Data Structures and Algorithms

LECTURE 07: GRAPHS AND SHORTEST PATHS

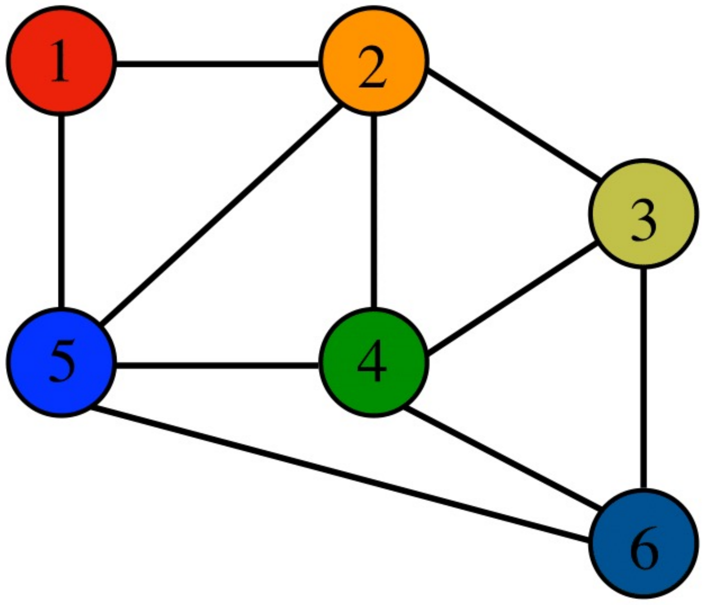
UNIVERSITY of GREENWICH

Alliance with FPT Education

Pearson BTEC

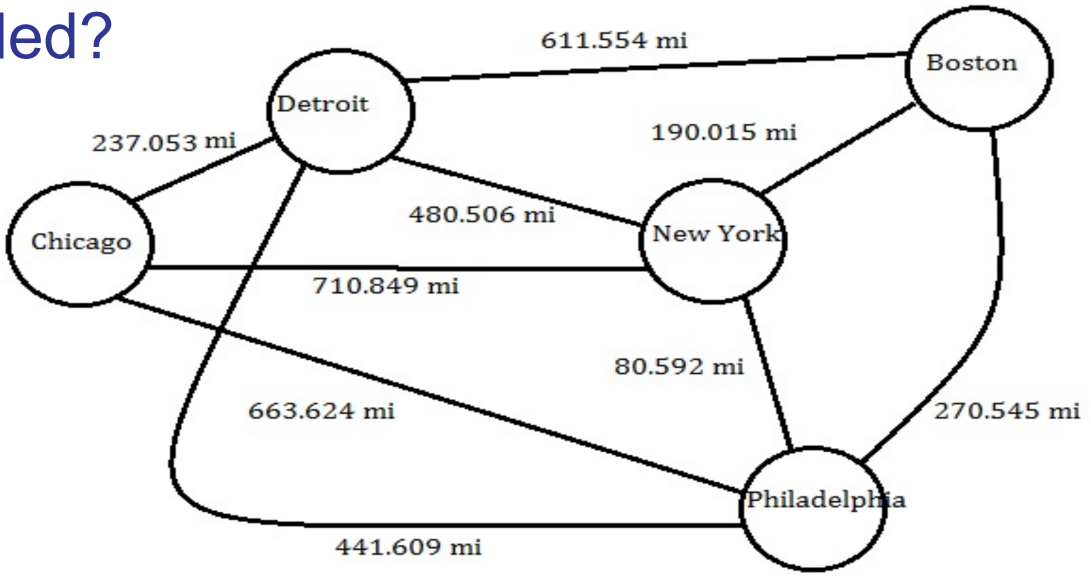
Contents

Graphs



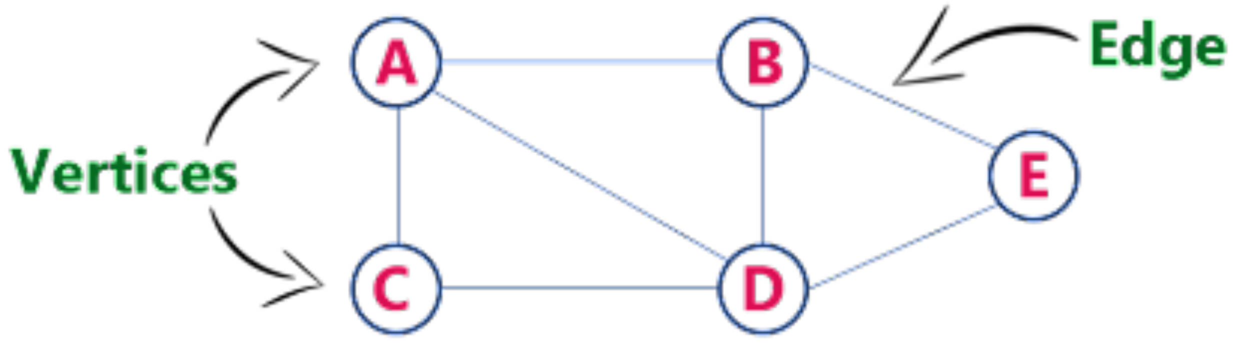
Graphs

A salesman spends his time visiting *n* cities (or nodes) cyclically. In one tour he visits each city just once, and finishes up where he started. In what order should he visit them to minimize the distance travelled?



Graphs: Definition

- 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



Graphs: Formal Definition

- A graph *G* is defined as follows:

$$G = (V, E)$$

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

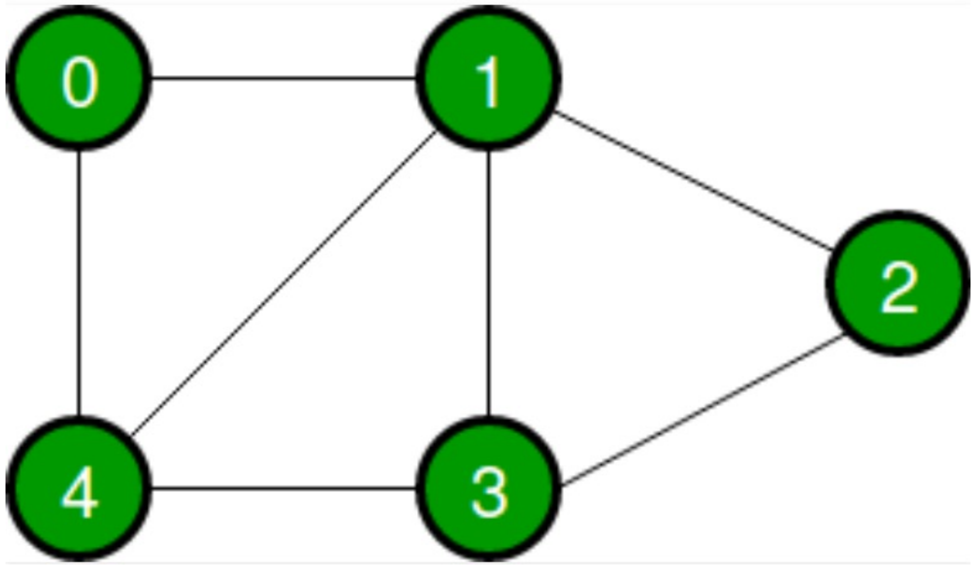
E(G): a set of edges (pairs of vertices)

Graphs

- Types of graphs:
  - UnDirected Graphs
  - Directed Graphs
  - Weighted Graphs

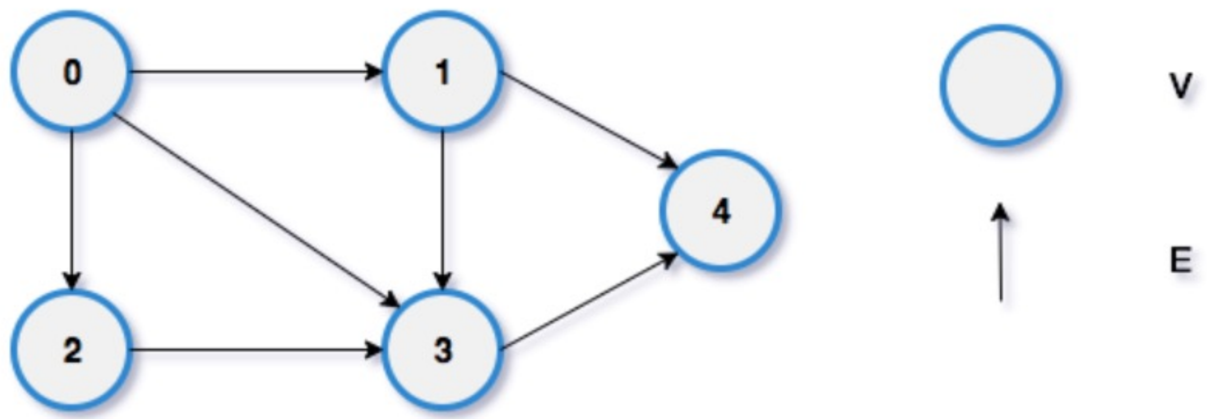
Graphs: UnDirected Graph

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



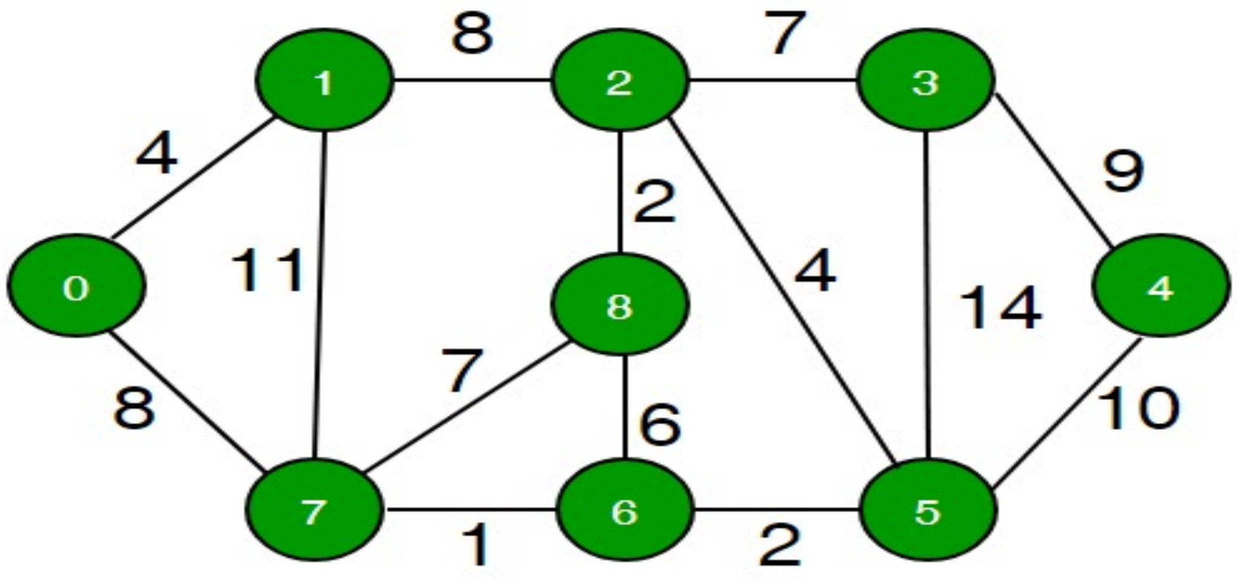
Graphs: Directed Graph

- When the edges in a graph have a direction, the graph is called directed (or digraph)
- Warning: if the graph is directed, the order of the vertices in each edge is important!



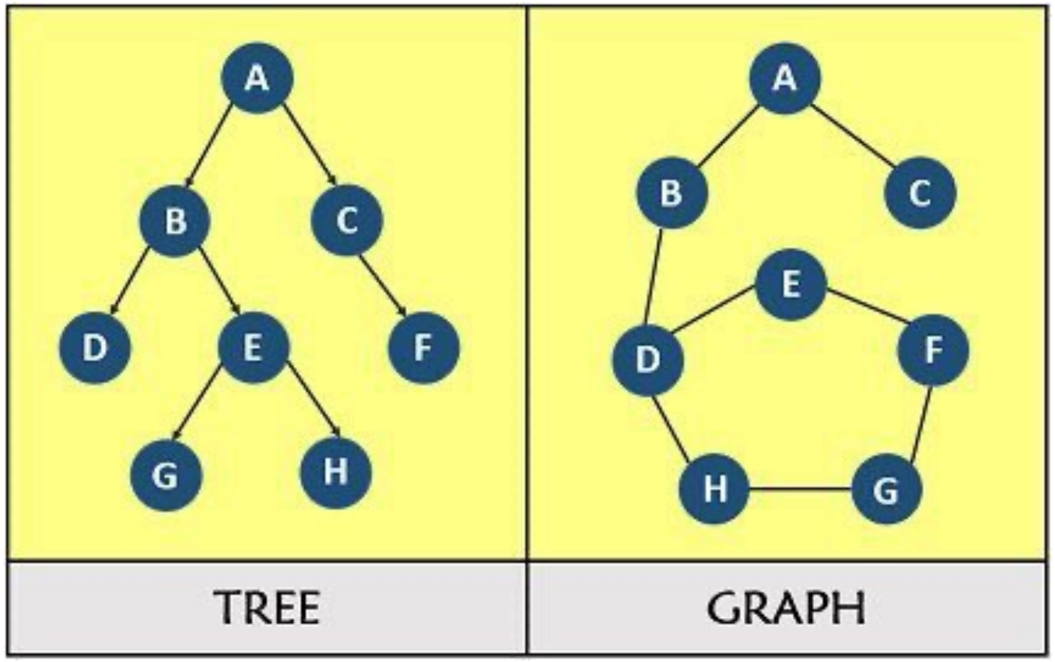
Graphs: Weighted Graph

- A graph in which each edge carries a value (is called **weight** or **cost**). Some graphs allow negative values.



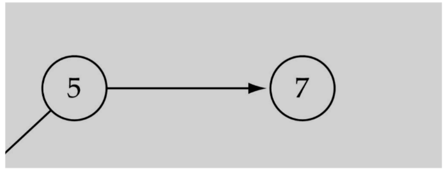
Graphs vs Trees

- Trees are special cases of graphs



Graph Terminology

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



5 is adjacent to 7
7 is adjacent from 5

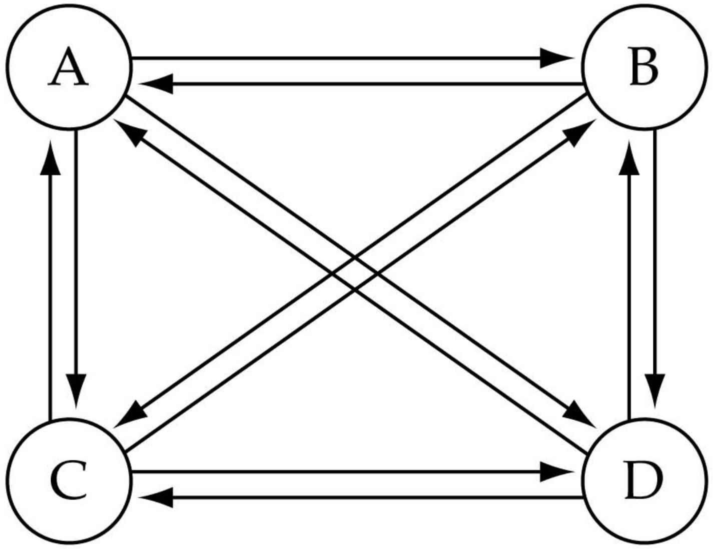
- Path: A sequence of edges that allows you to go from vertex A to vertex B is called a path

Graph Terminology

- Complete graph: a graph in which every vertex is directly connected to every other vertex

What is the number of edges in a complete directed graph with N vertices? *N * (N-1)*

$$O(N^2)$$



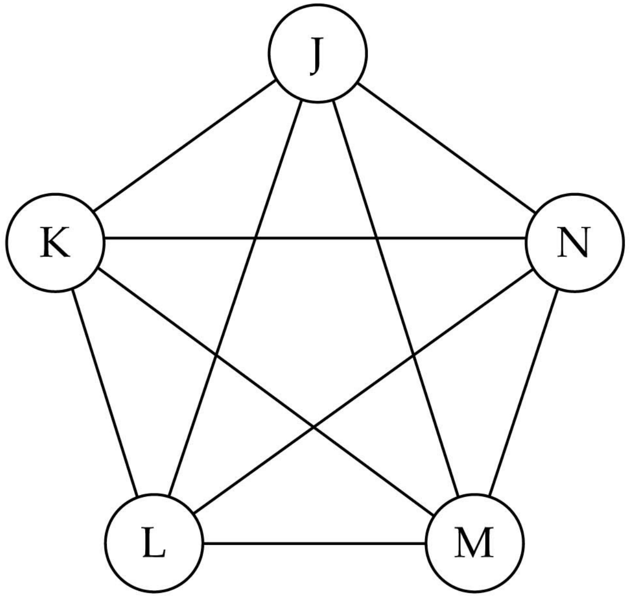
(a) Complete directed graph.

Graph Terminology

- Complete graph: a graph in which every vertex is directly connected to every other vertex

What is the number of edges in a complete undirected graph with N vertices? *N * (N-1) / 2*

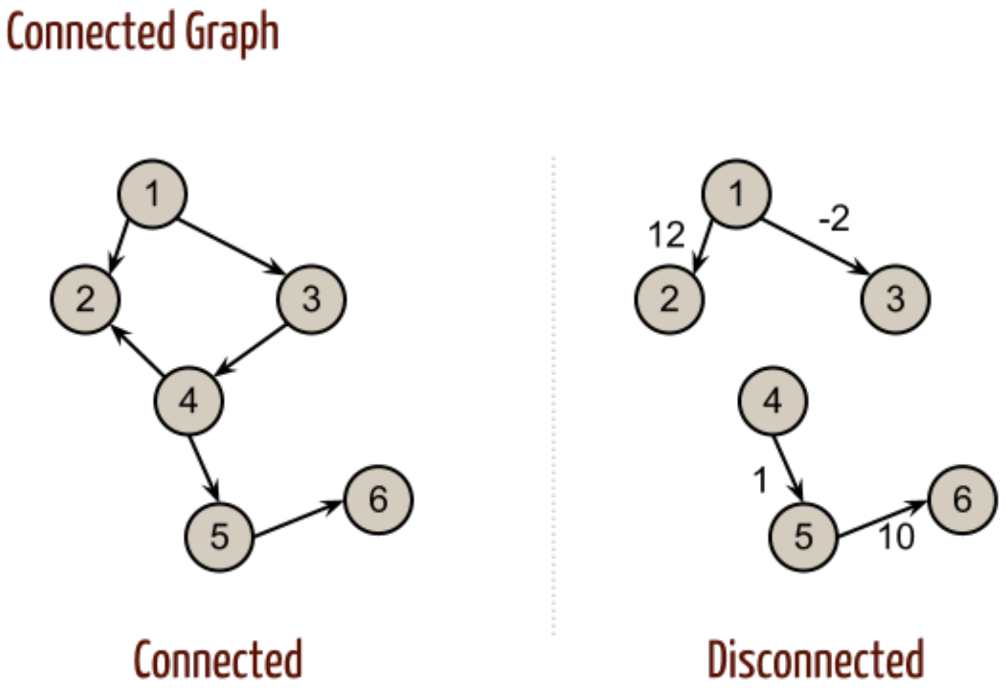
$$O(N^2)$$



(b) Complete undirected graph.

Graph Terminology

- A graph is said to be connected if there is at least one path from every vertex to every other vertex.
- A non-connected (disconnected) graph consists of several connected components.

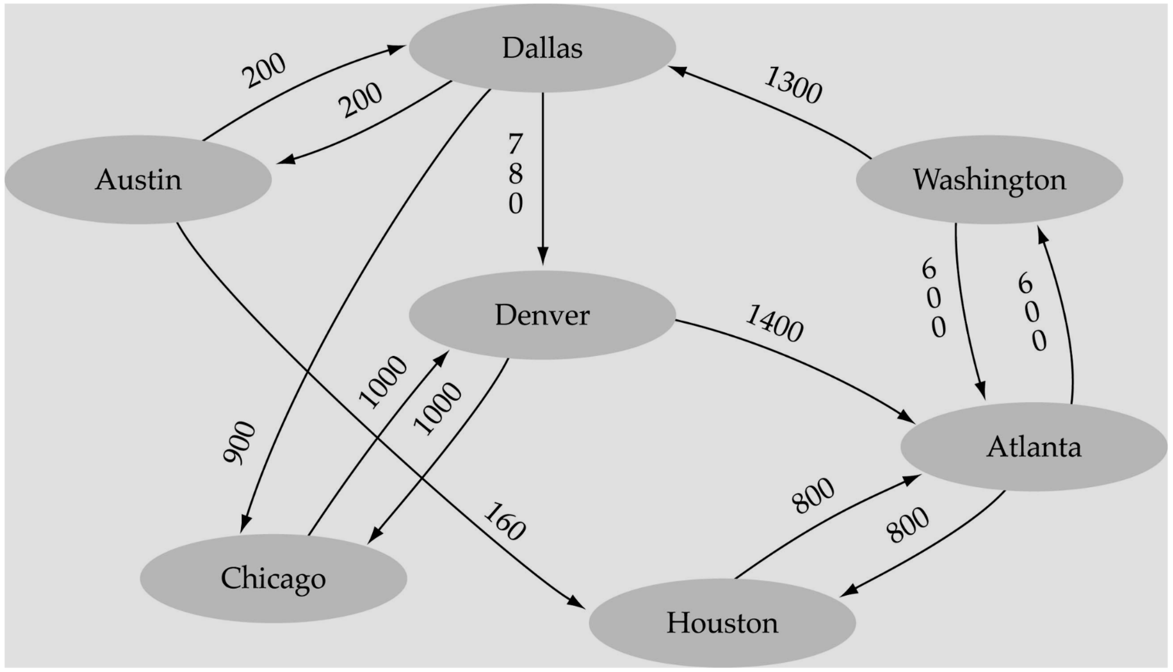


Graph Implementation

- Array-based implementation:
  - – A 1D array is used to represent the vertices
  - – A 2D array (adjacency matrix) is used to represent the edges

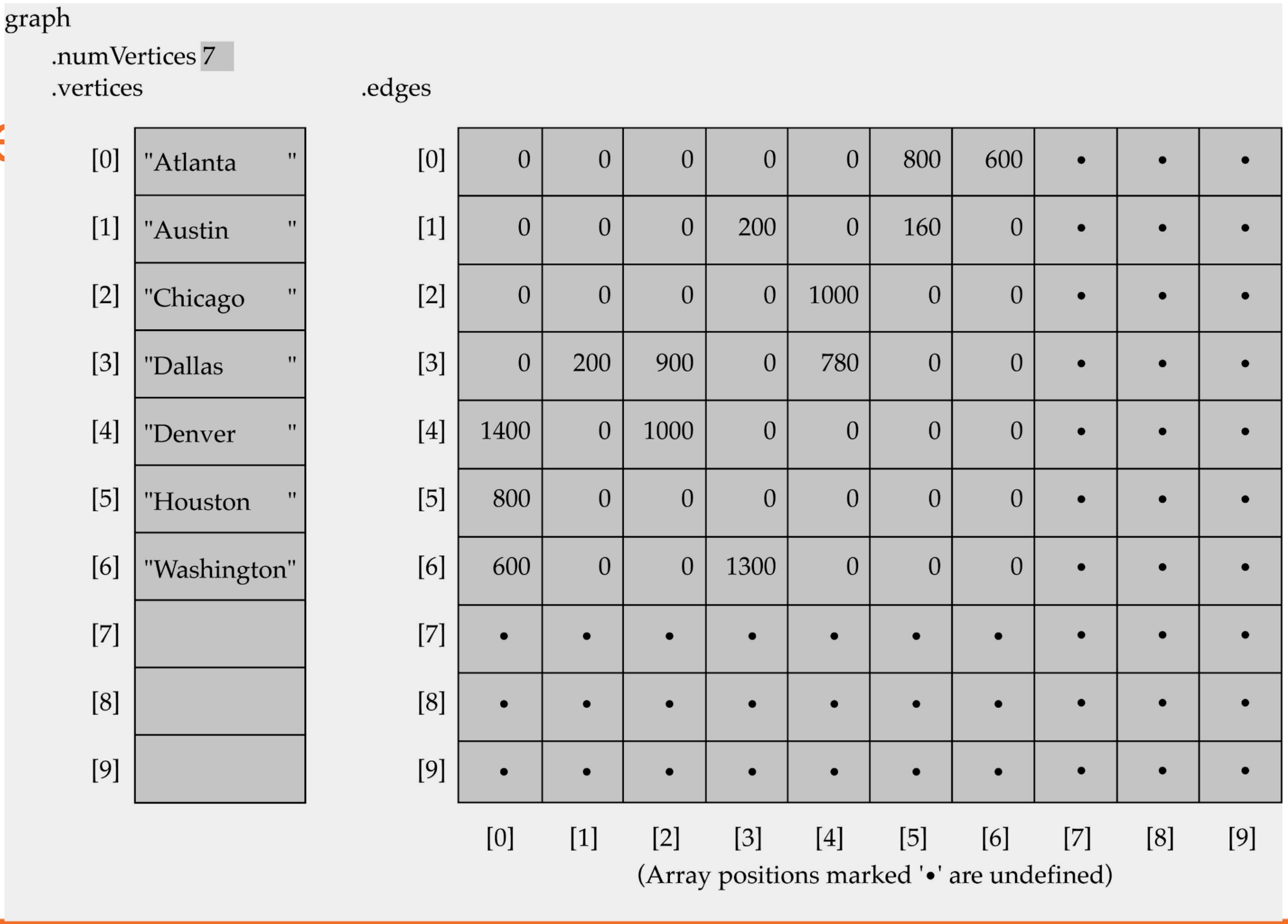
Graph Implementation

- Array-based implementation:



Graph Implementation

- Array-base



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]	•	•	•	•	•	•	•	•	•	•

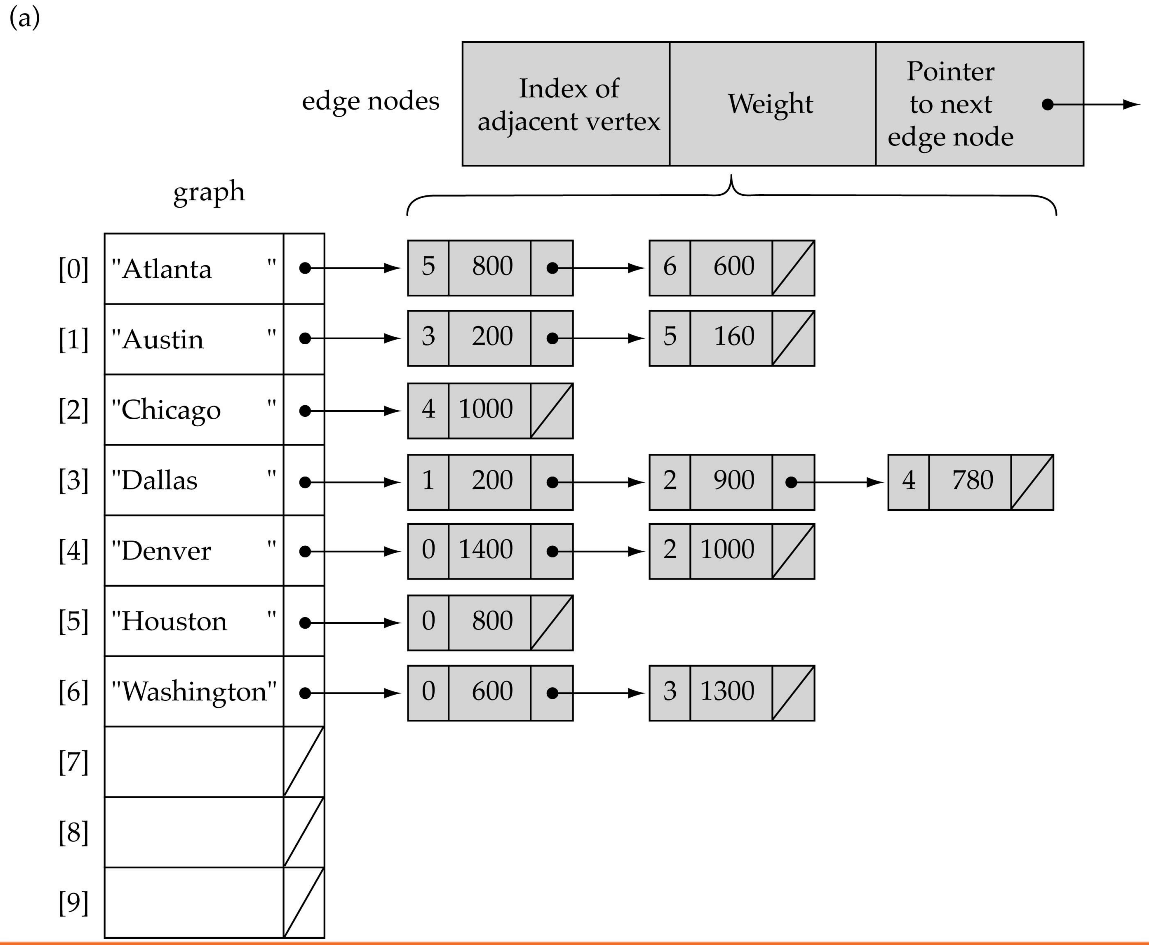
(Array positions marked '•' are undefined)

Graph Implementation

- 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 from v (adjacency list)

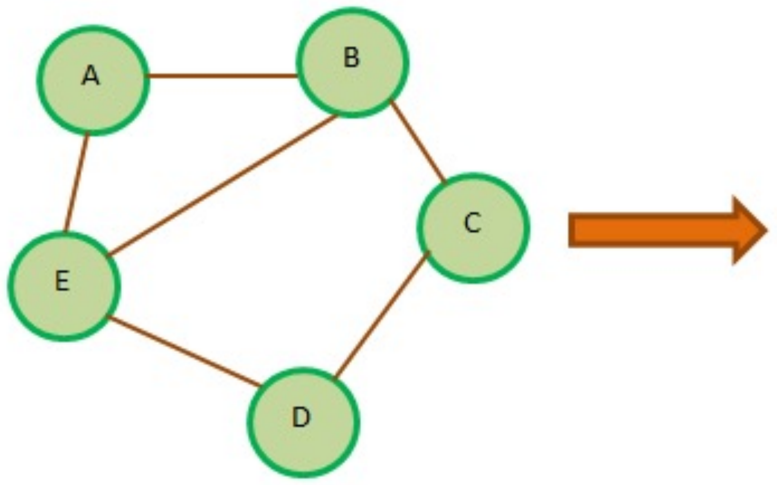
Graph Implementation

- Linked-list



Graph Implementation

- Examples: adjacency matrix



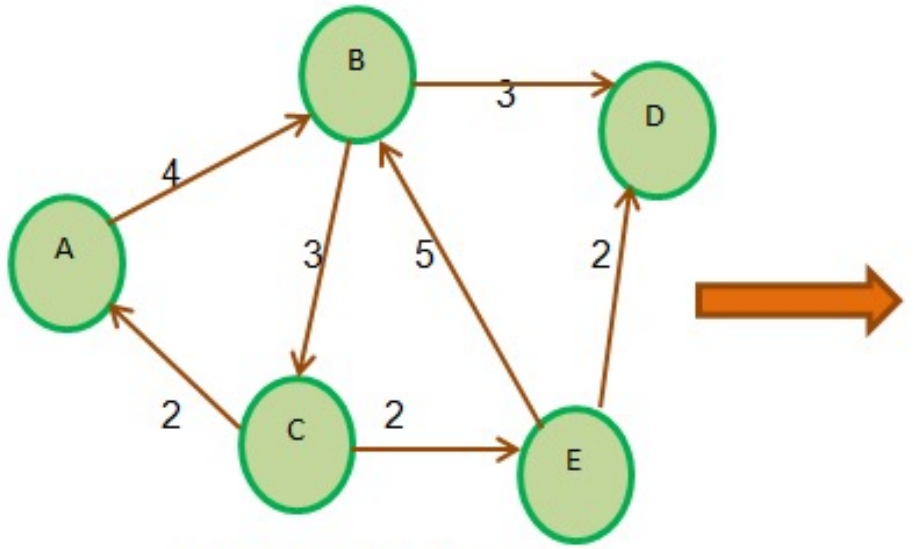
Undirected graph

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

Adjacency Matrix

Graph Implementation

- Examples: adjacency matrix



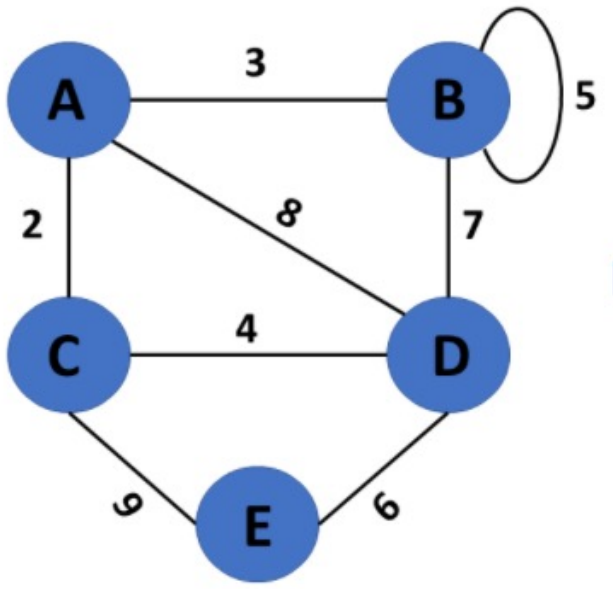
Directed weighted graph

$$
\begin{array}{c|ccccc}
 & A & B & C & D & E \\
A & 0 & 4 & 0 & 0 & 0 \\
B & 0 & 0 & 3 & 3 & 0 \\
C & 2 & 0 & 0 & 0 & 2 \\
D & 0 & 0 & 0 & 0 & 0 \\
E & 0 & 5 & 0 & 2 & 0
\end{array}
$$

adjacency matrix

Graph Implementation

- Examples: adjacency matrix



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 3 & 2 & 8 & 0 \\ B & 3 & 5 & 0 & 7 & 0 \\ C & 2 & 0 & 0 & 4 & 9 \\ D & 8 & 7 & 4 & 0 & 6 \\ E & 0 & 0 & 9 & 6 & 0 \end{array}$$

Applications of Graph

- Social Graphs: Facebook's Graph API
- Knowledge Graphs: Google's Knowledge Graph
- Recommendation Engines: Yelp's Local Graph
- Path Optimization Algorithms:
  - Google Maps Platform
  - Flight Networks
  - GPS Navigation Systems
- Scientific Computations

Graph Traversals

- Fundamental operation, to find out which vertices can be reached from a specified vertex.
- An algorithm that provides a systematic way to start at a specified vertex and then move along edges to other vertex.
- Every vertex that is connected to this vertex has to be visited at the end.

Graph Traversals

- Two common approaches:
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)

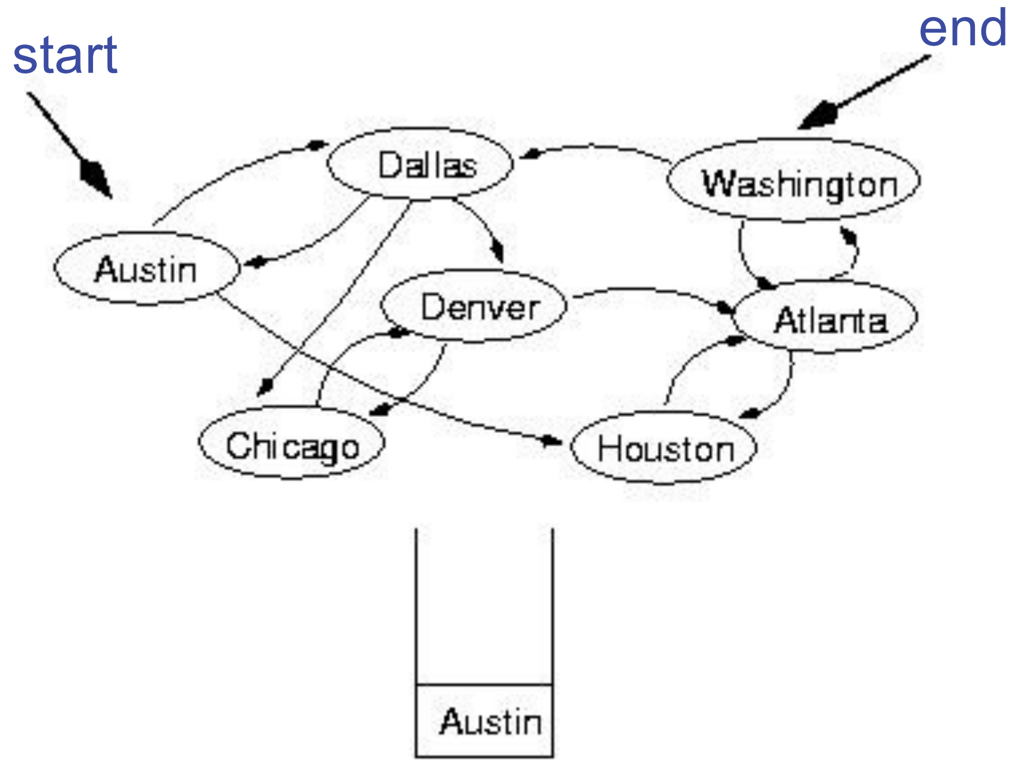
Graph Traversals: DFS

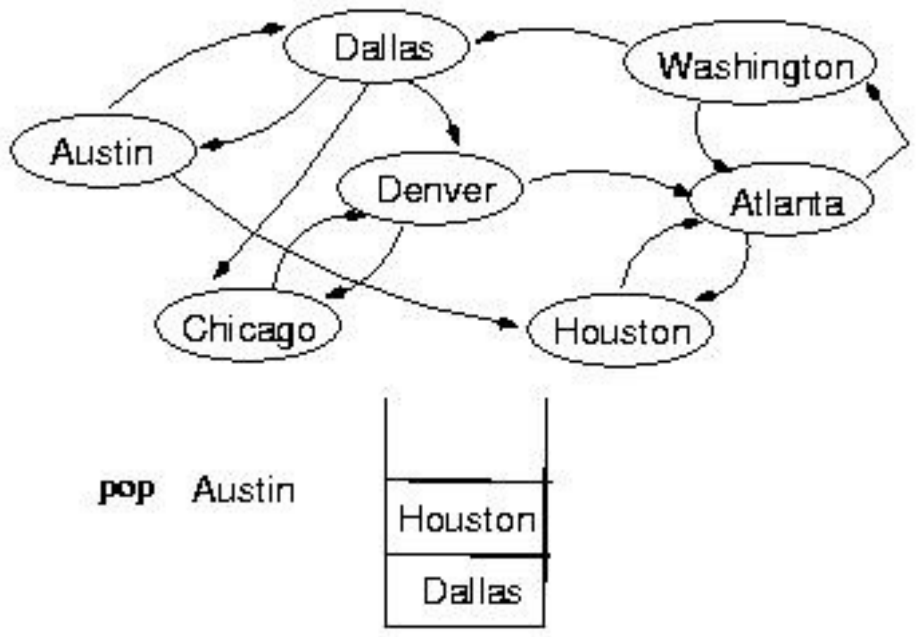
- What is the idea behind DFS?
  - Travel as far as you can down a path
  - Back up as little as possible when you reach a "dead end" (i.e., next vertex has been "marked" or there is no next vertex)
- DFS can be implemented efficiently using a **STACK**

Graph Traversals: DFS

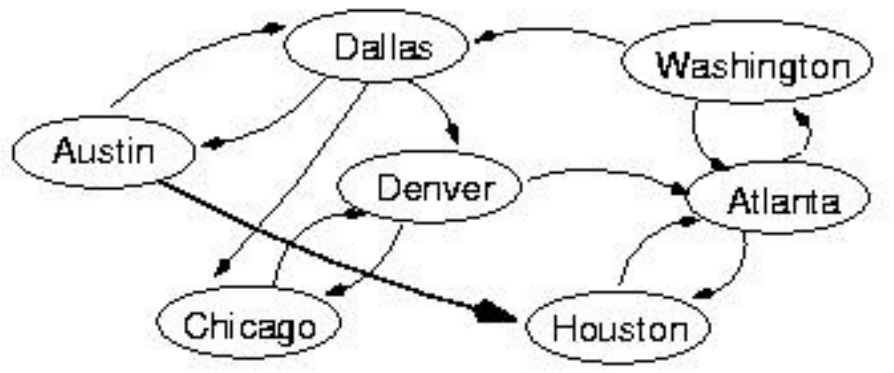
```
Set found to false
stack.Push(startVertex)
DO
  stack.Pop(vertex)
  IF vertex == endVertex
    Set found to true
  ELSE
    Push all adjacent vertices onto stack
WHILE !stack.IsEmpty() AND !found

IF(!found)
  Write "Path does not exist"
```

Graph Traversals: DFS

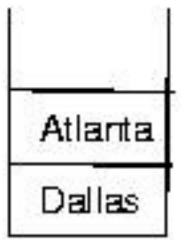


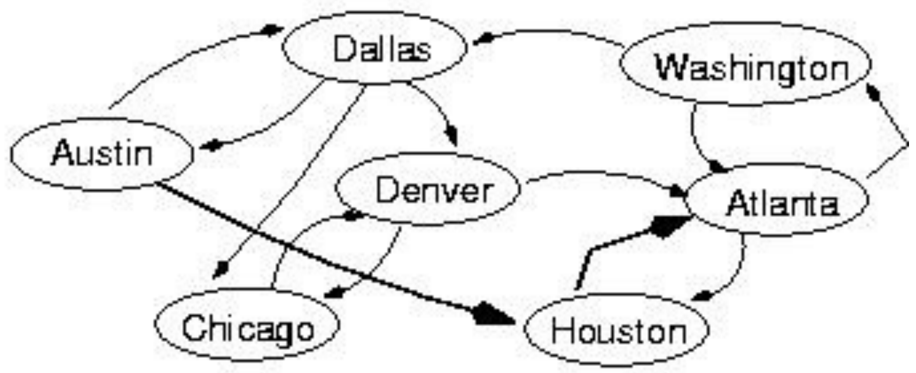


Graph Traversals: DFS

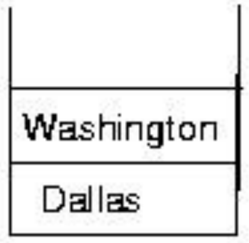


pop Houston

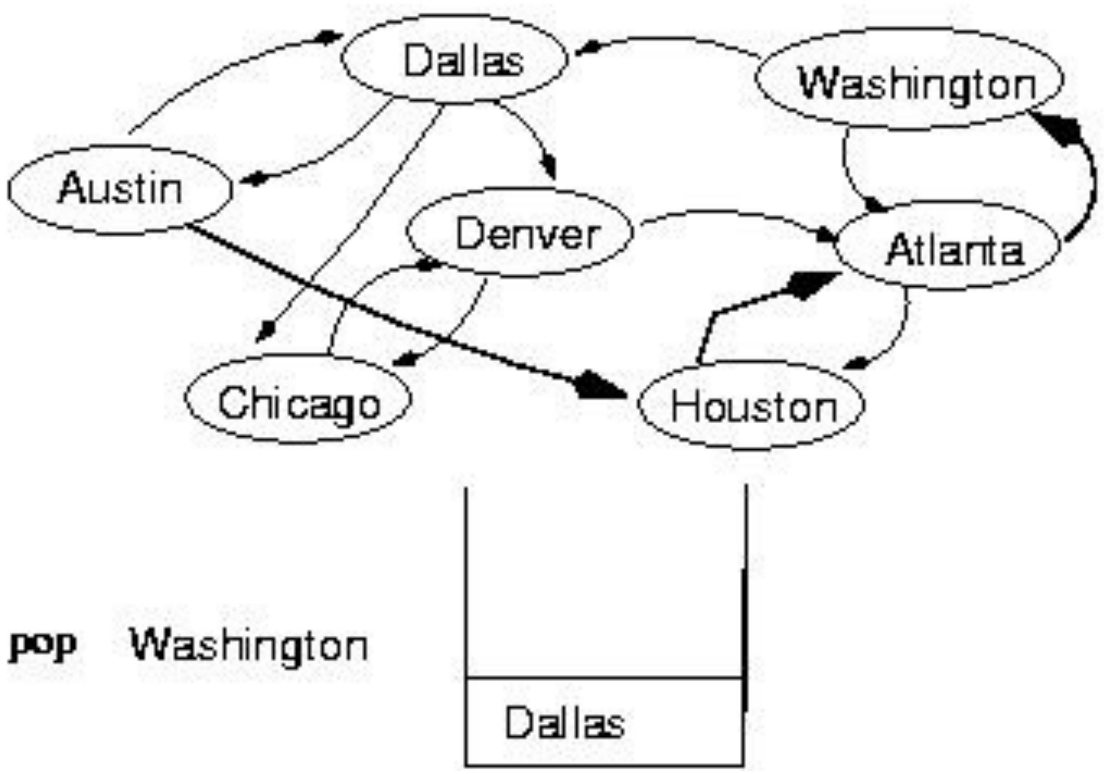




pop Atlanta



Graph Traversals: DFS



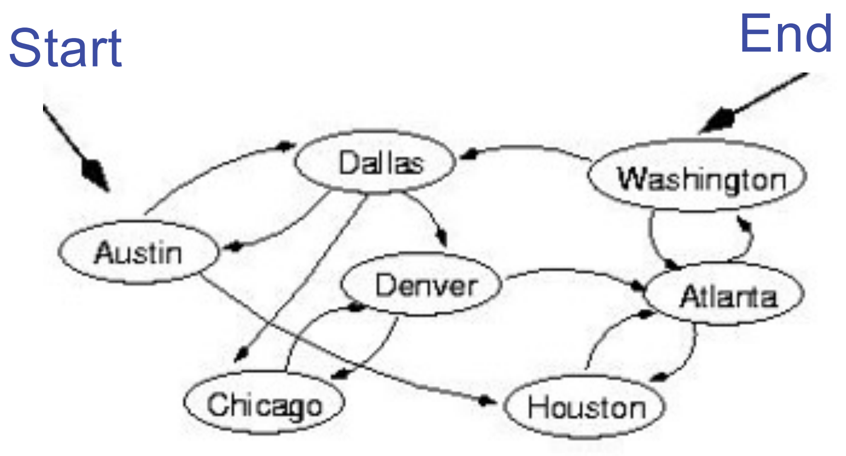
Graph Traversals: BFS

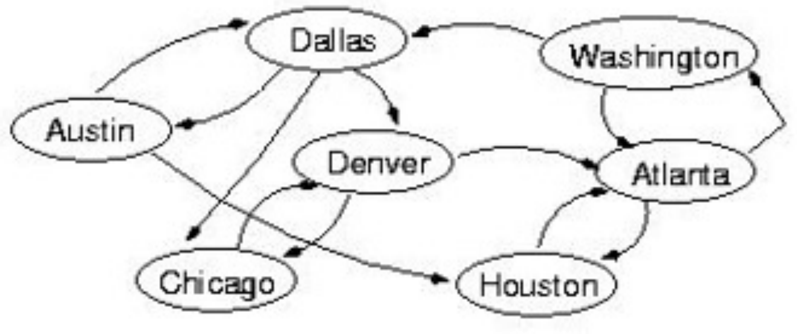
- What is the idea behind BFS?
  - Look at all possible paths at the same depth before you go at a deeper level
  - Back up as far as possible when you reach a "dead end" (i.e., next vertex has been "marked" or there is no next vertex)
- BFS can be implemented efficiently using a **QUEUE**

Graph Traversals: BFS

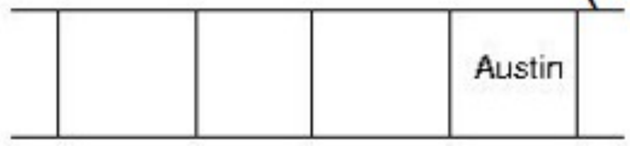
```
Set found to false
queue.Enqueue(startVertex)
DO
  queue.Dequeue(vertex)
  IF vertex == endVertex
    Set found to true
  ELSE
    Enqueue all adjacent vertices onto queue
WHILE !queue.IsEmpty() AND !found
IF(!found)
    Write "Path does not exist"
```

Graph Traversals: BFS

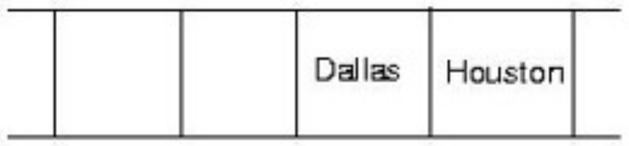




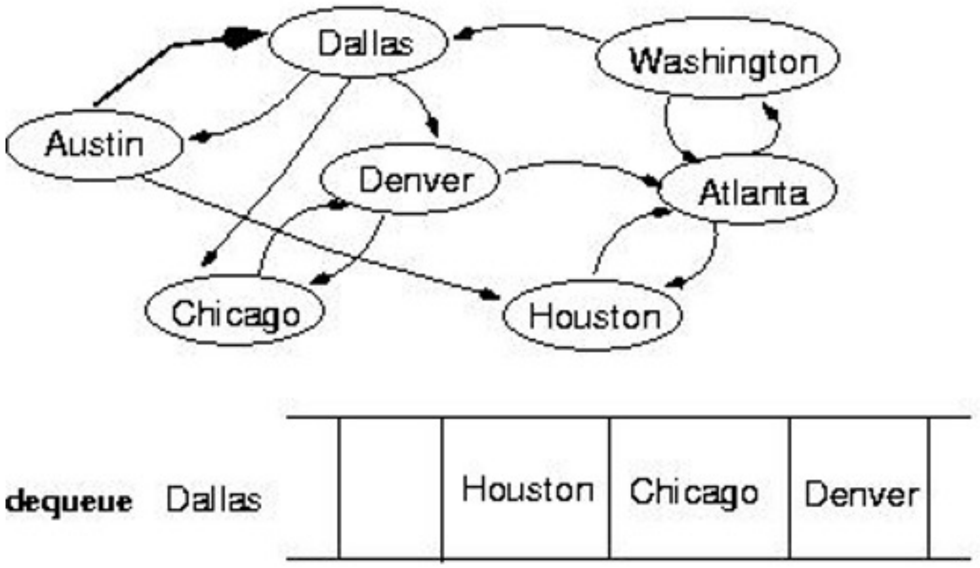
(initialization)

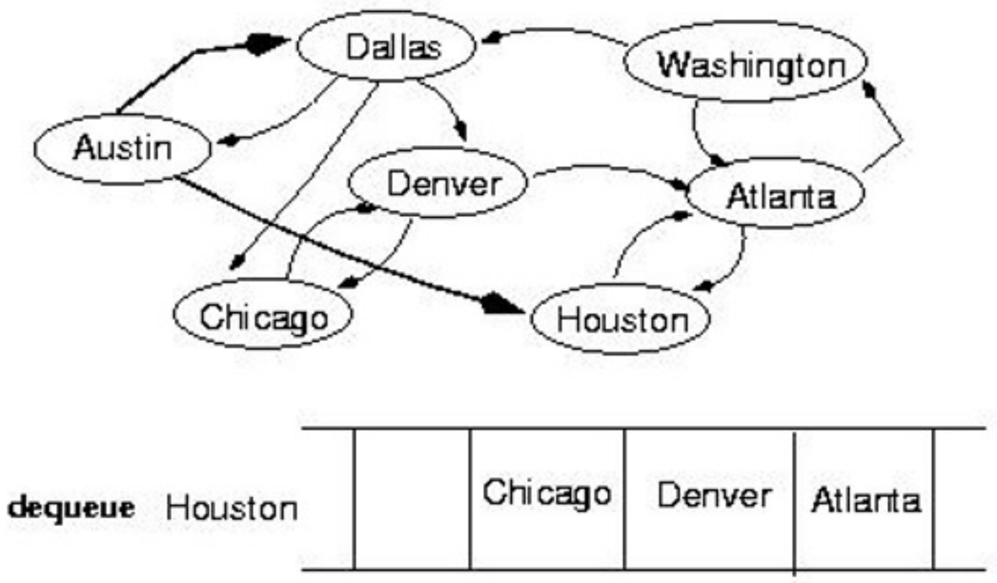


dequeue Austin

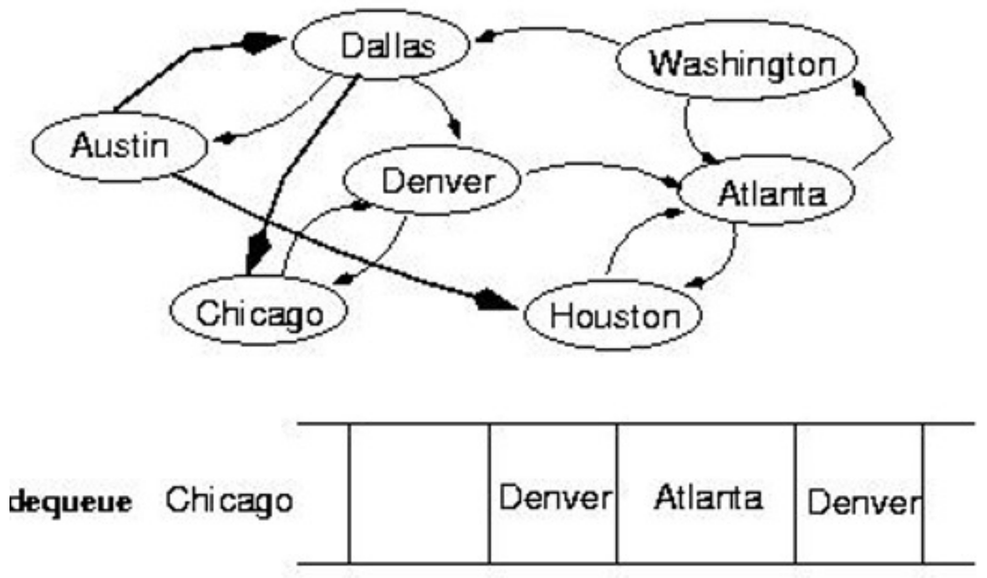


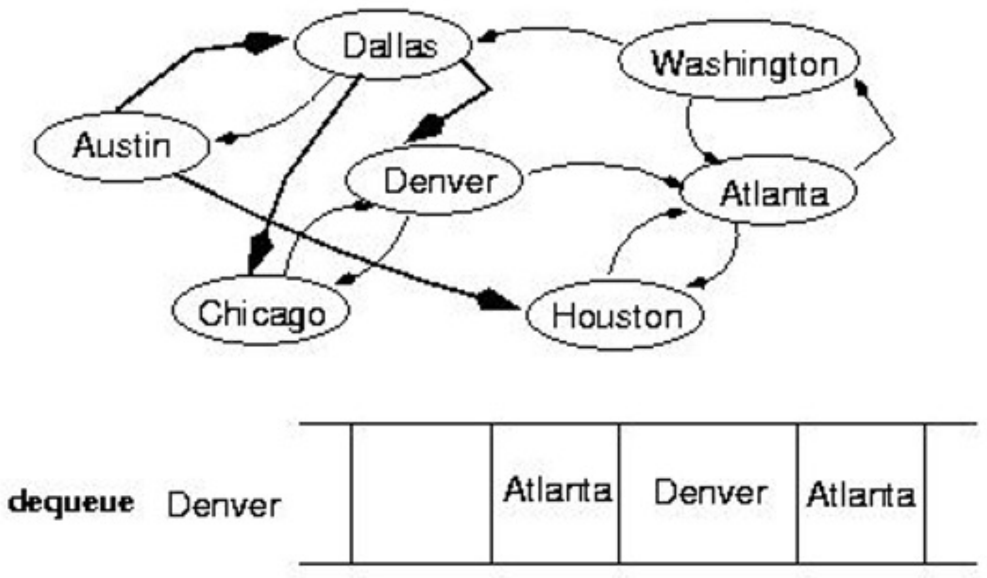
Graph Traversals: BFS



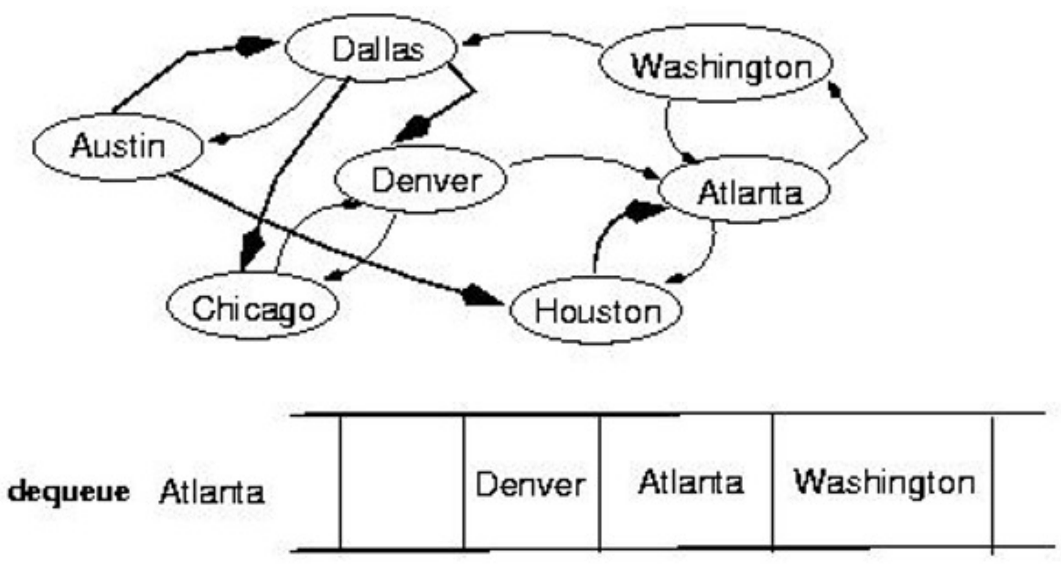


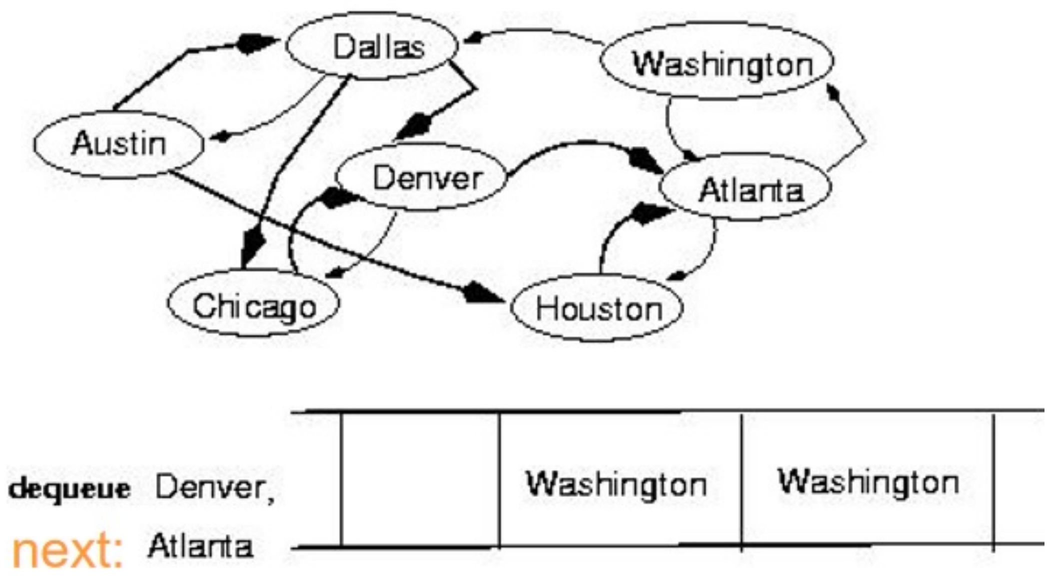
Graph Traversals: BFS



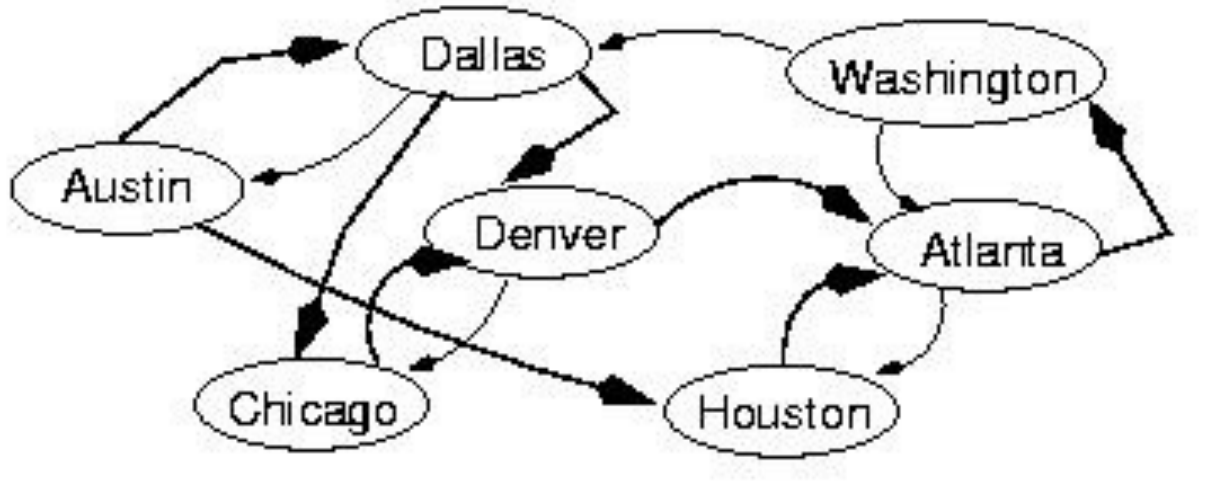


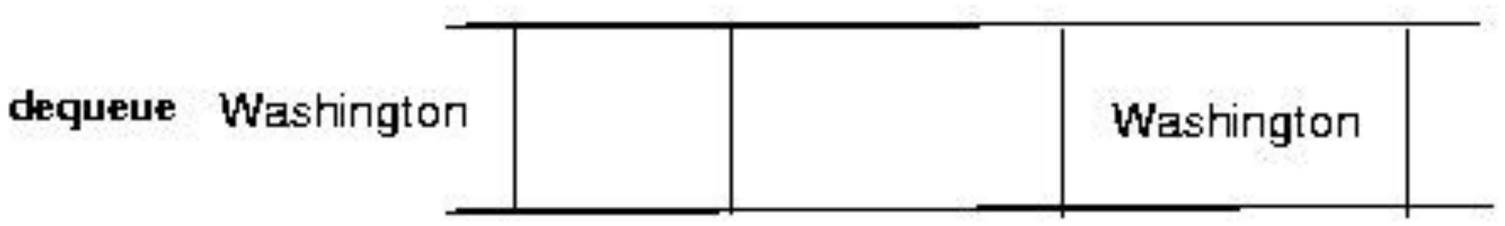
Graph Traversals: BFS





Graph Traversals: BFS





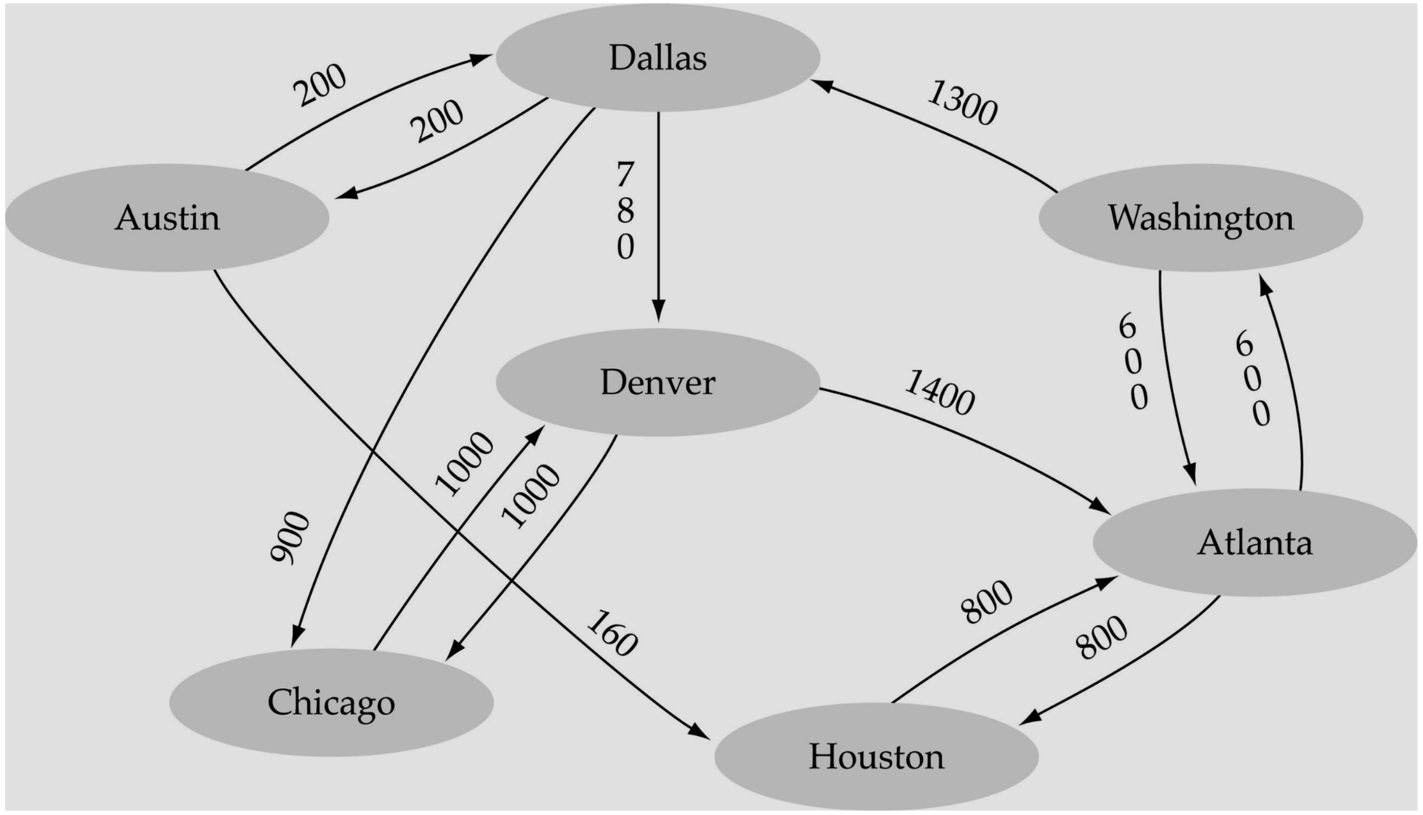
Graph Traversals: DFS/BFS Comparison

- BFS:
  - BFS always finds the shortest path (or "optimal solution") from the starting node to a target node
  - Storage for BFS can be extremely large
- DFS:
  - DFS can use less space in finding a path

Graphs: 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: 2980miles

Graphs: Shortest-Path Problem



Graphs: Shortest-Path Problem

- Common algorithms:
  - Dijkstra's algorithm
  - 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

Graph Traversals: 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)
- BFS can be implemented efficiently using a **QUEUE**

Summary

- Graphs and Types of Graphs
- Graph vs Tree
- Applications of Graphs in Real-Life
- DFS and BFS Algorithms
- Shortest Path Problem in Graphs