Graphs

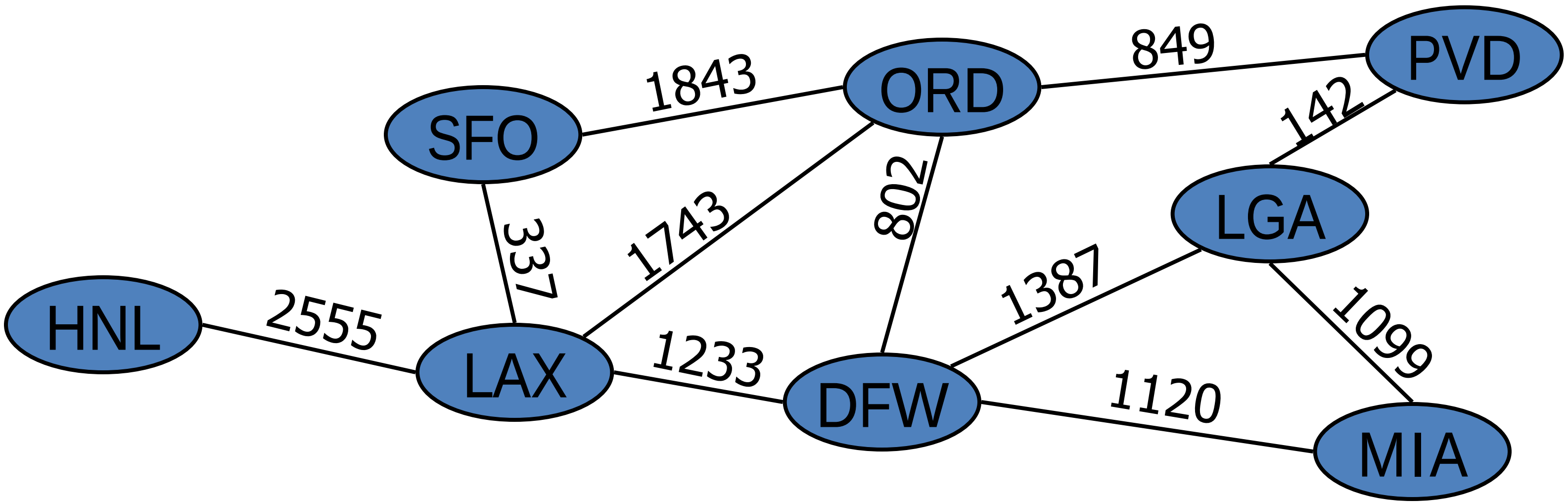
cse2011

sections 13.1 and 13.2 of textbook

Graphs

- 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 objects and store elements

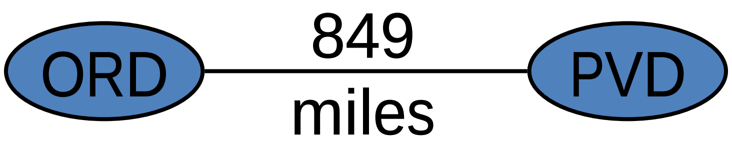
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route



Edge Types

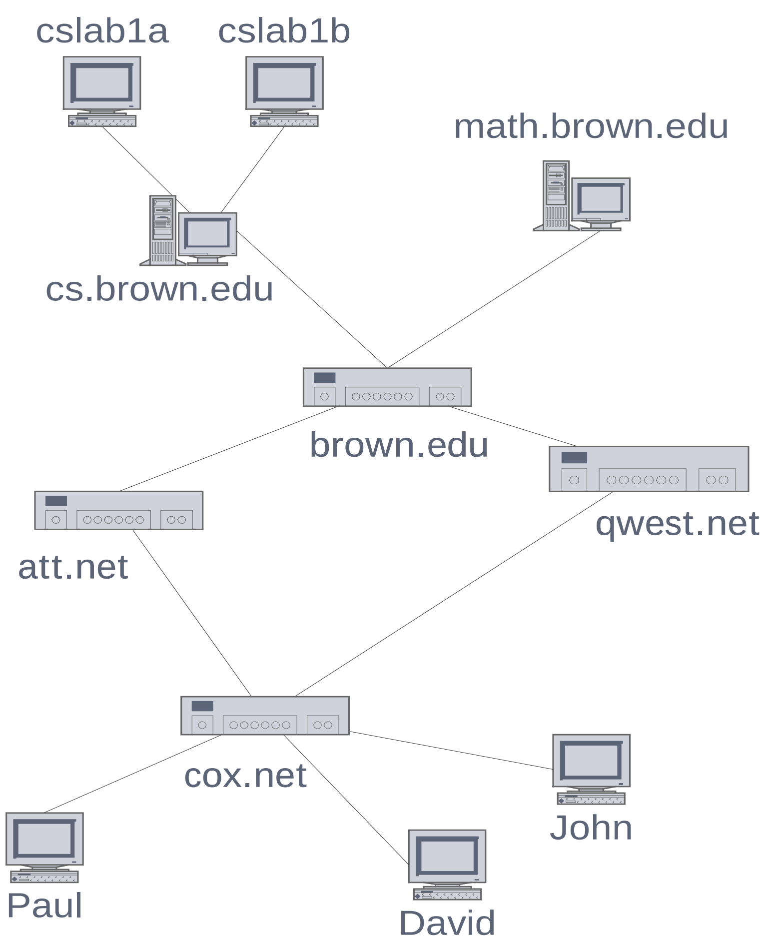
- **Directed edge**
  - ordered pair of vertices (u,v)
  - first vertex u is the origin
  - second vertex v is the destination
  - e.g., a flight
- **Undirected edge**
  - unordered pair of vertices (u,v)
  - e.g., a flight route
- **Directed graph** (digraph)
  - all the edges are directed
  - e.g., flight network
- **Undirected graph**
  - all the edges are undirected
  - e.g., route network
- **Mixed graph**

  contains both directed and undirected edges



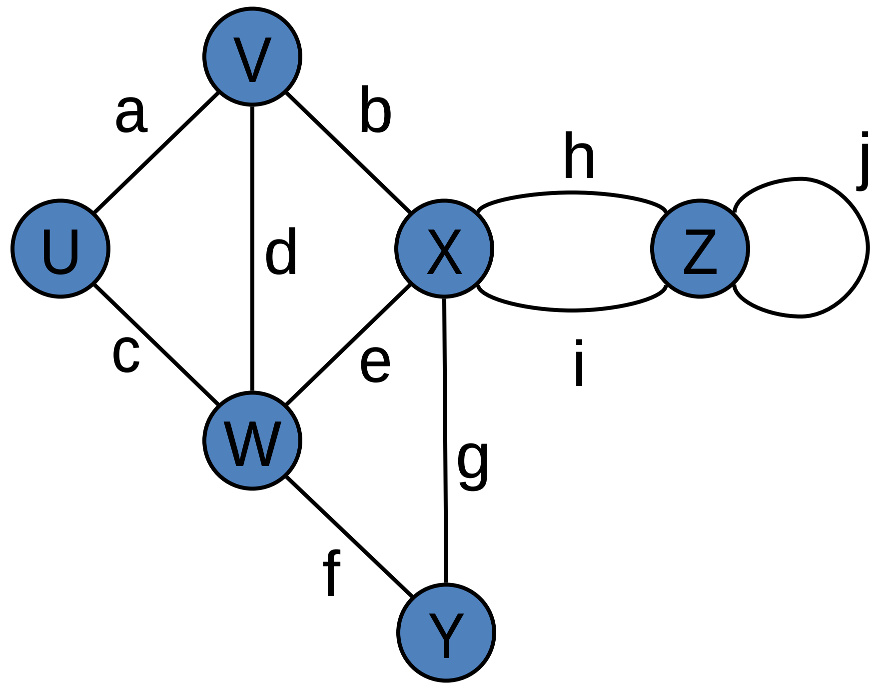
Applications

- 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

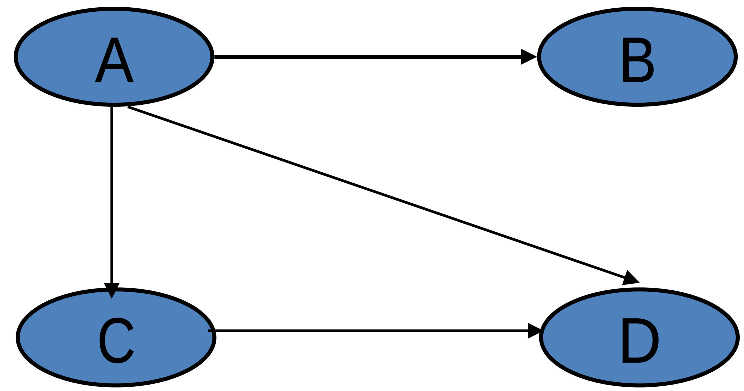
- **End vertices** (or endpoints) of an edge
  - U and V are the endpoints of a
- Edges **incident** on a vertex
  - a, d, and b are incident on V
- **Adjacent** vertices
  - U and V are adjacent
- **Degree** of a vertex
  - W has degree 4
- **Loop**
  - j is a loop
    (we will consider only loopless graphs)



Terminology (2)

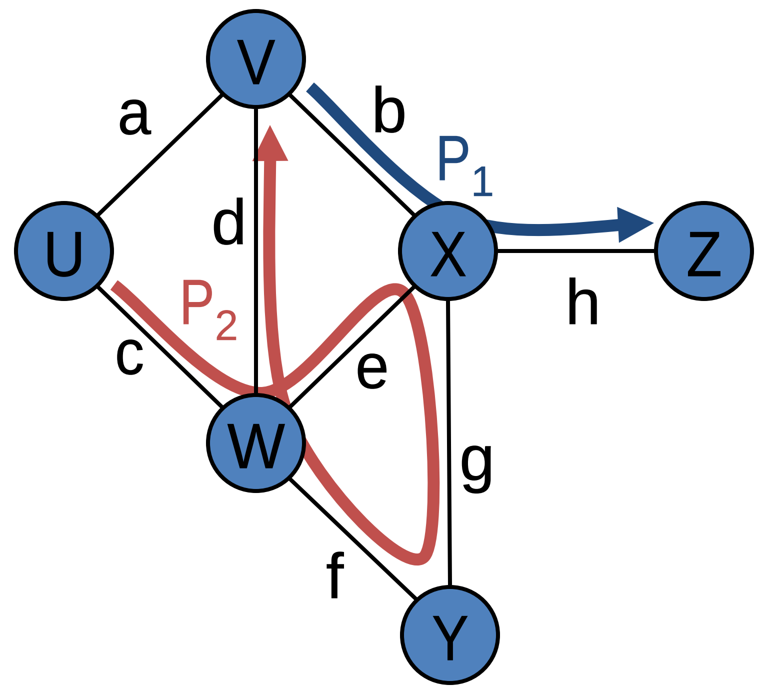
For directed graphs:

- **Origin**, destination of an edge
- **Outgoing** edge
- **Incoming** edge
- **Out-degree** of vertex v:
  - number of outgoing edges of v
- **In-degree** of vertex v:
  - number of incoming edges of v



Paths

- **Path**
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- **Path length**
  - the total number of edges on the path
- **Simple path**
  - path such that all vertices are distinct (except that the first and last could be the same)
- 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



Properties of Undirected Graphs

Property 1

$\sum_v \deg(v) = 2E$

Proof: each edge is counted twice

Notation

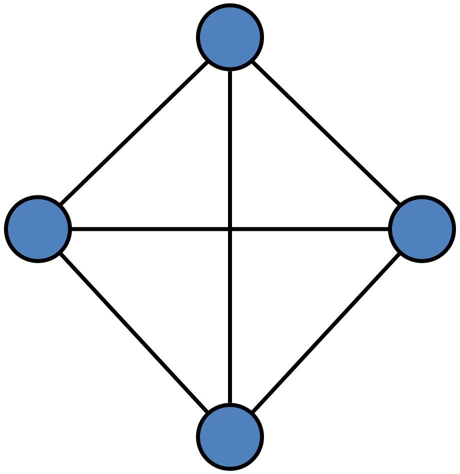
V	number of vertices
E	number of edges
$\deg(v)$	degree of vertex v

Property 2

In an undirected graph with no loops

$\mathbf{E} \leq \mathbf{V}\,(\mathbf{V} - 1)/2$

Proof: each vertex has degree at most $(\mathbf{V} - 1)$



Example

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

What is the bound for a directed graph?

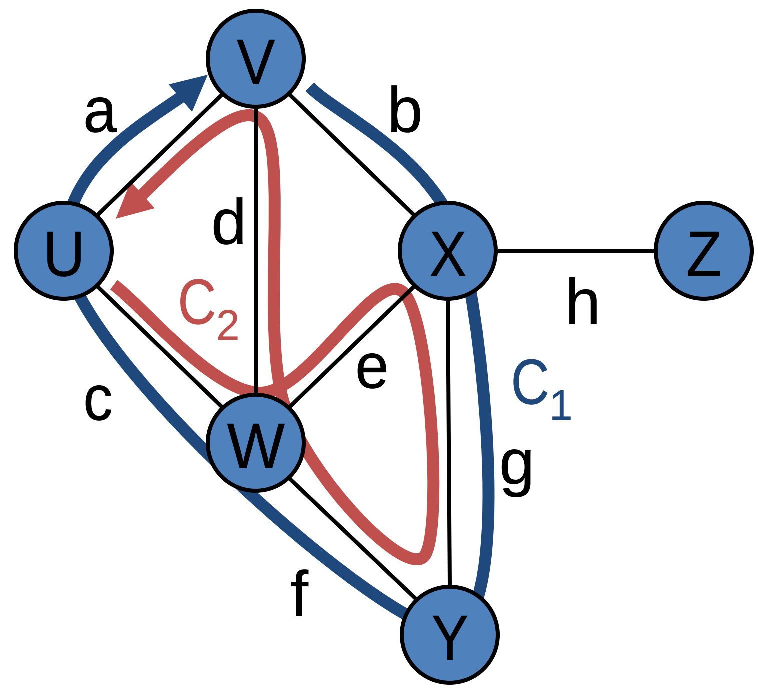
Cycles

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

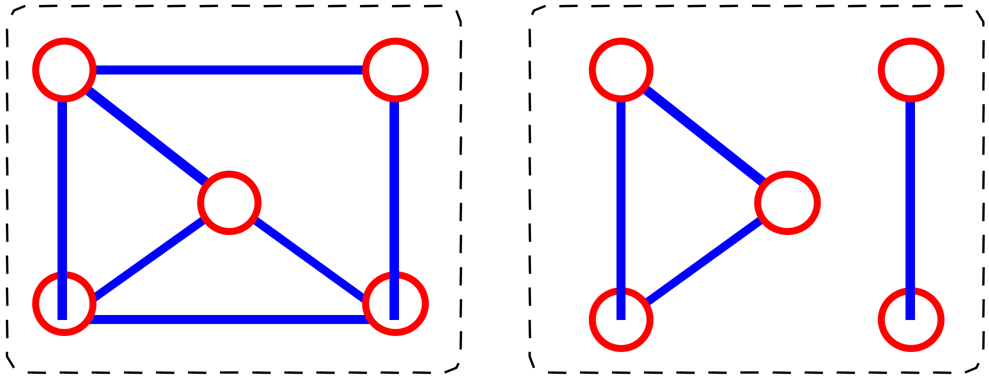
- **Simple cycle**
  - cycle such that all its vertices are distinct (except the first and the last)

- Examples
  - C_1=(V,b,X,g,Y,f,W,c,U,a,V) is a simple cycle
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,U) is a cycle that is not simple

- A directed graph is *acyclic* if it has no cycles ⇒ called DAG (directed acyclic graph)



Connectivity – Undirected Graphs

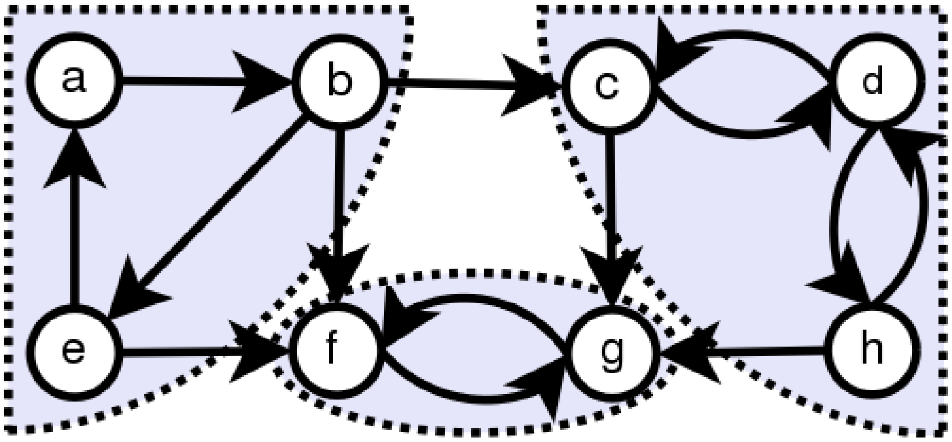


connected   not connected

- An undirected graph is *connected* if there is a path **from every** vertex **to every** other vertex.

Connectivity – Directed Graphs

- A **directed** graph is called ***strongly*** *connected* if there is a path from every vertex to every other vertex.
- If a directed graph is not strongly connected, but the corresponding undirected graph is connected, then the directed graph is said to be ***weakly*** *connected*.



Graph ADT and Data Structures

CSE 2011

Representation of Graphs

- Two popular computer representations of a graph: Both represent the vertex set and the edge set, but in different ways.

1. **Adjacency Matrices**

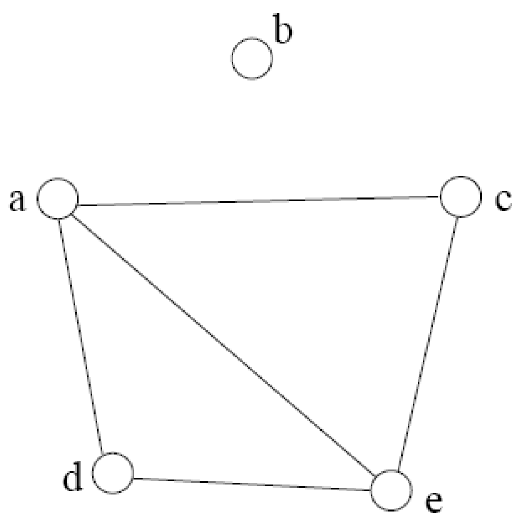
   Use a 2D matrix to represent the graph

2. **Adjacency Lists**

   Use a set of linked lists, one list per vertex

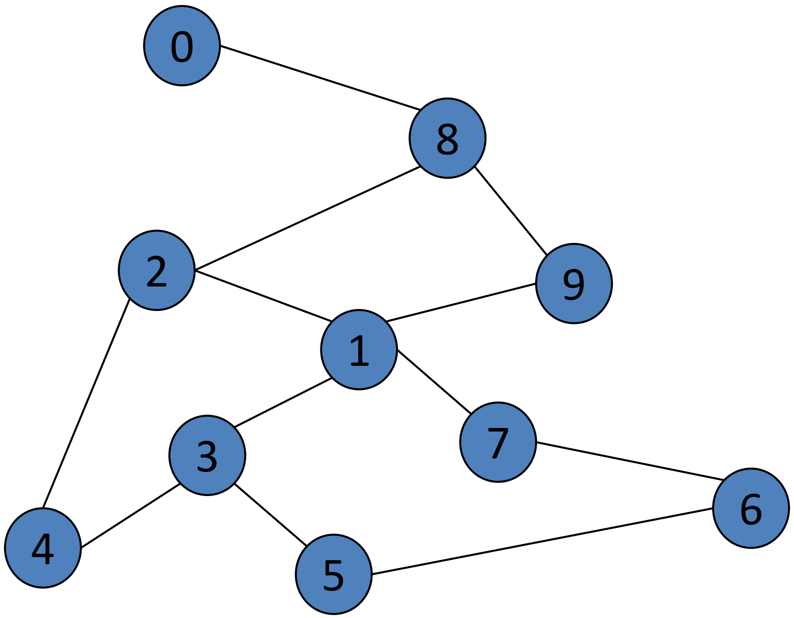
Adjacency Matrix Representation

- 2D array of size ***n*** x ***n*** where ***n*** is the number of vertices in the graph
- A[i][j]=1 if there is an edge connecting vertices i and j; otherwise, A[i][j]=0



	a	b	c	d	e
a	0	0	1	1	1
b	0	0	0	0	0
c	1	0	0	0	1
d	1	0	0	0	1
e	1	0	1	1	0

Adjacency Matrix Example



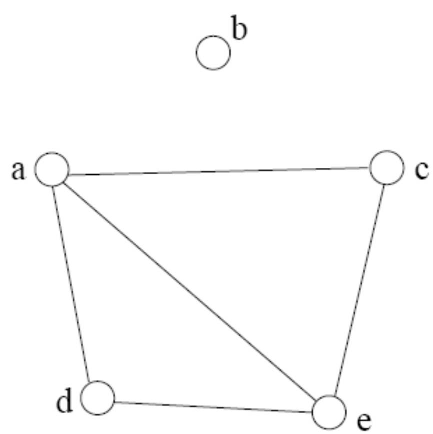
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0	1	0	1
2	0	1	0	0	1	0	0	0	1	0
3	0	1	0	0	1	1	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0
5	0	0	0	1	0	0	1	0	0	0
6	0	0	0	0	0	1	0	1	0	0
7	0	1	0	0	0	0	1	0	0	0
8	1	0	1	0	0	0	0	0	0	1
9	0	1	0	0	0	0	0	0	1	0

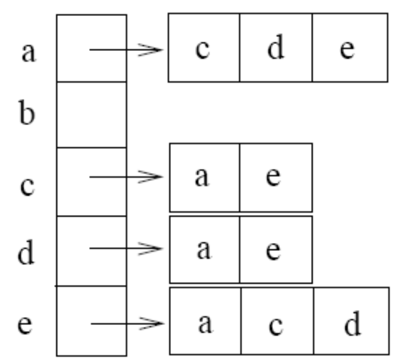
Adjacency Matrices: Analysis

- The storage requirement is $\Theta(V^2)$.
  - not efficient if the graph has few edges.
  - appropriate if the graph is dense; that is E= $\Theta(V^2)$
- If the graph is undirected, the matrix is symmetric. There exist methods to store a symmetric matrix using only half of the space.
  - Note: the space requirement is still $\Theta(V^2)$.
- We can detect in O(1) time whether two vertices are connected.

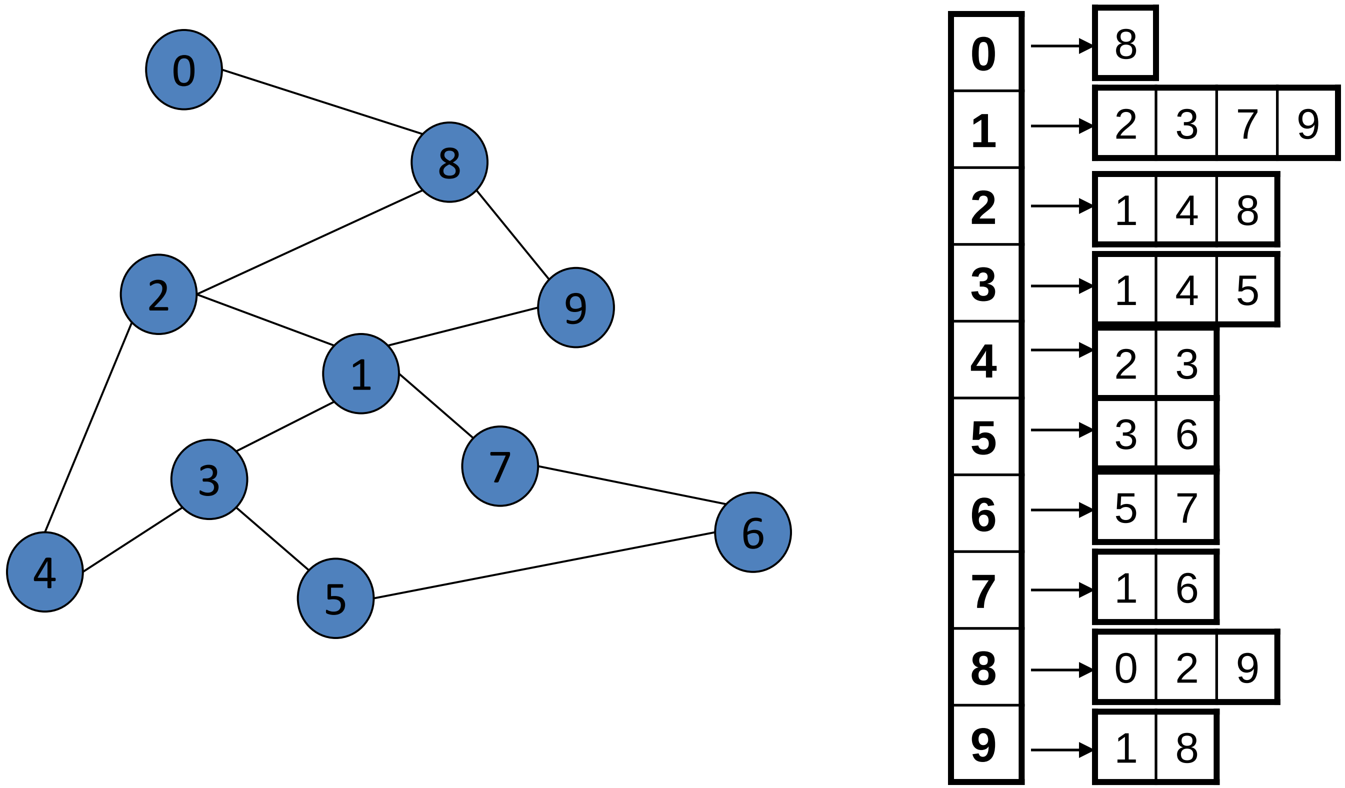
Adjacency Lists

- If the graph is sparse, a better solution is an adjacency list representation.
- For each vertex ***v*** in the graph, we keep a list of vertices adjacent to ***v***.

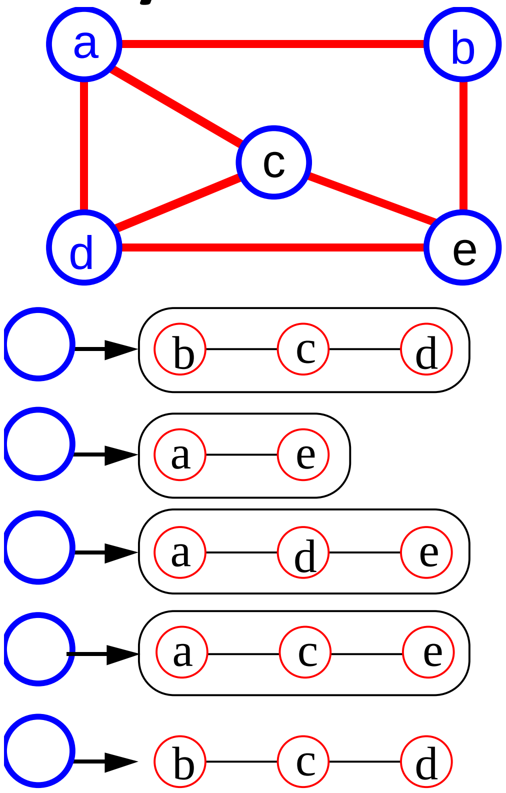




Adjacency List Example



Adjacency Lists: Analysis



Space =

$\Theta\,(V + \Sigma_v \deg(v)) = \Theta\,(V + E)$

- Testing whether u is adjacency to v takes time O(deg(v)) or O(deg(u)).

Adjacency Lists vs. Adjacency Matrices

- An adjacency list takes $\Theta(V + E)$.
  - If $E = O(V^2)$ (dense graph), both use $\Theta(V^2)$ space.
  - If $E = O(V)$ (sparse graph), adjacency lists are more space efficient.

- Adjacency lists
  - More compact than adjacency matrices if graph has few edges
  - Requires more time to find if an edge exists

- Adjacency matrices
  - Always require $\Theta(V^2)$ space
    - This can waste lots of space if the number of edges is small
  - Can quickly find if an edge exists

(Undirected) Graph ADT

- Vertices and edges
  - are positions
  - store elements
- Define Vertex and Edge interfaces, each extending Position interface
- Accessor methods
  - endVertices(e): returns an array of the two end vertices of e
  - opposite(v, e): returns the vertex opposite of v on e
  - areAdjacent(v, w): returns 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

(Undirected) Graph ADT (2)

- Update methods
  - insertVertex(o): inserts a vertex storing element o
  - insertEdge(v, w, o): inserts an edge (v, w) storing element o
  - removeVertex(v): removes vertex v (and its incident edges)
  - removeEdge(e): removes edge e
- Iterator methods
  - incidentEdges(v): returns the edges incident to v
  - vertices(): returns all vertices in the graph
  - edges(): returns all edges in the graph

Homework

- Prove the big-Oh running time of the graph methods shown in the next slide.

Running Time of Graph Methods

• 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

Next lectures …

- Graph traversal
  - Breadth first search (BFS)
    - Applications of BFS
  - Depth first search (DFS)
- Review
- Final exam (Thursday August 8th in CLH E from 14:00 to 17:00.)