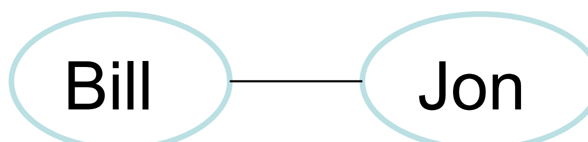
CSCI 136
Data Structures &
Advanced Programming

Lecture 29

Spring 2018

Instructors:


Last Time

- BSTs
  - Balance is important to maintain height (logn)
    - AVL Trees
      - Rotate left, rotate right
    - One of many types of balanced trees
- Game Trees
  - Backwards induction

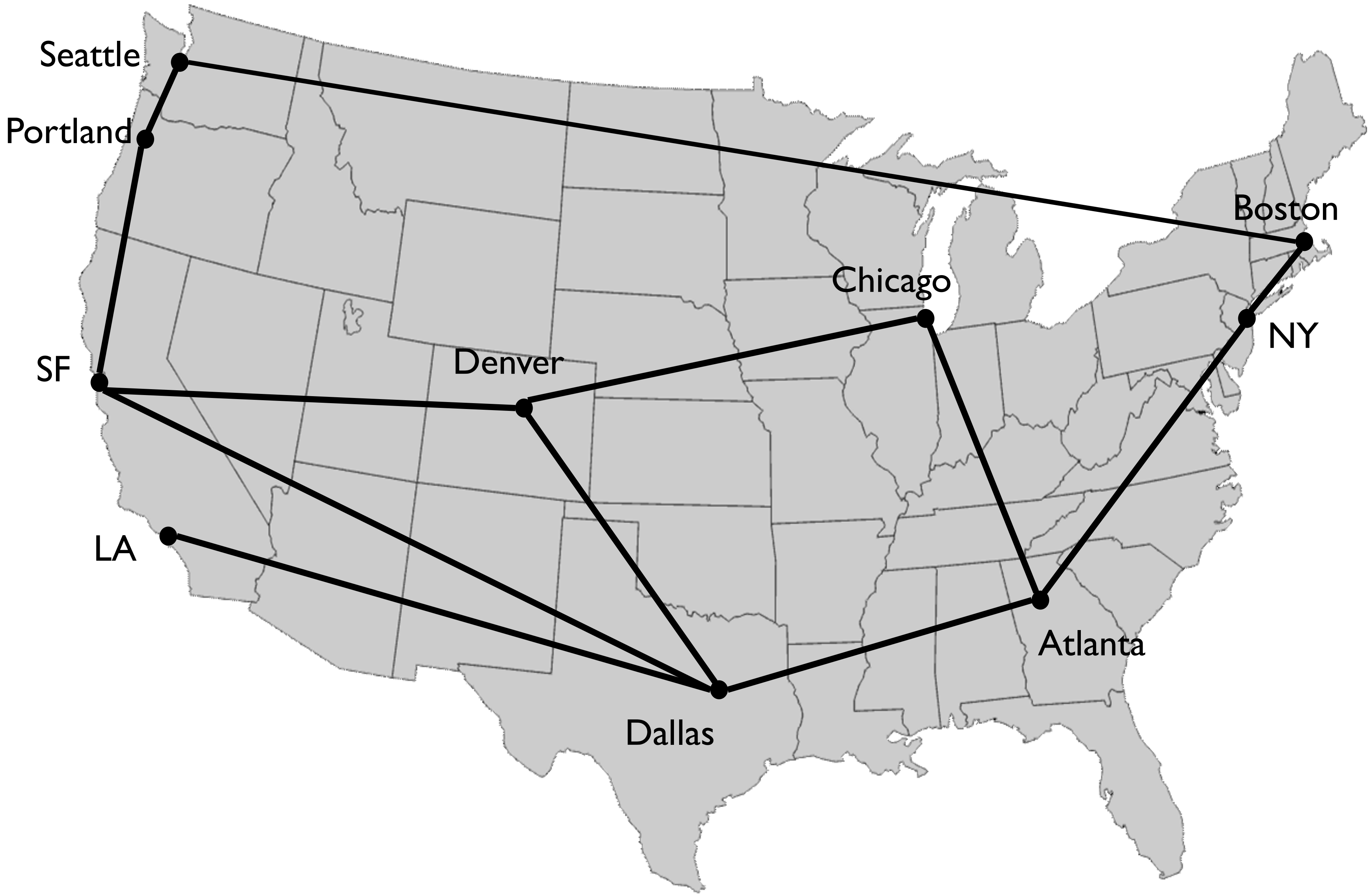
Today's Outline

- Introduction To Graphs
  - Definitions and Properties: Undirected Graphs
  - Small Proofs
  - Rechability
  - Graph Interface in Structure5

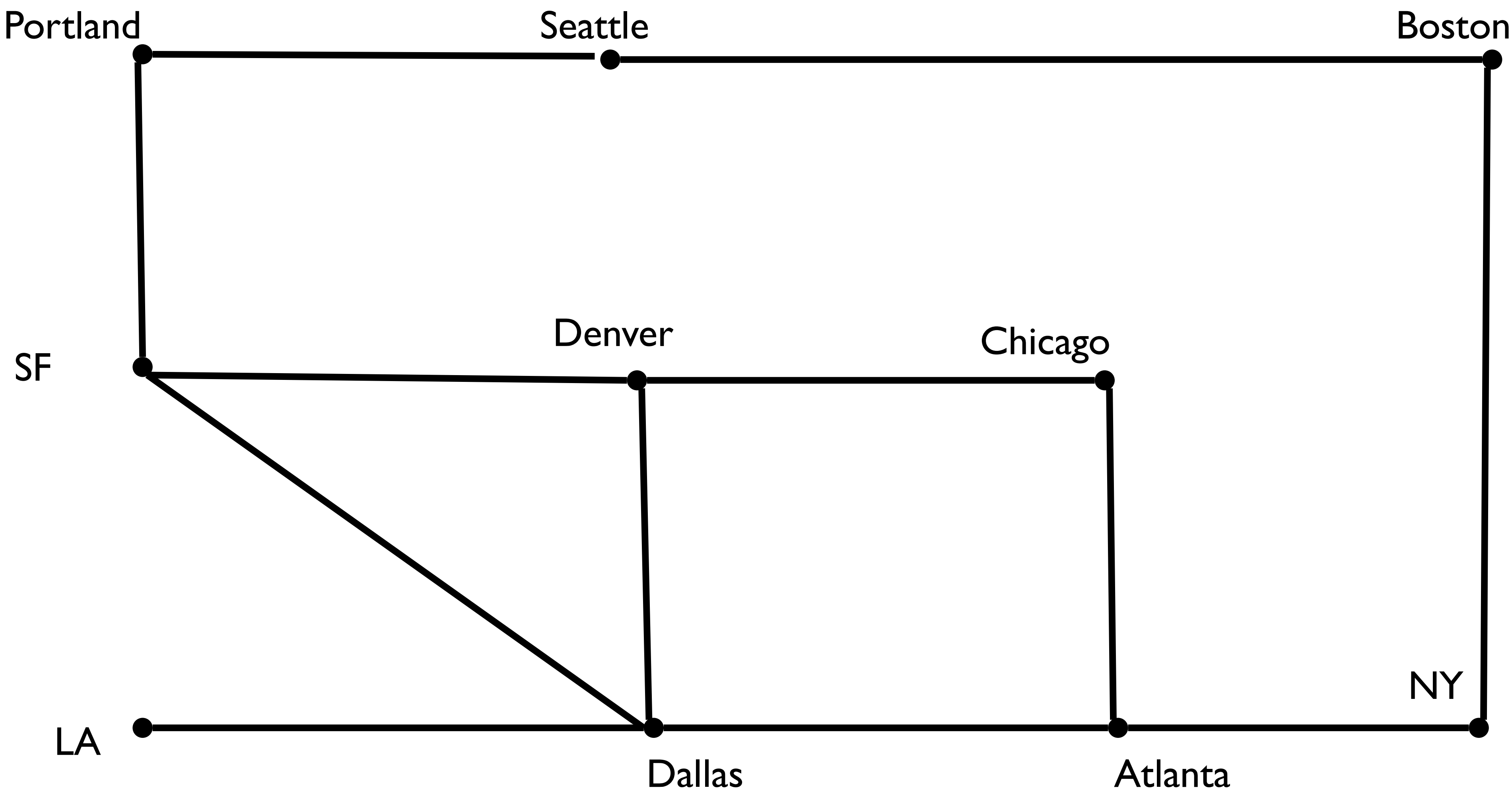
Graphs Describe the World[1]

- Transportation Networks
- Communication Networks
- Social Networks
- Molecular structures
- Dependency structures
- Scheduling
- Matching
- Graphics Modeling
- ....

Nodes = subway stops; Edges = subway lines

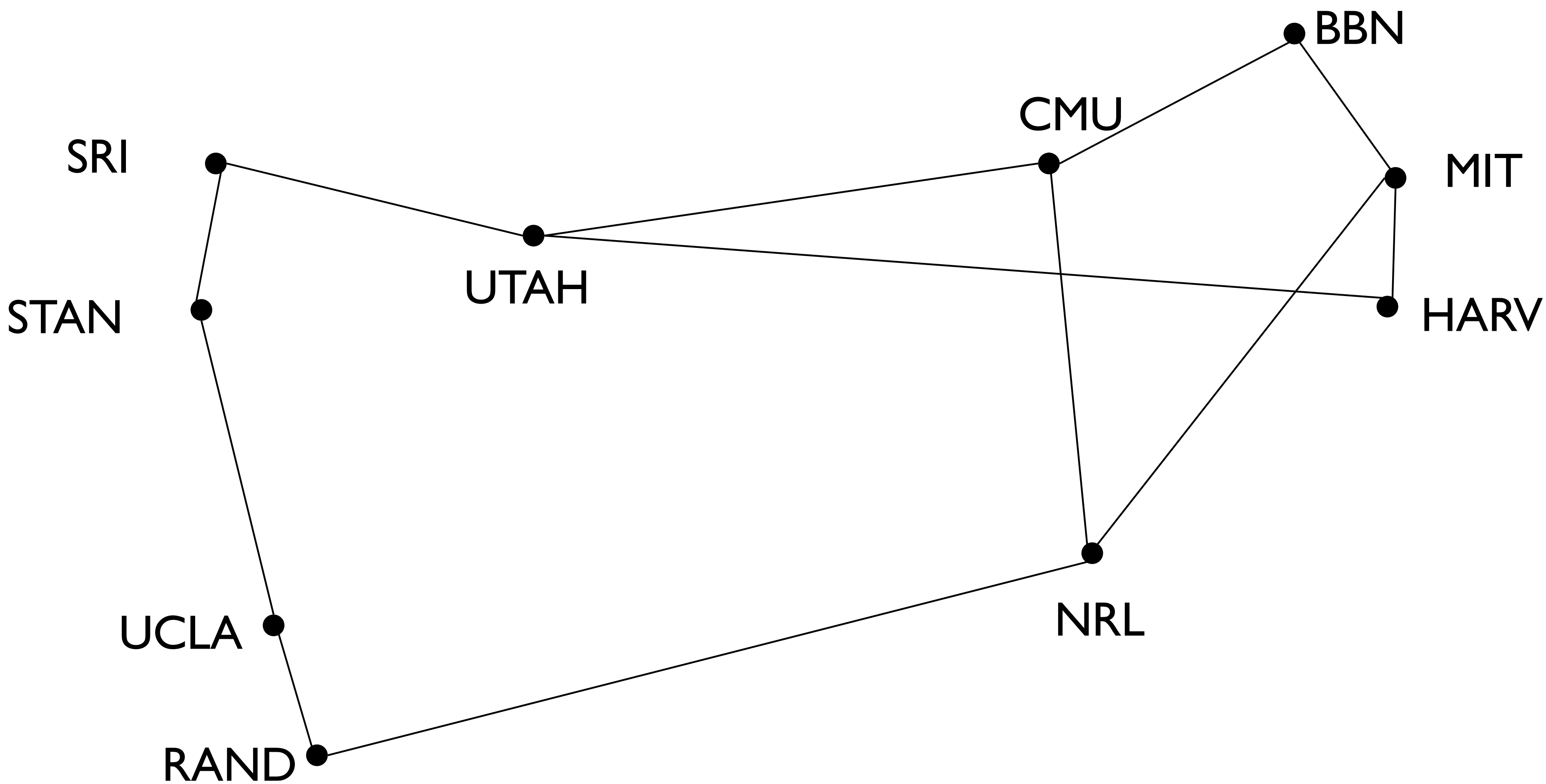


Nodes = cities; Edges = rail lines connecting cities

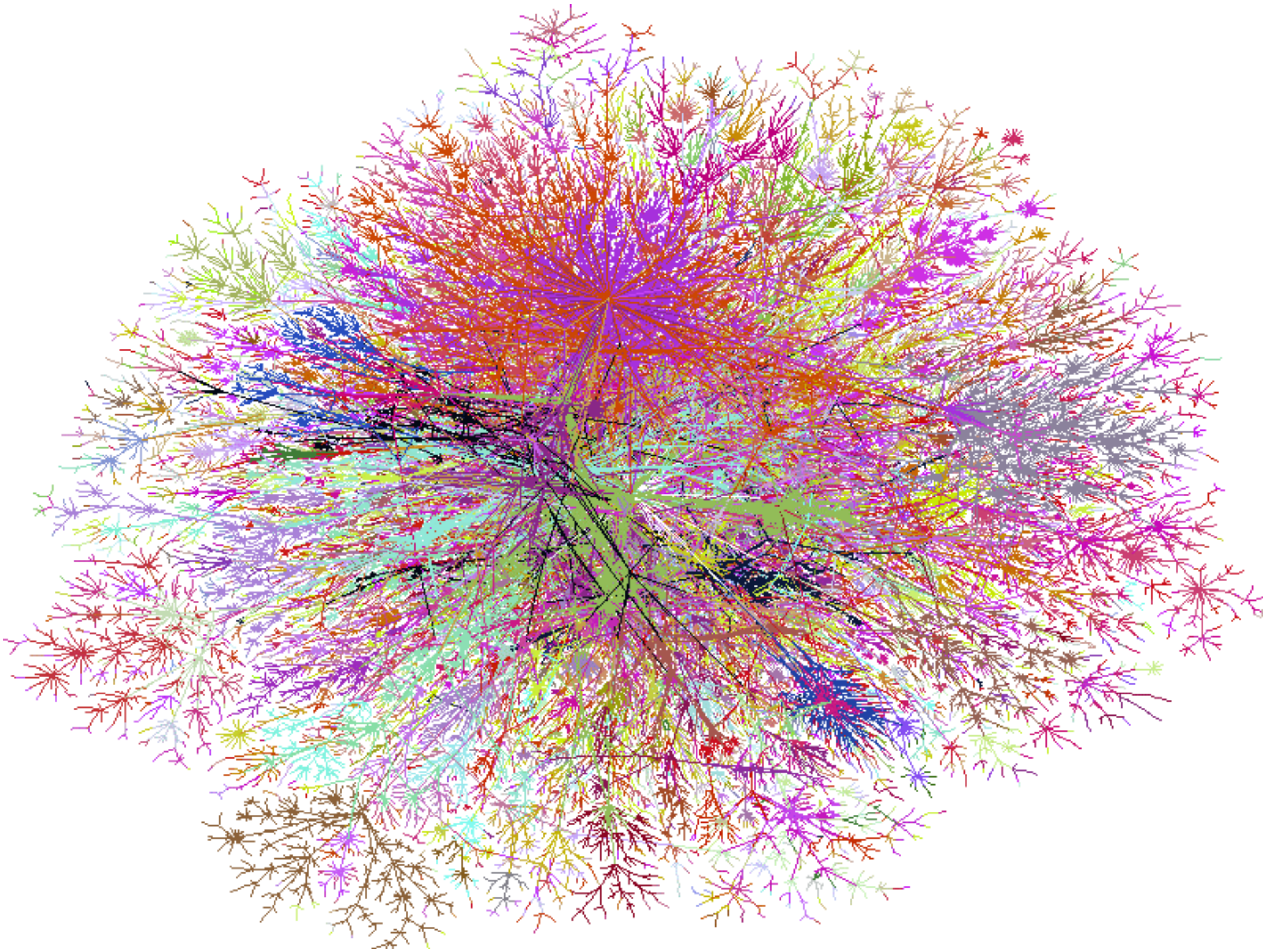


Note: Connections in graph matter, not precise locations of nodes

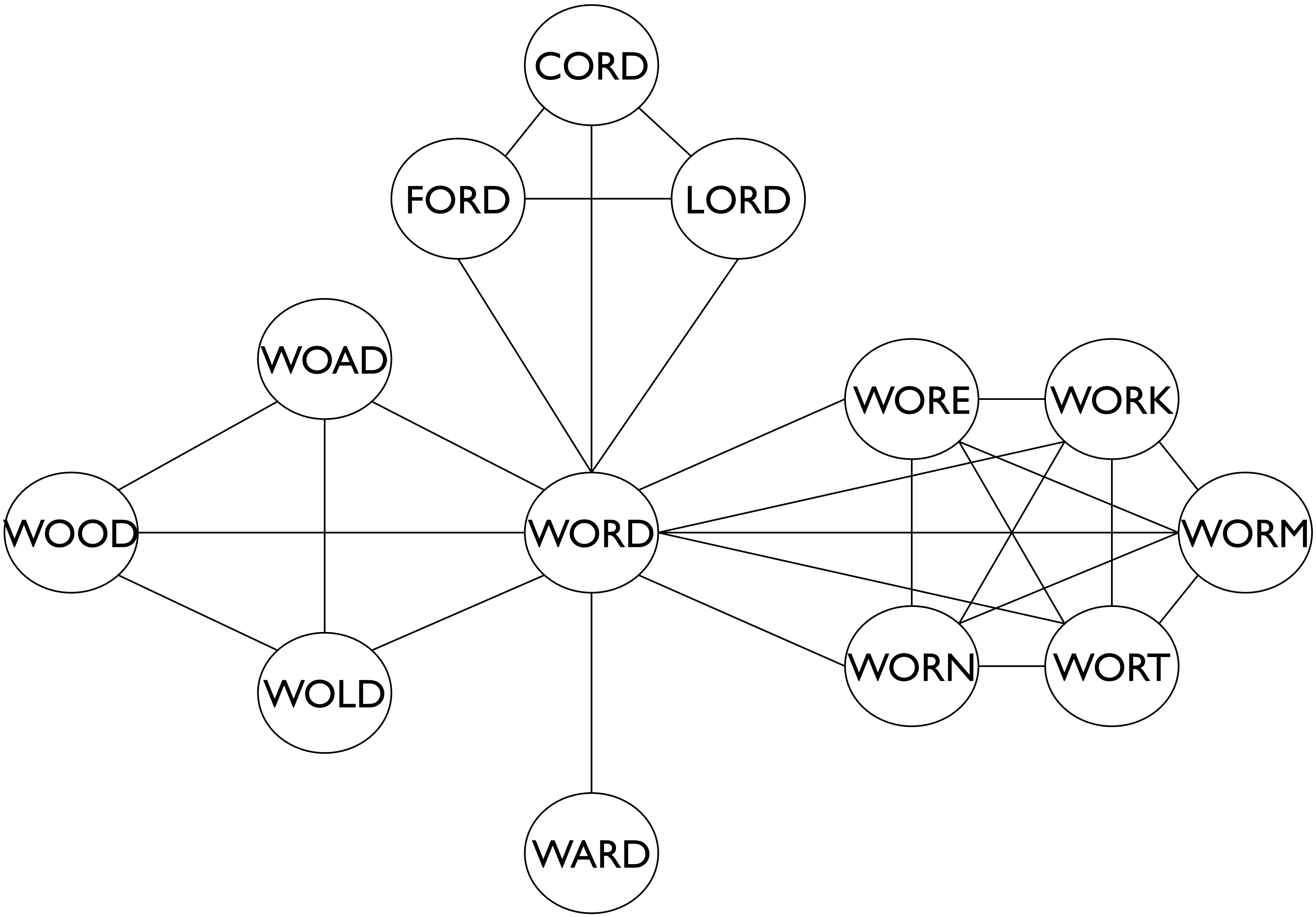
Internet (~1972)



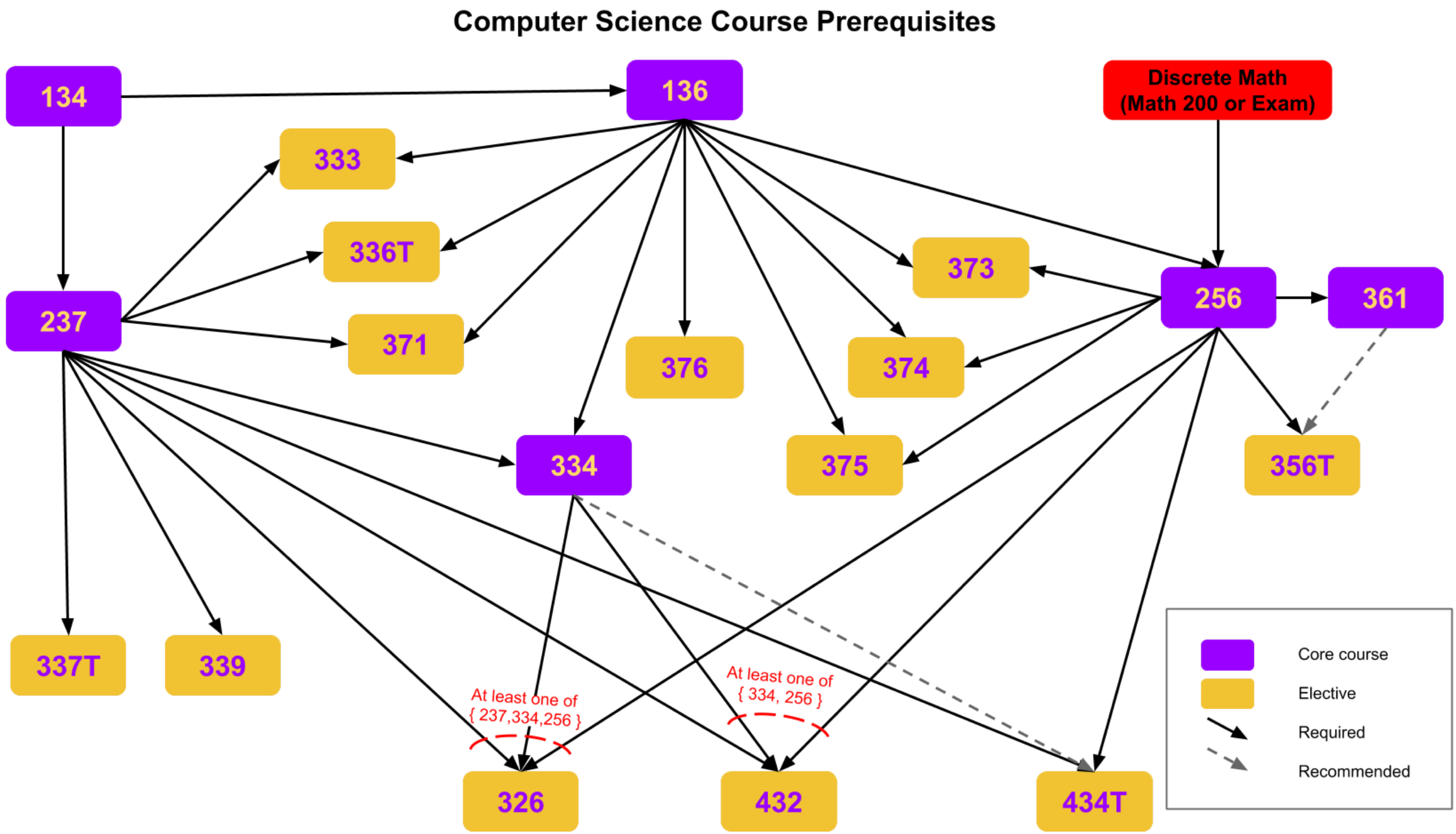
Internet (~1998)



Word Game

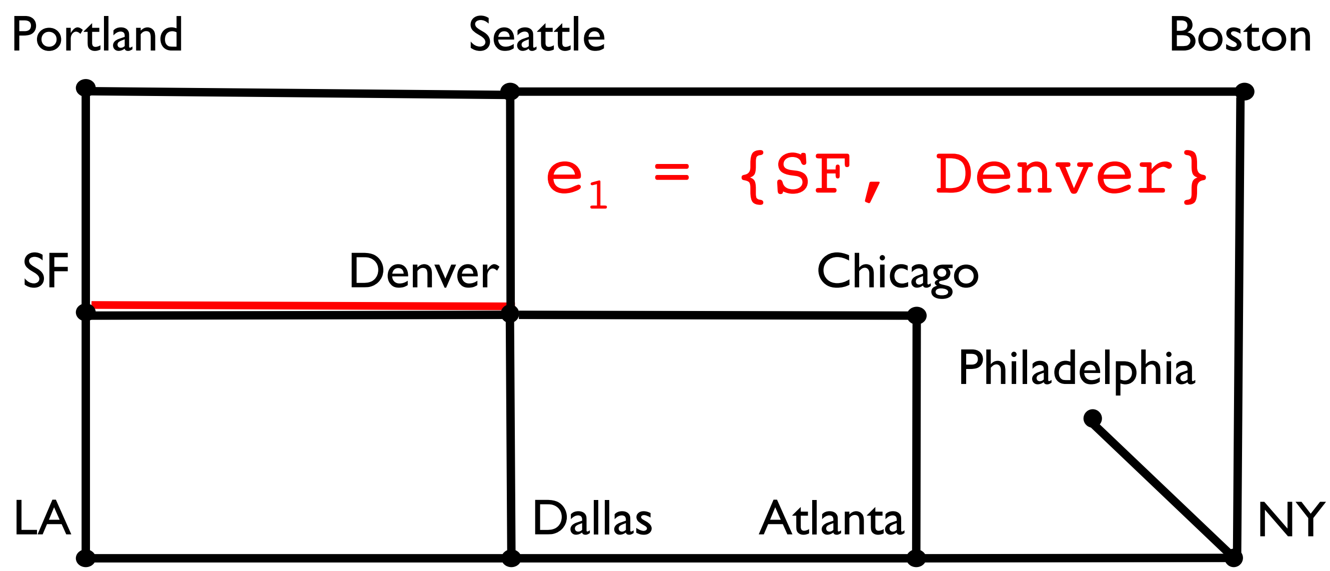


Nodes = words; Edges = words that differ by exactly one letter



Nodes = courses; Edges = prerequisites ***

Basic Definitions & Concepts



Definition: An *undirected graph* `G = (V,E)` consists of two sets

- `V` : the *vertices* of `G`, and `E` : the *edges* of `G`
- Each edge `e` in `E` is defined by a set of two vertices: its *incident vertices*.
- We write `e = {u,v}` and say that `u` and `v` are *adjacent.*

Basic Definitions & Concepts

- **Definition:** An *undirected graph* `G = (V,E)` consists of two sets:
  - `V` : the *vertices* of `G`
  - `E` : the *edges* of `G`

- Each edge `e` in `E` is defined by a set of two vertices: its *incident vertices*
- We write `e={u,v}` and say that `u` and `v` are *adjacent*
- The *degree* of a vertex is the number of *incident edges* (loops counted twice)

Walking Along a Graph

- A *walk from u to v* in a graph $G = (V,E)$ is an *alternating* sequence of vertices and edges

  $$u = v_0, e_1, v_1, e_2, v_2, \ldots , v_{k-1}, e_k, v_k = v$$

  such that each $e_i = \{v_i , v_{i+1}\}$ for $i = 1, \ldots , k$

  - (Note a walk starts and ends on a vertex)
- If no *edge* appears more than once then the walk is called a *path*
- If no *vertex* appears more than once then the walk is a *simple path*

Walking In Circles

- A *closed walk* in a graph $G = (V,E)$ is a *walk*

  $$v_0, e_1, v_1, e_2, v_2, \ldots , v_{k-1}, e_k, v_k$$

  such that $v_0 = v_k$ (it ends at the starting v)
- A *circuit* is a *path* where $v_0 = v_k$
  - Circuit vs. closed walk? Circuit has no repeat edges
- A *cycle* is a *simple* path where $v_0 = v_k$
  - Circuit vs. cycle? Cycle has no repeated vertices.
- The length of any of these is the number of *edges* in the sequence

Little Tiny Theorems

- If there is a *walk* from u to v, then there is a *walk* from v to u.
- If there is a *walk* from u to v, then there is a *path* from u to v (and from v to u)
- If there is a *path* from u to v, then there is a *simple path* from u to v (and v to u)
- Every *circuit* through v contains a *cycle* through v
- Not every *closed walk* through v contains a *cycle* through v! [Try to find an example!]

A Basic Graph Fact

- Denote the degree of a vertex *v* by *deg(v)*.
- **Theorem**: For any graph `G = (V,E)`

$$\sum_{v \in V} \deg(v) = 2|E|$$

where |`E`| is the number of edges in `G`

- Proof Hint: Induction on |`E`|: How does removing an edge change the equation?
  - Instead: Count pairs *(v,e)* where *v* is incident with e

Reachability and Connectedness

- **Definition:** A vertex `v` in G is *reachable* from a vertex `u` in G if there is a path from `u` to `v`
  - `v` is reachable from `u` *iff* `u` is reachable from `v`
- **Definition:** An undirected graph G is *connected* if for every pair of vertices (`u`, `v`) in G, `v` is reachable from `u` (and vice versa)
- The set of all vertices reachable from `v`, along with all edges of G connecting any two of them, is called the *connected component* of `v`

Basic Graph Algorithms

- We'll look at a number of graph algorithms
  - Connectedness: Is G connected?
    - If not, how many *connected components* does G have?
  - Cycle testing: Does G contain a cycle?
    - Does G contain a cycle through a given vertex?
  - If the edges of G have costs:
    - What is the cheapest subgraph connecting all vertices
      - Called a *connected, spanning subgraph*
    - What is a cheapest path from u to v?
  - And more.... (if not here, then in CSCI 256!)

Testing Connectedness

- How can we determine whether G is connected?
  - Pick a vertex v; see if every vertex u is reachable from v
- How could we do this?
  - Visit the neighbors of v, then visit their neighbors, etc. See if you reach all vertices
    - (Assume we can mark a vertex as "visited")
- How do we manage all of this visiting?

Reachability: Breadth-First Search

BFS(G, v)　　　// Do a breadth-first search of G starting at v

// pre: all vertices are marked as unvisited

count ← *0;*

Create empty queue Q; enqueue v; mark v as visited; count++

While Q isn't empty

　　current ← *Q.dequeue();*

　　for each unvisited neighbor u of current :

　　　　add u to Q; mark u as visited; count++

return count;

Now compare value returned from BFS(G,v) to size of V

BFS Reflections

- The BFS algorithm traced out a tree T_v: the edges connecting a visited vertex to (as yet) unvisited neighbors
- T_v is called a *BFS tree of G with root v* (or *from v*)
- The vertices of T_v are visited in *level-order*
- This reveals a natural measure of distance between vertices: the length of (any) shortest path between the vertices