Intro to Graphs

- **Definitions and Vocabulary**
  - A graph consists of a set of vertices (or nodes) and a set of edges (or arcs) where each edge connects a pair of vertices.
  - If the pair of vertices defining an edge is ordered, then it is a directed graph.
  - A vertex may have information called a label.
  - An edge may have information called a weight or cost.
  - A vertex i is adjacent to j if there is an edge from j to i.
**Intro to Graphs**

- **Definitions and Vocabulary**
  - A *path* is a sequence of adjacent vertices with a *length* equal to the number of edges on the path. This is also known as the *unweighted path length*. The *weighted path length* is the sum of the costs of the edges of the path.
  - A *cycle* is a path of at least length one where the first and last vertex are the same.
Graph Representation

- **Adjacency matrix**
  - Row and column numbers represent vertices
  - Cells represent edges
    - Use true/false for unweighted graphs
    - Use weights for weighted graphs with special value (infinity) for no connection
  - Can have separate vector of vertex labels
    Algorithms use integers as identifiers
  - $O(N^2)$ space: often sparse; much wasted space
# Graph Representation

- **How far from A to B?**

<table>
<thead>
<tr>
<th></th>
<th>Asheville</th>
<th>Durham</th>
<th>Greensboro</th>
<th>Manteo</th>
<th>Murphy</th>
<th>Raleigh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asheville</td>
<td>0</td>
<td>231</td>
<td>178</td>
<td>438</td>
<td>105</td>
<td>241</td>
</tr>
<tr>
<td>Durham</td>
<td>231</td>
<td>0</td>
<td>54</td>
<td>203</td>
<td>340</td>
<td>23</td>
</tr>
<tr>
<td>Greensboro</td>
<td>178</td>
<td>54</td>
<td>0</td>
<td>259</td>
<td>283</td>
<td>78</td>
</tr>
<tr>
<td>Manteo</td>
<td>438</td>
<td>203</td>
<td>259</td>
<td>0</td>
<td>543</td>
<td>197</td>
</tr>
<tr>
<td>Murphy</td>
<td>105</td>
<td>340</td>
<td>283</td>
<td>543</td>
<td>0</td>
<td>355</td>
</tr>
<tr>
<td>Raleigh</td>
<td>241</td>
<td>23</td>
<td>78</td>
<td>197</td>
<td>355</td>
<td>0</td>
</tr>
</tbody>
</table>
Graph Representation

- **Adjacency lists (Edge lists)**
  - Use vector to represent all vertices where index identifies vertex
    - Each node in the vector can include a vertex label
  - Use linked lists to represent edges from these vertices
    - Each node in the linked list identifies a vertex and, optionally, edge cost
  - $O(N)$ space when sparse; $O(N^2)$ when dense
Graph Representation

- Adjacency List

```
0  v0   ─ 1  ─ 4
1  v1   ─ 3
2  v2   ─ 0
3  v3   ─ 5  ─ 6
4  v4
5  v5
6  v6   ─ 1
```
Graphs

- Totally linked versions are also possible
- Special case
  - General Trees
  - "Naturally Corresponding" Binary Trees

- Working with graphs:

  Marking (I've been here! ... and more ...)
  - Cave or maze exploration
  - How have binary tree algorithms avoided the need for such marks?
Graph Traversals

- **Traversals: Depth First or Breadth First?**
  - What if vertices represent chess boards (i.e., positions)?
  - What is a pre-order traversal of a binary tree?
  - What is a level-order traversal of a binary tree?
Depth First Search (recursive)
- Un-mark all vertices (pre search initialization!!!)
- Process and mark starting vertex
- For each unmarked adjacent vertex do Depth First Search
Breadth First Search

- Un-mark all vertices
- Process and mark starting vertex and place in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each unmarked adjacent vertex
     - process it
     - mark it
     - place it on the queue
Breadth First Search

- What if we apply this to binary tree?

- What would you name this traversal?
Graph Algorithms

- **Topological Sort**
  - Produce a valid ordering of all nodes, given pairwise constraints
  - Solution usually not unique
  - When is solution impossible?

- **Topological Sort Example: Getting an AB in CPS**
  - Express prerequisite structure
  - This example, CPS courses only: 6, 100, 104, 108, 110, 130
  - Ignore electives or outside requirements (can add later)
Intro to Graphs

- **Topological Sort Algorithm**
  1. Find vertex with no incoming edges
  2. Remove (updating incoming edge counts) and Output
  3. Repeat 1 and 2 while vertices remain
     - Complexity?

- **Refine Algorithm**
  - Use queue? (and marking)
  - Complexity?

- **What is the minimum number of semesters required?**
  - Develop algorithm
Intro to Graphs

- Shortest Path
- Traveling Salesman