CSE 331:

Algorithms & Complexity

"Shortest Path... Again"

Prof. Charlie Anne Carlson (She/Her)

Lecture 32 & 33

Nov 17th & 19th, 2025

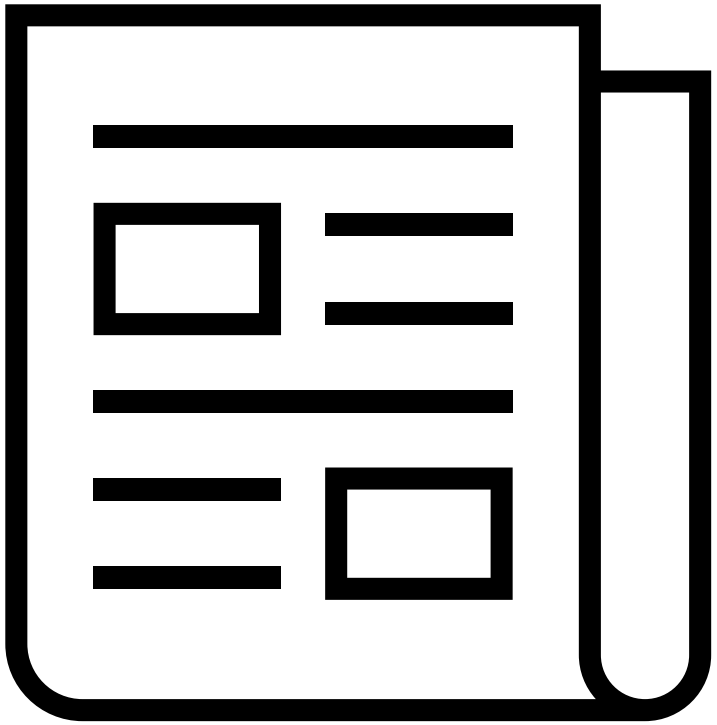
University at Buffalo®

Schedule

1. Course Updates
2. Dynamic Programming
3. Subset Sum -> Knapsack
4. Shortest Path Problem
5. Bellman-Ford Algorithm

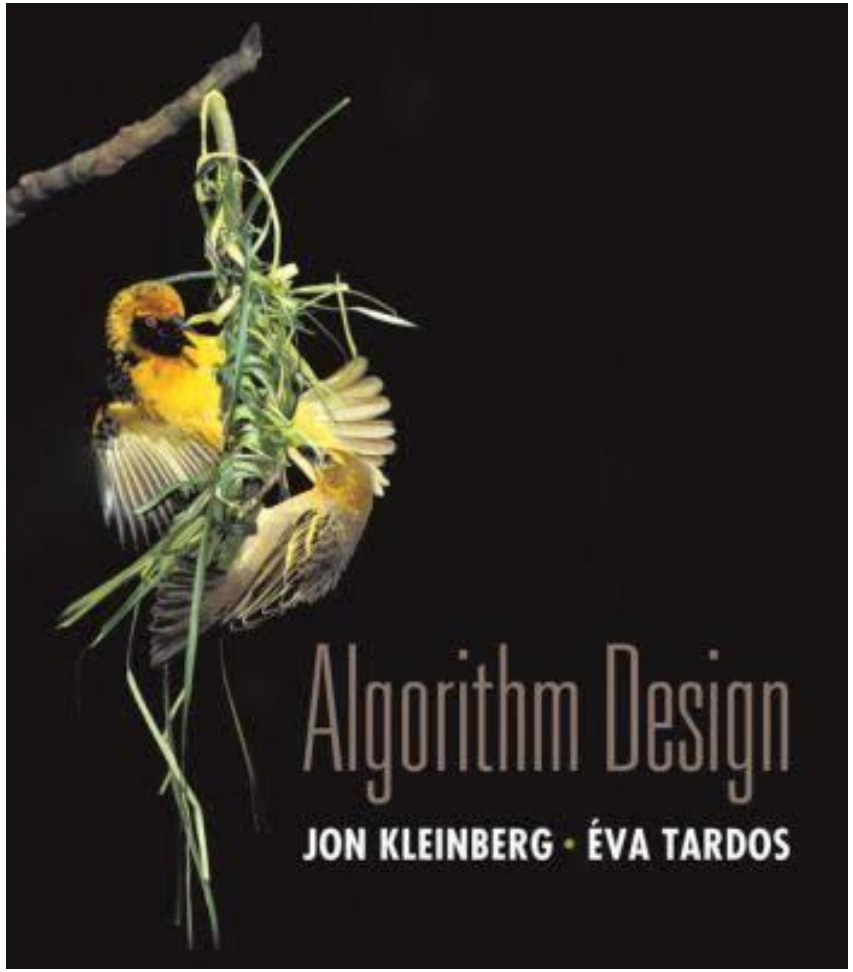
Course Updates

- HW 7 Out
  - Due November 19th
- Group Project (Autolab Up)
  - Groups Fixed
  - Code 3 Due November 24th
  - Reflections 3 Due December 1st
- Next Quiz is December 1st



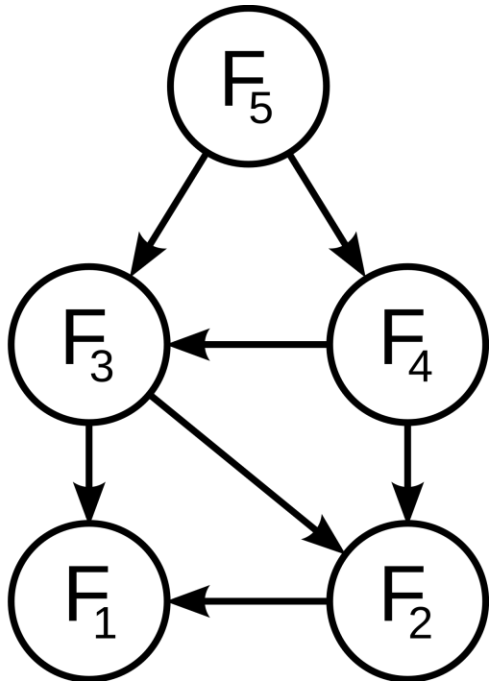
Reading

- You should have read:
  - Finished 6.1
  - Finished 6.2
  - Finished 6.4
  - Finished 6.8
- Before Next Class:
  - Start 8.1



Dynamic Programming

- Optimal Substructure
  - The solution to the original problem can be easily computed from the solutions to the subproblems.
- Overlapping Subproblems
  - At most polynomial subproblems to solve.
- Subproblem Ordering
  - There exists a natural order to solve subproblems without conflict.



$$F_i = \begin{cases} 0 & \text{i} = 0 \\ 1 & \text{i} = 1 \\ F_{i-1} + F_{i-2} & \text{i} \geq 2 \end{cases}$$

Subset Sum & Subproblems

Let OPT(i, W') be the optimal with only first i items and bound of W'.

If i = 0 then

OPT(i,W') = 0

If $W' > w_i$ then

OPT(i,W') = max{w_i + OPT(i-1,W'- w_i), OPT(i-1,W')}

Otherwise

OPT(i,W') = OPT(i-1,W')

Knapsack

- **Input**: A list of n items {1,…,n} and a bound W.
  - Each item i has a non-negative weight w_i.
  - Each item i has a non-negative value v_i.
- **Goal**: Find the max-value subset S such that $\sum_{i \in S} w_i \leq W$.

Knapsack & Subproblems

Let OPT(i, W') be the optimal with only first i items and bound of W'. Then we can write:

If i = 0 then

OPT(i,W') = 0

If $W' > w_i$ then

OPT(i,W') = max{$\boldsymbol{v_i}$ + OPT(i-1,W'- w_i), OPT(i-1,W')}

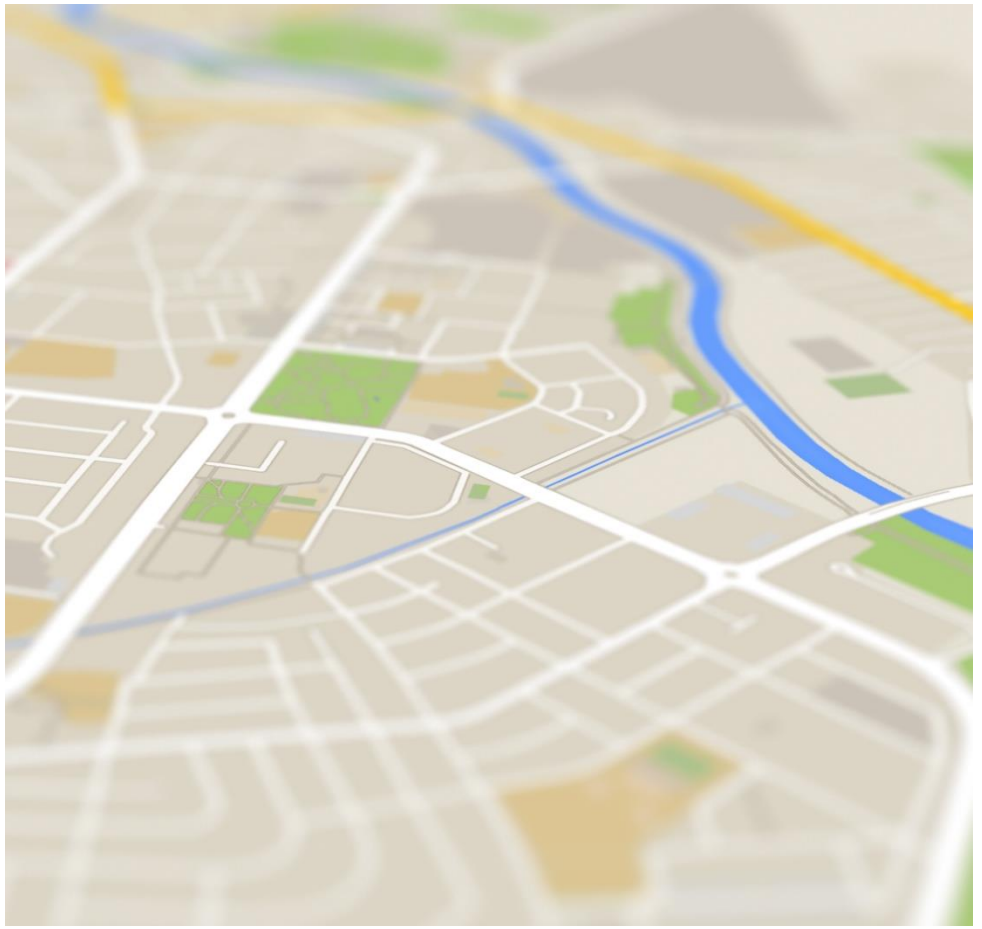
Otherwise

OPT(i,W') = OPT(i-1,W')

Shortest Path

- **Input**:
  - A directed graph $G = (V, E)$.
  - A weight for each edge $(i, j) \in E, c_{ij}$.
  - Source vertex $s \in V$.
  - Destination vertex $t \in V$.
- **Goal**: Find a path from s to t with minimum total cost:

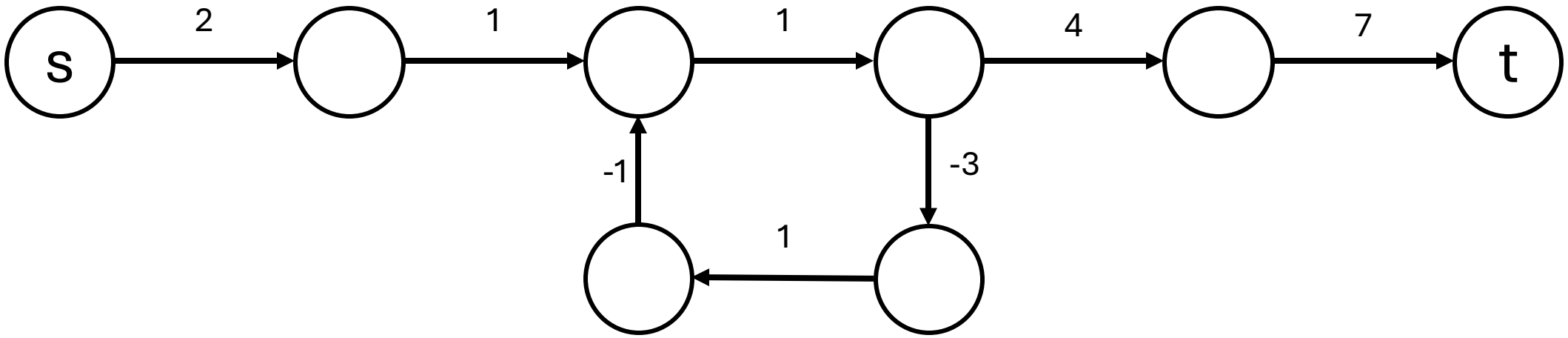
$$\text{cost}(P) = \sum_{ij \in P} c_{ij}$$



Negative Cycles

- A **negative cycle** in G is a cycle C such that

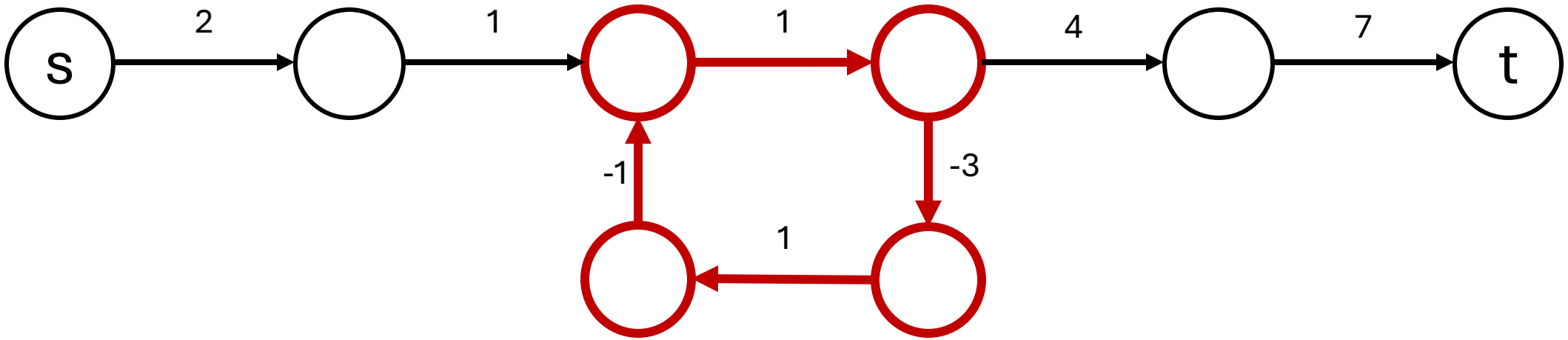
$$\text{cost}(\text{C}) = \sum_{\text{ij} \in \text{P}} c_{\text{ij}} < 0.$$



Negative Cycles

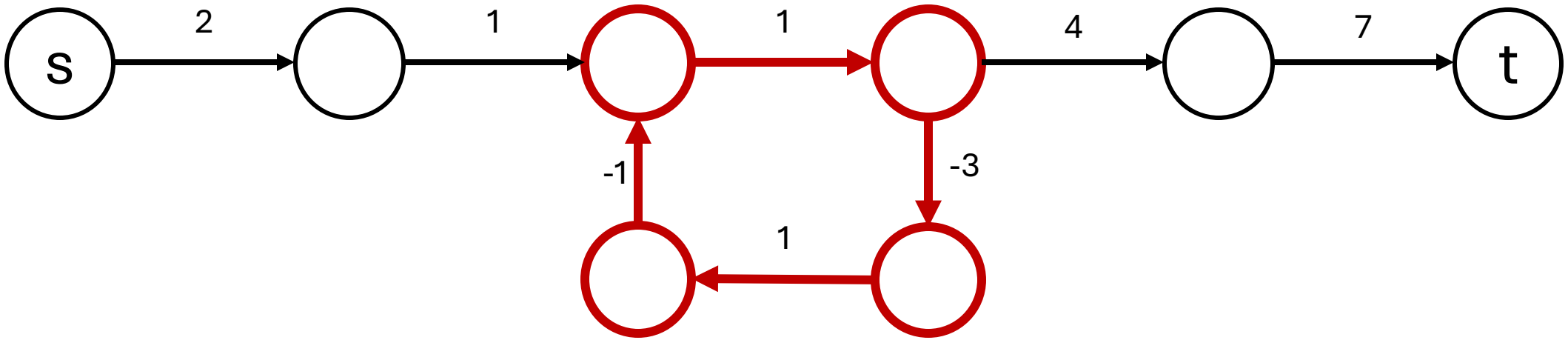
- A **negative cycle** in G is a cycle C such that

$$\text{cost}(\text{C}) = \sum_{\text{ij} \in \text{P}} c_{\text{ij}} < 0.$$



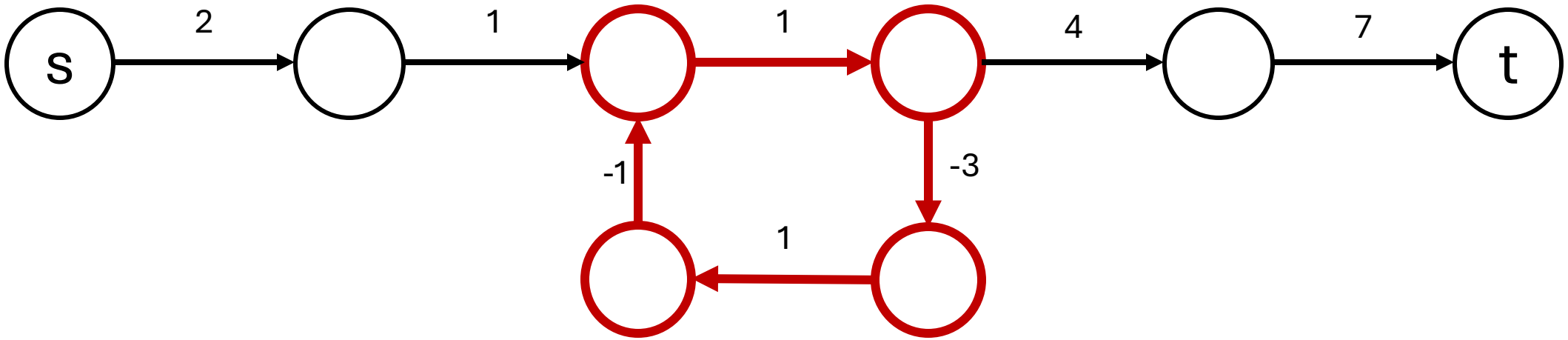
Negative Cycles

- **Observation**: If a graph has a negative cycle, then there may be no shortest path from s to t.



Negative Cycles

- **Claim**: If a graph has a no negative cycle, then there exists a shortest path from s to t is a simple path.
  - Such a shortest path will use at most $n - 1$ edges.

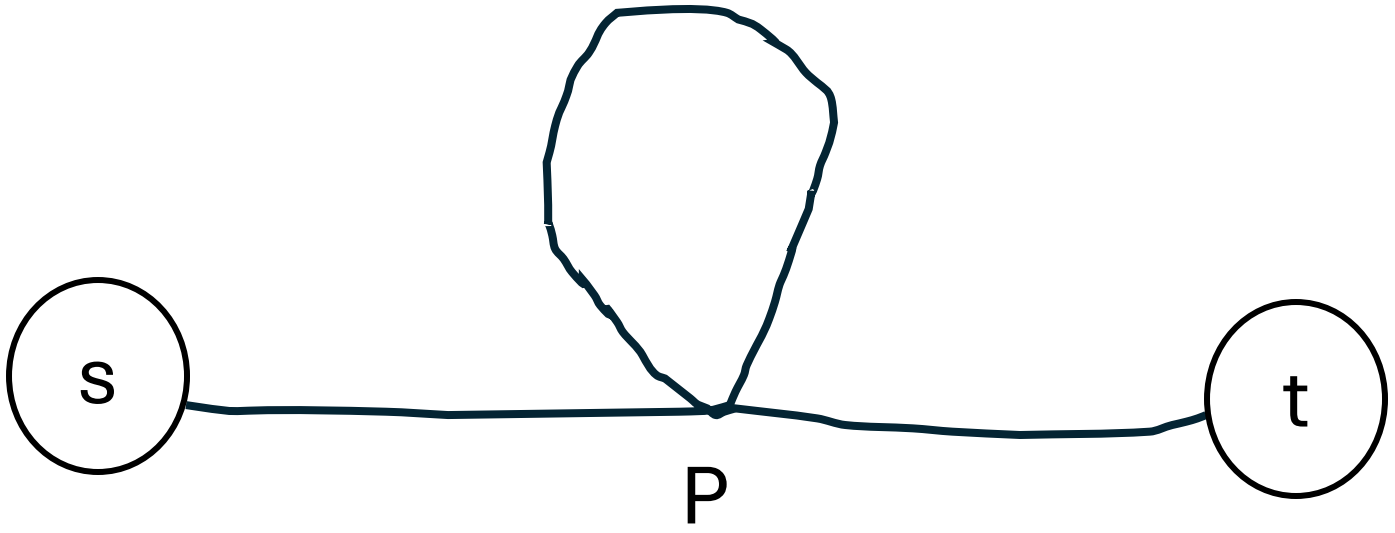


Negative Cycles

- **Claim**: If a graph has a no negative cycle, then the shortest path from s to t is a simple path.
- **Proof:**
  - Since every cycle has nonnegative cost, the shortest path P from s to t with the fewest number of edges doesn't repeat a vertex v.
  - If it did repeat a vertex, then we could remove the portion of P between consecutive visits to v, resulting in a path of no greater cost and strictly fewer edges.

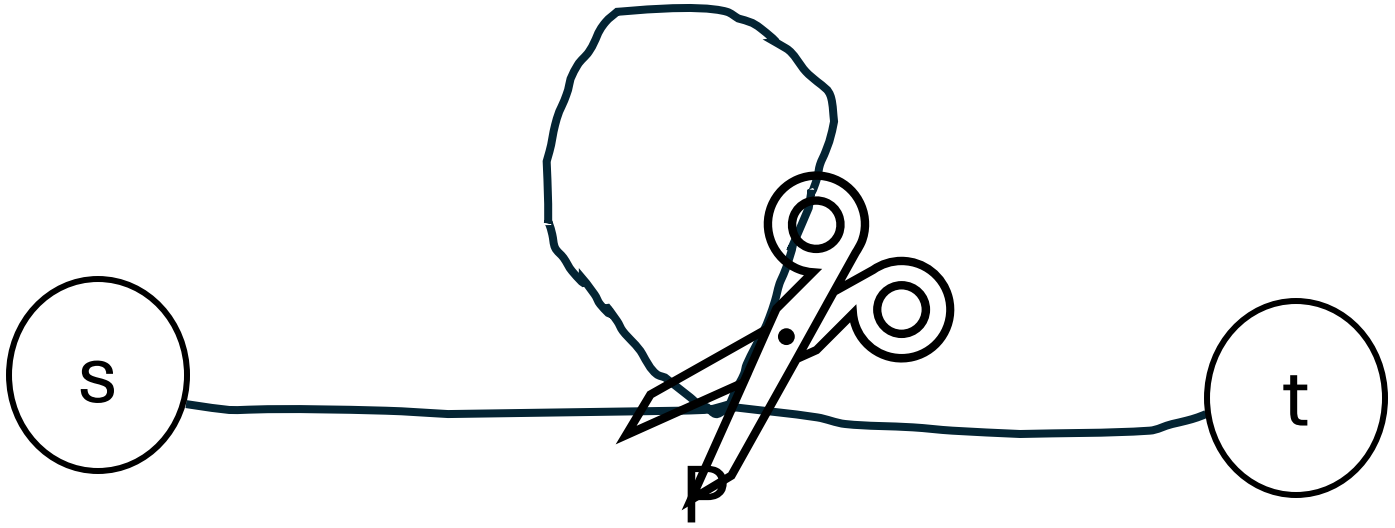
Negative Cycles

- **Claim**: If a graph has a no negative cycle, then the shortest path from s to t is a simple path.
- **Proof:**



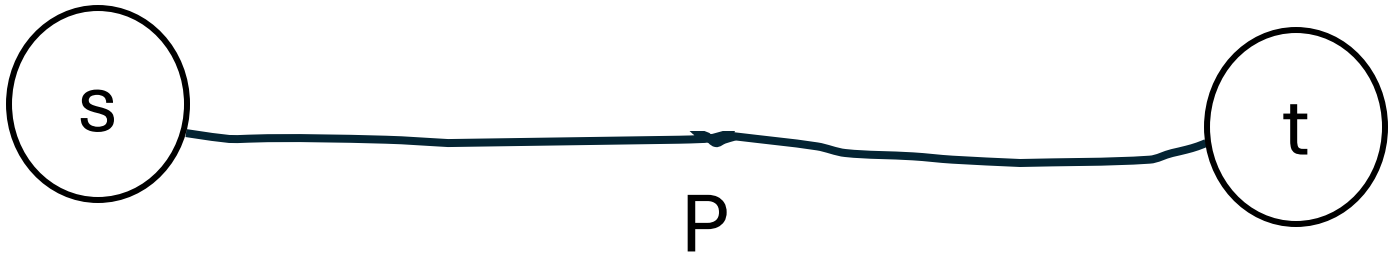
Negative Cycles

- **Claim**: If a graph has a no negative cycle, then the shortest path from s to t is a simple path.
- **Proof:**



Negative Cycles

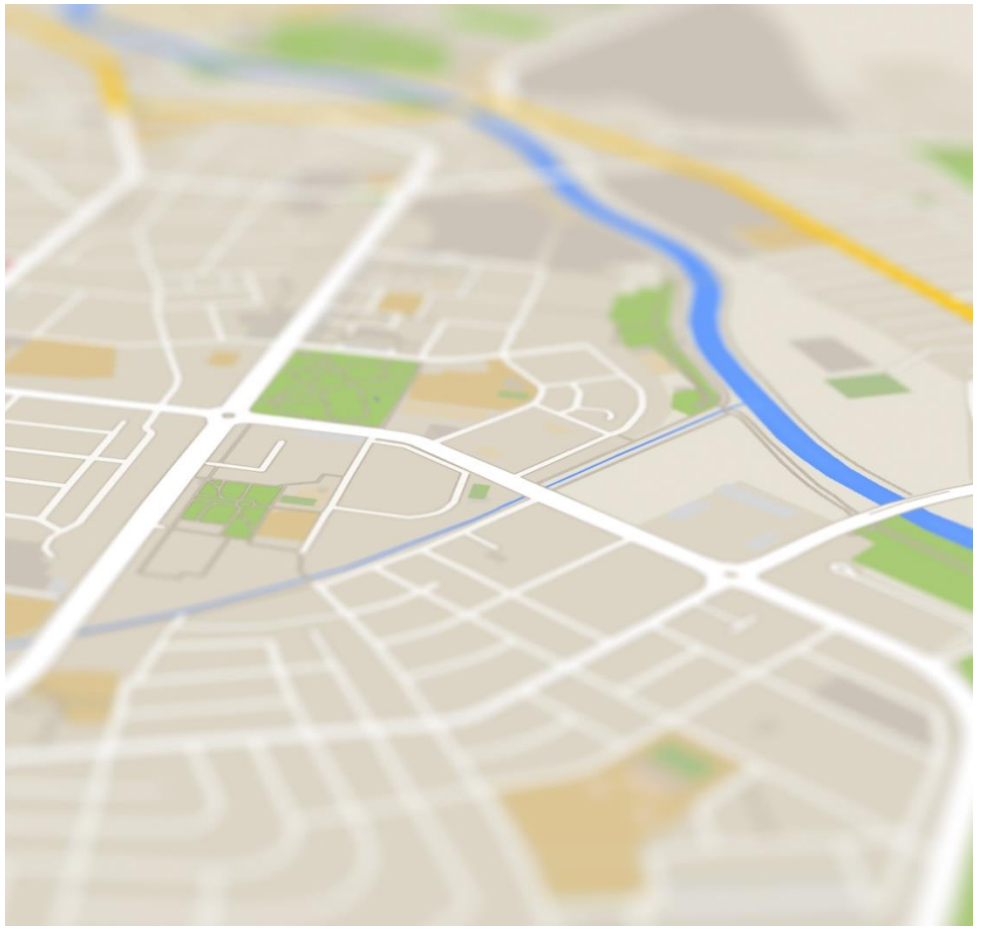
- **Claim**: If a graph has a no negative cycle, then the shortest path from s to t is a simple path.
- **Proof:**



Shortest Path

- **Input**:
  - A directed graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$.
  - A weight for each edge $(\mathrm{i}, \mathrm{j}) \in \mathrm{E}, c_{\mathrm{ij}}$.
  - Source vertex $s \in \mathrm{V}$.
  - Destination vertex $\mathrm{t} \in \mathrm{V}$.
  - **No negative cycles!**
- **Goal**: Find a path from s to t with minimum total cost:

$$\mathrm{cost}(P) = \sum_{\mathrm{ij} \in \mathrm{P}} c_{\mathrm{ij}}$$



Shortest Path Subproblem

- **Input**:
  - A directed graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$.
  - A weight for each edge $(\mathrm{i}, \mathrm{j}) \in \mathrm{E}, c_{ij}$.
  - Source vertex $s \in \mathrm{V}$.
  - Destination vertex $\mathrm{t} \in \mathrm{V}$.
  - **No negative cycles!**
- **Goal**: Find a path from s to t with minimum total cost:

$$\mathrm{cost}(P) = \sum_{\mathrm{ij} \in \mathrm{P}} c_{ij}$$

Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.
- **Question**: What is the answer the our overall problem?
- **Question**: What are the base/easy cases?
- **Question**: What is a nice way to write the answer as a function of subproblems?

Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- **Question**: What is the answer to our overall problem?
  - **Answer**: $OPT(n-1, s)$
- **Question**: What are the base/easy cases?

- **Question**: What is a nice way to write the answer as a function of subproblems?

Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- **Question**: What is the answer to our overall problem?
  - **Answer**: $OPT(n-1, s)$
- **Question**: What are the base/easy cases?
  - **Answer**: $OPT(0, \mathrm{u}) = \infty$ if $u \neq \mathrm{t}$ and $OPT(0, \mathrm{t}) = 0$ otherwise.
- **Question**: What is a nice way to write the answer as a function of subproblems?

Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- **Question**: What is the answer to our overall problem?
  - **Answer**: $OPT(n-1, s)$
- **Question**: What are the base/easy cases?
  - **Answer**: $OPT(0, \mathrm{u}) = \infty$ if $u \neq \mathrm{t}$ and $OPT(0, \mathrm{t}) = 0$ otherwise.
- **Question**: What is a nice way to write the answer as a function of subproblems?
  - **"Answer":** We should try to identify a choice as a way to break the problem into smaller subproblems.

Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- **Question**: What is a nice way to write the answer as a function of subproblems?
  - **"Answer":** We should try to identify a choice as a way to break the problem into smaller subproblems.
  - **Question**: What can we say about the shortest path from v to t if $v \neq t$?

Shortest Path Subproblem

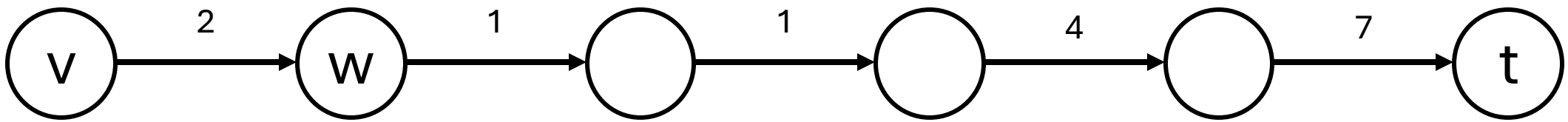
- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- **Question**: What is a nice way to write the answer as a function of subproblems?
  - **"Answer":** We should try to identify a choice as a way to break the problem into smaller subproblems.
  - **Question**: What can we say about the shortest path from v to t that uses at least one edge if $v \neq t$?
    - **Answer**: There must be an edge from v to another vertex w.

Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

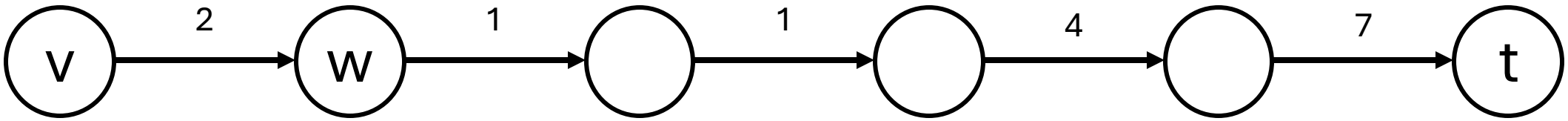
- **Observation**: There must be an edge from v to another vertex w to any shortest path using at least one edge from v to t if v is not t.



Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

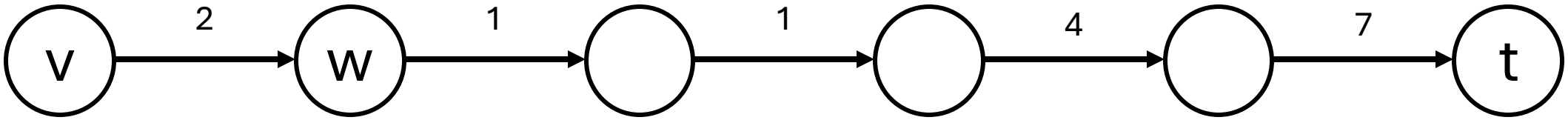
- **Question**: Is the shortest path always length n-1?



Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

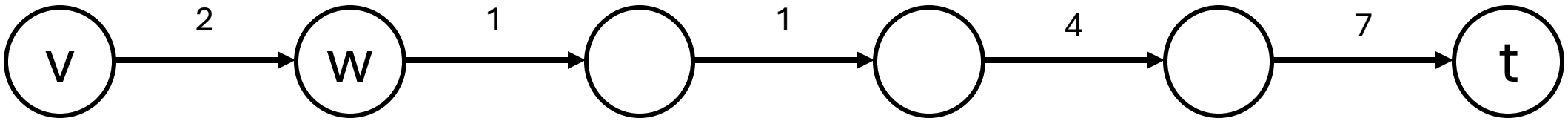
- **Question**: Is the shortest path always length n-1?
  - **Answer**: No, the path could be much shortest and not visit every vertex.



Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- **Observation**: The shortest path from v to t that uses at most i edges might be the shortest path from v to t that uses at most i-1 edges.



Shortest Path Subproblem

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.
- **Observation**: There must be an edge from v to another vertex w to any shortest path using at least one edge from v to t if v is not t.
- **Observation**: The shortest path from v to t that uses at most i edges might be the shortest path from v to t that uses at most i-1 edges.

Shortest Path Recurrence

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.
- **Observation**: There must be an edge from v to another vertex w to any shortest path using at least one edge from v to t if v is not t.
- **Observation**: The shortest path from v to t that uses at most i edges might be the shortest path from v to t that uses at most i-1 edges.
- If i > 0:

$$OPT(i, v) = min\{OPT(i-1, v), min_{\mathrm{w} \in \mathrm{V}}\{c_{v\mathrm{w}} + OPT(i-1, w)\}\}$$

Shortest Path Ordering

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- Then
  $OPT(0, \mathrm{u}) = \infty$ if $u \neq \mathrm{t}$ and $OPT(0, \mathrm{t}) = 0$.
- Moreover, if i > 0, then
  $OPT(i, v) = min\{OPT(i-1, v), min_{\mathrm{w} \in \mathrm{V}}\{c_{v\mathrm{w}} + OPT(i-1, w)\}\}$

Shortest Path Ordering

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- Then
  $OPT(0, \mathrm{u}) = \infty$ if $u \neq \mathrm{t}$ and $OPT(0, \mathrm{t}) = 0$.
- Moreover, if i > 0, then
  $OPT(i, v) = min\{OPT(i-1, v), min_{\mathrm{w} \in \mathrm{V}}\{c_{v\mathrm{w}} + OPT(i-1, w)\}\}$

- **Observation**: To solve $OPT(i, v)$ we need to know $OPT(i-1, \mathrm{w})$ for "all" w.

Shortest Path Ordering

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- Then
  $OPT(0, \mathrm{u}) = \infty$ if $u \neq \mathrm{t}$ and $OPT(0, \mathrm{t}) = 0$.
- Moreover, if i > 0, then
  $OPT(i, v) = min\{OPT(i-1, v), min_{\mathrm{w} \in \mathrm{V}}\{c_{v\mathrm{w}} + OPT(i-1, w)\}\}$

- **Observation**: To solve $OPT(i, v)$ we need to know $OPT(i-1, \mathrm{w})$ for "all" w… **and we don't need to know anything about $\boldsymbol{OPT(j, w)}$ for $\boldsymbol{j < i-1}$ and any w.**

Shortest Path Algorithm

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length n x n
3. Set M[0,t] = ?
4. Set M[0,v] = ? for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[i,v] using ?
8. Return ?
```

Shortest Path Algorithm

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length n x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[i,v] using recurrence
8. Return M[n-1,s]
```

Shortest Path Algorithm

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length n x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[i,v] using recurrence
8. Return M[n-1,s]
```

It is easy to see this algorithm has runtime at most O(n^3).

Shortest Path Algorithm

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length n x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[i,v] using recurrence
8. Return M[n-1,s]
```

It is easy to see this algorithm has runtime at most O(n^3).

Shortest Path Algorithm

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length n x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[i,v] using recurrence
8. Return M[n-1,s]
```

It isn't too hard to see this algorithm has runtime at most O(nm) if we only consider neighbors on line 6.

Getting the Shortest Path with M

Question: How do we find the shortest path of length at most l form s to t given M?

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length n x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[i,v] using recurrence
8. Return M[n-1,s]
```

Getting the Shortest Path with M

Answer: It takes O(in) time to track back optimal choices.

```
Find-Shortest-Path(i,v,M):
1. If i == 0 (and v == t):
2.   Return [t]
3. Else if M[i,v] == M[i-1,v]:
4.   Return Find-Shortest-Path(i-1,v,M)
5. Else:
6.   Find w such that M[i,v] == c[v,w] + M[i-1,w]
7.   Return [w] ++ Find-Shortest-Path(i-1,w,M)
```

Shortest Path Better Space

- Let $OPT(i, v)$ denotes the minimum cost of a v to t path using at most i edges.

- Then
  $OPT(0, \mathrm{u}) = \infty$ if $u \neq \mathrm{t}$ and $OPT(0, \mathrm{t}) = 0$.
- Moreover, if i > 0, then
  $OPT(i, v) = min\{OPT(i-1, v), min_{\mathrm{w} \in \mathrm{V}}\{c_{v\mathrm{w}} + OPT(i-1, w)\}\}$

- **Observation**: To solve $OPT(i, v)$ we need to know $OPT(i-1, \mathrm{w})$ for "all" w… **and we don't need to know anything about $\boldsymbol{OPT(j, w)}$ for $\boldsymbol{j < i-1}$ and any w.**

Shortest Path Better Space

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length 2 x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1:
6.   For v in V:
7.     Compute M[1,v] using recurrence
8.   M[0,v] = M[1,v] for all v.
9. Return M[0,s]
```

Shortest Path Better Space

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length 2 x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1: <- Still doing this n-1 times
6.   For v in V:
7.     Compute M[1,v] using recurrence
8.   M[0,v] = M[1,v] for all v.
9. Return M[0,s]
```

Bellman-Ford Algorithm

- Runs in time O(n(n+m)) time and uses at most O(n) space.

```
Shortest-Path(G,s,t):
1. N = number of nodes in G
2. 2D Array M of length 2 x n
3. Set M[0,t] = 0
4. M[0,v] = INF for all v not t
5. For i = 1,...,n-1: <- Still doing this n-1 times
6.   For v in V:
7.     Compute M[1,v] using recurrence
8.   M[0,v] = M[1,v] for all v.
9. Return M[0,s]
```