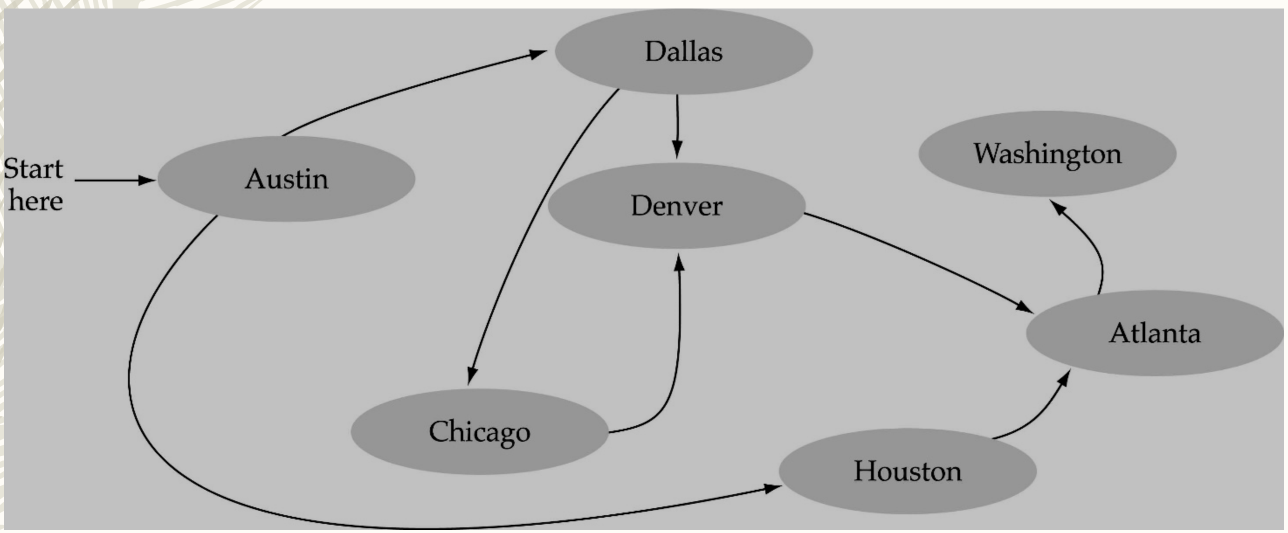
Graphs Data Structure

What is a graph?

– A data structure that consists of a set of nodes (*vertices*) and a set of edges that relate the nodes to each other

– The set of edges describes relationships among the vertices



Formal definition of graphs

- A graph *G* is defined as follows:

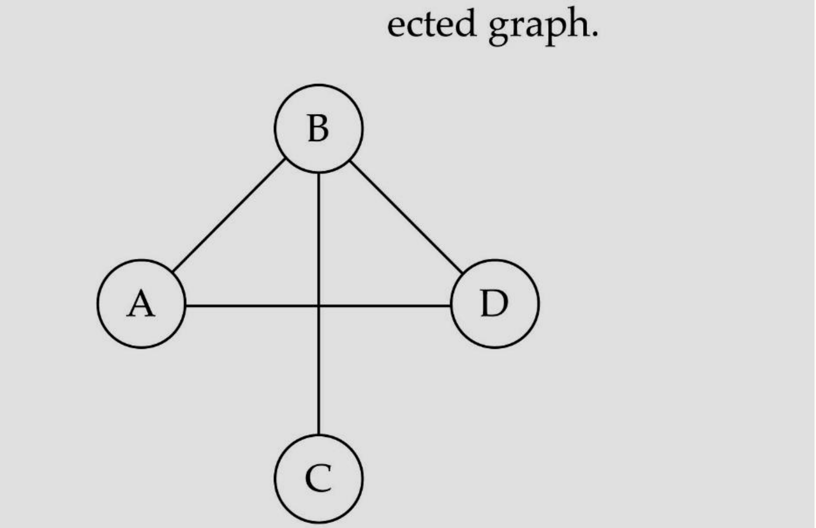
$$G=(V,E)$$

V(G): a finite, nonempty set of vertices

E(G): a set of edges (pairs of vertices)

Directed vs. undirected graphs

- When the edges in a graph have no direction, the graph is called *undirected*

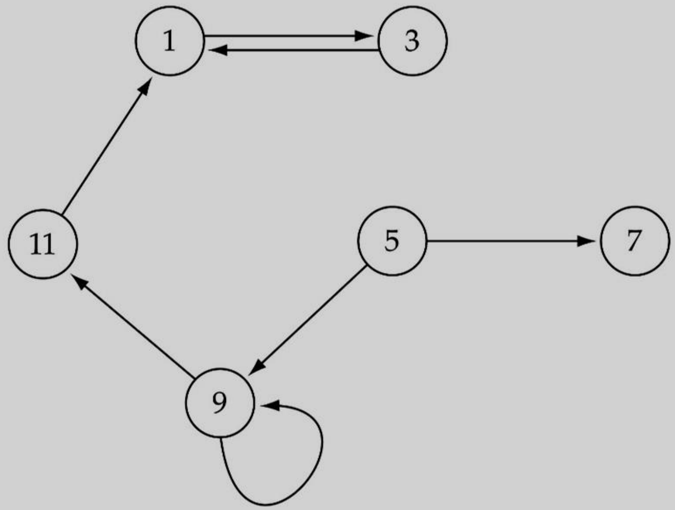


V(Graph1) = { A, B, C, D }
E(Graph1) = { (A, B), (A, D), (B, C), (B, D) }

Directed vs. undirected graphs (cont.)

- When the edges in a graph have a direction, the graph is called *directed* (or *digraph*)

(b) Graph2 is a directed graph.

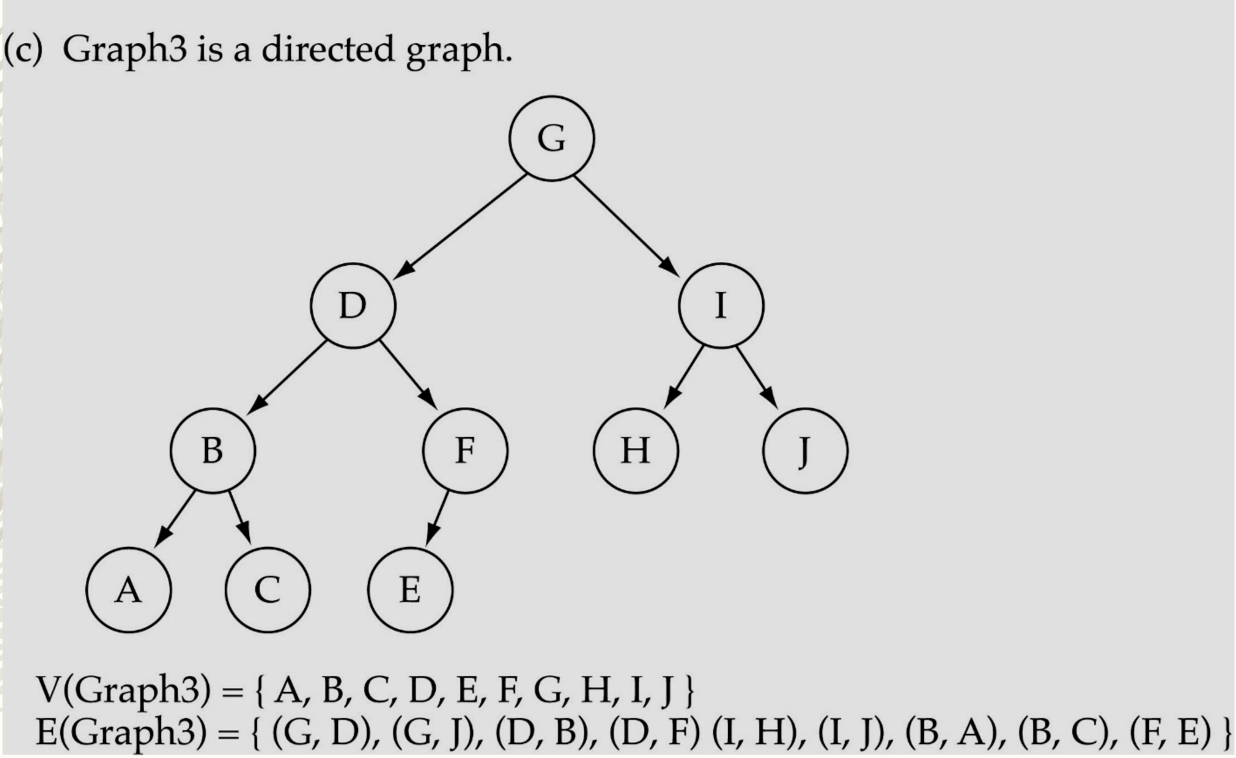


V(Graph2) = { 1, 3, 5, 7, 9, 11 }
E(Graph2) = {(1,3) (3,1) (5,9) (9,11) (5,7), (9, 9), (11, 1) }

Warning: if the graph is directed, the order of the vertices in each edge is important !!

Trees vs graphs

– Trees are special cases of graphs!!

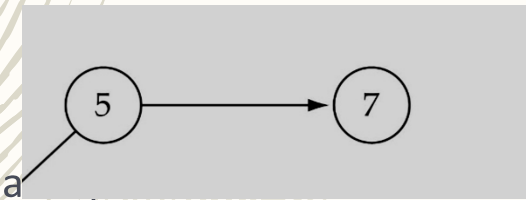


DiGraph terminology

- A directed graph (G) is therefore a pair (V,E) where E is a binary relation on V
  - The set V is called **Vertex set** of G (Vertices)
  - the set E is called **Edge set** of G (edges)
- Vertices are represented by circles and edges by arrows.
- **Self-loops** are edges from a vertex to itself

DiGraph terminology

- Adjacent nodes: two nodes are adjacent if they are connected by an edge



5 is adjacent to 7
7 is adjacent from 5

- Path: a [illegible] onnect two nodes in a graph
- Length of a path is the number of edges in the path
- Out-degree of a vertex is the number of edges leaving it and in-degree is the number entering it
- Degree of a vertex =out-degree + in-degree
- Complete graph: a graph in which every vertex is directly connected to every other vertex
- Other terms: reachable, subpath, simple, cycle

Undirected Graph terminology

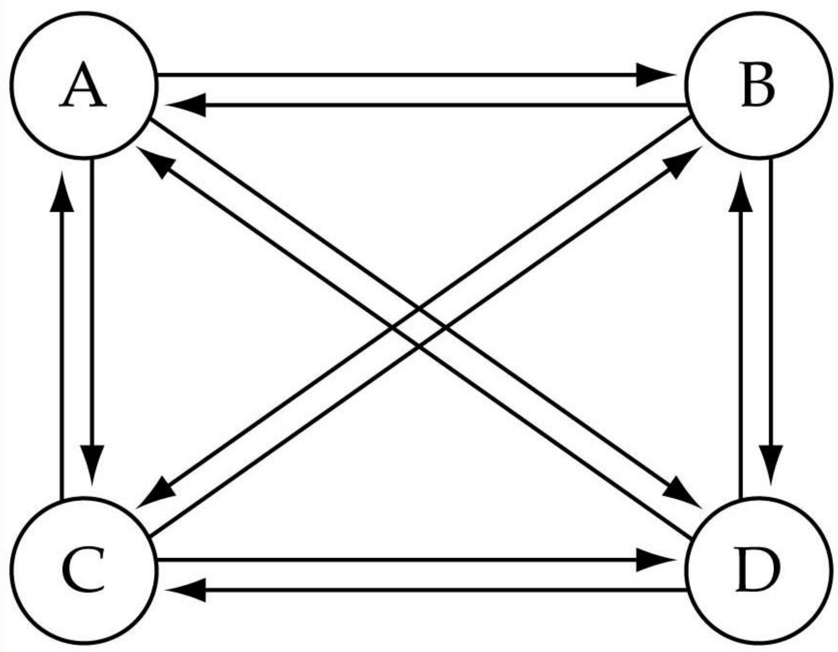
- An Undirected graph (G)= (V,E),
  - E consists of unordered pairs therefore we write (u,v) instead of {u,v} and (u,v) is considered same as (v,u)
- Self-loops are forbidden
- Adjacent relation is symmetric
- Given an edge (u,v) then (u,v) is <u>incident on</u> vertices u and v
- <u>Degree</u> of a vertex is the number of edges incident on it
- A vertex with degree 0 is said to be <u>isolated</u>
- <u>Connected</u> graph – every pair of vertices is connected by a path
- A connected, acyclic graph is called a <u>(free) tree;</u> it is <u>a forest</u> if not connected
- A <u>complete graph</u> – every pair of vertices is adjacent

Graph terminology (cont.)

– What is the number of edges in a complete directed graph with N vertices?

*N * (N-1)*

$O(N^2)$



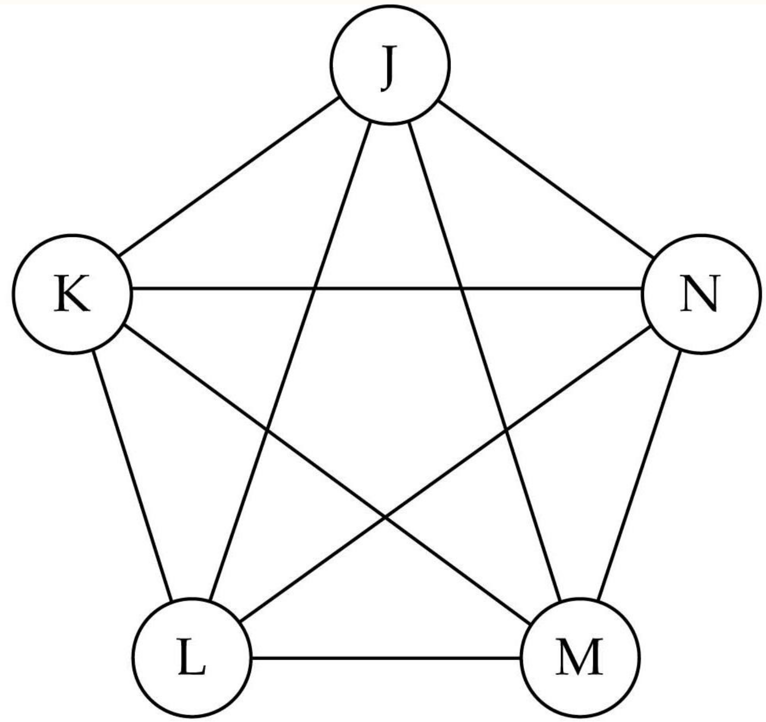
(a) Complete directed graph.

Graph terminology (cont.)

– What is the number of edges in a complete undirected graph with N vertices?

*N * (N-1) / 2*

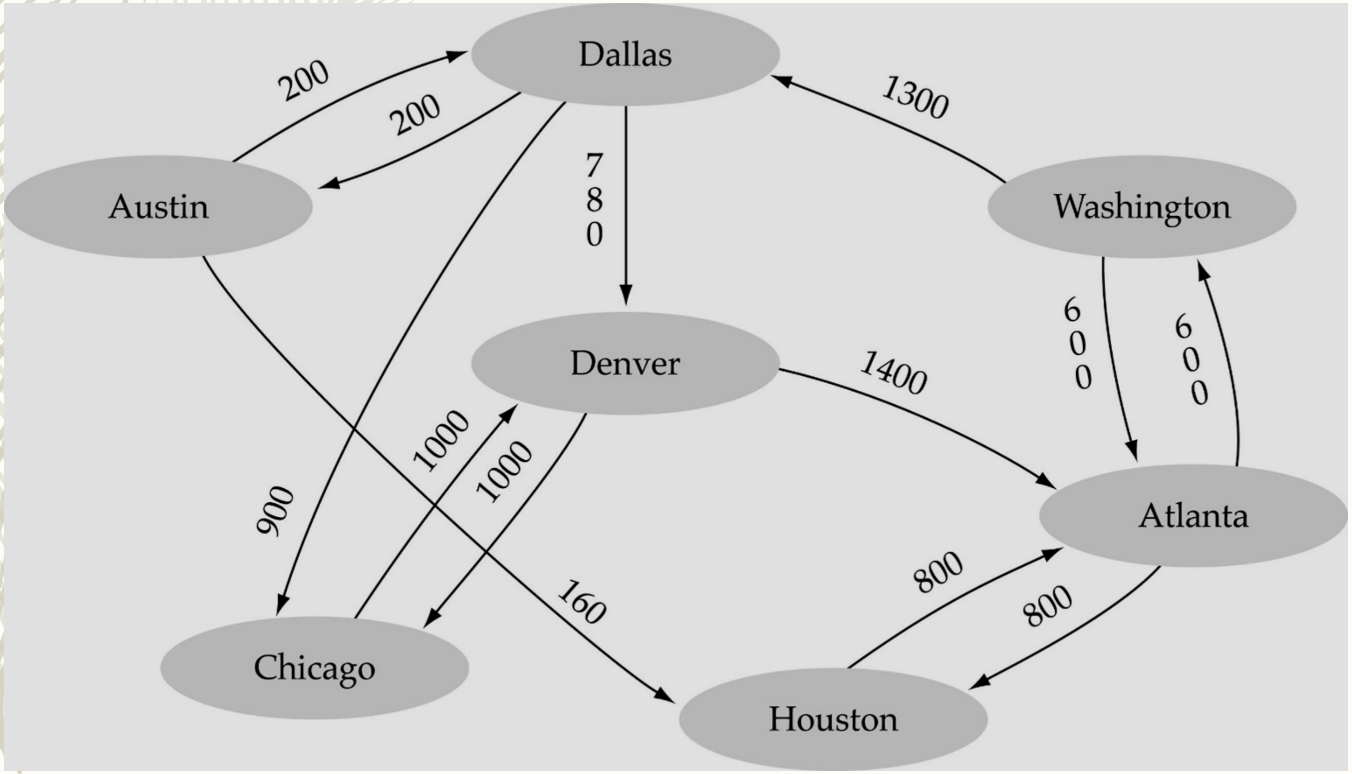
$O(N^2)$



(b) Complete undirected graph.

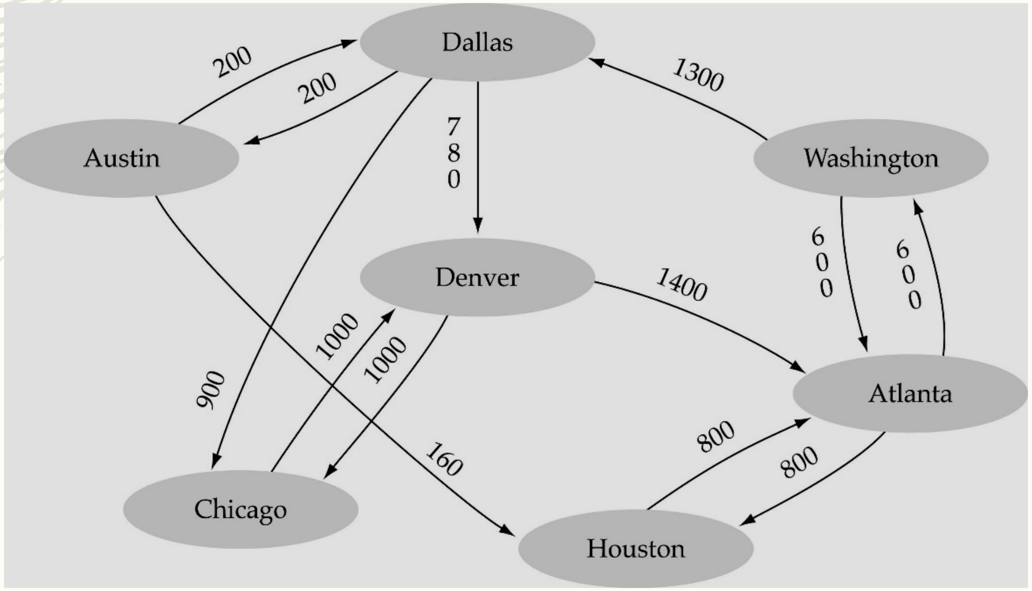
Graph terminology (cont.)

– Weighted graph: a graph in which each edge carries a value



Graph implementation

- <u>Array-based implementation</u>
  - A 1D array is used to represent the vertices
  - A 2D array (adjacency matrix) is used to represent the edges



Array-based implementation

graph

.numVertices 7

.vertices

[0]	"Atlanta "
[1]	"Austin "
[2]	"Chicago "
[3]	"Dallas "
[4]	"Denver "
[5]	"Houston "
[6]	"Washington"
[7]	
[8]	
[9]	

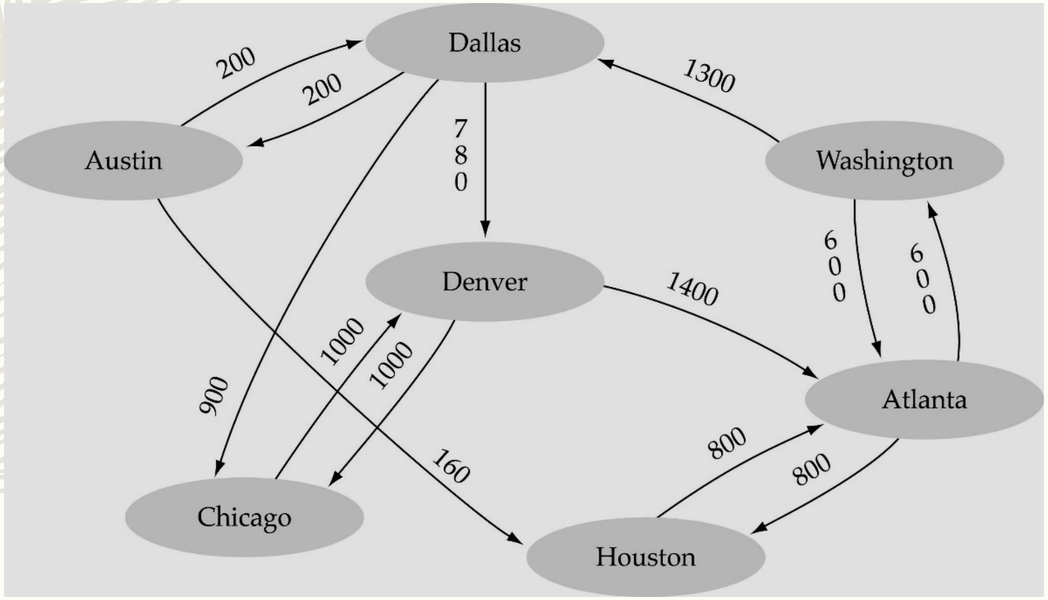
.edges

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[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	•	•	•
[7]	•	•	•	•	•	•	•	•	•	•
[8]	•	•	•	•	•	•	•	•	•	•
[9]	•	•	•	•	•	•	•	•	•	•

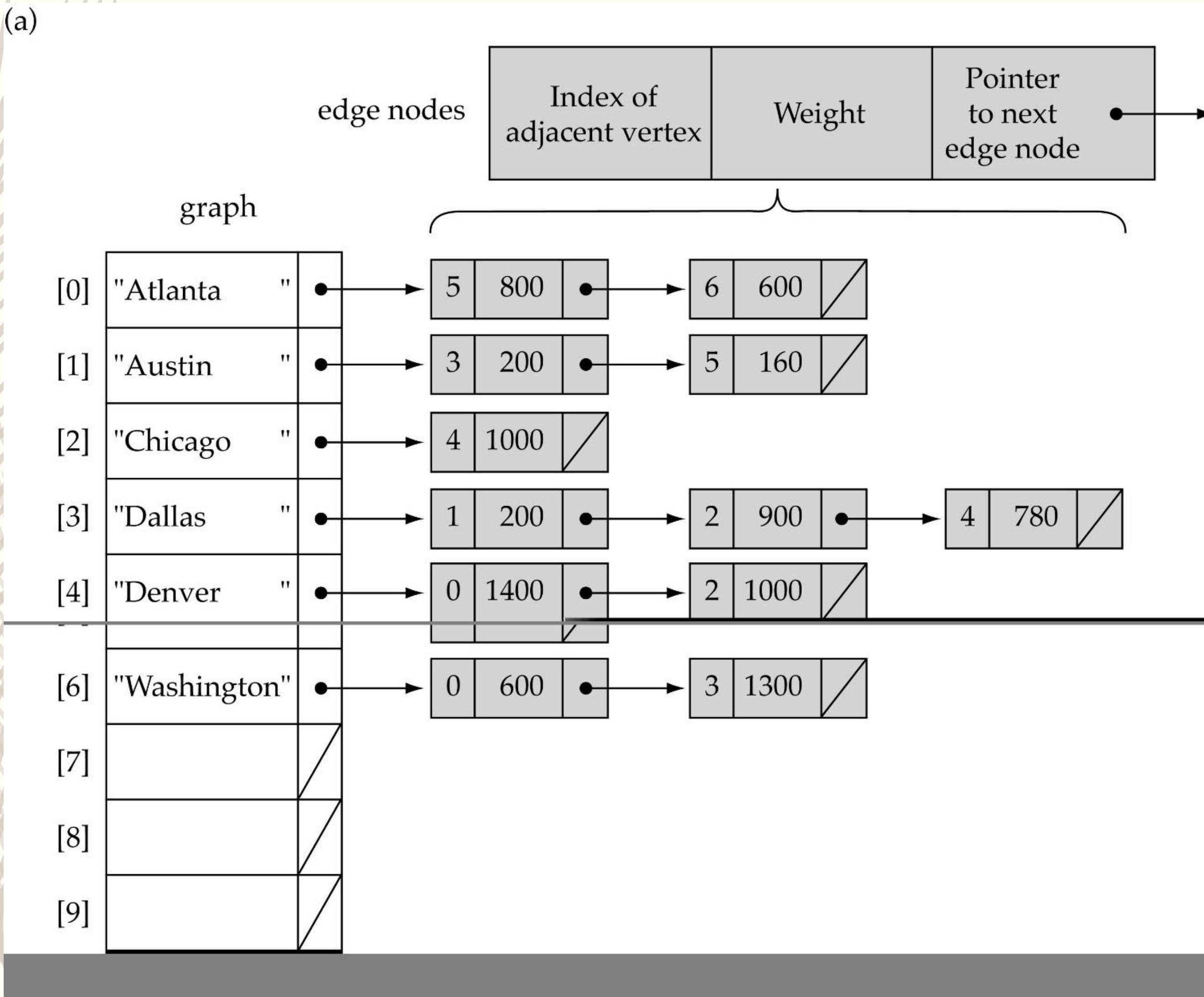
(Array positions marked '•' are undefined)

Graph implementation (cont.)

- Linked-list implementation
  - A 1D array is used to represent the vertices
  - A list is used for each vertex *v* which contains the vertices which are adjacent to *v* (adjacency list)



Linked-list implementation



Adjacency matrix vs. adjacency list representation

– **Adjacency matrix**

  – Good for dense graphs

  Connectivity between two vertices can be tested quickly

– **Adjacency list**

  – Good for sparse graphs

  Vertices adjacent to another vertex can be found quickly

Q. Which of the two do you think requires more memory?

Graph searching

- *Problem*: find a path between two nodes of the graph (e.g., Austin and Washington)
- *Methods*: Depth-First-Search (DFS) or Breadth-First-Search (BFS)

Depth-First-Search (DFS)

- What is the idea behind DFS?
  - Travel as far as you can down a path
  - Back up *as little as possible* when you reach a "dead end" (i.e., next vertex has been "marked" or there is no next vertex)
- DFS can be implemented efficiently using a *stack*

Breadth-First-Searching (BFS)

- What is the idea behind BFS?
  - Look at all possible paths at the same depth before you go at a deeper level
  - Back up *as far as possible* when you reach a "dead end" (i.e., next vertex has been "marked" or there is no next vertex)
- We will be looking at BFS and DFS later in the course

Single-source shortest-path problem

- There are multiple paths from a source vertex to a destination vertex
- *Shortest path*: the path whose total weight (i.e., sum of edge weights) is minimum
- Examples:
  - Austin->Houston->Atlanta->Washington: 1560 miles
  - Austin->Dallas->Denver->Atlanta->Washington: 2980 miles

Single-source shortest-path problem (cont.)

- Common algorithms: *Dijkstra's* algorithm and *Bellman-Ford* algorithm
- BFS can be used to solve the shortest graph problem when the graph is **weightless** or all the weights are the same

Rooted Trees

- A free tree in which one of the vertices is distinguished from others; the root
- Terms like ancestor, descendant, parent, child, siblings leaf (external node) and internal node are used