Graphs

- Representation of things and relationships between them
- Vertices: people
- Edges: relationships

Undirected

Simple graph: no self-loops, at most one edge per pair of vertices (per directed, if directed)

Paths: seq of verts s.t. edge exists between adj verts
JG SG KK JB

Simple path: no repeated vertices
JG SG JG SG KK JB

Cycle: path w/ same start/end
JG SG JG JG

Simple cycle: only repeat is start/end

Path:
Simple path:
Cycle:
Simple cycle:
Weighted Graph

Each edge is labeled with weight.

- **Vertices**: cities
- **Edges**: roads between cities
- **Weights**: distance along roads

Source: Rand McNally 2012 Road Atlas
int foo(int n, int c)
{
    if (n == c)
    {
        return 0;
    }

    int i = 1;
    while (i < n)
    {
        if (i % c == 3)
        {
            if (n % 2 == 1)
            {
                return 0;
            }
        }
        i++;
    }
}
Feedback Arc Set

Y, D, Col, H, Pr

vertices: teams

edges: u -> v means v beat u in a game

Feedback Arc Set: what is min num edges you need to remove to make graph acyclic (no cycles)

NP-complete (like TSP)

is there a cycle?

if not, find ordering so all edges go in same dir

if so, find ordering to minimize # of wrong-way edges
Graph Representation

Adjacency Matrix:

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>D</th>
<th>Pr</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Pr</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>H</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

**Space:** \(\Theta(n^2)\)

Unweighted - entries are T/F
Weight - entries are weights or "not this"

Adjacency List:

- Y: \(Y, D\)
- D: \(Y, D, H\)
- Pr: \(Col, D, Y\)
- H: \(\) (empty)

**Adj List:** use hash table or set representation

- Worst case: \(\Theta(n^2)\)
- Best case: \(\Theta(n)\)
- \(n\): vertices, \(m\): edges

Has-edge (from, to):
- Adj matrix: array lookup \(O(1)\)
- Adj list: seg. search \(O(n)\)

Add-edge (from, to):
- Adj matrix: array update \(O(1)\)
- Adj list: add to list \(O(1)\) amortised

For each edge:
- Adj matrix:
  - for each row \(r\)
    - for each col \(c\): \(\Theta(n^2)\)
  - if \(adj[r][c] = T\) \(proceed\) edge \((r,c)\)

- Adj list:
  - \(\Theta(nm)\)
  - for each vertex \(u\)
    - \(\) for each \(v\) in \(adj[u]\)
      - \(\) for each \(v\) in \(adj[u]\)
        - process edge \((u,v)\)
  - worst case: \(\Theta(n^2)\)
\[ \Theta(nm) \] for each \( v \) in \( \text{adj}(u) \)
\[ \text{process_edge}(u,v) \]
\[ \Theta(n) \] for sparse
\[ \Theta(n^2) \] for dense

\[ \sum_{i=0}^{n-1} (1 + \text{outdegree}(v; i)) \]
\[ = \sum_{i=0}^{n-1} 1 + \sum_{i=0}^{n-1} \text{outdegree}(v; i) \]
\[ = n + m \]

Sparse: \( m \in \Theta(n) \)
Dense: \( m \in \Theta(n^2) \)
## Graph Implementation Time/Space Complexity

<table>
<thead>
<tr>
<th></th>
<th>Adj Matrix</th>
<th>Adj List</th>
<th>Adj Set (Hash)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n+m)$</td>
<td>$\Theta(n+m)$</td>
</tr>
<tr>
<td>has_edge ($u$, $v$)</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$ expected</td>
</tr>
<tr>
<td>add_edge</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$ expected</td>
</tr>
<tr>
<td>for_each_out-neighbor</td>
<td>$O(n)$</td>
<td>$O(n)$ worst</td>
<td>$O(n)$ worst</td>
</tr>
<tr>
<td>for each edge</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n+m)$</td>
<td>$\Theta(n+m)$</td>
</tr>
</tbody>
</table>