Dijkstra($G, w, s$)

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

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

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

- In transition graphs, negative weights make sense
- If we sell a item: ‘having the item’ $\rightarrow$ ‘not having the item’, weight is negative (we gain money)
**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 a item: ‘having the item’ \( \rightarrow \) ‘not having the item’, weight is negative (we gain money).
- Dijkstra’s algorithm does not work any more!
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

![Graph with nodes and edges with weights](image)
What is the length of the shortest path from \( s \) to \( d \)?

**Def.**
An edge \( e \) is negative if its weight is negative.

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 \textbf{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 $s$
- $w : E \to \mathbb{R}$

**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 $v \in V$

- first try: $f[v]$: length of shortest path from $s$ to $v$
## 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 $v \in V$

- first try: $f[v]$: length of shortest path from $s$ to $v$
- issue: do not know in which order we compute $f[v]$’s
### Single Source Shortest Paths, Weights May be Negative

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

\[ w : E \rightarrow \mathbb{R} \]

**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

\[ f^\ell[v], \ \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, \ v \in V : \text{ length of shortest path from } s \text{ to } v \text{ that uses at most } \ell \text{ edges} \]
The length of the shortest path from \( s \) to \( v \) that uses at most \( l \) edges is defined as:

\[
fe[l](v) \in \{0, 1, 2, 3, \ldots, n-1\}, v \in V:
\]

where \( l \) is the number of edges used in the path.
\[ f^\ell[v], \, \ell \in \{0, 1, 2, 3 \cdots, n - 1\}, \, v \in V: \]
length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges

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

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

- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)
\[ f^\ell[v], \ \ell \in \{0, 1, 2, 3 \cdots, n-1\}, \ v \in V : \]

- length of shortest path from \( s \) to \( v \) that uses at most \( \ell \) edges
- \( f^2[a] = 6 \)
- \( f^3[a] = 2 \)

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

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

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

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

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

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

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

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

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

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

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

\[
\begin{array}{cccccc}
  & s & a & b & c & d \\
 f^0 & 0 & \infty & \infty & \infty & \infty \\
 s & 7 & 6 & \text{length-0 edge} & 8 & -2 \\
 b & 8 & \text{length-0 edge} & 7 & -3 & -4 \\
 a & 6 & 8 & -2 & 7 & -4 \\
 c & -2 & -3 & 7 & -4 & -3 \\
 d & -4 & -2 & -4 & -3 & -3 \\
\end{array}
\]
Dynamic Programming: Example

```
7 6
8
-2
-4
-3
7

s a b c d

length-0 edge
```

```
0

6 7
8
-4
-3
7

s a b c d

f^0

f^1
```

70/88
Dynamic Programming: Example

\[ f^0 \]

\[ f^1 \]

length-0 edge
Dynamic Programming: Example

\[ f^0 \]
\[ f^1 \]

length-0 edge
Dynamic Programming: Example

\[
\begin{align*}
&f^0 & s & a & b \quad & \quad & f^1 & a & b \quad & \quad & c \quad & \quad & d \\
&0 & 6 & 7 & \infty & 7 & \infty & 8 & -4 & \infty & -2 & \infty
\end{align*}
\]

- length-0 edge
Dynamic Programming: Example

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

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

length-0 edge
Dynamic Programming: Example

Diagram showing a graph with nodes labeled s, a, b, c, and d, and edges with weights. The text explains the concept of dynamic programming with a focus on the example.
Dynamic Programming: Example

![Diagram of a network with nodes and edges labeled with weights.](image-url)
Dynamic Programming: Example

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

length-0 edge
Dynamic Programming: Example

Length-0 edge
Dynamic Programming: Example

- Diagram of a graph with nodes labeled s, b, a, c, d, and edges labeled with weights.
- Nodes s, a, b, c, d are connected by edges with weights.
- The graph includes a length-0 edge.

- Dynamic Programming formulation with functions $f^0, f^1, f^2$.
  - $f^0(s) = 0$, $f^1(a) = 6$, $f^2(b) = 7$.
  - Edges with weights -2, -3, -4.

- Example problem with a network of nodes and edges, illustrating dynamic programming concepts.

- The goal is to find the optimal path from s to d.
Dynamic Programming: Example

![Graph with labeled vertices and edges](image)

- $s$: Source vertex
- $a$: Vertex with weight 6
- $b$: Vertex with weight -4
- $c$: Vertex with weight -3
- $d$: Vertex with weight 7

Weighted edges:
- $s 	o a$: 7
- $s 	o b$: 8
- $s 	o c$: 6
- $s 	o d$: 7
- $a 	o b$: -2
- $a 	o c$: -3
- $a 	o d$: -4
- $b 	o c$: 6
- $b 	o d$: 1
- $c 	o d$: 1

Length-0 edge:
- $s$: Source vertex

$f^0$: Initial step
- $s$: 0
- $a$: 6
- $b$: $\infty$
- $c$: $\infty$
- $d$: $\infty$

$f^1$: Second step
- $s$: 6
- $a$: 7
- $b$: 8
- $c$: $\infty$
- $d$: $\infty$

$f^2$: Third step
- $s$: 6
- $a$: 7
- $b$: 8
- $c$: 2
- $d$: 4

$f^3$: Final step
- $s$: 6
- $a$: 7
- $b$: 8
- $c$: 7
- $d$: 7

The final step shows the optimal path from $s$ to each vertex.
Dynamic Programming: Example

- Graph representation:
  - Vertices: s, a, b, c, d
  - Edges and weights:
    - sa: 7
    - sb: 6
    - sa
    - sb
    - sc: 8
    - sd: 6
    - ba: 8
    - bd: -3
    - bc: -2
    - cd: -4

- Dynamic Programming:
  - Initial function $f^0$:
    - $s$: 0
    - $a$: $\infty$
    - $b$: $\infty$
    - $c$: $\infty$
    - $d$: $\infty$

- Iterative updates:
  - $f^1$: $s$: 0, $a$: 6, $b$: 8, $c$: 7, $d$: $\infty$
  - $f^2$: $s$: 0, $a$: 6, $b$: 7, $c$: 2, $d$: 4
  - $f^3$: $s$: 0, $a$: 6, $b$: 7, $c$: 2, $d$: 4

- Length-0 edge:
  - The length-0 edge is a path that connects two vertices without any intermediate vertices.
Dynamic Programming: Example

![Graph showing dynamic programming example](image)
Dynamic Programming: Example

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

length-0 edge
Dynamic Programming: Example

\[
\begin{array}{c}
\text{length-0 edge}
\end{array}
\]
Dynamic Programming: Example

The diagram illustrates the concept of dynamic programming through an example graph. The graph consists of nodes labeled s, a, b, c, and d, connected by edges with weights. The algorithm progresses through stages, labeled as $f^0, f^1, f^2, f^3$, to find the optimal path.

- **Stage $f^0$**: The initial state, where $s$ is the source node.
- **Stage $f^1$**: The graph expands with added edges.
- **Stage $f^2$**: Further expansion, increasing the complexity.
- **Stage $f^3$**: The final stage, showing the complete graph.

The graph includes a length-0 edge, which is a special case in the algorithm.

The values at each node represent the cost or weight of reaching that node from the source, with the goal of minimizing the total cost.
Dynamic Programming: Example
Dynamic Programming: Example

```
7  6
8  7
-4 -3
-2
7  6 8
-3 7 -2
-4 6
```

```
s
0 6 8
6 7 8
8 7 -4
6 7 -3
6 7 -2
6 7 -2
6 7 -2
6 7 -2
```

```
length-0 edge
```

```
  s
  a
  b
  c
  d
```

```
f^0
f^1
f^2
f^3
f^4
```

```
0 6 8
6 7 8
8 7 -4
6 7 -3
6 7 -2
6 7 -2
6 7 -2
6 7 -2
```
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: 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
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. \[\Box\]
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: \hspace{1em} copy \(f^{\text{old}} \rightarrow f^{\text{new}}\)
4: for each \((u, v) \in E\) do
5: \hspace{1em} if \(f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v]\) then
6: \hspace{1em} \hspace{1em} \(f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)\)
7: \hspace{1em} 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**

```latex
\texttt{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} $\ell \leftarrow 1$ \textbf{to} $n - 1$ \textbf{do}
3: \hspace{1em} \text{copy} $f^{\text{old}} \rightarrow f^{\text{new}}$
4: \hspace{1em} \textbf{for} each $(u, v) \in E$ \textbf{do}
5: \hspace{2em} \textbf{if} $f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v]$ \textbf{then}
6: \hspace{3em} $f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)$
7: \hspace{1em} \text{copy} $f^{\text{new}} \rightarrow f^{\text{old}}$
8: \textbf{return} $f^{\text{old}}$

$\ell$ 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: \(\text{for } \ell \leftarrow 1\ \text{to } n - 1 \text{ do}\)
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. \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