What is a Graph?

- A graph \( G = (V,E) \) is composed of:
  - \( V \): set of vertices or nodes
  - \( E \): set of edges connecting the vertices in \( V \)

- An edge, link or arc \( e = (u,v) \) is a pair of vertices

- Example:

\[
\begin{align*}
V &= \{a,b,c,d,e\} \\
E &= \{(a,b),(a,c),(a,d), (b,e),(c,d),(c,e), (d,e)\}
\end{align*}
\]
Edge Types

- **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 flight route

- **Directed graph or Digraph**
  - all the edges are directed
  - e.g., route network

- **Undirected graph**
  - all the edges are undirected
  - e.g., flight network

![Diagram of directed and undirected edges with examples]
Applications

• electronic circuits

find the path of least resistance to CS210

• networks (roads, flights, communications)
A typical student day

- wake up
- eat
- work
- more cs2210
- play
- cs2210 project
- watch TV
- sleep
- dream of cs2210
- make cookies for cs2210 TA
- cs2210: meditation
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
Graph Terminology

- **adjacent vertices**: connected by an edge
- **degree (of a vertex)**: # of adjacent vertices

\[ \sum_{v \in V} \text{deg}(v) = 2(\# \text{ edges}) \]

- Since adjacent vertices each count the adjoining edge, it will be counted twice

**path**: sequence of vertices \( v_1, v_2, \ldots, v_k \) such that consecutive vertices \( v_i \) and \( v_{i+1} \) are adjacent.
More Graph Terminology

• **simple path**: no repeated vertices

- a b
- c
d e

- a c
d e

• **cycle**: simple path, except that the last vertex is the same as the first vertex

- a c d a
- b e c
Even More Terminology

- **connected graph**: any two vertices are connected by some path

- **subgraph**: subset of vertices and edges forming a graph

- **connected component**: maximal connected subgraph. E.g., the graph below has 3 connected components.
Another Terminology Slide!

• (free) tree - connected graph without cycles
• forest - collection of trees
Connectivity

Let $n = \#\text{vertices}$
$n = \#\text{edges}$

Complete graph - all pairs of vertices are adjacent

$m = (1/2)\sum_{v \in V} \deg(v) = (1/2)\sum_{v \in V} (n - 1) = n(n-1)/2$

- Each of the $n$ vertices is incident to $n - 1$ edges, however, we would have counted each edge twice. Therefore, intuitively, $m = n(n-1)/2$.

$n = 5$
$m = (5 \times 4)/2 = 10$

- Therefore, if a graph is not complete, $m < n(n-1)/2$.
More Connectivity

\[ n = \#\text{vertices} \]
\[ m = \#\text{edges} \]

- For a tree \( m = n - 1 \)

\[ n = 5 \]
\[ m = 4 \]

- If \( m < n - 1 \), \( G \) is not connected

\[ n = 5 \]
\[ m = 3 \]
Spanning Tree

• A **spanning tree** of $G$ is a subgraph which
  - is a tree
  - contains all vertices of $G$

![Graph $G$](image1)

![Spanning tree of $G$](image2)

• Failure on any edge disconnects system (least fault tolerant)
The Graph ADT

**Accessor methods**
- `endVertices(e)`: an array of the two endvertices of `e`
- `opposite(v, e)`: the vertex opposite of `v` on `e`
- `areAdjacent(v, w)`: true iff `v` and `w` are adjacent
- `replace(v, x)`: replace element at vertex `v` with `x`
- `replace(e, x)`: replace element at edge `e` with `x`

**Update methods**
- `insertVertex(o)`: insert a vertex storing element `o`
- `insertEdge(v, w, o)`: insert an edge `(v,w)` storing element `o`
- `removeVertex(v)`: remove vertex `v` (and its incident edges)
- `removeEdge(e)`: remove edge `e`

**Iterator methods**
- `incidentEdges(v)`: edges incident to `v`
- `vertices()`: all vertices in the graph
- `edges()`: all edges in the graph