i206: Lecture 14:
Graphs

Tapan Parikh

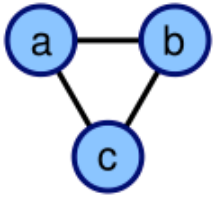
Spring 2013

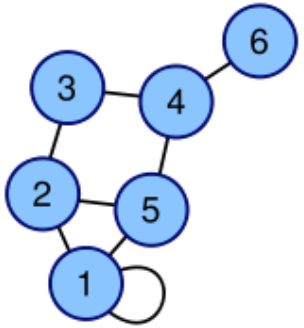
Some slides courtesy Marti Hearst, John Chuang and others

Outline

- What is a data structure
- Basic building blocks: arrays and linked lists
- Data structures (uses, methods, performance):
  - List, stack, queue
  - Dictionary
  - Tree
  - Graph

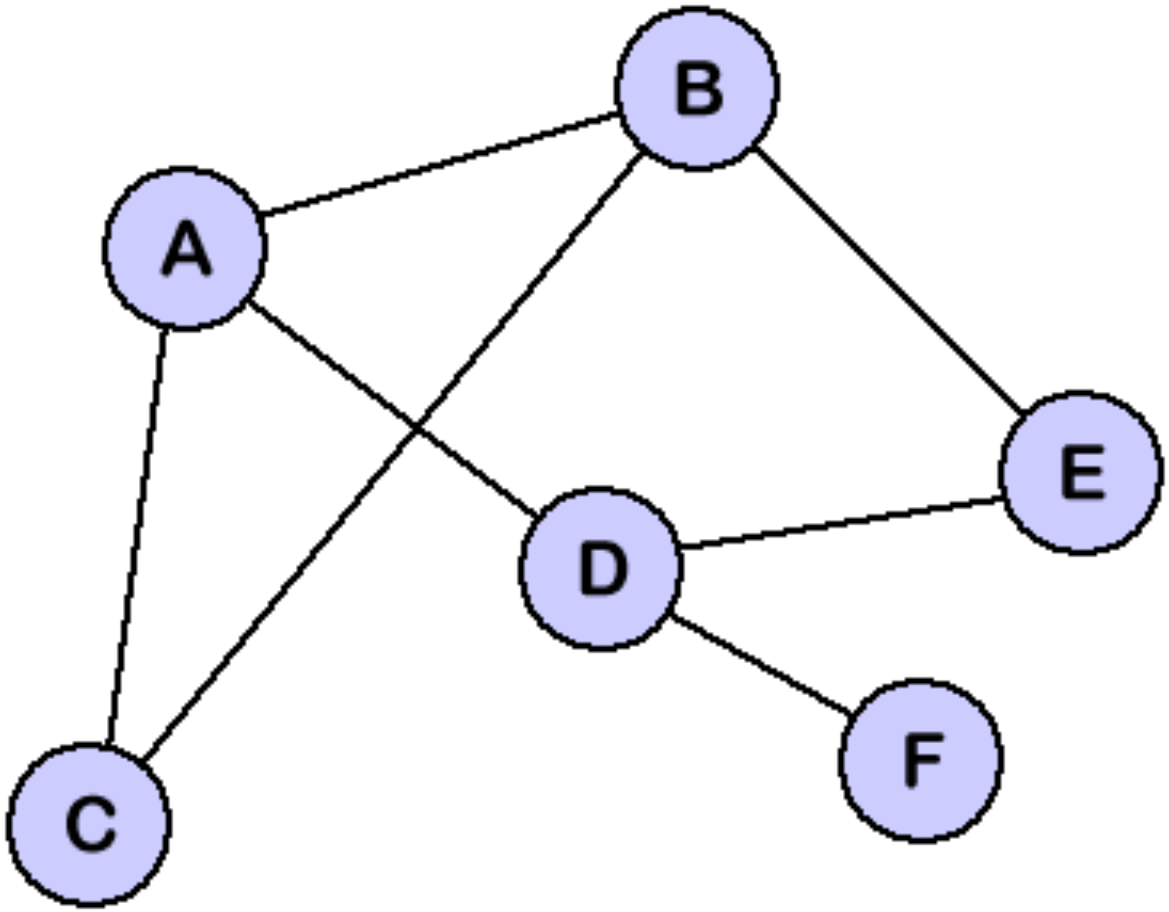
Graph



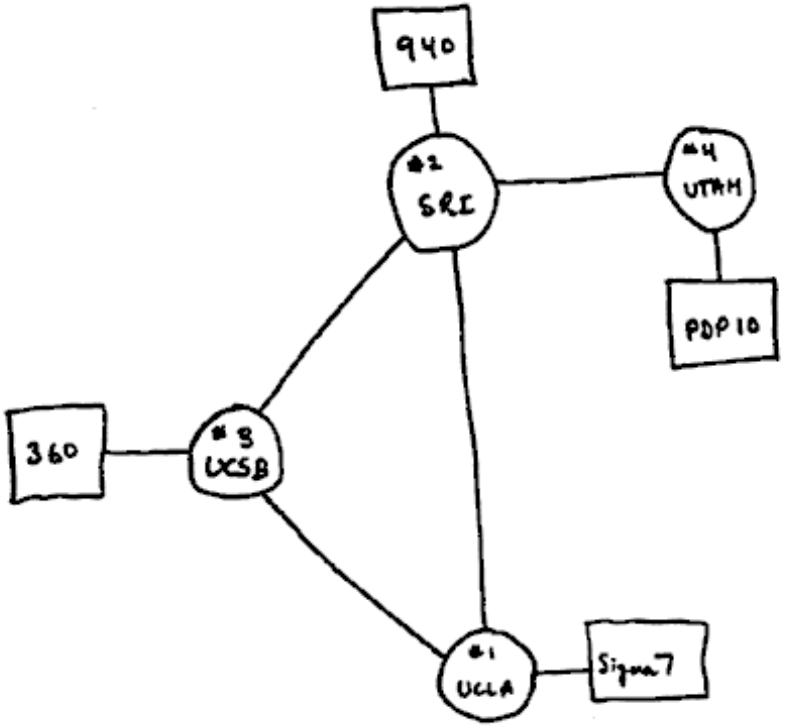


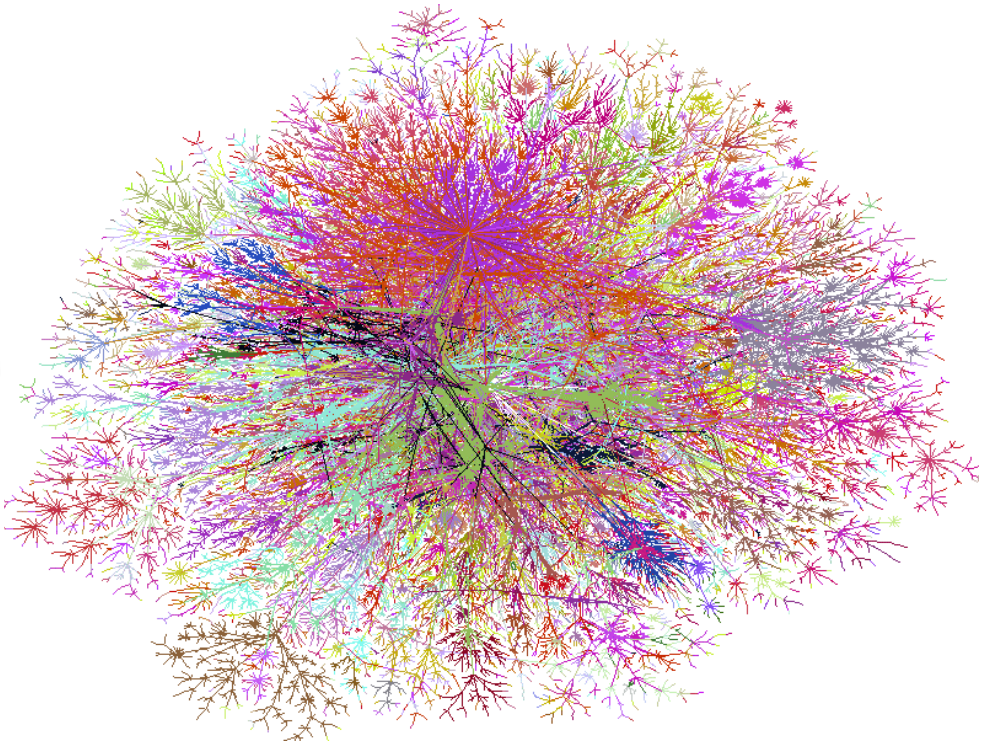
- A graph consists of a set of nodes (vertices) and a set of links (edges) that establish relationships (connections) between the nodes
- Represented/stored using adjacency list or adjacency matrix data structures
  - Adjacency list for Graph 1: {a,b}, {a,c}, {b,c}
  - Adjacency matrix for Graph 2:
- Edges can be directed/undirected
- Edges can have weights
- Tree is a special case of graph

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$


B
A
E
D
F
C

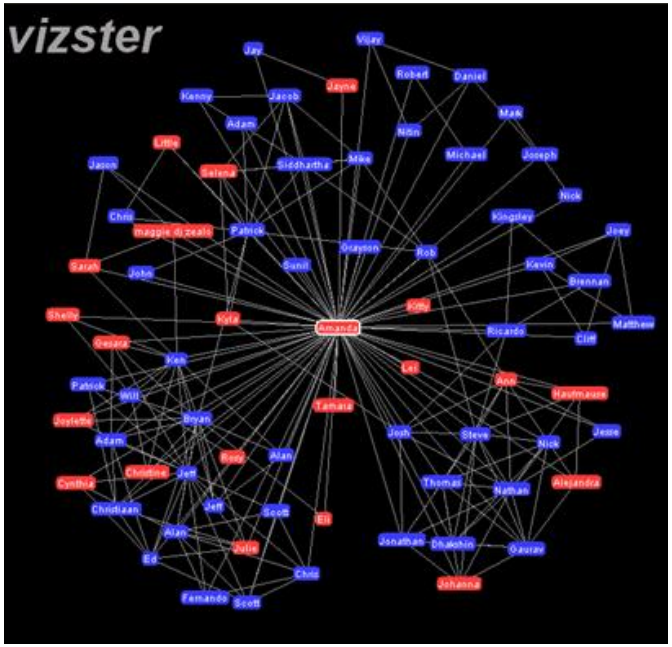
Internet Graphs

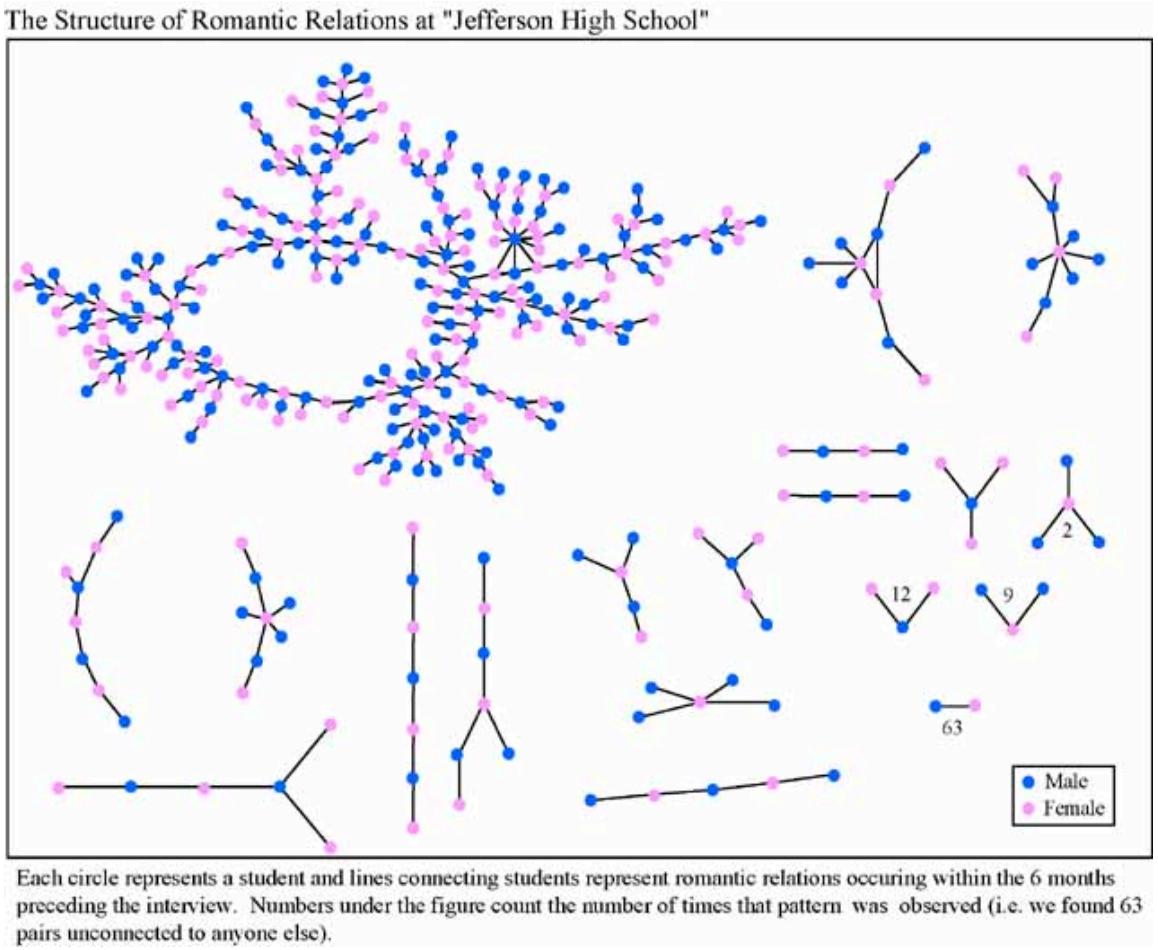




Source: Cheswick and Burch

Social Network Graphs





Each circle represents a student and lines connecting students represent romantic relations occuring within the 6 months preceding the interview. Numbers under the figure count the number of times that pattern was observed (i.e. we found 63 pairs unconnected to anyone else).

American Journal of Sociology, Vol. 100, No. 1. "Chains of affection: The structure of adolescent romantic and sexual networks," Bearman PS, Moody J, Stovel K.

Why Use Graphs?

- Graphs serve as models of a wide range of objects:
  - A roadmap
  - A map of airline routes
  - A layout of an adventure game world
  - A schematic of the computers and connections that make up the Internet
  - The links between pages on the Web
  - The relationship between students and courses
  - A diagram of the flow capacities in a communications or transportation network

Google Maps

Search Maps | Show search options

Find businesses, addresses and places of interest. Learn more.

Get Directions | My Maps

Print | Send | Link

A: Berkeley, ca

B: stanford, ca

Add Destination - Show options

By car | Get Directions

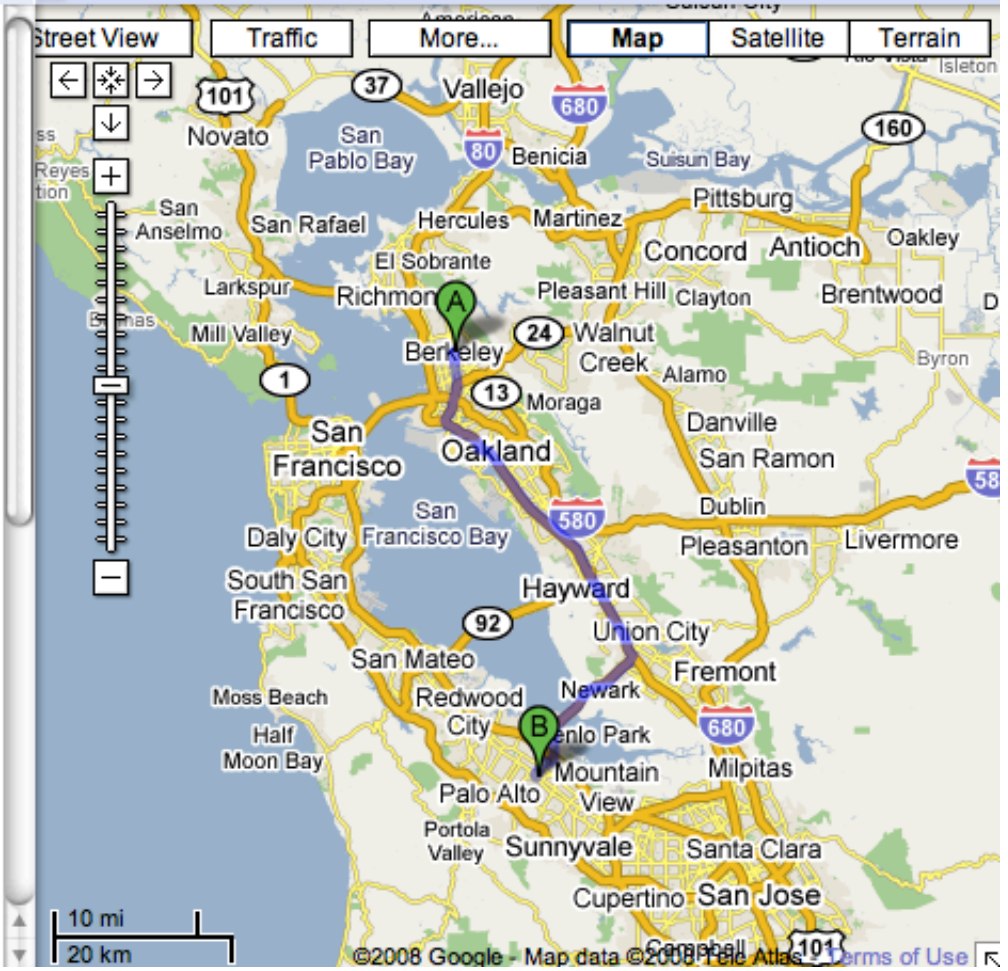
Also available: Public Transit

Driving directions to Stanford, CA

39.0 mi – about **54 mins** (up to 1 hour 10 mins in traffic)

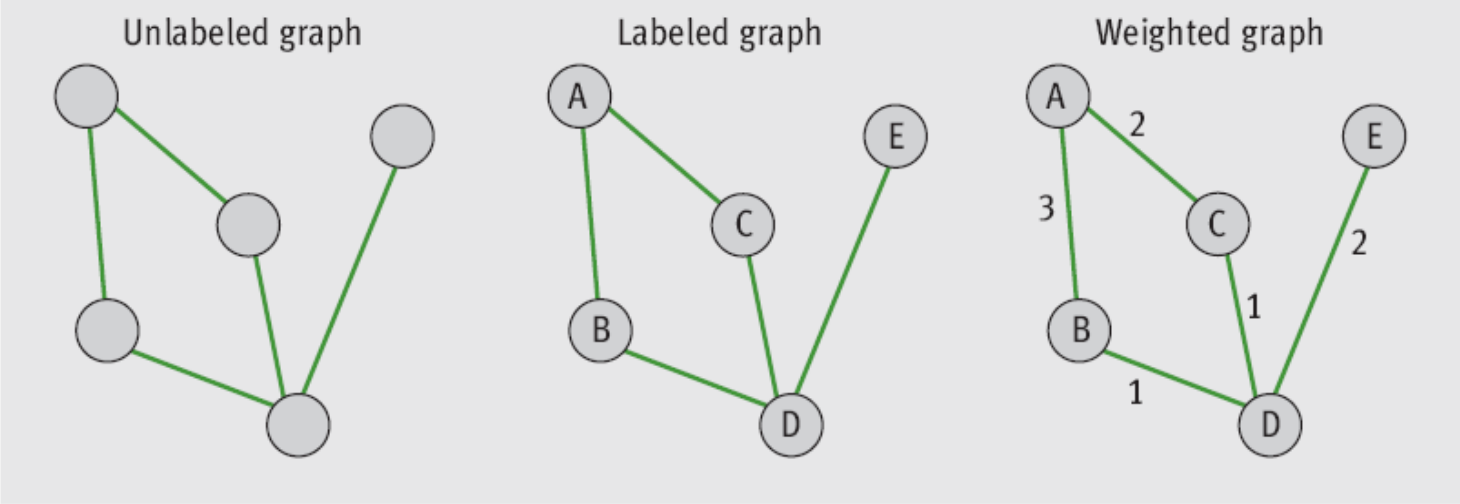
A Berkeley, CA

1. Head **south** on **Shattuck Ave/Shattuck Square** toward **University Ave** — 2.4 mi
   Continue to follow Shattuck Ave
2. Turn **right** at **52nd St** — 59 ft
3. Take the ramp to **CA-24 W** — 0.5 mi
4. Keep **right** at the fork, follow signs for **I-980** and merge onto **CA-24 W** — 0.6 mi
5. Continue on **I-980 W** — 2.3 mi
6. Merge onto **I-880 S** — 20.9 mi



Graph Terminology

- Mathematically, a graph is a set *V* of **vertices** and a set *E* of **edges**, such that each edge in *E* connects two of the vertices in *V*
  - We use **node** as a synonym for vertex



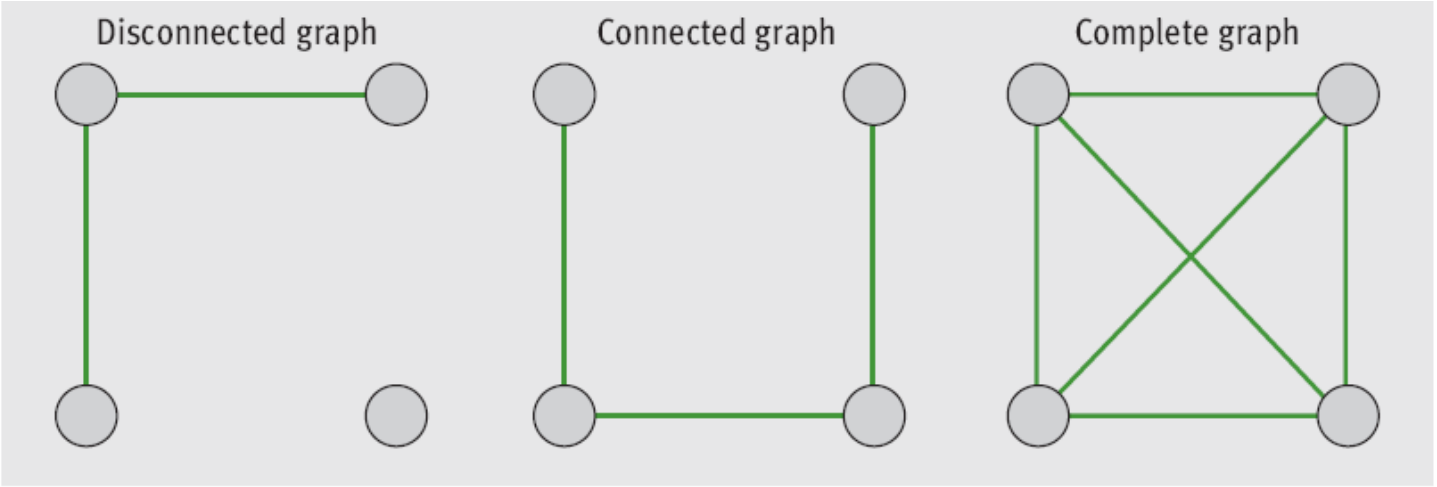
[FIGURE 20.1] Unlabeled, labeled, and weighted graphs

Graph Terminology (continued)

- One vertex is **adjacent** to another vertex if there is an edge connecting the two vertices (**neighbors**)
- **Path:** Sequence of edges that allows one vertex to be reached from another vertex in a graph
- A vertex is **reachable** from another vertex if and only if there is a path between the two
- **Length of a path:** Number of edges on the path
- **Degree of a vertex:** Number of edges connected to it
  - In a complete graph: Number of vertices minus one

Graph Terminology (continued)

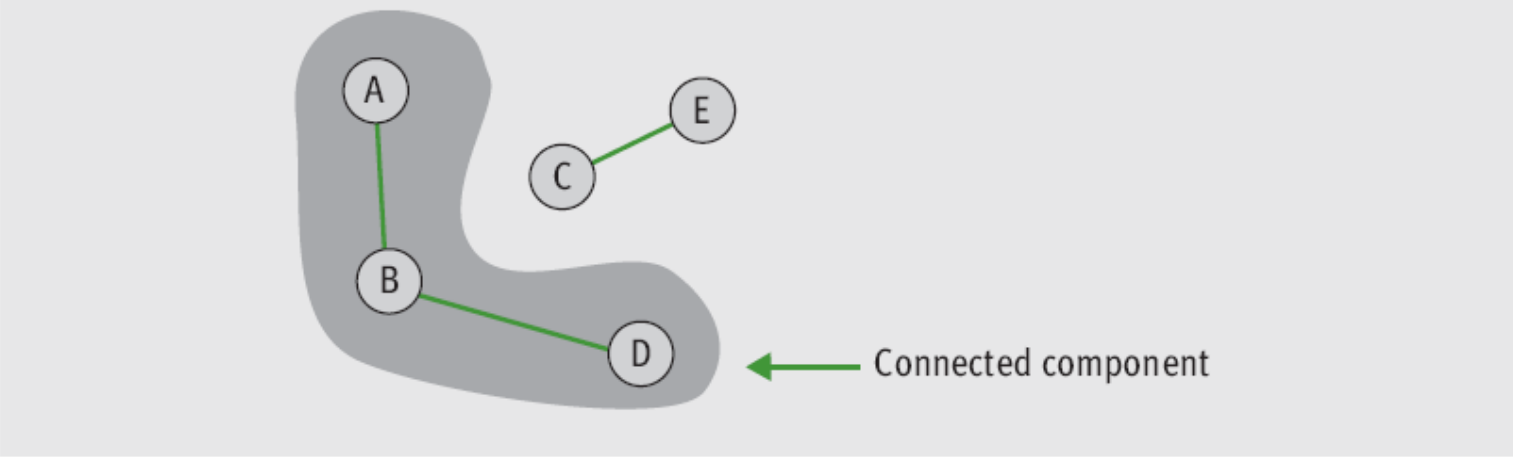
- A graph is connected if there is a path from each vertex to every other vertex
- A graph is complete if there is an edge from each vertex to every other vertex



[FIGURE 20.2] Disconnected, connected but not complete, and complete graphs

Graph Terminology (continued)

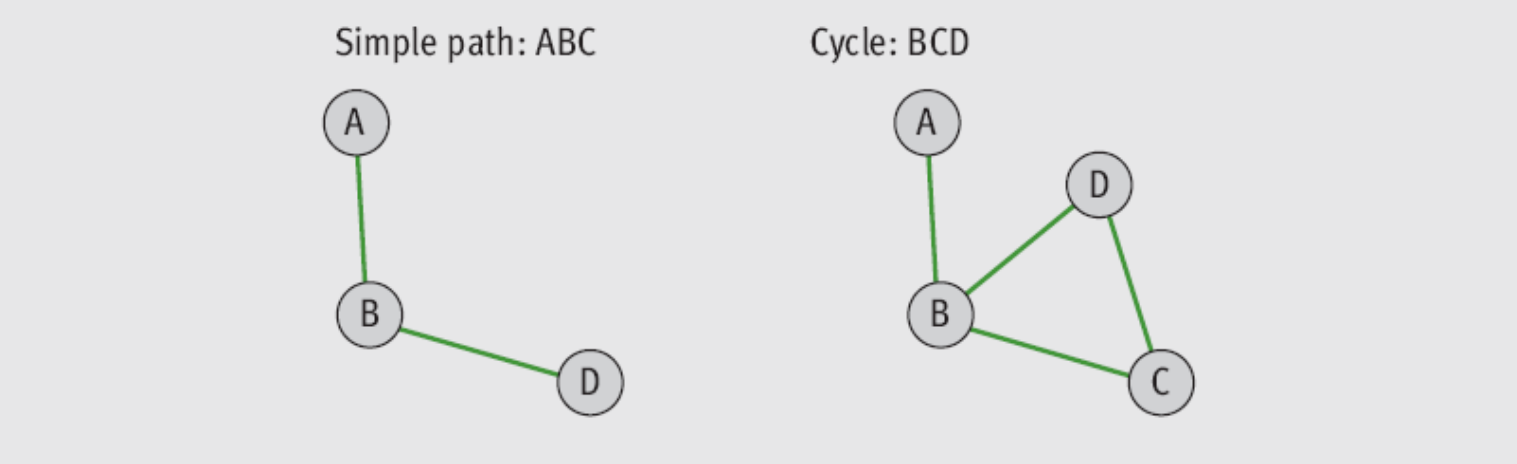
- **Subgraph:** Subset of the graph's vertices and the edges connecting those vertices
- **Connected component:** Subgraph consisting of the set of vertices reachable from a given vertex



[FIGURE 20.3] A connected component of a graph

Graph Terminology (continued)

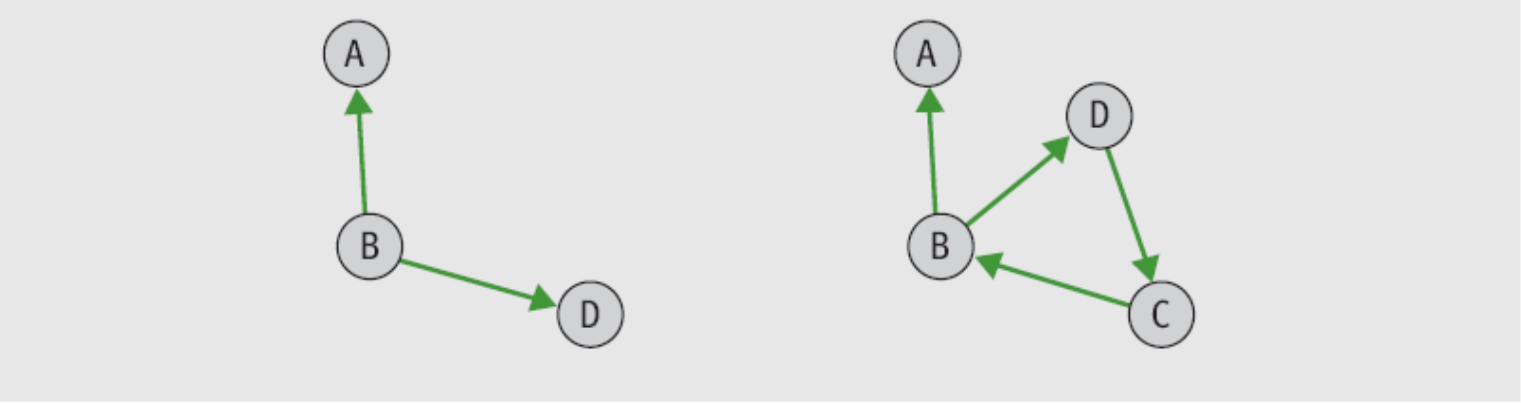
- **Simple path:** Path that does not pass through the same vertex more than once
- **Cycle:** Path that begins and ends at the same vertex



[FIGURE 20.4] A simple path and a cycle

Graph Terminology (continued)

- Graphs can be **undirected** or **directed** (**digraph**)
- A **directed edge** has a **source vertex** and a **destination vertex**
- Edges emanating from a given source vertex are called its **incident edges**



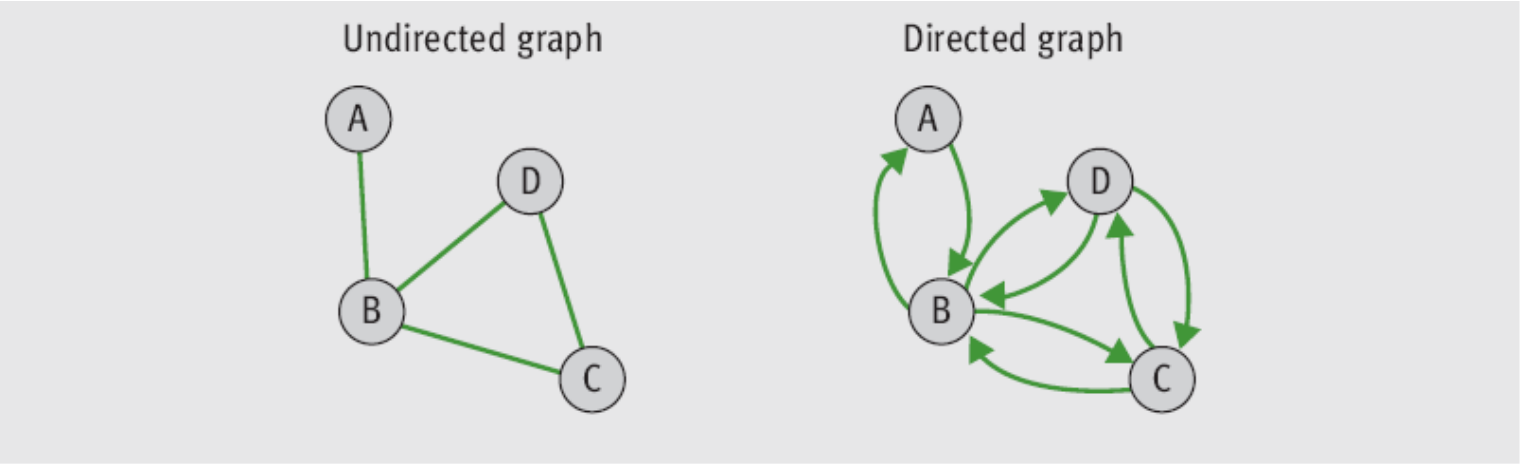
[FIGURE 20.5] Directed graphs (digraphs)

Graph Terminology (continued)

- Is Twitter a directed or undirected graph?
- How about Facebook?

Graph Terminology (continued)

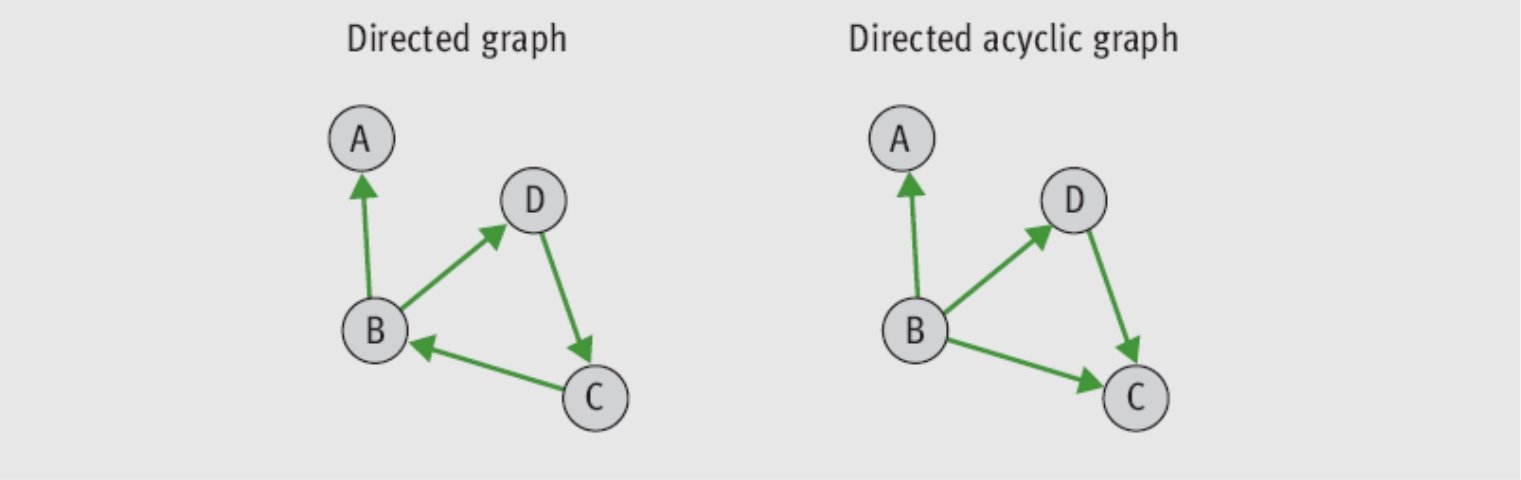
- To convert undirected graph to equivalent directed graph, replace each edge in undirected graph with a pair of edges pointing in opposite directions



[FIGURE 20.6] Converting an undirected graph to a directed graph

Graph Terminology (continued)

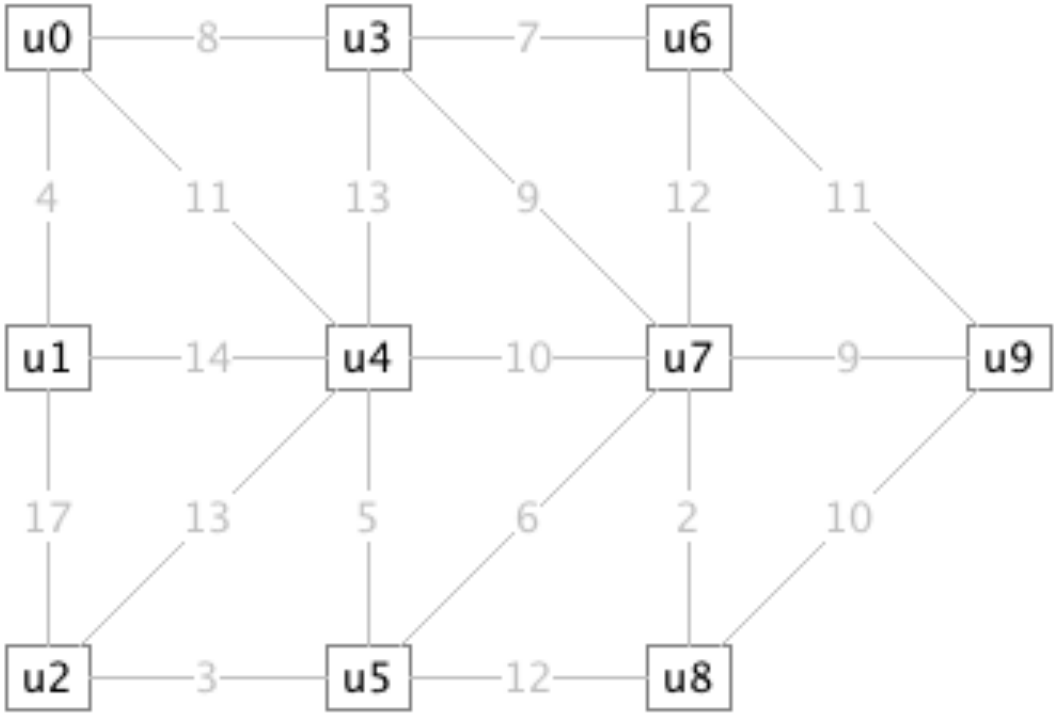
- A special case of digraph that contains no cycles is known as a **directed acyclic graph**, or **DAG**



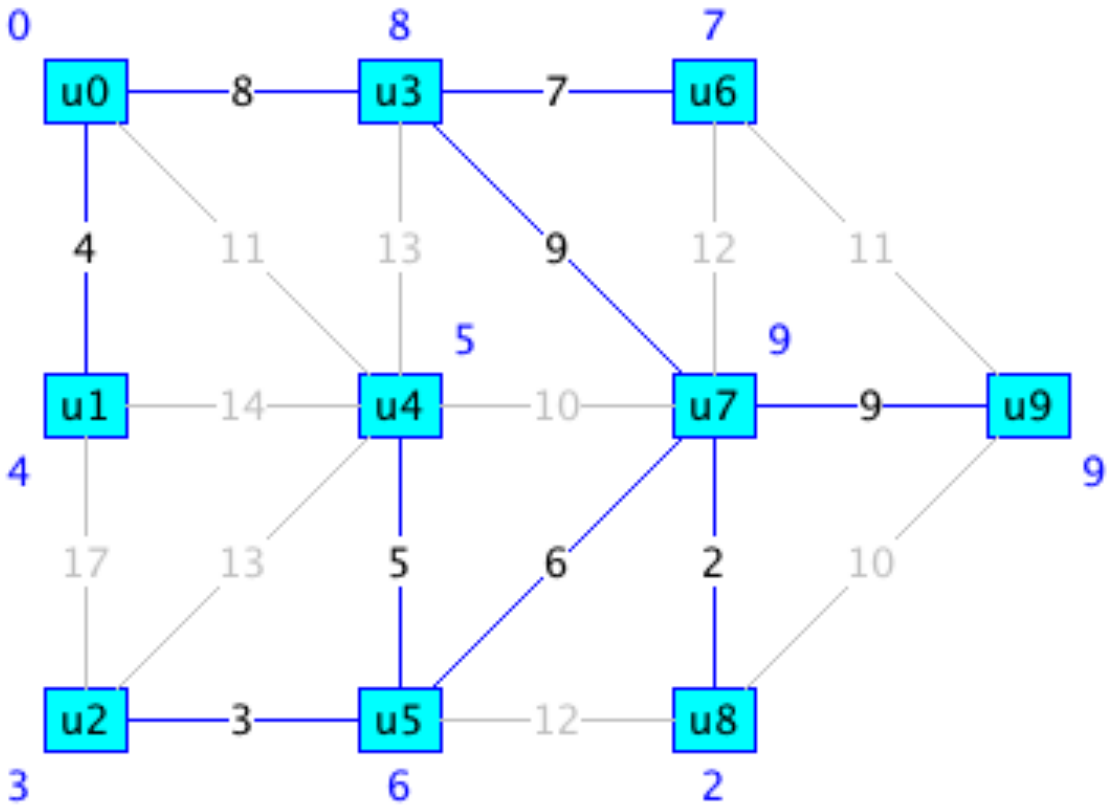
[FIGURE 20.7] A directed graph and a directed acyclic graph (DAG)

- Lists and trees are special cases of directed graphs

Given this graph, how would you find the shortest path that links every node?



Minimum Spanning Tree



http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/PrimApp.shtml?demo3

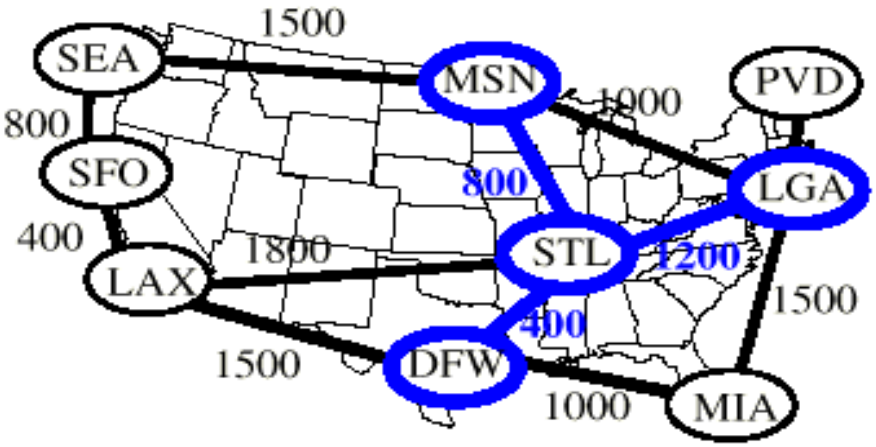
Minimum Spanning Tree

- In a weighted graph, you can sum the weights for all edges in a spanning tree and attempt to find a spanning tree that minimizes this sum
  - There are several algorithms for finding a **minimum spanning tree** for a component.

- For example, to determine how an airline can service all cities, while minimizing the total length of the routes it needs to support

Minimum Spanning Tree

- Prim-Jarnik algorithm
- Kruskal algorithm



Slide adapted from Goodrich & Tamassia

Algorithms for Minimum Spanning Trees

- There are two well-known algorithms for finding a minimum spanning tree:
  - One developed by Robert C. Prim in 1957
    - https://en.wikipedia.org/wiki/Prim%27s_algorithm
  - The other by Joseph Kruskal in 1956
- Here is Prim's algorithm:

```
minimumSpanningTree(graph):
    mark all vertices and edges as unvisited
    mark some vertex, say v, as visited
    repeat until all vertices have been visited:
      find the least weight edge from any visited vertex to an
    unvisited vertex, say w.  Mark the edge and w as visited.
```

  - Maximum running time is O(*m* * *n)*
  - Animation: http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/PrimApp.shtml?demo3

Prim's MST Algorithm (continued)

- Improvement to algorithm: use a heap of edges where the smallest edge weight is on top of the heap.

```
1   minimumSpanningTree(Graph g):
2       mark all edges as unvisited
3       mark all vertices as unvisited
4       mark some vertex, say v, as visited
5       for each edge leading from v:
6           add the edge to the heap
7       k = 1
8       while k < number of vertices:
9           remove an edge from the heap
10          if one end of this edge, say vertex w, is not visited:
11              mark the edge and w as visited
12              for each edge leading from w:
13                  add the edge to the heap
14              k += 1
```

- Maximum running time: O(*m* log *n*)

Graph Traversals

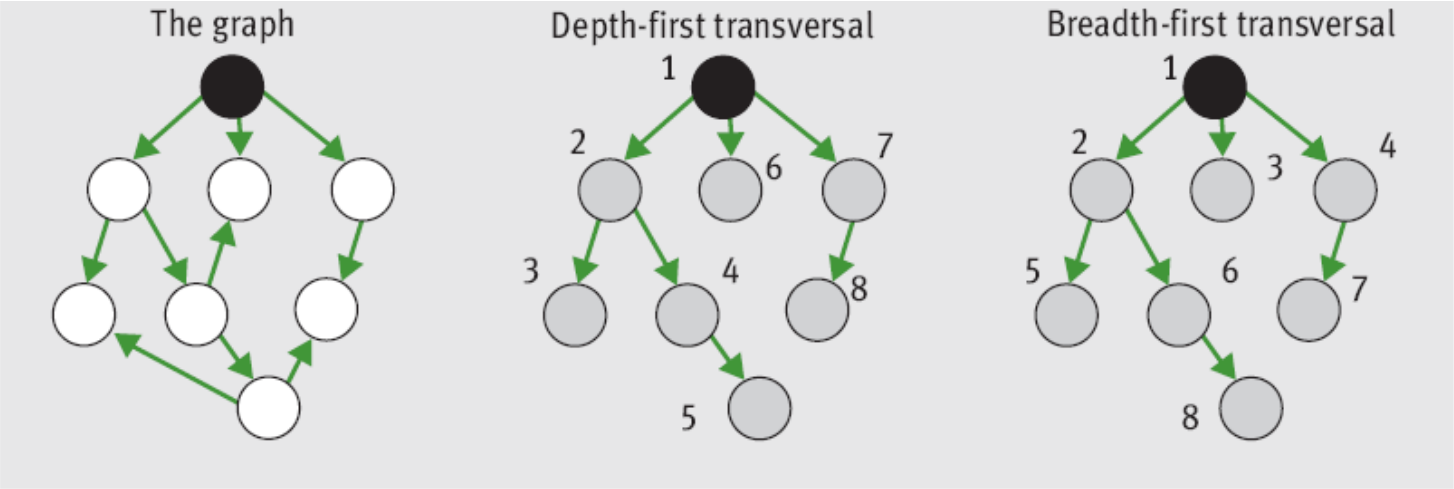
- In addition to the insertion and removal of items, important graph-processing operations include:
  - Finding the shortest path to a given item in a graph
  - Finding all of the items to which a given item is connected by paths
  - Traversing all of the items in a graph
    - One starts at a given vertex and, from there, visits all vertices to which it connects
    - Different from tree traversals, which always visit all of the nodes in a given tree

A Generic Traversal Algorithm

```
traverseFromVertex(graph, startVertex):
    mark all vertices in the graph as unvisited
    insert the startVertex into an empty collection
    while the collection is not empty:
        remove a vertex from the collection
        if the vertex has not been visited:
            mark the vertex as visited
            process the vertex
            insert all adjacent unvisited vertices into the collection
```

Breadth-First and Depth-First Traversals

- A **depth-first traversal**, uses a stack as the collection in the generic algorithm
- A **breadth-first traversal**, uses a queue as the collection in the generic algorithm

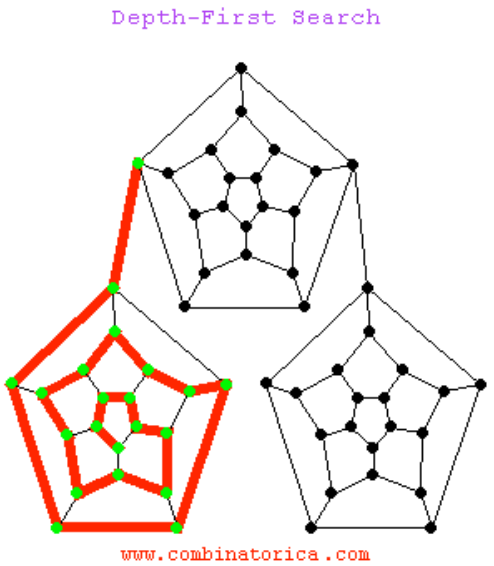


[FIGURE 20.13] Depth-first and breadth-first traversals of a given graph

Depth-First Traversal

- DFS prefers to visit undiscovered vertices immediately, so the search trees tend to be deeper rather than balanced as with BFS.
- Recursive depth-first traversal:

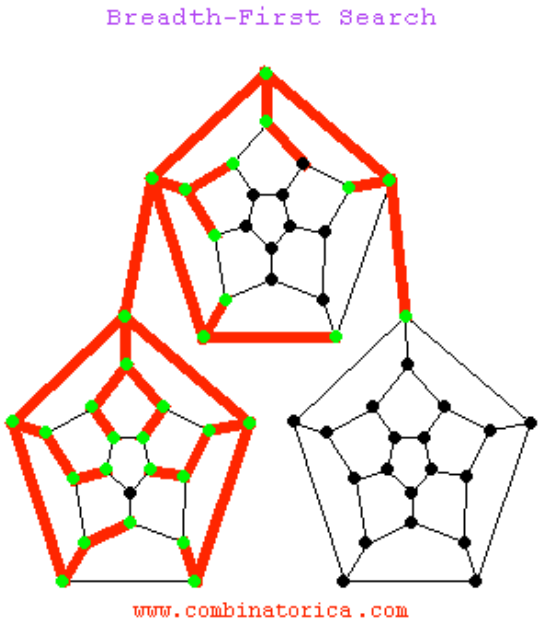
```
traverseFromVertex(graph, startVertex):
    mark all vertices in the graph as unvisited
    dfs(graph, startVertex)

dfs(graph, v):
    mark v as visited
    process v
    for each vertex, w, adjacent to v:
        if w has not been visited:
            dfs(graph, w)
```



http://www.cs.sunysb.edu/~skiena/combinatorica/animations/search.html

Breadth-First Traversal

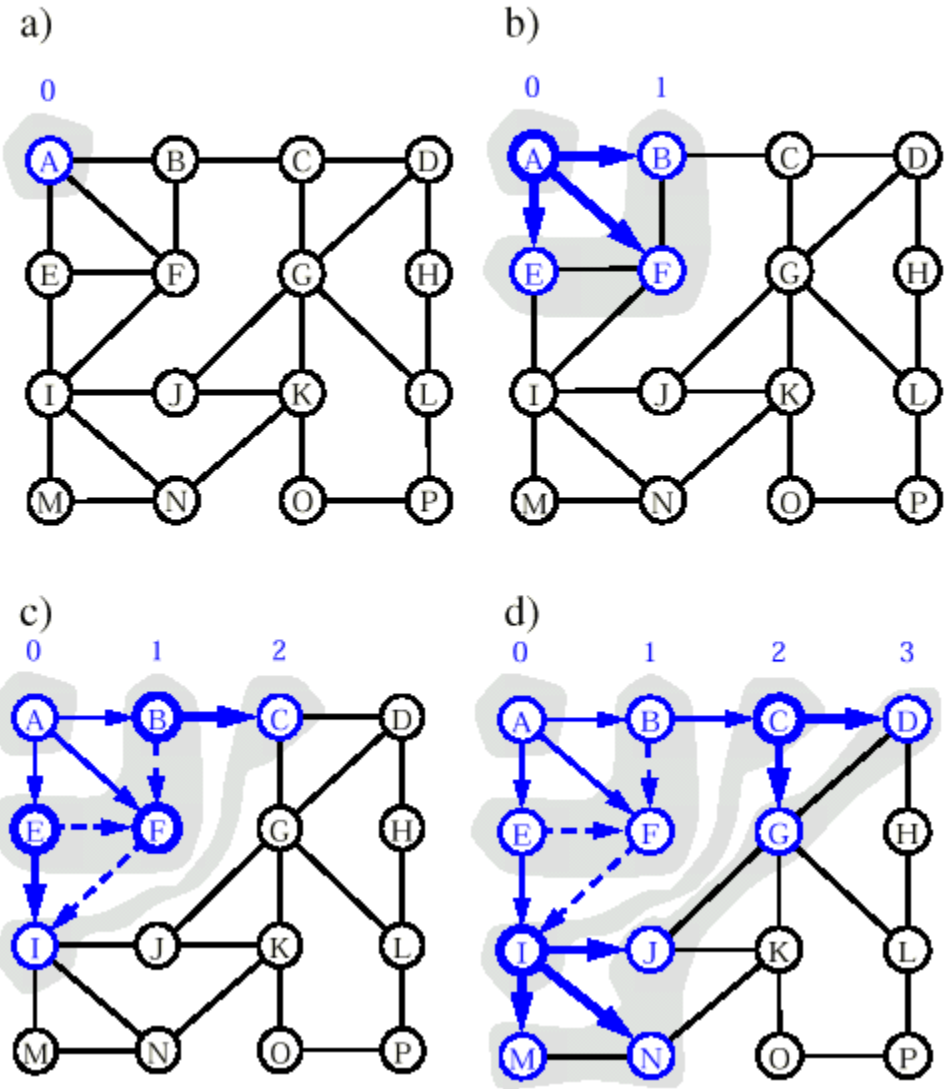
- Breadth-first traversal (BFS) is a graph traversal algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.



- Breadth-first traversal:
  - Select a node, mark it, and enqueue it
  - While the queue is not empty:
    - Dequeue a node and mark it
      - Enqueue all nodes that have an edge unvisited (unmarked)

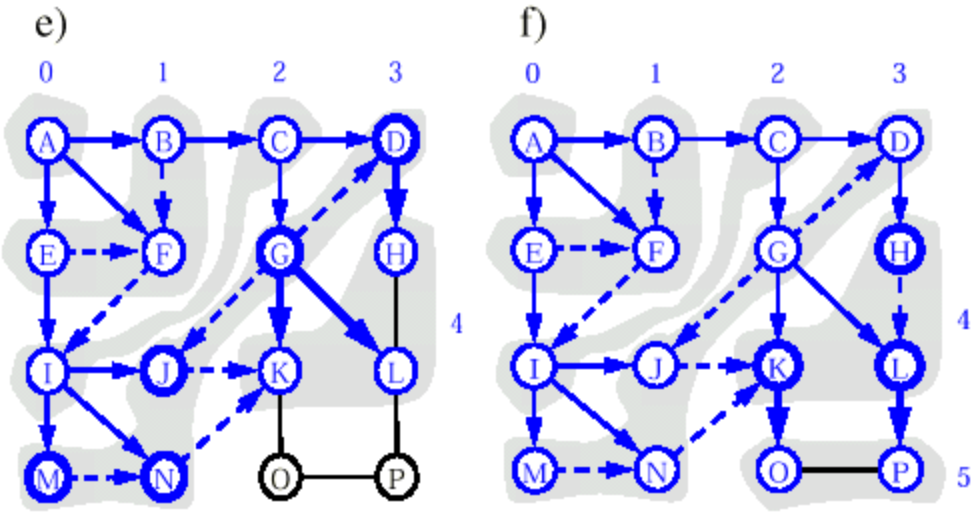
http://www.cs.sunysb.edu/~skiena/combinatorica/animations/search.html

BFS - A Graphical Representation



Slide adapted from Goodrich & Tamassia

More BFS

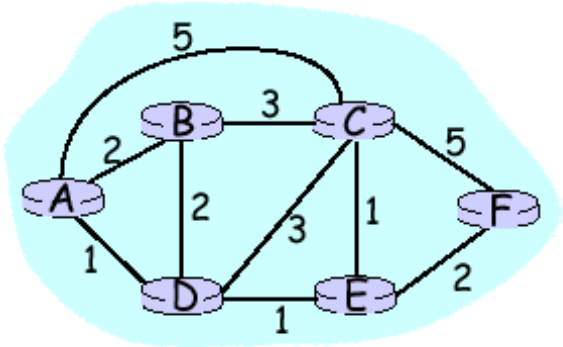


Slide adapted from Goodrich & Tamassia

Greedy Algorithms

- An algorithm which always takes the best immediate, or local, solution in looking for an answer.
- Greedy algorithms sometimes find less-than-optimal solutions
- Some greedy algorithms always find the optimal solution
  - Dijkstra's shortest paths algorithm
  - Prim's algorithm for minimum spanning trees

Shortest Path



- Problem: network routers have to forward data packets toward destination; must determine next hop
- Algorithm: Dijkstra's algorithm
  - Shortest Path First (SPF) algorithm
  - Greedy algorithm
  - Input: graph with nodes and weighted edges
  - Output: shortest paths from source node *i* to **every** other node; cost of each path

Dijkstra's Algorithm

- Inputs: a directed acyclic graph with edge weights > 0 and the source vertex
- Computes the distances of the shortest paths from source vertex to all other vertices in graph
- Output: a two-dimensional grid, **`results`**
  - *N* rows, where *N* is the number of vertices
- Steps: initialization step and computation step

Algorithm Intuition

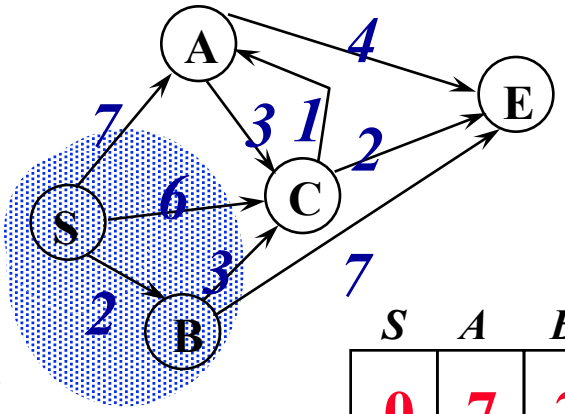
- Start at source node
- Move outward
- At each step:
  - Find node *u* that
    - has not been considered before; and
    - is closest to source
  - Compute:
    - Distance from u to each neighbor *v*
    - If distance shorter, make path to *v* go through *u*

- http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/DijkstraApp.shtml?demo

Shortest paths, more details

- Single-source shortest path
  - Start at some vertex S
  - Find shortest path to every reachable vertex from S
- A set of vertices is *processed*
  - Initially just S is processed
  - Each pass processes a vertex

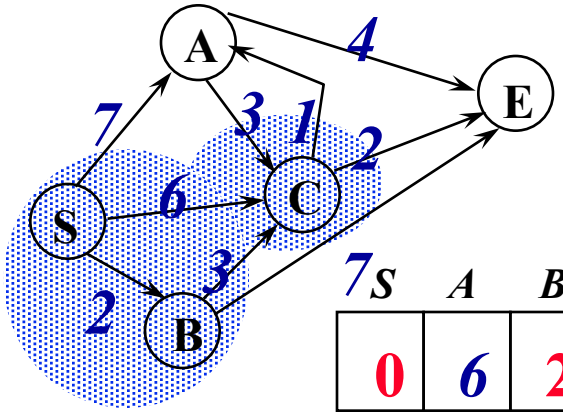
  After each pass, shortest path from S to any vertex using just vertices from processed set (except for last vertex) is always known
- Next processed vertex is closest to S still needing processing



S	A	B	C	E	
0	7	2	6	∞	

process B

0 7 2 5 9



S	A	B	C	E	
0	6	2	5	7	

process C

Analysis

- The initialization step must process every vertex, so it is O(n)
- The outer loop of the computation step also iterates through every vertex
  - The inner loop of this step iterates through every vertex not included thus far
  - Hence, the overall behavior of the computation step resembles that of other O(n^2) algorithms
- Dijkstra's algorithm is O(n^2)

Graph Algorithms

- Search/traversal: breadth-first or depth-first -- $O(|V|+|E|)$
- Routing: shortest path between two points (Dijkstra's algorithm) -- $O(|V|^2+|E|)$
- Minimum spanning tree -- $O(|E|)$
- Maximum Flow -- $O(|V|^3)$, $O(|V|^2|E|)$, $O(|V||E|^2)$