T minus 1 class

- “Homework” 10 will be out shortly
  - About 6 questions
- No more homeworks after that!
- No quiz next week
- Next class will be review

CMSC 203: Lecture 26

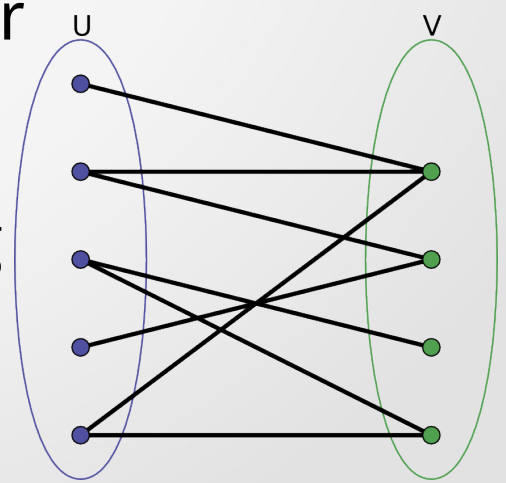
Graphs Cont.

Special Graphs

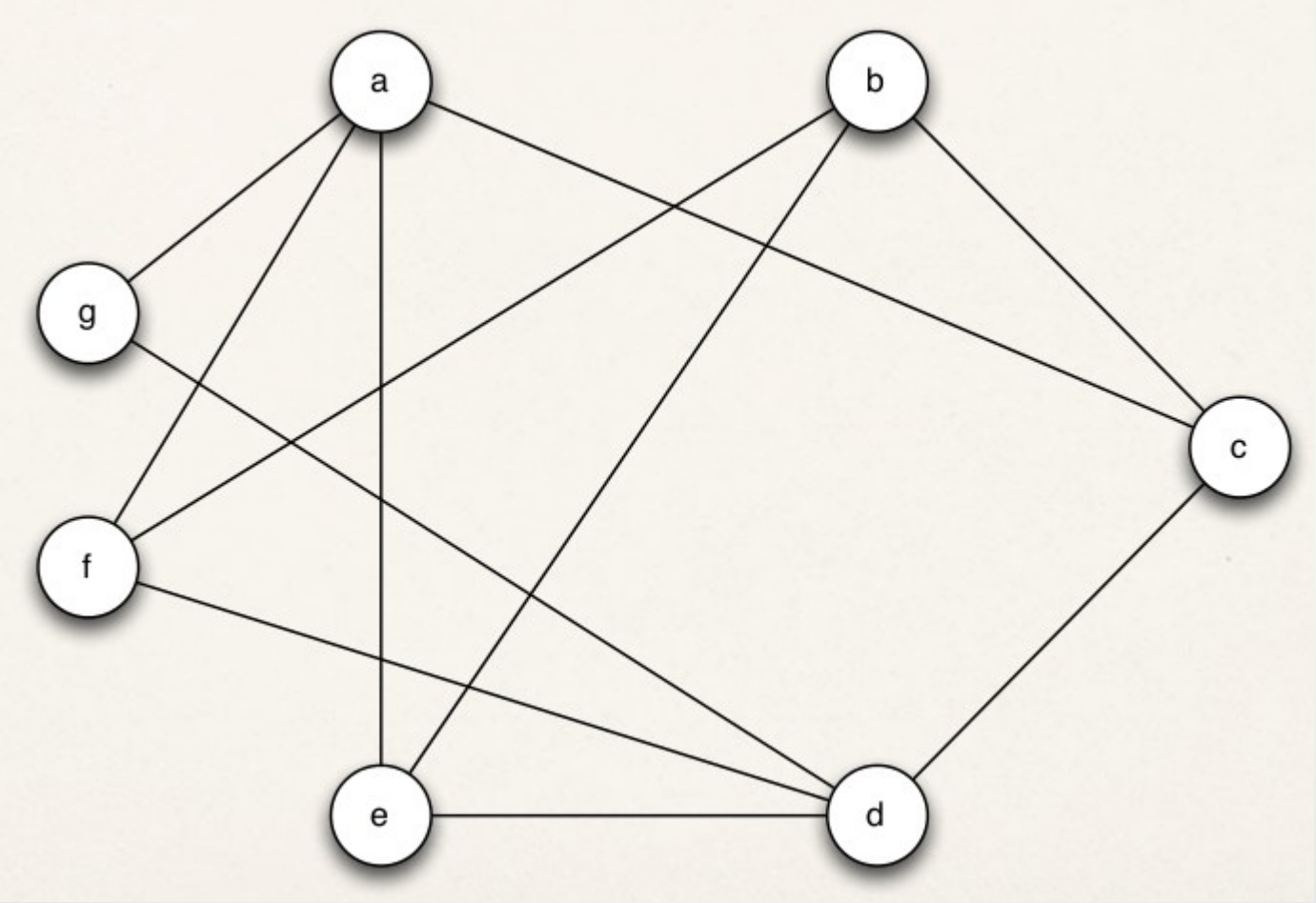
- Complete Graph: K_n
  - Simple graph that contains one edge between each pair of distinct vertices
- Cycles: C_n
  - Consists of *n* vertices $v_1, v_2, \ldots, v_n$ and edges such that $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \text{ and } \{v_n, v_1\}$
- Wheel: W_n
  - Add additional vertex to C_n and connect vertex to each of the n vertices

Bipartite Graphs

- A simple graph is bipartite if its vertices can be partitioned into two disjoint sets such that every edge connects a vertex in both sets
  - No edge connects vertices in the same set
- Can be verified if and only if it is possible to assign one of two different colors to each vertex of the graph so no two adjacent vertices are assigned the same color
  - Known as **Graph coloring**
- Useful for applications that involve matching elements between sets



Bipartite Example

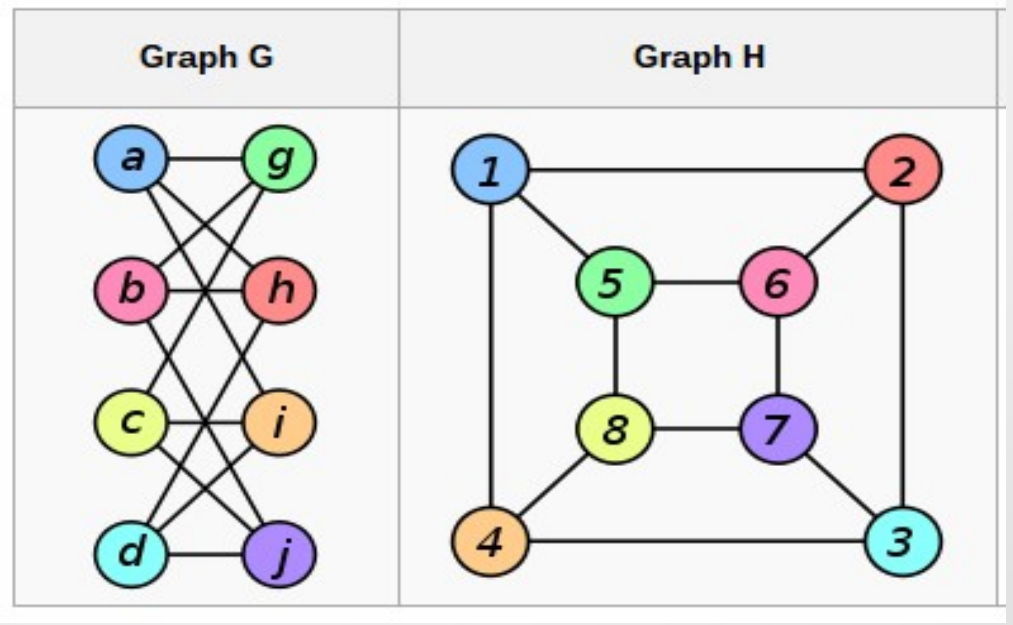


Representing Graphs

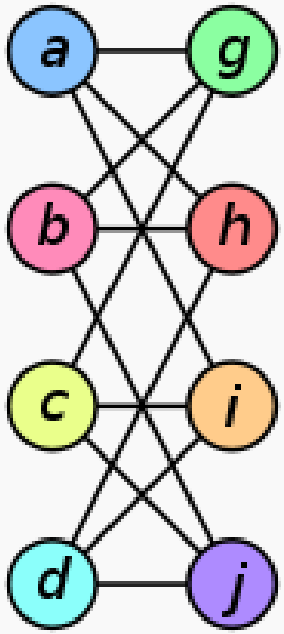
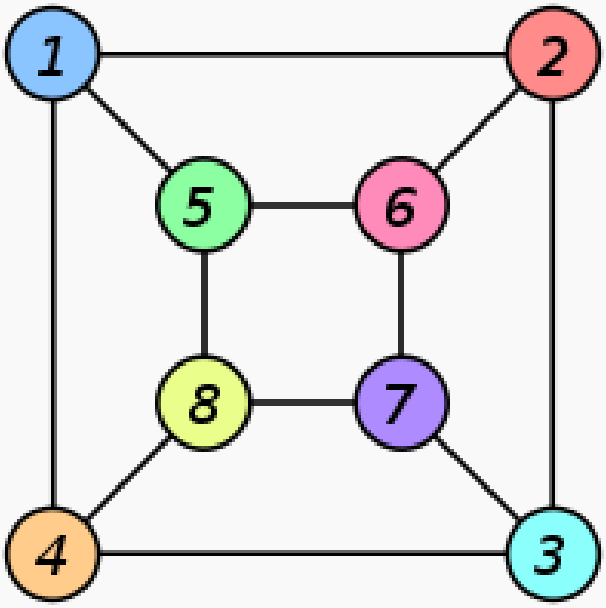
- Adjacency List
  - For each vertex, list the adjacent vertices
  - Good for **sparse** graphs (few edges)
- Adjacency Matrix
  - 2D matrix of {0, 1}, similar to relation matrix
    - 1 if edge from *i* to *j*
  - Good for **dense** graph (many edges)

Isomorphism

- Two graphs are **isomorphic** if they are same structure
- You can re-arrange vertices and have identical graphs
- G_1 and G_2 are isomoprhic iff ∃ a bijection *f* from V_1 to V_2 such that *a* and *b* are adjacent in G_1 iff f(a) and f(b) are adjacent in G_2



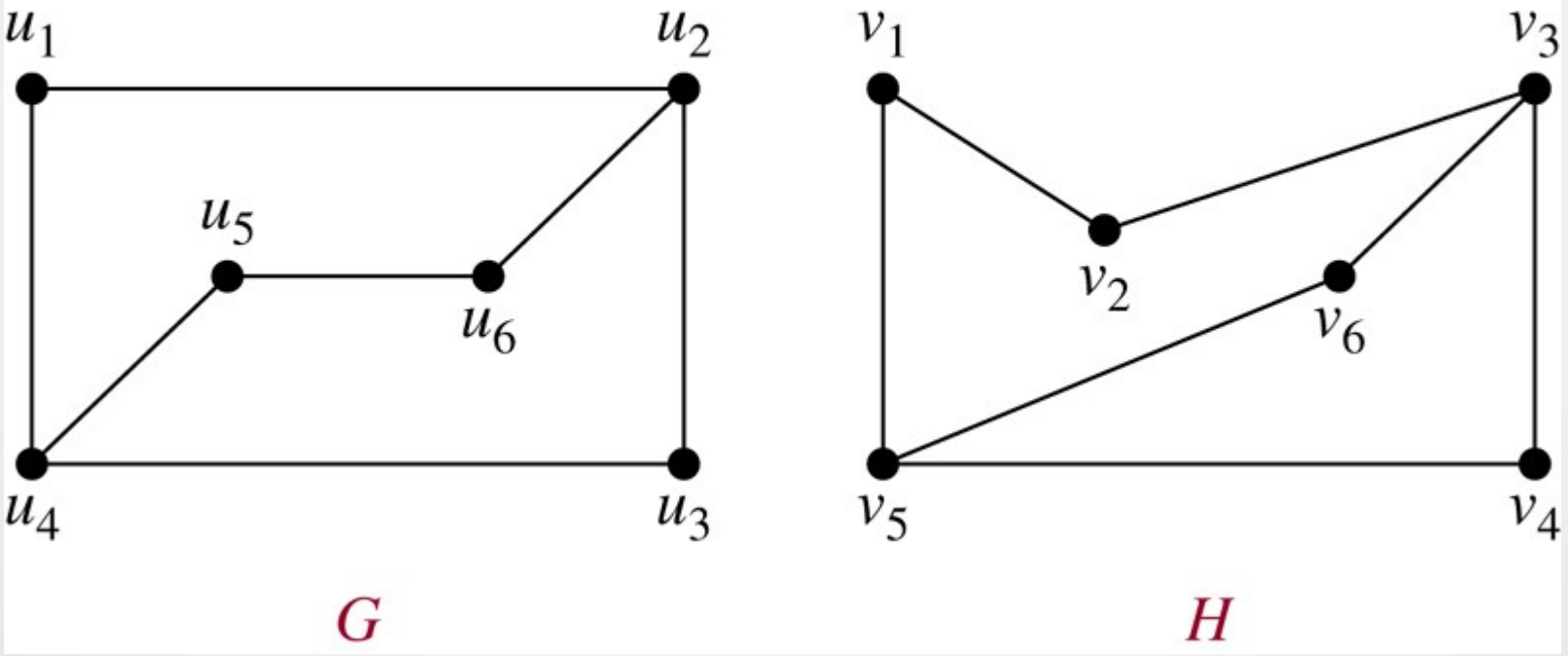
Isomorphism Example

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Graph Invariants

- A property preserved by isomorphism is a **graph invariant**
- Examples:
  - Number of vertices
  - Number of edges
  - Degrees of vertices must be the same
- Failing above invariants means graphs are not isomorphic (but this is not true in reverse)

Isomorphism Example



Path

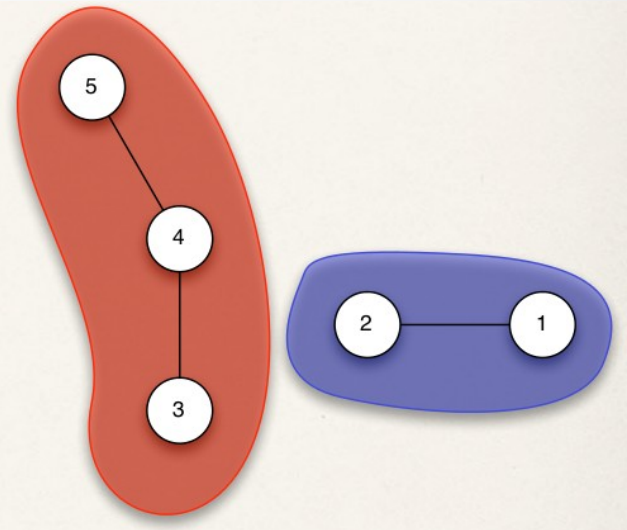
- A **path** is *p* is a set of edges such that:
  - p.start = e_0.start
  - p.end = e_n.end
  - e_n.end = e_{n+1}.start
- Basically, a series of edges from a start vertex, following connected edges, until an end vertex
- If p.start = p.end, it is a **cycle**
- If no edge is repeated, it is **simple**

Connectedness

- Two nodes are **connected** iff there is a path between
  - Otherwise, nodes are **disconnected**
- If every pair of nodes is connected, then the graph is **connected**
  - Otherwise, graph is **disconnected**
- A connected graph contains a simple path between every pair of vertices

Components

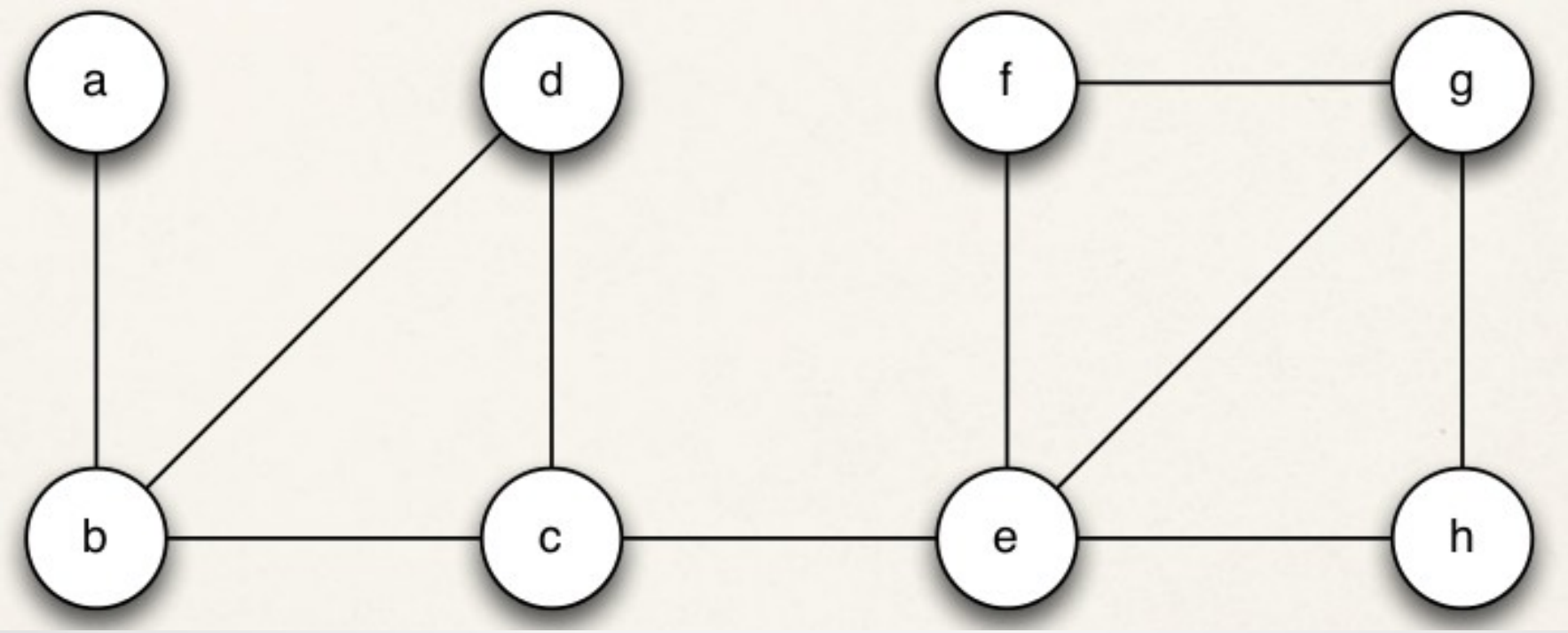
- A connected subgraph of G that is not a proper subgraph of any other connected subgraph of G is a **connected component**
- A connected graph has 1 connected component, whereas a disconnected graph has 2+



Cuts

- If removing a vertex increases the number of connected components, that vertex is a **cut vertex**
- If removing an edge increases the number of connected components, that edge is a **cut edge**
- If no such cut vertex or cut edge exists, the graph is called **nonseparable**

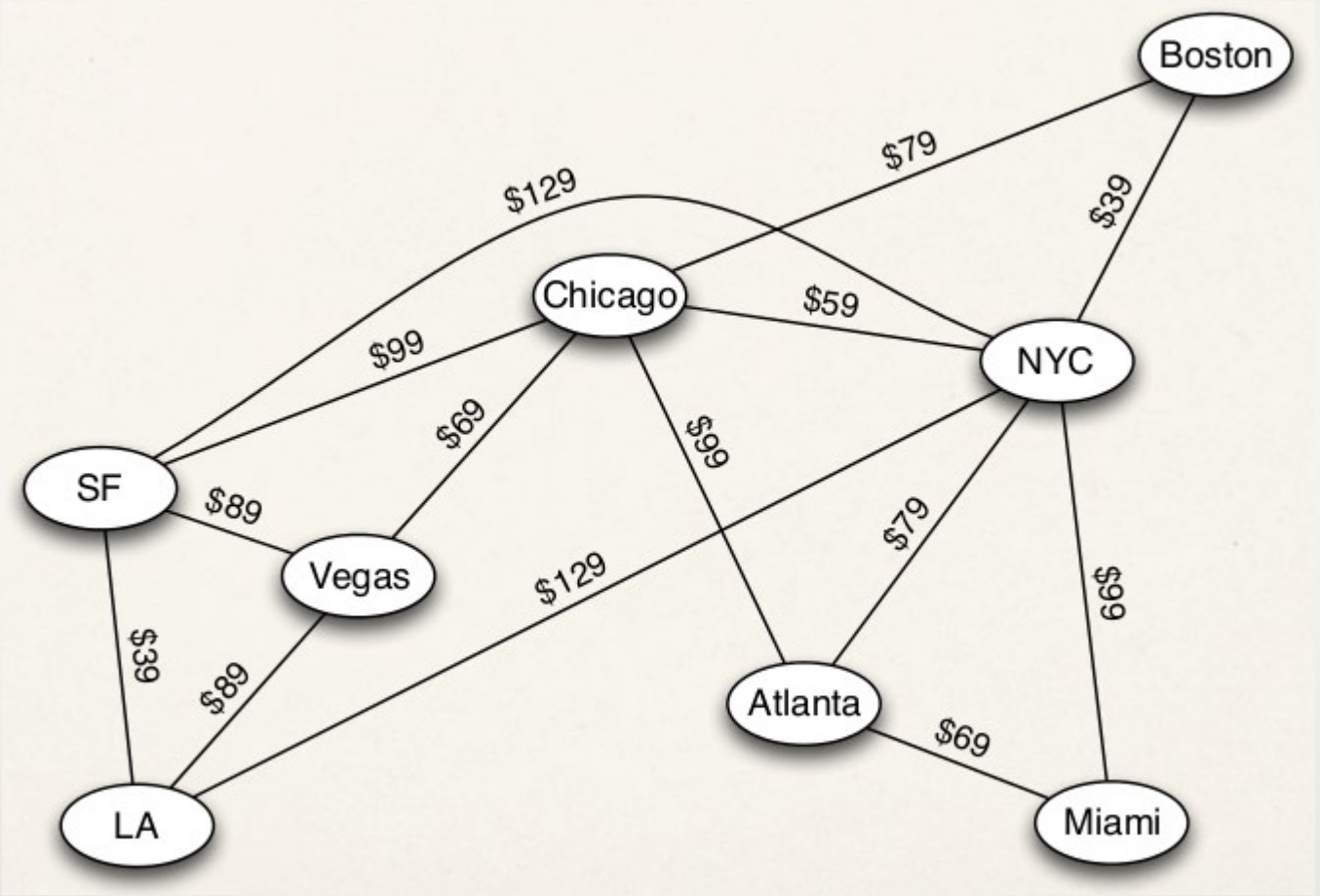
Cut Practice



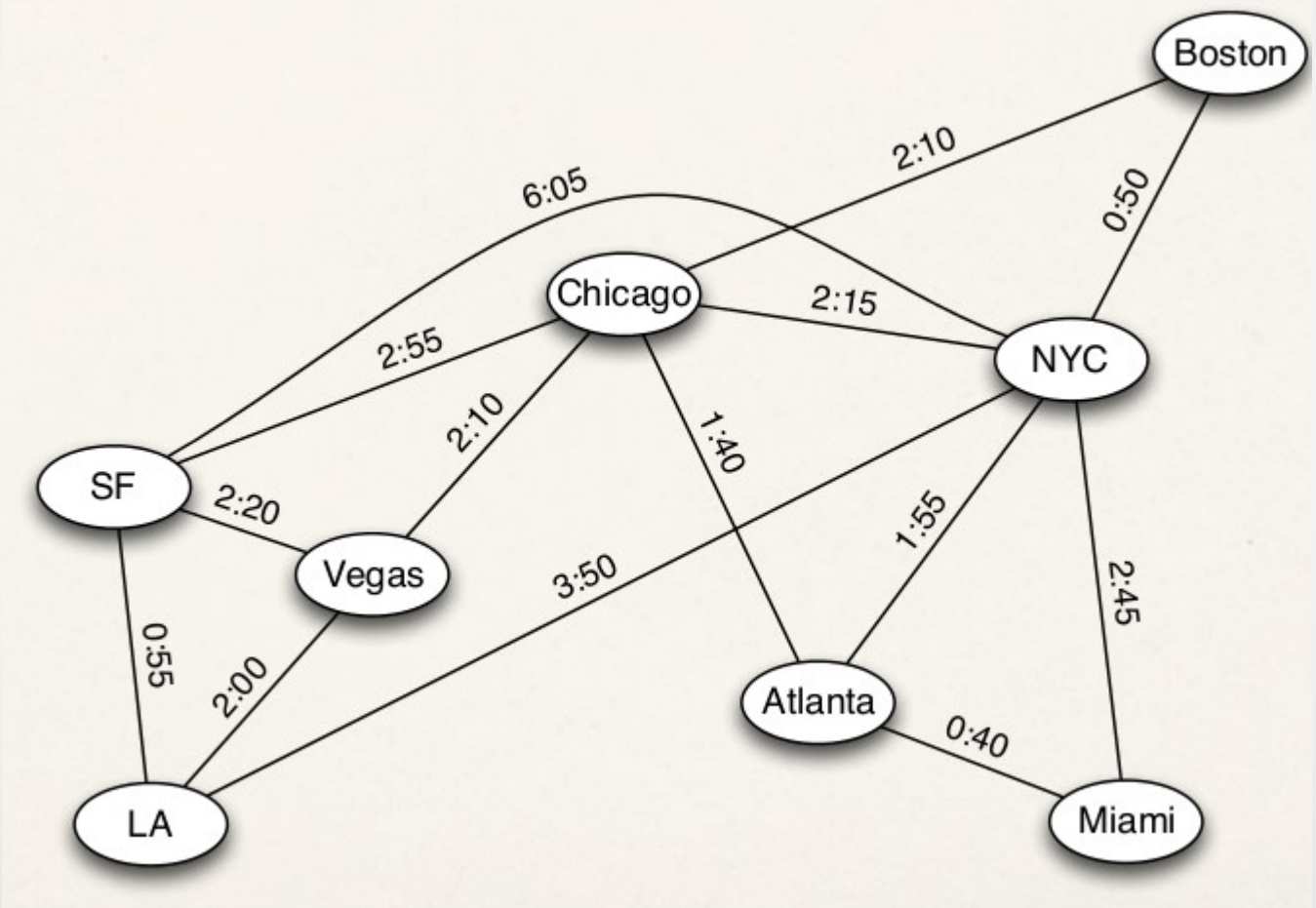
Shortest-Path Problems

- Useful on **weighted graphs** where weights are **costs**
- The **length** of the path is the sum of the weights of edges in the path
- We minimize the path length between two vertices
- Examples:
  - Airplane flight time
  - Driving to minimize time, gas, fares, etc.
  - Time to send messages

Cost as Fare


Boston
Chicago
NYC
SF
Vegas
Atlanta
LA
Miami
$129
$79
$39
$99
$59
$69
$99
$89
$79
$99
$39
$89
$129
$69

Cost as Time


Boston
Chicago
NYC
SF
Vegas
Atlanta
LA
Miami
6:05
2:10
0:50
2:55
2:15
2:10
1:40
2:20
1:55
2:45
3:50
0:55
2:00
0:40

Dijkstra's Algorithm

- Finds shortest path in $O(n^2)$ time
- Asymptotically fastest shortest-path algorithm, given:
  - single-source
  - arbitrary directed graphs
  - unbounded weights
  - non-negative weights

Dijkstra's Algorithm

- Greedy Algorithm
  - Start with C = {X}
  - Search for neighbors of C for next closest vertex not already in C
    - Add the new vertex to C
    - Repeat until end point is the next closest point

Dijkstra Example