

# CSE 221: Algorithms

## Introduction to graphs

Mumit Khan

Computer Science and Engineering  
BRAC University

### References

- 1 Jon Kleinberg and Éva Tardos, *Algorithm Design*. Pearson Education, 2006.
- 2 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.
- 3 M. Goodrich and R. Tamassia, *Algorithm Design*. John-Wiley and Sons. 2002.

Last modified: September 10, 2009



This work is licensed under the *Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License*.

# Contents

## 1 Introduction to Graphs

- Graph basics
- Graph traversal

# Contents

- 1 Introduction to Graphs
  - Graph basics
  - Graph traversal

# What is a Graph?

- **Graph**  $G$  is a pair  $(V, E)$ , where  $V$  is a finite set (set of vertices) and  $E$  is a finite set of pairs from  $V$  (set of edges). We will often denote  $n = |V|$ ,  $m = |E|$ .
- Graph  $G$  can be **directed**, if  $E$  consists of ordered pairs, or **undirected**, if  $E$  consists of unordered pairs. If  $(u, v) \in E$ , then vertices  $u$  and  $v$  are **adjacent**.
- We can assign weight function to the edges:  $w_G(e)$  is a weight of edge  $e \in E$ . The graph which has such function assigned is called **weighted**.
- **Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

# Examples of graphs

**Transportation networks** How should you design the highway network in a country? What is the quickest way to drive from Badda to Naraynganj?

**Communication networks** How to send a network packet from Bracu intranet to Yahoo mail server?

**Information networks** Is the World wide a directed or undirected network?

**Social networks** Facebook, Myspace, Flickr, . . . .

**Dependency networks** What courses must you take before you can take CSE-423?

# Examples of graphs

**Transportation networks** How should you design the highway network in a country? What is the quickest way to drive from Badda to Naraynganj?

**Communication networks** How to send a network packet from Bracu intranet to Yahoo mail server?

**Information networks** Is the World wide a directed or undirected network?

**Social networks** Facebook, Myspace, Flickr, . . . .

**Dependency networks** What courses must you take before you can take CSE-423?

# Examples of graphs

**Transportation networks** How should you design the highway network in a country? What is the quickest way to drive from Badda to Naraynganj?

**Communication networks** How to send a network packet from Bracu intranet to Yahoo mail server?

**Information networks** Is the World wide a directed or undirected network?

**Social networks** Facebook, Myspace, Flickr, . . . .

**Dependency networks** What courses must you take before you can take CSE-423?

# Examples of graphs

**Transportation networks** How should you design the highway network in a country? What is the quickest way to drive from Badda to Naraynganj?

**Communication networks** How to send a network packet from Bracu intranet to Yahoo mail server?

**Information networks** Is the World wide a directed or undirected network?

**Social networks** Facebook, Myspace, Flickr, . . . .

**Dependency networks** What courses must you take before you can take CSE-423?



# Examples of graphs

**Transportation networks** How should you design the highway network in a country? What is the quickest way to drive from Badda to Naraynganj?

**Communication networks** How to send a network packet from Bracu intranet to Yahoo mail server?

**Information networks** Is the World wide a directed or undirected network?

**Social networks** Facebook, Myspace, Flickr, . . . .

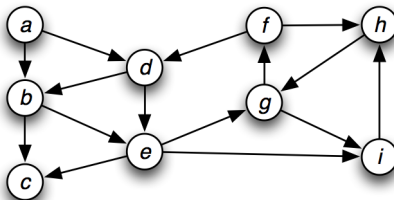
**Dependency networks** What courses must you take before you can take CSE-423?

# What is a Graph?

## Definition (Graph)

**Graph**  $G$  is a pair  $(V, E)$ , where  $V$  is a finite set (set of vertices) and  $E$  is a finite set of pairs from  $V$  (set of edges). We will often denote  $n = |V|$ ,  $m = |E|$ .

## Example

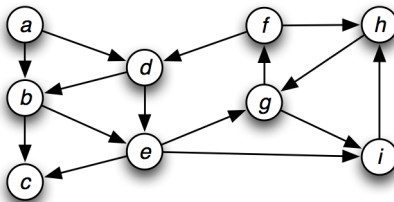


# What is a Graph?

## Definition (Graph)

**Graph**  $G$  is a pair  $(V, E)$ , where  $V$  is a finite set (set of vertices) and  $E$  is a finite set of pairs from  $V$  (set of edges). We will often denote  $n = |V|$ ,  $m = |E|$ .

## Example



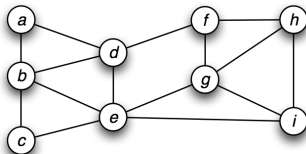
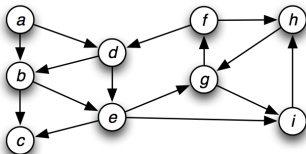
$V = \{a, \dots, i\}, n = 9, m = 15.$

# Directed vs. undirected graphs

## Definition (Directed vs. undirected graph)

Graph  $G$  can be **directed**, if  $E$  consists of ordered pairs, or **undirected**, if  $E$  consists of unordered pairs. If  $(u, v) \in E$ , then vertices  $u$  and  $v$  are **adjacent**.

## Example

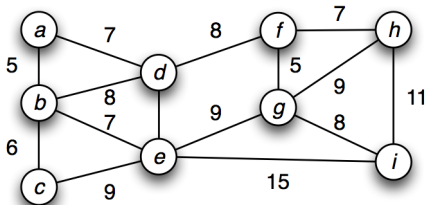


# Weighted graphs

## Definition (Weighted graph)

We can assign weight function to the edges:  $w_G(e)$  is a weight of edge  $e \in E$ . The graph which has such function assigned is called **weighted**.

## Example

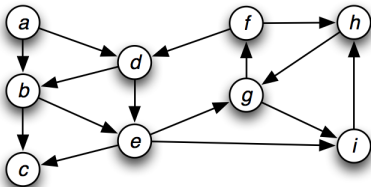


# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Directed graph)



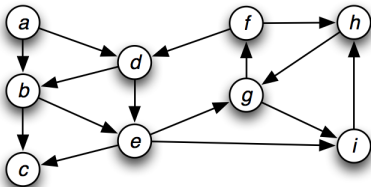
Vertex	in	out
--------	----	-----

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Directed graph)



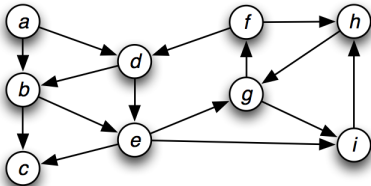
Vertex	in	out
a	0	2

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Directed graph)



Vertex	in	out
<i>a</i>	0	2
<i>b</i>	2	2

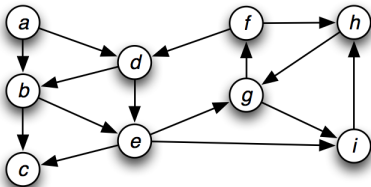


# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Directed graph)



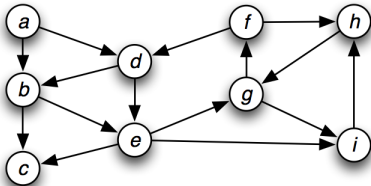
Vertex	in	out
<i>a</i>	0	2
<i>b</i>	2	2
<i>c</i>	2	0

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Directed graph)



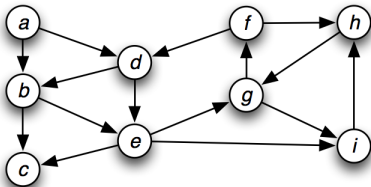
Vertex	in	out
<i>a</i>	0	2
<i>b</i>	2	2
<i>c</i>	2	0
<i>d</i>	2	2

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Directed graph)



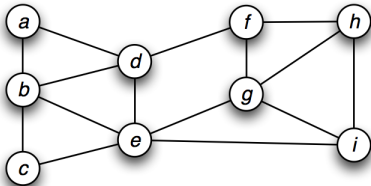
Vertex	in	out
<i>a</i>	0	2
<i>b</i>	2	2
<i>c</i>	2	0
<i>d</i>	2	2
<i>e</i>	2	3

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Undirected graph)



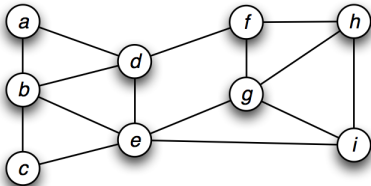
Vertex	$\deg$
--------	--------

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Undirected graph)



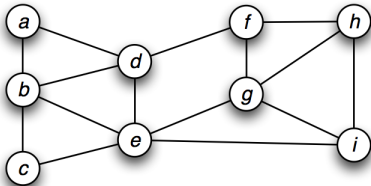
Vertex	$\deg$
$a$	2

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Undirected graph)



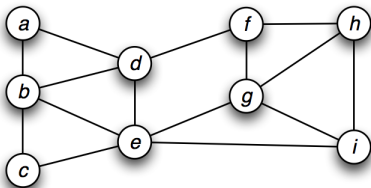
Vertex	$\deg$
$a$	2
$b$	4

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Undirected graph)



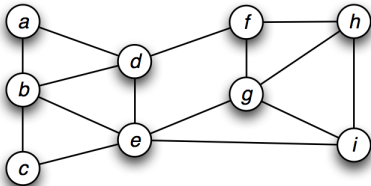
Vertex	$\deg$
$a$	2
$b$	4
$c$	2

# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Undirected graph)



Vertex	$\deg$
$a$	2
$b$	4
$c$	2
$d$	4

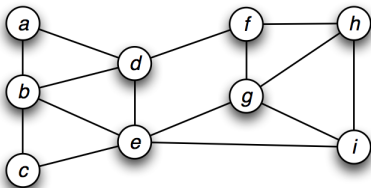


# Degree of vertices

## Definition (Degree of vertices)

**Degree** of a vertex  $v$  is the number of vertices  $u$  for which  $(u, v) \in E$  or  $(v, u) \in E$  (denote  $\deg(v)$ ). The number of **incoming edges** to a vertex  $v$  is called **in-degree** of the vertex (denote  $\text{indeg}(v)$ ). The number of **outgoing edges** from a vertex is called **out-degree** (denote  $\text{outdeg}(v)$ ).

## Example (Undirected graph)



Vertex	$\deg$
<i>a</i>	2
<i>b</i>	4
<i>c</i>	2
<i>d</i>	4
<i>e</i>	5

# Relationship of $n = |V|$ and $m = |E|$

## Theorem

*If  $G$  is a graph with  $m$  edges, then*

$$\sum_{v \in G} \deg(v) = 2m.$$

# Relationship of $n = |V|$ and $m = |E|$

## Theorem

*If  $G$  is a graph with  $m$  edges, then*

$$\sum_{v \in G} \deg(v) = 2m.$$

## Proof.

An edge  $(u, v)$  is counted twice in the above summation; once by its endpoint  $u$  and once by its endpoint  $v$ . Thus, the total contribution of the edges to the degrees of the vertices is twice the number of edges. □

Relationship of  $n = |V|$  and  $m = |E|$ 

## Theorem

*If  $G$  is a directed graph (digraph) with  $m$  edges, then*

$$\sum_{v \in G} \text{indeg}(v) = \sum_{v \in G} \text{outdeg}(v) = m.$$

# Relationship of $n = |V|$ and $m = |E|$

## Theorem

If  $G$  is a directed graph (digraph) with  $m$  edges, then

$$\sum_{v \in G} \text{indeg}(v) = \sum_{v \in G} \text{outdeg}(v) = m.$$

## Proof.

In a directed graph, an edge  $(u, v)$  contributes one unit to the **out-degree** of its origin  $u$  and one unit to the **in-degree** of its destination  $v$ . Thus, the total contribution of the edges to the out-degrees of the vertices is equal to the number of edges, and similarly for the in-degrees. □

# Relationship of $n = |V|$ and $m = |E|$

## Theorem

If  $G$  is a simple **undirected** graph with  $n$  vertices and  $m$  edges, then  $m \leq n(n-1)/2$ .

# Relationship of $n = |V|$ and $m = |E|$

## Theorem

If  $G$  is a simple **undirected** graph with  $n$  vertices and  $m$  edges, then  $m \leq n(n-1)/2$ .

## Proof.

Since  $G$  is a simple undirected graph without self-loops, the maximum degree of a vertex is  $n-1$ . And, since there are  $n$  vertices, and each edge is counted twice (once for each end of an edge  $(u, v)$ ), the maximum number of edges is  $n(n-1)/2$ . Thus,  $m \leq n(n-1)/2$ , or  $m = \Theta(n^2)$ . □

# Relationship of $n = |V|$ and $m = |E|$

## Theorem

If  $G$  is a simple **directed** graph with  $n$  vertices and  $m$  edges, then  $m \leq n(n - 1)$ .



# Relationship of $n = |V|$ and $m = |E|$

## Theorem

If  $G$  is a simple **directed** graph with  $n$  vertices and  $m$  edges, then  $m \leq n(n - 1)$ .

## Proof.

Since  $G$  is a simple graph without self-loops, the maximum in-degree of a vertex is  $n - 1$ . And, since there are  $n$  vertices, and no two edges can have the same origin and destination,  $m \leq n(n - 1)$ , or  $m = \Theta(n^2)$ . □

# Paths and connectivity

## Definition (Path)

A **path** in a undirected graph  $G = (V, E)$  is sequence  $P$  of vertices  $v_1, v_2, \dots, v_{k-1}, v_k$  with the property that for  $i \in \{1, \dots, k-1\}$ ,  $(v_i, v_{i+1}) \in E$ .

# Paths and connectivity

## Definition (Path)

A **path** in a undirected graph  $G = (V, E)$  is sequence  $P$  of vertices  $v_1, v_2, \dots, v_{k-1}, v_k$  with the property that for  $i \in \{1, \dots, k-1\}$ ,  $(v_i, v_{i+1}) \in E$ .

## Definition (Simple and cyclic paths)

A path is **simple** if all vertices are distinct. A **cycle** is a path  $v_1, v_2, \dots, v_{k-1}, v_k$  in which  $k \geq 2$ , and the first  $k-1$  nodes are all distinct, and  $v_1 = v_k$ .

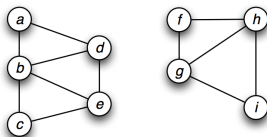
# Connected vs. unconnected graphs

## Definition (Connected graph)

The undirected graph  $G$  is **connected**, if for every pair of vertices  $u, v$  there exists a **path** from  $u$  to  $v$ .

If a graph is not connected, the vertices of the graph can be divided into **connected components**. Two vertices are in the same **connected component** *iff* they are connected by a **path**.

## Example (Connected components)



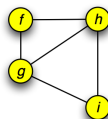
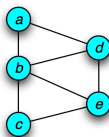
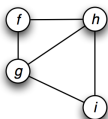
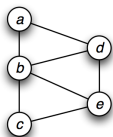
# Connected vs. unconnected graphs

## Definition (Connected graph)

The undirected graph  $G$  is **connected**, if for every pair of vertices  $u, v$  there exists a **path** from  $u$  to  $v$ .

If a graph is not connected, the vertices of the graph can be divided into **connected components**. Two vertices are in the same **connected component** *iff* they are connected by a **path**.

## Example (Connected components)



# Trees

## Definition (Trees)

A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

# Trees

## Definition (Trees)

A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

*For undirected graph  $G$ , any two of the following implies the third.*

- ①  $G$  is connected.
- ②  $G$  does not contain a cycle.
- ③  $G$  has  $n - 1$  edges.

# Trees

## Definition (Trees)

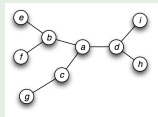
A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

For undirected graph  $G$ , any two of the following implies the third.

- 1  $G$  is connected.
- 2  $G$  does not contain a cycle.
- 3  $G$  has  $n - 1$  edges.

## Example (Trees (or not))





# Trees

## Definition (Trees)

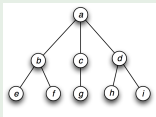
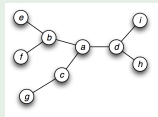
A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

For undirected graph  $G$ , any two of the following implies the third.

- 1  $G$  is connected.
- 2  $G$  does not contain a cycle.
- 3  $G$  has  $n - 1$  edges.

## Example (Trees (or not))



# Trees

## Definition (Trees)

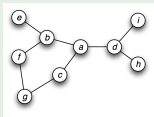
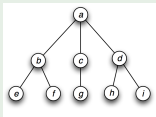
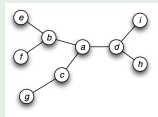
A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

For undirected graph  $G$ , any two of the following implies the third.

- ❶  $G$  is connected.
- ❷  $G$  does not contain a cycle.
- ❸  $G$  has  $n - 1$  edges.

## Example (Trees (or not))



# Trees

## Definition (Trees)

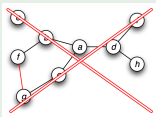
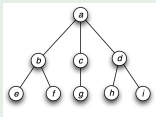
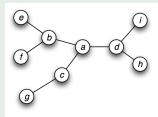
A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

For undirected graph  $G$ , any two of the following implies the third.

- 1  $G$  is connected.
- 2  $G$  does not contain a cycle.
- 3  $G$  has  $n - 1$  edges.

## Example (Trees (or not))



# Trees

## Definition (Trees)

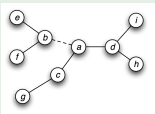
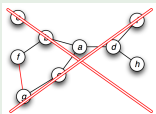
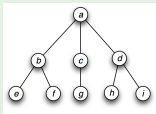
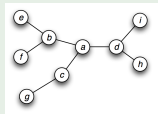
A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

For undirected graph  $G$ , any two of the following implies the third.

- ❶  $G$  is connected.
- ❷  $G$  does not contain a cycle.
- ❸  $G$  has  $n - 1$  edges.

## Example (Trees (or not))



# Trees

## Definition (Trees)

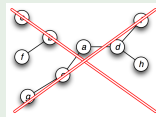
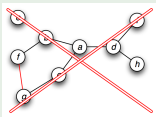
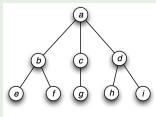
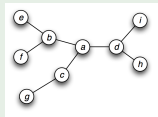
A undirected graph is a *tree* if it is *connected* and does not contain a *cycle* (implies that every  $n$ -node tree has exactly  $n - 1$  edges).

## Theorem

For undirected graph  $G$ , any two of the following implies the third.

- ❶  $G$  is connected.
- ❷  $G$  does not contain a cycle.
- ❸  $G$  has  $n - 1$  edges.

## Example (Trees (or not))



# Representation of graphs

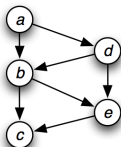
## Definition (Adjacency Matrix)

Represents the graph as an  $n \times n$  matrix  $A = (a_{i,j})$ , where

$$a_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

The matrix is symmetric in case of undirected graph, while it may be asymmetric if the graph is directed.

## Example (Directed graph)



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>	0	1	0	1	0
<i>b</i>	0	0	1	0	1
<i>c</i>	0	0	0	0	0
<i>d</i>	0	1	0	0	1
<i>e</i>	0	0	1	0	0

# Representation of graphs

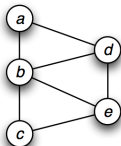
## Definition (Adjacency Matrix)

Represents the graph as an  $n \times n$  matrix  $A = (a_{i,j})$ , where

$$a_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

The matrix is symmetric in case of undirected graph, while it may be asymmetric if the graph is directed.

## Example (Undirected graph)



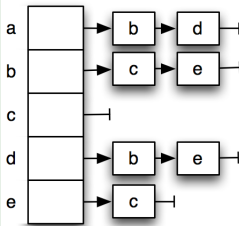
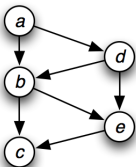
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>	0	1	0	1	0
<i>b</i>	1	0	1	1	1
<i>c</i>	0	1	0	0	1
<i>d</i>	1	1	0	0	1
<i>e</i>	0	1	1	1	0

# Representation of graphs

## Definition (Adjacency List)

Represent the graph by listing for each vertex  $v_i$  its outgoing vertices in a list  $out(v_i)$ . (Representation can be linked list, or another appropriate structure.)

## Example (Directed graph)





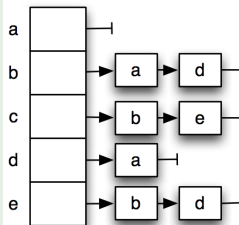
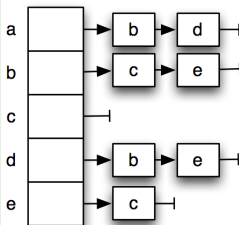
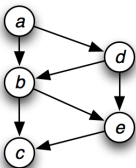
# Representation of graphs

## Definition (Adjacency List)

Represent the graph by listing for each vertex  $v_i$  its outgoing vertices in a list  $out(v_i)$ . (Representation can be linked list, or another appropriate structure.)

If the graph is directed, it makes sense to build for each vertex  $v_i$  also list of its incoming vertices  $in(v_i)$ .

## Example (Directed graph)



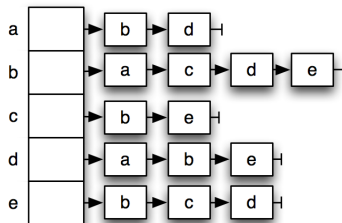
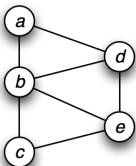
# Representation of graphs

## Definition (Adjacency List)

Represent the graph by listing for each vertex  $v_i$  its outgoing vertices in a list  $out(v_i)$ . (Representation can be linked list, or another appropriate structure.)

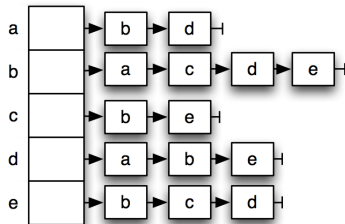
If the graph is undirected, all incident edges are listed for each vertex  $v_i$ .

## Example (Undirected graph)



# Comparison of graph representations

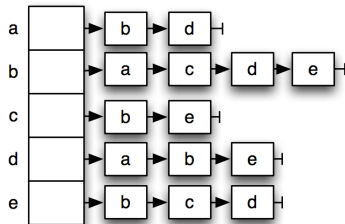
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>	0	1	0	1	0
<i>b</i>	1	0	1	1	1
<i>c</i>	0	1	0	0	1
<i>d</i>	1	1	0	0	1
<i>e</i>	0	1	1	1	0



Operation	Adjacency matrix	Adjacency list
Is $(u, v) \in E$ ?	$\Theta(1)$	$\Theta(\text{outdeg}(u))$
List edges outgoing from $u$	$\Theta(n)$	$\Theta(\text{outdeg}(u))$
Memory	$\Theta(n^2)$	$\Theta(m + n)$

# Comparison of graph representations

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>	0	1	0	1	0
<i>b</i>	1	0	1	1	1
<i>c</i>	0	1	0	0	1
<i>d</i>	1	1	0	0	1
<i>e</i>	0	1	1	1	0



Operation	Adjacency matrix	Adjacency list
Is $(u, v) \in E$ ?	$\Theta(1)$	$\Theta(\text{outdeg}(u))$
List edges outgoing from $u$	$\Theta(n)$	$\Theta(\text{outdeg}(u))$
Memory	$\Theta(n^2)$	$\Theta(m + n)$

## Representation note

We will be using adjacency lists to represent graphs, unless stated otherwise.

# Some very basic questions for a graph

- 1 Starting from a given *source* vertex, can I reach a *target* vertex? *s-t connectivity problem*.

# Some very basic questions for a graph

- 1 Starting from a given *source* vertex, can I reach a *target* vertex? *s-t connectivity problem*.
- 2 Starting from a given *source* vertex, how can I visit all the other vertices that have a path from it?

# Some very basic questions for a graph

- 1 Starting from a given *source* vertex, can I reach a *target* vertex? *s-t connectivity problem*.
- 2 Starting from a given *source* vertex, how can I visit all the other vertices that have a path from it?
- 3 Is the graph *connected*? If not, what are the *connected components*?

# Some very basic questions for a graph

- 1 Starting from a given *source* vertex, can I reach a *target* vertex? *s-t connectivity problem*.
- 2 Starting from a given *source* vertex, how can I visit all the other vertices that have a path from it?
- 3 Is the graph *connected*? If not, what are the *connected components*?
- 4 Is there a cycle in this graph?



# Some very basic questions for a graph

- 1 Starting from a given *source* vertex, can I reach a *target* vertex? *s-t connectivity problem*.
- 2 Starting from a given *source* vertex, how can I visit all the other vertices that have a path from it?
- 3 Is the graph *connected*? If not, what are the *connected components*?
- 4 Is there a cycle in this graph?
- 5 Given a *source* vertex in weighted graph, what is the *shortest distance* to all the other vertices? *Single-source shortest path problem*.

# Some very basic questions for a graph

- 1 Starting from a given *source* vertex, can I reach a *target* vertex? *s-t connectivity problem*.
- 2 Starting from a given *source* vertex, how can I visit all the other vertices that have a path from it?
- 3 Is the graph *connected*? If not, what are the *connected components*?
- 4 Is there a cycle in this graph?
- 5 Given a *source* vertex in weighted graph, what is the *shortest distance* to all the other vertices? *Single-source shortest path problem*.
- 6 How to compute the *minimum spanning tree*?

# Contents

## 1 Introduction to Graphs

- Graph basics
- Graph traversal

# Graph traversal techniques

## The graph traversal problem

Given a graph  $G = (V, E)$ , and a *distinguished vertex*  $s$ , how do you visit each vertex  $v \in V$  exactly once.

# Graph traversal techniques

## The graph traversal problem

Given a graph  $G = (V, E)$ , and a *distinguished vertex*  $s$ , how do you visit each vertex  $v \in V$  exactly once.

### Breadth-first search (BFS)

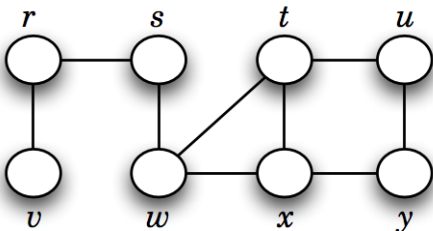
- Start with the source vertex, and *fan out* from there.
- Finds the distance (in terms of edges) of each node from the source.

### Depth-first search (DFS)

- Start with the source vertex, *recursively* visit each neighbor, only backtracking when choices are exhausted.
- Remember *pre-order* tree traversal?

# Basic idea behind breadth first search (BFS)

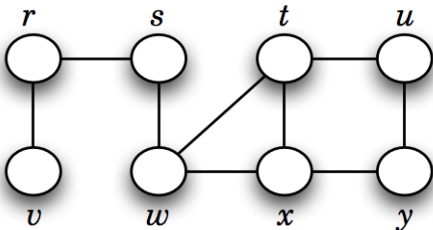
Given a undirected graph  $G = (V, E)$ , and a distinguished vertex  $s$ , systematically visit each vertex  $v \in V$  that is reachable from  $s$ .



# Breadth first search (BFS)

**BFS( $G, s$ )**     $\triangleright G = (V, E)$

```
1   $Q \leftarrow \{s\}$ 
2   $d[v] \leftarrow \infty, \forall v \in V; d[s] \leftarrow 0$ 
3  while  $Q \neq \emptyset$ 
4      do  $u \leftarrow \text{DEQUEUE}(Q)$ 
5          for each  $v \in \text{Adj}[u]$ 
6              do if  $d[v] = \infty$ 
7                  then  $d[v] = d[u] + 1$ 
8                       $\text{ENQUEUE}(Q, v)$ 
```

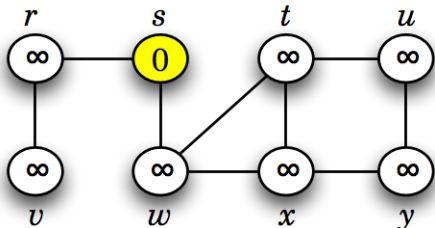


# Breadth first search (BFS)

**BFS( $G, s$ )**     $\triangleright G = (V, E)$

```

1   $Q \leftarrow \{s\}$ 
2   $d[v] \leftarrow \infty, \forall v \in V; d[s] \leftarrow 0$ 
3  while  $Q \neq \emptyset$ 
4      do  $u \leftarrow \text{DEQUEUE}(Q)$ 
5          for each  $v \in \text{Adj}[u]$ 
6              do if  $d[v] = \infty$ 
7                  then  $d[v] = d[u] + 1$ 
8                       $\text{ENQUEUE}(Q, v)$ 
  
```





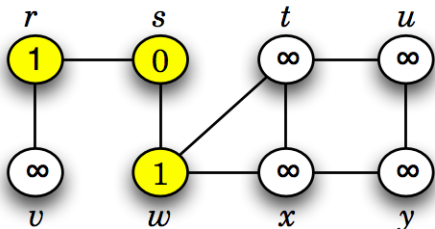
# Breadth first search (BFS)

**BFS( $G, s$ )**     $\triangleright G = (V, E)$

```

1   $Q \leftarrow \{s\}$ 
2   $d[v] \leftarrow \infty, \forall v \in V; d[s] \leftarrow 0$ 
3  while  $Q \neq \emptyset$ 
4      do  $u \leftarrow \text{DEQUEUE}(Q)$ 
5          for each  $v \in \text{Adj}[u]$ 
6              do if  $d[v] = \infty$ 
7                  then  $d[v] = d[u] + 1$ 
8                       $\text{ENQUEUE}(Q, v)$ 

```



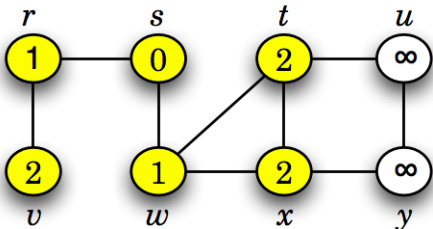
# Breadth first search (BFS)

**BFS( $G, s$ )**     $\triangleright G = (V, E)$

```

1   $Q \leftarrow \{s\}$ 
2   $d[v] \leftarrow \infty, \forall v \in V; d[s] \leftarrow 0$ 
3  while  $Q \neq \emptyset$ 
4      do  $u \leftarrow \text{DEQUEUE}(Q)$ 
5          for each  $v \in \text{Adj}[u]$ 
6              do if  $d[v] = \infty$ 
7                  then  $d[v] = d[u] + 1$ 
8                       $\text{ENQUEUE}(Q, v)$ 

```



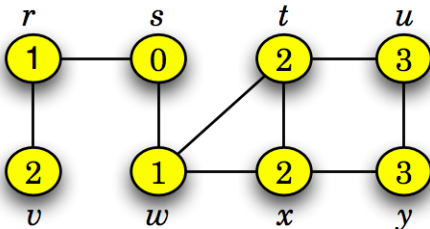
# Breadth first search (BFS)

**BFS( $G, s$ )**     $\triangleright G = (V, E)$

```

1   $Q \leftarrow \{s\}$ 
2   $d[v] \leftarrow \infty, \forall v \in V; d[s] \leftarrow 0$ 
3  while  $Q \neq \emptyset$ 
4      do  $u \leftarrow \text{DEQUEUE}(Q)$ 
5          for each  $v \in \text{Adj}[u]$ 
6              do if  $d[v] = \infty$ 
7                  then  $d[v] = d[u] + 1$ 
8                       $\text{ENQUEUE}(Q, v)$ 

```



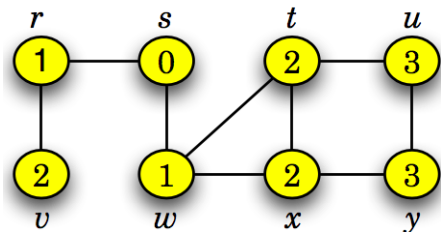
# Breadth first search (BFS)

```
BFS( $G, s$ )    ▷  $G = (V, E)$ 
1   $Q \leftarrow \{s\}$ 
2   $d[v] \leftarrow \infty, \forall v \in V; d[s] \leftarrow 0$ 
3  while  $Q \neq \emptyset$ 
4      do  $u \leftarrow \text{DEQUEUE}(Q)$ 
5          for each  $v \in \text{Adj}[u]$ 
6              do if  $d[v] = \infty$ 
7                  then  $d[v] = d[u] + 1$ 
8                       $\text{ENQUEUE}(Q, v)$ 
```

## Analysis

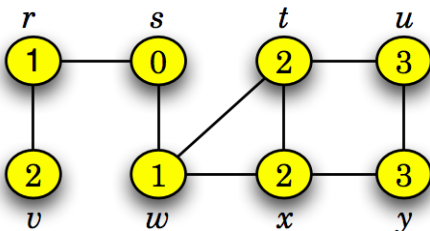
**Time** =  $O(V + E)$ .

# What does *this* $\text{BFS}(G, s)$ algorithm tell us?



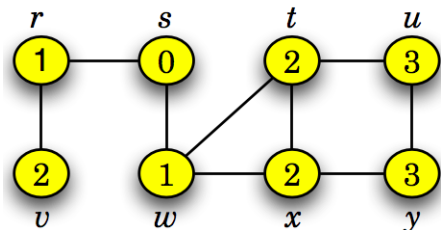
- This graph  $G = (V, E)$  is connected since BFS visits all vertices  $v \in V$  starting from  $s$ .

# What does *this* $\text{BFS}(G, s)$ algorithm tell us?



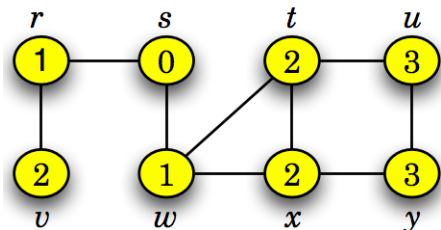
- This graph  $G = (V, E)$  is connected since BFS visits all vertices  $v \in V$  starting from  $s$ .
- The minimum number of *hops* or edges between  $s$  and any vertex  $t \in V$ .

# What does *this* $\text{BFS}(G, s)$ algorithm *not* tell us?



- Is there a cycle in this graph?

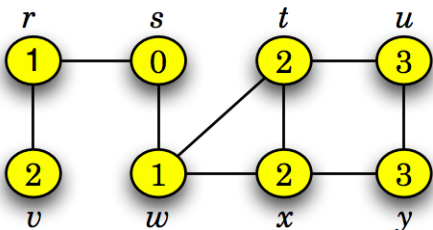
# What does *this* $\text{BFS}(G, s)$ algorithm *not* tell us?



- Is there a cycle in this graph?
- Given a target vertex  $t \in V$ , what is the path with smallest number of edges back to  $s$ ?

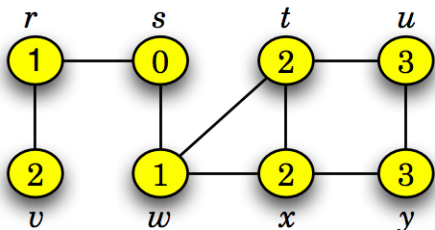


# What does *this* $\text{BFS}(G, s)$ algorithm *not* tell us?



- Is there a cycle in this graph?
- Given a target vertex  $t \in V$ , what is the path with smallest number of edges back to  $s$ ?
- If this were not a connected graph, how to visit all the vertices?

# What does *this* $\text{BFS}(G, s)$ algorithm *not* tell us?



- Is there a cycle in this graph?
- Given a target vertex  $t \in V$ , what is the path with smallest number of edges back to  $s$ ?
- If this were not a connected graph, how to visit all the vertices?
- Equivalently, how to compute all the connected components of a graph?

# Computing connected components

## Idea

Since each node is visited no more than once ( $d[v] \neq \infty$  after the first visit), simply apply the DFS algorithm over each vertex  $v \in V$ , creating a new component each time a vertex is found for which  $d[v] = \infty$ .

# Computing connected components

## Idea

Since each node is visited no more than once ( $d[v] \neq \infty$  after the first visit), simply apply the DFS algorithm over each vertex  $v \in V$ , creating a new component each time a vertex is found for which  $d[v] = \infty$ .

## BFS( $G$ )    $\triangleright G = (V, E)$

```
1   $d[v] \leftarrow \infty, \forall v \in V$ 
2  for each  $v \in V$ 
3      do if  $d[v] = \infty$ 
4          then BFS( $G, v$ )
```

# A better Breadth first search (BFS) algorithm

## Information during traversal

- 1 *color*[*v*]: the *color* of each vertex visited

# A better Breadth first search (BFS) algorithm

## Information during traversal

- 1  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**

# A better Breadth first search (BFS) algorithm

## Information during traversal

- ① *color*[*v*]: the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing

# A better Breadth first search (BFS) algorithm

## Information during traversal

- ① *color*[*v*]: the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing



# A better Breadth first search (BFS) algorithm

## Information during traversal

- 1  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing
- 2  $\pi[v]$ : the **predecessor** pointer

# A better Breadth first search (BFS) algorithm

## Information during traversal

- ①  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing
- ②  $\pi[v]$ : the **predecessor** pointer
- ③  $d[v]$ : the **discovery time** (or *hops* from the start)

# A better Breadth first search (BFS) algorithm

## Information during traversal

- 1  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing
- 2  $\pi[v]$ : the **predecessor** pointer
- 3  $d[v]$ : the **discovery time** (or *hops* from the start)

$BFS(G) \triangleright G = (V, E)$

▷ Initialize the vertex colors and predecessors

```
1 for each vertex  $v \in V$ 
2     do  $color[v] = \text{WHITE}$ 
3      $\pi[v] = \text{NULL}$ 
4 for each vertex  $v \in V$ 
5     do if  $color[v] = \text{WHITE}$ 
6         then  $BFS(G, v)$ 
```

# A better Breadth first search (BFS) algorithm

$\text{BFS}(G, s) \triangleright G = (V, E)$

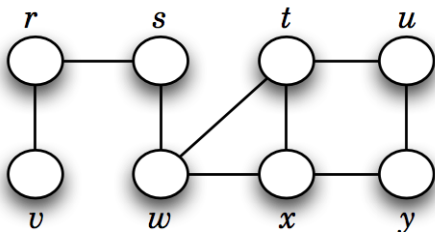
- 1  $color[s] = \text{GRAY}; \pi[s] = \text{NULL}; d[s] = 0$
- 2  $Q = \emptyset; \text{ENQUEUE}(Q, s)$

# A better Breadth first search (BFS) algorithm

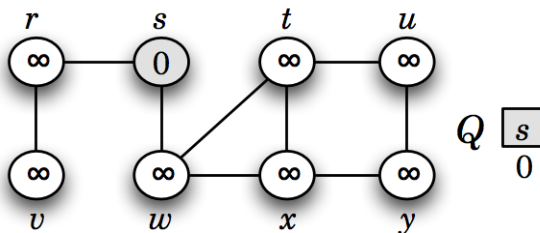
$\text{BFS}(G, s) \triangleright G = (V, E)$

```
1   $color[s] = \text{GRAY}; \pi[s] = \text{NULL}; d[s] = 0$ 
2   $Q = \emptyset; \text{ENQUEUE}(Q, s)$ 
3  while  $Q \neq \emptyset$ 
4      do  $u = \text{DEQUEUE}(Q)$ 
5          for each vertex  $v \in \text{Adj}[u]$ 
6              do if  $color[v] = \text{WHITE}$ 
7                  then  $color[v] = \text{GRAY}$ 
8                       $d[v] = d[u] + 1$ 
9                       $\pi[v] = u$ 
10                      $\text{ENQUEUE}(Q, v)$ 
11      $color[u] = \text{BLACK}$ 
```

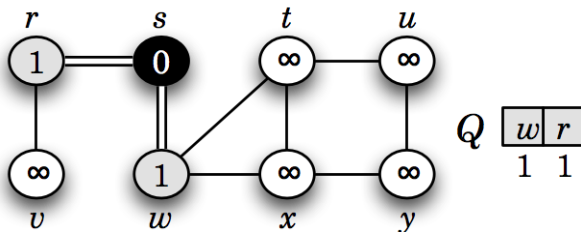
# Application of BFS



# Application of BFS

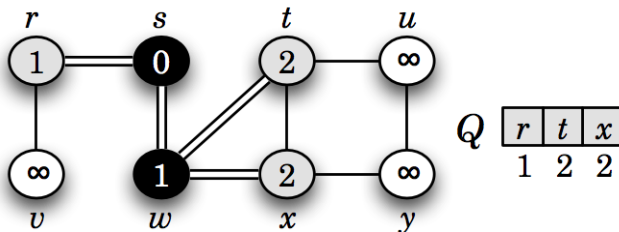


# Application of BFS

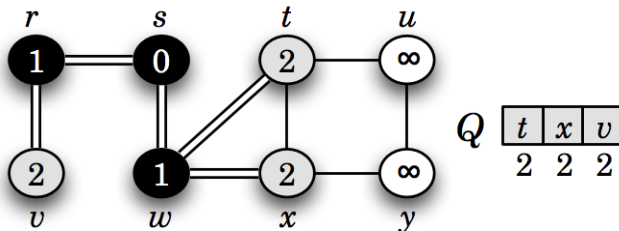




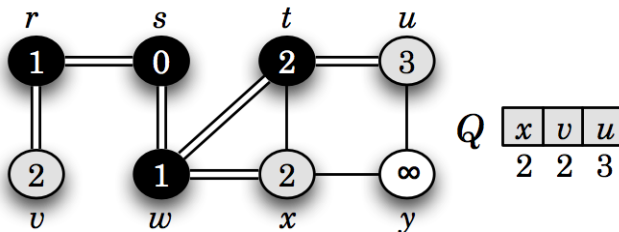
# Application of BFS



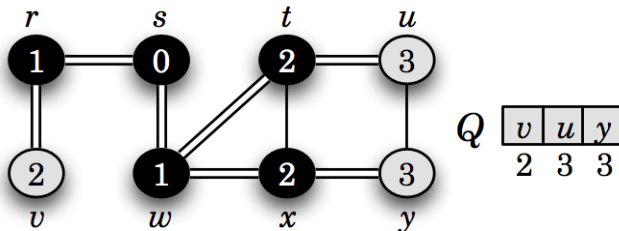
# Application of BFS



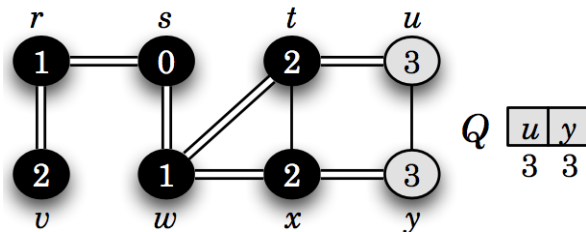
# Application of BFS



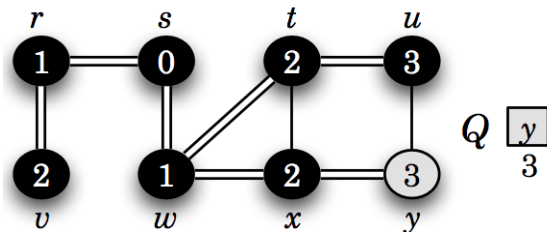
# Application of BFS



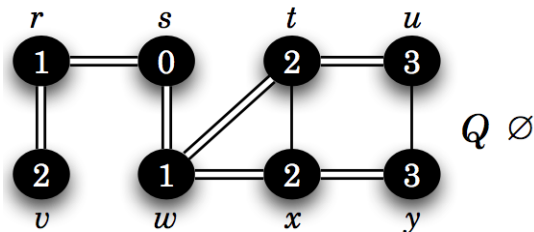
# Application of BFS



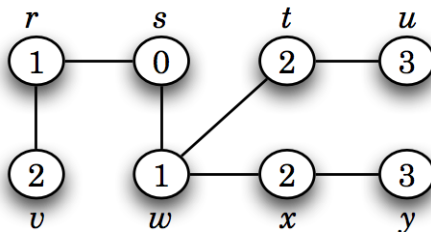
# Application of BFS



# Application of BFS

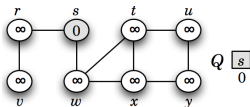


# Application of BFS

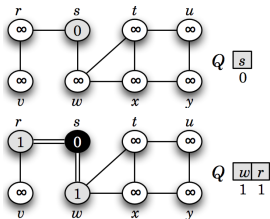




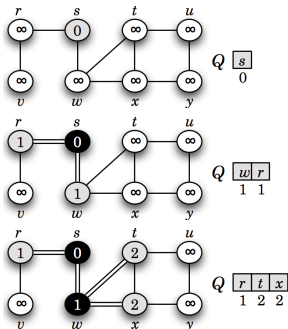
# Application of BFS



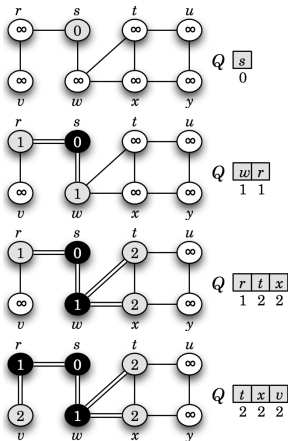
# Application of BFS



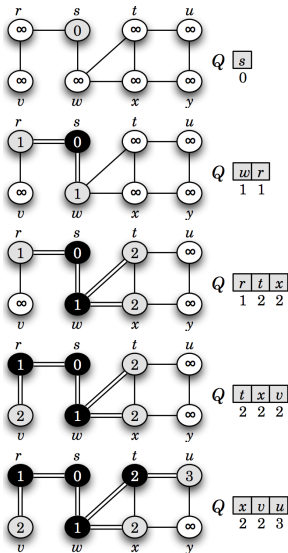
# Application of BFS



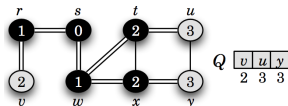
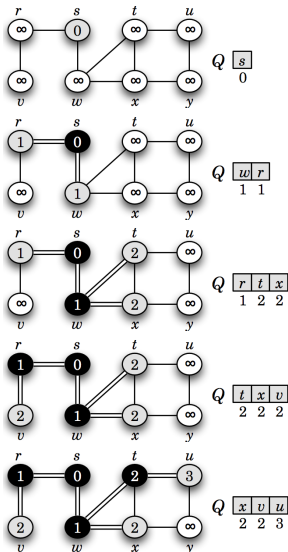
# Application of BFS



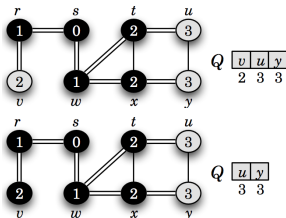
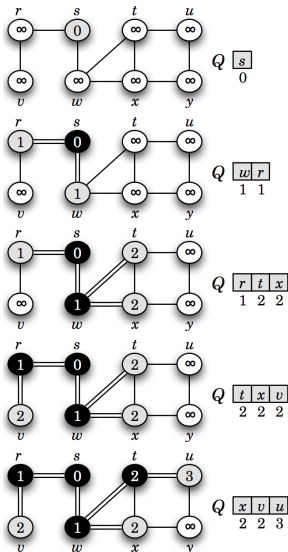
# Application of BFS



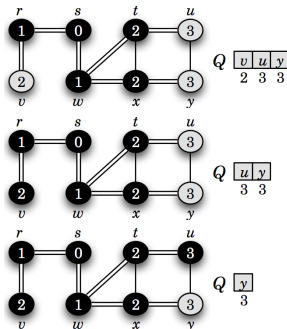
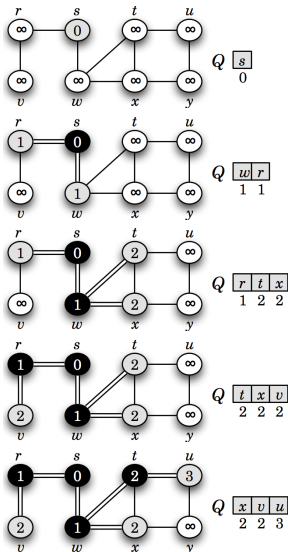
# Application of BFS



# Application of BFS

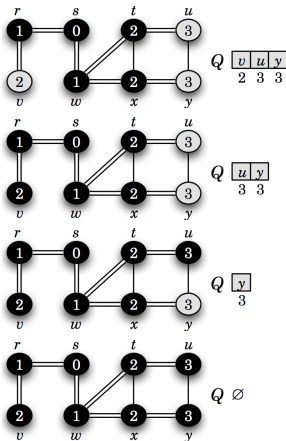
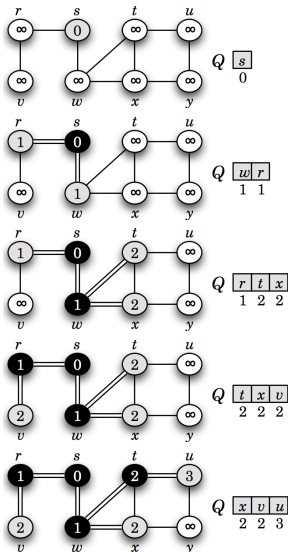


# Application of BFS

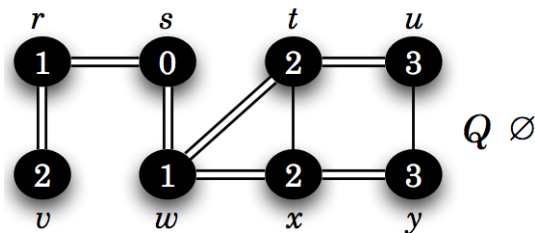
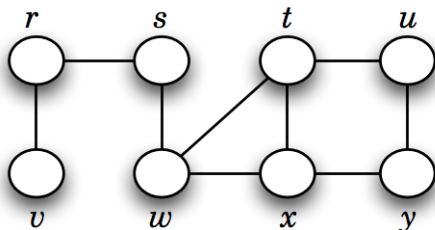




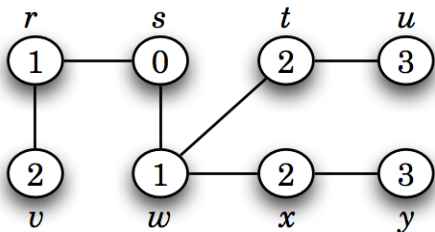
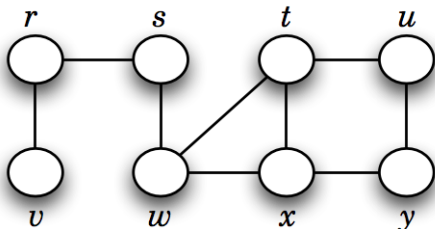
# Application of BFS



## Breadth first search (BFS) tree



# Breadth first search (BFS) tree



# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1 **color**[*v*]: the **color** of each vertex visited

# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1 **color**[*v*]: the **color** of each vertex visited
  - white: **undiscovered**

# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1 **color**[*v*]: the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing

# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1 **color**[*v*]: the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing



# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing
- 2  $\pi[v]$ : the **predecessor** pointer

# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing
- 2  $\pi[v]$ : the **predecessor** pointer
- 3  $d[v]$ : the **discovery time** (or *hops* from the start)

# Depth first search (DFS)

- Starts from an initial vertex.
- Recursively visits each adjacent vertex.
- Strategy to search **deeper** into the graph, unlike BFS which fans out.

## Information during traversal

- 1  $color[v]$ : the **color** of each vertex visited
  - white: **undiscovered**
  - gray: **discovered** but not finished processing
  - black: **finished** processing
- 2  $\pi[v]$ : the **predecessor** pointer
- 3  $d[v]$ : the **discovery time** (or *hops* from the start)
- 4  $f[v]$ : the **finish time** – when processing of  $v$  and all its descendants have finished.

# DFS algorithm

$\text{DFS}(G) \triangleright G = (V, E)$

```
1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3       $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
```

$\text{DFS-VISIT}(G, u) \triangleright G = (V, E), u \in V$

```
1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
```

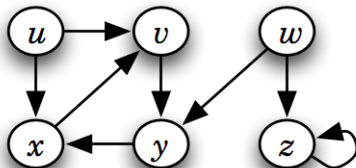
# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

```
1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
```

$\text{DFS-VISIT}(G, u)$

```
1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

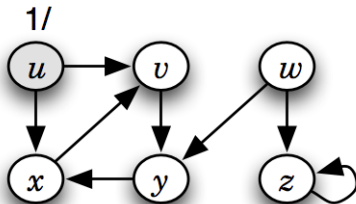
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3          $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

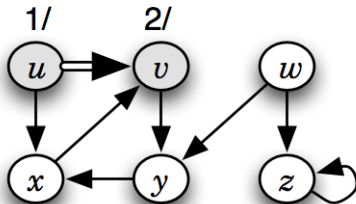
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3       $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

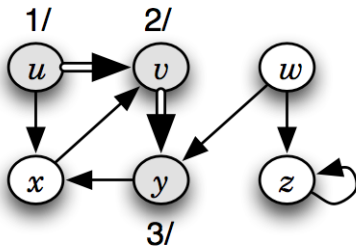
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3       $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```





# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

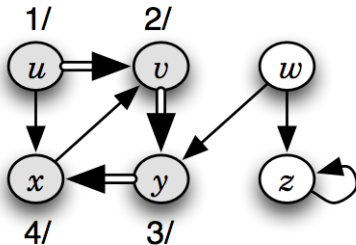
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

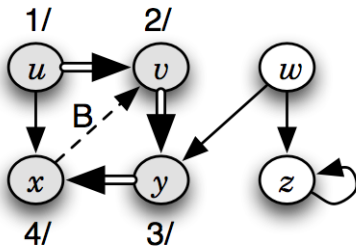
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

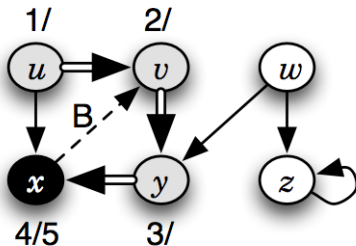
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3          $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

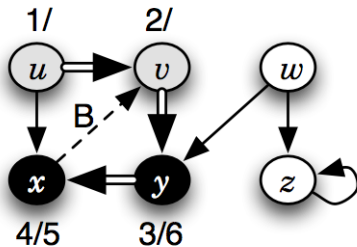
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

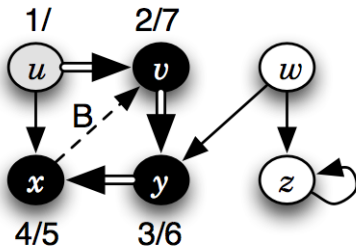
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3          $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

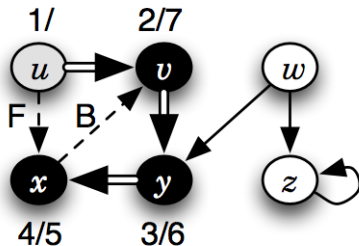
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

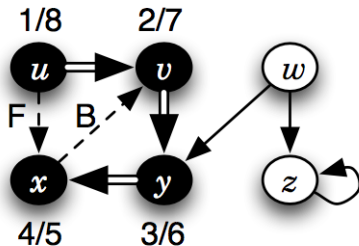
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

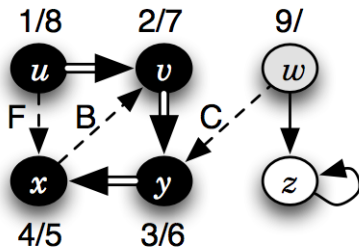
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```





# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

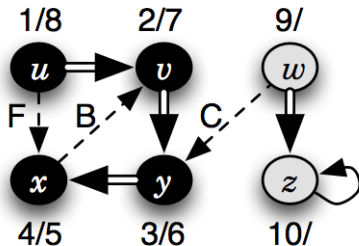
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

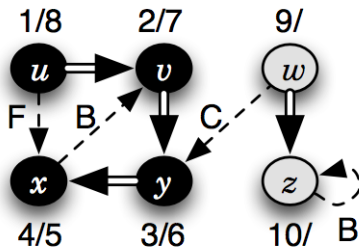
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3          $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

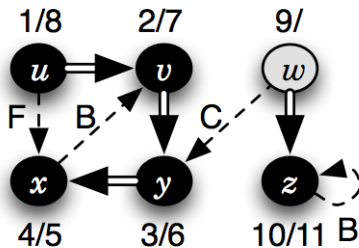
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3           $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

$\text{DFS-VISIT}(G, u)$

```

1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



# Application of DFS

$\text{DFS}(G) \triangleright G = (V, E)$

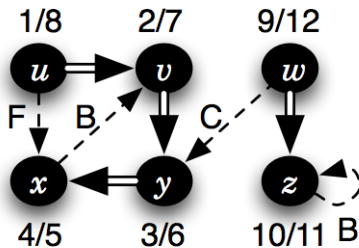
```

1  for each vertex  $u \in V[G]$ 
2      do  $\text{color}[u] \leftarrow \text{WHITE}$ 
3          $\pi[u] \leftarrow \text{NIL}$ 
4   $\text{time} \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $\text{color}[u] = \text{WHITE}$ 
7          then  $\text{DFS-VISIT}(u)$ 
  
```

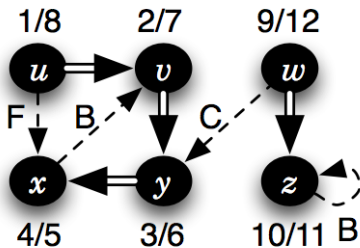
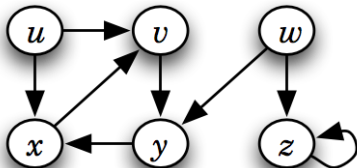
$\text{DFS-VISIT}(G, u)$

```

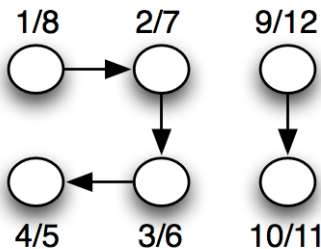
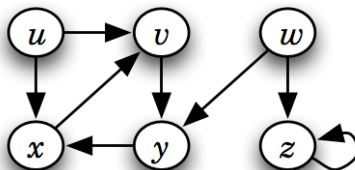
1   $\text{color}[u] \leftarrow \text{GRAY}$ 
2   $\text{time} \leftarrow \text{time} + 1$ 
3   $d[u] \leftarrow \text{time}$ 
4  for each vertex  $v \in \text{Adj}[u]$ 
5      do if  $\text{color}[v] = \text{WHITE}$ 
6          then  $\pi[v] \leftarrow u$ 
7               $\text{DFS-VISIT}(v)$ 
8   $\text{color}[u] \leftarrow \text{BLACK}$ 
9   $f[u] \leftarrow \text{time} + 1$ 
  
```



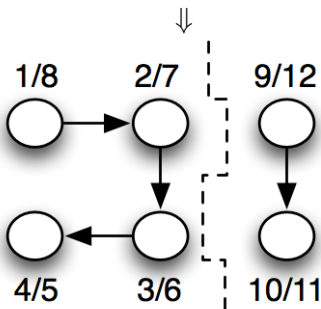
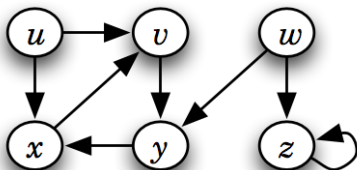
# Breadth first search (DFS) tree (or, forest)



# Breadth first search (DFS) tree (or, forest)



# Breadth first search (DFS) tree (or, forest)



# Questions to answer using DFS

- How to detect a cycle in the graph?



# Questions to answer using DFS

- How to detect a cycle in the graph?
- How to compute the connected components? (Hint: line 7 in the  $\text{DFS}(G)$  starts a new tree, with each tree representing a connected component)

# Questions to answer using DFS

- How to detect a cycle in the graph?
- How to compute the connected components? (Hint: line 7 in the  $\text{DFS}(G)$  starts a new tree, with each tree representing a connected component)
- Of what use are the *discovery* and *finish* times?

# Questions to answer using DFS

- How to detect a cycle in the graph?
- How to compute the connected components? (Hint: line 7 in the  $\text{DFS}(G)$  starts a new tree, with each tree representing a connected component)
- Of what use are the *discovery* and *finish* times?
- What are the different types of *non-tree* edges in a BFS or DFS tree of an undirected graph? What about a directed graph?

# Questions to answer using DFS

- How to detect a cycle in the graph?
- How to compute the connected components? (Hint: line 7 in the  $\text{DFS}(G)$  starts a new tree, with each tree representing a connected component)
- Of what use are the *discovery* and *finish* times?
- What are the different types of *non-tree* edges in a BFS or DFS tree of an undirected graph? What about a directed graph?
- How to use DFS to *topologically sort* a directed acyclic graph (DAG)?

# Questions to answer using DFS

- How to detect a cycle in the graph?
- How to compute the connected components? (Hint: line 7 in the  $\text{DFS}(G)$  starts a new tree, with each tree representing a connected component)
- Of what use are the *discovery* and *finish* times?
- What are the different types of *non-tree* edges in a BFS or DFS tree of an undirected graph? What about a directed graph?
- How to use DFS to *topologically sort* a directed acyclic graph (DAG)?
- ...