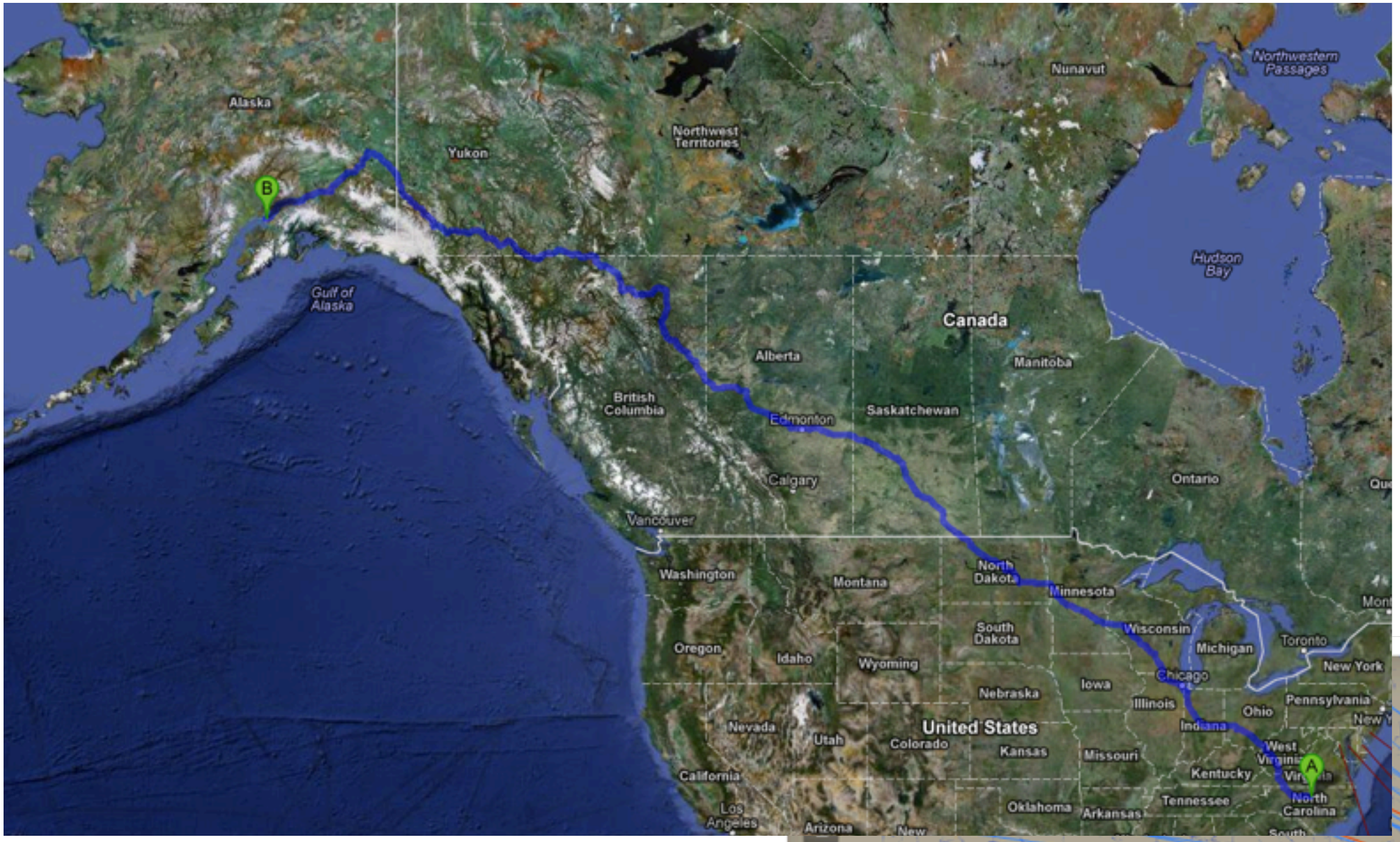
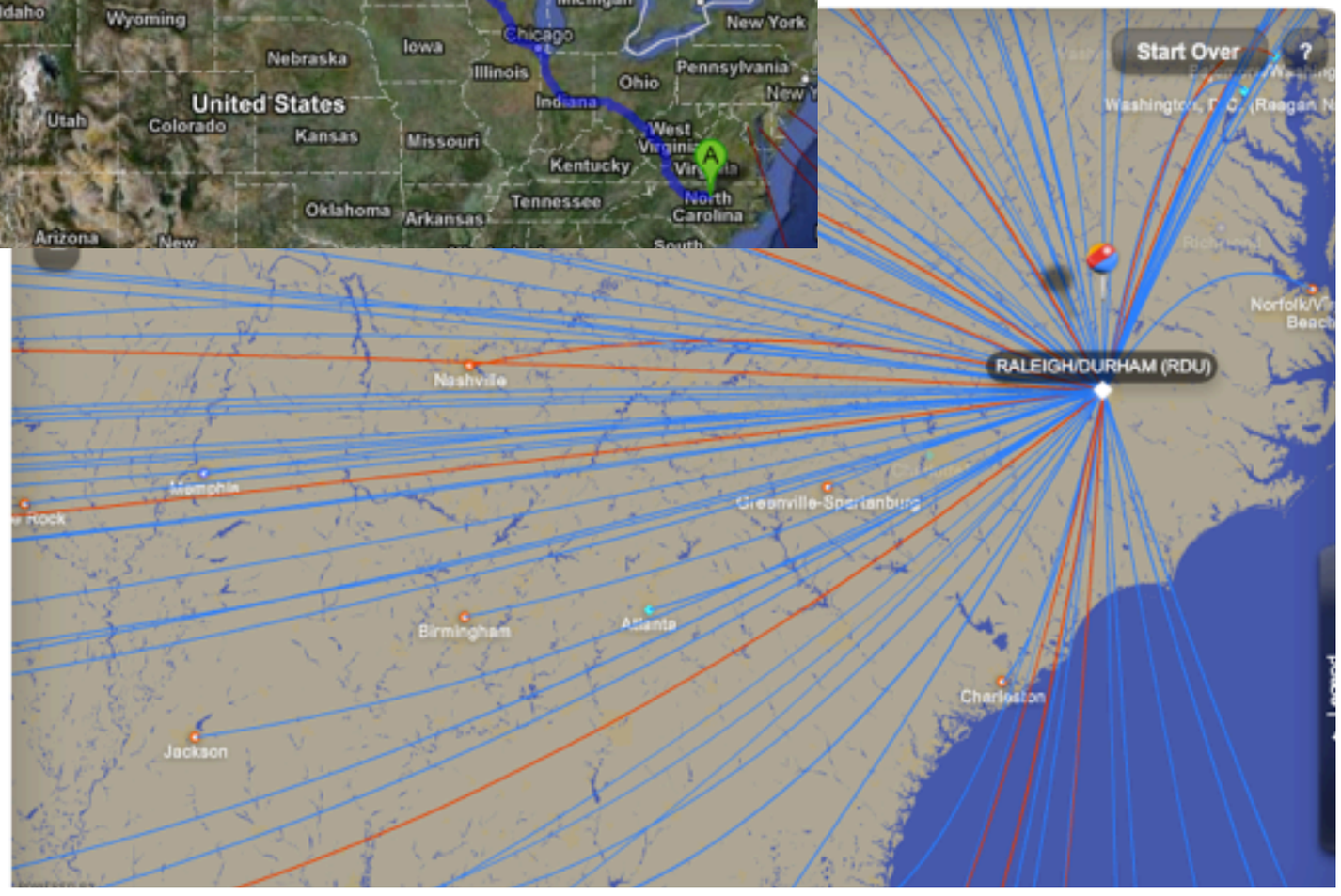
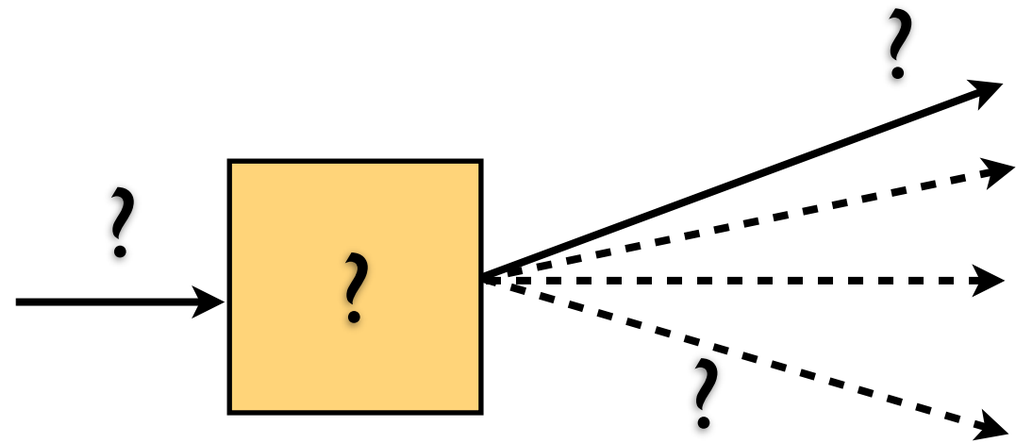

Alaska
Yukon
Northwest Territories
Nunavut
Northwestern Passages
Hudson Bay
Gulf of Alaska
Canada
Alberta
Manitoba
British Columbia
Edmonton
Saskatchewan
Calgary
Ontario
Vancouver
Washington
Montana
North Dakota
Minnesota
Wisconsin
Michigan
Toronto
New York
Oregon
Idaho
Wyoming
South Dakota
Nebraska
Iowa
Illinois
Ohio
Pennsylvania
Chicago
Indiana
Nevada
Utah
Colorado
United States
Kansas
Missouri
West Virginia
Kentucky
California
Los Angeles
Arizona
Oklahoma
Arkansas
Tennessee
North Carolina
B
A


Start Over
?
RALEIGH/DURHAM (RDU)
Nashville
Memphis
Greenville-Spartanburg
Birmingham
Atlanta
Charleston
Jackson
Norfolk/Virginia Beach

Graphs!

We've seen...

Linked Lists
Binary (Search) Trees
Heaps
Tries

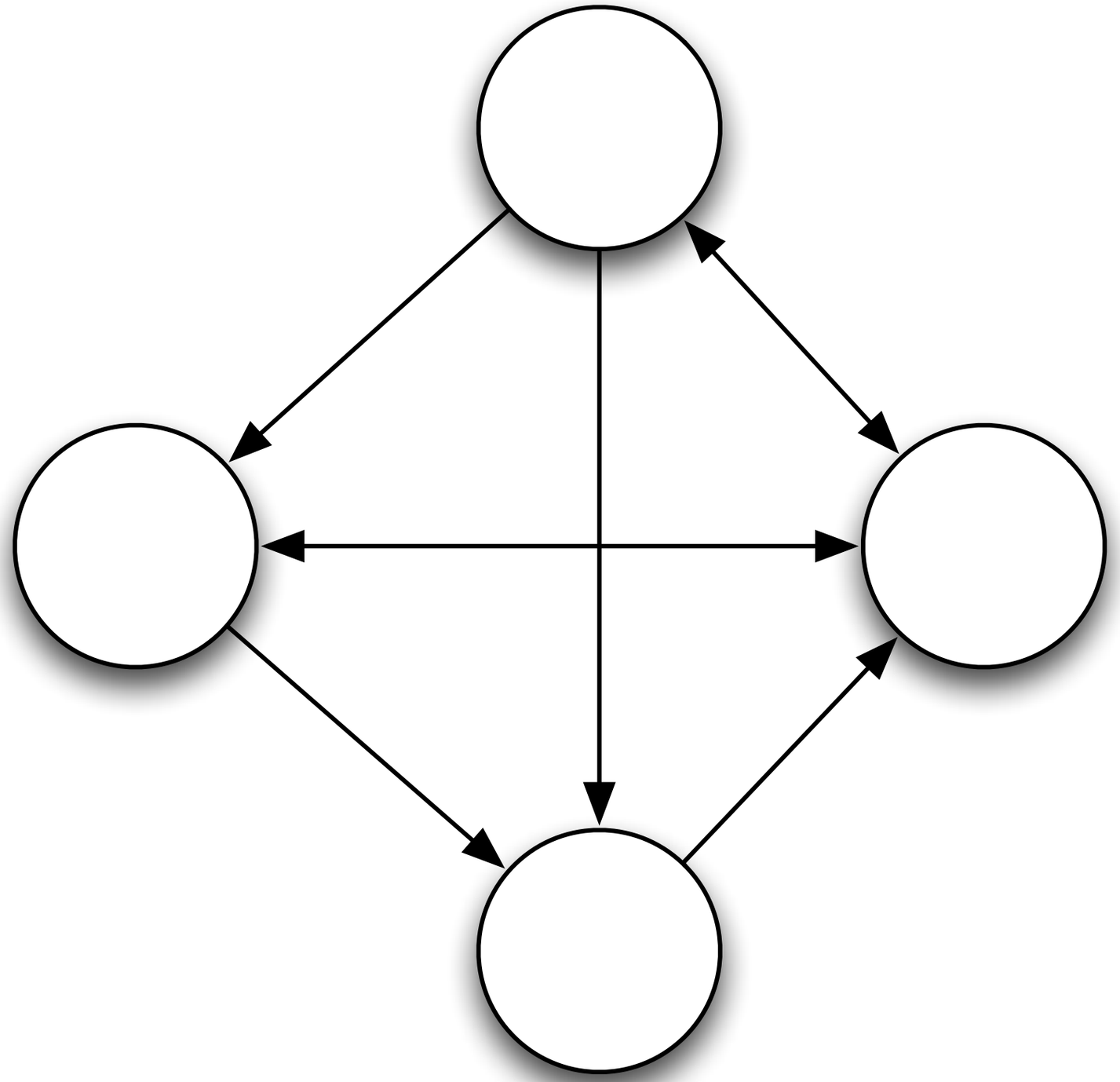


Graphs

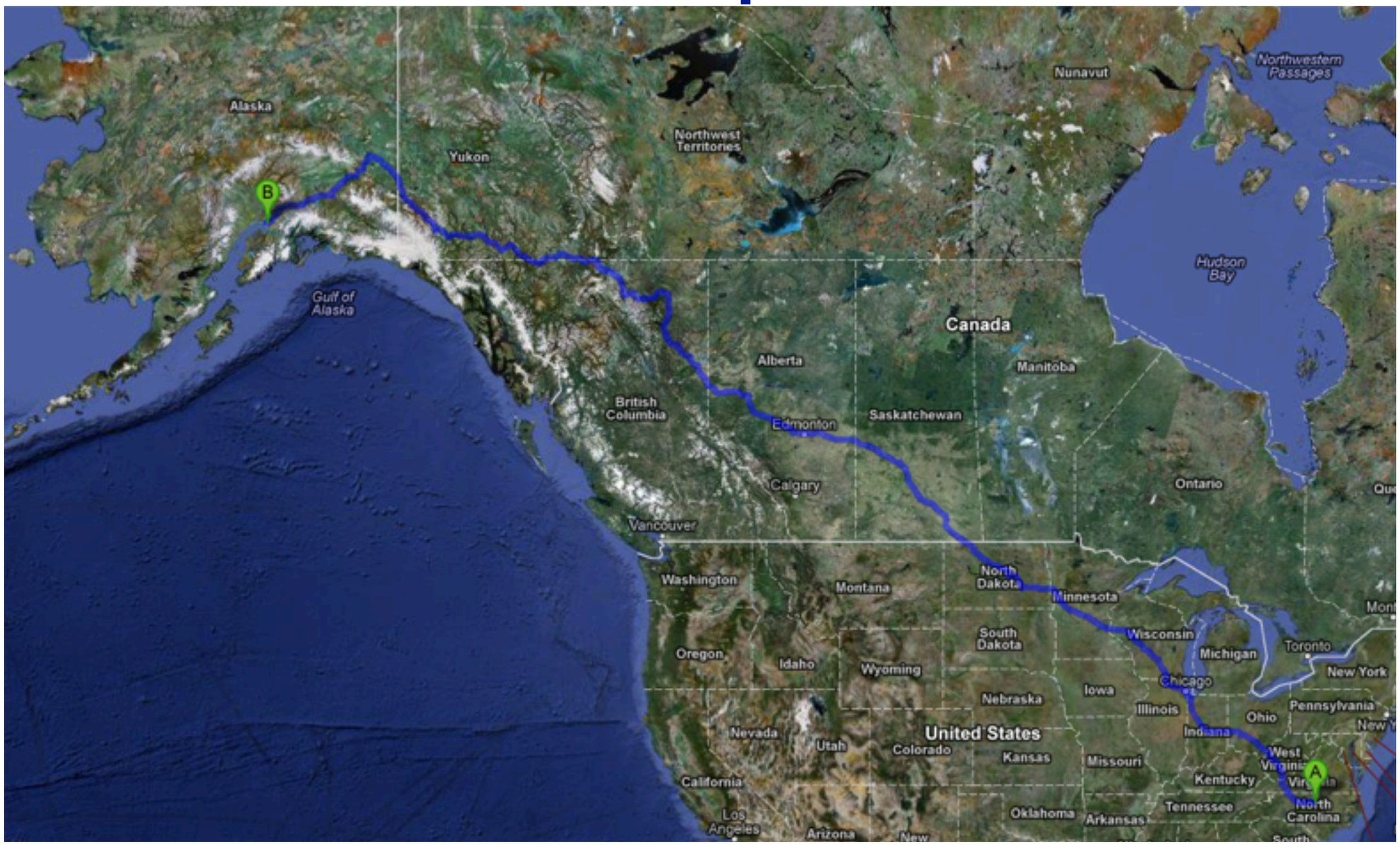
A set of *nodes* and a set of *edges*.

Edges can be *directed* or *undirected*.

Nodes and edges can have labels, or values, or whatever.

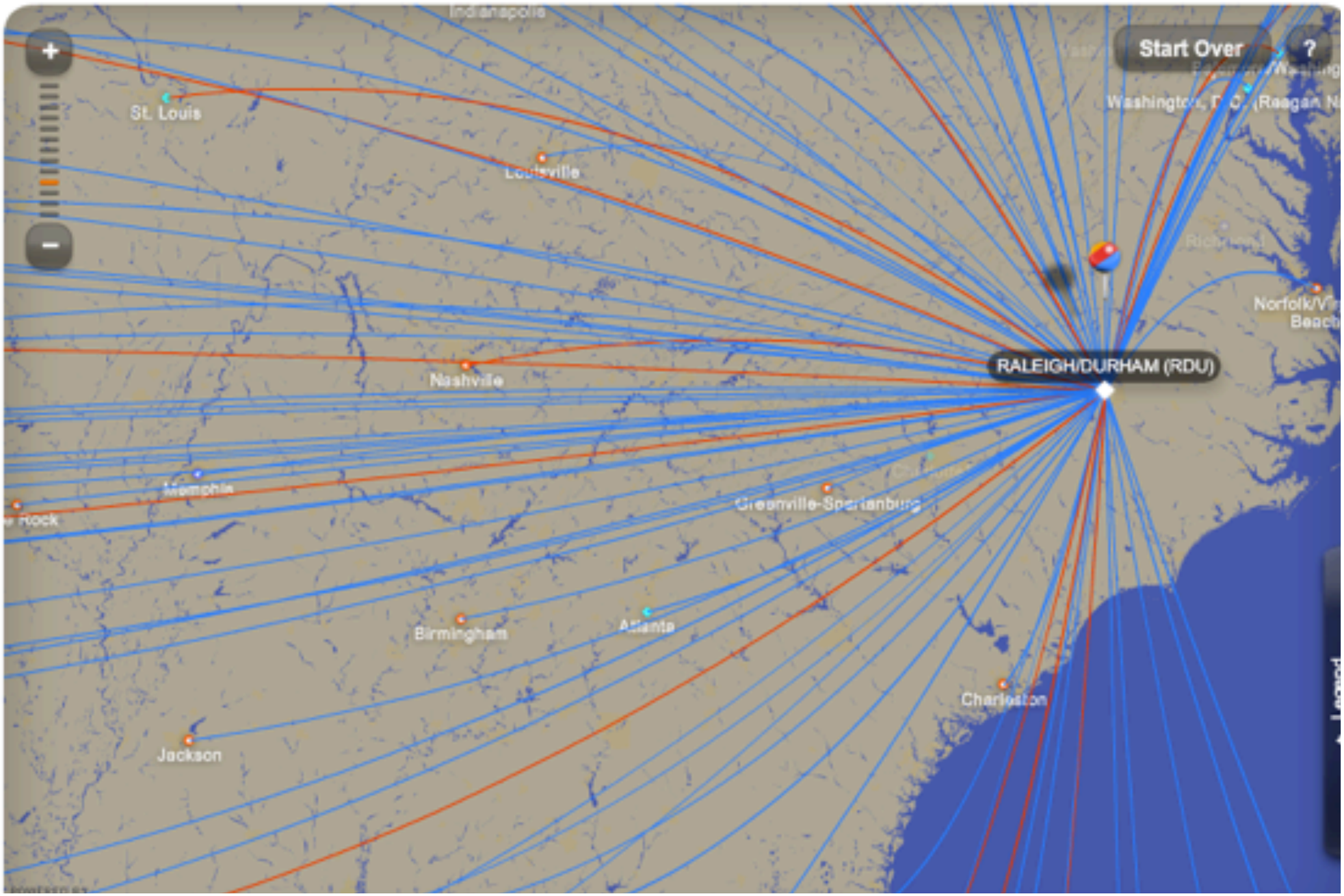


Examples



Nodes are cities (or addresses, maybe).
Edges are roads. Edges have weights.

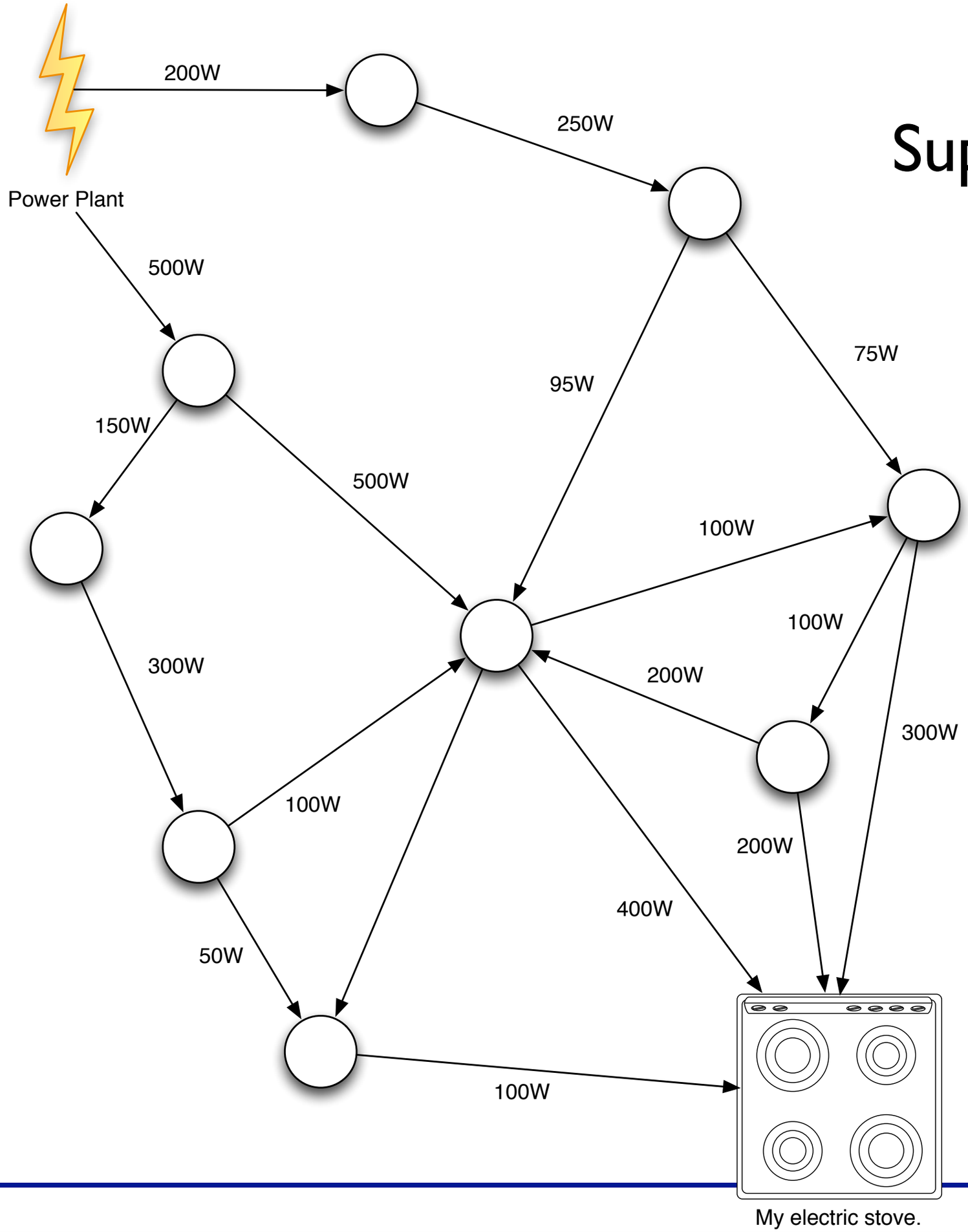
Examples



Nodes are cities. Edges are routes. Edges have weights.

Examples

Suppose you're building a power grid.

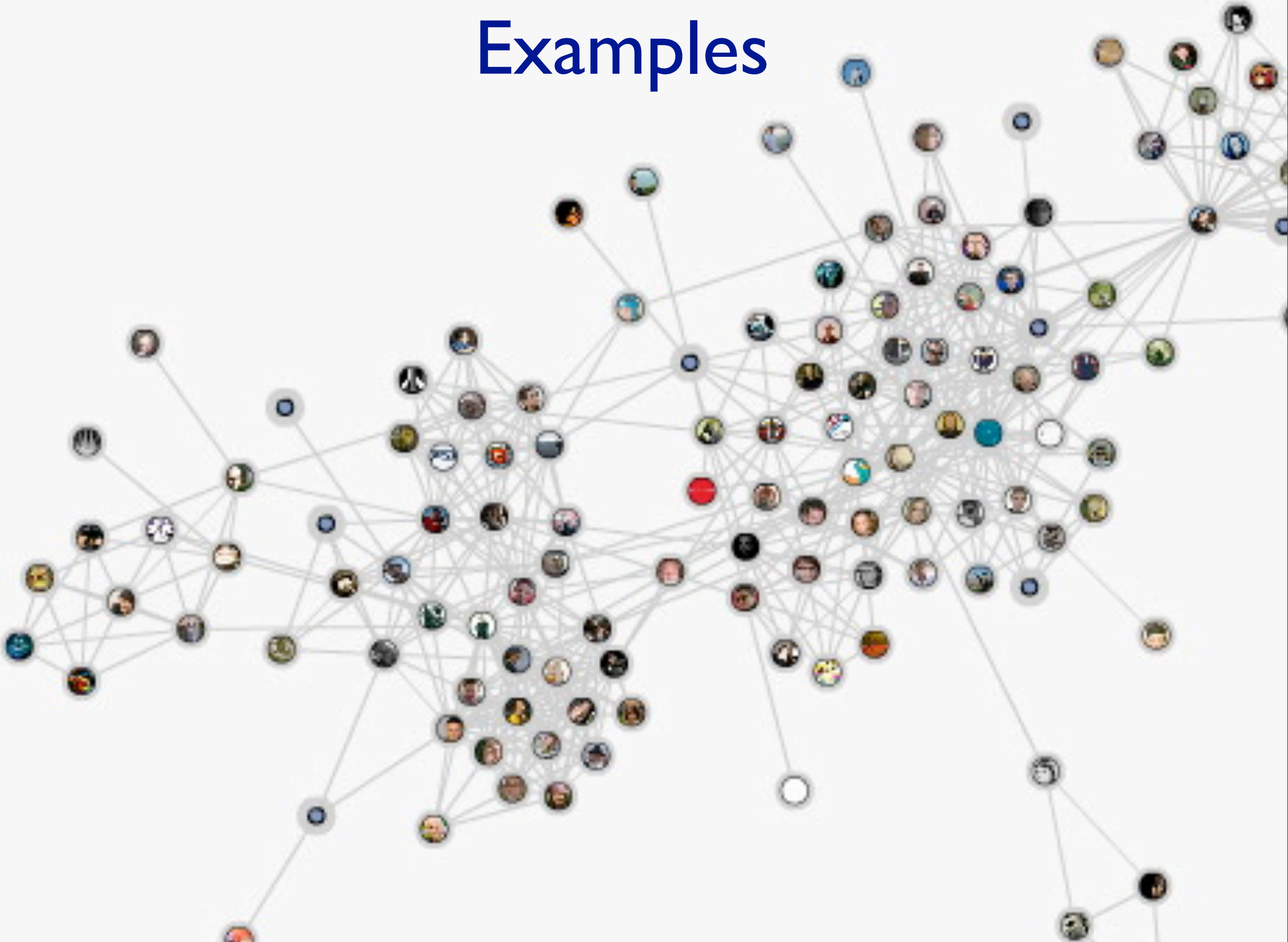


My electric stove.

How much power can my stove draw?

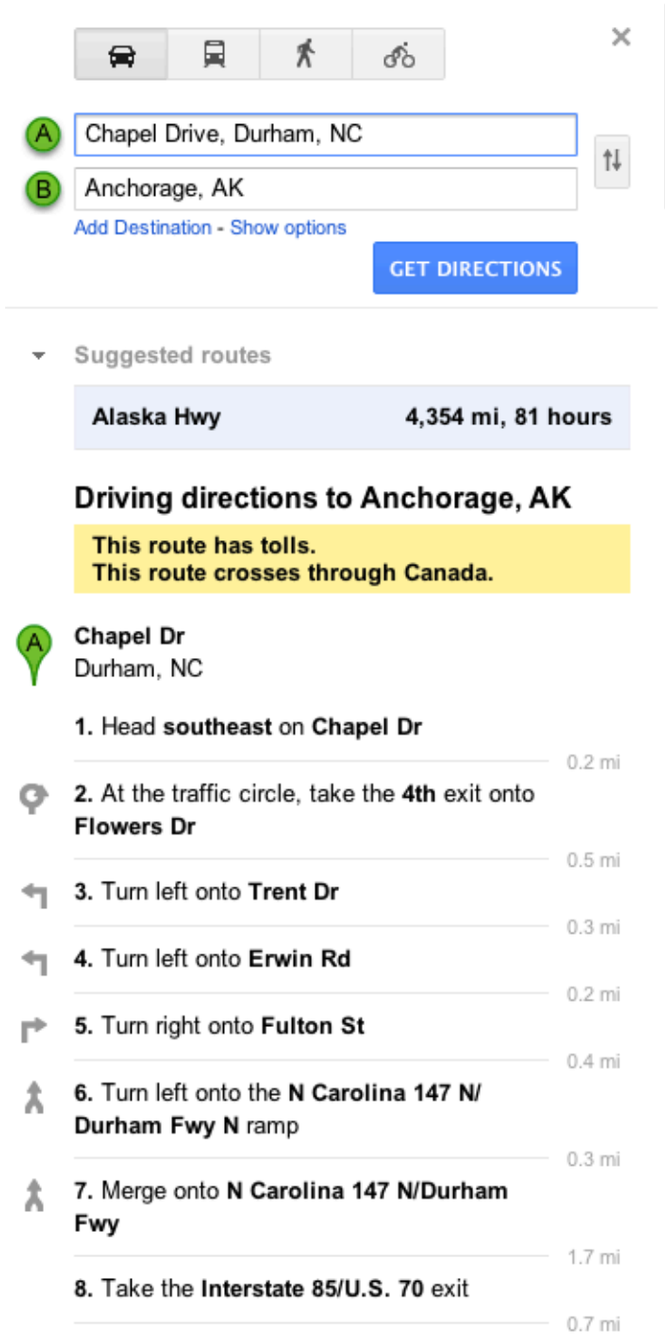
See the Ford-Fulkerson algorithm for details.

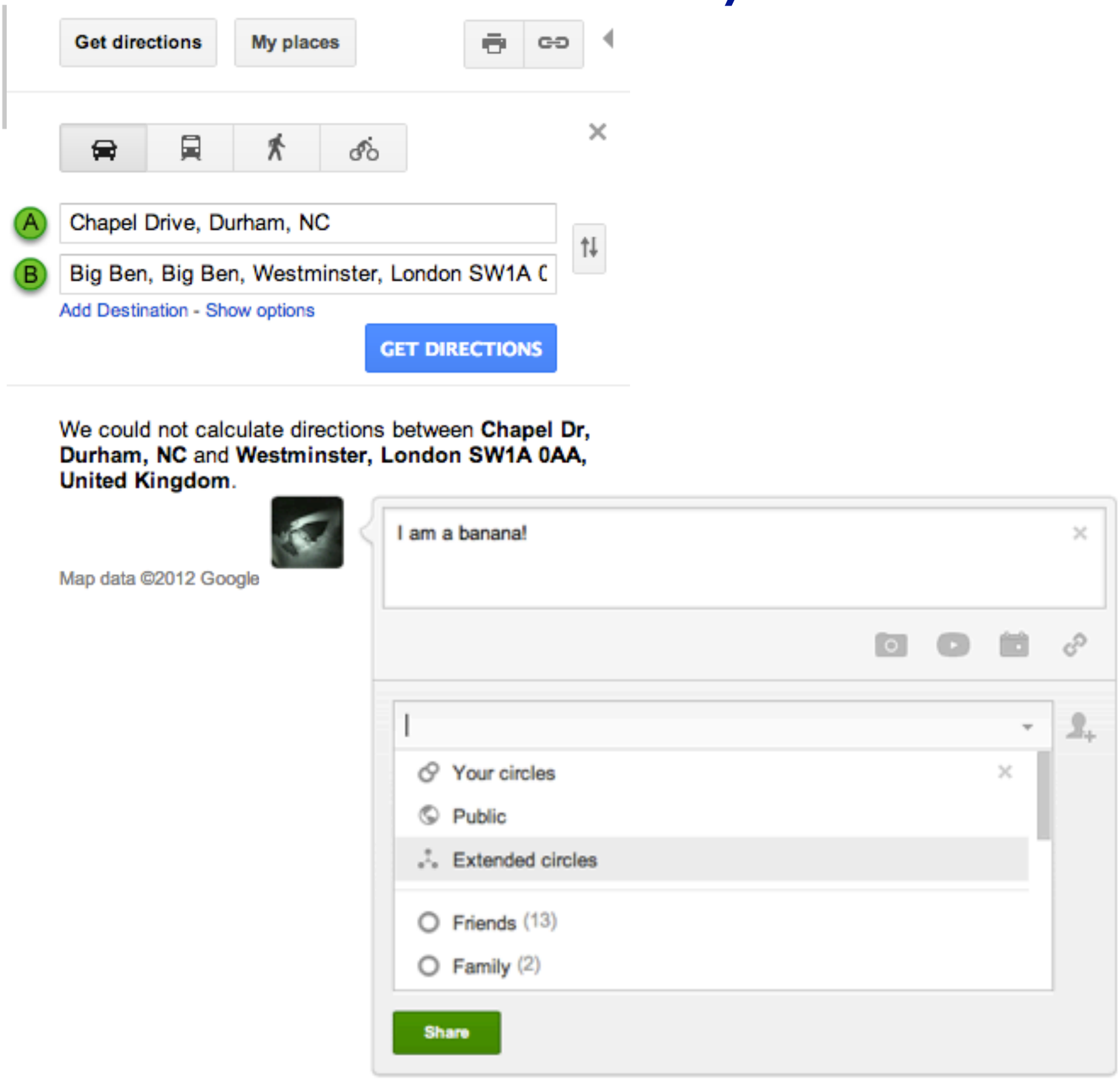
Examples



An example from Facebook. From http://asawicki.info

First question: *connectivity*





Second question: representation

So, if you were going to write a Graph class, what data would you store?

Operations you'll need to support:

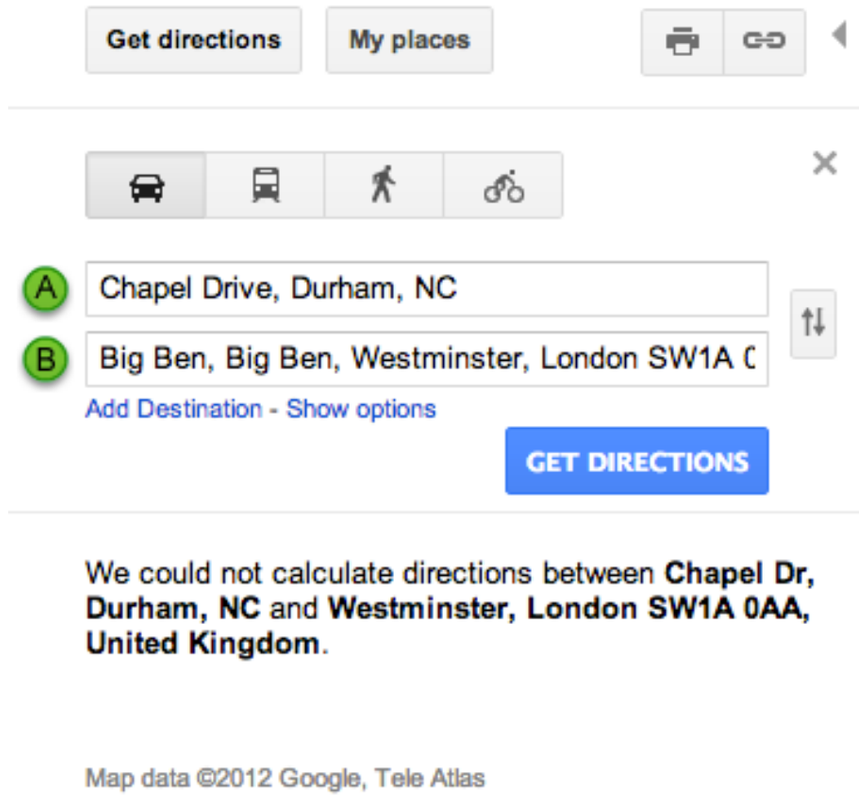
1. Iterating through the nodes.
2. Assigning each node a label.
3. Getting the neighbors of a node.
4. Assigning each edge a label.

Tell us!

http://goo.gl/p1PKN

Back to the first question

Complete connectedTo.



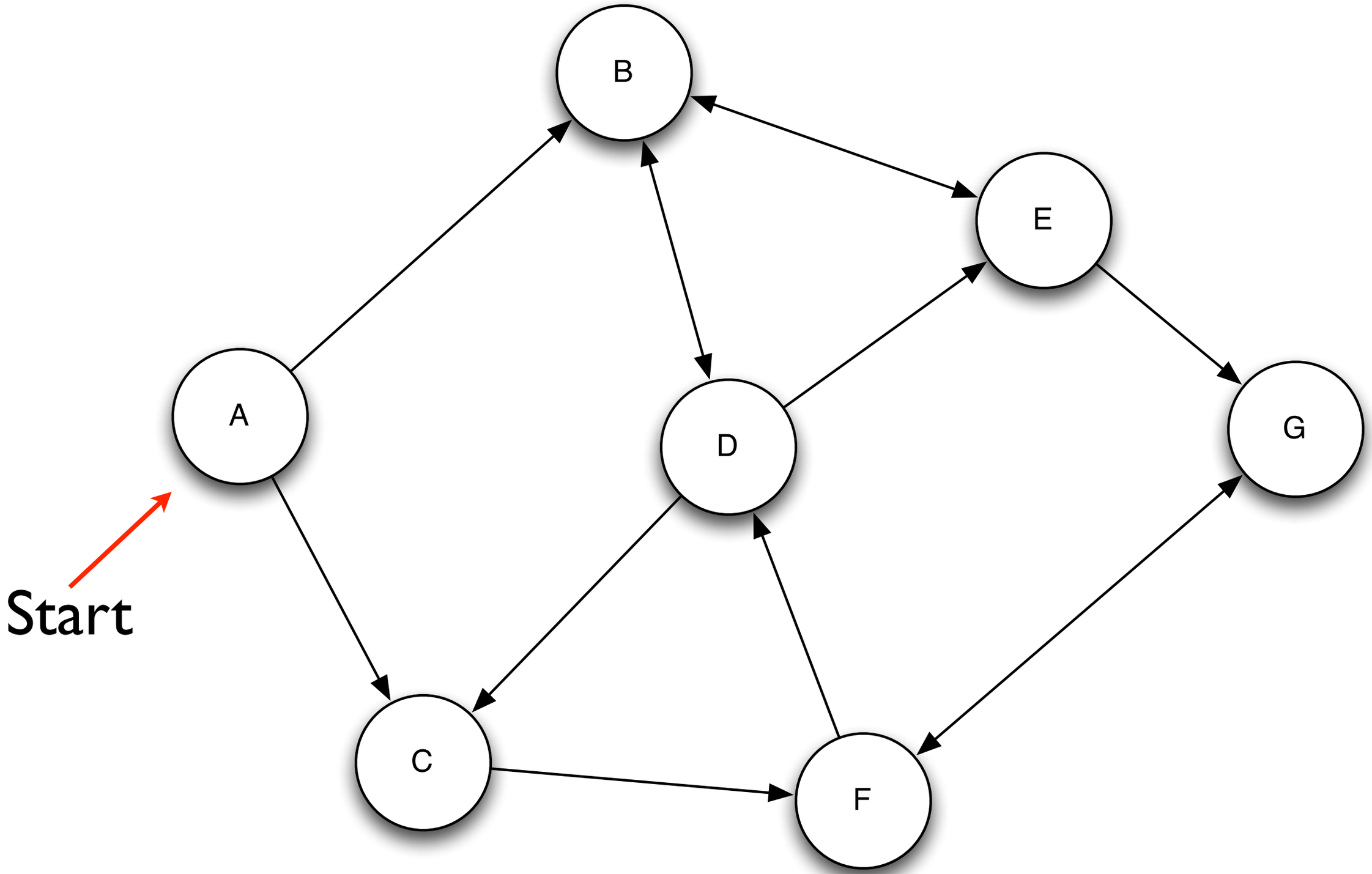

```
/* A node in a generic, directed, graph. */
public class GraphNode {
   private String myLabel;
   private ArrayList<GraphNode> myNeighbors;

   public GraphNode(String l) {
      myLabel = l;
   }

   boolean connectedTo(GraphNode gn) {
      // Can you get to gn from this node?
   }
}
```

This particular implementation is called an *adjacency list*.

Breadth-first & Depth-first Search

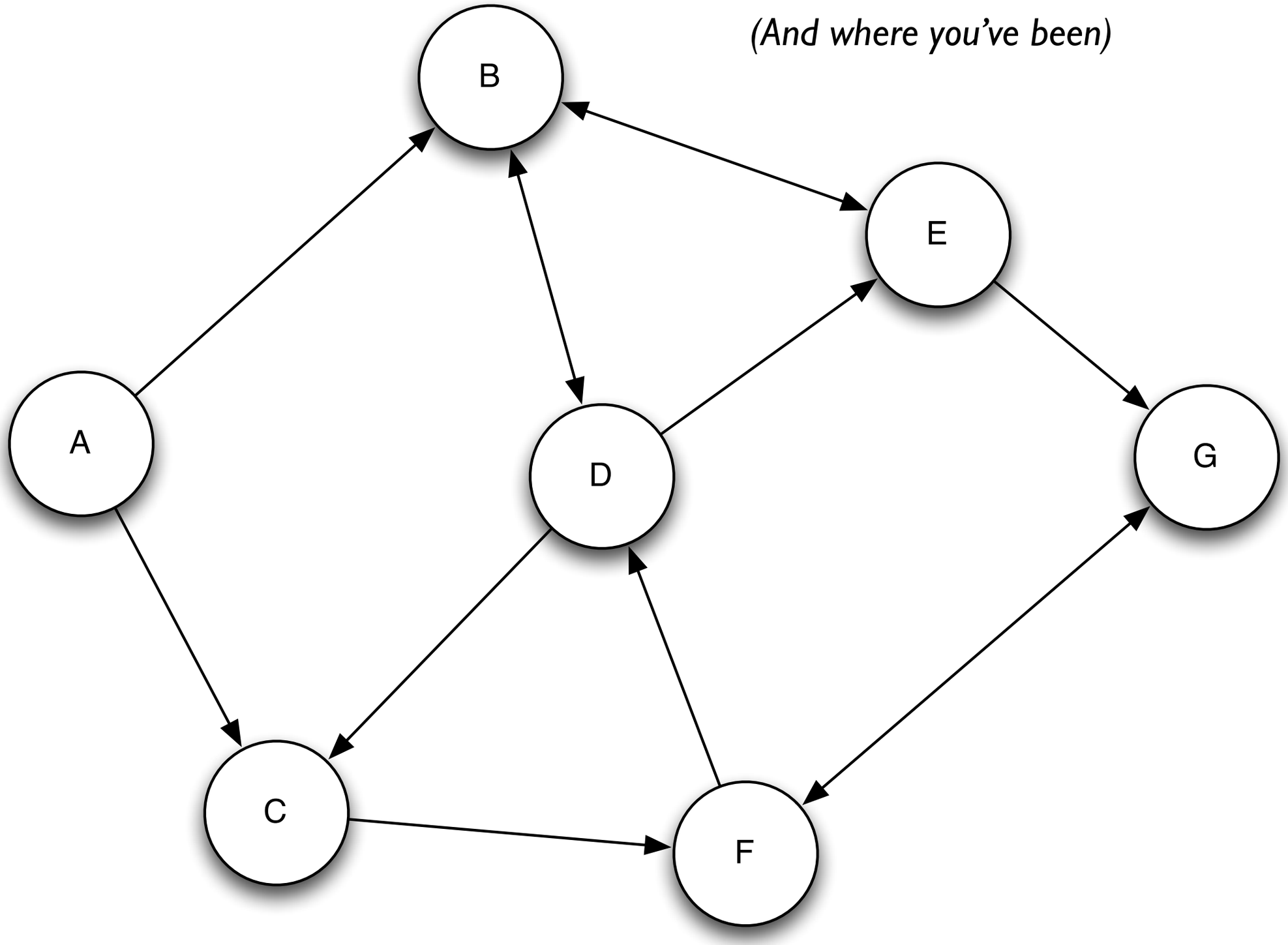


In what order would your code visit these nodes?

Breadth-first & Depth-first Search

Keep track of the *frontier*.

(And where you've been)

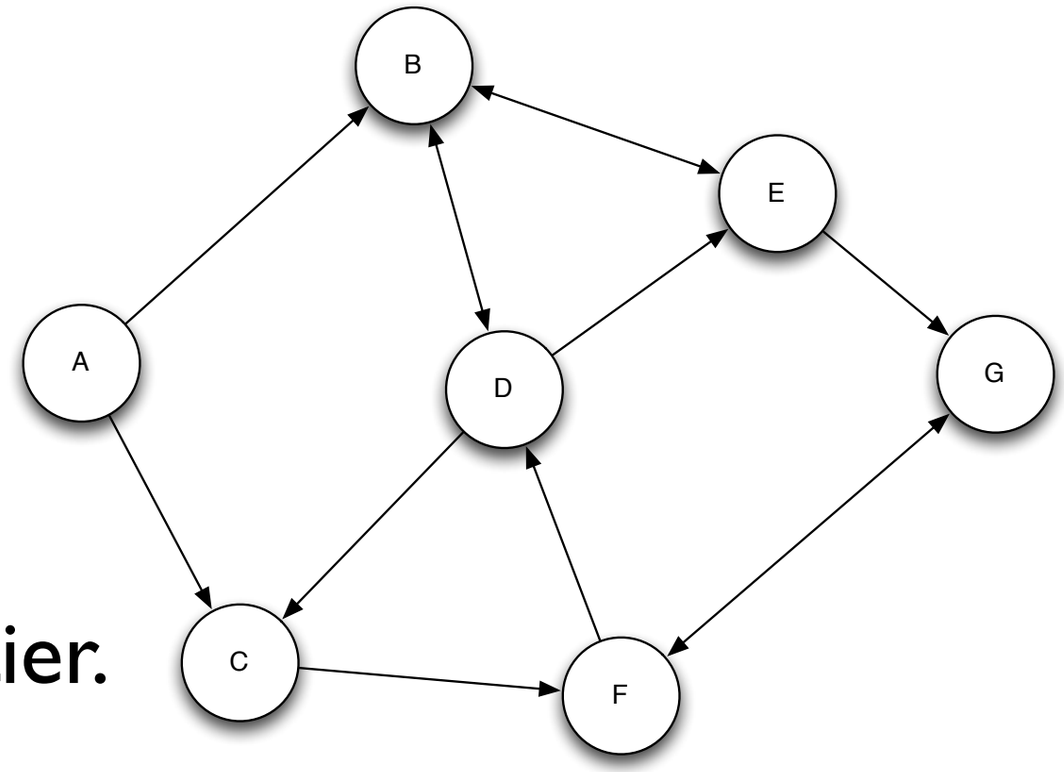


This may remind you of a test question. Funny how that works...

Breadth-first & Depth-first Search

Keep track of the *frontier*.

(And where you've been)



- Add start to your frontier.
- while the frontier isn't empty
  - Pop the first element off the frontier.
  - Process that element.
  - Add that element's neighbors to the frontier.

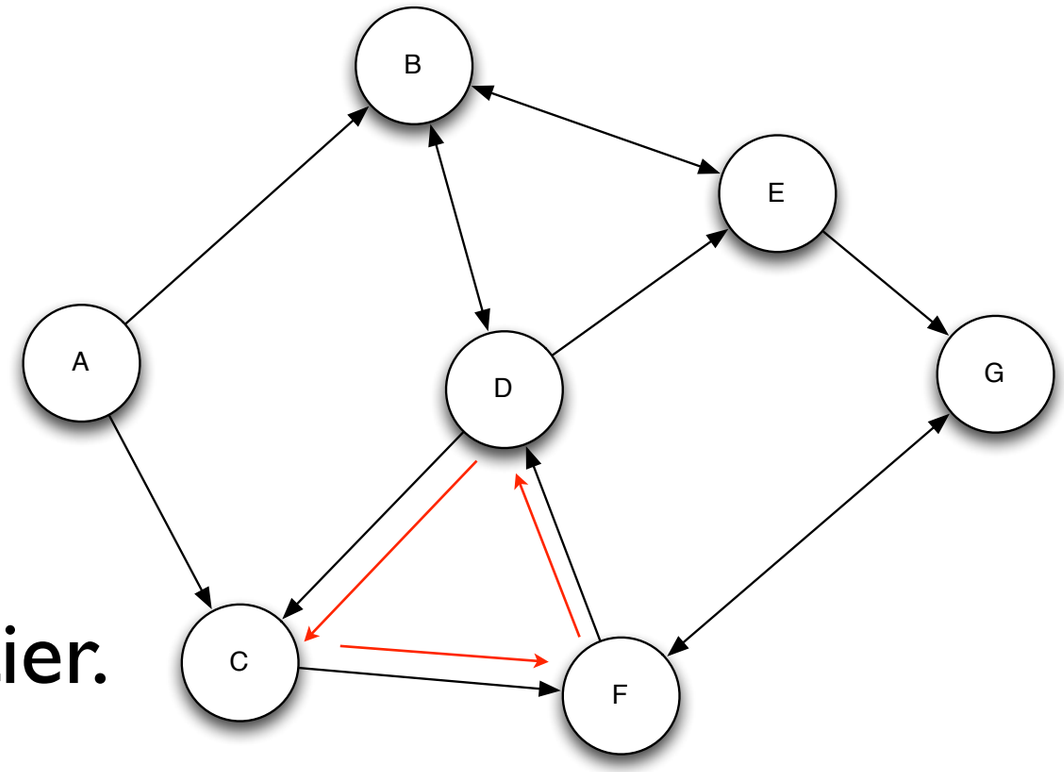
(skipping those you've seen before)

This may remind you of a test question. Funny how that works...

Breadth-first & Depth-first Search

Keep track of the *frontier*.

(And where you've been)



- Add start to your frontier.
- while the frontier isn't empty
  - Pop the first element off the frontier.
  - Process that element.
  - Add that element's neighbors to the frontier.

(skipping those you've seen before)

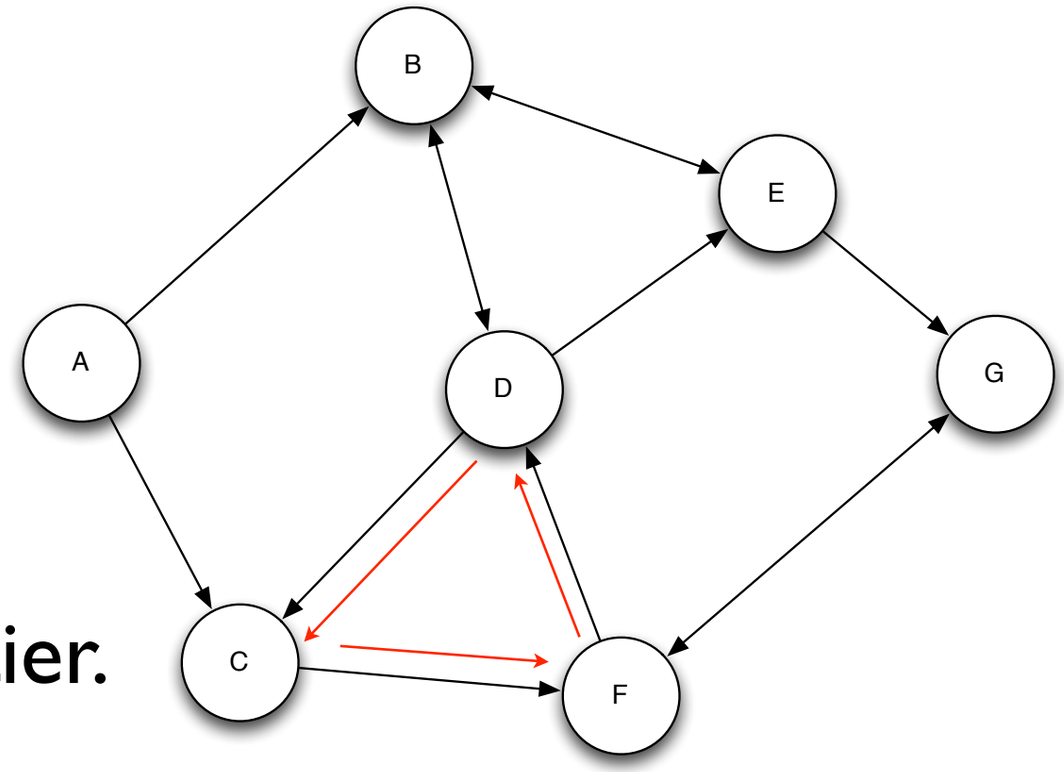
This check is new. Why didn't we use to have to do this?

This may remind you of a test question. Funny how that works...

Breadth-first & Depth-first Search

Keep track of the *frontier*.

(And where you've been)



- Add start to your frontier.
- while the frontier isn't empty
  - Pop the first element off the frontier.
  - Process that element.
  - Add that element's neighbors to the frontier.

(skipping those you've seen before)

This check is new. Why didn't we use to have to do this?

Trees are *directed acyclic graphs*.

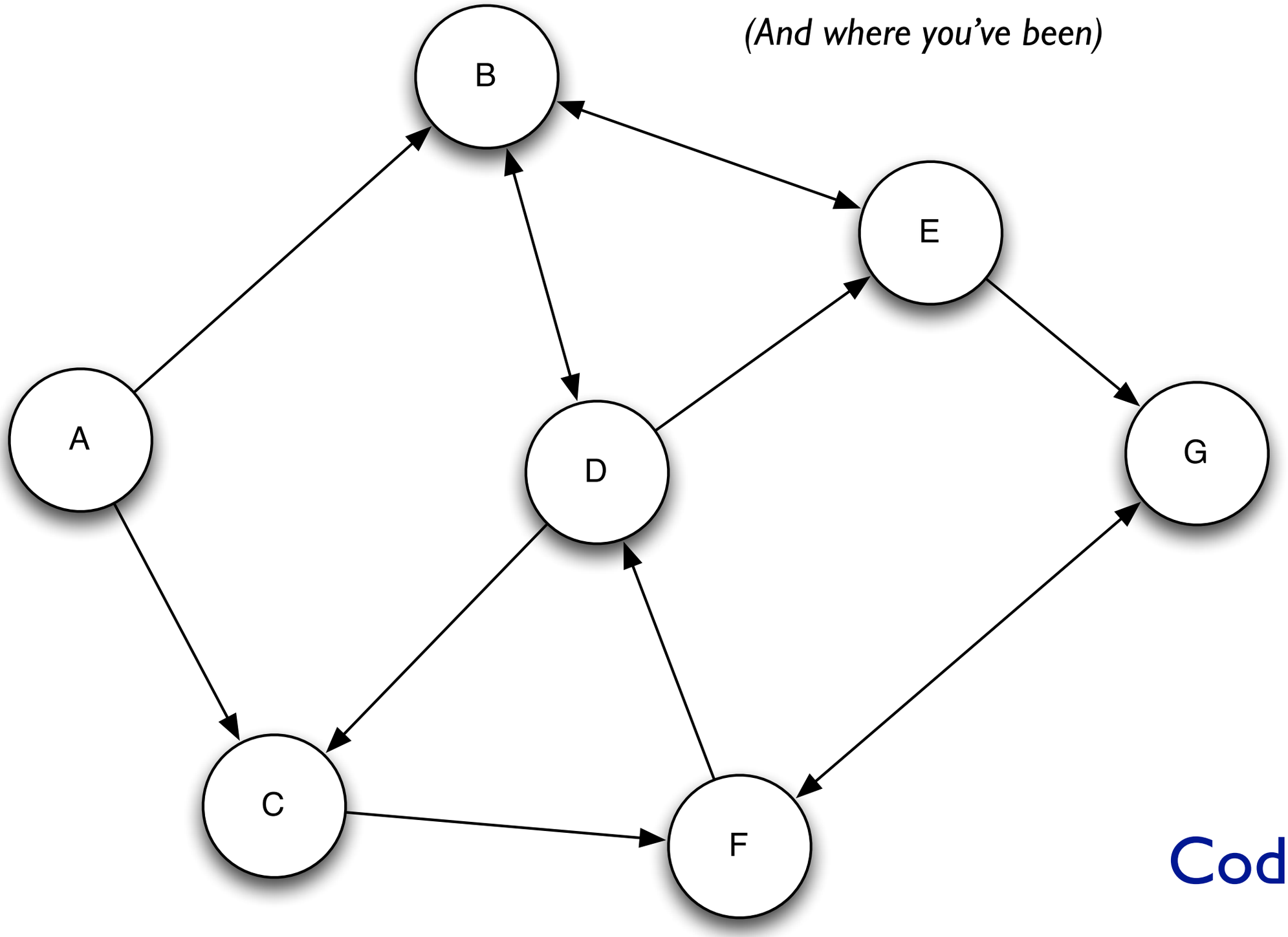
Demo time!

This may remind you of a test question. Funny how that works...

Breadth-first & Depth-first Search

Keep track of the *frontier*.

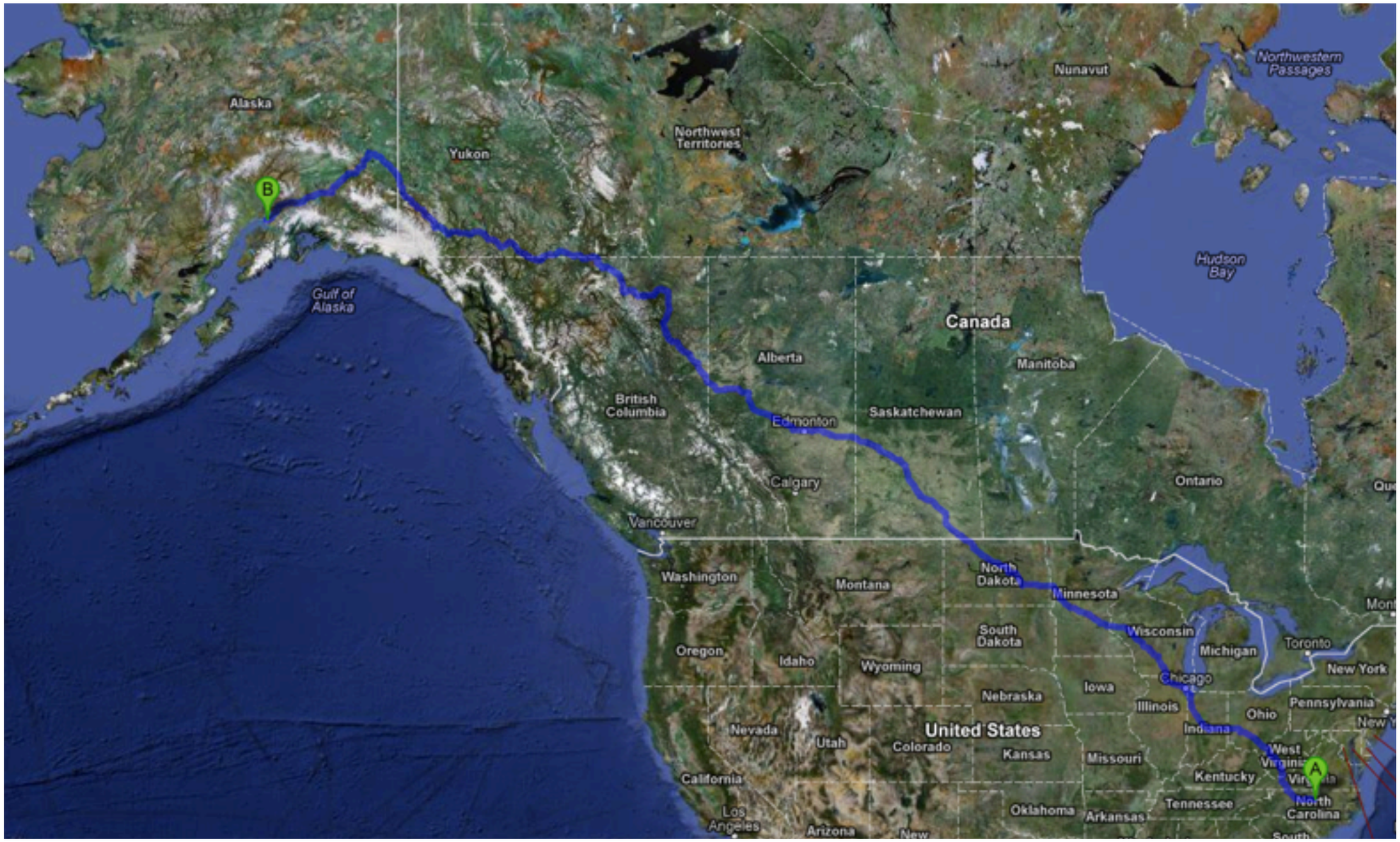
(And where you've been)



Code!

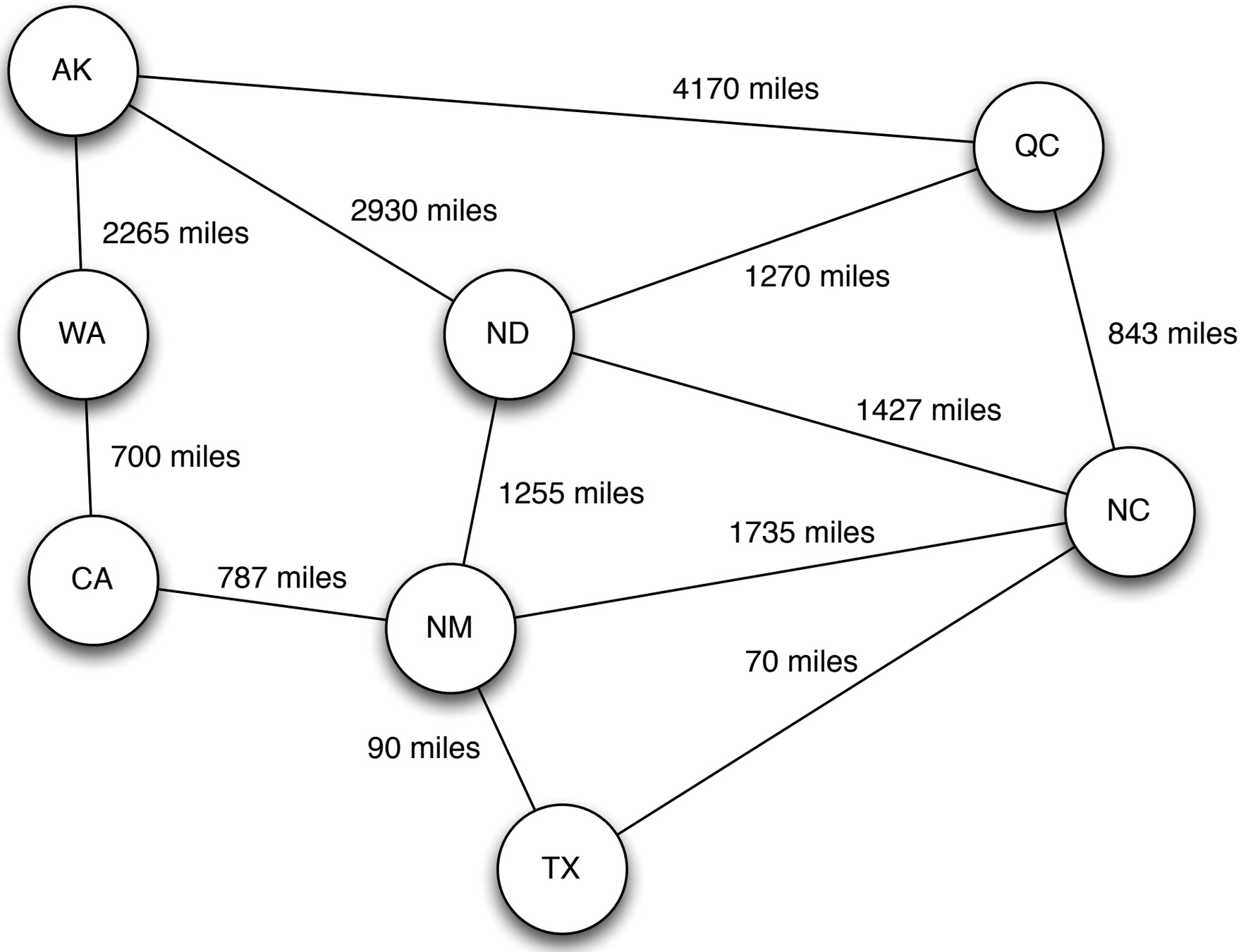
This may remind you of a test question. Funny how that works...

North to Alaska



Nodes are cities (or addresses, maybe).
Edges are roads. Edges have weights.

Connectivity isn't enough.



Shortest path problem.

DFS? BFS?

An aside

Inventor of or advocate for:

- Semaphores (used in parallel computation)
- The switchyard algorithm (used in parsing)
- Loops.
- Not using goto. See "Goto considered harmful."
- And a great many funny ways of telling people off:

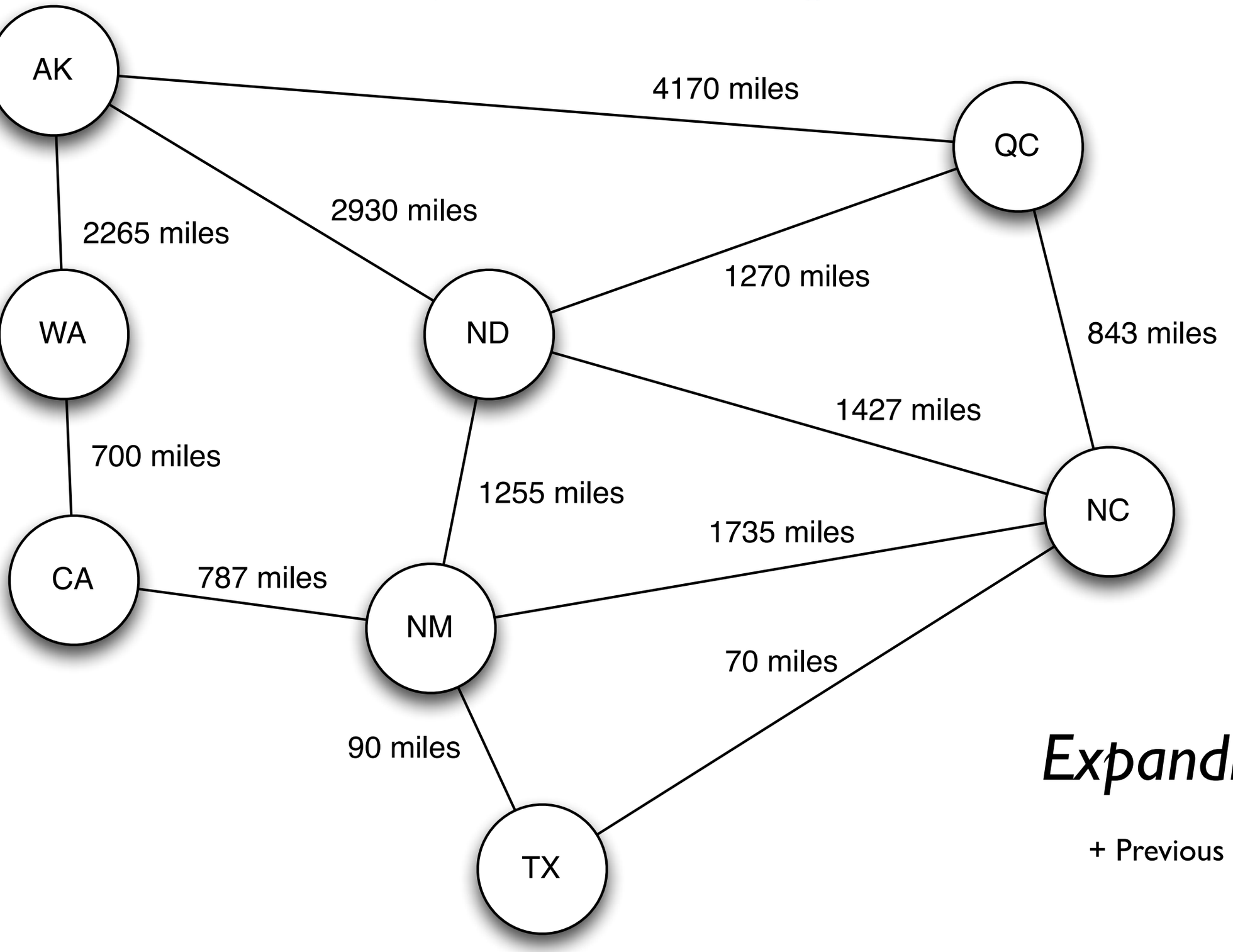
> *It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.*

Edsger W. Dijkstra

On a more philosophical note:

> The job [of operating or using a computer] was actually beyond the electronic technology of the day, and, as a result, the question of how to get and keep the physical equipment more or less in working condition became in the early days the all-overriding concern. As a result, the topic became —primarily in the USA— prematurely known as "computer science" —which, actually is like referring to surgery as "knife science"— and it was firmly implanted in people's minds that computing science is about machines and their peripheral equipment.

Dijkstra's Algorithm

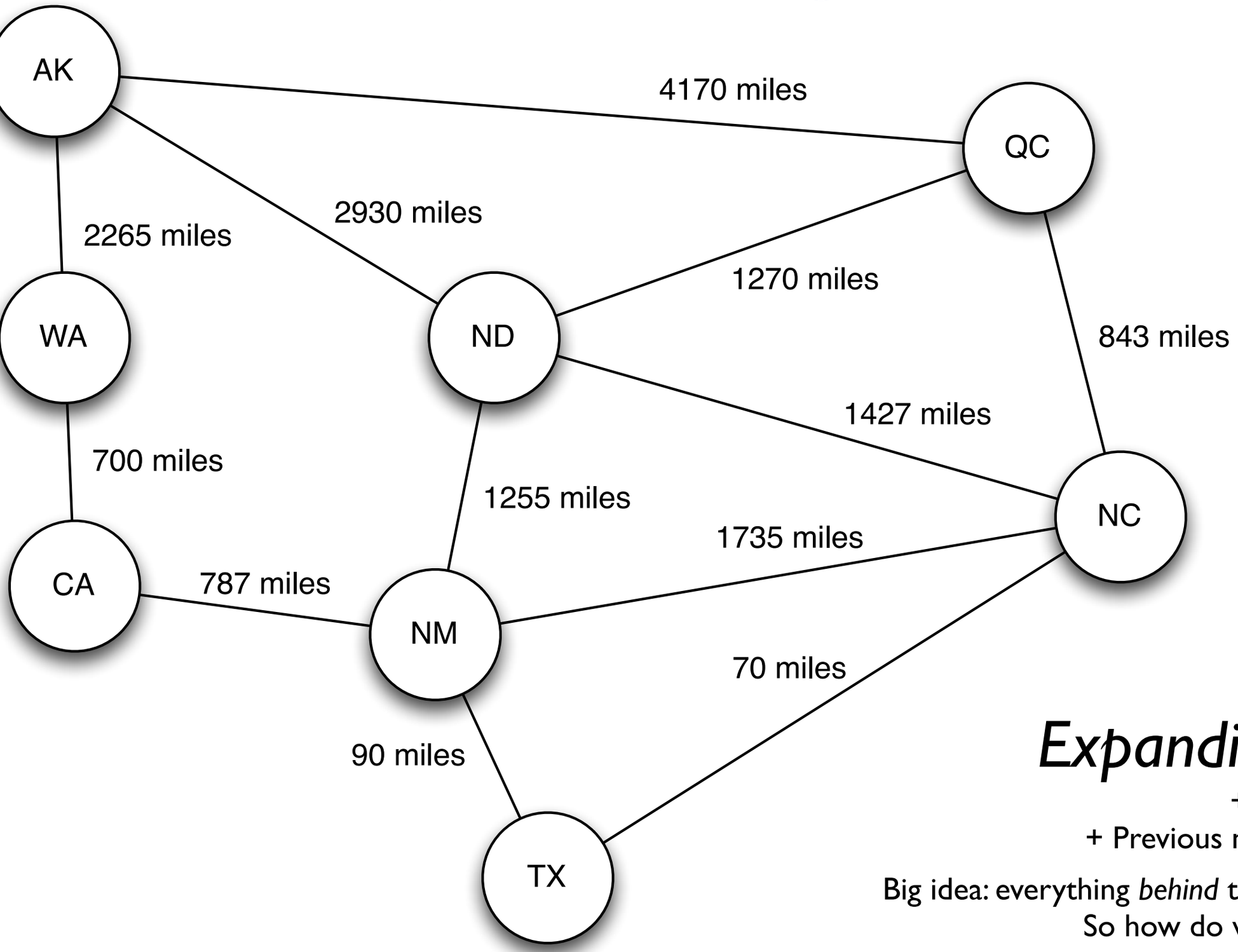


Expanding frontier.

+ Distance from start
+ Previous node in shortest path

You'll need to assume that your edge weights are non-negative.

Dijkstra's Algorithm



Expanding frontier.

+ Distance from start
+ Previous node in shortest path

Big idea: everything *behind* the frontier is correct.
So how do we grow the frontier?

Demo time!

You'll need to assume that your edge weights are non-negative.