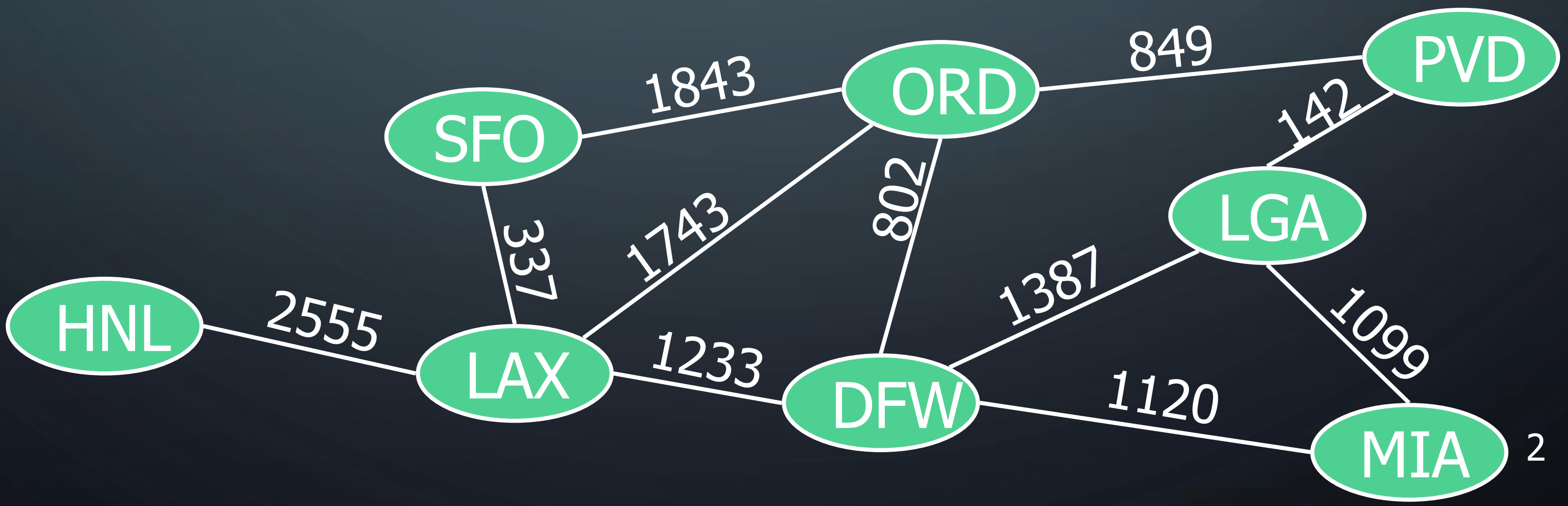
GRAPHS

DR. LORRAINE (LORI) JACQUES

SPRING 2019

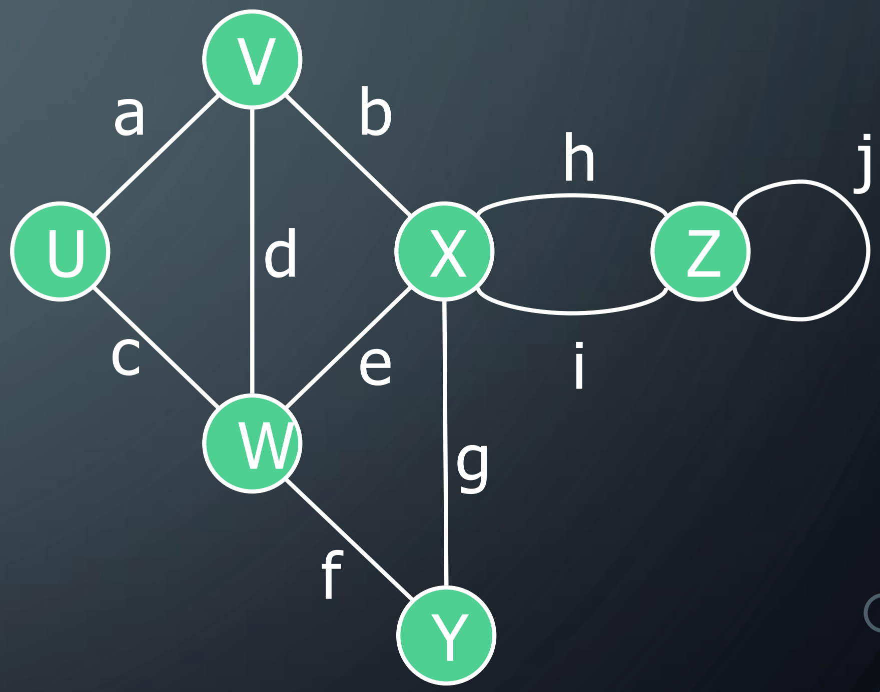
DEFINITION

- 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
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route



TERMS

- **Vertex** – a node in the graph (e.g., U, V, X)
- **Edge** – a path connecting 2 vertices (e.g., a, b, d)
- **Degree** of a vertex – the number of edges on a vertex (e.g., X has degree 5)
- End vertices – endpoints of an edge (e.g., U and V are the endpoints of a)
- Incident edges – edges on a vertex (e.g., a, d, and b are incident on V)
- Adjacent vertices – vertices connected by an edge (e.g., U and V are adjacent)
- Parallel edges – edges with the same end vertices (e.g., h and i are parallel edges)
- Self-loop – an edge connecting only one vertex (e.g., j is a self-loop)

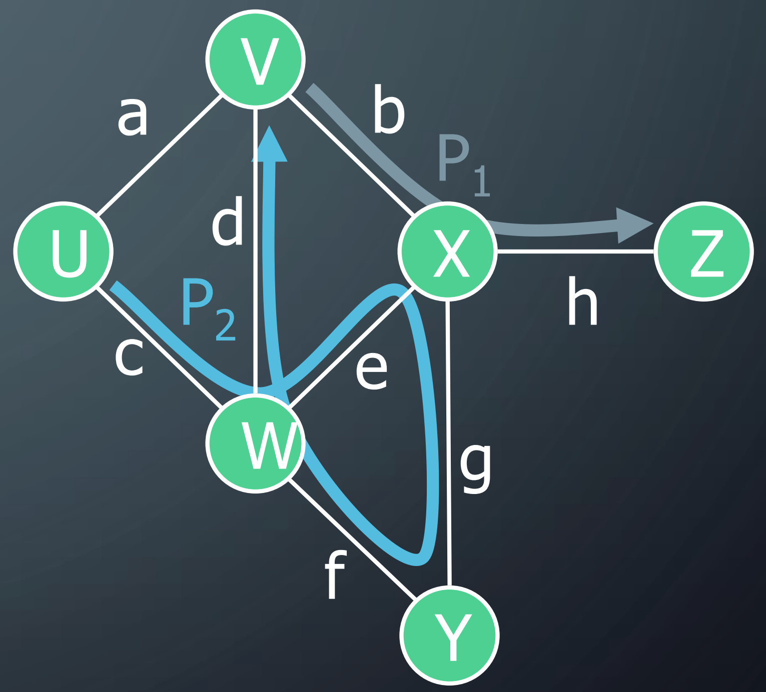


MORE TERMS

- **Path**
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- Simple path
  - path such that all its vertices and edges are distinct
- Examples

  P_1=(V,b,X,h,Z) is a simple path

  P_2=(U,c,W,e,X,g,Y,f,W,d,V) is a path that is not simple

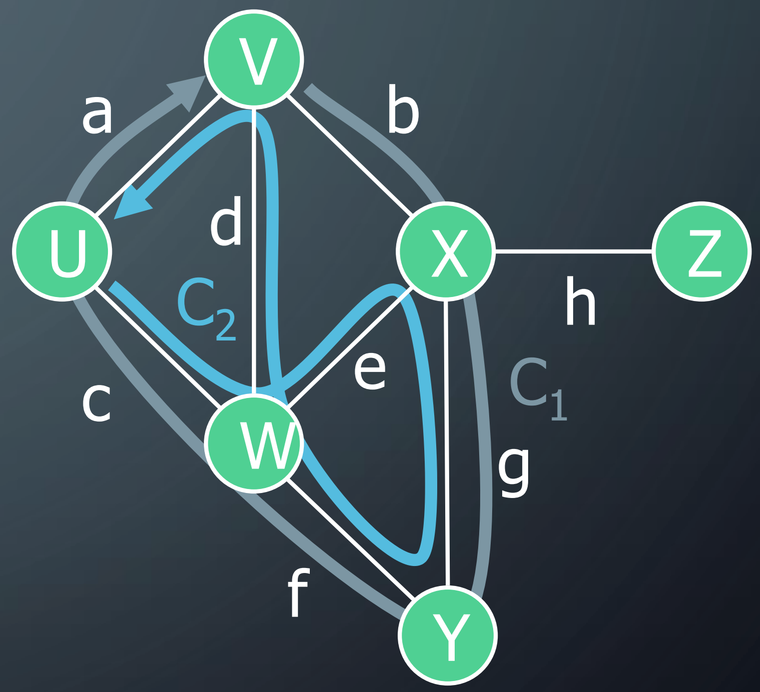


EVEN MORE TERMS

- **Cycle**
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints
- Simple cycle
  - cycle such that all its vertices and edges are distinct
- Examples

  C_1=(V,b,X,g,Y,f,W,c,U,a,↵) is a simple cycle

  C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,↵) is a cycle that is not simple



TYPES OF EDGES

- Directed – An ordered pair of vertices (u,v) where the first vertex is the origin and the second vertex is the destination
  - So travel is only in one direction on the edge – from u to v
  - E.g., a flight
- Undirected – An unordered pair of vertices (u,v)
  - So travel can be from u to v or from v to u
  - E.g., a flight path
- Weighted – An ordered or unordered pair of vertices (u, v) with an additional value indicating that path's cost
  - Cost can be time to travel, distance, etc.
  - E.g., miles between airports

So edges can be directed and weighted, directed and unweighted, undirected and weighted, or undirected and unweighted

TYPES OF GRAPHS – BASED ON THE EDGES

- Directed graph - all the edges are directed
  - e.g., route network
- Undirected graph - all the edges are undirected
  - e.g., flight network
- Weighted graph – all the edges have weights
- Unweighted graph – none of the edges have weights

Like edges, a graph can be a combination of these.

PROPERTIES

Property 1 – Sum of Degrees

$\Sigma_v \deg(\boldsymbol{v}) = 2\boldsymbol{m}$

Why: each edge is counted twice

Property 2 – Max # of Edges

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

$\boldsymbol{m} \leq \boldsymbol{n}\,(\boldsymbol{n} - 1)/2$

Why: each vertex has degree at most $(\boldsymbol{n} - 1)$

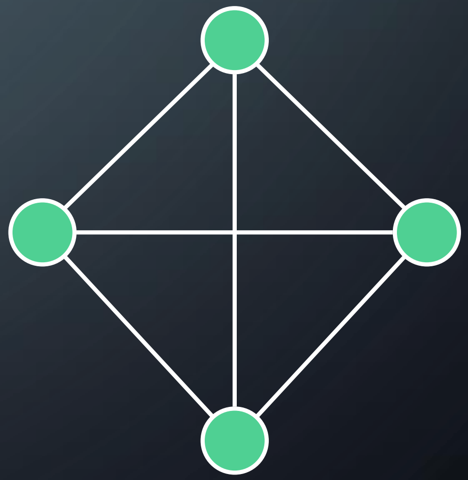
What is max # of edges for a directed graph?

Notation

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

Example

- $\boldsymbol{n} = 4$
- $\boldsymbol{m} = 6$
- $\deg(\boldsymbol{v}) = 3$



Reminder: Degree of a vertex – the number of edges on a vertex (e.g., X has degree 5)

PROGRAMMING A GRAPH

VERTEX CLASS

- Considered a lightweight object because it doesn't do much
- Stores the information for the vertex (like Node does)
- And only has methods for getting or setting that information, nothing else

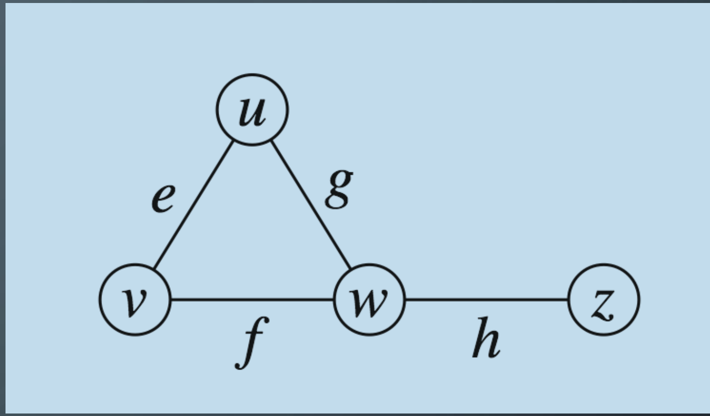
EDGE CLASS

- Stores the endpoint vertices – one for origin and one for destination
  - Yes, it assumes a directed graph
- For weighted graphs, also stores the weight
- Methods (in addition for getters and setters)
  - Endpoints – returns both vertices
  - Opposite – returns the vertex that is not the same as the vertex passed as a parameter

SO HOW DO EDGES AND VERTICES KEEP TRACK OF EACH OTHER?

Option 1: Adjacency matrix

- 2D-array of edge objects
  - Reference to edge object for adjacent vertices
  - Null for non nonadjacent vertices
  - *In 220, you just used 0 for no edge and 1 for edge*
- So each vertex gets associated with a row number (and identical column number)
  - Vertex class thus needs an “index” property

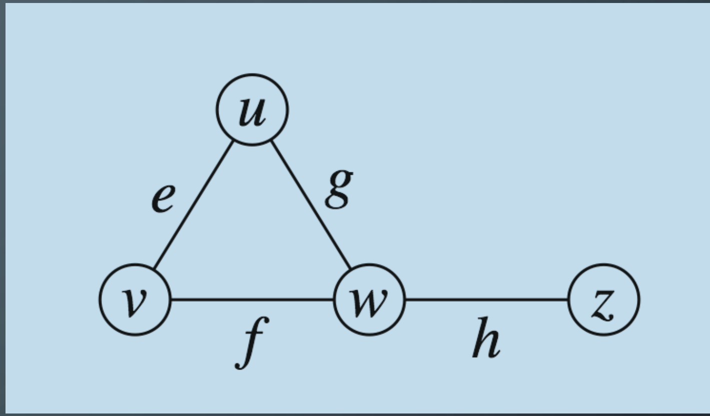


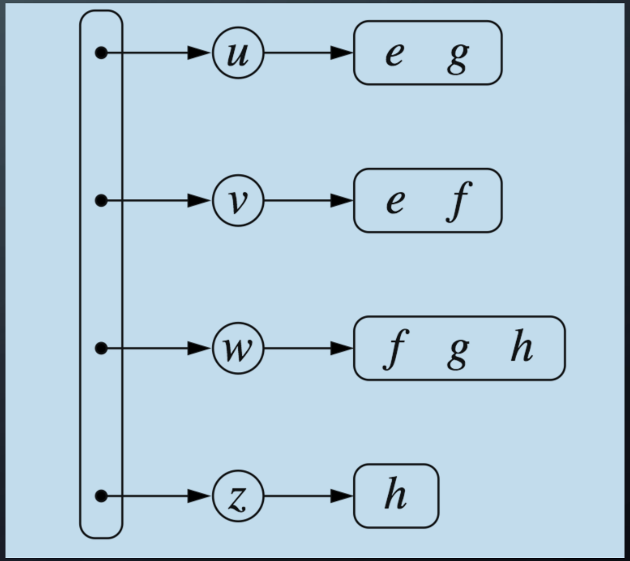
		0	1	2	3
$u \longrightarrow$	0		e	g	
$v \longrightarrow$	1	e		f	
$w \longrightarrow$	2	g	f		h
$z \longrightarrow$	3			h	

SO HOW DO EDGES AND VERTICES KEEP TRACK OF EACH OTHER?

Option 2: Adjacency list

- An array or linked list
- Each element in the array/list contains a vertex and a list
  - That list is a list of the vertex's edges
  - Can find out who's adjacent to whom by using that opposite method in the Edge class





GRAPH CLASS

Properties:

- numV – number of vertices
- adj – adjacency matrix or list
- directed – true/false indicating if a directed graph

Methods:

- Getters and setters, including a method to list all edges
- Traversing the graph (will output the vertices)
- getEdge – takes 2 vertices as parameters and returns the edge connecting them or null if not adjacent
- addEdge / removeEdge – adds/removes an edge to adj
  - Include code to put edge in twice if graph is undirected – once for origin to destination and once for reverse
- addVertex / removeVertex – adds/removes a vertex <u>and its edges</u> (can't have an edge to/from nowhere)
- degree – returns the degree of a vertex
- incidentEdges – returns all (outgoing) edges of a vertex
- edgeSum – sum of the weights of the edges

Other properties and methods may be added depending on what you need the graph to do

CONCLUSION – APPLICATIONS OF GRAPHS

- Electronic circuits
  - Printed circuit board, Integrated circuit
- Transportation networks
  - Highway network, Flight network
- Computer networks
  - Local area network, Internet, Web
- Databases
  - Entity-relationship diagram