CS 4102: Algorithms

Lecture 21: Shortest Path Algorithms

David Wu

Fall 2019

Warm-Up

Show how to use Dijkstra's algorithm to compute a path $s \to t$ in a graph G which minimizes the product of edge weights along the path. You may assume that all edge weights are greater than or equal to 1.

Recall log rules: $\log ab = \log a + \log b$

Warm-Up

Show how to use Dijkstra's algorithm to compute a path $s \to t$ in a graph G which minimizes the product of edge weights along the path. You may assume that all edge weights are greater than or equal to 1.

- Construct a graph $G' = (V, E)$ where the weight of each edge $w'(e)$ in G' is $\log w(e)$ and run Dijkstra's on G'
- The multiplicative cost of a path $(v_0, v_1, \dots, v_t)$ in G is $\prod_{i=1}^{t} w(v_{i-1}, v_i)$
- Since all edge weights are positive, minimizing this quantity is equivalent to minimizing $\log \prod_{i=1}^{t} w(v_{i-1}, v_i) = \sum_{i=1}^{t} \log w(v_{i-1}, v_i)$
- This coincides with the minimization objective of Dijkstra on G' (all weights in G' are non-negative since edge weights in G are ≥ 1)

Today's Keywords

Graphs

Shortest paths algorithms

Bellman-Ford

Floyd-Warshall

Dynamic programming algorithms

CLRS Readings: Chapter 22, 23, 24

Homework

HW7 due **Thursday, November 14, 11pm**

- Graph algorithms
- Written (use LaTeX!) – Submit both **zip** and **pdf** (two separate attachments)!

HW10B due **Thursday, November 14, 11pm**

- No late submissions allowed (no exceptions)

HW8 out **Thursday, November 14**, due **Thursday, November 21, 11pm**

- Programming assignment (Python or Java)
- Graph algorithms

Currency Exchanges and Arbitrage

Currency code ▲▼	Currency name ▲▼	Units per USD	USD per Unit
USD	US Dollar	1.0000000000	1.0000000000
EUR	Euro	0.8783121137	1.1385474303
GBP	British Pound	0.6956087704	1.4375896950
INR	Indian Rupee	66.1909310706	0.0151078098
AUD	Australian Dollar	1.3050318080	0.7662648480
CAD	Canadian Dollar	1.2997506294	0.7693783541
SGD	Singapore Dollar	1.3478961522	0.7418969172
CHF	Swiss Franc	0.9590451582	1.0427037678
MYR	Malaysian Ringgit	3.8700000000	0.2583979328
JPY	Japanese Yen	112.5375383115	0.0088859239
CNY	Chinese Yuan Renminbi	6.4492409303	0.1550570076
NZD	New Zealand Dollar	1.4480018872	0.6906068347
THB	Thai Baht	35.1005319022	0.0284895968
HUF	Hungarian Forint	275.7012427385	0.0036271146
AED	Emirati Dirham	3.6730000000	0.2722570106

1 Dollar = 0.8783121137 Euro

1 Dollar = 3.87 Ringgit

Conversion rates starting from USD

Currency Exchanges and Arbitrage

Currency code ▲▼	Currency name ▲▼	Units per EUR	EUR per Unit
USD	US Dollar	1.1386632306	0.8782227907
EUR	Euro	1.0000000000	1.0000000000
GBP	British Pound	0.7921136388	1.2624451227
INR	Indian Rupee	75.3658843112	0.0132686030
AUD	Australian Dollar	1.4859561878	0.6729673514
CAD	Canadian Dollar	1.4796754127	0.6758238945
SGD	Singapore Dollar	1.5347639238	0.6515660060
CHF	Swiss Franc	1.0917416715	0.9159676012
MYR	Malaysian Ringgit	4.4140052400	0.2265516114
JPY	Japanese Yen	128.1388820287	0.0078040325
CNY	Chinese Yuan Renminbi	7.3411003512	0.1362193612
NZD	New Zealand Dollar	1.6484648003	0.6066250246
THB	Thai Baht	39.9627318192	0.0250233143
HUF	Hungarian Forint	313.9042436792	0.0031856849
AED	Emirati Dirham	4.1823100458	0.2391023117

Currency code ▲▼	Currency name ▲▼	Units per AED	AED per Unit
USD	US Dollar	0.2722570106	3.6730000000
EUR	Euro	0.2391289974	4.1818433177
GBP	British Pound	0.1893997890	5.2798369266
INR	Indian Rupee	18.0207422309	0.0554916100
AUD	Australian Dollar	0.3552996418	2.8145257760
CAD	Canadian Dollar	0.3538334124	2.8261887234
SGD	Singapore Dollar	0.3669652245	2.7250538559
CHF	Swiss Franc	0.2610686193	3.8304105746
MYR	Malaysian Ringgit	1.0548325619	0.9480177576
JPY	Japanese Yen	30.6399242607	0.0326371564
CNY	Chinese Yuan Renminbi	1.7555154332	0.5696332719
NZD	New Zealand Dollar	0.3941937299	2.5368237088
THB	Thai Baht	9.5553789460	0.1046530970
HUF	Hungarian Forint	75.0637936939	0.0133220019
AED	Emirati Dirham	1.0000000000	1.0000000000

1 Dollar = 3.87 Ringgit

Arbitrage opportunity: Profit by exploiting uneven exchange rates for the same asset (e.g., currencies, stocks, bonds, etc.)

But what we go from USD → EUR → MYR?

1 Dollar = 0.8783121137 Euro

1 Euro= 4.1823100458 Dirham

1 Dirham= 1.0548325619 Ringgit

1 Dollar = 0.8783121137 * 4.1823100458 * 1.0548325619 Ringgit

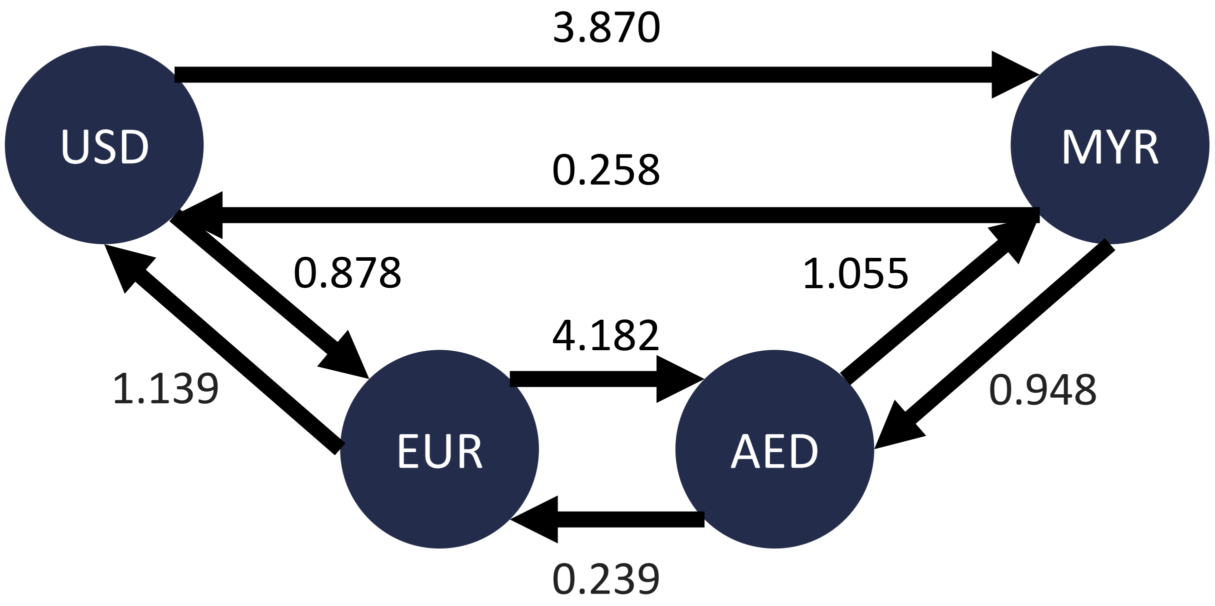
= **3.87479406049 Ringgit**

= **1.00123877526 Dollar**

Best Currency Exchange

Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates

Product of edge weights along a path from $s \rightarrow t$ gives amount of currency t that can be exchanged for 1 unit of currency s



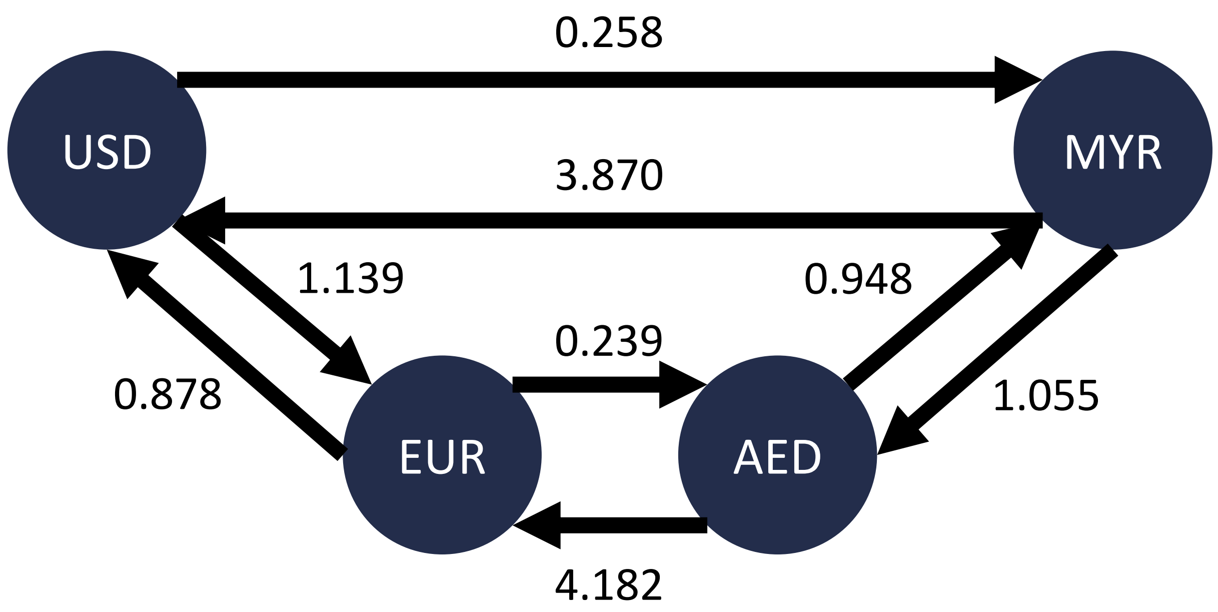
Best currency exchange:

$$\max_p \prod_{e \in p} w(e)$$

Best Currency Exchange

Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates

Product of edge weights along a path from $s \to t$ gives amount of currency t that can be exchanged for 1 unit of currency s



Best currency exchange:

$$\max_p \prod_{e \in p} w(e)$$

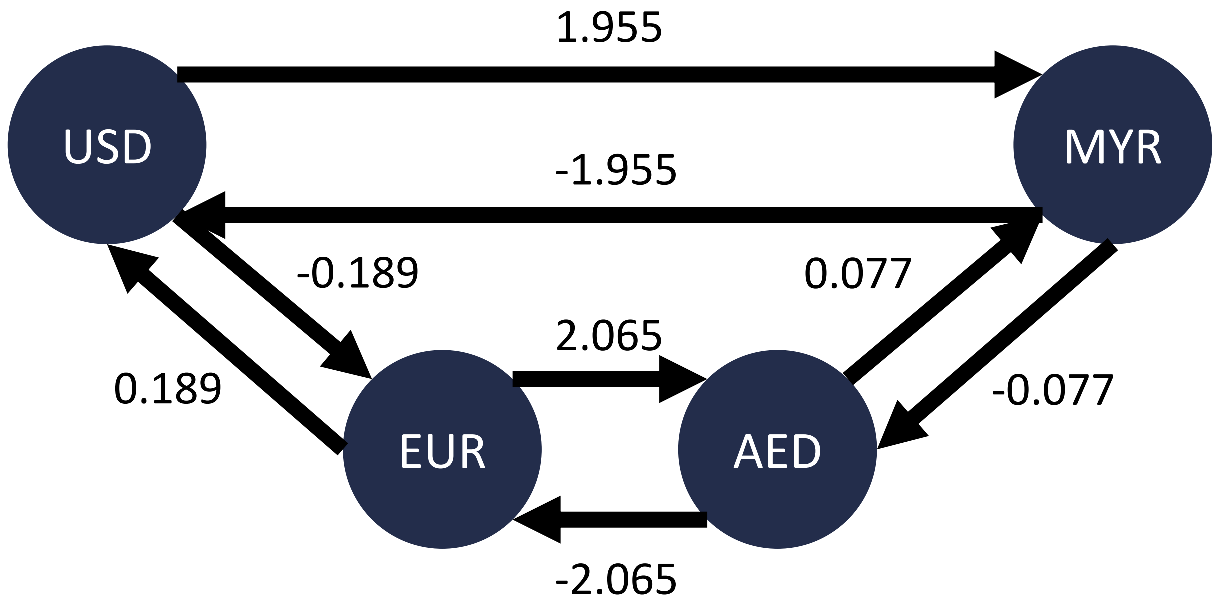
Equivalently:

$$\min_p \prod_{e \in p} \frac{1}{w(e)}$$

Best Currency Exchange

Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates

Product of edge weights along a path from $s \to t$ gives amount of currency t that can be exchanged for 1 unit of currency s



Best currency exchange:

$$\max_p \prod_{e \in p} w(e)$$

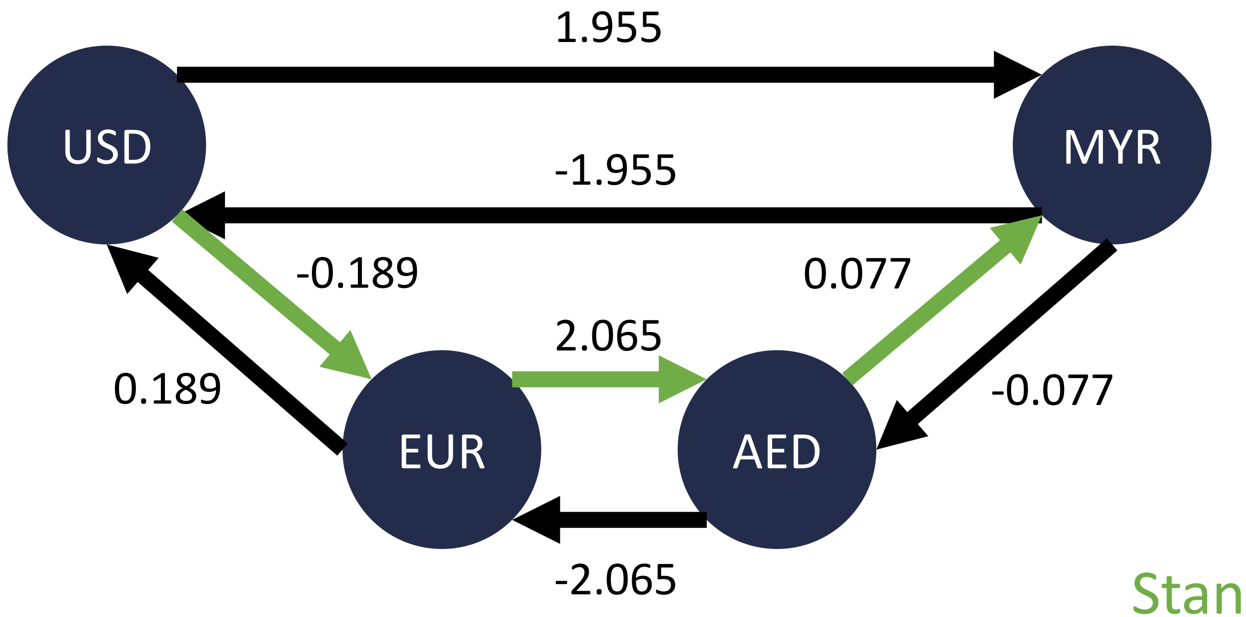
Equivalently:

$$\min_p \sum_{e \in p} -\log w(e)$$

Best Currency Exchange

Consider a directed graph where nodes correspond to currencies and edges correspond to exchange rates

Product of edge weights along a path from $s \rightarrow t$ gives amount of currency t that can be exchanged for 1 unit of currency s



Best currency exchange:

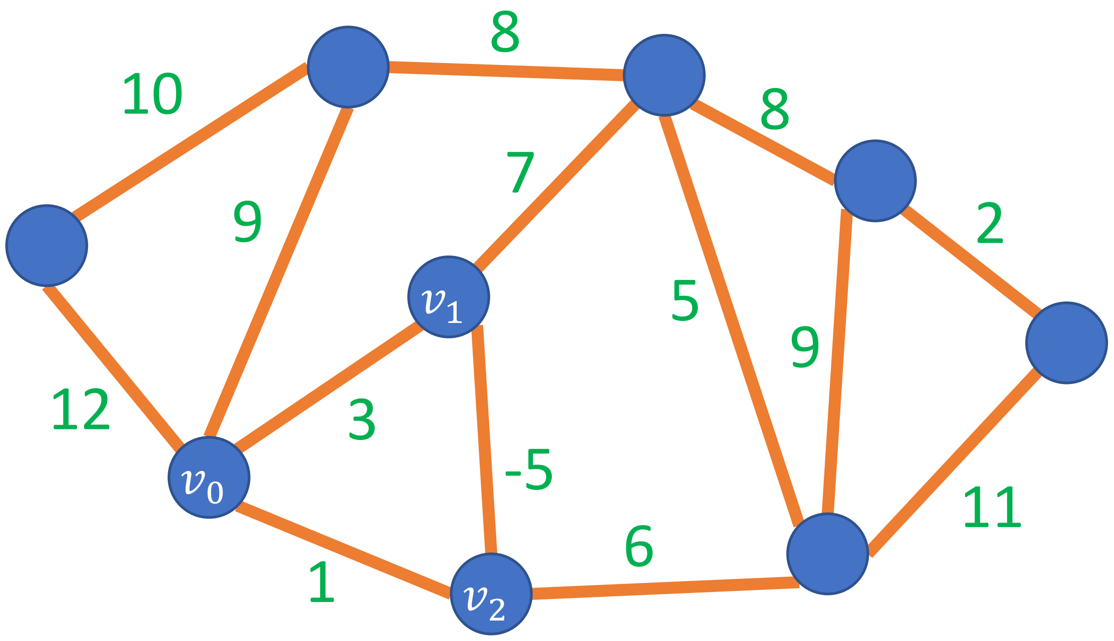
$$\max_{p} \prod_{e \in p} w(e)$$

Equivalently:

$$\min_{p} \sum_{e \in p} -\log w(e)$$

Standard shortest path problem!

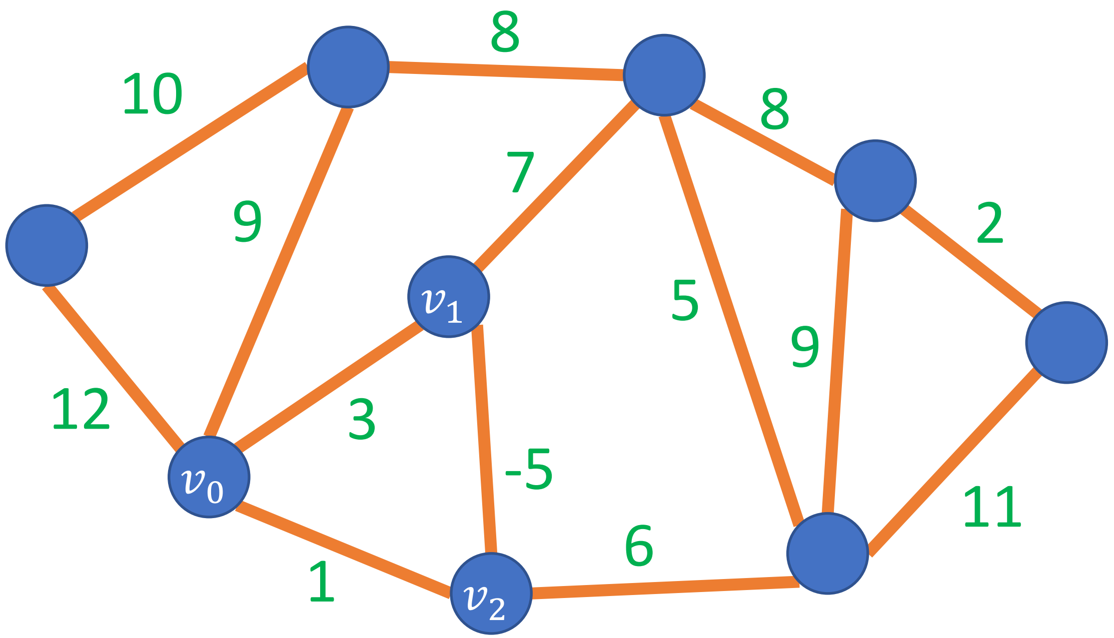
Problem with Negative Edges



If a graph has a negative-weight cycle, then there does not exist a shortest path from $s \to t$ (whenever the negative-weight cycle is reachable from s)

Proof: Suppose there a negative-weight cycle $(v \to v)$ of cost $k < 0$. Consider any path of the form $s \to v \to u \to t$. We can decrease the cost of this path by k by replacing $s \to v \to u$ with $s \to (v \to v) \to u$. This decreases the weight of the path from $s \to t$ by k. This can be repeated to make the weight of the path arbitrarily negative.

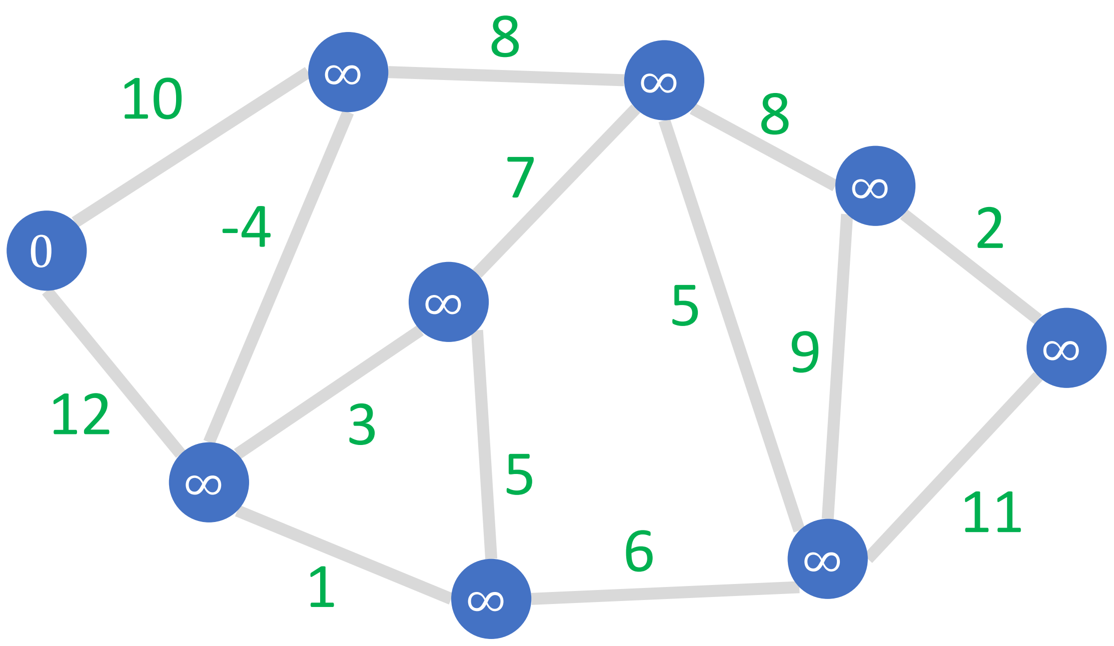
Problem with Negative Edges



If a graph has a negative-weight cycle, then there does <u>not</u> exist a shortest path from $s \rightarrow t$ (whenever the negative-weight cycle is reachable from s)

Important Note: Shortest path is still well-defined if the graph has negative-weight edges, as long as it does not have a negative-weight cycle

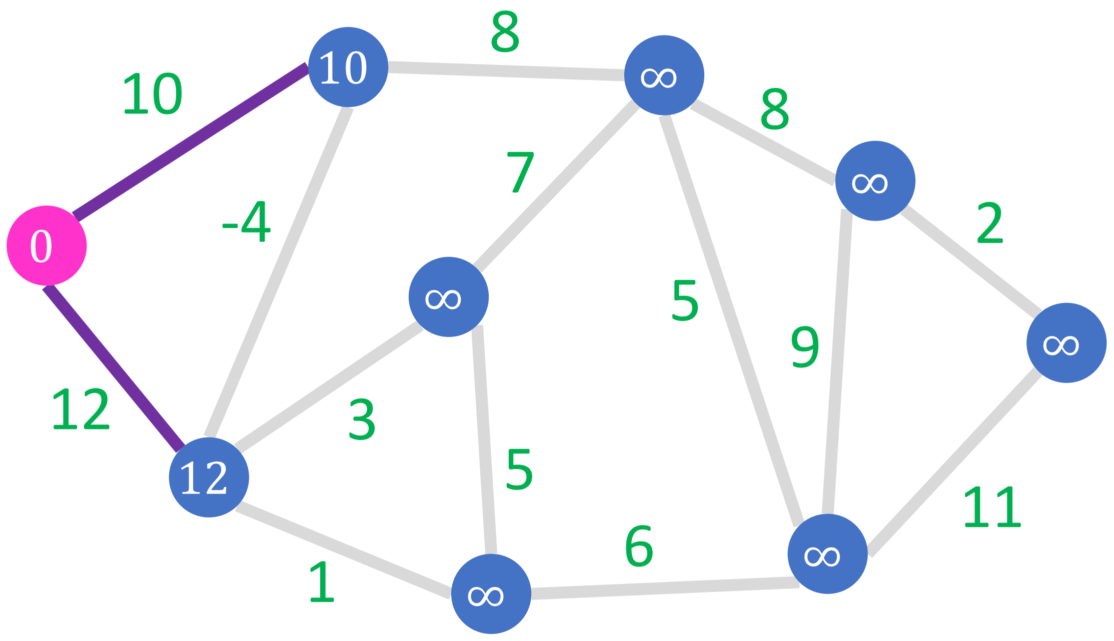
Problem with Negative Edges



Recall: Dijkstra's algorithm does not work if there are edges of negative weight

Dijkstra's algorithm is <u>greedy</u>: it constructs a shortest-path tree by always choosing the <u>current</u> closest node

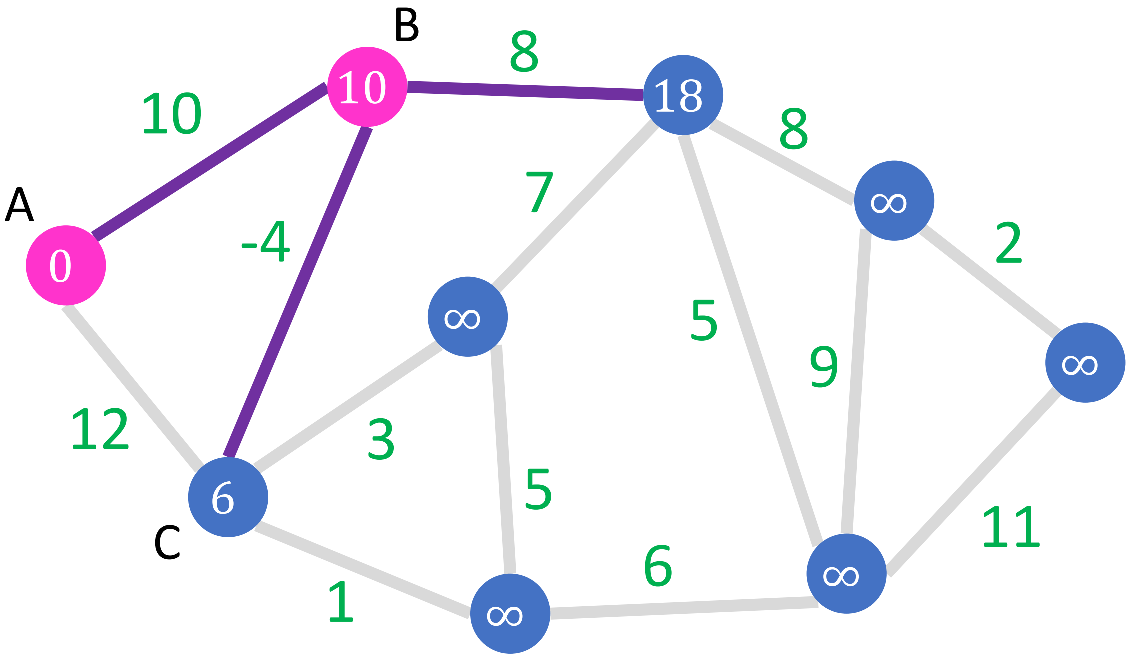
Problem with Negative Edges



Recall: Dijkstra's algorithm does not work if there are edges of negative weight

Dijkstra's algorithm is <u>greedy</u>: it constructs a shortest-path tree by always choosing the <u>current</u> closest node

Problem with Negative Edges



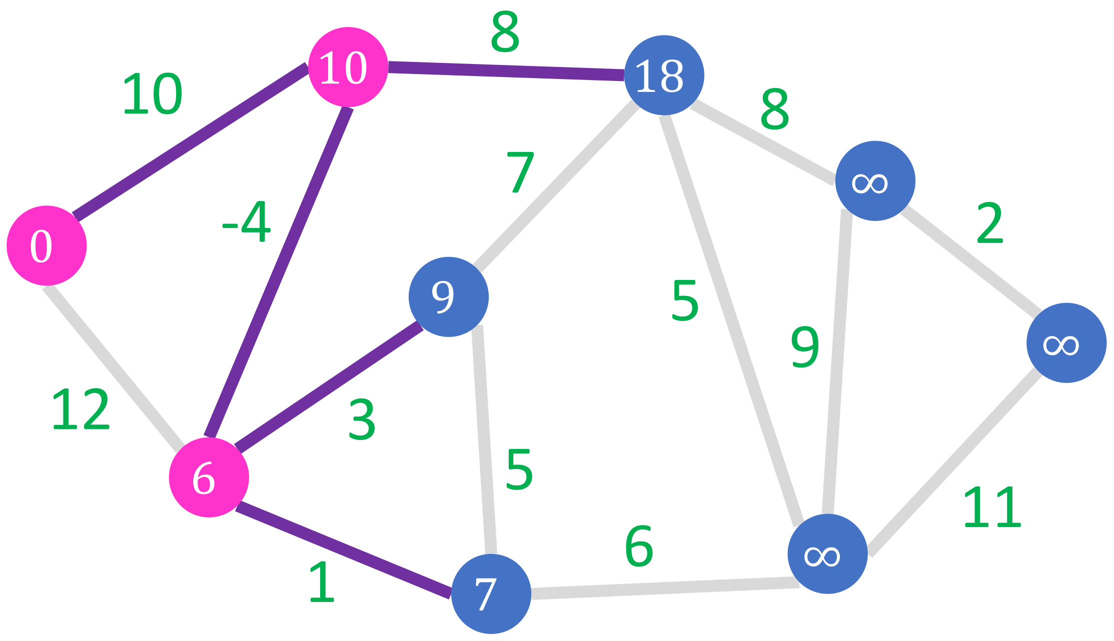
Recall: Dijkstra's algorithm does not work if there are edges of negative weight

Dijkstra's algorithm is <u>greedy</u>: it constructs a shortest-path tree by always choosing the <u>current</u> closest node

Problem: Dijkstra assumes that it has now found the shortest path to node B

- When weights are positive, then every other path must have greater weight because they require first taking a path that is longer than the current distance from $A \to B$ (e.g, $A \to C$)
- But if edge weights can be negative, the weight of later edges (e.g., $C \to B$) can offset the cost of the initial longer path – hence, the greedy heuristic is <u>suboptimal</u>

Problem with Negative Edges



Recall: Dijkstra's algorithm does not work if there are edges of negative weight

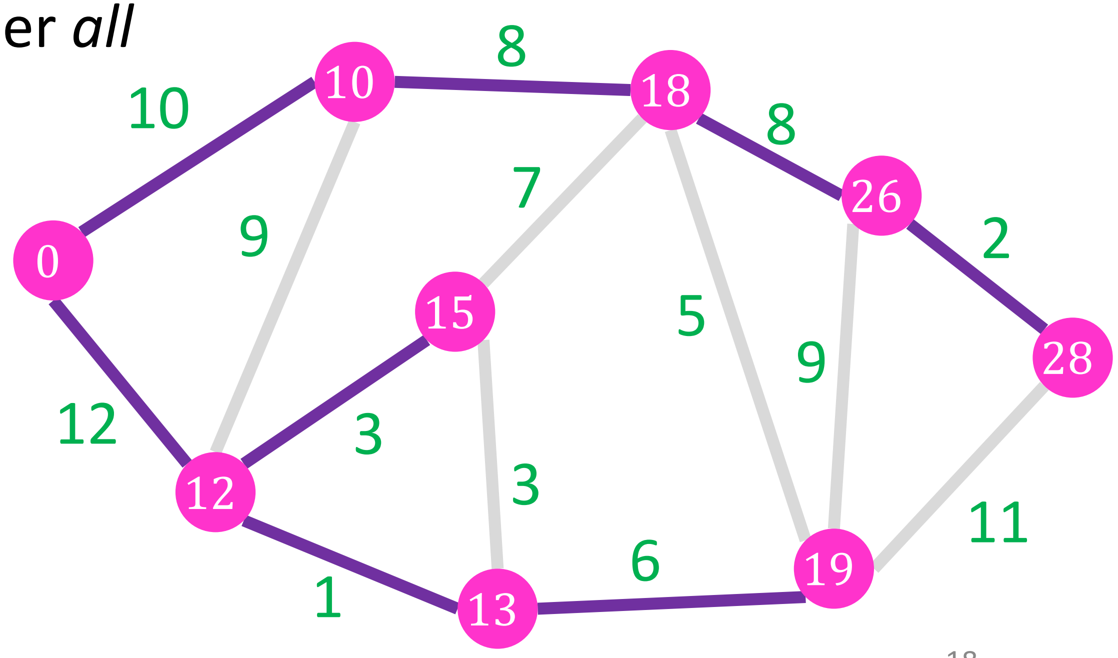
Dijkstra's algorithm is <u>greedy</u>: it constructs a shortest-path tree by always choosing the <u>current</u> closest node

Optimal Substructure of Shortest Path Trees

Recall the structure of the "shortest-path" tree from the previous lecture:

Shortest paths from a source has optimal substructure

- Greedy choice (choose the "closest" node to the source) is suboptimal with negative-weight edges
- **Idea:** use dynamic programming and consider *all* possible subproblems



Every subpath of a shortest path is itself a shortest path (optimal substructure)

Observe: shortest paths from a source forms a tree, but **not** a minimum spanning tree

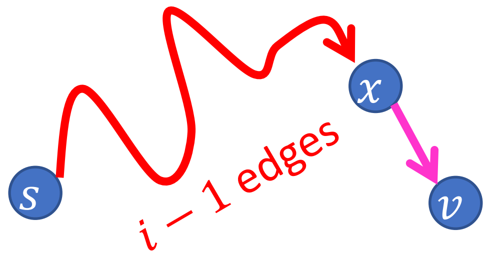
Bellman-Ford Shortest Path Algorithm

When greedy does not work… try dynamic programming!

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

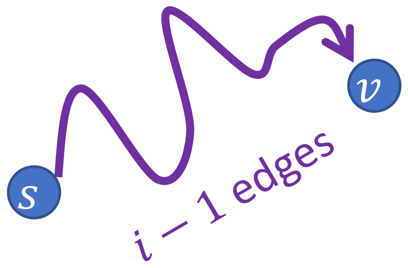
Two possibilities:

A path of $i-1$ edges from s to some node x, then edge (x, v)



OR

A path from s to v of at most $i-1$ edges



$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}\left(\text{Short}(i-1, x) + w(x, v)\right) \\ \text{Short}(i-1, v) \end{cases}$$

Maximum value of i?

Number of Hops in Shortest Path

Claim: In every graph $G = (V, E)$ that does not contain a negative-weight cycle, there exists a shortest path between any two connected nodes with at most $|V| - 1$ edges

Proof: Follows by Pigeonhole principle:

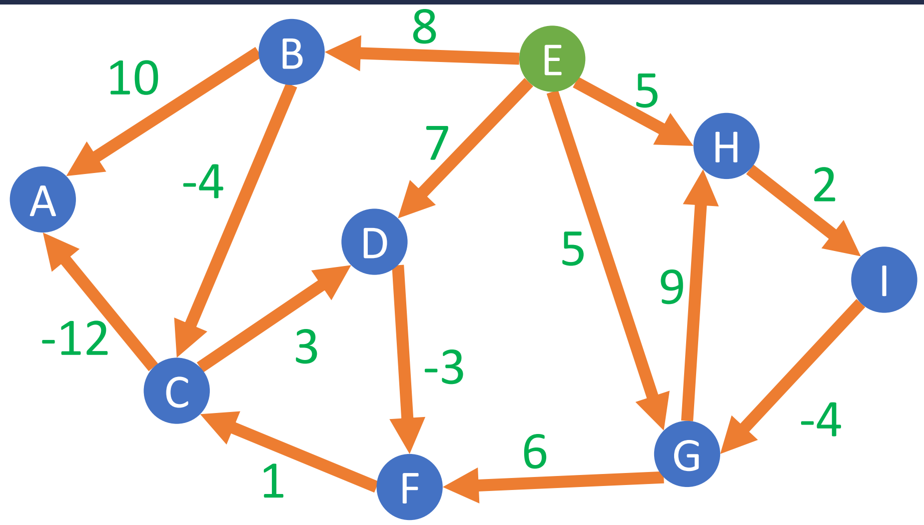
- If there is a shortest path with more than $|V| - 1$ edges, at least one node appears twice in the path, which means there is a cycle
- Since there are no negative-weight cycles in G, the weight of the cycle is ≥ 0
- Thus, removing the cycle will yield a path of equal or smaller weight
- Each cycle can be removed in this manner to obtain a path satisfying the desired property

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0									
1									
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

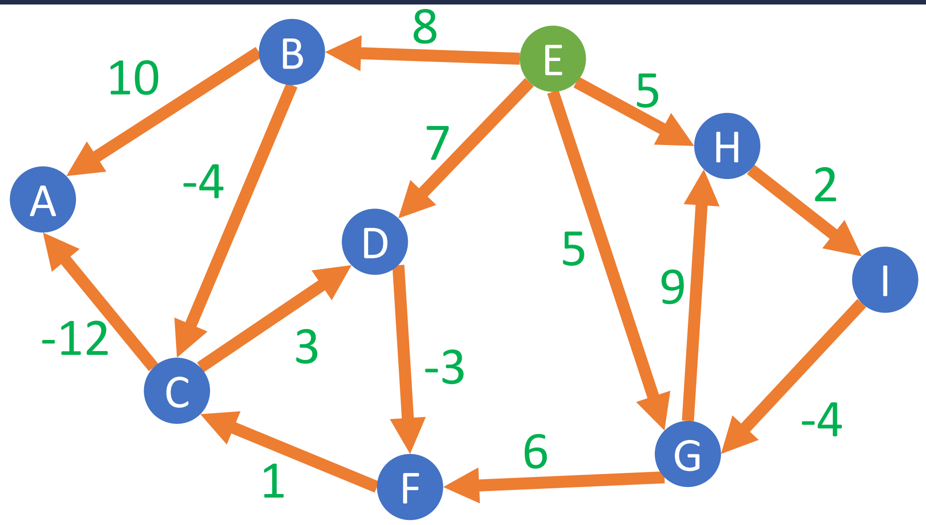
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\,\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1									
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

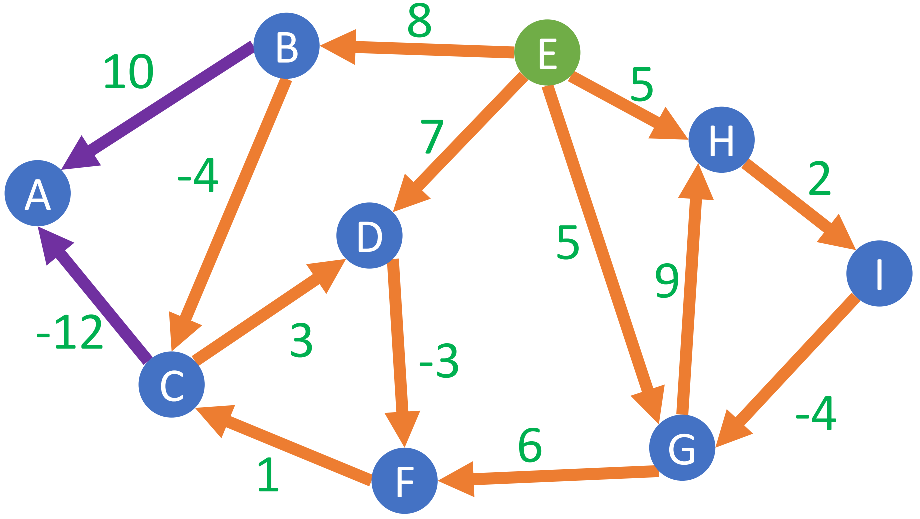
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} \left(\text{Short}(i-1, x) + w(x, v) \right) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1									
2									
3									
4									
5									
6									
7									
8									



Base case:

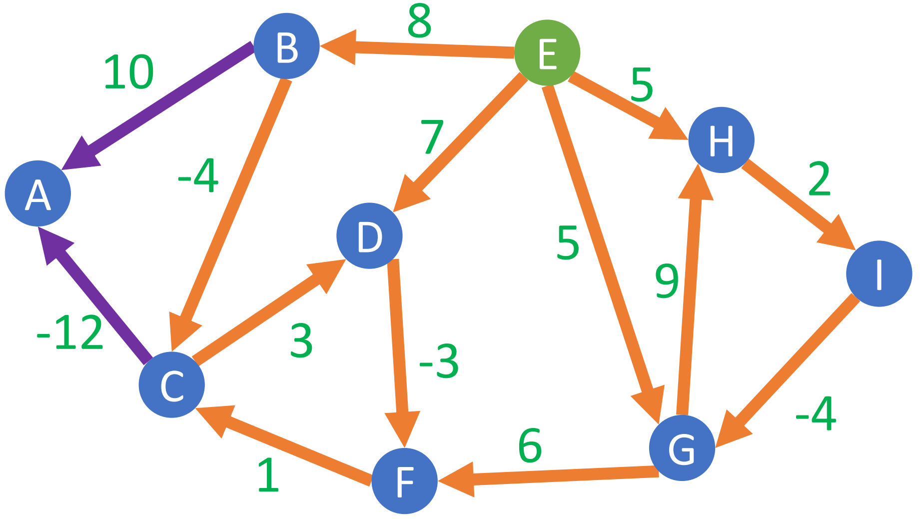
- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞								
2									
3									
4									
5									
6									
7									
8									

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

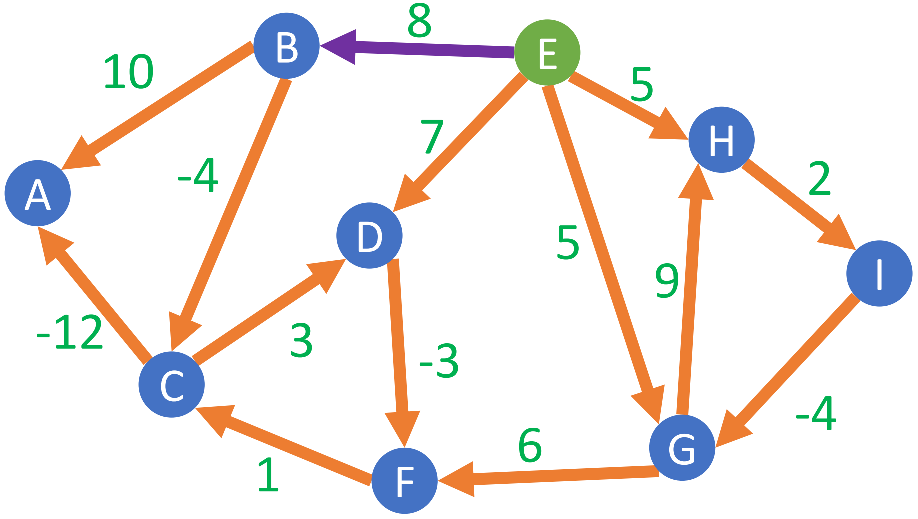
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞								
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

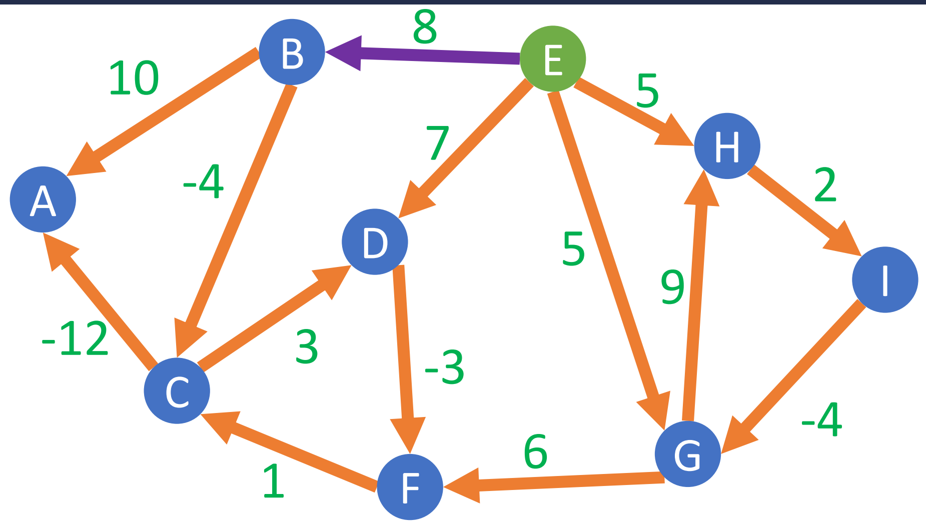
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**							
2									
3									
4									
5									
6									
7									
8									



Base case:

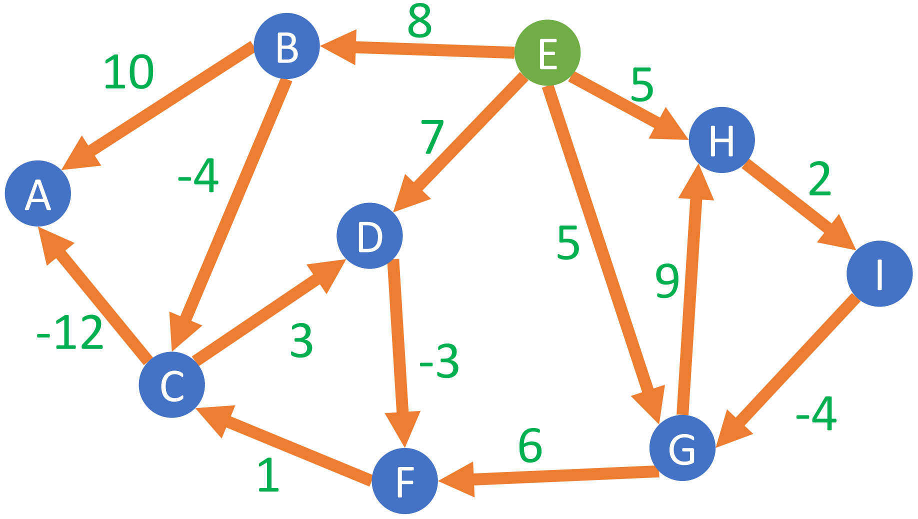
- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞						
2									
3									
4									
5									
6									
7									
8									

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

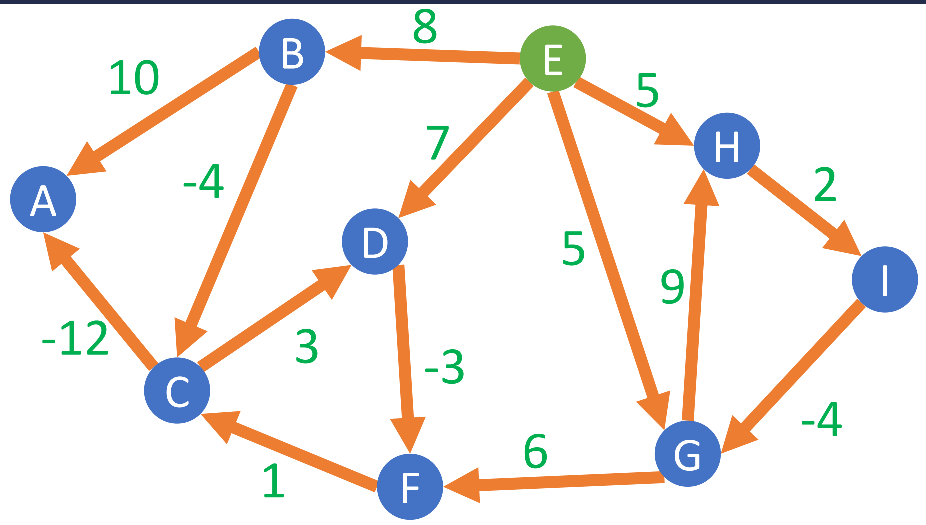
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**					
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

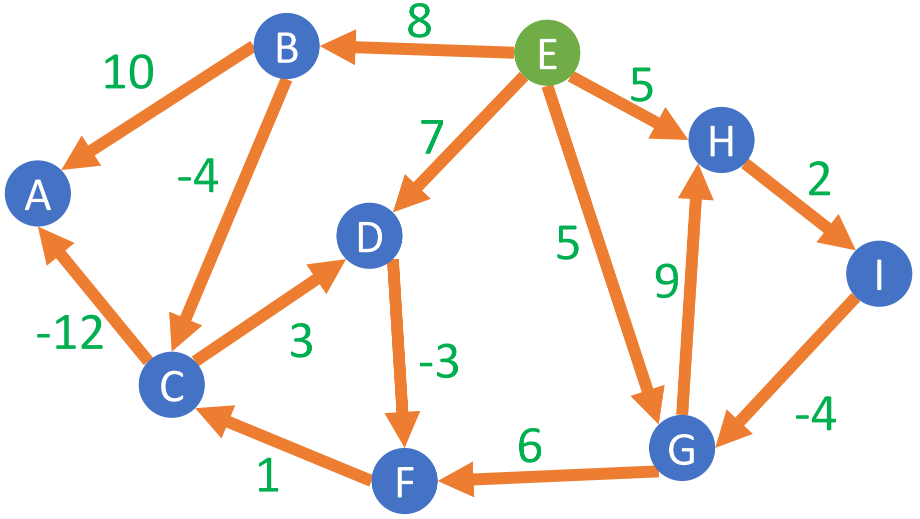
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ / $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**				
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

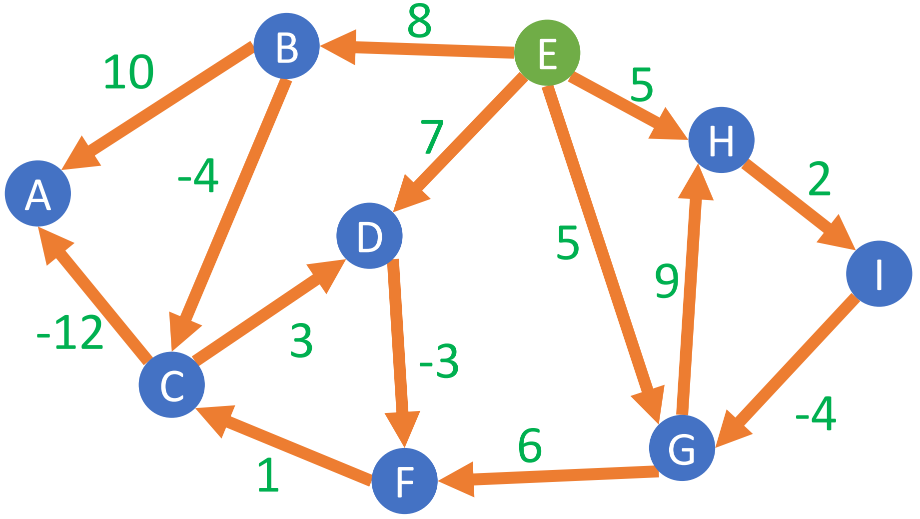
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞			
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

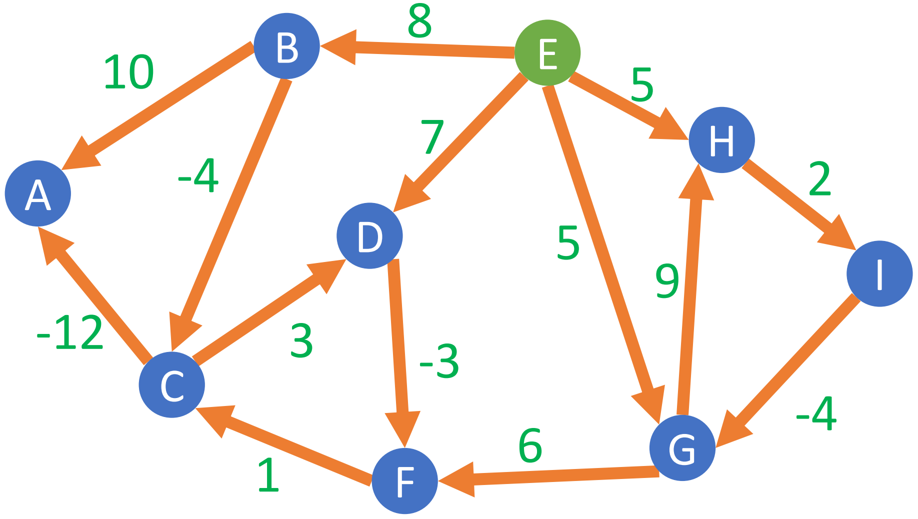
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i,v) = \min \begin{cases} \min\limits_{x \in V}\left(\text{Short}(i-1, x) + w(x,v)\right) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**		
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

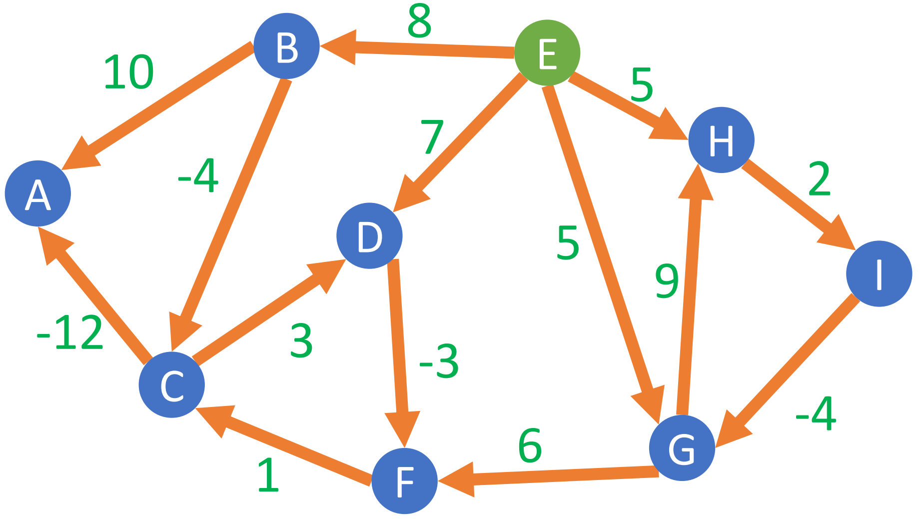
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	
2									
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

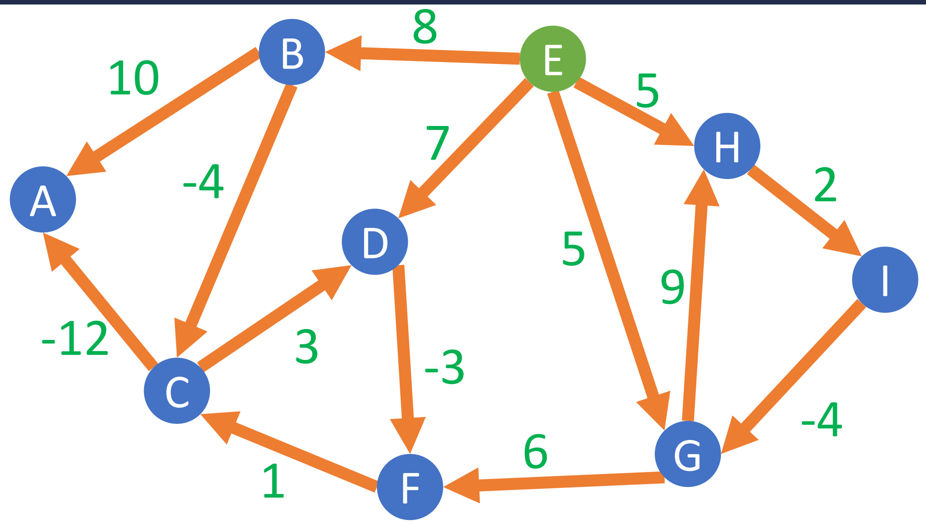
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2									
3									
4									
5									
6									
7									
8									



Base case:

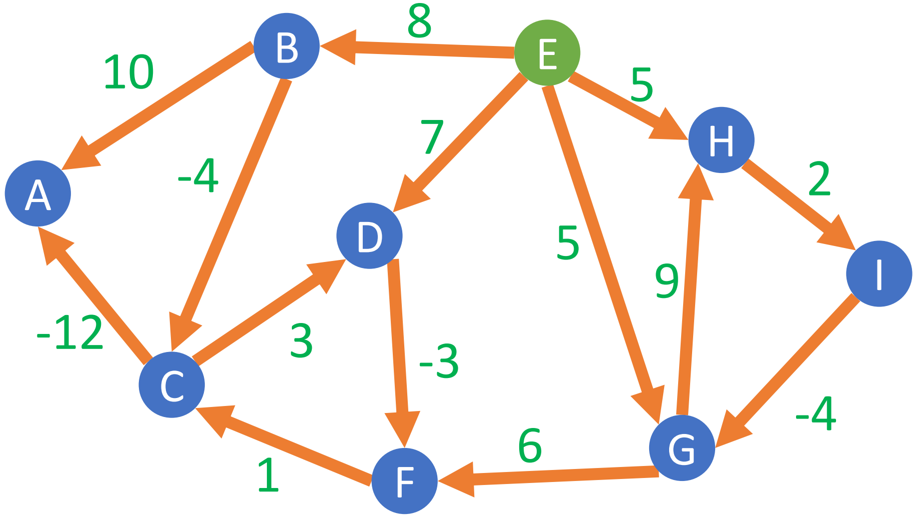
- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**4**	**5**	**5**	**7**
3									
4									
5									
6									
7									
8									

Base case:

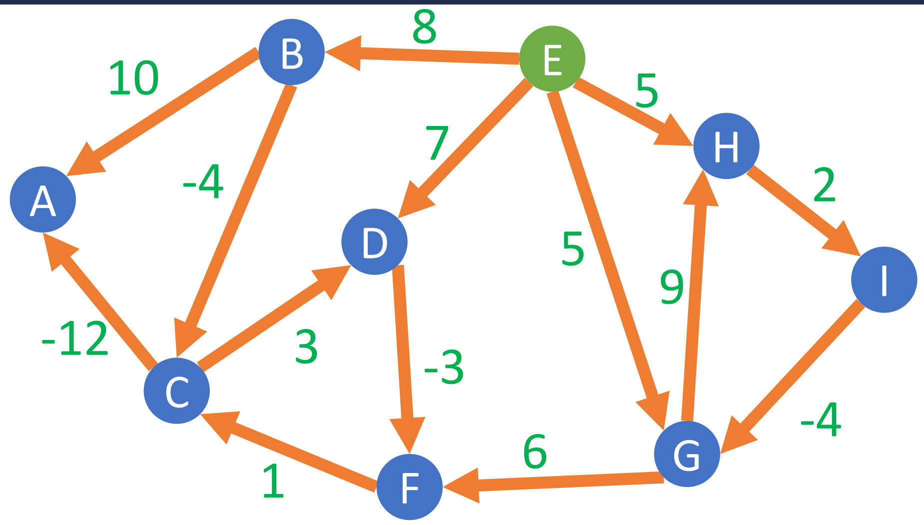
- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**4**	**5**	**5**	**7**
3	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
4									
5									
6									
7									
8									

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

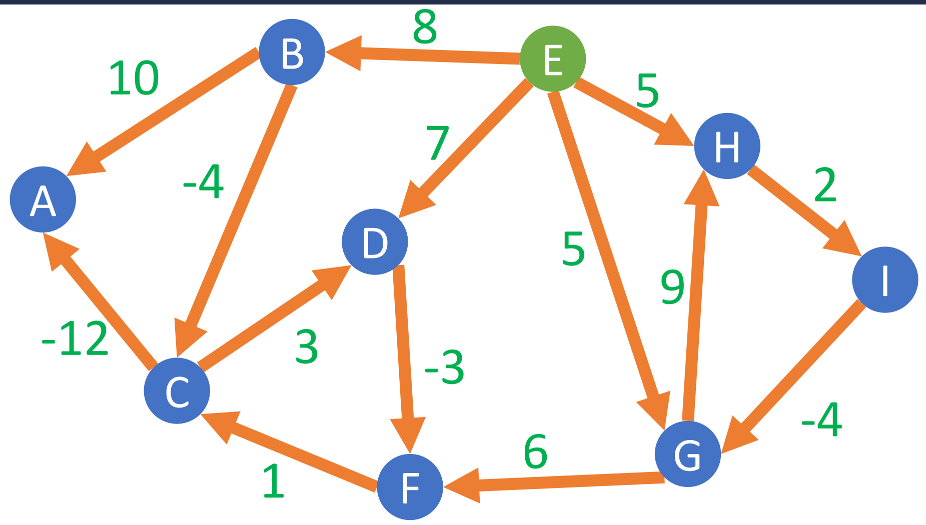
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} \left(\text{Short}(i-1, x) + w(x, v) \right) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**4**	**5**	**5**	**7**
3	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
4	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

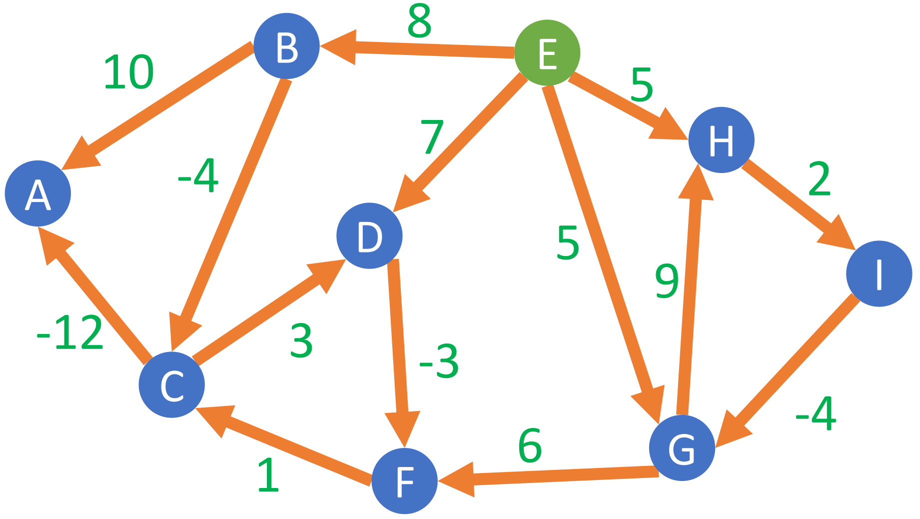
Suppose source node is E

Bellman-Ford Shortest Path Algorithm

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\,\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	0	∞	∞	∞	∞
1	∞	8	∞	7	0	∞	5	5	∞
2	18	8	4	7	0	4	5	5	7
3	-8	8	4	7	0	4	3	5	7
4	-8	8	4	7	0	4	3	5	7
5	-8	8	4	7	0	4	3	5	7
6	-8	8	4	7	0	4	3	5	7
7	-8	8	4	7	0	4	3	5	7
8	-8	8	4	7	0	4	3	5	7



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

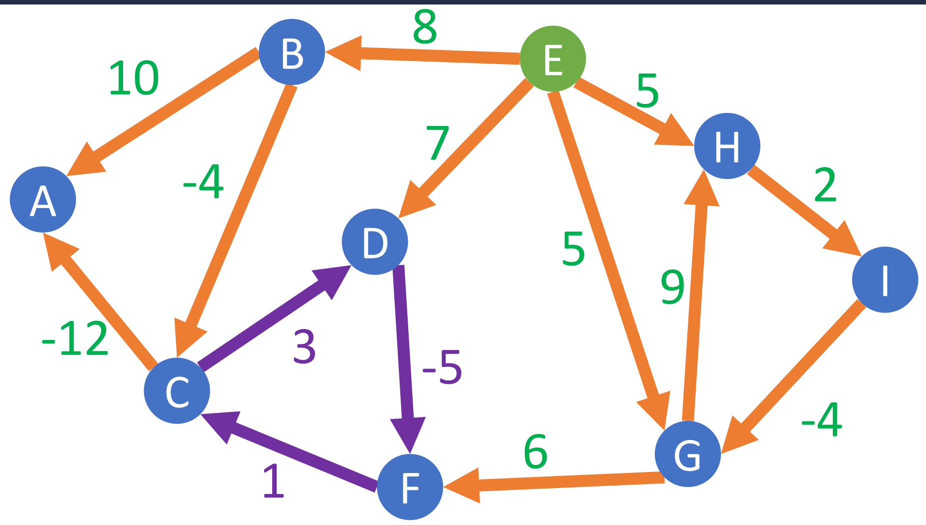
Suppose source node is E

Backtrack to reconstruct shortest path

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\,\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0									
1									
2									
3									
4									
5									
6									
7									
8									

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

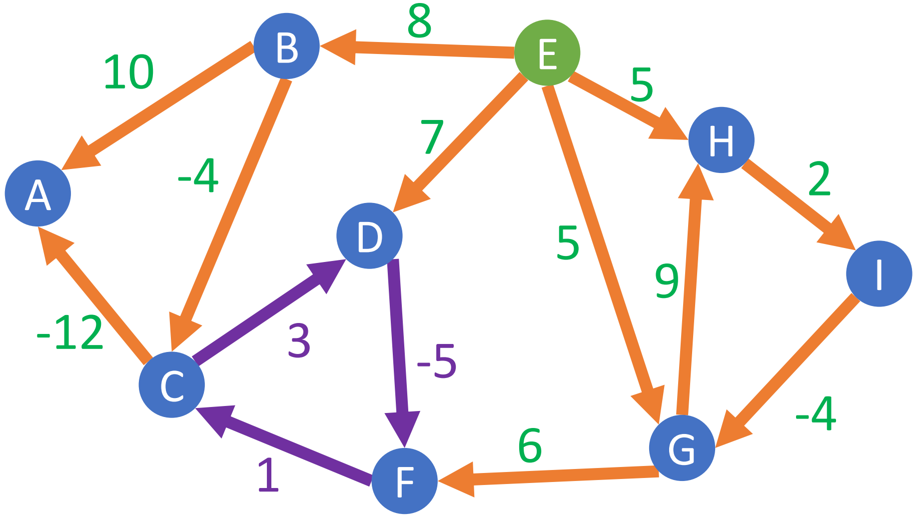
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1									
2									
3									
4									
5									
6									
7									
8									



Base case:

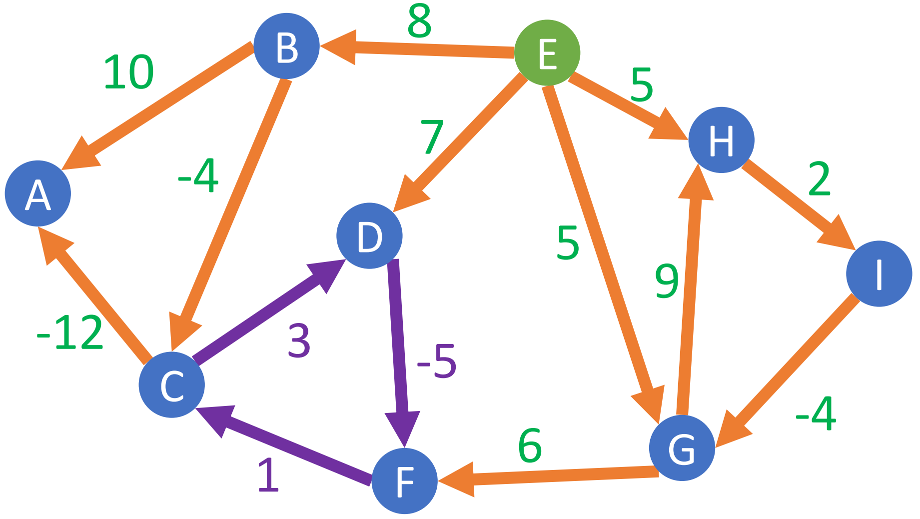
- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2									
3									
4									
5									
6									
7									
8									

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

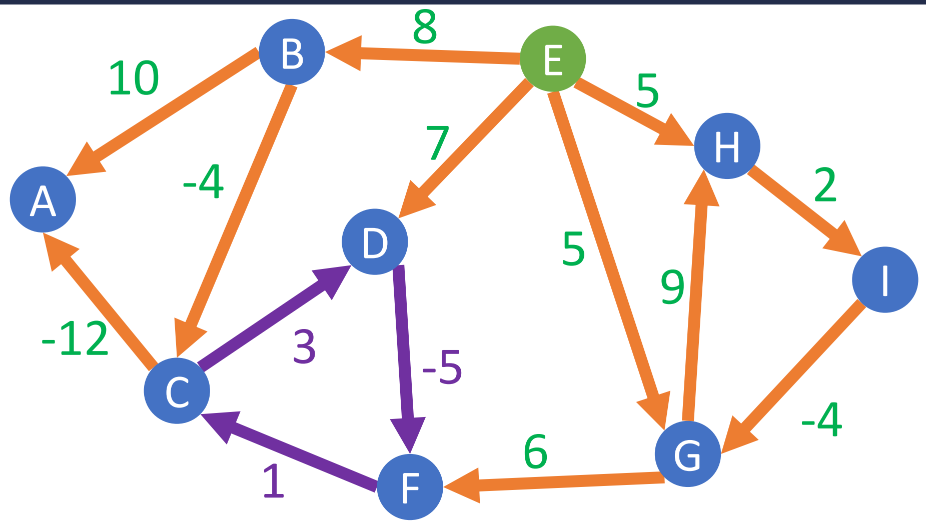
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**2**	**5**	**5**	**7**
3									
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

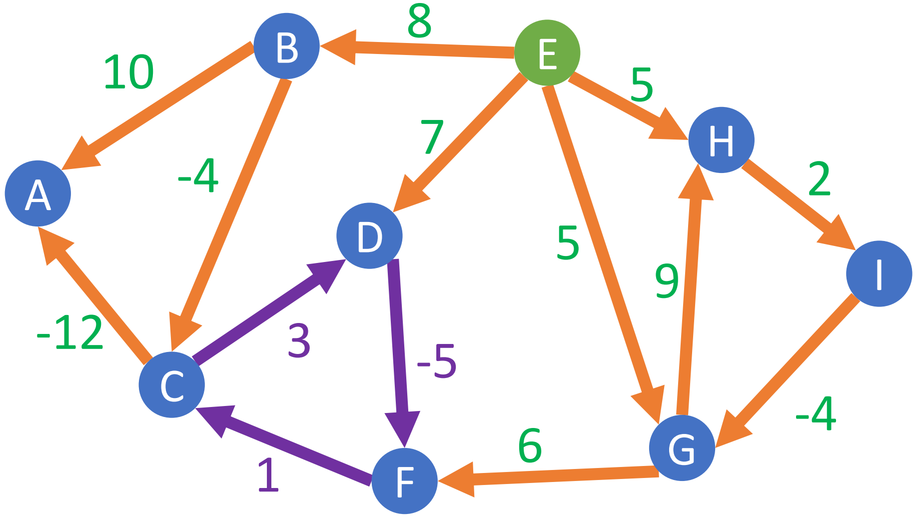
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**2**	**5**	**5**	**7**
3	**-8**	**8**	**3**	**7**	**0**	**2**	**3**	**5**	**7**
4									
5									
6									
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

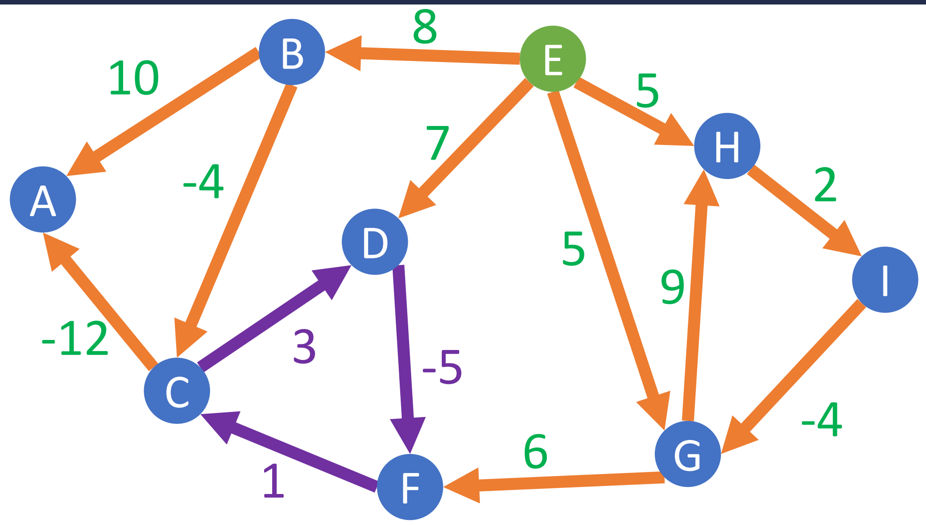
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}\left(\text{Short}(i-1, x) + w(x, v)\right) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**2**	**5**	**5**	**7**
3	**-8**	**8**	**3**	**7**	**0**	**2**	**3**	**5**	**7**
4	**-9**	**8**	**3**	**6**	**0**	**2**	**3**	**5**	**7**
5									
6									
7									
8									



Base case:

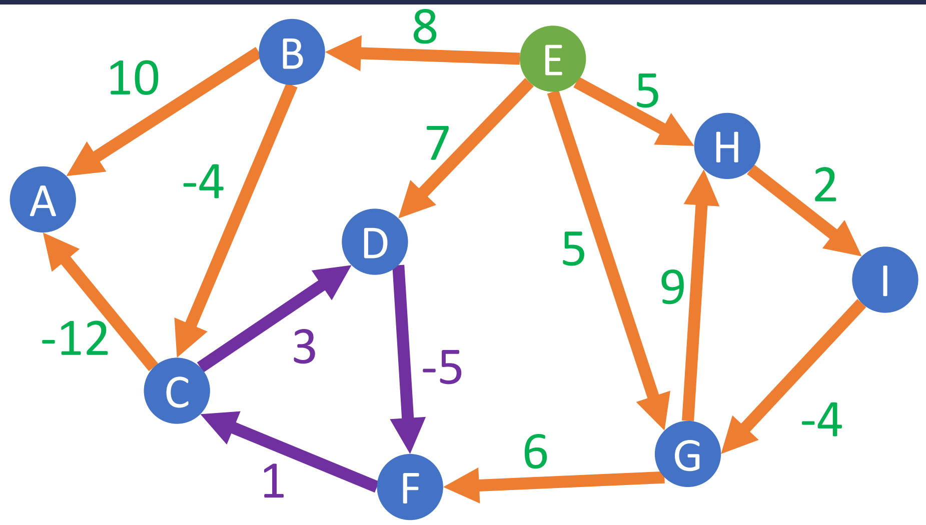
- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**2**	**5**	**5**	**7**
3	**-8**	**8**	**3**	**7**	**0**	**2**	**3**	**5**	**7**
4	**-9**	**8**	**3**	**6**	**0**	**2**	**3**	**5**	**7**
5	**-9**	**8**	**3**	**6**	**0**	**1**	**3**	**5**	**7**
6									
7									
8									

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

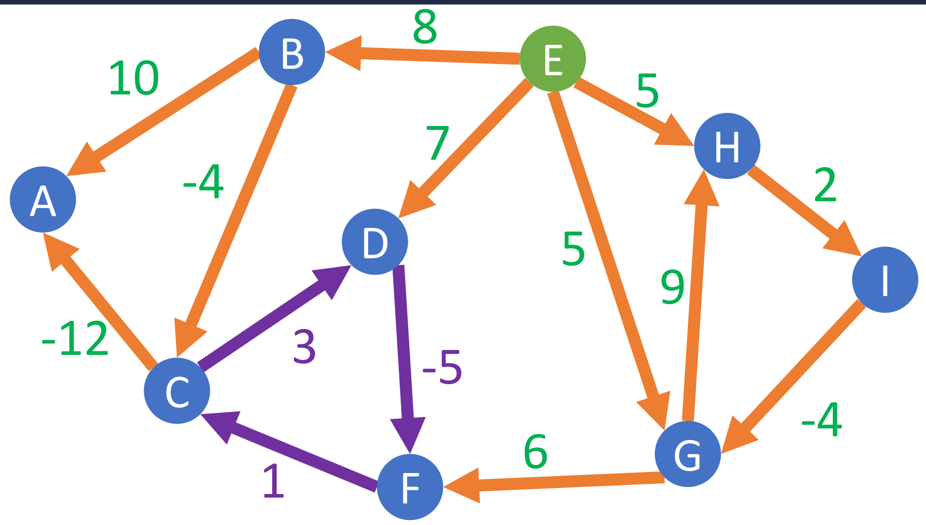
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**2**	**5**	**5**	**7**
3	**-8**	**8**	**3**	**7**	**0**	**2**	**3**	**5**	**7**
4	**-9**	**8**	**3**	**6**	**0**	**2**	**3**	**5**	**7**
5	**-9**	**8**	**3**	**6**	**0**	**1**	**3**	**5**	**7**
6	**-9**	**8**	**2**	**6**	**0**	**1**	**3**	**5**	**7**
7									
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

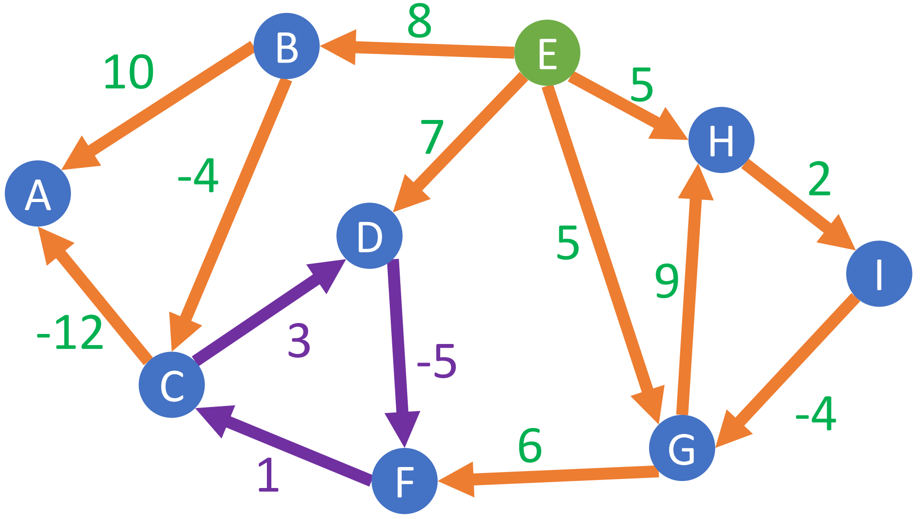
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**8**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**18**	**8**	**4**	**7**	**0**	**2**	**5**	**5**	**7**
3	**-8**	**8**	**3**	**7**	**0**	**2**	**3**	**5**	**7**
4	**-9**	**8**	**3**	**6**	**0**	**2**	**3**	**5**	**7**
5	**-9**	**8**	**3**	**6**	**0**	**1**	**3**	**5**	**7**
6	**-9**	**8**	**2**	**6**	**0**	**1**	**3**	**5**	**7**
7	**-10**	**8**	**2**	**5**	**0**	**1**	**3**	**5**	**7**
8									



Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

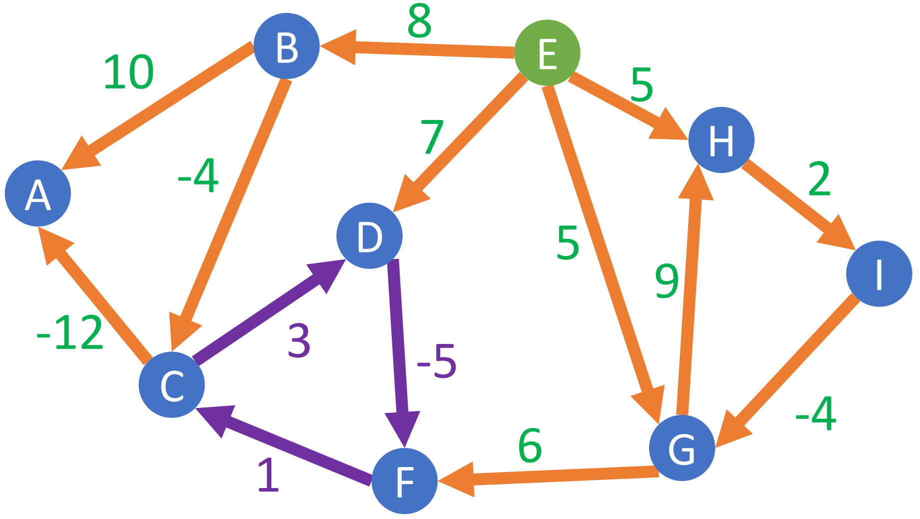
Suppose source node is E

Detecting Negative-Weight Cycles

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	0	∞	∞	∞	∞
1	∞	8	∞	7	0	∞	5	5	∞
2	18	8	4	7	0	2	5	5	7
3	-8	8	3	7	0	2	3	5	7
4	-9	8	3	6	0	2	3	5	7
5	-9	8	3	6	0	1	3	5	7
6	-9	8	2	6	0	1	3	5	7
7	-10	8	2	5	0	1	3	5	7
8	-10	8	2	5	0	0	3	5	7



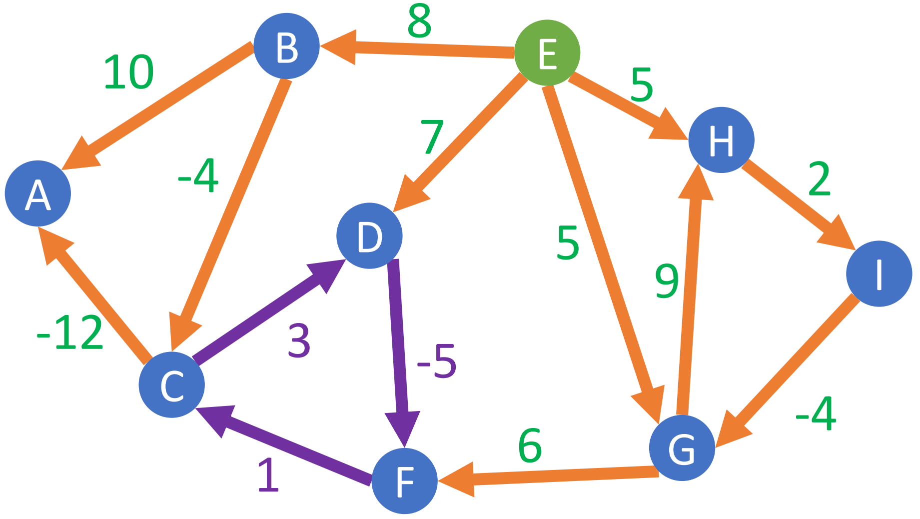
Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Detecting Negative-Weight Cycles

Observation: After $|V| - 1$ iterations, if the lengths of the shortest paths has not converged (e.g., shortest path change after one more iteration of Bellman-Ford), there must exist a negative-weight cycle (since without negative-weight cycles, the shortest path requires at most $|V| - 1$ hops)



$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	0	∞	∞	∞	∞
1	∞	8	∞	7	0	∞	5	5	∞
2	18	8	4	7	0	2	5	5	7
3	-8	8	3	7	0	2	3	5	7
4	-9	8	3	6	0	2	3	5	7
5	-9	8	3	6	0	1	3	5	7
6	-9	8	2	6	0	1	3	5	7
7	-10	8	2	5	0	1	3	5	7
8	-10	8	2	5	0	0	3	5	7

Base case:

- $\text{Short}(0, s) = 0$
- $\text{Short}(0, v) = \infty$ if $v \neq s$

Suppose source node is E

Bellman-Ford Implementation

```
allocate short[n][n]
initialize short[0][v] = ∞ for each v
initialize short[0][s] = 0
for i = 1,...,n - 1:
     for each e = (x, y) in E:
          short[i][y] = min(
               short[i-1][x] + w[x][y],
               short[i-1][y]
          )
```

Bellman-Ford update

$|V| = n$

Bellman-Ford Run Time

```
allocate short[n][n]
initialize short[0][v] = ∞ for each v
initialize short[0][s] = 0
for i = 1,...,n - 1:
    for each e = (x, y) in E:
        short[i][y] = min(
            short[i-1][x] + w[x][y],
            short[i-1][y]
        )
```

$O(|V|^2)$

$O(|V|)$

$O(1)$

$|V|$ times

$|E|$ times

$O(1)$

Running time (naïve) : $O(|V|^2 + |E||V|)$

$|V| = n$

Bellman-Ford Run Time

```
allocate short[n][n]                              O(|V|^2)
initialize short[0][v] = ∞ for each v             O(|V|)
initialize short[0][s] = 0                        O(1)
for i = 1,...,n - 1:                              |V| times
    for each e = (x, y) in E:                     |E| times
        short[i][y] = min(
            short[i-1][x] + w[x][y],              O(1)
            short[i-1][y]
        )
```

Observation: update for row i only depends on update for row $i - 1$
Optimization: only need to store **two** rows of `short` (previous row and current row)

Overall running time of Bellman-Ford: $O(|E||V|)$

Dijkstra vs. Bellman-Ford

Both algorithms solve the single-source shortest path (SSSP) problem

Both algorithms handle directed and undirected graphs

Dijkstra:

- Greedy algorithm that always adds the node that is "closest" to the nodes that have been considered so far
- Only works for graphs with non-negative weights
- Updates require keeping track of shortest path to all nodes in the graph (changing graph weight essentially requires re-running the algorithm)
- **Running time:** $O(|E| \log|V|)$

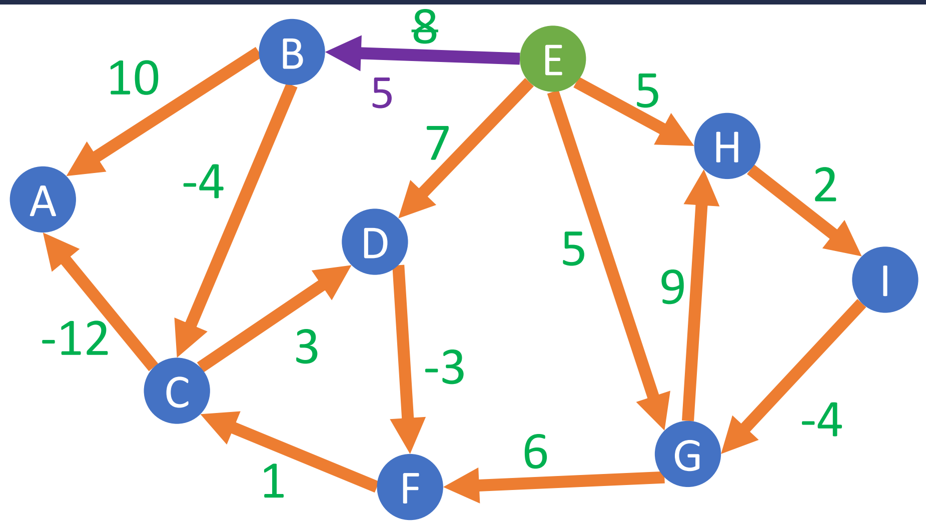
Bellman-Ford:

- Dynamic programming algorithm that updates the costs of all paths based on the current shortest distance to all nodes in the graph
- Handles graphs with negative weights, can also be used to detect negative-weight cycles
- Updates can be distributed (each node only needs to know shortest path from/to each of its neighbors) – used in old routing protocols (e.g., the Routing Information Protocol)
- **Running time:** $O(|E||V|)$

Bellman-Ford in Dynamic Graphs (Update)

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V} (\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



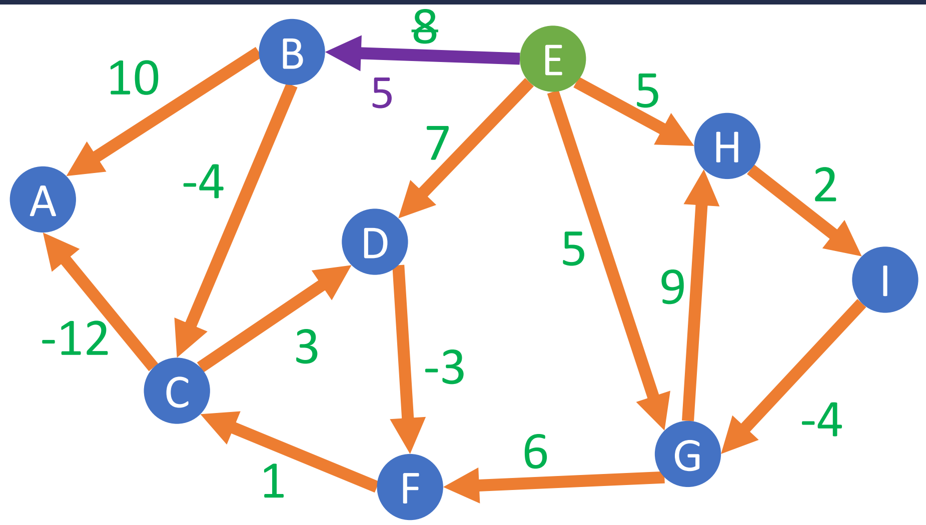
$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	0	∞	∞	∞	∞
1	∞	8	∞	7	0	∞	5	5	∞
2	18	8	4	7	0	4	5	5	7
3	-8	8	4	7	0	4	3	5	7
4	-8	8	4	7	0	4	3	5	7
5	-8	8	4	7	0	4	3	5	7
6	-8	8	4	7	0	4	3	5	7
7	-8	8	4	7	0	4	3	5	7
8	-8	8	4	7	0	4	3	5	7

Bellman-Ford in Dynamic Graphs (Update)

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

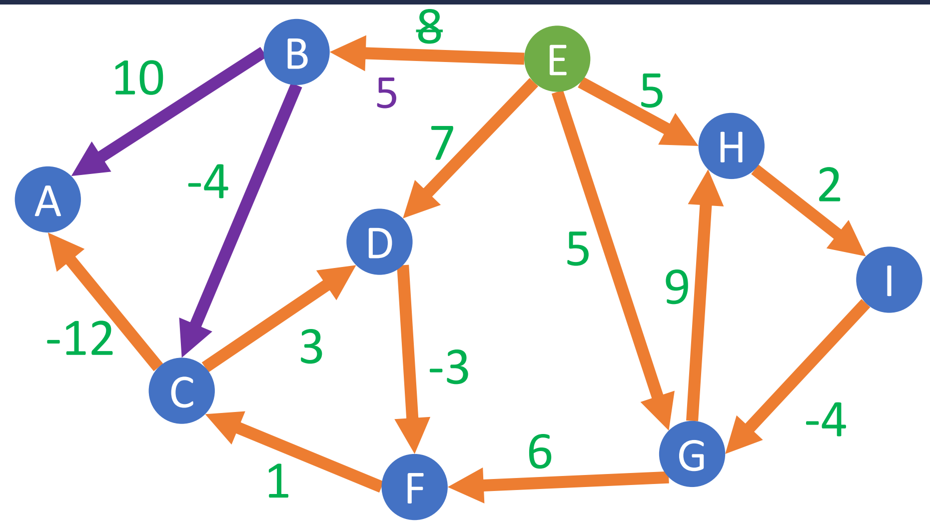
$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	0	∞	∞	∞	∞
1	∞	5	∞	7	0	∞	5	5	∞
2	18	8	4	7	0	4	5	5	7
3	-8	8	4	7	0	4	3	5	7
4	-8	8	4	7	0	4	3	5	7
5	-8	8	4	7	0	4	3	5	7
6	-8	8	4	7	0	4	3	5	7
7	-8	8	4	7	0	4	3	5	7
8	-8	8	4	7	0	4	3	5	7



Bellman-Forford in Dynamic Graphs (Update)

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$



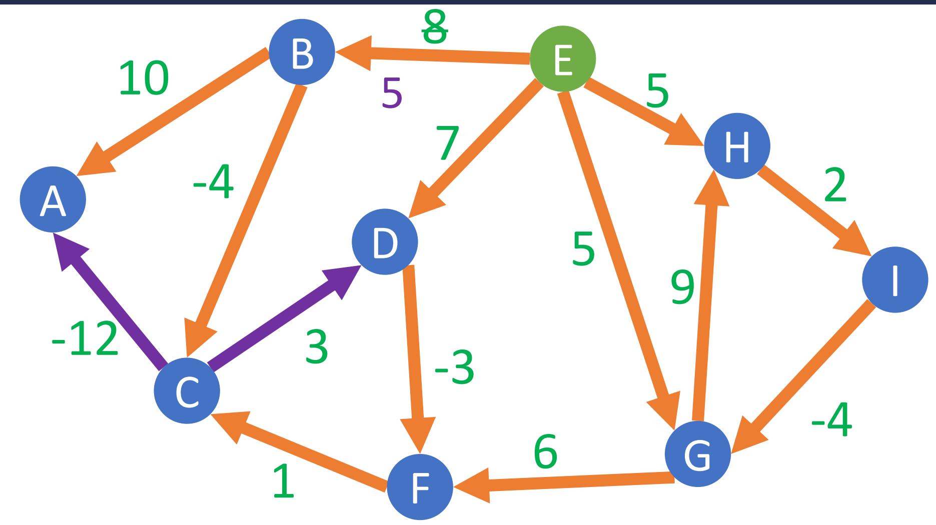
$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**5**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**15**	**5**	**1**	**7**	**0**	**4**	**5**	**5**	**7**
3	**-8**	**5**	**1**	**7**	**0**	**4**	**3**	**5**	**7**
4	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
5	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
6	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
7	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
8	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**

Bellman-Ford in Dynamic Graphs (Update)

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}\left(\text{Short}(i-1, x) + w(x, v)\right) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**5**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**15**	**5**	**1**	**7**	**0**	**4**	**5**	**5**	**7**
3	**-11**	**5**	**1**	**4**	**0**	**4**	**3**	**5**	**7**
4	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
5	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
6	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
7	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**
8	**-8**	**8**	**4**	**7**	**0**	**4**	**3**	**5**	**7**

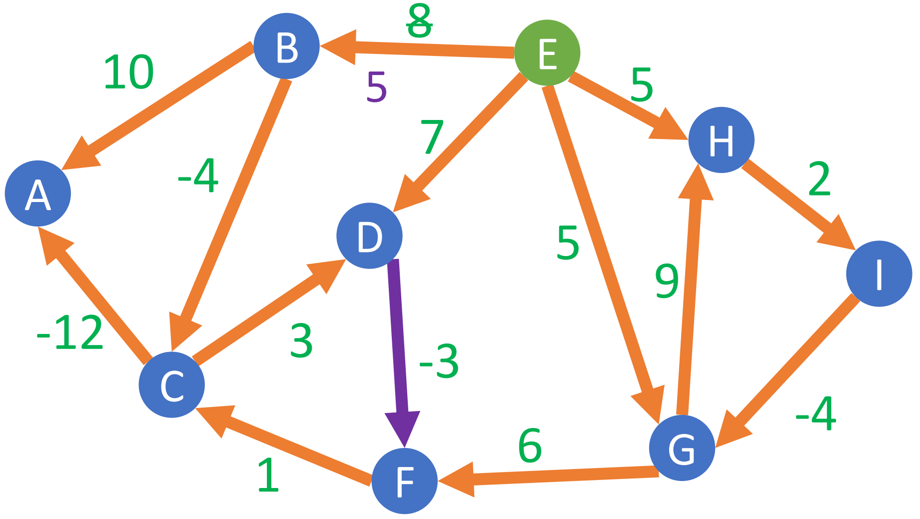


Bellman-Ford in Dynamic Graphs (Update)

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

$$\text{Short}(i, v) = \min \begin{cases} \min_{x \in V}(\text{Short}(i-1, x) + w(x, v)) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	0	∞	∞	∞	∞
1	∞	5	∞	7	0	∞	5	5	∞
2	15	5	1	7	0	4	5	5	7
3	-11	5	1	4	0	4	3	5	7
4	-11	5	1	4	0	1	3	5	7
5	-8	8	4	7	0	4	3	5	7
6	-8	8	4	7	0	4	3	5	7
7	-8	8	4	7	0	4	3	5	7
8	-8	8	4	7	0	4	3	5	7

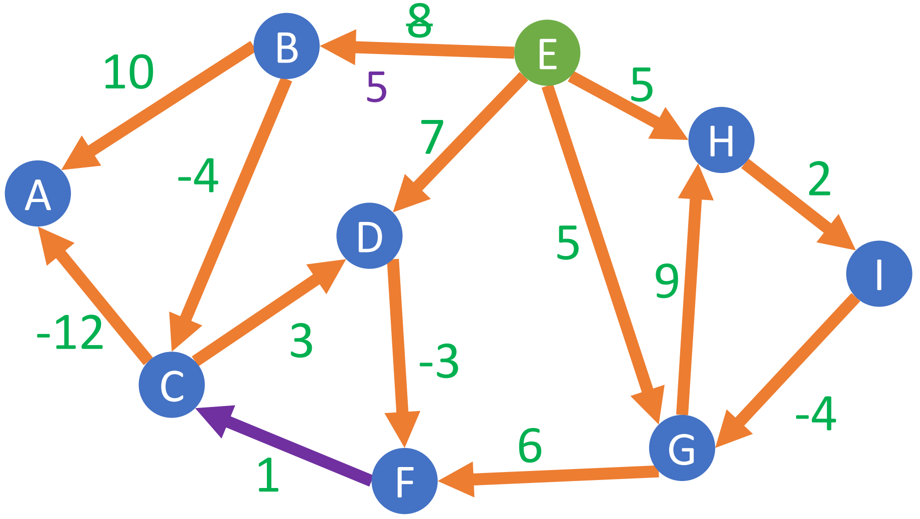


Bellman-Ford in Dynamic Graphs (Update)

$\text{Short}(i, v) =$ weight of the shortest path from s to v using at most i edges

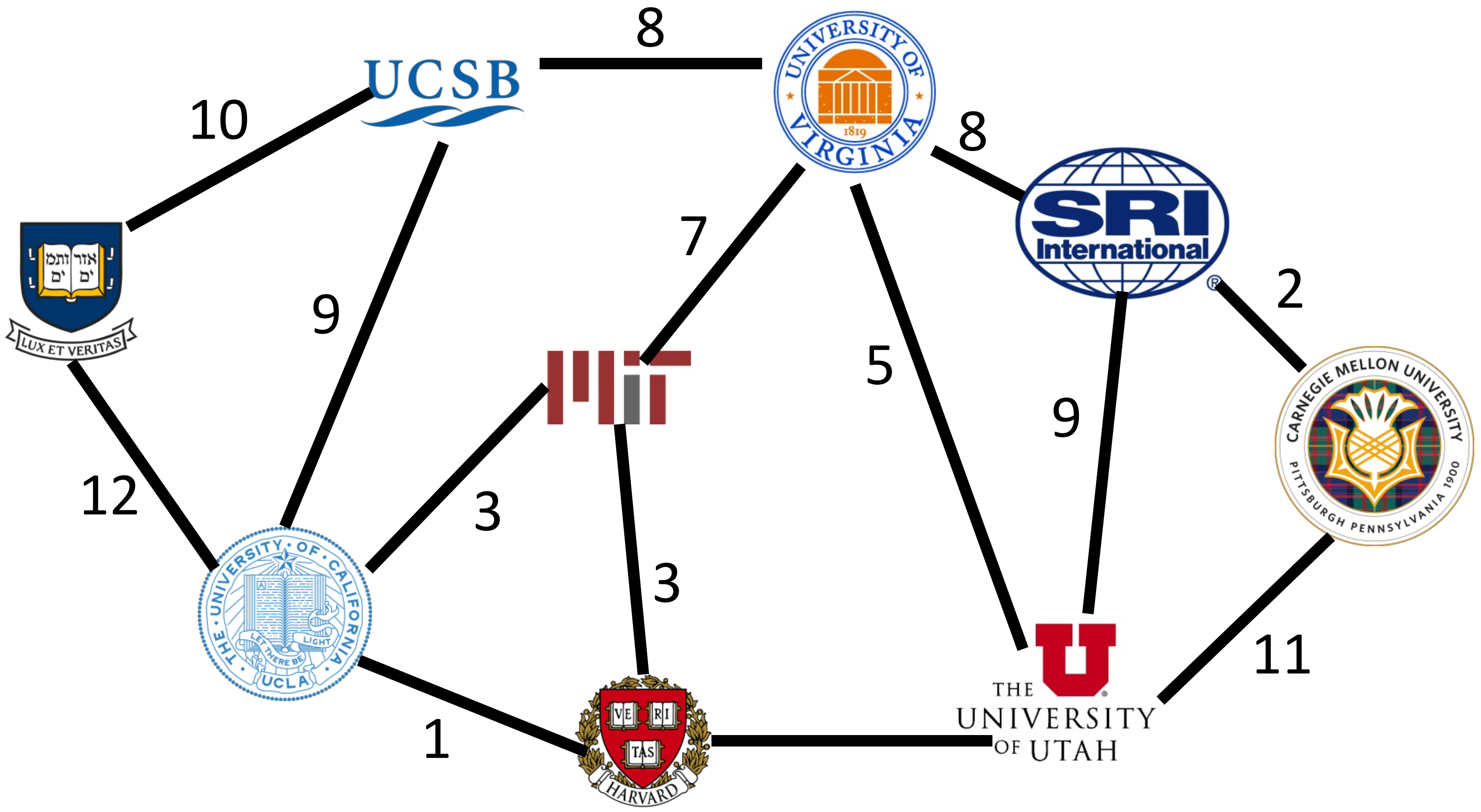
$$\text{Short}(i, v) = \min \begin{cases} \min\limits_{x \in V}\left(\text{Short}(i-1, x) + w(x, v)\right) \\ \text{Short}(i-1, v) \end{cases}$$

$i =$ \ $v =$	A	B	C	D	E	F	G	H	I
0	∞	∞	∞	∞	**0**	∞	∞	∞	∞
1	∞	**5**	∞	**7**	**0**	∞	**5**	**5**	∞
2	**15**	**5**	**1**	**7**	**0**	**4**	**5**	**5**	**7**
3	**-11**	**5**	**1**	**4**	**0**	**4**	**3**	**5**	**7**
4	**-11**	**5**	**1**	**4**	**0**	**1**	**3**	**5**	**7**
5	**-11**	**5**	**1**	**4**	**0**	**1**	**3**	**5**	**7**
6	**-11**	**5**	**1**	**4**	**0**	**1**	**3**	**5**	**7**
7	**-11**	**5**	**1**	**4**	**0**	**1**	**3**	**5**	**7**
8	**-11**	**5**	**1**	**4**	**0**	**1**	**3**	**5**	**7**



Recomputing shortest paths only requires local updates (only need to update paths that emanate from a node whose shortest path has changed)

All-Pairs Shortest Path



Thus far: <u>single-source</u> shortest path algorithms (Dijkstra, Bellman-Ford)

All-pairs shortest-paths: find shortest path between <u>every</u> pair of nodes

All-Pairs Shortest Path

Naïvely: Run single-source shortest paths algorithm for each node s (to compute shortest path from s to every other node in the graph)

- If edge weights are all non-negative, can use Dijkstra (running time $O(|V||E|\log|V|)$
- If edge weights can be negative, can use Bellman-Ford (running time $O(|V|^2|E|)$

When $|E| = \Omega(|V|^2)$, both of these algorithms are $O(|V|^3 \log|V|)$ or $O(|V|^4)$

Can we do better?

Floyd-Warshall All-Pairs Shortest Paths

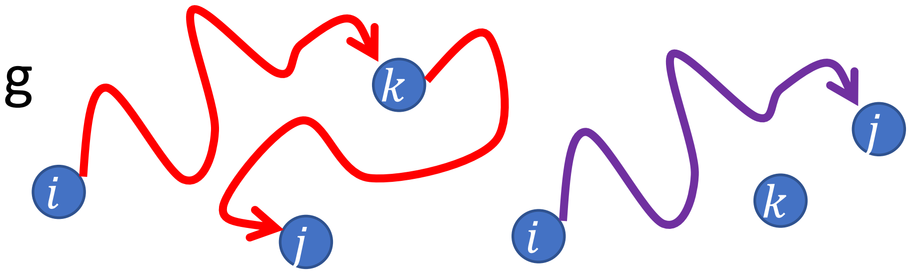
Finds all-pairs shortest paths in $\Theta(|V|^3)$ using dynamic programming

Also works if graph has negative-weight edges

Same observation as before: Every subpath of a shortest path is itself a shortest path (optimal substructure)

- Namely if shortest path from i to j goes through k, then the $i \to j$ and $j \to k$ subpaths must themselves be a shortest path

$\mathrm{Short}(i, j, k) =$ weight of shortest path from $i \to j$ using nodes $1, \ldots, k$ as intermediate hops



Two possibilities for node k:

Shortest path from i to j includes k

$\mathrm{Short}(i, k, k-1) + \mathrm{Short}(k, j, k-1)$

OR

Shortest path from i to j excludes k

$\mathrm{Short}(i, j, k-1)$

Floyd-Warshall All-Pairs Shortest Paths

Finds all-pairs shortest paths in $\Theta(|V|^3)$ using dynamic programming

Also works if graph has negative-weight edges

Same observation as before: Every subpath of a shortest path is itself a shortest path (optimal substructure)

- Namely if shortest path from i to j goes through k, then the $i \to k$ and $k \to j$ subpaths must themselves be a shortest path

$\text{Short}(i, j, k) =$ weight of shortest path from $i \to j$ using nodes $1, \ldots, k$ as intermediate hops

$$\text{Short}(i, j, k) = \min \begin{cases} \text{Short}(i, k, k-1) + \text{Short}(k, j, k-1) \\ \text{Short}(i, j, k-1) \end{cases}$$

Floyd-Warshall All-Pairs Shortest Paths

```
allocate short[n][n][n] (initialized to ∞)
for (i, j) in E:
  short[i][j][0] = w[i][j]
for i = 1,...,n:
  short[i][i][0] = 0

for k = 1,...,n:
  for i = 1,...,n:
    for j = 1,...,n:
      short[i][j][k] = min(short[i][k][k-1] + short[k][j][k-1], short[i][j][k-1])
```

$\boldsymbol{k = 0}$**:** shortest path cannot use any intermediate nodes (must be direct path)

$\boldsymbol{k = 0}$**:** shortest path from node to itself is always 0

$$\text{Short}(i, j, k) = \min \begin{cases} \text{Short}(i, k, k-1) + \text{Short}(k, j, k-1) \\ \text{Short}(i, j, k-1) \end{cases}$$

Floyd-Warshall All-Pairs Shortest Paths

```
allocate short[n][n] (initialized to ∞)
for (i, j) in E:
  short[i][j] = w[i][j]
for i = 1,...,n:
  short[i][i] = 0
```

In this case, the initialization step is constructing the adjacency matrix of the graph (this step is not needed if graph already represented in this form!)

```
for k = 1,...,n:
  for i = 1,...,n:
    for j = 1,...,n:
      short[i][j] = min(short[i][k] + short[k][j], short[i][j])
```

$$\text{Short}(i, j, k) = \min \begin{cases} \text{Short}(i, k, k-1) + \text{Short}(k, j, k-1) \\ \text{Short}(i, j, k-1) \end{cases}$$

Observation: `short[i][j][k]` only depends on values for `short[][][k - 1]`, so we can just use a single two-dimensional array

Floyd-Warshall All-Pairs Shortest Paths

```
allocate short[n][n] (initialized to ∞)
for (i, j) in E:
  short[i][j] = w[i][j]
for i = 1,...,n:
  short[i][i] = 0
```

$\boldsymbol{k = 0}$**:** shortest path cannot use any intermediate nodes (must be direct path)

$\boldsymbol{k = 0}$**:** shortest path from node to itself is always 0

```
for k = 1,...,n:
  for i = 1,...,n:
    for j = 1,...,n:
      short[i][j] = min(short[i][k] + short[k][j], short[i][j])
```

$$\mathrm{Short}(i, j, k) = \min \begin{cases} \mathrm{Short}(i, k, k-1) + \mathrm{Short}(k, j, k-1) \\ \mathrm{Short}(i, j, k-1) \end{cases}$$

Very simple implementation!

Running time: $O(n^3) = O(|V|^3)$

Shortest Paths Review

Single Source Shortest Paths

- Dijkstra: $\Theta(|E| \log |V|)$
  - Greedy algorithm (choose closest node to current explored nodes)
  - No negative edge weights
- Bellman-Ford: $\Theta(|E||V|)$
  - Dynamic programming algorithm
  - Supports negative edge weights (and finds negative weight cycles)
  - Supports local updates when edge weights change

All Pairs Shortest Paths

- Floyd-Warshall: $\Theta(|V|^3)$
  - Supports negative edge weights