A graph is a pair \((V, E)\), where
- \(V\) is a set of nodes, called vertices
- \(E\) is a collection of pairs of vertices, called edges

Example:
- A vertex represents an airport and stores the airport code
- An edge represents a flight route between two airports

Directed edge:
- ordered pair of vertices \((u, v)\)
  - first vertex \(u\) is the origin
  - second vertex \(v\) is the destination
  - e.g., a flight

Undirected edge:
- unordered pair of vertices \((u, v)\)
  - e.g., a street

Directed graph: all edges are directed

Weighted edge: has a real number associated to it
  - e.g., distance between cities
  - e.g., bandwidth between internet routers

Weighted graph: all edges have weights

Labeled graphs: vertices have identifiers

Unlabeled graph: vertices have no identifiers

Applications

Terminology

Endpoints of an edge
- \(U\) and \(V\) are the endpoints of a

Edges incident on a vertex
- \(a, b, c, d\) and \(e\) are incident on \(V\)

Adjacent vertices
- Lines by an edge
- \(U\) and \(V\) are adjacent

Degree of a vertex
- Number of incident edges
  - \(X\) has degree 5

Parallel edges
- \(h\) and \(i\) are parallel edges

Self-loop
- \(j\) is a self-loop
**Terminology (cont.)**

- **Path**
  - sequence of adjacent vertices
  - Simple path
  - path such that all its vertices are distinct
- **Examples**
  - $P_1 = (U, X, Z)$ is a simple path
  - $P_2 = (U, W, X, Y, W, V)$ is a path that is not simple
- **Graph is connected if**
  - For all pair of vertices $u$ and $v$, there is a path between $u$ and $v$

**Example**

- $|V| = 4$
- $|E| = 6$
- $\deg(v) = 3$

**Cycle**

- path that starts and ends at the same vertex
- **Simple cycle**
  - cycle where each vertex is distinct
- **Examples**
  - $C_1 = (V, X, Y, W, U, V)$ is a simple cycle
  - $C_2 = (U, W, X, Y, W, V, V)$ is a cycle that is not simple
  - A tree is a connected acyclic graph

**Properties**

**Property 1**

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

**Proof?**

**Property 2**

In an undirected graph with no self-loops and no multiple edges

$$|E| \leq \frac{|V|(|V| - 1)}{2}$$

**Proof?**

**Data structure for graphs - Adjacency lists**

- Graph can be stored as
  - A dictionary of pairs (key, info) where
    - key = vertex identifier
    - info contains a list (called adj) of adjacent vertices
- Example: if the dictionary is implemented as a linked-list,

**Adjacency lists - Operations**

- **addVertex(key k):**
  - vertices.insert(k, emptyList)
- **addEdge(key k, key l):**
  - vertices.find(k).adj.insert(l)
  - vertices.find(l).adj.insert(k)
- **areAdjacent(key k, key l):**
  - return vertices.find(k).adj.find(l)

**Data structure for graphs - Adjacency matrix**

- Define some order on the vertices, for example:
  - DFW, LAX, LGA, ORD, SFO
- Graph with $n$ vertices is stored as
  - $n \times n$ array $M$ of boolean, where
    - $M[i][j] = 1$ if there is an edge between $i$-th and $j$-th vertices
    - $M[i][j] = 0$ otherwise

**Example**

<table>
<thead>
<tr>
<th></th>
<th>DFW</th>
<th>LAX</th>
<th>LGA</th>
<th>ORD</th>
<th>SFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFW</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LAX</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LGA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ORD</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SFO</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Adjacency matrix - Operations

- **addEdge(i,j):** \( \text{matrix}[i][j] = 1 \)
- **removeEdge(i,j):** \( \text{matrix}[i][j] = 0 \)
- Not very good for inserting/removing vertices: requires shifting elements of matrix.
- Requires space \( O(n^2) \)

## Lists vs Matrices

- **Adjacency lists are better if:**
  - You frequently need to add/remove vertices
  - The graph has few edges
  - Need to traverse the graph
- **Adjacency matrices are better if:**
  - You frequently need to
    - add/remove edges
  - Check for the presence/absence of an edge between i,j
  - Matrix is small enough to fit in memory