Last Time

- Strongly connected components
- Minimum spanning trees
- Lazy and Eager Prim Algorithms
- Kruskal’s Algorithm
But Wait, We Did Shortest Path with BFS...

• Yes, we did, but now our edges have weights!
  – A path that takes 4 edges could have a smaller total weight than a path that takes 2 edges.

• A **shortest path** from vertex $s$ to vertex $t$ in an edge-weighted digraph is a directed path from $s$ to $t$ with the property that no other such path has a lower weight.
Shortest Paths

edge-weighted digraph

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-&gt;5</td>
<td>0.35</td>
</tr>
<tr>
<td>5-&gt;4</td>
<td>0.35</td>
</tr>
<tr>
<td>4-&gt;7</td>
<td>0.37</td>
</tr>
<tr>
<td>5-&gt;7</td>
<td>0.28</td>
</tr>
<tr>
<td>7-&gt;5</td>
<td>0.28</td>
</tr>
<tr>
<td>5-&gt;1</td>
<td>0.32</td>
</tr>
<tr>
<td>0-&gt;4</td>
<td>0.38</td>
</tr>
<tr>
<td>0-&gt;2</td>
<td>0.26</td>
</tr>
<tr>
<td>7-&gt;3</td>
<td>0.39</td>
</tr>
<tr>
<td>1-&gt;3</td>
<td>0.29</td>
</tr>
<tr>
<td>2-&gt;7</td>
<td>0.34</td>
</tr>
<tr>
<td>6-&gt;2</td>
<td>0.40</td>
</tr>
<tr>
<td>3-&gt;6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-&gt;0</td>
<td>0.58</td>
</tr>
<tr>
<td>6-&gt;4</td>
<td>0.93</td>
</tr>
</tbody>
</table>

shortest path from 0 to 6

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-&gt;2</td>
<td>0.26</td>
</tr>
<tr>
<td>2-&gt;7</td>
<td>0.34</td>
</tr>
<tr>
<td>7-&gt;3</td>
<td>0.39</td>
</tr>
<tr>
<td>3-&gt;6</td>
<td>0.52</td>
</tr>
</tbody>
</table>
Shortest Path Assumptions

• Paths are directed. Undirected edges can be considered as a pair of directed edges. We must respect the direction shown by the edge.
• Weights are (again) not necessarily distances, nor are they necessarily costs (but they could be).
• Not all vertices need to be reachable. If no path exists from $s$ to $t$, then it follows that there is no shortest path.
• Negative weights make things tough, so we’ll assume all weights are positive or zero.
• Shortest paths are not necessarily unique.
Relaxation

- **Relaxation** is a process for determining the shortest path by a bit of induction.
  - In **edge relaxation**, we test whether the best known way from $s$ to $w$ is to go from $s$ to $v$, then take the edge from $v$ to $w$, and if so, update our data structures to indicate that this is the case.
  - In **vertex relaxation**, we relax all of the edges from a given vertex at the same time, keeping any of the new shortest paths found in our data structures.
Relaxation

black edges are in edgeTo[]

weight of v→w is 1.3

distTo[v]
distTo[w]

before

after

still ineligible

now ineligible

edgeTo[w]

no longer in SPT

04/14/2015

Shortest Paths
Dijkstra’s Algorithm

• Basic idea:
  – Build off of the idea behind Eager Prim.
  – Initialize distTo[s] (starting vertex) to 0, and distTo[everything_else] to positive infinity.
  – Relax and add to the tree a non-tree vertex with the lowest distTo[ ] value.
  – Repeat until all vertices are on the tree, or no non-tree vertex has a finite distTo[ ] value.
Dijkstra’s Algorithm

```java
private DirectedEdge[] edgeTo;
private double[] distTo;
private IndexMinPQ<Double> pq;

public DijkstraSP(EdgeWeightedDigraph G, int s) {
    edgeTo = new DirectedEdge[G.V()];
    distTo = new double[G.V()];
    pq = new IndexMinPQ<Double>(G.V());
    for (int v = 0; v < G.V(); v++) {
        distTo[v] = Double.POSITIVE_INFINITY;
    } //for
    distTo[s] = 0.0;
    pq.insert(s, 0.0);
    while !(pq.isEmpty()) {
        relax(G, pq.delMin());
    } //while
} //DijkstraSP
```
Dijkstra’s Algorithm

```java
void relax(EdgeWeightedDigraph G, int v) {
    for (DirectedEdge e: G.adj(v)) {
        int w = e.to();
        if (distTo[w] > distTo[v] + e.weight()) {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) {
                pq.changeKey(w, distTo[w]);
            } else {
                pq.insert(w, distTo[w]);
            } //if-else
        } //if
    } //for
} //DijkstraSP
```
Dijkstra’s Algorithm – Visual
Dijkstra’s Algorithm – Analysis

- **Theorem:** Dijkstra’s algorithm uses extra space proportional to $V$ and time proportional to $E \times \log(V)$ (in the worst case)
- **Proof:** Identical to the proof for Eager Prim.■
Can We Do Better?

• In some cases, yeah.
• What if we know that our digraph is acyclic?
  – Now we can handle negative weights. (why?)
  – We can now solve the problem in linear time.
  – Make use of our knowledge of Topological sort.
  – We can also solve the longest path problem! (how?)
AcyclicSP

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;
    } //for
    distTo[s] = 0.0;
    Topological top = new Topological(G);
    for (int v: top.order()) {
        relax(G, v);
    } //for
} //AcyclicSP
AcyclicSP – Visual

topological sort
5 1 3 6 4 7 0 2

thick black: on tree

red: add to tree

gray: ineligible

edgeTo[]
0
1 5->1
2
3
4 5->4
5
6
7 5->7

0 4->0
1 5->1
2 6->2
3 1->3
4 5->4
5
6 3->6
7 5->7

0 4->0
1 5->1
2 7->2
3 1->3
4 5->4
5
6 3->6
7 5->7

0 4->0
1 5->1
2 6->2
3 1->3
4 5->4
5
6 3->6
7 5->7

04/14/2015 Shortest Paths
AcyclicSP – Analysis

- **Theorem:** By relaxing vertices in topological order, we can solve the shortest path problem for edge-weighted DAGs in time proportional to $E + V$.

- **Proof:**
  - Every edge $v\rightarrow w$ is relaxed exactly once, when $v$ is relaxed, leaving $\text{distTo}[w] \leq \text{distTo}[v] + e\.weight()$.
  - This inequality holds until the algorithm completes, because $\text{distTo}[v]$ never updates (no edge pointing to $v$ will be relaxed after $v$ is relaxed).
  - After all vertices reachable from $s$ have been added to the tree, the shortest-path optimality condition holds.
  - The Topological sort pass takes $E + V$, and the second relaxation pass completes the job by relaxing each edge once: another $E + V$. ■
Let’s Make Thing Complicated

• What could happen if we introduce negative edges into a graph with a cycle?
  – (bad things)
  – We can only have a shortest path if there are no negative cycles in the graph.
Bellman-Ford Algorithm

Staring at the ceiling, she asked me what I was thinking about.

I should have made something up.

The Bellman-Ford algorithm makes terrible pillow talk.
Any Questions?