**dynamic-programming** \((G, w, s)\)

1. \(f^0[s] \leftarrow 0\) and \(f^0[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2. **for** \(\ell \leftarrow 1\) **to** \(n - 1\) **do**
   3. copy \(f^{\ell-1} \rightarrow f^\ell\)
4. **for** each \((u, v) \in E\) **do**
   5. if \(f^{\ell-1}[u] + w(u, v) < f^\ell[v]\) then
   6. \(f^\ell[v] \leftarrow f^{\ell-1}[u] + w(u, v)\)
7. return \((f^{n-1}[v])_{v \in V}\)

**Obs.** Assuming there are no negative cycles, then a shortest path contains at most \(n - 1\) edges

**Proof.**

If there is a path containing at least \(n\) edges, then it contains a cycle. Removing the cycle gives a path with the same or smaller length. \(\square\)
dynamic-programming($G, w, s$)

1: $f^{old}[s] \leftarrow 0$ and $f^{old}[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
2: for $\ell \leftarrow 1$ to $n - 1$ do
3: copy $f^{old} \rightarrow f^{new}$
4: for each $(u, v) \in E$ do
5: if $f^{old}[u] + w(u, v) < f^{new}[v]$ then
6: $f^{new}[v] \leftarrow f^{old}[u] + w(u, v)$
7: copy $f^{new} \rightarrow f^{old}$
8: return $f^{old}$

- $f^\ell$ only depends on $f^{\ell-1}$: only need 2 vectors
Dynamic Programming with Better Space Usage

dynamic-programming\( (G, w, s) \)

1. \( f^{\text{old}}[s] \leftarrow 0 \) and \( f^{\text{old}}[v] \leftarrow \infty \) for any \( v \in V \setminus \{s\} \)
2. \textbf{for } \ell \leftarrow 1 \text{ to } n - 1 \text{ do}
3. \text{ copy } f^{\text{old}} \rightarrow f^{\text{new}}
4. \textbf{for each } (u, v) \in E \text{ do}
5. \text{ if } f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v] \text{ then}
6. \quad f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)
7. \text{ copy } f^{\text{new}} \rightarrow f^{\text{old}}
8. \textbf{return } f^{\text{old}}

- \( f^{\ell} \) only depends on \( f^{\ell-1} \): only need 2 vectors
- only need 1 vector!
Dynamic Programming with Better Space Usage

**dynamic-programming**($G, w, s$)

1: $f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
2: for $\ell \leftarrow 1$ to $n - 1$ do
3:   copy $f \rightarrow f$
4:   for each $(u, v) \in E$ do
5:     if $f[u] + w(u, v) < f[v]$ then
6:       $f[v] \leftarrow f[u] + w(u, v)$
7:   copy $f \rightarrow f$
8: return $f$

- $f^\ell$ only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!
Dynamic Programming with Better Space Usage

dynamic-programming\((G, w, s)\)

1: \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2: \textbf{for } \ell \leftarrow 1 \textbf{ to } n - 1 \textbf{ do}
3: \textbf{ for each } (u, v) \in E \textbf{ do}
4: \quad \textbf{if } f[u] + w(u, v) < f[v] \textbf{ then}
5: \quad \quad f[v] \leftarrow f[u] + w(u, v)
6: \textbf{return } f

- \(f^\ell\) only depends on \(f^{\ell-1}\): only need 2 vectors
- only need 1 vector!
Bellman-Ford Algorithm

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

1. \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2. \textbf{for} \(\ell \leftarrow 1\) to \(n - 1\) \textbf{do}
3. \hspace{1em} \textbf{for} each \((u, v) \in E\) \textbf{do}
4. \hspace{2em} \textbf{if} \(f[u] + w(u, v) < f[v]\) \textbf{then}
5. \hspace{3em} \(f[v] \leftarrow f[u] + w(u, v)\)
6. \textbf{return} \(f\)

- \(f^\ell\) only depends on \(f^{\ell-1}\): only need 2 vectors
- only need 1 vector!
Bellman-Ford Algorithm

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

1: $f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
2: for $\ell \leftarrow 1$ to $n - 1$ do
3: for each $(u, v) \in E$ do
4: if $f[u] + w(u, v) < f[v]$ then
5: $f[v] \leftarrow f[u] + w(u, v)$
6: return $f$

Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration
Bellman-Ford Algorithm

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

1: \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2: \(\text{for } \ell \leftarrow 1\ \text{to } n - 1\ \text{do}\)
3: \(\quad\text{for each } (u, v) \in E \ \text{do}\)
4: \(\quad\text{if } f[u] + w(u, v) < f[v] \ \text{then}\)
5: \(\quad\quad f[v] \leftarrow f[u] + w(u, v)\)
6: \(\text{return } f\)

- Issue: when we compute \(f[u] + w(u, v)\), \(f[u]\) may be changed since the end of last iteration
- This is OK: it can only “accelerate” the process!
Bellman-Ford Algorithm

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

1: \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2: \textbf{for} \(\ell \leftarrow 1\) to \(n - 1\) \textbf{do}
3: \hspace{1em} \textbf{for} each \((u, v) \in E\) \textbf{do}
4: \hspace{2em} \textbf{if} \(f[u] + w(u, v) < f[v]\) \textbf{then}
5: \hspace{3em} \(f[v] \leftarrow f[u] + w(u, v)\)
6: \textbf{return} \(f\)

- Issue: when we compute \(f[u] + w(u, v)\), \(f[u]\) may be changed since the end of last iteration
- This is OK: it can only “accelerate” the process!
- After iteration \(\ell\), \(f[v]\) is \textbf{at most} the length of the shortest path from \(s\) to \(v\) that uses at most \(\ell\) edges
Bellman-Ford Algorithm

$$\text{Bellman-Ford}(G, w, s)$$

1: \( f[s] \leftarrow 0 \) and \( f[v] \leftarrow \infty \) for any \( v \in V \setminus \{s\} \)
2: \textbf{for} \ \ell \leftarrow 1 \ \textbf{to} \ n - 1 \ \textbf{do}
3: \quad \textbf{for} \ \text{each} \ (u, v) \in E \ \textbf{do}
4: \quad \quad \textbf{if} \ f[u] + w(u, v) < f[v] \ \textbf{then}
5: \quad \quad \quad f[v] \leftarrow f[u] + w(u, v)
6: \quad \textbf{return} \ f

- Issue: when we compute \( f[u] + w(u, v) \), \( f[u] \) may be changed since the end of last iteration
- This is OK: it can only “accelerate” the process!
- After iteration \( \ell \), \( f[v] \) is at most the length of the shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges
- \( f[v] \) is always the length of some path from \( s \) to \( v \)
Bellman-Ford Algorithm

- After iteration $\ell$:
  
  \[
  \text{length of shortest } s-v \text{ path} \\
  \leq f[v] \\
  \leq \text{length of shortest } s-v \text{ path using at most } \ell \text{ edges}
  \]

- Assuming there are no negative cycles:
  
  \[
  \text{length of shortest } s-v \text{ path} \\
  = \text{length of shortest } s-v \text{ path using at most } n - 1 \text{ edges}
  \]

- So, assuming there are no negative cycles, after iteration $n - 1$:
  
  \[
  f[v] = \text{length of shortest } s-v \text{ path}
  \]
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

vertices
\[
\begin{array}{cccccc}
\text{f} & s & a & b & c & d \\
0 & \infty & \infty & \infty & \infty & \infty \\
\end{array}
\]
order in which we consider edges: 
(s, a), (s, b), (a, b), (a, c), (b, d), 
(c, d), (d, a)

vertices

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
order in which we consider edges:

(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>6</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
order in which we consider edges:
\((s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)\)

<table>
<thead>
<tr>
<th>vertices</th>
<th>(s)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>0</td>
<td>6</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
</tbody>
</table>
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
order in which we consider edges:

\[(s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)\]

<table>
<thead>
<tr>
<th>vertices</th>
<th>0</th>
<th>6</th>
<th>7</th>
<th>(\infty)</th>
<th>(\infty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
order in which we consider edges:

$(s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)$

<table>
<thead>
<tr>
<th>vertices</th>
<th>$s$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

vertices

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>∞</td>
</tr>
</tbody>
</table>
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>∞</td>
</tr>
</tbody>
</table>
order in which we consider edges:

\( (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a) \)

<table>
<thead>
<tr>
<th>vertices</th>
<th>( s )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
order in which we consider edges:

\((s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)\)

<table>
<thead>
<tr>
<th>vertices</th>
<th>(s)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
order in which we consider edges:

$(s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)$

<table>
<thead>
<tr>
<th>vertices</th>
<th>$s$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
order in which we consider edges: 
(s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

end of iteration 1: 0, 2, 7, 2, 4

diameter

end of iteration 2: 0, 2, 7, -2, 4

Algorithm terminates in 3 iterations,
instead of 4.

end of iteration 3: 0, 2, 7, -2, 4

vertices | s | a | b | c | d
----------|---|---|---|---|---
f         | 0 | 2 | 7 | 2 | 4
order in which we consider edges: 
\((s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)\)

end of iteration 1: 0, 2, 7, 2, 4

Algorithm terminates in 3 iterations, instead of 4.

<table>
<thead>
<tr>
<th>vertices</th>
<th>(s)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
order in which we consider edges:

$(s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)$

<table>
<thead>
<tr>
<th>vertices</th>
<th>$s$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4
order in which we consider edges: 
(s, a), (s, b), (a, b), (a, c), (b, d), 
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4

Algorithm terminates in 3 iterations, 
instead of 4.
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

vertices
\[ \begin{array}{|c|c|c|c|c|c|}
\hline
 f & s & a & b & c & d \\
\hline
 0 & 2 & 7 & 2 & 4 \\
\hline
\end{array} \]

end of iteration 1: 0, 2, 7, 2, 4

Algorithm terminates in 3 iterations, instead of 4.
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

vertices

<table>
<thead>
<tr>
<th>f</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4
order in which we consider edges: 
(s, a), (s, b), (a, b), (a, c), (b, d), 
(c, d), (d, a)

vertices

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4

Algorithm terminates in 3 iterations, instead of 4.
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4

Algorithm terminates in 3 iterations,
instead of 4.
order in which we consider edges: 
(s, a), (s, b), (a, b), (a, c), (b, d), 
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

end of iteration 1: 0, 2, 7, 2, 4
end of iteration 2: 0, 2, 7, -2, 4
order in which we consider edges: 
(s, a), (s, b), (a, b), (a, c), (b, d), 
(c, d), (d, a)

end of iteration 1: 0, 2, 7, 2, 4
end of iteration 2: 0, 2, 7, -2, 4
end of iteration 3: 0, 2, 7, -2, 4

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>
order in which we consider edges:
(s, a), (s, b), (a, b), (a, c), (b, d),
(c, d), (d, a)

<table>
<thead>
<tr>
<th>vertices</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>-2</td>
<td>4</td>
</tr>
</tbody>
</table>

divide the vertices as follows:
end of iteration 1: 0, 2, 7, 2, 4
end of iteration 2: 0, 2, 7, -2, 4
end of iteration 3: 0, 2, 7, -2, 4
Algorithm terminates in 3 iterations, instead of 4.
Bellman-Ford Algorithm

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

1: \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2: for \(\ell \leftarrow 1\) to \(n\) do
3: \(\text{updated } \leftarrow \text{false}\)
4: for each \((u, v) \in E\) do
5: \(\text{if } f[u] + w(u, v) < f[v] \text{ then}\)
6: \(f[v] \leftarrow f[u] + w(u, v)\)
7: \(\text{updated } \leftarrow \text{true}\)
8: if not \(\text{updated}\), then return \(f\)
9: output “negative cycle exists”
Bellman-Ford Algorithm

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

1: $f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
2: for $\ell \leftarrow 1$ to $n$ do
3: \hspace{1em} updated $\leftarrow$ false
4: \hspace{1em} for each $(u, v) \in E$ do
5: \hspace{2em} if $f[u] + w(u, v) < f[v]$ then
6: \hspace{3em} $f[v] \leftarrow f[u] + w(u, v)$, $\pi[v] \leftarrow u$
7: \hspace{1em} updated $\leftarrow$ true
8: \hspace{1em} if not updated, then return $f$
9: output “negative cycle exists”

- $\pi[v]$: the parent of $v$ in the shortest path tree
Bellman-Ford Algorithm

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

1: \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2: for \(\ell \leftarrow 1\) to \(n\) do
3: \(\text{updated} \leftarrow \text{false}\)
4: for each \((u, v) \in E\) do
5: \(\text{if } f[u] + w(u, v) < f[v] \text{ then}\)
6: \(f[v] \leftarrow f[u] + w(u, v), \pi[v] \leftarrow u\)
7: \(\text{updated} \leftarrow \text{true}\)
8: if not \(\text{updated}\), then return \(f\)
9: output “negative cycle exists”

\(\pi[v]\): the parent of \(v\) in the shortest path tree

Running time = \(O(nm)\)
Outline

1 Minimum Spanning Tree
   • Kruskal’s Algorithm
   • Reverse-Kruskal’s Algorithm
   • Prim’s Algorithm

2 Single Source Shortest Paths
   • Dijkstra’s Algorithm

3 Shortest Paths in Graphs with Negative Weights

4 All-Pair Shortest Paths and Floyd-Warshall
All-Pair Shortest Paths

**Input:** directed graph $G = (V, E)$,

$w : E \rightarrow \mathbb{R}$ (can be negative)

**Output:** shortest path from $u$ to $v$ for every $u, v \in V$

1. For every starting point $s \in V$
2. Run Bellman-Ford ($G, w, s$)

Running time $= O(n^2 m)$
All-Pair Shortest Paths

**Input:** directed graph $G = (V, E)$,
\[ w : E \to \mathbb{R} \text{ (can be negative)} \]

**Output:** shortest path from $u$ to $v$ for every $u, v \in V$

1. **for** every starting point $s \in V$ **do**
2. run Bellman-Ford($G, w, s$)
All Pair Shortest Paths

**Input:** directed graph $G = (V, E)$, 
\[ w : E \rightarrow \mathbb{R} \text{ (can be negative)} \]

**Output:** shortest path from $u$ to $v$ for every $u, v \in V$

1: for every starting point $s \in V$ do
2: run Bellman-Ford($G, w, s$)

- Running time $= O(n^2m)$
### Summary of Shortest Path Algorithms we learned

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Graph Type</th>
<th>Weight Set</th>
<th>SS?</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple DP</td>
<td>DAG</td>
<td>$\mathbb{R}$</td>
<td>SS</td>
<td>$O(n + m)$</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>U/D</td>
<td>$\mathbb{R}_{\geq 0}$</td>
<td>SS</td>
<td>$O(n \log n + m)$</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>U/D</td>
<td>$\mathbb{R}$</td>
<td>SS</td>
<td>$O(nm)$</td>
</tr>
<tr>
<td>Floyd-Warshall</td>
<td>U/D</td>
<td>$\mathbb{R}$</td>
<td>AP</td>
<td>$O(n^3)$</td>
</tr>
</tbody>
</table>

- DAG = directed acyclic graph
- U = undirected
- D = directed
- SS = single source
- AP = all pairs

- $O(n + m)$ is the time complexity for Simple DP and Dijkstra Algorithms.
- $O(n \log n + m)$ is the time complexity for Dijkstra Algorithm.
- $O(nm)$ is the time complexity for Bellman-Ford Algorithm.
- $O(n^3)$ is the time complexity for Floyd-Warshall Algorithm.
It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$
- For simplicity, extend the $w$ values to non-edges:

$$w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E
\end{cases}$$
It is convenient to assume \( V = \{1, 2, 3, \ldots, n\} \)

For simplicity, extend the \( w \) values to non-edges:

\[
w(i, j) = \begin{cases} 
0 & \text{if } i = j \\
\text{weight of edge } (i, j) & \text{if } i \neq j, (i, j) \in E \\
\infty & \text{if } i \neq j, (i, j) \notin E
\end{cases}
\]

For now assume there are no negative cycles
It is convenient to assume $V = \{1, 2, 3, \cdots, n\}$

For simplicity, extend the $w$ values to non-edges:

$$w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge} \ (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E
\end{cases}$$

For now assume there are no negative cycles
Design a Dynamic Programming Algorithm

- It is convenient to assume \( V = \{1, 2, 3, \ldots, n\} \)
- For simplicity, extend the \( w \) values to non-edges:

\[
w(i, j) = \begin{cases} 
0 & \text{if } i = j \\
\text{weight of edge } (i, j) & \text{if } i \neq j, (i, j) \in E \\
\infty & \text{if } i \neq j, (i, j) \notin E
\end{cases}
\]

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

- First try: \( f[i, j] \) is length of shortest path from \( i \) to \( j \)
Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \ldots, n\}$
- For simplicity, extend the $w$ values to non-edges:

$$w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E
\end{cases}$$

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

- First try: $f[i, j]$ is length of shortest path from $i$ to $j$
- Issue: do not know in which order we compute $f[i, j]$’s
Design a Dynamic Programming Algorithm

- It is convenient to assume $V = \{1, 2, 3, \ldots, n\}$
- For simplicity, extend the $w$ values to non-edges:

\[
\begin{align*}
    w(i, j) = \begin{cases} 
        0 & i = j \\
        \text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
        \infty & i \neq j, (i, j) \notin E
    \end{cases}
\end{align*}
\]

- For now assume there are no negative cycles

Cells for Floyd-Warshall Algorithm

- First try: $f[i, j]$ is length of shortest path from $i$ to $j$
- Issue: do not know in which order we compute $f[i, j]$’s

$f^k[i, j]$: length of shortest path from $i$ to $j$ that only uses vertices \(\{1, 2, 3, \ldots, k\}\) as intermediate vertices
Example for Definition of $f^k[i, j]$’s

\[
\begin{align*}
f^0[1, 4] &= \infty \\
f^1[1, 4] &= \infty \\
f^2[1, 4] &= 140 \quad (1 \rightarrow 2 \rightarrow 4) \\
f^3[1, 4] &= 90 \quad (1 \rightarrow 3 \rightarrow 2 \rightarrow 4) \\
f^4[1, 4] &= 90 \quad (1 \rightarrow 3 \rightarrow 2 \rightarrow 4) \\
f^5[1, 4] &= 60 \quad (1 \rightarrow 3 \rightarrow 5 \rightarrow 4)
\end{align*}
\]
\[ w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E 
\end{cases} \]

- \( f^k[i, j] \): length of shortest path from \( i \) to \( j \) that only uses vertices \( \{1, 2, 3, \ldots, k\} \) as intermediate vertices
\[ w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E 
\end{cases} \]

- \( f^k[i, j] \): length of shortest path from \( i \) to \( j \) that only uses vertices \( \{1, 2, 3, \ldots, k\} \) as intermediate vertices

\[ f^k[i, j] = \begin{cases} 
k = 0 \\
k = 1, 2, \ldots, n 
\end{cases} \]
\[ w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E 
\end{cases} \]

- \( f^k[i, j] \): length of shortest path from \( i \) to \( j \) that only uses vertices \( \{1, 2, 3, \ldots, k\} \) as intermediate vertices

\[ f^k[i, j] = \begin{cases} 
\text{weight of edge } (i, j) & k = 0 \\
\min\left( f^k[i, j], f^{k-1}[i, k] + f^{k-1}[k, j] \right) & k = 1, 2, \ldots, n 
\end{cases} \]
\[
\begin{align*}
    w(i, j) &= \begin{cases} 
        0 & \text{if } i = j \\
        \text{weight of edge } (i, j) & \text{if } i \neq j, (i, j) \in E \\
        \infty & \text{if } i \neq j, (i, j) \notin E
        \end{cases}
\end{align*}
\]

\[f^k[i, j]: \text{ length of shortest path from } i \text{ to } j \text{ that only uses vertices } \{1, 2, 3, \cdots, k\} \text{ as intermediate vertices}\]

\[
\begin{align*}
    f^k[i, j] &= \begin{cases} 
        w(i, j) & \text{if } k = 0 \\
        \min \left\{ f^k[i, j] \right\} & \text{if } k = 1, 2, \cdots, n
        \end{cases}
\end{align*}
\]
\[ w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E 
\end{cases} \]

- \( f^k[i, j] \): length of shortest path from \( i \) to \( j \) that only uses vertices \( \{1, 2, 3, \cdots, k\} \) as intermediate vertices

\[ f^k[i, j] = \begin{cases} 
w(i, j) & k = 0 \\
\min \left\{ f^{k-1}[i, j] \right\} & k = 1, 2, \cdots, n 
\end{cases} \]
\[ w(i, j) = \begin{cases} 
0 & i = j \\
\text{weight of edge } (i, j) & i \neq j, (i, j) \in E \\
\infty & i \neq j, (i, j) \notin E 
\end{cases} \]

- \( f^k[i, j] \): length of shortest path from \( i \) to \( j \) that only uses vertices \( \{1, 2, 3, \cdots, k\} \) as intermediate vertices

\[
f^k[i, j] = \begin{cases} 
  w(i, j) & k = 0 \\
  \min \left\{ f^{k-1}[i, j], f^{k-1}[i, k] + f^{k-1}[k, j] \right\} & k = 1, 2, \cdots, n
\end{cases}
\]
Floyd-Warshall($G, w$)

1: $f^0 \leftarrow w$
2: for $k \leftarrow 1$ to $n$ do
3: copy $f^{k-1} \rightarrow f^k$
4: for $i \leftarrow 1$ to $n$ do
5: for $j \leftarrow 1$ to $n$ do
6: if $f^{k-1}[i, k] + f^{k-1}[k, j] < f^k[i, j]$ then
7: $f^k[i, j] \leftarrow f^{k-1}[i, k] + f^{k-1}[k, j]$
Floyd-Warshall\((G, w)\)

1: \(f^{old} \leftarrow w\)
2: \textbf{for } \(k \leftarrow 1 \text{ to } n \) \textbf{do}
3: \hspace{1em} \text{copy } f^{old} \rightarrow f^{new}\)
4: \textbf{for } \(i \leftarrow 1 \text{ to } n \) \textbf{do}
5: \hspace{2em} \textbf{for } \(j \leftarrow 1 \text{ to } n \) \textbf{do}
6: \hspace{3em} \textbf{if } f^{old}[i, k] + f^{old}[k, j] < f^{new}[i, j] \textbf{ then}
7: \hspace{4em} f^{new}[i, j] \leftarrow f^{old}[i, k] + f^{old}[k, j]\)

Lemma

Assume there are no negative cycles in \(G\). After iteration \(k\), for \(i, j \in V\), \(f[i, j]\) is exactly the length of shortest path from \(i\) to \(j\) that only uses vertices in \(\{1, 2, 3, \ldots, k\}\) as intermediate vertices.

Running time = \(O(n^3)\).
Floyd-Warshall($G, w$)

1: \( f^{\text{old}} \leftarrow w \)
2: \textbf{for} \( k \leftarrow 1 \) to \( n \) \textbf{do}
3: \hspace{1em} \text{copy} \ f^{\text{old}} \rightarrow f^{\text{new}}
4: \textbf{for} \ i \leftarrow 1 \) to \( n \) \textbf{do}
5: \hspace{1em} \textbf{for} \ j \leftarrow 1 \) to \( n \) \textbf{do}
6: \hspace{2em} \textbf{if} \ f^{\text{old}}[i, k] + f^{\text{old}}[k, j] < f^{\text{new}}[i, j] \textbf{ then}
7: \hspace{3em} f^{\text{new}}[i, j] \leftarrow f^{\text{old}}[i, k] + f^{\text{old}}[k, j]
Floyd-Warshall\((G, w)\)

1: \(f \leftarrow w\)
2: \textbf{for} \(k \leftarrow 1\) \textbf{to} \(n\) \textbf{do}
3: \hspace{1em} \text{copy } f \rightarrow f
4: \textbf{for} \(i \leftarrow 1\) \textbf{to} \(n\) \textbf{do}
5: \hspace{2em} \textbf{for} \(j \leftarrow 1\) \textbf{to} \(n\) \textbf{do}
6: \hspace{3em} \textbf{if} \(f[i, k] + f[k, j] < f[i, j]\) \textbf{then}
7: \hspace{4em} \(f[i, j] \leftarrow f[i, k] + f[k, j]\)

Lemma
Assume there are no negative cycles in \(G\). After iteration \(k\), for \(i, j \in V\), \(f[i, j]\) is exactly the length of shortest path from \(i\) to \(j\) that only uses vertices in \(\{1, 2, 3, \ldots, k\}\) as intermediate vertices.

Running time \(= O(n^3)\).
Floyd-Warshall($G, w$)

1: $f \leftarrow w$
2: for $k \leftarrow 1$ to $n$ do
3: for $i \leftarrow 1$ to $n$ do
4: for $j \leftarrow 1$ to $n$ do
5: if $f[i, k] + f[k, j] < f[i, j]$ then
6: $f[i, j] \leftarrow f[i, k] + f[k, j]$

Lemma
Assume there are no negative cycles in $G$. After iteration $k$, for $i, j \in V$, $f[i, j]$ is exactly the length of shortest path from $i$ to $j$ that only uses vertices in \{1, 2, 3, $\cdots$, $k$\} as intermediate vertices.

Running time $= O(n^3)$.
Floyd-Warshall($G, w$)

1: $f \leftarrow w$
2: for $k \leftarrow 1$ to $n$ do
3:     for $i \leftarrow 1$ to $n$ do
4:         for $j \leftarrow 1$ to $n$ do
5:             if $f[i, k] + f[k, j] < f[i, j]$ then
6:                 $f[i, j] \leftarrow f[i, k] + f[k, j]$

**Lemma**  Assume there are no negative cycles in $G$. After iteration $k$, for $i, j \in V$, $f[i, j]$ is exactly the length of shortest path from $i$ to $j$ that only uses vertices in $\{1, 2, 3, \cdots, k\}$ as intermediate vertices.
Floyd-Warshall($G, w$)

1: $f \leftarrow w$
2: for $k \leftarrow 1$ to $n$ do
3:     for $i \leftarrow 1$ to $n$ do
4:         for $j \leftarrow 1$ to $n$ do
5:             if $f[i, k] + f[k, j] < f[i, j]$ then
6:                 $f[i, j] \leftarrow f[i, k] + f[k, j]$ 

Lemma  Assume there are no negative cycles in $G$. After iteration $k$, for $i, j \in V$, $f[i, j]$ is exactly the length of shortest path from $i$ to $j$ that only uses vertices in $\{1, 2, 3, \cdots, k\}$ as intermediate vertices.

- Running time $= O(n^3)$. 
\( i = 2, \; k = 1, \; j = 3 \)
\[ i = 2, \ k = 1, \ j = 3 \]
\( i = 1, \; k = 2, \; j = 4 \)
\( i = 1, \ k = 2, \ j = 4 \)}
$i = 3, \ k = 2, \ j = 1,$
\[i = 3, \ k = 2, \ j = 1,\]
\[ i = 3, \; k = 2, \; j = 4 \]
\( i = 3, \ k = 2, \ j = 4 \)
\[ i = 1, \quad k = 3, \quad j = 2 \]
\[ i = 1, \quad k = 3, \quad j = 2 \]
Floyd-Warshall\((G, w)\)

1: \(f \leftarrow w, \pi[i, j] \leftarrow \perp\) for every \(i, j \in V\)

2: \textbf{for} \(k \leftarrow 1\) to \(n\) \textbf{do}

3: \hspace{1em} \textbf{for} \(i \leftarrow 1\) to \(n\) \textbf{do}

4: \hspace{2em} \textbf{for} \(j \leftarrow 1\) to \(n\) \textbf{do}

5: \hspace{3em} \textbf{if} \(f[i, k] + f[k, j] < f[i, j]\) \textbf{then}

6: \hspace{3em} \(f[i, j] \leftarrow f[i, k] + f[k, j], \pi[i, j] \leftarrow k\)
Recovering Shortest Paths

**Floyd-Warshall**\((G, w)\)

\[
\begin{align*}
1: \quad & f \leftarrow w, \pi[i, j] \leftarrow \bot \text{ for every } i, j \in V \\
2: \quad & \textbf{for } k \leftarrow 1 \text{ to } n \textbf{ do} \\
3: \quad & \textbf{for } i \leftarrow 1 \text{ to } n \textbf{ do} \\
4: \quad & \textbf{for } j \leftarrow 1 \text{ to } n \textbf{ do} \\
5: \quad & \textbf{if } f[i, k] + f[k, j] < f[i, j] \textbf{ then} \\
6: \quad & f[i, j] \leftarrow f[i, k] + f[k, j], \pi[i, j] \leftarrow k \\
\end{align*}
\]

**print-path**\((i, j)\)

\[
\begin{align*}
1: \quad & \textbf{if } \pi[i, j] = \bot \textbf{ then then} \\
2: \quad & \textbf{if } i \neq j \textbf{ then print}(i,“,” ) \\
3: \quad & \textbf{else} \\
4: \quad & \textbf{print-path}(i, \pi[i, j]), \textbf{ print-path}(\pi[i, j], j) \\
\end{align*}
\]
Detecting Negative Cycles

Floyd-Warshall($G, w$)

1: $f \leftarrow w$, $\pi[i, j] \leftarrow \bot$ for every $i, j \in V$
2: for $k \leftarrow 1$ to $n$ do
3: for $i \leftarrow 1$ to $n$ do
4: for $j \leftarrow 1$ to $n$ do
5: if $f[i, k] + f[k, j] < f[i, j]$ then
6: $f[i, j] \leftarrow f[i, k] + f[k, j]$, $\pi[i, j] \leftarrow k$
Detecting Negative Cycles

Floyd-Warshall(G, w)

1. \( f \leftarrow w, \pi[i, j] \leftarrow \bot \) for every \( i, j \in V \)
2. for \( k \leftarrow 1 \) to \( n \) do
3. for \( i \leftarrow 1 \) to \( n \) do
4. for \( j \leftarrow 1 \) to \( n \) do
5. if \( f[i, k] + f[k, j] < f[i, j] \) then
6. \( f[i, j] \leftarrow f[i, k] + f[k, j], \pi[i, j] \leftarrow k \)
7. for \( k \leftarrow 1 \) to \( n \) do
8. for \( i \leftarrow 1 \) to \( n \) do
9. for \( j \leftarrow 1 \) to \( n \) do
10. if \( f[i, k] + f[k, j] < f[i, j] \) then
11. report “negative cycle exists” and exit