Graphs
An Introduction
A familiar place...
Graphs (and Networks)

Graphs are made up of

- **nodes** (or vertices) and
- **connections** between them (or edges)

Vertices typically have a name or label, e.g., 3 or Duo

Edges are referenced by the pair of vertices they connect, e.g., (3,2) or (Duo, Cat)

When nodes represent entities, connections represent relationships.

In this case graphs are also called Networks.
Undirected Graphs

Edges are bidirectional. Like two-way streets
(Undirected) Graph Definition

- Our first non-linear data structure!

- An **undirected graph** $G$ consists of two sets $G = \{V, E\}$
  - A set of $V$ **vertices**, or nodes
  - A set of $E$ **edges**, relationships between nodes

- A **subgraph** $G'$ consists of a subset of the vertices and edges of $G$

- **Adjacent** are two vertices connected by an edge

- An edge that connects a vertex to itself is called a *self-loop* or *sling*. We will avoid them.

- $V = \{\}$

- $E = \{(\ ,\ ),(\ ,\ ),(\ ,\ ),(\ ,\ ),(\ ,\ )\}$
Paths and Cycles

- **A path** between two vertices is a sequence of edges that begins at the first vertex and ends at the other vertex. (The edges in the path could be required to be distinct or not.)

- **A simple path**
  - is a path that passes through a vertex at most once

- **A cycle**
  - is a path that begins and ends at the same vertex

- **A simple cycle**
  - A cycle that does not pass through a vertex more than once

- A graph that has no cycle is called **Acyclic**
A **connected** graph is a graph that has a path between each pair of vertices.

A **disconnected** graph is a graph that has at least one pair of vertices without a path between them.

A **connected component** is a connected subgraph of the graph.
A complete graph is a graph that has an edge between every pair of distinct vertices.

How many edges does a complete graph with \( n \) vertices have?
Tree
a connected Graph without cycles

How many simple paths are there between two *tree* nodes?

How many *edges* does a tree with *n* nodes have?
Directed Graphs (aka: DiGraphs)

Edges ("arcs") are uni-directional. Like one-way streets
Directed Graphs and DAGs

- Directed graph $G = \{V, A\}$
- Arcs (or links) are directed edges between vertices
- A vertex $y$ is adjacent to vertex $x$ iff (if and only if) there is an arc (directed edge) from $x$ to $y$

- Directed path is a sequence of arcs between two vertices
- Directed cycle is a directed path from a vertex to itself

- Directed Acyclic Graph (DAG) is a digraph without directed cycles

- You could turn a digraph into a DAG by removing some arcs to break cycles
- How few arcs can you remove to turn this digraph into a DAG?

\[ V = \{ \} \]
\[ A = \{ (\ , \), (\ , \), (\ , \), (\ , \), (\ , \) \} \]
Visualizing Graphs with yEd

- yEd: A simple graph visualization
- Download it: https://www.yworks.com/products/yed
- You can create any graph by clicking (for vertices) and clicking-and-dragging (for edges)
- Lots of graph formats supported. Use .tgf for simplicity
- Once you upload a file, choose Layout > Circular to see it laid out nicely. Explore more layouts for fun!
A strongly connected graph
- A graph that has a directed path between any pair of vertices

A strongly connected component of a graph
- A maximally strongly connected subgraph

How many strongly connected components do you see in this digraph?
Implementing Graphs

An **undirected** graph $G$ consists of two sets $G = \{V, E\}$, a set $V$ of vertices and a set $E$ of edges.

A **digraph** $G$ consists of two sets $G = \{V, A\}$, a set $V$ of vertices and a set $A$ of arcs (directed edges).
public interface DiGraph<T> {

public int getNumVertices()   // Returns number of vertices
public int getNumArcs()       // Returns the number of arcs

public void addVertex(T v)    // Insert a vertex in a graph
public void removeVertex(T v) // Delete a vertex along with any arcs between v and other vertices

public void addArc(T v1, T v2)  // Adds an arc from v1->v2
public void removeArc(T v1, T v2) // Deletes the arc between two given vertices in a graph

public boolean isArc(T v1, T v2) // Returns true iff an arc exists between vertices v1 and v2

public boolean isEmpty()     // Returns true iff a graph is empty
public String toString()     // Returns a String representation

public void saveToTGF(String fName) // Saves graph fName.tgf
Implementing (Di)Graphs with Adjacency Matrix

NOTE: If a **digraph** has between every pair of vertices either *both* arcs or *none*, then it can be considered **undirected**

<table>
<thead>
<tr>
<th>Arcs</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
What do you need to add to turn this digraph into an undirected graph?
What property does the matrix of an undirected graph have?
public class AdjMatDiGraph<T> implements DiGraph<T> {
    private final int DEFAULT_CAPACITY = 10;

    private boolean[][] arcs; // adjacency matrix of arcs
    private T[] vertices; // array of vertices (could be a Vector)
    private int n; // number of vertices in the graph

    public AdjMatDiGraph(){  // constructor
        this.n = 0;
        this.arcs = new boolean[DEFAULT_CAPACITY][DEFAULT_CAPACITY];
        this.vertices = (T[])(new Object[DEFAULT_CAPACITY]);
    }
    public boolean isEmpty(){... // returns true if a graph is empty
}
    public int getNumVertices(){... // returns the number of vertices
}
    public int getNumArcs(){... //returns the number of arcs
        //count them!
    }
    etc...
Implementing (Di)Graphs with Adjacency Lists

NOTE: If a digraph has between every pair of vertices either both arcs or none, then it can be considered undirected.
Adjacency Lists

- An adjacency list for a DiGraph with
  - \( n \) vertices numbered 0, 1, …, \( n - 1 \)
  - \( arcs \): array (or Vector) of \( n \) linked lists
    - The \( i^{th} \) linked list has a list entry for vertex \( j \)
      iff the graph contains an arc from vertex \( i \) to vertex \( j \)

```
  a \rightarrow b \rightarrow c \rightarrow d
```

```
  Arcs
  a   c   d
  b   c   /
  c   a   b
  d   c   /
```
Undirected & Directed Graph Representation

- We can use either `AdjMatDiGraph` or `AdjListDiGraph` to represent both undirected and directed graphs.
- In an undirected graph every edge v–w appears as two arcs v -> w and w -> v in the adjacency lists.

What do you need to add to turn this digraph into an undirected graph?
```java
public class AdjListDiGraph<T> implements DiGraph<T> {

    private Vector<T> vertices;
    private Vector<LinkedList<T>> arcs; // adjacency lists of arcs

    public AdjListDiGraph(){ // constructor
        this.arcs = new Vector<LinkedList<T>>() ;
        this.vertices = new Vector<T> ();
    }

    public boolean isEmpty(){... // returns true if a graph is empty
    }

    public int getNumVertices(){... // returns the number of vertices
    }

    public int getNumArcs(){... //returns the number of arcs
    //count them!
    }

    etc...
```
Practicing with the Wellesley Campus
WC Campus DiGraph
WC Campus DAG

Diagram of a Directed Acyclic Graph (DAG) with nodes labeled from 0 to 8.