Graph Algorithms

- **Topological Sort**
  - Produce a valid ordering of all nodes, given pair-wise 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)
Topological Sort

- **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

- **Refine Algorithm**
  - Use priority queue?
  - Complexity?

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

<table>
<thead>
<tr>
<th>Semester</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPS 6</td>
</tr>
<tr>
<td>1</td>
<td>CPS 100</td>
</tr>
<tr>
<td>2</td>
<td>CPS 104</td>
</tr>
<tr>
<td>3</td>
<td>CPS 108</td>
</tr>
<tr>
<td>4</td>
<td>CPS 110</td>
</tr>
<tr>
<td>5</td>
<td>CPS 130</td>
</tr>
<tr>
<td>6</td>
<td>AB</td>
</tr>
</tbody>
</table>
Shortest Path (Unweighted)

- Mark all vertices with infinity (⋆) exec. starting vertex with 0
- Place starting vertex in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each adjacent vertex marked with ⋆,
     - process it,
     - mark it with source distance + 1
     - place it on the queue.
Shortest Path (Unweighted)

- Mark all vertices with infinity (*)
- Mark starting vertex with 0
- Place starting vertex in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each adjacent vertex marked with *,
     - process it,
     - mark it with source distance + 1
     - place it on the queue.

How do we get actual “Path”?
Shortest Path (Weighted): Dijkstra

- Unmark all vertices and give infinite weight
- Set weight of starting vertex at 0 and place in priority queue
- Repeat until priority queue is empty:
  1. Remove a vertex from priority queue
     - Process and mark (weight now permanent)
  2. For each adjacent unmarked vertex
     - Set weight at lesser of current weight and (source weight + path weight)
     - (This may involve reducing previous weight setting)
     - Place in priority queue (if not there already)
Shortest Path (Weighted): Dijkstra

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

Edges:
- v0 → v1: 2
- v1 → v2: 3
- v2 → v3: 4
- v3 → v0: 1
- v3 → v1: 2
- v3 → v2: 5
- v3 → v4: 2
- v4 → v3: 6
- v4 → v5: 6
- v4 → v6: 1
Shortest Path (Weighted): Dijkstra

Mark all vertices with infinity (*).

Mark starting vertex with 0.

Place starting vertex in queue.

Repeat until queue is empty:

1. Remove a vertex from front of queue.
2. For each adjacent vertex marked with *, process it, mark it with source distance + 1, and place it on the queue.

How do we get actual “Path”?
Other Graph Algorithms

- Traveling Salesman
- Spanning Trees
- Paths with negative weights