Graph Search:

Depth-First Search and Topological Sort

Fatemeh Panahi
Department of Computer Science
University of Toronto

CSC263-Fall 2017
Lecture 9

Today

- Depth First Search (DFS)
- Edge classifications
- Topological sort

Anouncements

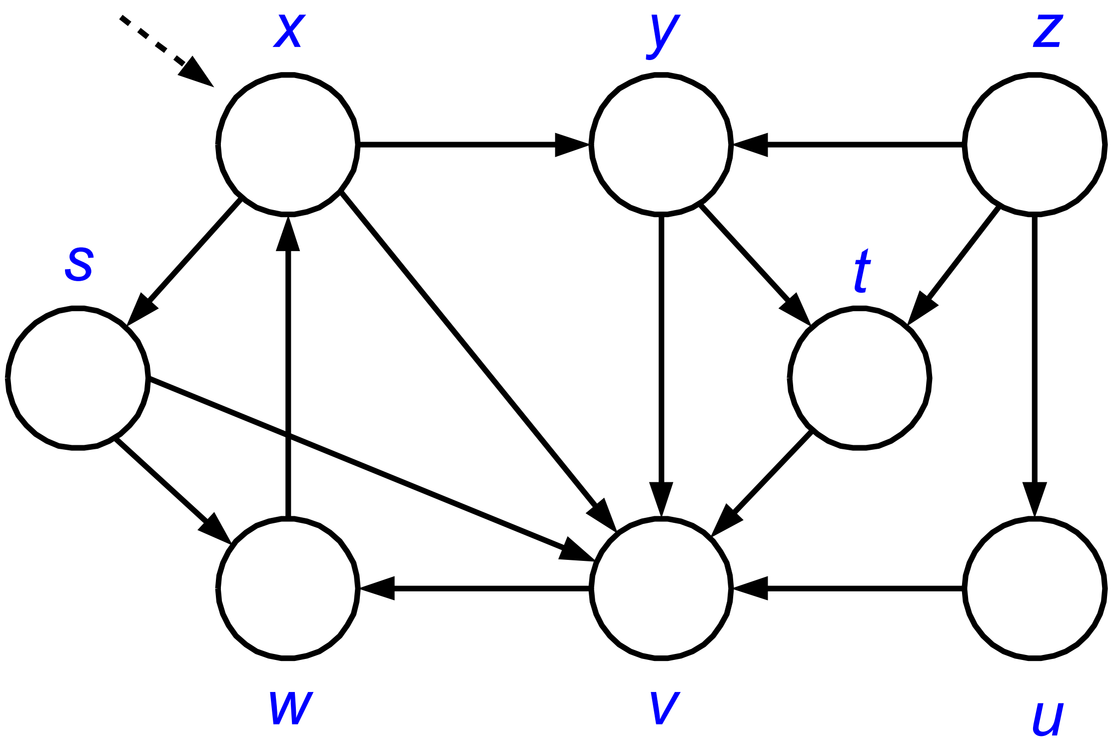
- A3 due to Friday
- Quiz next week
- Tutorial on Friday: 22.3-5, 22.3-8, 22.4-2, 22-3

Breadth-First Search

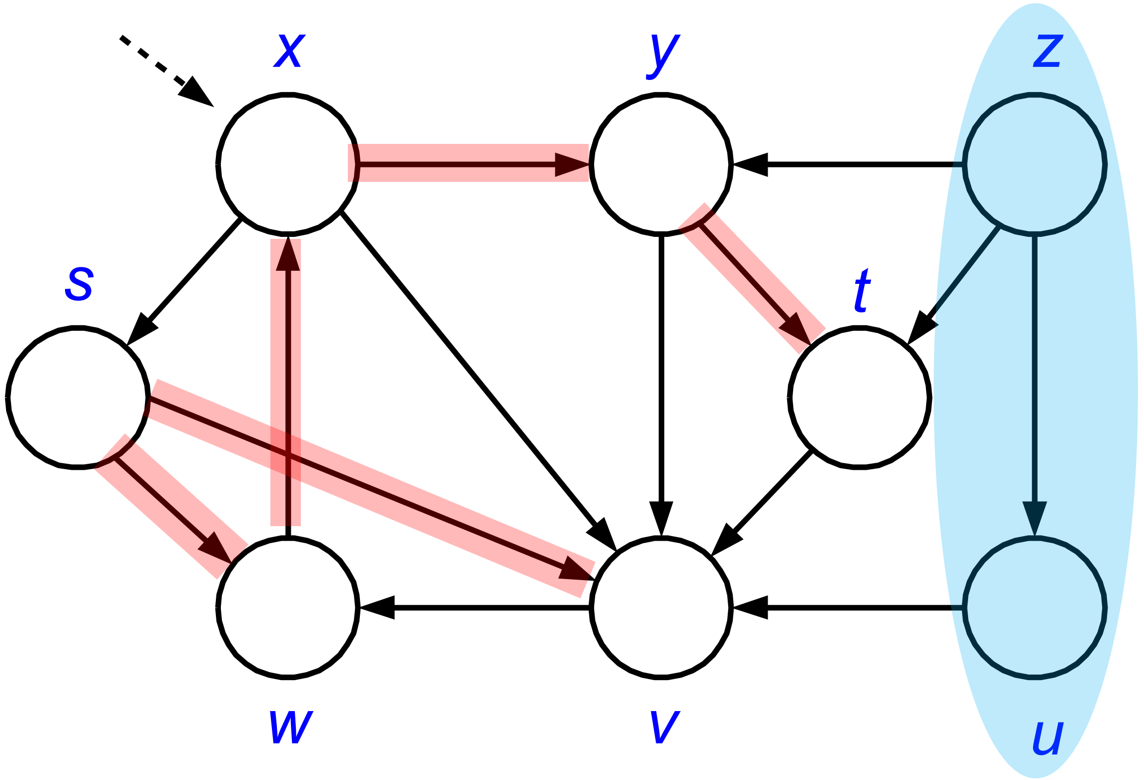
GOAL: Explores edges of G to

- discover every vertex reachable from the source vertex s
- compute the shortest path distance of every vertex from the source vertex s
- produce a breadth-first tree (BFT) G_{Π} with root s

BFS (s)



BFS (s)



Depth-First Search

- Graph $G=(V,E)$ directed or undirected
- Adjacency list representation
- Goal: Systematically explore every vertex and every edge
- Idea: search deeper whenever possible
  - Using a LIFO stack (FIFO queue used in BFS)

Depth-First Search

- Like BFS, colors the vertices to indicate their states.
  Each vertex is
  - Initially white,
  - grayed when **discovered**,
  - blackened when **finished**
- Like BFS, records **discovery** of a white v during scanning $Adj[u]$ by $v.\pi \leftarrow u$

Depth-First Search

- Unlike BFS, predecessor graph G_π produced by DFS forms spanning forest
- $G_\pi = (V, E_\pi)$ where

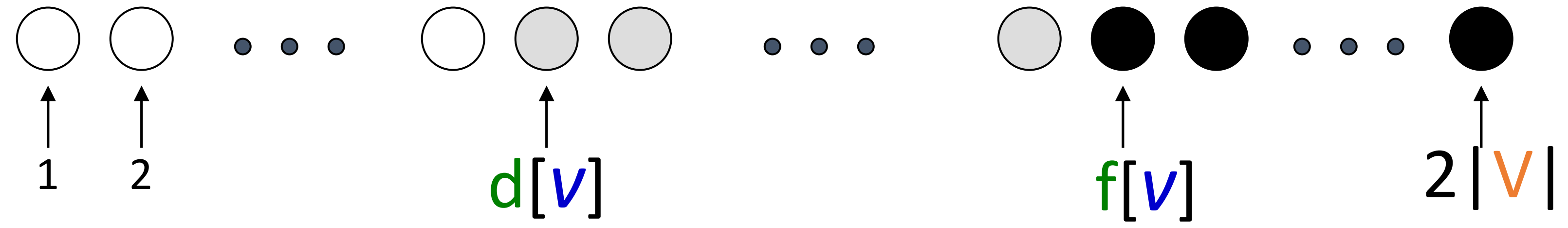
$$E_\pi = \{(v.\pi, v):\ v \in V \ and \ v.\pi \neq NIL\}$$

- G_π = depth-first forest (DFF) is composed of disjoint depth-first trees (DFTs)

Depth-First Search

- DFS also timestamps for each vertex with two timestamps
- $v.d$: records when v is first discovered and grayed
- $v.f$: records when v is finished and blackened
- Since there is only one discovery event and finishing event for each vertex we have $1 \leq v.d < v.f \leq 2|V|$

Time axis for the color of a vertex



Algorithm

DFS-VISIT(G, u)

$u.color \leftarrow gray$

$time \leftarrow time + 1$

$u.d \leftarrow time$

for each $v \in \mathrm{Adj}[u]$ **do**

  if $v.color = white$

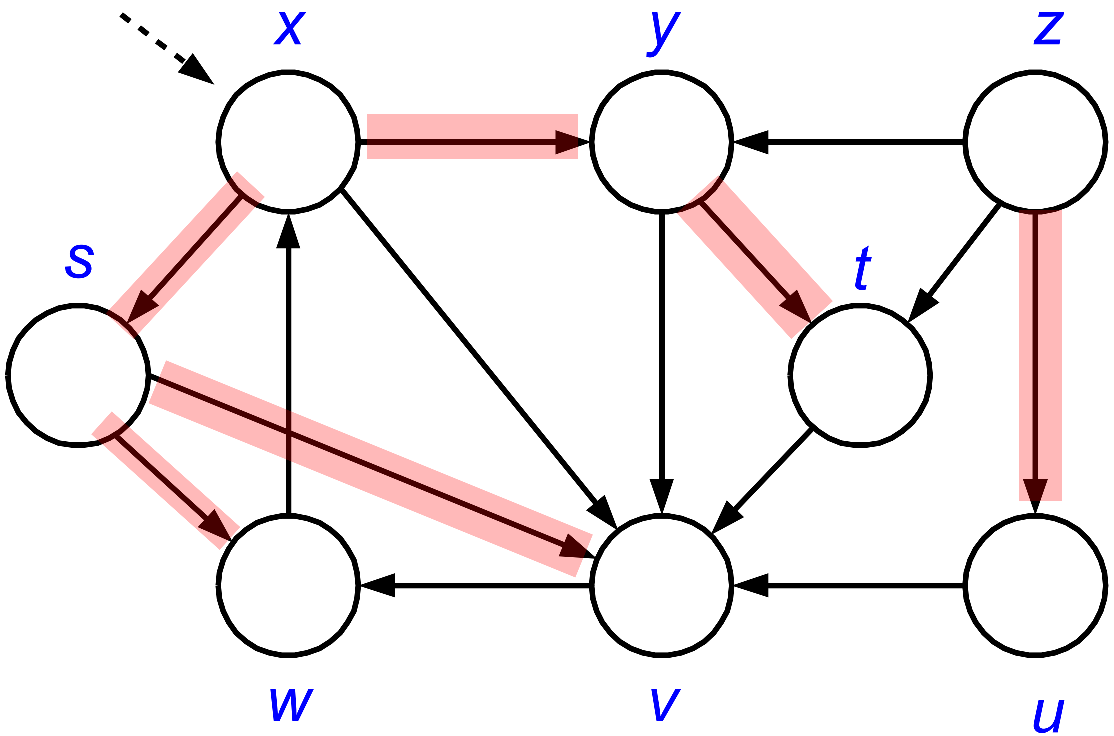
    $v.\pi \leftarrow u$

    DFS-VISIT(G, v)

$u.color \leftarrow black$

$time \leftarrow time + 1$

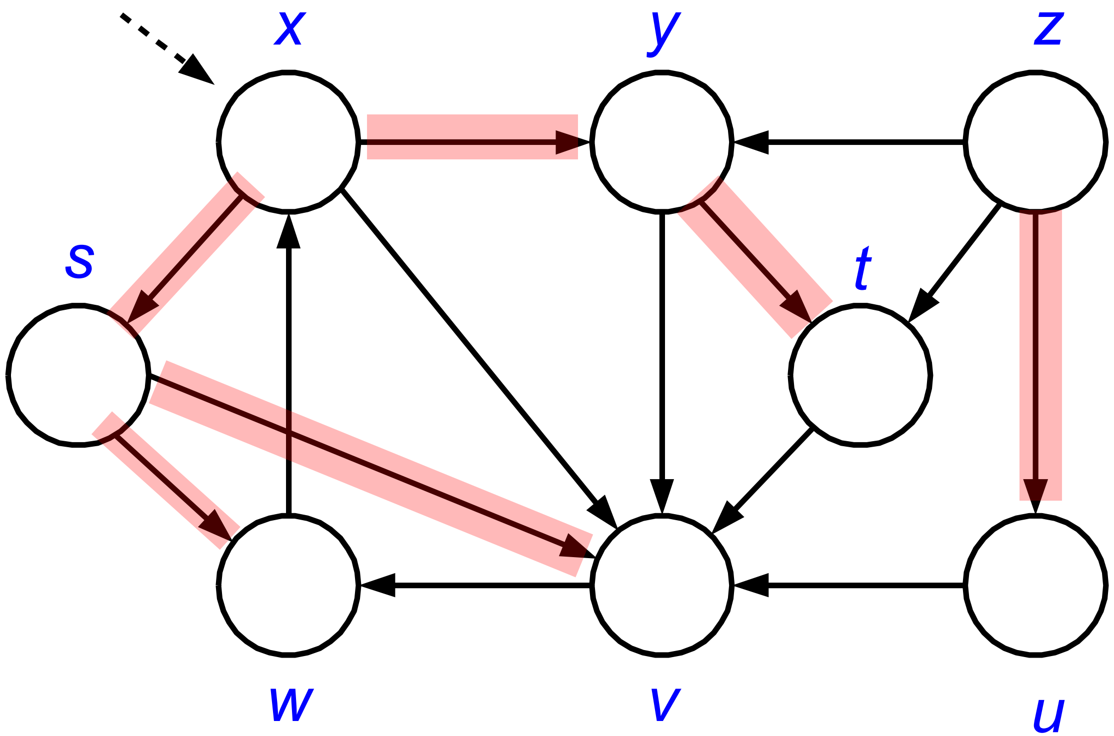
$u.f \leftarrow time$



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

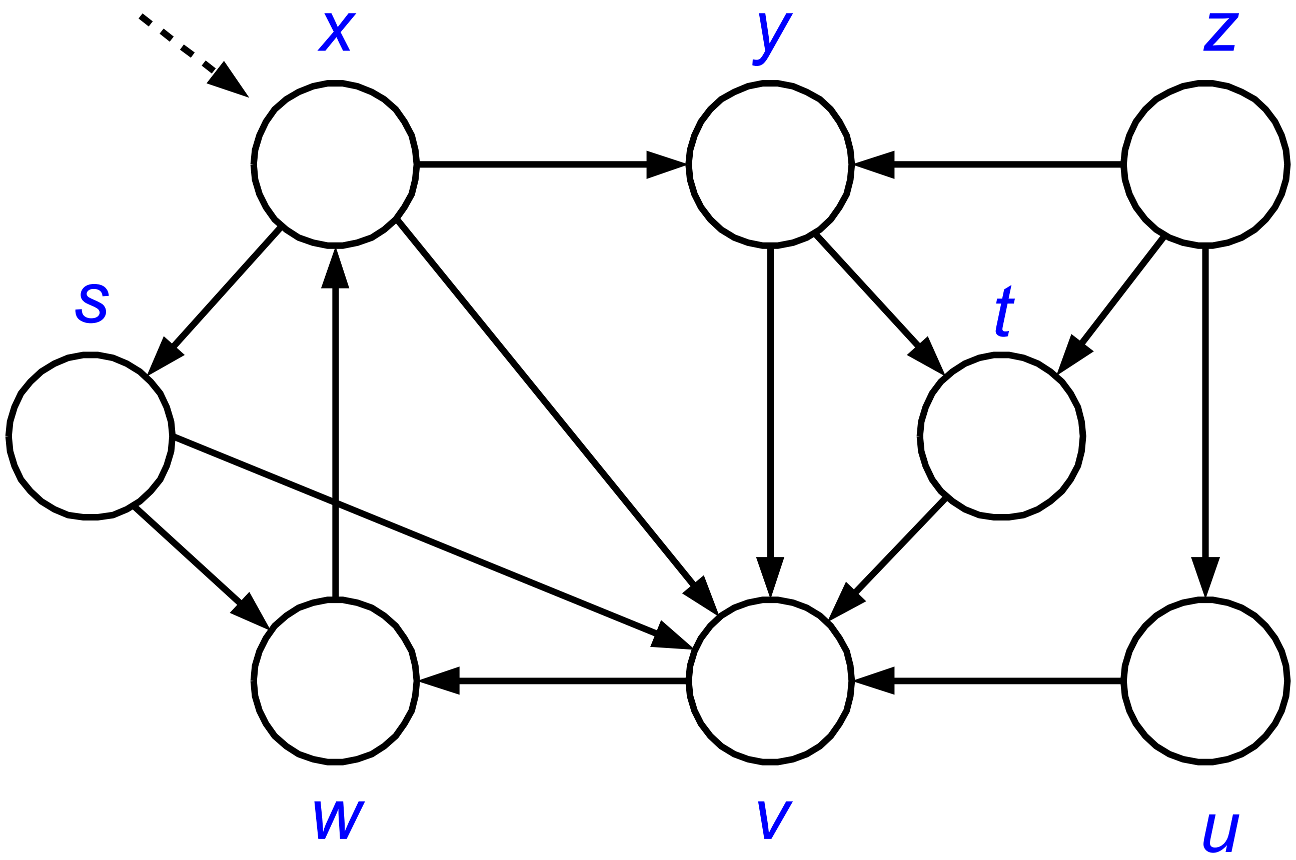
Algorithm

DFS(G)
for each $u \in V$ **do**
  $u.color \leftarrow$ white
  $u.\pi \leftarrow$ NIL
$time \leftarrow 0$
for each $u \in V$ do
  if $u.color =$ white **then**
    DFS-VISIT(G, u)

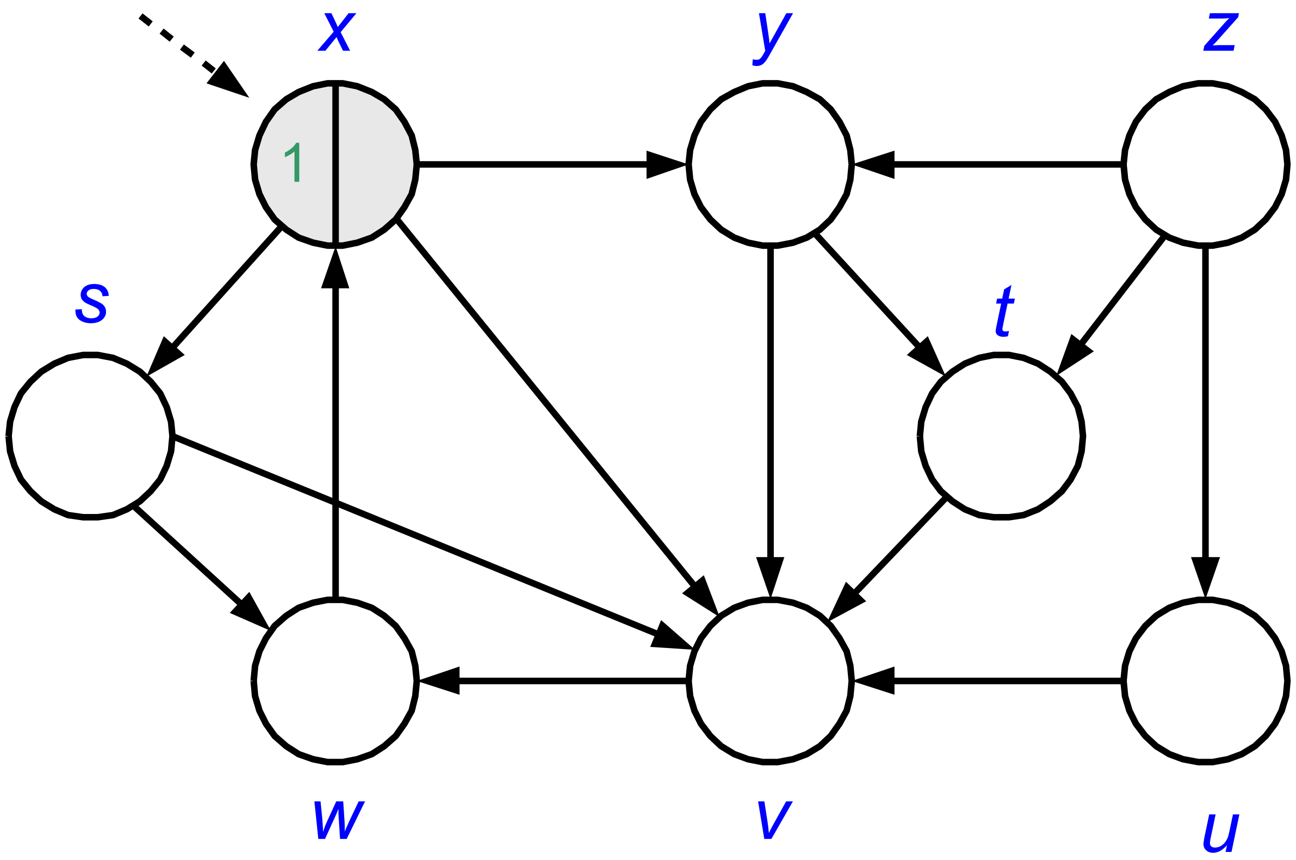


Running time: $\Theta(V + E)$

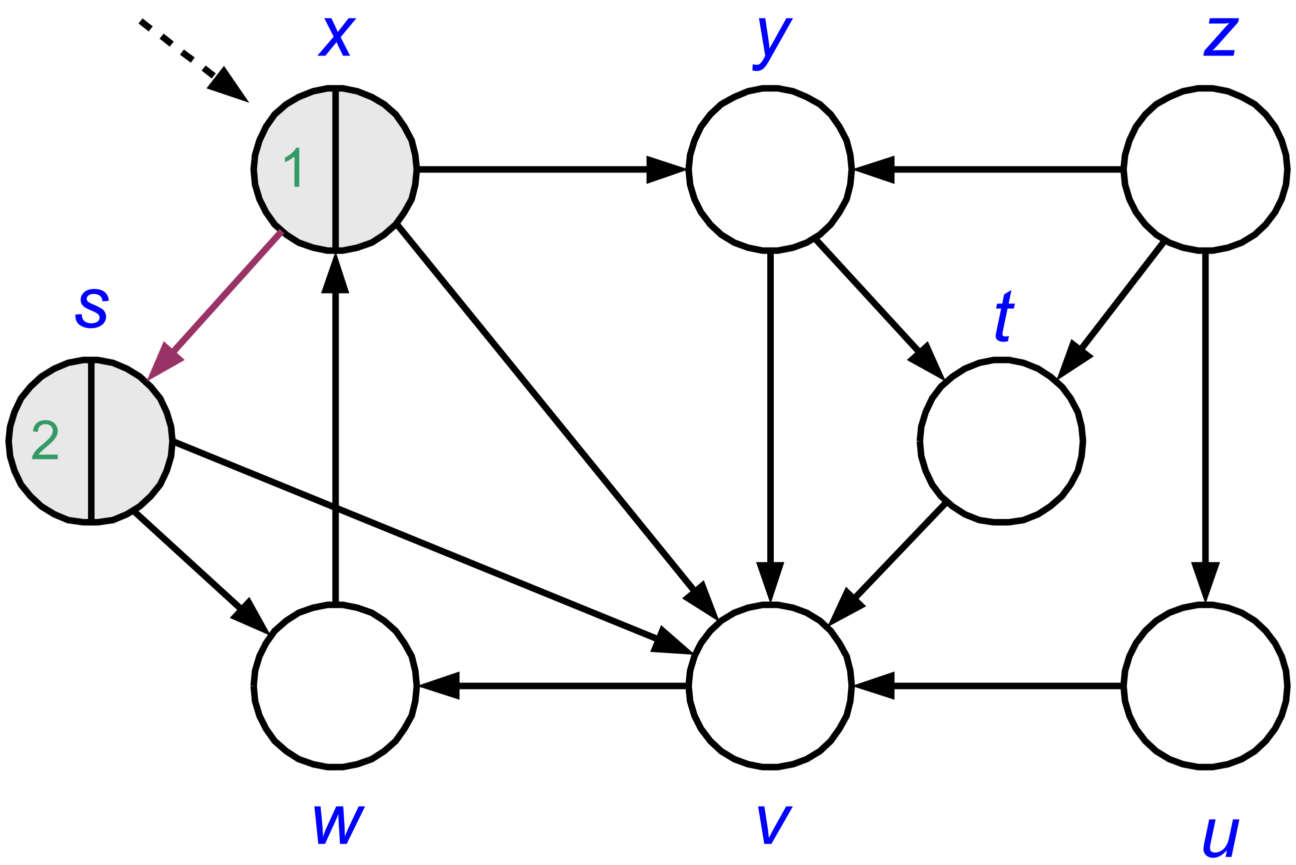
DFS Example



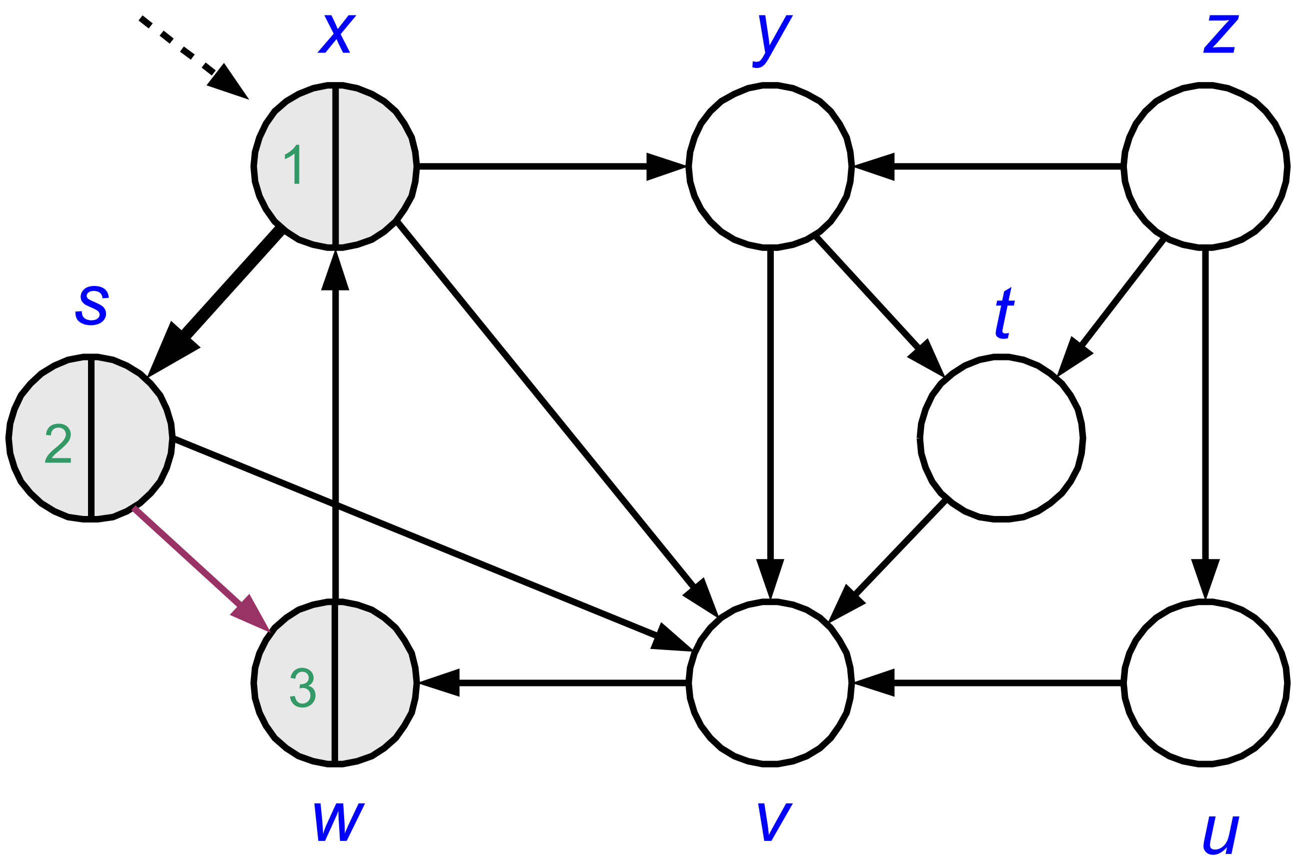
DFS Example



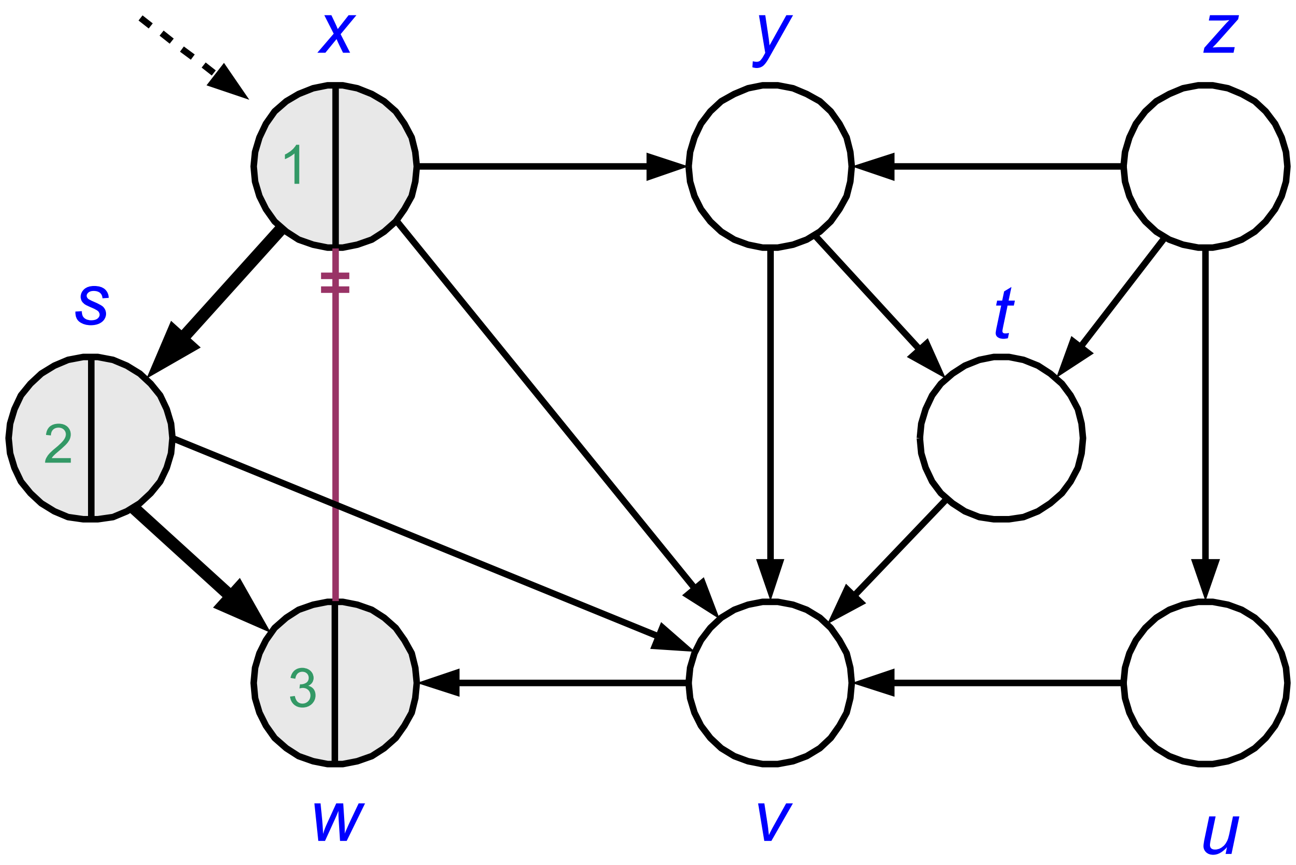
DFS Example



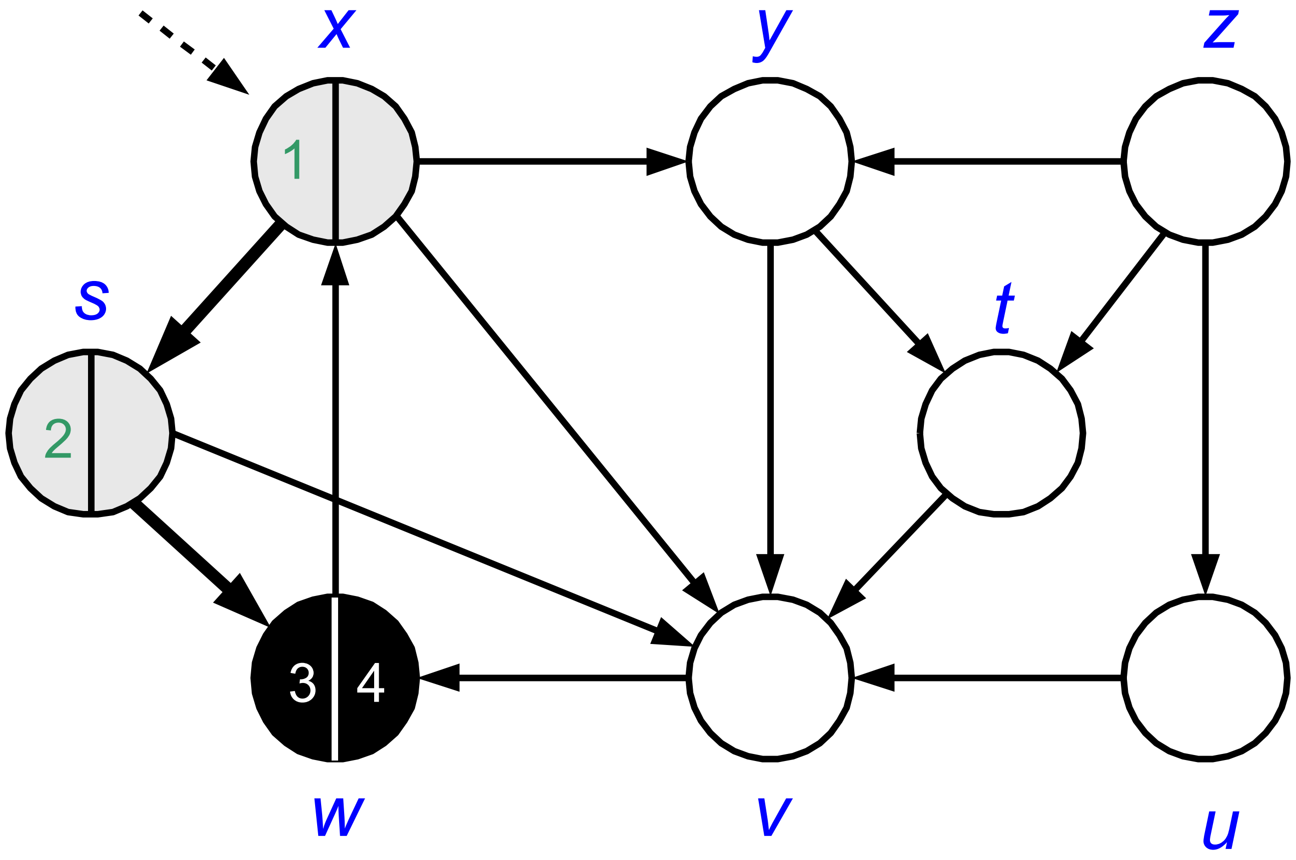
DFS Example



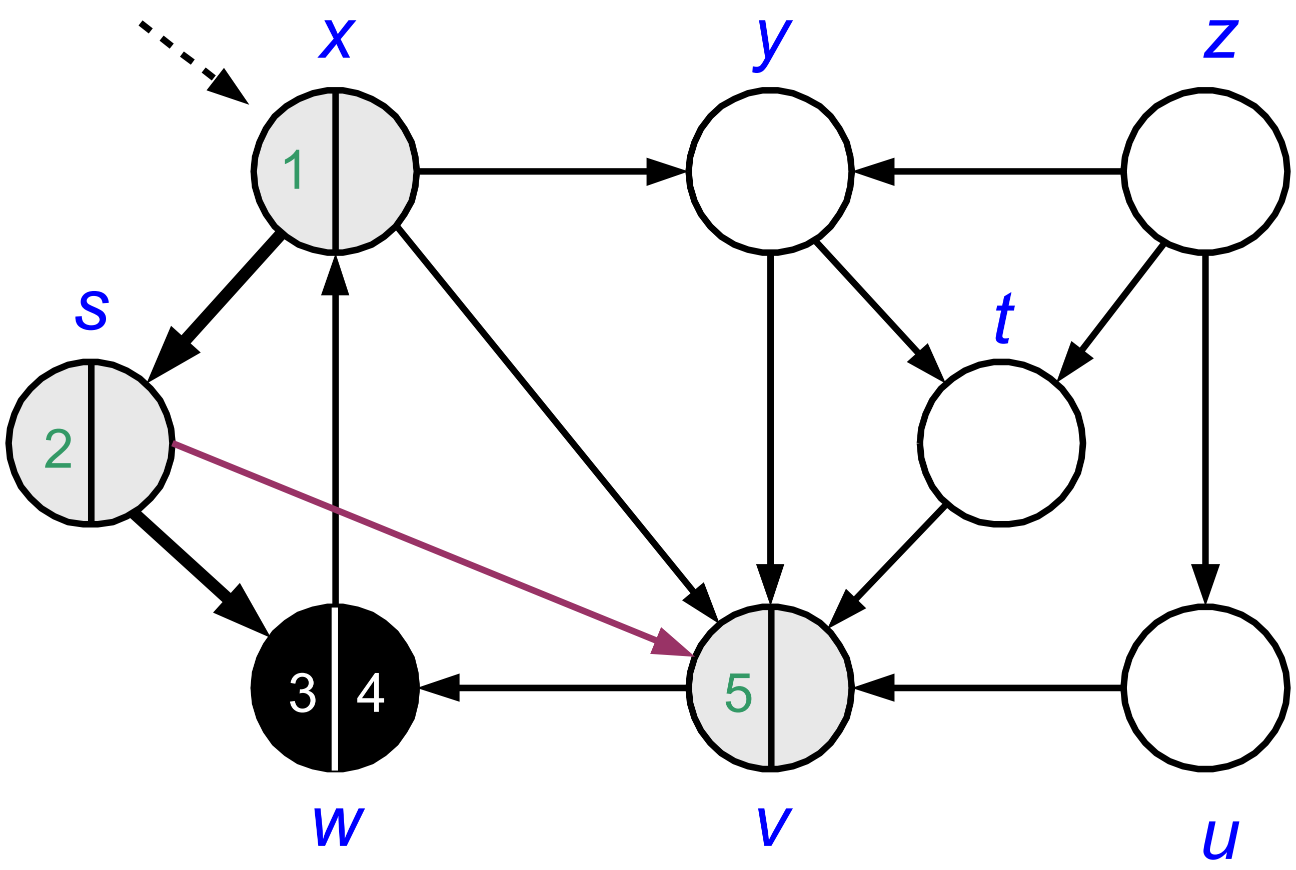
DFS Example



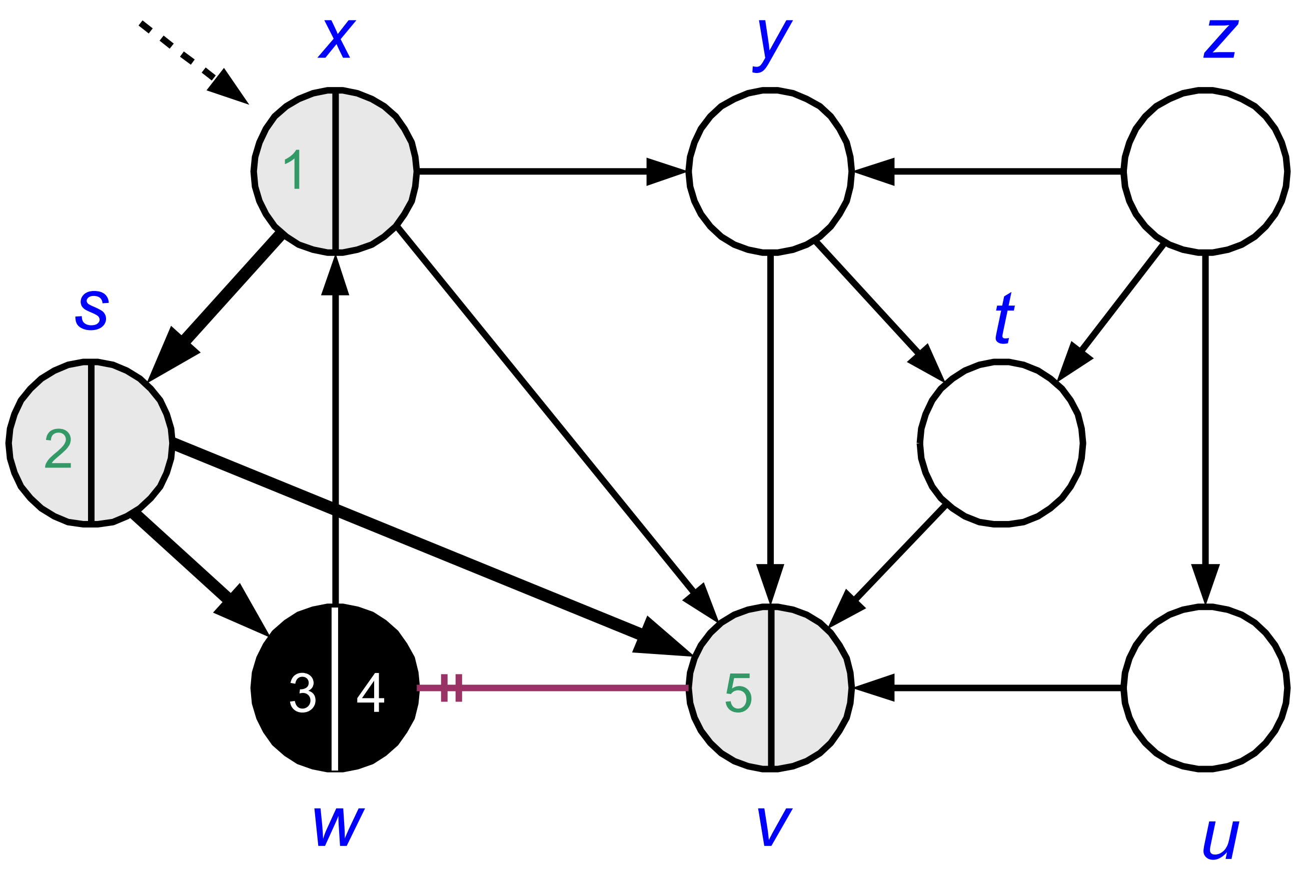
DFS Example



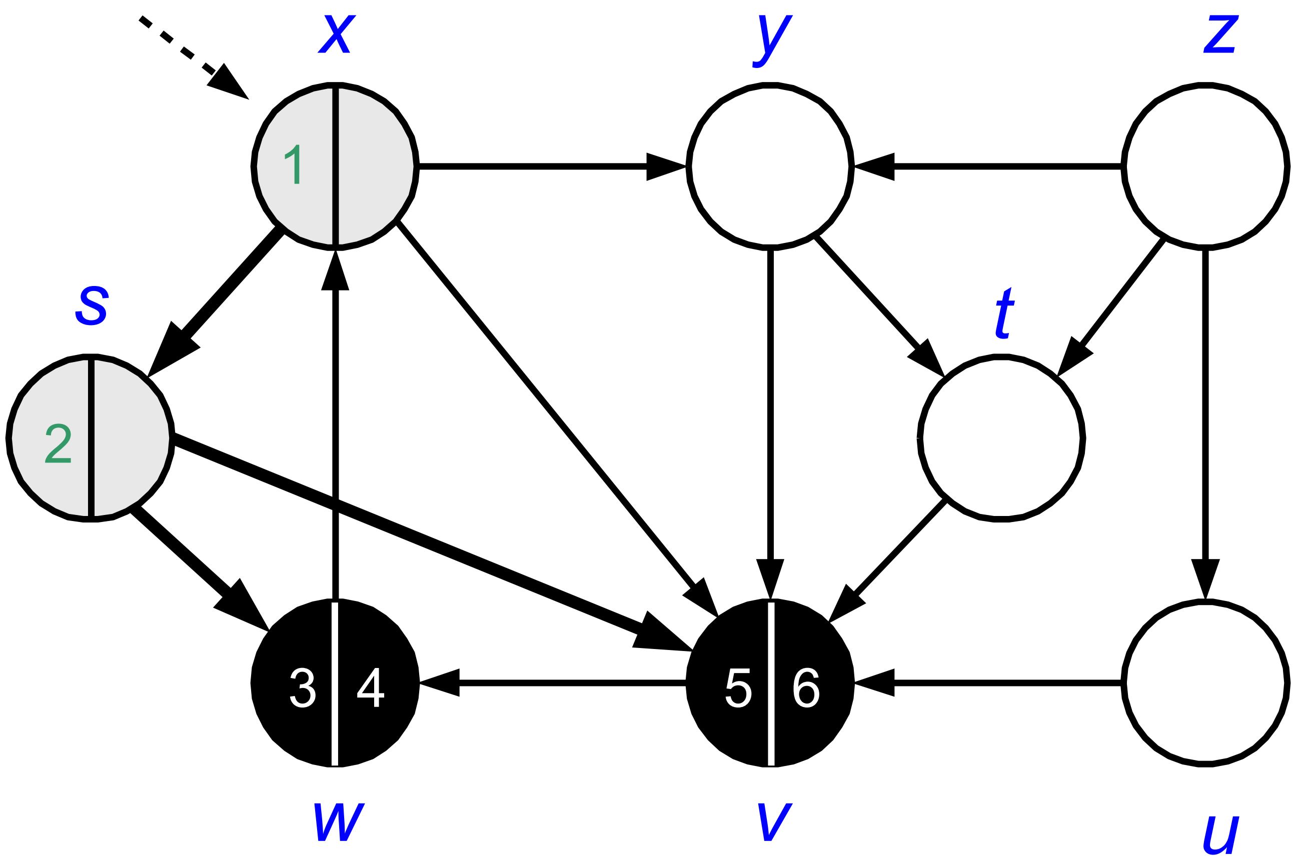
DFS Example



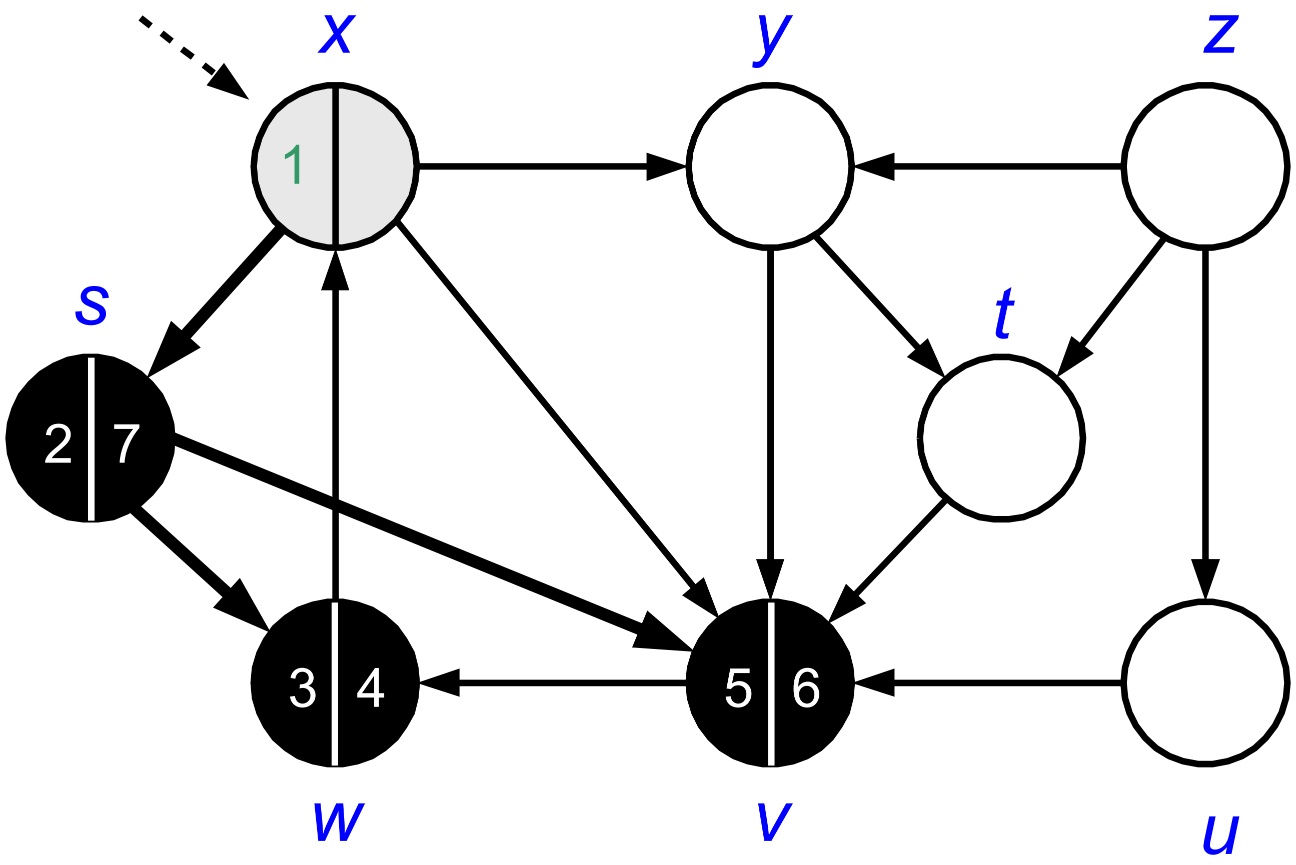
DFS Example



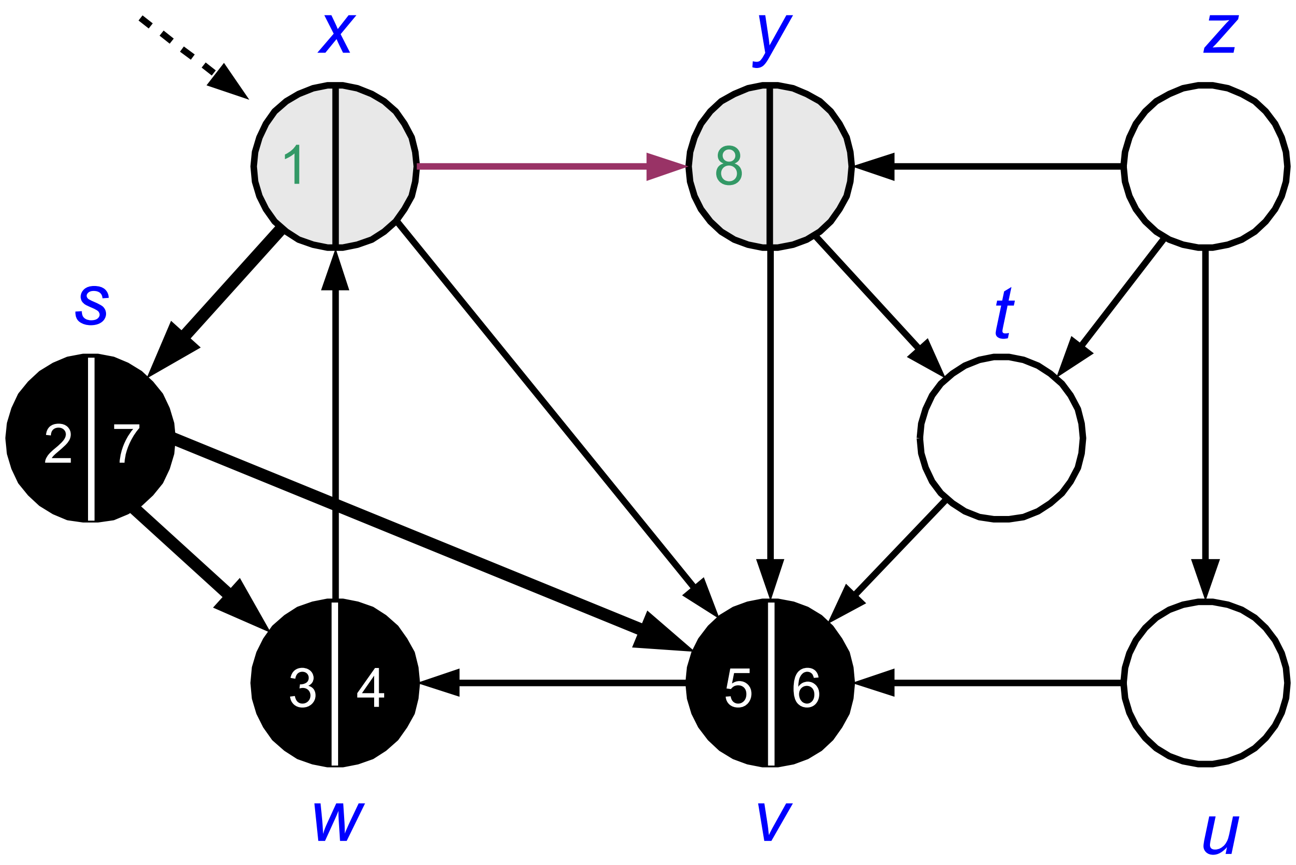
DFS Example



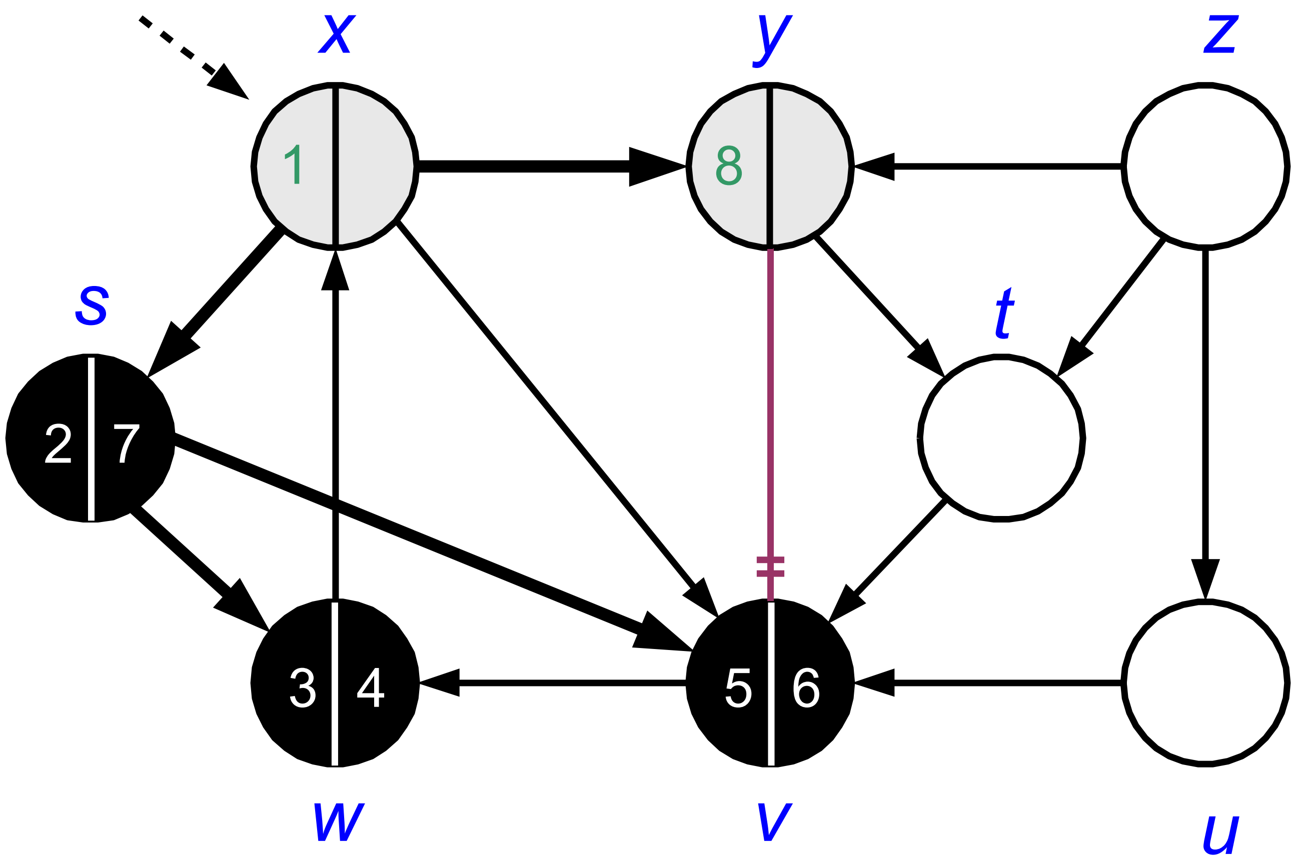
DFS Example



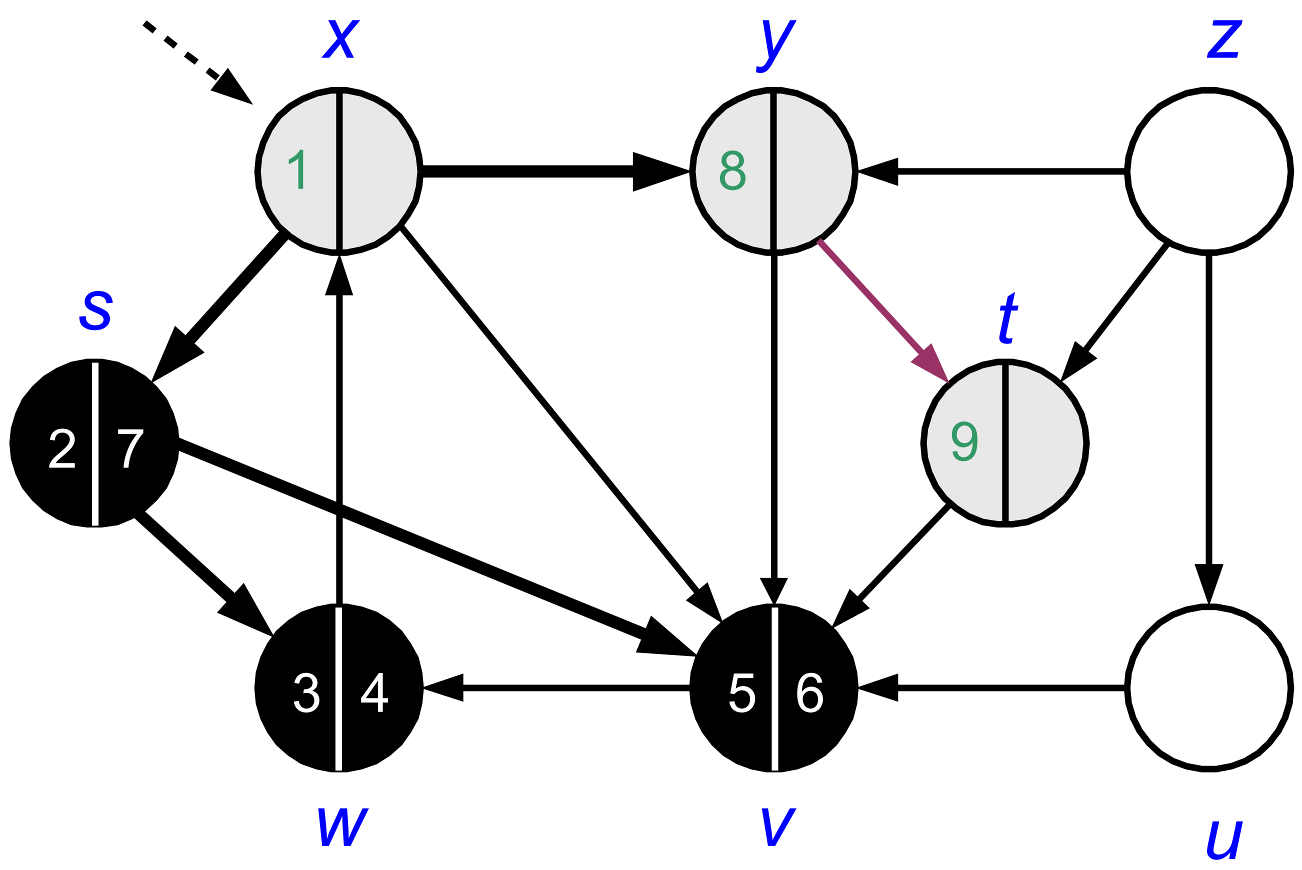
DFS Example



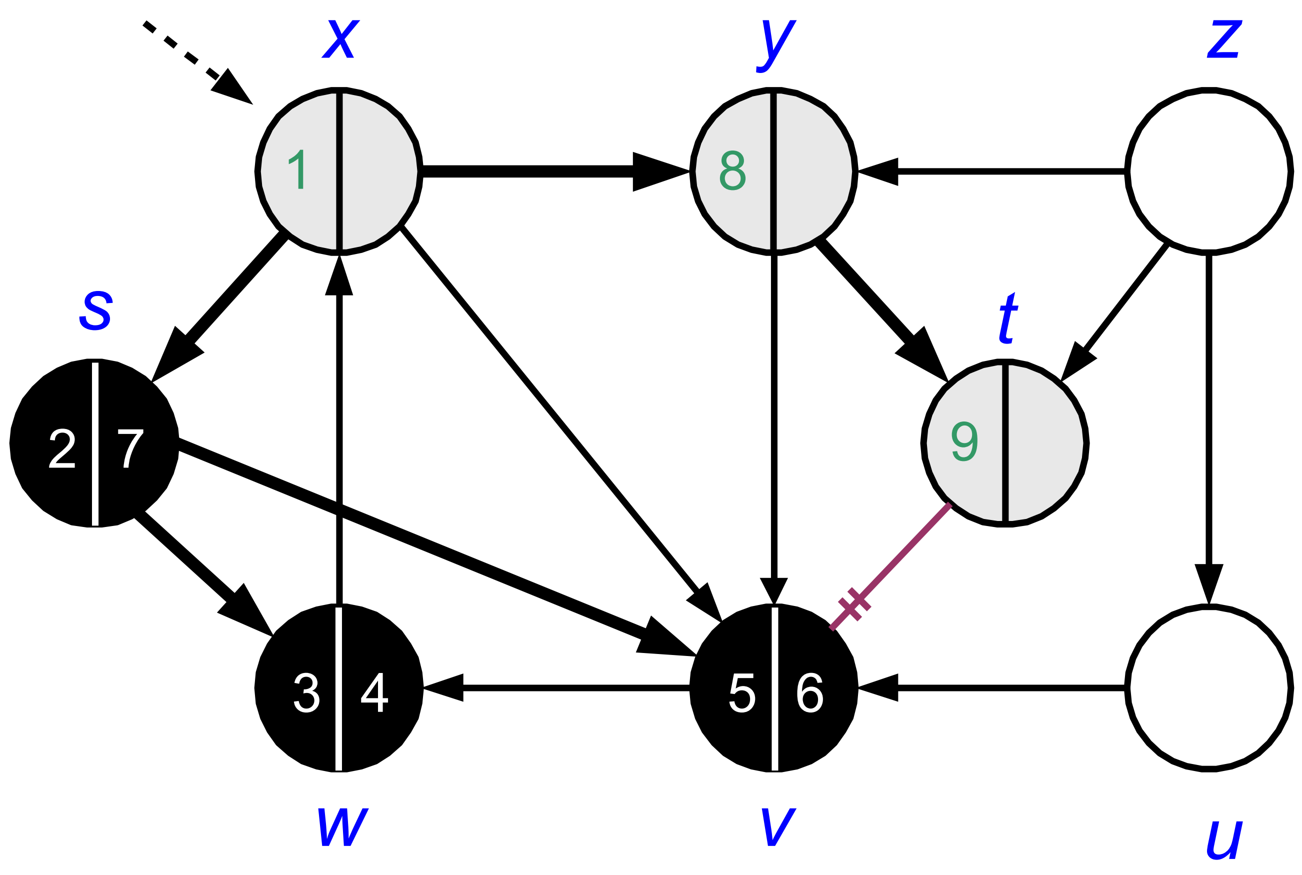
DFS Example



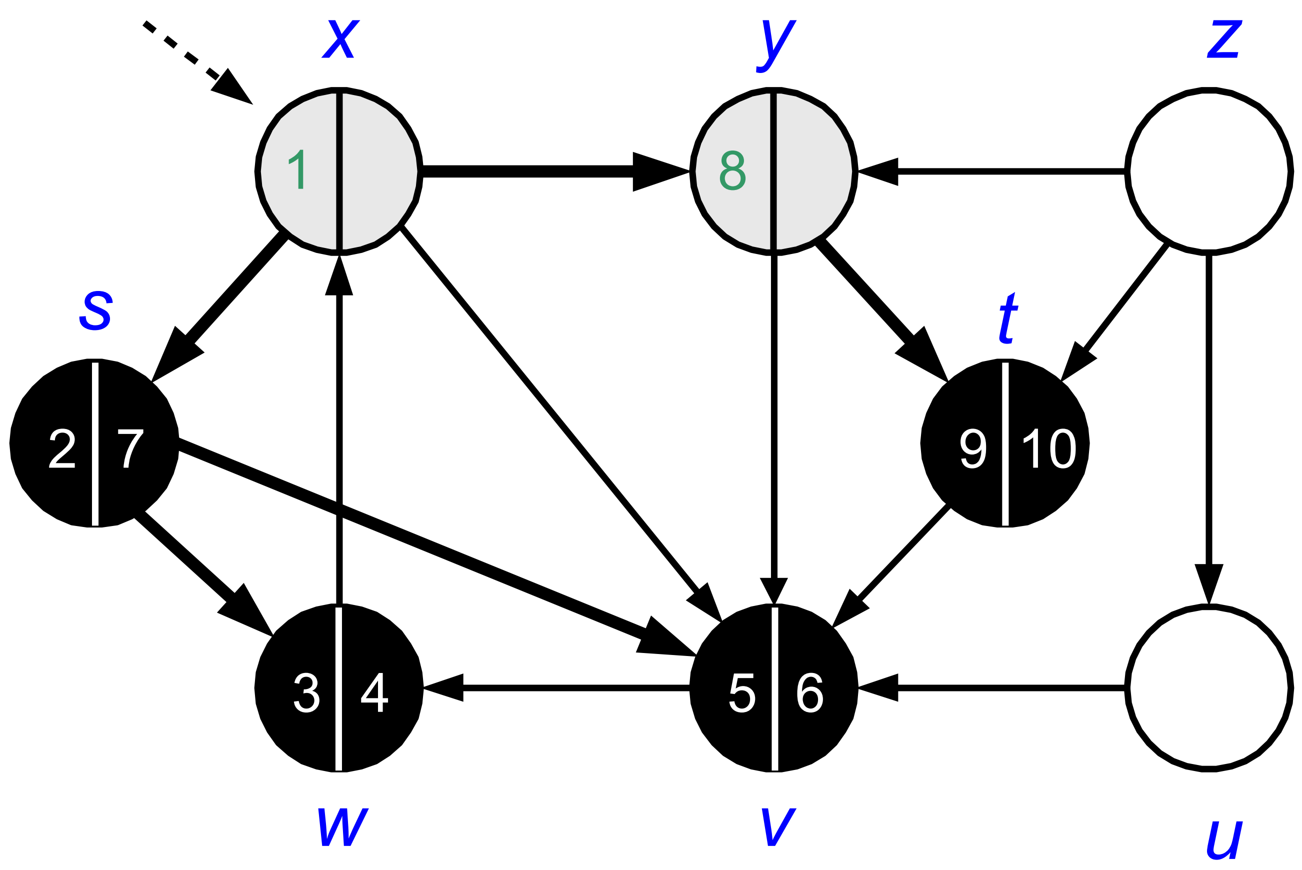
DFS Example



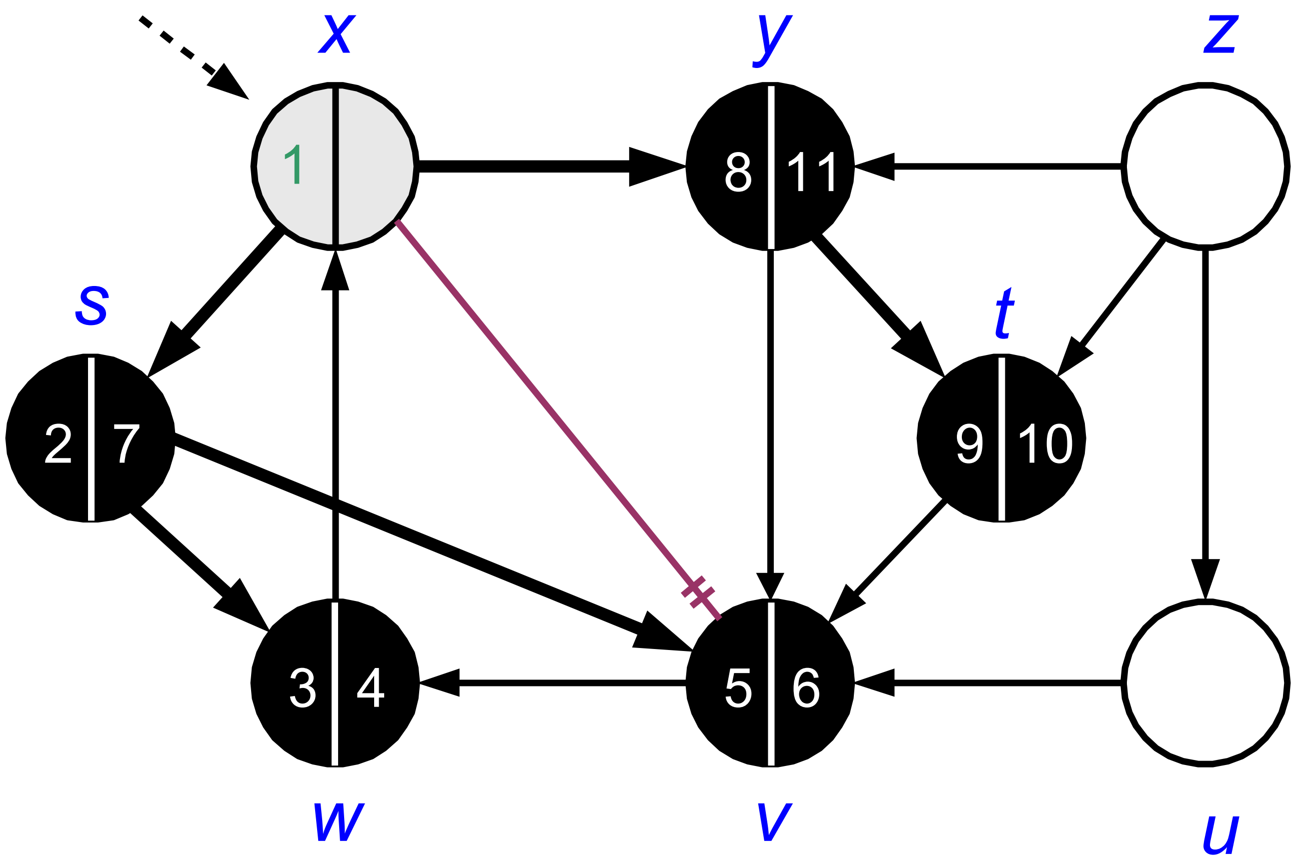
DFS Example



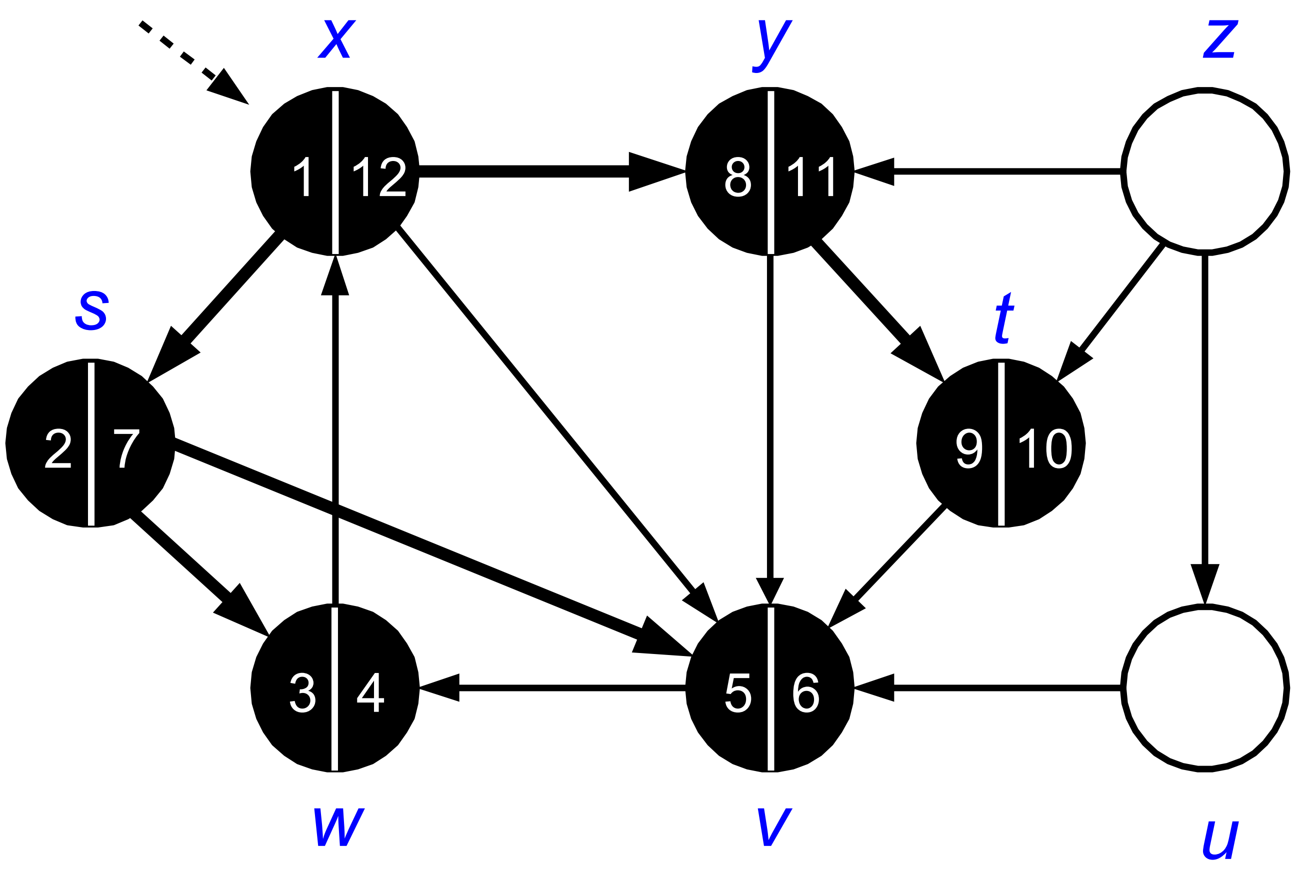
DFS Example



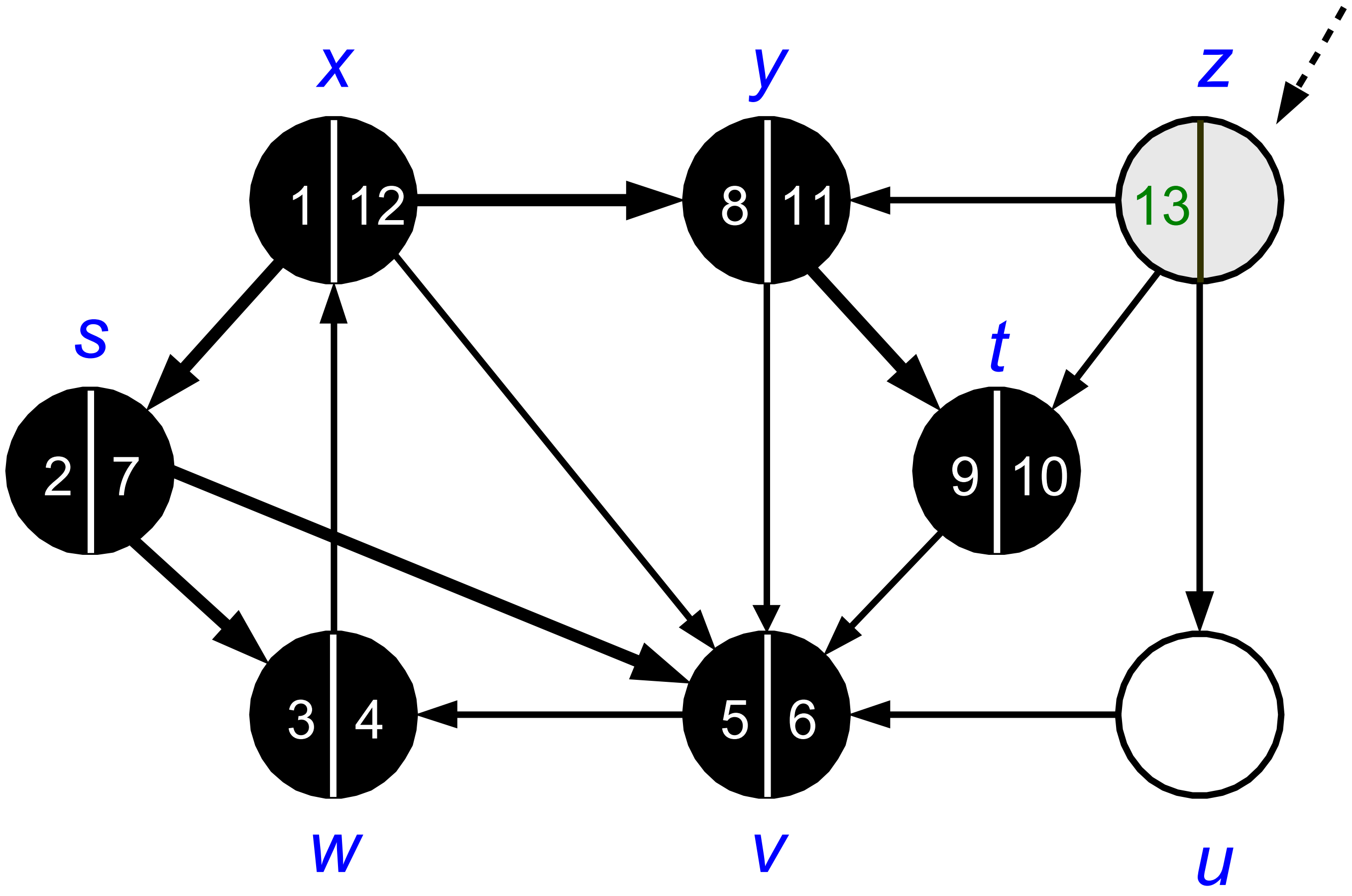
DFS Example



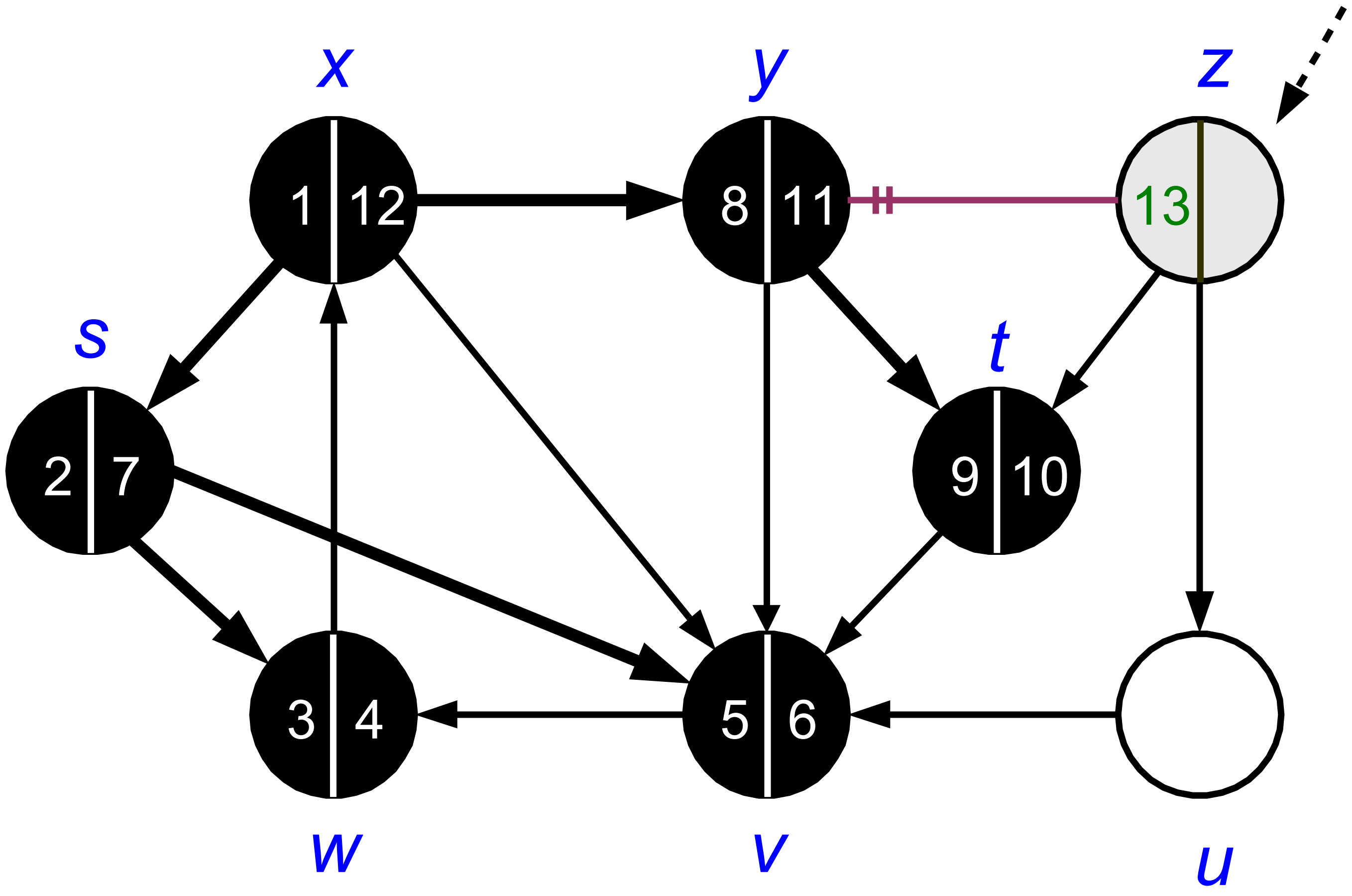
DFS Example



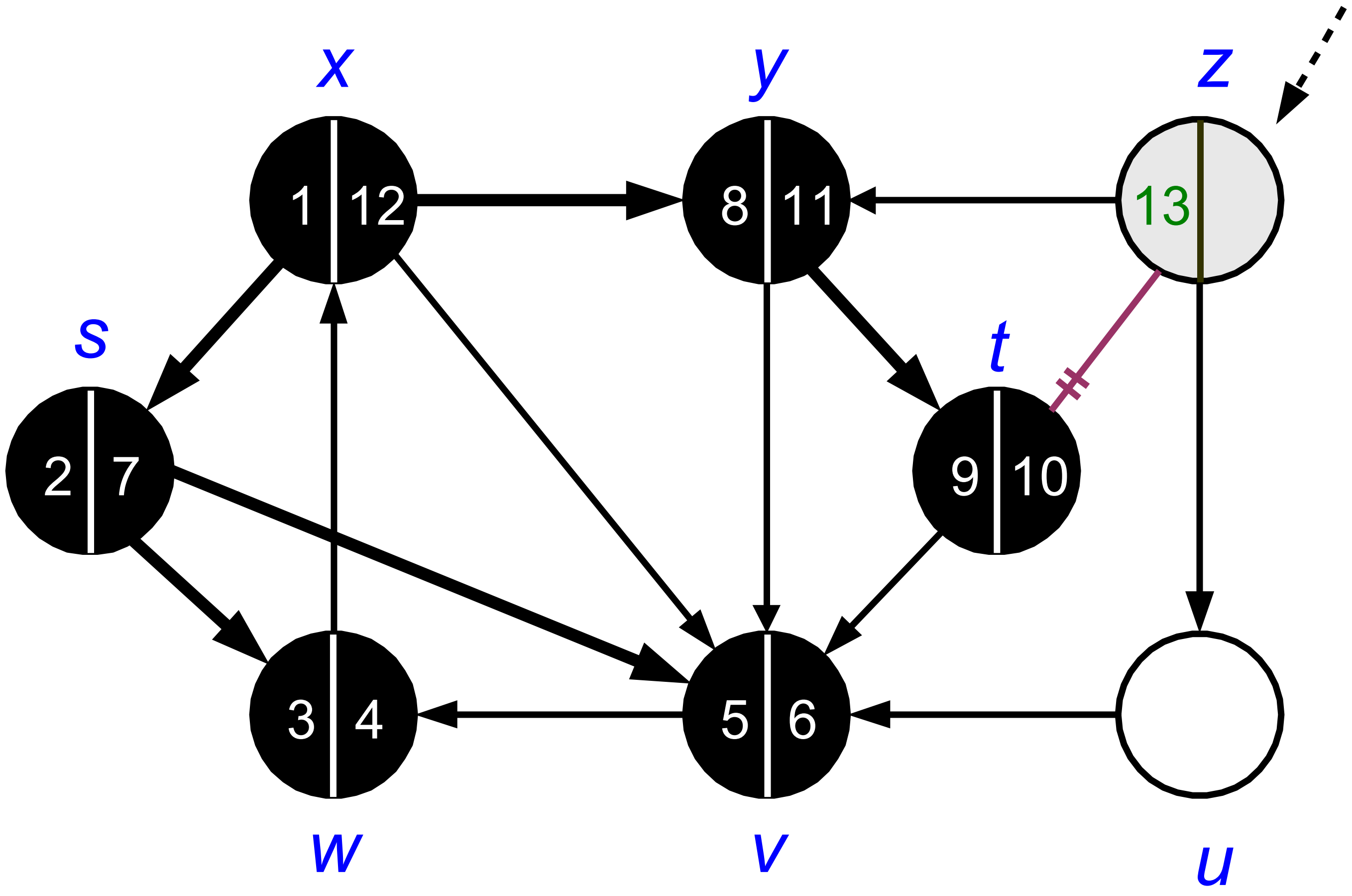
DFS Example



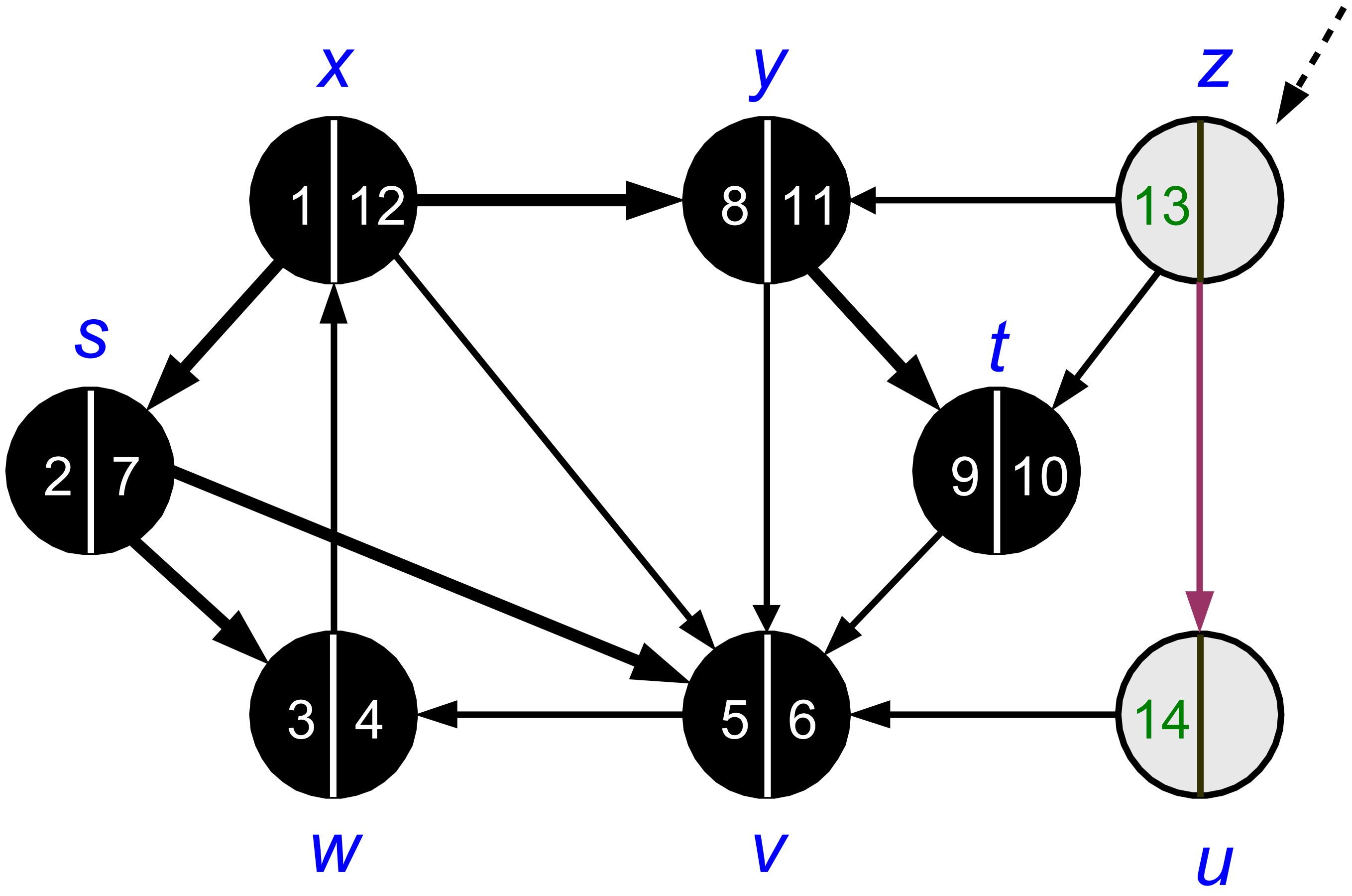
DFS Example



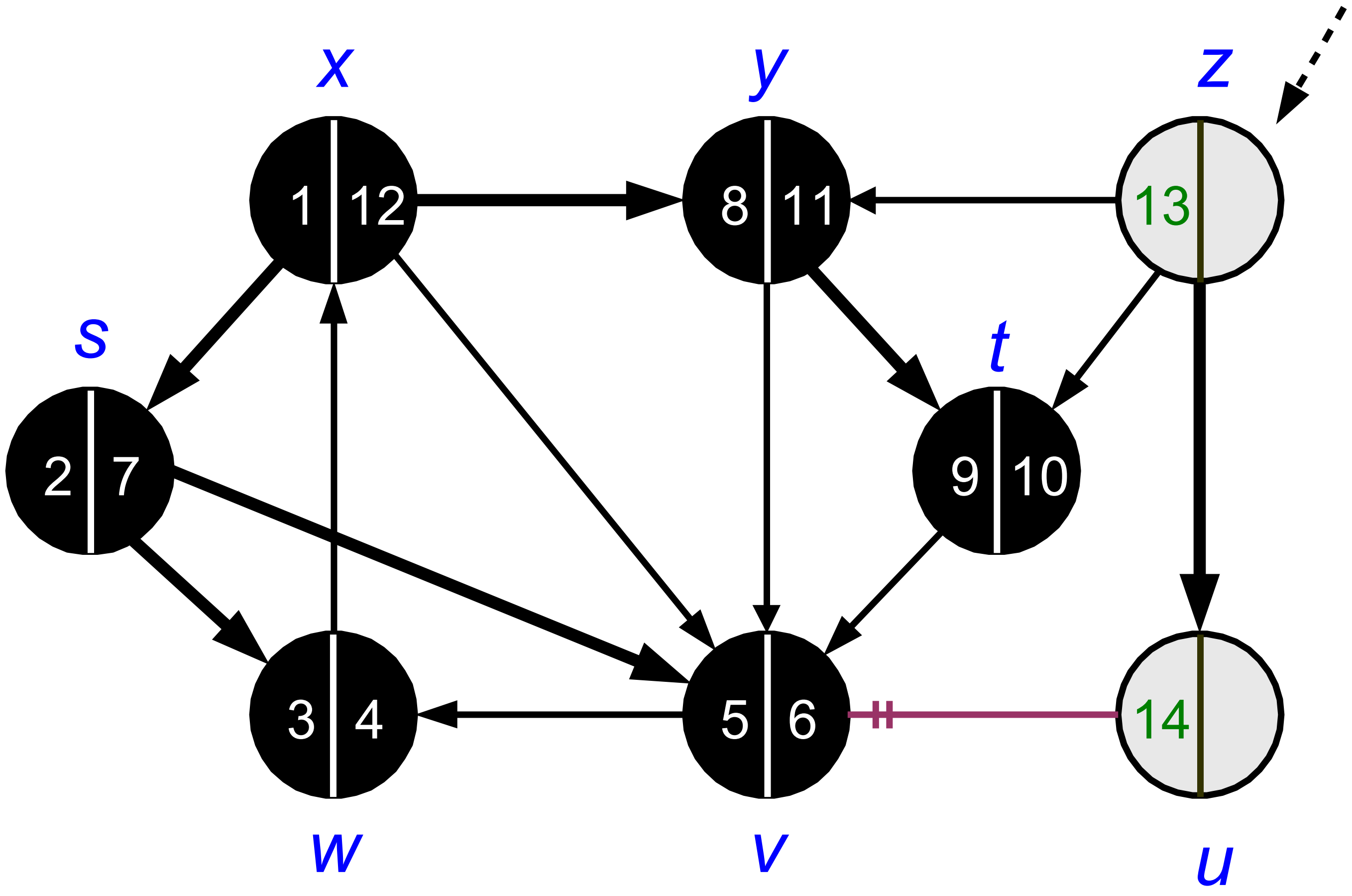
DFS Example



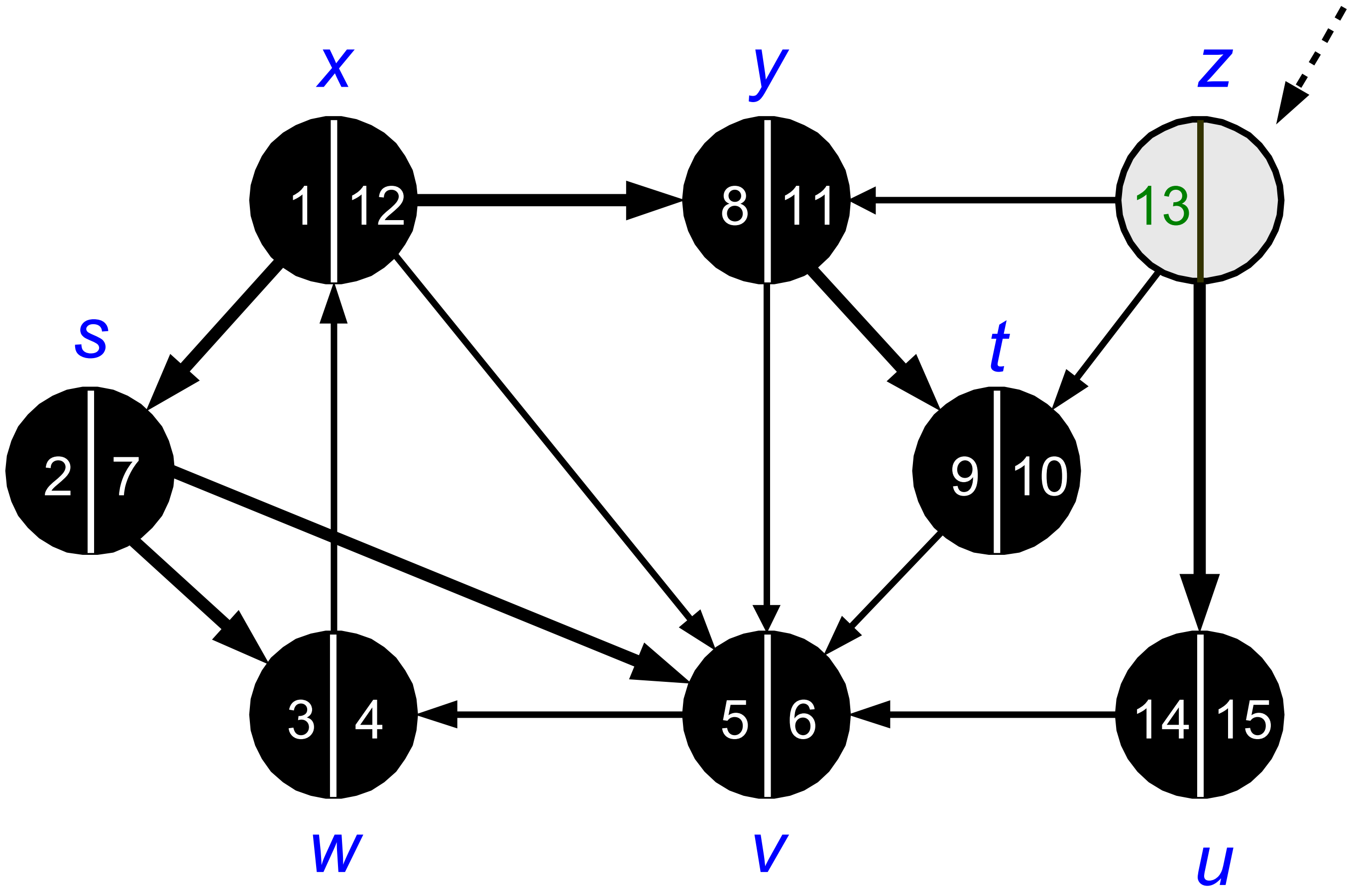
DFS Example



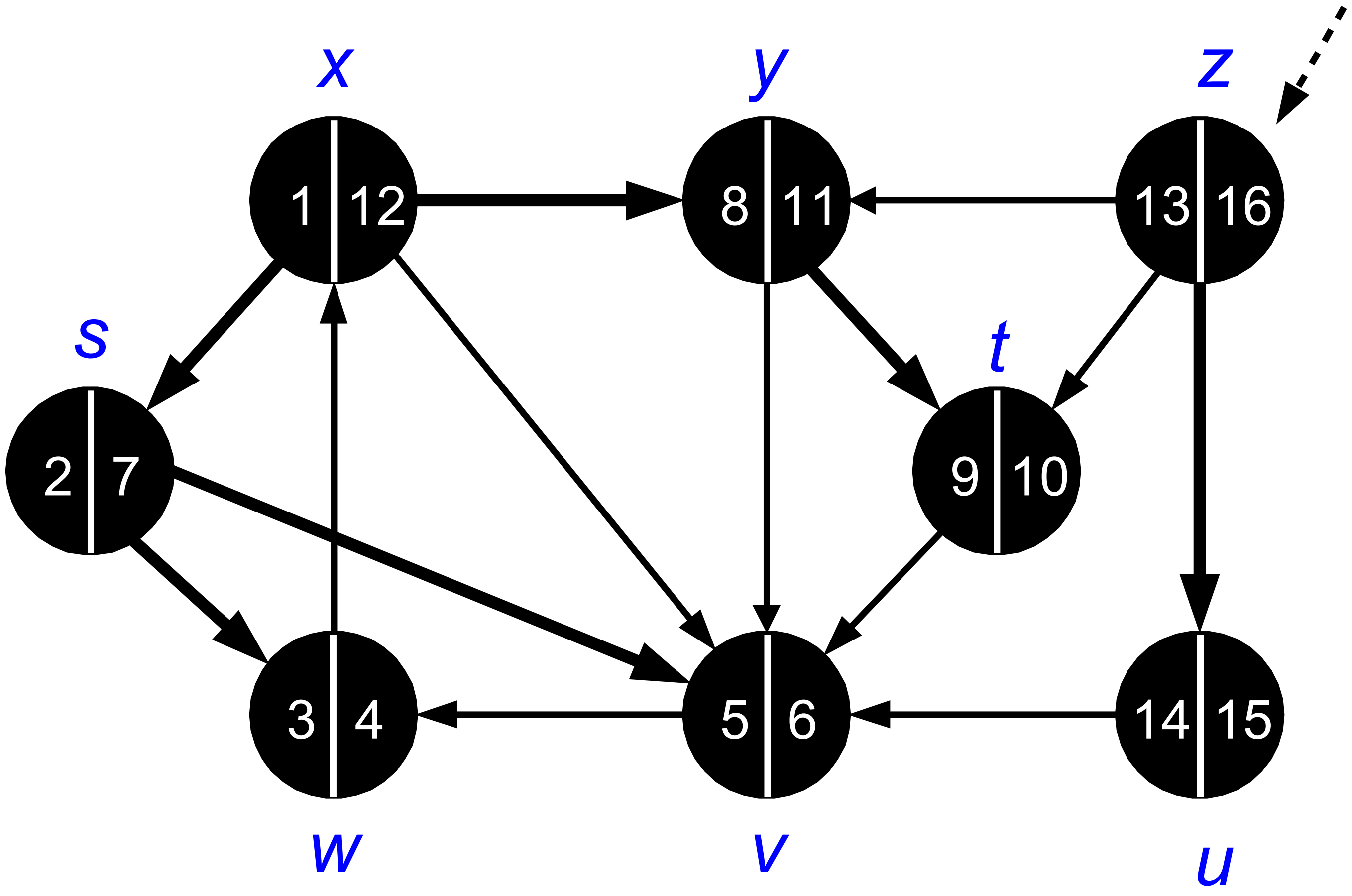
DFS Example



DFS Example

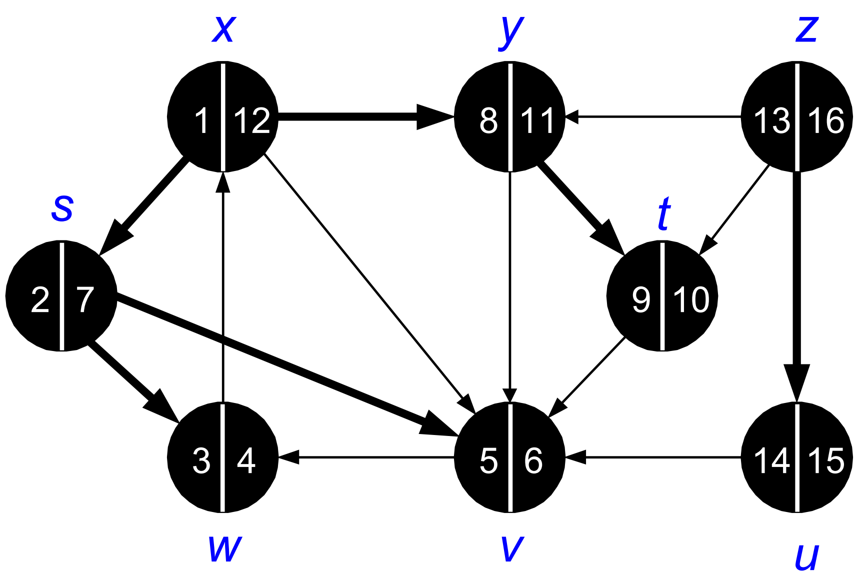


DFS Example

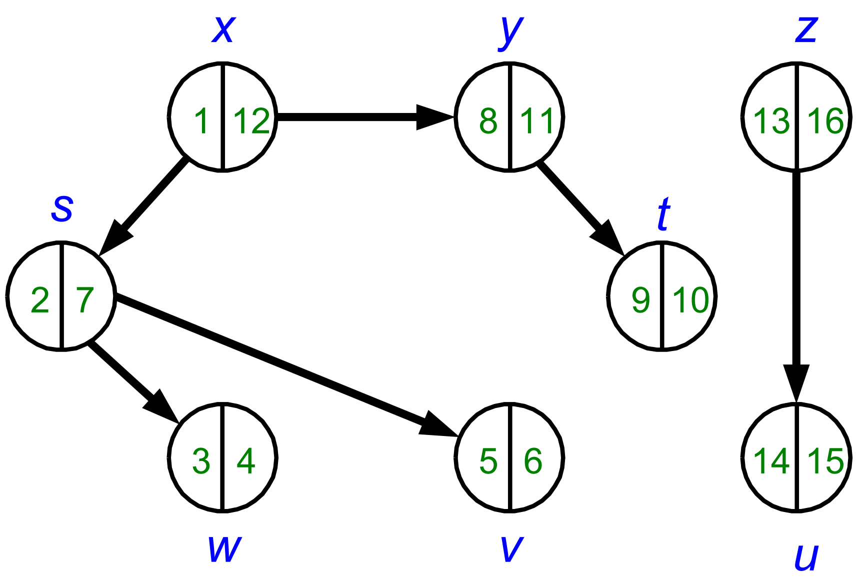


DFS Example

$DFS(G)$ terminated



Depth-first forest (DFF)

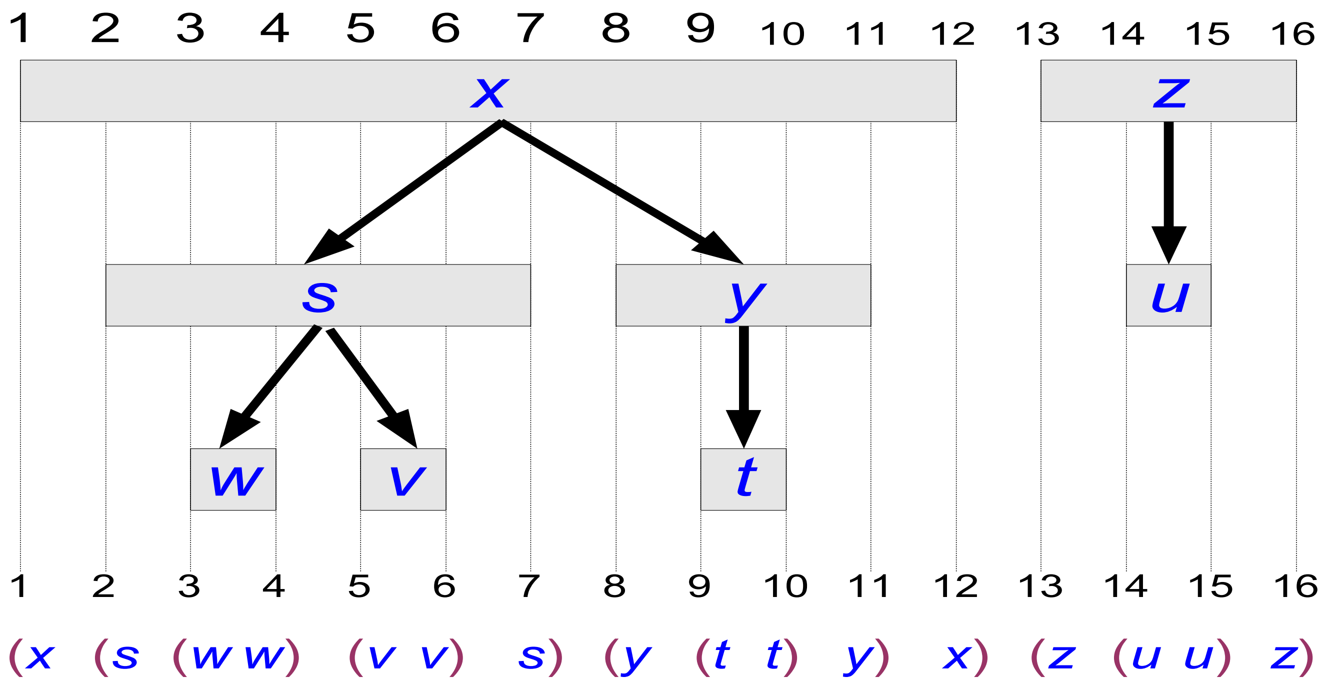


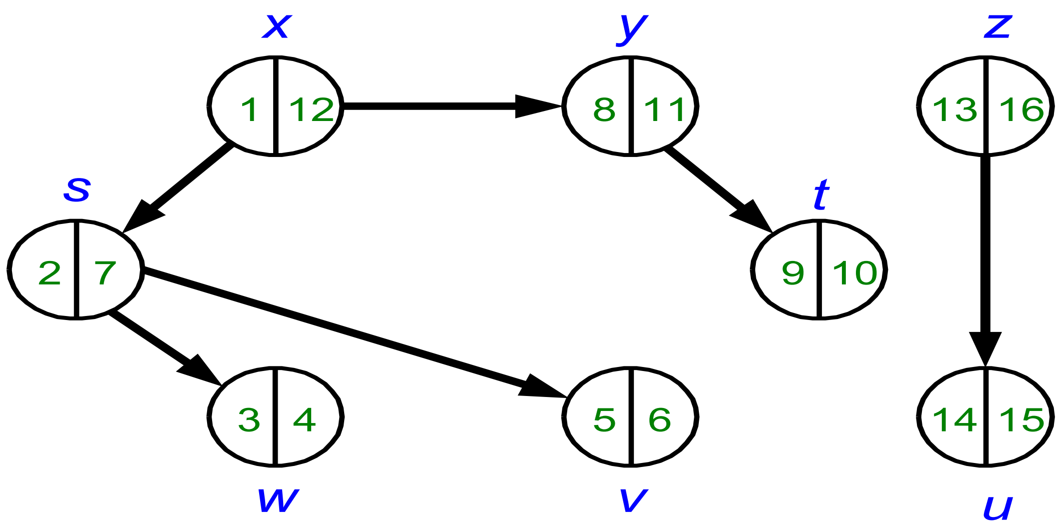
DFS Property: Parenthesis Structure

In any DFS of $\boldsymbol{G}$=($\boldsymbol{V}, \boldsymbol{E}$), let $\boldsymbol{int_v}$= [$\boldsymbol{v.d}, \boldsymbol{v.f}$] then

- $\boldsymbol{int_u}$ and $\boldsymbol{int_v}$ are entirely disjoint
- $\boldsymbol{int_v}$ is entirely contained in $\boldsymbol{int_u}$ and v is a descendant of u in a DFT
- $\boldsymbol{int_u}$ is entirely contained in $\boldsymbol{int_v}$ and u is a descendant of v in a DFT

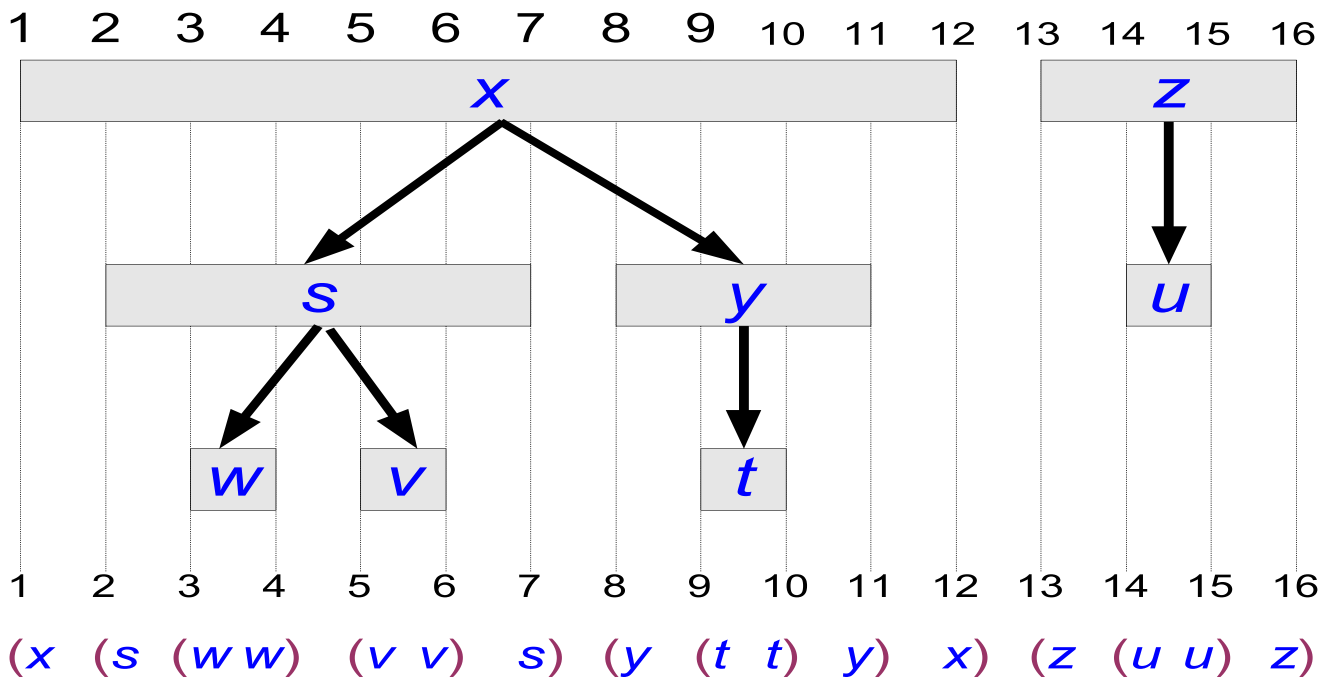
DFS Property: Parenthesis Structure

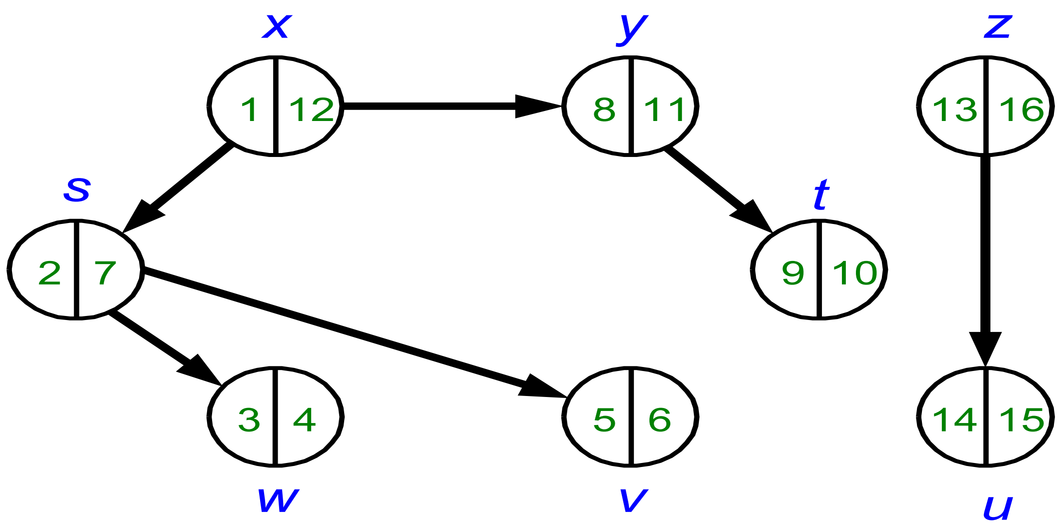




(( )) This overlapping never happens!

DFS Property: Parenthesis Structure

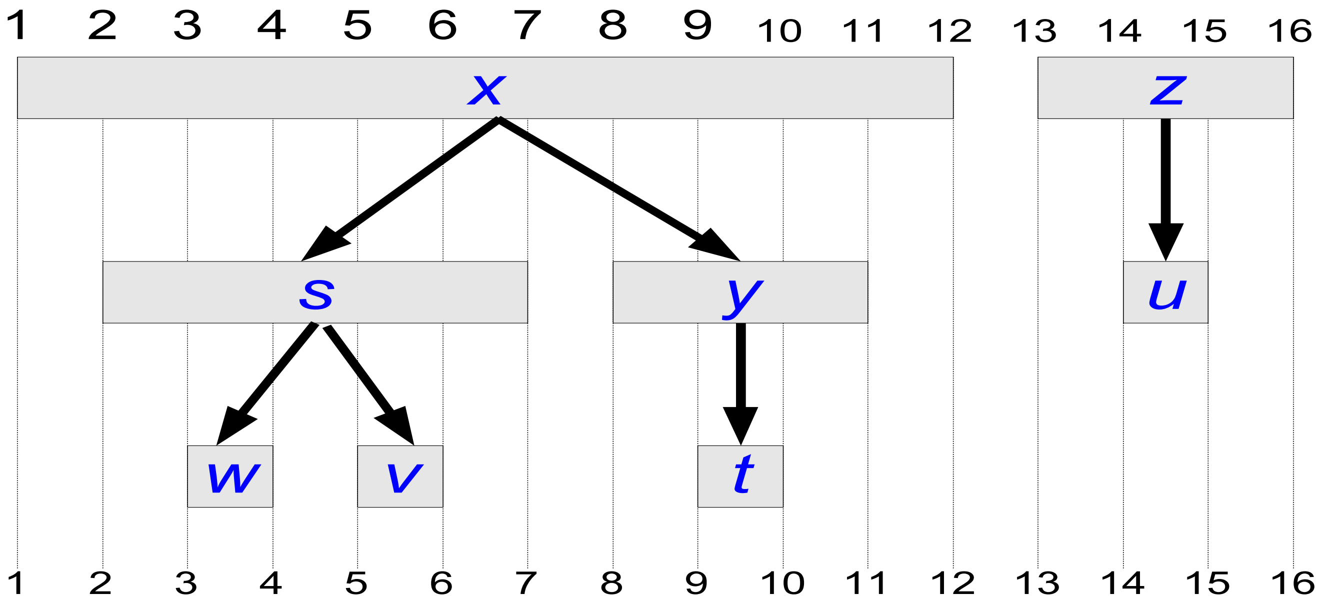


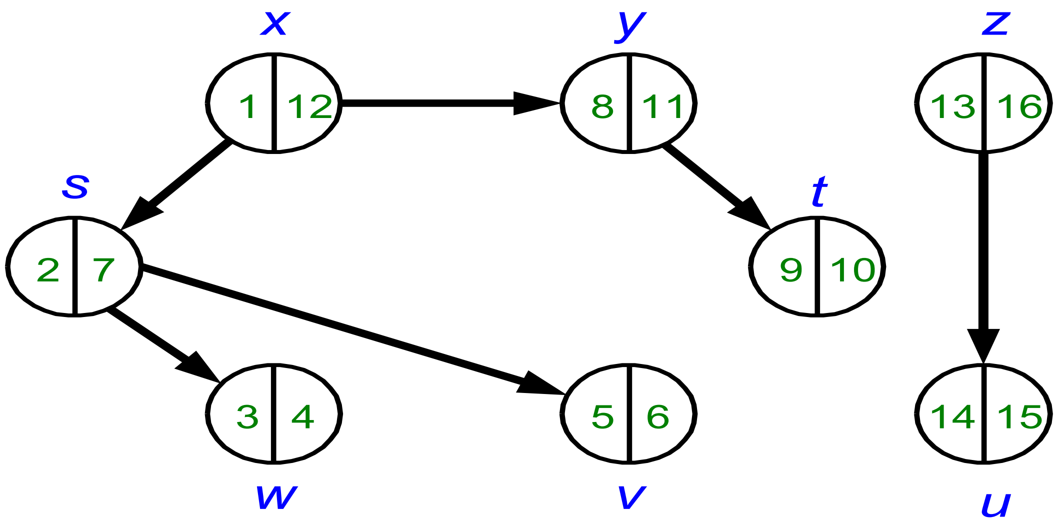


Question: Given a graph and two vertices x, v how would you decide if x is a descendant of v?

If $\boldsymbol{int_x}$ is contained $\boldsymbol{int_v}$

Parenthesis Theorem– Example





(*x* (*s* (*w* *w*) (*v* *v*) *s*) (*y* (*t* *t*) *y*) *x*) (*z* (*u* *u*) *z*)

(( )) This overlapping never happens!

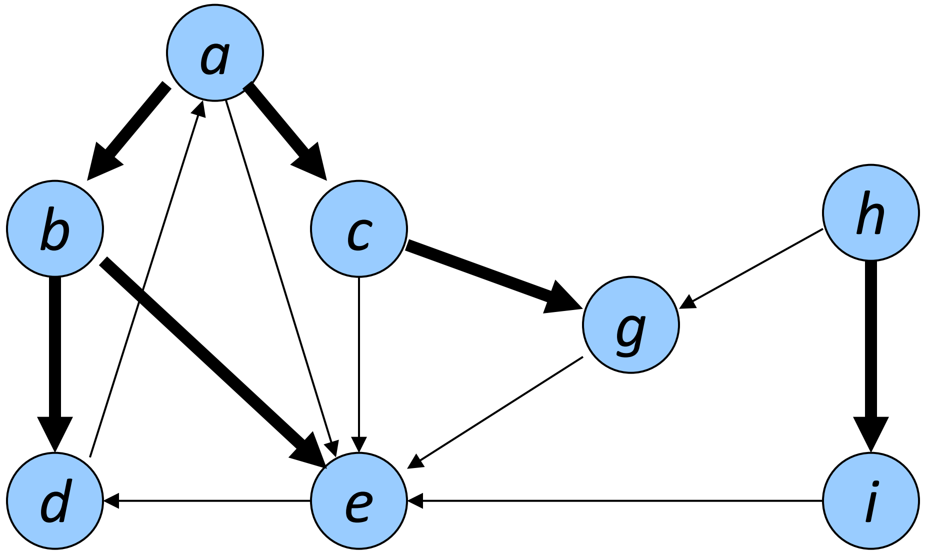
Edge Classification in a DFF

Tree Edge: discover a new (WHITE) vertex ▹GRAY to WHITE◃

Back Edge: from a descendent to an ancestor in DFF ▹GRAY to GRAY◃

Forward Edge: from ancestor to descendent in DFF ▹GRAY to BLACK◃

Cross Edge: remaining edges (between trees and subtrees) ▹GRAY to BLACK◃



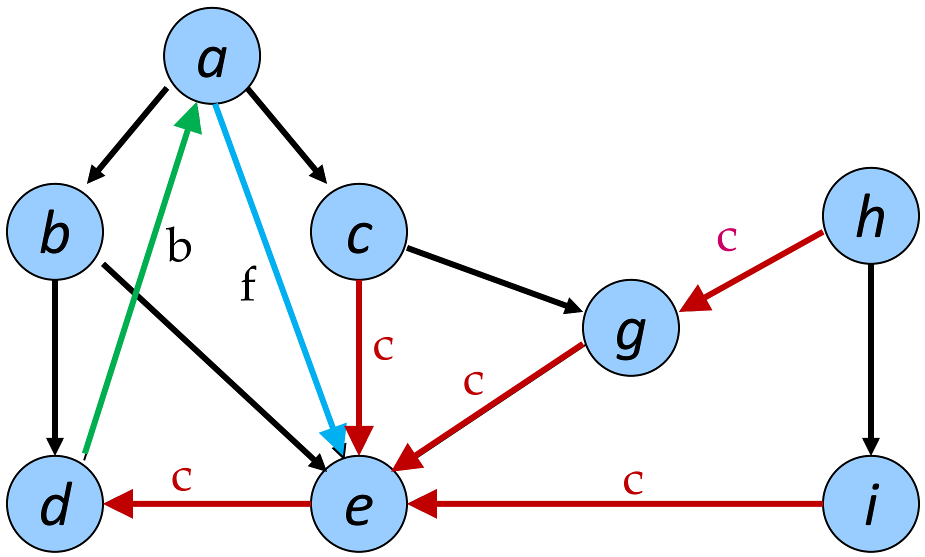
Edge Classification in a DFF

Tree Edge: discover a new (WHITE) vertex ▷ GRAY to WHITE ◁

Back Edge: from a descendent to an ancestor in DFF ▷ GRAY to GRAY ◁

Forward Edge: from ancestor to descendent in DFF ▷ GRAY to BLACK ◁

Cross Edge: remaining edges (between trees and subtrees) ▷ GRAY to BLACK ◁

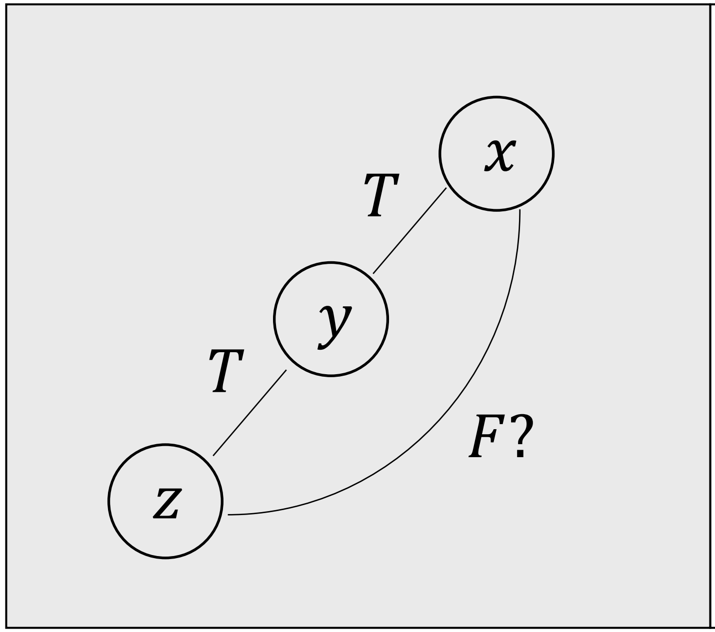


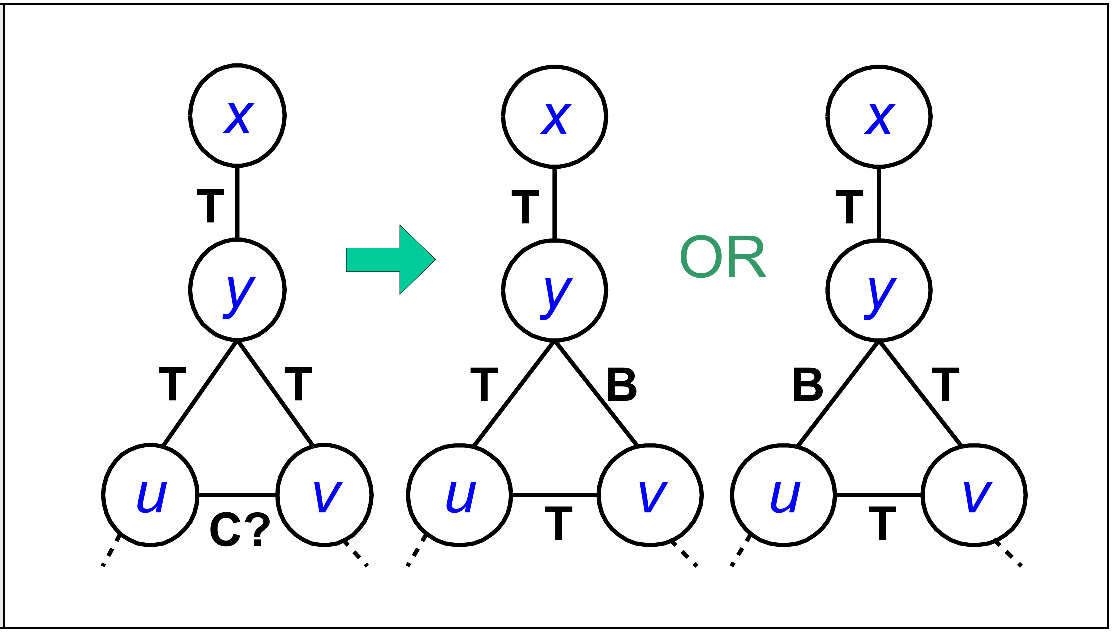
Edge Classification in a DFF

	Directed Graph	Undirected Graph
DFS	all	
BFS		

Edge Classification in a DFF

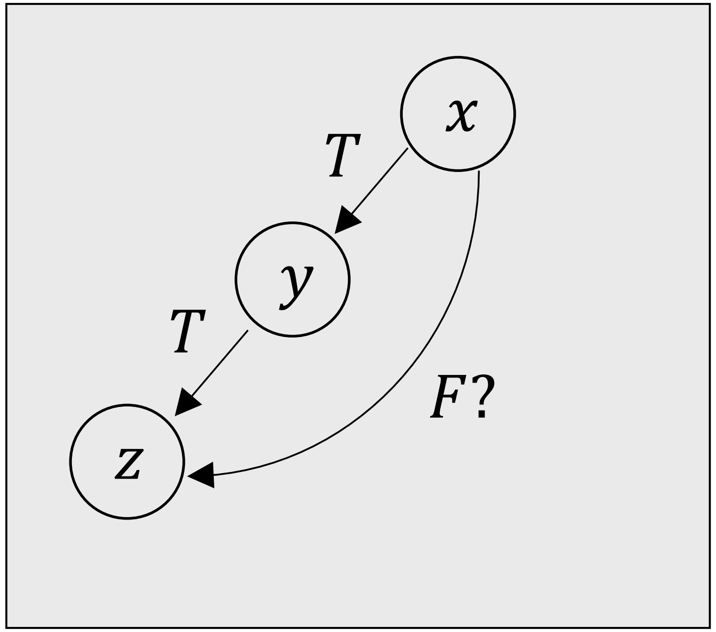
any DFS on an undirected graph produces only Tree and Back edges

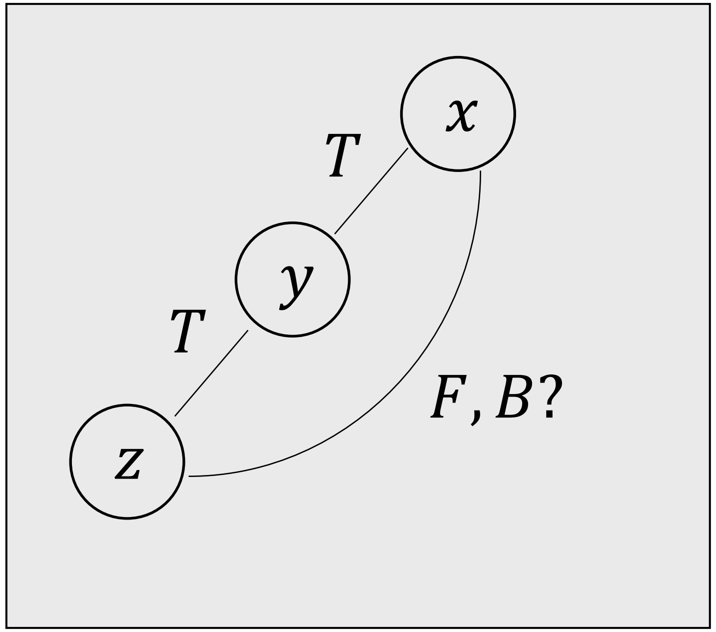




Edge Classification in a DFF

- any BFS on a **directed** graph produces only Tree, back and cross edges
- any BFS on an **undirected** graph produces only Tree and cross edges.

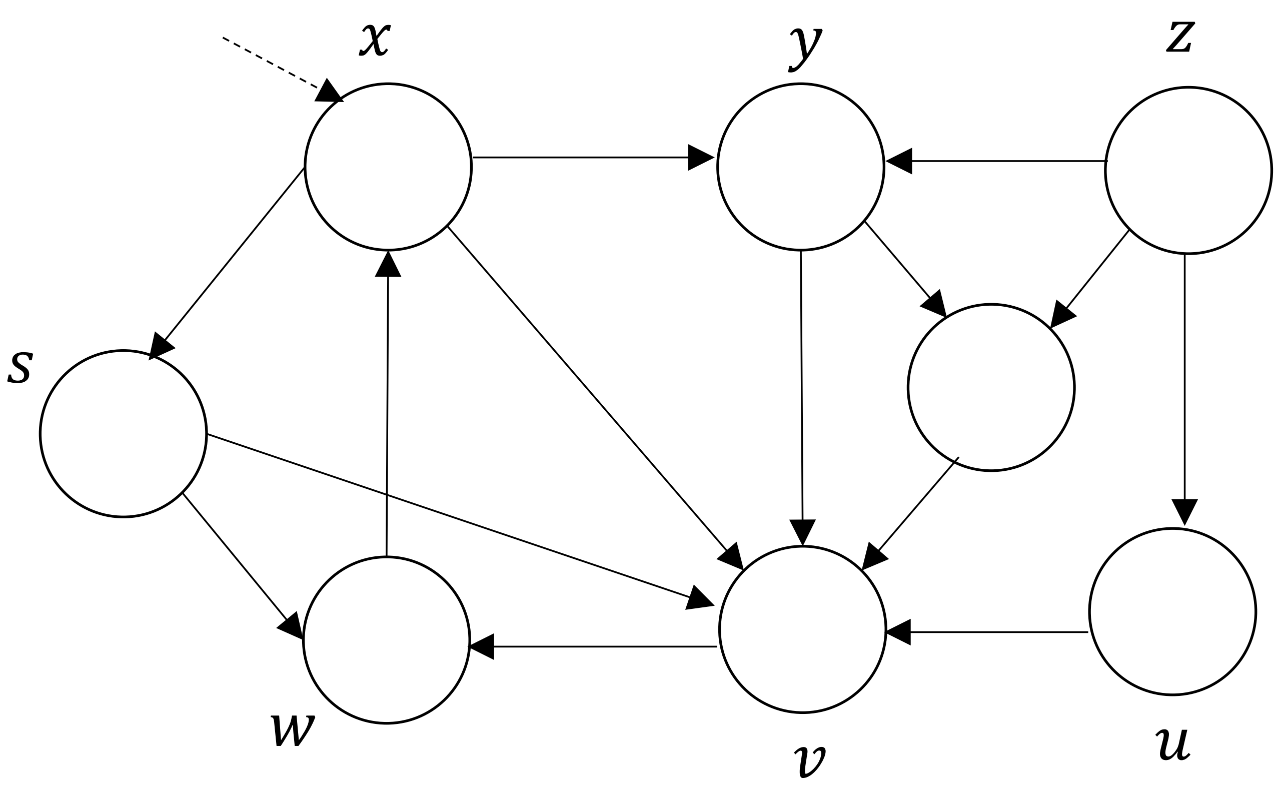




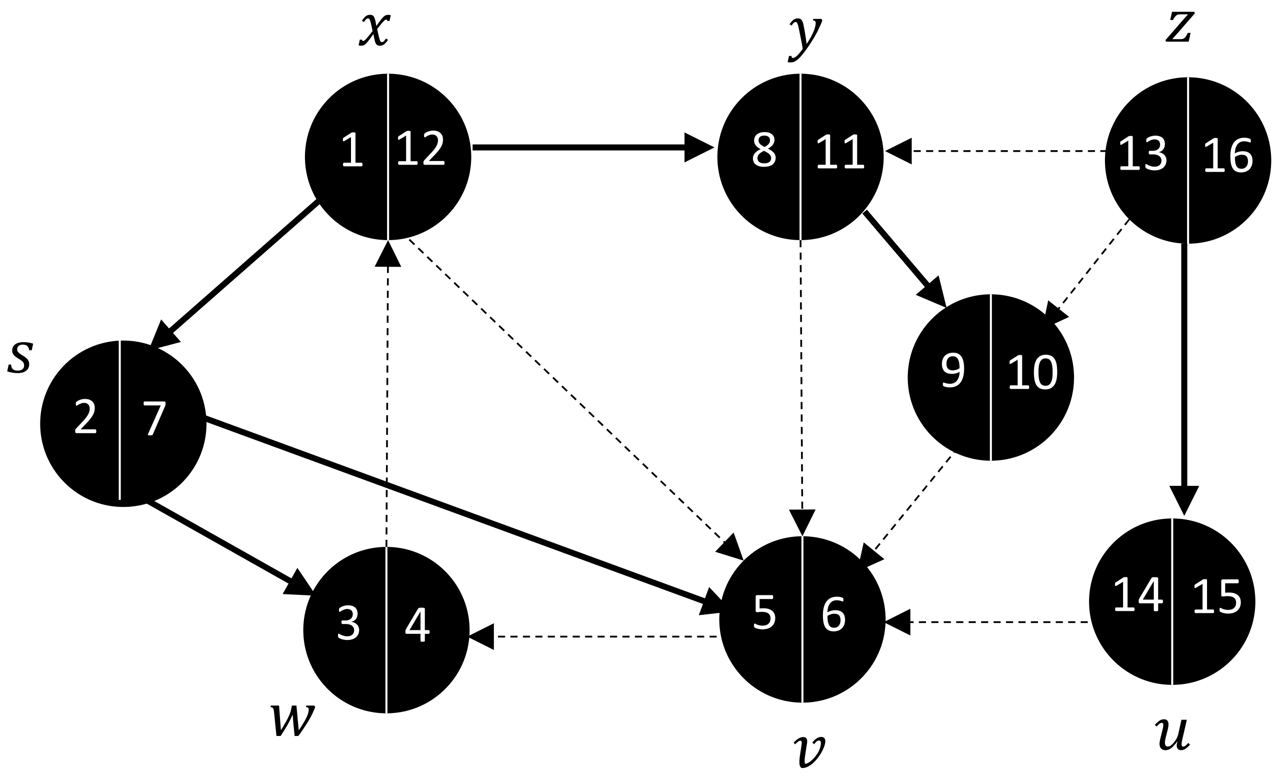
Edge Classification in a DFF

	Directed Graph	Undirected Graph
DFS	all	tree , back
BFS	tree, back, cross	tree, cross

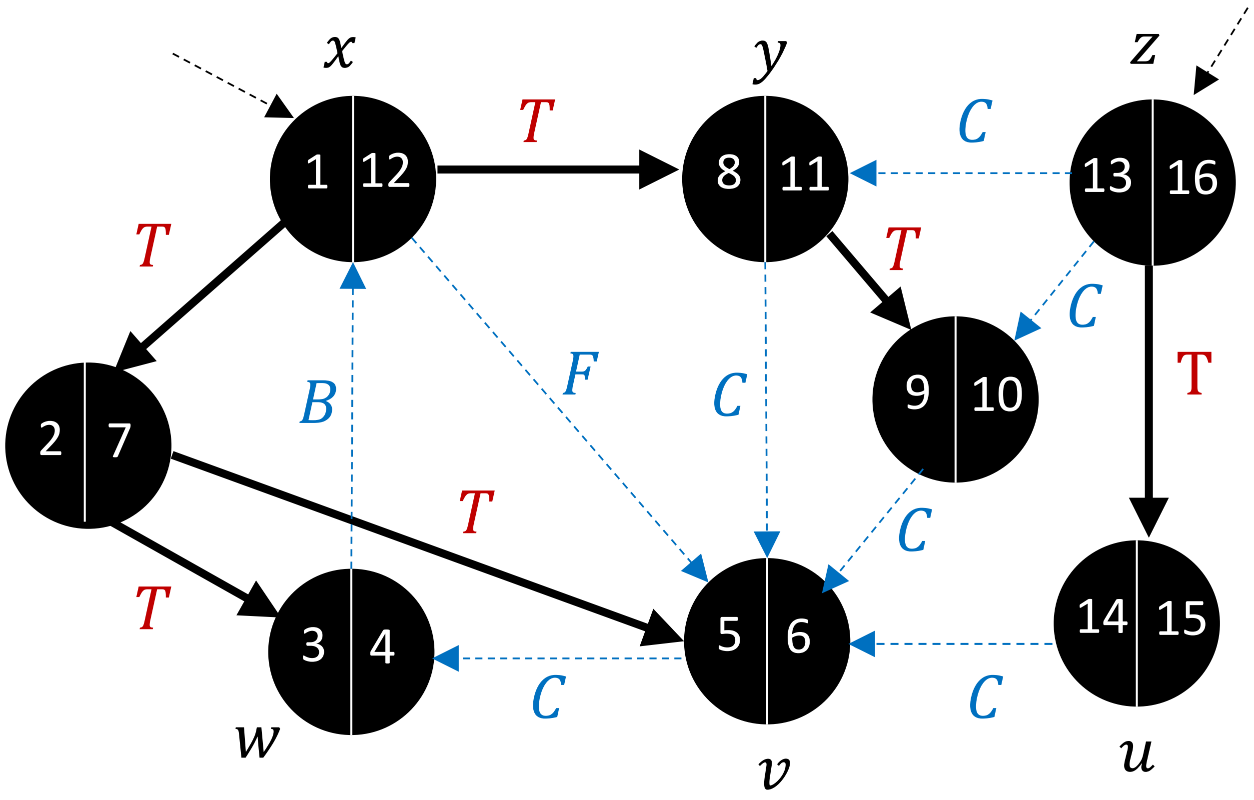
Edge Classification - Example



Edge Classification - Example



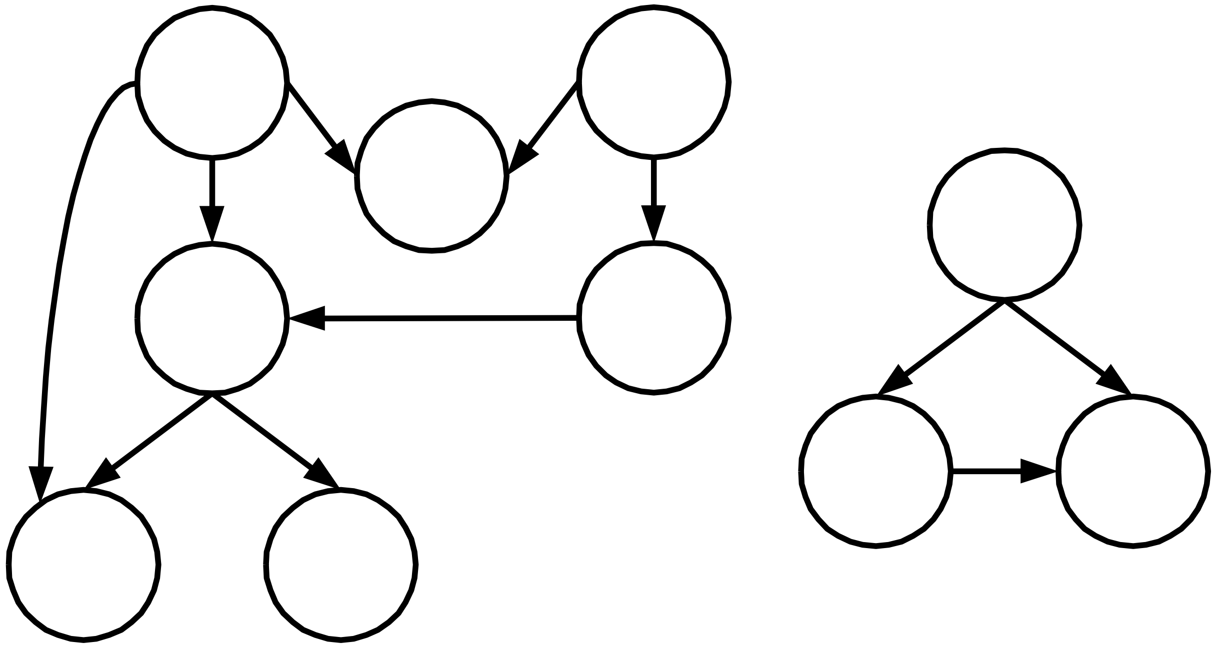
Edge Classification - Example



Directed Acyclic Graphs (DAG)

No directed cycles

Example:



Directed Acyclic Graphs (DAG)

Theorem: a directed graph G is acyclic iff DFS on G yields no Back edges

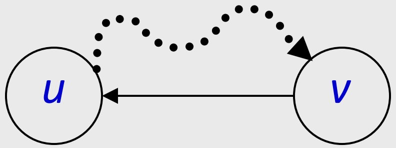
Proof (acyclic $\Rightarrow$ no Back edges; by contradiction):

Let (v, u) be a Back edge visited during scanning $Adj[v]$

$\Rightarrow v.color = u.color =$ Gray and $u.d < v.d$

v is descendent of $u \Rightarrow \exists$ a path from u to v in a DFT and hence in G

$\therefore$ edge (v, u) will create a cycle (Back edge $\Rightarrow$ cycle)



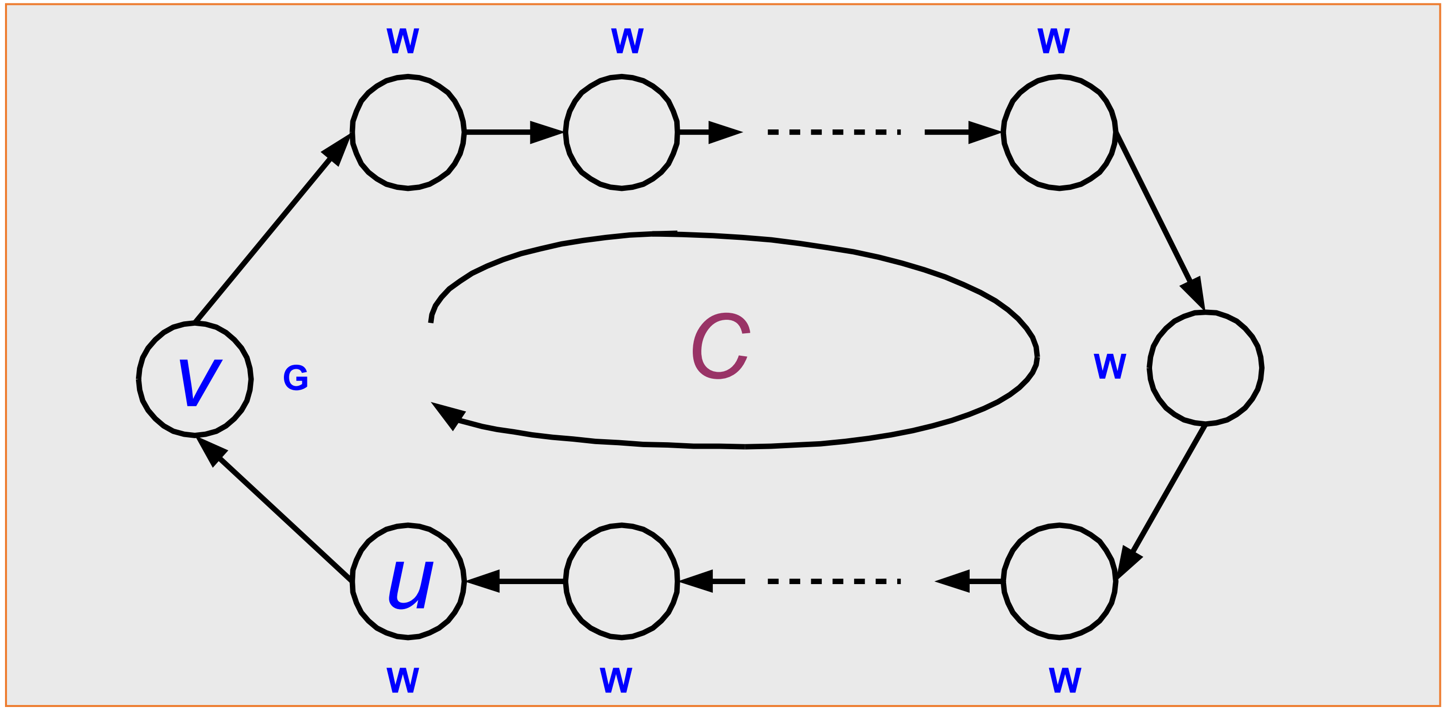
path from u to v in a DFT and hence in G

Proof (no Back edges $\Rightarrow$ acyclic):

Suppose G contains a cycle C

(Show that a DFS on G yields a Back edge; proof by contradiction)

Let v be the first vertex discovered in C and let (u, v) be proceeding edge in C



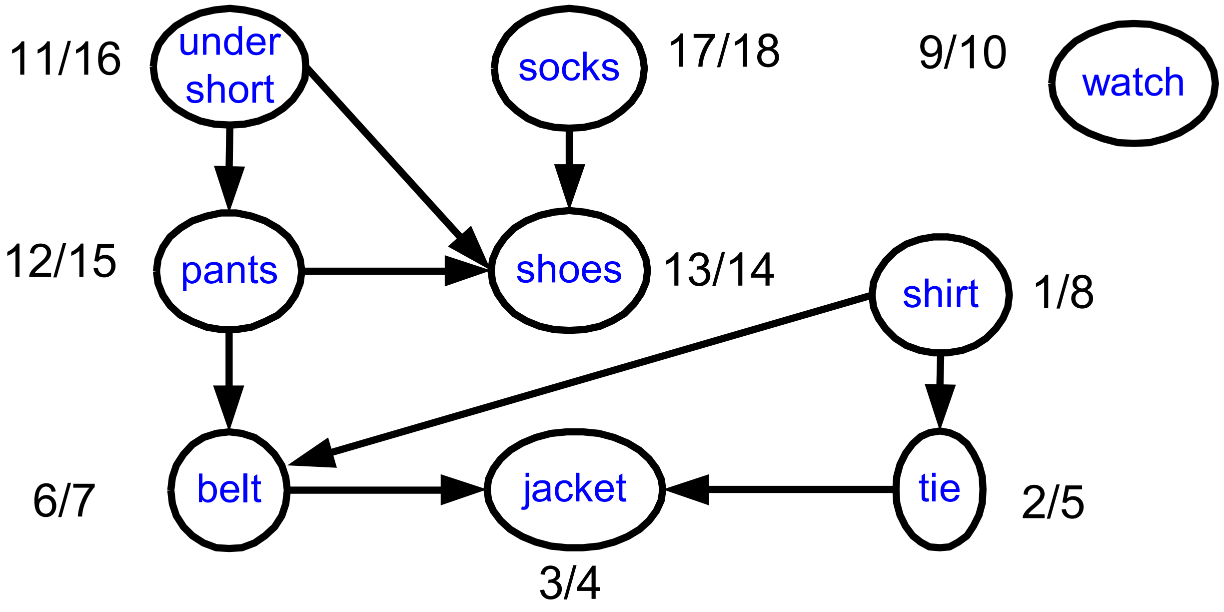
Topological Sort of a DAG

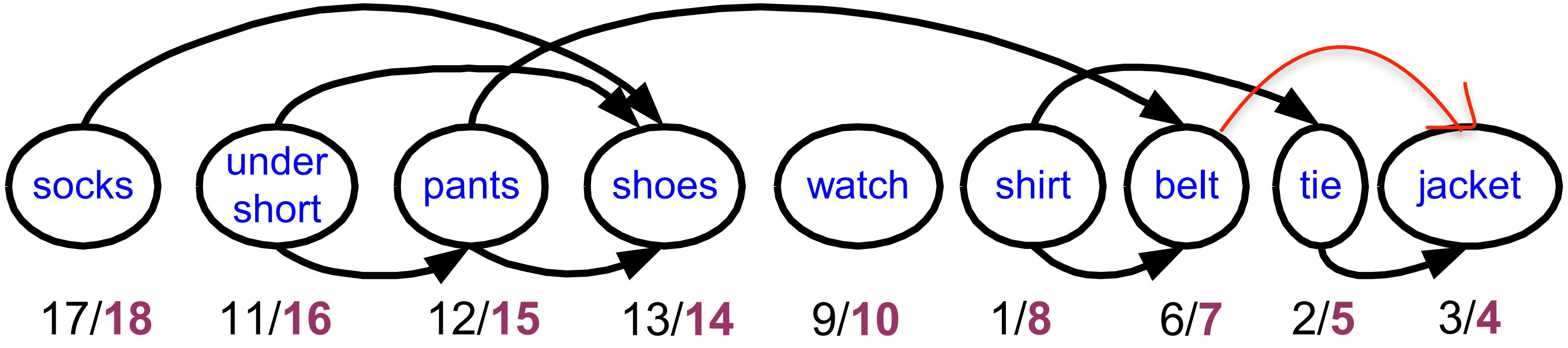
- Linear ordering of V such that

  $(u, v) \in E \implies u < v$ in ordering

  - Ordering may not be unique
  - i.e., mapping the partial ordering to total ordering may yield more than one orderings

Topological Sort of a DAG





Topological Sort of a DAG

Algorithm

run DFS(G)

when a vertex finished, output it

vertices output in reverse topologically sorted order

Runs in $O(V + E)$ time

Topological Sort of a DAG

Correctness of the Algorithm:

Claim: $(u, v) \in E \;\Rightarrow u.f > v.f$

Proof: consider any edge (u, v) explored by DFS

when (u, v) is explored, u is GRAY

- if v is GRAY, (u, v) is a Back edge (contradicting acyclic theorem)
- if v is WHITE, v becomes a descendent of u (b WPT) $\Rightarrow v.f < u.f$
- if v is BLACK, $v.f < u.f \Rightarrow v.f < u.f$

Questions