**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 \( i \).

- **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
  - **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
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

- 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, and place it on the queue.
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**