Topics for this week

- Python classes:
  - constructor _init_
  - string representation _str_

- Graphs
  - a generalization of trees (another useful datastructure)
  - adjacency lists representation
  - depth-first-search traversal (DFS)
  - path reconstruction
  - time complexity of DFS
Problem: Message Routing

Above: IBM’s network backbone (a few years ago).
- How do we find a path between two locations?
- How do we represent the problem?
Graphs

- a data structure with a set of **nodes** and **edges**, each edge connects two nodes (directed or undirected connection).

Examples:

Note 1: Nodes are also known as **vertices**.

Note 2: **Neighbors** of a node $u$ are all nodes that are connected (adjacent) to $u$ by an edge.
**Graphs: adjacency lists representation**

An adjacency list representation:

For each node, we store a list of its neighbors:

<table>
<thead>
<tr>
<th>Node</th>
<th>Adjacency List of the Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B, C, G</td>
</tr>
<tr>
<td>B</td>
<td>A, D</td>
</tr>
<tr>
<td>C</td>
<td>A, F</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>C, G, E</td>
</tr>
<tr>
<td>G</td>
<td>A, F</td>
</tr>
</tbody>
</table>

For directed:

<table>
<thead>
<tr>
<th>Node</th>
<th>Adjacency List of the Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>A, B, D</td>
</tr>
<tr>
<td>D</td>
<td>null</td>
</tr>
</tbody>
</table>

**Adjacency matrix representation:**

\[
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
Graphs: Depth-first-search (DFS)

How do we find all possible nodes that are reachable from the a given node? (I.e., how do we traverse the graph from the given node - the start?)

Pseudo code (the first attempt):

idea: recursively check all neighbors

define func DFS(start):
    go through the neighbors of start:
        DFS(neighrbor)
Graphs: Depth-first-search (DFS)

How do we find all possible nodes that are reachable from the a given node? (I.e., how do we traverse the graph from the given node - the start?)

Pseudo code:

```python
define func DFSmain(start):
    initialize every node as not visited
    DFS(start)

define func DFS(start):
    mark start as visited
    for a neighbor x of start: (i.e. loop through start's neighbors)
        if x has not been visited:
            DFS(x)
```
Graphs: Depth-first-search (DFS)

How do we find all possible nodes that are reachable from the a given node? (I.e., how do we traverse the graph from the given node - the start?)

Pseudo code:

```
stack frame "diagram":

DFS(G)
  DFS(A)
    DFS(B)
      no DFS(A)
    no DFS(B)
  DFS(C)
    no DFS(A)
    DFS(F)
      no DFS(C)
      no DFS(G)
    DFS(E)
      no DFS(F)
  no DFS(F)
```

adj lists:

- A: B, C, G
- B: A, D
- C: A, F
- D: B
- E: F
- F: C, G, E
- G: A, F

No DFS means that node already visited.
Graphs: Depth-first-search (DFS)

We are almost done 😊 The last thing: how do we reconstruct the path from the start to the finish?

Pseudo code:

```python
def DFS_paths_main(start, finish):
    initialize every node as not visited
    return DFS_paths(start, finish)

def DFS_paths(start, finish):
    mark start as visited
    for a neighbor x of start:
        if x has not been visited:
            p = DFS_paths(x, finish)
            if p ≠ None:
                return [start] + p
    return None
```
Graphs: Depth-first-search (DFS)

The last bits...

Time complexity:

Testing: see the nodes