A graph is a pair \((V, E)\), where

- \(V\) is a set of nodes or vertices
- \(E\) is a collection of pairs of vertices \((u, v)\), called edges, links, or arcs

\[
V = \{a, b, c, d, e\}
\]
\[
E = \{(a, b), (a, c), (a, d),
(b, e), (c, d), (c, e), (d, e)\}
\]
Edge Types

- Directed edge
  - ordered pair of vertices \((u,v)\)
  - first vertex \(u\) is the origin
  - second vertex \(v\) is the destination

- Undirected edge
  - unordered pair of vertices \((u,v)\)

- Directed graph or digraph
  - all the edges are directed

- Undirected graph
  - all the edges are undirected

- Mixed graph
  - directed and undirected edges
Applications

- Computer networks
Applications

- Transportation networks
Applications

- Scheduling tasks

A typical student day

- wake up
- CS2210 meditation
- go to class
- play
- think about CS2210
- eat
- CS2210 assignment
- watch TV
- Make cookies for CS2210 TA
- sleep
- dream of CS2210
Terminology

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a
- Edges incident on a vertex
  - a, d, and b are incident on V
- Adjacent vertices
  - U and V are adjacent
- Degree of a vertex
  - X has degree 5
- Parallel edges
  - h and i are parallel edges
- Self-loop
  - j is a self-loop

A graph without parallel edges or self-loops is called a simple graph
Terminology

- **Path**
  - sequence of adjacent vertices

- **Simple path**
  - path such that all its vertices and edges are distinct

- **Examples**
  - $P_1 = (V, X, Z)$ is a simple path
  - $P_2 = (U, W, X, Y, W, V)$ is a path that is not simple
**Terminology**

- **Cycle**
  - circular sequence of adjacent vertices

- **Simple cycle**
  - cycle such that all its vertices are distinct (except first and last)

- **Examples**
  - $C_1=(V,X,Y,W,U,V)$ is a simple cycle
  - $C_2=(U,W,X,Y,W,V,U)$ is a cycle that is not simple
Properties

Notation

- $n$: number of vertices
- $m$: number of edges
- $\text{deg}(v)$: degree of vertex $v$

Property 1

\[ \sum_v \text{deg}(v) = 2m \]

Example

- $n = 4$
- $m = 6$
- $\text{deg}(v) = 3$
- $\sum_v \text{deg}(v) = 12$
Complete Graph or Clique

Each vertex is connected to every other vertex.

$$m = \frac{n(n-1)}{2}$$
A tree is a graph without cycles.

$m = n-1$
Forest

A forest is a set of trees.

\[ m \leq n - 1 \]
Subgraph

A subgraph is a subset of vertices and edges that forms a graph.
Subgraph

A subgraph is a subset of vertices and edges that forms a graph.
Connected Graph

A graph is connected if there is a path from each vertex to every other vertex.
Connected Component

A connected component is a \textbf{maximal} connected subgraph.

2 connected components
Connected Component

A connected component is a **maximal** connected subgraph.

3 connected components
Graph ADT

numVertices(): number of vertices of the graph
getEdge(u,v): returns the edge between vertices u and v
opposite(v,e): returns the vertex other than v that is incident on e
insertVertex(x): creates and returns a new vertex storing value x
insertEdge(u,v,x): creates an edge between u and v storing value x
removeVertex(v): removes vertex v and all edges incident on it
removeEdge(e): removes edge e
areAdjacent(u,v): returns true if u and v are adjacent; false otherwise
incidentEdges(u): returns an iterator of all edges incident on vertex u.
Edge List Structure

- **Vertex object**
  - element
- **Edge object**
  - element
  - origin vertex object
  - destination vertex object
- **Vertex sequence**
  - sequence of vertex objects
- **Edge sequence**
  - sequence of edge objects
Adjacency List Structure

- Incidence sequence for each vertex
  - list of incident edges
Adjacency Matrix Structure

- 2D-array adjacency array
  - Reference to edges for adjacent vertices
  - Null for non-adjacent vertices
- The "old fashioned" version just has 0 for no edge and 1 for edge

![Adjacency Matrix Diagram]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>e</td>
<td>g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>e</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>g</td>
<td>f</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>
# Performance

- **$n$** vertices, **$m$** edges

<table>
<thead>
<tr>
<th></th>
<th>Edge List</th>
<th>Adjacency List</th>
<th>Adjacency Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>incidentEdges($v$)</td>
<td>$O(m)$</td>
<td>$O(\deg(v))$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>areAdjacent ($v, w$)</td>
<td>$O(m)$</td>
<td>$O(\min{\deg(v), \deg(w)})$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insertVertex($v$)</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>insertEdge($v, w, o$)</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>removeVertex($v$)</td>
<td>$O(m)$</td>
<td>$O(\deg(v))$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>removeEdge($v, w$)</td>
<td>$O(m)$</td>
<td>$O(\deg(u) + \deg(v))$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>