CS 2124: DATA STRUCTURES
Spring 2024

DFS is a great way to solve mazes and other puzzles that have a single solution.

- Lecture 11.1
  - Introduction to Graphs – I

Topics

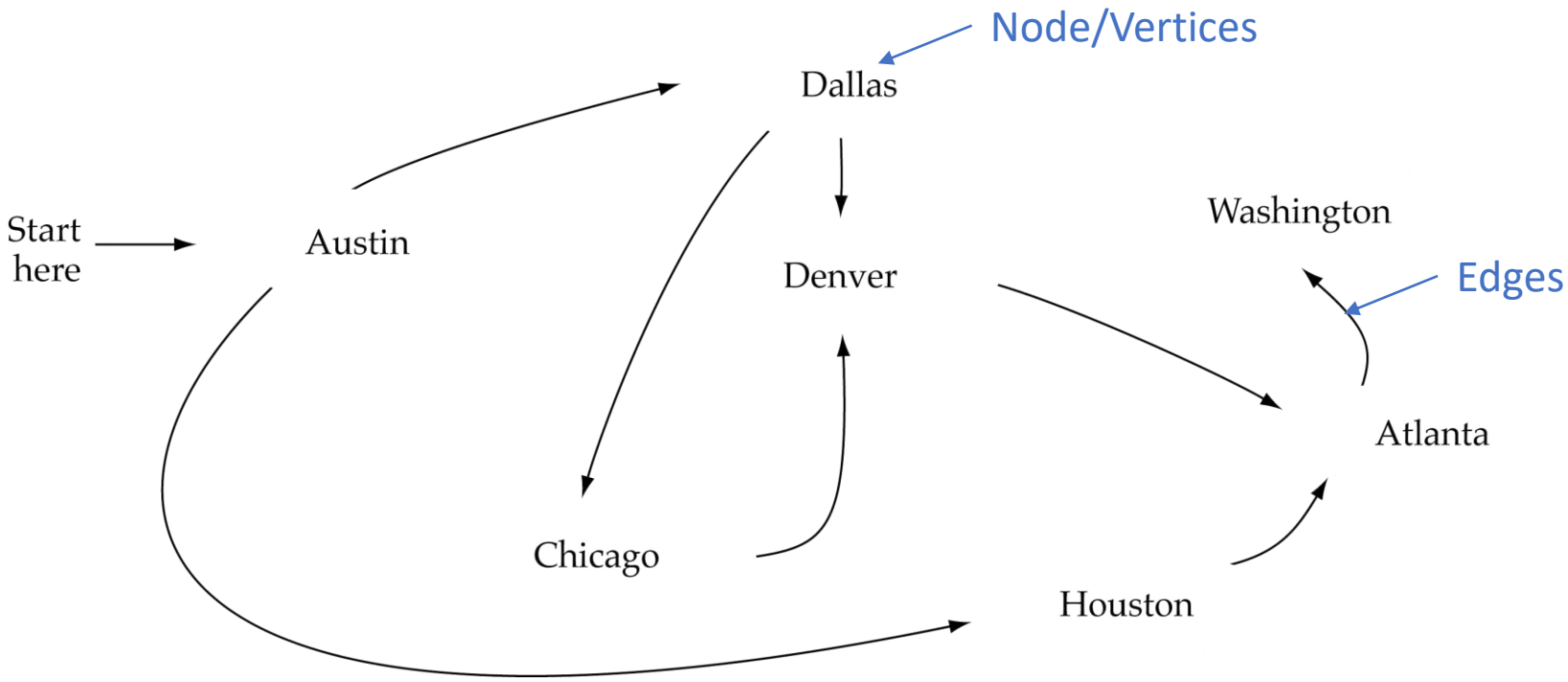
1. Graphs
    a) Directed Graph
    b) Undirected graphs
2. Applications of Graphs
3. Bounds on the number of edges
4. Degree of a vertex
5. Weighted Graphs
6. Handshaking Lemma

Next Lecture

7. Adjacency-matrix representation
8. Adjacency-list representation
9. Graphs in C
10. Using Adjacency Matrix for Path Matrix
11. Warshall's Algorithm
12. Graph Representation

What is a Graph?

- A data structure that consists of a set of nodes (*vertices*) and a set of edges that relate the nodes to each other
- The set of edges describes relationships among the vertices



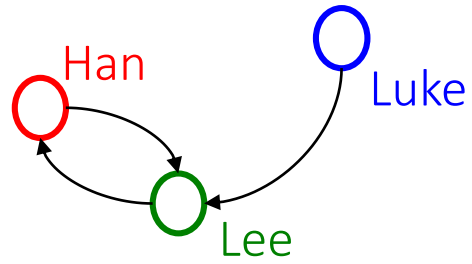
Graphs

A graph is a way for representing relationships among items

- Very general definition
- Very general concept

A graph is a pair: G = (V, E)

- G = graph
- A set of vertices, also known as nodes: $V = \{v_1, v_2, \ldots, v_n\}$
- A set of edges $E = \{e_1, e_2, \ldots, e_m\}$
  - Each edge e_i is a pair of vertices $E = (v_j, v_k)$
  - An edge "connects" the vertices



$G = \{V, E\}$
$V = \{\text{Han}, \text{Lee}, \text{Luke}\}$
$E = \{(\text{Luke}, \text{Lee}), (\text{Han}, \text{Lee}), (\text{Lee}, \text{Han})\}$

A Graph ADT?

We can think of graphs as an ADT*

- Operations would include is Edge(v_j,v_k)
- But it is unclear what the "standard operations" would be for such an ADT
- As in computer science, a graph is an ADT that is meant to implement the undirected graph and directed graph concepts from the field of graph theory within mathematics.

- Instead we tend to develop algorithms over graphs and then use data structures that are efficient for those algorithms

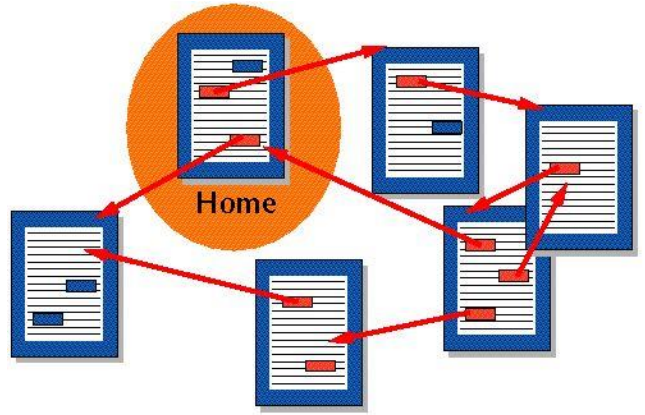
Many important problems can be solved by:

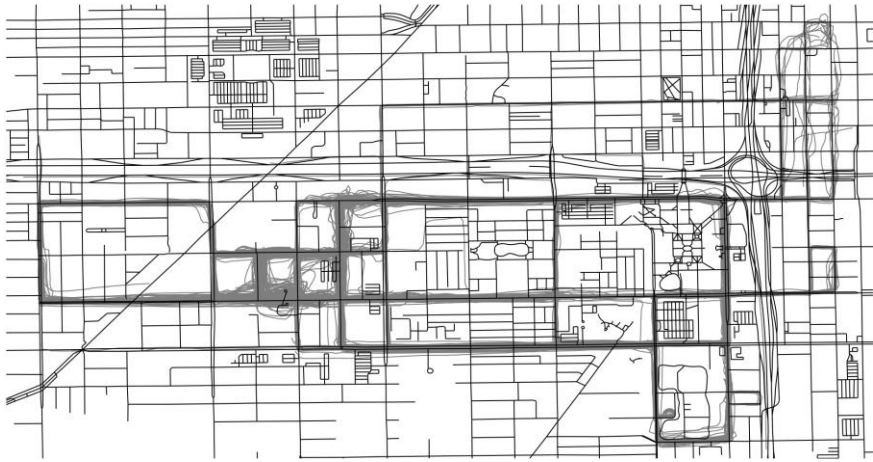
1. Formulating them in terms of graphs
2. Applying a standard graph algorithm

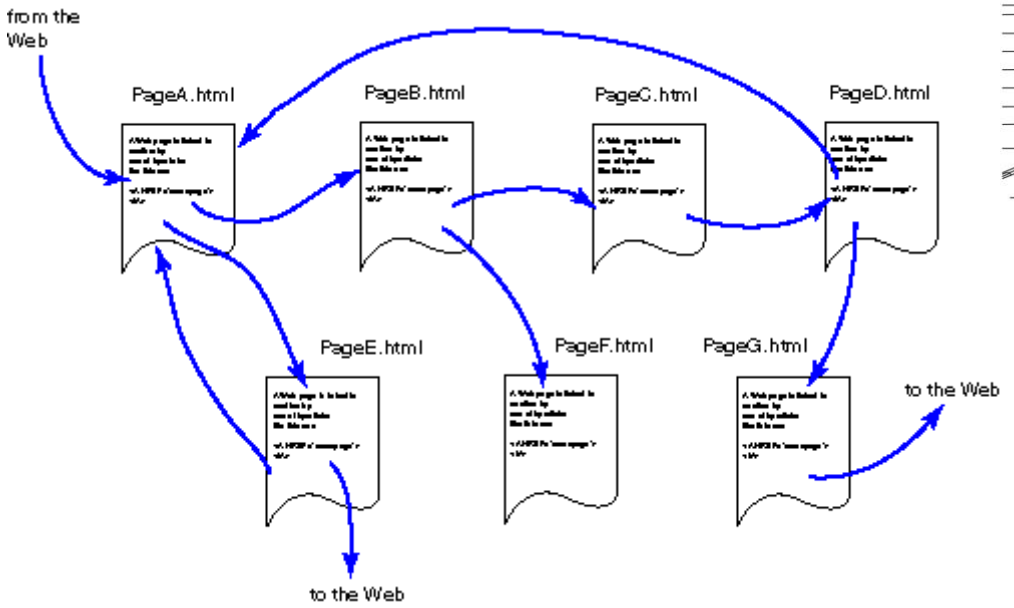
**An ADT (abstract data type) is a mathematical model for data types.*

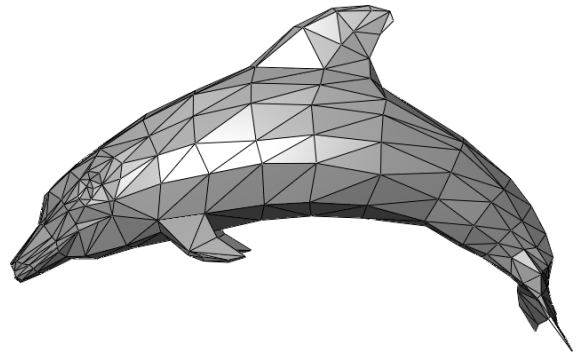
Applications of Graphs

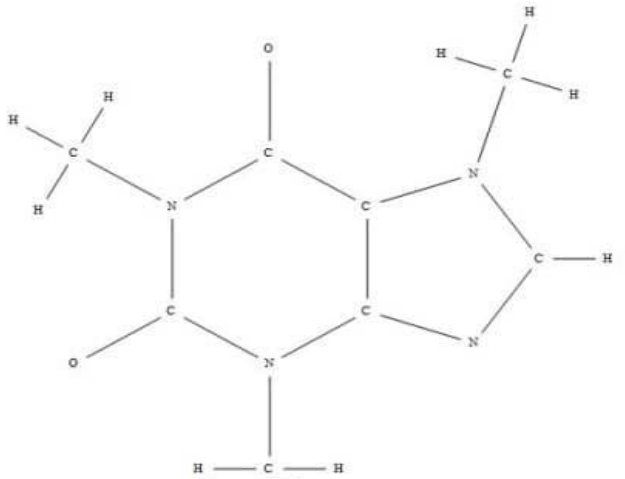
- Web pages with links
- Social media friends
- "Input data" for the Kevin Bacon game (Link)
- Methods in a program that call each other
- Road maps
- Airline routes
- Family trees
- Course pre-requisites



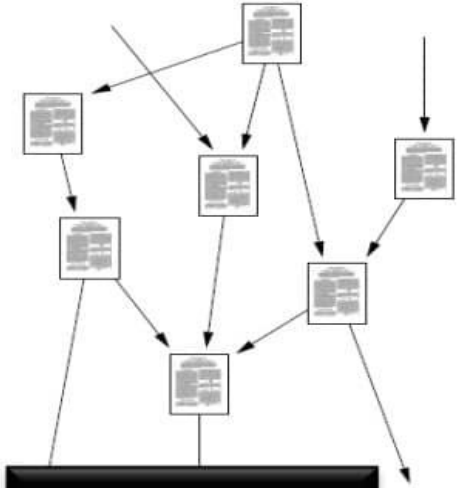




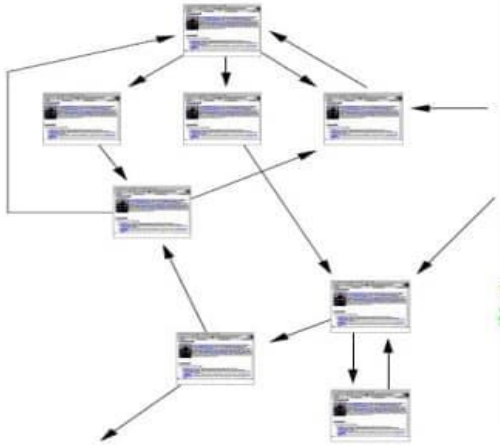




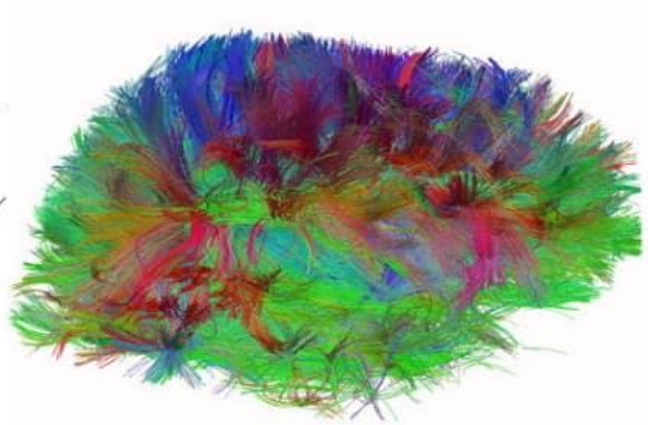
Molecules



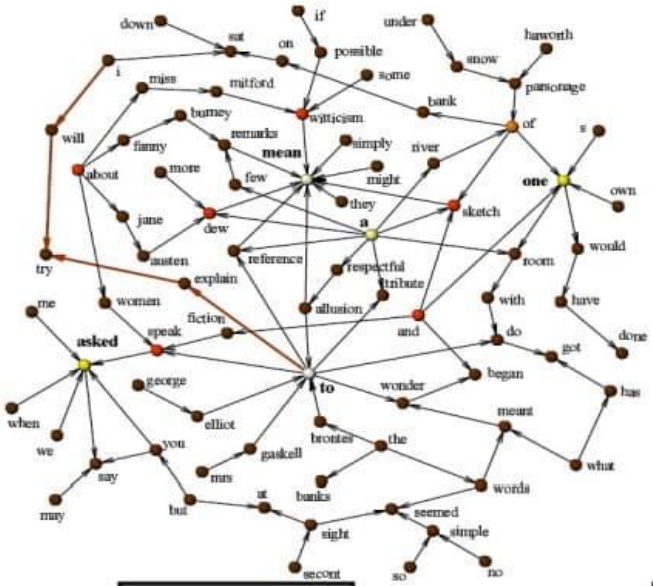
Knowledge



Information



Brain/neurons



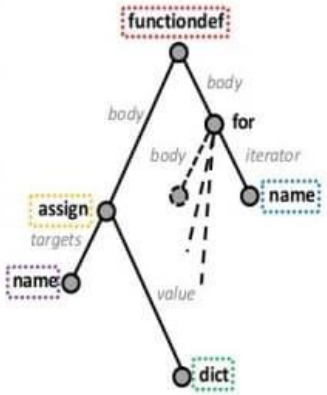
Genes

Communication

```
def encode(obj):
    """
    Encode a (possibly nested)
    dictionary containing complex values
    into a form that can be serialized
    using JSON.
    """
    e = {}
    for key,value in obj.items():
        if isinstance(value,dict):
            e[key] = encode(value)
        elif isinstance(value,complex):
            e[key] = {'type' : 'complex',
                      'r' : value.real,
                      'i' : value.imag}
    return e
```

```
import ast
tree = ast.parse("")
...
```

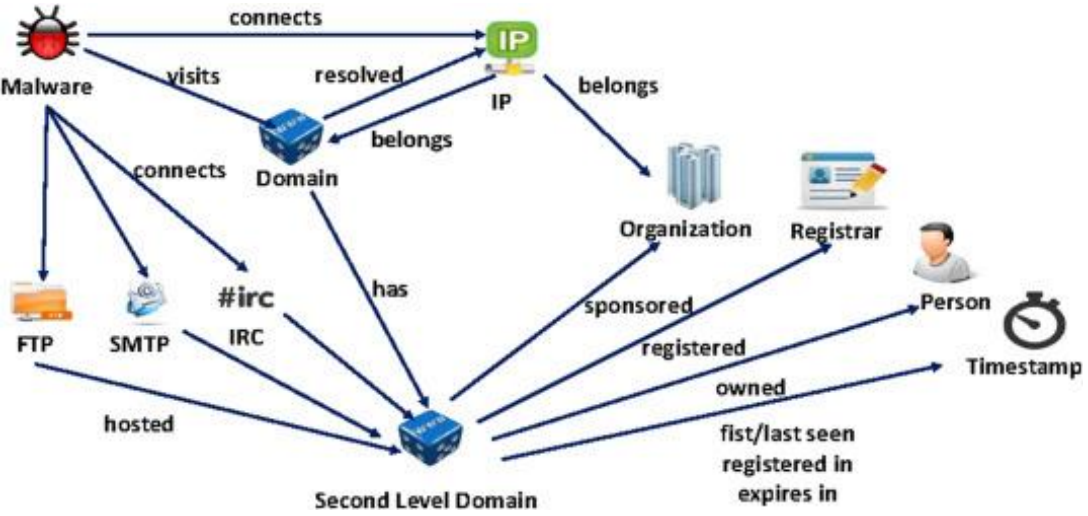
Software

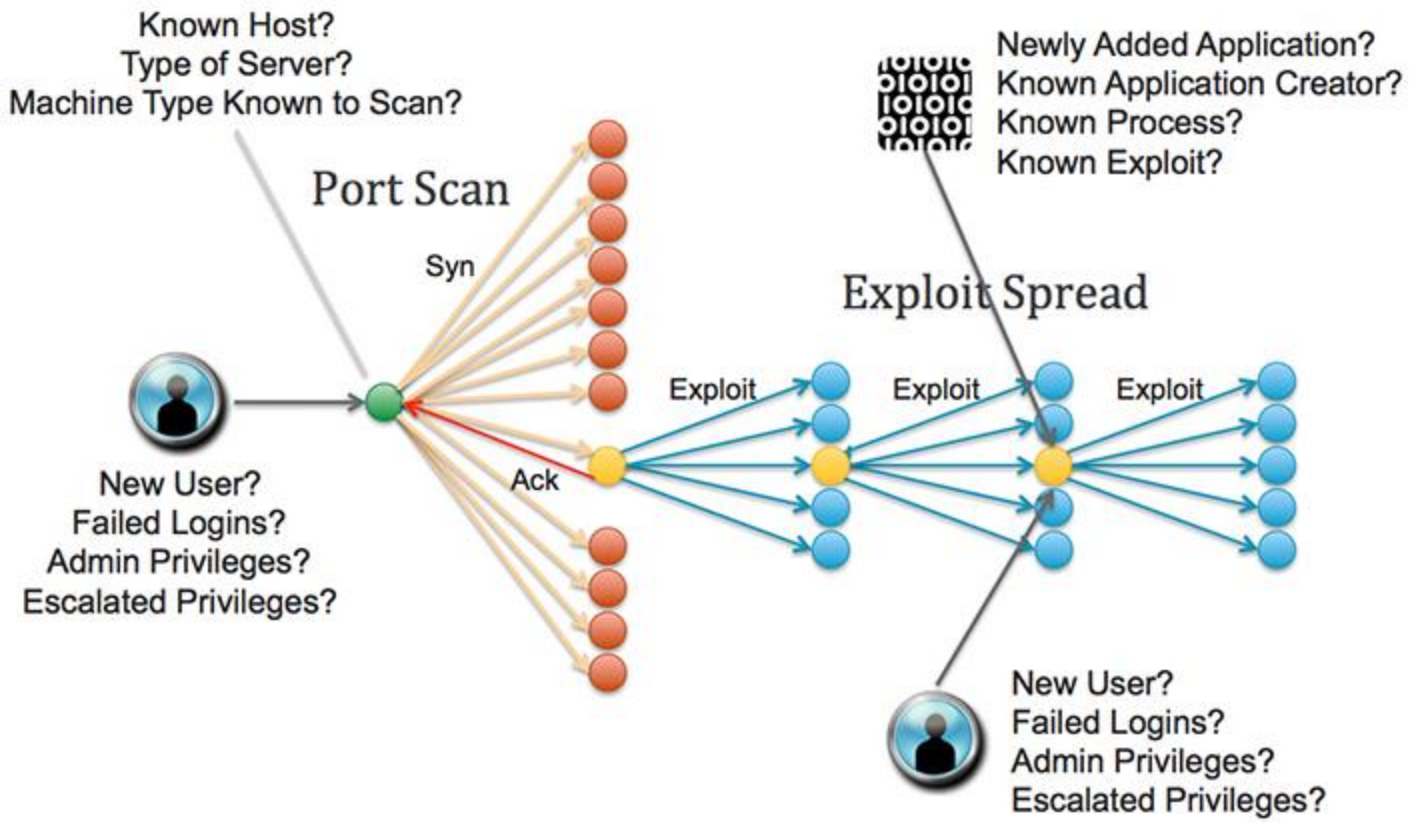


Social

[Source]

Vulnerability Assessment
(Graph Theory)





Google A.I

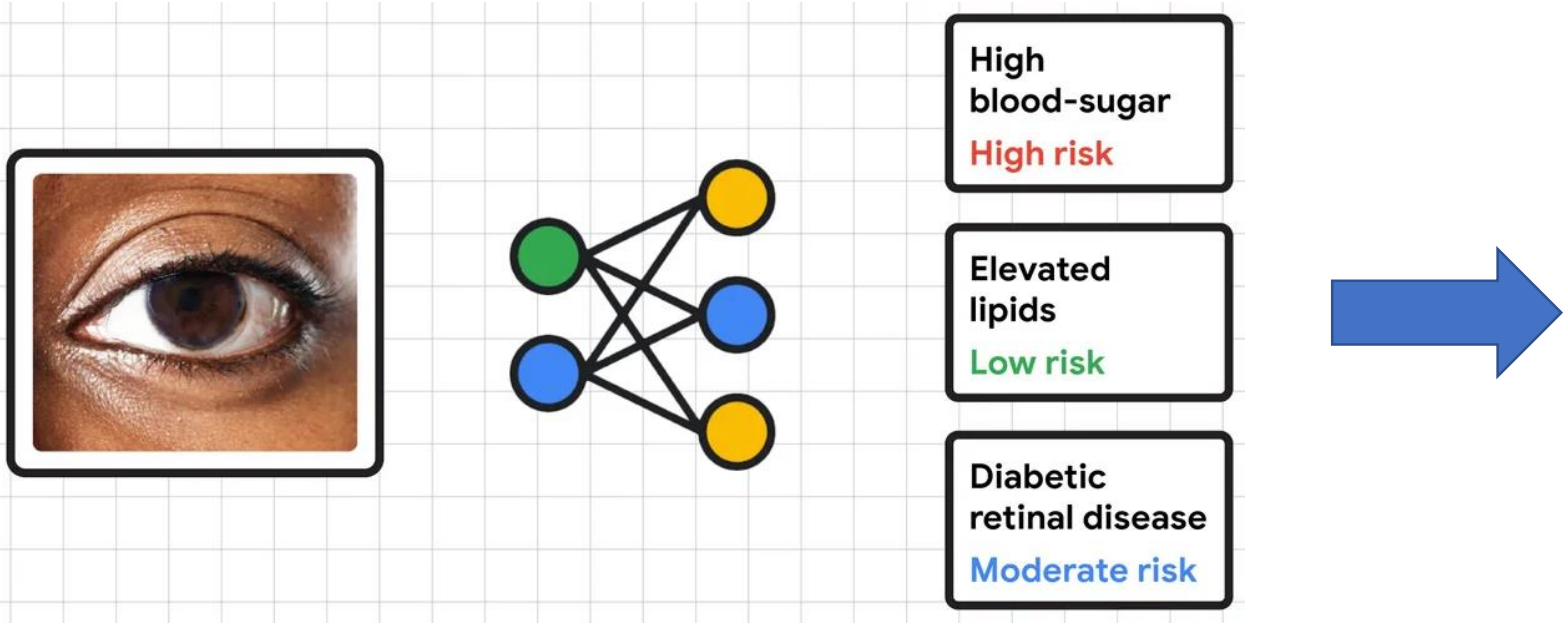


Image Source: Link

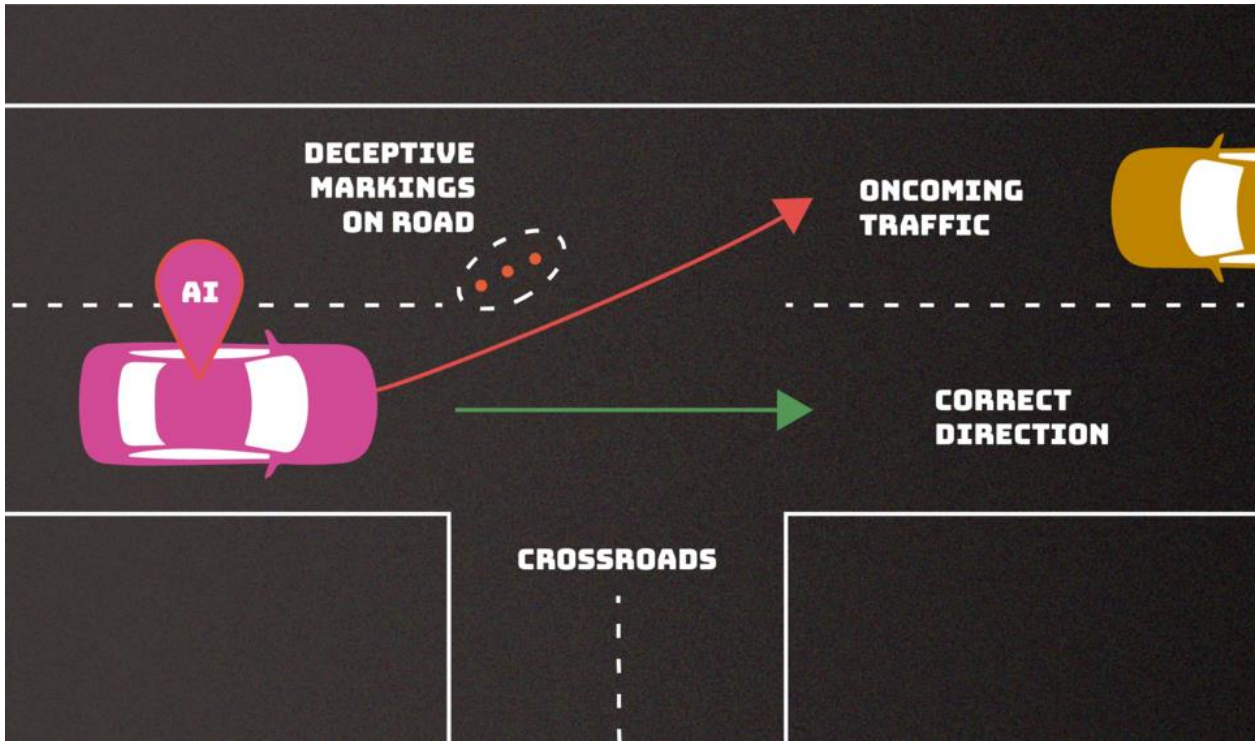
The AI often answered medical questions on par with how a doctor did. Google is expanding use of its health care artificial intelligence, including helping detect diseases, such as cancer, earlier and answering medical questions.
Mar 15, 2023

NIST Identifies Types of Cyberattacks That Manipulate Behavior of AI Systems (Link)

January 04, 2024

- AI systems can malfunction when exposed to untrustworthy data, and attackers are exploiting this issue.
- New guidance documents the types of these attacks, along with mitigation approaches.
- No foolproof method exists as yet for protecting AI from misdirection, and AI developers and users should be wary of any who claim otherwise.

An AI system can malfunction if an adversary finds a way to confuse its decision making.
In this example, errant markings on the road mislead a driverless car, potentially making it veer into oncoming traffic.
This “evasion” attack is one of numerous adversarial tactics described in a new NIST publication intended to help outline the types of attacks we might expect along with approaches to mitigate them.



Not Part of Exam

Undirected graphs

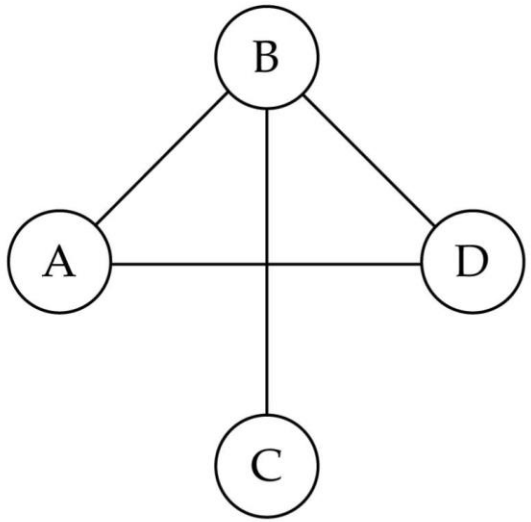
A graph G is defined as follows:

G=(V,E)

V(G): a finite, nonempty set of vertices

E(G): a set of edges (pairs of vertices)

- When the edges in a graph have no direction, the graph is called *undirected*



V(Graph1) = { A, B, C, D }
E(Graph1) = { (A, B), (A, D), (B, C), (B, D) }

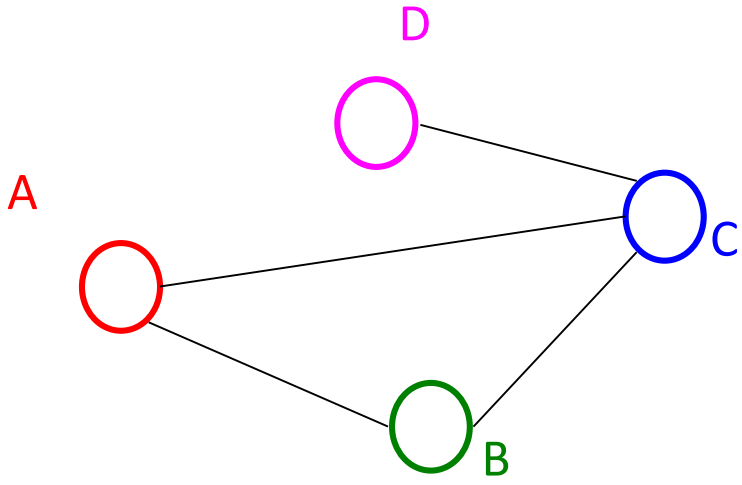
Undirected Graphs

In undirected graphs, edges have no specific direction

- Edges (E) are always "two-way"

Thus, (A, B) ∈ E implies (B, A) ∈ E.

- Only one of these edges needs to be in the set
- The other is implicit, so normalize how you check for it

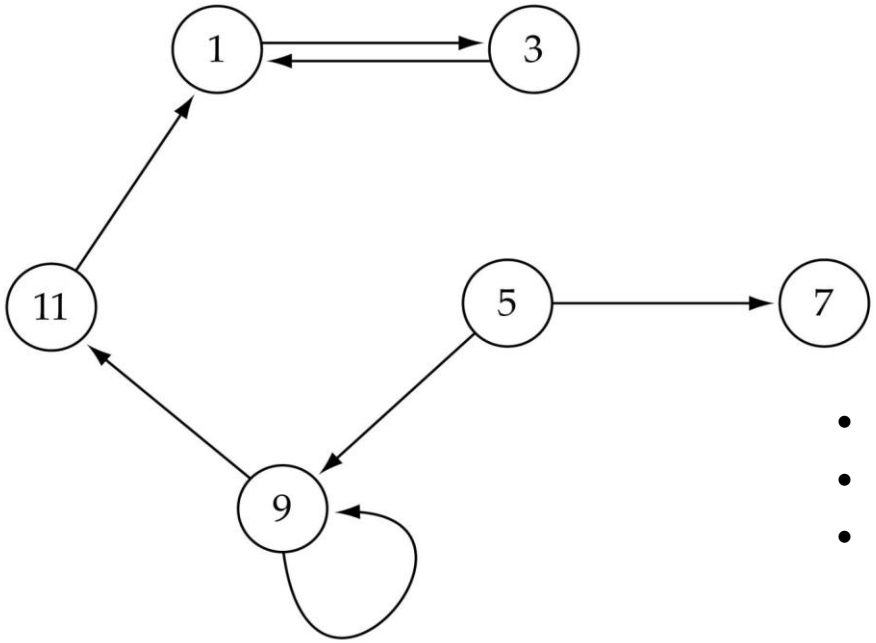


Degree of a vertex: number of edges containing that vertex

- Put another way: the number of adjacent vertices
  - deg(A) = 2 , deg(B) = 2, deg(C) = 3, deg(D) = 1
- Any two nodes connected by an edge or any two edges connected by a node are said to be **adjacent** e.g. A is adjacent to B and C

Directed graphs

- When the edges in a graph have a direction, the graph is called *directed* (or *digraph*)



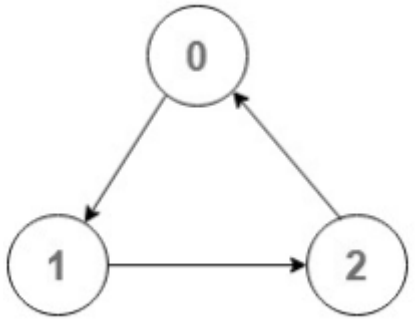
- If (5, 7) ∈ E , then 7 is a neighbor of 5 (i.e., 7 is not adjacent to 5)
- Order matters for directed edges: 5 is not adjacent to 7 unless (7, 5) ∈ E
- A self-edge a.k.a. a loop edge is of the form (9, 9)

V(Graph2) = { 1, 3, 5, 7, 9, 11 }

E(Graph2) = {(1,3) (3,1) (5,9) (9,11) (5,7) (9,9) (11,1)}

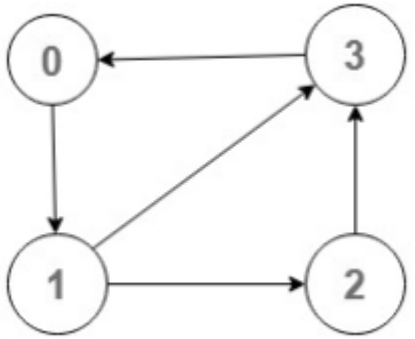
Warning: if the graph is directed, the order of the vertices in each edge is important !!

Directed graphs



Adjacency list :

i. 0 -> 1
ii. 1 -> 2
iii. 2 -> 0

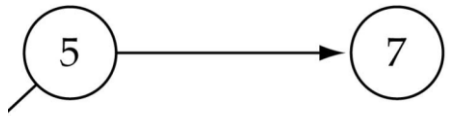


Adjacency list :

I. 0 -> 1
II. 1 -> 2, 3
III. 2 -> 3
IV. 3 -> 0

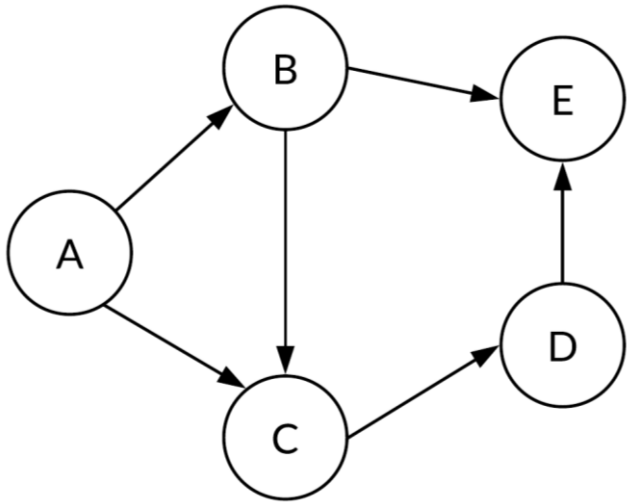
Graph terminology

- Adjacent nodes: Two nodes are adjacent if they are connected by an edge



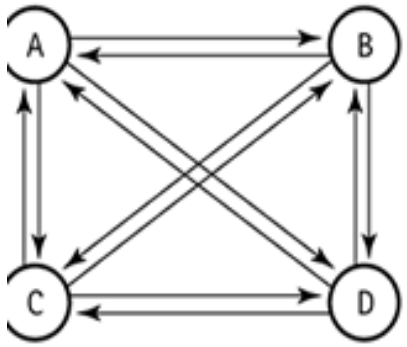
5 is adjacent to 7

- Path: A sequence of vertices that connect two nodes in a graph
- Complete graph: A graph in which every vertex is directly connected to every other vertex

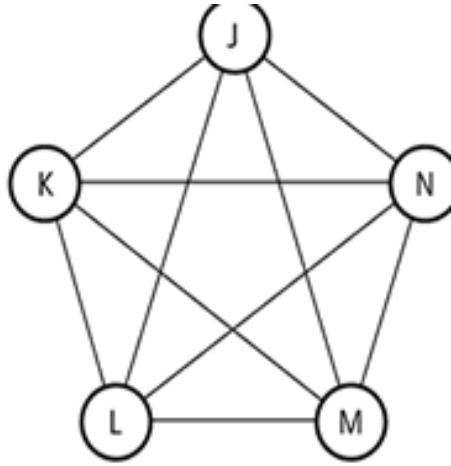


Adjacency list:

```
graph = {
'A': ['B', 'C'],
'B': ['C', 'E'],
'C': ['D'],
'D': ['E'],
}
```



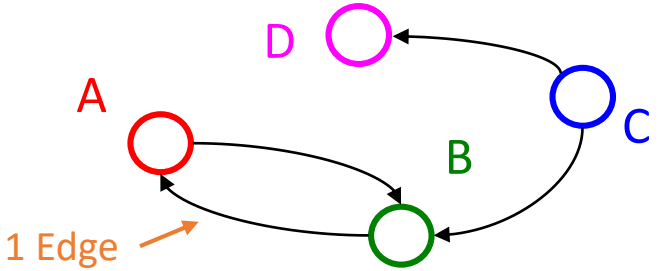
a) Complete directed graph.



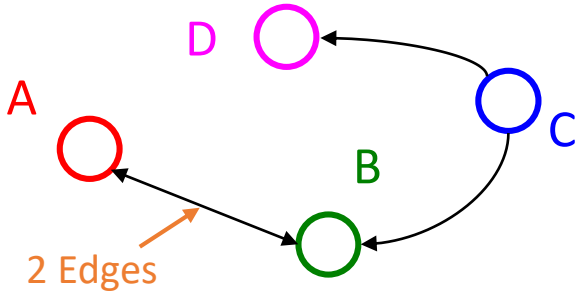
(b) Complete undirected graph.

Directed Graphs

In directed graphs (or digraphs), edges have direction



or



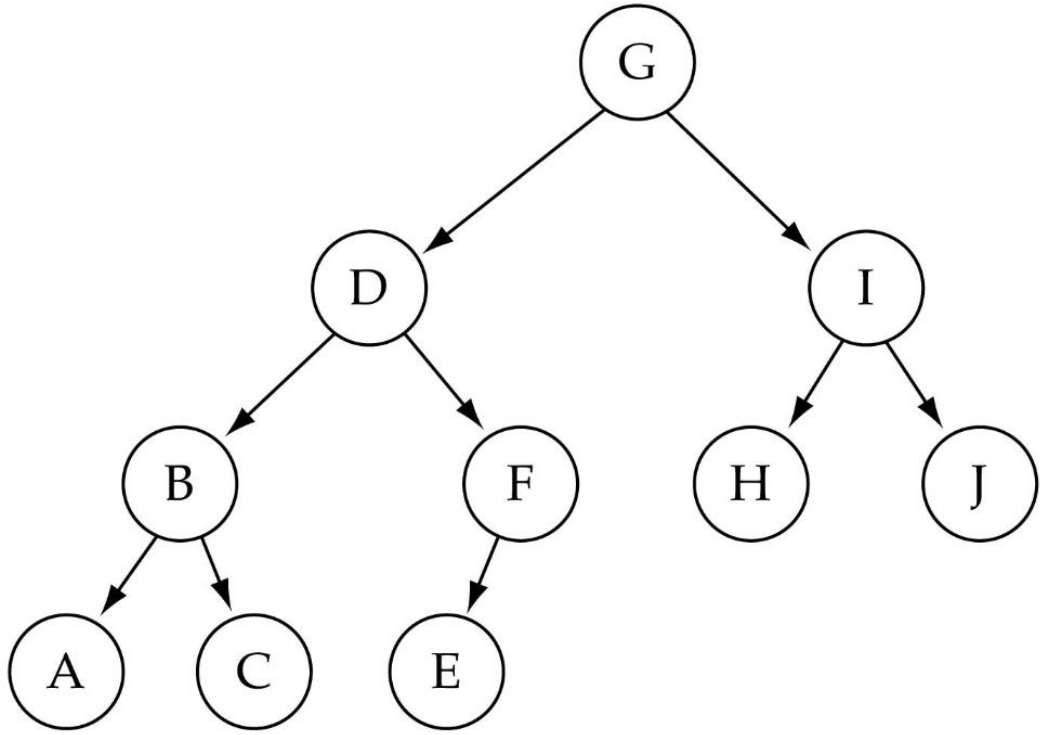
Thus, $(C, D) \in E$ does not imply $(D, C) \in E$.

Let $(C, D) \in E$ mean $C \rightarrow D$

- Call C the source and D the destination
- **In-Degree of a vertex**: number of in-bound edges (edges where the vertex is the destination)
- **Out-Degree of a vertex**: number of out-bound edges (edges where the vertex is the source)
  - Vertex A has 1 in-degree and outdegree

Trees as Graphs

- Trees are special cases of graphs!!



Warning: if the graph is directed, the order of the vertices in each edge is important !!

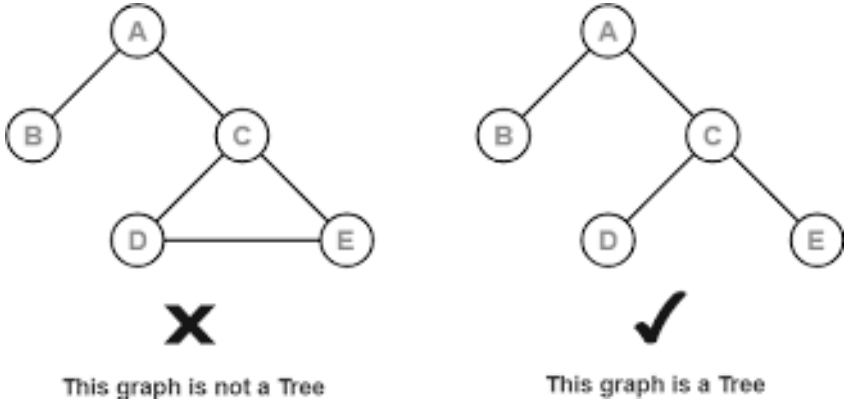
V(Graph3) = { A, B, C, D, E, F, G, H, I, J }
E(Graph3) = { (G, D), (G, J), (D, B), (D, F) (I, H), (I, J), (B, A), (B, C), (F, E) }

Trees VS Graphs

When talking about graphs, we say a tree is a graph that is:

1. Undirected
2. A-cyclic
3. Connected

All trees are graphs, but NOT all graphs are trees



This graph is not a Tree

This graph is a Tree

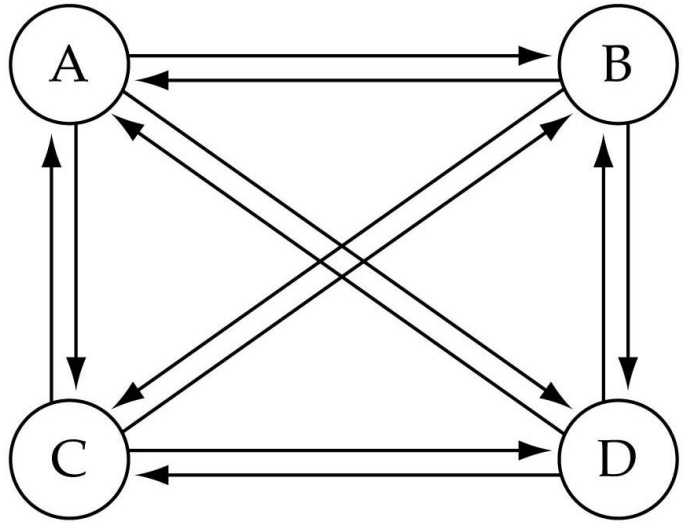
Comparison	Graph	Tree
Relationship of Node	No Root node No Parent, Child relation	Root node Parent, Child relation
Path	One or more path between two nodes	One path among two nodes
Loop	May have Loop among nodes	Loop do not exist among nodes
Traversal	BFS, DFS	Pre-Order, In-Order, Post-Order
Model type	Network	Hierarchical

Graph terminology

- What is the number of edges in a **complete directed graph** with N vertices?

*Number of Edges = N * (N-1)*

- *N = 4 (i.e. A, B, C, D)*
- *4*(4-1) = 12*
- *Edges = 12*



(a) Complete directed graph.

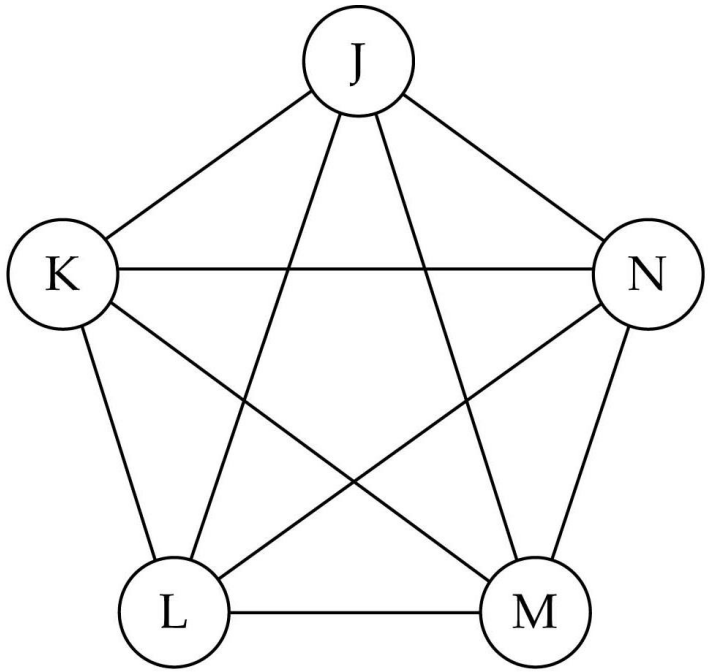
Graph terminology

- What is the number of edges in a **complete undirected graph** with N vertices?

*Number of Edges = N * (N-1) / 2*

- *N = 5 (i.e. J, K, L, M, N)*
- *N * (N-1) / 2 => 5*(5-1) / 2*
- *Edges = 10*

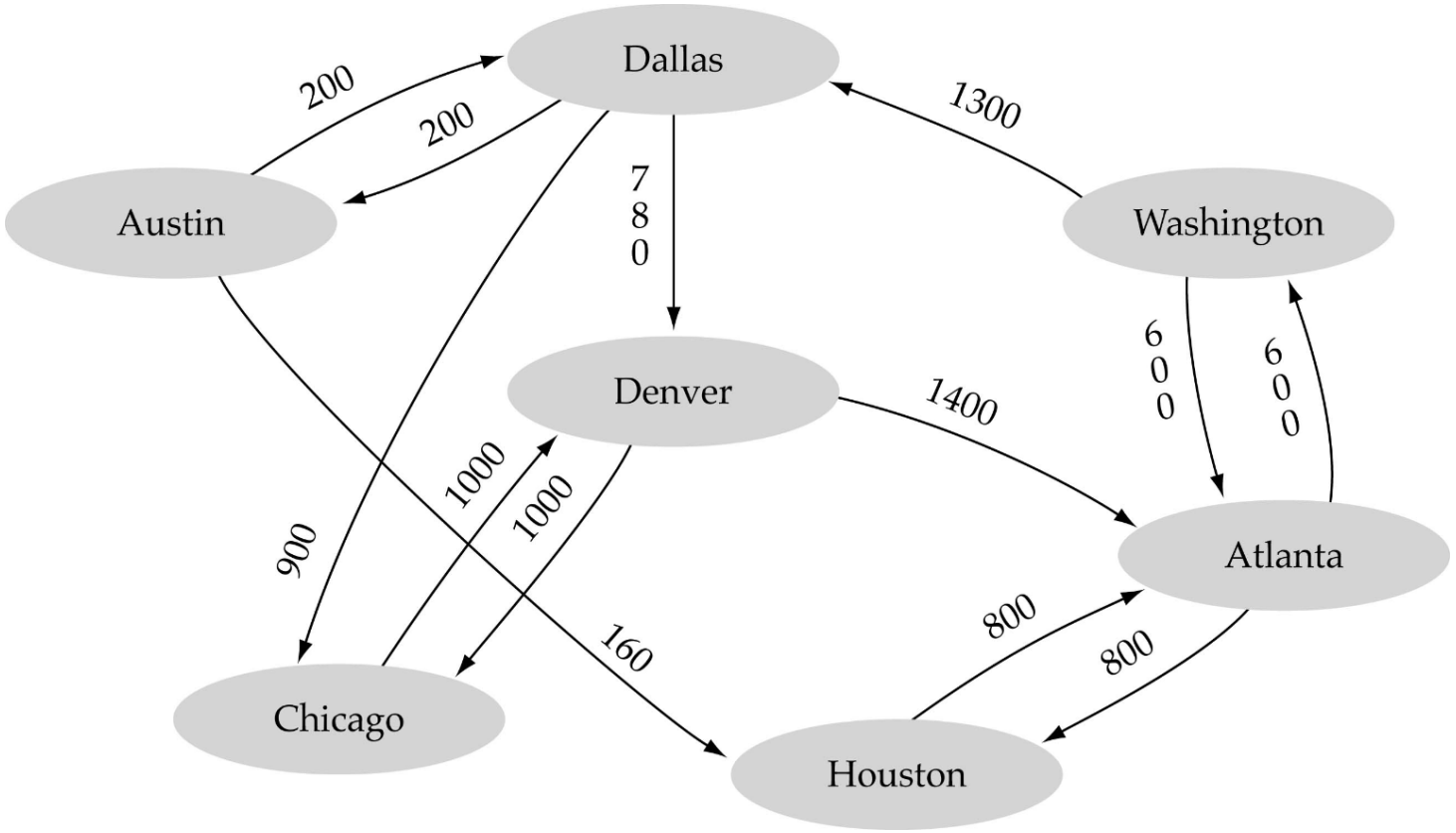
E = {(J, K) (J, N) (J, N) (J, M) (K, N) (K, M) (K, L) (L, N) (L, M) (M, N)}

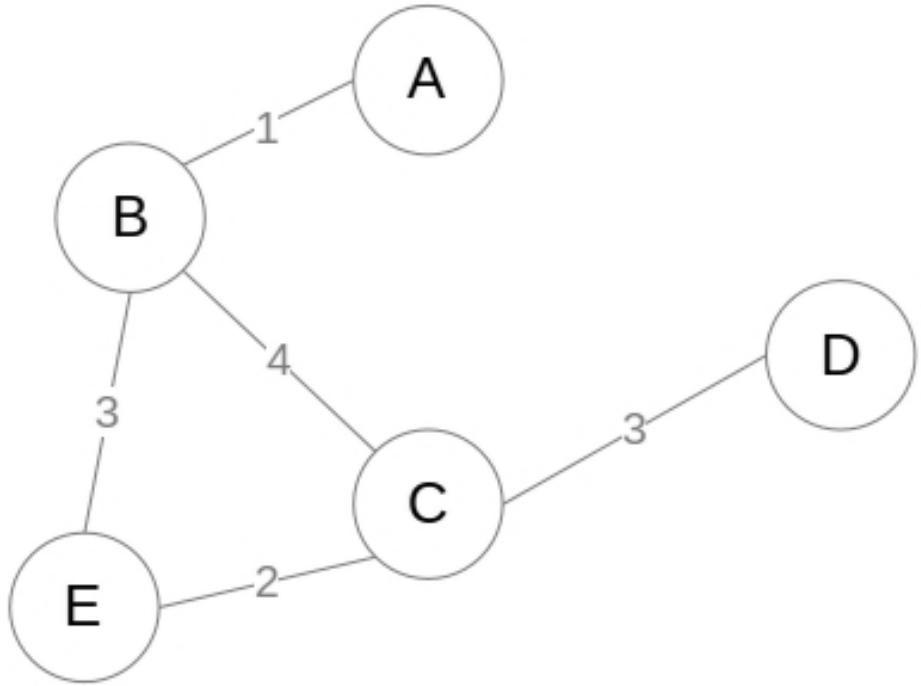


(b) Complete undirected graph.

Graph terminology

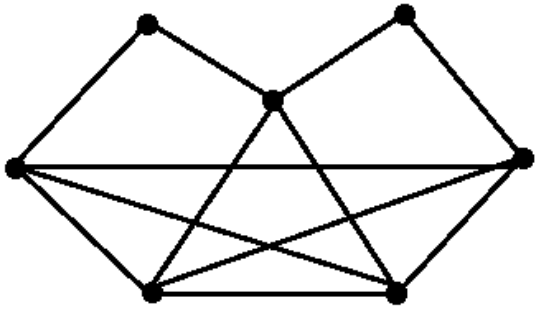
- Weighted graph: Each edge carries a value



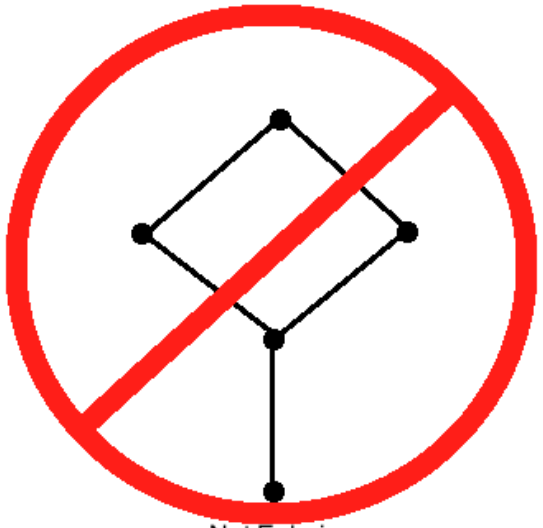


Handshaking Lemma

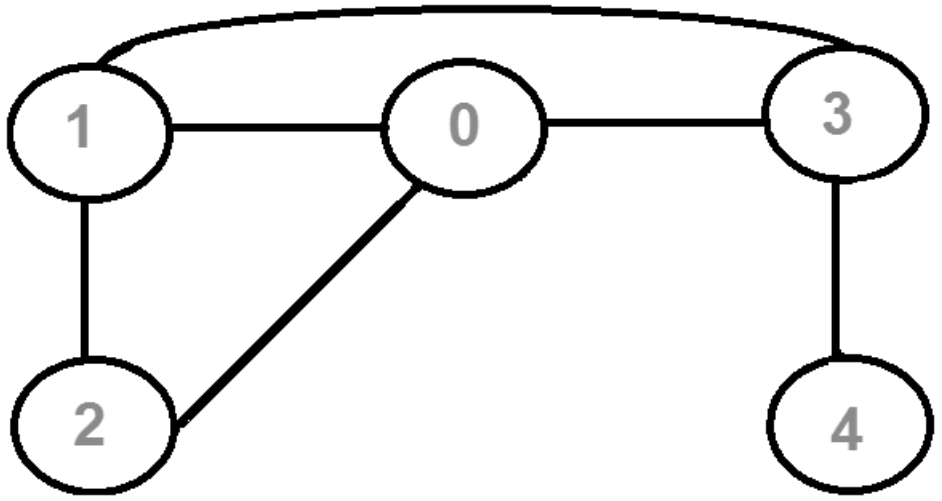
- The Handshaking Lemma is a useful tool in solving graph-related problems.
  - For example, it can be used to determine whether a graph has an Eulerian path or cycle.
  - An **Eulerian path** is a path that visits every edge of a graph exactly once, while an **Eulerian cycle** is a cycle that visits every vertex of a graph exactly once.
  - If a graph has an Eulerian path or cycle, then the sum of the **degrees of all vertices must be even.**



Eulerian



Not Eulerian



The graph is not Eulerian. Note that there are four vertices with odd degree (0, 1, 3 and 4)

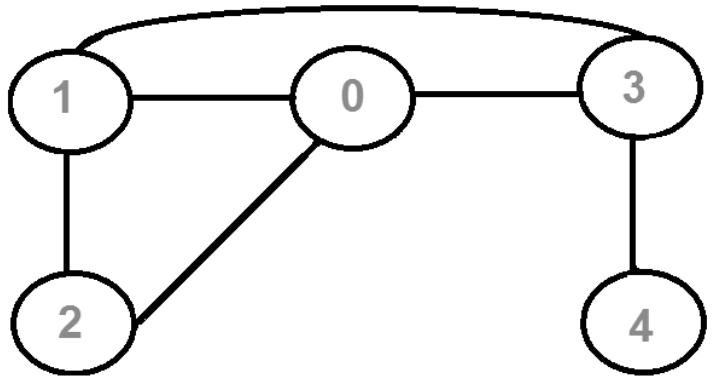
Handshaking Lemma

- Handshaking lemma is about an undirected graph.
- In every finite undirected graph, an even number of vertices will always have an odd degree.
- The handshaking lemma is a consequence of the degree sum formula (also sometimes called the handshaking lemma)
  - $\sum_{u \in v} \deg(v) = 2|E|$
    - *Edges = N * (N-1) / 2*
    - *N = 5*
    - *Edges=5*(5-1)/2*
    - *Edges = 10*

$$\sum_{u \in v} deg(v) = 2|E| = 20$$

- *Vertices = 5 (i.e. 0,1,2,3,4)*
- *Edges = 6*
  - *0 have 3 edge connections*
  - *1 have 3 edge connections*
  - *2 have 2 edge connections*
  - *3 have 3 edge connections*
  - *4 have 1 edge connections*
- $\sum_{u \in v} deg(v) = 20$ *(even number)*

Not a Complete undirected graph



The graph is not Eulerian. Note that there are four vertices with odd degree (0, 1, 3 and 4)

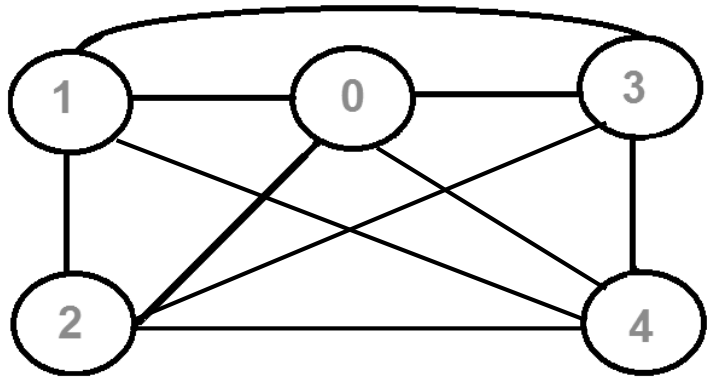
Application: The handshaking lemma is one of the important branches of graph theory. The content is widely applied in topology and computer science.

Handshaking Lemma

- Handshaking lemma is about an undirected graph.
- In every finite undirected graph, an even number of vertices will always have an odd degree.
- The handshaking lemma is a consequence of the degree sum formula (also sometimes called the handshaking lemma)
  - $\sum_{u \in v} \deg(v) = 2|E|$
    - *Edges = N * (N-1) / 2*
    - *N = 5*
    - *Edges=5*(5-1)/2*
    - *Edges = 10*

$$\sum_{u \in v} deg(v) = 2|E| = 20$$

- *Vertices = 5 (i.e. 0,1,2,3,4)*
- *Edges = 6*
  - *0 have 4 edge connections*
  - *1 have 4 edge connections*
  - *2 have 4 edge connections*
  - *3 have 4 edge connections*
  - *4 have 4 edge connections*
- $\sum_{u \in v} deg(v) = 20$ *(even number)*



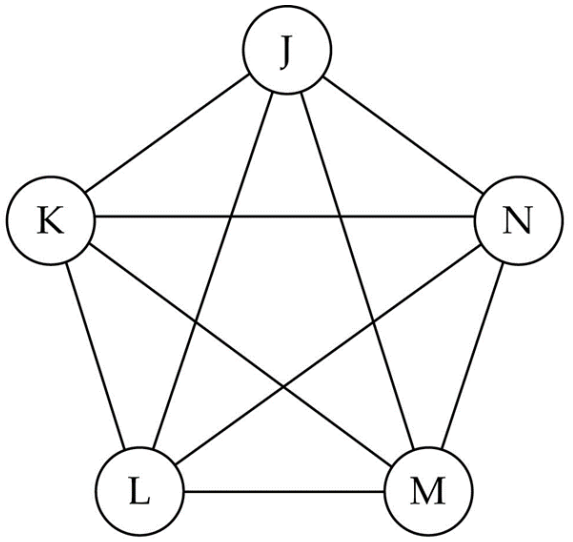
Application: The handshaking lemma is one of the important branches of graph theory. The content is widely applied in topology and computer science.

Handshaking Lemma

- Handshaking lemma is about an undirected graph.
- In every finite undirected graph, an even number of vertices will always have an odd degree.
- The handshaking lemma is a consequence of the degree sum formula (also sometimes called the handshaking lemma)
  - $\sum_{u \in v} \deg(v) = 2|E|$
    - *Edges = N * (N-1) / 2*
    - *N = 5*
    - *Edges=5*(5-1)/2*
    - *Edges = 10*

$$\sum_{u \in v} deg(v) = 2|E| = 20$$

- *Vertices = 5 (i.e. J, K, L, M, N)*
- *Edges = 10*
- *Each Vertices have 4 degree*
  - *i.e.*
  - *K have 4 edge connections*
  - *J have 4 edge connections*
  - *L have 4 edge connections*
  - *M have 4 edge connections*
  - *N have 4 edge connections*
- $\sum_{u \in v} deg(v) = 20$ *(even number)*

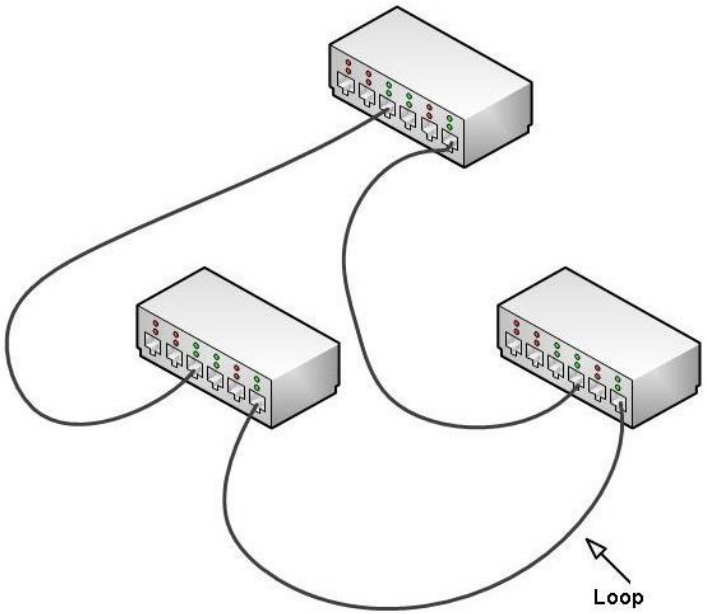


(b) Complete undirected graph.

Application: The handshaking lemma is one of the important branches of graph theory. The content is widely applied in topology and computer science.

Handshaking Lemma -Example

- In a network, a loop provides a never-ending path for network traffic to follow.
- In Ethernet, frames never die; there's no time-to-live function attached to a traditional Ethernet frame.
- Therefore, if a loop exists, frames with no explicit destination (like broadcast frames) circle around the network forever

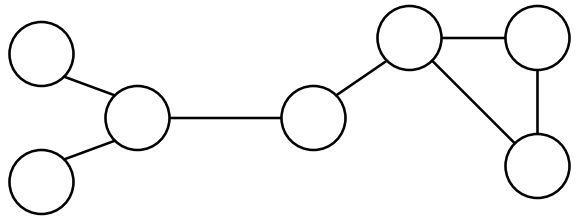


- The sender keeps a record of each packet it sends and maintains a timer from when the packet was sent.
- The sender re-transmits a packet if the timer expires before receiving the acknowledgement.
- The timer is needed in case a packet gets lost or corrupted.

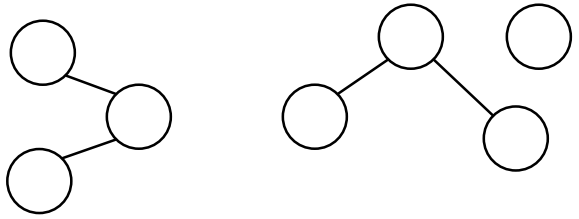
Damage	Data Network down Voice Network down Local ERP, MIS sites relying on MPLS will be effected
Fix	Find cable responsible for loop Spanning Tree Protocol (STP)

Undirected Graph Connectivity

- A graph is said to be connected if every pair of vertices in the graph is connected. This means that there is a path between every pair of vertices. An undirected graph that is not connected is called disconnected.



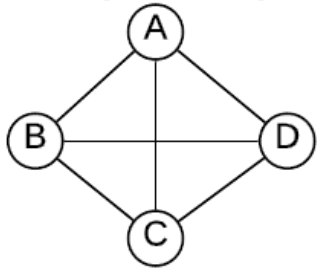
Connected graph



Disconnected graph

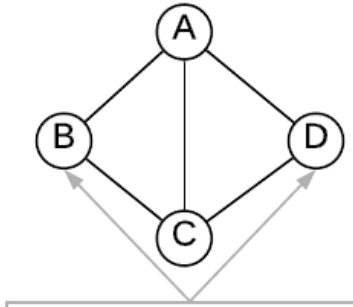
- An undirected graph is complete, or fully connected, if for all pairs of vertices there exists an *edge* from u to v
- A complete (undirected) graph is known to have exactly V(V-1)/2 edges where V is the number of vertices. So, you can simply check that you have exactly V(V-1)/2 edges.

Complete Graph



All vertices connect to all other vertices

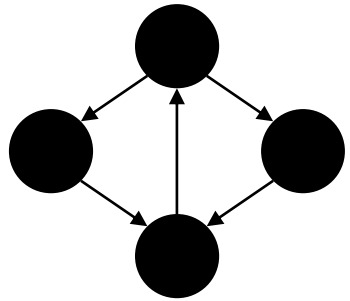
Not a Complete Graph



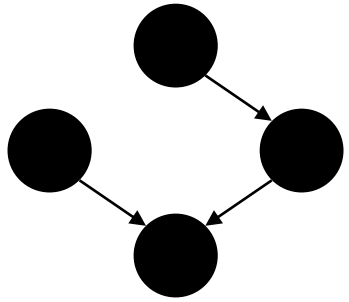
No edge connect B and D

Directed Graph Connectivity

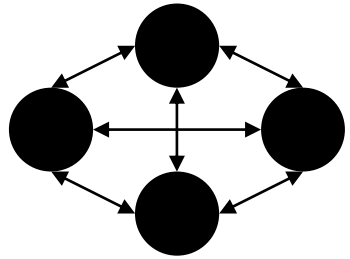
A directed graph is strongly connected if there is a path from every vertex to every other vertex



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



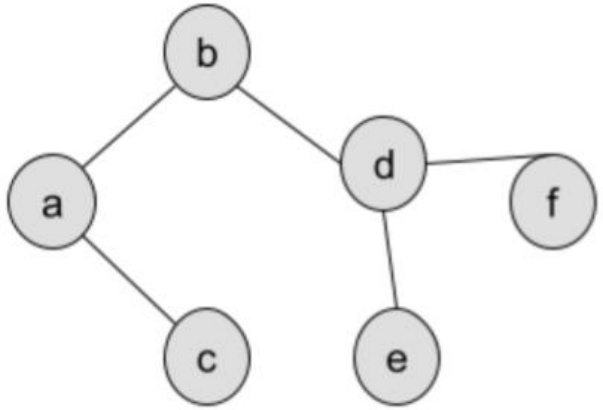
A direct graph is complete or fully connected, if for all pairs of vertices, there exists an *edge* from u to v



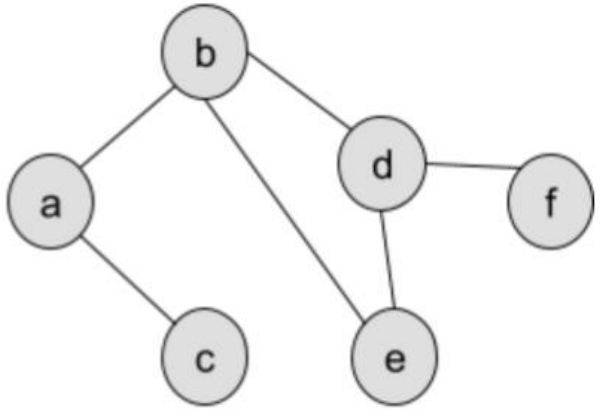
Based on the Presence of Cycles

- **Cyclic Graphs:** If a graph contains one or more cycles, it is deemed a **cyclic graph**. In other words, a cyclic graph contains at least one node, which has a path that connects it back to itself.
- If the cyclic graph contains only one cycle, it is called a **unicyclic graph**.

- **Acyclic Graphs:** An acyclic graph, on the other hand, has no cycles. Since there are zero cycles, no node has a path to be traced back to itself.
- An undirected graph that's acyclic is a forest graph.



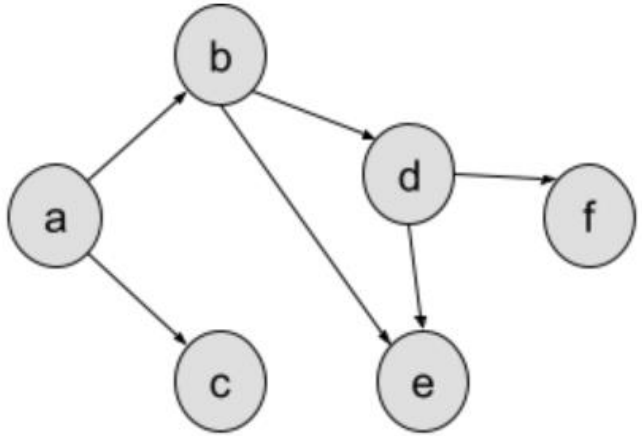
Acyclic (Undirected) Graph



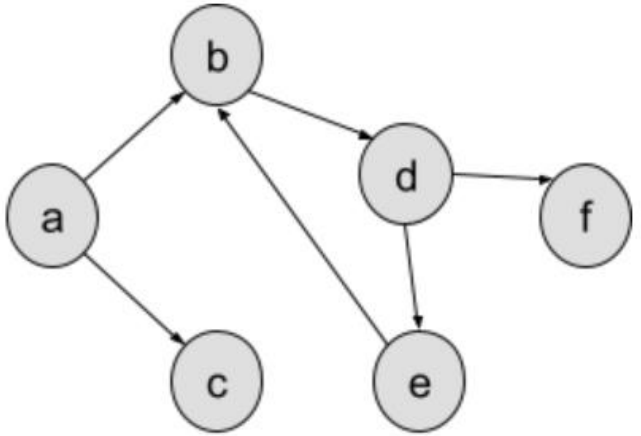
Cyclic (Undirected) Graph

Based on the Presence of Cycles

- A directed graph with no directed cycles is called a **directed acyclic graph**.
- By **directed cycles**, we mean that following the directions in the graph will never form a closed loop. In the figure, the first directed graph is acyclic, and the second directed graph is cyclic.
- **Note** that just the direction of one edge has been reversed to create the cyclic graph out of the acyclic graph. Just that one change leads to a directed cycle being formed in the graph.
- To connect the concept of a tree and graphs, if a connected undirected graph is also acyclic, it is a tree graph.



Acyclic (Directed) Graph

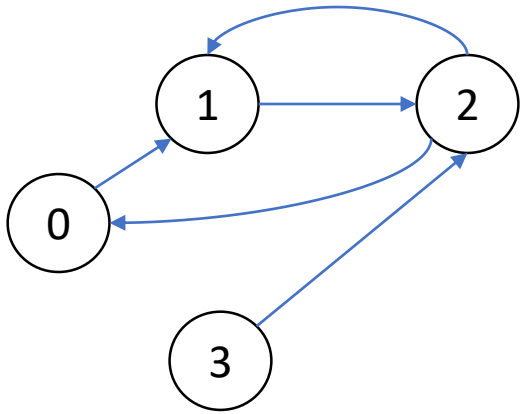


Cyclic (Directed) Graph

Directed Graph

```
76      // input array containing edges of the graph (as per the above diagram)
77      // (x, y) pair in the array represents an edge from x to y
78      struct Edge edges[] =
79      {
80          {0, 1}, {1, 2}, {2, 0}, {2, 1}, {3, 2}
81      };
```

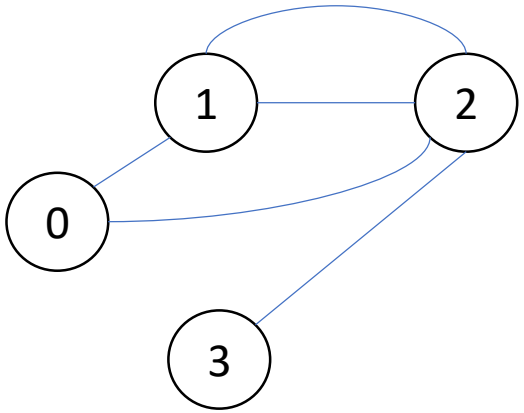
```
(0 -> 1)
(1 -> 2)
(2 -> 1)          (2 -> 0)
(3 -> 2)
```



Un-Directed Graph

```
90      // input array containing edges of the graph (as per the above diagram)
91      // (x, y) pair in the array represents an edge from x to y
92      struct Edge edges[] =
93      {
94          {0, 1}, {1, 2}, {2, 0}, {2, 1}, {3, 2}
95      };
```

```
(0 -> 2)        (0 -> 1)
(1 -> 2)        (1 -> 2)        (1 -> 0)
(2 -> 3)        (2 -> 1)        (2 -> 0)        (2 -> 1)
(3 -> 2)
```



Weighted Directed Graph

```
80      // input array containing edges of the graph (as per the above diagram)
81      // (x, y, w) tuple represents an edge from x to y having weight `w`
82      struct Edge edges[] =
83      {
84          {0, 1, 6}, {1, 2, 7}, {2, 0, 5}, {2, 1, 4}, {3, 2, 10}
85      };
```

```
0 -> 1 (6)
1 -> 2 (7)
2 -> 1 (4)        2 -> 0 (5)
3 -> 2 (10)
```