### Shortest Paths in Graphs

**Bellman-Ford Algorithm**

Slides courtesy of Erik Demaine and Carola Wenk

---

**Negative-weight cycles**

**Recall:** If a graph \( G = (V, E) \) contains a negative-weight cycle, then some shortest paths may not exist.

**Example:**

\[
\begin{array}{c}
\text{u} \\
\text{< 0} \\
\text{v}
\end{array}
\]

**Bellman-Ford algorithm:** Finds all shortest-path lengths from a source \( s \in V \) to all \( v \in V \) or determines that a negative-weight cycle exists.

---

**Bellman-Ford and Undirected Graphs**

Bellman-Ford algorithm is designed for directed graphs.

If \( G \) is undirected, replace every edge \((u, v)\) with two directed edges \((u, v)\) and \((v, u)\), both with weight \( w(u, v) \)

---

**Bellman-Ford algorithm**

\[
\begin{align*}
d[s] &\leftarrow 0 \\
\text{for each } v \in V - \{s\} &\text{ do } d[v] \leftarrow \infty
\end{align*}
\]

**initialization**

\[
\begin{align*}
\text{for } i &\leftarrow 1 \text{ to } |V| - 1 \text{ do} \\
\text{for each edge } (u, v) &\in E \text{ do} \\
\text{if } d[v] > d[u] + w(u, v) &\text{ then } \\
\quad d[v] &\leftarrow d[u] + w(u, v) \\
\quad \pi[v] &\leftarrow u
\end{align*}
\]

**relaxation step**

for each edge \((u, v)\) \(\in E\)

\[
\text{if } d[v] > d[u] + w(u, v) \quad \text{then } \text{report that a negative-weight cycle exists}
\]

At the end, \( d[v] = \delta(s, v) \). Time = \( O(|V| |E|) \).

---

**Example of Bellman-Ford**

Order of edges: \((B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)\)

---

**Example of Bellman-Ford**

Order of edges: \((B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)\)
Example of Bellman-Ford

Order of edges: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

\[
\begin{array}{cccccc}
& A & B & C & D & E \\
0 & 0 & \infty & \infty & \infty & \infty \\
0 & -1 & \infty & \infty & \infty & \infty \\
0 & -1 & 4 & \infty & \infty & \infty \\
0 & -1 & 2 & \infty & 1 & 1 \\
\end{array}
\]
Correctness

**Theorem.** If \( G = (V, E) \) contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, \( d[v] = \delta(s, v) \) for all \( v \in V \).

**Proof.** Let \( v \in V \) be any vertex, and consider a shortest path \( p \) from \( s \) to \( v \) with the minimum number of edges.

\[
p: s \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v
\]

Since \( p \) is a shortest path, we have
\[
\delta(s, v) = \delta(s, v_{i-1}) + w(v_{i-1}, v) \quad \text{for every } i.
\]

Correctness (continued)

Let \( p \) be the shortest path from \( s \) to a vertex \( v \). Let's re-label the vertices along \( p \) so \( s=v_0, v_1, \ldots, v_k=v \).

Note that (a portion of) \( p \) is also the shortest path to each \( v_i \) since \( G \) contains no negative-weight cycles, \( p \) is simple. The shortest simple path has \(|V| - 1\) edges.

Detection of negative-weight cycles

**Corollary.** If a value \( d[v] \) fails to converge after \(|V| - 1 \) passes, there exists a negative-weight cycle in \( G \) reachable from \( s \).

**DAG shortest paths**

If the graph is a directed acyclic graph (DAG), we first topologically sort the vertices, (give \( l(v) \) to each vertex \( v \) such that \( (u, v) \) vertex such that \( l(u) < l(v) \) for every edge \( (u, v) \). (e.g. use Kuhn Algorithm).

To find shortest path from \( s \) to \( v \):
- Init as Bellman-Ford.
- Walk through the vertices in topological order. For every vertex \( v \) relaxing the edges in \( Adj[v] \)

\[
d[v] = \min(d[w] + \text{weight}(v, w))
\]

Thereby obtaining the shortest paths from \( s \) in a total of \( O(V + E) \) time. Homework: Prove correctness.

Homework:
- Thereby obtaining the shortest paths from \( s \) in a total of \( O(V + E) \) time. (prove correctness.)
Topological sorting of a graph.

- Given a DAG $G(V,E)$
- Output: A topological order. It should label each vertex $v$ in $V$ with a unique number $\text{lbl}[v]$ such that
- If $(u,v)$ is in $E$ then $\text{lbl}[u] < \text{lbl}[v]$.

InDegree – definition:

- For every vertex $v$, we store $\text{InDegree}(v, E)$, a number specifying how many edges "enter" $v$.

Kahn algorithm for finding a topological order in a DAG:

1. $L$ ← Empty list that will contain the sorted elements
2. $S$ ← Set of all nodes currently with no incoming edge in $E$.
3. $\text{cnt} = 0$; $\text{lbl}(v) =$ NULL for every vertex $v$
4. While $S$ is non-empty
   a. remove a node $u$ from $S$
   b. add $u$ to tail of $L$
   c. $\text{lbl}(u) =$ $\text{cnt}$; $\text{cnt}++$
   d. for each node $v$ with an edge $(u,v)$ in $E$ (each nbr of $u$)
      i. remove $(u,v)$ from $E$
      ii. $\text{InDegree}(v) = \text{InDegree}(v) - 1$
   e. if $v$ has no other incoming edges then insert $v$ into $S$
5. If $E$ is not empty, there are cycles.