Recall log rules: \( \log ab = \log a + \log b \)

Show how to use Dijkstra’s algorithm to compute a path \( s \to t \) in a graph \( G \) which minimizes the product of edge weights along the path. You may assume that all edge weights are greater than or equal to 1.
Show how to use Dijkstra’s algorithm to compute a path $s \rightarrow t$ in a graph $G$ which minimizes the product of edge weights along the path. You may assume that all edge weights are greater than or equal to 1.

- Construct a graph $G' = (V, E)$ where the weight of each edge $w'(e)$ in $G'$ is $\log w(e)$ and run Dijkstra’s on $G'$
- The multiplicative cost of a path $(v_0, v_1, ..., v_t)$ in $G$ is $\prod_{i=1}^{t} w(v_{i-1}, v_i)$
- Since all edge weights are positive, minimizing this quantity is equivalent to minimizing $\log \prod_{i=1}^{t} w(v_{i-1}, v_i) = \sum_{i=1}^{t} \log w(v_{i-1}, v_i)$
- This coincides with the minimization objective of Dijkstra on $G'$ (all weights in $G'$ are non-negative since edge weights in $G$ are $\geq 1$)
Today’s Keywords

Graphs
Shortest paths algorithms
Bellman-Ford
Floyd-Warshall

**CLRS Readings:** Chapter 22, 23, 24
Homework

HW7 due Thursday, November 14, 11pm
  • Graph algorithms
  • Written (use LaTeX!) – Submit both zip and pdf (two separate attachments)!

HW10B due Thursday, November 14, 11pm
  • No late submissions allowed (no exceptions)

HW8 out Thursday, November 14, due Thursday, November 21, 11pm
  • Programming assignment (Python or Java)
  • Graph algorithms
# Currency Exchanges and Arbitrage

Conversion rates starting from USD

<table>
<thead>
<tr>
<th>Currency code</th>
<th>Currency name</th>
<th>Units per USD</th>
<th>USD per Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>US Dollar</td>
<td>1.0000000000</td>
<td>1.0000000000</td>
</tr>
<tr>
<td>EUR</td>
<td>Euro</td>
<td>0.8783121137</td>
<td>1.1385474303</td>
</tr>
<tr>
<td>GBP</td>
<td>British Pound</td>
<td>0.6956087704</td>
<td>1.4375896950</td>
</tr>
<tr>
<td>INR</td>
<td>Indian Rupee</td>
<td>66.1909310706</td>
<td>0.0151078098</td>
</tr>
<tr>
<td>AUD</td>
<td>Australian Dollar</td>
<td>1.3050318080</td>
<td>0.7662648480</td>
</tr>
<tr>
<td>CAD</td>
<td>Canadian Dollar</td>
<td>1.2997506294</td>
<td>0.7693783541</td>
</tr>
<tr>
<td>SGD</td>
<td>Singapore Dollar</td>
<td>1.3478961522</td>
<td>0.7418969172</td>
</tr>
<tr>
<td>CHF</td>
<td>Swiss Franc</td>
<td>0.9590451582</td>
<td>1.0427037678</td>
</tr>
<tr>
<td>MYR</td>
<td>Malaysian Ringgit</td>
<td>3.8700000000</td>
<td>0.2583979328</td>
</tr>
<tr>
<td>JPY</td>
<td>Japanese Yen</td>
<td>112.5375383115</td>
<td>0.0088859239</td>
</tr>
<tr>
<td>CNY</td>
<td>Chinese Yuan Renminbi</td>
<td>6.4492409303</td>
<td>0.1550570076</td>
</tr>
<tr>
<td>NZD</td>
<td>New Zealand Dollar</td>
<td>1.4480018872</td>
<td>0.6906068347</td>
</tr>
<tr>
<td>THB</td>
<td>Thai Baht</td>
<td>35.1005319022</td>
<td>0.0284895968</td>
</tr>
<tr>
<td>HUF</td>
<td>Hungarian Forint</td>
<td>275.7012427385</td>
<td>0.0036271146</td>
</tr>
<tr>
<td>AED</td>
<td>Emirati Dirham</td>
<td>3.6730000000</td>
<td>0.2722570106</td>
</tr>
</tbody>
</table>

1 Dollar = 0.8783121137 Euro

1 Dollar = 3.87 Ringgit
Currency Exchanges and Arbitrage

1 Euro = 4.1823100458 Dirham
1 Dirham = 1.0548325619 Ringgit
1 Dollar = $0.8783121137 * 4.1823100458 * 1.0548325619 Ringgit
= 3.87479406049 Ringgit
= 1.00123877526 Dollar

But what we go from USD → EUR → MYR?

1 Dollar = 0.8783121137 Euro
1 Euro = 4.1823100458 Dirham
1 Dirham = 1.0548325619 Ringgit
1 Dollar = 0.8783121137 * 4.1823100458 * 1.0548325619 Ringgit
= 3.87479406049 Ringgit
= 1.00123877526 Dollar

Arbitrage opportunity: Profit by exploiting uneven exchange rates for the same asset (e.g., currencies, stocks, bonds, etc.)
Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates.

Product of edge weights along a path from $s \rightarrow t$ gives amount of currency $t$ that can be exchanged for 1 unit of currency $s$.

**Best currency exchange:**

$$\max_p \prod_{e \in p} w(e)$$
Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates.

Product of edge weights along a path from $s \to t$ gives the amount of currency $t$ that can be exchanged for 1 unit of currency $s$.

**Best currency exchange:**

$$\max_p \prod_{e \in p} w(e)$$

**Equivalently:**

$$\min_p \prod_{e \in p} \frac{1}{w(e)}$$
Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates.

Product of edge weights along a path from $s \to t$ gives amount of currency $t$ that can be exchanged for 1 unit of currency $s$.

**Best currency exchange:**
\[
\max_p \prod_{e \in p} w(e)
\]

**Equivalently:**
\[
\min_p \sum_{e \in p} -\log w(e)
\]
Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates.

Product of edge weights along a path from $s \rightarrow t$ gives amount of currency $t$ that can be exchanged for 1 unit of currency $s$.

**Best currency exchange:**

$$\max_D \prod_{e \in p} w(e)$$

**Equivalently:**

$$\min_D \sum_{e \in p} - \log w(e)$$

Standard shortest path problem!
Problem with Negative Edges

If a graph has a negative-weight cycle, then there does not exist a shortest path from $s \to t$ (whenever the negative-weight cycle is reachable from $s$).

**Proof:** Suppose there a negative-weight cycle $(v \to v)$ of cost $k < 0$. Consider any path of the form $s \to v \to u \to t$. We can decrease the cost of this path by $k$ by replacing $s \to v \to u$ with $s \to (v \to v) \to u$. This decreases the weight of the path from $s \to t$ by $k$. This can be repeated to make the weight of the path arbitrarily negative.
If a graph has a negative-weight cycle, then there does not exist a shortest path from $s \rightarrow t$ (whenever the negative-weight cycle is reachable from $s$).

**Important Note:** Shortest path is still well-defined if the graph has negative-weight edges, as long as it does not have a negative-weight cycle.
Recall: Dijkstra’s algorithm does not work if there are edges of negative weight.

Dijkstra’s algorithm is greedy: it constructs a shortest-path tree by always choosing the current closest node.
Recall: Dijkstra’s algorithm does not work if there are edges of negative weight.

Dijkstra’s algorithm is greedy: it constructs a shortest-path tree by always choosing the current closest node.
Recall: Dijkstra’s algorithm does not work if there are edges of negative weight.

Dijkstra’s algorithm is greedy: it constructs a shortest-path tree by always choosing the current closest node.

Problem: Dijkstra assumes that it has now found the shortest path to node B.

- When weights are positive, then every other path must have greater weight because they require first taking a path that is longer than the current distance from A → B (e.g., A → C).
- But if edge weights can be negative, the weight of later edges (e.g., C → B) can offset the cost of the initial longer path—hence, the greedy heuristic is suboptimal.
Recall: Dijkstra’s algorithm does not work if there are edges of negative weight.

Dijkstra’s algorithm is greedy: it constructs a shortest-path tree by always choosing the current closest node.
Recall the structure of the “shortest-path” tree from the previous lecture:

Shortest paths from a source has **optimal substructure**
- Greedy choice (choose the “closest” node to the source) is suboptimal with negative-weight edges
- **Idea:** use dynamic programming and consider *all* possible subproblems

Every subpath of a shortest path is itself a shortest path (optimal substructure)

**Observe:** shortest paths from a source forms a tree, but **not** a minimum spanning tree
When greedy does not work... try dynamic programming!

Short($i, v$) = weight of the shortest path from $s$ to $v$ using at most $i$ edges

A path of $i - 1$ edges from $s$ to some node $x$, then edge $(x, v)$

Two possibilities: OR

A path from $s$ to $v$ of at most $i - 1$ edges

$$\begin{align*}
\text{Short}(i, v) &= \min \left\{ \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right), \right.
\text{Short}(i - 1, v) \}
\end{align*}$$

Maximum value of $i$?
Claim: In every graph $G = (V, E)$ that does not contain a negative-weight cycle, there exists a shortest path between any two connected nodes with at most $|V| - 1$ edges.

Proof: Follows by Pigeonhole principle:

- If there is a shortest path with more than $|V| - 1$ edges, at least one node appears twice in the path, which means there is a cycle.
- Since there are no negative-weight cycles in $G$, the weight of the cycle is $\geq 0$.
- Thus, removing the cycle will yield a path of equal or smaller weight.
- Each cycle can be removed in this manner to obtain a path satisfying the desired property.
Bellman-Ford Shortest Path Algorithm

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

Short\( (i, v) = \min \left\{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)), \text{Short}(i - 1, v) \right\} \)

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Bellman-Ford Shortest Path Algorithm

Short($i$, $v$) = weight of the shortest path from $s$ to $v$ using at most $i$ edges

$$Short(i, v) = \min \begin{cases} \min_{x \in V} (Short(i - 1, x) + w(x, v)) \\ Short(i - 1, v) \end{cases}$$

Base case:
- $Short(0, s) = 0$
- $Short(0, v) = \infty$ if $v \neq s$

Suppose source node is $E$
Bellman-Ford Shortest Path Algorithm

\[ \text{Short}(i, v) = \begin{cases} \text{weight of the shortest path from } s \text{ to } v \\ \text{using at most } i \text{ edges} \end{cases} \]

\[ \text{Short}(i, v) = \min \left\{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) , \text{Short}(i - 1, v) \right\} \]

Base case:

- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Bellman-Ford Shortest Path Algorithm

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

\[
\text{Short}(i, v) = \min\left\{ \begin{array}{l}
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\
\text{Short}(i - 1, v)
\end{array} \right\}
\]

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Bellman-Ford Shortest Path Algorithm

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[ \text{Short}(i, v) = \min \begin{cases} \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right) \\ \text{Short}(i - 1, v) \end{cases} \]

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is \(E\)
Bellman-Ford Shortest Path Algorithm

Short\( (i, \nu) \) = weight of the shortest path from \( s \) to \( \nu \) using at most \( i \) edges

Short\( (i, \nu) = \min \left\{ \begin{array}{ll}
\min_{x \in V} & (\text{Short} (i-1, x) + w(x, \nu)) \\
\text{Short} (i-1, \nu)
\end{array} \right. \)

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, \nu) = \infty \) if \( \nu \neq s \)

Suppose source node is \( E \)
Bellman-Ford Shortest Path Algorithm

\[
\text{Short}(i, v) = \begin{cases} \text{weight of the shortest path from } s \text{ to } v \\
\text{using at most } i \text{ edges} \\
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\
\text{Short}(i - 1, v) \end{cases}
\]

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Bellman-Ford Shortest Path Algorithm

Short(i, v) = weight of the shortest path from s to v using at most i edges

Short(i, v) = \min_{x \in V} \left\{ \min(Short(i - 1, x) + w(x, v)) \right\}
\quad \text{Short}(i - 1, v)

Base case:
- Short(0, s) = 0
- Short(0, v) = \infty if v ≠ s

Suppose source node is E
Bellman-Ford Shortest Path Algorithm

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[
\text{Short}(i, v) = \min \left\{ \begin{array}{c}
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\
\text{Short}(i - 1, v)
\end{array} \right\}
\]

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is \(E\)
# Bellman-Ford Shortest Path Algorithm

- **Short**\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

  \[
  \text{Short}(i, v) = \begin{cases} 
  \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right) \\
  \text{Short}(i - 1, v)
  \end{cases}
  \]

**Base case:**

- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is \(E\)
Bellman-Ford Shortest Path Algorithm

\[
\text{Short}(i, v) = \begin{cases} 
\text{weight of the shortest path from } s \text{ to } v \\
\text{using at most } i \text{ edges}
\end{cases}
\]

\[
\text{Short}(i, v) = \min \left\{ \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right), \text{Short}(i - 1, v) \right\}
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( v = \text{A} )</th>
<th>( \text{B} )</th>
<th>( \text{C} )</th>
<th>( \text{D} )</th>
<th>( \text{E} )</th>
<th>( \text{F} )</th>
<th>( \text{G} )</th>
<th>( \text{H} )</th>
<th>( \text{I} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>( \infty )</td>
<td>8</td>
<td>( \infty )</td>
<td>7</td>
<td>0</td>
<td>( \infty )</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Bellman-Ford Shortest Path Algorithm

Short($i$, $v$) = weight of the shortest path from $s$ to $v$ using at most $i$ edges

Short($i$, $v$) = \[ \min_{x \in V} \left( \min(\text{Short}($i - 1$, $x$) + w(x, v)) \right) \]

\[ \text{Short}($i - 1$, $v$) \]

\[ \text{Base case:} \]
- Short(0, $s$) = 0
- Short(0, $v$) = $\infty$ if $v \neq s$

Suppose source node is $E$
Bellman-Ford Shortest Path Algorithm

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[
\text{Short}(i, v) = \min \left\{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)), \text{Short}(i - 1, v) \right\}
\]

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty \) if \(v \neq s\)

Suppose source node is \(E\)
Bellman-Ford Shortest Path Algorithm

\[
\text{Short}(i, v) = \begin{cases} 
\text{weight of the shortest path from } s \text{ to } v \\
\text{using at most } i \text{ edges}
\end{cases}
\]

\[
\text{Short}(i, v) = \min \left\{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \right\}
\]

\[
\text{Short}(i, v) = \begin{cases} 
\text{Short}(i - 1, v)
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>∞</td>
<td>8</td>
<td>∞</td>
<td>7</td>
<td>0</td>
<td>∞</td>
<td>5</td>
<td>5</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty \text{ if } v \neq s\)

Suppose source node is E
Bellman-Ford Shortest Path Algorithm

\[ \text{Short}(i, v) = \begin{cases} \text{weight of the shortest path from } s \text{ to } v \\ \text{using at most } i \text{ edges} \end{cases} \]

\[ \text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\ \text{Short}(i - 1, v) \end{cases} \]

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)

<table>
<thead>
<tr>
<th>( v )</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>( \infty )</td>
<td>8</td>
<td>( \infty )</td>
<td>7</td>
<td>0</td>
<td>( \infty )</td>
<td>5</td>
<td>5</td>
<td>( \infty )</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>( \infty )</td>
<td>8</td>
<td>( \infty )</td>
<td>4</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>5</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>6</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>7</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>8</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>
Bellman-Ford Shortest Path Algorithm

\[ \text{Short}(i, v) = \text{weight of the shortest path from } s \text{ to } v \text{ using at most } i \text{ edges} \]

\[ \text{Short}(i, v) = \min_{x \in V} \left\{ \min(\text{Short}(i - 1, x) + w(x, v)) \right\} \]

\[ \text{Short}(i, v) = \min(\text{Short}(i - 1, v)) \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>∞</td>
<td>8</td>
<td>∞</td>
<td>7</td>
<td>0</td>
<td>∞</td>
<td>5</td>
<td>5</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base case:
- \[ \text{Short}(0, s) = 0 \]
- \[ \text{Short}(0, v) = \infty \text{ if } v \neq s \]

Suppose source node is E
Bellman-Ford Shortest Path Algorithm

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

\[
\text{Short}(i, v) = \min \begin{cases} 
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\
\text{Short}(i - 1, v)
\end{cases}
\]

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)

Backtrack to reconstruct shortest path
Detecting Negative-Weight Cycles

\[ \text{Short}(i, v) = \begin{cases} \text{weight of the shortest path from } s \text{ to } v \\ \text{using at most } i \text{ edges} \end{cases} \]

\[ \text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\ \text{Short}(i - 1, v) \end{cases} \]

\[
\begin{array}{cccccccccc}
\hline
v & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
A & & & & & & & & & \\
B & & & & & & & & & \\
C & & & & & & & & & \\
D & & & & & & & & & \\
E & & & & & & & & & \\
F & & & & & & & & & \\
G & & & & & & & & & \\
H & & & & & & & & & \\
I & & & & & & & & & \\
\hline
\end{array}
\]

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Detecting Negative-Weight Cycles

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

\[
\begin{align*}
\text{Short}(i, v) &= \min\left\{ \min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \right. \\
&\quad \left. \quad \text{Short}(i-1, v) \right\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( v = )</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Detecting Negative-Weight Cycles

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

\[
\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\ \text{Short}(i - 1, v) \end{cases}
\]

Base case:
- \( \text{Short}(0, s) = 0 \)
- \( \text{Short}(0, v) = \infty \) if \( v \neq s \)

Suppose source node is \( E \)
Detecting Negative-Weight Cycles

Short($i, v$) = weight of the shortest path from $s$ to $v$ using at most $i$ edges

Short($i, v$) = \[
\min \{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \}
\]
\[
\text{Short}(i - 1, v)
\]

\[\begin{array}{cccccccccc}
\text{v} & A & B & C & D & E & F & G & H & I \\
\hline
0 & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\
1 & \infty & 8 & \infty & 7 & 0 & \infty & 5 & 5 & \infty \\
2 & 18 & 8 & 4 & 7 & 0 & 2 & 5 & 5 & 7 \\
3 & & & & & & & & & \\
4 & & & & & & & & & \\
5 & & & & & & & & & \\
6 & & & & & & & & & \\
7 & & & & & & & & & \\
8 & & & & & & & & & \\
\end{array}\]

Base case:
- Short(0, $s$) = 0
- Short(0, $v$) = $\infty$ if $v \neq s$

Suppose source node is $E$
Detecting Negative-Weight Cycles

Short(i, v) = weight of the shortest path from s to v using at most i edges

\[
\text{Short}(i, v) = \min \left\{ \begin{array}{l}
\min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right) \\
\text{Short}(i - 1, v)
\end{array} \right. 
\]

Base case:
- Short(0, s) = 0
- Short(0, v) = \infty if v \neq s

Suppose source node is E
Detecting Negative-Weight Cycles

\[
\text{Short}(i, v) = \text{weight of the shortest path from } s \text{ to } v \text{ using at most } i \text{ edges}
\]

\[
\text{Short}(i, v) = \min \begin{cases} 
\min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\
\text{Short}(i-1, v)
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>∞</td>
<td>8</td>
<td>∞</td>
<td>7</td>
<td>0</td>
<td>∞</td>
<td>5</td>
<td>5</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-9</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is E
Detecting Negative-Weight Cycles

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[
\text{Short}(i, v) = \begin{cases} 
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) & \text{if } v \neq s \\
\text{Short}(i - 1, v) & \text{if } v = s 
\end{cases}
\]

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is \(E\)
Detecting Negative-Weight Cycles

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[
\text{Short}(i, v) = \min \begin{cases} 
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\
\text{Short}(i - 1, v)
\end{cases}
\]

<table>
<thead>
<tr>
<th>(i)</th>
<th>(v = A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
<th>(E)</th>
<th>(F)</th>
<th>(G)</th>
<th>(H)</th>
<th>(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>0</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>1</td>
<td>(\infty)</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>(\infty)</td>
<td>5</td>
<td>5</td>
<td>(\infty)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-9</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-9</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>-9</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is \(E\)
Detecting Negative-Weight Cycles

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

Short\((i, v)\) = min \(\{\min_{x \in V}(\text{Short}(i - 1, x) + w(x, v))\) \(\text{Short}(i - 1, v)\)\)

<table>
<thead>
<tr>
<th>(v)</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>∞</td>
<td>8</td>
<td>∞</td>
<td>7</td>
<td>0</td>
<td>∞</td>
<td>5</td>
<td>5</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-9</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-9</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>-9</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>-10</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Base case:
- \(\text{Short}(0, s) = 0\)
- \(\text{Short}(0, v) = \infty\) if \(v \neq s\)

Suppose source node is \(E\)
Detecting Negative-Weight Cycles

Short(i, v) = weight of the shortest path from s to v using at most i edges

\[
\text{Short}(i, v) = \min \left\{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)), \text{Short}(i - 1, v) \right\}
\]

Base case:
- Short(0, s) = 0
- Short(0, v) = \infty if v \neq s

Suppose source node is E
Detecting Negative-Weight Cycles

**Observation:** After $|V| - 1$ iterations, if the lengths of the shortest paths has not converged (e.g., shortest path change after one more iteration of Bellman-Ford), there must exist a negative-weight cycle (since without negative-weight cycles, the shortest path requires at most $|V| - 1$ hops).

**Base case:**
- Short$(0,s) = 0$
- Short$(0,v) = \infty$ if $v \neq s$

Suppose source node is E.
allocate short[n][n]
initialize short[0][v] = \infty for each v
initialize short[0][s] = 0
for i = 1, ..., n - 1:
    for each e = (x, y) in E:
        short[i][y] = \min(
            short[i-1][x] + w[x][y],
            short[i-1][y]
        )

\mid V \mid = n
allocate short[n][n]
initialize short[0][v] = ∞ for each v
initialize short[0][s] = 0
for i = 1,...,n - 1:
    for each e = (x, y) in E:
        short[i][y] = min(
            short[i-1][x] + w[x][y],
            short[i-1][y]
        )

$|V| = n$

Running time (naïve): $O(|V|^2 + |E||V|)$
Bellman-Ford Run Time

allocate short[n][n]
initialize short[0][v] = \infty for each v
initialize short[0][s] = 0
for i = 1,\ldots,n - 1:
    for each e = (x, y) in E:
        short[i][y] = \min(
            short[i-1][x] + w[x][y],
            short[i-1][y]
        )

\textbf{Observation:} update for row i only depends on update for row i − 1
\textbf{Optimization:} only need to store two rows of short (previous row and current row)
\textbf{Overall running time of Bellman-Ford:} \mathcal{O}(|E||V|)
Both algorithms solve the single-source shortest path (SSSP) problem
Both algorithms handle directed and undirected graphs

**Dijkstra:**
- **Greedy** algorithm that always adds the node that is “closest” to the nodes that have been considered so far
- Only works for graphs with non-negative weights
- Updates require keeping track of shortest path to all nodes in the graph (changing graph weight essentially requires re-running the algorithm)
- **Running time:** $O(|E| \log |V|)$

**Bellman-Ford:**
- **Dynamic programming** algorithm that updates the costs of all paths based on the current shortest distance to all nodes in the graph
- Handles graphs with negative weights, can also be used to detect negative-weight cycles
- Updates can be distributed (each node only needs to know shortest path from/to each of its neighbors) – used in old routing protocols (e.g., the Routing Information Protocol)
- **Running time:** $O(|E||V|)$
Bellman-Ford in Dynamic Graphs (Update)

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

\[
\text{Short}(i, v) = \min \left\{ \begin{array}{l}
\min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right) \\
\text{Short}(i - 1, v)
\end{array} \right\}
\]

\[
\text{Short}(i, v) = \begin{array}{cccccccc}
0 & 18 & 0 & 0 & 0 & 0 & 0 & 0 \\
18 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 4 & 0 & 0 & 0 & 0 & 0 \\
0 & 8 & 4 & 0 & 0 & 0 & 0 & 0
\end{array}
\]
Bellman-Ford in Dynamic Graphs (Update)

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[
\text{Short}(i, v) = \min \begin{cases} 
\min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)) \\
\text{Short}(i - 1, v)
\end{cases}
\]

<table>
<thead>
<tr>
<th>(i)</th>
<th>(v) = A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>0</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>1</td>
<td>(\infty)</td>
<td>5</td>
<td>(\infty)</td>
<td>7</td>
<td>0</td>
<td>(\infty)</td>
<td>5</td>
<td>5</td>
<td>(\infty)</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Bellman-Ford in Dynamic Graphs (Update)

Short\( (i, v) \) = weight of the shortest path from \( s \) to \( v \) using at most \( i \) edges

\[
\text{Short}(i, v) = \min \left\{ \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right), \text{Short}(i - 1, v) \right\}
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( v = 0 )</th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( D )</th>
<th>( E )</th>
<th>( F )</th>
<th>( G )</th>
<th>( H )</th>
<th>( I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( \infty )</td>
<td>5</td>
<td>( \infty )</td>
<td>7</td>
<td>0</td>
<td>( \infty )</td>
<td>5</td>
<td>5</td>
<td>( \infty )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-8</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Bellman-Ford in Dynamic Graphs (Update)

Short\((i, v)\) = weight of the shortest path from \(s\) to \(v\) using at most \(i\) edges

\[
\text{Short}(i, v) = \min \left\{ \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right), \text{Short}(i - 1, v) \right\}
\]

<table>
<thead>
<tr>
<th>(i)</th>
<th>(v = A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
<th>(E)</th>
<th>(F)</th>
<th>(G)</th>
<th>(H)</th>
<th>(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>0</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>1</td>
<td>(\infty)</td>
<td>5</td>
<td>(\infty)</td>
<td>7</td>
<td>0</td>
<td>(\infty)</td>
<td>5</td>
<td>5</td>
<td>(\infty)</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Bellman-Ford in Dynamic Graphs (Update)

Short($i, v$) = weight of the shortest path from $s$ to $v$
using at most $i$ edges

Short($i, v$) = \[ \min \left\{ \min_{x \in V} \left( \text{Short}(i - 1, x) + w(x, v) \right), \text{Short}(i - 1, v) \right\} \]

<table>
<thead>
<tr>
<th>$i$</th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( D )</th>
<th>( E )</th>
<th>( F )</th>
<th>( G )</th>
<th>( H )</th>
<th>( I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>( \infty )</td>
<td>5</td>
<td>( \infty )</td>
<td>7</td>
<td>0</td>
<td>( \infty )</td>
<td>5</td>
<td>5</td>
<td>( \infty )</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Bellman-Ford in Dynamic Graphs (Update)

\[ \text{Short}(i, v) = \text{weight of the shortest path from } s \text{ to } v \text{ using at most } i \text{ edges} \]

\[ \text{Short}(i, v) = \min \left\{ \min_{x \in V} (\text{Short}(i - 1, x) + w(x, v)), \text{Short}(i - 1, v) \right\} \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>$\infty$</td>
<td>5</td>
<td>$\infty$</td>
<td>7</td>
<td>0</td>
<td>$\infty$</td>
<td>5</td>
<td>5</td>
<td>$\infty$</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>-11</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Recomputing shortest paths only requires local updates (only need to update paths that emanate from a node whose shortest path has changed)
Thus far: single-source shortest path algorithms (Dijkstra, Bellman-Ford)

All-pairs shortest-paths: find shortest path between every pair of nodes
All-Pairs Shortest Path

**Naïvely:** Run single-source shortest paths algorithm for each node $s$ (to compute shortest path from $s$ to every other node in the graph)

- If edge weights are all non-negative, can use Dijkstra (running time $O(|V||E| \log |V|)$)
- If edge weights can be negative, can use Bellman-Ford (running time $O(|V|^2|E|)$)

When $|E| = \Omega(|V|^2)$, both of these algorithms are $O(|V|^3 \log |V|)$ or $O(|V|^4)$

**Can we do better?**
Floyd-Warshall All-Pairs Shortest Paths

Finds all-pairs shortest paths in $\Theta(|V|^3)$ using dynamic programming.

Also works if graph has negative-weight edges.

**Same observation as before:** Every subpath of a shortest path is itself a shortest path (optimal substructure)
- Namely if shortest path from $i$ to $j$ goes through $k$, then the $i \rightarrow j$ and $j \rightarrow k$ subpaths must themselves be a shortest path.

\[
\text{Short}(i, j, k) = \begin{cases} 
\text{weight of shortest path from } i \rightarrow j \text{ using nodes } 1, \ldots, k \text{ as intermediate hops} 
\end{cases}
\]

Two possibilities for node $k$:
- **Shortest path from $i$ to $j$ includes $k$**
  \[
  \text{Short}(i, k, k - 1) + \text{Short}(k, j, k - 1)
  \]
- **Shortest path from $i$ to $j$ excludes $k$**
  \[
  \text{Short}(i, j, k - 1)
  \]
Floyd-Warshall All-Pairs Shortest Paths

Finds all-pairs shortest paths in $\Theta(|V|^3)$ using **dynamic programming**

Also works if graph has negative-weight edges

**Same observation as before:** Every subpath of a shortest path is itself a shortest path (optimal substructure)

- Namely if shortest path from $i$ to $j$ goes through $k$, then the $i \rightarrow k$ and $k \rightarrow j$ subpaths must themselves be a shortest path

$$
\text{Short}(i, j, k) = \text{weight of shortest path from } i \rightarrow j \text{ using nodes } 1, \ldots, k \text{ as intermediate hops}
$$

$$
\text{Short}(i, j, k) = \min \left\{ \text{Short}(i, k, k - 1) + \text{Short}(k, j, k - 1), \text{Short}(i, j, k - 1) \right\}
$$
allocate short[n][n][n] (initialized to $\infty$)

for (i, j) in E:
    short[i][j][0] = w[i][j]

for i = 1,...,n:
    short[i][i][0] = 0

for k = 1,...,n:
    for i = 1,...,n:
        for j = 1,...,n:
            short[i][j][k] = min(short[i][k][k-1] + short[k][j][k-1], short[i][j][k-1])

$k = 0$: shortest path cannot use any intermediate nodes (must be direct path)

$k = 0$: shortest path from node to itself is always 0

Short($i, j, k$) = min \{ Short($i, k, k - 1$) + Short($k, j, k - 1$), Short($i, j, k - 1$) \}
Floyd-Warshall All-Pairs Shortest Paths

allocate short[n][n] (initialized to ∞)
for (i, j) in E:
    short[i][j] = w[i][j]
for i = 1,...,n:
    short[i][i] = 0

for k = 1,...,n:
    for i = 1,...,n:
        for j = 1,...,n:
            short[i][j] = min(short[i][k] + short[k][j], short[i][j])

Observation: short[i][j][k] only depends on values for short[][][k - 1], so we can just use a single two-dimensional array

In this case, the initialization step is constructing the adjacency matrix of the graph (this step is not needed if graph already represented in this form!)
Floyd-Warshall All-Pairs Shortest Paths

allocate short[n][n] (initialized to $\infty$)

for (i, j) in E:
    short[i][j] = w[i][j]

for $i = 1, \ldots, n$:
    short[i][i] = 0

for $k = 1, \ldots, n$:
    for $i = 1, \ldots, n$:
        for $j = 1, \ldots, n$:
            short[i][j] = min(short[i][k] + short[k][j], short[i][j])

$k = 0$: shortest path cannot use any intermediate nodes (must be direct path)

$k = 0$: shortest path from node to itself is always 0

Very simple implementation!

Running time: $O(n^3) = O(|V|^3)$
Shortest Paths Review

Single Source Shortest Paths

• Dijkstra: $\Theta(|E| \log |V|)$
  • Greedy algorithm (choose closest node to current explored nodes)
  • No negative edge weights
• Bellman-Ford: $\Theta(|E||V|)$
  • Dynamic programming algorithm
  • Supports negative edge weights (and finds negative weight cycles)
  • Supports local updates when edge weights change

All Pairs Shortest Paths

• Floyd-Warshall: $\Theta(|V|^3)$
  • Supports negative edge weights