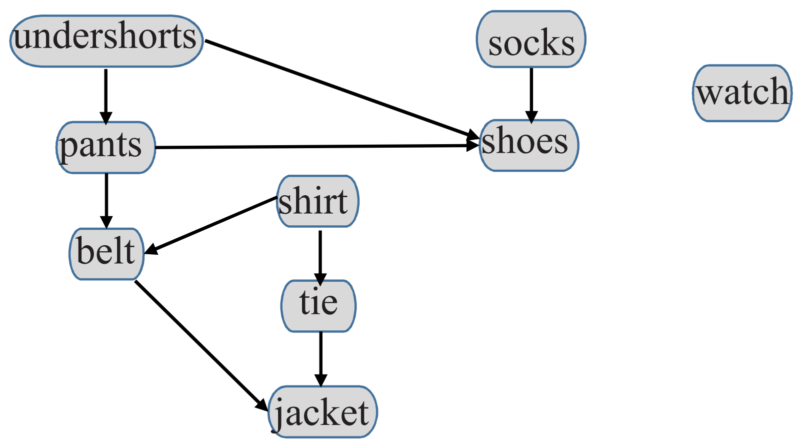
Topological sort Example: dressing of a person



Topological sort Example: dressing of a person

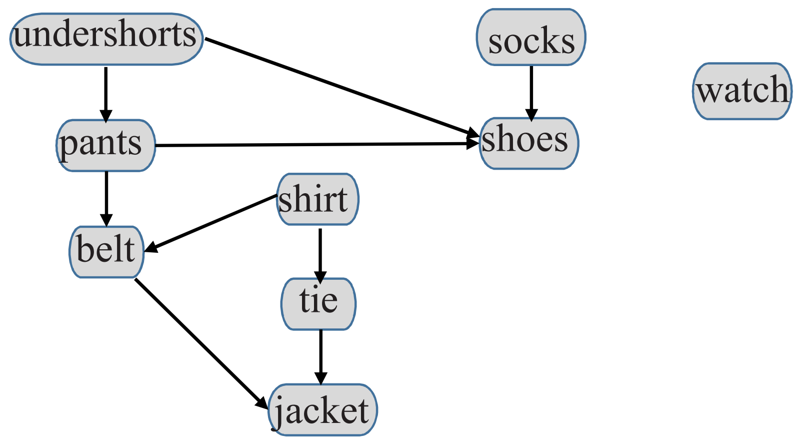


Topological sort Example: dressing of a person

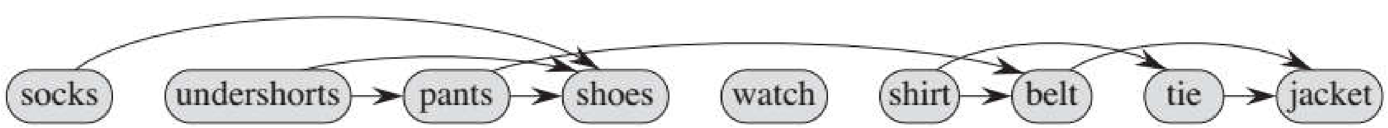


DAG representation of dressing

Topological sort Example: dressing of a person

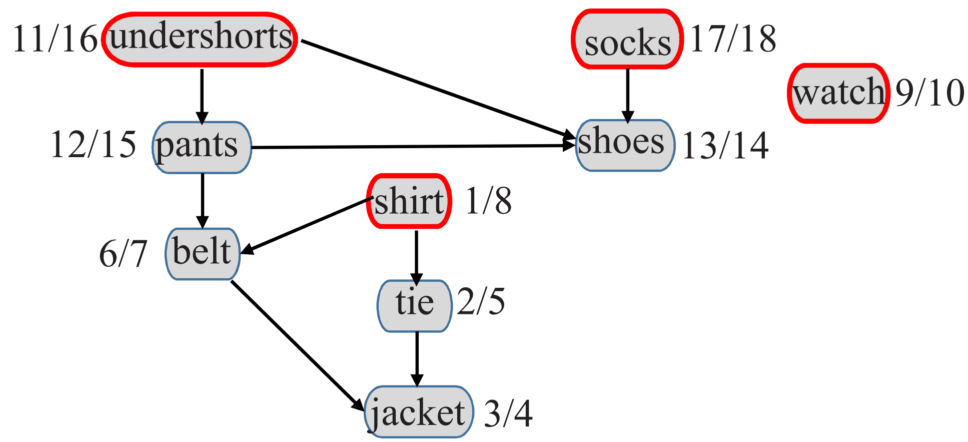


DAG representation of dressing



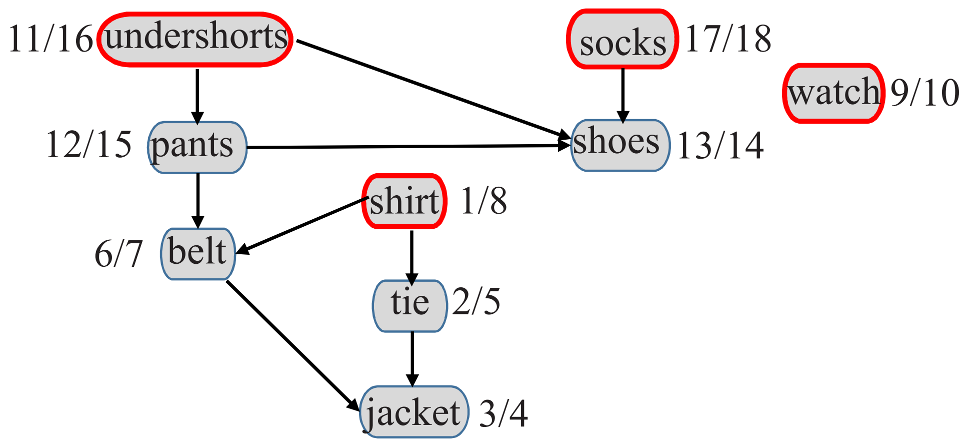
Topologically sorted actions

Topological sort Example: dressing of a person

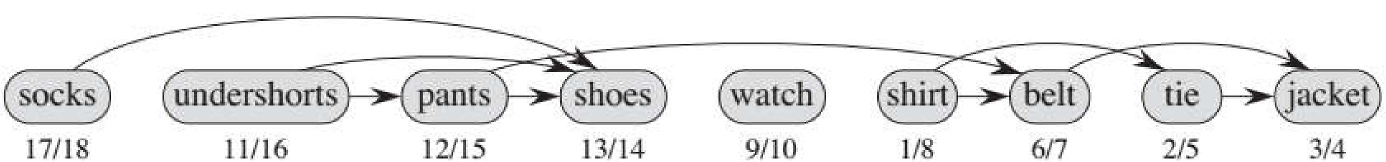


Find finishing times by DFS of the DAG

Topological sort Example: dressing of a person



Find finishing times by DFS of the DAG



sorted by finishing times: use linked list

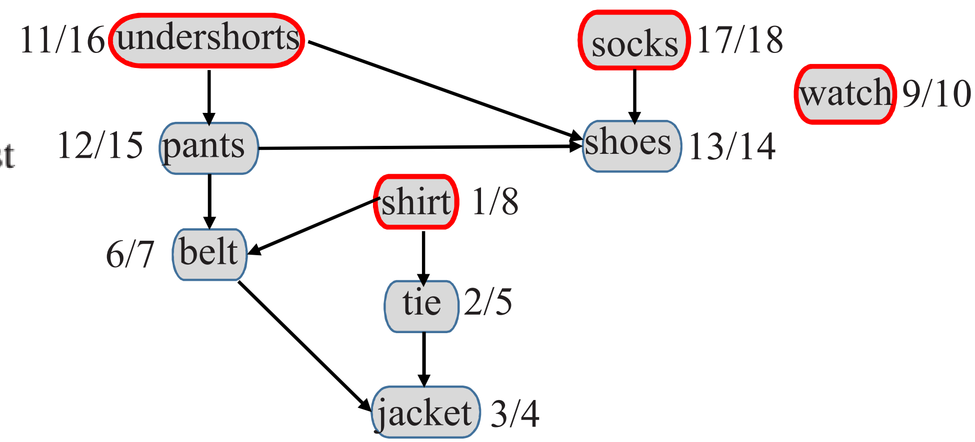
Topological sort Algorithm

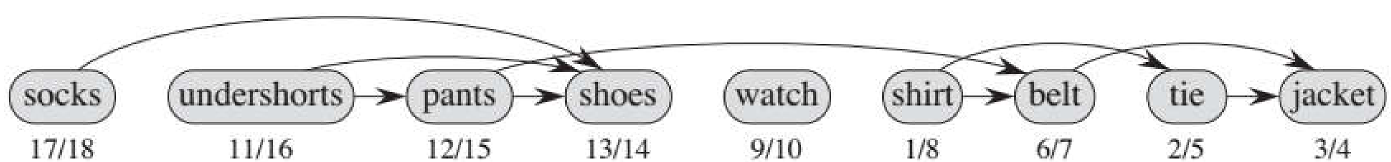
```
TOPOLOGICAL-SORT(G)
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

Topological sort Algorithm

TOPOLOGICAL-SORT(G)

call DFS(G) to compute finishing times $v.f$ for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices





sorted by finishing times: use linked list

Topological sort Algorithm: Complexity

```
TOPOLOGICAL-SORT(G)
    call DFS(G) to compute finishing times v.f for each vertex v  ⟶ O(V+E)
    as each vertex is finished, insert it onto the front of a linked list  ⟶ O(V)
    return the linked list of vertices
```

Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

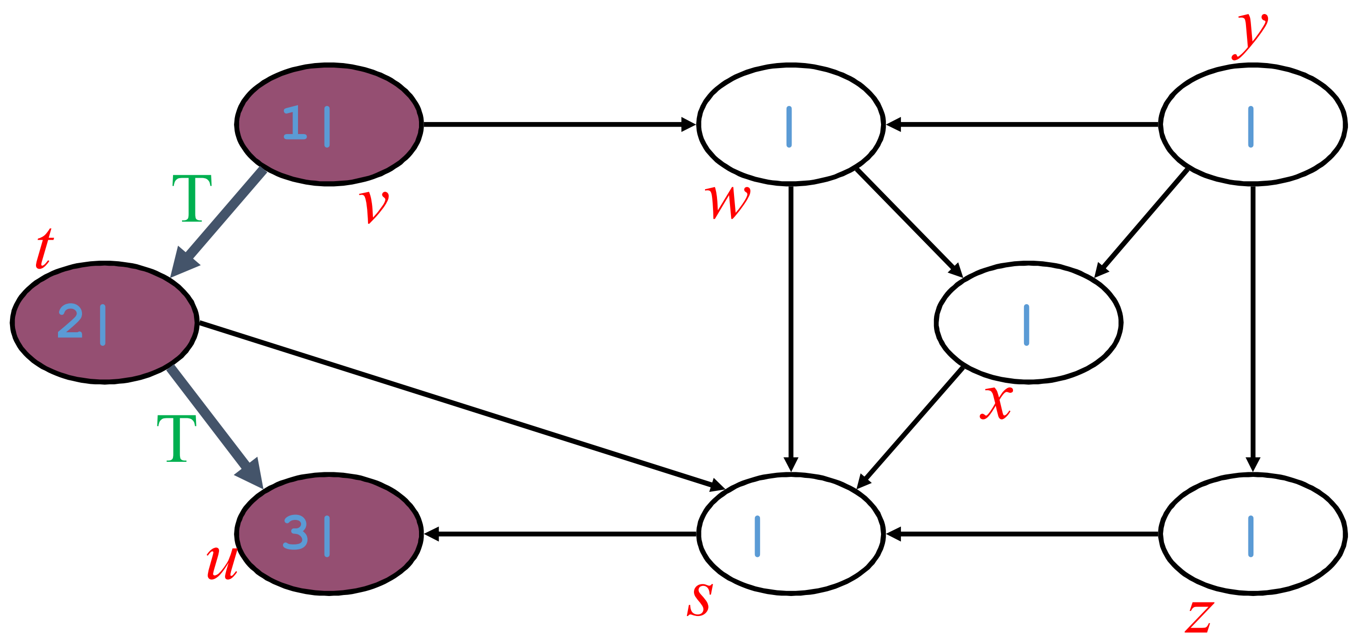
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If P then Q:

Let G is a DAG. Prove that G has no back edge.



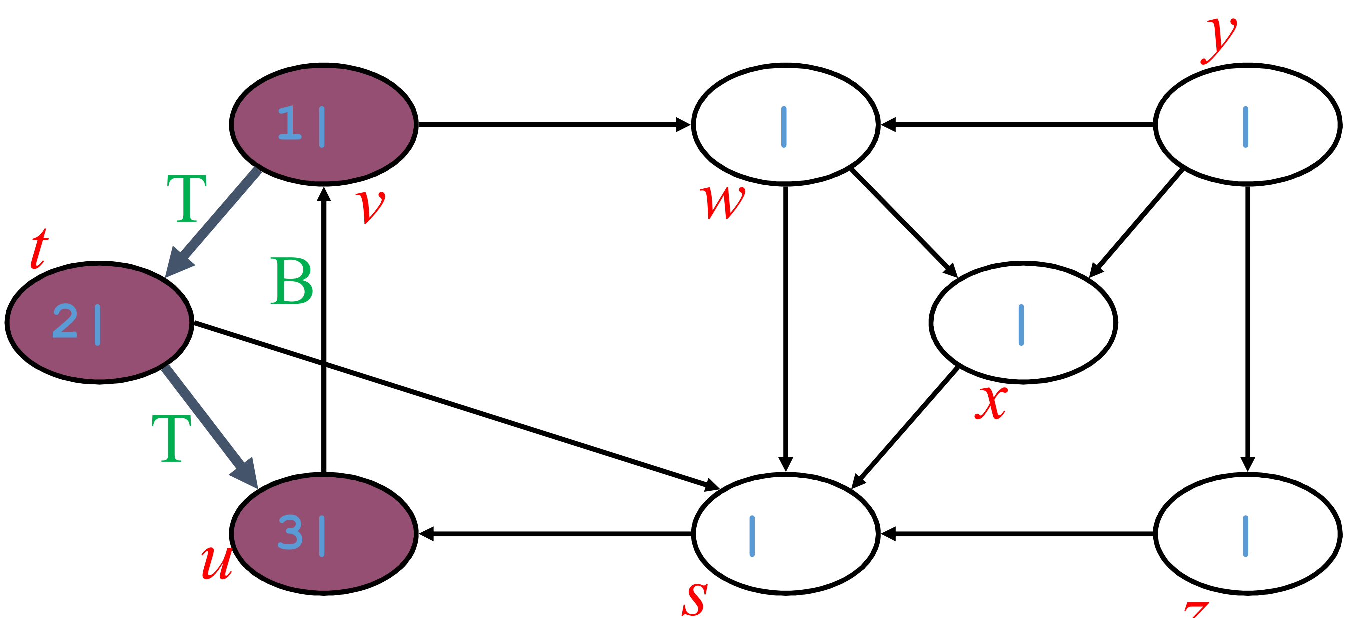
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If P then Q:

Let G is a DAG.
If G has a back edge (u, v)
=> v is an ancestor of u.
⇒ There is path from v to u



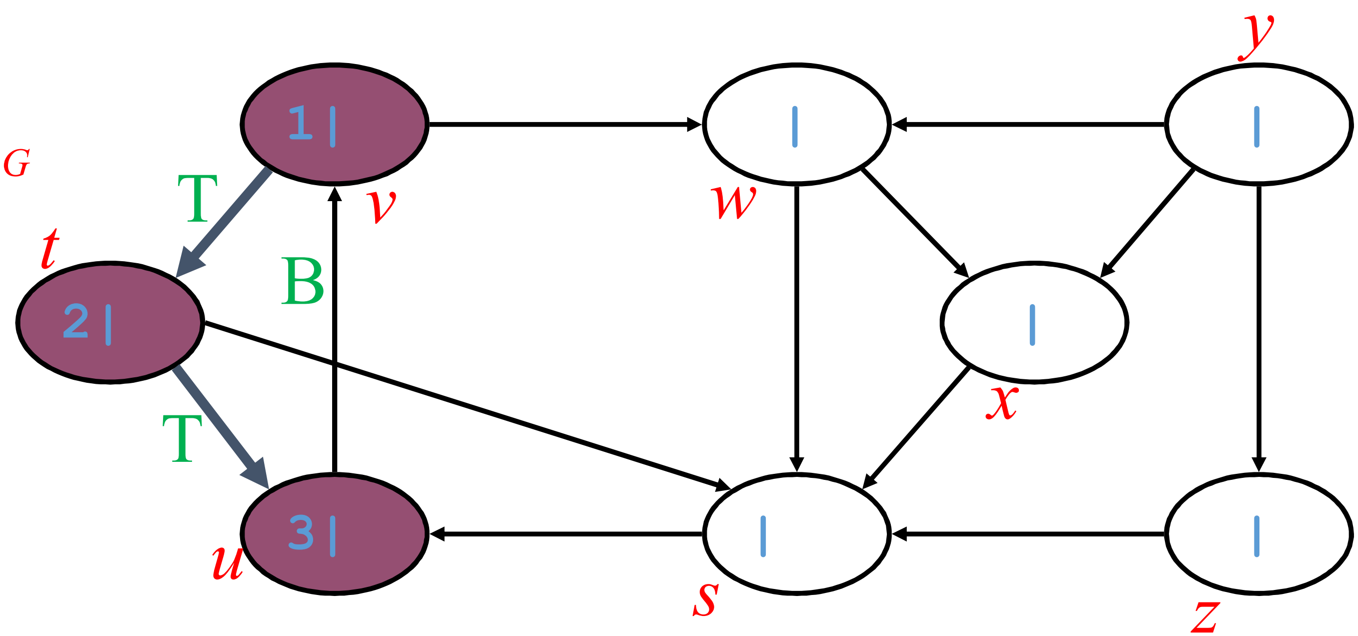
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If P then Q:

Let G is a DAG.
If G has a back edge (u, v)
=> v is an ancestor of u.
⇒ There is path from v to u
⇒ adding an edge (u, v) makes a cycle in G



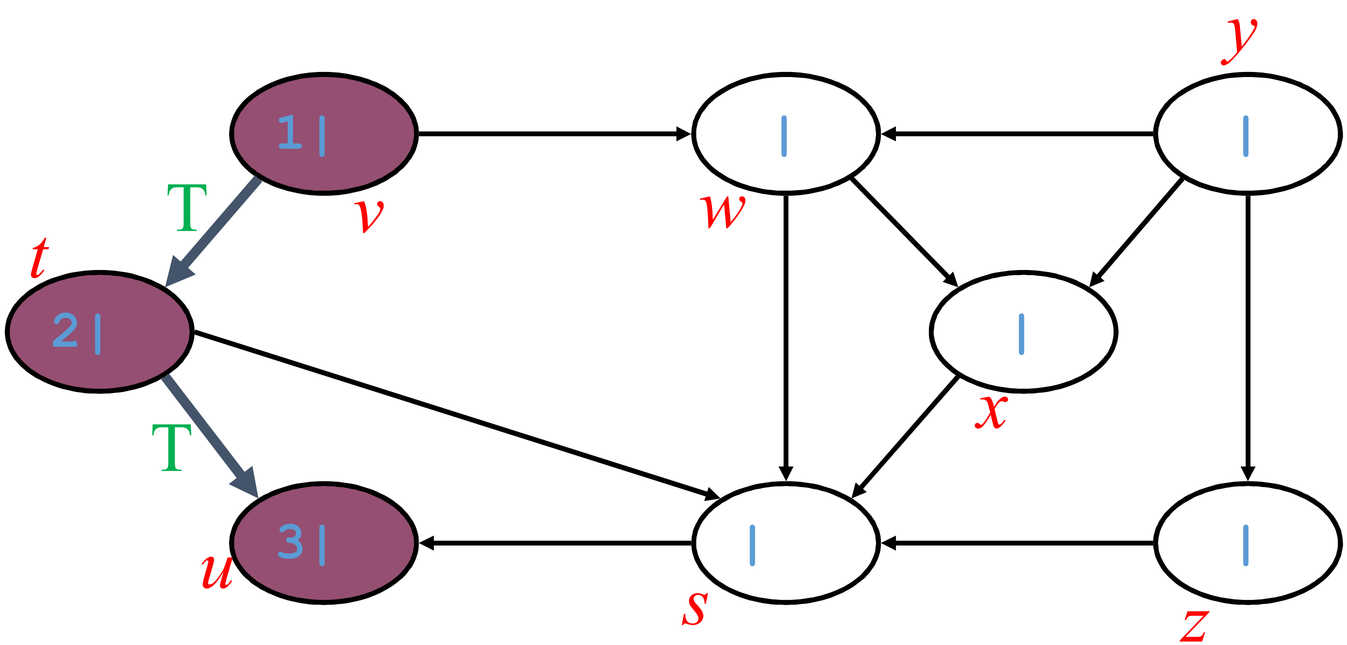
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P   Q

If Q then P:

Let G has no back edge. Prove that G is a DAG.



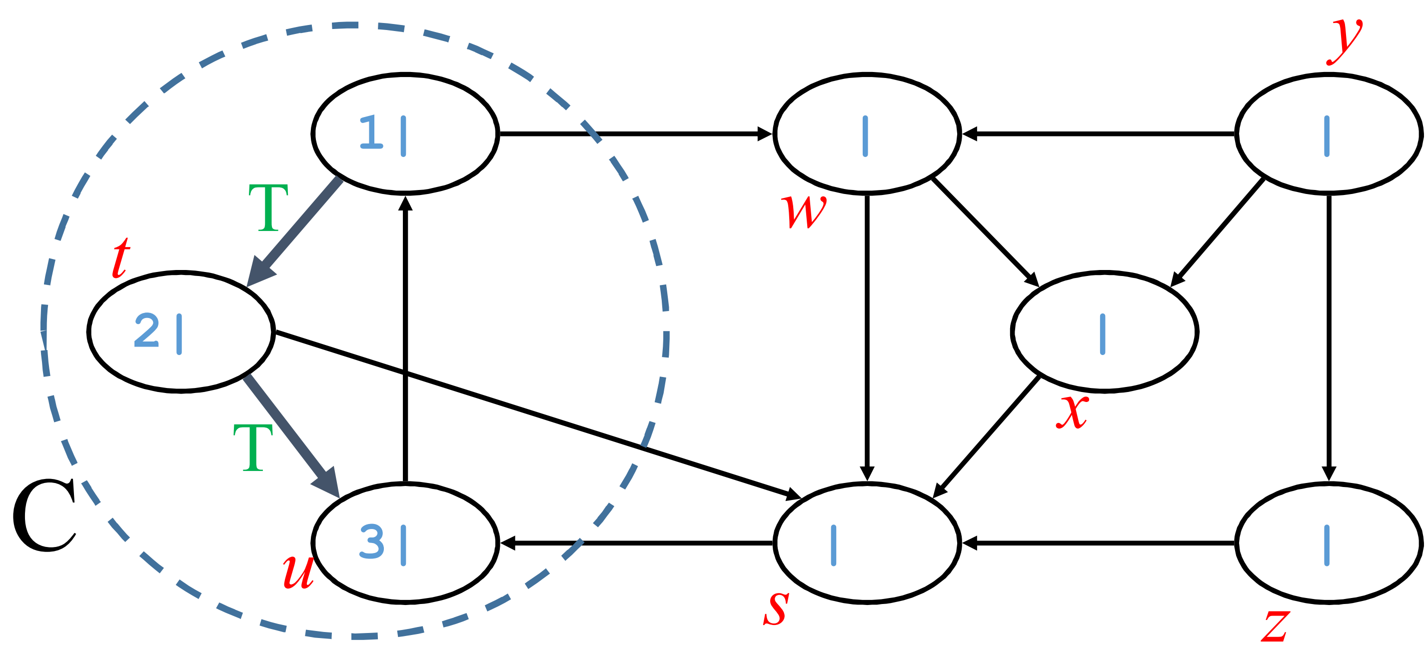
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If Q then P:

Let G has no back edge.
Assume that G has cycle C.



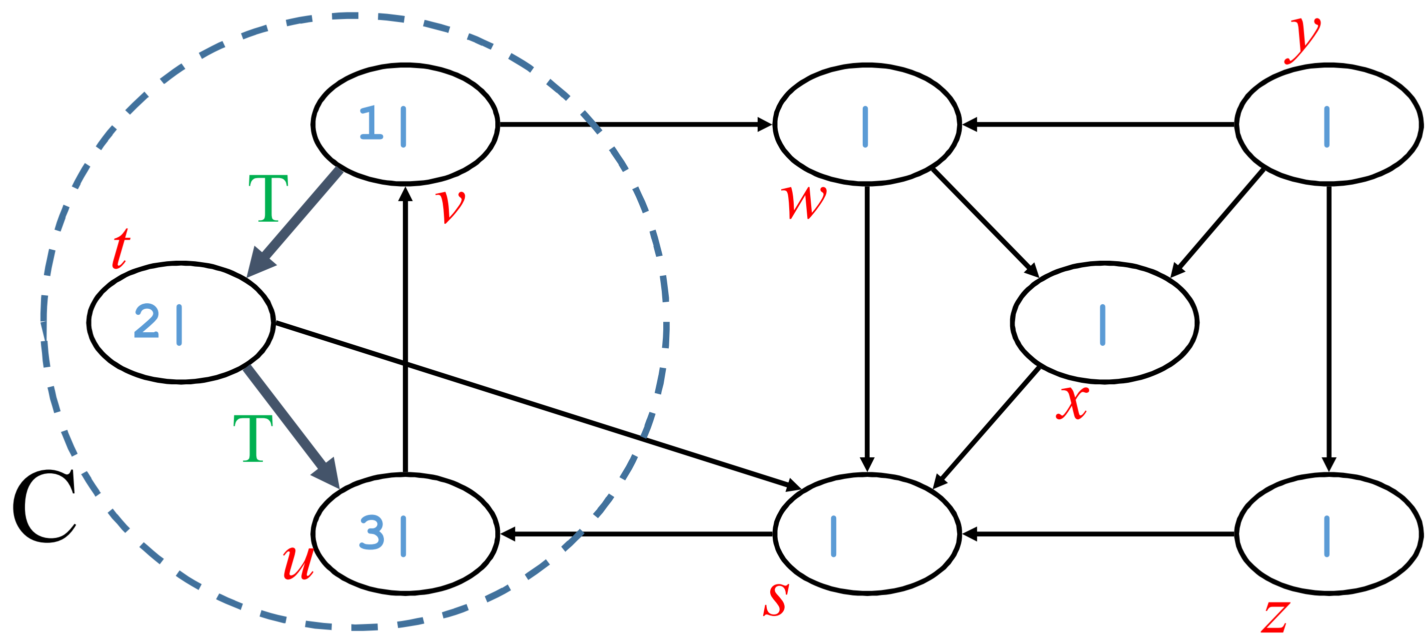
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If Q then P:

Let G has no back edge.
Assume that G has cycle C
Let v be the first vertex in C and
(u, v) is the preceding edge.



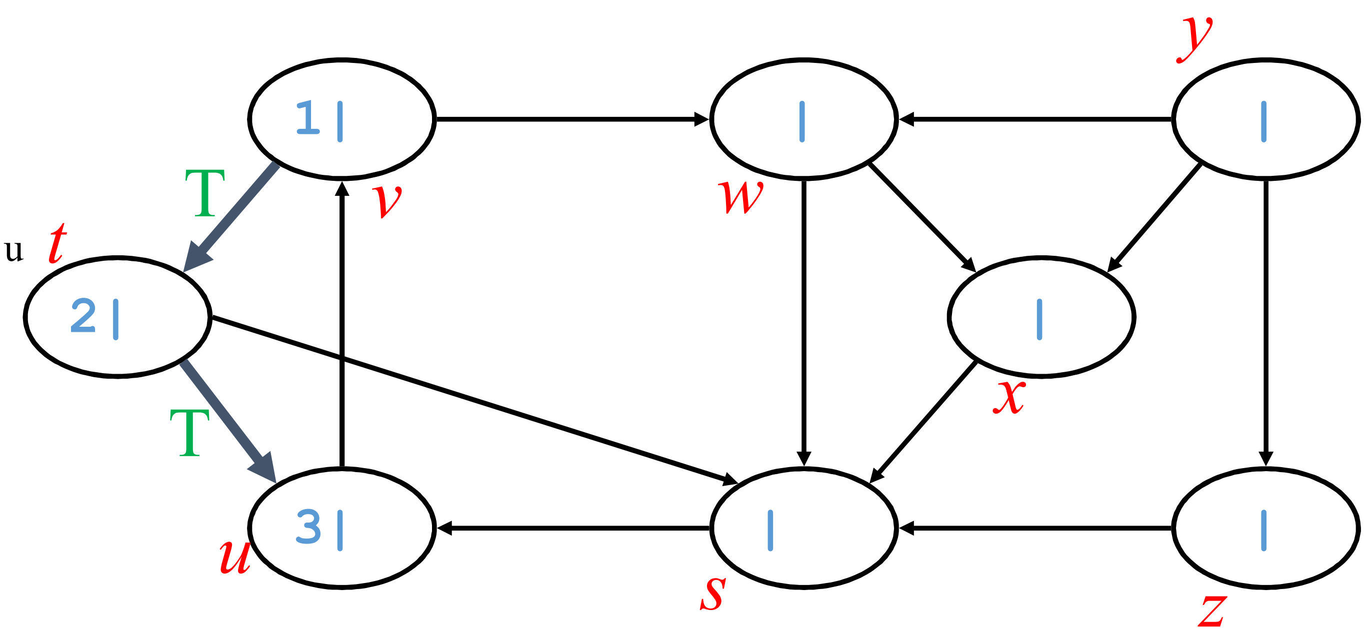
Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.
P Q

If Q then P:

Let G has no back edge.
Assume that G has cycle C
Let v be the first vertex in C and
(u, v) is the preceding edge.

At $v.d$, v, t, u are all white
$\Rightarrow$ There is path of white vertices from v *to* u



Lemma 22.11

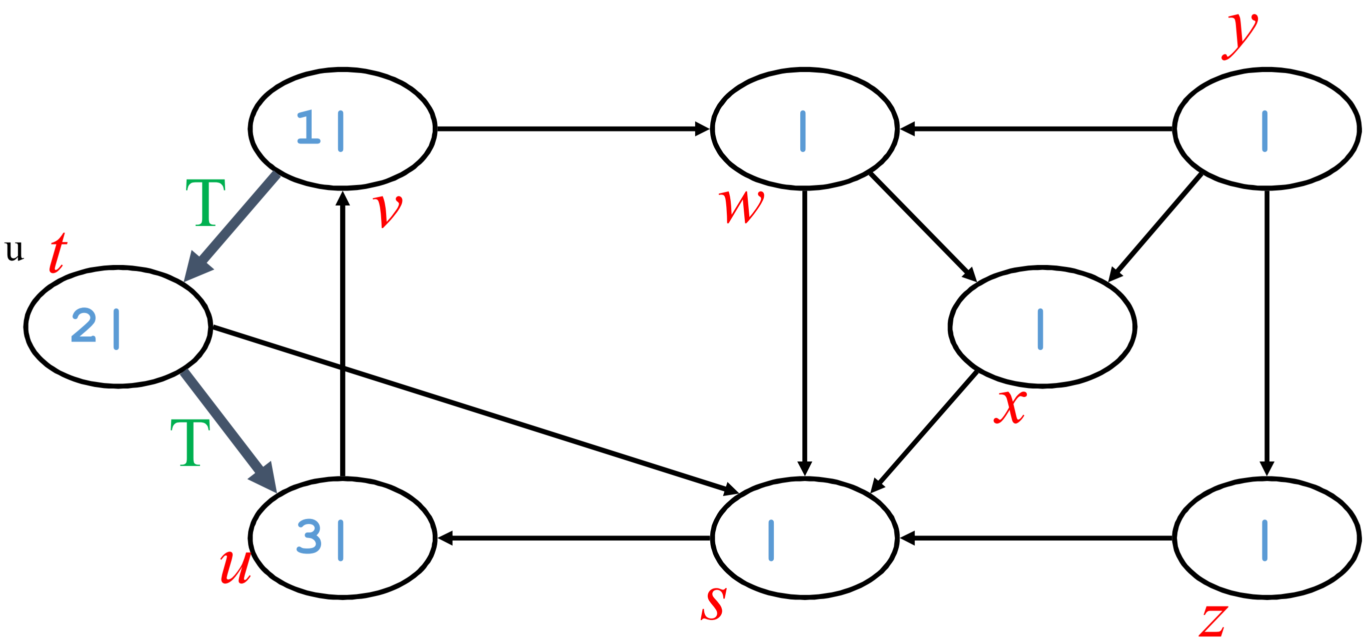
A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If Q then P:

Let G has no back edge.
Assume that G has cycle C.
Let v be the first vertex in C and
(u, v) is the preceding edge.

At $v.d$, v, t, u are all white
$\Rightarrow$ There is path of white vertices from v *to* u
$\Rightarrow$ u is a descendant of v



Lemma 22.11

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

P Q

If Q then P:

Let G has no back edge.
Assume that G has cycle C.
Let v be the first vertex in C and
(u, v) is the preceding edge.

At $v.d$, v, t, u are all white
$\Rightarrow$ There is path of white vertices from v *to* u
$\Rightarrow$ u is a descendant of v
$\Rightarrow$ (u, v) is a back edge