A graph consists of:

- A set of nodes or vertices (terms are interchangeable)
- A set of edges or arcs (terms are interchangeable)

Edges in graph can be either directed or undirected.
Edges can be labeled or unlabeled
• Edge labels are typically the cost associated with an edge
• e.g., Nodes are cities, edges are roads between cities, edge label is the length of road
There are several problems that are “naturally” graph problems:

- Networking problems
- Route planning
- etc

Problems that don’t *seem* like graph problems can also be solved with graphs:

- Register allocation using graph coloring
15-3: Connected Undirected Graph

- Path from every node to every other node

- Connected
15-4: Connected Undirected Graph

- Path from every node to every other node

- Connected
15-5: Connected Undirected Graph

- Path from every node to every other node

- *Not Connected*
15-6: **Strongly Connected Graph**

- Directed Path from every node to every other node

```
1 -> 2
1 -> 3
2 -> 1
3 -> 4
3 -> 5
4 -> 1
5 -> 3
```

- Strongly Connected
15-7: **Strongly Connected Graph**

- Directed Path from every node to every other node

- Strongly Connected
15-8: **Strongly Connected Graph**

- Directed Path from every node to every other node

- Not Strongly Connected
Weakly Connected Graph

- Directed graph w/ connected backbone
- Weakly Connected
15-10: Cycles in Graphs

- Undirected cycles

![Diagram of a graph with undirected cycle]

- Contains an undirected cycle
Cycles in Graphs

- Undirected cycles
- Contains an undirected cycle
15-12: Cycles in Graphs

- Undirected cycles

Contains no undirected cycle
15-13: Cycles in Graphs

- Undirected cycles

Contains no undirected cycle
• Directed cycles

• Contains a directed cycle
Cycles in Graphs

- Directed cycles

- Contains a directed cycle
Cycles in Graphs

- Directed cycles

Contains a directed cycle
Directed cycles

Contains *no* directed cycle
Must a connected, undirected graph contain a cycle?
15-19: Cycles & Connectivity

- Must a connected, undirected graph contain a cycle?
  - No.
- Can an unconnected, undirected graph contain a cycle?
Cycles & Connectivity

- Must a connected, undirected graph contain a cycle?
  - No.

- Can an unconnected, undirected graph contain a cycle?
  - Yes.

- Must a strongly connected graph contain a cycle?
Must a connected, undirected graph contain a cycle?
- No.

Can an unconnected, undirected graph contain a cycle?
- Yes.

Must a strongly connected graph contain a cycle?
- Yes! (why?)
If a graph is weakly connected, and contains a cycle, must it be strongly connected?
If a graph is weakly connected, and contains a cycle, must it be strongly connected?
• No.
If a graph is weakly connected, and contains a cycle, must it be strongly connected?

- No.

If a graph contains a cycle which contains all nodes, must the graph be strongly connected?
If a graph is weakly connected, and contains a cycle, must it be strongly connected?
• No.

If a graph contains a cycle which contains all nodes, must the graph be strongly connected?
• Yes. (why?)
**Adjacency Matrix**

- Represent a graph with a two-dimensional array $G$
  - $G[i][j] = 1$ if there is an edge from node $i$ to node $j$
  - $G[i][j] = 0$ if there is no edge from node $i$ to node $j$
- If graph is undirected, matrix is symmetric
- Can represent edges labeled with a cost as well:
  - $G[i][j] = \text{cost of link between } i \text{ and } j$
  - If there is no direct link, $G[i][j] = \infty$
Examples:

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 \\
2 & 1 & 0 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 \\
\end{array}
\]
15-28: **Adjacency Matrix**

- **Examples:**

```
0 1 2 3
0 0 1 0 0
1 1 0 1 1
2 0 0 0 0
3 1 0 0 0
```
15-29: Adjacency Matrix

- **Examples:**

```
0 1 2 3 0
0  0  0  0 0
1  1  0  0 0
2  0  1  0 0
3  0  0  0 1
```
15-30: Adjacency Matrix

- Examples:

```
0 1 2 3
0 ∞ ∞ ∞ 5
1 4 ∞ ∞ ∞
2 ∞ 7 ∞ ∞
3 ∞ ∞ -2 ∞
```
Adjacency List

Maintain a linked-list of the neighbors of every vertex.

- $n$ vertices
- Array of $n$ lists, one per vertex
- Each list $i$ contains a list of all vertices adjacent to $i$. 
Examples:
15-33: Adjacency List

- Examples:

- Note – lists are not always sorted
15-34: **Sparse vs. Dense**

- **Sparse graph** – relatively few edges
- **Dense graph** – lots of edges
- **Complete graph** – contains all possible edges
  - These terms are fuzzy. “Sparse” in one context may or may not be “sparse” in a different context
15-35: Nodes with Labels

- If nodes are labeled with strings instead of integers
  - Internally, nodes are still represented as integers
  - Need to associate string labels & vertex numbers
    - Vertex number $\rightarrow$ label
    - Label $\rightarrow$ vertex number
Nodes with Labels

- Vertex numbers → labels
15-37: **Nodes with Labels**

- Vertex numbers $\rightarrow$ labels
  - Array
    - Vertex numbers are indices into array
    - Data in array is string label
Nodes with Labels

- Labels $\rightarrow$ vertex numbers
Nodes with Labels

- Labels $\rightarrow$ vertex numbers
  - Use a hash table
    - Key is the vertex label
    - Data is vertex number

Examples!
15-40: **Topological Sort**

- Directed Acyclic Graph, Vertices $v_1 \ldots v_n$
- Create an ordering of the vertices
  - If there a path from $v_i$ to $v_j$, then $v_i$ appears before $v_j$ in the ordering
- Example: Prerequisite chains
15-41: **Topological Sort**

- Which node(s) could be first in the topological ordering?
Topological Sort

- Which node(s) could be first in the topological ordering?
  - Node with no incident (incoming) edges
**15-43: Topological Sort**

- Pick a node $v_k$ with no incident edges
- Add $v_k$ to the ordering
- Remove $v_k$ and all edges from $v_k$ from the graph
- Repeat until all nodes are picked.
Topological Sort

- How can we find a node with no incident edges?
- Count the incident edges of all nodes
for (i=0; i < NumberOfVertices; i++)
    NumIncident[i] = 0;

for(i=0; i < NumberOfVertices; i++)
    each node k adjacent to i
    NumIncident[k]++
for(i=0; i < NumberofVertices; i++)
    NumIncident[i] = 0;

for(i=0; i < NumberofVertices; i++)
    for(tmp=G[i]; tmp != null; tmp=tmp.next())
        NumIncident[tmp.neighbor()]++
15-47: Topological Sort

- Create NumIncident array
- Repeat
  - Search through NumIncident to find a vertex $v$ with $\text{NumIncident}[v] == 0$
  - Add $v$ to the ordering
  - Decrement NumIncident of all neighbors of $v$
  - Set $\text{NumIncident}[v] = -1$
- Until all vertices have been picked
In a graph with $V$ vertices and $E$ edges, how long does this version of topological sort take?
In a graph with $V$ vertices and $E$ edges, how long does this version of topological sort take?

- $\Theta(V^2 + E) = \Theta(V^2)$
- Since $E \in O(V^2)$
Where are we spending “extra” time
Topological Sort

• Where are we spending “extra” time
  • Searching through NumIncident each time looking for a vertex with no incident edges
  • Keep around a set of all nodes with no incident edges
  • Remove an element \( v \) from this set, and add it to the ordering
  • Decrement NumIncident for all neighbors of \( v \)
    • If NumIncident\([k]\) is decremented to 0, add \( k \) to the set.
  • How do we implement the set of nodes with no incident edges?
Topological Sort

- Where are we spending “extra” time
  - Searching through NumIncident each time looking for a vertex with no incident edges
  - Keep around a set of all nodes with no incident edges
  - Remove an element \( v \) from this set, and add it to the ordering
  - Decrement NumIncident for all neighbors of \( v \)
    - If NumIncident[\( k \)] is decremented to 0, add \( k \) to the set.
- How do we implement the set of nodes with no incident edges?
  - Use a stack
Examples!!

- Graph
- Adjacency List
- NumIncident
- Stack