**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: \(\text{copy } f^{\ell-1} \rightarrow f^{\ell}\)
4: **for** each \((u, v) \in E\) **do**
5: \(\text{if } f^{\ell-1}[u] + w(u, v) < f^{\ell}[v] \text{ then}\)
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. □
**Dynamic Programming with Better Space Usage**

**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. for $\ell \leftarrow 1$ to $n - 1$ do
3.     copy $f^{\text{old}} \rightarrow f^{\text{new}}$
4.     for each $(u, v) \in E$ do
5.         if $f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v]$ then
6.             $f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)$
7.     copy $f^{\text{new}} \rightarrow f^{\text{old}}$
8. 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\((G, w, s)\)

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

- \(f^\ell\) 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. 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
- **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: 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
- 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
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. \(\text{for } \ell \leftarrow 1\) to \(n - 1\) \(\text{do}\)
3. \(\text{for each } (u, v) \in E \text{ do}\)
4. \(\text{if } f[u] + w(u, v) < f[v] \text{ then}\)
5. \(f[v] \leftarrow f[u] + w(u, v)\)
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!
- **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)

<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>∞</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>∞</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>(\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 | s | a | b | c | d
--- | --- | --- | --- | --- | ---
f | 0 | 6 | 7 | ∞ | ∞
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:
\((s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)\)

<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>(\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>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

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.
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
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
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

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.
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

day 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)

<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
end of iteration 3: 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
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: \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} $f[v] \leftarrow f[u] + w(u, v)$
7: \hspace{3em} $updated \leftarrow$ true
8: \hspace{1em} if not $updated$, 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:     updated $\leftarrow$ false
4:     for each $(u, v) \in E$ do
5:         if $f[u] + w(u, v) < f[v]$ then
6:             $f[v] \leftarrow f[u] + w(u, v)$, $\pi[v] \leftarrow u$
7:     updated $\leftarrow$ true
8:     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: \textbf{for} \(\ell \leftarrow 1\) to \(n\) \textbf{do}
3: \hspace{1em} \textit{updated} \leftarrow \text{false}
4: \hspace{1em} \textbf{for each} \((u, v) \in E\) \textbf{do}
5: \hspace{2em} \textbf{if} \(f[u] + w(u, v) < f[v]\) \textbf{then}
6: \hspace{3em} \(f[v] \leftarrow f[u] + w(u, v), \pi[v] \leftarrow u\)
7: \hspace{1em} \textit{updated} \leftarrow \text{true}
8: \hspace{1em} \textbf{if not} \textit{updated}, \textbf{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 $u, v \in V$

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

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

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

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

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

1: for every starting point $s \in V$ do
2: run Bellman-Ford$(G, w, s)$
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 $s \in V$ 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</th>
<th>weights</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
Design a Dynamic Programming 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:

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

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

- For now assume there are no negative cycles

**Cells for Floyd-Warshall Algorithm**
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:

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

- 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 $V = \{1, 2, 3, \cdots, n\}$
- 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

**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:

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

- 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