Australian National University

# COMP3600/6466 Algorithms

## Lecture 23

S2 2016

Dr. Hassan Hijazi

Prof. Weifa Liang

# Breadth-First Search (BFS)

**INPUT**: $G = (V, E)$, source node $s \in V$.

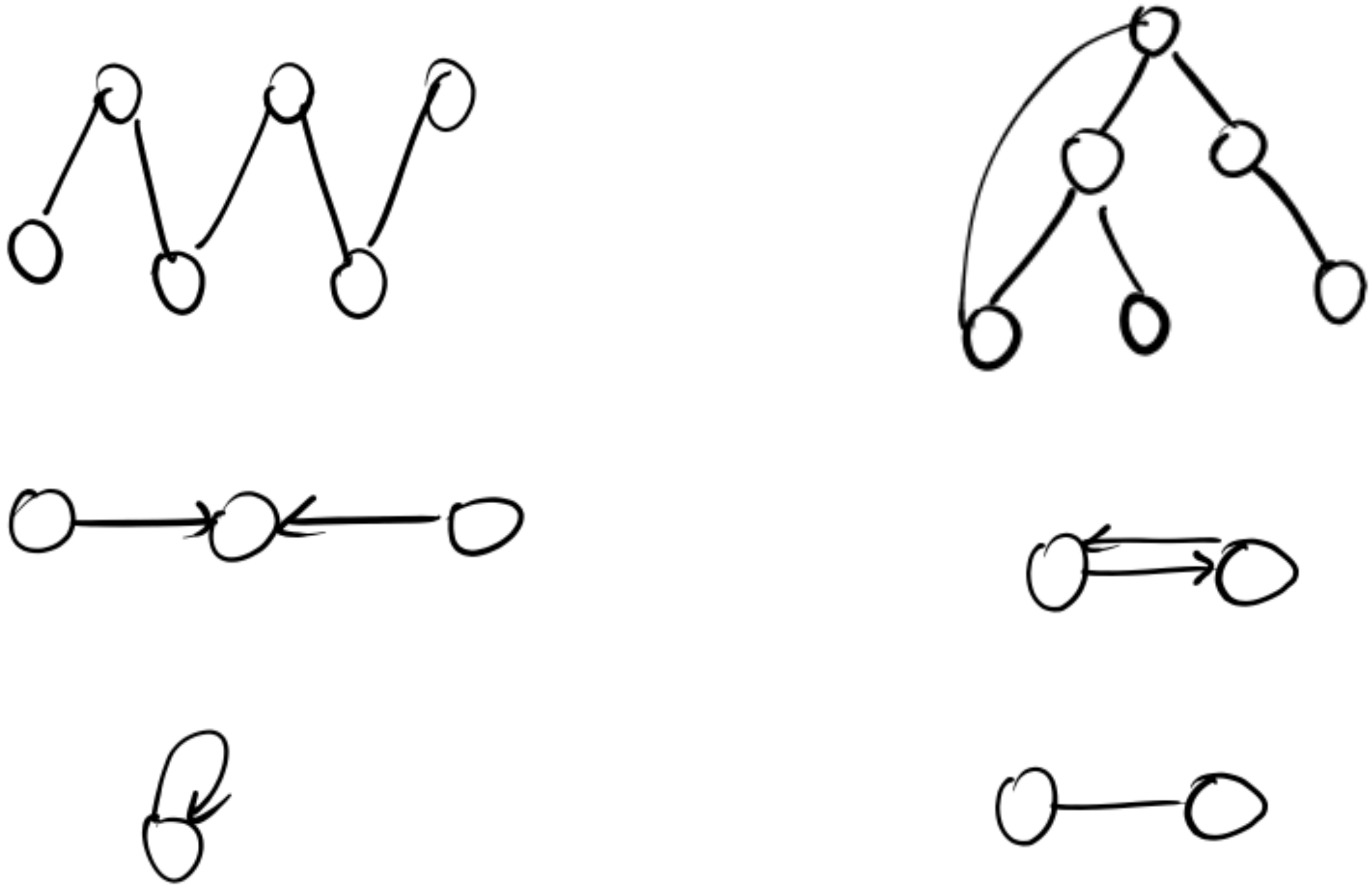
**OUTPUT**: The shortest distance (in number of edges) from $s$ to all its reachable nodes.

**OUTPUT**: A breadth-first tree of $s$ in $G$.

**When is a graph called a tree?**

When all nodes are connected
and
number of edges = number of nodes - 1

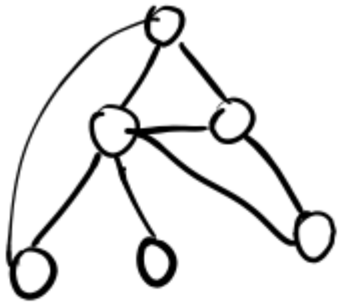
# Trees?

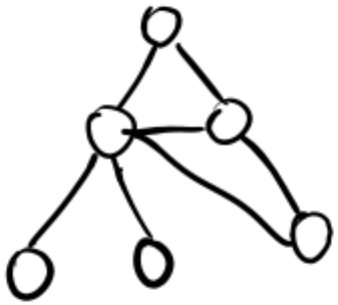


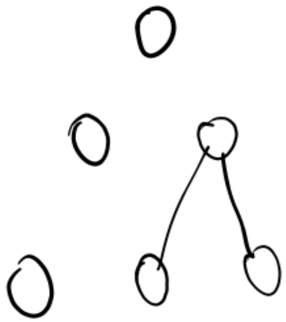
# Subgraphs and BFS trees

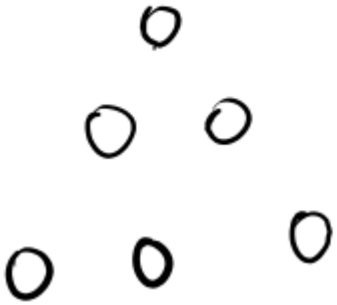
**What is a subgraph?**

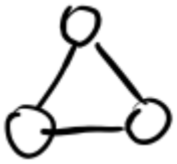
A subgraph $G_\pi = (V_\pi, E_\pi)$ of $G = (V, E)$ is a graph such as $V_\pi \subseteq V$ and $E_\pi \subseteq E$.











# Subgraphs and BFS trees

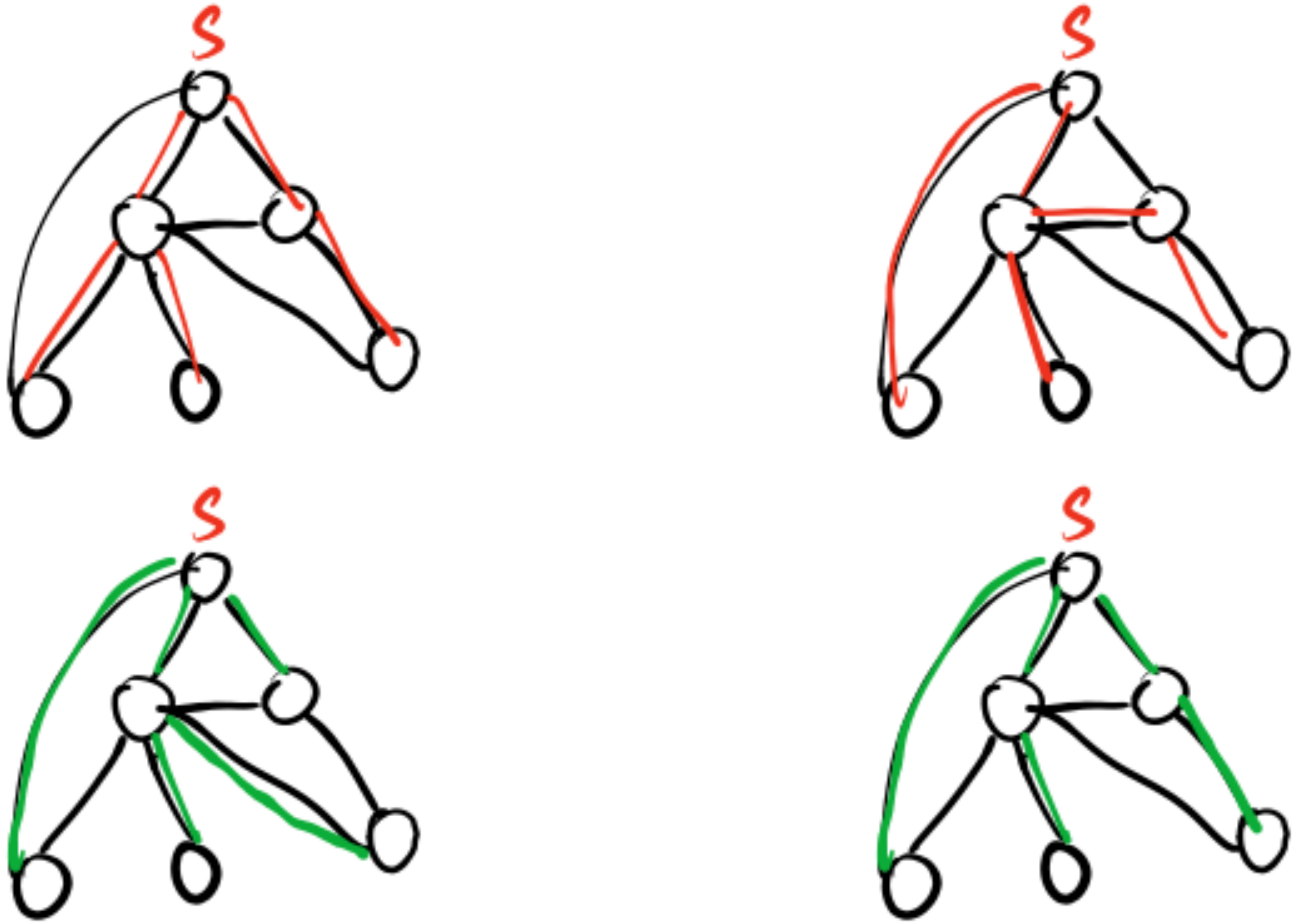
**What is a breadth-first tree?**

Given $G = (V, E)$, and a source node $s \in V$,

A subgraph $G_\pi = (V_\pi, E_\pi)$ of $G$
is a breadth-first tree of $s$ if and only if:

$V_\pi$ consists of the vertices reachable from $s$ and,
$\forall v \in V$, the subgraph $G_\pi$ contains a unique simple path (no cycles) from $s$ to $v$ that is also a shortest path from $s$ to $v$ in $G$.

# BFS trees ?



# Predecessor Graph in BFS

We define the **predecessor subgraph** of $s$ in $G$ as $G_\pi = (V_\pi, E_\pi)$, where

$$V_\pi = \{v \in V : \; v.\pi \neq NIL\} \cup \{s\} \text{ and } E_\pi = \{(v.\pi, v) : \; v \in V_\pi \setminus \{s\}\}$$

**Main Theorem**

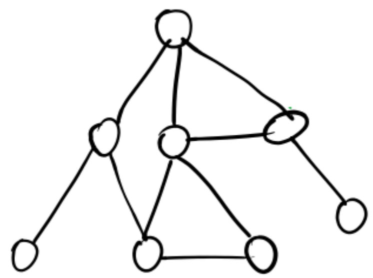
After applying algorithm BFS on a graph $G = (V, E)$,
the predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is a breadth-first tree of $G$.

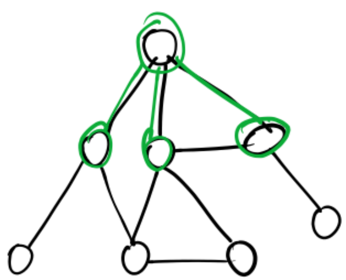
**Proof.** 1. BFS sets $v.\pi = u$ if and only if $(u, v) \in E$ and $\delta(s, v) < \infty$, thus $V_\pi$ consists of the vertices reachable from $s$ and $E_\pi \subseteq E$.

2. Since $G_\pi$ forms a tree, it contains a unique simple path from $s$ to each vertex in $V_\pi$.

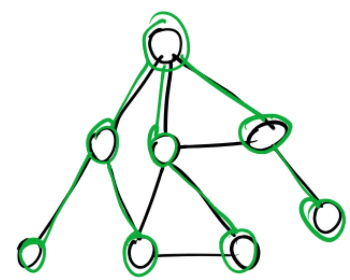
3. We have proved that $v.d = \delta(s, v)$, $\forall v \in V_\pi$ which implies that every edge along which $v$ is discovered is part of the shortest path from the source.

# Depth-First Search (DFS)



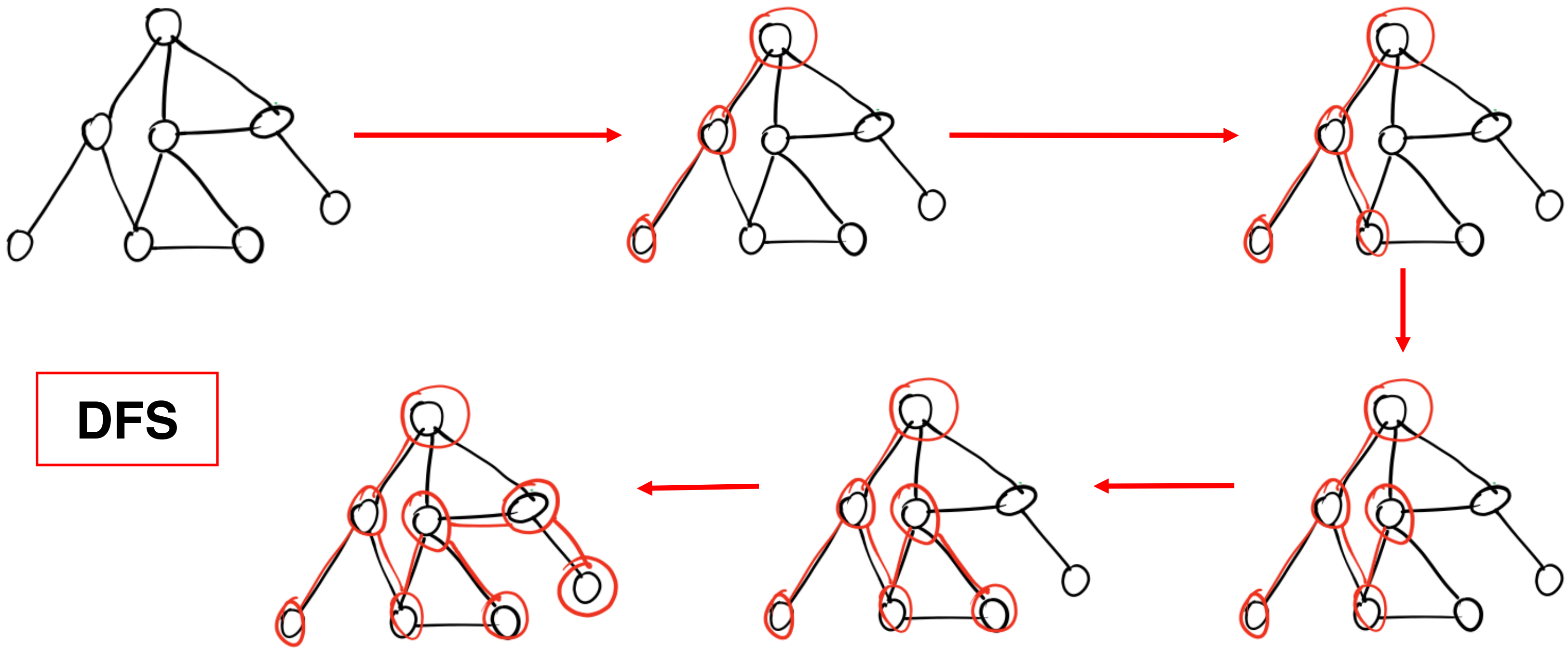


**BFS**



If nodes are grouped in levels, where level $i$ contains vertices that are at distance $i$ from the source, then BFS explores the nodes at a given level before moving to the next, while DFS dives all the way to the last level and then backtracks to dive again.

# Depth-First Search (DFS)



**DFS**

If nodes are grouped in levels, where level $i$ contains vertices that are at distance $i$ from the source, then BFS explores the nodes at a given level before moving to the next, while DFS dives all the way to the last level and then backtracks to dive again.

# DFS Implementation and Properties

We introduce a global variable denoted $time$.
Initially, $time = 0$.

$v$

$v.colour \in \{\text{white, grey, black}\}$
$v.\pi$, previous node on the path to $v$
$v.d$, discovery time (when $v$ is first visited)
$v.f$, finishing time (all the neighbours of $v$ have been examined)

# DFS Implementation and Properties

```
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)
```

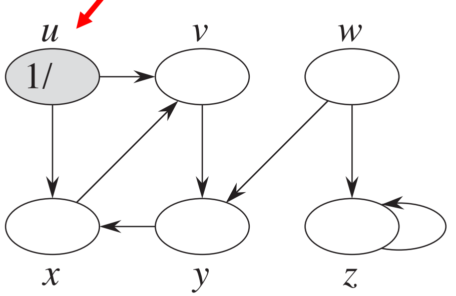
**Initialisation**

**Run DFS on all unvisited nodes**

```
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
```

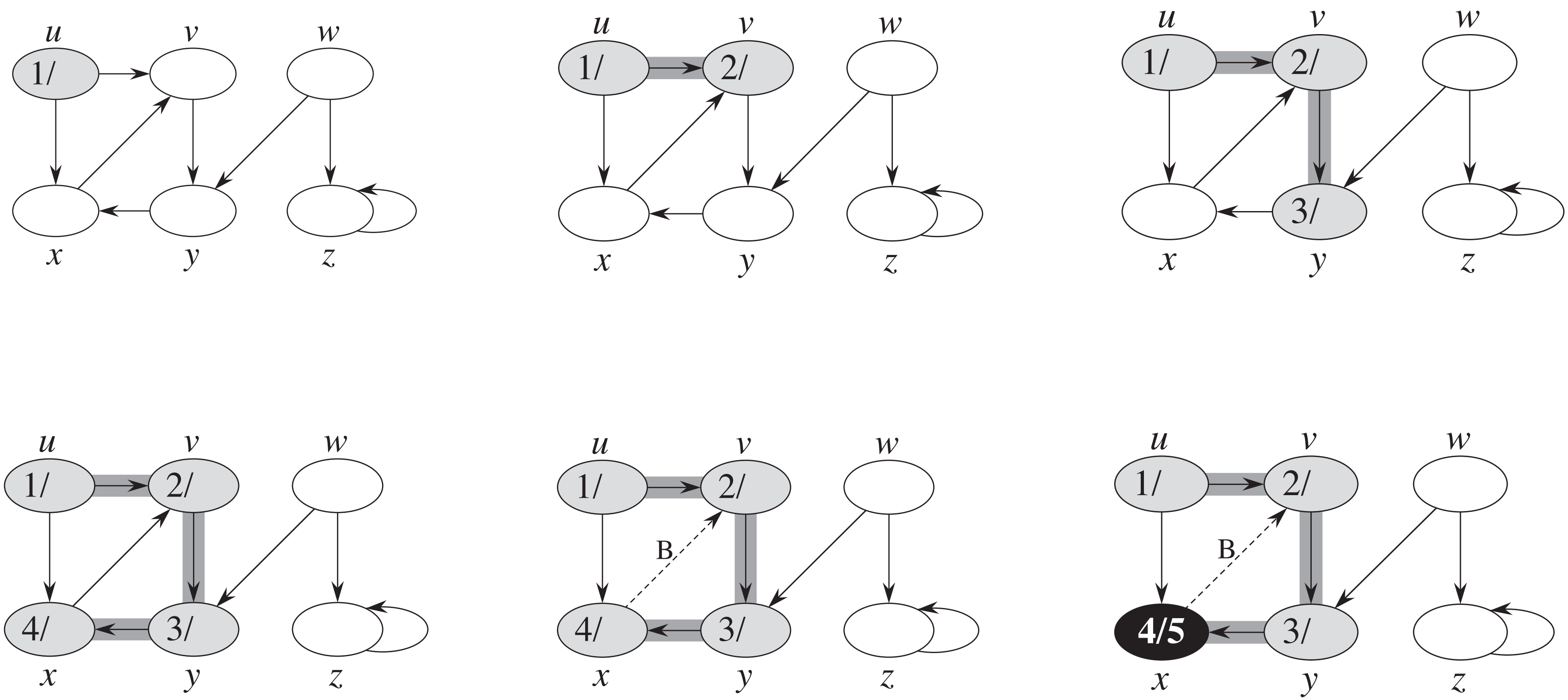
**DFS starting from root u**

**Grey** = node visited

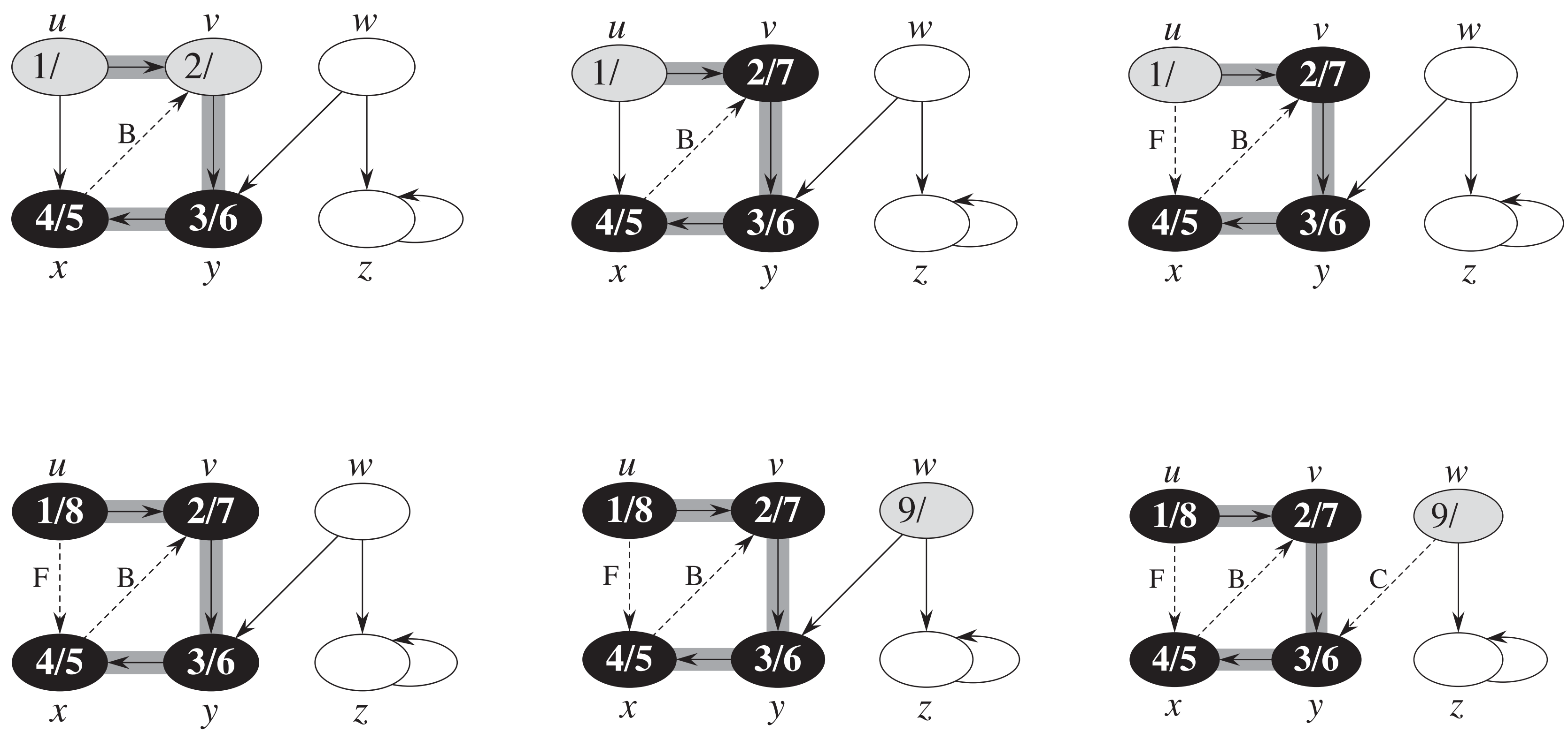



**White** = unvisited yet

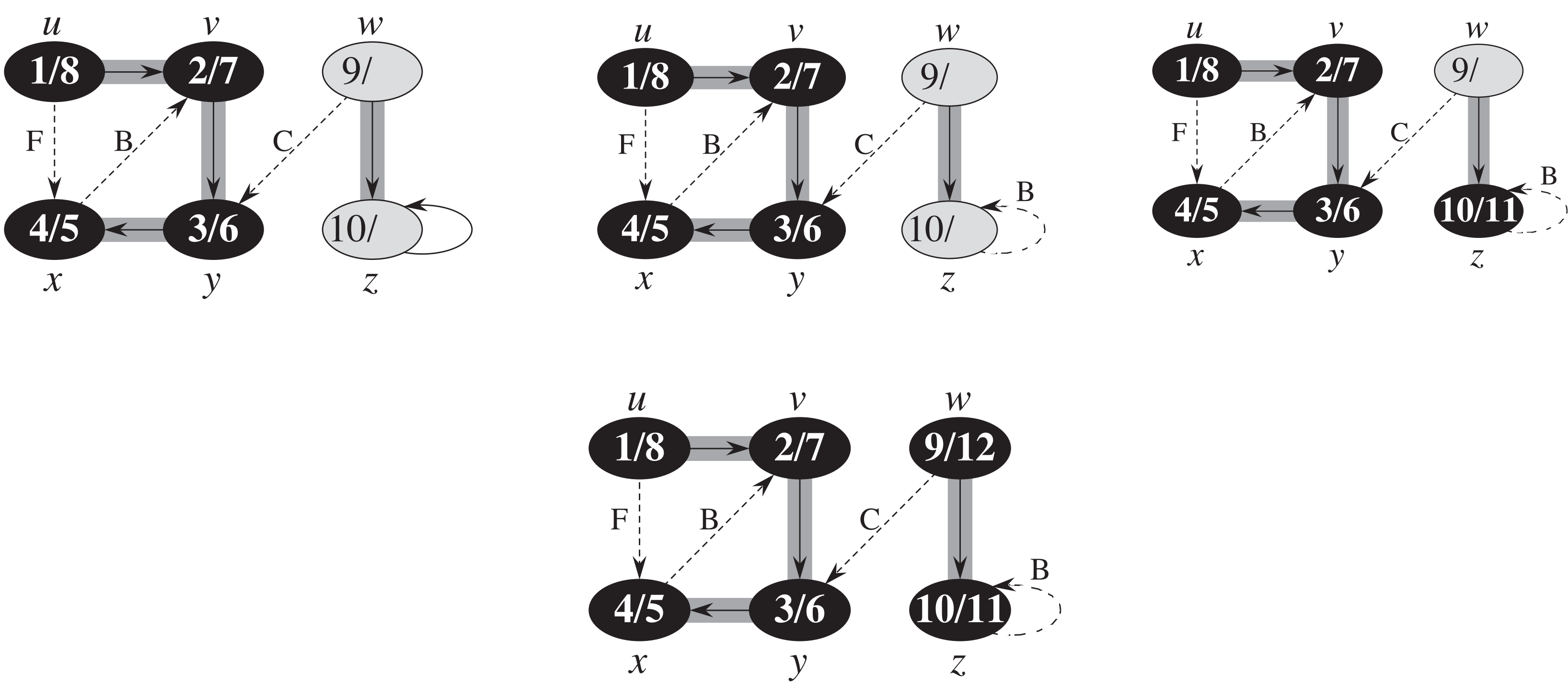
# DFS Implementation and Properties



# DFS Implementation and Properties



# DFS Implementation and Properties



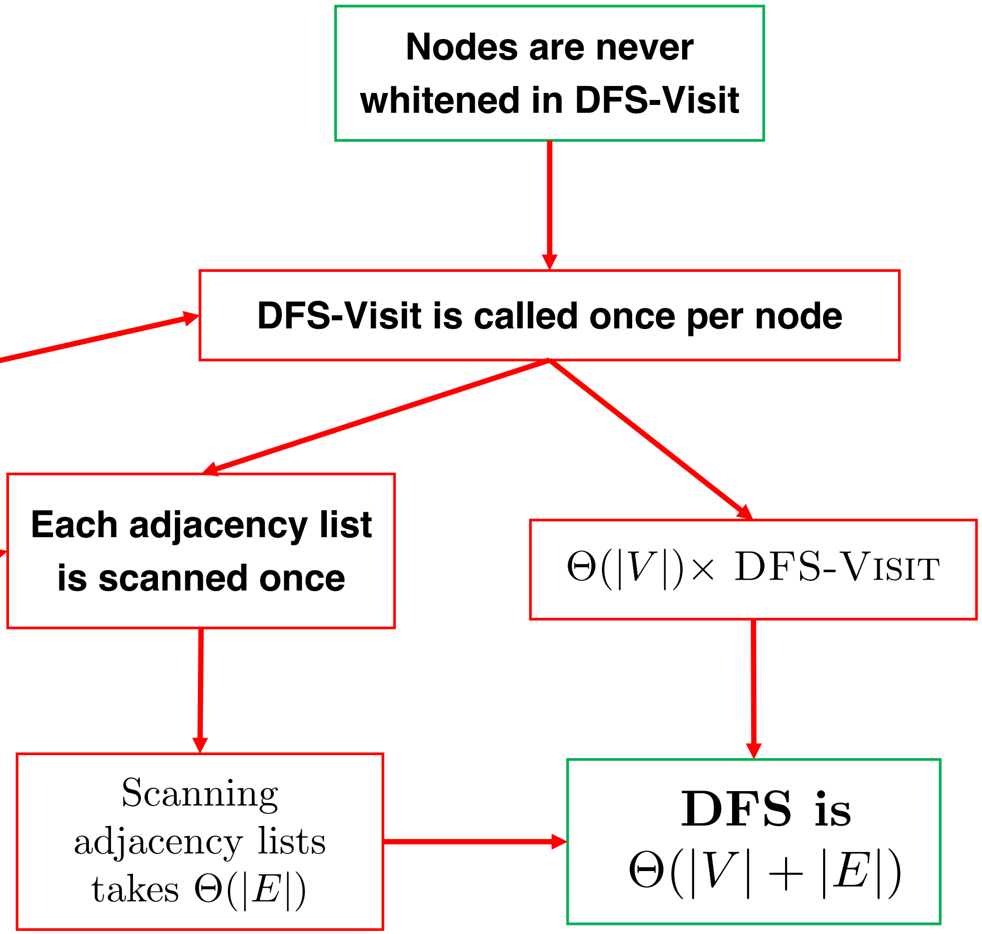
# DFS Analysis

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
```

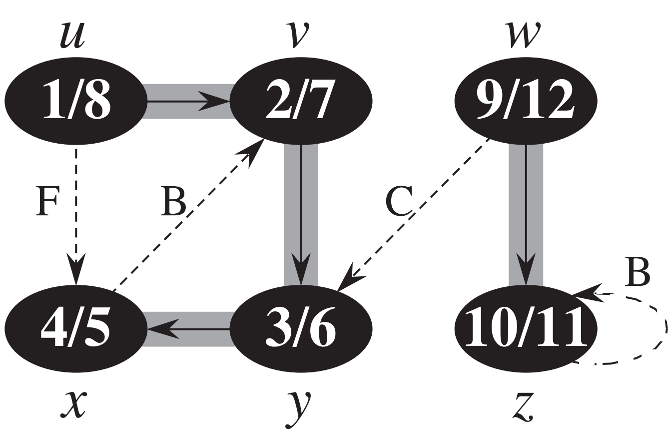


# Predecessor Graph for DFS

We define the **predecessor subgraph** of $G$ as $G_\pi = (V, E_\pi)$, where

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

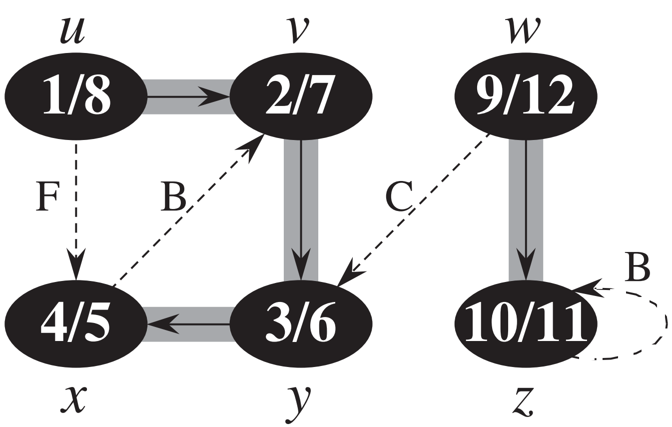
The predecessor subgraph of a depth-first search forms a **depth-first forest** comprising several depth-first trees. The edges in $E_\pi$ are **tree edges**.



# Descendants and Ancestors in DFS

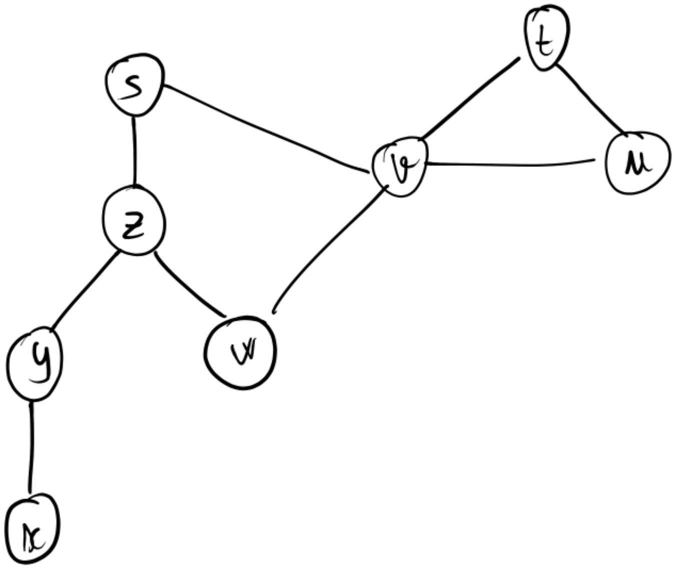
Vertex $v$ is a **descendant** of vertex $u$ in the depth-first forest for a graph $G$ if and only if $u.d < v.d < v.f < u.f$.

Vertex $u$ is an **ancestor** of $v$ if $v$ is a descendant of $u$.



# Exercise 23.1

Apply DFS on the following graph:




$s \to z \to v$

$z \to y \to w$

$y \to x$

$x$

$w \to z \to v$

$v \to u \to t \to s$

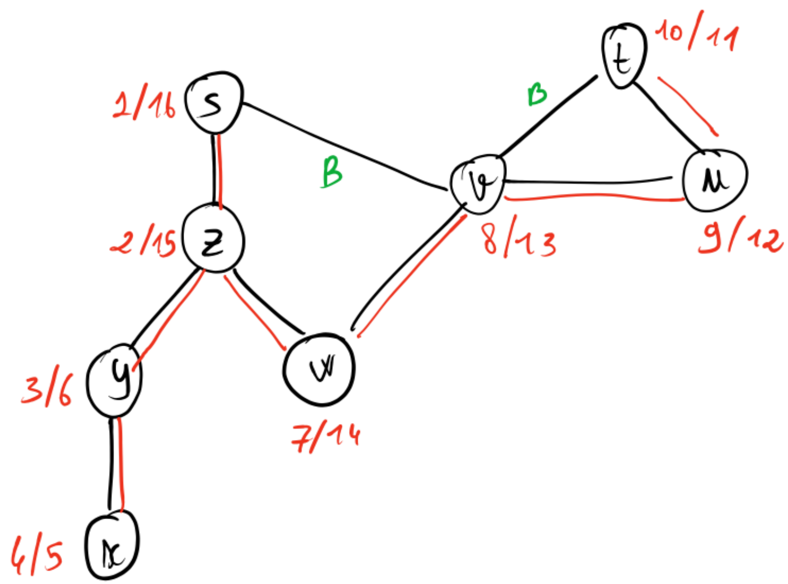
$u \to v \to t$

$t \to u \to v$

# Edge Classification (Undirected Graph)

DFS classifies the edges in an undirected graph into two types.

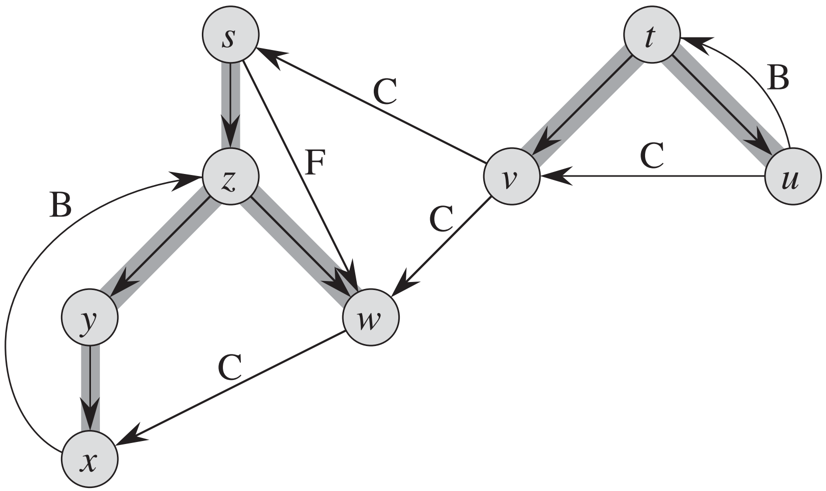
- Tree edges (edges along which a vertex is first discovered)
- Back edges (edges from a descendant to an ancestor)



# Edge Classification (Directed Graph)

DFS classifies the edges in a directed graph into four types.

- Tree edges (edges along which a vertex is first discovered)
- Back edges (edges from a descendant to an ancestor)
- Forward edges (non-tree edges from an ancestor to a descendant)
- Cross edges (all other edges)



# Acyclic Graphs

A directed graph $G(V, E)$ is called acyclic if it does not contain a directed cycle. Such a graph is also referred to as a **Directed Acyclic Graph (DAG)**.

**Theorem.** A directed graph $G$ has a directed cycle if and only if a depth-first search of $G$ yields a back edge.

**Proof:**
$(\Leftarrow)$ If there is a back edge $(u, v)$, then $v$ is an ancestor of $u$. The path of tree edges from $v$ to $u$, together with $(u, v)$, forms a directed cycle.

# Acyclic Graphs

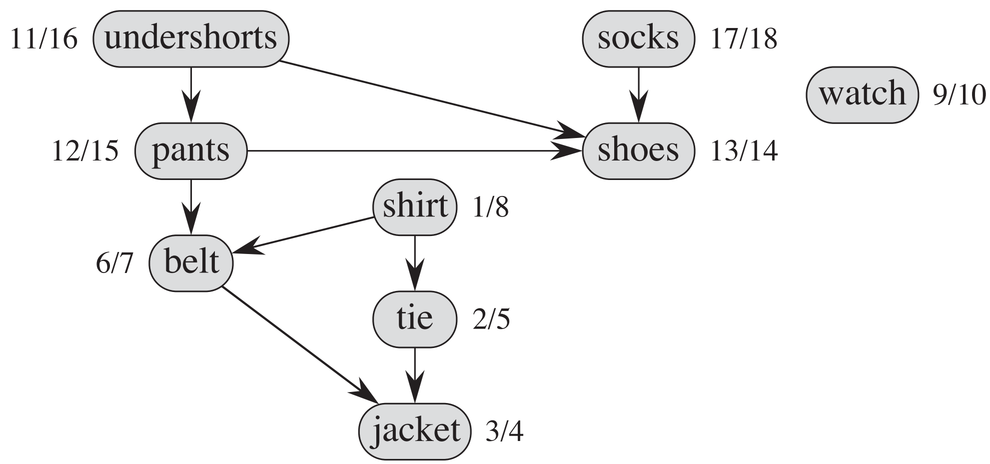
**Theorem.** A directed graph $G$ has a directed cycle if and only if a depth-first search of $G$ yields a back edge.

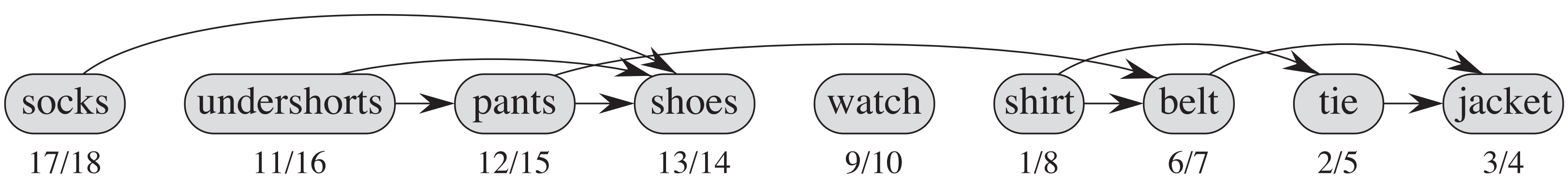
**Proof:**
$(\Leftarrow)$ If there is a back edge $(u, v)$, then $v$ is an ancestor of $u$. The path of tree edges from $v$ to $u$, together with $(u, v)$, forms a directed cycle.

$(\Rightarrow)$ Let $v_1, v_2, \ldots, v_k, v_1$ be a directed cycle in $G$, where $v_1$ is the first vertex discovered during DFS. By induction, vertices $v_2, \ldots, v_k$ are descendants of $v_1$. Therefore $(v_k, v_1)$ is a back edge.

# Topological Sort

A **topological sort** of a DAG is a linear ordering of the vertices s.t. every edge with endpoints in the ordered sequence is directed from left to right.



# DFS and Topological Sort

**Theorem.** Perform DFS on a directed acyclic graph $G$. Then, the vertex sequence listed by the decreasing order of their finishing times forms a topological sort for $G$.

**Proof.**
Let $(u, v)$ be an edge of $G$. We need to show that $v.f < u.f$.

Since $G$ is acyclic, we cannot have back edges,
thus we will only consider two cases:

**Case 1.** $(u, v)$ is a tree edge or a forward edge
Note that the finish time of a vertex $x$ is when the call DFS-VISIT$(G, x)$ returns. The recursive structure of DFS-VISIT implies that descendants always finish before their ancestors.
**Case 2.** $(u, v)$ is a cross edge
This implies that $v$ is not reachable from $u$ (otherwise we would have a cycle), and that $v$ is in the connected component that was explored before the one containing $u$ (otherwise $v$ would be a descendant of $u$). Therefore, the call to DFS-VISIT$(G, v)$ returns before visiting $u$, and we have $v.f < u.f$.