Single-Source Shortest Paths

• **Given:** A single source vertex in a weighted, directed graph.

• Want to compute a shortest path for each possible destination.
  – Similar to BFS.

• We will assume either
  – no negative-weight edges, or
  – no reachable negative-weight cycles.

• Algorithm will compute a shortest-path tree.
  – Similar to BFS tree.
Outline

• General Lemmas and Theorems.
• Bellman-Ford algorithm.
• DAG algorithm.
• Dijkstra’s algorithm.
General Results (Relaxation)

**Lemma 24.1:** Let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a SP from $v_1$ to $v_k$. Then, $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a SP from $v_i$ to $v_j$, where $1 \leq i \leq j \leq k$.

So, we have the **optimal-substructure property**.

Bellman-Ford’s algorithm uses **dynamic programming**.

Dijkstra’s algorithm uses the **greedy approach**.

Let $\delta(u, v) =$ weight of SP from $u$ to $v$.

**Corollary:** Let $p = $ SP from $s$ to $v$, where $p = s \rightarrow u \rightarrow v$. Then, $\delta(s, v) = \delta(s, u) + w(u, v)$.

**Lemma 24.10:** Let $s \in V$. For all edges $(u,v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u,v)$. 

p'
• Lemma 24.1 holds because one edge gives the shortest path, so the other edges must give sums that are at least as large.
Relaxation

Algorithms keep track of $d[v], \pi[v]$. **Initialized** as follows:

```plaintext
Initialize(G, s)
    for each $v \in V[G]$ do
        $d[v] := \infty$;
        $\pi[v] := \text{NIL}$
    end;
    $d[s] := 0$
```

These values are changed when an edge $(u, v)$ is **relaxed**:

```plaintext
Relax(u, v, w)
    if $d[v] > d[u] + w(u, v)$ then
        $d[v] := d[u] + w(u, v)$;
        $\pi[v] := u$
    end
```
Properties of Relaxation

- $d[v]$, if not $\infty$, is the length of some path from $s$ to $v$.
- $d[v]$ either stays the same or decreases with time.
- Therefore, if $d[v] = \delta(s, v)$ at any time, this holds thereafter.
- Note that $d[v] \geq \delta(s, v)$ always.
- After $i$ iterations of relaxing on all $(u,v)$, if the shortest path to $v$ has $i$ edges, then $d[v] = \delta(s, v)$.
Properties of Relaxation

Consider any algorithm in which \( d[v] \), and \( \pi[v] \) are first initialized by calling Initialize(\( G, s \)) [\( s \) is the source], and are only changed by calling Relax. We have:

**Lemma 24.11**: \((\forall \ v:: d[v] \geq \delta(s, v))\) is an invariant.

Implies \( d[v] \) doesn’t change once \( d[v] = \delta(s, v) \).

**Proof:**
Initialize(\( G, s \)) establishes invariant. If call to Relax(u, v, w) changes \( d[v] \), then it establishes:
\[
\begin{align*}
d[v] &= d[u] + w(u, v) \\
&\geq \delta(s, u) + w(u, v) \quad \text{, invariant holds before call.} \\
&\geq \delta(s, v) \quad \text{, by Lemma 24.10.}
\end{align*}
\]

**Corollary 24.12**: If there is no path from \( s \) to \( v \), then \( d[v] = \delta(s, v) = \infty \) is an invariant.
• For lemma 24.11, note that initialization makes the invariant true at the beginning.
More Properties

**Lemma 24.13:** Immediately after relaxing edge \((u, v)\) by calling Relax\((u, v, w)\), we have \(d[v] \leq d[u] + w(u, v)\).

**Lemma 24.14:** Let \(p = SP\) from \(s\) to \(v\), where \(p = s \xrightarrow{u} v\). If \(d[u] = \delta(s, u)\) holds at any time prior to calling Relax\((u, v, w)\), then \(d[v] = \delta(s, v)\) holds at all times after the call.

**Proof:**

After the call we have:
\[
\begin{align*}
d[v] & \leq d[u] + w(u, v) & \text{, by Lemma 24.13.} \\
& = \delta(s, u) + w(u, v) & \text{, } d[u] = \delta(s, u) \text{ holds.} \\
& = \delta(s, v) & \text{, by corollary to Lemma 24.1.}
\end{align*}
\]

By Lemma 24.11, \(d[v] \geq \delta(s, v)\), so \(d[v] = \delta(s, v)\).
- Lemma 24.13 follows simply from the structure of Relax.
- Lemma 24.14 shows that the shortest path will be found one vertex at a time, if not faster. Thus after a number of iterations of Relax equal to $V(G) - 1$, all shortest paths will be found.
• Bellman-Ford returns a compact representation of the set of shortest paths from $s$ to all other vertices in the graph reachable from $s$. This is contained in the predecessor subgraph.
Predecessor Subgraph

Lemma 24.16: Assume given graph $G$ has no negative-weight cycles reachable from $s$. Let $G_{\pi} = \text{predecessor subgraph}$. $G_{\pi}$ is always a tree with root $s$ (i.e., this property is an invariant).

Proof:
Two proof obligations:

(1) $G_{\pi}$ is acyclic.

(2) There exists a unique path from source $s$ to each vertex in $V_{\pi}$.

Proof of (1):
Suppose there exists a cycle $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$.
We have $\pi[v_i] = v_{i-1}$ for $i = 1, 2, \ldots, k$.

Assume relaxation of $(v_{k-1}, v_k)$ created the cycle.
We show cycle has a negative weight.

Note: Cycle must be reachable from $s$. 
Proof of (1) (Continued)

Before call to \text{Relax}(v_{k-1}, v_k, w):

\[ \pi[v_i] = v_{i-1} \text{ for } i = 1, \ldots, k-1. \]

Implies \( d[v_i] \) was last updated by “\( d[v_i] := d[v_{i-1}] + w(v_{i-1}, v_i) \)” for \( i = 1, \ldots, k-1. \) [Because \text{Relax} updates \( \pi \).]

Implies \( d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i) \) for \( i = 1, \ldots, k-1. \) [Lemma 24.13]

Because \( \pi[v_k] \) is changed by call, \( d[v_k] > d[v_{k-1}] + w(v_{k-1}, v_k). \) Thus,

\[
\sum_{i=1}^{k} d[v_i] > \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))
= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)
\]

Because \( \sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}], \sum_{i=1}^{k} w(v_{i-1}, v_i) < 0, \text{ i.e., neg.-weight cycle!} \)
Comment on Proof

• \(d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i)\) for \(i = 1, \ldots, k-1\) because when Relax\(v_{i-1}, v_i, w) was called, there was an equality, and \(d[v_{i-1}]\) may have gotten smaller by further calls to Relax.

• \(d[v_k] > d[v_{k-1}] + w(v_{k-1}, v_k)\) before the last call to Relax because that last call changed \(d[v_k]\).
Lemma 24.17

**Lemma 24.17:** Same conditions as before. Call Initialize & repeatedly call Relax until $d[v] = \delta(s, v)$ for all $v$ in $V$. Then, $G_\pi$ is a shortest-path tree rooted at $s$.

**Proof:**

**Key Proof Obligation:** For all $v$ in $V_\pi$, the unique simple path $p$ from $s$ to $v$ in $G_\pi$ (path exists by Lemma 24.16) is a shortest path from $s$ to $v$ in $G$.

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$.

We have $d[v_i] = \delta(s, v_i)$

$$d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i) \quad \text{(reasoning as before)}$$

Implies $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. 
Proof (Continued)

\[ w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \leq \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) \]
\[ = \delta(s, v_k) - \delta(s, v_0) = \delta(s, v_k) \]

So, equality holds and \( p \) is a shortest path.
• And note that this shortest path tree will be found after $V(G) - 1$ iterations of Relax.
Bellman-Ford Algorithm

Can have negative-weight edges. Will “detect” reachable negative-weight cycles.

```
Initialize(G, s);
for i := 1 to |V[G]| – 1 do
    for each (u, v) in E[G] do
        Relax(u, v, w)
    od
od;
for each (u, v) in E[G] do
    if d[v] > d[u] + w(u, v) then
        return false
    fi
od;
return true
```

Time Complexity is O(VE).
So if Bellman-Ford has not converged after $V(G) - 1$ iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.
Example
Example

The image shows a weighted graph with nodes labeled 0, 6, 7, ∞, u, v, z, x, y. The edges are labeled with weights: 6, 8, 7, 2, 9, -2, -3, -4, -5, ∞.
Example
Example
Example
Another Look

**Note:** This is essentially *dynamic programming*.

Let \( d(i, j) = \) cost of the shortest path from \( s \) to \( i \) that is at most \( j \) hops.

\[
d(i, j) = \begin{cases} 
0 & \text{if } i = s \land j = 0 \\
\infty & \text{if } i \neq s \land j = 0 \\
\min(\{d(k, j-1) + w(k, i) : i \in \text{Adj}(k)\} \cup \{d(i, j-1)\}) & \text{if } j > 0 
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>∞</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>-2</td>
<td></td>
</tr>
</tbody>
</table>
Lemma 24.2

**Lemma 24.2:** Assuming no negative-weight cycles reachable from s, $d[v] = \delta(s, v)$ holds upon termination for all vertices v reachable from s.

**Proof:**

Consider a SP $p$, where $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$.

Assume $k \leq |V| - 1$, otherwise $p$ has a cycle.

**Claim:** $d[v_i] = \delta(s, v_i)$ holds after the $i^{th}$ pass over edges.

Proof follows by induction on $i$.

By Lemma 24.11, once $d[v_i] = \delta(s, v_i)$ holds, it continues to hold.
Correctness

**Claim:** Algorithm returns the correct value.

(Part of Theorem 24.4. Other parts of the theorem follow easily from earlier results.)

**Case 1:** There is no reachable negative-weight cycle.

Upon termination, we have for all \((u, v)\):
\[
\begin{align*}
    d[v] &= \delta(s, v) & \text{, by lemma 24.2 (last slide) if } v \text{ is reachable;} \\
    &= \delta(s, v) = \infty \text{ otherwise.}
\end{align*}
\]
\[
\begin{align*}
    \leq \delta(s, u) + w(u, v) & \text{, by Lemma 24.10.} \\
    = d[u] + w(u, v)
\end{align*}
\]

So, algorithm returns **true**.
Case 2

**Case 2:** There exists a reachable negative-weight cycle 
\[ c = \langle v_0, v_1, \ldots, v_k \rangle, \text{ where } v_0 = v_k. \]

We have 
\[ \sum_{i = 1, \ldots, k} w(v_{i-1}, v_i) < 0. \]  
\[ (*) \]

Suppose algorithm returns true. Then, 
\[ d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i) \]
for 
\[ i = 1, \ldots, k. \] (because Relax didn’t change any \( d[v_i] \)). Thus,

\[ \sum_{i = 1, \ldots, k} d[v_i] \leq \sum_{i = 1, \ldots, k} d[v_{i-1}] + \sum_{i = 1, \ldots, k} w(v_{i-1}, v_i) \]

But, 
\[ \sum_{i = 1, \ldots, k} d[v_i] = \sum_{i = 1, \ldots, k} d[v_{i-1}] \]

Can show no \( d[v_i] \) is infinite. Hence, 
\[ 0 \leq \sum_{i = 1, \ldots, k} w(v_{i-1}, v_i). \]

Contradicts \( (*) \). Thus, algorithm returns false.
Shortest Paths in DAGs

Topologically sort vertices in G; Initialize(G, s);
for each u in V[G] (in order) do
  for each v in Adj[u] do
    Relax(u, v, w)
  od
od
Example
Example
Example

\[
\begin{array}{cccc}
\infty & 0 & 2 & 6 \\
5 & 2 & 7 & -1 & -2 \\
3 & & & 4 & 2 & 1 & -2 \\
\end{array}
\]
Example
Example
Example
Example
Dijkstra’s Algorithm

Assumes no negative-weight edges.
Maintains a set $S$ of vertices whose SP from $s$ has been determined.
Repeatedly selects $u$ in $V\setminus S$ with minimum SP estimate (greedy choice).
Store $V\setminus S$ in priority queue $Q$.

Initialize($G$, $s$);
$S := \emptyset$;
$Q := V[G]$;
while $Q \neq \emptyset$ do
  $u := \text{Extract-Min}(Q)$;
  $S := S \cup \{u\}$;
  for each $v \in \text{Adj}[u]$ do
    Relax($u$, $v$, $w$)
  od
od
Example
Example
Example
Example
Example
Example
## Correctness

**Theorem 24.6:** Upon termination, $d[u] = \delta(s, u)$ for all $u$ in $V$ (assuming non-negative weights).

**Proof:**

By Lemma 24.11, once $d[u] = \delta(s, u)$ holds, it continues to hold. **We prove:** For each $u$ in $V$, $d[u] = \delta(s, u)$ when $u$ is inserted in $S$.

Suppose not. Let $u$ be the first vertex such that $d[u] \neq \delta(s, u)$ when inserted in $S$.

Note that $d[s] = \delta(s, s) = 0$ when $s$ is inserted, so $u \neq s$.

$\Rightarrow S \neq \emptyset$ just before $u$ is inserted (in fact, $s \in S$).
Proof (Continued)

Note that there exists a path from \( s \) to \( u \), for otherwise \( d[u] = \delta(s, u) = \infty \) by Corollary 24.12.

\( \Rightarrow \) there exists a SP from \( s \) to \( u \). SP looks like this:
Proof (Continued)

**Claim:** \( d[y] = \delta(s, y) \) when \( u \) is inserted into \( S \).

We had \( d[x] = \delta(s, x) \) when \( x \) was inserted into \( S \).

Edge \((x, y)\) was relaxed at that time.

By Lemma 24.14, this implies the claim.

Now, we have: \( d[y] = \delta(s, y) \), by Claim.

\[ \leq \delta(s, u) \], nonnegative edge weights.

\[ \leq d[u] \], by Lemma 24.11.

Because \( u \) was added to \( S \) before \( y \), \( d[u] \leq d[y] \).

Thus, \( d[y] = \delta(s, y) = \delta(s, u) = d[u] \).

**Contradiction.**
Complexity

Running time is

\[ O(V^2) \] using linear array for priority queue.

\[ O((V + E) \lg V) \] using binary heap.

\[ O(V \lg V + E) \] using Fibonacci heap.

(See book.)