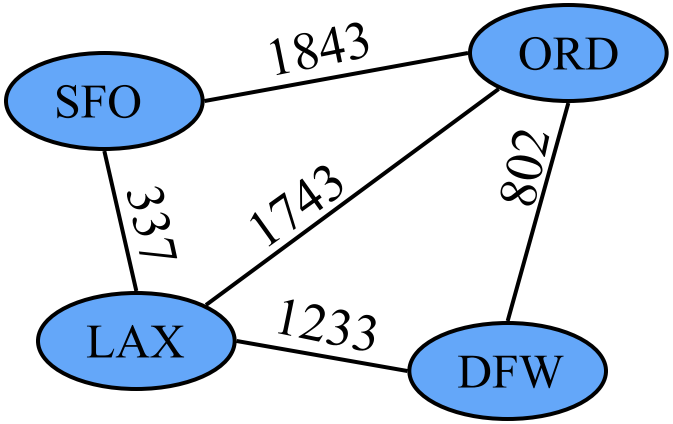
Graphs



Outline / Reading

Graphs (6.1)
- Definition
- Applications
- Terminology
- Properties
- ADT

Data structures for graphs (6.2)
- Edge list structure
- Adjacency list structure
- Adjacency matrix structure

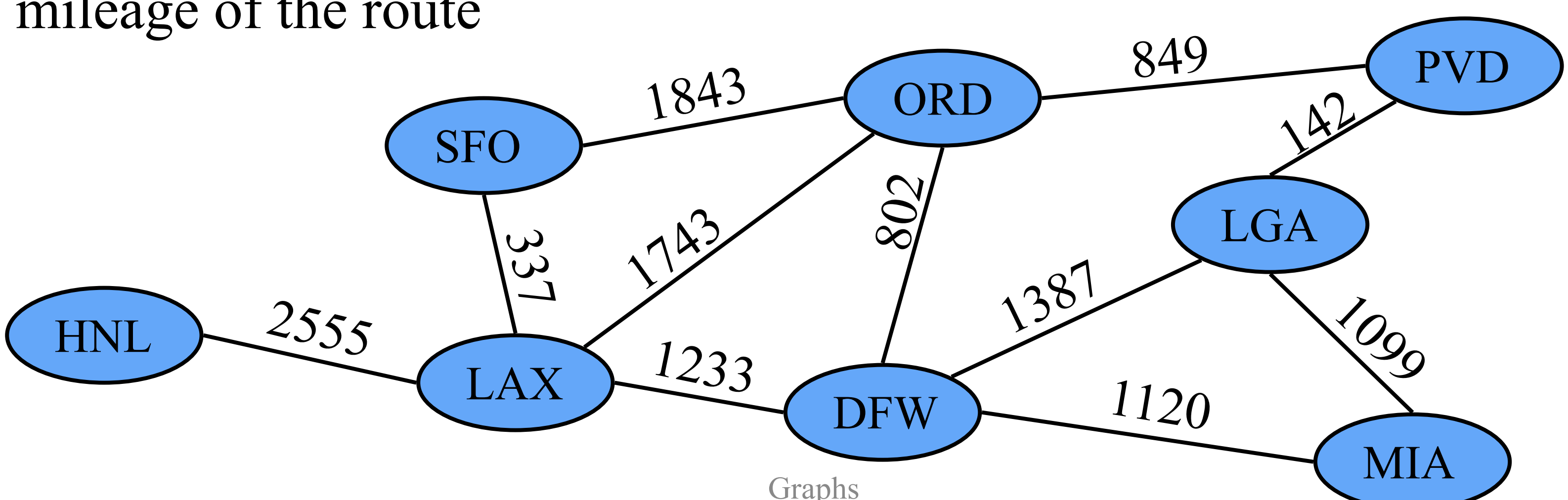
Graph

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



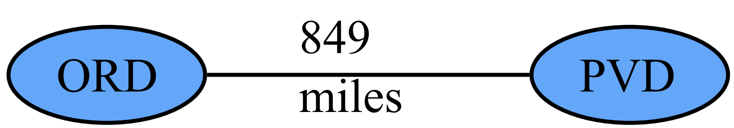
Edge Types

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

Directed graph
- all the edges are directed
- e.g., flight network

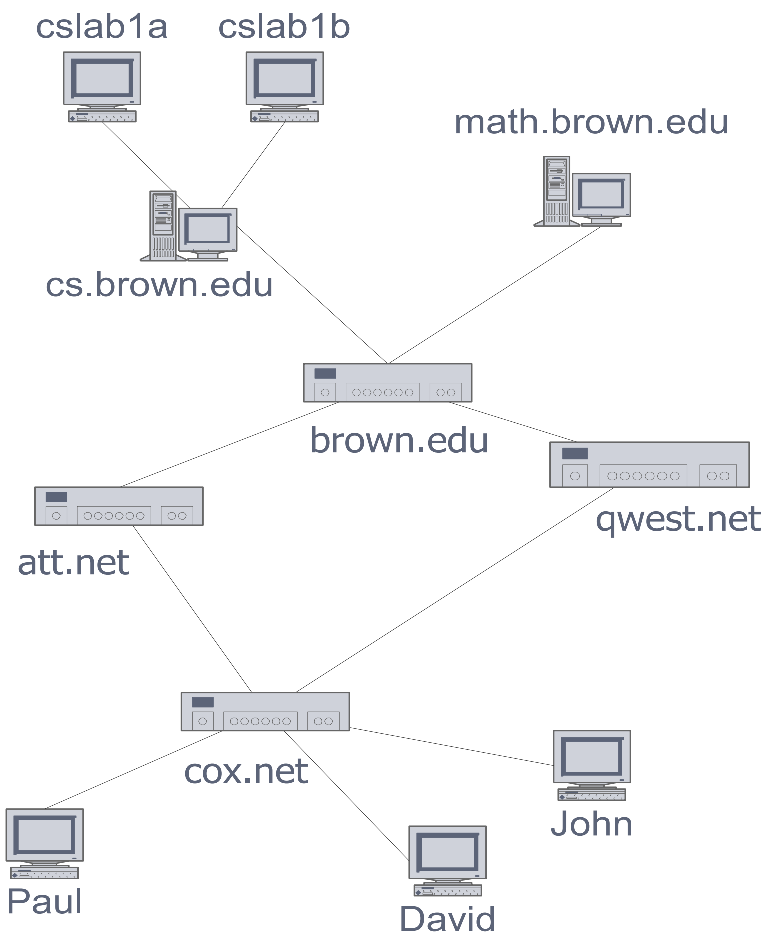
Undirected edge
- unordered pair of vertices $(\boldsymbol{u},\boldsymbol{v})$
- e.g., a flight route

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



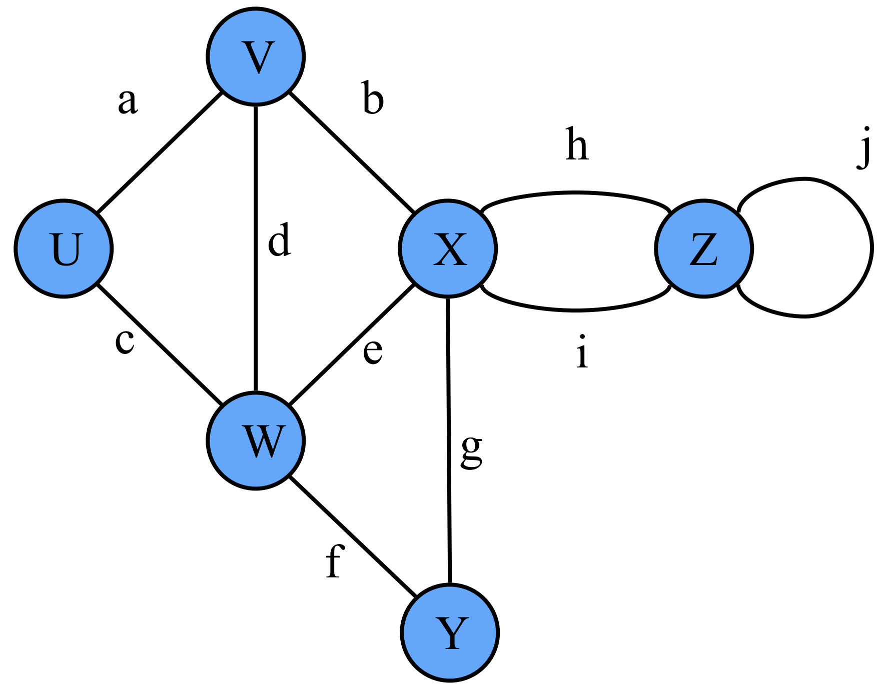
Applications

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



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



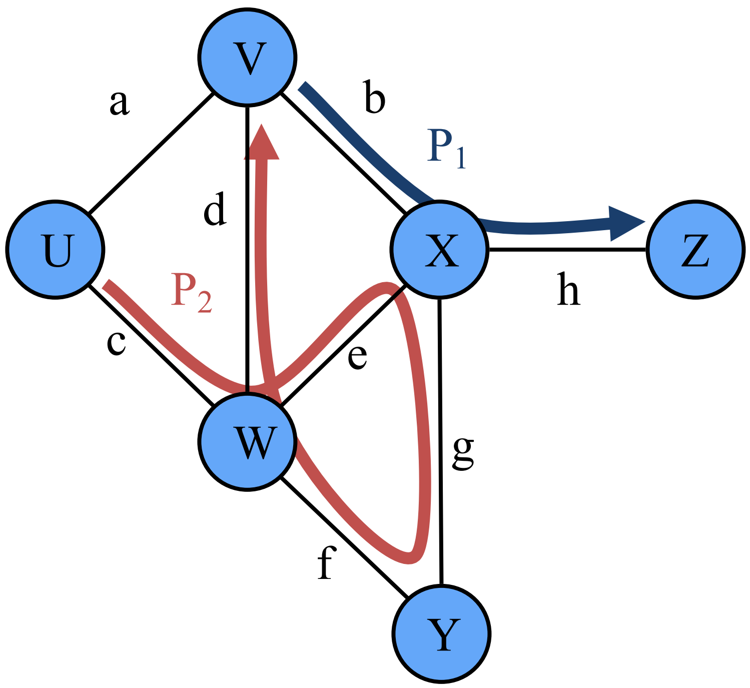
Terminology (cont.)

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

Terminology (cont.)

Cycle

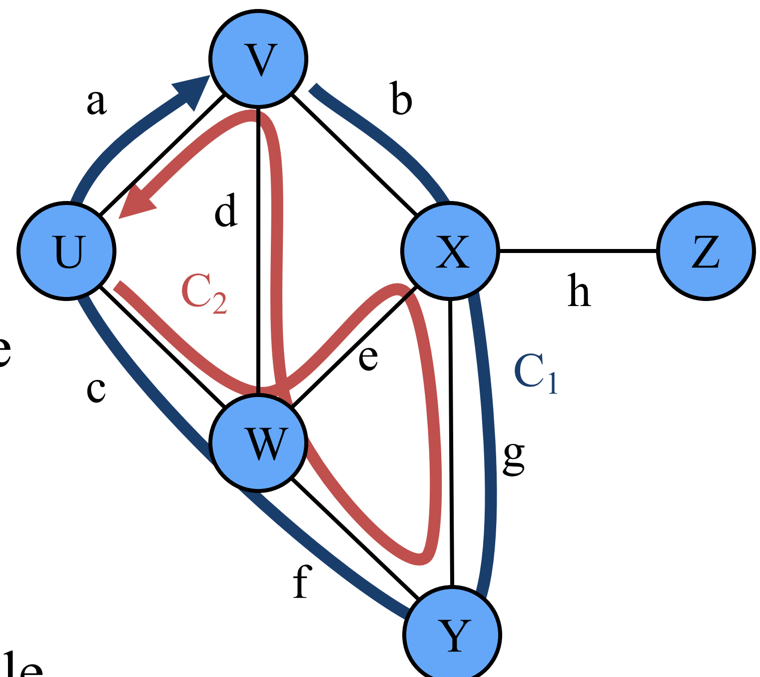
- circular sequence of alternating vertices and edges
- each edge is preceded and followed by its endpoints

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



Properties

Notation

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

Property 1. $\Sigma_v \deg(v) = 2m$

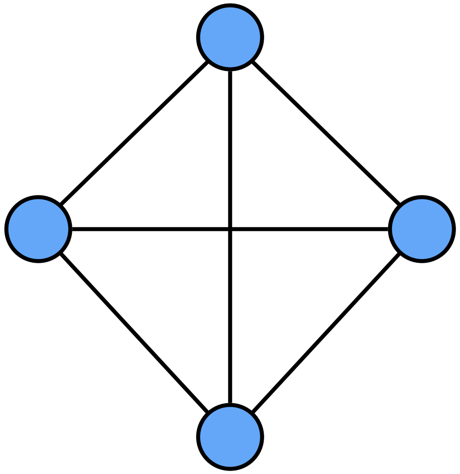
Proof: each edge is counted twice

Property 2. In an undirected graph with no self-loops and no multiple edges

$$m \leq n(n-1)/2$$

Proof: each vertex has degree at most $(n-1)$

What is the bound for a directed graph?



Ex: $n = 4$; $m = 6$; $\deg(v) = 3$

Main Methods of the Graph ADT

Vertices and edges
- are positions
- store elements

Accessor methods
- aVertex()
- incidentEdges(v)
- endVertices(e)
- isDirected(e)
- origin(e)
- destination(e)
- opposite(v, e)
- areAdjacent(v, w)

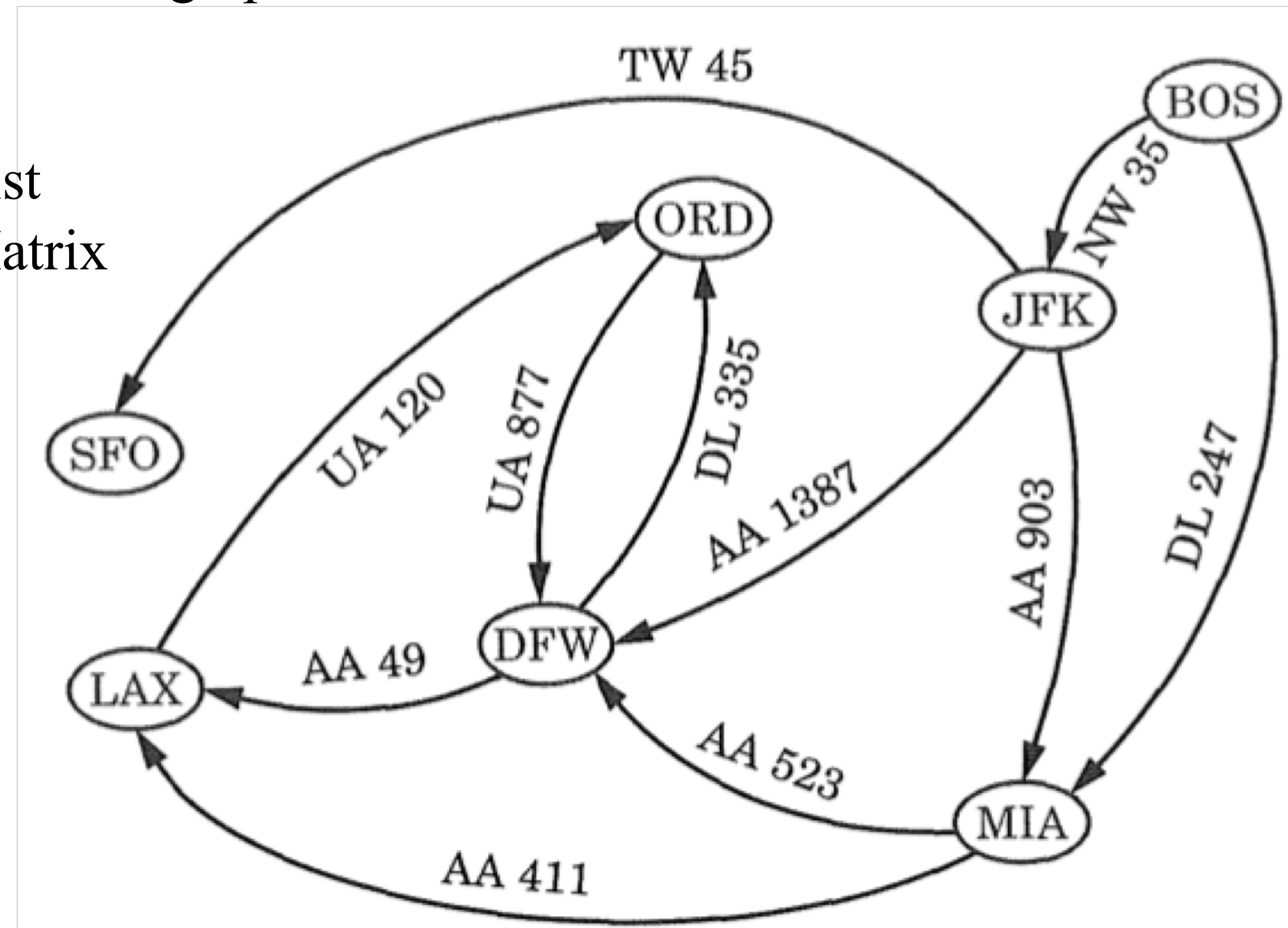
Update methods
- insertVertex(o)
- insertEdge(v, w, o)
- insertDirectedEdge(v, w, o)
- removeVertex(v)
- removeEdge(e)

Generic methods
- numVertices()
- numEdges()
- vertices()
- edges()

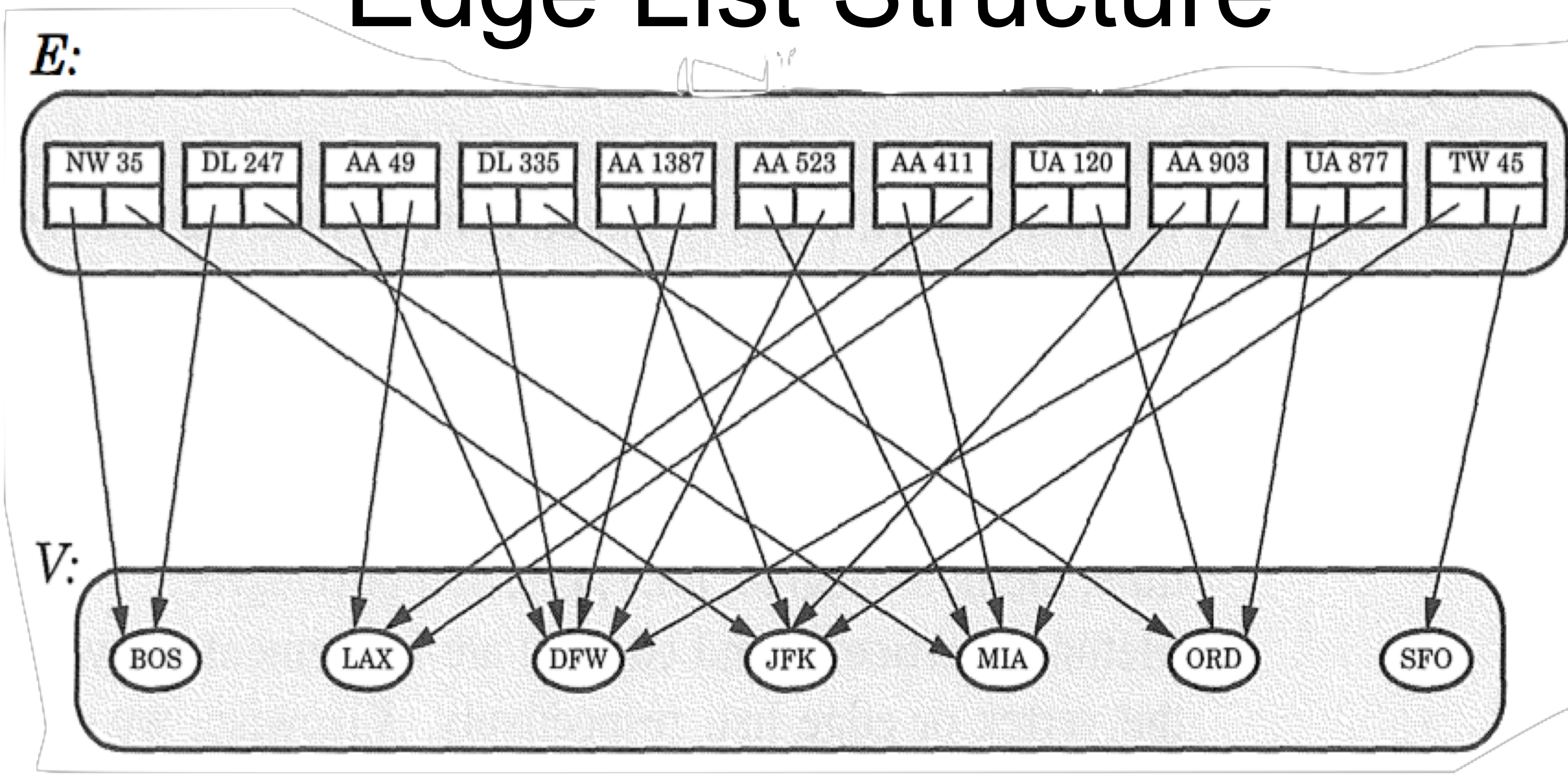
Data Structures

Structures to represent a graph:

1. Edge List
2. Adjacency List
3. Adjacency Matrix

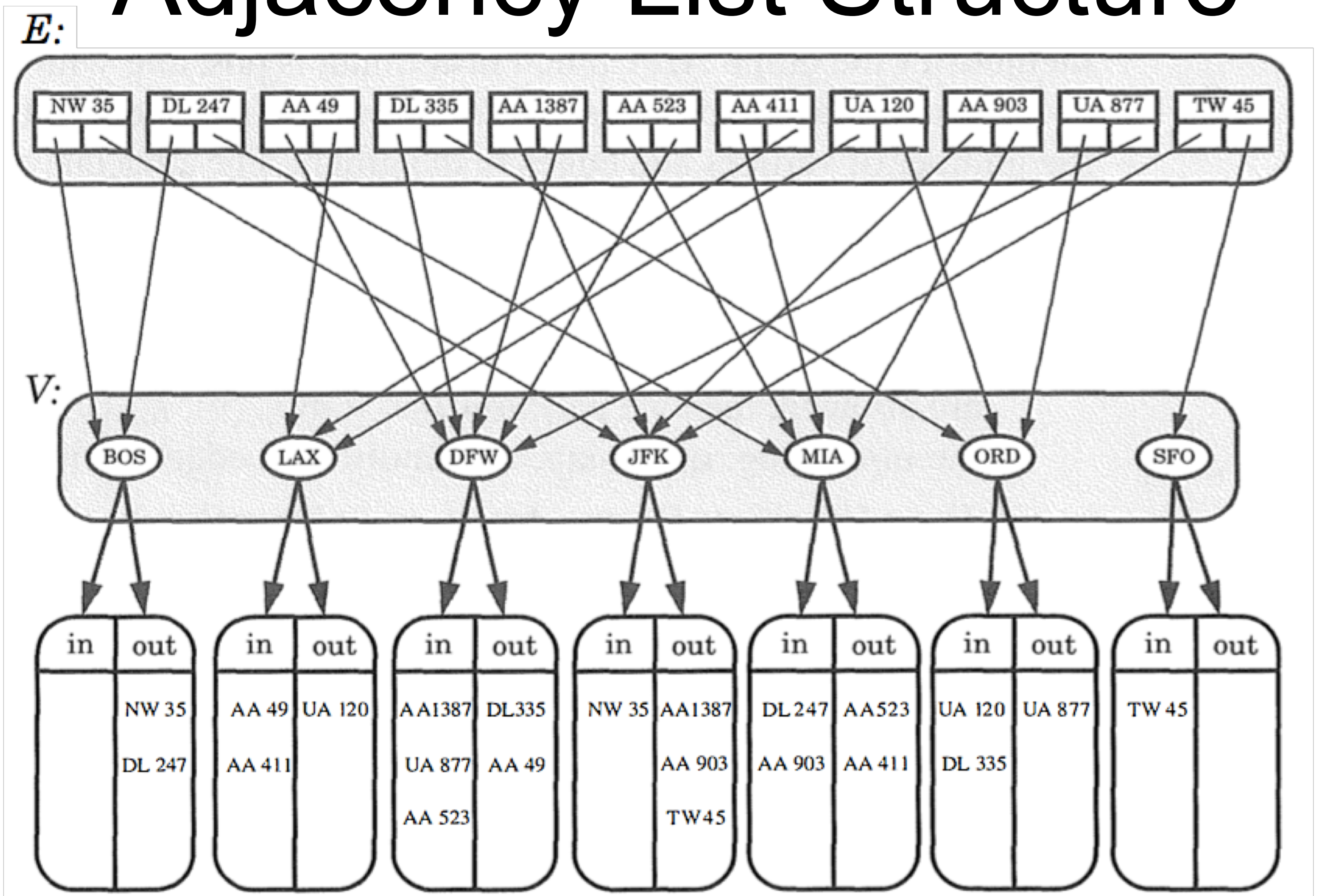


Edge List Structure



A container of edge objects, where each edge object references the origin and destination vertex object

Adjacency List Structure



An edge list structure, where additionally each vertex object *v* references an incidence container which stores references to the edges incident on *v*.

Adjacency Matrix Structure

		0 BOS	1 DFW	2 JFK	3 LAX	4 MIA	5 ORD	6 SFO
		0	1	2	3	4	5	6
BOS	0	Ø	Ø	NW 35	Ø	DL 247	Ø	Ø
DFW	1	Ø	Ø	Ø	AA 49	Ø	DL 335	Ø
JFK	2	Ø	AA 1387	Ø	Ø	AA 903	Ø	TW 45
LAX	3	Ø	Ø	Ø	Ø	Ø	UA 120	Ø
MIA	4	Ø	AA 523	Ø	AA 411	Ø	Ø	Ø
ORD	5	Ø	UA 877	Ø	Ø	Ø	Ø	Ø
SFO	6	Ø	Ø	Ø	Ø	Ø	Ø	Ø

A 2D array of all vertex pairs, where cell A[u,v] stores edge e incident on vertices u,v if such an edge exists.

Asymptotic Performance

◆ n vertices, m edges ◆ no parallel edges ◆ no self-loops ◆ Bounds are "big-Oh"	Edge List	Adjacency List	Adjacency Matrix
Space	$n + m$	$n + m$	n^2
incidentEdges(v)	m	$\deg(v)$	n
areAdjacent (v, w)	m	$\min(\deg(v), \deg(w))$	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	$\deg(v)$	n^2
removeEdge(e)	1	1	1