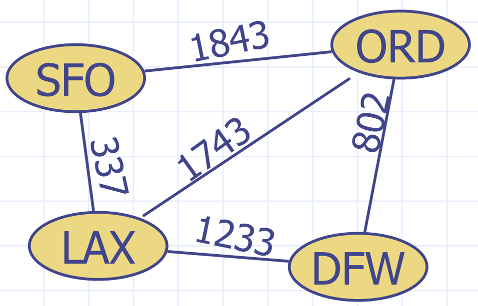
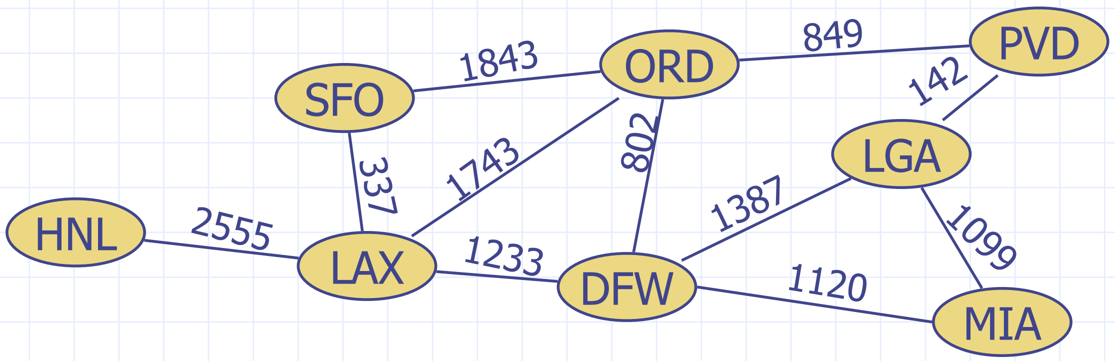
Graphs



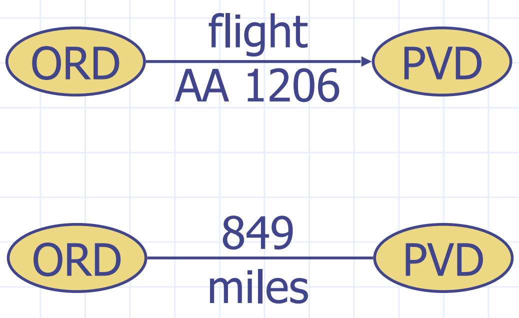
Graphs

- A graph is a pair (V, E), where
  - V is a set of nodes, called vertices
  - E is a collection of pairs of vertices, called edges
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route



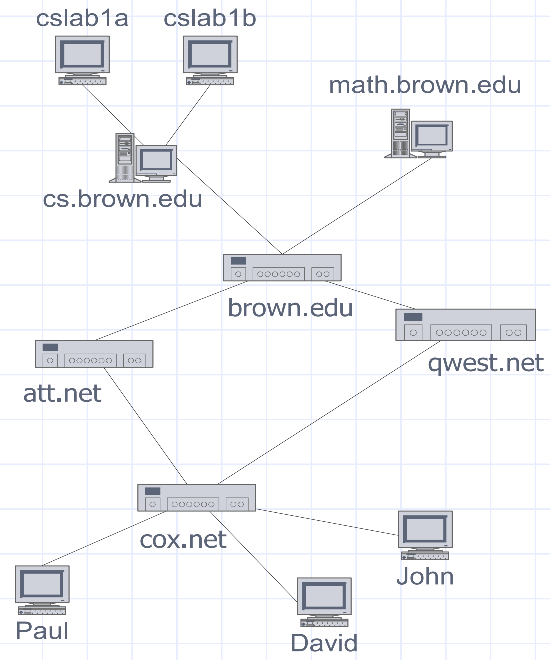
Edge Types

- Directed edge
  - ordered pair of vertices (u,v)
  - first vertex u is the origin
  - second vertex v is the destination
  - e.g., a flight
- Undirected edge
  - unordered pair of vertices (u,v)
  - e.g., a flight route
- Directed graph
  - all the edges are directed
  - e.g., route network
- Undirected graph
  - all the edges are undirected
  - e.g., flight network



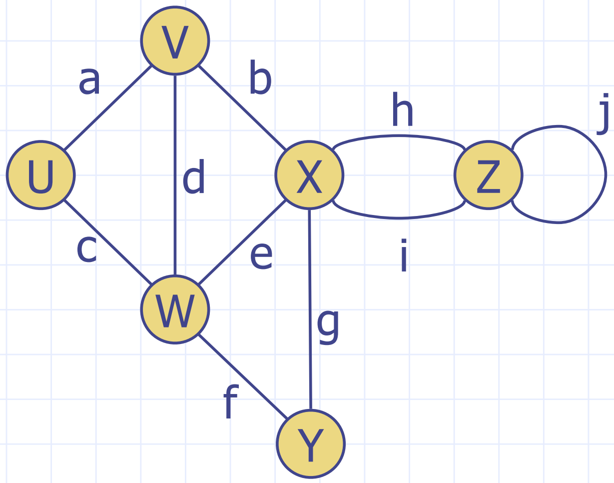
Applications

- Electronic circuits
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  - Highway network
  - Flight network
- Computer networks
  - Local area network
  - Internet
  - Web
- Databases
  - Entity-relationship diagram



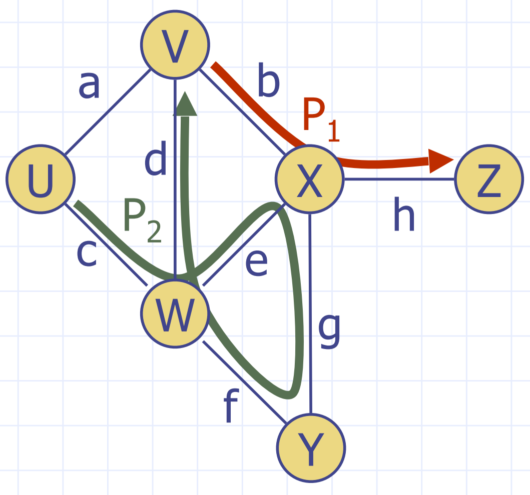
Terminology

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a
- Edges incident on a vertex
  - a, d, and b are incident on V
- Adjacent vertices
  - U and V are adjacent
- Degree of a vertex
  - X has degree 5
- Parallel edges
  - h and i are parallel edges
- Self-loop
  - j is a self-loop



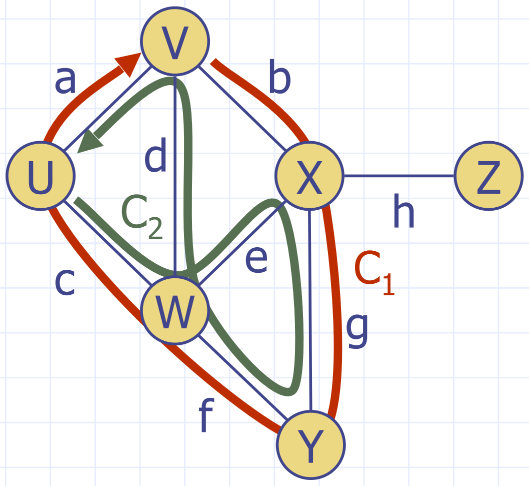
Terminology (cont.)

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- Simple path
  - path such that all its vertices and edges are distinct
- Examples
  - P_1=(V,b,X,h,Z) is a simple path
  - P_2=(U,c,W,e,X,g,Y,f,W,d,V) is a path that is not simple
  - Importance of distinction ?



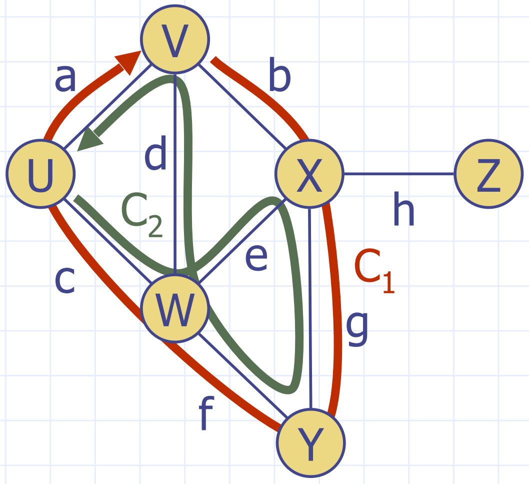
Terminology (cont.)

- Cycle
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints
  - First and last vertex is same.



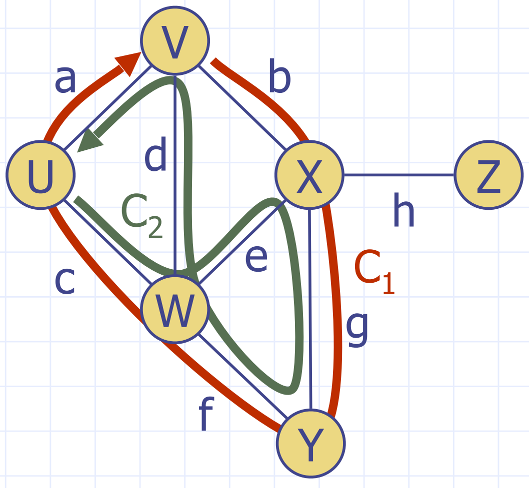
Terminology (cont.)

- Simple cycle
  - cycle such that all its vertices and edges are distinct
  - a cycle in an undirected graph may not have repeated edges why ?
  - What about directed graph ?
- Examples
  - C_1=(V,b,X,g,Y,f,W,c,U,a,↵) is a simple cycle
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,↵) is a cycle that is not simple



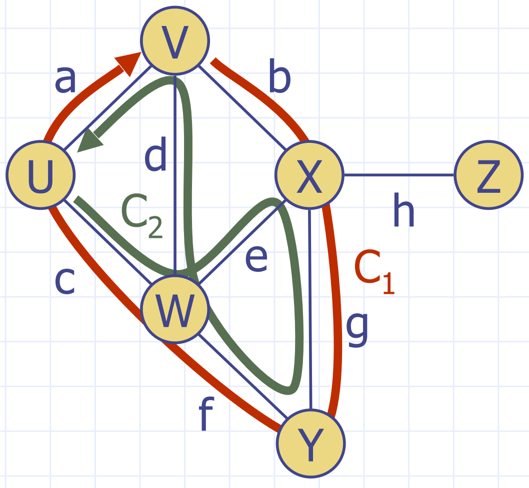
Terminology (cont.)

- The length of a path or a cycle is its number of edges.
- We say that one vertex is connected to another if there exists a path that contains both of them.
- A graph is connected if there is a path from every vertex to every other vertex.
- An acyclic graph is a graph with no cycles.
- Example of a graph that is acyclic ??



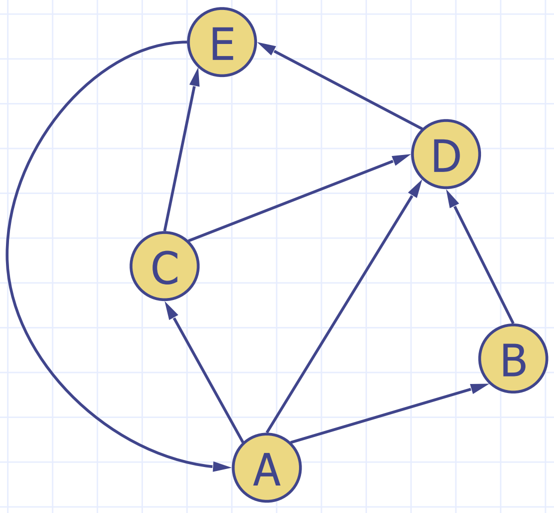
Terminology (cont.)

- A tree is an acyclic connected graph.
- A forest is a disjoint set of trees.



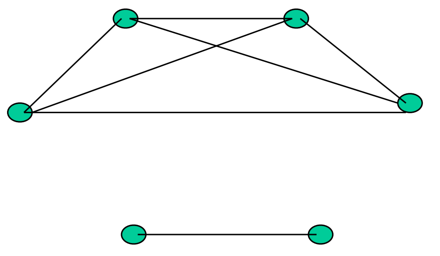
Digraphs

- A digraph is a graph whose edges are all directed
  - Short for "directed graph"
- Applications
  - one-way streets
  - flights
  - task scheduling

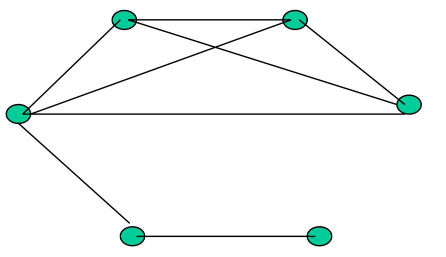


Graph Connectivity

- An undirected graph is said to be *connected* if there is a path between every pair of nodes. Otherwise, the graph is *disconnected*
- Informally, an undirected graph is connected if it hangs in one piece



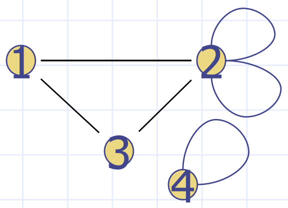
Disconnected

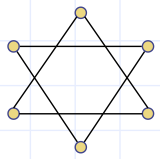


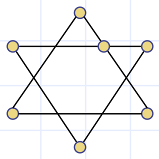
Connected

Connectivity

Q: Which of the following graphs are connected?

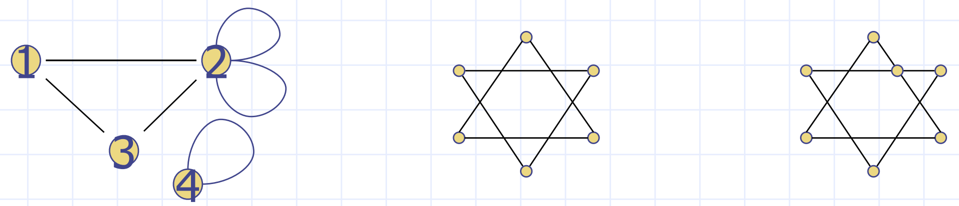




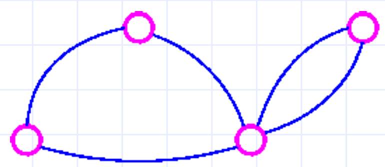


Connectivity

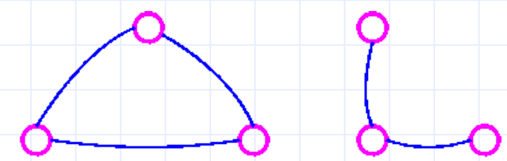
A: First and second are disconnected. Last is connected.



Connectivity



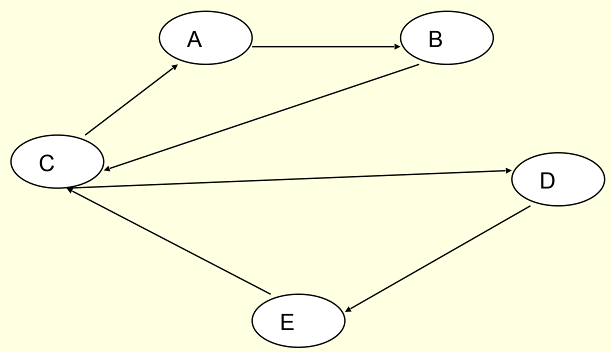
Connected Non–Simple Graph



Disconnected Simple Graph

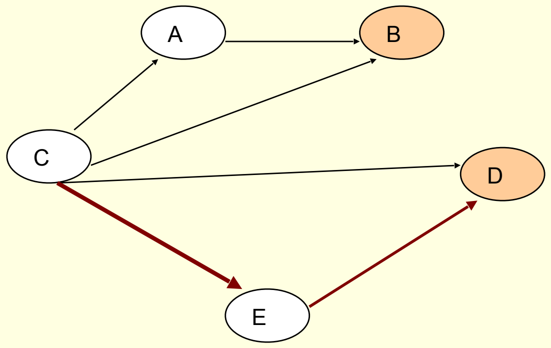
Connectivity in Directed Graphs (I)

Definition: A directed graph is said to be
strongly connected
if for any pair of nodes there is a path
from each one to the other



Connectivity in Directed Graphs (II)

Definition: A directed graph is said to be **weakly connected** if the underlying undirected graph is connected

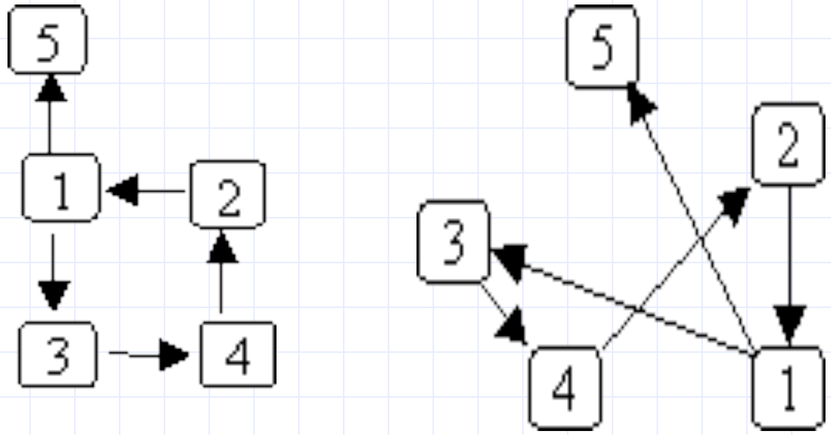


Each unilaterally connected graph is also weakly connected

There is no path between B and D

Directed Graphs

- Are these two graphs same ?



Properties

Property 1

$\sum_v \deg(v) = 2m$

Proof: each edge is counted twice

Property 2

In an undirected graph with no self-loops and no multiple edges

$m \leq n(n-1)/2$

Proof: each vertex has degree at most $(n-1)$

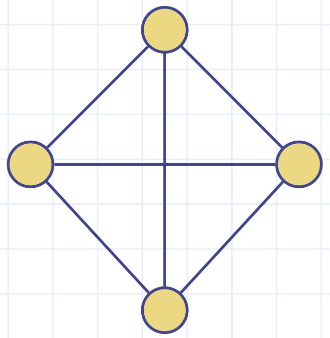
What is the bound for a directed graph?

Notation

n number of vertices

m number of edges

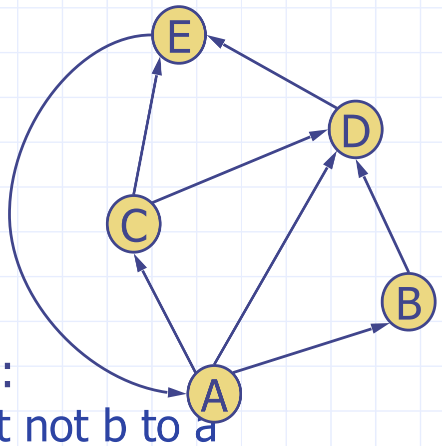
$\deg(v)$ degree of vertex v



Example

- $n = 4$
- $m = 6$
- $\deg(v) = 3$

Digraph Properties



- A graph G=(V,E) such that
  - Each edge goes in one direction:
  - Edge (a,b) goes from a to b, but not b to a
- If G is simple, $m \le n \cdot (n - 1)$
- If we keep in-edges and out-edges in separate adjacency lists, we can perform listing of incoming edges and outgoing edges in time proportional to their size

Main Methods of the Graph ADT

- Vertices and edges
  - are positions
  - store elements
- Accessor methods
  - e.endVertices(): a list of the two endvertices of e
  - e.opposite(v): the vertex opposite of v on e
  - u.isAdjacentTo(v): true iff u and v are adjacent
  - *v: reference to element associated with vertex v
  - *e: reference to element associated with edge e
- Update methods
  - insertVertex(o): insert a vertex storing element o
  - insertEdge(v, w, o): insert an edge (v,w) storing element o
  - eraseVertex(v): remove vertex v (and its incident edges)
  - eraseEdge(e): remove edge e
- Iterable collection methods
  - incidentEdges(v): list of edges incident to v
  - vertices(): list of all vertices in the graph
  - edges(): list of all edges in the graph

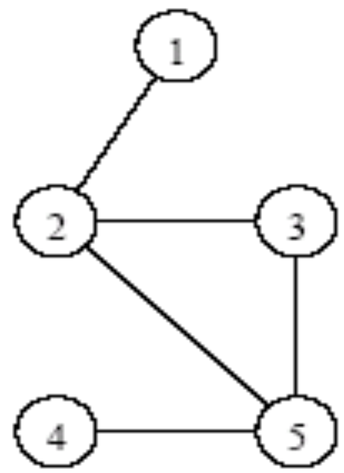
Graph Representation

- For graphs to be computationally useful, they have to be conveniently represented in programs
- There are two computer representations of graphs:
  - Adjacency matrix representation
  - Adjacency lists representation

Adjacency Matrix Representation

- In this representation, each graph of n nodes is represented by an n x n matrix A, that is, a two-dimensional array A
- The nodes are (re)-labeled 1,2,…,n
- A[i][j] = 1 if (i,j) is an edge
- A[i][j] = 0 if (i,j) is not an edge

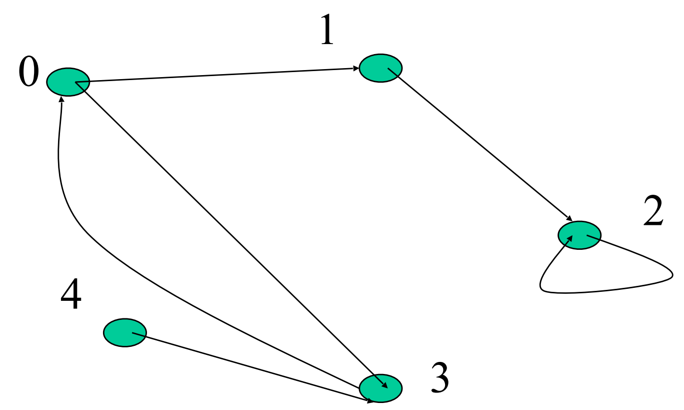
Adjacency Matrix – undirected graph



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

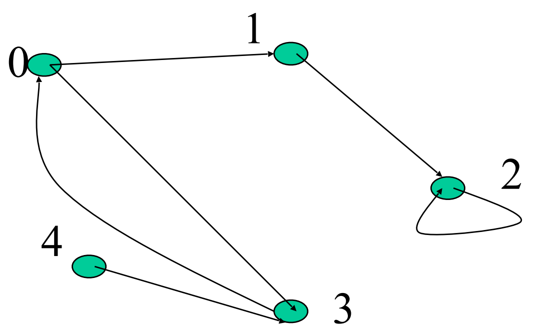
An undirected graph and its adjacency matrix representation.

Example of Adjacency Matrix
Directed graph ??



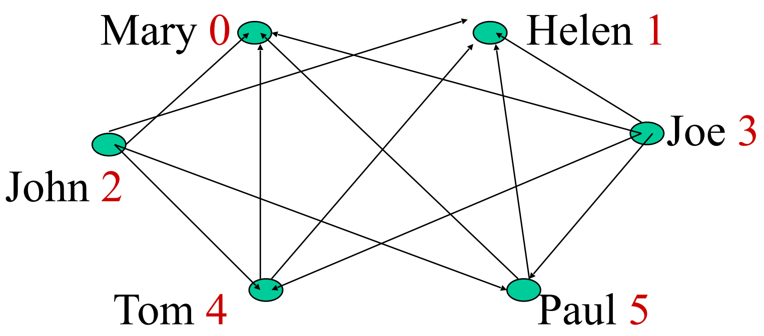
Example of Adjacency Matrix Directed graph

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Another Example of Adj. Matrix

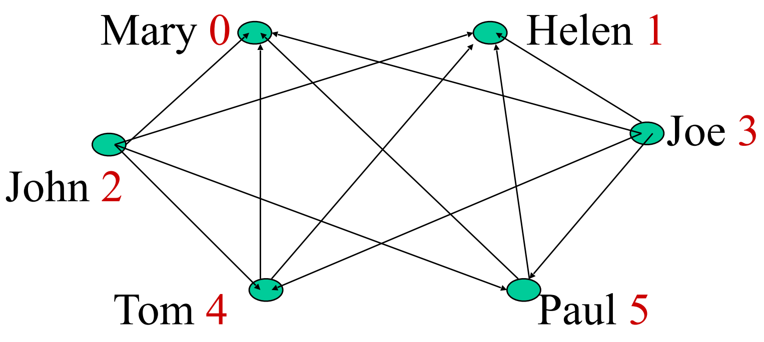
- Adj Matrix



Another Example of Adj. Matrix

- Re-label the nodes with numerical labels

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Representing Graphs

Representing Graphs

Definition: Let $G = (V, E)$ be an undirected graph with $|V| = n$. Suppose that the vertices and edges of G are listed in arbitrary order as $v_1, v_2, \ldots, v_n$ and $e_1, e_2, \ldots, e_m$, respectively.

Representing Graphs

Definition: Let $G = (V, E)$ be an undirected graph with $|V| = n$. Suppose that the vertices and edges of G are listed in arbitrary order as $v_1, v_2, \ldots, v_n$ and $e_1, e_2, \ldots, e_m$, respectively.

The **incidence matrix** of G with respect to this listing of the vertices and edges is the $n \times m$ zero-one matrix with 1 as its (i, j) entry when edge e_j is incident with v_i, and 0 otherwise.

Representing Graphs

Definition: Let $G = (V, E)$ be an undirected graph with $|V| = n$. Suppose that the vertices and edges of G are listed in arbitrary order as $v_1, v_2, \ldots, v_n$ and $e_1, e_2, \ldots, e_m$, respectively.

The **incidence matrix** of G with respect to this listing of the vertices and edges is the $n \times m$ zero-one matrix with 1 as its (i, j) entry when edge e_j is incident with v_i, and 0 otherwise.

In other words, for an incidence matrix $M = [m_{ij}]$,

Representing Graphs

Definition: Let $G = (V, E)$ be an undirected graph with $|V| = n$. Suppose that the vertices and edges of G are listed in arbitrary order as $v_1, v_2, \ldots, v_n$ and $e_1, e_2, \ldots, e_m$, respectively.

The **incidence matrix** of G with respect to this listing of the vertices and edges is the $n \times m$ zero-one matrix with 1 as its (i, j) entry when edge e_j is incident with v_i, and 0 otherwise.

In other words, for an incidence matrix $M = [m_{ij}]$,

$m_{ij} = 1$ if edge e_j is incident with v_i

$m_{ij} = 0$ otherwise.

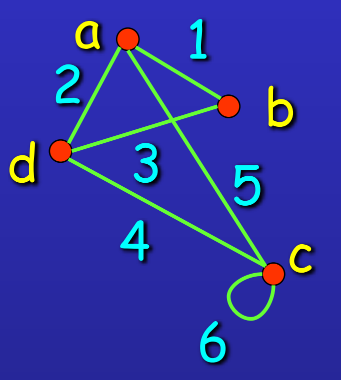
Representing Graphs

Representing Graphs

Example: What is the incidence matrix M for the following graph G based on the order of vertices a, b, c, d and edges 1, 2, 3, 4, 5, 6?

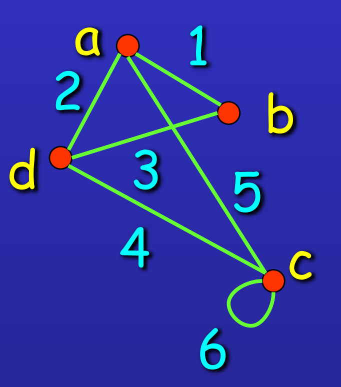
Representing Graphs

Example: What is the incidence matrix M for the following graph G based on the order of vertices a, b, c, d and edges 1, 2, 3, 4, 5, 6?



Representing Graphs

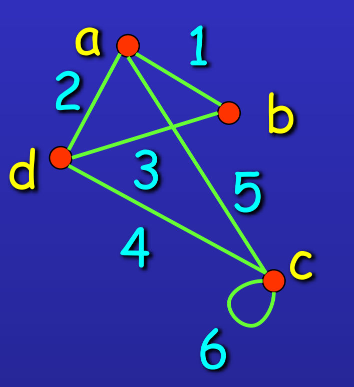
Example: What is the incidence matrix M for the following graph G based on the order of vertices a, b, c, d and edges 1, 2, 3, 4, 5, 6?



Solution:

Representing Graphs

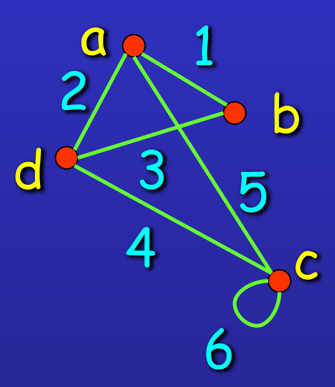
Example: What is the incidence matrix M for the following graph G based on the order of vertices a, b, c, d and edges 1, 2, 3, 4, 5, 6?



Solution: $$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Representing Graphs

Example: What is the incidence matrix M for the following graph G based on the order of vertices a, b, c, d and edges 1, 2, 3, 4, 5, 6?



Solution:

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Note: Incidence matrices of undirected graphs contain two 1s per column for edges connecting two vertices and one 1 per column for loops.

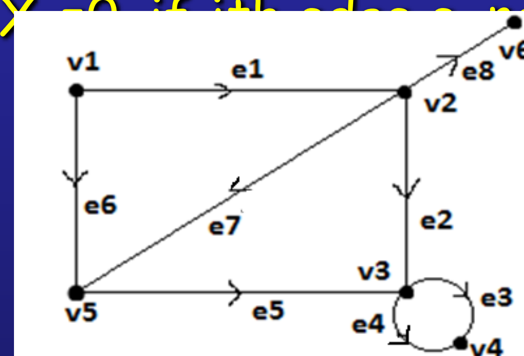
Incident matrix of diagraph

Given a graph G with n, e & no self loops is matrix $x(G)=[X_{ij}]$ or order n*e where n vertices are rows & e edges are columns such that,

$X_{ij}=1$, if jth edge e_j is incident out i^{th} vertex v_i

$X_{ij}=-1$, if jth edge e_j is incident into i^{th} vertex v_i

$X_{ij}=0$, if jth edge e_j not incident on i^{th} vertex v_i.

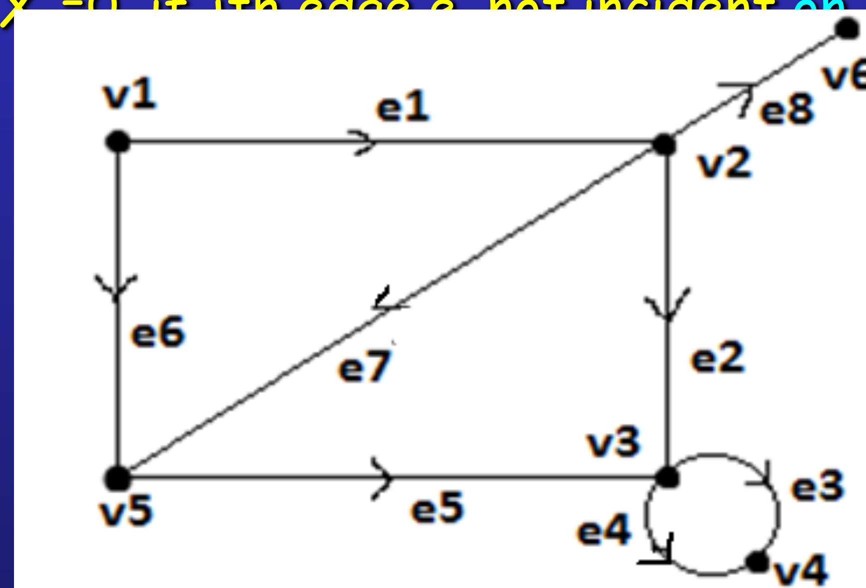


Incident matrix of diagraph

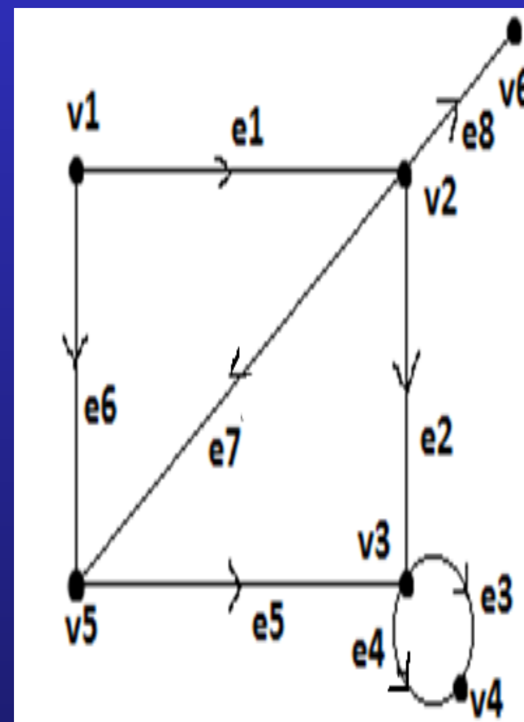
X_{ij}=1, if jth edge e_j is incident out i^{th} vertex v_i

X_{ij}=-1, if jth edge e_j is incident into i^{th} vertex v_i

X_{ij}=0, if jth edge e_j not incident on i^{th} vertex v_i.



Incident matrix of diagraph



	e1	e2	e3	e4	e5	e6	e7	e8
v1	1	0	0	0	0	1	0	0
v2	-1	1	0	0	0	0	1	1
v3	0	-1	1	1	-1	0	0	0
v4	0	0	-1	-1	0	0	0	0
v5	0	0	0	0	1	-1	-1	0
v6	0	0	0	0	0	0	0	-1

Adjacency Matrices

- Can you determine if it is a directed graph ?

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Pros and Cons of Adjacency Matrices

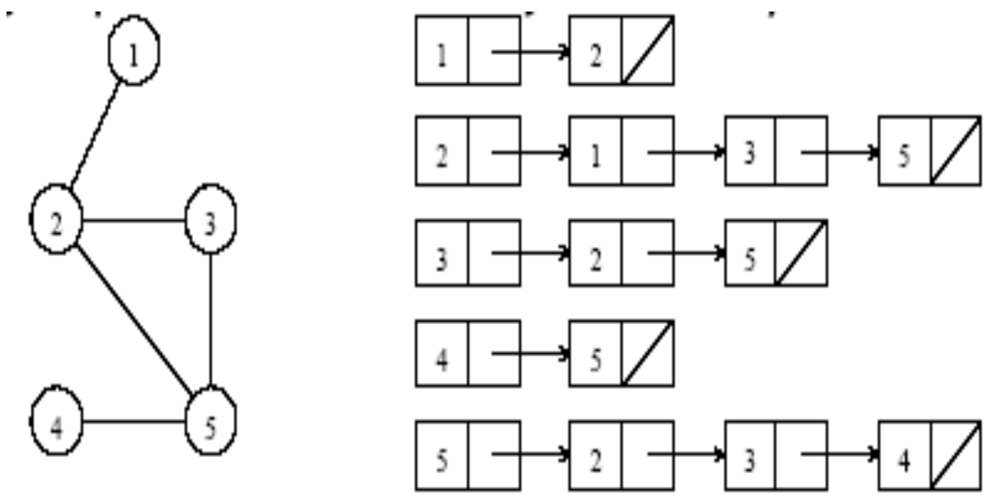
- Pros:
  - Simple to implement
  - Easy and fast to tell if a pair (i,j) is an edge: simply check if A[i][j] is 1 or 0
- Cons:
  - No matter how few edges the graph has, the matrix takes $O(n^2)$ in memory

Adjacency Lists Representation

- A graph of n nodes is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent from node i.
  - The nodes in the list L[i] are in no particular order

Adjacency list – undirectd graph

An undirected graph and its adjacency list representation.



Example of Linked Representation directed graph

L[0]: empty

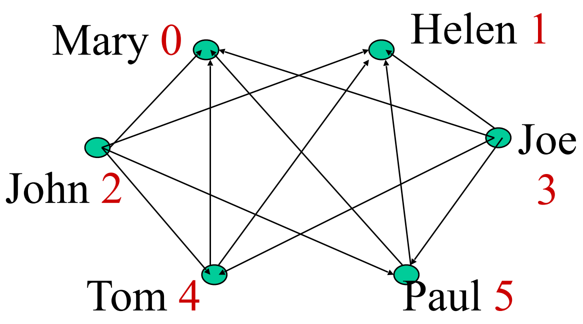
L[1]: empty

L[2]: 0, 1, 4, 5

L[3]: 0, 1, 4, 5

L[4]: 0, 1

L[5]: 0, 1

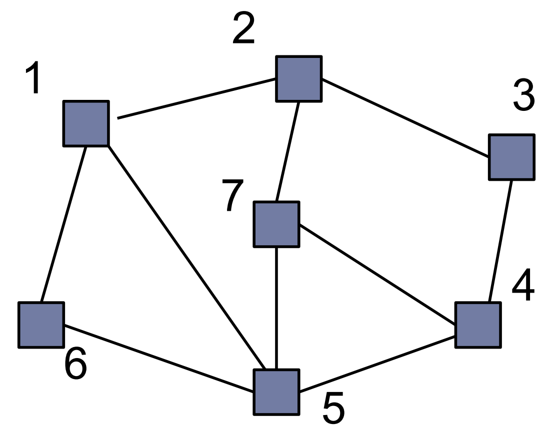


Pros and Cons of Adjacency Lists

- Pros:
  - Saves on space (memory): the representation takes as many memory words as there are nodes and edge.
- Cons:
  - It can take up to O(n) time to determine if a pair of nodes (i,j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].

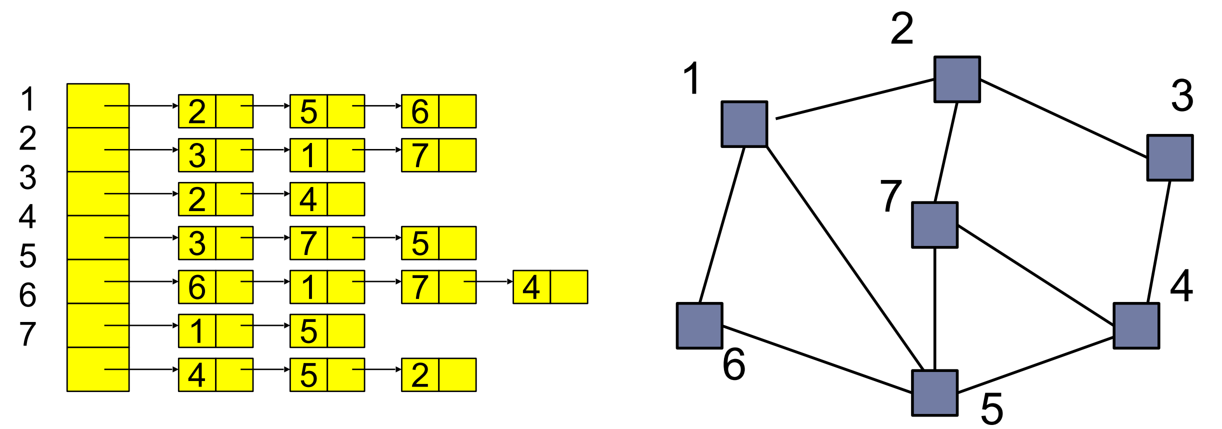
Adjacency List Example

- What is the adjacency list for the following graph

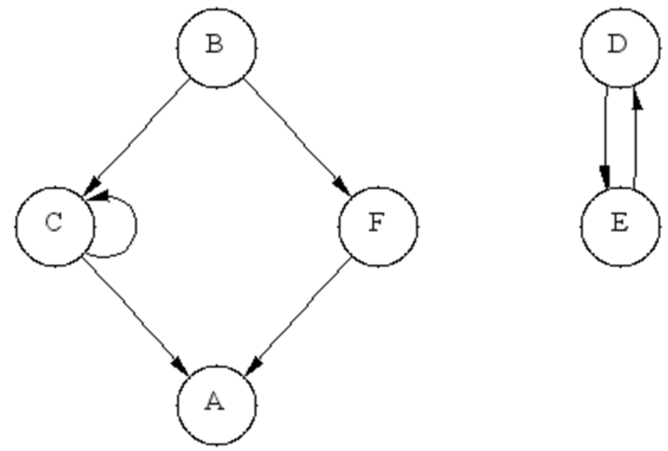


Adjacency List Example

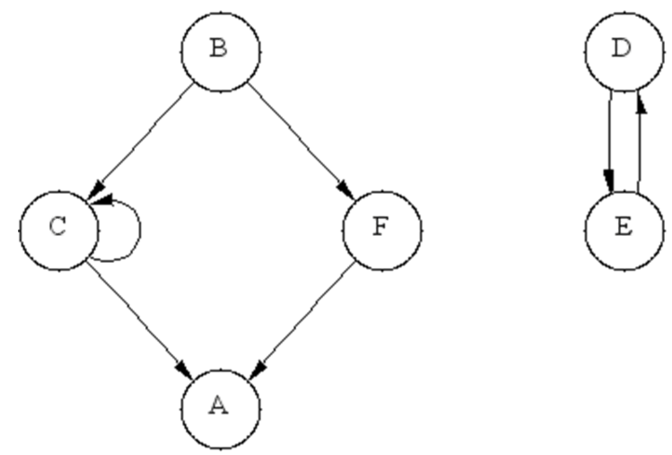
Adjacency list.

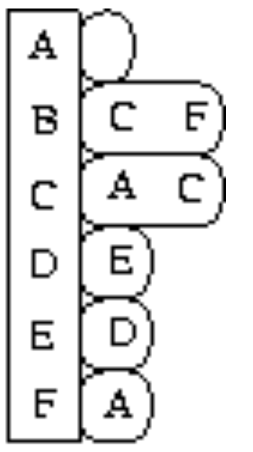


- What is the adjacency list of the following directed graph



- What is the adjacency list of the following directed graph





EXERCISE 3.3

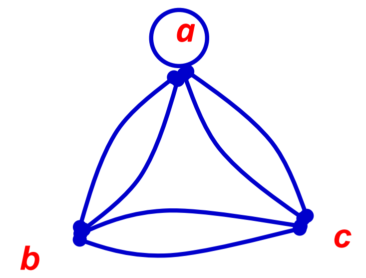
3. Draw a graph with the given adjacency matrix

a)

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

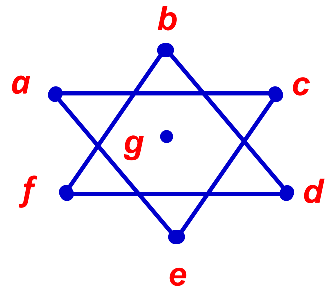
EXERCISE

1. For the following graph,
   - Find the degree of each vertex



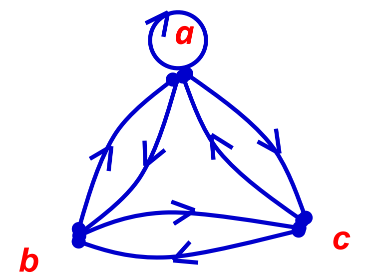
EXERCISE

1. For the following graph,
   - Find the degree of each vertex



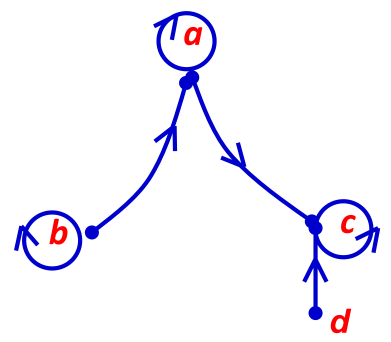
EXERCISE

2. For the following directed graph,
    - Find the in-degree and out-degree of each vertex



EXERCISE

2. For the following directed graph,
   - Find the in-degree and out-degree of each vertex

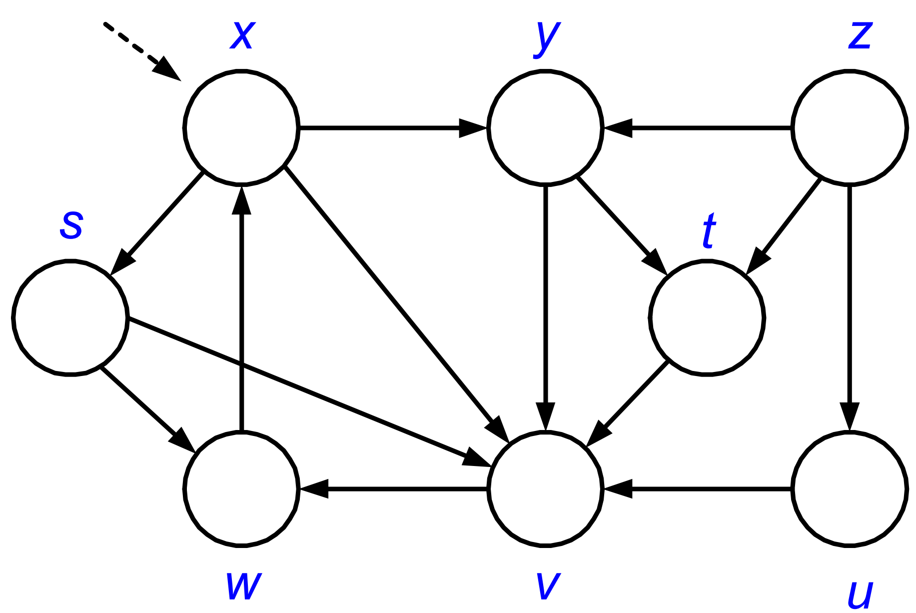


- A program to create the adjacency matrix for a graph

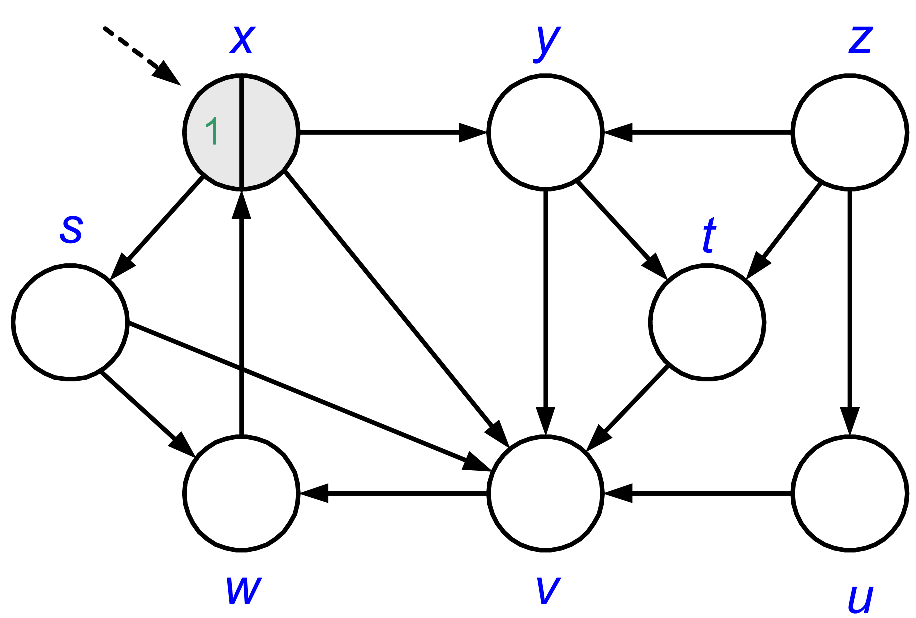
Graph Traversals

Depth-First Search
Breadth First Search

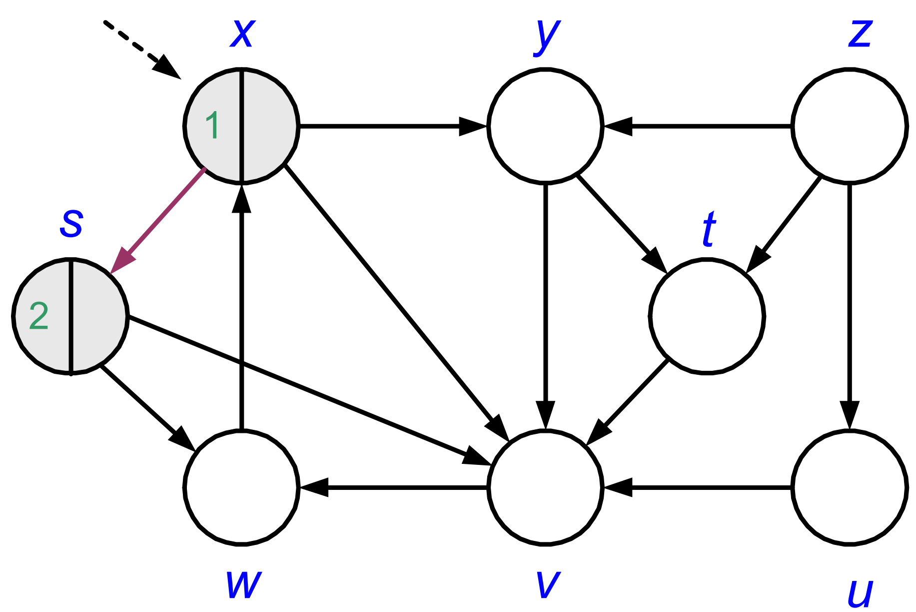
Depth-First Search: Example



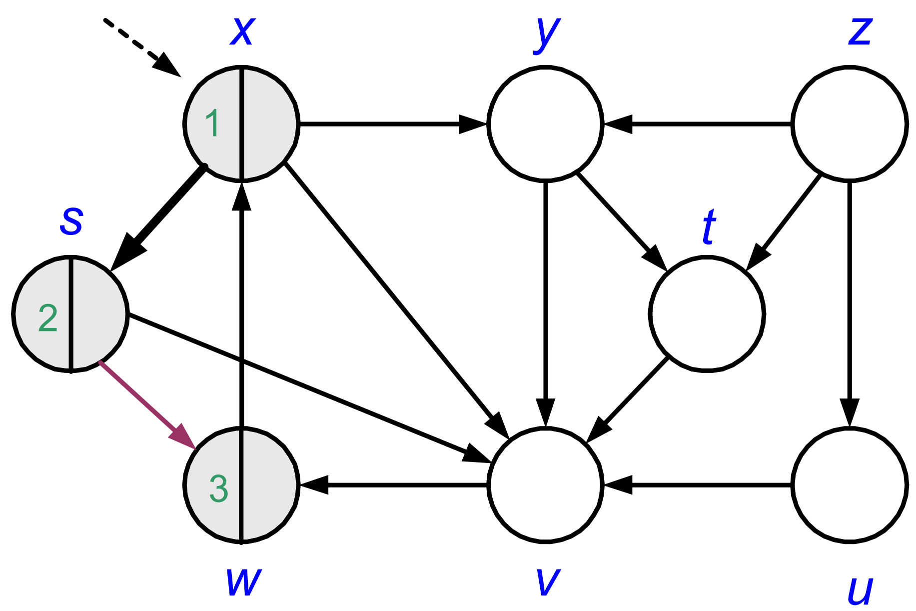
Depth-First Search: Example



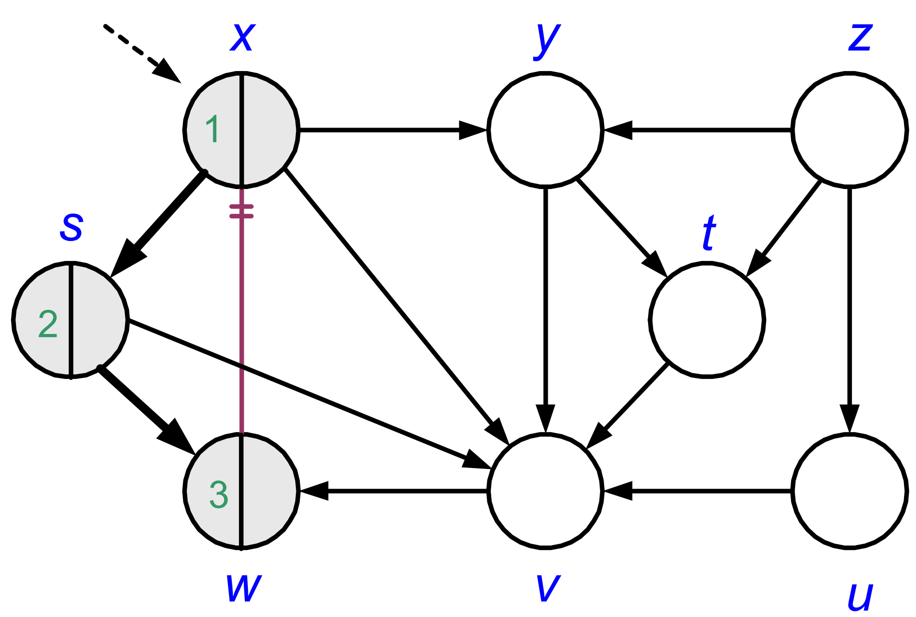
Depth-First Search: Example



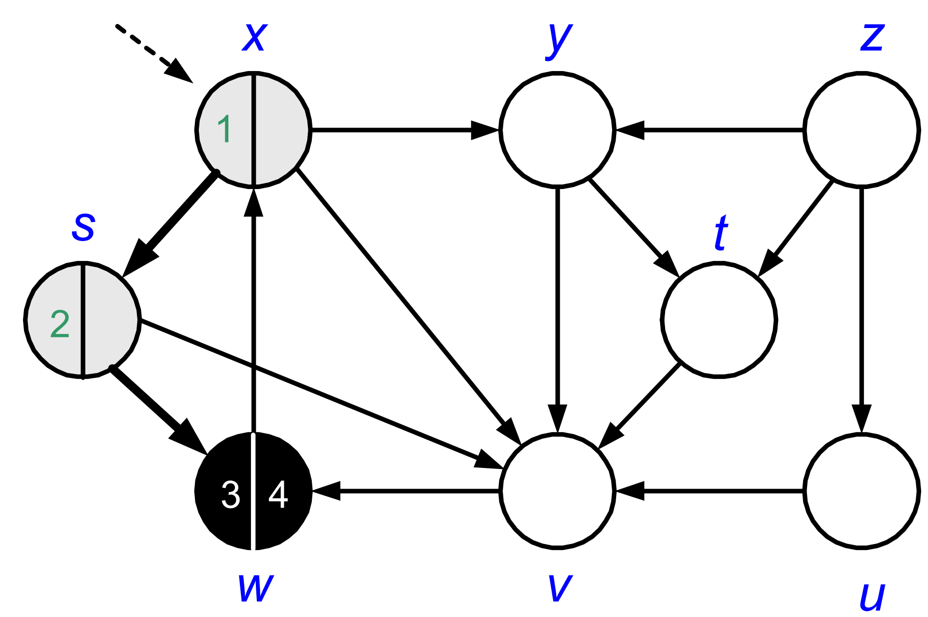
Depth-First Search: Example



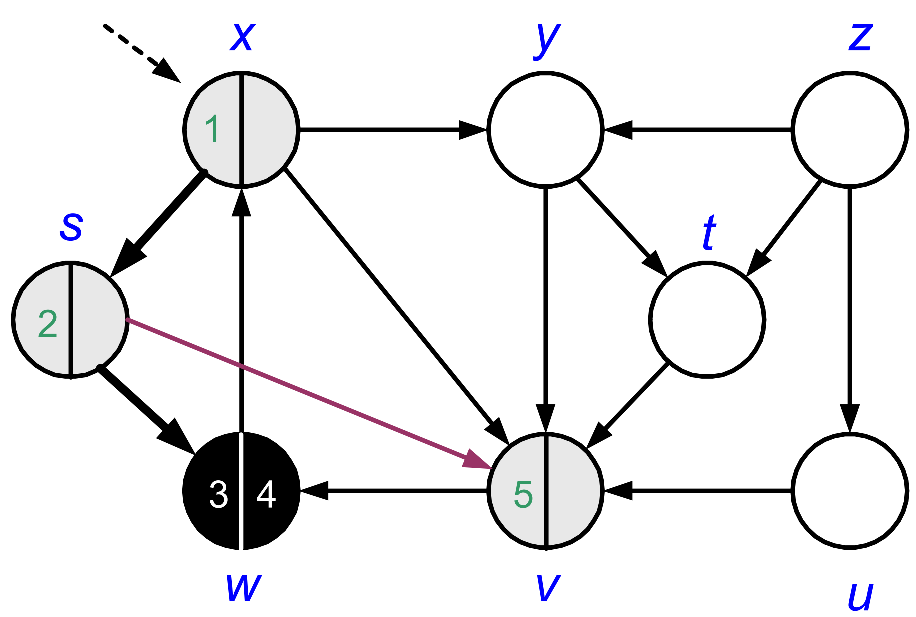
Depth-First Search: Example



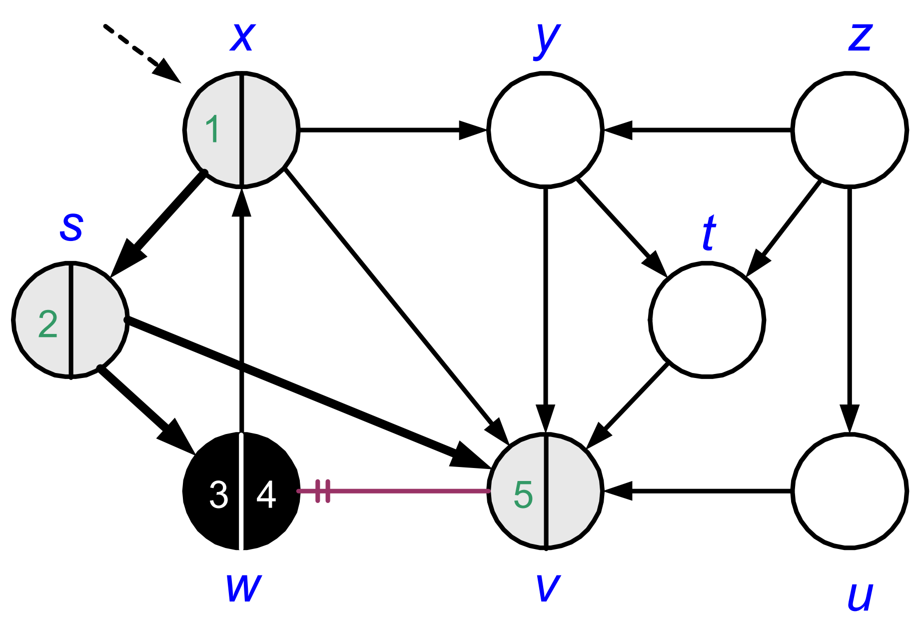
Depth-First Search: Example



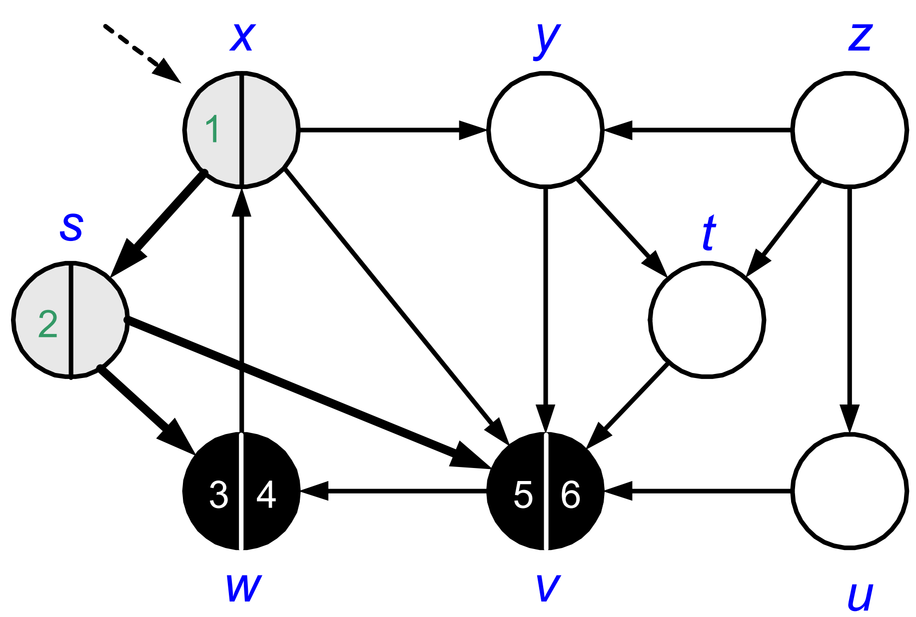
Depth-First Search: Example



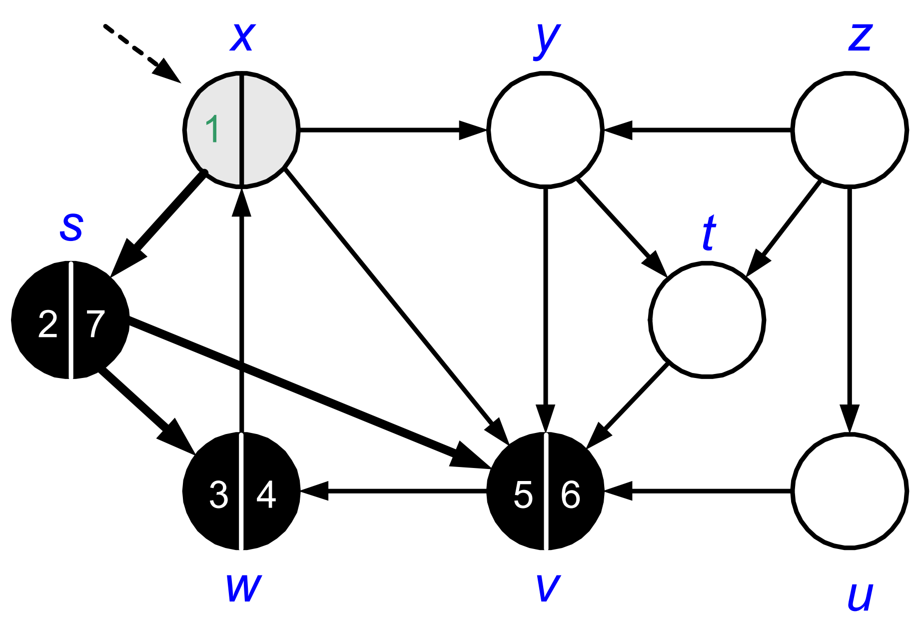
Depth-First Search: Example



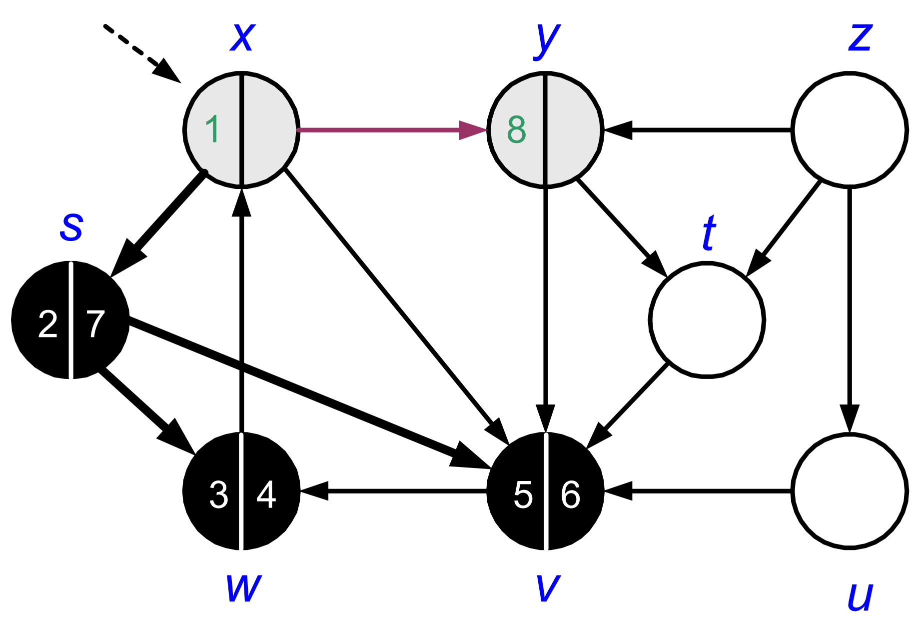
Depth-First Search: Example



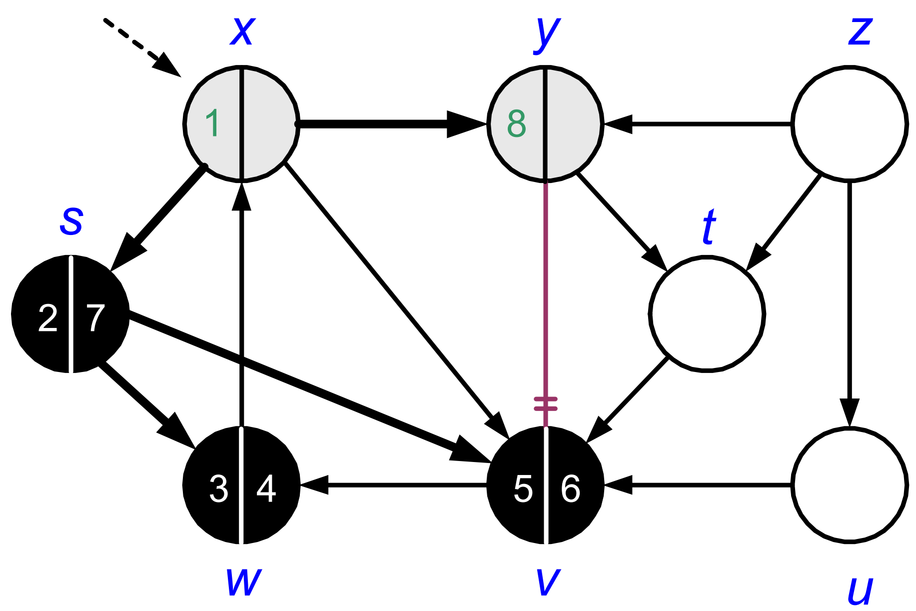
Depth-First Search: Example



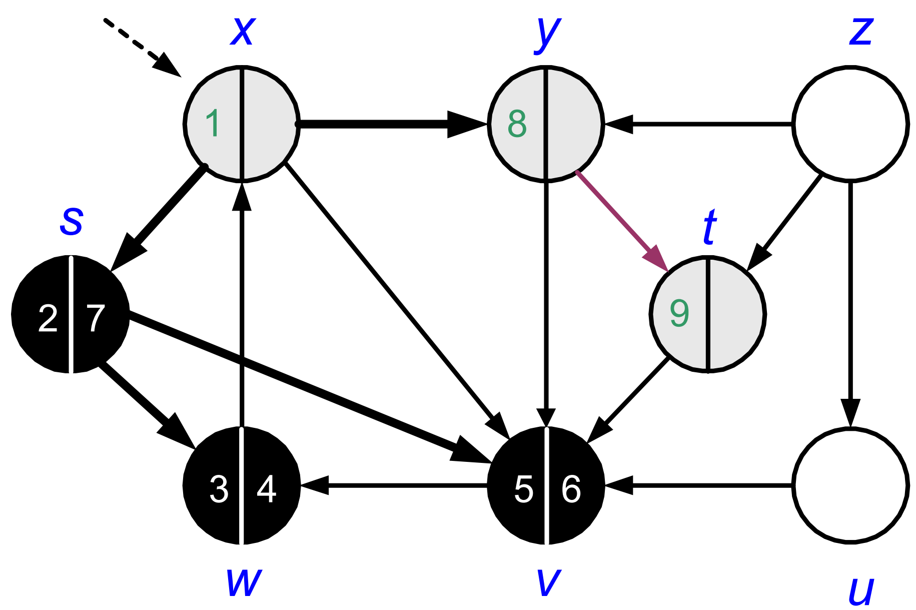
Depth-First Search: Example



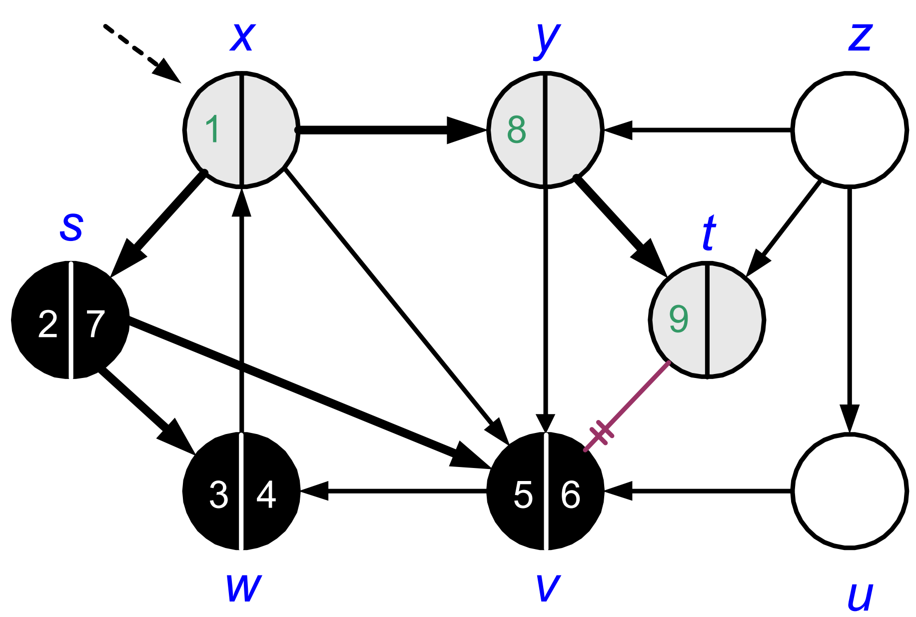
Depth-First Search: Example



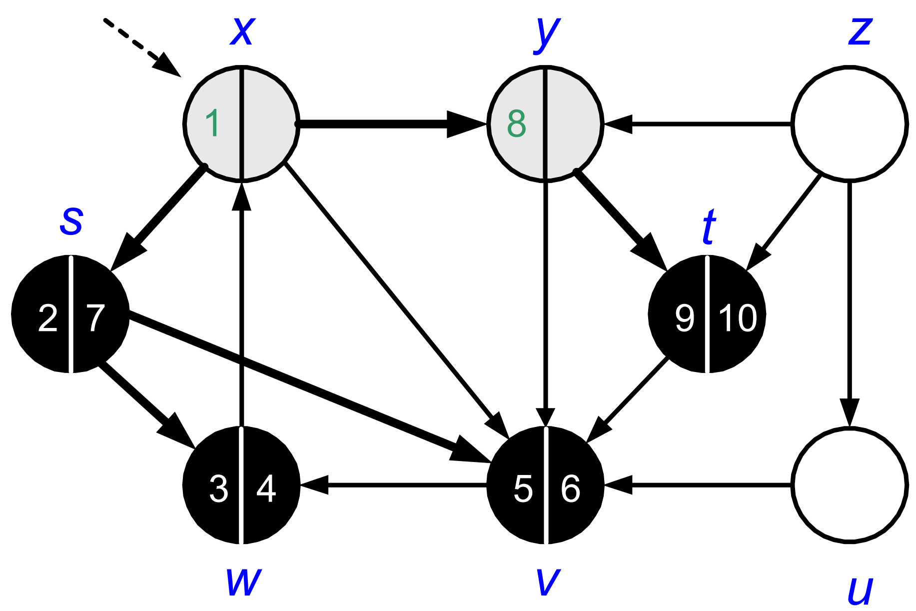
Depth-First Search: Example



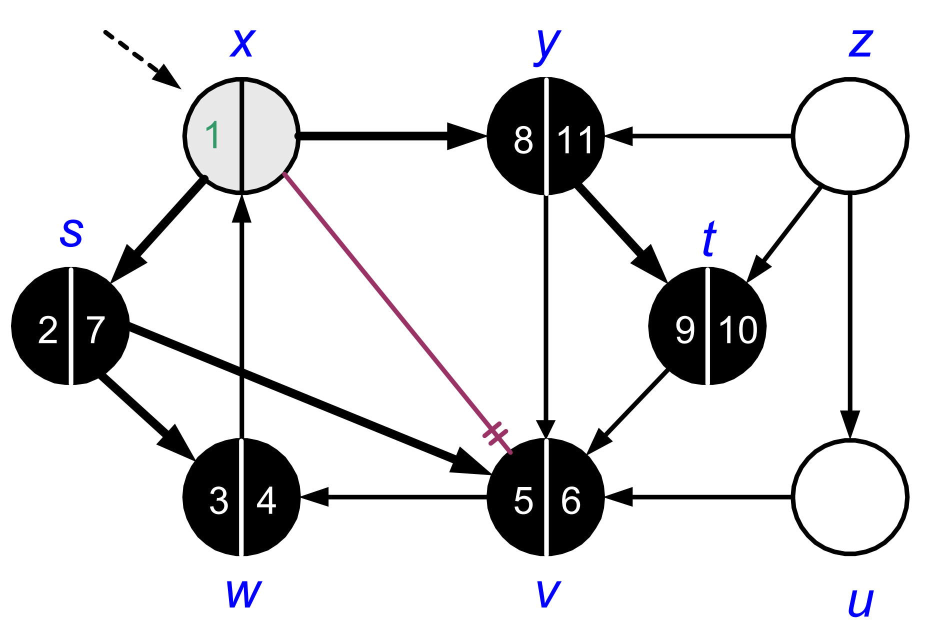
Depth-First Search: Example



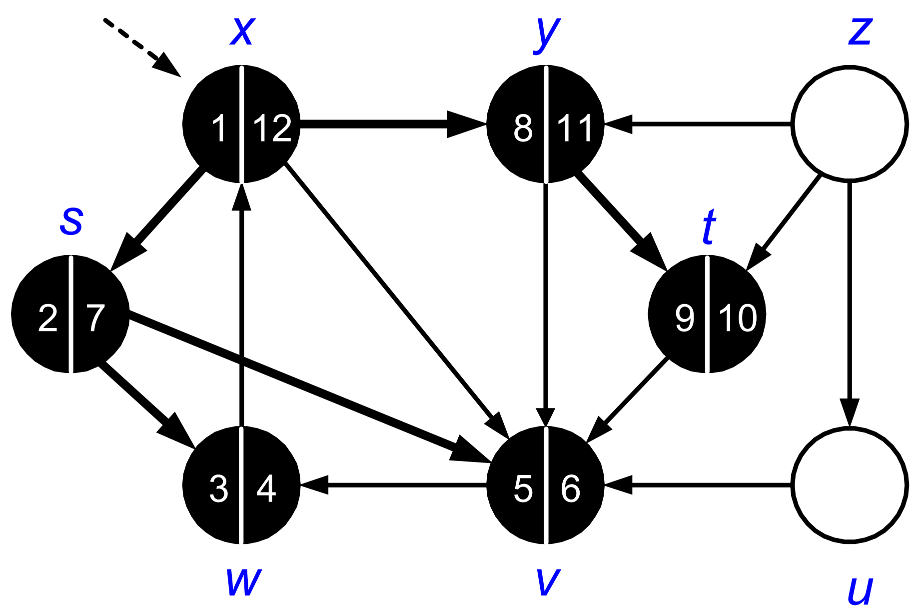
Depth-First Search: Example



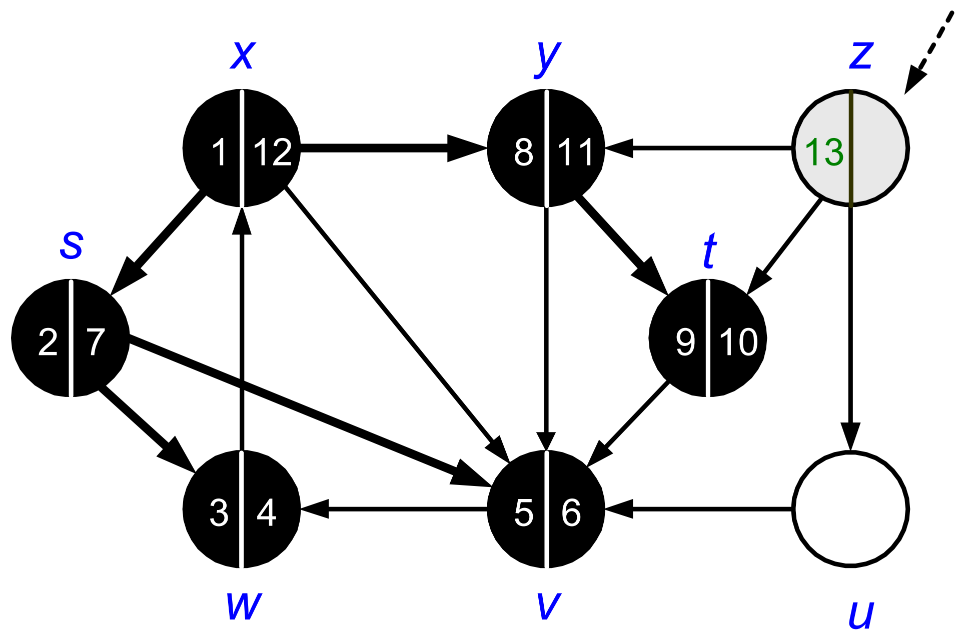
Depth-First Search: Example



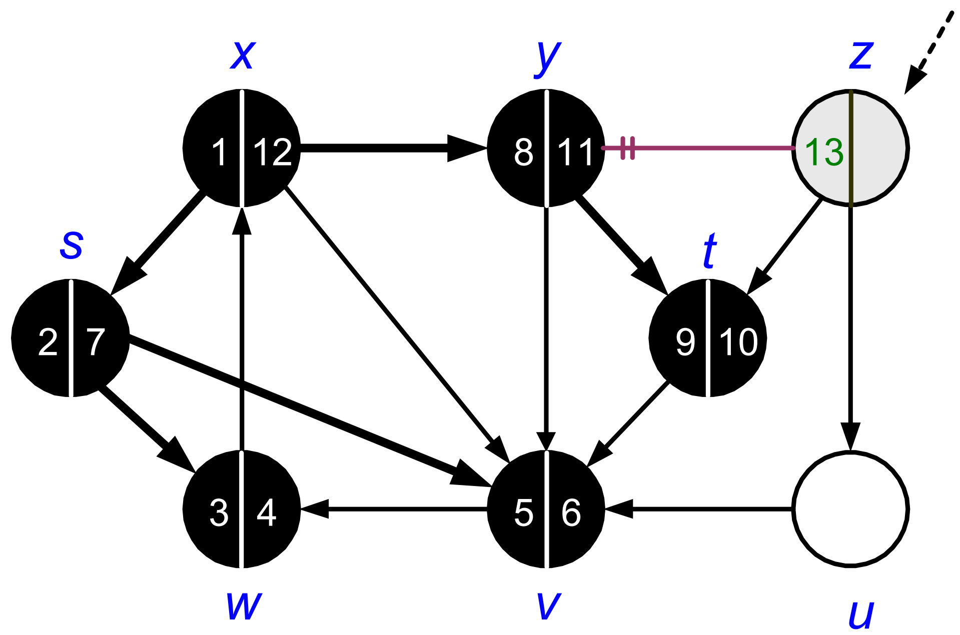
Depth-First Search: Example



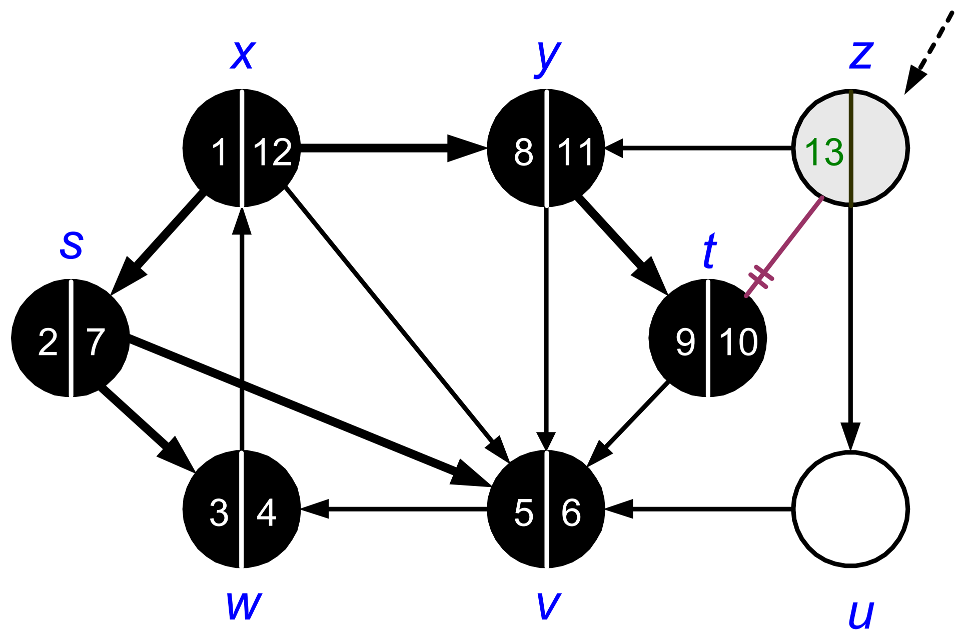
Depth-First Search: Example



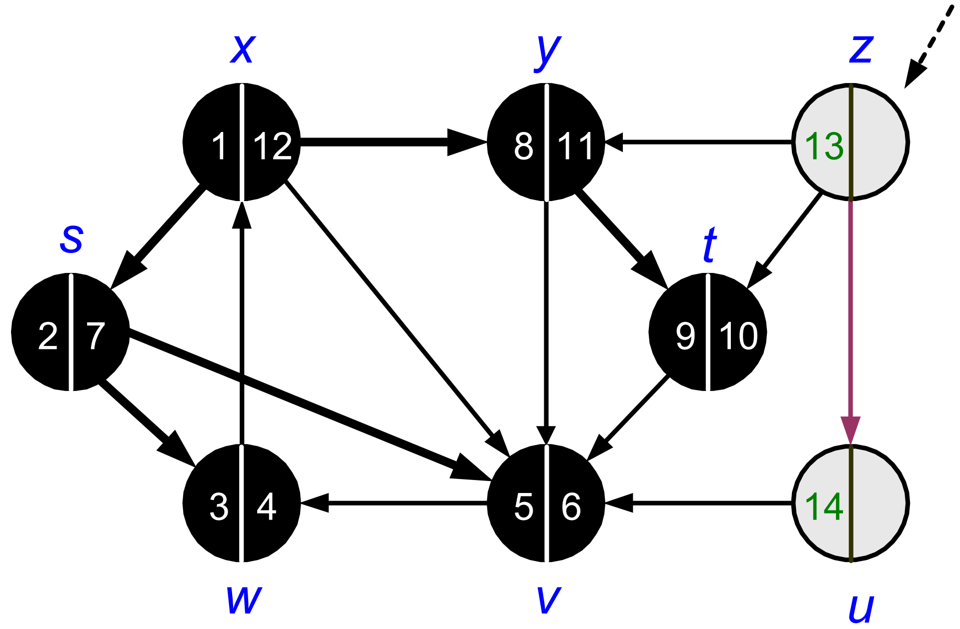
Depth-First Search: Example



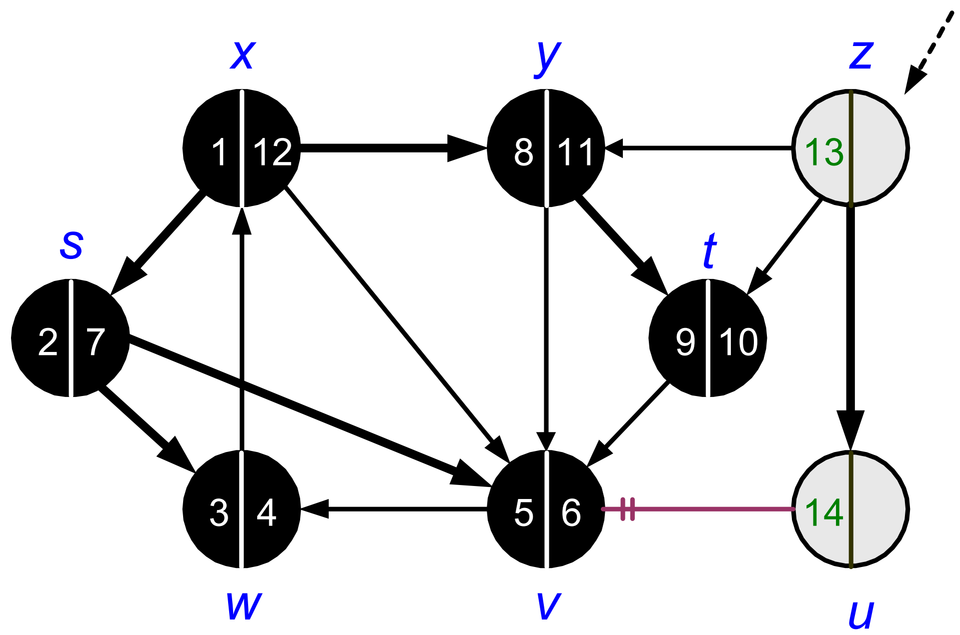
Depth-First Search: Example



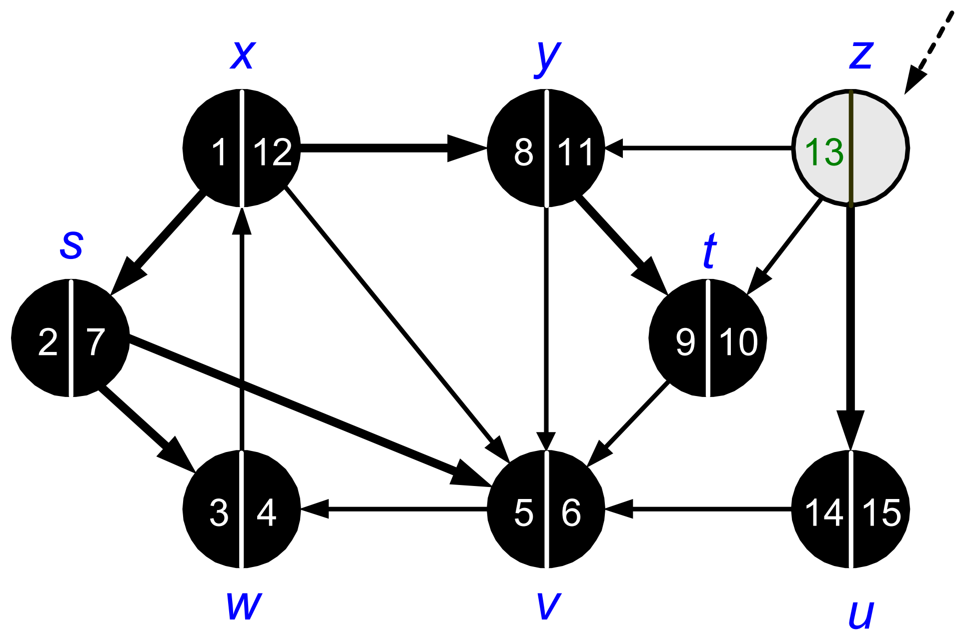
Depth-First Search: Example



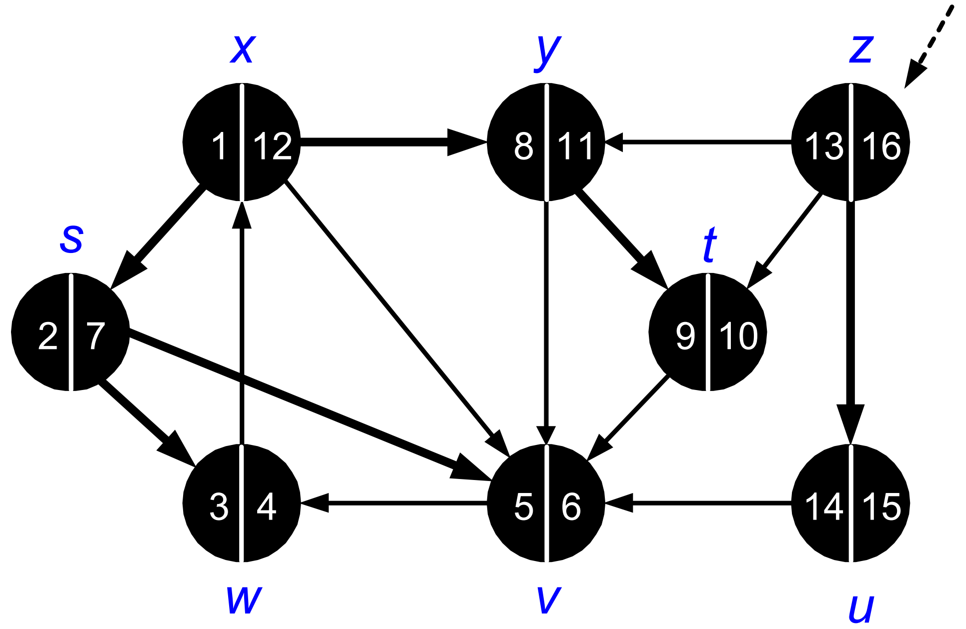
Depth-First Search: Example



Depth-First Search: Example

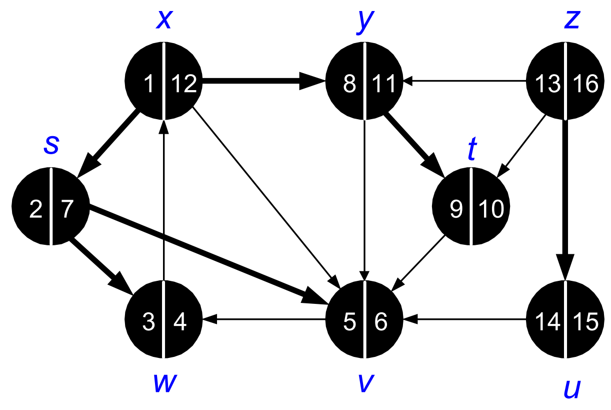


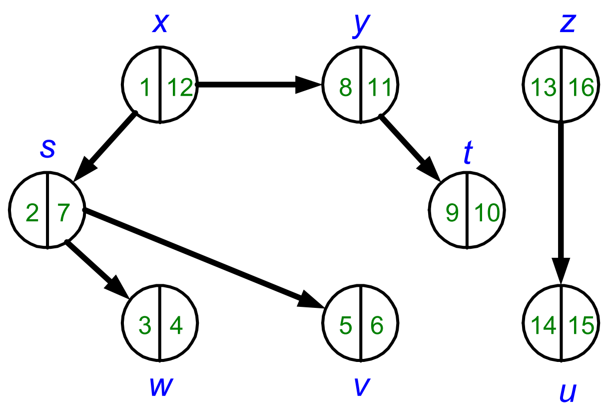
Depth-First Search: Example



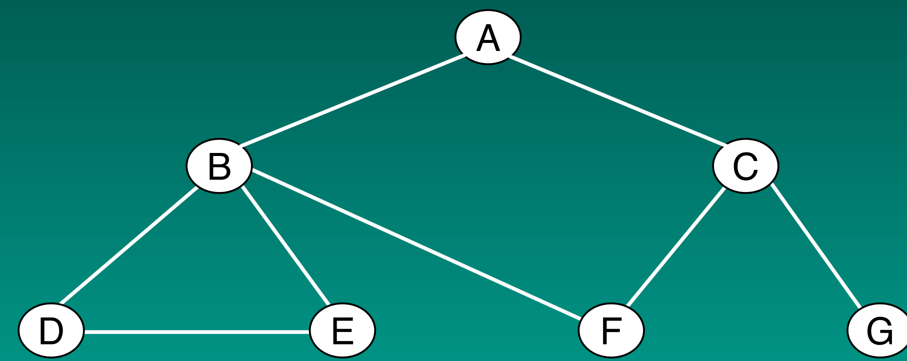
Depth-First Search: Example

DFS(G) terminated	Depth-first forest (DFF)



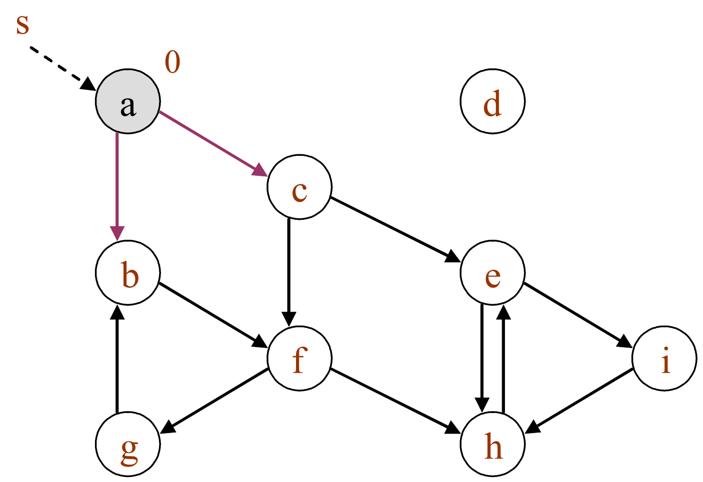


Depth-First Search



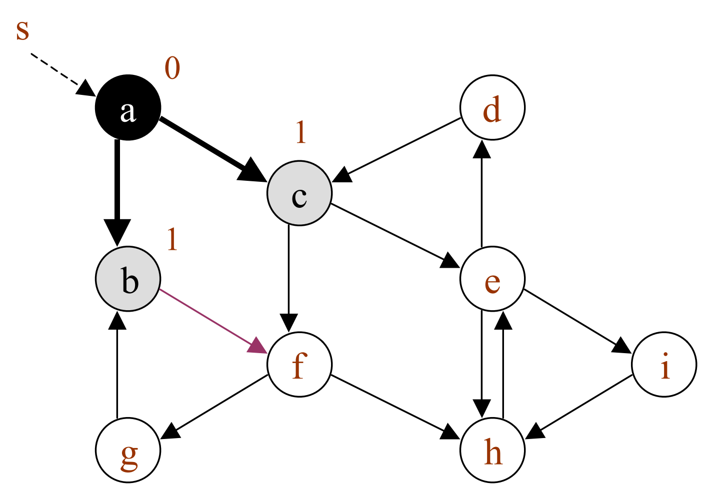
Breadth-First Search

Sample Graph:



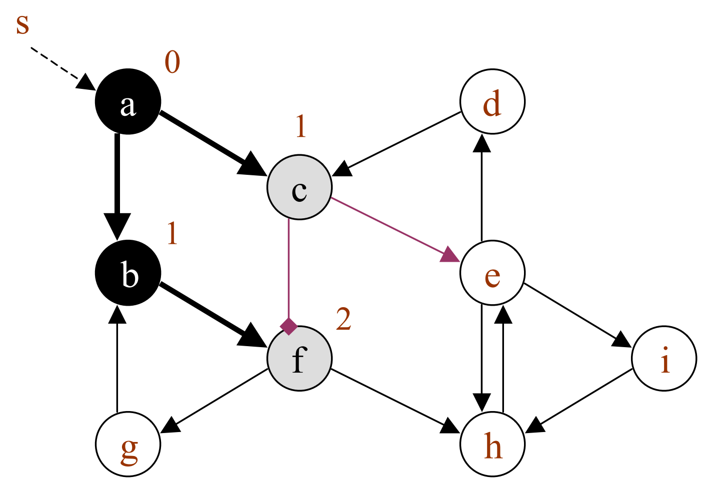
FIFO queue Q	just after processing vertex
⟨a⟩	-

Breadth-First Search



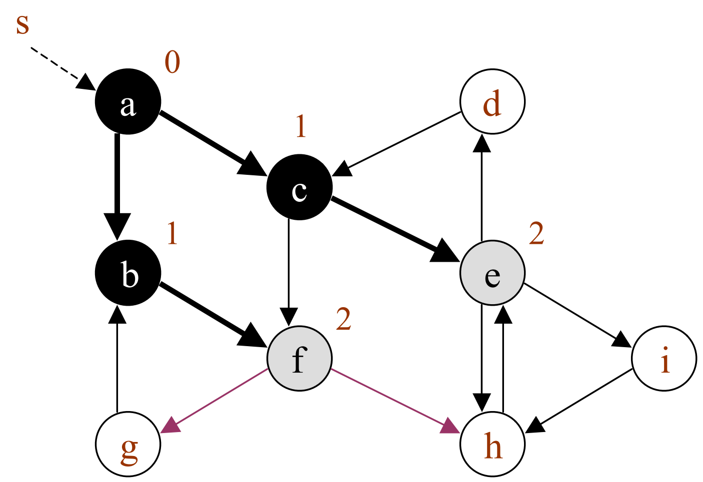
FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a

Breadth-First Search



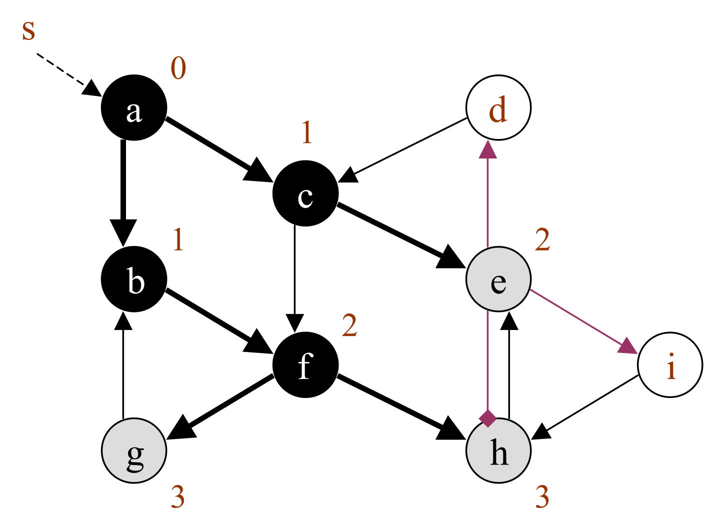
FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b

Breadth-First Search



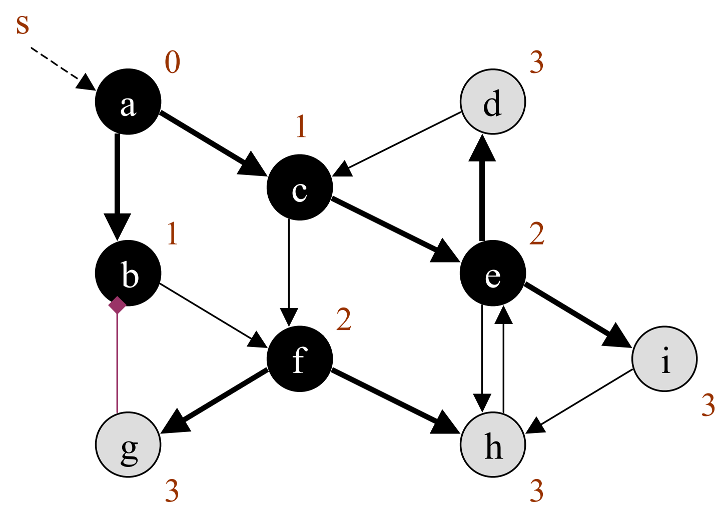
FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c

Breadth-First Search



FIFO <u>queue *Q*</u>	just after <u>processing vertex</u>
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f

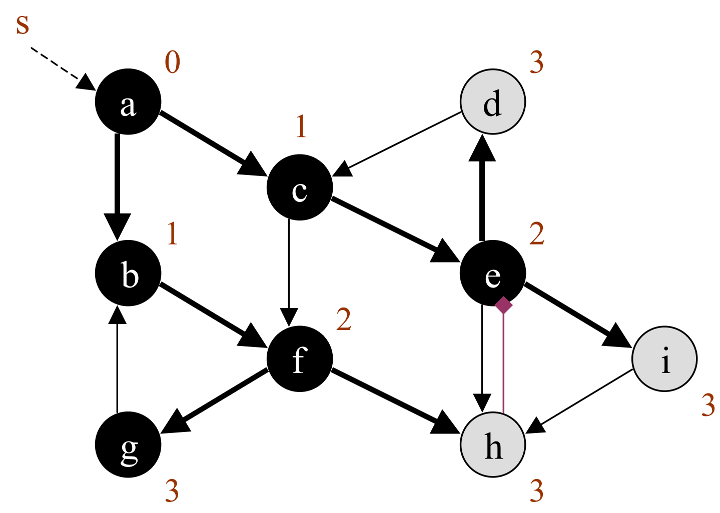
Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	e

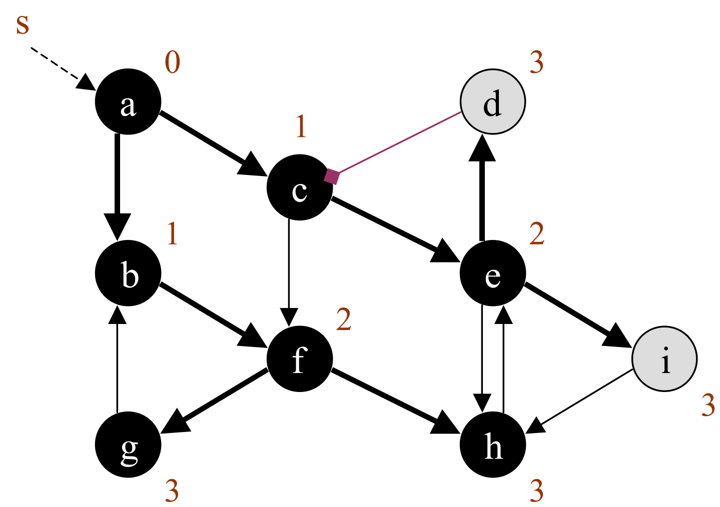
all distances are filled in after processing e

Breadth-First Search



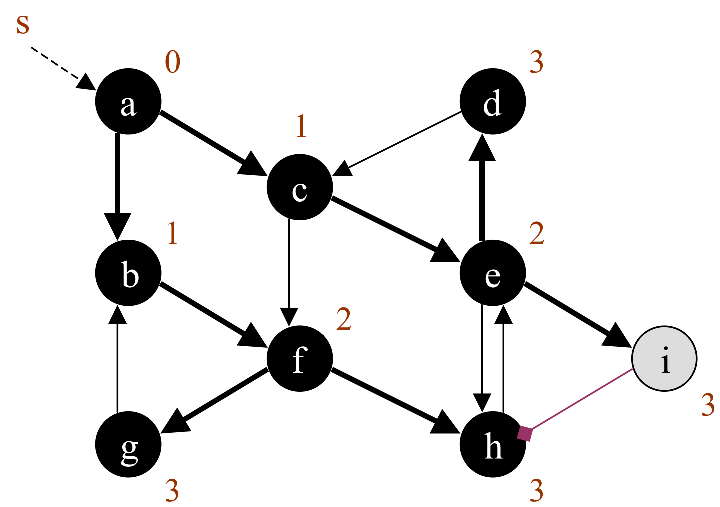
FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	g

Breadth-First Search



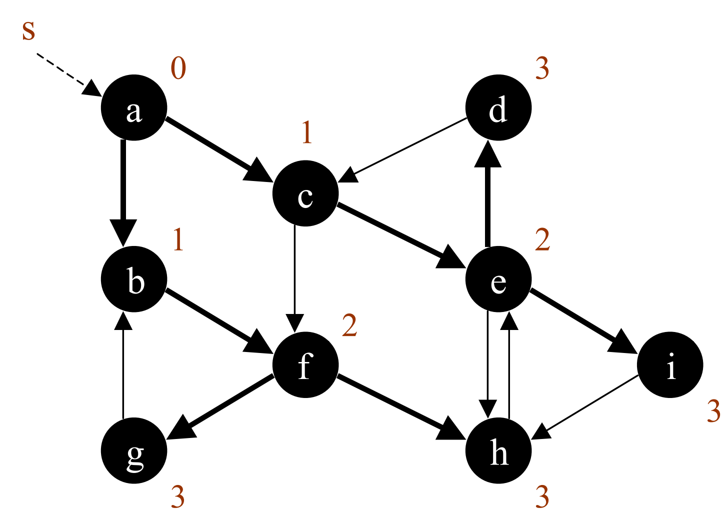
FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	h

Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	d

Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	i

algorithm terminates: all vertices are processed

Breadth first search - analysis

- Enqueue and Dequeue happen only once for each node. - O(V).
- Sum of the lengths of adjacency lists – O(E) (for a directed graph)
- Initialization overhead O(V)

Total runtime O(V+E)

Depth first search - analysis

- initialization take time O(V).
- DFS-VISIT is called only once for each node (since it's called only for white nodes and the first step in it is to paint the node gray).
- the total cost of DFS-VISIT it O(E)

The total cost of DFS is O(V+E)

BFS and DFS - comparison

- Space complexity of DFS is lower than that of BFS.
- Time complexity of both is same – O(|V|+|E|).