Topics for this week

- Data structures:
  - stack (first in last out)
  - queue (first in first out)

- Iterative searches:
  - depth-first search (via a stack)
  - breadth-first search (via a queue)
  - shortest paths (in an unweighted graph)
  - time complexity
Problem: Message Routing Redux

Above: IBM's network backbone (a few years ago).

- How do we find a **path with the smallest number of hops** between two locations?
Problem: Message Routing Redux

How do we find a path with the smallest number of hops between two locations?

BFS (breadth-first-search)

DFS from G
process neighbors alphabetically
Data structure: Stack

- a data structure that stores a sequence of objects
- operations:
  - **push**: insert a new object
  - **pop**: remove the last-inserted object
  - check if **empty**
  - **top**: returns the top of the stack but does not remove it from the stack

Time complexity:
Depth-first search via a stack

- removing recursion from DFS

   define func DFS nonrec (graph, start)
   Create an empty stack
   push the start onto the stack
   mark start as visited
   While stack is not empty:
     current node = pop from the stack
     for n a neighbor of the current node:
       if n not visited:
         push n onto the stack
         mark n as visited

   recursive:

   def DFS (node, visited):
     for n a neighbor of node:
       if n not visited:
         mark n as visited
         DFS (n, visited)

Note: in essence the same as the recursive DFS but nodes processed in slightly different order
Data structure: Queue

- a data structure that stores a sequence of objects

- operations:
  - enqueue: insert a new object
  - dequeue: remove the first-inserted object
  - check if empty

```
\[\text{\begin{array}{c}
7 & 4 & 6 & 8 & 2
\end{array}}\]
```

enqueue: 3, 7, 4, 6
dequeue ← 3
enqueue: 8, 2
dequeue 4 times ← 7 then 4 then 6 then 8

Time complexity: \(O(1)\)  😊
Breadth-first search

- like the iterative DFS but uses a queue instead of a stack

Exactly like the DFS, just replace the stack, push, pop by queue, enqueue and dequeue!
Breadth-first search: finding the path

- how to find the path from start to finish?

Keep track of the node that you came from (so-called predecessor).

Trace the predecessors from finish to start.
Time complexity of BFS and DFS

- crude bound:

- better bound: