Lecture 14: Introduction to Graphs

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Announcements

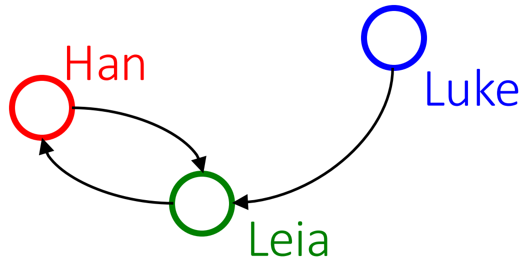
- EX05 due today
- EX06 due Friday
- Don’t talk about Exam 1!
  - Still makeups to proctor
- Exam 2 information posted on website

Today

- Graphs
  - Introduction
  - Terminologies
- Graph Data Structures
  - Adjacency Matrix
  - Adjacency List

Graphs: Basic Mathematical

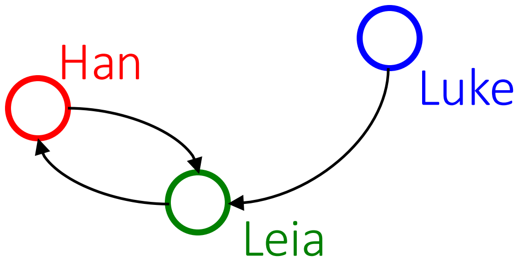
- A graph is a mathematical representation of a set of objects (vertices/nodes) connected by links (edges).
- A graph G is a pair of sets (V, E) where:
  - $V = \{v_1, v_2, \ldots, v_n\}$, a set of vertices (or nodes)
  - $E = \{e_1, e_2, \ldots, e_m\}$, a set of edges
    - Where each edge $e_i = (v_j, v_k)$, a pair of vertices
    - An edge "connects" the vertices




```
V = {Han,Leia,Luke}
E = {(Luke,Leia),
     (Han,Leia),
     (Leia,Han)}
```

Graphs: Basic Intuition

- A bunch of circles and arrows




```
V = {Han,Leia,Luke}
E = {(Luke,Leia),
     (Han,Leia),
     (Leia,Han)}
```

Graphs: Terminology Vomit (Memorize!)

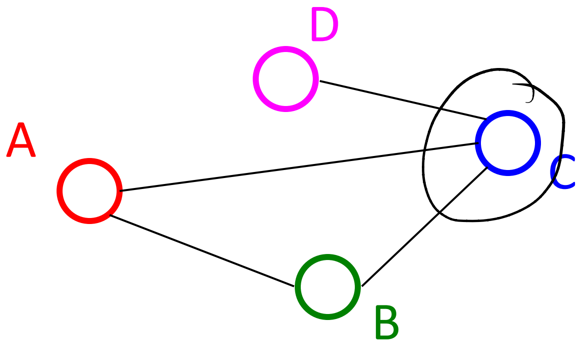
- Vertex (or Nodes) ✓
- Edges ✓
- Directed vs Undirected
- Weighted vs Unweighted
- Degree (of a Vertex)
  - In-Degree
  - Out-Degree
- Walk vs Path (or Simple Path) vs Cycles
  - Cyclic vs Acyclic
- Connected vs Disconnected
- Sparse vs Dense
- and many more...

Graphs: Yet Another Internet Warning

There are millions of different terminologies, algorithms, etc. with graphs. Use lecture slides.

Graphs: Undirected Graphs

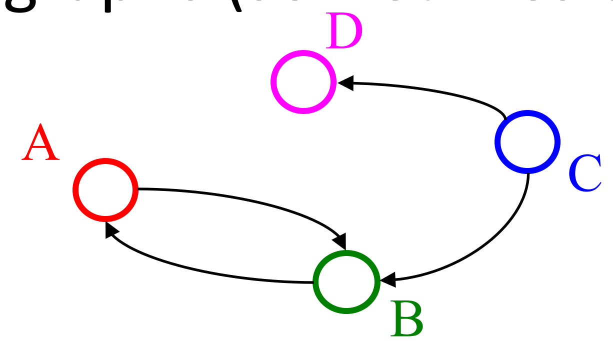
- In Undirected graphs, edges have no specific direction
  - Edges are always "two-way"



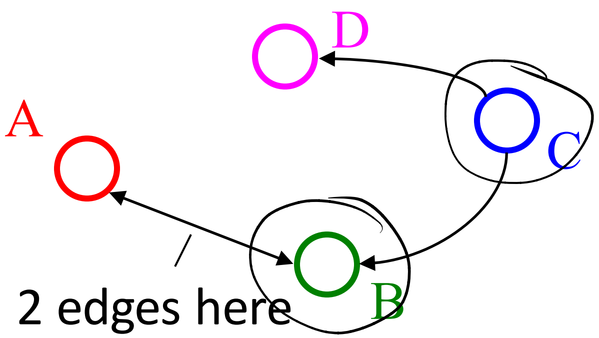
- Thus, $(v, u) \in E$ imply $(u, v) \in E$
  - Only one of these edges needs to be in the set; the other is implicit
- Degree of a vertex: number of edges containing that vertex
  - Put another way: the number of adjacent vertices

Graphs: Directed Graphs

- In Directed graphs ~~(sometimes called digraphs)~~, edges have a direction

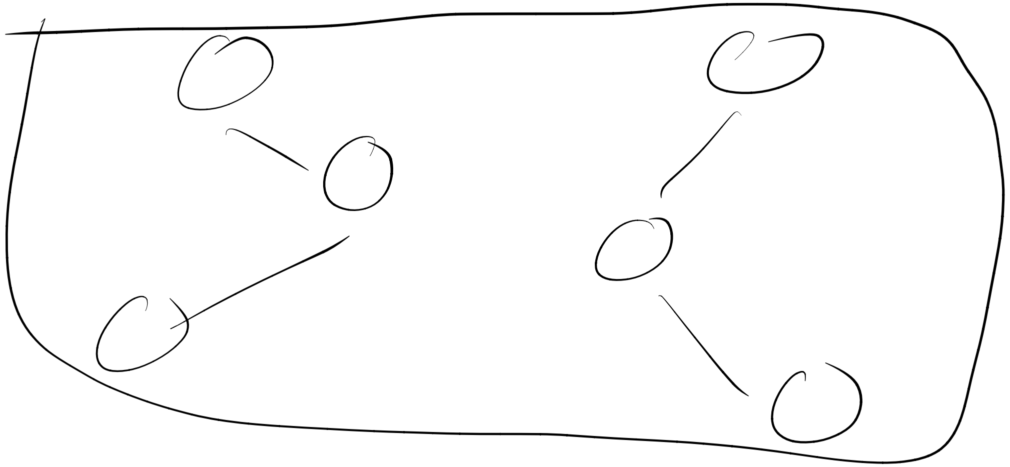


or

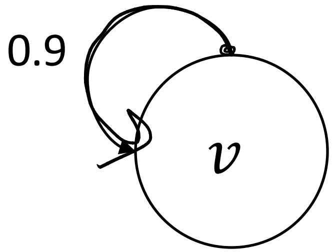


- Thus, $(v, u) \in E$ DOES NOT imply $(u, v) \in E$
  - $(v, u) \in E$ intuitively means $v \rightarrow u$
  - v is the source and u is the destination
- In-Degree of a vertex w: number of In-bound edges
  - i.e., edges where w is the destination
- Out-Degree of a vertex w: number of Out-bound edges
  - i.e., edges where w is the source

Graphs: Self-Edges



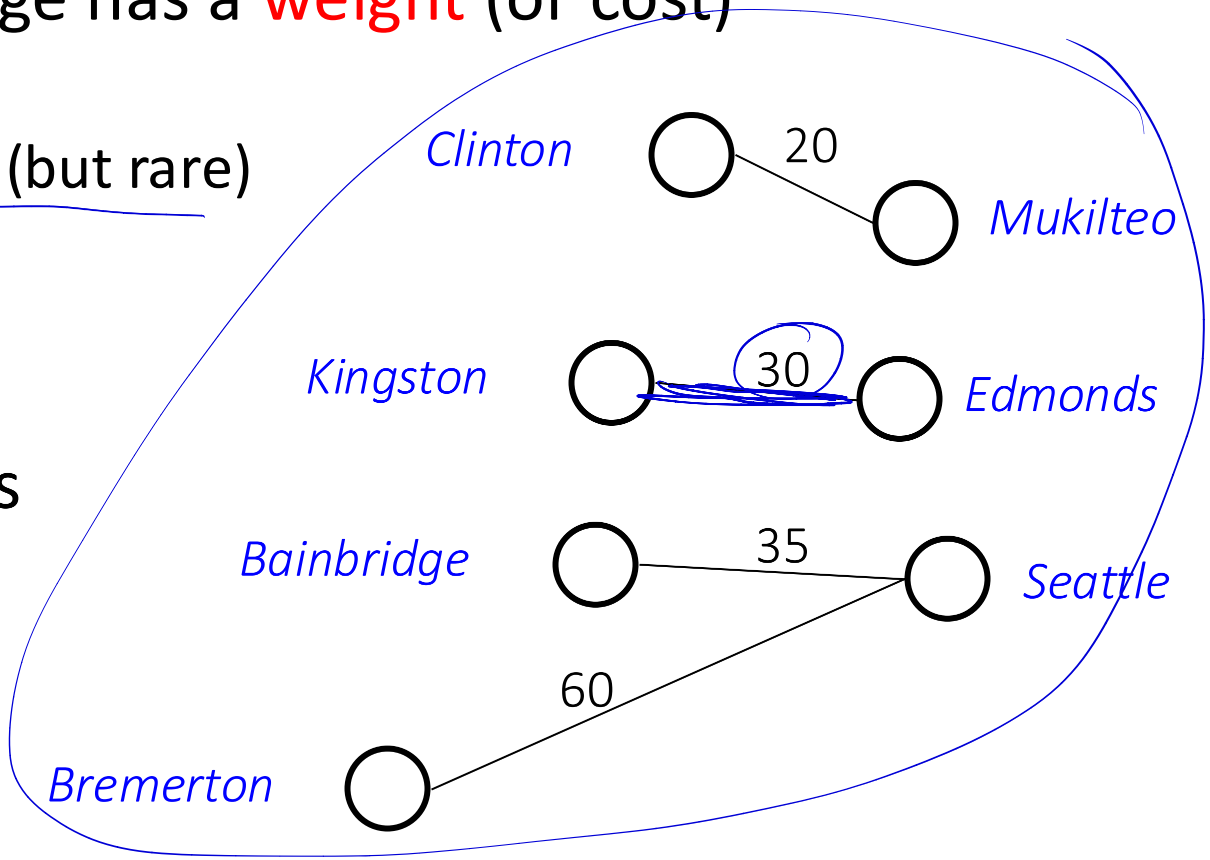
- We pretend they don't exist
- A self-edge a.k.a. a self-loop is an edge of the form (v, v)
  - Depending on the use/algorithm, a graph may have:
    - No self-edges
    - Some self-edges
    - All self-edges (often therefore implicit, but we will be explicit)



- A node can have a degree / in-degree / out-degree of zero
- A graph does not have to be connected (In an undirected graph, this means we can follow edges from any node to every other node), even if every node has non-zero degree

Graphs: Weighted Graphs

- In a weighted graph, each edge has a weight (or cost)
  - Typically, a number (int)
  - Negative weights are possible (but rare)



So far, possible graph types:

Undirected Unweighted graphs

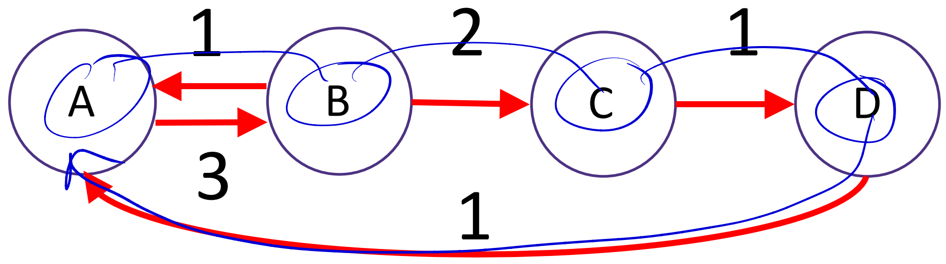
Undirected Weighted graphs

Directed Unweighted graphs

Directed Weighted graphs

Any Questions?

Graphs: (Walks) vs Paths vs Cycles



- Walk: Sequence of adjacent vertices
  - e.g., ABA, ABCD, ABC, etc.
- Path (or Simple Path): A walk that doesn't repeat a vertex
  - e.g., ABCD, ABC, AB
  - NOT ABA
- Cycle: A walk that doesn't repeat a vertex except the first and last vertex
  - e.g., ABCDA
  - NOT ABCD

__X__ Length: Number of edges in __X__

__X__ Cost: Sum of weights of each edge in __X__

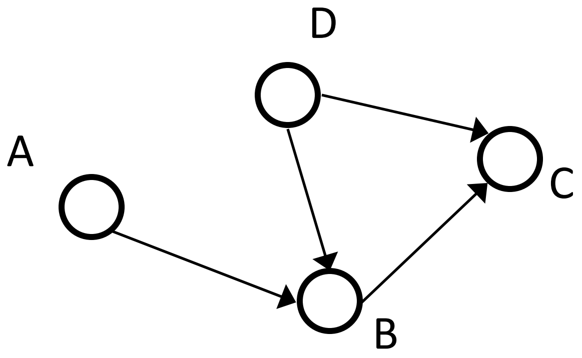
Graphs: Paths vs Cycles Example

- Is there a path from A to D?

NO

- Does the graph contain any cycles?

NO



- What if undirected?

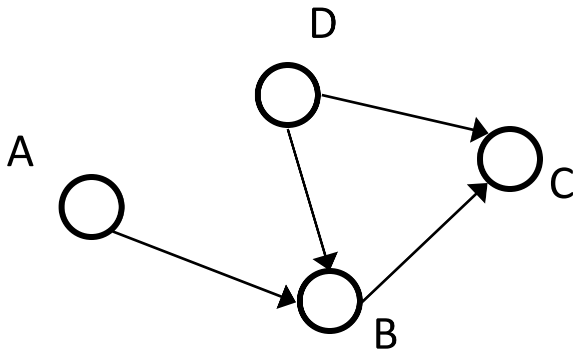
Yes (A B D) , Yes, (B D C B)

Graphs: Paths vs Cycles Example (Soln.)

- Is there a path from A to D?

No

- Does the graph contain any cycles? No

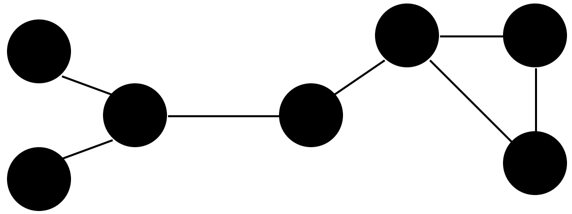


- What if undirected?

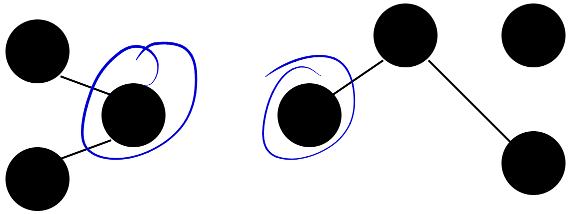
Yes, Yes

Graphs: Undirected Graph Connectivity

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

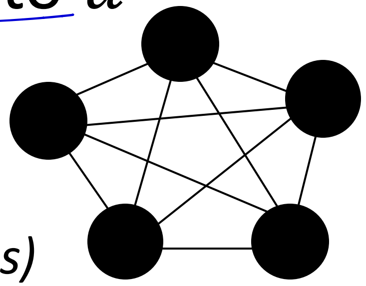


Connected graph



Disconnected graph

- An undirected graph is complete, a.k.a. fully connected if for all pairs of vertices (v, u), there exists an edge from v to u

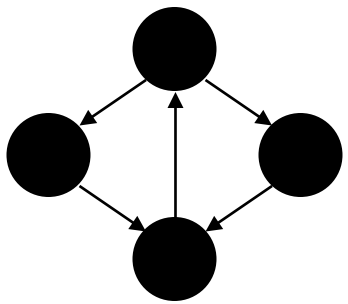


(plus self-edges)

K_5

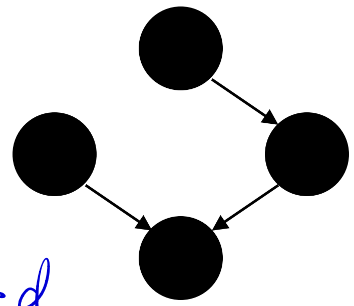
Graphs: Directed Graph Connectivity

- A directed graph is strongly connected if there is a path from every vertex to every other vertex



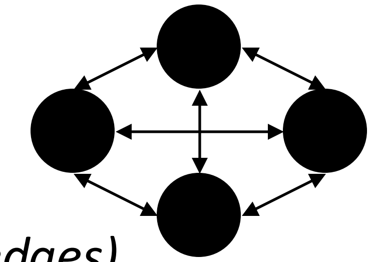
(v, u)   (u, v)

- A directed graph is weakly connected if there is a path from every vertex to every other vertex ignoring direction of edges



if undirected version is connected, weakly connected

- A directed graph is complete a.k.a. fully connected if for all pairs of vertices (v, u), there exists an edge from v to u



(plus self-edges)

(v, u)   (u, v)

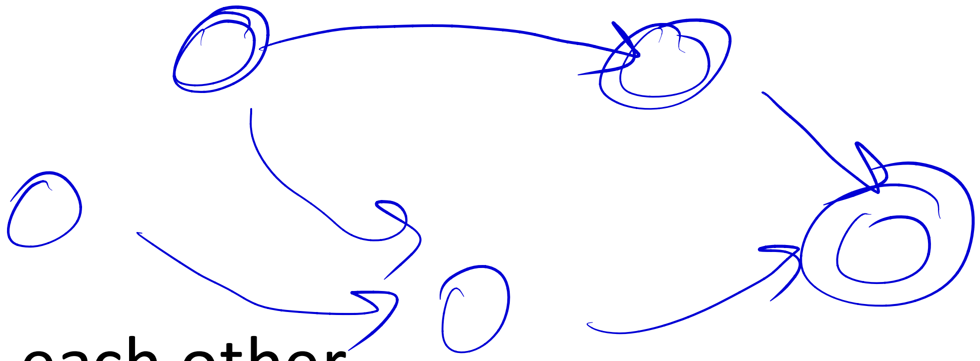
Graphs: Practical Examples

For undirected graphs: connected?

For directed graphs: strongly connected? weakly connected?

weighted?

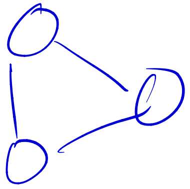
- Web pages with links
- Facebook friends
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Course pre-requisites
- …



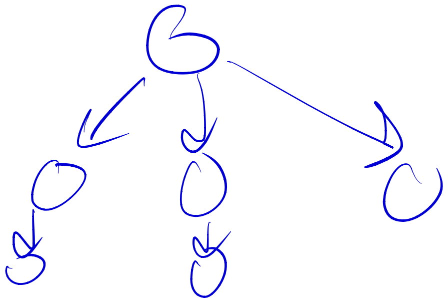
Graphs: Trees

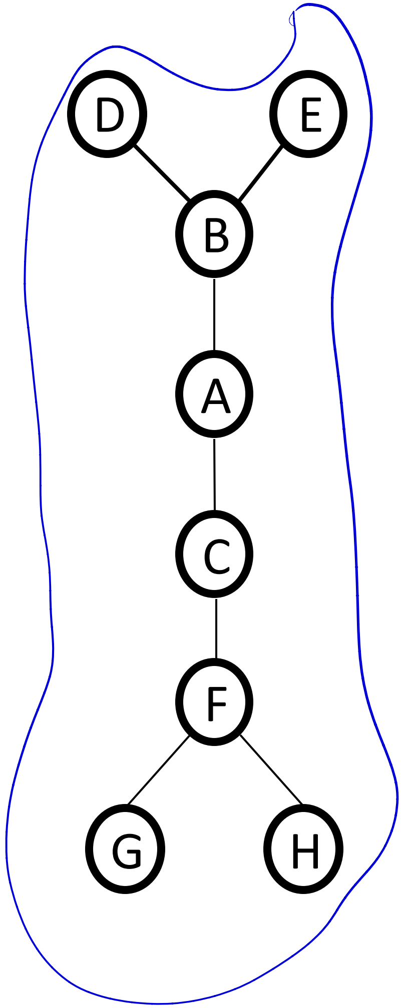
- When talking about graphs, we say a tree is a graph that is:
  - undirected
  - acyclic
  - connected

- So all trees are graphs, but not all graphs are trees



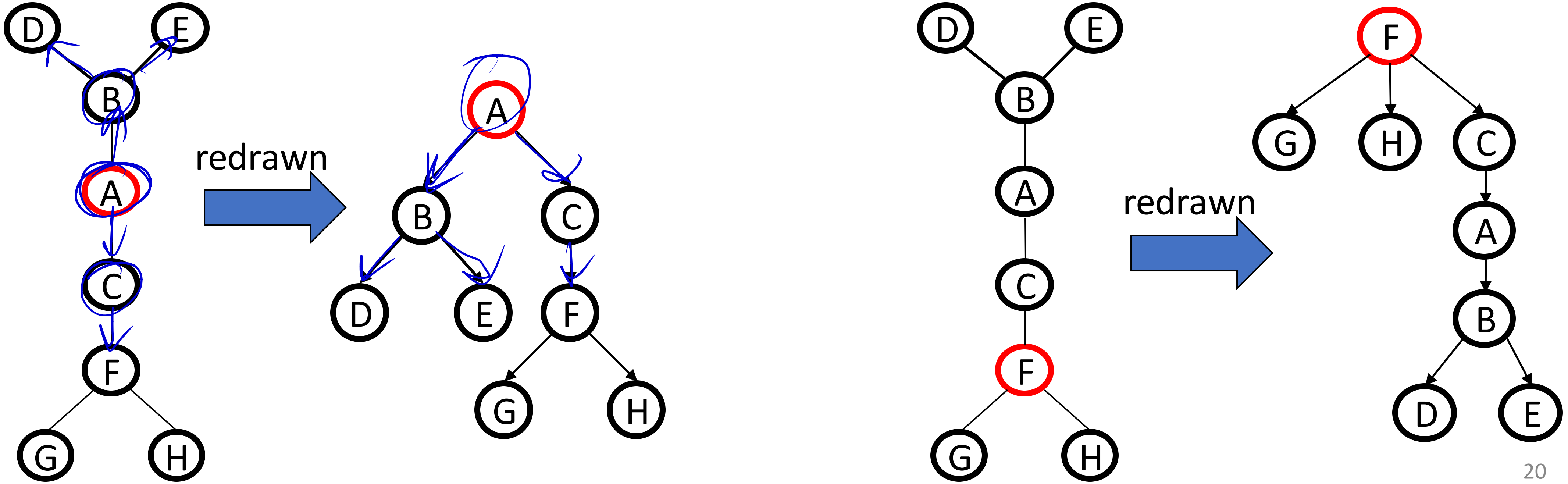
- How does this relate to the trees we know and love?...





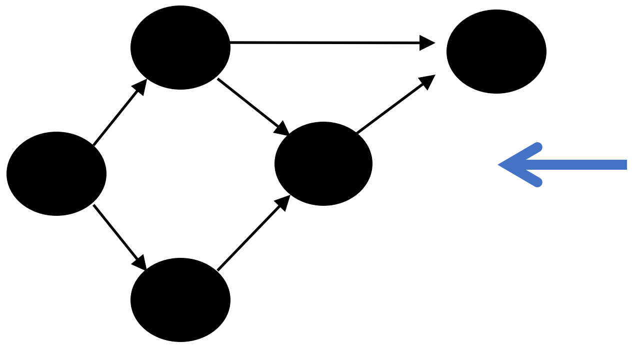
Graphs: Rooted Trees

- We are more accustomed to rooted trees where:
  - We identify a unique ("special") root
  - We think of edges as **directed**: parent to children
- Given a tree, once you pick a root, you have a unique rooted tree (just drawn differently and with undirected edges)



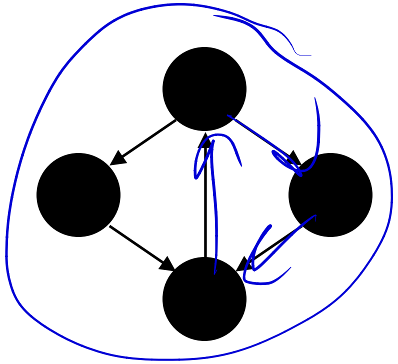
Graphs: Directed Acyclic Graphs (DAGs)

- A DAG is a directed graph with no cycles (Acyclic)
  - Every rooted directed tree is a DAG
    - But not every DAG is a rooted directed tree:



Not a rooted directed tree,
Has a cycle (in the undirected sense)

- Every DAG is a directed graph
  - But not every directed graph is a DAG:



Graphs: Number of Vertices vs Edges (Math)

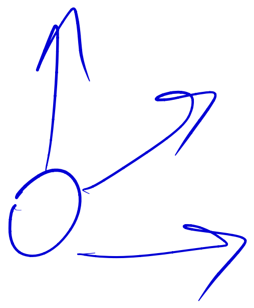
- Correct Mathematical Notation:
  - Number of Vertices = $|\{v_1, v_2, \ldots, v_n\}| = |V|$
  - Number of Edges = $|\{e_1, e_2, \ldots, e_m\}| = |E|$
- Common Notation: V or E
- Given $|V|$ vertices, what is:
  - Minimum number of Edges?

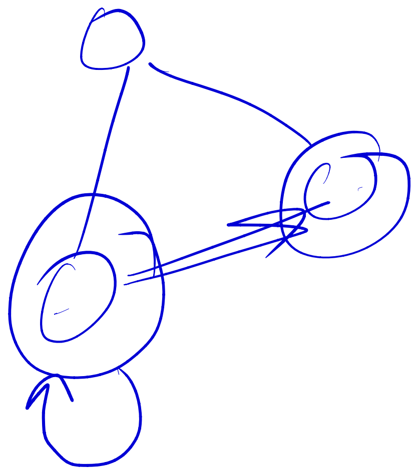
    0
  - Maximum for undirected?
  - Maximum for directed?

$v(v-1)$

$S = \{o_1, o_2, o_3\}$

$|S| = 3$





$\frac{2+2+2}{2} + 3$

Graphs: Number of Vertices vs Edges (Math)

- Correct Mathematical Notation:
  - Number of Vertices = $|\{v_1, v_2, \ldots, v_n\}| = |V|$
  - Number of Edges = $|\{e_1, e_2, \ldots, e_m\}| = |E|$
- Common Notation: V or E
- Given $|V|$ vertices, what is:
  - Minimum number of Edges?
    - 0
  - Maximum for undirected?
    - $\frac{V(V+1)}{2}$ (with self-edges) or $\frac{V(V+1)}{2} - V$ (no self-edges)
  - Maximum for directed?
    - V^2

(w/ self-edges) $V(V-1)$ w/o self edges

Graphs: Sparse vs Dense Graphs

- In a graph,
  - Undirected, $0 \le |E| < |V|^2$
  - Directed: $0 \le |E| \le |V|^2$
- So: $|E| \in \mathcal{O}(|V|^2)$
- Sparse: when $|E| \in \Theta(|V|)$ i.e., "few edges"
- Dense: when $|E| \in \Theta(|V|^2)$ i.e., "many edges"

Sparse **Dense**

0 edges $\mathcal{O}(|V|)$ edges $\mathcal{O}(|V|^2)$ edges

Any Questions?

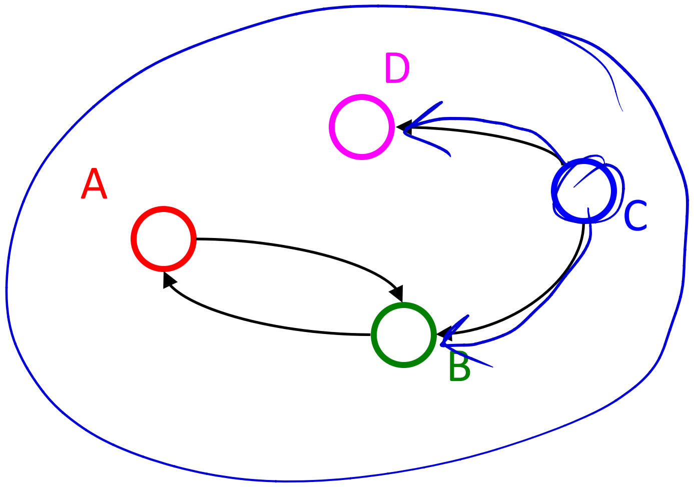
Graphs: The Data Structure

- Many data structures, tradeoffs
- Exploits graph properties
- Common operations:
  - "Is (v, u) an edge?"
  - "What are the neighbors of v?"

- Two standards:
  - Adjacency Matrix
  - Adjacency List

Graphs: Adjacency Matrix

- Assign each node a number from 0 to $|V| - 1$
- A $|V|$ by $|V|$ matrix `M` (2-D array) of Booleans
- `M[v][u]==true` means there is an edge from `v` to `u`



To

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

Any Questions?

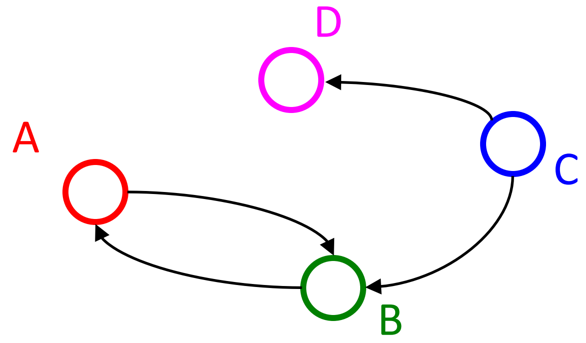
Adjacency Matrix: Properties

- Running time to:
  - Get a vertex's out-bound edges: $O(V)$
  - Get a vertex's in-bound edges: $O(V)$
  - Decide if some edge exists: $O(1)$
  - Insert an edge: $O(1)$
  - Delete an edge: $O(1)$
- Space requirements: $O(|V|^2)$
- Better for Sparse or Dense Graphs?

Dense

To

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



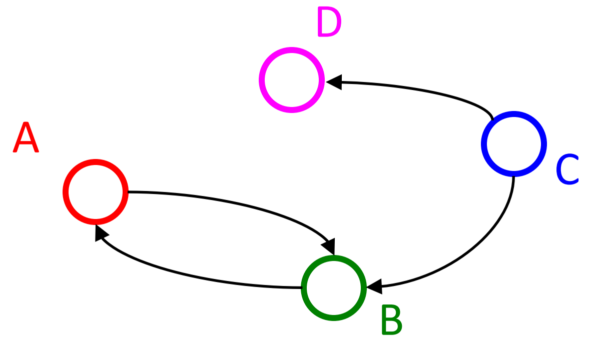
Adjacency Matrix: Properties (Soln.)

- Running time to:
  - Get a vertex's out-bound edges: $\mathcal{O}(|V|)$
  - Get a vertex's in-bound edges: $\mathcal{O}(|V|)$
  - Decide if some edge exists: $\mathcal{O}(1)$
  - Insert an edge: $\mathcal{O}(1)$
  - Delete an edge: $\mathcal{O}(1)$
- Space requirements: $\mathcal{O}(|V|^2)$
- Better for Sparse or Dense Graphs? Dense

To

From

	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: Adaptability

- How does it work for undirected graph?
- How does it work for weighted graph?

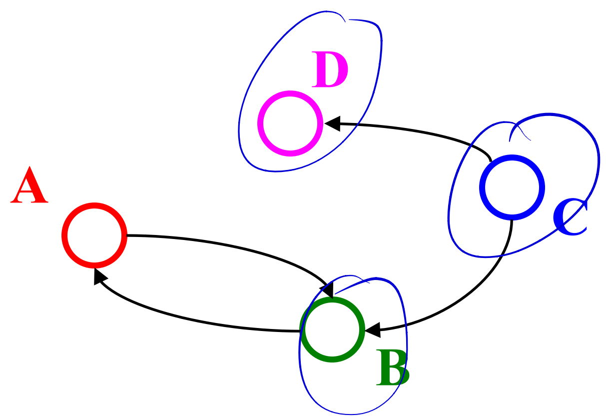
Adjacency Matrix: Adaptability (Soln.)

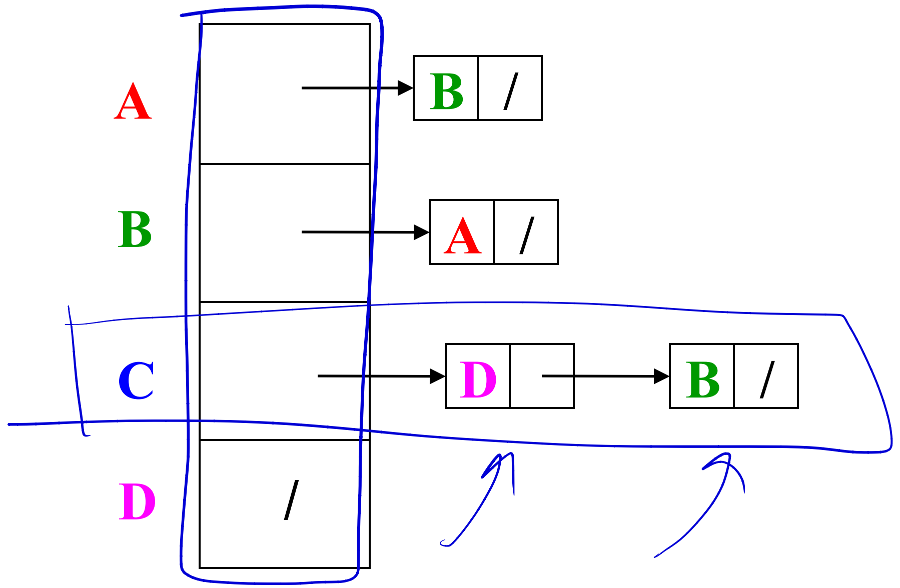
- How does it work for undirected graph?
  - Symmetric in diagonal axis (e.g., `M[v][u]==true`, then `M[u][v]==true`)
- How does it work for weighted graph?
  - Instead of boolean, use integer
  - "not an edge" can be 0, -1, infinite, etc.

M[u][v] = 5

Graphs: Adjacency List

- Assign each node a number from 0 to $|V| - 1$
- An array `arr` of length $|V|$ where `arr[i]` stores a (linked) list of all adjacent vertices





Any Questions?

Adjacency List: Properties

- Running time to:
  - Get a vertex's out-bound edges:

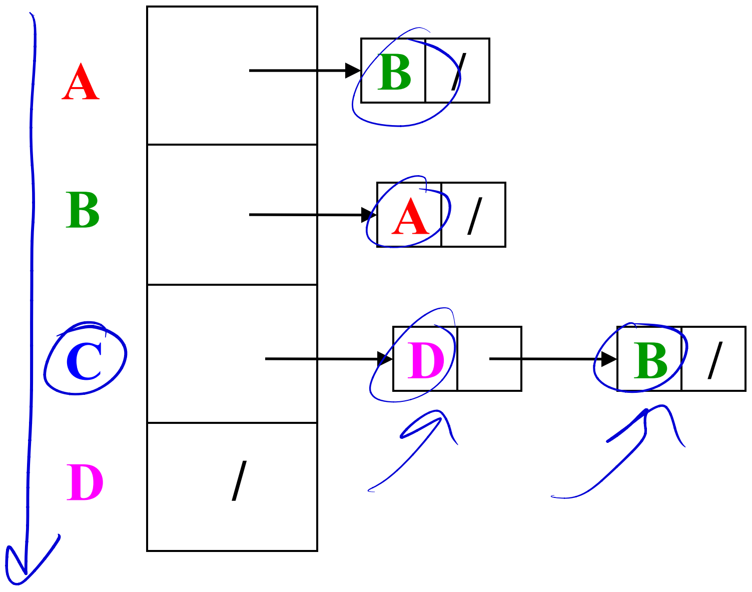
    $O(d)$ d = degree
  - Get a vertex's in-bound edges:

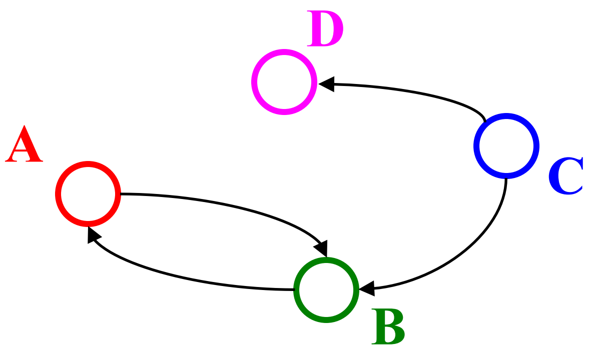
    $O(|V| + |E|)$
  - Decide if some edge exists:

    $O(d)$
  - Insert an edge:

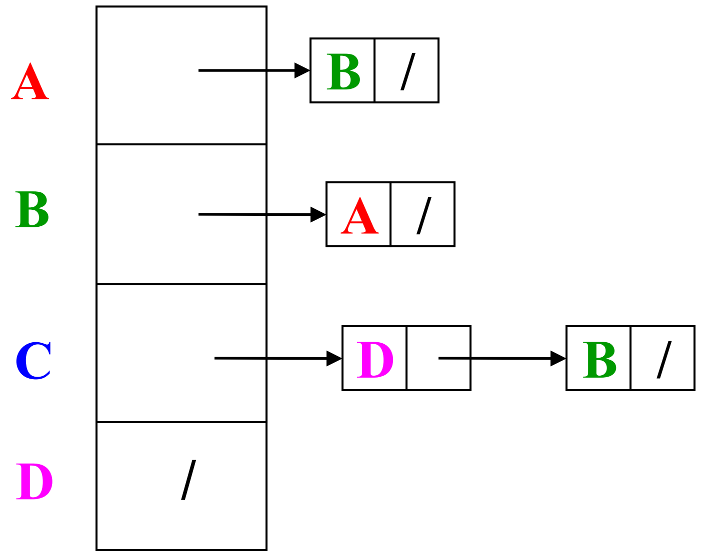
    $O(1)$
  - Delete an edge:

    $O(d)$
- Space requirements: $O(|V| + |E|)$
- Better for Sparse or Dense Graphs? Sparse

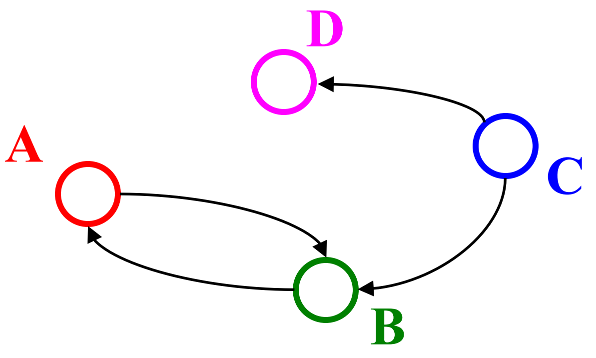




Adjacency List: Properties (Soln.)



- Running time to:
  - Get a vertex's out-bound edges:
    - $\mathcal{O}(d)$, where d is out-degree of vertex
  - Get a vertex's in-bound edges:
    - $\mathcal{O}(|V| + |E|)$, note: can keep 2nd "reverse" adjacency list for faster
  - Decide if some edge exists:
    - $\mathcal{O}(d)$, where d is out-degree of source vertex
  - Insert an edge:
    - $\mathcal{O}(1)$, unless you need to check for duplicates then $\mathcal{O}(d)$
  - Delete an edge:
    - $\mathcal{O}(d)$
- Space requirements: $\mathcal{O}(|V| + |E|)$
- Better for Sparse or Dense Graphs? Sparse



Any Questions?

Matrix vs List, which is better?

- Graphs are often sparse:
  - Streets form grids
    - every corner is not connected to every other corner
  - Airlines rarely fly to all possible cities
    - or if they do it is to/from a hub rather than directly to/from all small cities to other small cities

- Adjacency lists should generally be your default choice
  - Slower performance compensated by greater space savings

Matrix vs List, which is better?

- Graphs are often sparse:
  - Streets form grids
    - every corner is not connected to every other corner
  - Airlines rarely fly to all possible cities
    - or if they do it is to/from a hub rather than directly to/from all small cities to other small cities

- Adjacency lists should generally be your default choice
  - Slower performance compensated by greater space savings