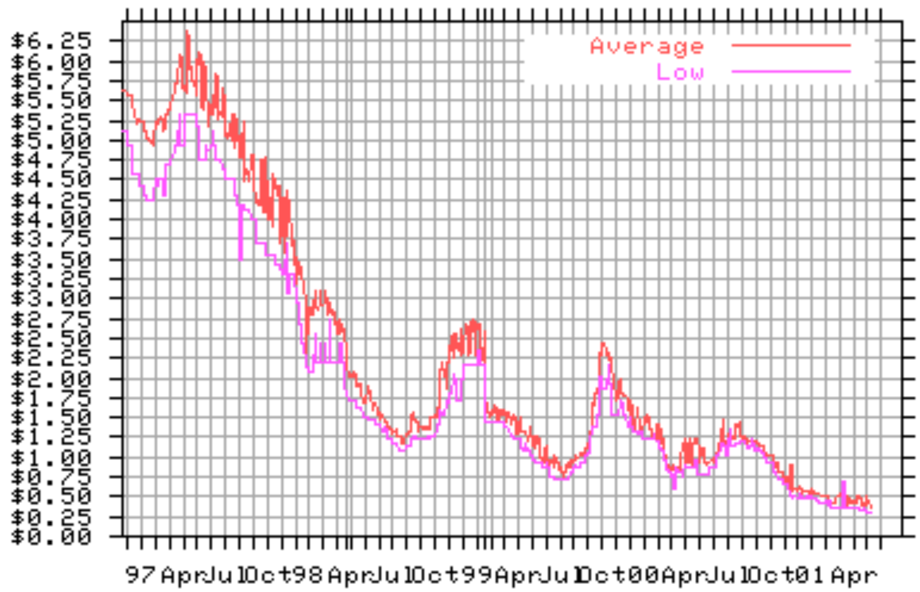
CSE 373: Data Structures and Algorithms

Lecture 19: Graphs

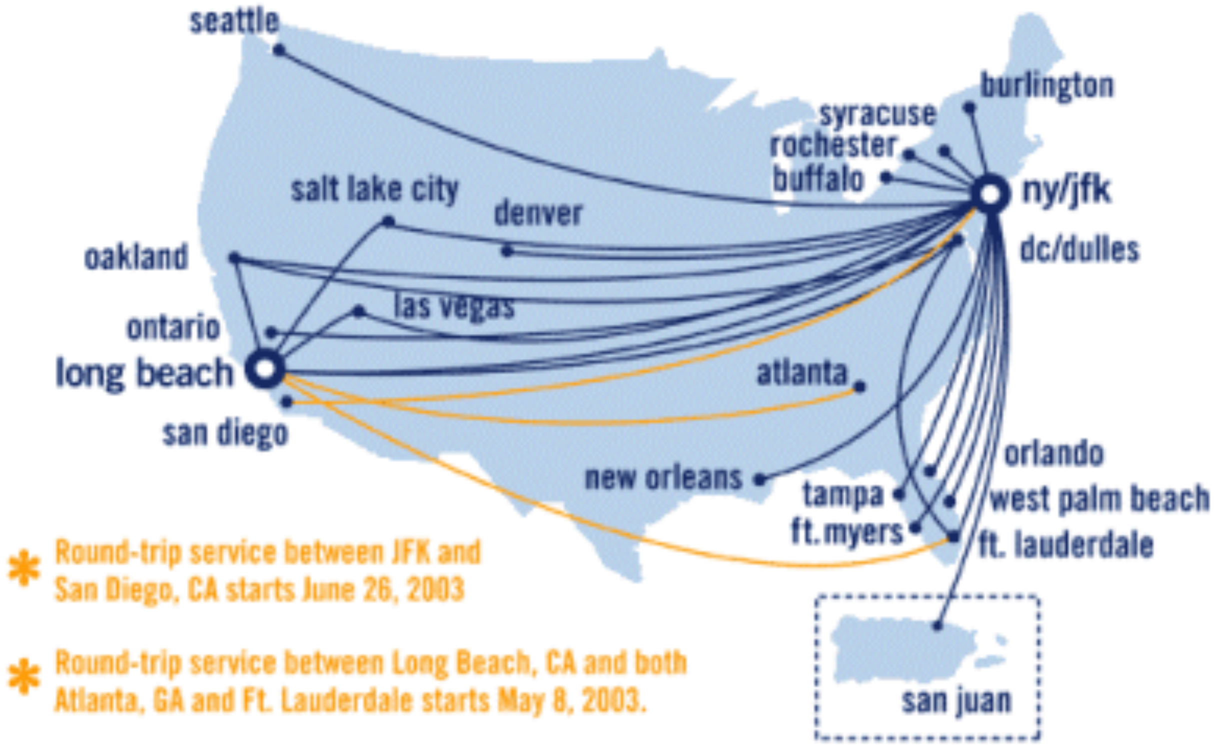
What are graphs?

- Yes, this is a graph….



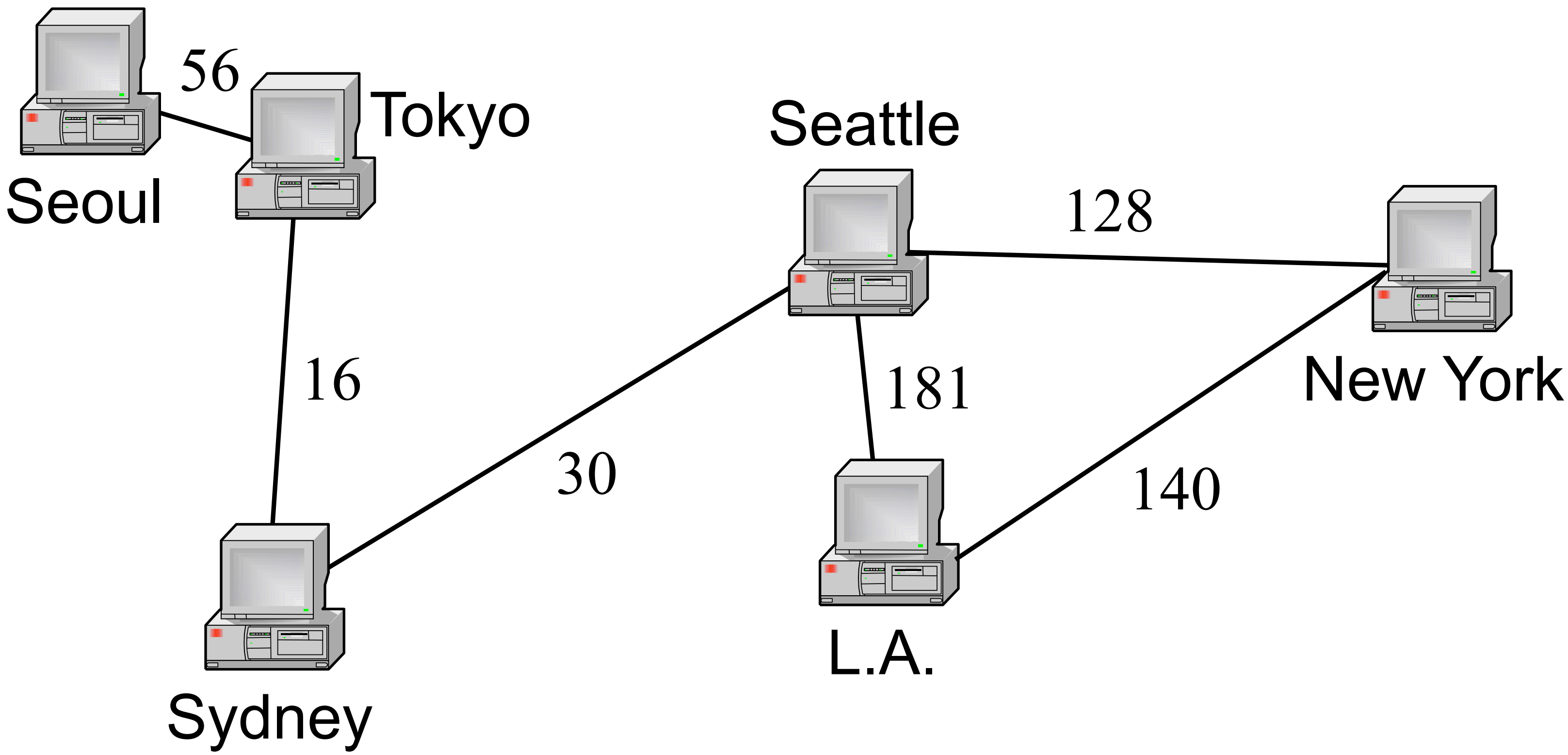
- But we are interested in a different kind of “graph”

Airline Routes



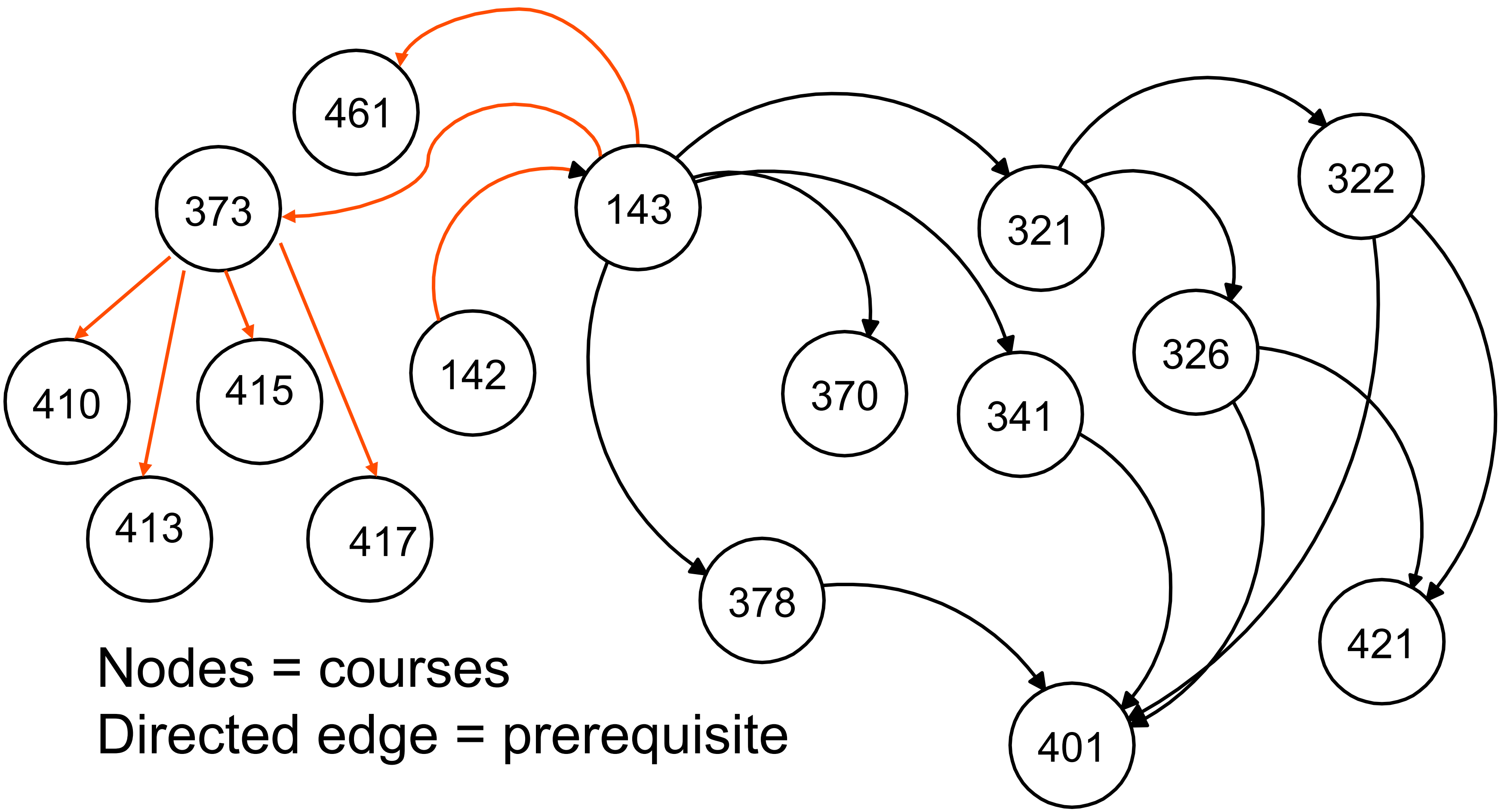
Nodes = cities
Edges = direct flights

Computer Networks



Nodes = computers
Edges = transmission rates

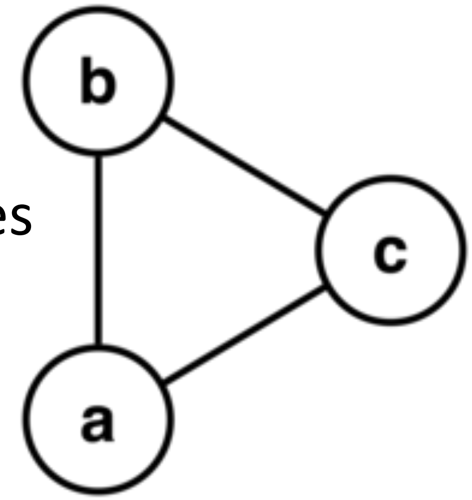
CSE Course Prerequisites at UW



Nodes = courses
Directed edge = prerequisite

Graphs

- **graph**: a data structure containing
  - a set of vertices *V*
  - a set of edges *E*, where an edge represents a connection between 2 vertices
  - $G = (V, E)$
  - edge is a pair (v, w) where v, w in V



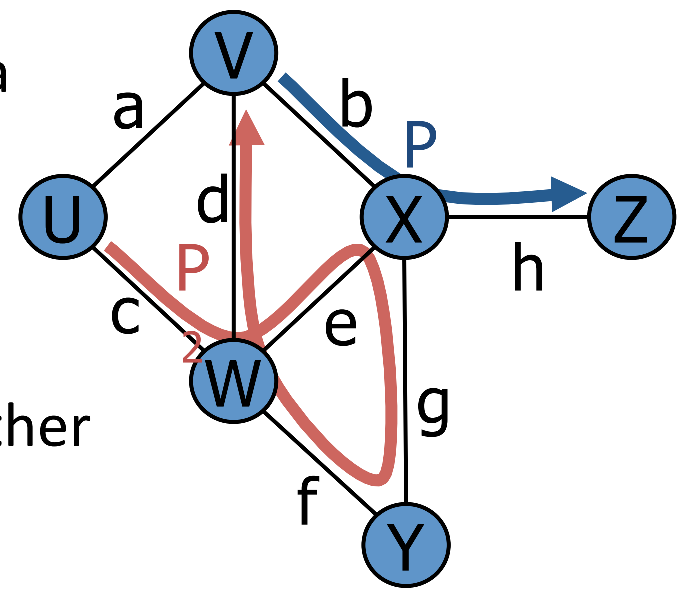
- the graph at right: $V = \{a, b, c\}$ and $E = \{(a, b), (b, c), (c, a)\}$
  - Assuming that a graph can only have one edge between a pair of vertices and cannot have an edge to itself, what is the maximum number of edges a graph can contain, relative to the size of the vertex set V?

Paths

- **path**: a path from vertex *A* to *B* is a sequence of edges that can be followed starting from *A* to reach *B*
  - can be represented as vertices visited or edges taken
  - example: path from V to Z: {b, h} or {V, X, Z}

- **reachability**: v_1 is *reachable* from v_2 if a path exists from V1 to V2

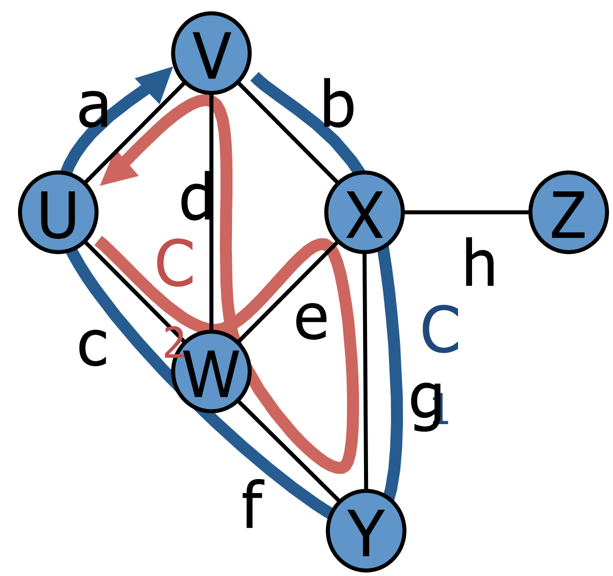
- **connected** graph: one in which it's possible to reach any node from any other
  - is this graph connected?



Cycles

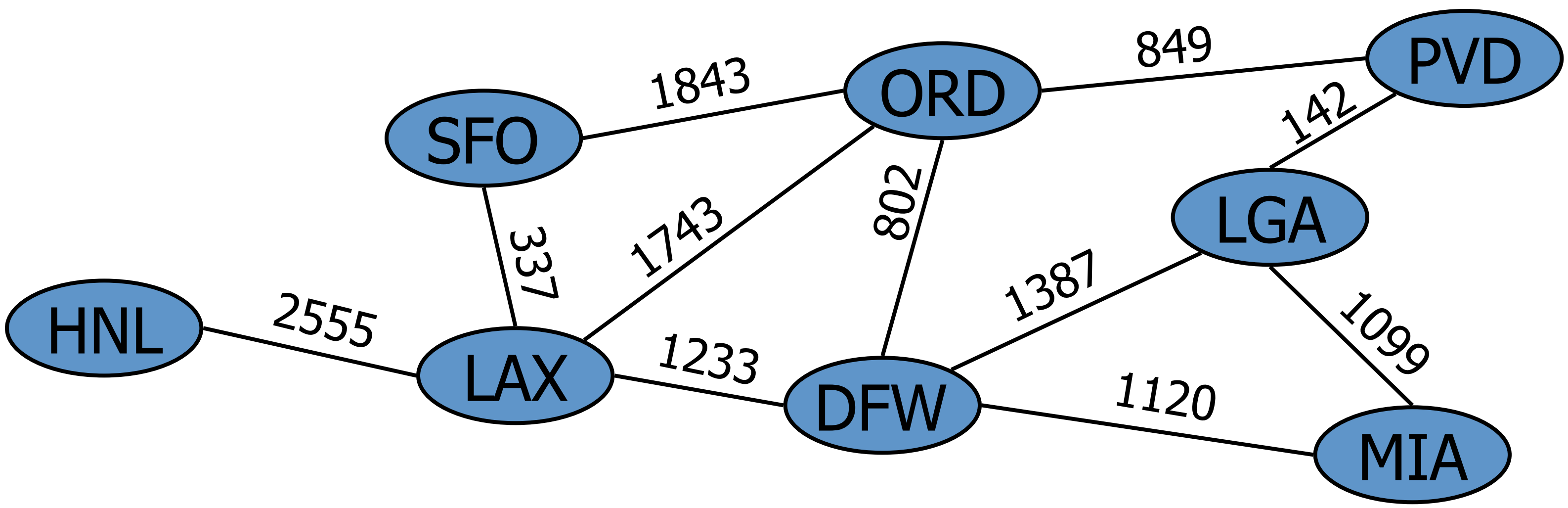
- **cycle**: path from one node back to itself
  - example: {b, g, f, c, a} or {V, X, Y, W, U, V}

- **loop**: edge directly from node to itself
  - many graphs don't allow loops



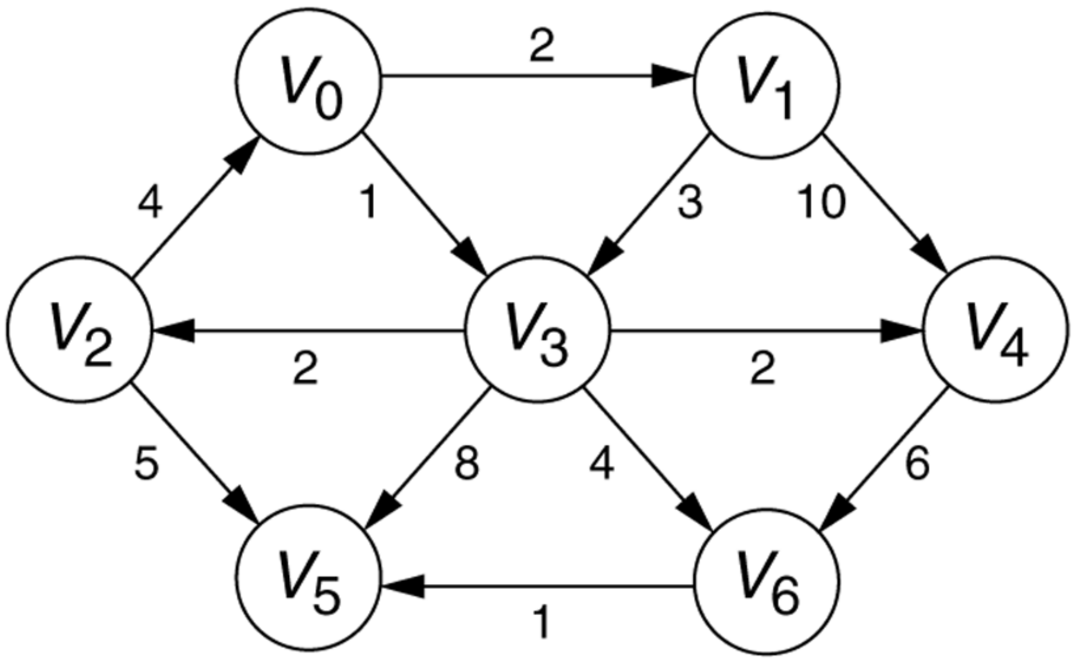
Weighted graphs

- **weight**: (optional) cost associated with a given edge
- example: graph of airline flights
  - if we were programming this graph, what information would we have to store for each vertex / edge?



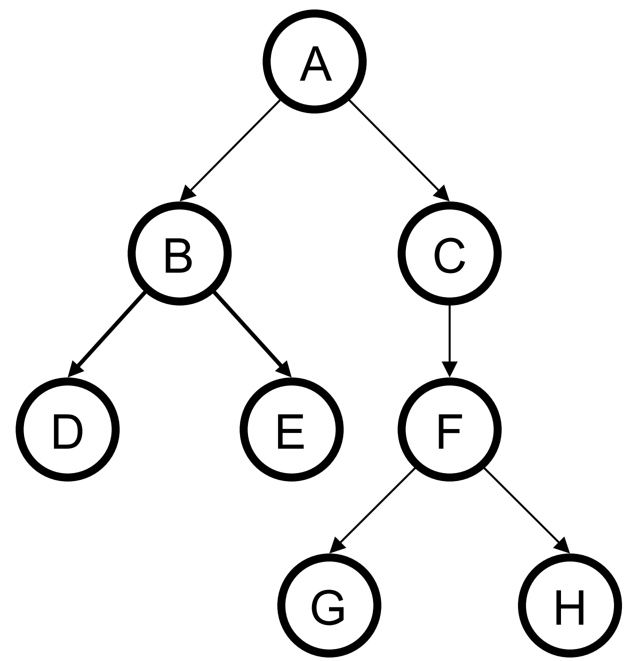
Directed graphs

- **directed graph (digraph)**: edges are one-way connections between vertices
  – if graph is directed, a vertex has a separate *in/out degree*



Trees as Graphs

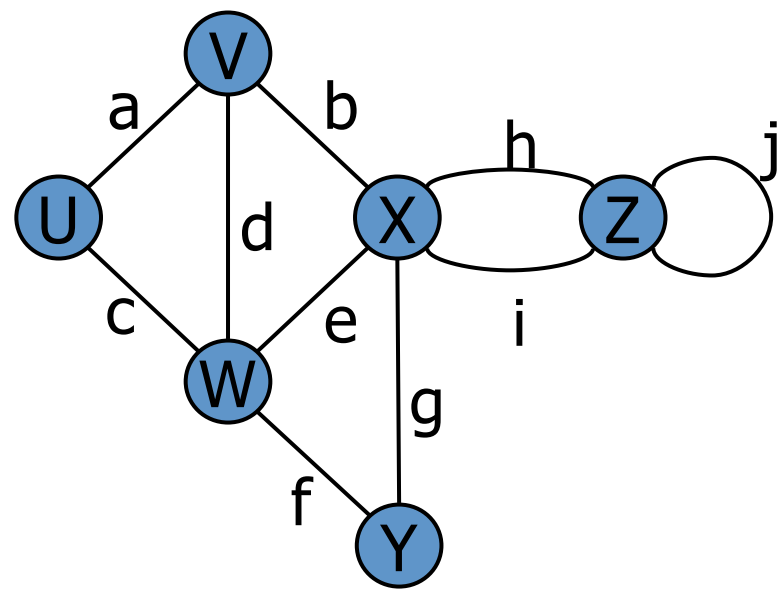
- Every tree is a graph with some restrictions:
  - the tree is directed
  - there is exactly one directed path from the root to every node



More terminology

- **degree**: number of edges touching a vertex
  – example: W has degree 4
  – what is the degree of X? of Z?

- **adjacent** vertices: connected directly by an edge



Graph questions

- Are the following graphs directed or not directed?
  - Buddy graphs of instant messaging programs? (vertices = users, edges = user being on another's buddy list)
  - bus line graph depicting all of Seattle's bus stations and routes
  - graph of movies in which actors have appeared together

- Are these graphs potentially cyclic? Why or why not?

Graph exercise

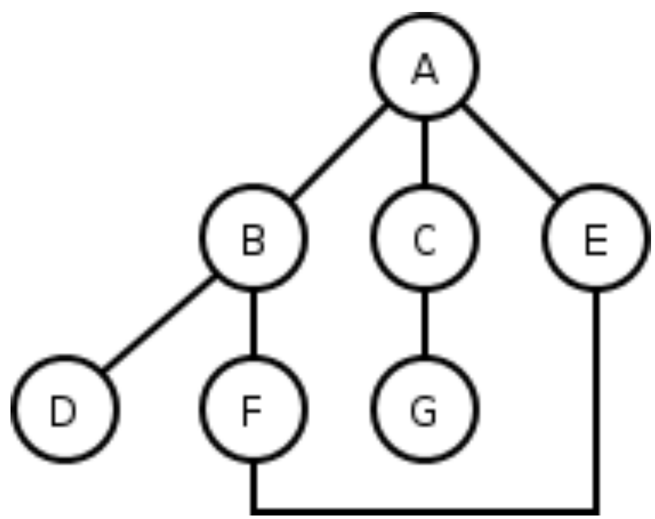
- Consider a graph of instant messenger buddies.
  - What do the vertices represent? What does an edge represent?
  - Is this graph directed or undirected? Weighted or unweighted?
  - What does a vertex's degree mean? In degree? Out degree?
  - Can the graph contain loops? cycles?

- Consider this graph data:
  - Jessica's buddy list: Meghan, Alan, Martin.
  - Meghan's buddy list: Alan, Lori.
  - Toni's buddy list: Lori, Meghan.
  - Martin's buddy list: Lori, Meghan.
  - Alan's buddy list: Martin, Jessica.
  - Lori's buddy list: Meghan.

  - Compute the in/out degree of each vertex. Is the graph connected?
  - Who is the most popular? Least? Who is the most antisocial?
  - If we're having a party and want to distribute the message the most quickly, who should we tell first?

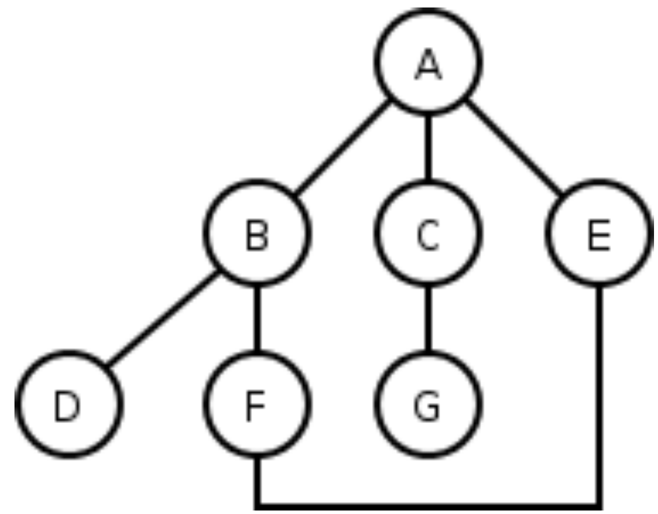
Depth-first search

- **depth-first search (DFS)**: finds a path between two vertices by exploring each possible path as many steps as possible before backtracking
  - often implemented recursively



DFS example

- All DFS paths from A to others (assumes ABC edge order)
  - A
  - A -> B
  - A -> B -> D
  - A -> B -> F
  - A -> B -> F -> E
  - A -> C
  - A -> C -> G



- What are the paths that DFS did not find?

DFS pseudocode

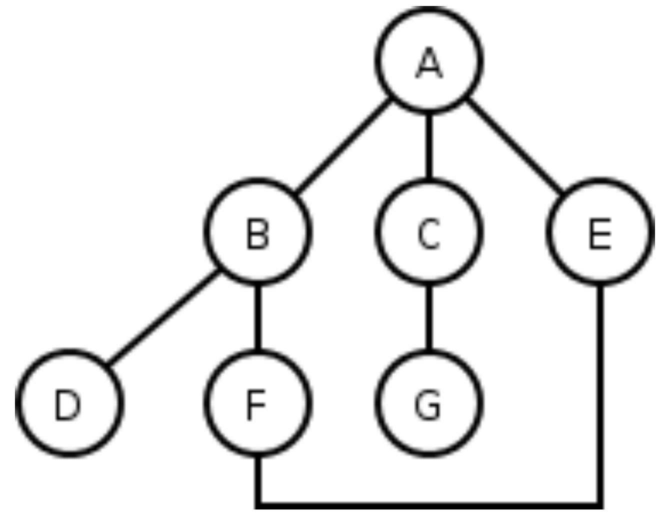
- Pseudo-code for depth-first search:

  dfs(v1, v2):
      dfs(v1, v2, {})

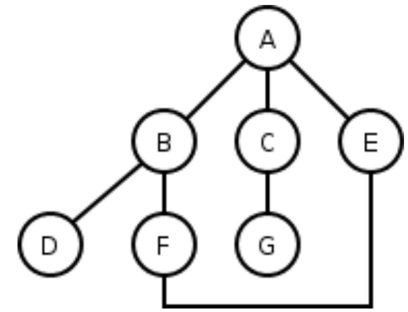
  dfs(v1, v2, path):
      path += v1.
      mark v1 as visited.
      if v1 is v2:
          path is found.

      for each unvisited neighbor v_i *of v1*
      where there is an edge from v1 to v_i*:*
          if dfs(v_i*, v2, path) finds a path, path is found.*

      path -= v1.   path is not found.



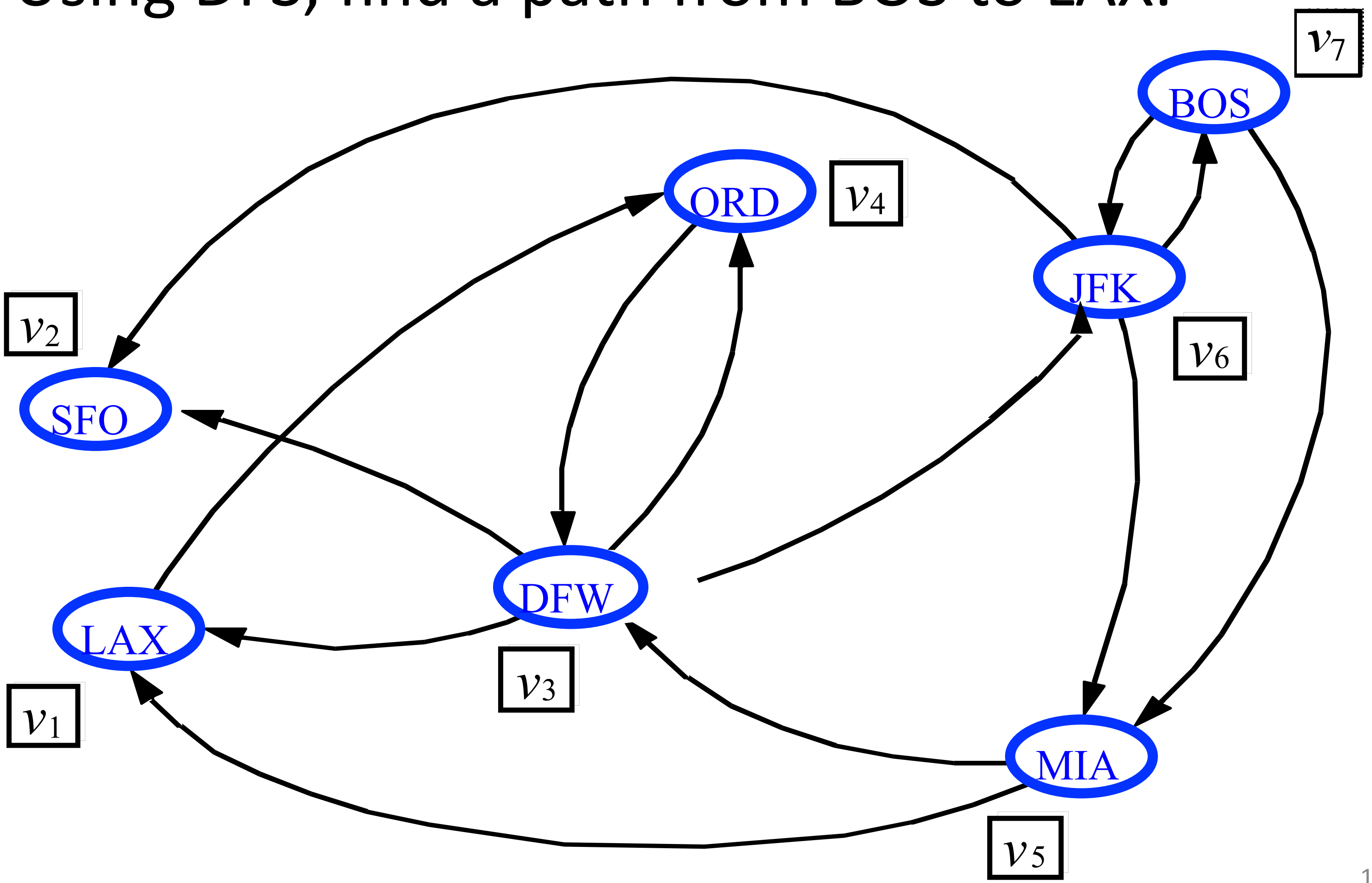
DFS observations



- guaranteed to find a path if one exists
- easy to retrieve exactly what the path is (to remember the sequence of edges taken) if we find it
- *optimality*: not optimal. DFS is guaranteed to find <u>a</u> path, not necessarily <u>the</u> best/shortest path
  - Example: DFS(A, E) may return
    A -> B -> F -> E

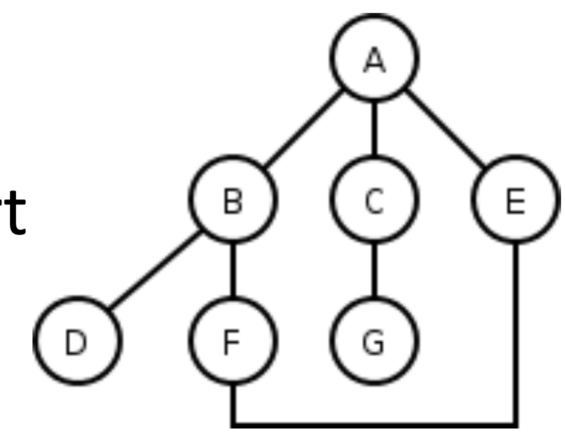
Another DFS example

- Using DFS, find a path from BOS to LAX.



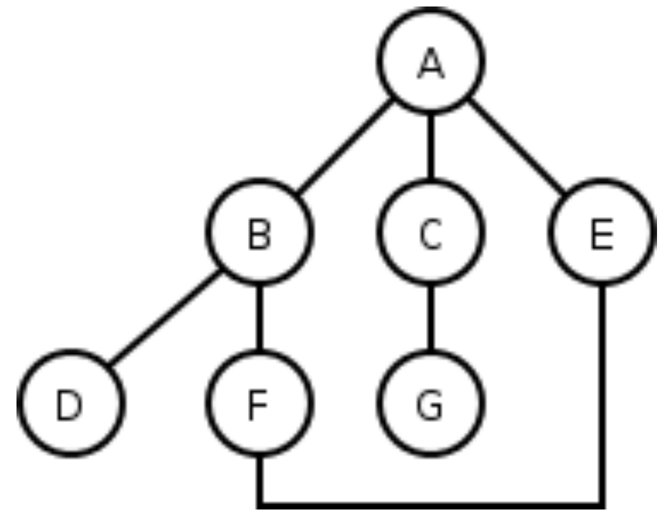
Breadth-first search

- **breadth-first search (BFS)**: finds a path between two nodes by taking one step down all paths and then immediately backtracking
  - often implemented by maintaining a list or queue of vertices to visit
  - BFS always returns the path with the fewest edges between the start and the goal vertices



BFS example

- All BFS paths from A to others (assumes ABC edge order)
  - A
  - A -> B
  - A -> C
  - A -> E
  - A -> B -> D
  - A -> B -> F
  - A -> C -> G



- What are the paths that BFS did not find?

BFS pseudocode

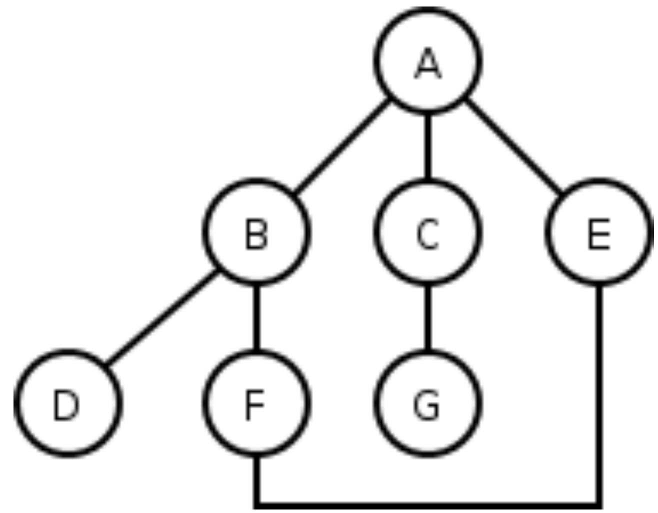
- Pseudo-code for breadth-first search:

```
bfs(v1, v2):
   List := {v1}.
   mark v1 as visited.

   while List not empty:
      v := List.removeFirst().
      if v is v2:
         path is found.

      for each unvisited neighbor v_i of v
      where there is an edge from v to v_i:
         mark v_i as visited
         List.addLast(v_i).

   path is not found.
```



BFS observations

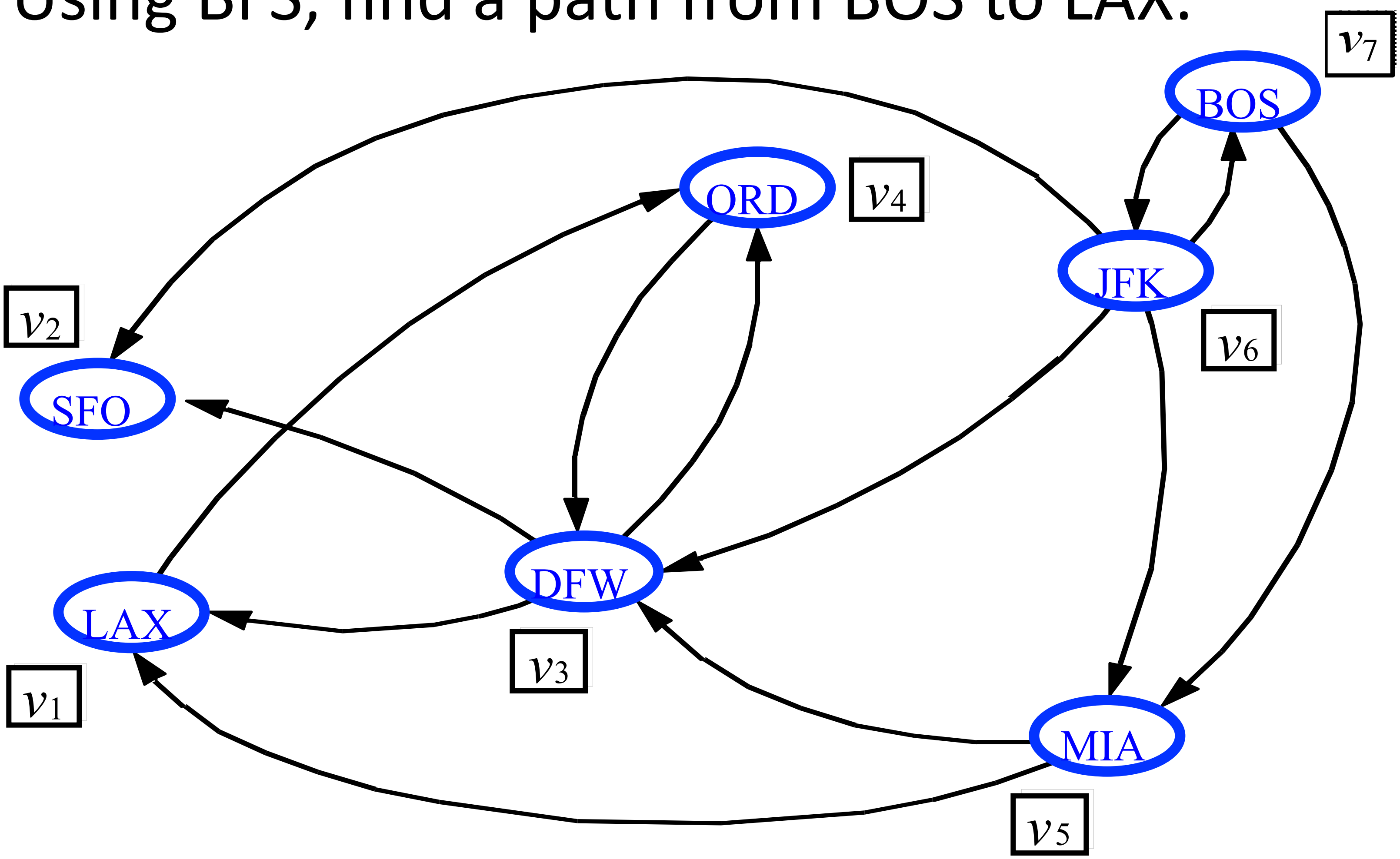
- *optimality*:
  - in unweighted graphs, optimal. (fewest edges = best)
  - In weighted graphs, not optimal.
    (path with fewest edges might not have the lowest weight)

- *disadvantage*: harder to reconstruct what the actual path is once you find it
  - conceptually, BFS is exploring many possible paths in parallel, so it's not easy to store a Path array/list in progress

- observation: any particular vertex is only part of one partial path at a time
  - We can keep track of the path by storing *predecessors* for each vertex (references to the previous vertex in that path)

Another BFS example

- Using BFS, find a path from BOS to LAX.



DFS, BFS runtime

- What is the expected runtime of DFS, in terms of the number of vertices V and the number of edges E ?
- What is the expected runtime of BFS, in terms of the number of vertices V and the number of edges E ?
- Answer: $O(|V| + |E|)$
  - each algorithm must potentially visit every node and/or examine every edge once.
  - why not $O(|V| * |E|)$?
- What is the space complexity of each algorithm?