4.4 Shortest Paths

- APIs
- shortest-paths properties
- Dijkstra’s algorithm
- edge-weighted DAGs
- negative weights
Acyclic edge-weighted digraphs (i.e., DAGs)

Q. Suppose that an edge-weighted digraph has no directed cycles, so it’s a DAG (directed acyclic graph). Is it easier to find shortest paths than in a general digraph?

A. Yes!
Acyclic shortest paths algorithm

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.

**an edge-weighted DAG (edge weights can be negative)**
Acyclic shortest paths algorithm

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.

<table>
<thead>
<tr>
<th>v</th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0</td>
<td>0→1</td>
</tr>
<tr>
<td>2</td>
<td>14.0</td>
<td>5→2</td>
</tr>
<tr>
<td>3</td>
<td>17.0</td>
<td>2→3</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0→4</td>
</tr>
<tr>
<td>5</td>
<td>13.0</td>
<td>4→5</td>
</tr>
<tr>
<td>6</td>
<td>25.0</td>
<td>2→6</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0→7</td>
</tr>
</tbody>
</table>
Proposition. Topological sort algorithm computes SPT in any edge-weighted DAG in time proportional to $E + V$.

Compare to $E \log V$ for Dijkstra’s algorithm--

Where $E > V$ (most cases), $O(E \log V) > O(E + V) = O(E)$, So this is faster. It takes advantage of the acyclic property.
Shortest paths in edge-weighted DAGs

```java
public class AcyclicSP {
    private DirectedEdge[] edgeTo;
    private double[] distTo;

    public AcyclicSP(EdgeWeightedDigraph G, int s) {
        edgeTo = new DirectedEdge[G.V()];
        distTo = new double[G.V()];

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        Topological topological = new Topological(G);
        for (int v : topological.order())
            for (DirectedEdge e : G.adj(v))
                relax(e);
    }
}
```
Content-aware resizing

**Seam carving.** [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.

http://www.youtube.com/watch?v=vIFCV2spKtg
Content-aware resizing

**Seam carving.** [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.

*In the wild.* Photoshop CS 5, Imagemagick, GIMP, ...
Longest paths in edge-weighted DAGs: application

**Parallel job scheduling.** Given a set of jobs with durations and precedence constraints, schedule the jobs (by finding a start time for each) so as to achieve the minimum completion time, while respecting the constraints.

<table>
<thead>
<tr>
<th>job</th>
<th>duration</th>
<th>must complete before</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41.0</td>
<td>1 7 9</td>
</tr>
<tr>
<td>1</td>
<td>51.0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>36.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>38.0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>21.0</td>
<td>3 8</td>
</tr>
<tr>
<td>7</td>
<td>32.0</td>
<td>3 8</td>
</tr>
<tr>
<td>8</td>
<td>32.0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>29.0</td>
<td>4 6</td>
</tr>
</tbody>
</table>

Parallel job scheduling solution
Critical path method

**CPM.** To solve a parallel job-scheduling problem, create edge-weighted DAG:

- **Source and sink vertices.**
- **Two vertices (begin and end) for a job.**
  - begin to end (weighted by duration)
  - source to begin (0 weight)
  - end to sink (0 weight)

One edge for each precedence constraint (0 weight).

<table>
<thead>
<tr>
<th>job</th>
<th>duration</th>
<th>must complete before</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41.0</td>
<td>1 7 9</td>
</tr>
<tr>
<td>1</td>
<td>51.0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>36.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>38.0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>21.0</td>
<td>3 8</td>
</tr>
<tr>
<td>7</td>
<td>32.0</td>
<td>3 8</td>
</tr>
<tr>
<td>8</td>
<td>32.0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>29.0</td>
<td>4 6</td>
</tr>
</tbody>
</table>
Critical path method

**CPM.** Use longest path from the source to schedule each job.
4.4 Shortest Paths

- APIs
- shortest_paths_properties
- Dijkstra's algorithm
- edge-weighted DAGs
- negative weights
Shortest paths with negative weights: failed attempts

**Dijkstra.** Doesn’t work with negative edge weights.

Dijkstra selects vertex 3 immediately after 0. But shortest path from 0 to 3 is 0→1→2→3.

**Re-weighting.** Add a constant to every edge weight doesn’t work.

Adding 8 to each edge weight changes the shortest path from 0→1→2→3 to 0→3.

**Conclusion.** Need a different algorithm.
**Proposition.** A SPT exists iff no negative cycles.

*assuming all vertices reachable from s*
Bellman-Ford algorithm

Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.

Repeat V times:
  - Relax each edge.

```java
for (int i = 0; i < G.V(); i++)
  for (int v = 0; v < G.V(); v++)
    for (DirectedEdge e : G.adj(v))
      relax(e);
```
### Single source shortest-paths implementation: cost summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction</th>
<th>Typical case</th>
<th>Worst case</th>
<th>Extra space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topological sort</td>
<td>no directed cycles</td>
<td>$E + V$</td>
<td>$E + V$</td>
<td>$V$</td>
</tr>
<tr>
<td><strong>Dijkstra</strong> (binary heap)</td>
<td>no negative weights</td>
<td>$E \log V$</td>
<td>$E \log V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>no negative cycles</td>
<td>$E \ V$</td>
<td>$E \ V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Bellman-Ford (queue-based)</td>
<td>no directed cycles</td>
<td>$E + V$</td>
<td>$E \ V$</td>
<td>$V$</td>
</tr>
</tbody>
</table>

**Remark 1.** Directed cycles make the problem harder.
**Remark 2.** Negative weights make the problem harder.
**Remark 3.** Negative cycles makes the problem intractable.
Shortest paths summary

Nonnegative weights.
• Arises in many application.
• Dijkstra's algorithm is nearly linear-time.

Acyclic edge-weighted digraphs.
• Arise in some applications.
• Topological sort algorithm is linear time.
• Edge weights can be negative.

Negative weights and negative cycles.
• Arise in some applications.
• If no negative cycles, can find shortest paths via Bellman-Ford
• If negative cycles, can find one via Bellman-Ford.

Shortest-paths is a broadly useful problem-solving model.