“Evidence” for $e \in$ MST or $e \notin$ MST

**Assumption** Assume all edge weights are different.

- $e \in$ MST $\iff$ there is a cut in which $e$ is the lightest edge
- $e \notin$ MST $\iff$ there is a cycle in which $e$ is the heaviest edge

Exactly one of the following is true:
- There is a cut in which $e$ is the lightest edge
- There is a cycle in which $e$ is the heaviest edge

Thus, the minimum spanning tree is unique with assumption.
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
<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>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>R</td>
<td>SS</td>
<td>$O(nm)$</td>
</tr>
<tr>
<td>Floyd-Warshall</td>
<td>U/D</td>
<td>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
$s$-$t$ Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$w : E \to \mathbb{R}_{\geq 0}$

**Output:** shortest path from $s$ to $t$
**s-t Shortest Paths**

**Input:** (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest path from $s$ to $t$
**s-t Shortest Paths**

**Input:** (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest path from $s$ to $t$
Single Source Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$
Single Source Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s \in V$

$$w : E \rightarrow \mathbb{R}_{\geq 0}$$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths

- We do not know how to solve $s$-$t$ shortest path problem more efficiently than solving single source shortest path problem.
Single Source Shortest Paths

**Input:** (directed or undirected) graph \( G = (V, E) \), \( s \in V \)
\[ w : E \rightarrow \mathbb{R}_{\geq 0} \]

**Output:** shortest paths from \( s \) to all other vertices \( v \in V \)

Reason for Considering Single Source Shortest Paths

- We do not know how to solve \( s-t \) shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight
Single Source Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \to \mathbb{R}_{\geq 0}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve $s$-$t$ shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight
**Single Source Shortest Paths**

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

$$w : E \rightarrow \mathbb{R}_{\geq 0}$$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

**Reason for Considering Single Source Shortest Paths Problem**

- We do not know how to solve $s$-$t$ shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight
Single Source Shortest Paths

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

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:**
- $\pi[v], v \in V \setminus s$: the parent of $v$ in shortest path tree
- $d[v], v \in V \setminus s$: the length of shortest path from $s$ to $v$
Q: How to compute shortest paths from $s$ when all edges have weight 1?
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
**Assumption**  Weights $w(u, v)$ are integers (w.l.o.g.).
Assumption  Weights $w(u, v)$ are integers (w.l.o.g.).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges

```
1
u   1   1   1   1   v
```

```
4
u -> v
```
**Assumption**  Weights $w(u, v)$ are integers (w.l.o.g).

An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges

---

**Shortest Path Algorithm by Running BFS**

1. replace $(u, v)$ of length $w(u, v)$ with a path of $w(u, v)$ unit-weight edges, for every $(u, v) \in E$
2. run BFS
3. $\pi[v] \leftarrow$ vertex from which $v$ is visited
4. $d[v] \leftarrow$ index of the level containing $v$
**Assumption**  Weights $w(u, v)$ are integers (w.l.o.g.).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges

![Graph](image)

---

**Shortest Path Algorithm by Running BFS**

1. replace $(u, v)$ of length $w(u, v)$ with a path of $w(u, v)$ unit-weight edges, for every $(u, v) \in E$
2. run BFS
3. $\pi[v] \leftarrow$ vertex from which $v$ is visited
4. $d[v] \leftarrow$ index of the level containing $v$

- Problem: $w(u, v)$ may be too large!
Assumption  Weights $w(u, v)$ are integers (w.l.o.g).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges

\[
\begin{array}{c}
\text{Shortest Path Algorithm by Running BFS} \\
1: \text{replace } (u, v) \text{ of length } w(u, v) \text{ with a path of } w(u, v) \\
\quad \text{unit-weight edges, for every } (u, v) \in E \\
2: \text{run BFS virtually} \\
3: \pi[v] \leftarrow \text{vertex from which } v \text{ is visited} \\
4: d[v] \leftarrow \text{index of the level containing } v
\end{array}
\]

- Problem: $w(u, v)$ may be too large!
Shortest Path Algorithm by Running BFS Virtually

1: $S \leftarrow \{s\}, \ d(s) \leftarrow 0$
2: while $|S| \leq n$ do
3: find a $v \notin S$ that minimizes $\min_{u \in S: (u,v) \in E} \{d[u] + w(u, v)\}$
4: $S \leftarrow S \cup \{v\}$
5: $d[v] \leftarrow \min_{u \in S: (u,v) \in E} \{d[u] + w(u, v)\}$
Virtual BFS: Example
Virtual BFS: Example

Time 0
Virtual BFS: Example

Time 2
Virtual BFS: Example

Time 4
Virtual BFS: Example

Time 7
Virtual BFS: Example

Time 9
Virtual BFS: Example

Time 10
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
Dijkstra’s Algorithm

**Dijkstra**(*G, w, s*)

1. \( S \leftarrow \emptyset \), \( d(s) \leftarrow 0 \) and \( d[v] \leftarrow \infty \) for every \( v \in V \setminus \{s\} \)
2. while \( S \neq V \) do
3. \( u \leftarrow \text{vertex in } V \setminus S \text{ with the minimum } d[u] \)
4. add \( u \) to \( S \)
5. for each \( v \in V \setminus S \) such that \((u, v) \in E\) do
6. \( \text{if } d[u] + w(u, v) < d[v] \text{ then} \)
7. \( d[v] \leftarrow d[u] + w(u, v) \)
8. \( \pi[v] \leftarrow u \)
9. return \((d, \pi)\)
Dijkstra’s Algorithm

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

1: \(S \leftarrow \emptyset, d(s) \leftarrow 0\) and \(d[v] \leftarrow \infty\) for every \(v \in V \setminus \{s\}\)
2: \textbf{while} \(S \neq V\) \textbf{do}
3: \(u \leftarrow\) vertex in \(V \setminus S\) with the minimum \(d[u]\)
4: \(\text{add } u\) to \(S\)
5: \textbf{for} each \(v \in V \setminus S\) such that \((u, v) \in E\) \textbf{do}
6: \textbf{if} \(d[u] + w(u, v) < d[v]\) \textbf{then}
7: \(d[v] \leftarrow d[u] + w(u, v)\)
8: \(\pi[v] \leftarrow u\)
9: \textbf{return} \((d, \pi)\)

- Running time = \(O(n^2)\)
Improved Running Time using Priority Queue

**Dijkstra**($G, w, s$)

1: $s \leftarrow$ arbitrary vertex in $G$
2: $S \leftarrow \emptyset$, $d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
3: $Q \leftarrow$ empty queue, for each $v \in V$: $Q.insert(v, d[v])$
4: while $S \neq V$ do
5: \hspace{1em} $u \leftarrow Q.extract\_min()$
6: \hspace{1em} $S \leftarrow S \cup \{u\}$
7: \hspace{1em} for each $v \in V \setminus S$ such that $(u, v) \in E$ do
8: \hspace{2em} if $d[u] + w(u, v) < d[v]$ then
9: \hspace{3em} $d[v] \leftarrow d[u] + w(u, v)$, $Q.decrease\_key(v, d[v])$
10: \hspace{2em} $\pi[v] \leftarrow u$
11: return $(\pi, d)$
Recall: Prim’s Algorithm for MST

**MST-Prim**($G, w$)

1: $s \leftarrow$ arbitrary vertex in $G$
2: $S \leftarrow \emptyset$, $d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
3: $Q \leftarrow$ empty queue, for each $v \in V$: $Q$.insert($v, d[v]$)
4: while $S \neq V$ do
5: \hspace{2em} $u \leftarrow Q$.extract\_min()
6: \hspace{2em} $S \leftarrow S \cup \{u\}$
7: \hspace{2em} for each $v \in V \setminus S$ such that $(u, v) \in E$ do
8: \hspace{4em} if $w(u, v) < d[v]$ then
9: \hspace{6em} $d[v] \leftarrow w(u, v)$, $Q$.decrease\_key($v, d[v]$)
10: \hspace{6em} $\pi[v] \leftarrow u$
11: return $\{(u, \pi[u]) | u \in V \setminus \{s\}\}$
Improved Running Time

Running time: 
\[ O(n) \times \text{(time for extract\_min)} + O(m) \times \text{(time for decrease\_key)} \]

<table>
<thead>
<tr>
<th>Priority-Queue</th>
<th>extract_min</th>
<th>decrease_key</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O(m \log n))</td>
</tr>
<tr>
<td>Fibonacci Heap</td>
<td>(O(\log n))</td>
<td>(O(1))</td>
<td>(O(n \log n + m))</td>
</tr>
</tbody>
</table>
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
Single Source Shortest Paths, Weights May be Negative

**Input:** directed graph \( G = (V, E) \), \( s \in V \)

assume all vertices are reachable from \( s \)

\( w : E \rightarrow \mathbb{R} \)

**Output:** shortest paths from \( s \) to all other vertices \( v \in V \)
Single Source Shortest Paths, Weights May be Negative

**Input:** directed graph \( G = (V, E) \), \( s \in V \)
- assume all vertices are reachable from \( s \)
- \( w : E \rightarrow \mathbb{R} \)

**Output:** shortest paths from \( s \) to all other vertices \( v \in V \)

- In transition graphs, negative weights make sense
Single Source Shortest Paths, Weights May be Negative

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

Assume all vertices are reachable from $s$

$w : E \to \mathbb{R}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

- In transition graphs, negative weights make sense
- If we sell an item: ‘having the item’ $\rightarrow$ ‘not having the item’, weight is negative (we gain money)
Single Source Shortest Paths, Weights May be Negative

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

- assume all vertices are reachable from $s$

  $w : E \rightarrow \mathbb{R}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

- In transition graphs, negative weights make sense
- If we sell an item: ‘having the item’ $\rightarrow$ ‘not having the item’, weight is negative (we gain money)
- Dijkstra’s algorithm does not work anymore!
Dijkstra’s Algorithm Fails if We Have Negative Weights
Dijkstra’s Algorithm Fails if We Have Negative Weights
Dijkstra’s Algorithm Fails if We Have Negative Weights
Dijkstra’s Algorithm Fails if We Have Negative Weights
Q: What is the length of the shortest path from \( s \) to \( d \)?

A: 1

Definition: An edge \( e \) is a negative cycle if it has negative weight.

Dealing with Negative Cycles

Assume the input graph does not contain negative cycles, or allow the algorithm to report "negative cycle exists".
Q: What is the length of the shortest path from $s$ to $d$?
Q: What is the length of the shortest path from \( s \) to \( d \)?

A: \(-\infty\)
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$
Q: What is the length of the shortest path from \( s \) to \( d \)?

A: \(-\infty\)
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$
**Q:** What is the length of the shortest path from $s$ to $d$?

**A:** $-\infty$
Q: What is the length of the shortest path from \( s \) to \( d \)?

A: \(-\infty\)

Def. A negative cycle is a cycle in which the total weight of edges is negative.
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

Dealing with Negative Cycles
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

Dealing with Negative Cycles
- assume the input graph does not contain negative cycles, or
Q: What is the length of the shortest path from $s$ to $d$?

A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

Dealing with Negative Cycles
- assume the input graph does not contain negative cycles, or
- allow algorithm to report “negative cycle exists”
Q: What is the length of the shortest simple path from s to d?

A: Unfortunately, computing the shortest simple path between two vertices is an NP-hard problem.
Q: What is the length of the shortest simple path from $s$ to $d$?
Q: What is the length of the shortest simple path from $s$ to $d$?

A: 1
Q: What is the length of the shortest simple path from $s$ to $d$?

A: 1

Unfortunately, computing the shortest simple path between two vertices is an **NP-hard** problem.
<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
**Single Source Shortest Paths, Weights May be Negative**

**Input:** directed graph $G = (V, E)$, $s \in V$
- assume all vertices are reachable from $s$
- $w : E \rightarrow \mathbb{R}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$
Defining Cells of Table

Single Source Shortest Paths, Weights May be Negative

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

assume all vertices are reachable from $s$

$w : E \rightarrow \mathbb{R}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

- first try: $f[v]$: length of shortest path from $s$ to $v$
Defining Cells of Table

### Single Source Shortest Paths, Weights May be Negative

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

assume all vertices are reachable from $s$

$w : E \rightarrow \mathbb{R}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

- first try: $f[v]$: length of shortest path from $s$ to $v$
- issue: do not know in which order we compute $f[v]$’s
Defining Cells of Table

Single Source Shortest Paths, Weights May be Negative

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

- assume all vertices are reachable from $s$
- $w : E \rightarrow \mathbb{R}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

- first try: $f[v]$: length of shortest path from $s$ to $v$
- issue: do not know in which order we compute $f[v]$’s
- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \ldots, n - 1\}$, $v \in V$: length of shortest path from $s$ to $v$ that uses at most $\ell$ edges
\( f^\ell[v], \ell \in \{0, 1, 2, 3 \ldots , n - 1\}, v \in V : \) length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges
\[ \ell \in \{0, 1, 2, 3 \ldots, n - 1\}, \quad v \in V : \]
length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges

\[ f^\ell[v] = \]

\[ f^2[a] = \]
\[ f^\ell[v], \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, v \in V: \]
length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges

\[ f^2[a] = 6 \]
\[ f^\ell[v], \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, v \in V : \text{length of shortest path from } s \text{ to } v \text{ that uses at most } \ell \text{ edges} \]

- \[ f^2[a] = 6 \]
- \[ f^3[a] = \]
\[ f^\ell[v], \; \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, \; v \in V : \text{length of shortest path from } s \text{ to } v \text{ that uses at most } \ell \text{ edges} \]

- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)
\[ f^\ell[v], \ \ell \in \{0, 1, 2, 3 \cdots , n - 1\}, \ v \in V: \text{length of shortest path from } s \text{ to } v \text{ that uses at most } \ell \text{ edges} \]

- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)

\[
f^\ell[v] = \begin{cases} 
\ell = 0, v = s \\
\ell = 0, v \neq s \\
\ell > 0
\end{cases}
\]
\[ f^\ell[v], \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, v \in V : \]

length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges

- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)

\[
\begin{align*}
  f^\ell[v] &= \begin{cases} 
    0 & \ell = 0, v = s \\
    0 & \ell = 0, v \neq s \\
    \ell > 0 & \end{cases} 
\end{align*}
\]
$f^\ell[v], \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, v \in V:$ length of shortest path from $s$ to $v$ that uses at most $\ell$ edges

- $f^2[a] = 6$
- $f^3[a] = 2$

\[
f^\ell[v] = \begin{cases} 
0 & \ell = 0, v = s \\
\infty & \ell = 0, v \neq s \\
\ & \ell > 0
\end{cases}
\]
\[ f^\ell[v], \quad \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, \quad v \in V : \] length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges

- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)

\[
f^\ell[v] = \begin{cases} 
0 & \text{if } \ell = 0, v = s \\
\infty & \text{if } \ell = 0, v \neq s \\
\min\{ & \text{if } \ell > 0 
\end{cases}
\]
\[ f^\ell[v], \ell \in \{0, 1, 2, 3 \cdots , n - 1\}, v \in V : \]
length of shortest path from \( s \) to \( v \) that uses
at most \( \ell \) edges

- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)

\[ f^\ell[v] = \begin{cases} 
0 & \text{if } \ell = 0, v = s \\
\infty & \text{if } \ell = 0, v \neq s \\
\min \{ f^{\ell-1}[v] \} & \ell > 0
\end{cases} \]
\[ f^\ell[v], \; \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, \; v \in V : \]
length of shortest path from \( s \) to \( v \) that uses
at most \( \ell \) edges
\[ f^2[a] = 6 \]
\[ f^3[a] = 2 \]

\[ f^\ell[v] = \begin{cases} 
0 & \ell = 0, \; v = s \\
\infty & \ell = 0, \; v \neq s \\
\min \left\{ \min_{u: (u,v) \in E} \left( f^{\ell-1}[u] + w(u, v) \right) \right\} & \ell > 0 
\end{cases} \]
Dynamic Programming: Example

![Graph](image)

**f^0**

- **s**: 0
- **a**: ∞
- **b**: ∞
- **c**: ∞
- **d**: ∞

**Vertex Labels**

- **s**: 7
- **b**: 6
- **a**: 8
- **c**: -4
- **d**: -3

**Length-0 Edge**

- **c** to **d**: 7
Dynamic Programming: Example

\[ f^0 \]

\[ f^1 \]

\[
\begin{array}{c}
\text{s} \\
\text{b} \\
\text{c} \\
\text{d}
\end{array}
\]

\[
\begin{array}{c}
7 \quad 6 \\
8 \quad -2 \\
-4 \quad -3 \\
7
\end{array}
\]

\[
\begin{array}{c}
\text{s} \\
\text{a} \\
\text{b} \\
\text{c} \\
\text{d}
\end{array}
\]

\[
\begin{array}{c}
0 \\
\infty \\
\infty \\
\infty \\
\infty
\end{array}
\]

\[
\begin{array}{c}
6 \quad 7 \\
8 \quad -4 \\
-3 \\
7
\end{array}
\]

length-0 edge
Dynamic Programming: Example

\[ \text{length-0 edge} \]
Dynamic Programming: Example

\[
\begin{align*}
\text{length-0 edge} & \\
\end{align*}
\]
Dynamic Programming: Example

\[ \begin{array}{c}
\begin{array}{c}
\text{b} \\
\rightarrow \text{s} \\
\downarrow 7 \\
\text{a} \\
\downarrow 6 \\
\text{c} \\
\rightarrow \text{d} \\
\downarrow -2 \\
\end{array}
\end{array} \]

\[ \begin{array}{c}
\begin{array}{c}
\text{s} \\
\rightarrow \text{a} \\
\downarrow 8 \\
\text{b} \\
\rightarrow \text{d} \\
\downarrow 7 \\
\text{c} \\
\rightarrow \text{d} \\
\downarrow -3 \\
\end{array}
\end{array} \]

\[ \begin{array}{c}
\begin{array}{c}
\text{f}^0 \\
\text{s} \\
\rightarrow \text{a} \\
\downarrow 6 \\
\text{b} \\
\rightarrow \text{c} \\
\downarrow 7 \\
\text{c} \\
\rightarrow \text{d} \\
\downarrow -2 \\
\end{array}
\end{array} \]

\[ \begin{array}{c}
\begin{array}{c}
\text{f}^1 \\
\text{s} \\
\rightarrow \text{a} \\
\downarrow 8 \\
\text{b} \\
\rightarrow \text{c} \\
\downarrow -3 \\
\text{c} \\
\rightarrow \text{d} \\
\downarrow 7 \\
\end{array}
\end{array} \]

length-0 edge
Dynamic Programming: Example

\[ f^0 \]

\[ f^1 \]

length-0 edge
Dynamic Programming: Example

\begin{align*}
& \text{length-0 edge} \\
& f^0: s \rightarrow a, \quad a \rightarrow b, \quad b \rightarrow c, \quad c \rightarrow d \\
& f^1: s \rightarrow a, \quad a \rightarrow b, \quad b \rightarrow c, \quad c \rightarrow d \\
& f^2: s \rightarrow a, \quad a \rightarrow b, \quad b \rightarrow c, \quad c \rightarrow d
\end{align*}
Dynamic Programming: Example

\begin{itemize}
\item $f^0$
\item $f^1$
\item $f^2$
\end{itemize}

Length-0 edge
Dynamic Programming: Example

- Diagram of a graph with nodes labeled s, a, b, c, and d, with edges labeled with weights.
- Calculation of functions $f^0$, $f^1$, $f^2$.
- Length-0 edge indicated.
Dynamic Programming: Example

\begin{itemize}
\item $s$ to $a$ (length: 7)
\item $a$ to $b$ (length: 6)
\item $a$ to $d$ (length: 8)
\item $b$ to $c$ (length: -2)
\item $c$ to $d$ (length: -3)
\item $a$ to $d$ (length: -4)
\item $b$ to $d$ (length: -3)
\item $c$ to $d$ (length: -2)
\item $a$ to $c$ (length: 6)
\item $b$ to $c$ (length: 7)
\item $a$ to $b$ (length: 8)
\end{itemize}

The length of the shortest path from $s$ to $d$ is 0.
Dynamic Programming: Example

![Diagram of a graph with labeled nodes and edges]
Dynamic Programming: Example

- Graph with edges labeled with weights:
  - s to a: 7
  - a to d: 6
  - b to c: 8
  - c to d: 7
  - b to a: -2
  - c to b: -3
  - d to c: -4

- Dynamic Programming stages:
  - Initial stage ($f^0$):
    - s: 0
    - a: ∞
    - b: ∞
    - c: ∞
    - d: ∞
  - Stage 1 ($f^1$):
    - s: 0
    - a: 6
    - b: 7
    - c: ∞
    - d: ∞
  - Stage 2 ($f^2$):
    - s: 0
    - a: 6
    - b: 7
    - c: 2
    - d: 4

- Transition graph with edges labeled with weights:
  - s to a: 6
  - a to b: 8
  - b to c: -4
  - c to d: -3
  - b to c: -3
  - c to d: -2

- Definition of length-0 edge:
  - An edge with weight 0
Dynamic Programming: Example

length-0 edge
Dynamic Programming: Example

\[ f^0 \]
\[ f^1 \]
\[ f^2 \]
\[ f^3 \]

length-0 edge
Dynamic Programming: Example

- Diagram showing a graph with nodes and edges labeled with values. The values are likely weights of the edges.
- The text likely explains the process of dynamic programming, using the graph as an example.
- The term "length-0 edge" suggests the consideration of base cases or initial conditions in the dynamic programming approach.
Dynamic Programming: Example

Diagram of a directed graph with vertices labeled as $s$, $a$, $b$, $c$, and $d$. The graph includes edges with weights $7$, $6$, $8$, $-3$, $-4$, and $-2$. The diagram also shows four levels labeled $f^0$, $f^1$, $f^2$, and $f^3$, each representing a different stage of the dynamic programming algorithm.

The diagram illustrates the progression from the initial state $f^0$ to a final state, with each level showing the cumulative cost as the algorithm moves from one vertex to another. The length-0 edge is highlighted, indicating the starting point of the algorithm.
Dynamic Programming: Example

![Graph Diagram]

- $f^0$
- $f^1$
- $f^2$
- $f^3$

length-0 edge
Dynamic Programming: Example

![Graph](image)

- $f^0$: Initial state
- $f^1$: First iteration
- $f^2$: Second iteration
- $f^3$: Third iteration

Length-0 edge
Dynamic Programming: Example

```
\[ f^0 \]
\[ f^1 \]
\[ f^2 \]
\[ f^3 \]
\[ f^4 \]

length-0 edge
```
Dynamic Programming: Example

length-0 edge
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 $(f^{n-1}[v])_{v \in V}$
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: \textbf{for } \ell \leftarrow 1 \textbf{ to } n - 1 \textbf{ do}
3: \quad \text{copy } f^{\ell-1} \rightarrow f^\ell
4: \textbf{for each } (u, v) \in E \textbf{ do}
5: \quad \textbf{if } f^{\ell-1}[u] + w(u, v) < f^\ell[v] \textbf{ then}
6: \quad \quad f^\ell[v] \leftarrow f^{\ell-1}[u] + w(u, v)
7: \textbf{return } (f^{n-1}[v])_{v \in V}

\textbf{Obs.} Assuming there are no negative cycles, then a shortest path contains at most \(n - 1\) edges
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. □
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
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: \textbf{for } \ell \leftarrow 1 \text{ to } n - 1 \text{ do}
3: \hspace{1em} \text{copy } f^{old} \rightarrow f^{new}
4: \hspace{1em} \textbf{for each} \ (u, v) \in E \ \textbf{do}
5: \hspace{2em} \textbf{if} \ f^{old}[u] + w(u, v) < f^{new}[v] \ \textbf{then}
6: \hspace{3em} f^{new}[v] \leftarrow f^{old}[u] + w(u, v)
7: \hspace{1em} \text{copy } f^{new} \rightarrow f^{old}
8: \hspace{1em} \textbf{return} \ f^{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: \(\text{for } \ell \leftarrow 1\ \text{to}\ n - 1\ \text{do}\)
3: \(\text{copy } f \rightarrow f\)
4: \(\text{for each } (u, v) \in E\ \text{do}\)
5: \(\text{if } f[u] + w(u, v) < f[v] \text{ then}\)
6: \(f[v] \leftarrow f[u] + w(u, v)\)
7: \(\text{copy } f \rightarrow f\)
8: \(\text{return } f\)

- \(f^\ell\) 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: 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 $f$

- $f^\ell$ 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 \(f\)

- \(f^\ell\) 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. \hspace{1em} $f[v] \leftarrow f[u] + w(u, v)$
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**

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

1. \(f[s] \leftarrow 0\) and \(f[v] \leftarrow \infty\) for any \(v \in V \setminus \{s\}\)
2. \textbf{for} \(\ell \leftarrow 1\) 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 **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 \(v\).