CSCI 136
Data Structures &
Advanced Programming

Bill Jannen

Lecture 32

May 3, 2017

Administrative Details

- Lab 10 Today/Thursday
  - Bring design documents to lab!
- Quick Hexapawn Demo?

- Lab 10 is the last lab…
  - Next week we will post a sample exam instead
- Review session:
  - Monday 5/15 from 7-9pm in Physics 203

Last Time (Monday)

- Finished Binary Search Trees
  - predecessor(), remove()
  - Game Trees
    - For lab, you don't need to implement backwards induction
    - ComputerPlayer moves randomly, but…
    - Each time ComputerPlayer loses, prune losing node

Today's Outline

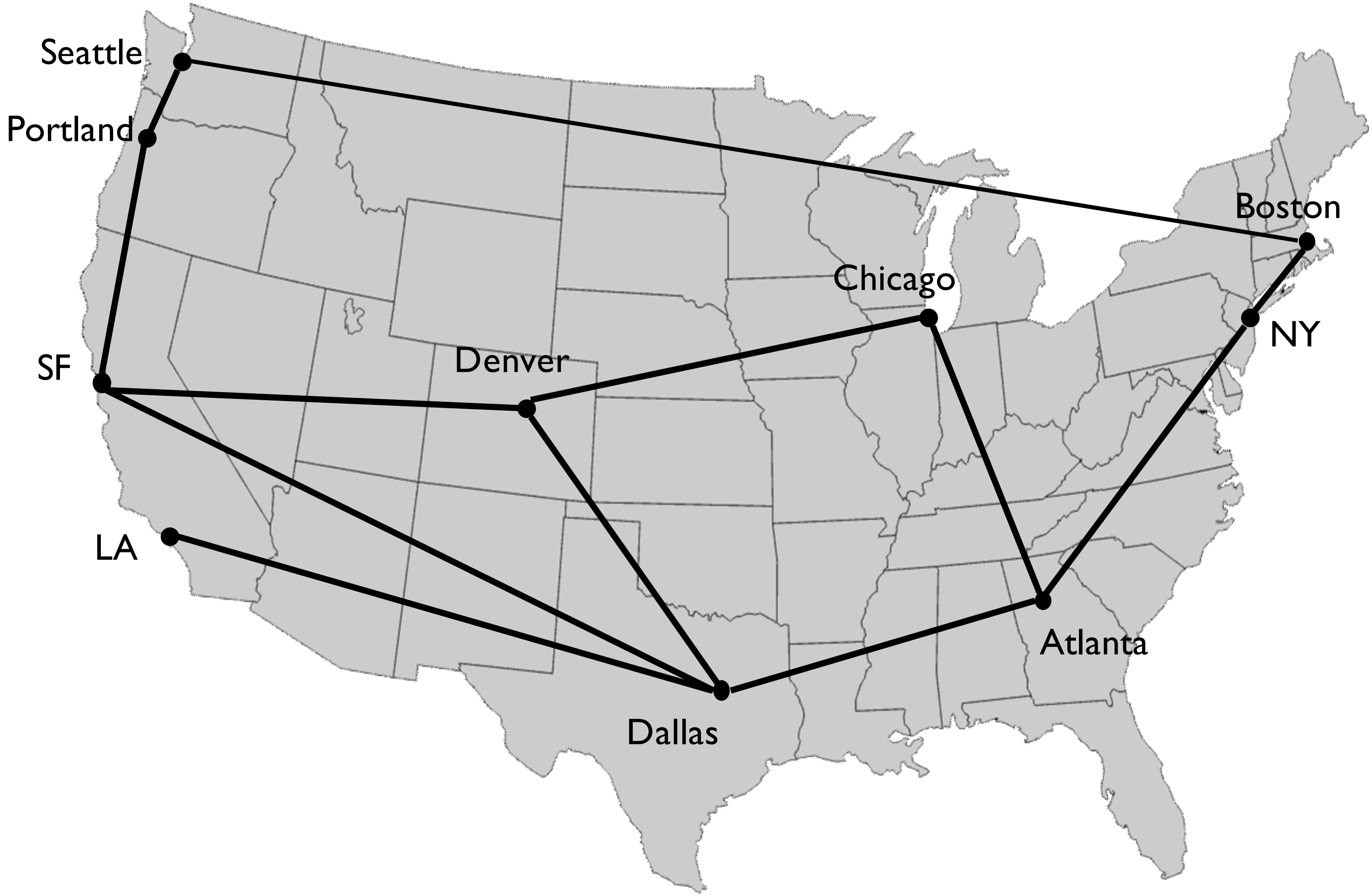
- Introduction to Graphs!
  - Examples
  - Terminology
  - 2 Representations

Putting Data Structures in Context

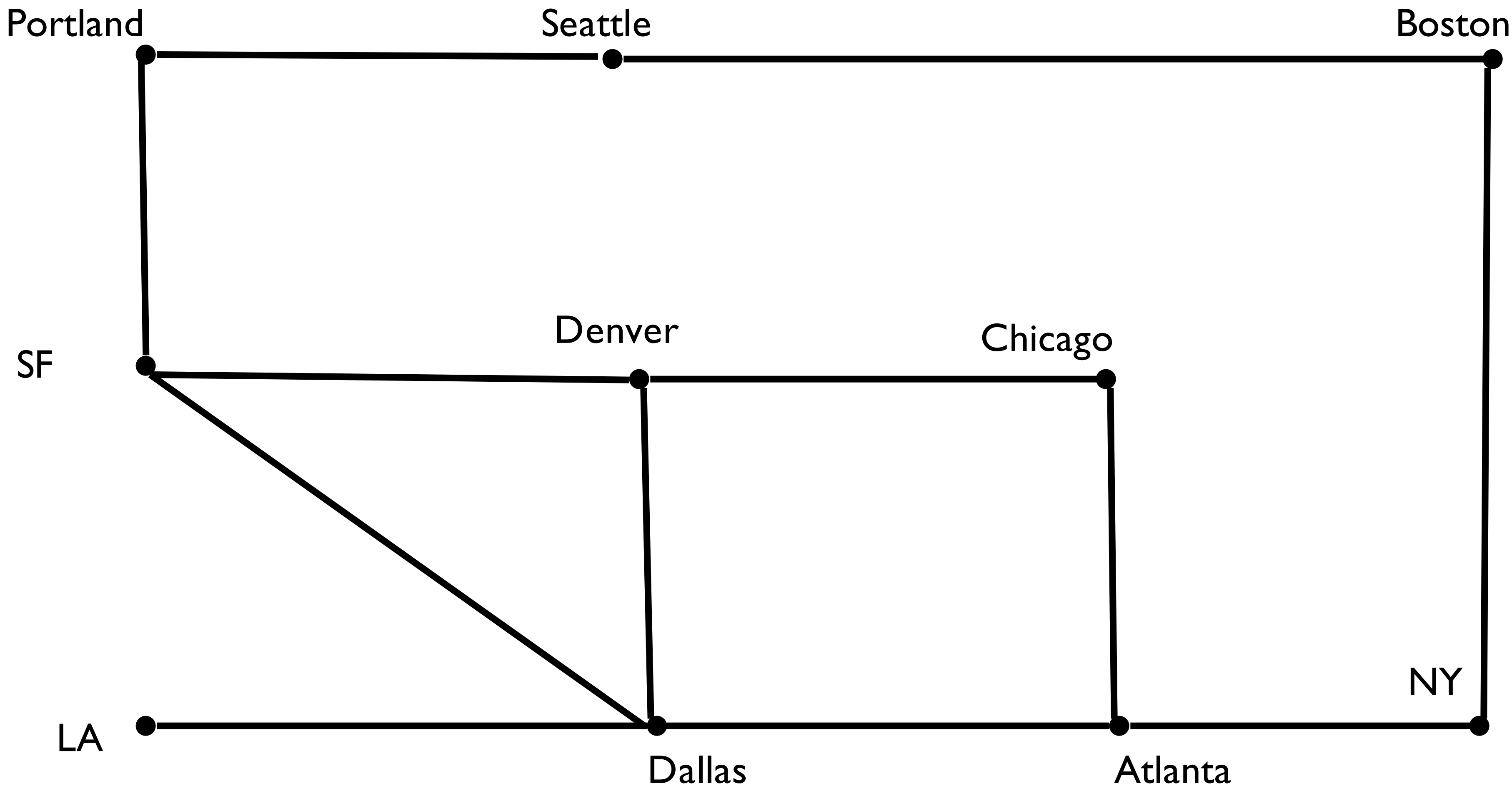
- Types of data structures
  - Basic - Lists/Vectors (no ordering relation)
  - Linear – Stacks/Queues (ordered by insertion)
  - Ordered Structures – value ordering
  - Tree - hierarchical ordering
  - BST - value ordering (in a hierarchical fashion)
- Next up: Graphs
  - The most general way to describe relationships between data

Graphs

- Definition
  - A graph is a collection of **vertices** (nodes) and **edges** (links) connecting them
- Let's use real world examples for intuitions



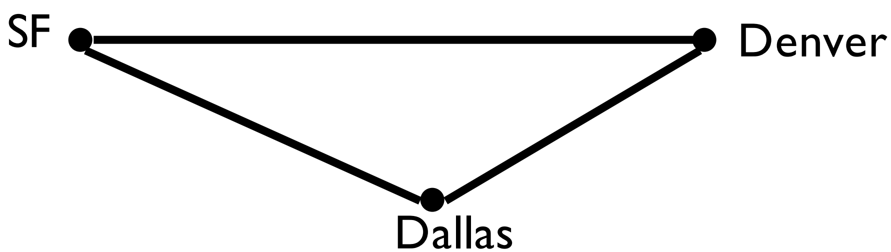
Nodes = cities; Edges = lines connecting cities



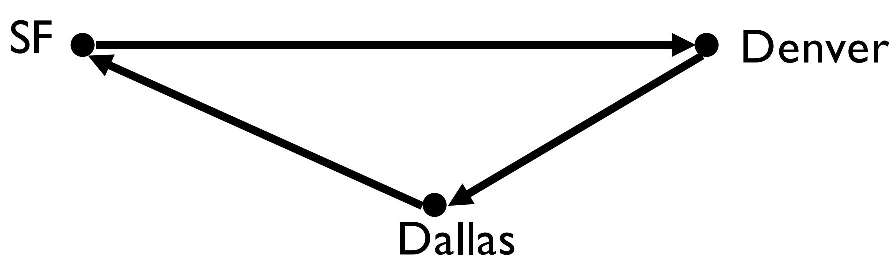
Note: Structure of graph matters, not actual placement of nodes

Types of Graphs

- Undirected
  - All edges are bi-directional

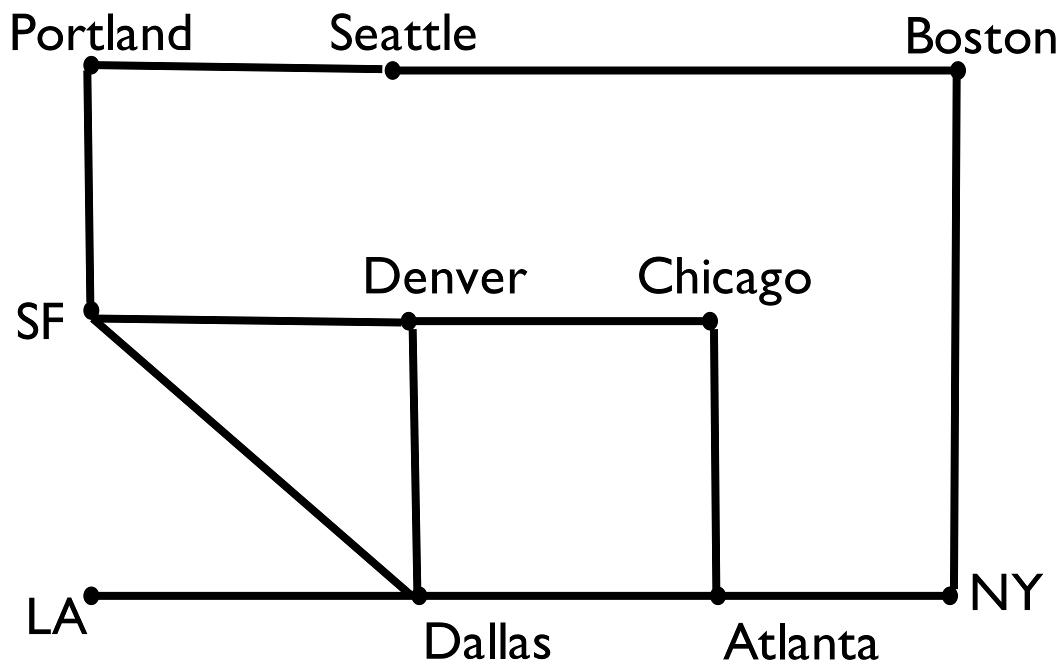


- Directed
  - Edges have a source and destination



Paths

- A *path* is a sequence of distinct edges between two nodes
- A cycle is a path that starts and ends at the same node

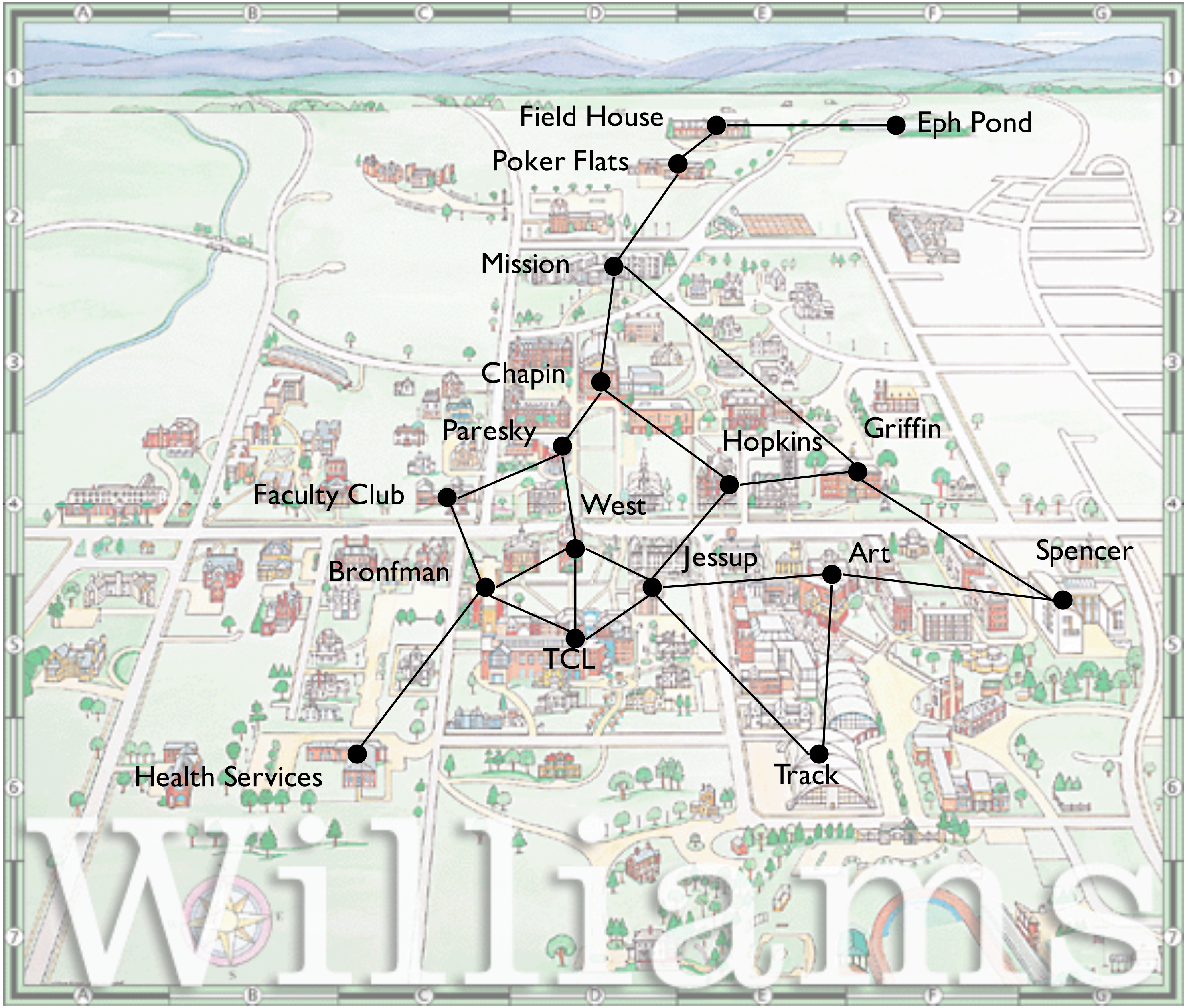


- Questions:
  - What is the shortest path from SF to Boston?
  - What is the shortest *cycle* from SF to SF that goes through Dallas and Chicago?

Connectedness

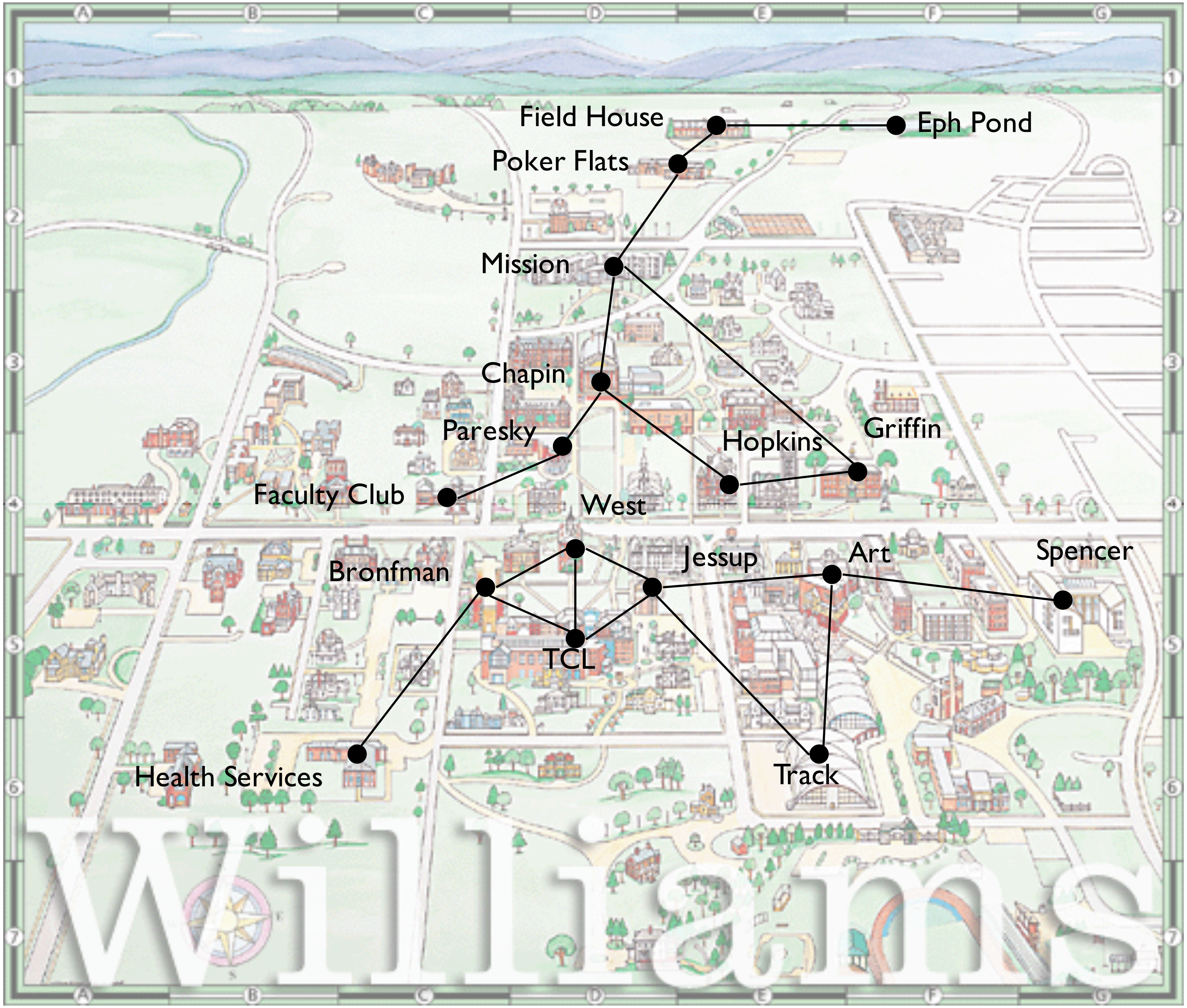
- Nodes U and V are **connected** if there is an edge between U and V

- A **connected component** is a set *S* where there is a path between every vertex pair in *S*
- A **fully connected component** is a set *S* where there is an edge between every pair of vertices in *S*



Field House
Eph Pond
Poker Flats
Mission
Chapin
Paresky
Griffin
Hopkins
Faculty Club
West
Jessup
Art
Spencer
Bronfman
TCL
Health Services
Track
Williams

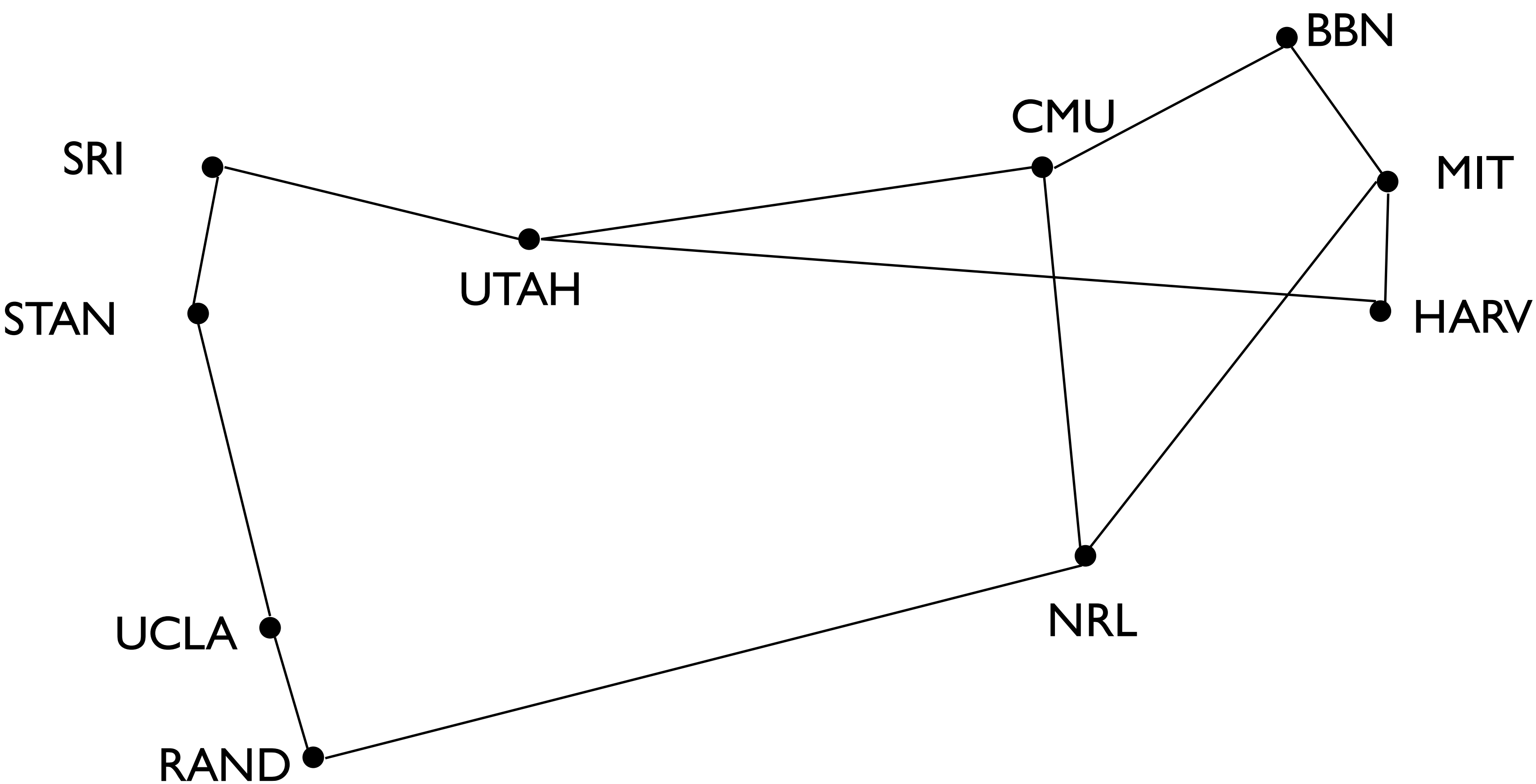
What's *reachable* from TCL?



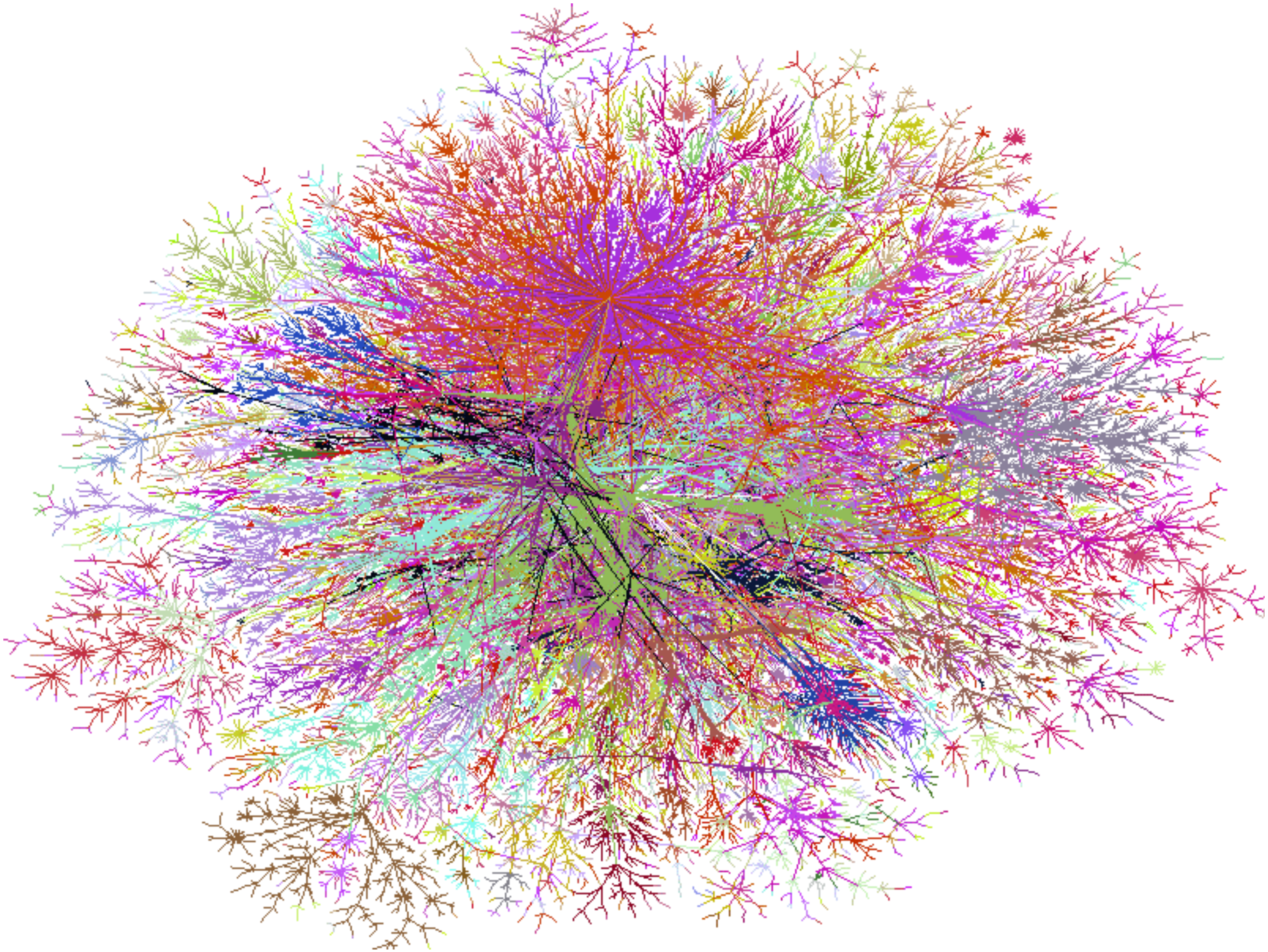
Graph Applications

- Connectedness in the real world
  - Flights, campus, (social) networks, etc.
  - Useful to finding shortest number of steps/hops

Internet (~1972)

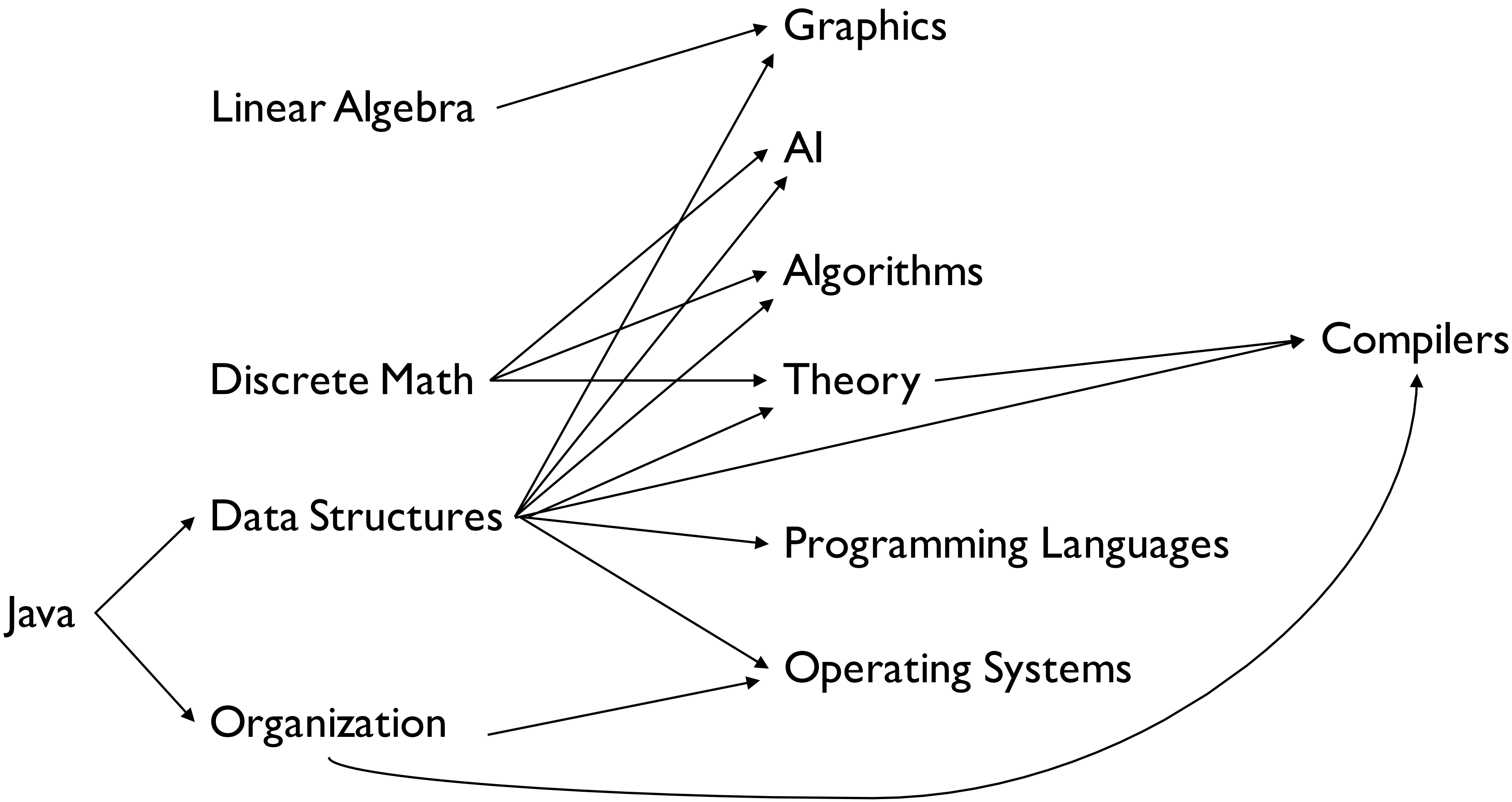


Internet (~1998)



Graph Applications

- Connectedness in the real world
  - Flights, campus, (social) networks, etc.
  - Often useful to find shortest number of steps/hops
- CS Courses
  - In edges/out edges indicate prerequisite relationships (why no cycles?)



Question: Is this a tree?

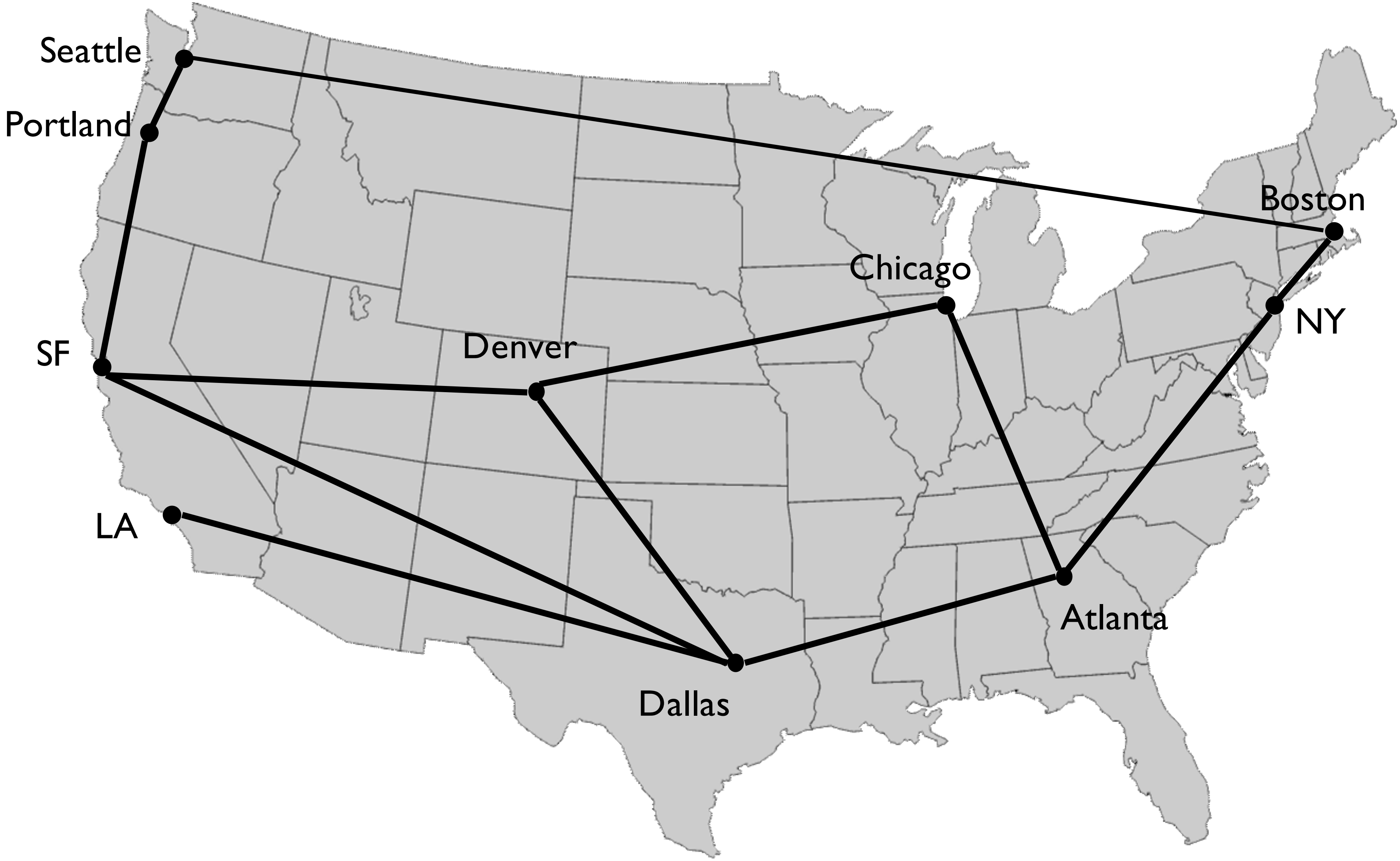
- No! No root node, Courses can have multiple parents

Vertices and Edges

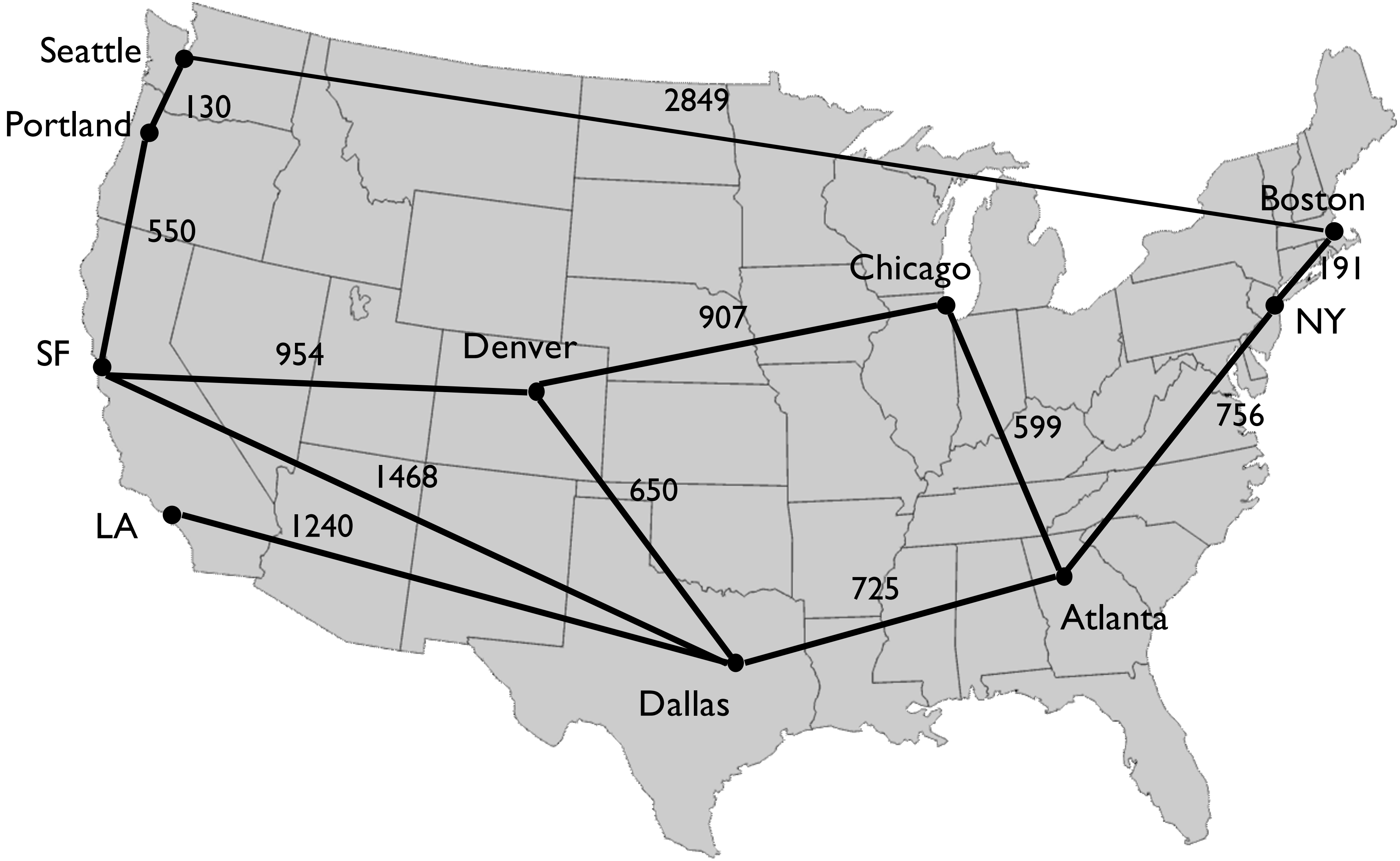
- Vertices represent "things"
- Edges encode relationships between "things"
  - Not all edges are the same

Edges

- Edges can have different "weight"
- Weight = the *cost* of traversing that edge
    - Cost may be a function of time, distance, price to pay, probability, etc.
- May lead to different solutions to previously answered questions
    - What is shortest path between SF and NY given edge weights?



- What is the shortest path from SF to Boston?



- What is the shortest path from SF to Boston?

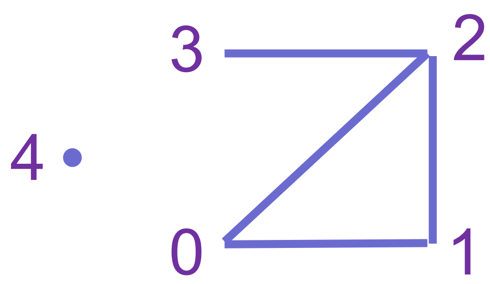
Representing Graphs

Let’s think back to the ways we represented trees:

- Nodes store explicit pointers to parent/children
- Nodes are stored in a Vector
  - Which was better for sparse trees?
  - Which was better for dense trees?

Adjacency Matrix

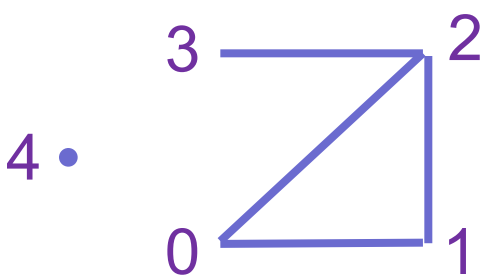
- Let $G = (V, E)$ be a graph with n vertices
  - Number the vertices $0 \ldots n\text{-}1$
  - The adjacency matrix of G is an n x n matrix where each (x,y) coordinate is T if there is an edge between v_x, v_y and F otherwise.
- Example:

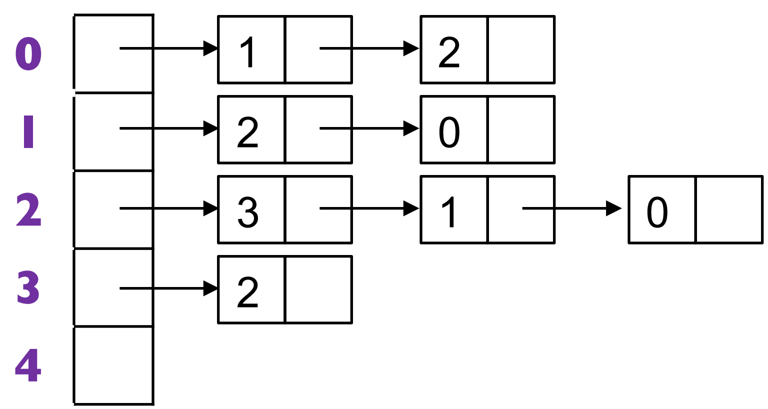


	0	**1**	**2**	**3**	**4**
0	F	T	T	F	F
1	T	F	T	F	F
2	T	T	F	T	F
3	F	F	T	F	F
4	F	F	F	F	F

Adjacency List

- Let $G = (V, E)$ be a graph with n vertices
  - Number the vertices $0 \ldots n\text{-}1$
  - The adjacency list of G is a Vector of length n where each entry in the Vector contains a list of all adjacent vertices.
- Example:





Representation Tradeoffs

- Let $G = (V, E)$ be a graph with n vertices
- What is the maximum number of edges in G?
  - $m <= n^2$ (every node connected to every other node)
- G is dense if m is close to n^2
- G is sparse if m is far from n^2

Representation Tradeoffs

- $G = (V, E)$ with $|E| = m$, $|V| = n$

	Adjacency Matrix	Adjacency List
Space	$O(n^2)$	$O(n+m)$
Time to check if v_1 connected to v_2	$O(1)$	$O(\text{out-degree}(v_1))$
Time to find all v_i adjacent to v_1	$O(n)$	$O(\text{out-degree}(v_1))$
Time to visit all edges	$O(n^2)$	$O(n+m)$

No Clear Efficiency Winner

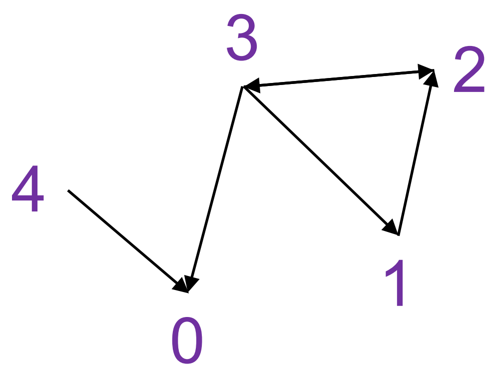
- Matrix is better for dense graphs
- List is better for sparse graphs
- Graphs “in the middle”?

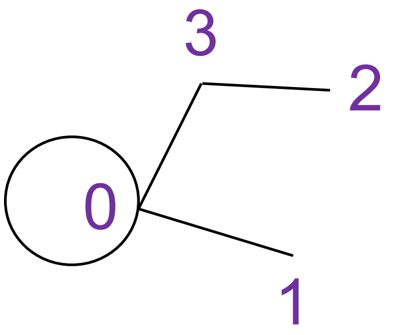
Other Considerations

- What API should graphs support?
  - Want to lookup vertices by label
  - Want extra information to manage traversals
    - "Visited" info for nodes and edges
- What does it mean for two vertices to be equal? Two edges?
- Next class we will talk about implementation details and traversal strategies…

Practice

- Draw the adjacency matrix and adjacency list representations of the following graphs:





- What does it mean for an adjacency matrix to be symmetric?