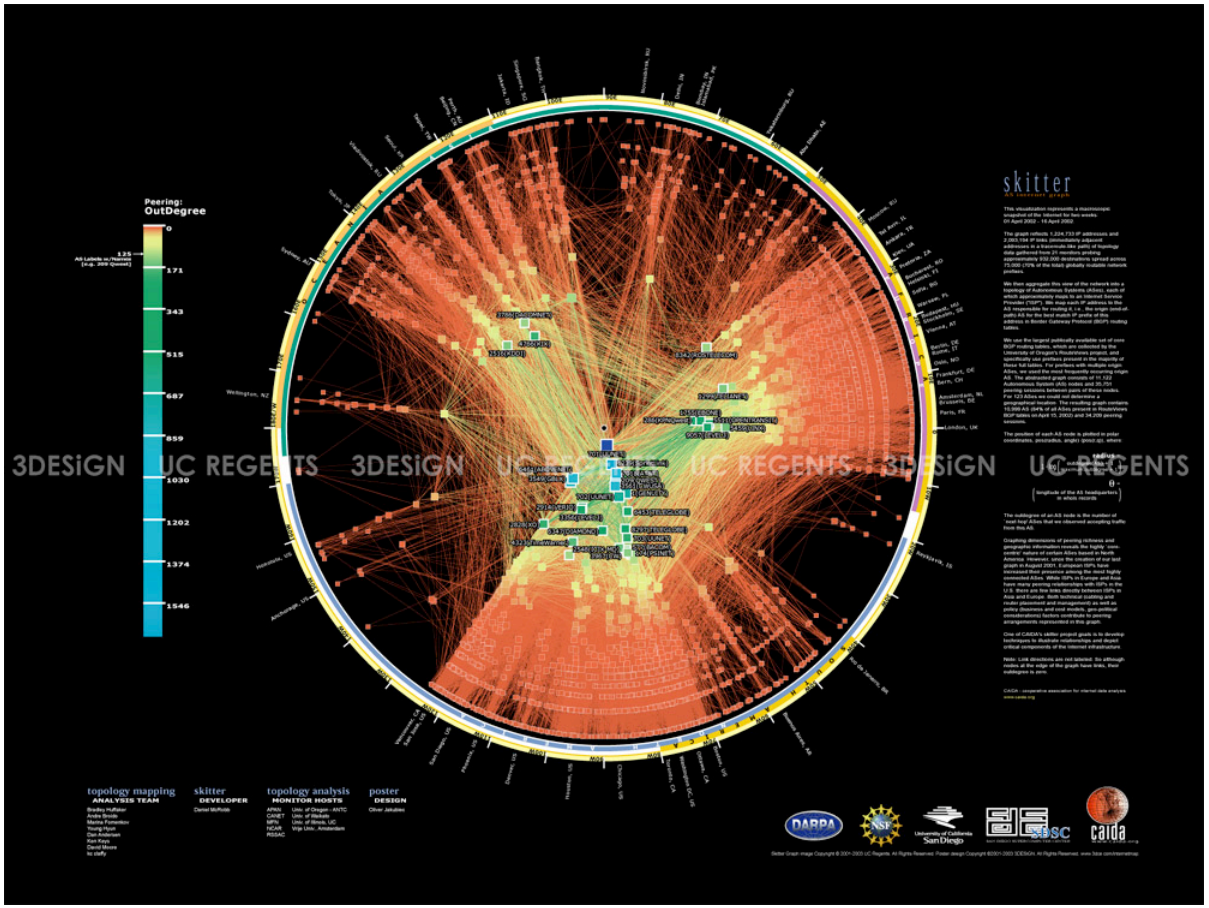
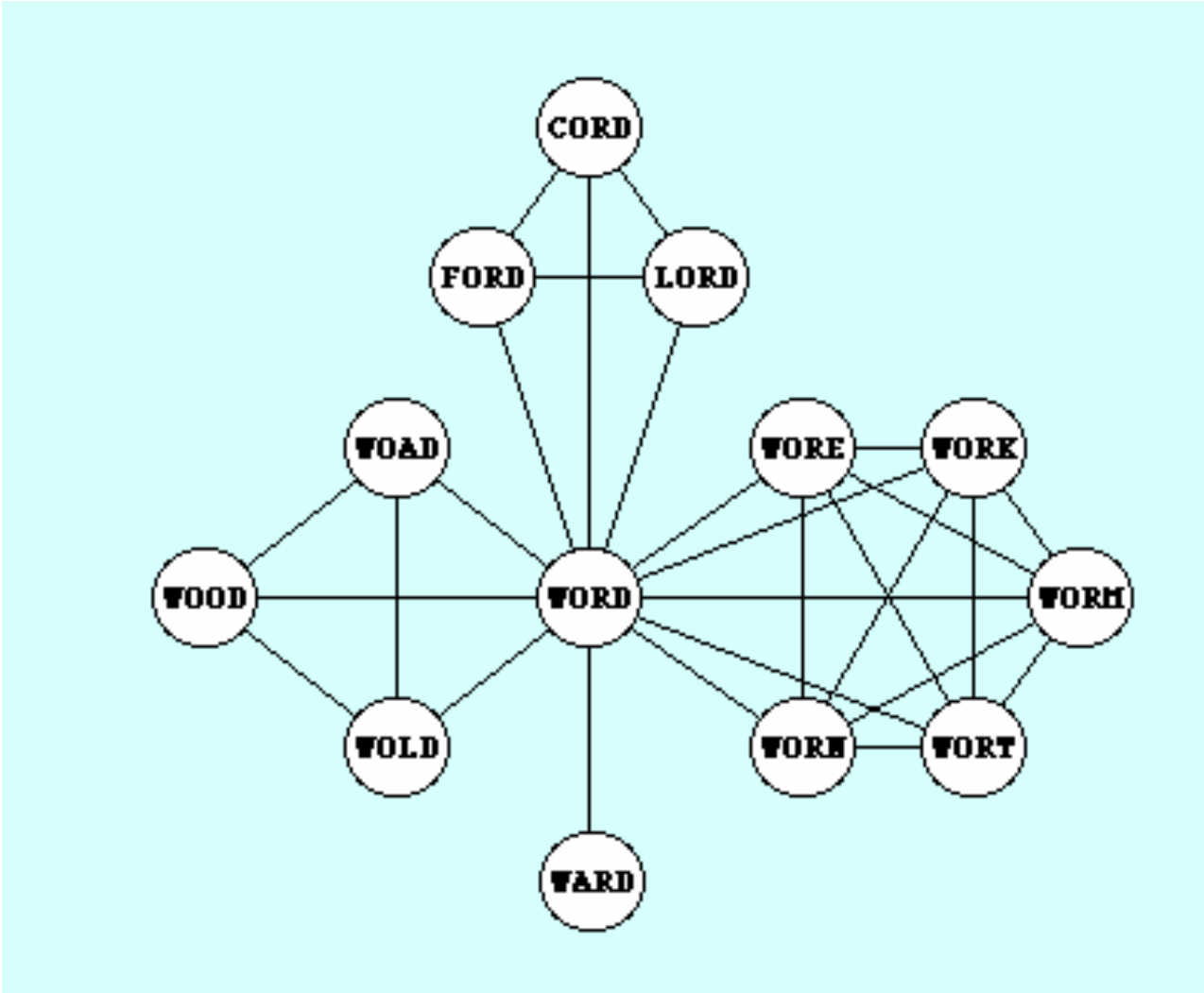
Graphs, the Internet, and Everything



http://www.caida.org/

Airline routes

Word ladder


CORD
FORD
LORD
WOAD
WORE
WORK
WOOD
WORD
WORM
WOLD
WORN
WORT
WARD

Tim Berners-Lee

I want you to realize that, if you can imagine a computer doing something, you can program a computer to do that.

Unbounded opportunity... limited only by your imagination. And a couple of laws of physics.

- TCP/IP, HTTP
  - How, Why, What, When?

Graphs: Structures and Algorithms

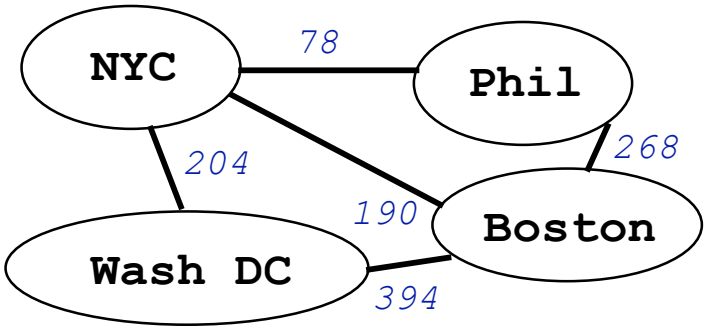
- How do packets of bits/information get routed on the internet
  - Message divided into packets on client (your) machine
  - Packets sent out using routing tables toward destination
    - Packets may take different routes to destination
    - What happens if packets lost or arrive out-of-order?
  - Routing tables store local information, not global (why?)

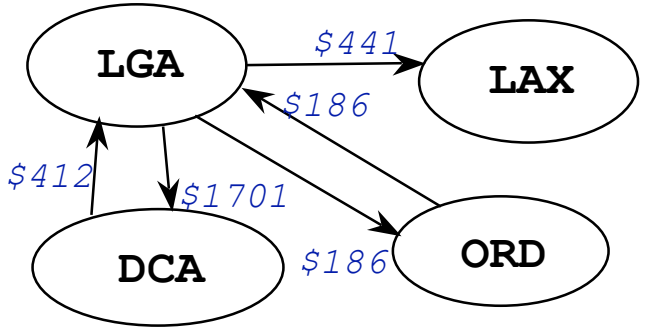
- What about The Oracle of Bacon, Erdos Numbers, and Word Ladders?
  - All can be modeled using graphs
  - What kind of connectivity does each concept model?

- Graphs are everywhere in the world of algorithms (world?)

Vocabulary

- Graphs are collections of *vertices* and *edges* (vertex also called node)
  - Edge connects two *vertices*
    - Direction can be important, *directed edge, directed graph*
    - Edge may have associated weight/cost
- A vertex sequence $v_0, v_1, \ldots, v_{n-1}$ is a *path* where v_k and v_{k+1} are connected by an edge.
  - If some vertex is repeated, the path is a *cycle*
  - A graph is *connected* if there is a path between any pair of vertices





Graph questions/algorithms

- What vertices are reachable from a given vertex?
  - Two standard traversals: depth-first, breadth-first
  - Find *connected components*, groups of connected vertices

- Shortest path between any two vertices (weighted graphs?)
  - Breadth first search is storage expensive
  - Dijkstra's algorithm is efficient, uses a priority queue too!

- Longest path in a graph
  - No known efficient algorithm

- Visit all vertices without repeating? Visit all edges?
  - With minimal cost? Hard!

Depth, Breadth, other traversals

- **We want to visit every vertex that can be reached from a specific starting vertex (we might try all starting vertices)**
  - **Make sure we don't visit a vertex more than once**
    - **Why isn't this an issue in trees?**
    - **Mark vertex as visited, use set/array/map for this**
      - Can keep useful information to help with visited status
  - **Order in which vertices visited can be important**
  - **Storage and runtime efficiency of traversals important**

- **What other data structures do we have: stack, queue, …**
  - **What happens when we traverse using priority queue?**

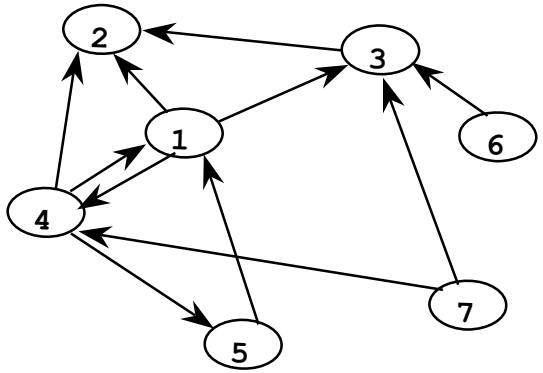
Breadth first search

- In an unweighted graph this finds the shortest path between a start vertex and every vertex
  - Visit every node one away from start
  - Visit every node two away from start
    - This is every node one away from a node one away
  - Visit every node three away from start, …

- Put vertex on queue to start (initially just one)
  - Repeat: take vertex off queue, put all adjacent vertices on
  - Don't put a vertex on that's already been visited (why?)
  - When are 1-away vertices enqueued? 2-away? 3-away?
  - How many vertices on queue?

Code for breadth first

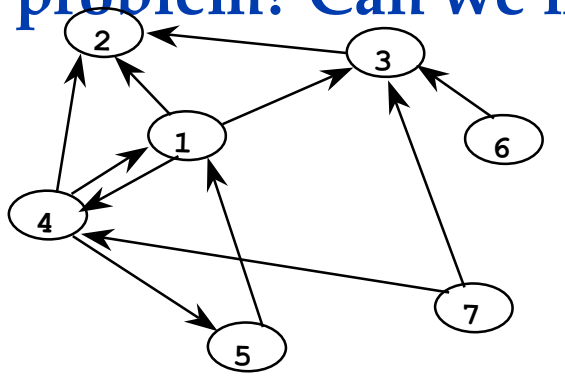
```
public void breadth(String vertex){
    Set<String> visited = new TreeSet<String>();
    Queue<String> q = new LinkedList<String>();
    q.add(vertex);
    visited.add(vertex);
    while (q.size() > 0) {
       String current = q.remove();
       // process current
       for(each v adjacent to current){
         if (!visited.contains(v)){// not visited
            visited.add(v);
            q.add(v);
         }
       }
    }
}
```



Pseudo-code for depth-first search

```
void depthfirst(String vertex){
    if (! alreadySeen(vertex))
    {
        markAsSeen(vertex);
        System.out.println(vertex);
        for(each v adjacent to vertex) {
           depthfirst(v);
        }
    }
}
```

- Clones are stacked up, problem? Can we make use of stack explicit?



BFS compared to DFS

```
public Set<Graph.Vertex> bfs(Graph.Vertex start){
    Set<Graph.Vertex> visited = new TreeSet<Graph.Vertex>();
    Queue<Graph.Vertex> qu = new LinkedList<Graph.Vertex>();
    visited.add(start);
    qu.add(start);

    while (qu.size() > 0){
        Graph.Vertex v = qu.remove();
        for(Graph.Vertex adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);
                qu.add(adj);
            }
        }
    }
    return visited;
}
```

BFS becomes DFS

```
public Set<Graph.Vertex> dfs(Graph.Vertex start){
    Set<Graph.Vertex> visited = new TreeSet<Graph.Vertex>();
    Queue<Graph.Vertex> qu = new LinkedList<Graph.Vertex>();
    visited.add(start);
    qu.add(start);

    while (qu.size() > 0){
        Graph.Vertex v = qu.remove();
        for(Graph.Vertex adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);
                qu.add(adj);
            }
        }
    }
    return visited;
}
```

DFS arrives

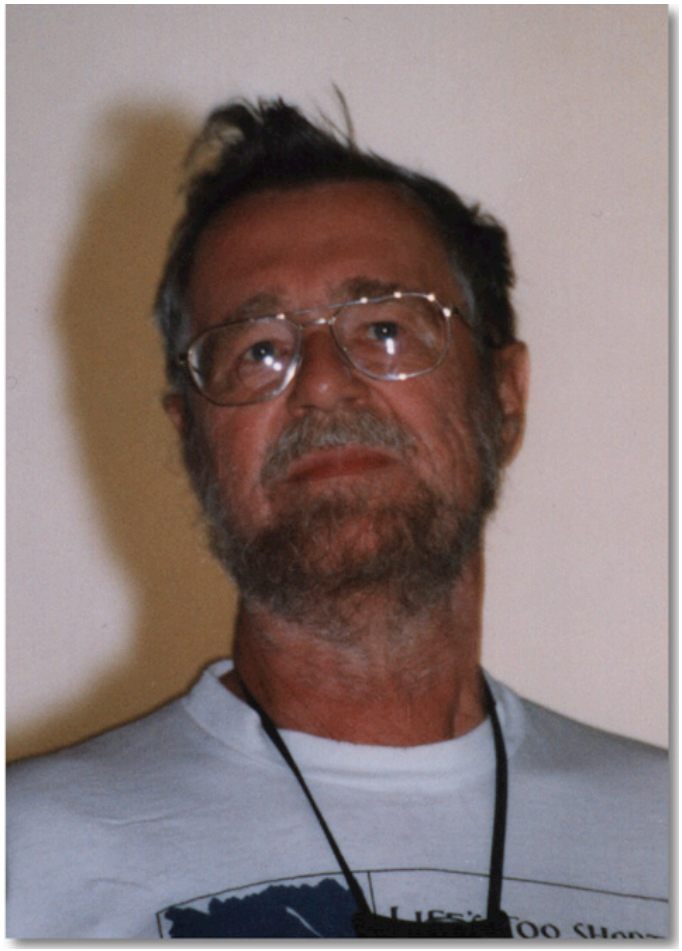
```
public Set<Graph.Vertex> dfs(Graph.Vertex start){
    Set<Graph.Vertex> visited = new TreeSet<Graph.Vertex>();
    Stack<Graph.Vertex> qu = new Stack<Graph.Vertex>();
    visited.add(start);
    qu.push(start);

    while (qu.size() > 0){
        Graph.Vertex v = qu.pop();
        for(Graph.Vertex adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);
                qu.push(adj);
            }
        }
    }
    return visited;
}
```

Edsger Dijkstra

- Turing Award, 1972
- Operating systems and concurrency
- Algol-60 programming language
- Goto considered harmful
- Shortest path algorithm
- Structured programming

  "Program testing can show the presence of bugs, but never their absence"
- A Discipline of programming

  "For the absence of a bibliography I offer neither explanation nor apology"



What is the Internet?

- The Internet was originally designed as an "overlay" network running on top of existing phone and other networks. It is based on a small set of software protocols that direct routers inside the network to forward data from source to destination, while applications run on the Internet to rapidly scale into a critical global service. However, this success now makes it difficult to create and test new ways of protecting it from abuses, or from implementing innovative applications and services.

http://www.intel.com/labs/features/idf09041.htm

How does the Internet work?

- Differences between the Internet and phone networks
  - Dedicated circuits/routes
  - Distributed, end-to-end

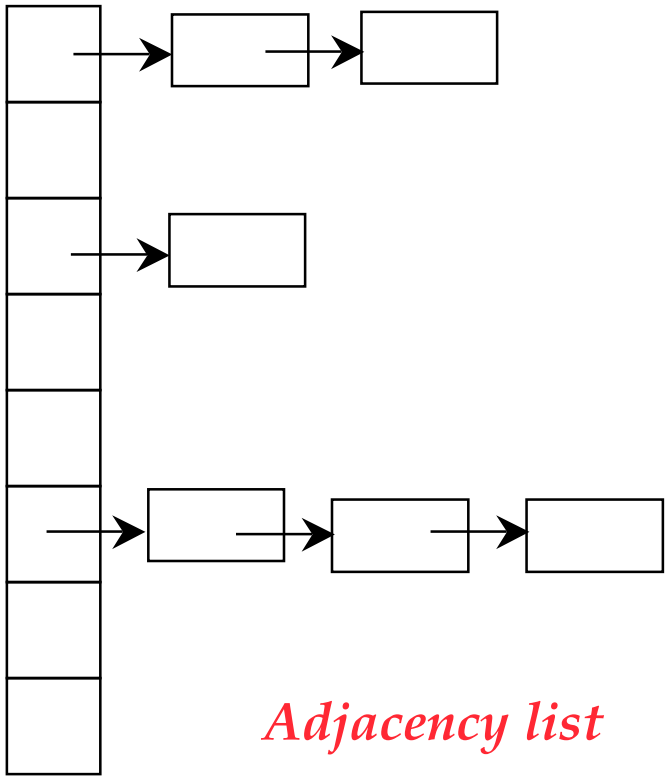
- Where is the intelligence?
  - Not in the network, per se, in the design and the ends
  - End-to-end Arguments in System Design

- Success of email, web, etc., relies on not building intelligence into the network
  - What about overlay networks?
  - What about PlanetLab?

Graph implementations

- Typical operations on graph:
  - Add vertex
  - Add edge (parameters?)
  - getAdjacent(vertex)
  - getVertices(..)
  - String->Vertex (vice versa)

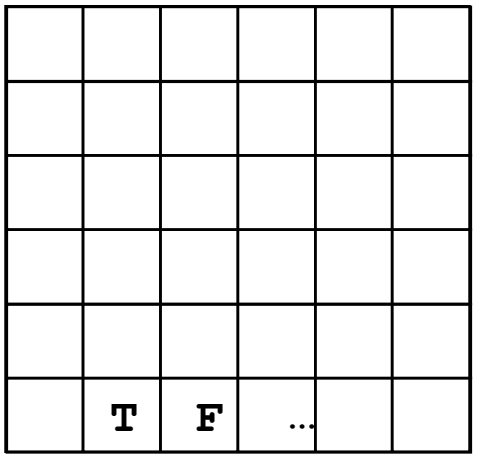
- Different kinds of graphs
  - Lots of vertices, few edges, *sparse* graph
    - Use adjacency list
  - Lots of edges (max # ?) *dense* graph
    - Use adjacency matrix



Adjacency list

Graph implementations (continued)

- Adjacency matrix
  - Every possible edge represented, how many?
- Adjacency list uses O(V+E) space
  - What about matrix?
  - Which is better?



- What do we do to get adjacent vertices for given vertex?
  - What is complexity?
  - Compared to adjacency list?

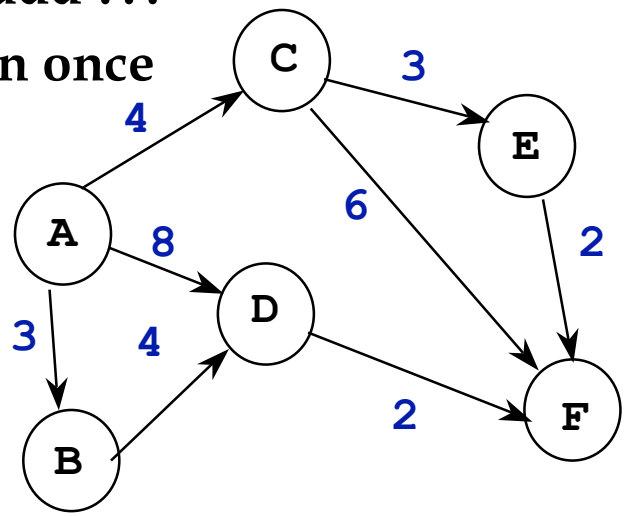
- What about weighted edges?

Shortest path in weighted graph

- We need to modify approach slightly for weighted graph
  - Edges have weights, breadth first by itself doesn't work
  - What's shortest path from A to F in graph below?

- Use same idea as breadth first search
  - Don't add 1 to current distance, add ???
  - Might adjust distances more than once
  - What vertex do we visit next?

- What vertex is next is key
  - Use greedy algorithm: closest
  - Huffman is greedy, ...



Greedy Algorithms

- A greedy algorithm makes a locally optimal decision that leads to a globally optimal solution
  - Huffman: choose two nodes with minimal weight, combine
    - Leads to optimal coding, optimal Huffman tree
  - Making change with American coins: choose largest coin possible as many times as possible
    - Change for $0.63, change for $0.32
    - What if we're out of nickels, change for $0.32?

- Greedy doesn't always work, but it does sometimes
- Weighted shortest path algorithm is *Dijkstra's* algorithm, greedy and uses priority queue