

# Intro to Graphs

CSE 332 Spring 2020

**Instructor:** Hannah C. Tang

**Teaching Assistants:**

Annie Mao

Howard Xiao

Lucy Jiang

Chris Choi

Jade Watkins

Nathan Lipiarski

Ethan Knutson

Khushi Chaudhari

Richard Jiang

# Warm-up

- ❖ Write your answers on a piece of scratch paper. For each of the following, which would use *directed edges*? Which might have *0-degree nodes*?
  - Web pages with links
  - Facebook friends
  - Methods in a program that call each other
  - Road maps (e.g., Google maps)
  - Airline routes
  - Family trees
  - Course pre-requisites

# Announcements

- ❖ Quiz 4 released, due Friday
- ❖ Exercises *actually* due in 1 week

# Learning Objectives

- ❖ Memorize a bunch of definitions 
- ❖ Learn to choose a graph representation based on requirements

# Lecture Outline

- ❖ **Graph Definitions: Part 2**
- ❖ Graph Representations
  - Adjacency Matrix
  - Adjacency List

*Lecture questions: [pollev.com/cse332](https://pollev.com/cse332)*

# Graphs

- ❖ A *graph* represents relationships among items
  - Very general definition because it's a very general concept

❖ A *graph* is a pair:  $G = (V, E)$

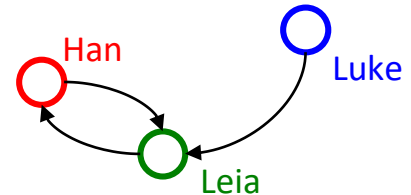
- A set of *vertices*, also known as *nodes*

$$V = \{v_1, v_2, \dots, v_n\}$$

- A set of *edges*, possibly *directed*

$$E = \{e_1, e_2, \dots, e_m\}$$

- Each edge  $e_i$  is a pair of vertices  $(v_j, v_k)$
- An edge “connects” the vertices



$$V = \{\text{Han}, \text{Leia}, \text{Luke}\}$$

$$E = \{(\text{Luke}, \text{Leia}), (\text{Han}, \text{Leia}), (\text{Leia}, \text{Han})\}$$

# Is a Graph an ADT?

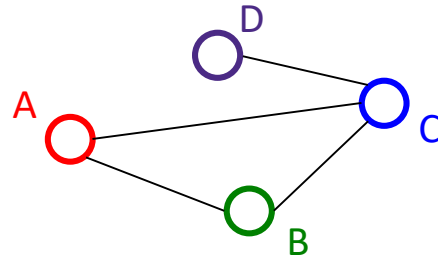
- ❖ tl;dr: maybe
  - They have operations like `hasEdge ( (vj, vk) )`
  - But it is unclear what the “standard operations” are
  
- ❖ Instead:
  - We tend to develop algorithms over graphs and then use whatever data structure is efficient for that algorithm
  
- ❖ Many important problems can be solved by:
  1. Formulating them in terms of graphs
  2. Applying a standard graph algorithm

# Some Graph Examples

- ❖ For each of the following, what are the vertices and the edges?
  - Web pages with links
  - Facebook friends
  - Methods in a program that call each other
  - Road maps (e.g., Google maps)
  - Airline routes
  - Family trees
  - Course pre-requisites
  
- ❖ **Wow!** Using the same algorithms for problems across so many domains sounds like “core computer science and engineering”

# Undirected Graphs

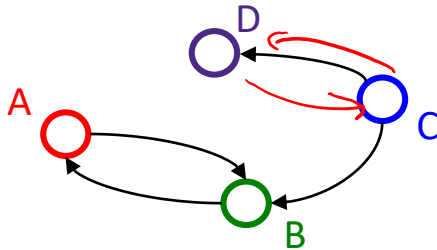
- ❖ In *undirected graphs*, edges have no specific direction
  - Edges are always “two-way”



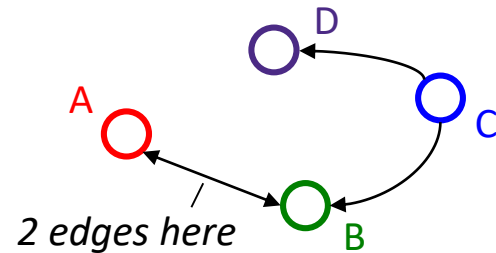
- ❖ Thus,  $(u, v) \in E$  implies  $(v, u) \in E$ 
  - Only one of these edges needs to be in the set; the other is implicit
- ❖ *Degree* of a vertex: number of edges containing that vertex
  - i.e.: the number of adjacent vertices

# Directed Graphs

- ❖ In *directed graphs* (aka *digraphs*), edges have a *direction*



or



- ❖ Thus,  $(u, v) \in E$  does not imply  $(v, u) \in E$ 
  - $(u, v) \in E$  means  $u \rightarrow v$ ;  $u$  is the *source* and  $v$  the *destination*
- ❖ *In-Degree* of a vertex: number of in-bound edges
  - i.e.: edges where the vertex is the destination
- ❖ *Out-Degree* of a vertex: number of out-bound edges
  - i.e.: edges where the vertex is the source

# Self-edges

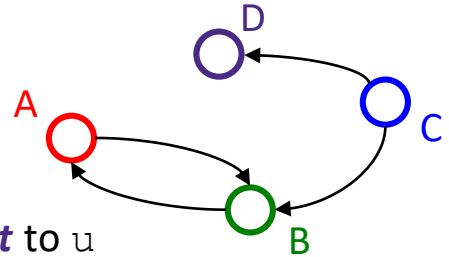
“Selfedge” ☺



- ❖ A **self-edge** (aka a **loop**) is an edge of the form  $(u, u)$
- ❖ Depending on the use/algorithm, a graph may have:
  - No self edges
  - Some self edges
  - All self edges (therefore often implicit, but we will be explicit)
- ❖ A node can have a degree / in-degree / out-degree of zero

# Adjacency (1 of 2)

- ❖ If  $(u, v) \in E$ 
  - Then  $v$  is a *neighbor* of  $u$ , i.e.,  $v$  is *adjacent* to  $u$
  - For directed edges, order matters
    - $u$  is not adjacent to  $v$  unless  $(v, u) \in E$



$$V = \{A, B, C, D\}$$

$$E = \{(C, B), (A, B), (B, A), (C, D)\}$$

# Adjacency (2 of 2)

❖ For a graph  $G = (V, E)$ :

- $|V|$  is the number of vertices

- $|E|$  is the number of edges

- Minimum size?

- 0

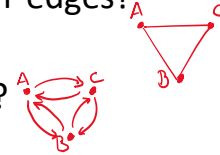
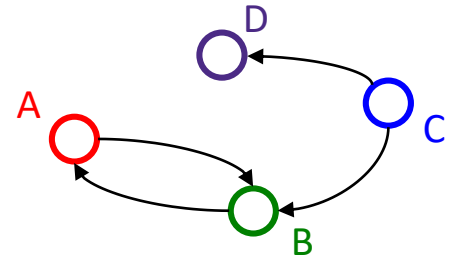
- Maximum size for an undirected graph with no self-edges?

- $|V||V-1|/2 \in O(|V|^2)$

- Maximum for a directed graph with no self-edges?

- $|V||V-1| \in O(|V|^2)$

- If self-edges are allowed, add  $|V|$  to the answers above (applies to both undirected and directed graphs)

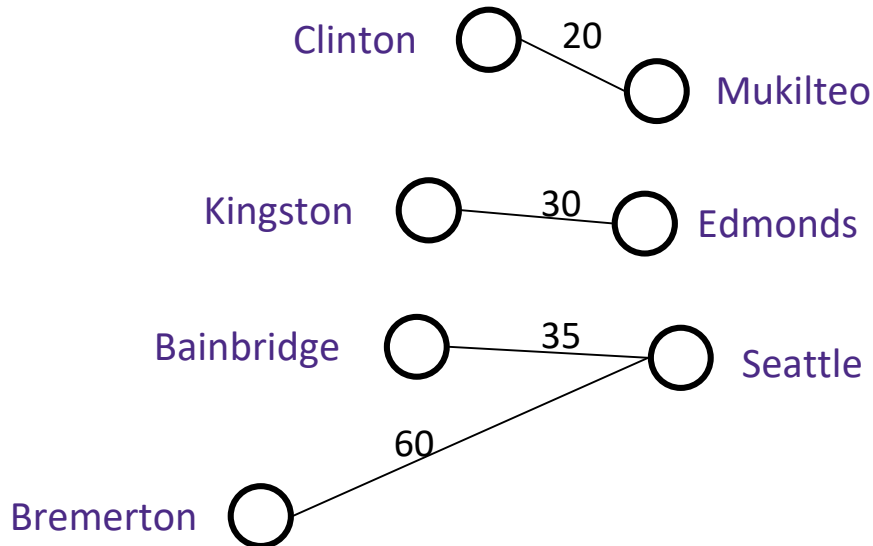


# Graph Examples, Again

- ❖ For each of the following, which would use *directed edges*? Which would have *self-edges*? Which might have *0-degree nodes*?
  - Web pages with links
  - Facebook friends
  - Methods in a program that call each other
  - Road maps (e.g., Google maps)
  - Airline routes
  - Family trees
  - Course pre-requisites

# Weighted Graphs

- ❖ In a weighed graph, each edge has a ***weight*** a.k.a. ***cost***
  - Typically numeric (most examples will use ints)
  - Some graphs allow *negative weights*; many don't



# Graph Examples, Once More Unto the Breach

❖ Do *weights* make sense for each of the following graphs? What would they represent, and could those weights be *negative*?

- Web pages with links
- Facebook friends
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course pre-requisites

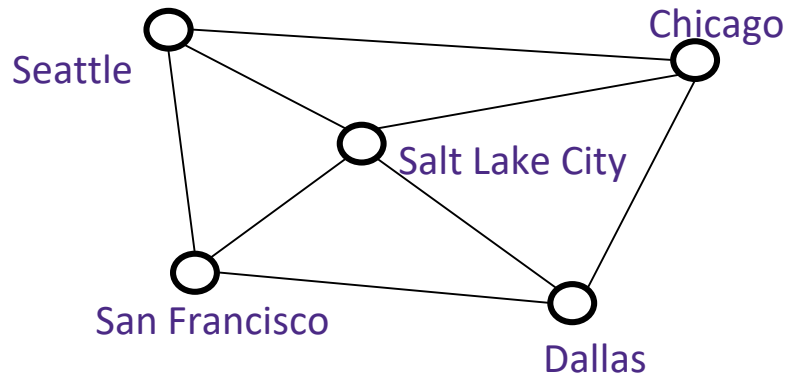
→ Depends on the weight defn!

- Flight duration: always positive
- Arrival time - departure time, using local time: might be negative!

# Paths and Cycles (1 of 2)

- ❖ A **path** is a list of vertices  $[v_0, v_1, \dots, v_n]$  such that  $(v_i, v_{i+1}) \in E$  for all  $0 \leq i < n$ 
  - You'd call it a path from  $v_0$  to  $v_n$

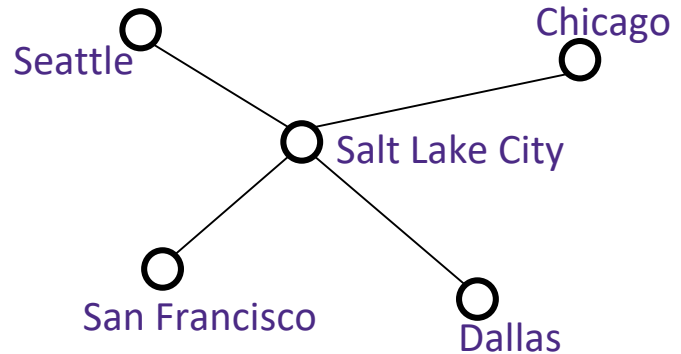
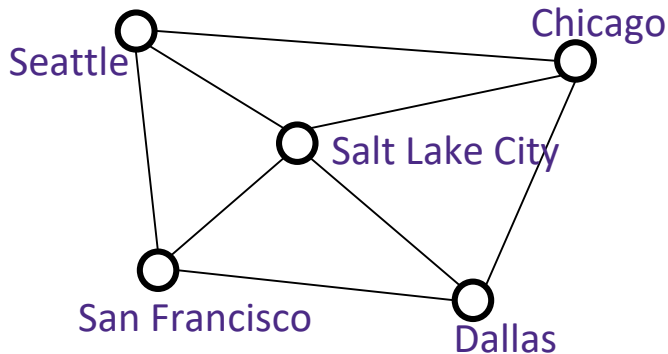
- ❖ A **cycle** is a path that begins and ends at the same node
  - i.e.,  $v_0 == v_n$



- ❖ Example path:
  - [Seattle, SLC, Chicago, Dallas, SF, Seattle]
  - Also happens to be a cycle!

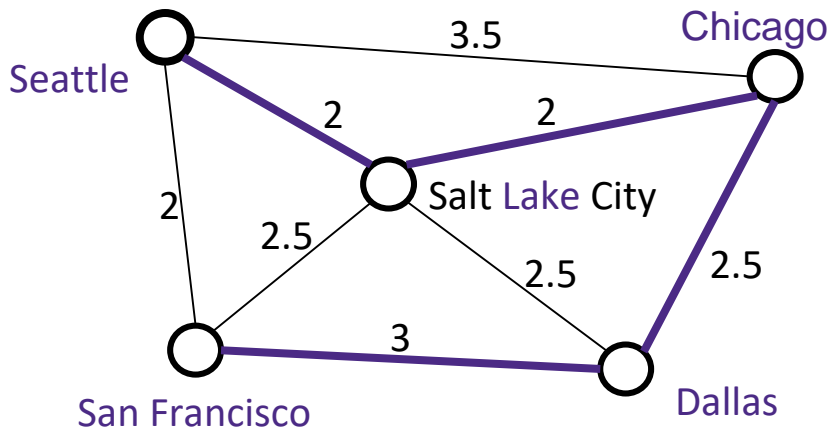
## Paths and Cycles (2 of 2)

- ❖ A graph that does not contain any cycles is *acyclic*



# Path Length and Cost

- ❖ **Path length:** Number of edges in a path
  - Also called “unweighted cost”
- ❖ **Path cost:** Sum of the weights of each edge in a path
- ❖ Example:  $P = [\text{Seattle}, \text{SLC}, \text{Chicago}, \text{Dallas}, \text{SF}]$



length( $P$ ) = 4  
cost( $P$ ) = 9.5

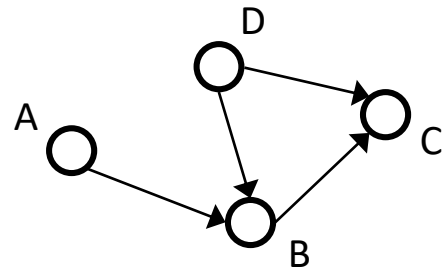


# Poll Everywhere

[pollev.com/cse332](https://pollev.com/cse332)

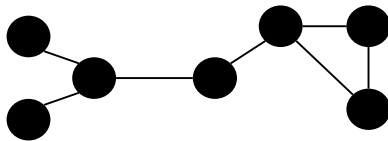
❖ Is there a path from A to D? Does the graph contain any cycles?

- A. Yes / Yes
- B. Yes / No
- C. No / Yes
- D. No / No
- E. I'm not sure ...

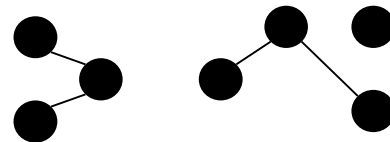


# Undirected Graph Connectivity

- ❖ An undirected graph is **connected** if for all pairs of vertices  $u, v$ , there exists a *path* from  $u$  to  $v$

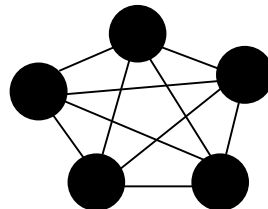


Connected graph



Disconnected graph

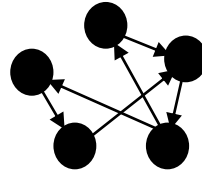
- ❖ An undirected graph is **complete** (aka **fully connected**) if for all pairs of vertices  $u, v$ , there exists an *edge* from  $u$  to  $v$



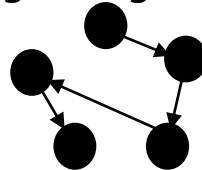
(not pictured: self edges)

# Directed Graph Connectivity

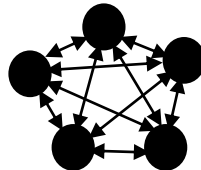
- ❖ A directed graph is **strongly connected** if for all pairs of vertices  $u, v$ , there exists a *path* from  $u$  to  $v$



- ❖ A directed graph is **weakly connected** if for all pairs of vertices  $u, v$ , there exists a path from  $u$  to  $v$  *ignoring direction of edges*



- ❖ A directed graph is **complete** (aka **fully connected**) if for all pairs of vertices  $u, v$ , there exists an *edge* from  $u$  to  $v$



(not pictured: self edges)

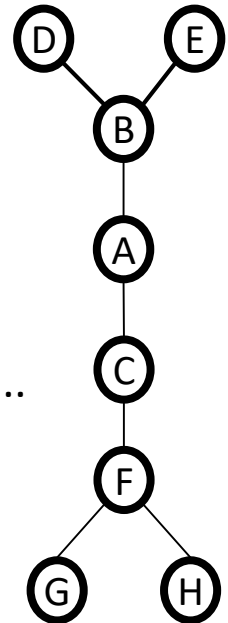
# Example Graphs: Dear Friends, Once More

- ❖ For undirected graphs: *connected?*
- ❖ For directed graphs: *strongly connected?* *weakly connected?*
  - Web pages with links
  - Facebook friends
  - Methods in a program that call each other
  - Road maps (e.g., Google maps)
  - Airline routes
  - Family trees
  - Course pre-requisites

# Trees as Graphs

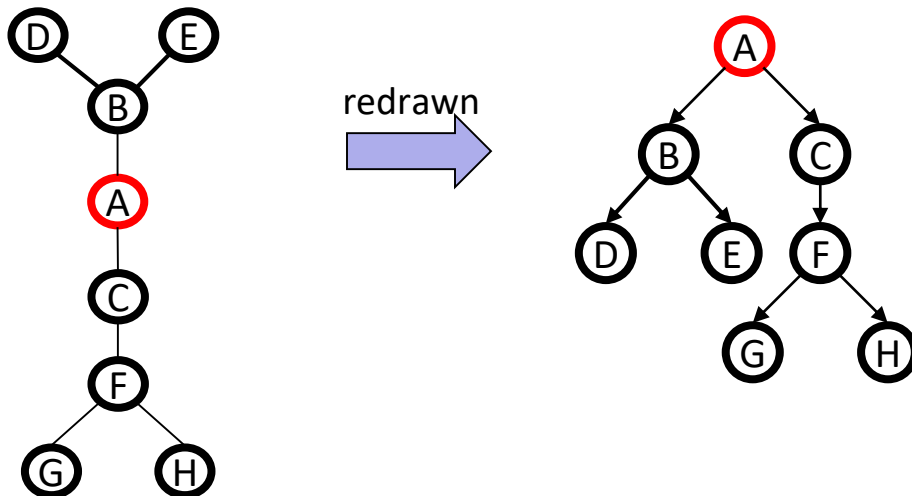
- ❖ A *tree* is a graph that is:
  - undirected
  - acyclic
  - connected
- ❖ So all trees are graphs, but not all graphs are trees
- ❖ How does this relate to the trees we know and love?...

Example:



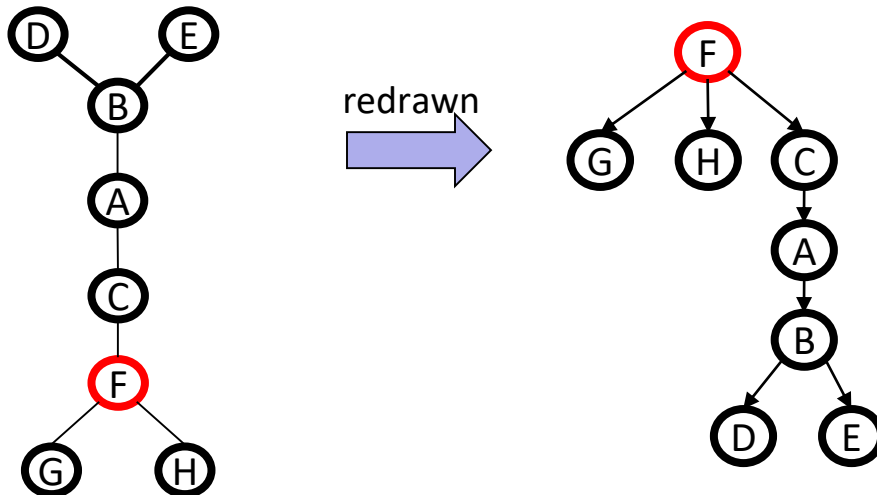
# Rooted Trees (1 of 2)

- ❖ We've previously worked with rooted trees, where:
  - We identify a unique ("special") vertex: the root
  - We think of edges as **directed**: parent to children
- ❖ The same tree can be redrawn as multiple rooted trees depending on which node you pick as the root



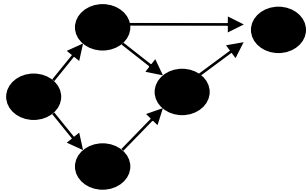
## Rooted Trees (2 of 2)

- ❖ We've previously worked with *rooted trees*, where:
  - We identify a unique ("special") vertex: the root
  - We think of edges as **directed**: parent to children
- ❖ The same tree can be redrawn as multiple rooted trees depending on which *node you pick as the root*



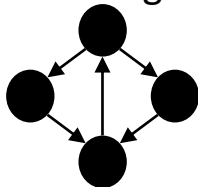
# Directed Acyclic Graphs (aka DAGs)

- ❖ A **DAG** is a directed graph with no directed cycles
- ❖ Every rooted directed tree is a DAG
  - But not every DAG is a rooted directed tree:



← Not a rooted directed tree;  
has an undirected cycle

- ❖ Every DAG is a directed graph (by definition!)
  - But not every directed graph is a DAG:



← Not a DAG; has a  
directed cycle

# Graph Examples: Or Close the Wall Up

- ❖ Which of our *directed*-graph examples do you expect to be a **DAG**?
  - Web pages with links
  - Facebook friends
  - Methods in a program that call each other
  - Road maps (e.g., Google maps)
  - Airline routes
  - Family trees
  - Course pre-requisites

# Density / Sparsity (1 of 2)

## ❖ Recall:

- In an undirected graph,  $0 \leq |E| < |V|^2$
- In a directed graph:  $0 \leq |E| \leq |V|^2$

So for any graph,  
 $|E| \in O(|V|^2)$

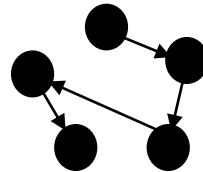
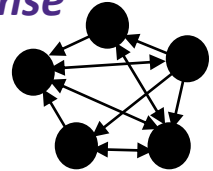
## ❖ One more fact:

- In a *connected* undirected graph,  $|E| \geq |V|-1$
- In a *weakly connected* directed graph,  $|E| \geq |V|-1$
- In a *strongly connected* directed graph,  $|E| \geq |V|$

So for any  
*connected* graph,  
 $|E| \in \Omega(|V|^2)$

## Density / Sparsity (2 of 2)

- ❖ We do not always approximate as  $|E|$  as  $O(|V|^2)$ 
  - This is a *correct* bound, it's just oftentimes not *tight*
- ❖ If it is tight, i.e.  $|E| \in \Theta(|V|^2)$ , we say the graph is *dense*
  - Intuitively: “lots of edges”
- ❖ If  $|E| \in O(|V|)$  we say the graph is *sparse*
  - Sparse: “most (of the possible) edges missing”



# Lecture Outline

- ❖ Graph Definitions: Part 2
- ❖ Graph Representations
  - **Adjacency Matrix**
  - Adjacency List

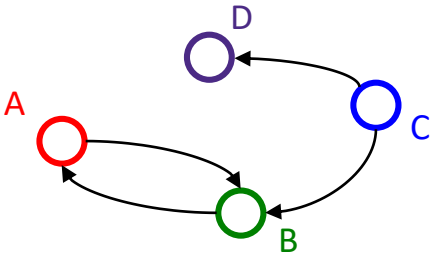
*Lecture questions: [pollev.com/cse332](https://pollev.com/cse332)*

# What is the Data Structure?

- ❖ Is a Graph an ADT? Maybe!
  - “Develop an algorithm over the graph, then use whatever data structure is efficient”
- ❖ The “best” data structure can depend on:
  - Properties of the graph (e.g., dense versus sparse)
  - Common queries
    - e.g., “is  $(\mathbf{u}, \mathbf{v})$  an edge?” vs “what are the neighbors of node  $\mathbf{u}$ ?”
- ❖ There are two standard graph representations:
  - *Adjacency Matrix* and *Adjacency List*
  - Different trade-offs, particularly time vs space

# Adjacency Matrix: Representation

- ❖ Assign each node a number from 0 to  $|\mathcal{V}| - 1$
- ❖ Graph is a  $|\mathcal{V}| \times |\mathcal{V}|$  matrix (ie, 2-D array) of booleans
  - $M[u][v] == \text{true}$  means there is an edge from  $u$  to  $v$

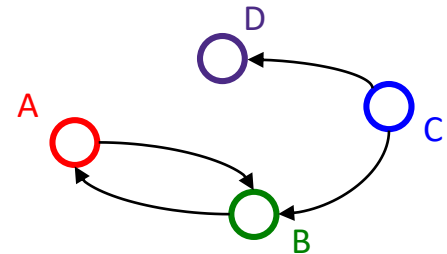


	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F

# Adjacency Matrix: Properties (1 of 3)

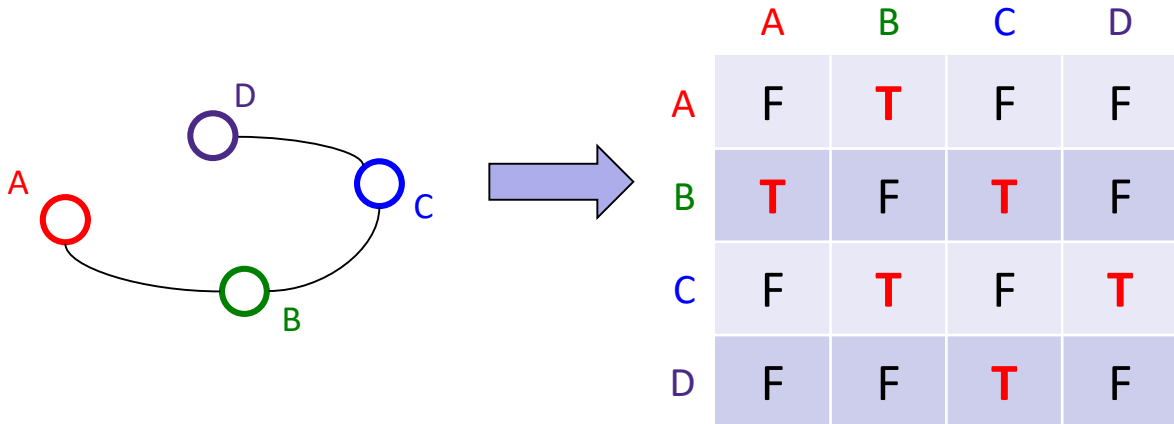
- ❖ Running time to:
  - Get a vertex's out-edges:
    - $O(|V|)$
  - Get a vertex's in-edges:
    - $O(|V|)$
  - Decide if some edge exists:
    - $O(1)$
  - Insert an edge:
    - $O(1)$
  - Delete an edge:
    - $O(1)$
- ❖ Space requirements:
  - $|V|^2$  bits
- ❖ Best for sparse or dense graphs?
  - Best for dense graphs

	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F



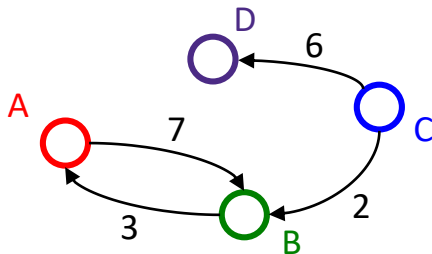
# Adjacency Matrix: Properties (2 of 3)

- ❖ How does the adjacency matrix vary for an *undirected graph*?
  - *Undirected graphs are symmetric about diagonal axis*
  - *Languages with array-of-array matrix representations can save ½ the space by omitting the symmetric half*
    - *Languages with “proper” 2D matrix representations (eg, C/C++) can’t do this*



# Adjacency Matrix: Properties (3 of 3)

- ❖ How can we adapt the representation for *weighted graphs*?
  - *Store the weight in each cell*
  - *Need some value to represent “not an edge”*
    - *In some situations, 0 or -1 works*



	A	B	C	D
A	0	7	0	0
B	3	0	0	0
C	0	2	0	6
D	0	0	0	0

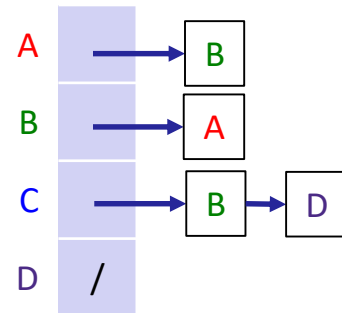
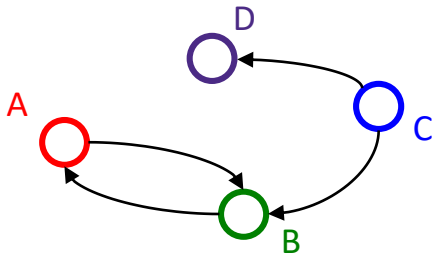
# Lecture Outline

- ❖ Graph Definitions: Part 2
- ❖ Graph Representations
  - Adjacency Matrix
  - **Adjacency List**

*Lecture questions: [pollev.com/cse332](https://pollev.com/cse332)*

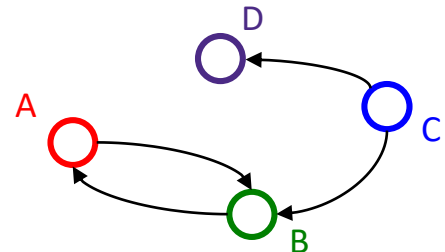
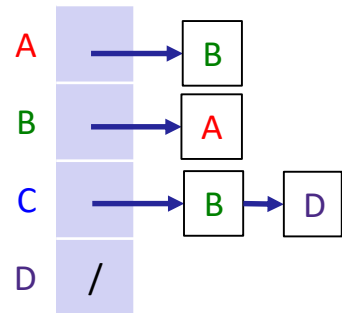
# Adjacency List: Representation

- ❖ Assign each node a number from 0 to  $|\mathcal{V}| - 1$
- ❖ Graph is an array of length  $|\mathcal{V}|$ ; each entry stores a list of all adjacent vertices
  - E.g. linked list



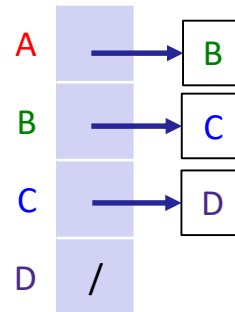
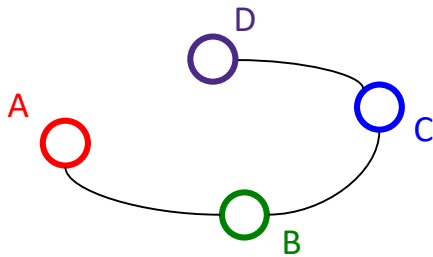
# Adjacency List: Properties (1 of 3)

- ❖ Running time to:
  - Get a vertex's out-edges:
    - $O(d)$  where  $d$  is out-degree of vertex
  - Get a vertex's in-edges:
    - $O(|V| + |E|)$
    - (but could keep a second adjacency list for this!)
  - Decide if some edge exists:
    - $O(d)$  where  $d$  is out-degree of source vertex
  - Insert an edge:
    - $O(1)$
    - (unless you need to check if it's there; then  $O(d)$ )
  - Delete an edge:
    - $O(d)$  where  $d$  is out-degree of source vertex
- ❖ Space requirements:
  - $O(|V| + |E|)$
- ❖ Best for sparse or dense graphs?
  - Best for sparse graphs, so usually just stick with linked lists for the buckets

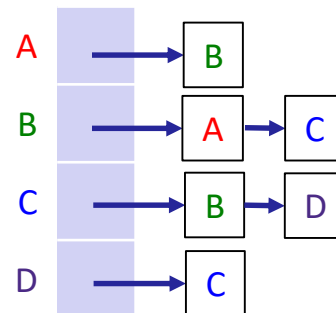


# Adjacency List: Properties (2 of 3)

- ❖ How does the adjacency list vary for an *undirected* graph?
  - *Optionally, can double the entries to increase edge lookup speed*

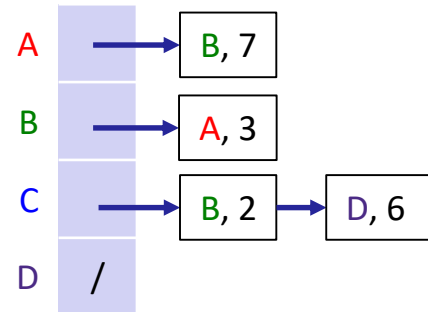
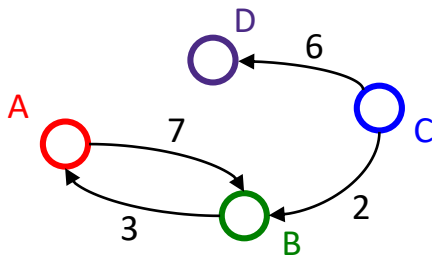


... or ...



# Adjacency List: Properties (3 of 3)

- ❖ How can we adapt the representation for *weighted graphs*?
  - *Store the weight alongside the destination vertex*
  - *No need for a special value to represent “not an edge”!*



# Summary: Which is Better?

- ❖ Graphs are often sparse:
  - Road networks are often grids
    - Every corner isn't connected to every other corner
  - Airlines rarely fly to all possible cities
    - Or if they do it is to/from a hub
- ❖ Adjacency lists should generally be your default choice
  - Slower performance compensated by greater space savings
  - Many graph algorithms rely heavily on `getAllEdgesFrom(v)`

	<code>getAllEdgesFrom(v)</code>	<code>hasEdge(v, w)</code>	<code>getAllEdges()</code>
Adjacency Matrix	$\Theta(V)$	$\Theta(1)$	$\Theta(V^2)$
Adjacency List	$O(V)$	$\Theta(\text{degree}(v))$	$\Theta(E + V)$