CS-2001
DATA STRUCTURE

Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

GRAPH

Graph

- A graph is a collection of nodes (or vertices) and edges (or arcs)
  - Each node contains an element
  - Each edge connects two nodes together (or possibly the same node to itself) and may contain an edge attribute

Graph

Graph: A data structure that consists of a set of nodes and a set of edges that relate the nodes to one another.

Vertex: A node in a graph is called vertex

Edges(arc): A pair of vertices representing a connection between two nodes in a graph

Undirected Graph: A graph in which the edges have no direction

Directed Graph: A graph in which each edge is directed from one vertex to another(or the same) vertex

Graph

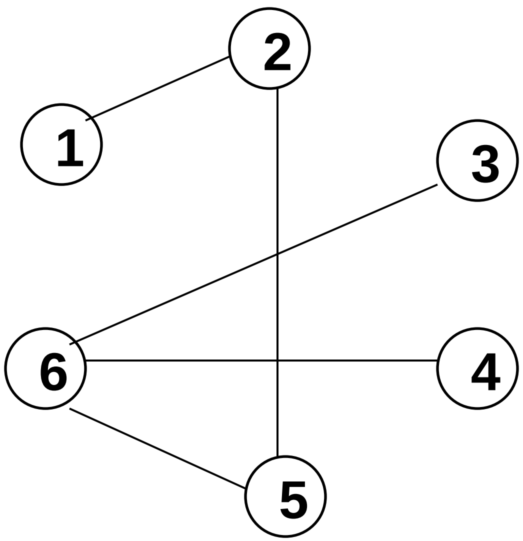
- There are **two kinds of graphs**:
  - Undirected graphs
  - Directed graphs

- A directed graph is one in which the edges do not have direction

- An undirected graph is one in which the edges have a direction.

Graph … Examples

Undirected Graph



V = {1, 2, 3, 4, 5, 6}
E = {(1, 2) (2, 5) (3, 6) (4, 6) (5, 6)}

Graph … Examples

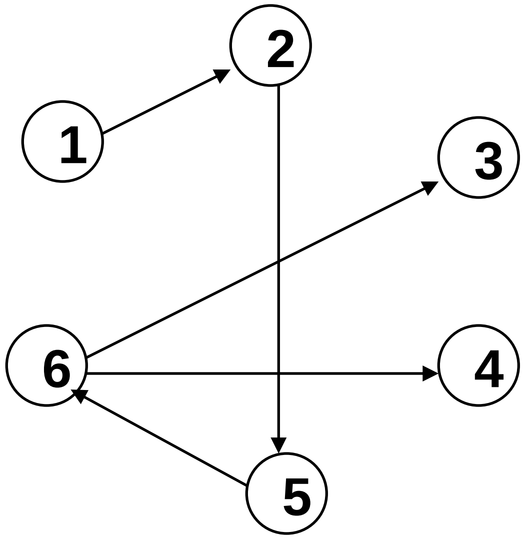
Directed Graph

If the pair of vertices is ordered then the graph is a ***directed* graph,**

G=(V,E):

V = {1, 2, 3, 4, 5, 6}

E = {(1, 2) (2, 5) (5, 6) (6, 3) (6, 4)}



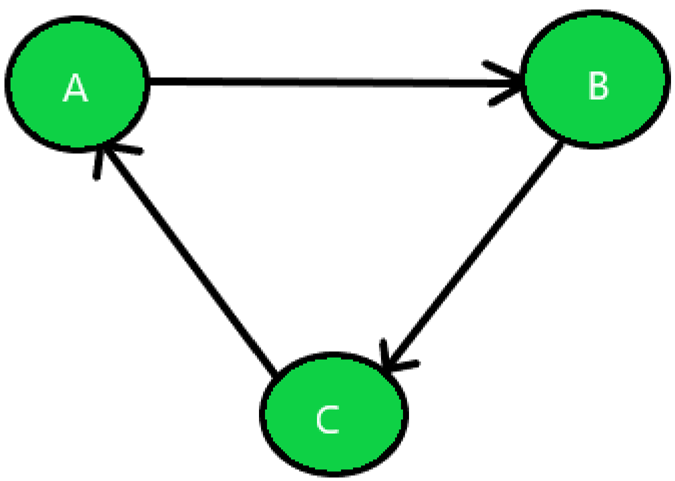
Terminologies

- A graph is said to be ***connected*** if there is a path from every vertex to every other vertex

- ***Strongly Connected:*** A graph is said to be **strongly connected** if *every pair of vertices(u, v) in the graph contains a path between each other*.
- ***Unilaterally Connected:*** A graph is said to be **unilaterally connected** if *it contains a directed path from u to v OR a directed path from v to u for every pair of vertices u, v.*
- ***Weakly Connected:*** A graph is said to be **weakly connected** if *there doesn't exist any path between any two pairs of vertices*.

Terminologies

Strongly Connected Graph

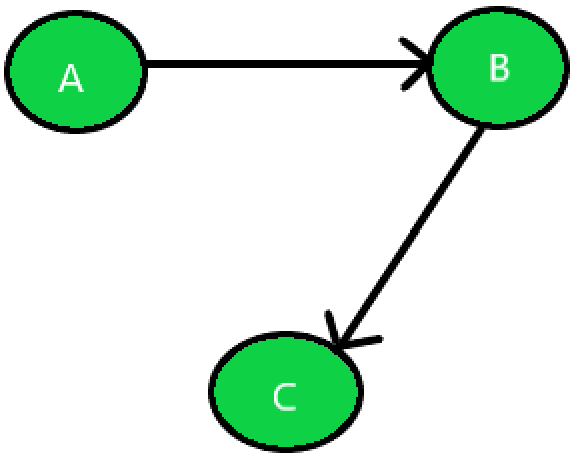


Path Matrix :

	A	B	C
A	1	1	1
B	1	1	1
C	1	1	1

Terminologies

Unilaterally Connected Graph

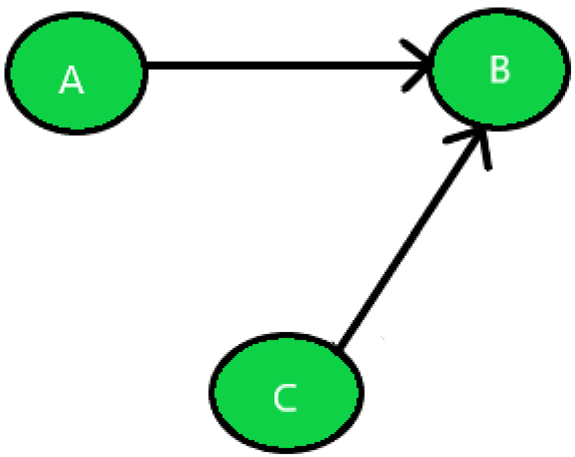


Path Matrix :

	A	B	C
A	0	1	1
B	0	0	1
C	0	0	0

Terminologies

Weakly Connected Graph

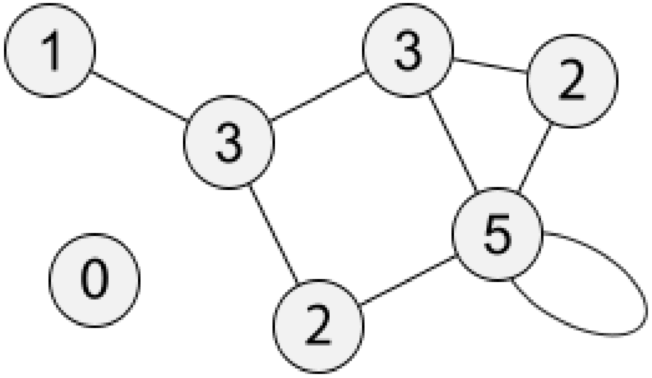


Path Matrix :

	A	B	C
A	0	1	0
B	0	0	0
C	0	1	0

Adjacency

- Vertices connected by an edge are ***adjacent***
- ***Degree*** of a vertex v, deg(v)
  - number of edges incident on v
- The ***degree (or valency)*** of a vertex of a graph is the number of edges incident to the vertex, with loops counted twice



IMPLEMENTATION

Graph Representations

- Adjacency-matrix Representation
- Adjacency Lists Representation

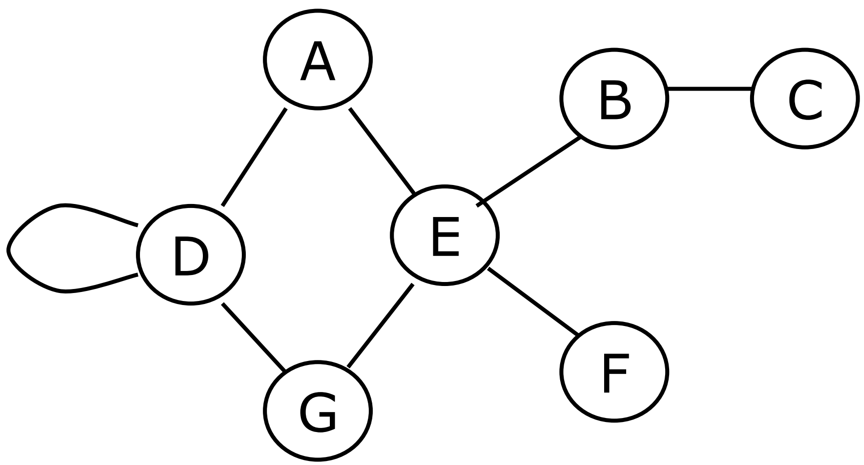
Adjacency Matrix

Undirected Graphs:

- One simple way of representing a graph is the **adjacency matrix**
- A 2-D array has a mark at $[i][j]$ if there is an edge from node i to node j
- The adjacency matrix is *symmetric* about the main diagonal.
- This representation is only suitable for small graphs!

Adjacency Matrix

Undirected Graphs:



	A	B	C	D	E	F	G
A				●	●		
B			●		●		
C		●					
D	●			●			●
E	●	●				●	●
F					●		
G				●	●		

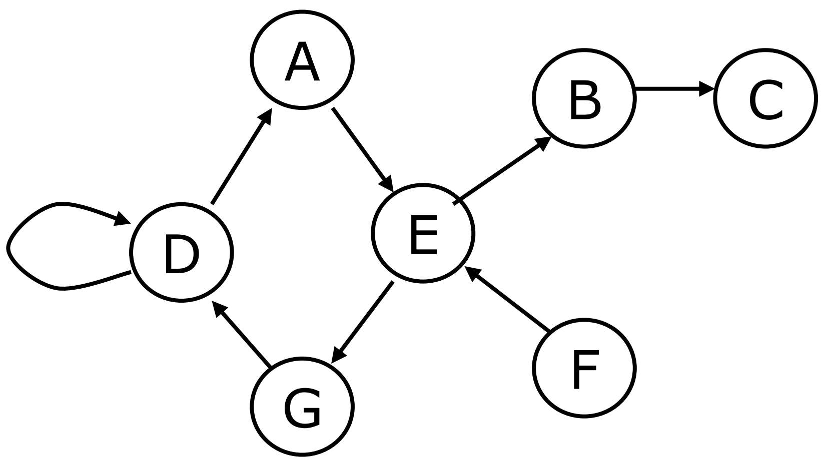
Adjacency Matrix

Directed Graphs:

- An adjacency matrix can equally well be used for directed graphs.
- A 2-D array has a mark at $[i][j]$ if there is an edge from node i to node j.

Adjacency Matrix

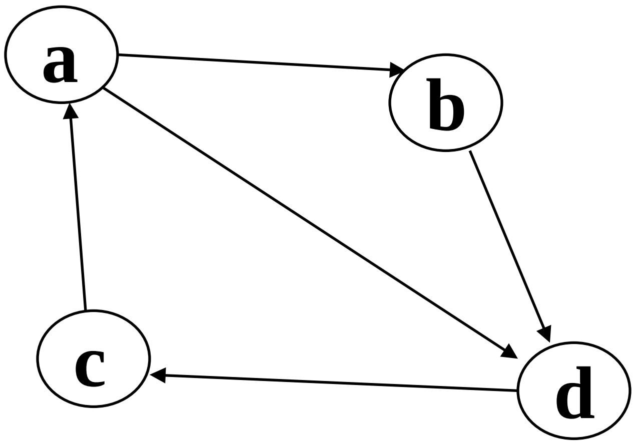
Directed Graphs:



	A	B	C	D	E	F	G
A					●		
B			●				
C							
D	●			●			
E		●					●
F					●		
G				●			

Adjacency Matrix

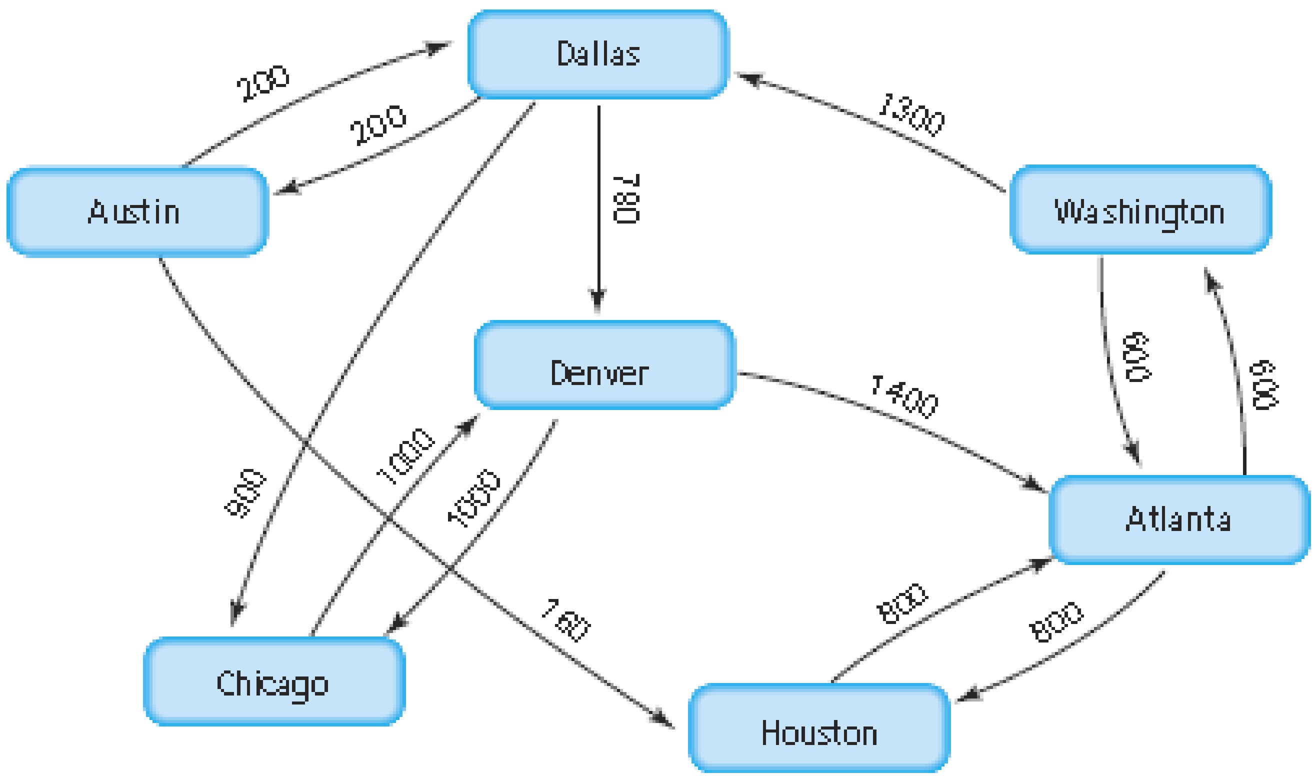
Directed Graphs:



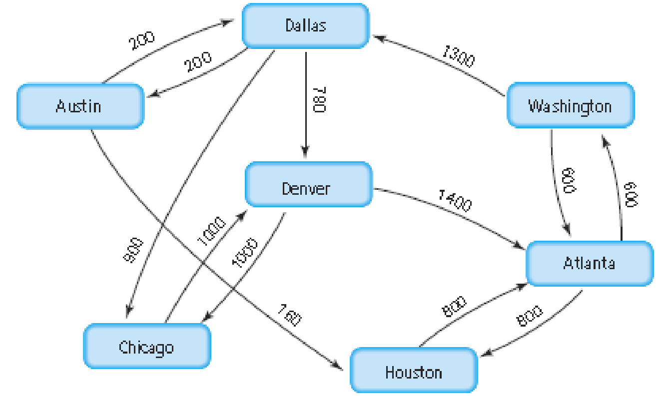
	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	1	0	0	0
d	0	0	1	0

Adjacency Matrix

<u>Weighted Directed Graphs:</u>



Adjacency Matrix

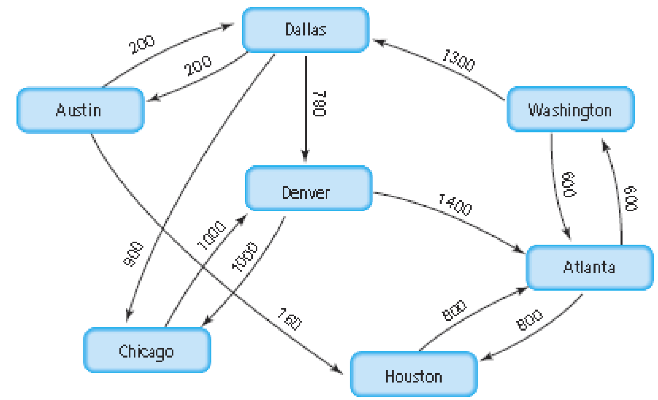


Weighted Directed Graphs:

Austin
160 miles
Houston
800 miles
Atlanta
600 miles
Washington

Austin
200 miles
Dallas
780 miles
Denver
1400 miles
Atlanta
600 miles
Washington

Adjacency Matrix



Weighted Directed Graphs:

.vertices

[0]	"Atlanta "
[1]	"Austin "
[2]	"Chicago "
[3]	"Dallas "
[4]	"Denver "
[5]	"Houston "
[6]	"Washington"

.edges

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	0	0	0	0	0	800	600
[1]	0	0	0	200	0	160	0
[2]	0	0	0	0	1000	0	0
[3]	0	200	900	0	780	0	0
[4]	1400	0	1000	0	0	0	0
[5]	800	0	0	0	0	0	0
[6]	600	0	0	1300	0	0	0

Adjacency Lists

Adjacency List

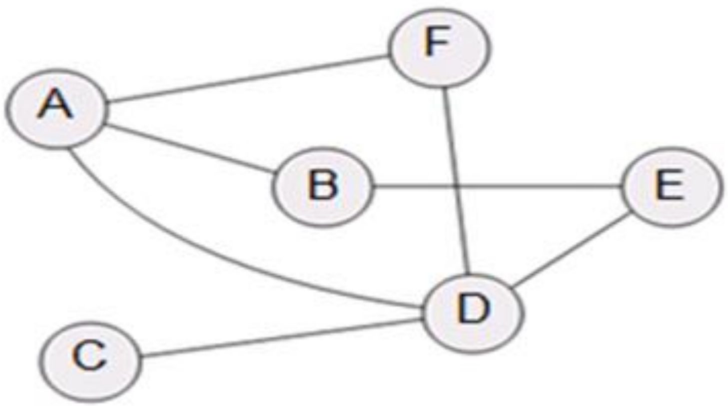
- Array of lists
- Each *vertex has an array entry*
- A vertex w is inserted in the list for vertex v if there is an outgoing edge from v to w

Adjacency Lists

- A graph of n nodes is represented by a one-dimensional array L of linked lists, where,
  - L[i] is the linked list containing all the nodes adjacent to node i
  - The nodes in the list L[i] are in NO particular order,
  - An adjacency list for a weighted graph should contain **two elements** in the list nodes – one element for the vertex and the second element for the weight of that edge

Adjacency Lists

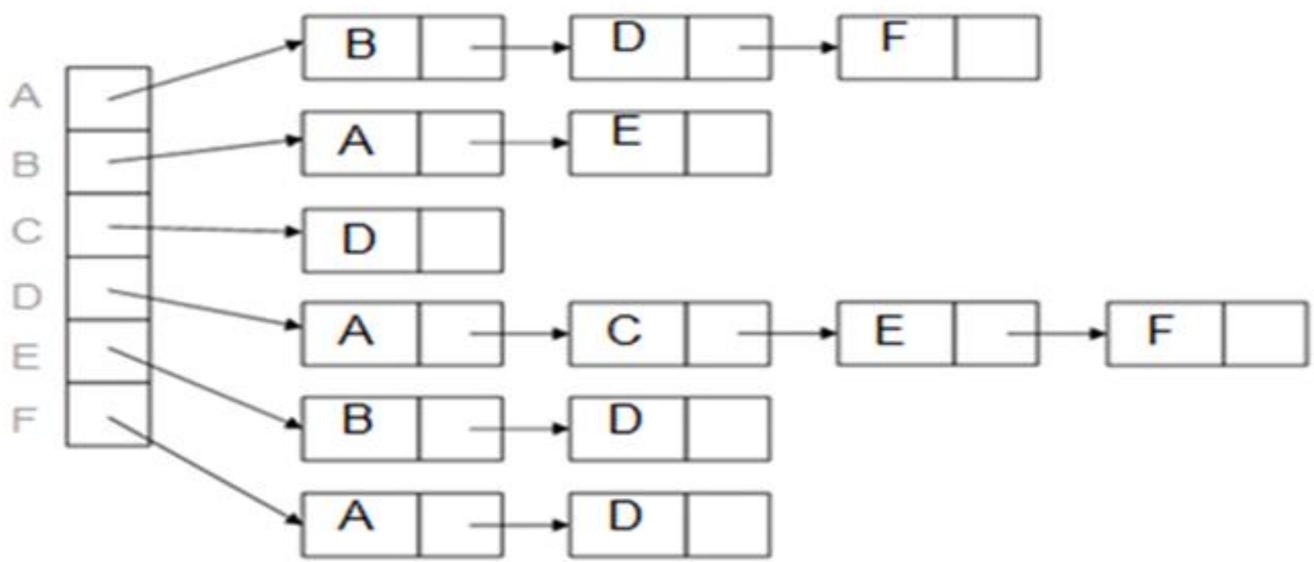
An (undirected) graph $G = (V,E)$



adjacency matrix for G

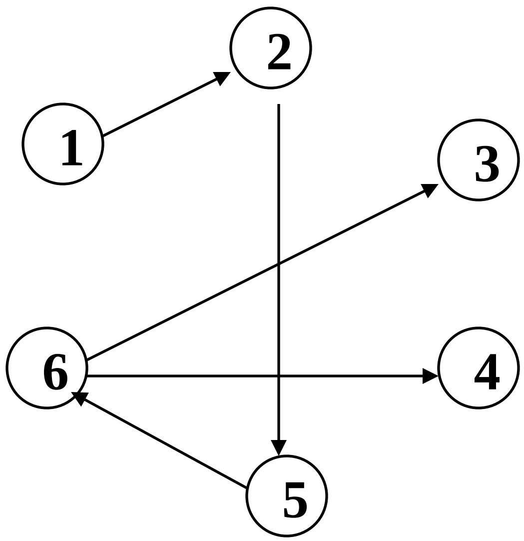
$$
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 1 & 0 & 1 & 0 & 1 \\
B & 1 & 0 & 0 & 0 & 1 & 0 \\
C & 0 & 0 & 0 & 1 & 0 & 0 \\
D & 1 & 0 & 1 & 0 & 1 & 1 \\
E & 0 & 1 & 0 & 1 & 0 & 0 \\
F & 1 & 0 & 0 & 1 & 0 & 0
\end{array}
$$

adjacency list for G

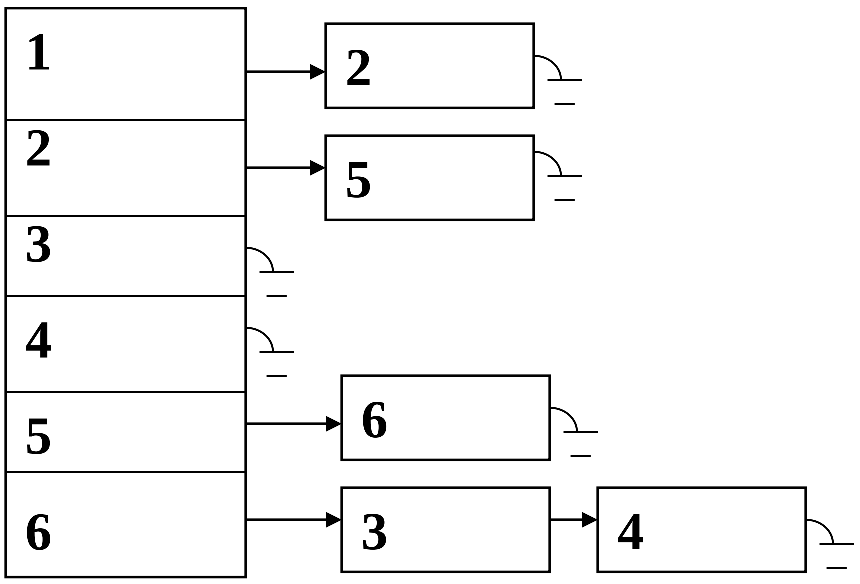


Adjacency Lists

Directed Graphs:



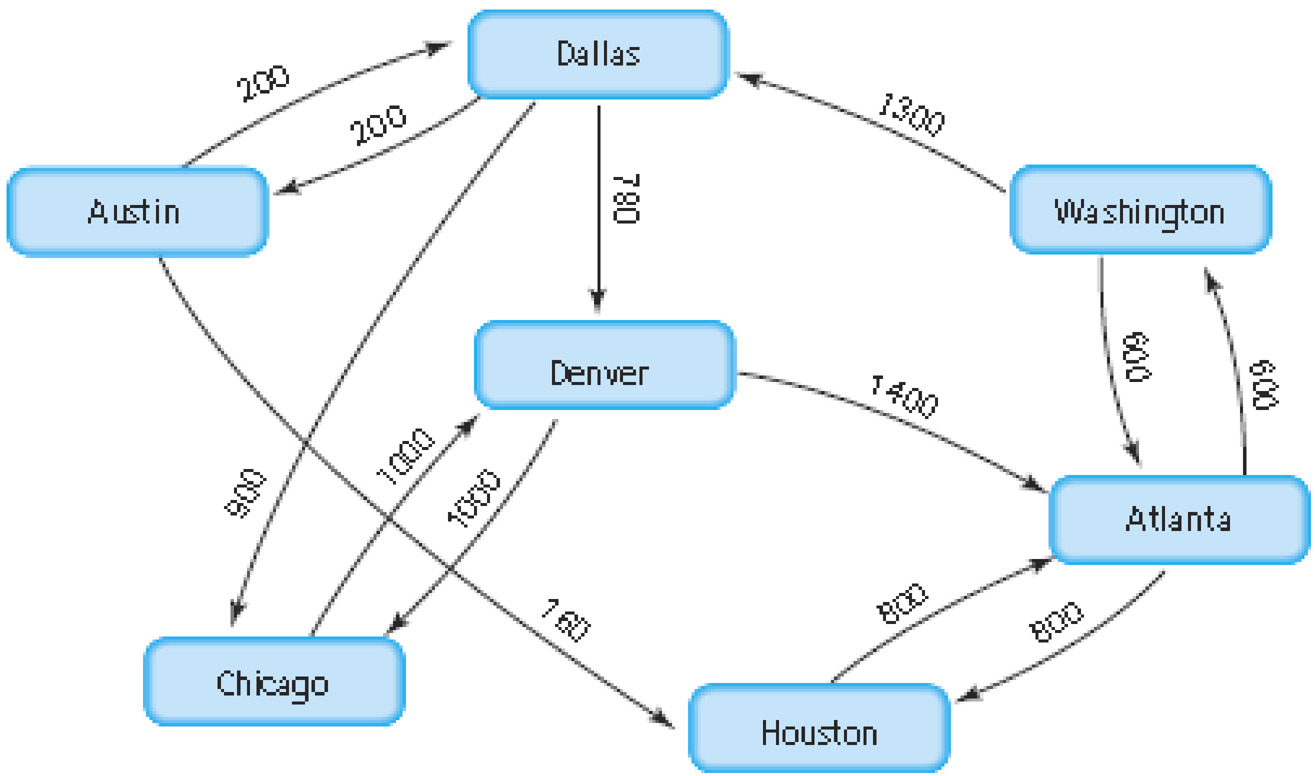
Graph



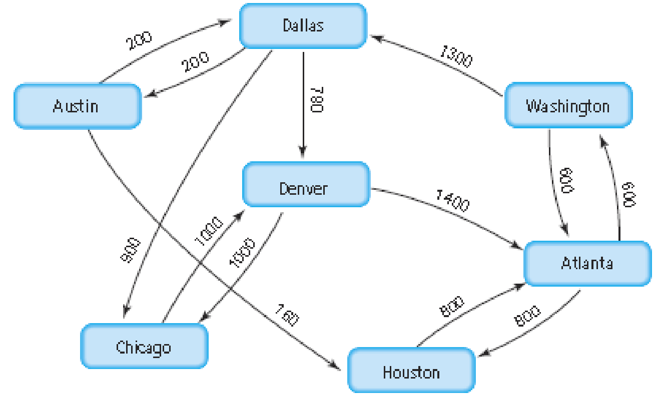
Adjacency List

Adjacency Lists

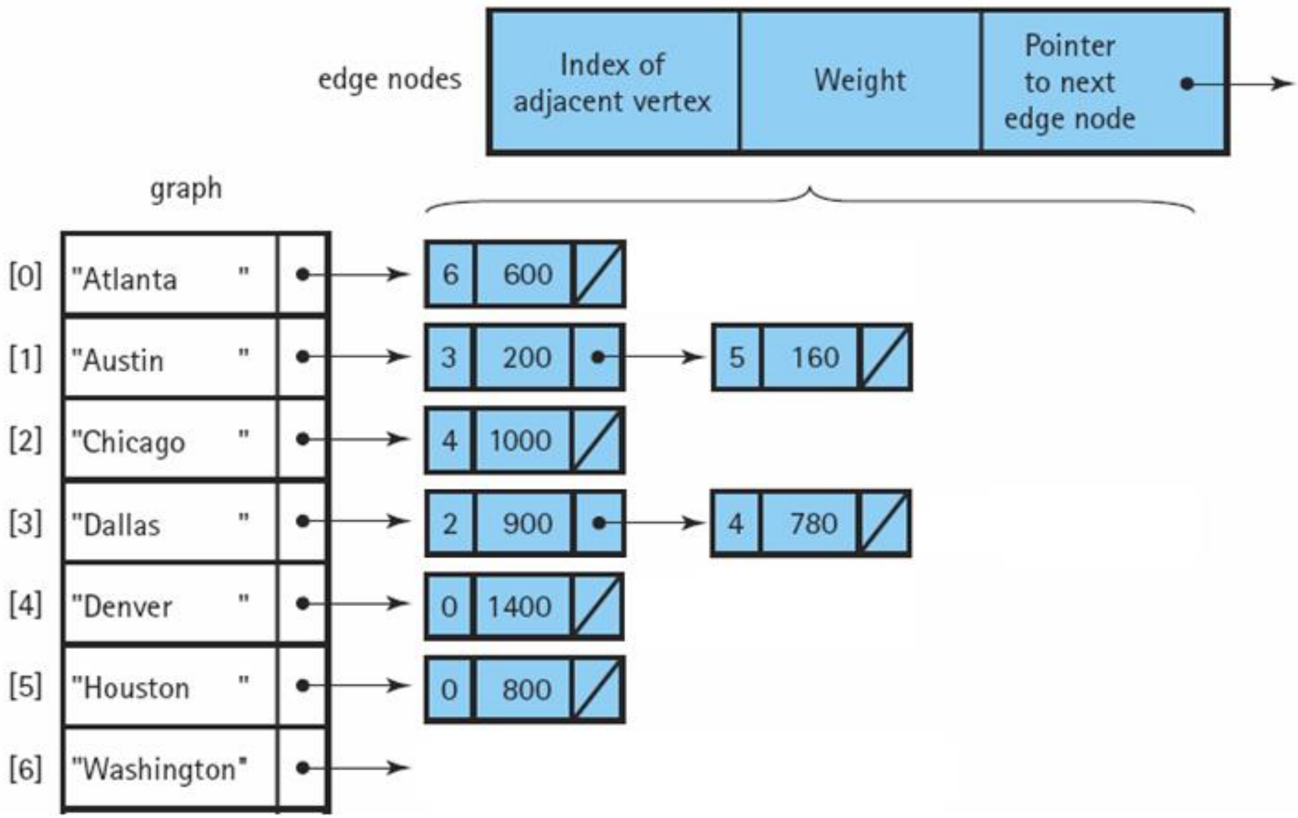
Weighted Directed Graphs:

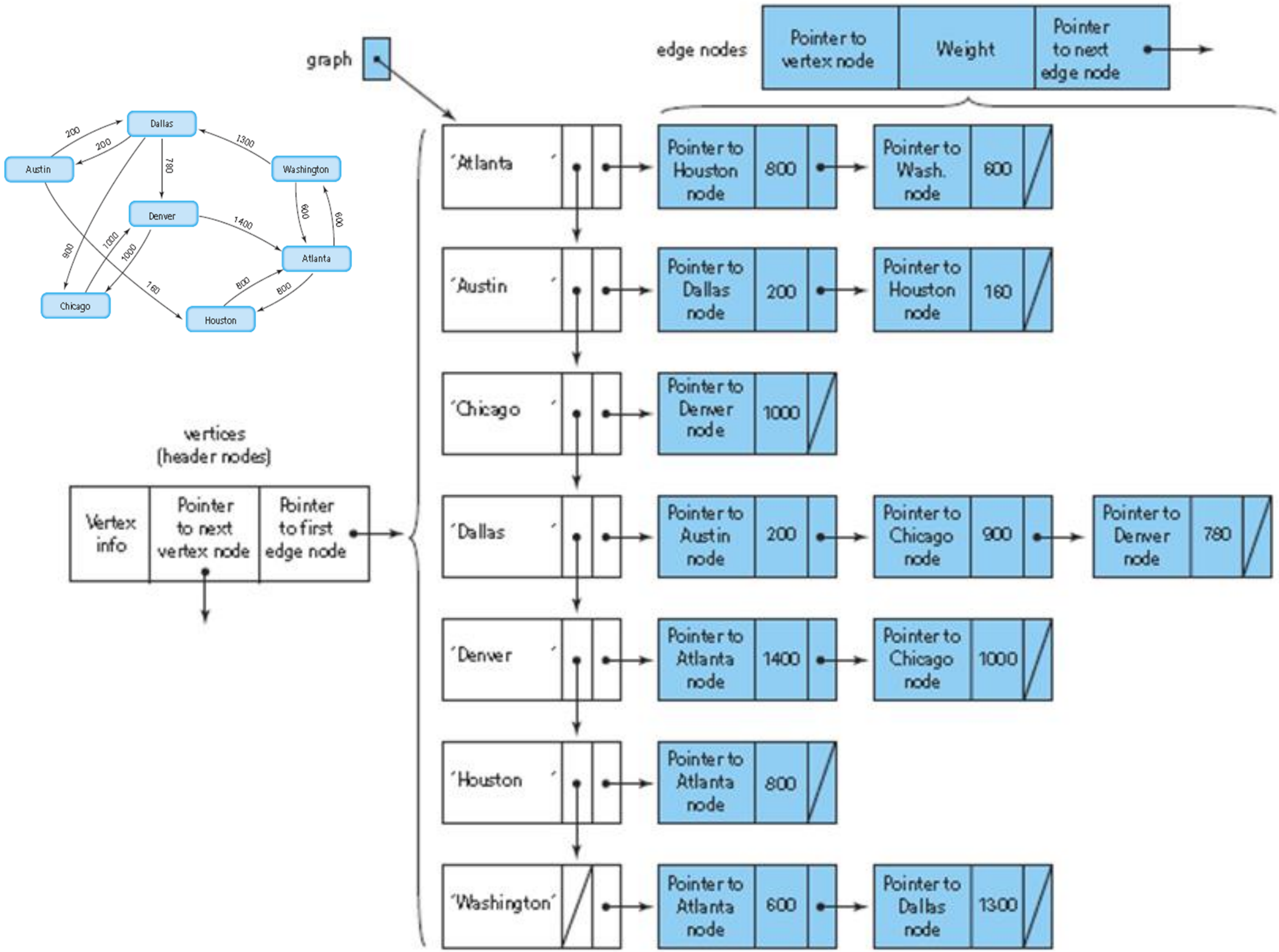


Adjacency Lists



Weighted Directed Graphs:




graph
edge nodes
Pointer to vertex node
Weight
Pointer to next edge node
Dallas
Austin
Washington
Denver
Atlanta
Chicago
Houston
200
200
1300
780
1400
600
600
900
1000
1000
160
800
800
vertices (header nodes)
Vertex info
Pointer to next vertex node
Pointer to first edge node
'Atlanta'
Pointer to Houston node
800
Pointer to Wash. node
600
'Austin'
Pointer to Dallas node
200
Pointer to Houston node
160
'Chicago'
Pointer to Denver node
1000
'Dallas'
Pointer to Austin node
200
Pointer to Chicago node
900
Pointer to Denver node
780
'Denver'
Pointer to Atlanta node
1400
Pointer to Chicago node
1000
'Houston'
Pointer to Atlanta node
800
'Washington'
Pointer to Atlanta node
600
Pointer to Dallas node
1300

Adjacency Matrix

- **Pros:**
  - Simple to implement
  - Easy and fast to tell if a pair (i,j) is an edge:
    - simply check if A[i][j] is 1 or 0
- **Cons:**
  - No matter how few edges the graph has, the matrix takes $\mathbf{O(n^2)}$ in memory

Adjacency Lists

Pros:

- **Save space (memory):** the representation takes as many memory words as there are nodes and edges.
- For both directed and undirected graphs, the adjacency-list representation has the desirable property that **the amount of memory** it requires, is O(V+E).

Cons:

- *It can take up to O(n) time to determine if a pair of nodes (i,j) has an edge:*
  - One would have to search the linked list L[i], which takes time proportional to the length of L[i].

Graph … Implementation

- Operations: The basic functions of graph
  - **MakeEmpty** →
    - *Initializes the graph to an empty state.*
  - **Boolean IsEmpty** →
    - *Tests whether the graph is empty*
  - **Boolean IsFull** →
    - *Tests whether the graph is full.*
  - **AddVertex**(VertexType vertex) →
    - *Add vertex to graph*

Graph … Implementation

- Operations: The basic functions of graph
  - **AddEdge** (VertexType fromVertex, VertexType toVertex, EdgeValueType weight) → *Adds an edge with the specified weight from fromVertex to toVertex.*
  - **EdgeValueType Weights** (VertexType fromVertex, VertexType toVertex) → *Determines the weight of the edge from fromVertex to toVertex.*
  - **GetToVertices** (VertexType vertex, QueType& vertexQ) →Returns a queue of the vertices that are adjacent from vertex.

Reading Materials

- Nell Dale: Chapter # 9