Shortest Paths

- **Input**: weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$.

- The **weight** of path $p = < v_0, v_1, \ldots, v_k >$ is the sum of the weights of its constituent edges:
  $$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) .$$

- The **shortest-path weight** from $u$ to $v$ is
  $$\delta(u, v) = \left\{ \begin{array}{ll}
  \min \{ w(p) \} & \text{if there is a path } p \text{ from } u \text{ to } v , \\
  \infty & \text{otherwise} .
  \end{array} \right.$$ 

- A **shortest path** from vertex $u$ to vertex $v$ is then defined as any path $p$ with weight $w(p) = \delta(u, v)$. 
Example
Solution
Shortest Paths

Shortest Path Variants

• Single Source-Single Sink
• Single Source (all destinations from a source s)
• All Pairs

Defs:

• Let \( \delta(v) \) be the real shortest path distance from \( s \) to \( v \)
• Let \( d(v) \) be a value computed by an algorithm

Edge Weights

• All non-negative
• Arbitrary

Note: Must have no negative cost cycles
Single Source Shortest Paths

Key Property: Subpaths of shortest paths are shortest paths  Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}$, let $p = < v_1, v_2, \ldots, v_k >$ be a shortest path from vertex $v_1$ to vertex $v_k$ and, for any $i$ and $j$ such that $1 \leq i \leq j \leq k$, let $p_{ij} = < v_i, v_{i+1}, \ldots, v_j >$ be the subpath of $p$ from vertex $v_i$ to vertex $v_j$. Then, $p_{ij}$ is a shortest path from $v_i$ to $v_j$.

Note: this is optimal substructure

Corollary 1  For all edges $(u, v) \in E$,
$$\delta(v) \leq \delta(u) + w(u, v)$$

Corollary 2  Shortest paths follow a tree of edges for which
$$\delta(v) = \delta(u) + w(u, v)$$

More precisely, any edge in a shortest path must satisfy
$$\delta(v) = \delta(u) + w(u, v)$$
Relaxation

Relax($u,v,w$)
1   if $d[v] > d[u] + w(u,v)$
2       then $d[v] \leftarrow d[u] + w(u,v)$
3       $\pi[v] \leftarrow u$ (keep track of actual path)

Lemma: Assume that we initialize all $d(v)$ to $\infty$, $d(s) = 0$ and execute a series of Relax operations. Then for all $v$, $d(v) \geq \delta(v)$.

Lemma: Let $P = e_1, \ldots, e_k$ be a shortest path from $s$ to $v$. After initialization, suppose that we relax the edges of $P$ in order (but not necessarily consecutively). Then $d(v) = \delta(v)$. 
Example

```
s  6  a
  6  3  b
  2  5  c
  5  5  d
  6  4  e
-8
  4
-1
```
Goal of an algorithm: Relax the edges in a shortest path in order (but not necessarily consecutively).
Goal of an algorithm: Relax the edges in a shortest path in order (but not necessarily consecutively).

Bellman-Ford\((G, w, s)\)

1. \text{Initialize-Single-Source}(G, s)
2. for \(i \leftarrow 1\) to \(|V[G]| - 1\)
3. do for each edge \((u, v) \in E[G]\)
4. do \text{Relax}(u, v, w)
5. for each edge \((u, v) \in E[G]\)
6. do if \(d[v] > d[u] + w(u, v)\)
7. then return \text{FALSE}
8. return \text{TRUE}

\text{Initialize – Single – Source}(G, s)

1. for each vertex \(v \in V[G]\)
2. do \(d[v] \leftarrow \infty\)
3. \(\pi[v] \leftarrow \text{NIL}\)
4. \(d[s] \leftarrow 0\)
Example
Correctness of Bellman Ford

- Every shortest path must be relaxed in order
- If there are negative weight cycles, the algorithm will return false

Running Time \( O(VE) \)
All edges non-negative

• Dijkstra’s algorithm, a greedy algorithm
• Similar in spirit to Prim’s algorithm
• Idea: Run a discrete event simulation of breadth-first-search. Figure out how to implement it efficiently
• Can relax edges out of each vertex exactly once.

\[Dijkstra(G, w, s)\]

1. \texttt{Initialize-Single-Source}(G, s)
2. \(S \leftarrow \emptyset\)
3. \(Q \leftarrow V[G]\)
4. while \(Q \neq \emptyset\)
5. do \(u \leftarrow \texttt{Extract-Min}(Q)\)
6. \(S \leftarrow S \cup \{u\}\)
7. for each vertex \(v \in Adj[u]\)
8. do \texttt{Relax}(u, v, w)
Example
Correctness of Dijkstra’s algorithm  Dijkstra’s algorithm, run on a weighted, directed graph $G = (V, E)$ with nonnegative weight function $w$ and source $s$, terminates with $d[u] = \delta(s, u)$ for all vertices $u \in V$.

- $E$ decrease keys and $V$ delete-min’s
- $O(E \log V)$ using a heap
- $O(E + V \log V)$ using a Fibonacci heap
**Shortest Path in a DAG**

Dag-Shortest-Paths\((G, w, s)\)

1. topologically sort the vertices of \(G\)
2. \textsc{Initialize-Single-Source}'\((G, s)\)
3. for each \(u\) taken in topological order
   4. do for each \(v \in \text{Adj}[u]\)
   5. do \textsc{Relax}(\(u, v, w\))
Example
Correctness and Running Time

**Correctness** If a weighted, directed graph $G = (V, E)$ has source vertex $s$ and no cycles, then at the termination of the **DAG-SHORTEST-PATHS** procedure, $d[v] = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph $G_\pi$ is a shortest-paths tree.

**Running Time**

- Topological sort is linear time
- Each edge is relaxed once
- No additional data structure overhead

$O(V + E)$ time.