Working with Graph Algorithms in Python

INTRODUCING THE GRAPH DATA STRUCTURE

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

What You Need in Your Toolkit

Prerequisites

Familiarity with the command line on a Mac, Linux or Windows machine

A basic understanding of algorithms and time complexity

Comfortable with writing programs in Python

Install and Setup

A Mac, Linux or Windows machine on which you code and run programs

A working version of Python 2.7.x or 3.x

Course Overview

Introduction to the graph data structure and its representation and traversal

Ordering of dependent nodes using topological sort

Shortest path algorithms in weighted and unweighted graphs

Spanning tree algorithms to connect all nodes in a graph

Overview

Graphs are excellent tools for modeling complex relationships

An adjacency matrix is the most common way of representing a graph

Adjacency lists and adjacency sets are alternative data representations

The two fundamental ways of traversing a graph are

- Depth-first
- Breadth-first

Graphs for Modeling Relationships

Two Big Trends

Bigger data

More and more data being collected and aggregated

Smaller world

More and more interconnections between actions and events

Modeling interconnections is increasingly important

Interconnections

Jim

Drives

Car

Relationships between entities

Interconnections

Jim

Drives

Car

Relationships between entities

Interconnections

Jim

Drives

Car

Relationships between entities

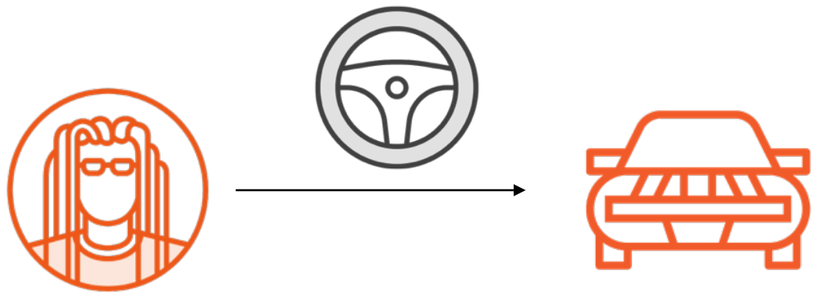
Graphs

Jim

Drives

Car

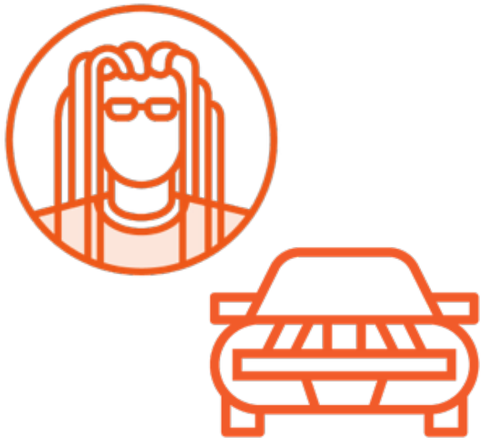
Graphs represent relationships between entities



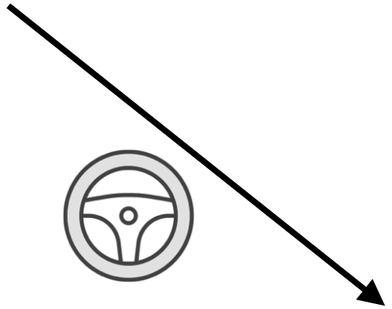
Graphs consist of

- Vertices (entities)
- Edges (relationships)

Modeling the Real World



Vertex

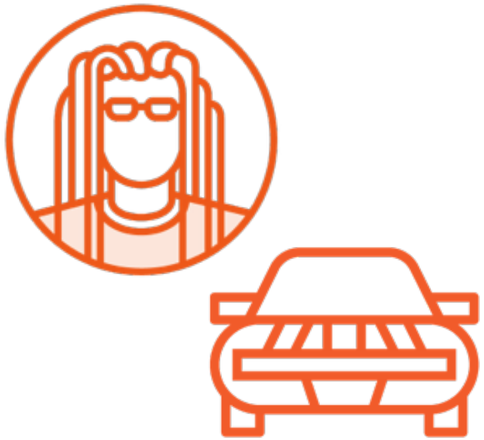


Edge

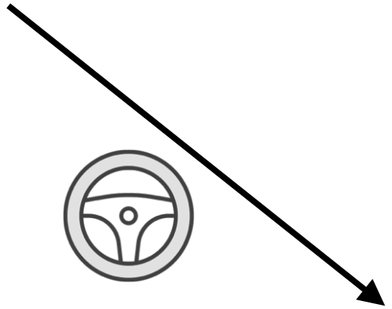
People

Social or professional relationships

Modeling the Real World



Vertex

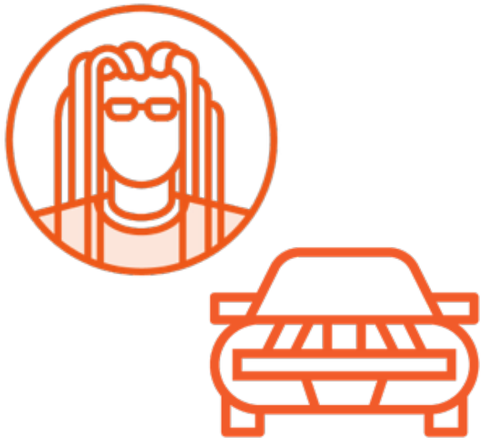


Edge

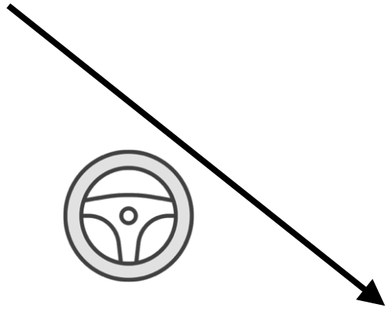
Locations

Means of transportation
i.e. road, rail air

Modeling the Real World



Vertex

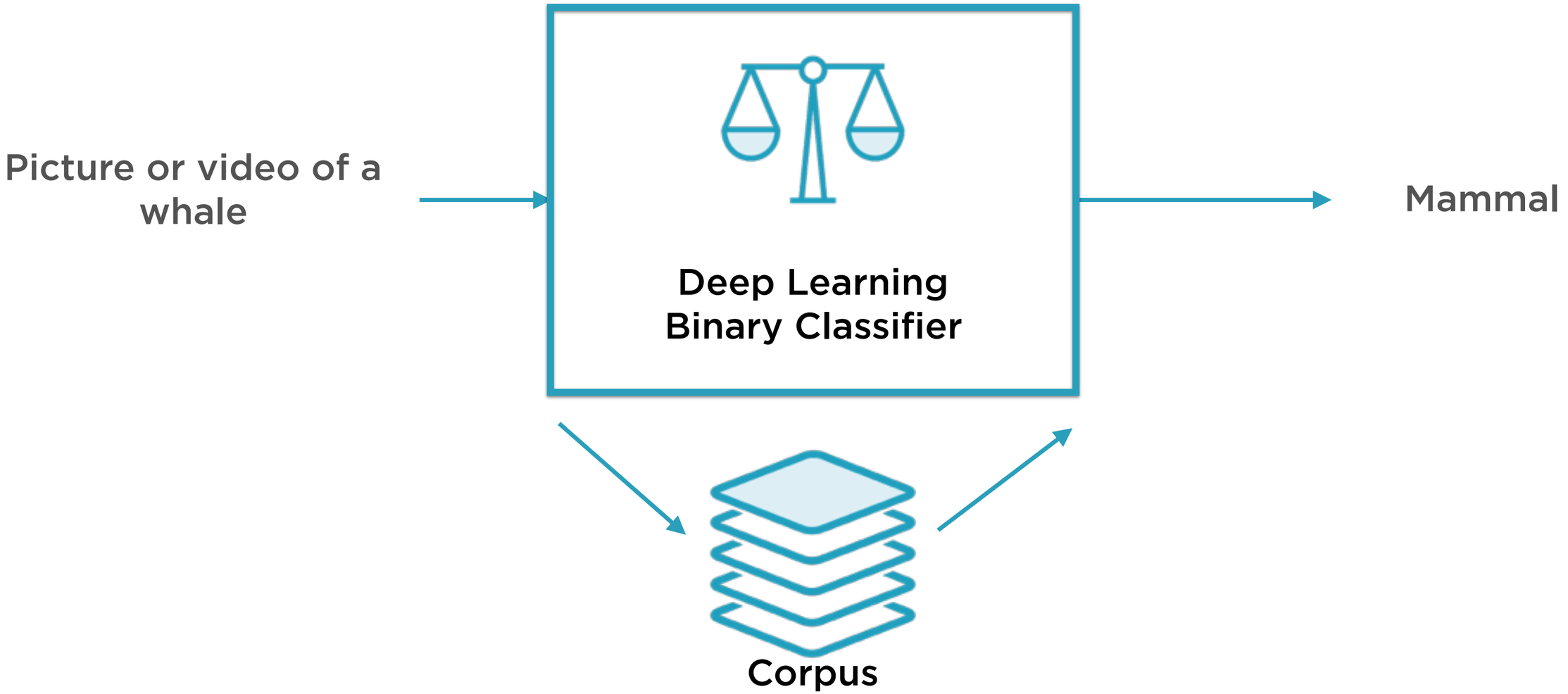


Edge

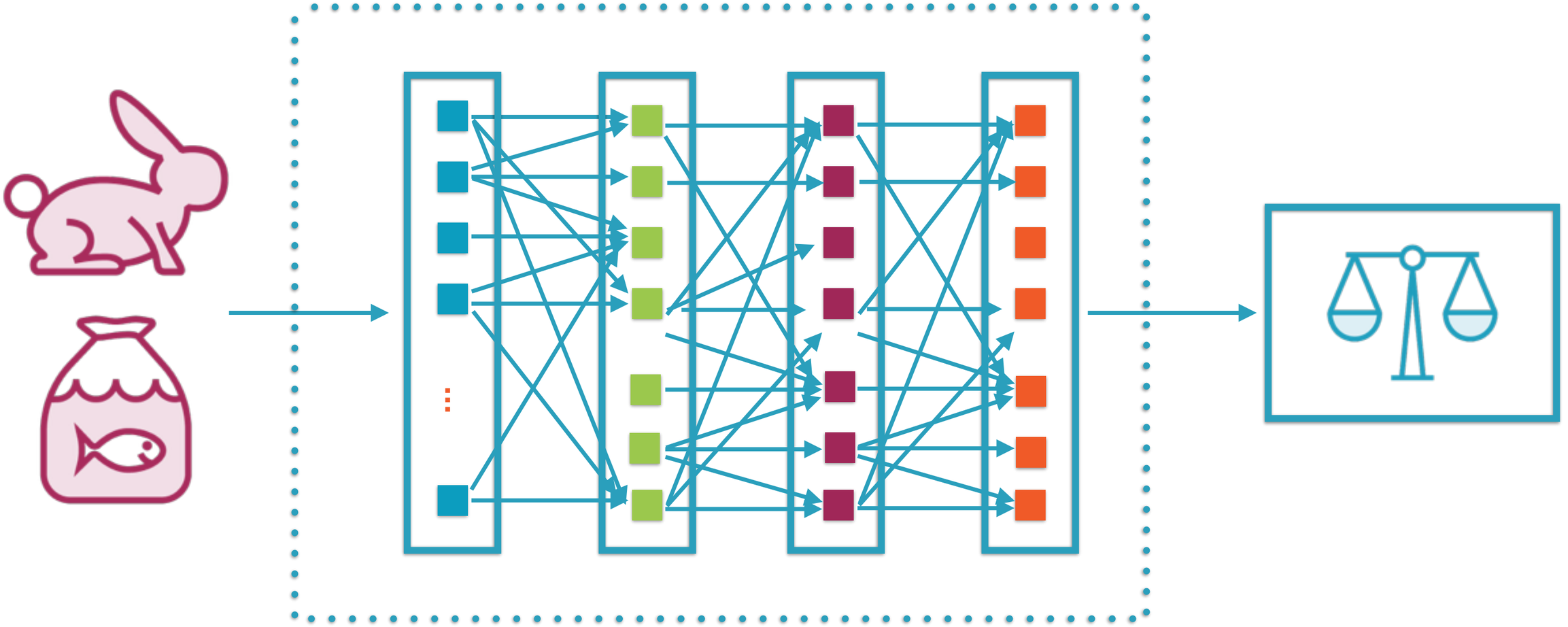
Phones - landlines

Phone network to carry voice calls

“Deep Learning” Binary Classifier



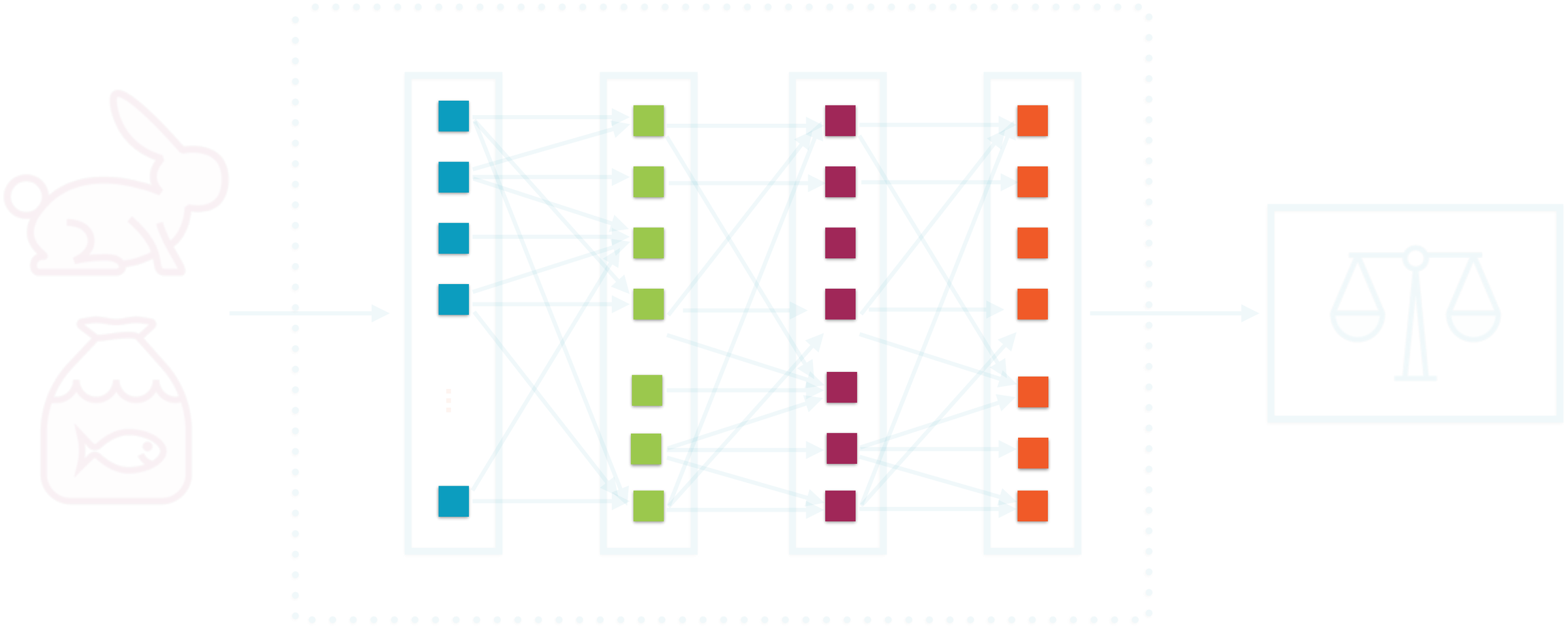
Neural Network Computation Graph



Corpus of Images

ML-based Classifier

Neural Network Computation Graph

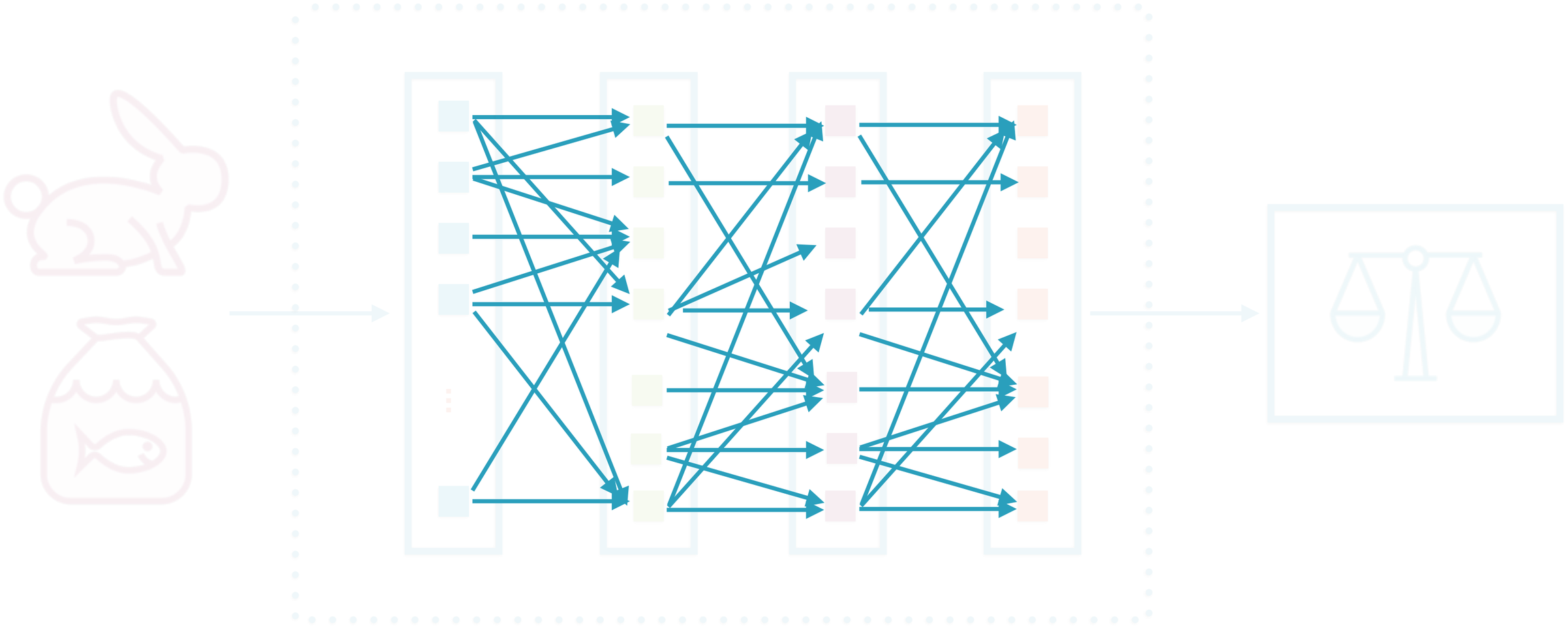


Corpus of Images

ML-based Classifier

The vertices in the computation graph are neurons (simple building blocks)

Neural Network Computation Graph



Corpus of Images

ML-based Classifier

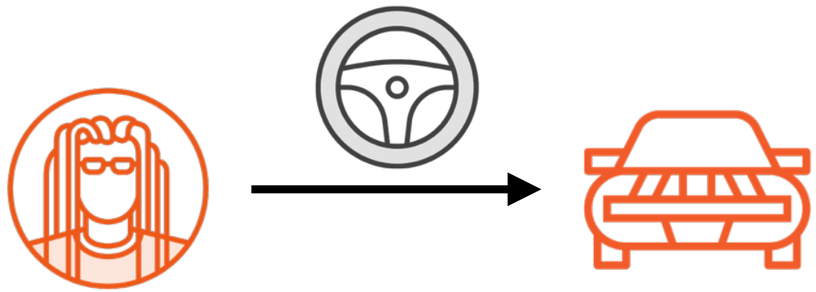
The edges in the computation graph are data items called tensors

Structure of a Graph

Graph (V,E)

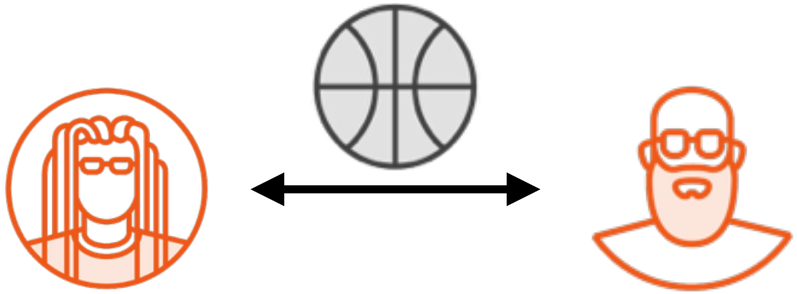
A set of vertices (V) and edges (E)

Directed and Undirected Graphs



“Jim drives his car”

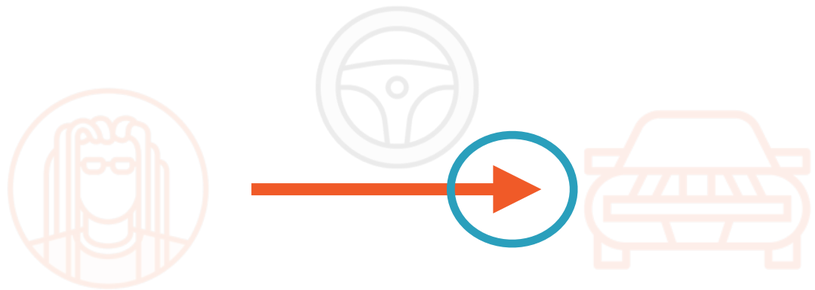
Relationship goes one way only



“Jim and Joe play ball”

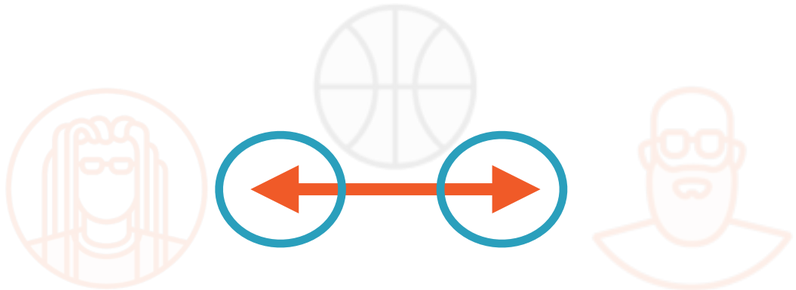
Relationship goes both ways

Directed and Undirected Graphs



"Jim drives his car"

Relationship goes one way only



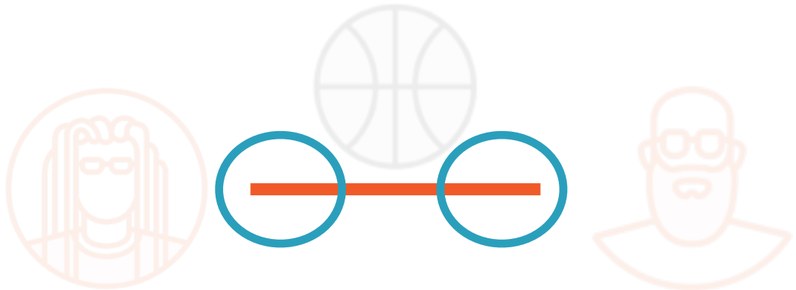
"Jim and Joe play ball"

Relationship goes both ways

Directed and Undirected Graphs

“Jim drives his car”

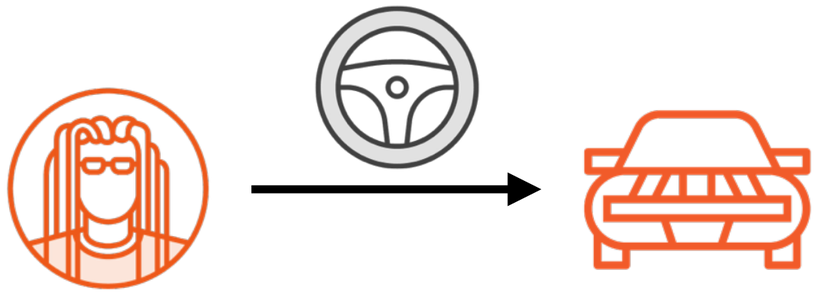
Relationship goes one way only



“Jim and Joe play ball”

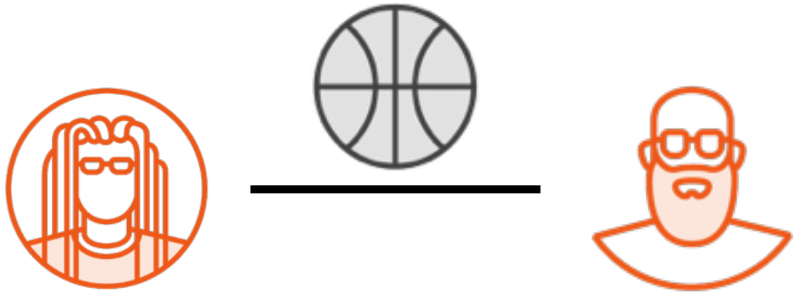
Relationship goes both ways

Directed and Undirected Graphs



“Jim drives his car”

Relationship goes one way only



“Jim and Joe play ball”

Relationship goes both ways

Directed and Undirected Graphs

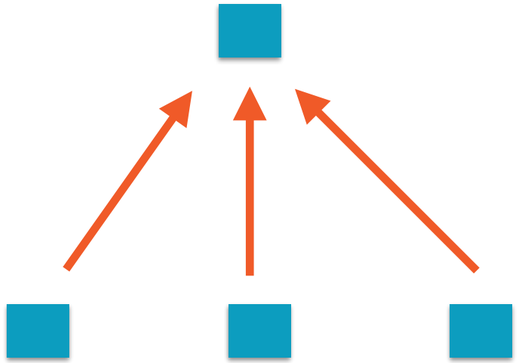
Directed Graph

Relationship goes one way only

Undirected Graph

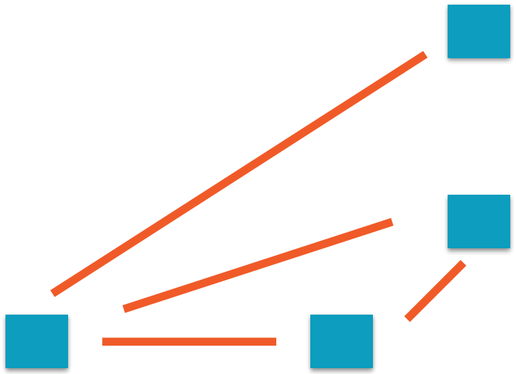
Relationship goes both ways

Directed and Undirected Graphs



Twitter Followers

Relationship goes one way only

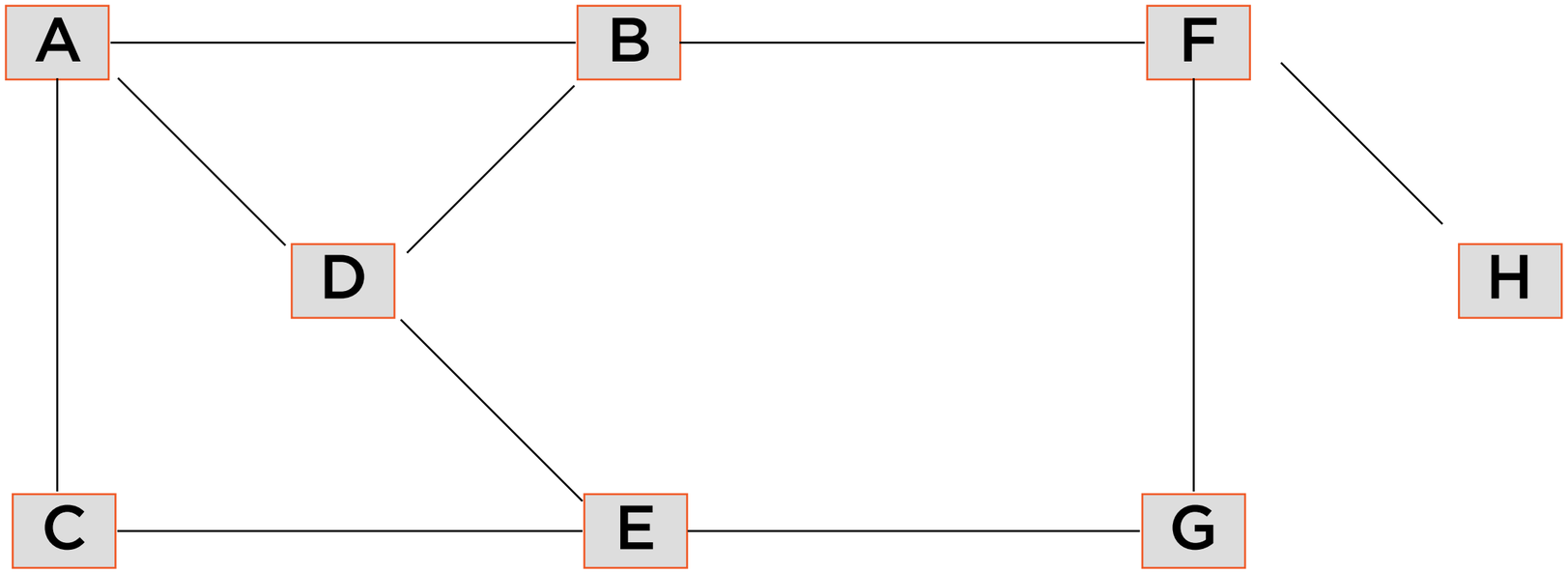


Facebook Friends

Relationship goes both ways

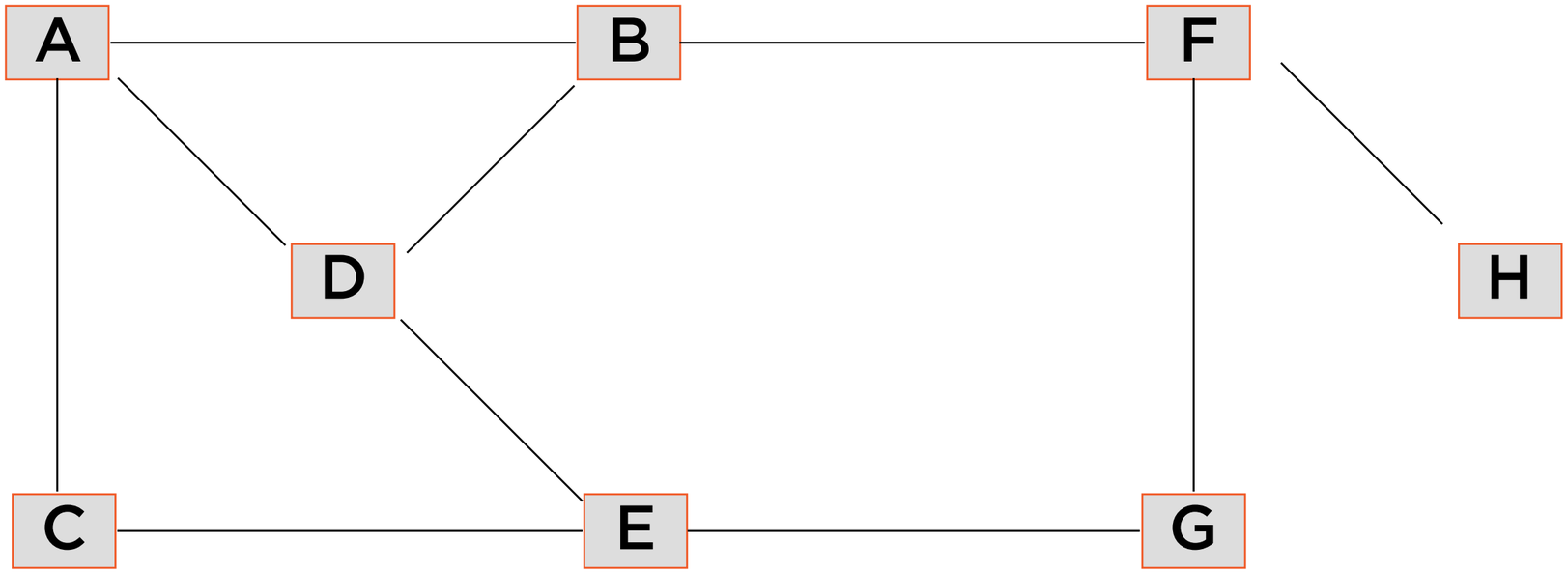
Undirected Graphs

An Undirected Graph



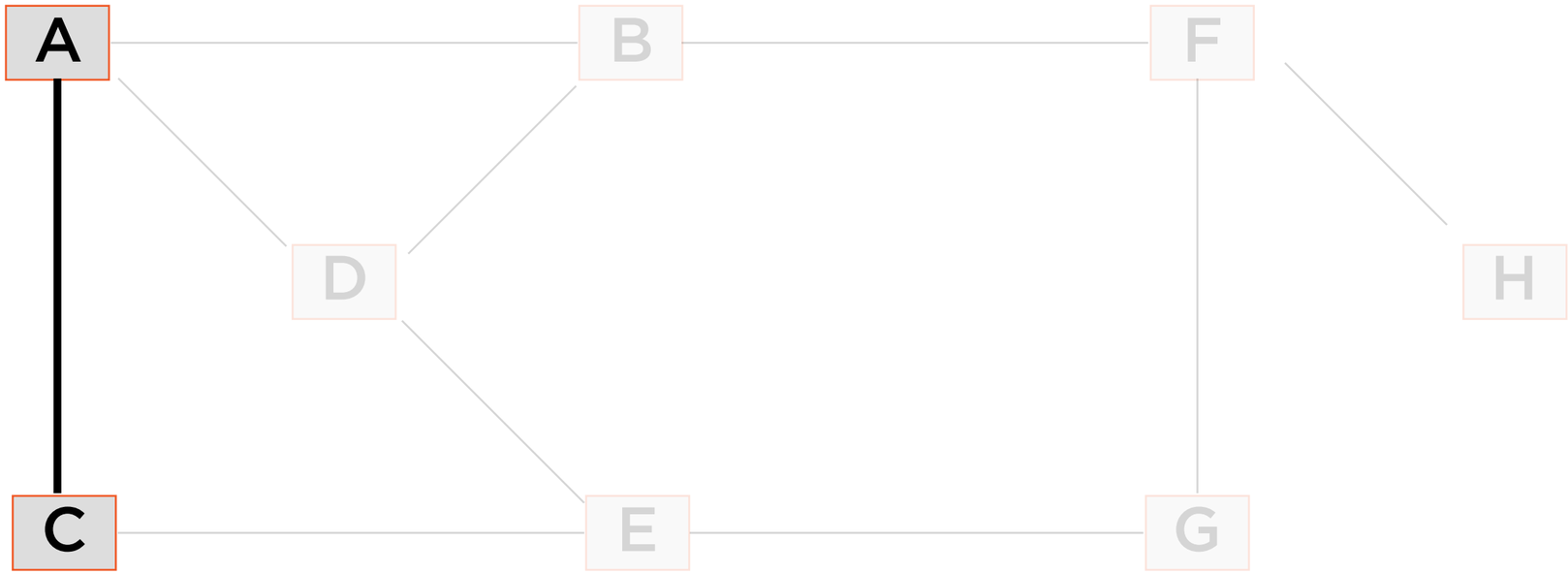
V = {A, B, C, D, E, F, G, H}

An Undirected Graph



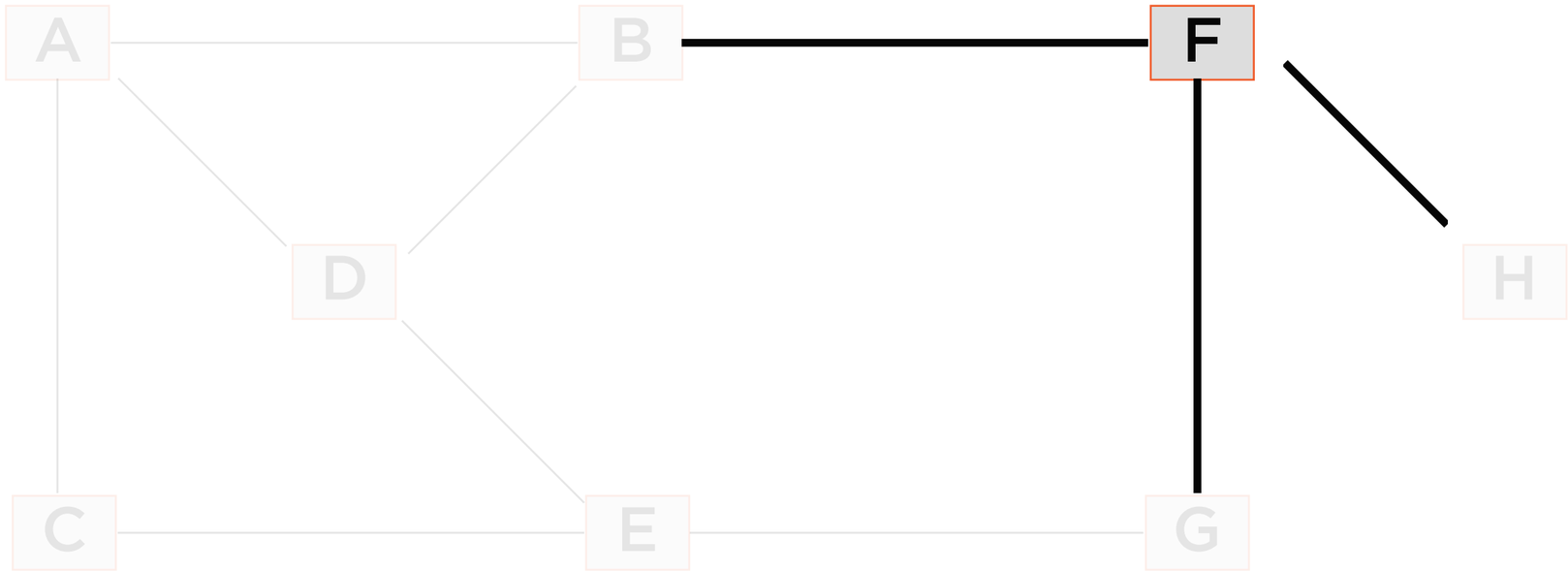
V = {A, B, C, D, E, F, G, H}

Adjacent Nodes



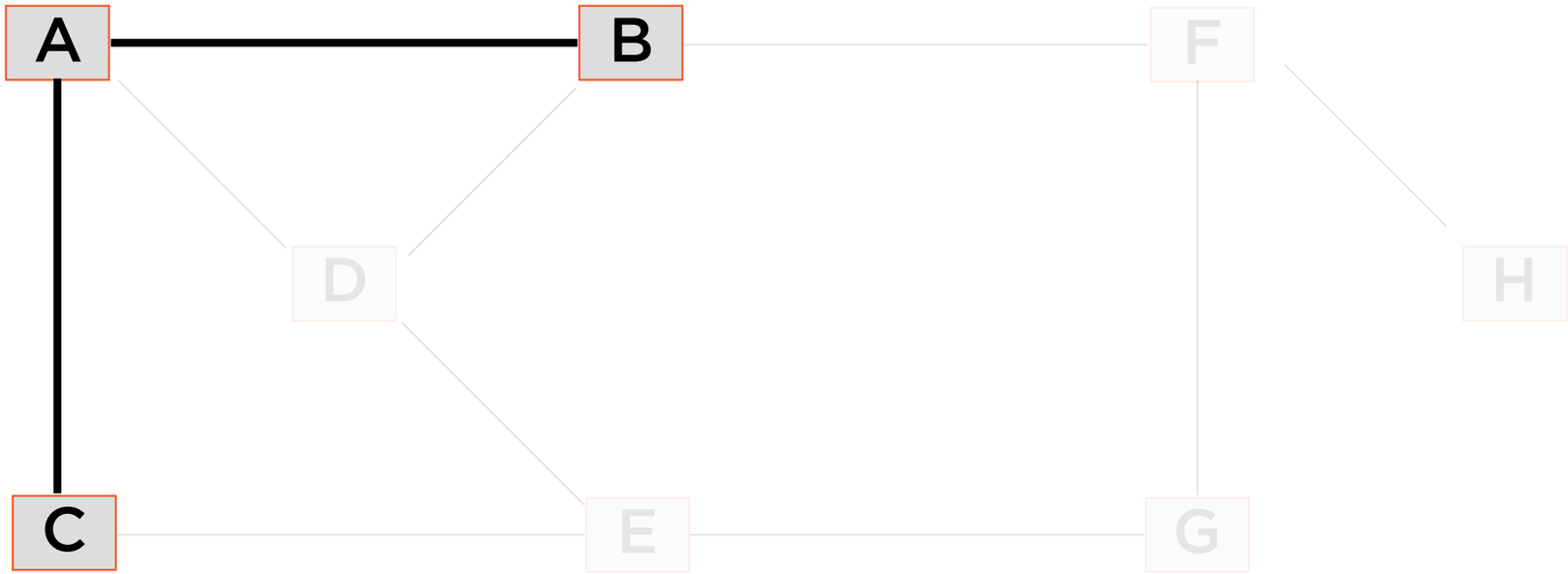
A and C are adjacent nodes - a single edge connects them

Degree of a Node



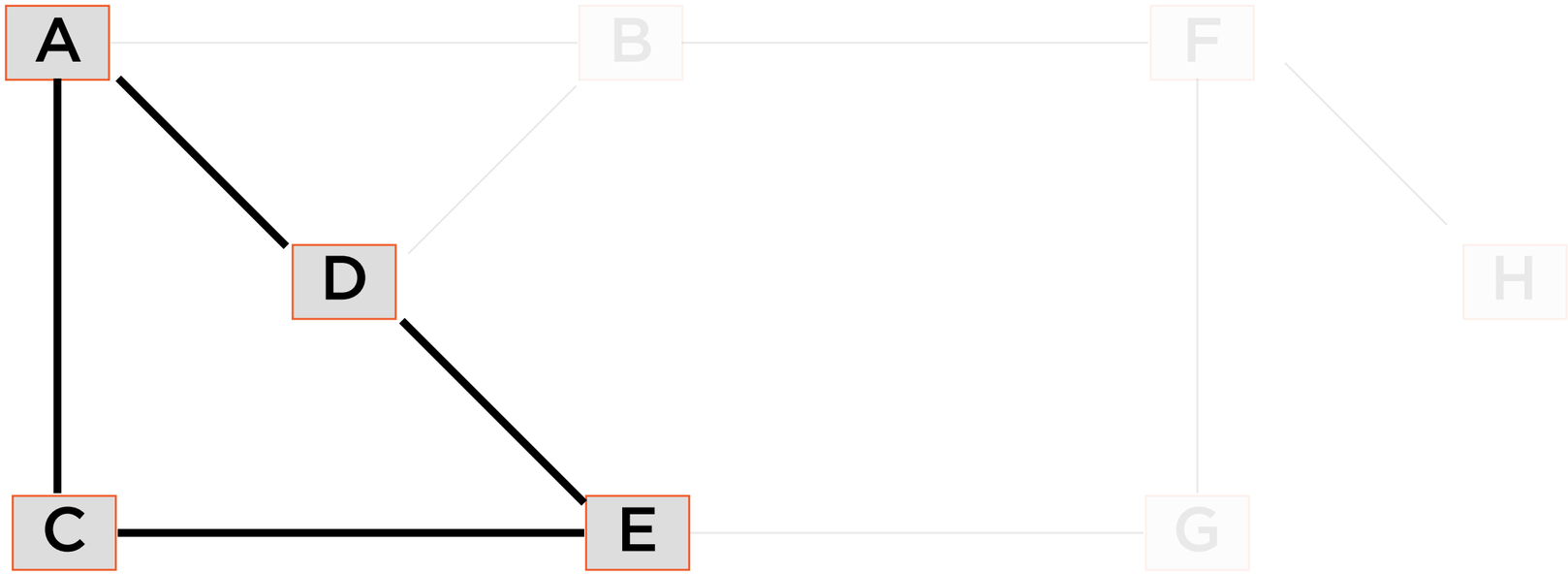
The degree of F is 3, since 3 edges are incident on F

Paths in a Graph



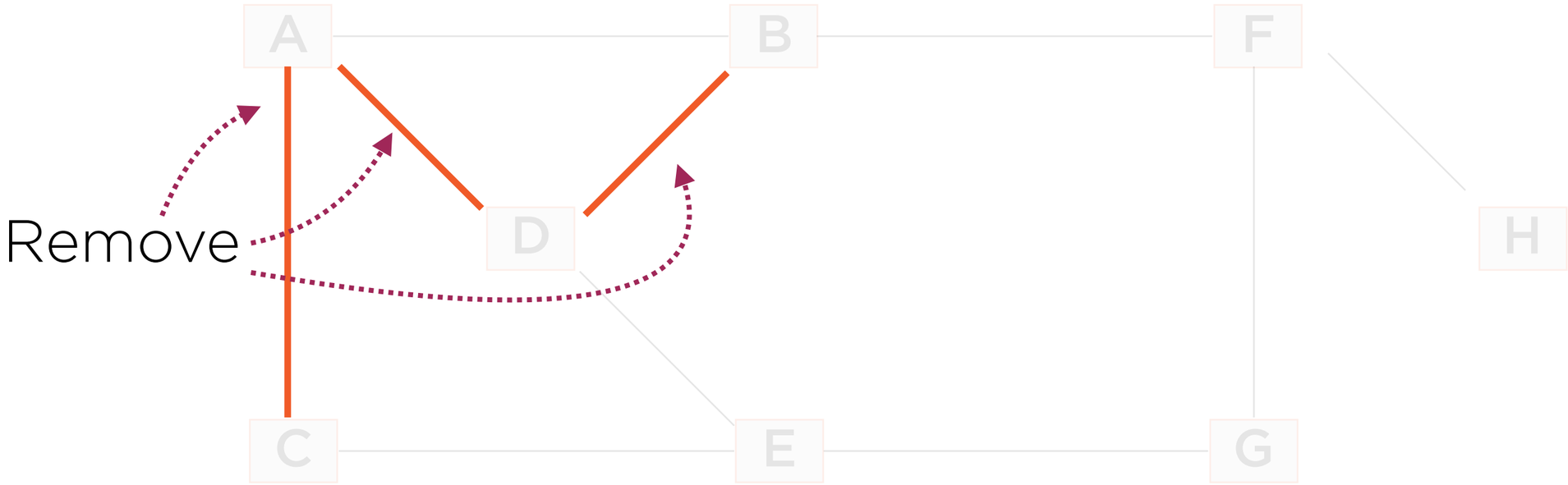
A series of edges links node C to node B - this is called a path

Undirected Cyclic Graph



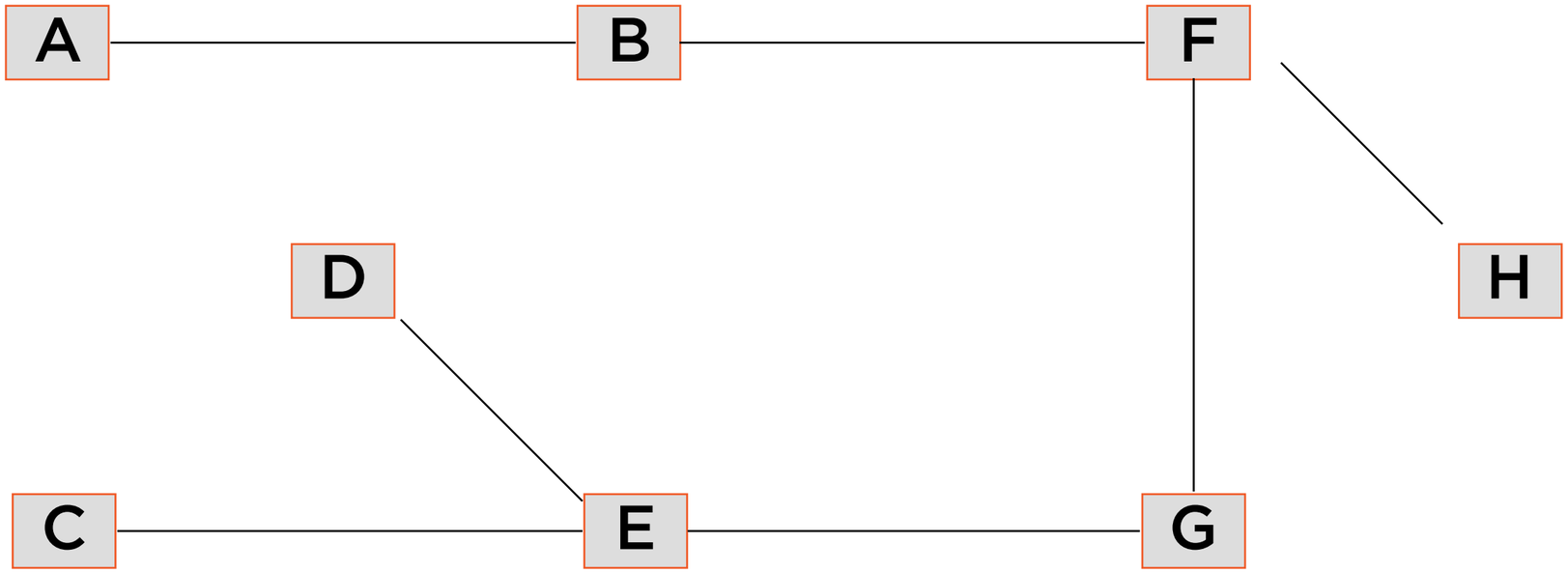
The nodes A, D, E, C and A form a cycle

Undirected Acyclic Graphs



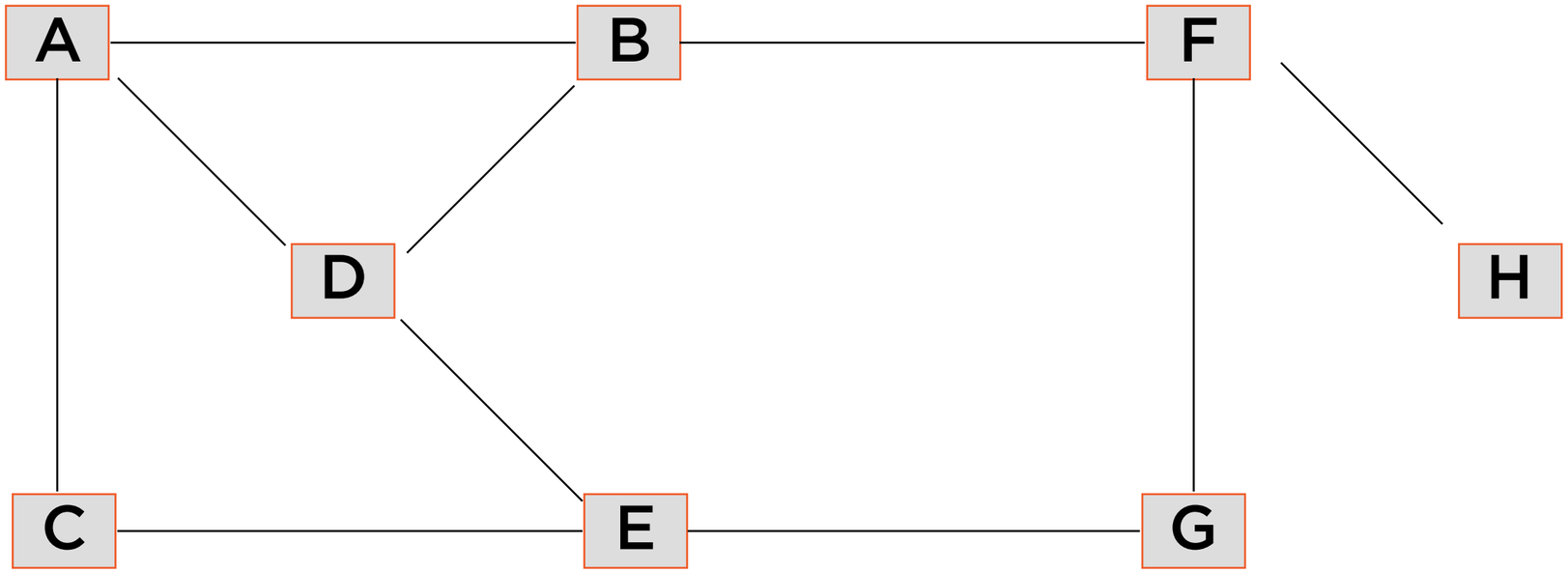
Removing edges A - C, A - D and B - D eliminates the cycles in this graph

Undirected Acyclic Graphs



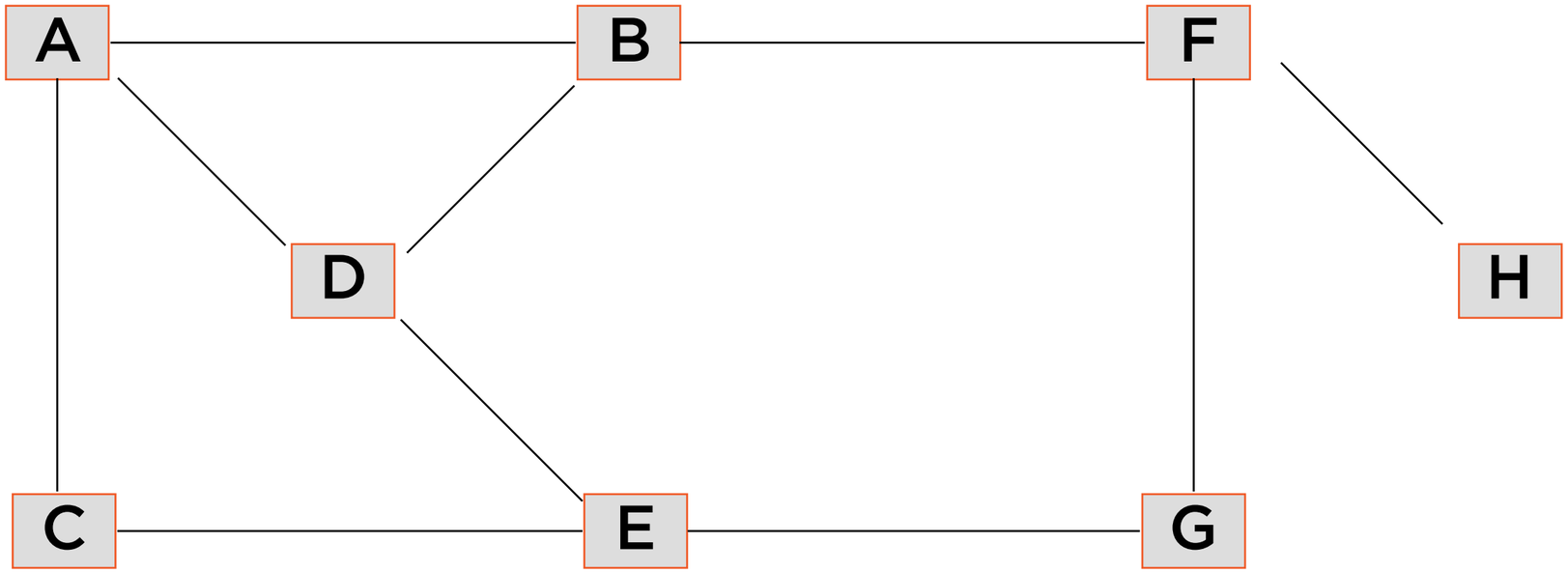
The graph is now an undirected acyclic graph

Connected Graph



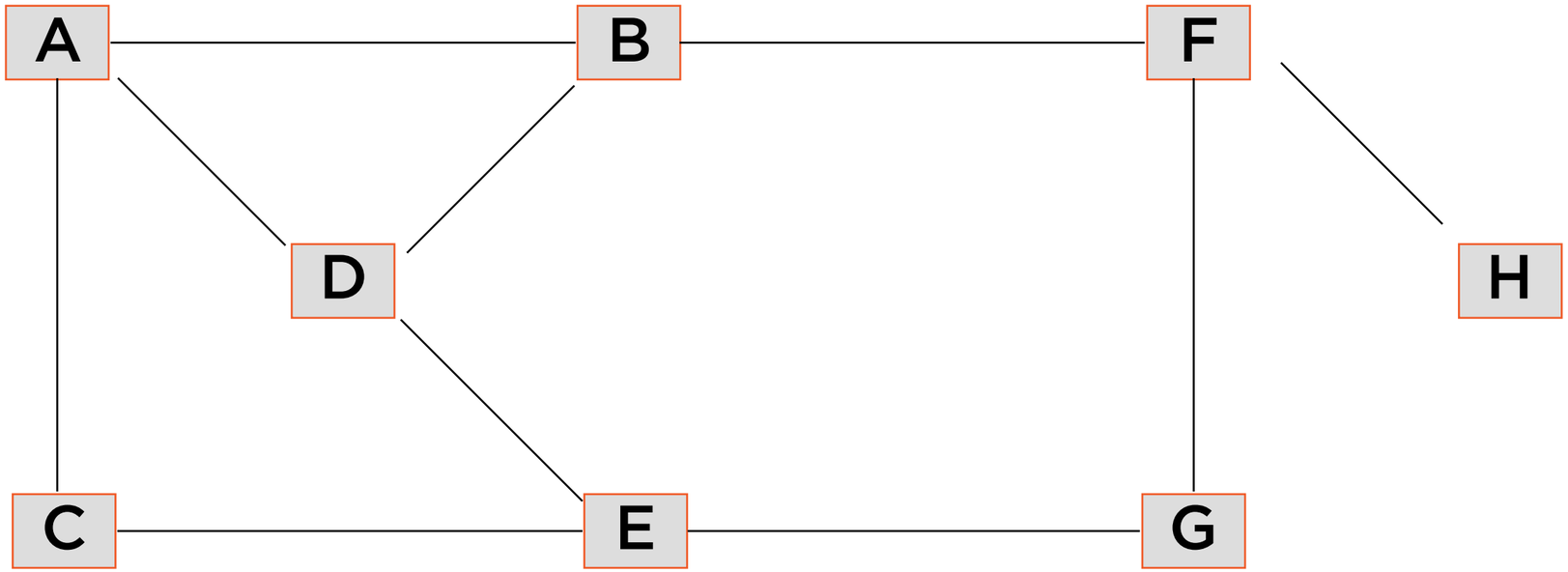
Every node is connected to every other node via a series of edges

Connected Graph



Equivalently, there is a path from every node to every other node

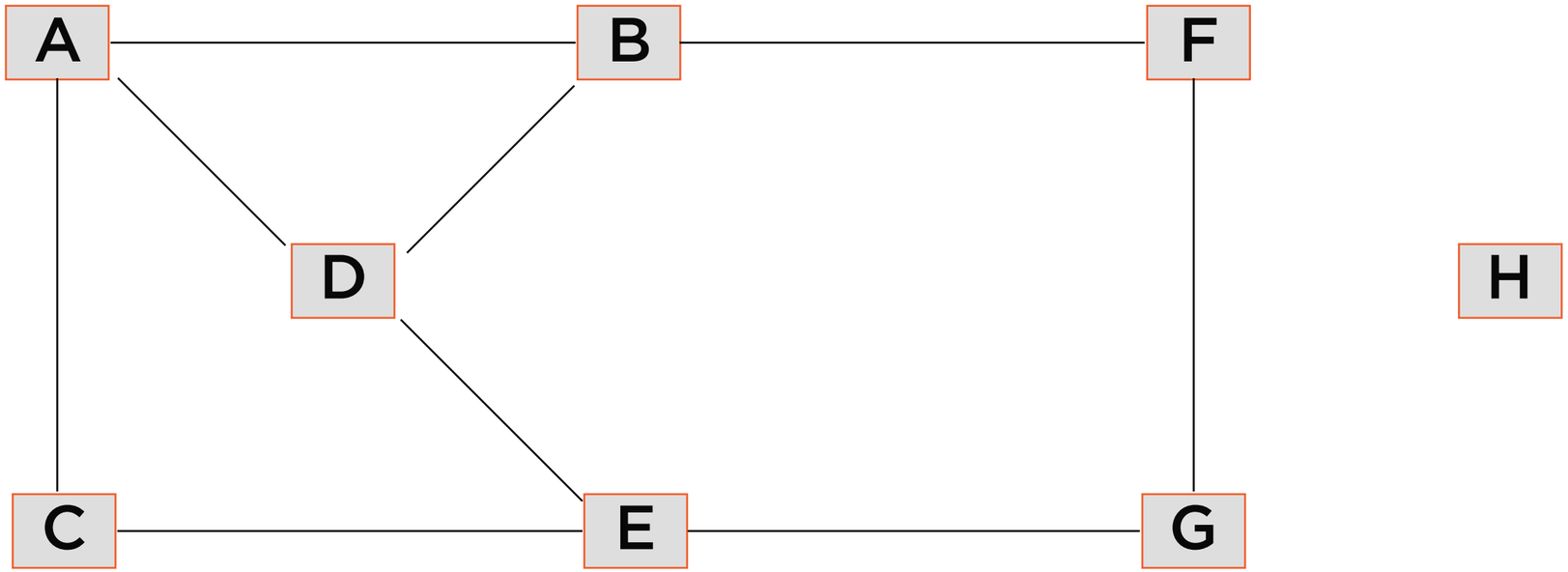
Disconnected Graph



Disconnected Graph

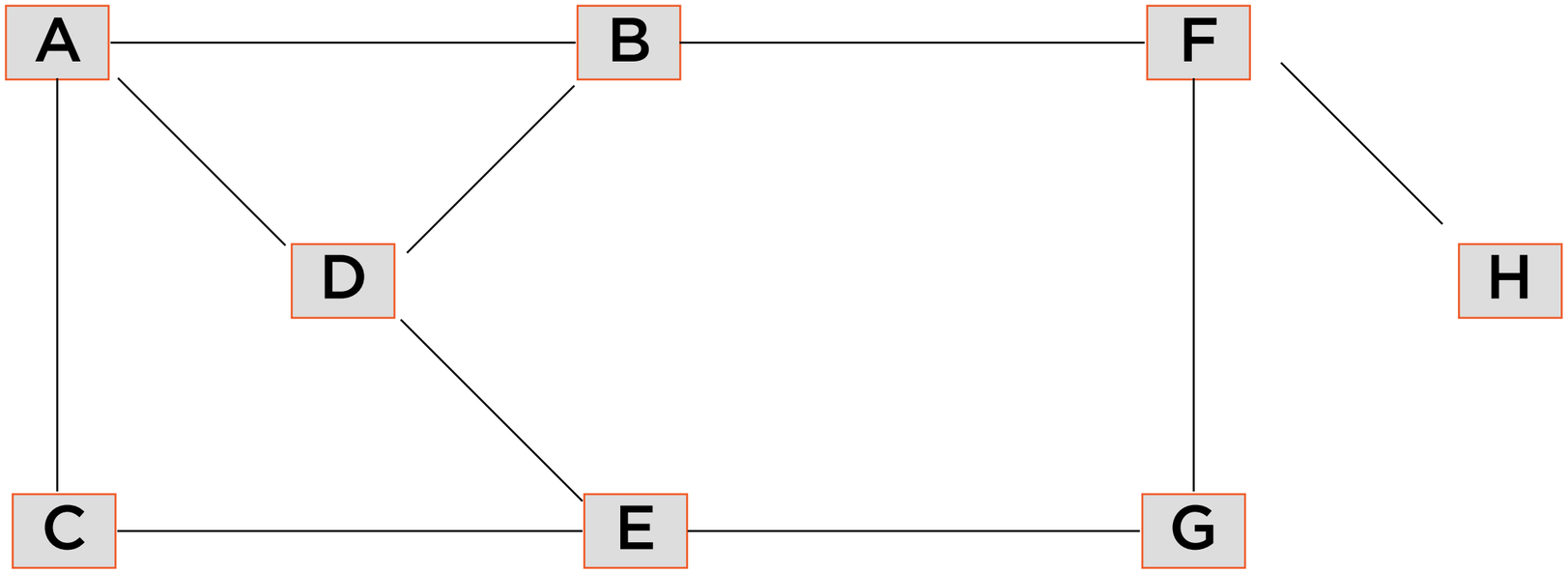
Removing the F - H edge leaves H without a path to the other nodes in the graph

Disconnected Graph

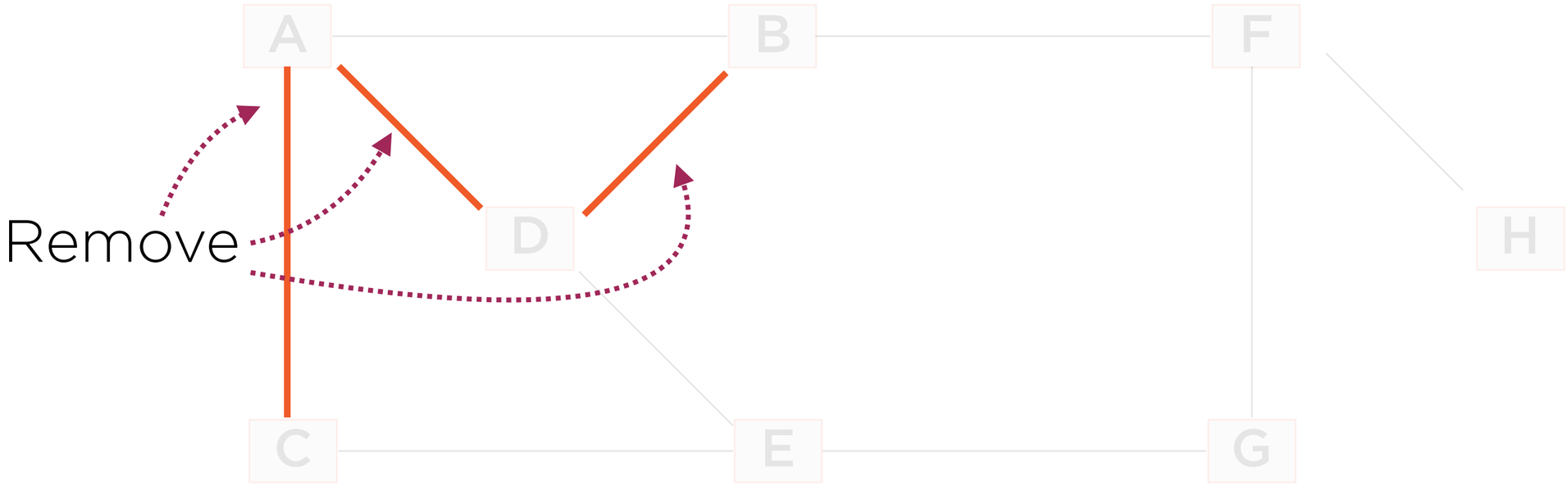


Disconnected

Connected Graph with Cycle

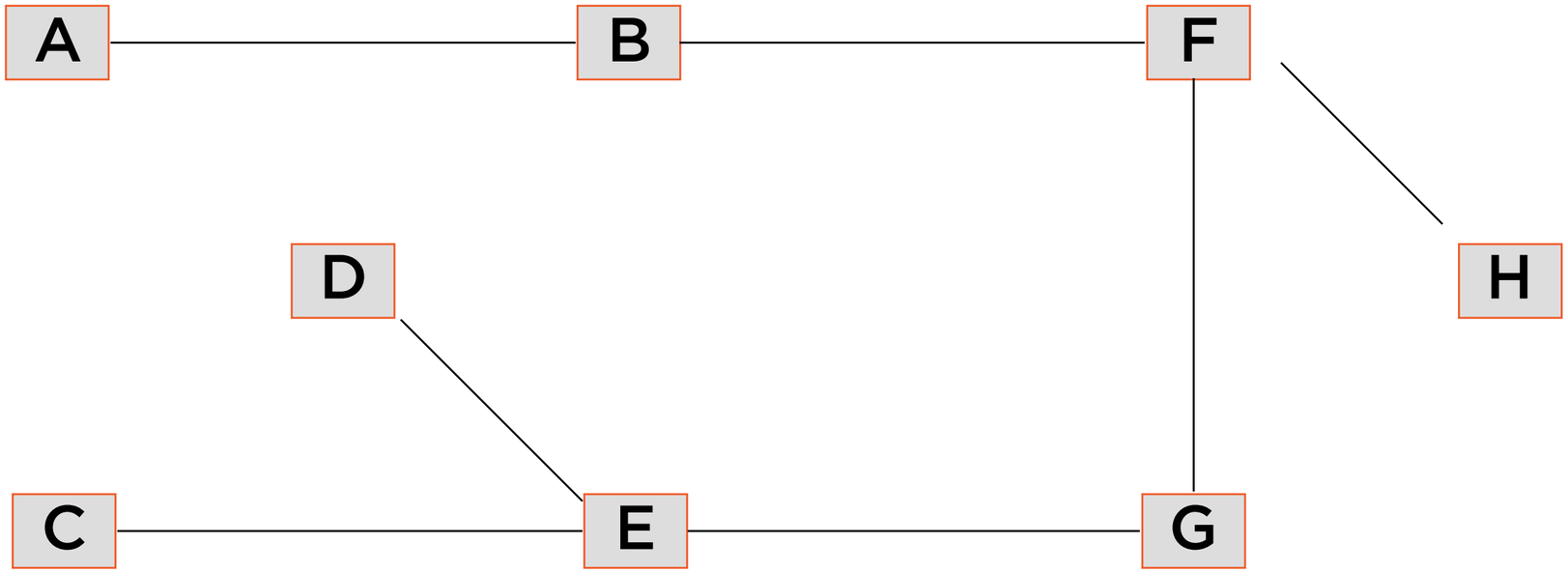


Connected Graph with no Cycle



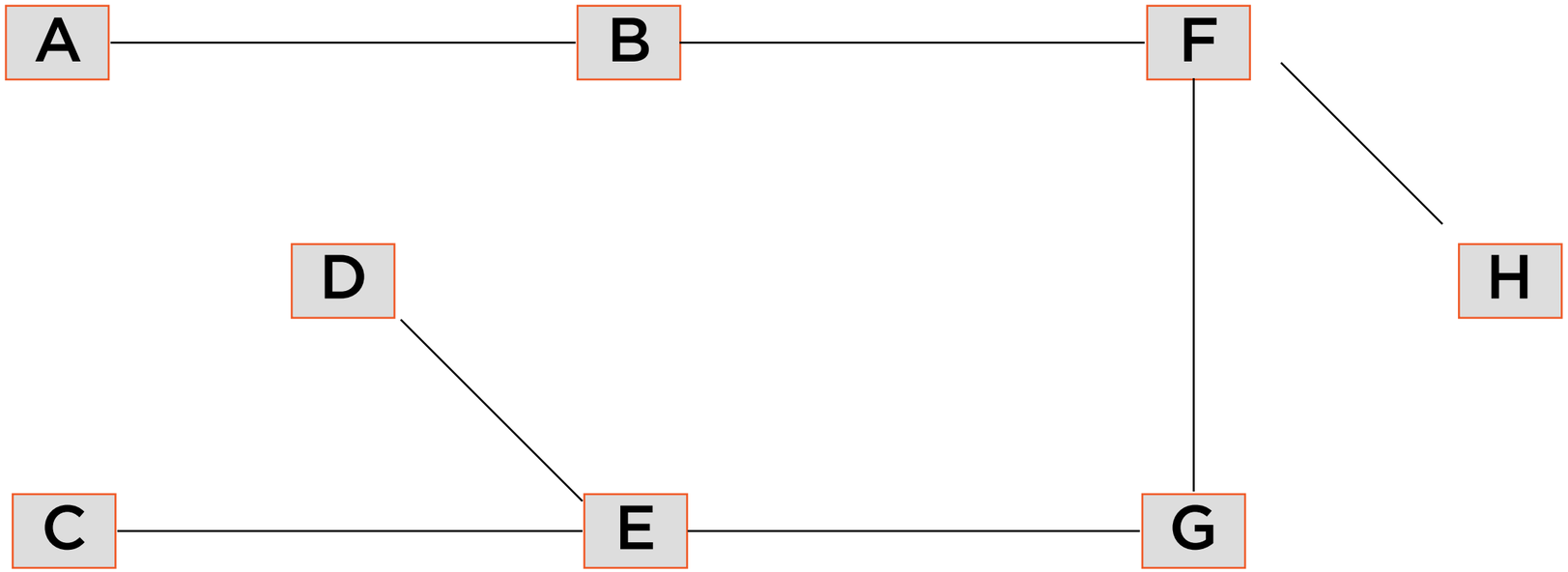
Removing edges A - C, A - D and B - D eliminates the cycles in this graph

Connected Graph with no Cycle



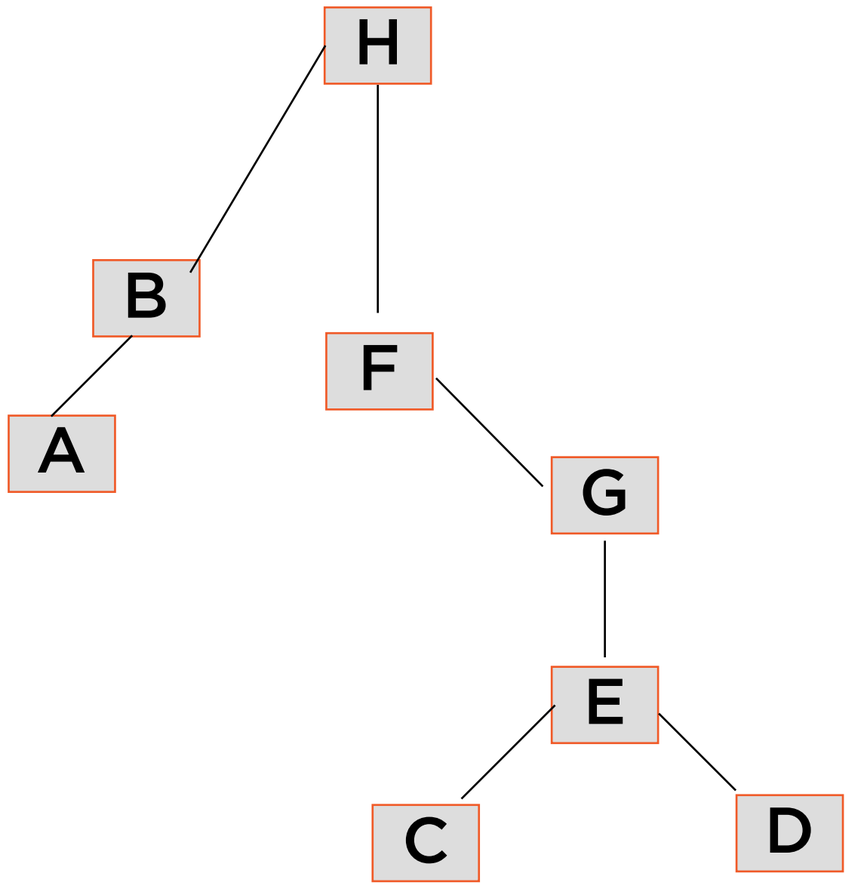
Such a graph is called a tree

Connected Graph with no Cycle



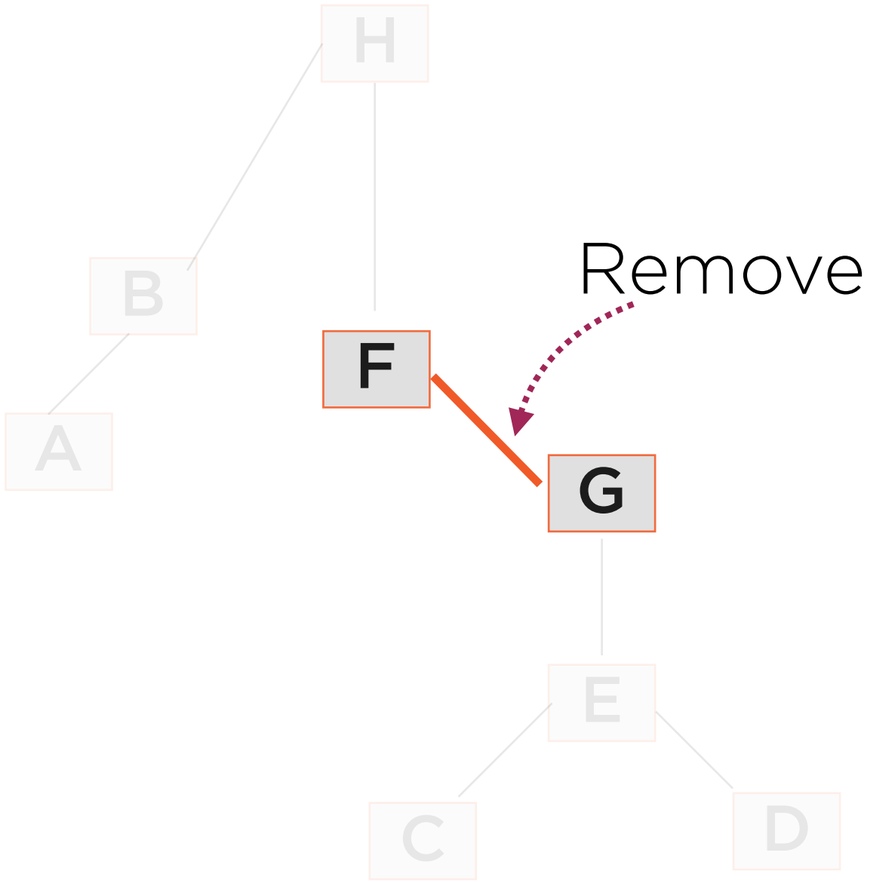
Trees are great for depicting
hierarchical relationships

Forest: Set of Disjoint Trees



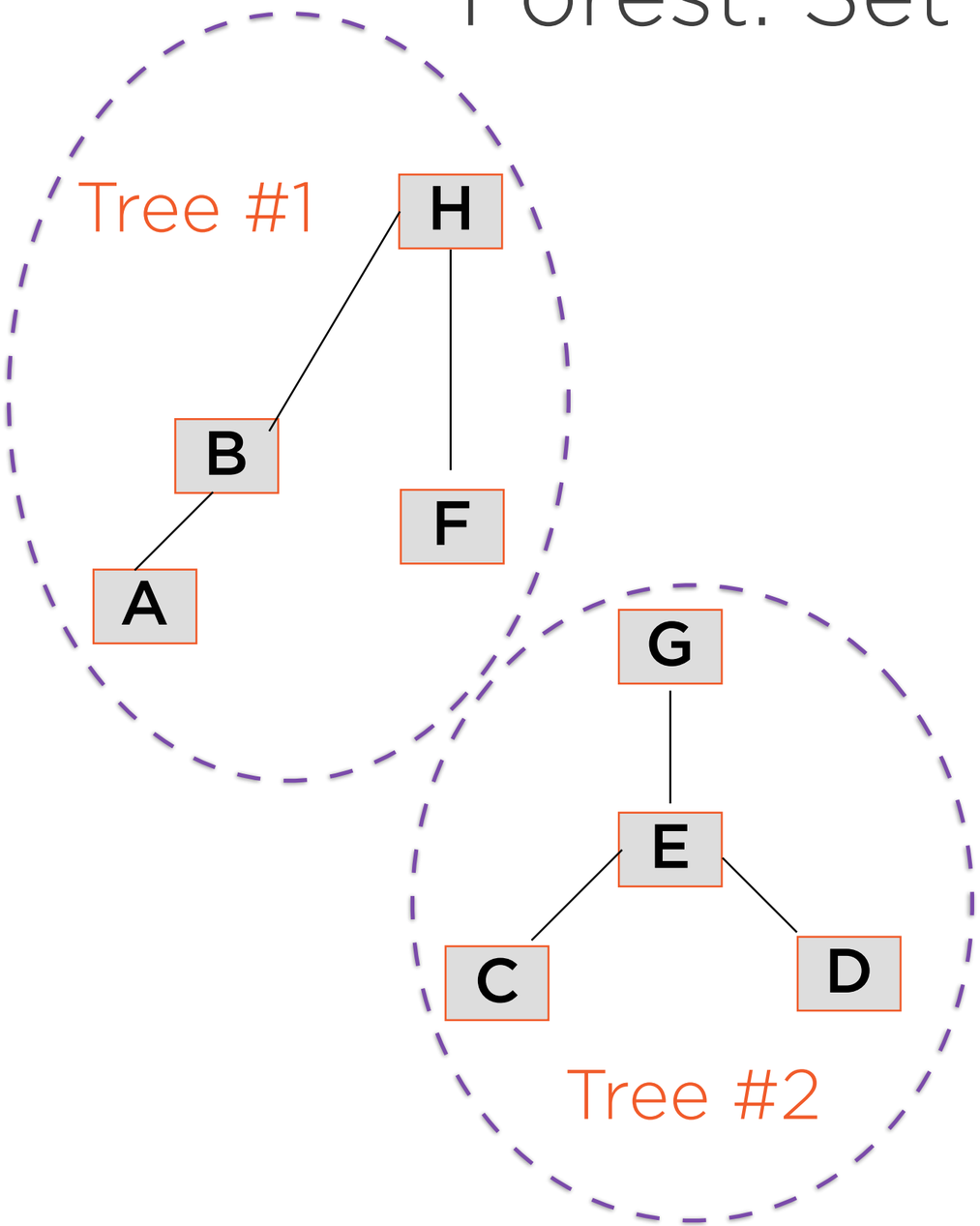
Trees are great for depicting hierarchical relationships

Forest: Set of Disjoint Trees



Removing F - G divides the original graph into two disjoint graphs

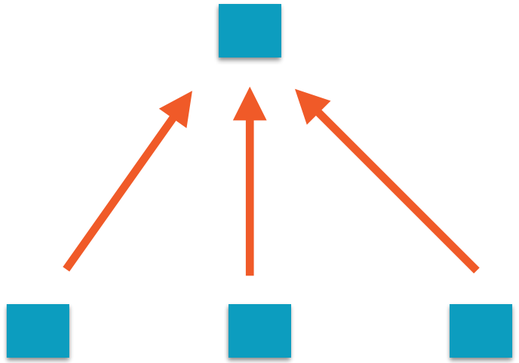
Forest: Set of Disjoint Trees



Such a set of disjoint trees is called a forest

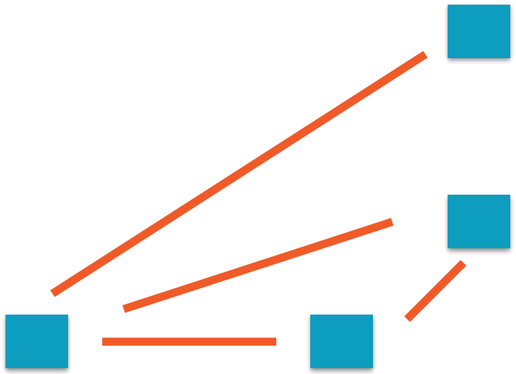
Directed Graphs

Directed and Undirected Graphs



Twitter Followers

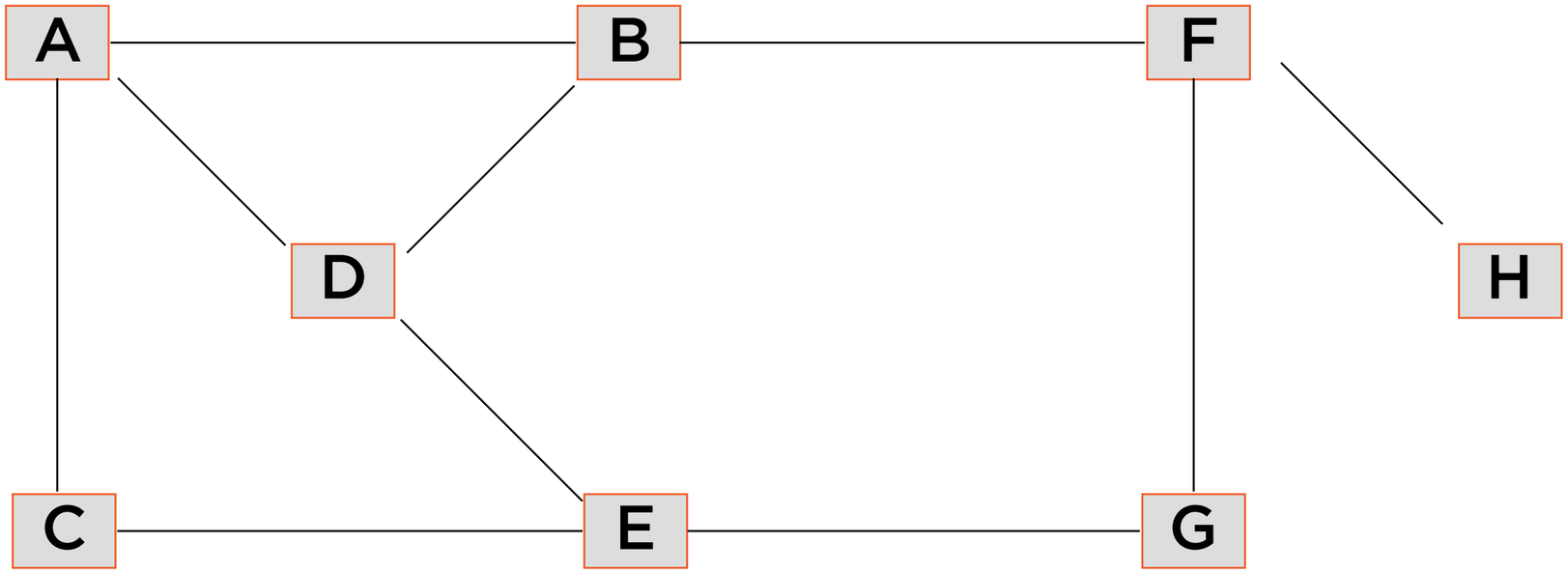
Relationship goes one way only



Facebook Friends

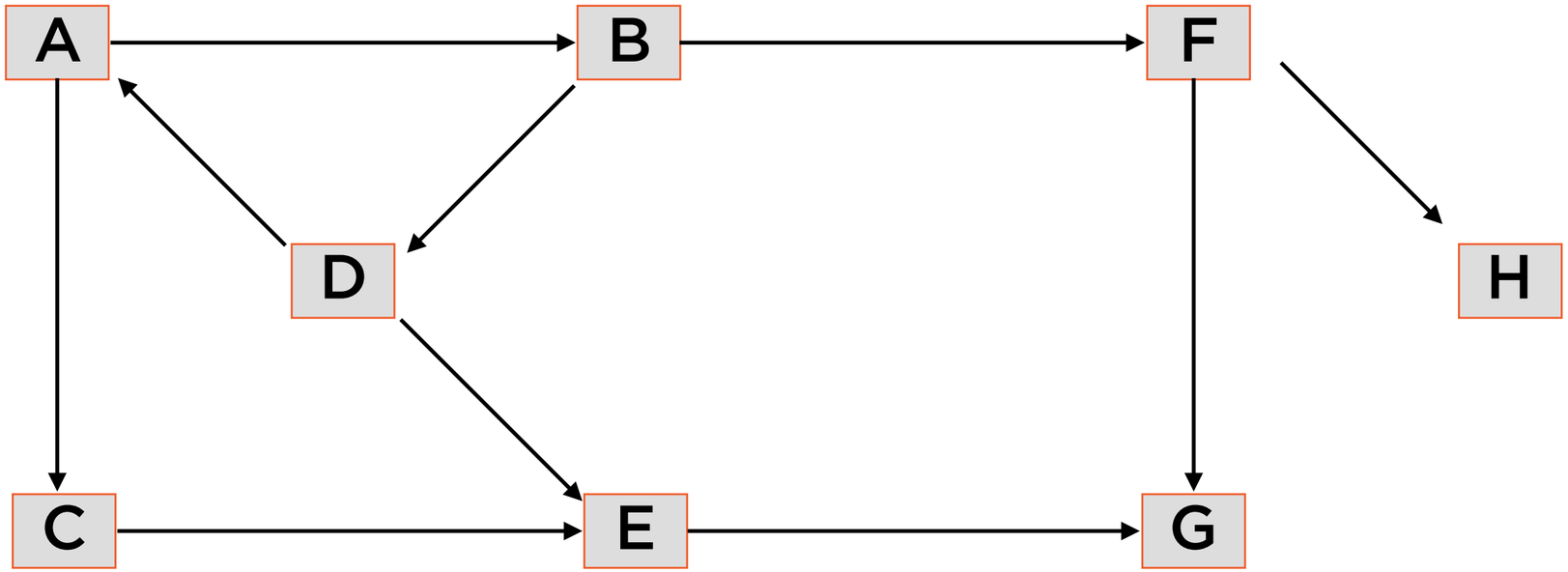
Relationship goes both ways

An Undirected Graph



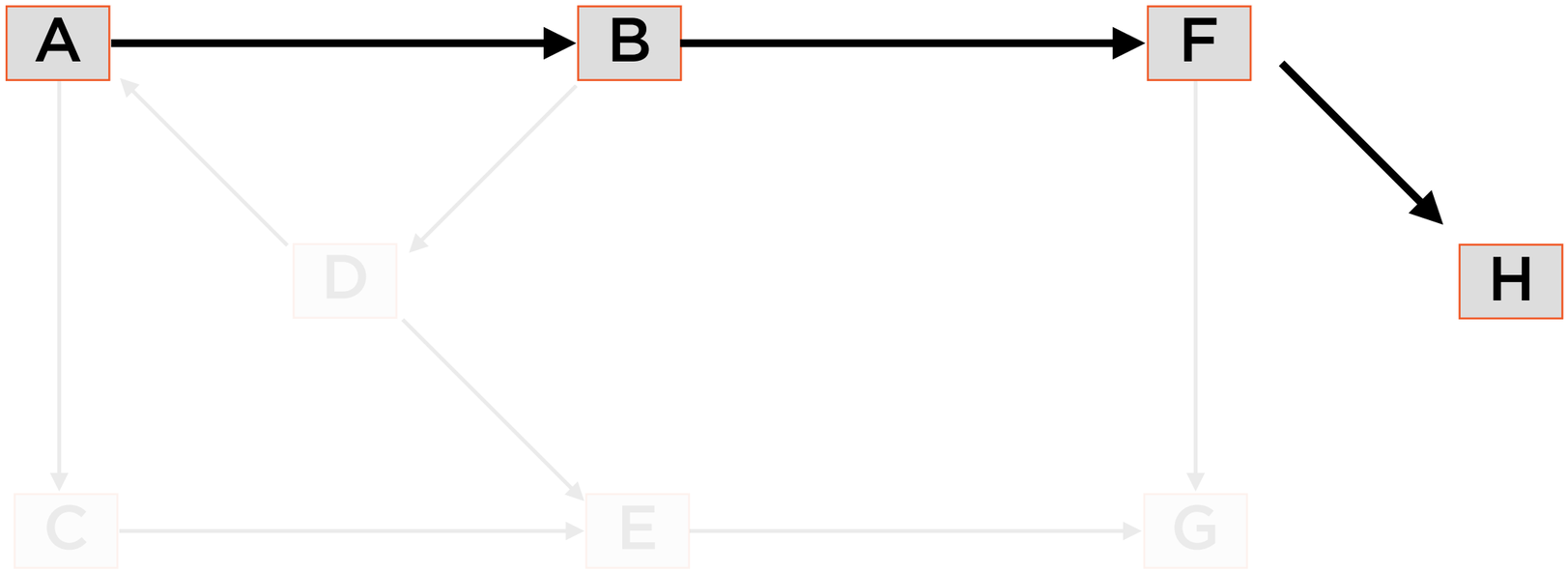
V = {A, B, C, D, E, F, G, H}

A Directed Graph



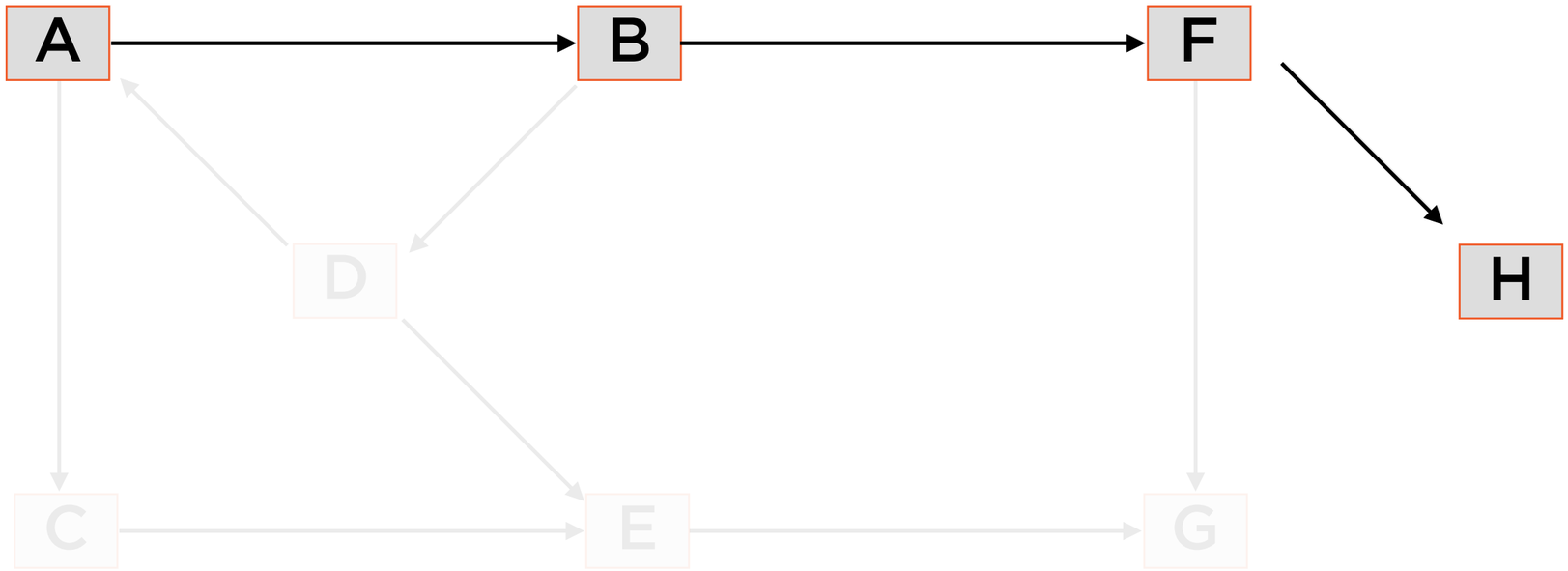
V = {A, B, C, D, E, F, G, H}

Paths in a Graph



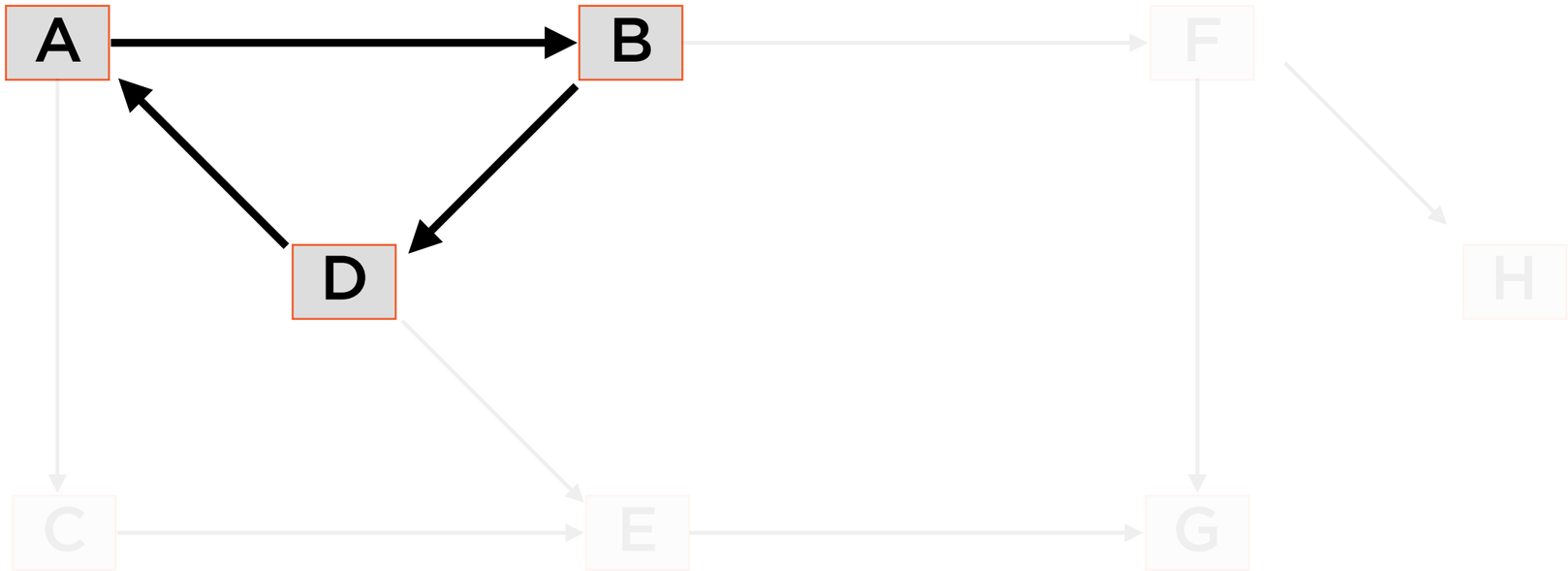
A series of edges links node A to node H - this is called a path

Paths in a Graph



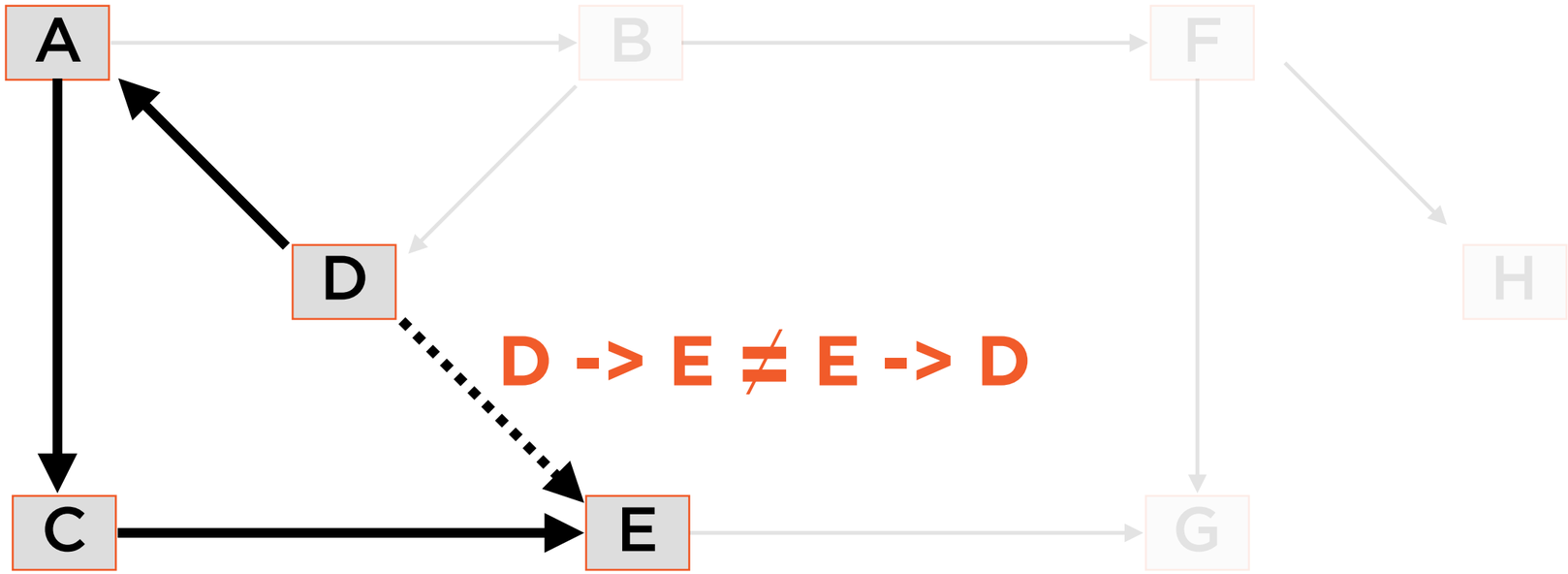
In a directed graph, the path must follow the direction of the arrows

Directed Cyclic Graph



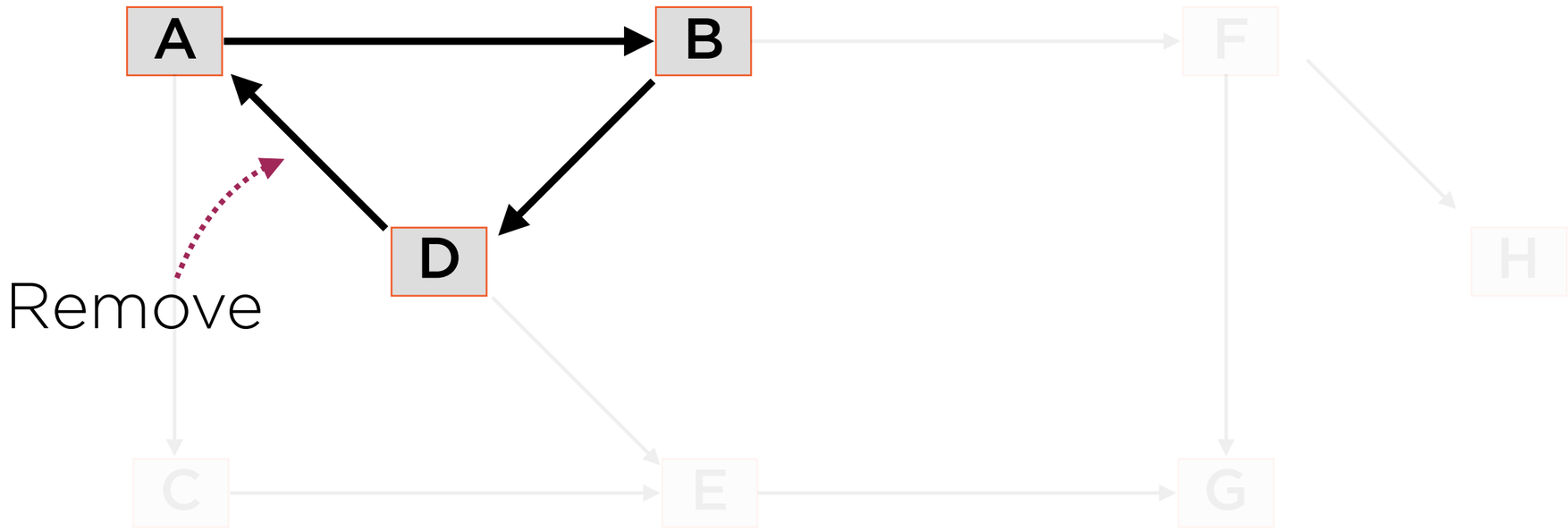
The nodes A,B, D and A form a cycle

Directed Cyclic Graph



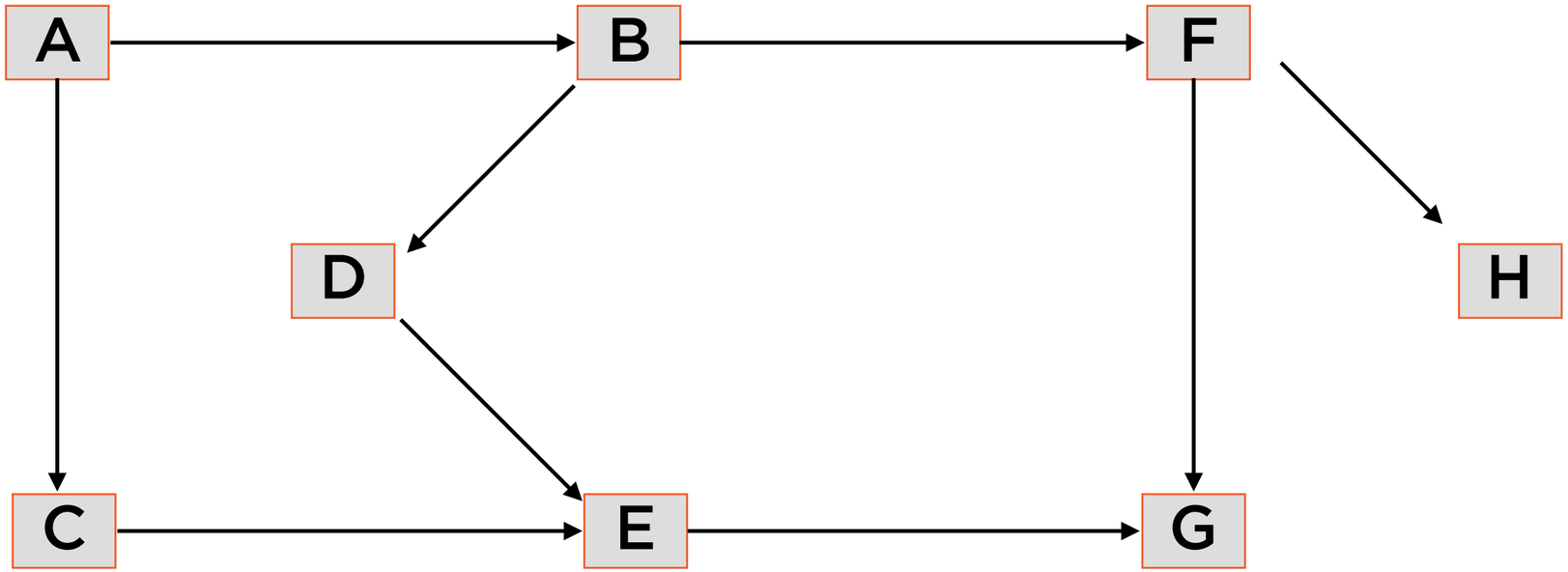
The nodes A, C, E, D and A do not form a cycle

Directed Acyclic Graph



Removing the edge D -> A eliminates the only cycle in this directed graph

Directed Acyclic Graph



Removing the edge D -> A eliminates the only cycle in this directed graph

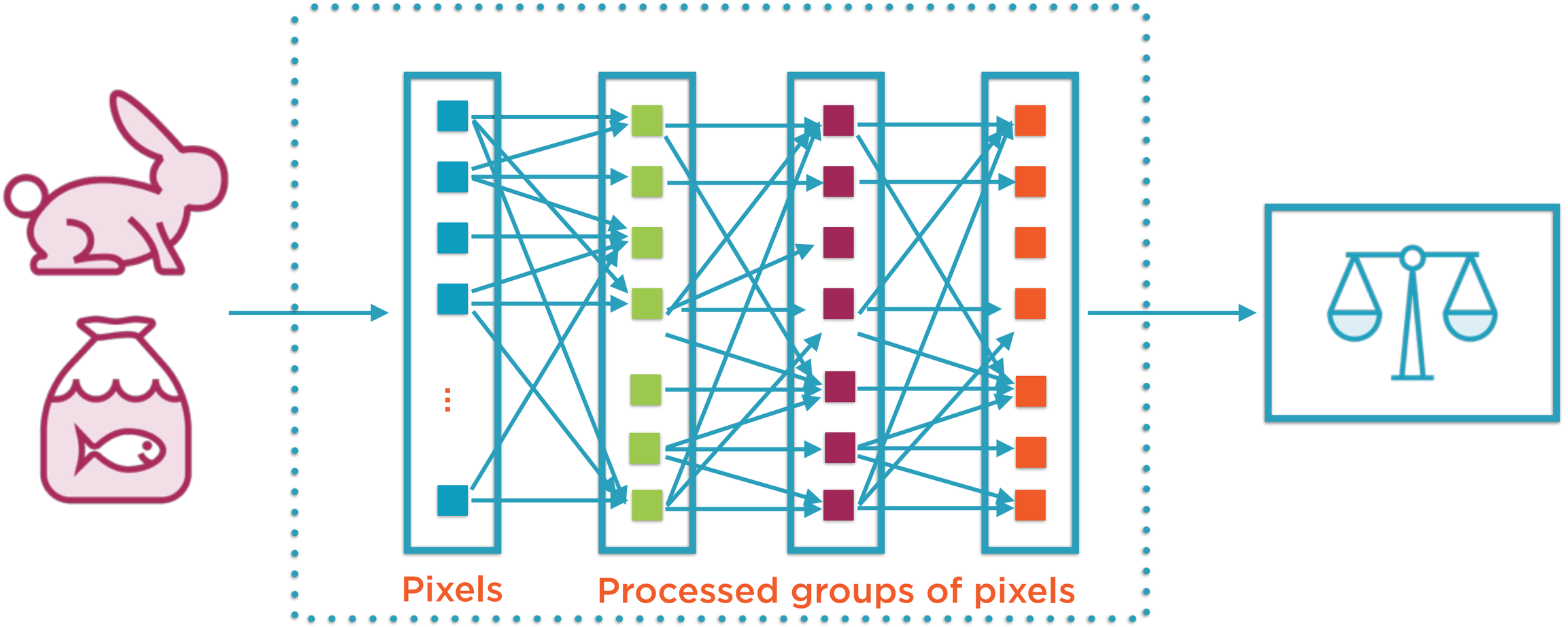
Directed Acyclic Graphs (DAGs)

Especially important type of graph

Common applications

- Scheduling tasks
- Evaluating expressions

Neural Network Computation Graph



Corpus of Images

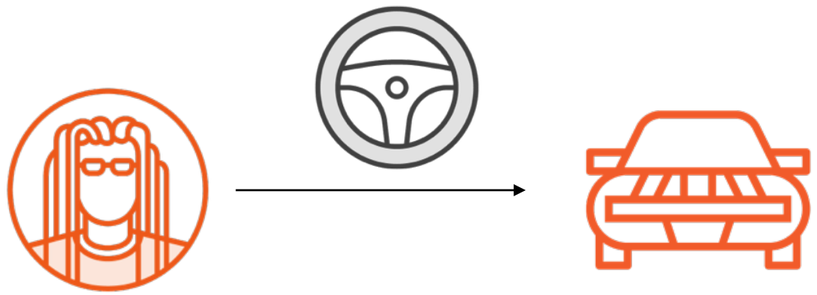
Operations (nodes) on data (edges)

ML-based Classifier

Adjacency Matrices

Graph (V,E)

A set of vertices (V) and edges (E)

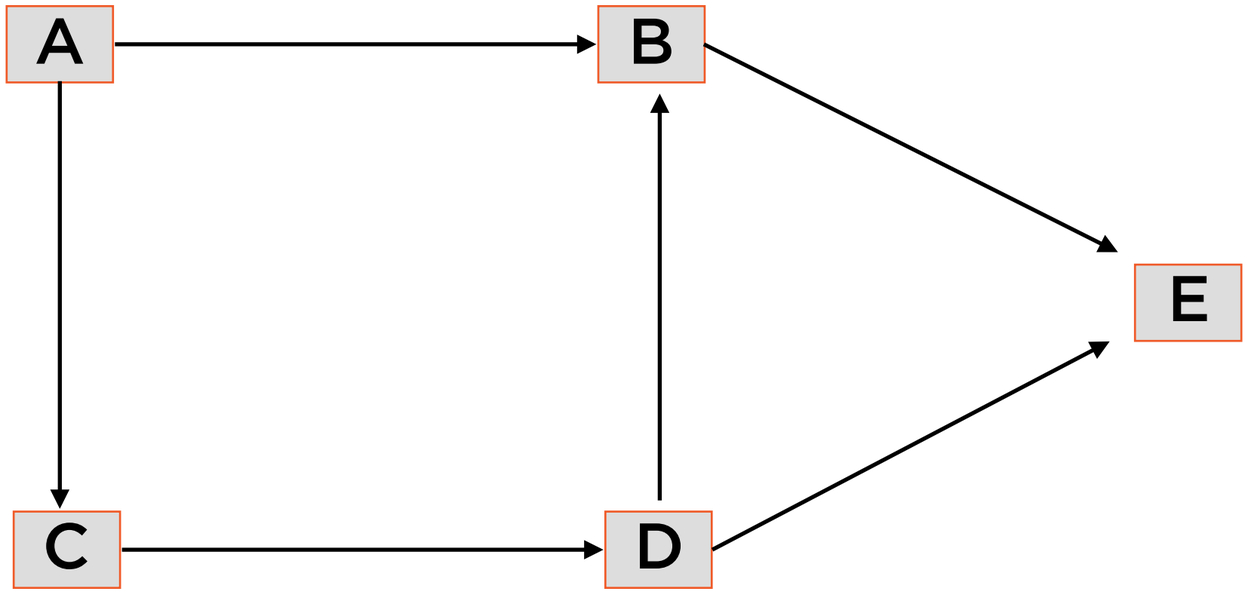


Three ways to represent graphs in code

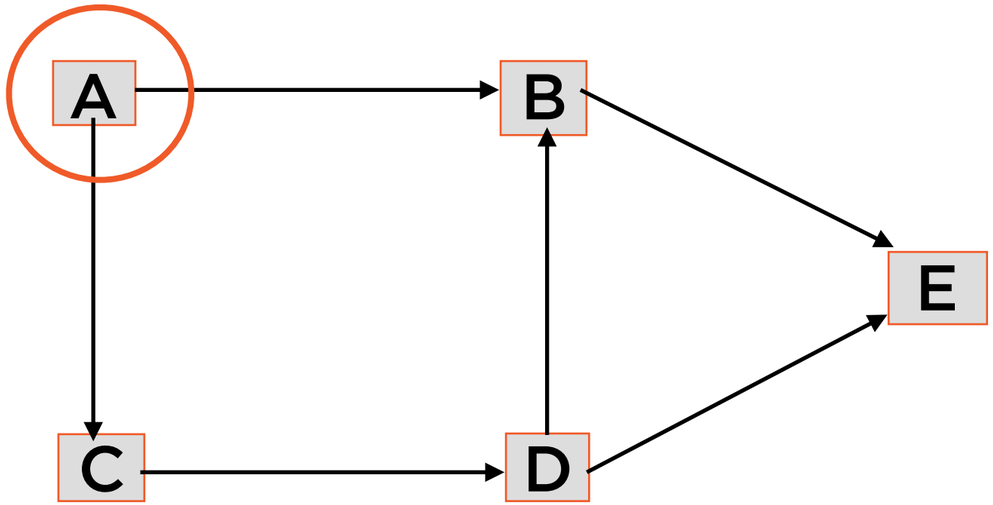
- Adjacency matrices
- Adjacency lists
- Adjacency sets

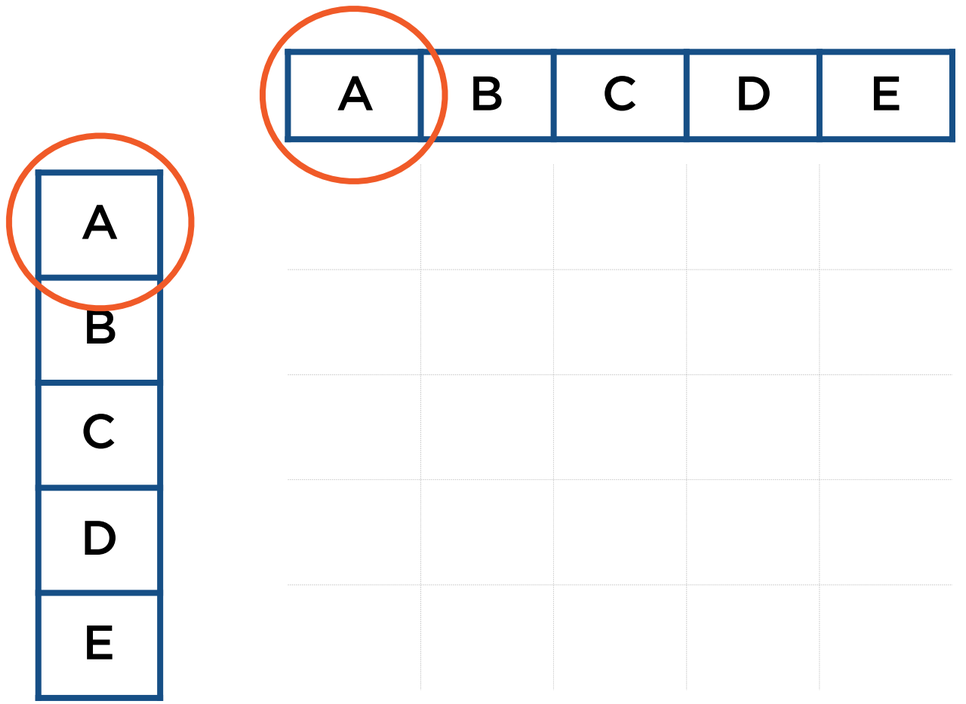
The adjacency matrix of a graph with N nodes is an N x N matrix

Adjacency Matrix for a Directed Graph



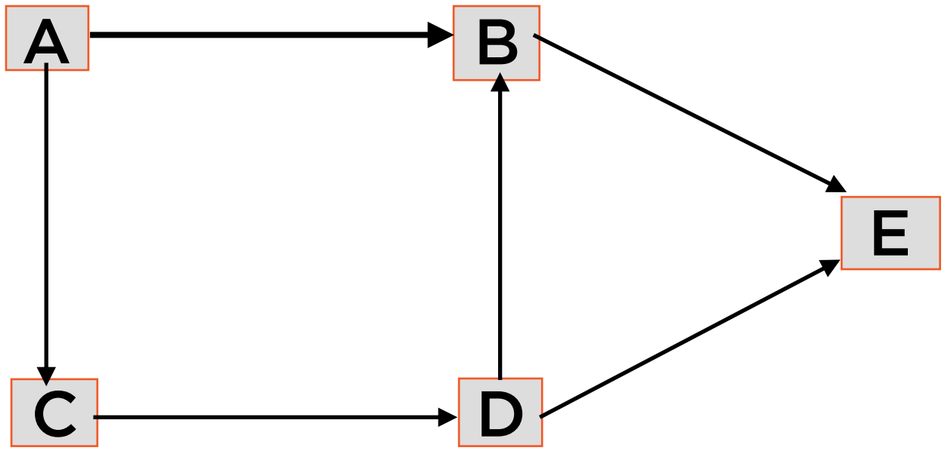
Adjacency Matrix for a Directed Graph





Each node has a corresponding row and column

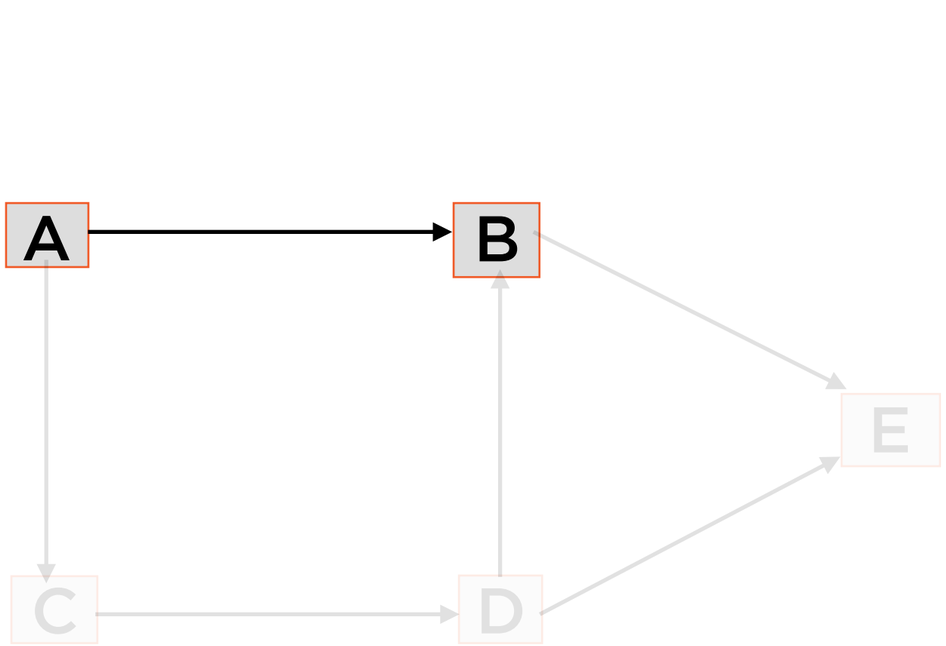
Adjacency Matrix for a Directed Graph

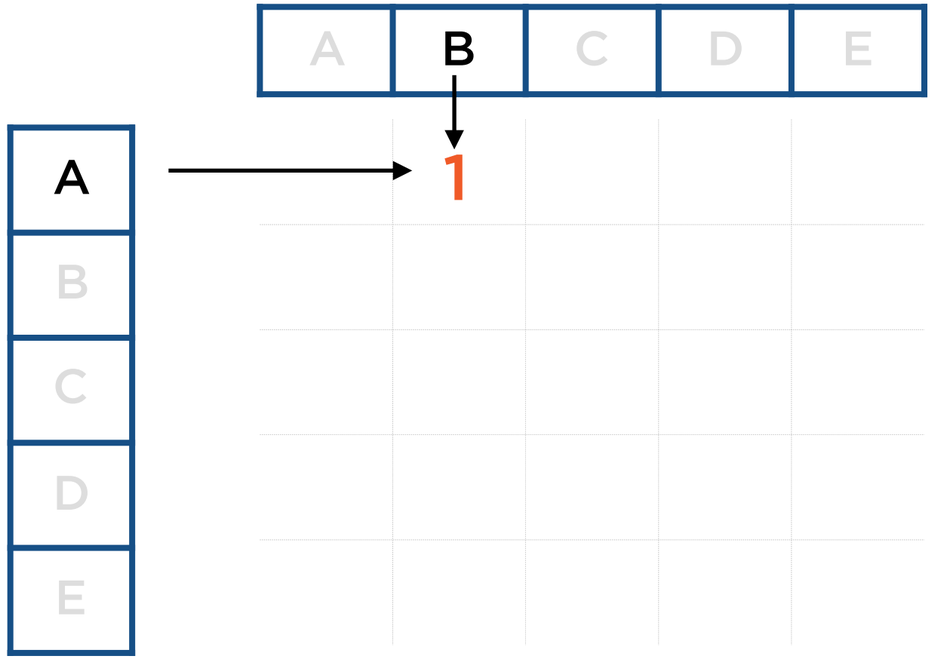


	A	B	C	D	E
A					
B					
C					
D					
E					

Value of 1 in (Row A, Column B) indicates an edge from A to B

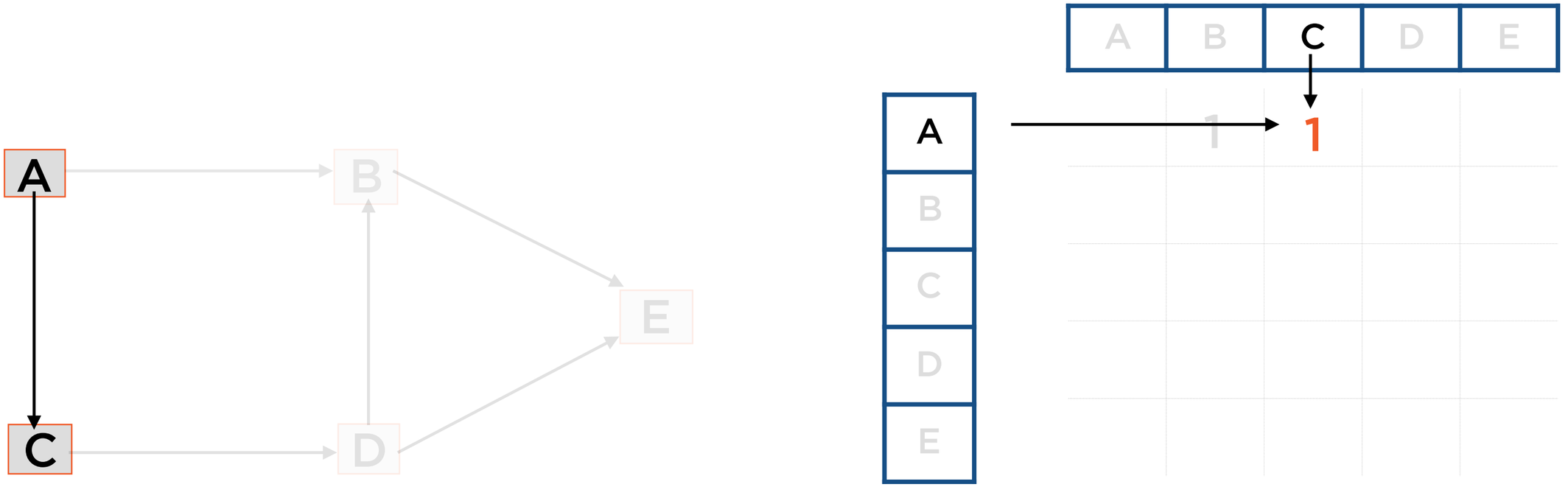
Adjacency Matrix for a Directed Graph





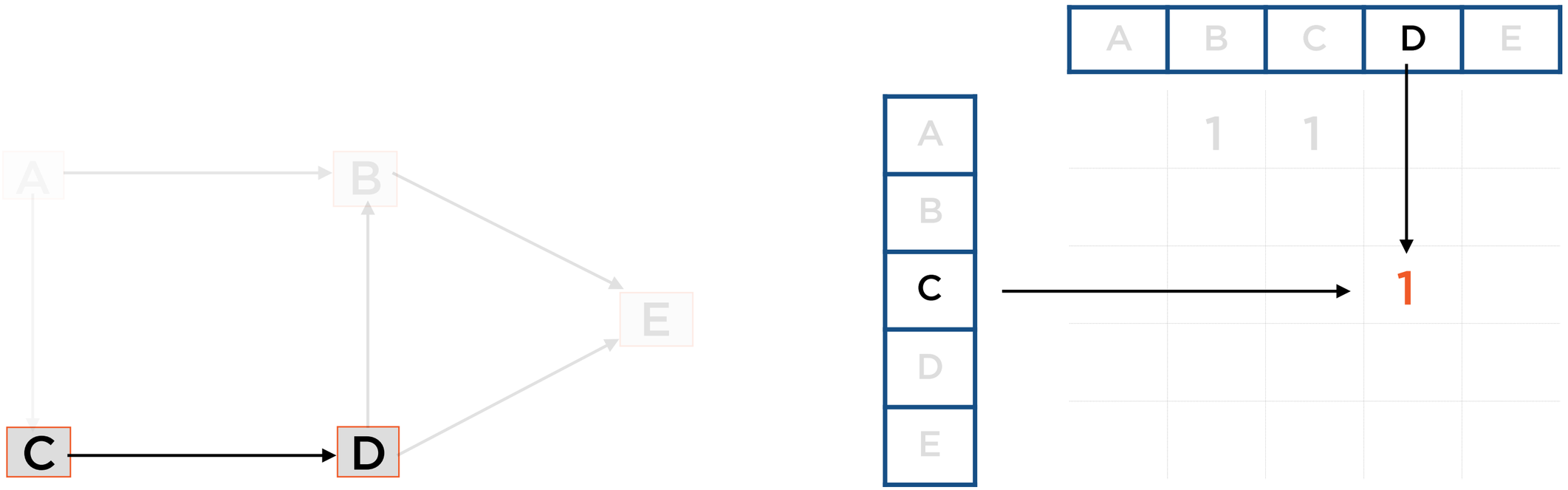
Value of 1 in (Row A, Column B) indicates an edge from A to B

Adjacency Matrix for a Directed Graph



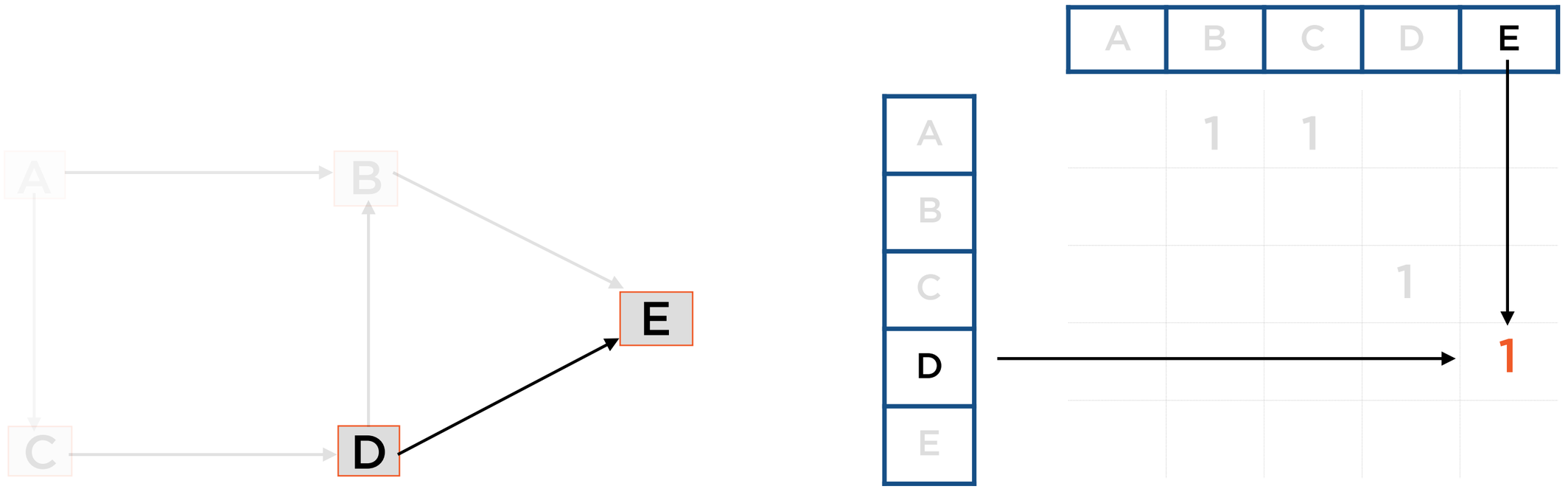
Edge A to C (Row A, Column C)

Adjacency Matrix for a Directed Graph



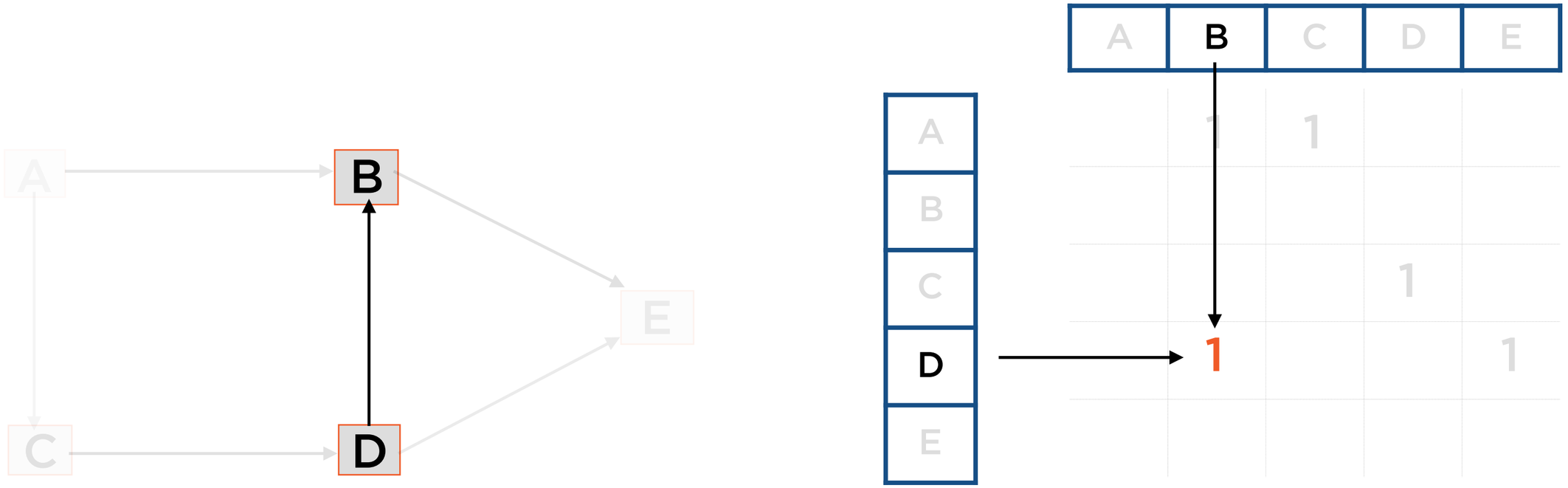
Edge C to D (Row C, Column D)

Adjacency Matrix for a Directed Graph



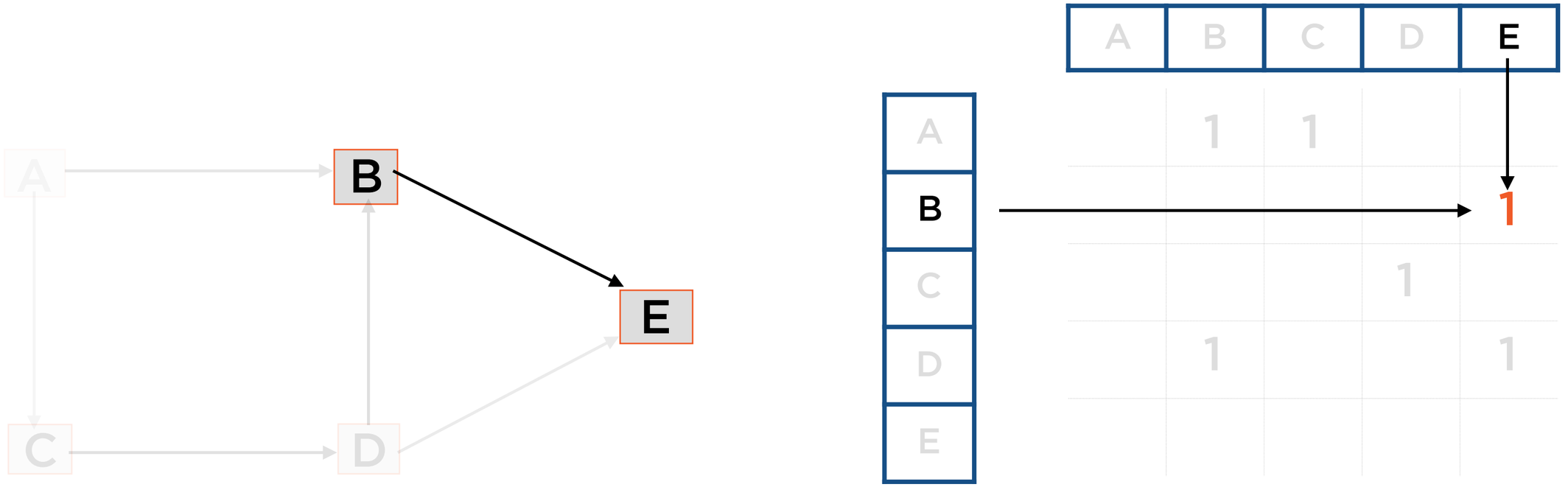
Edge D to E (Row D, Column E)

Adjacency Matrix for a Directed Graph



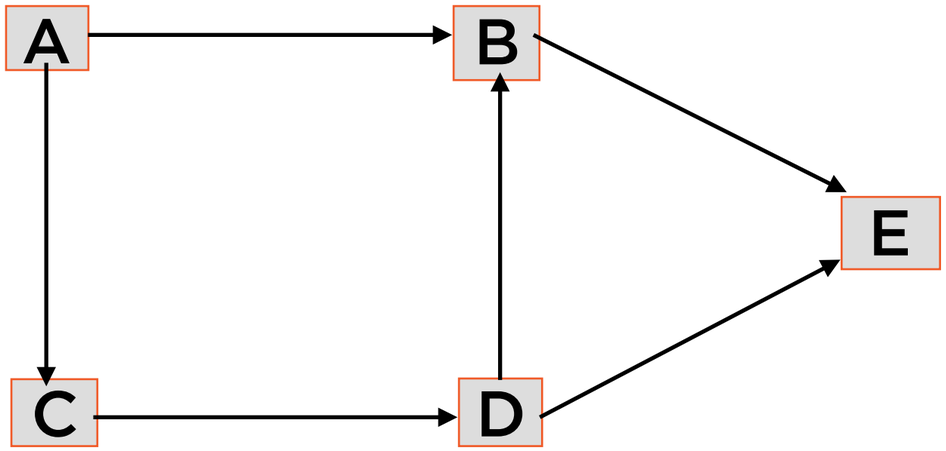
Edge D to B (Row D, Column B)

Adjacency Matrix for a Directed Graph



Edge B to E (Row B, Column E)

Adjacency Matrix for a Directed Graph

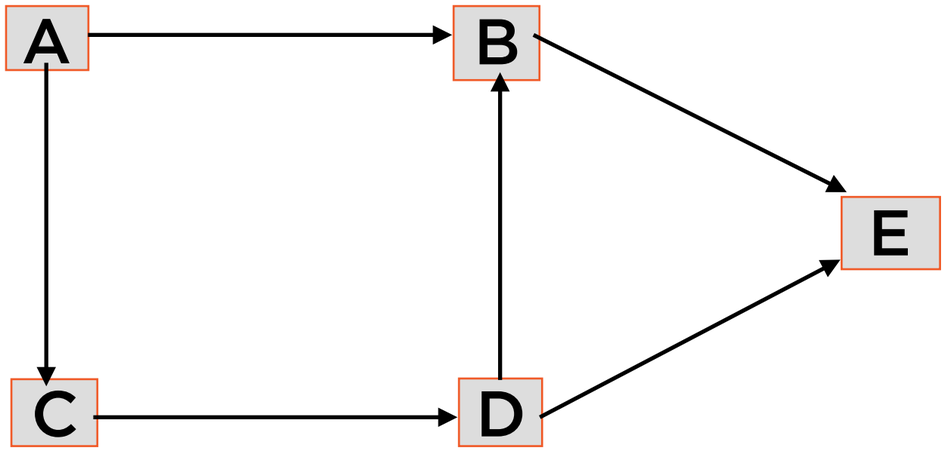


A
B
C
D
E

A	B	C	D	E
	1	1		
				1
			1	
	1			1

Six edges in graph => six 1s in adjacency matrix

Adjacency Matrix for a Directed Graph

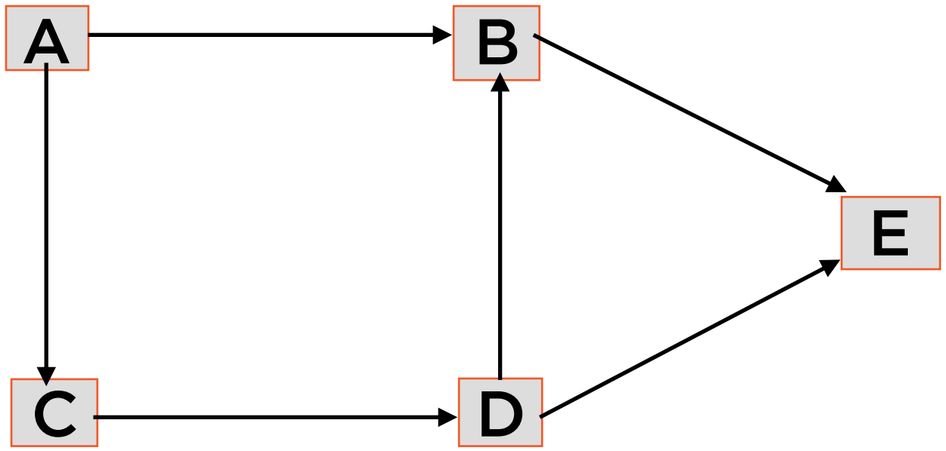


A
B
C
D
E

A	B	C	D	E
0	1	1	0	0
0	0	0	0	1
0	0	0	1	0
0	1	0	0	1
0	0	0	0	0

All other elements are zero

Adjacency Matrix for a Directed Graph

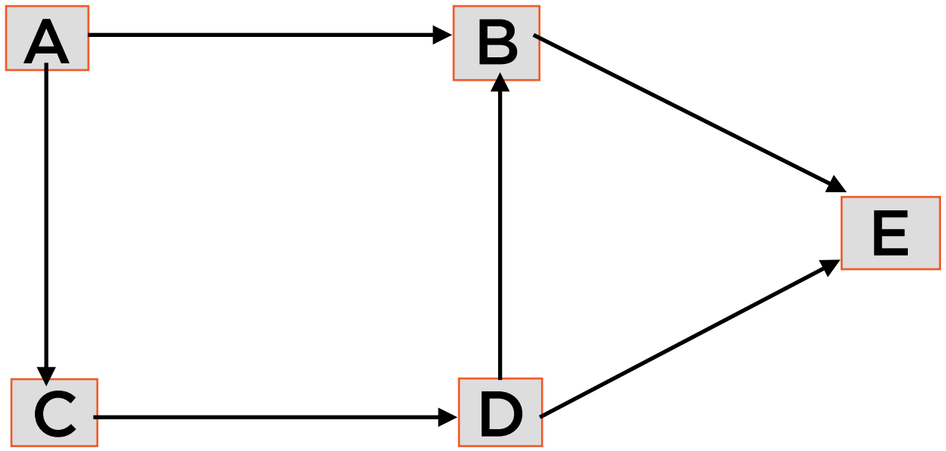


A
B
C
D
E

A	B	C	D	E
0	1	1	0	0
0	0	0	0	1
0	0	0	1	0
0	1	0	0	1
0	0	0	0	0

All diagonal elements are zero (since nodes are not connected to themselves)

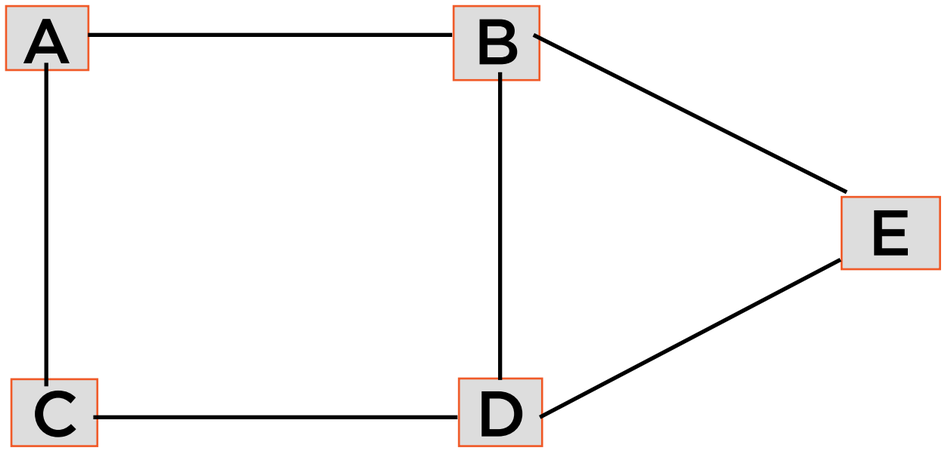
Adjacency Matrix for a Directed Graph



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	1	0	0	1
E	0	0	0	0	0

The adjacency matrix of a Directed Graph is not symmetric

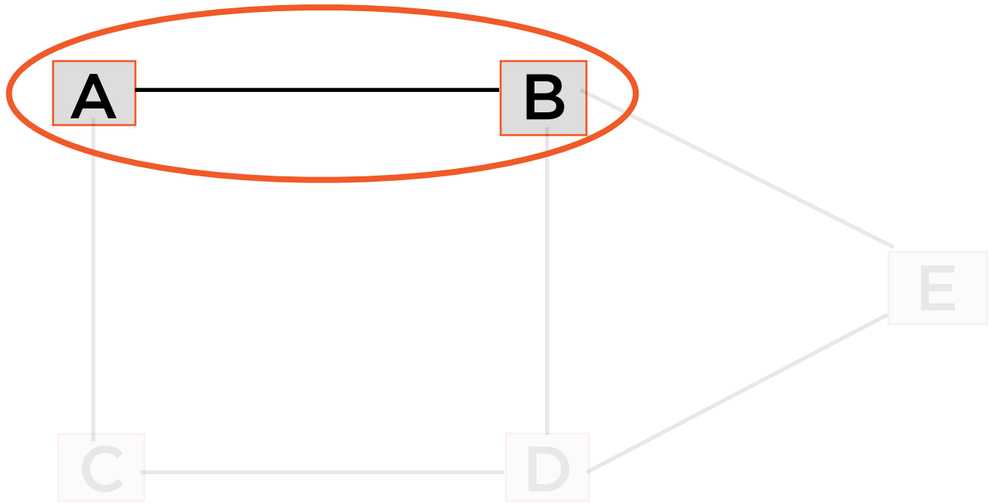
Adjacency Matrix for an Undirected Graph

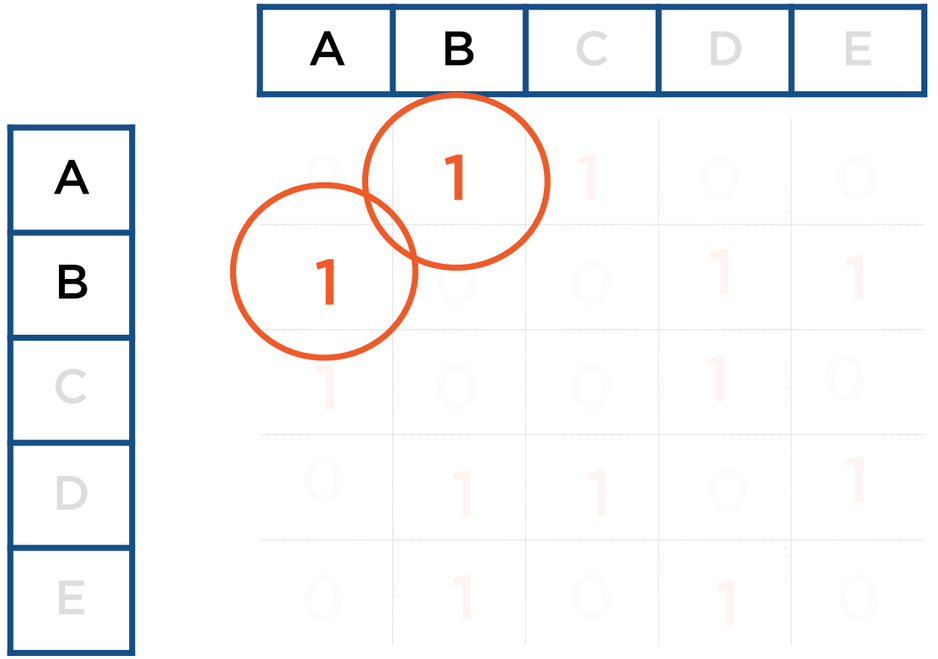


	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	1
C	1	0	0	1	0
D	0	1	1	0	1
E	0	1	0	1	0

Value(Row i, Column j) == Value(Row j, column i)

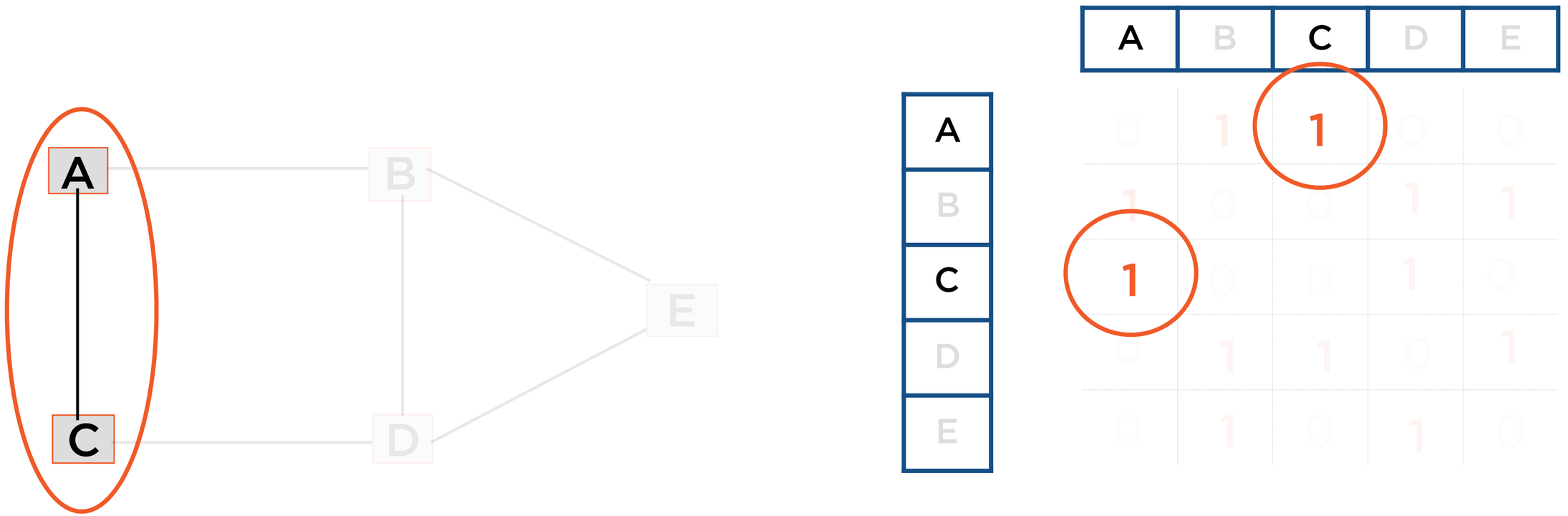
Adjacency Matrix for an Undirected Graph





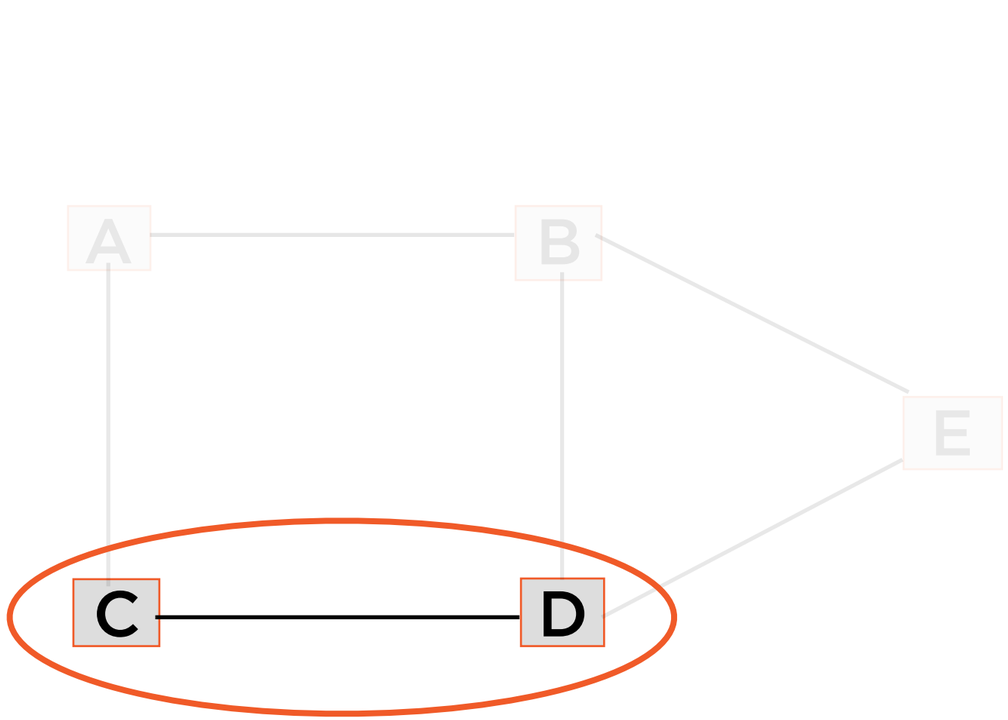
Value(Row i, Column j) == Value(Row j, column i)

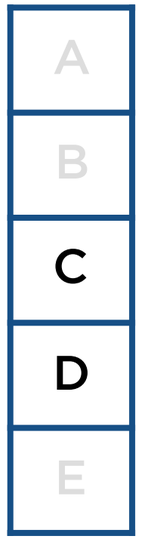
Adjacency Matrix for an Undirected Graph

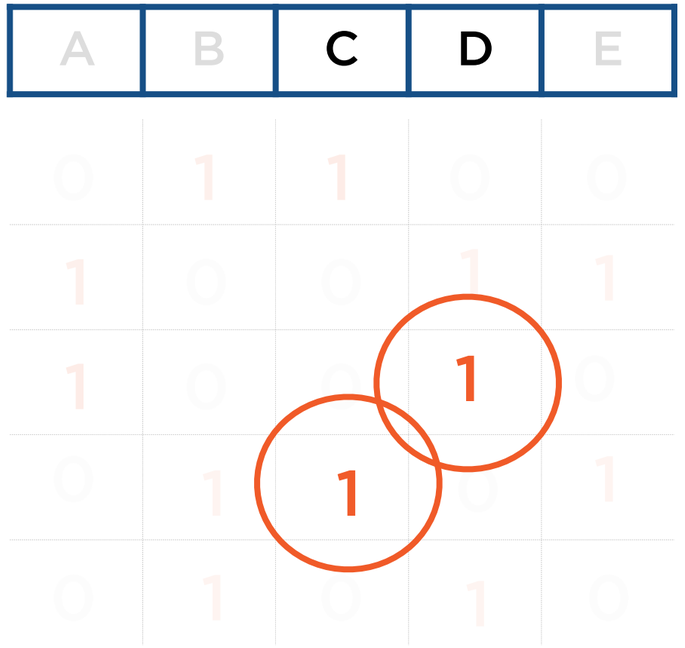


Value(Row i, Column j) == Value(Row j, column i)

Adjacency Matrix for an Undirected Graph

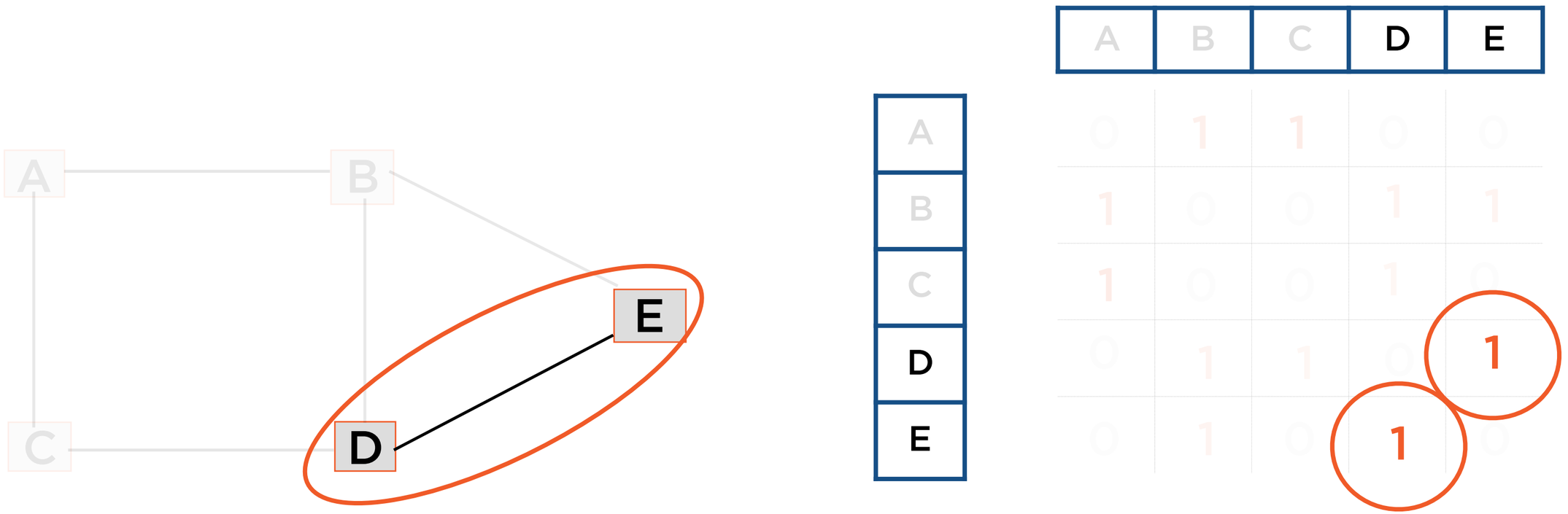






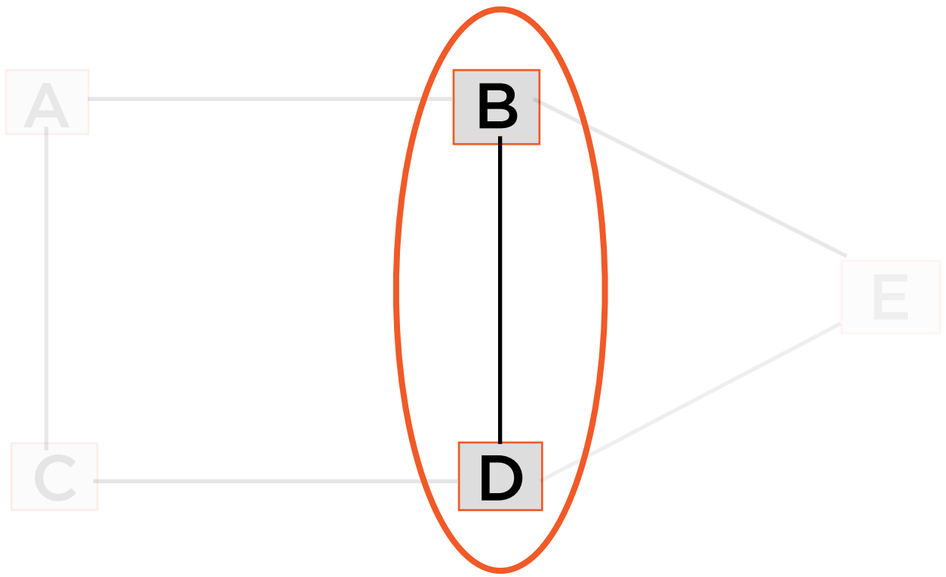
Value(Row i, Column j) == Value(Row j, column i)

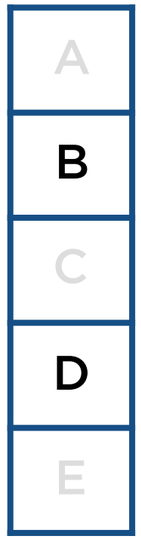
Adjacency Matrix for an Undirected Graph

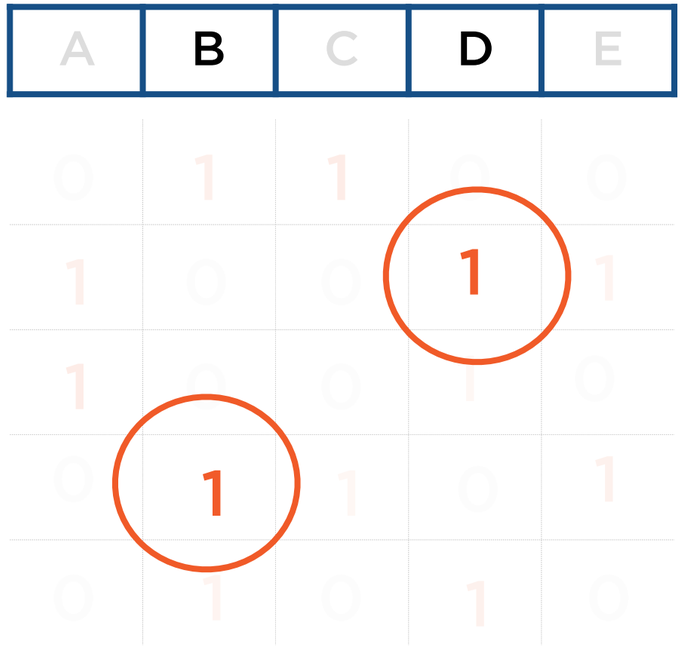


Value(Row i, Column j) == Value(Row j, column i)

Adjacency Matrix for an Undirected Graph

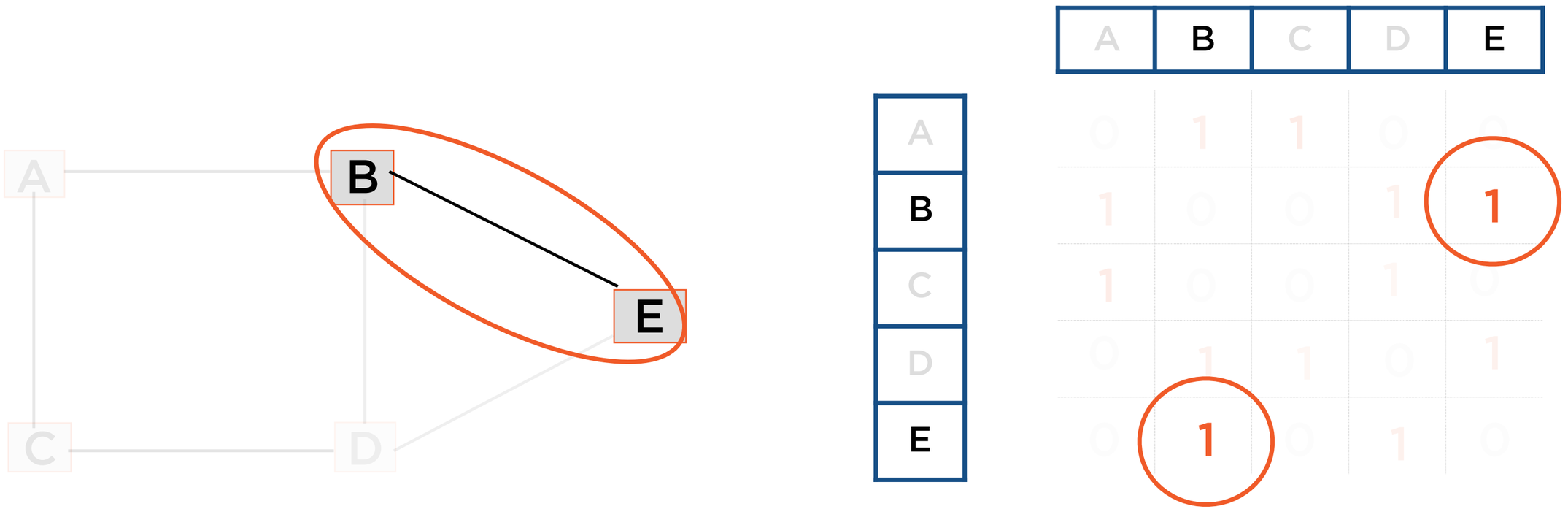






Value(Row i, Column j) == Value(Row j, column i)

Adjacency Matrix for an Undirected Graph



Value(Row i, Column j) == Value(Row j, column i)

The adjacency matrix of an **undirected** graph is **symmetric**

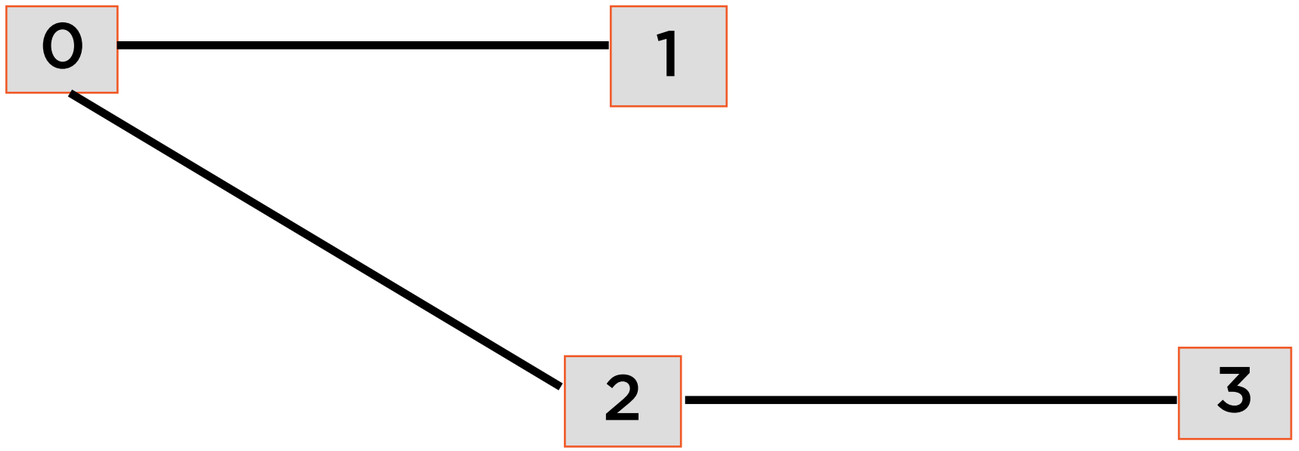
The adjacency matrix of a **directed** graph is **asymmetric**

Demo

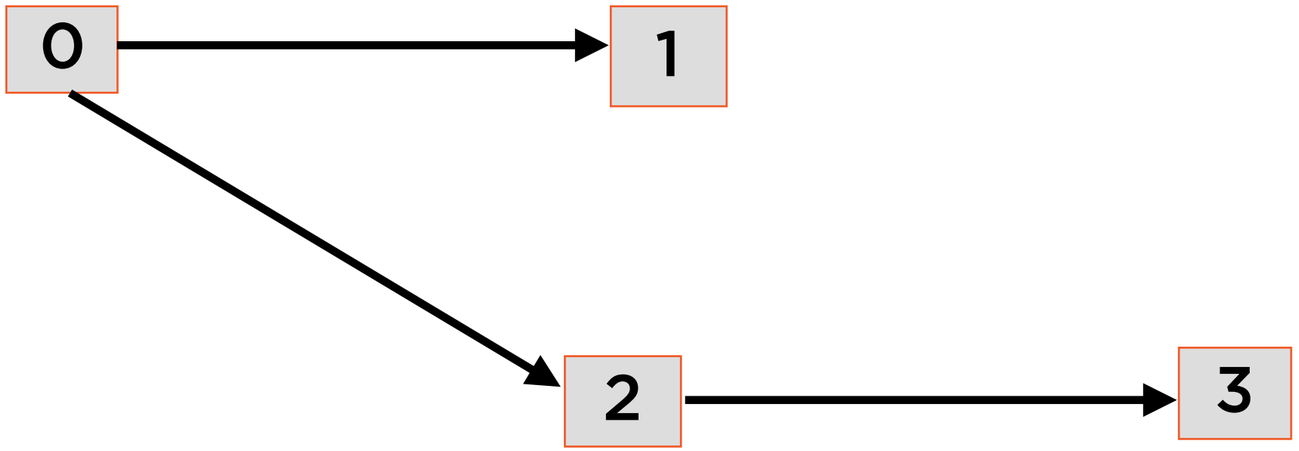
Set up a abstract base class to represent a Graph

Represent a Graph as an adjacency matrix

A Sample Undirected Graph



A Sample Directed Graph

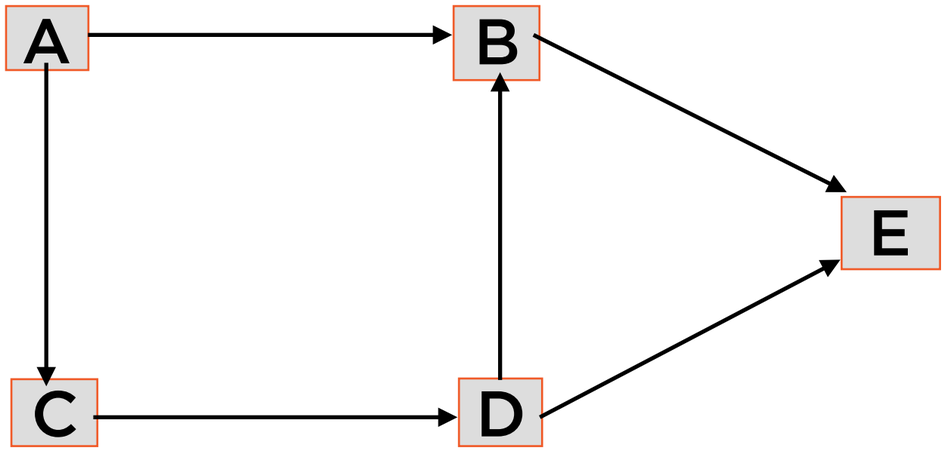


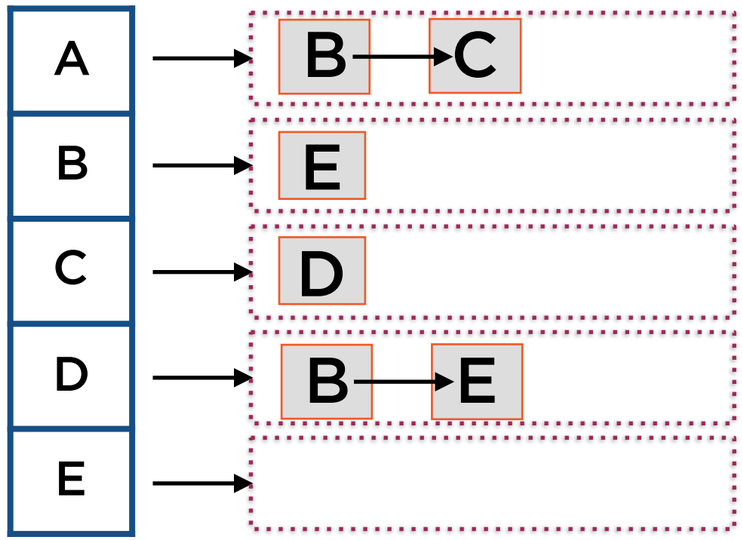
Adjacency Lists and Adjacency Sets

Adjacency List Representation

Each node maintains a linked list of its adjacent nodes

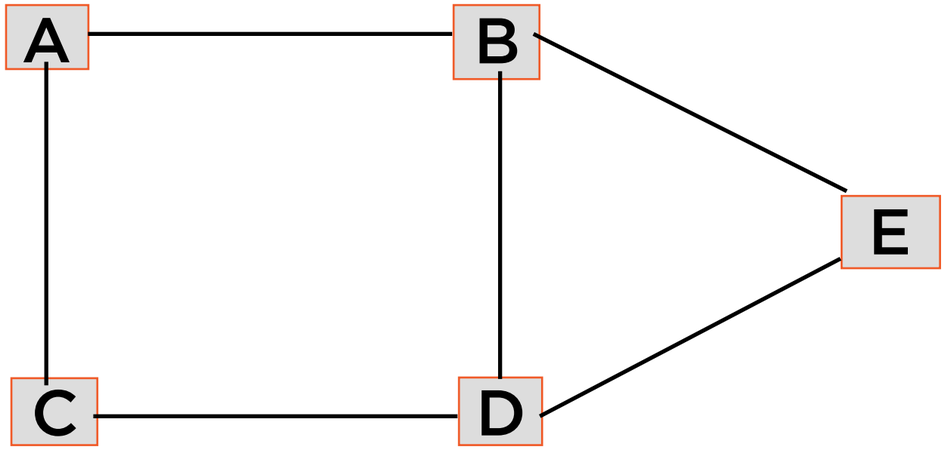
Adjacency List for a Directed Graph

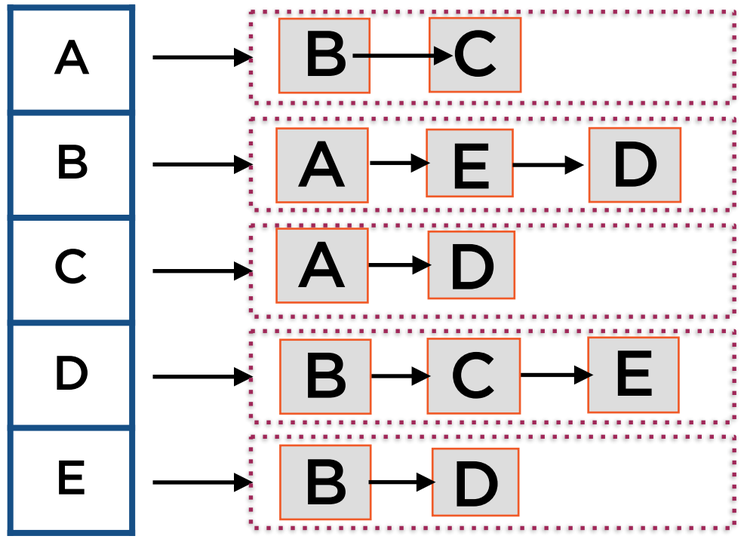




Each node maintains a linked list of adjacent nodes

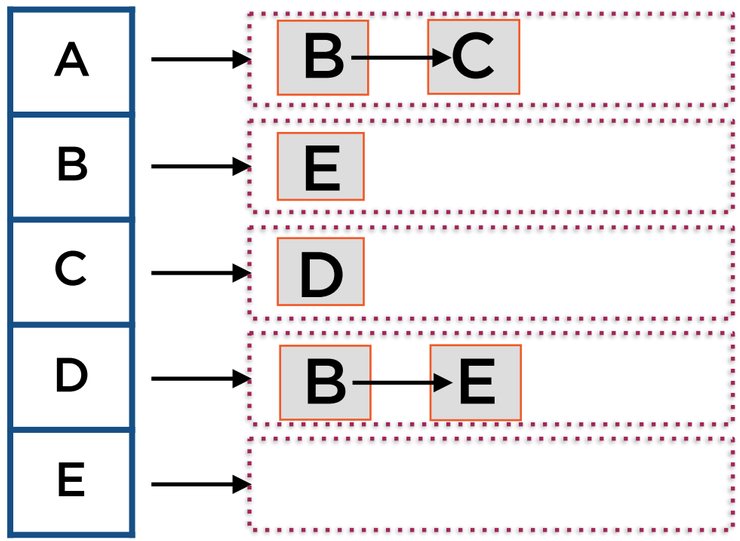
Adjacency List for an Undirected Graph





Each node maintains a linked list of adjacent nodes

Adjacency List Flaws

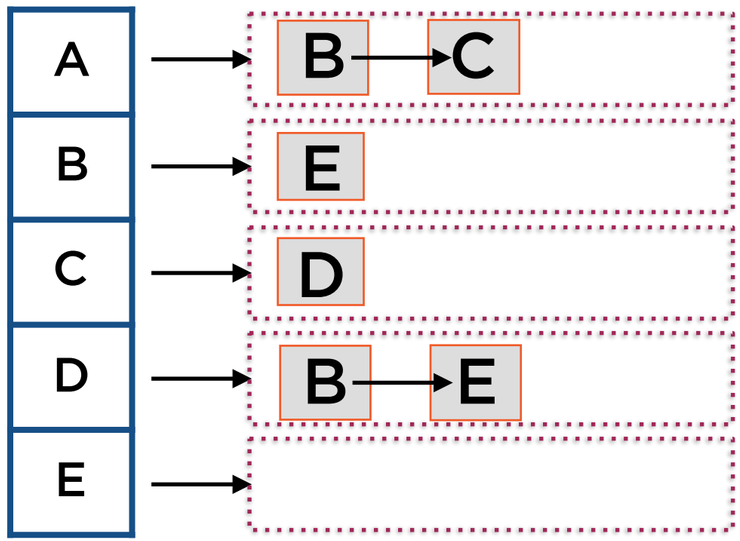


In a list, order matters

Thus, the same graph can have multiple representations

Deletion of a node is inefficient

Requires iterations through all adjacency lists



Adjacency lists have some serious flaws

Adjacency sets help address them

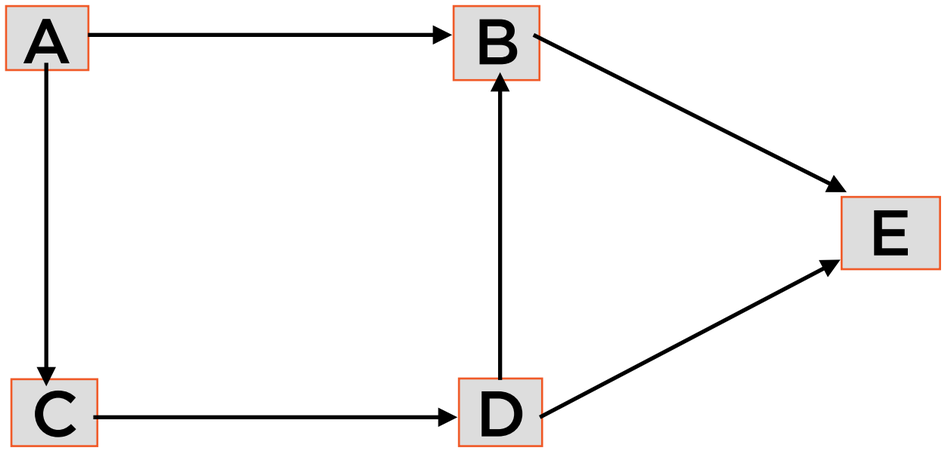
Adjacency List Representation

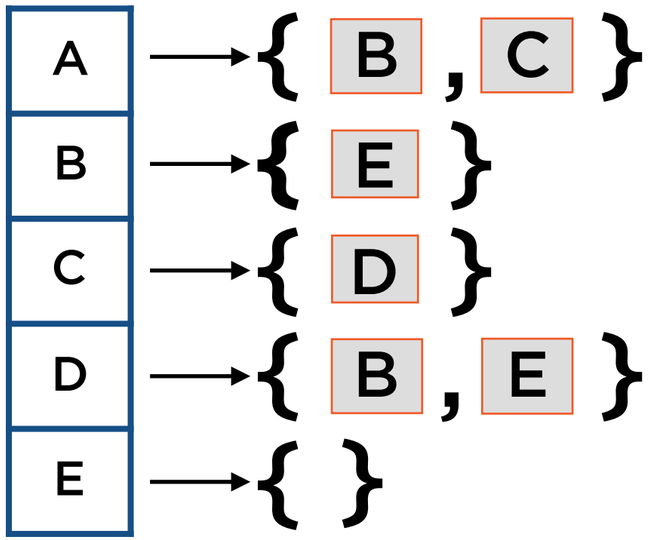
Each node maintains a linked list of its adjacent nodes

Adjacency Set Representation

Each node maintains a set of its adjacent nodes

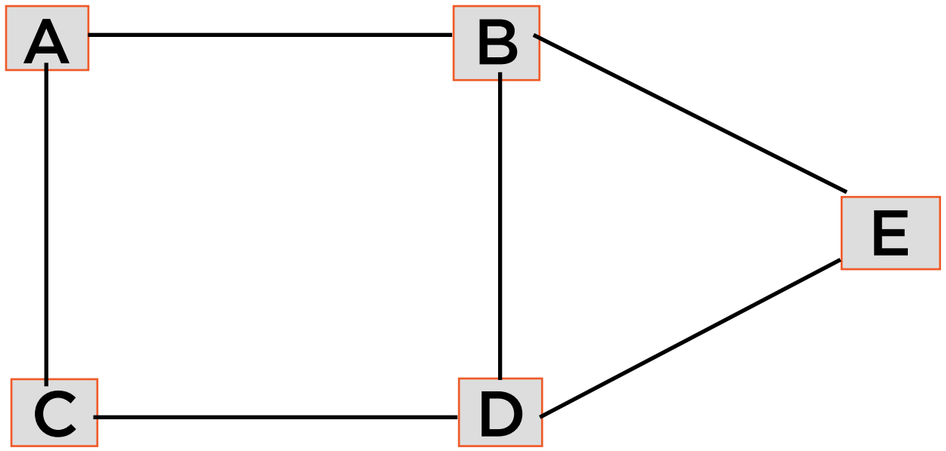
Adjacency Set for a Directed Graph

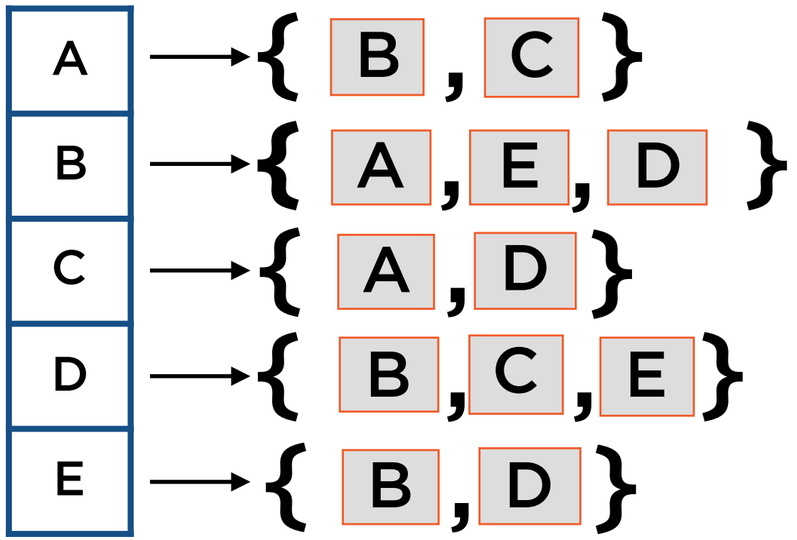




Each node maintains a set of adjacent nodes

Adjacency Set for a Undirected Graph



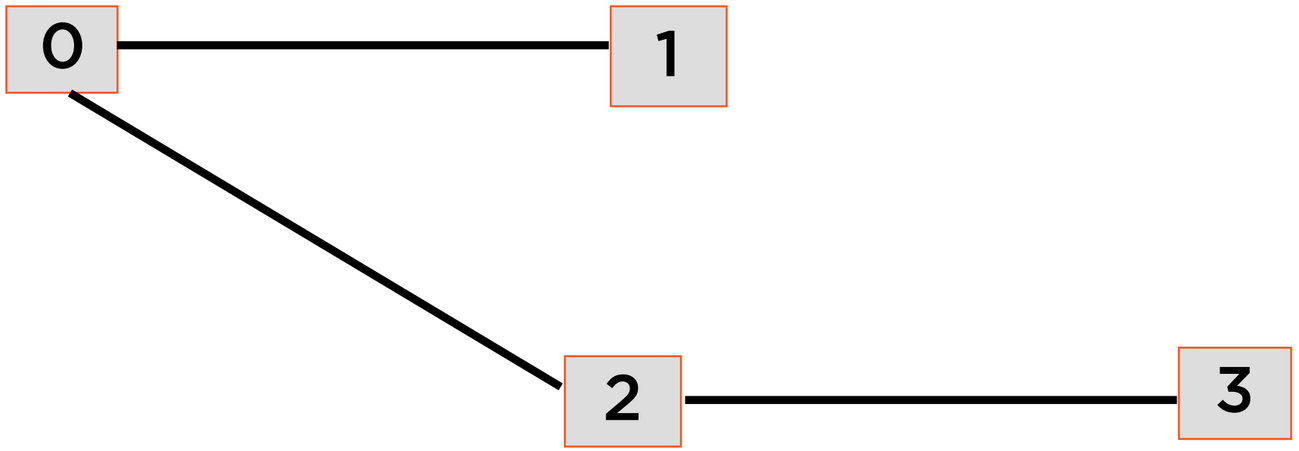


Each node maintains a set of adjacent nodes

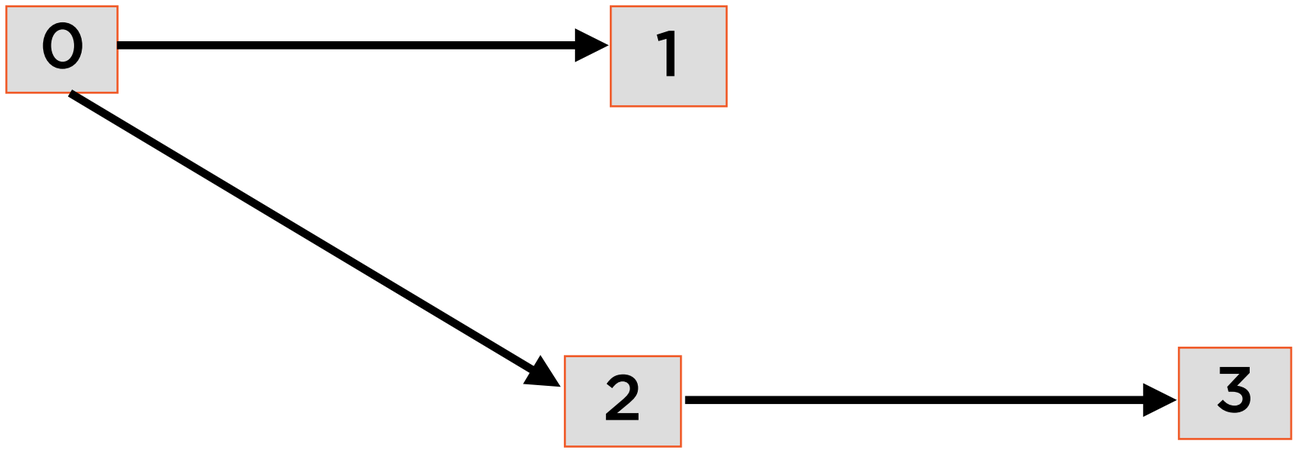
Demo

Represent a Graph as an adjacency set

A Sample Undirected Graph



A Sample Directed Graph

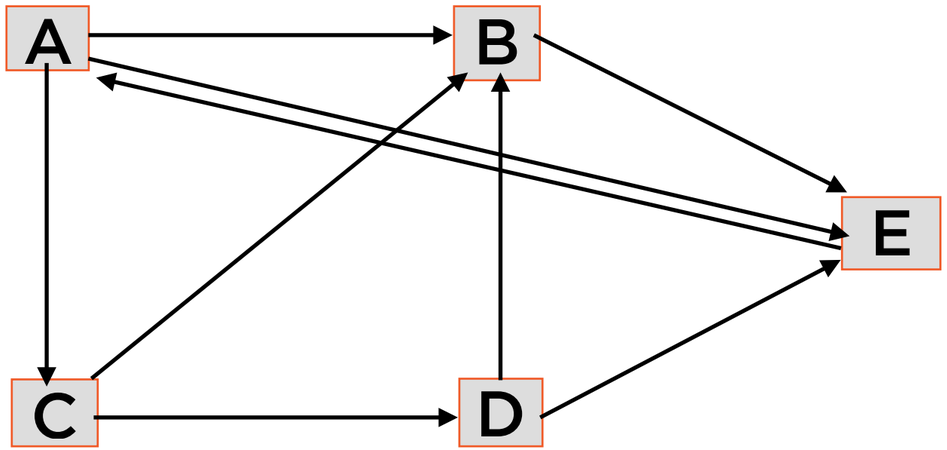


Comparing Graph Representations

Comparing Graph Representations

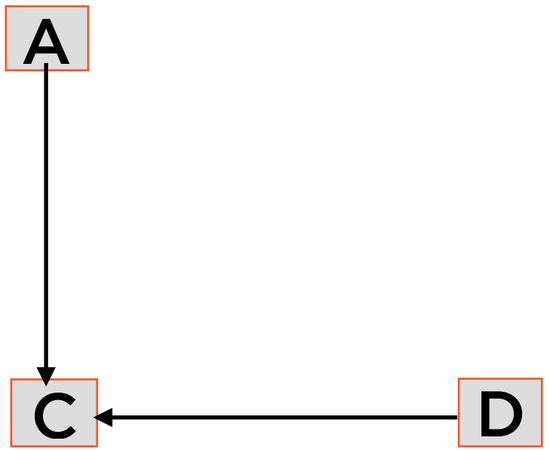
Adjacency Matrix

Makes sense for small, densely connected graphs



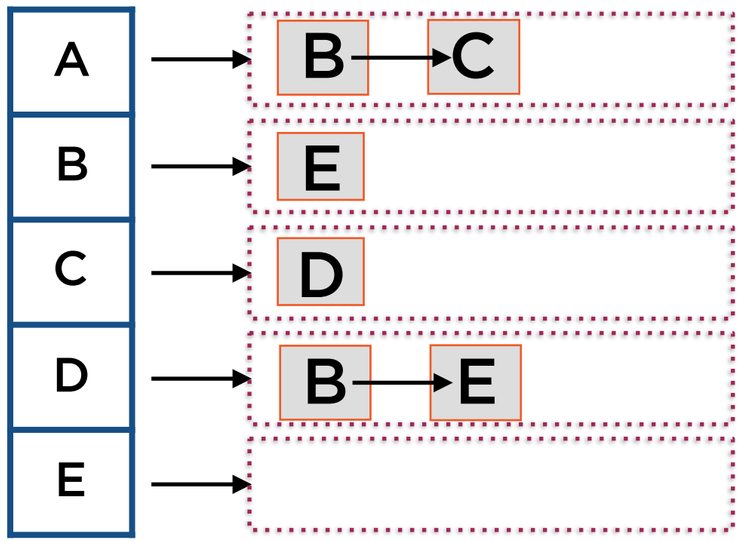
Adjacency List

Useful for large, sparsely connected graphs - saves on storage space



Comparing Graph Representations

	Adjacency Matrix	Adjacency List
Space required	$O(V^2)$	O(E+V)
Checking if edge is present	O(1)	O(degree V)
Iterating over edges	O(V)	O(degree V)



Adjacency lists have some serious flaws

Adjacency sets help address them

Comparing Graph Representations

	Adjacency Matrix	Adjacency Set
Space required	$O(V^2)$	O(E+V)
Checking if edge is present	O(1)	O(ln(degree V))
Iterating over edges	O(V)	O(degree V)

Depth-first and Breadth-first Graph Traversal

Two Ways of Conveying Information

“Answer first”

Headlines in a newspaper

“Drop the mic”

Punchlines in comedy

Two Ways of Traversing Graphs

Breadth-first

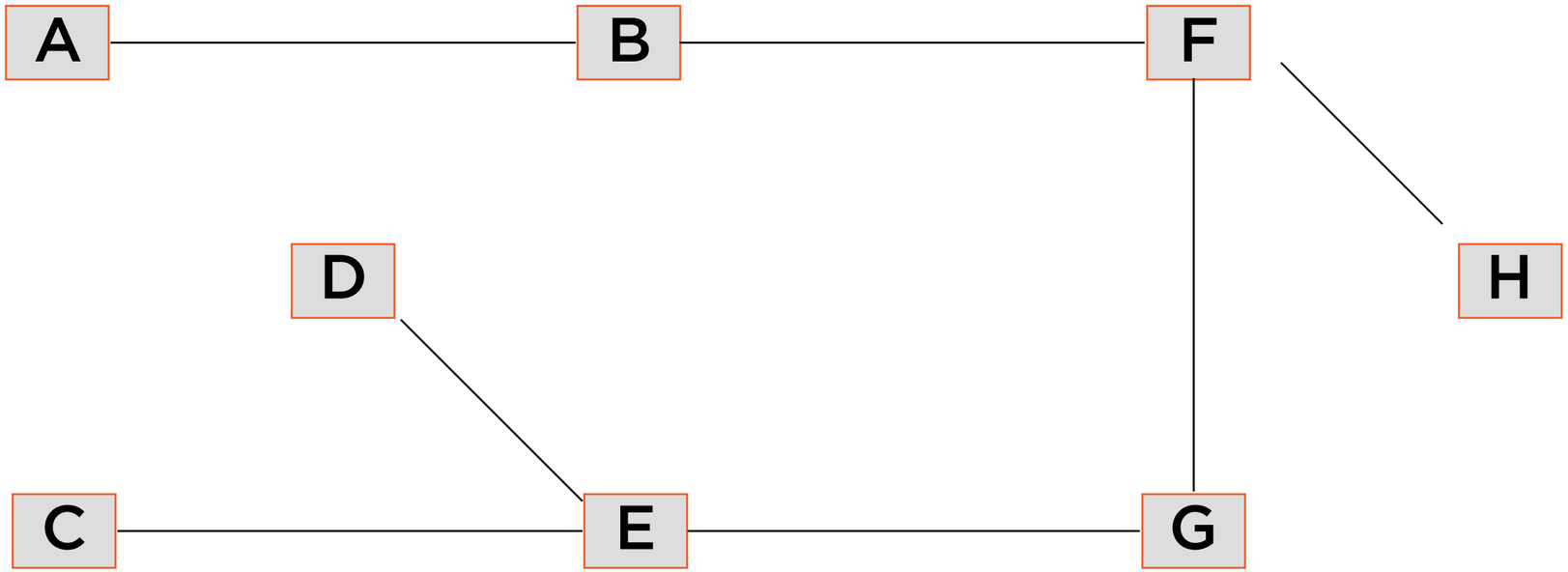
All nodes at same distance from origin visited together

Depth-first

All nodes in certain direction from origin visited together

Tree traversal is easier to understand than graph traversal - start there

Connected Graph with no Cycle



Such a graph is called a tree

Two Ways of Traversing Graphs

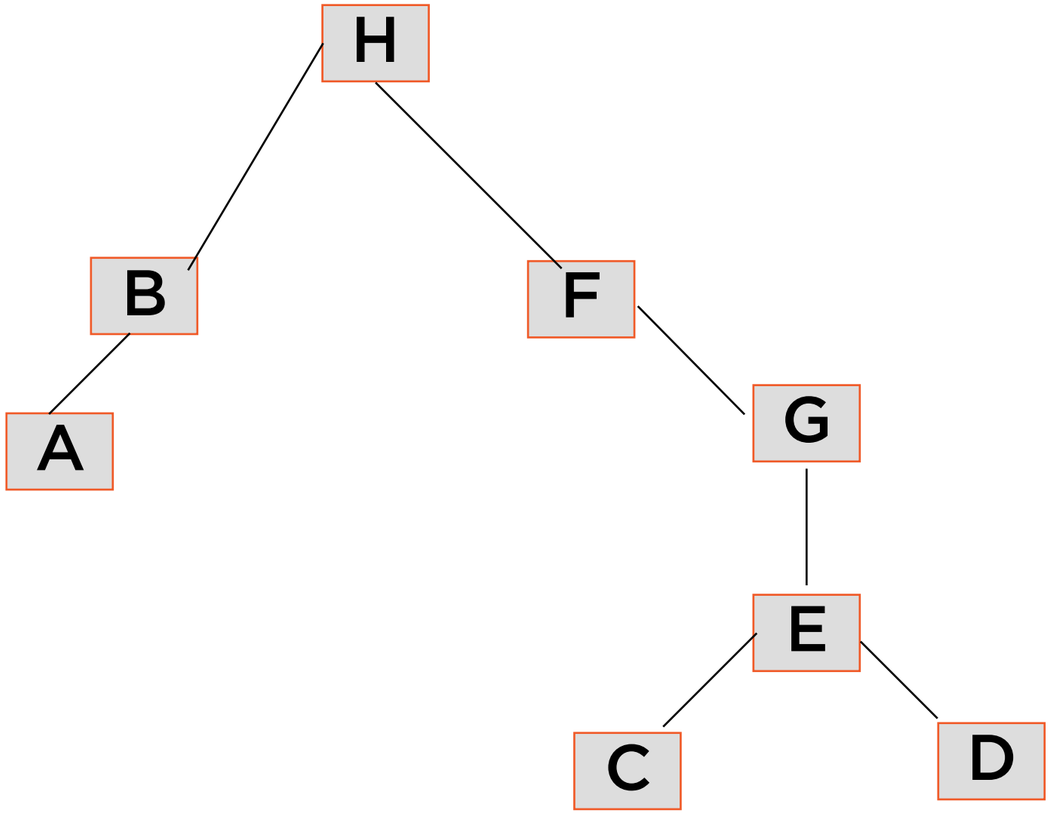
Breadth-first

All nodes at same distance from origin visited together

Depth-first

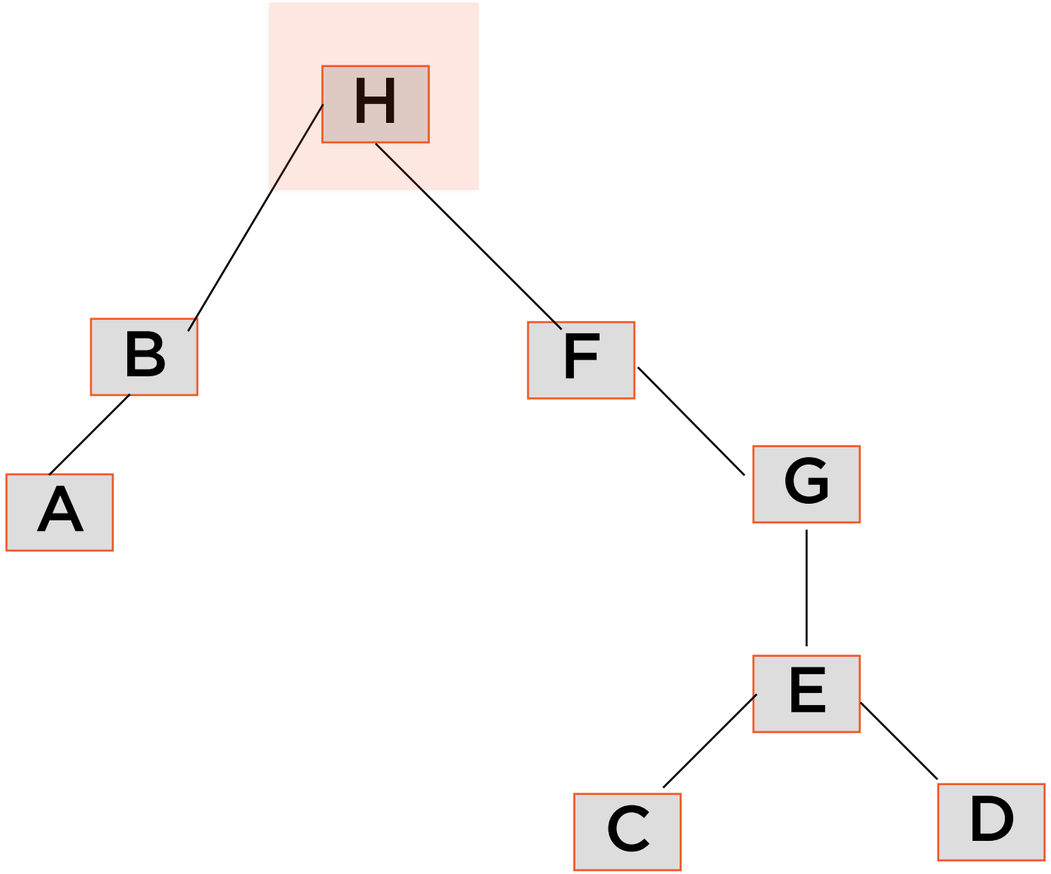
All nodes in certain direction from origin visited together

“Breadth-first” Tree Traversal



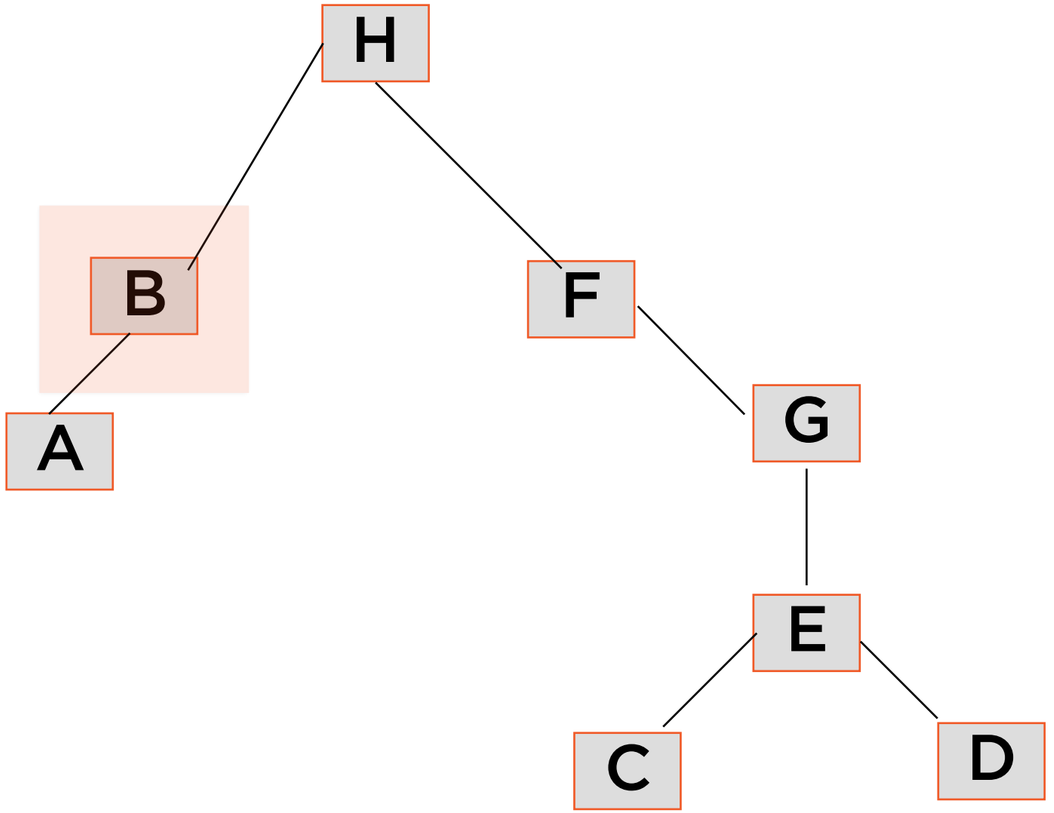
Nodes are visited level-by-level

“Breadth-first” Tree Traversal



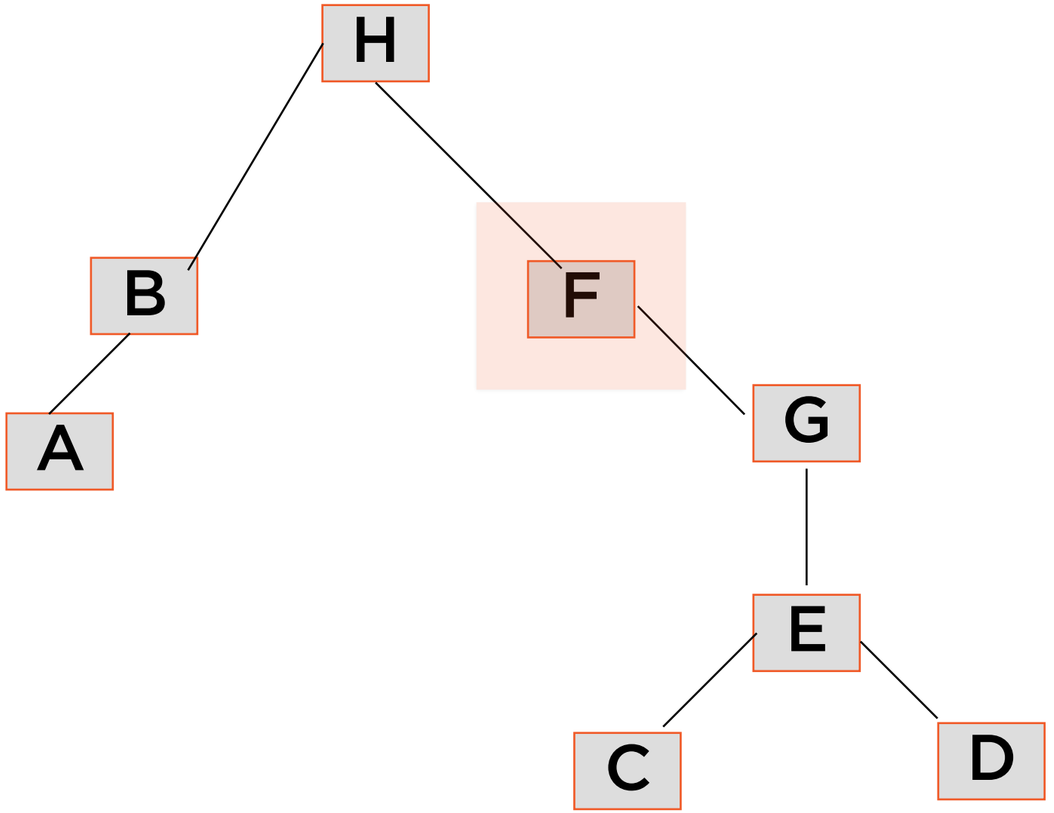
Visited H

“Breadth-first” Tree Traversal



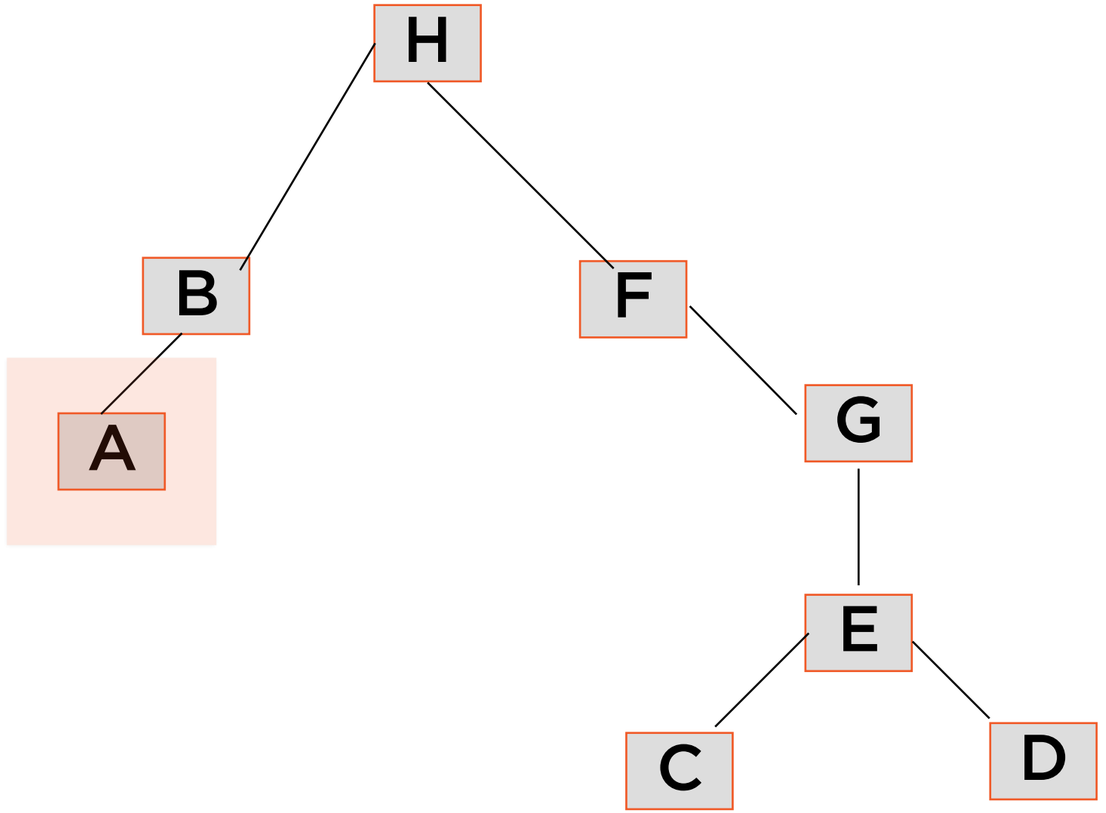
Visited H - B

“Breadth-first” Tree Traversal



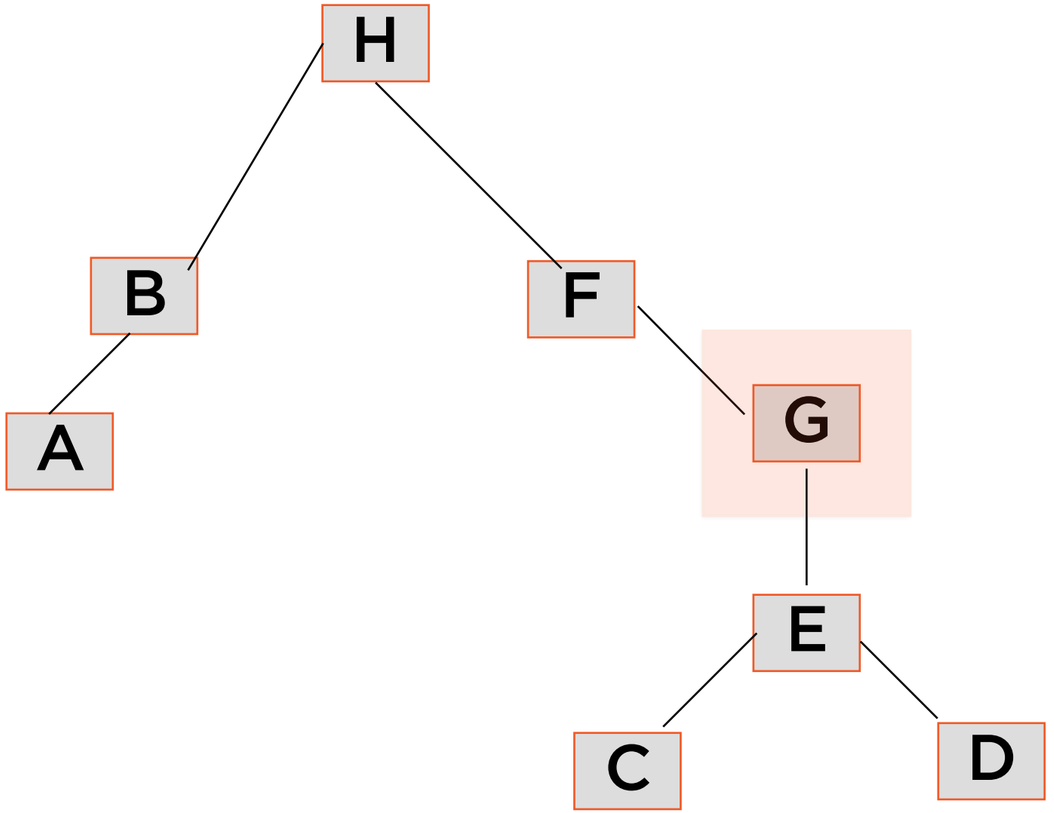
Visited H - B - F

“Breadth-first” Tree Traversal



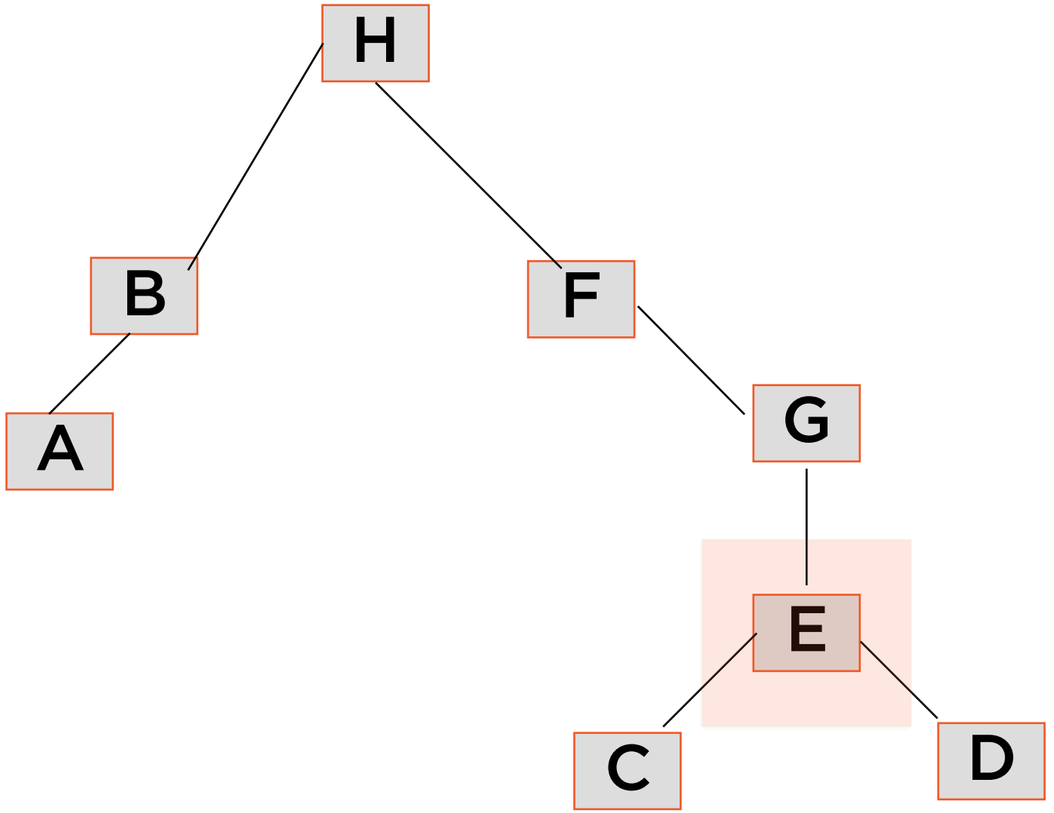
Visited H - B - F - A

“Breadth-first” Tree Traversal



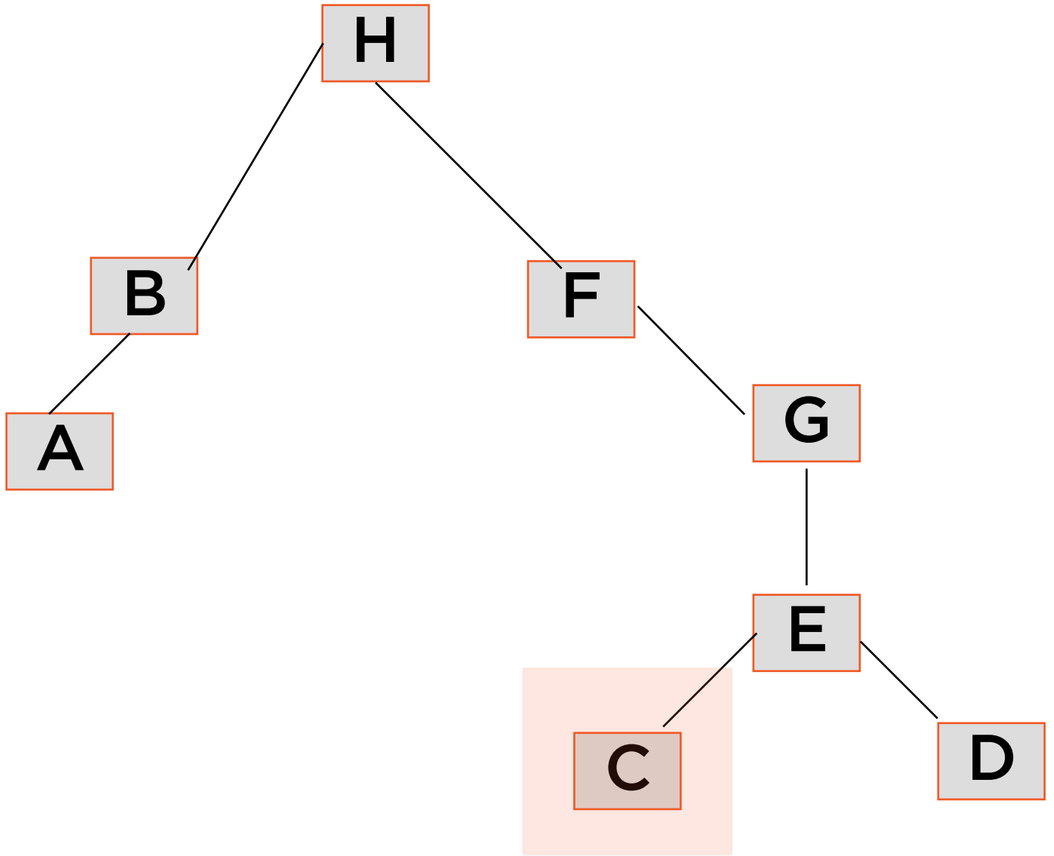
Visited H - B - F - A - G

“Breadth-first” Tree Traversal



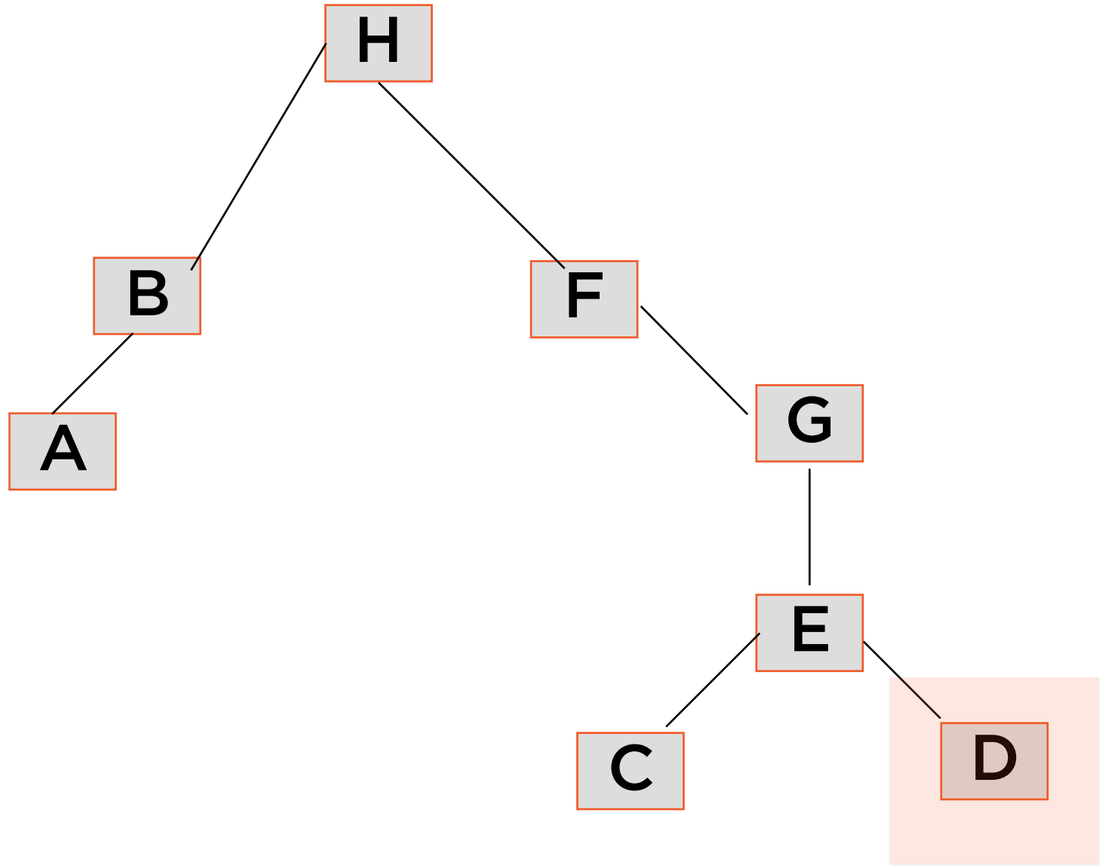
Visited H - B - F - A - G - E

“Breadth-first” Tree Traversal



Visited H - B - F - A - G - E - C

“Breadth-first” Tree Traversal



Visited H - B - F - A - G - E - C - D

Two Ways of Traversing Graphs

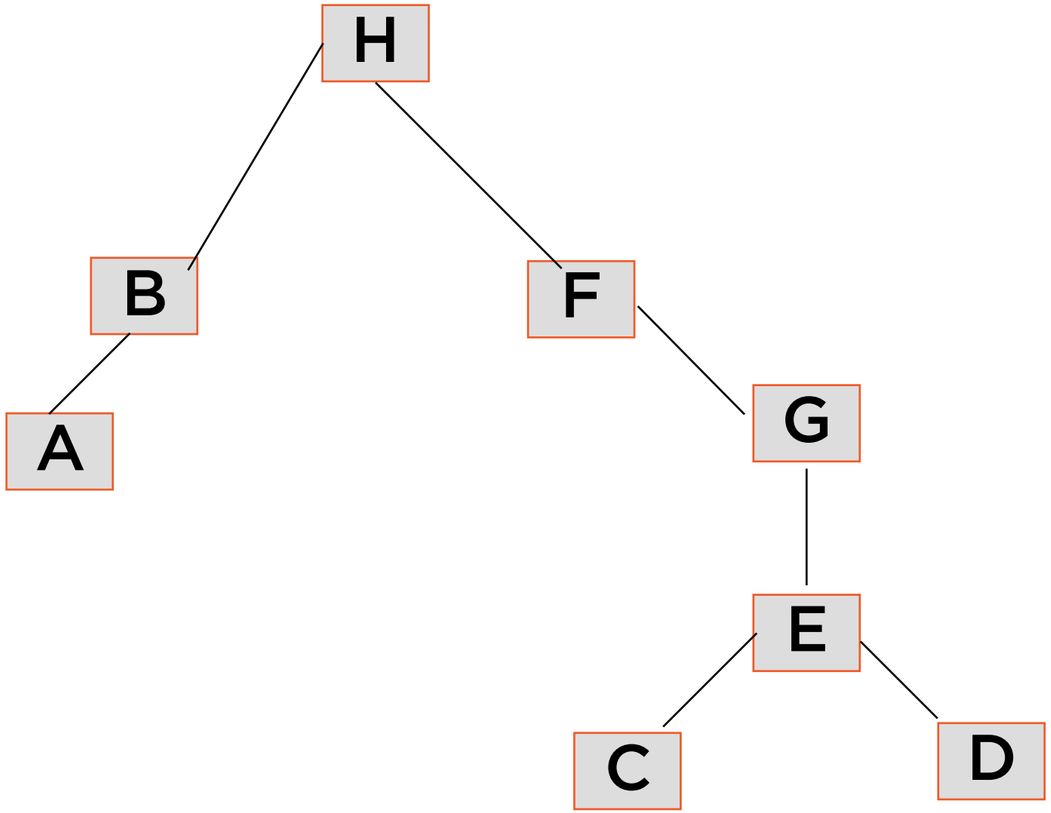
Breadth-first

All nodes at same distance from origin visited together

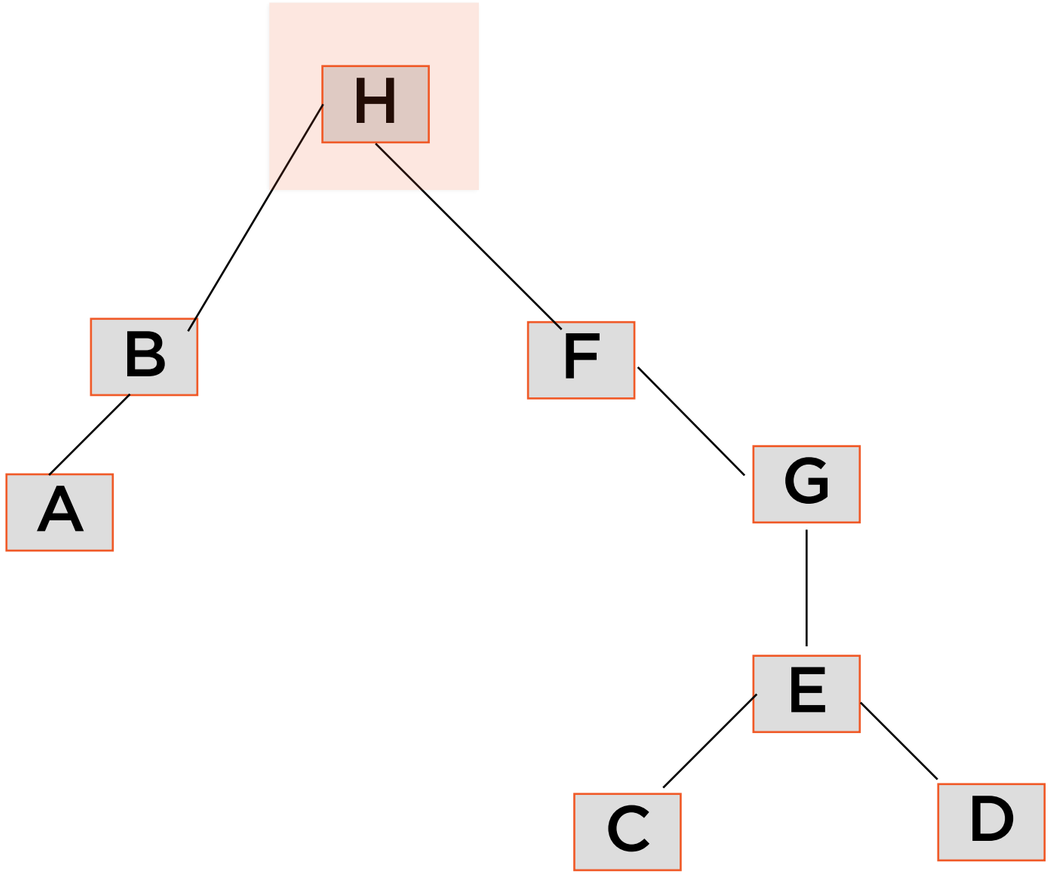
Depth-first

All nodes in certain direction from origin visited together

“Depth-first” Tree Traversal

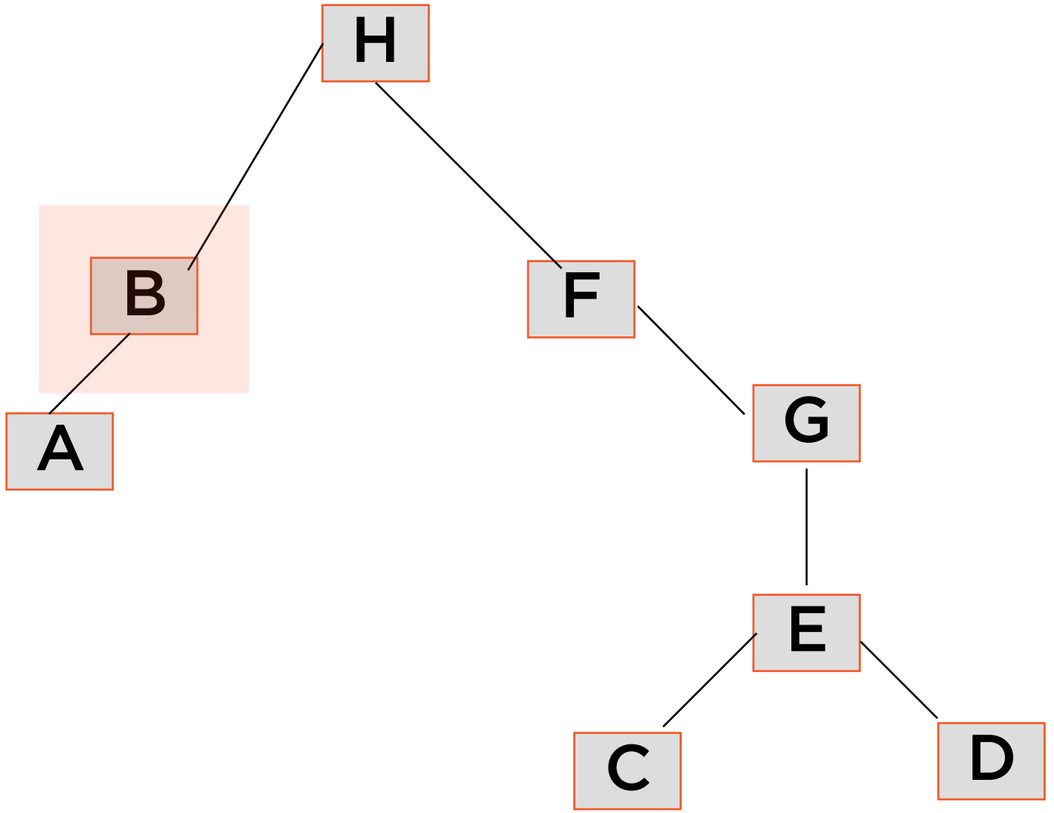


“Depth-first” Tree Traversal



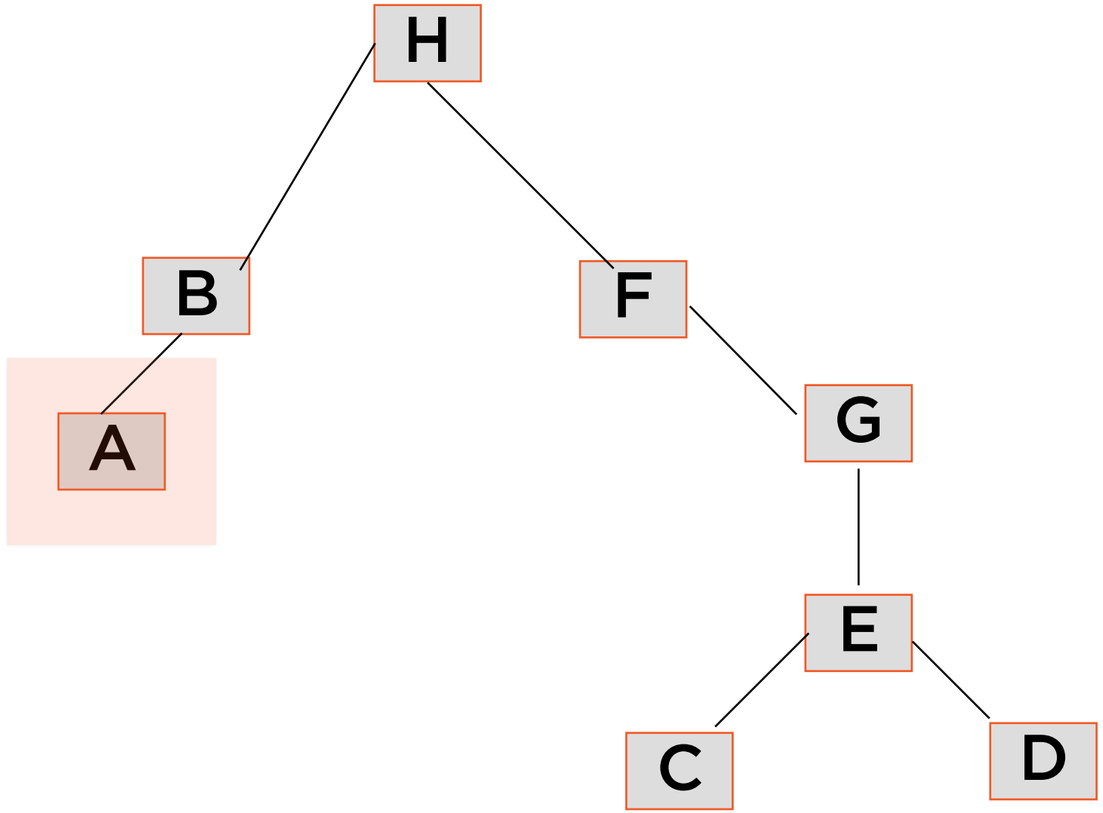
Visited H

“Depth-first” Tree Traversal



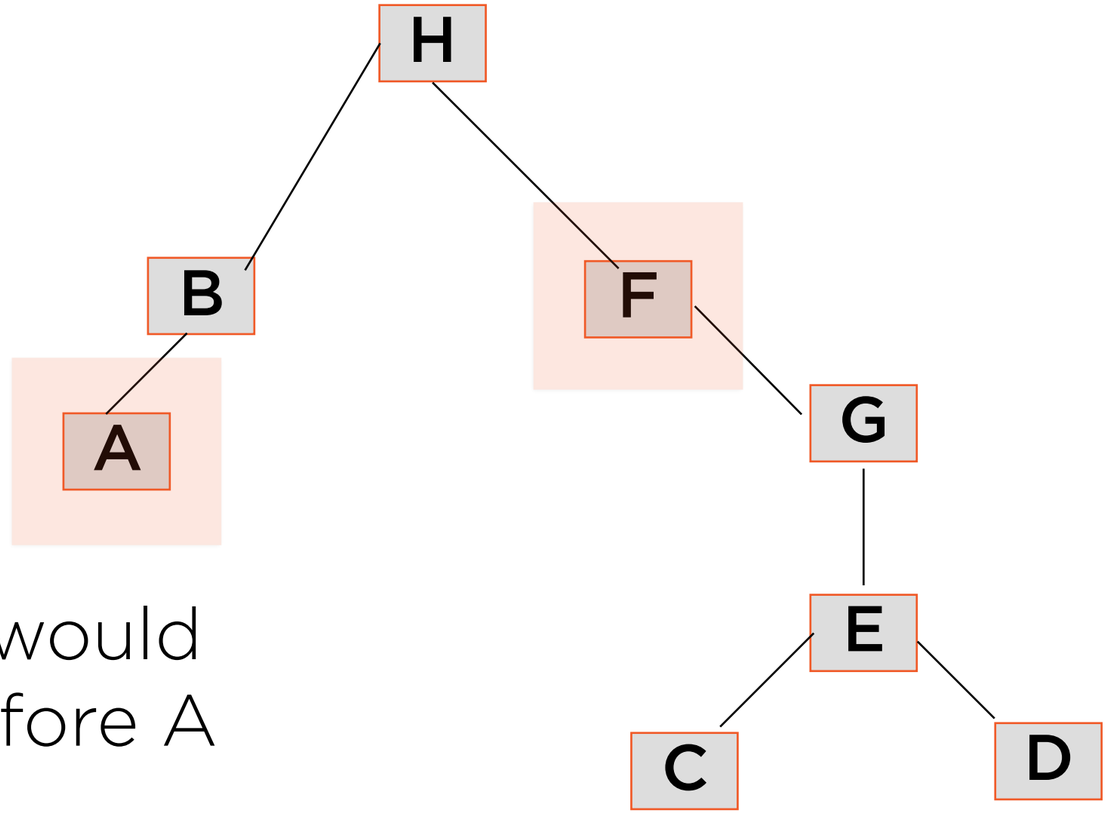
Visited H - B

“Depth-first” Tree Traversal



Visited H - B - A

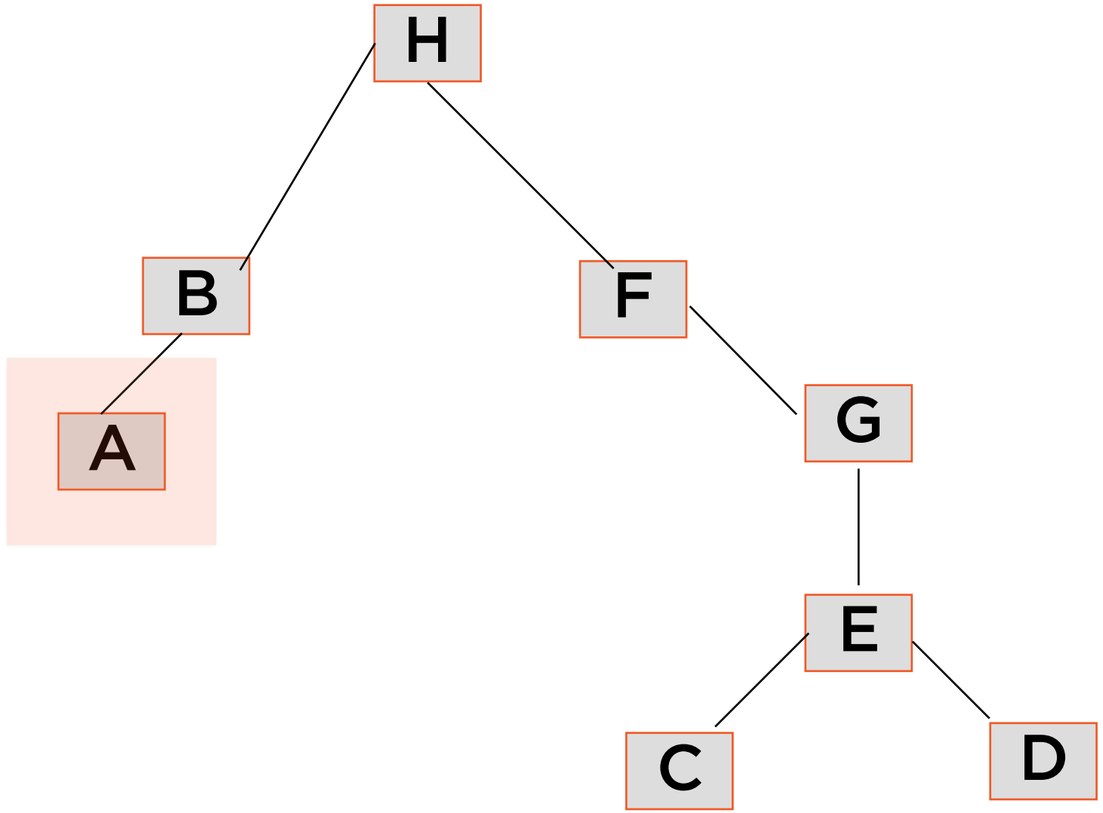
“Depth-first” Tree Traversal



In breadth-first, would have visited F before A

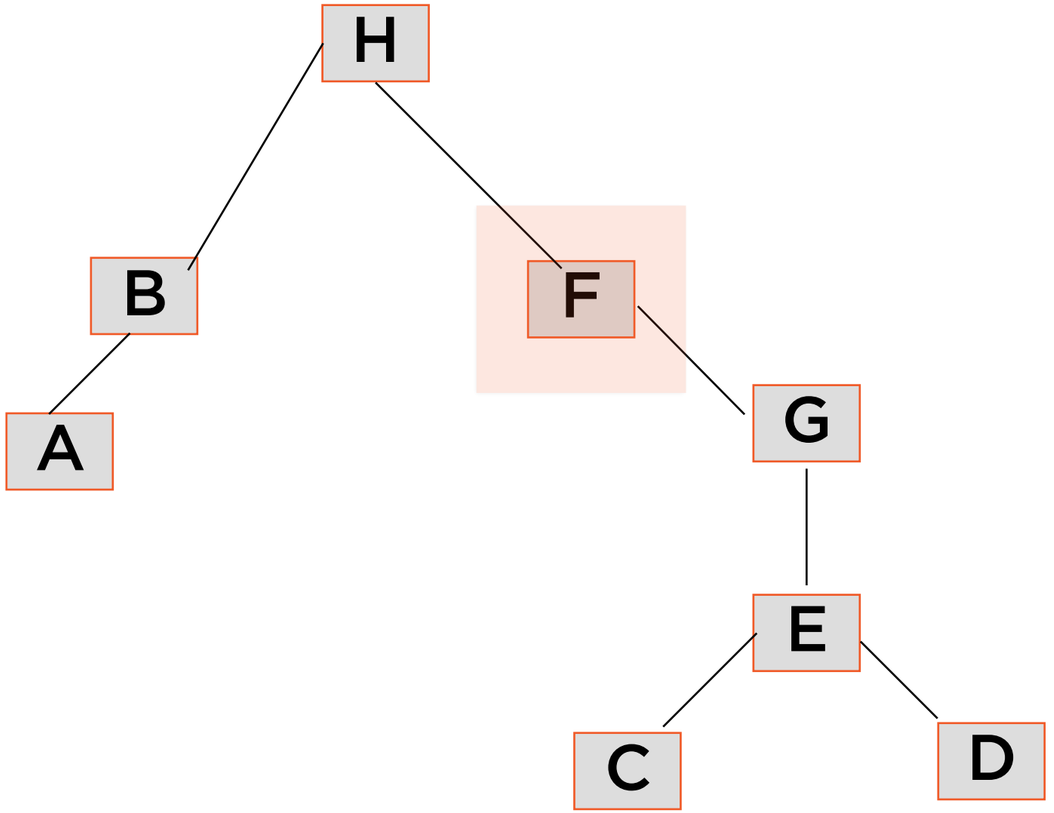
Visited H - B - A

“Depth-first” Tree Traversal



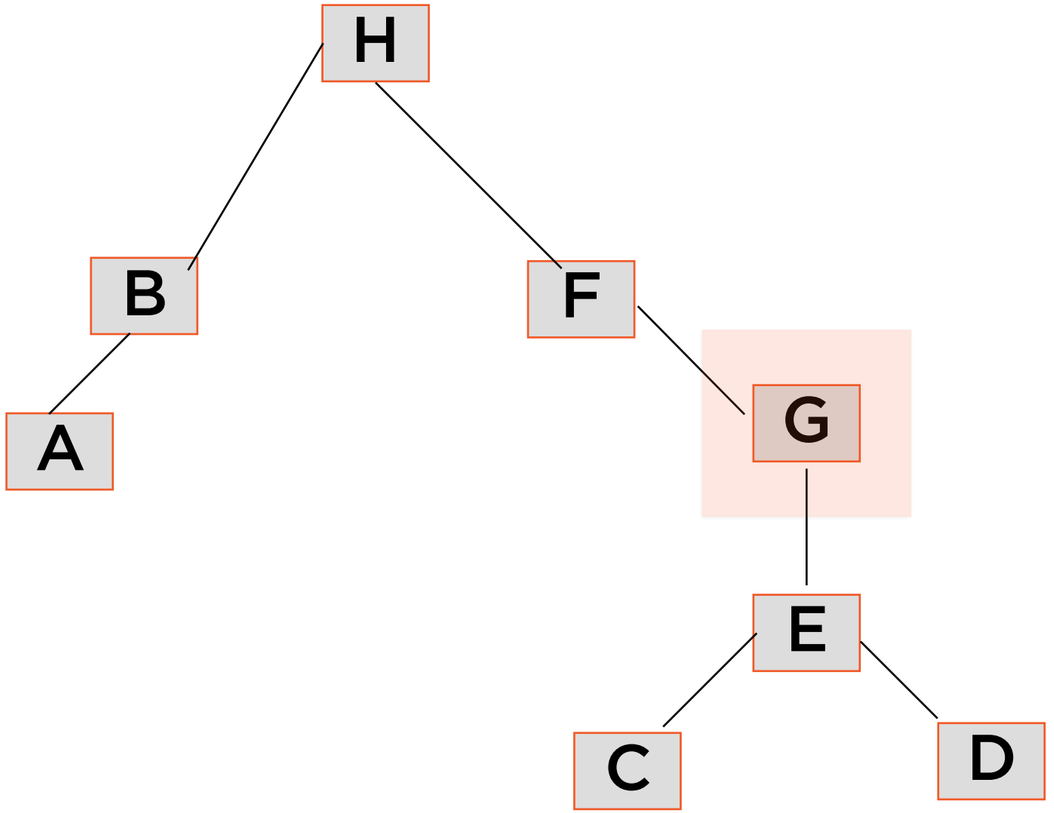
Visited H - B - A

“Depth-first” Tree Traversal



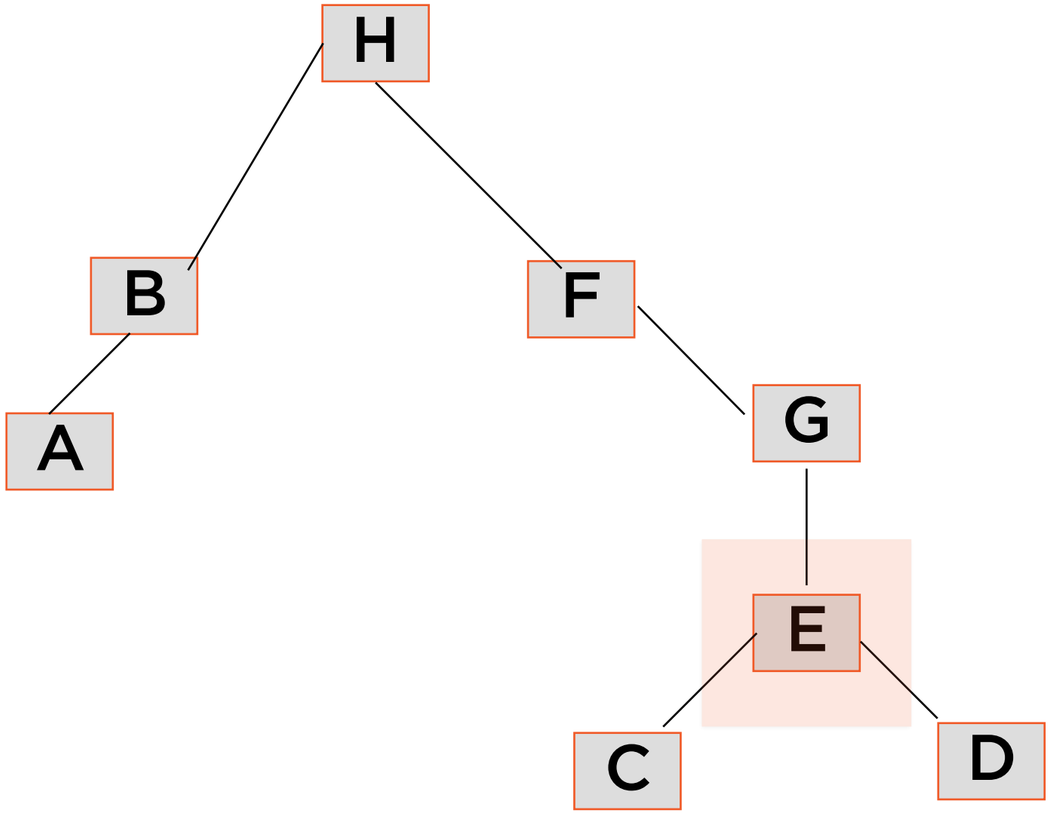
Visited H - B - A - F

“Depth-first” Tree Traversal



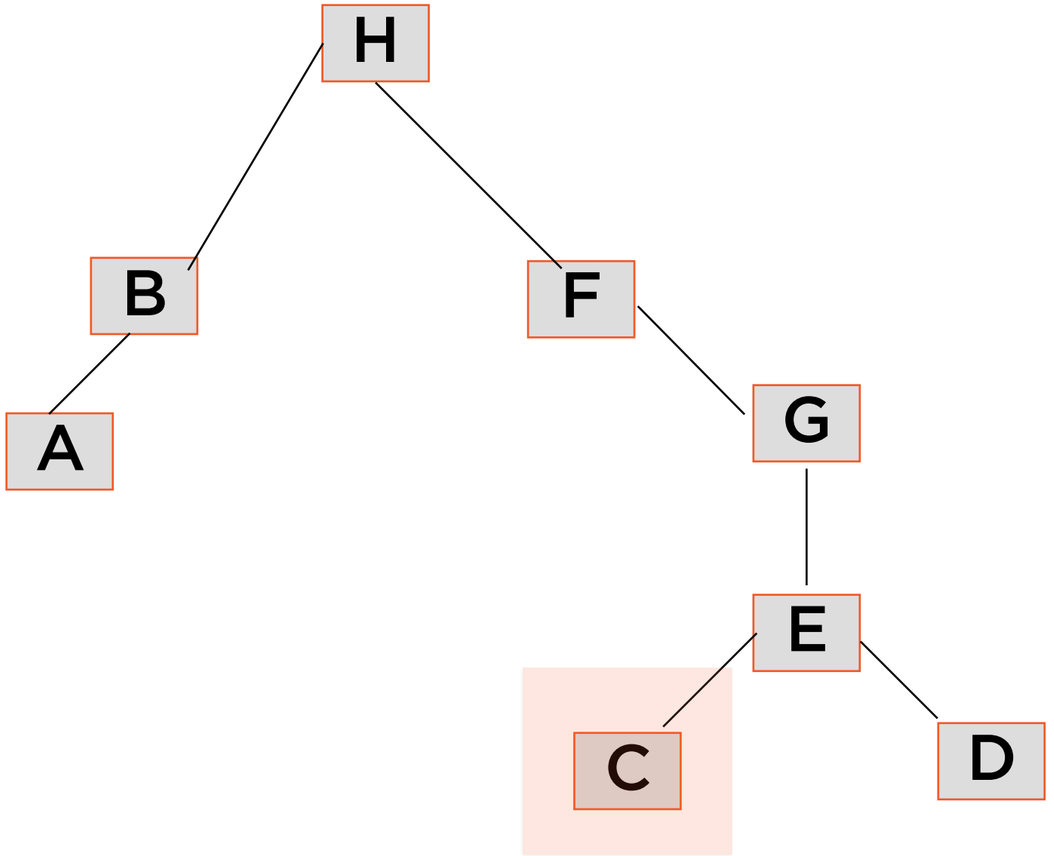
Visited H - B - A - F - G

“Depth-first” Tree Traversal



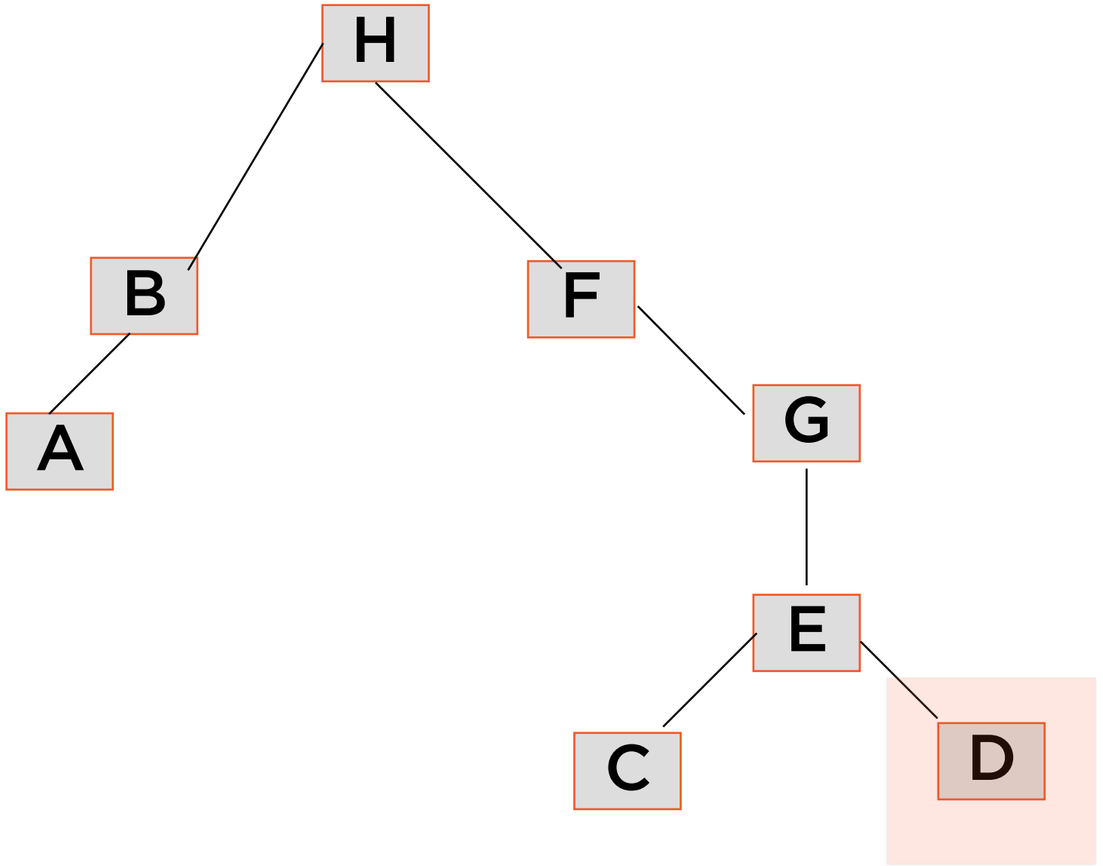
Visited H - B - A - F - G - E

“Depth-first” Tree Traversal



Visited H - B - A - F - G - E - C

“Depth-first” Tree Traversal



Visited H - B - A - F - G - E - C

Two Ways of Traversing Graphs

Breadth-first

All nodes at same distance from origin visited together

Depth-first

All nodes in certain direction from origin visited together

Tree traversal is easier to understand than graph traversal - start there

Traversal Algorithms

Traversing a Tree

One node is designated root

Only one specific path from root to any node

Traversing a Graph

No designated root

Multiple paths possible between any pair of nodes

Traversal Algorithms

Traversing a Tree

No cycles

Any node will be visited exactly once

No need to track which nodes already visited

Traversing a Graph

Cycles possible

Nodes could be visited multiple times (could lead to infinite loop)

Essential to track which nodes already visited

Traversal Algorithms

Traversing a Tree

No unconnected nodes possible

Traversing a Graph

Unconnected nodes possible

Traversal Algorithms

Traversing a Tree

No unconnected nodes possible

No need to track which nodes already visited

Traversing a Graph

Unconnected nodes possible

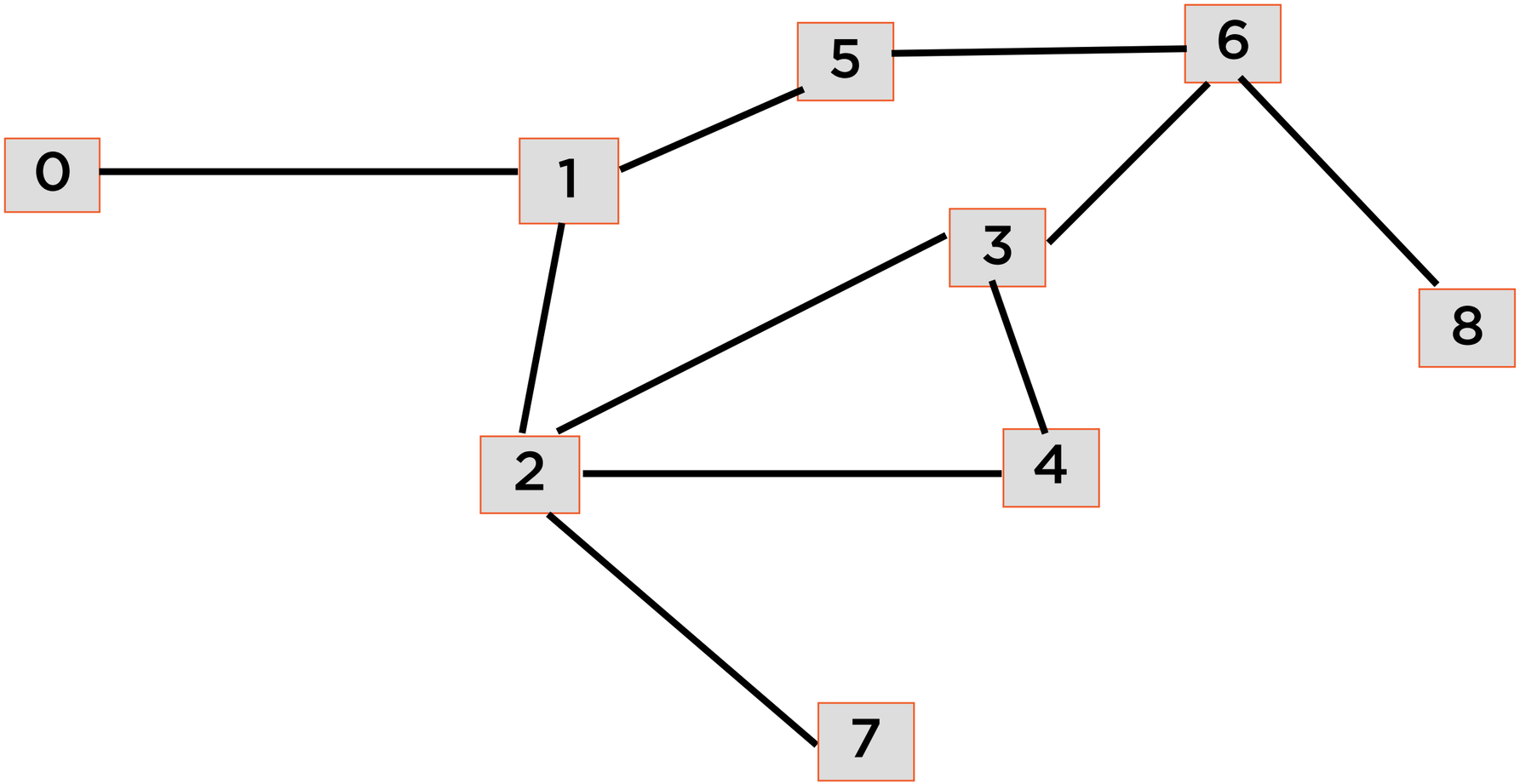
Algorithm can not terminate until all nodes have been visited

Graph traversal, unlike tree traversal, explicitly needs to ensure that each node is visited exactly once

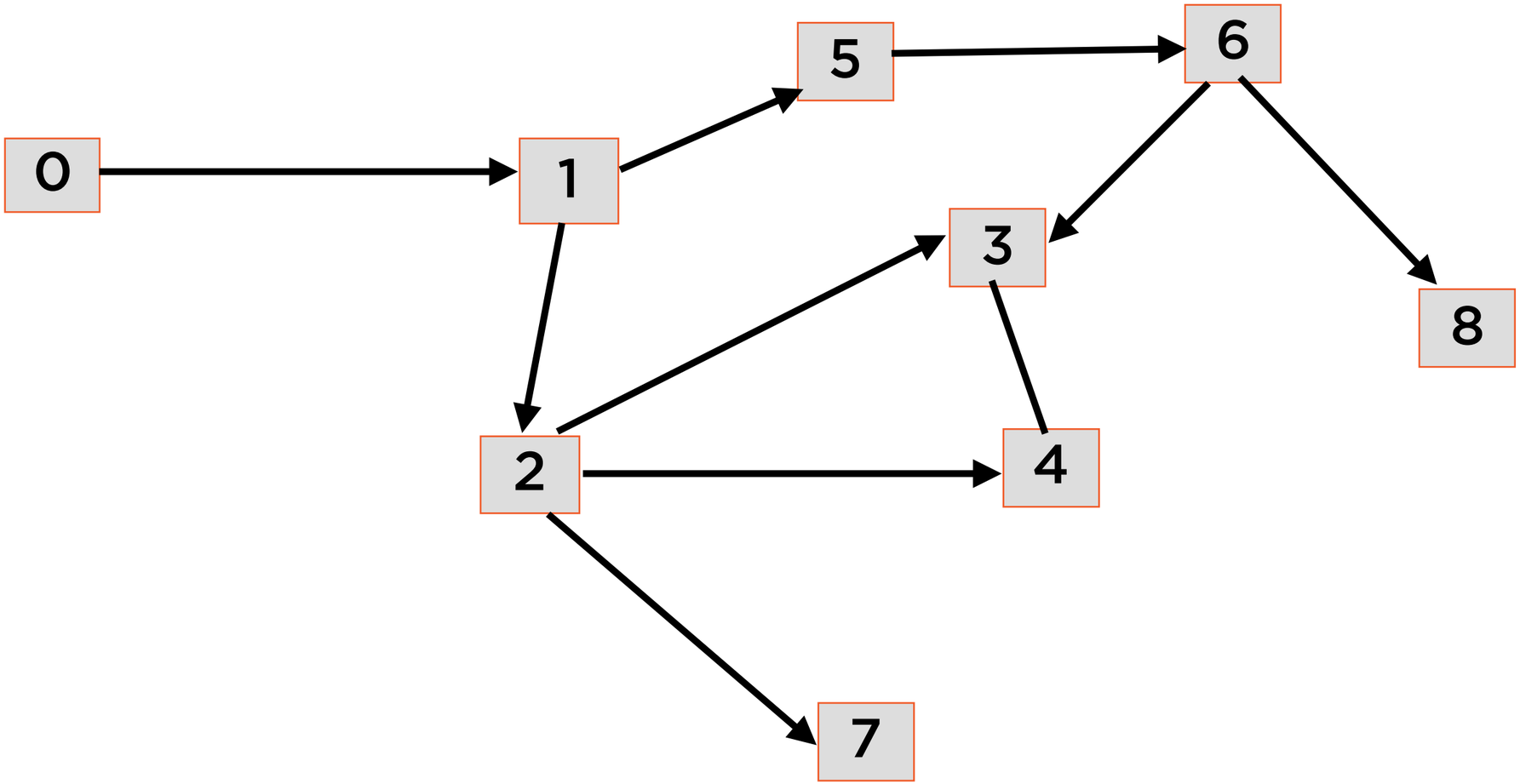
Demo

Breadth-first traversal of a graph

A Sample Undirected Graph



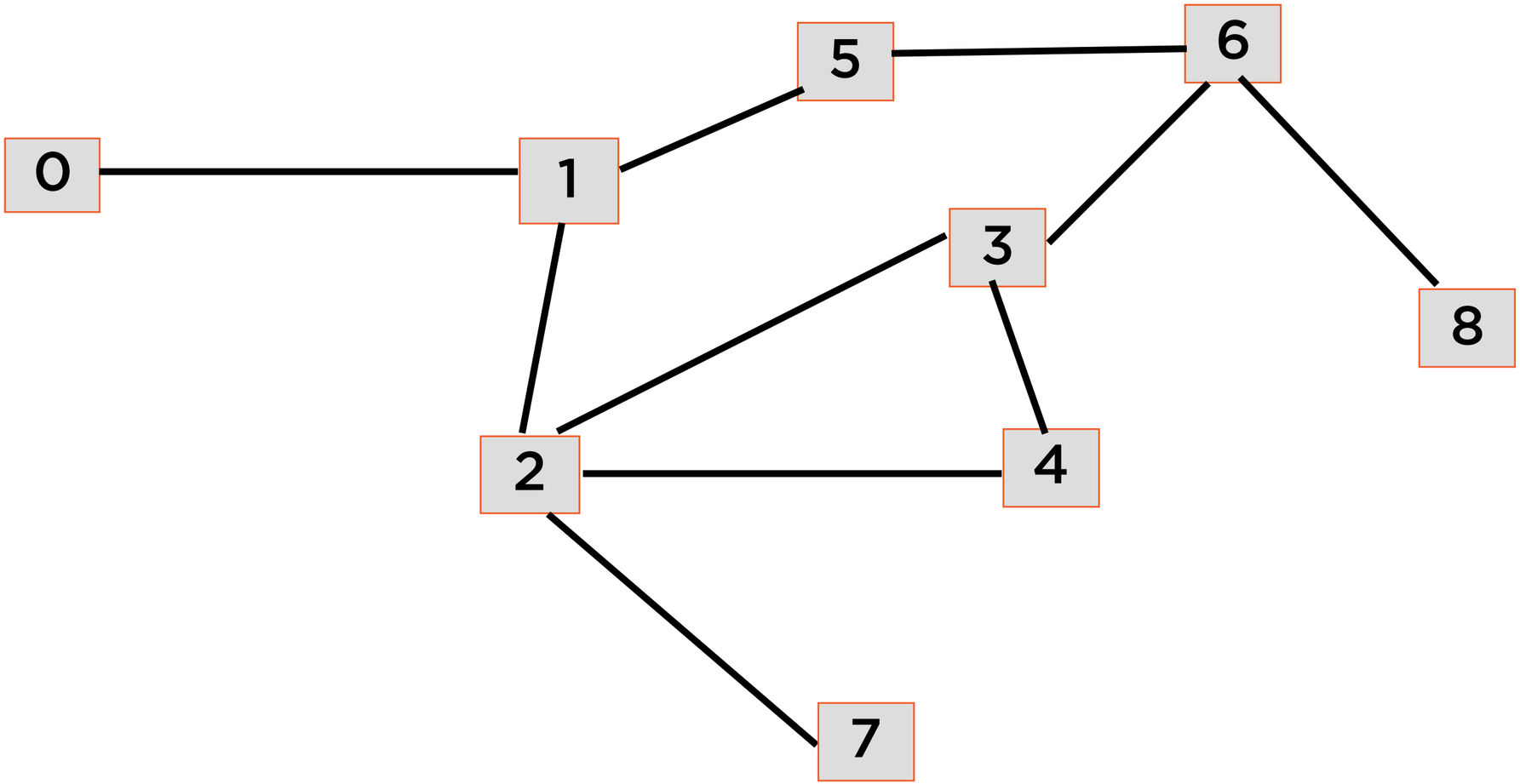
A Sample Directed Graph



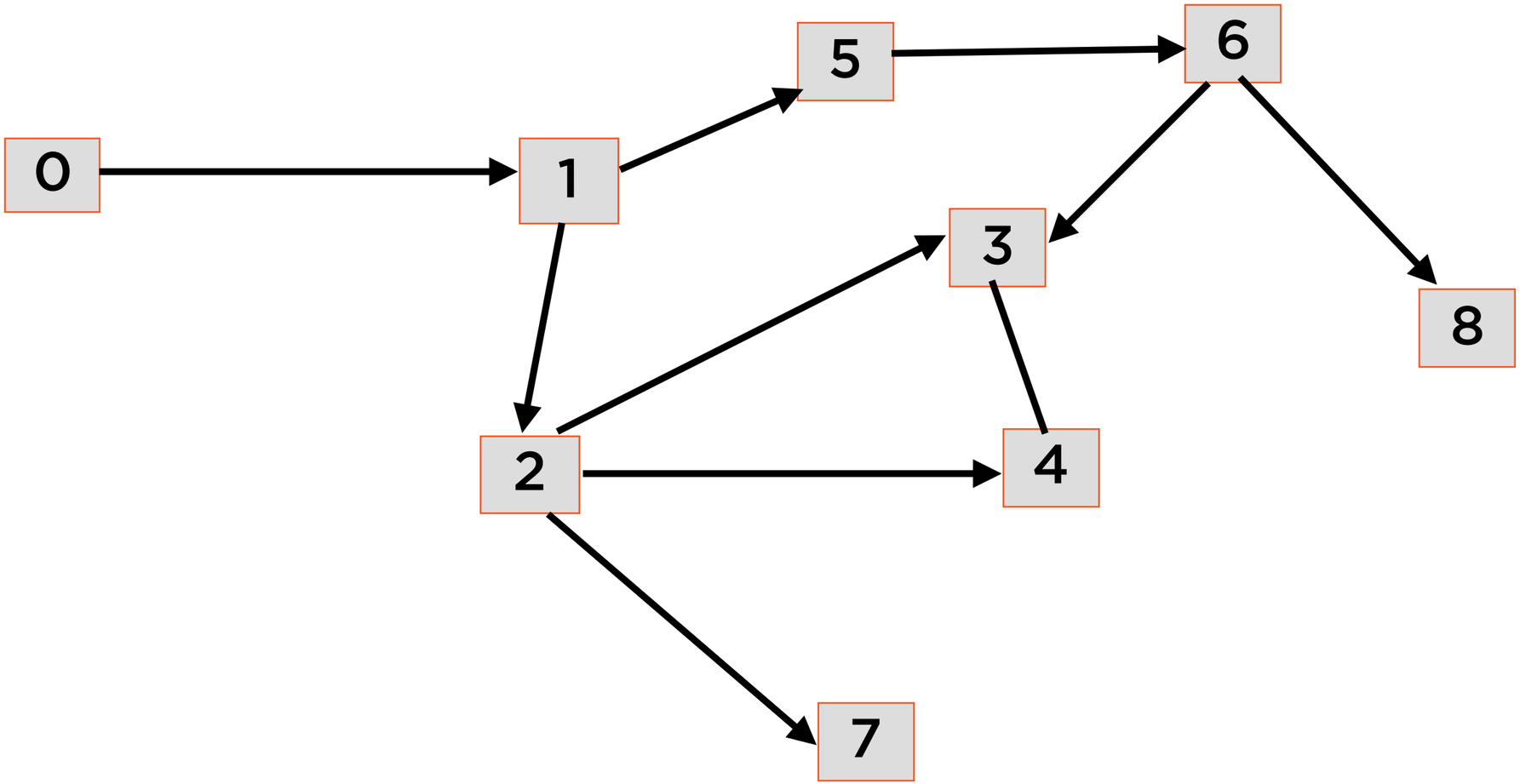
Demo

Depth-first traversal of a graph

A Sample Undirected Graph



A Sample Directed Graph



Summary

Graphs are excellent tools for modeling complex relationships

An adjacency matrix is the most common way of representing a graph

Adjacency lists and adjacency sets are alternative data representations

The two fundamental ways of traversing a graph are

- Depth-first
- Breadth-first