CSE 417 Autumn 2025

Lecture 17: Dynamic Programming – Graphs

Nathan Brunelle

Single-source shortest paths (Dijkstra's)

Given: an (un)directed graph $G = (V, E)$ with each edge e having a non-negative weight $w(e)$ and a vertex s

Find: (length of) shortest paths from s to each vertex in G

Single-source shortest paths (Today)

Given: an (un)directed graph $G = (V, E)$ with each edge e having a ~~non-negative~~ weight $w(e)$ and a vertex s

Find: (length of) shortest paths from s to each vertex in G

Dijkstra's Algorithm

- Maintain a set S of vertices whose shortest paths are known
  - initially $S = \{s\}$
- Maintaining current best lengths of paths that *only go through* S to each of the vertices in G
  - path-lengths to elements of S will be right, to $V \setminus S$ they might not be right
- Repeatedly add vertex v to S that has the shortest path-length of any vertex in $V \setminus S$
  - update path lengths based on new paths through v

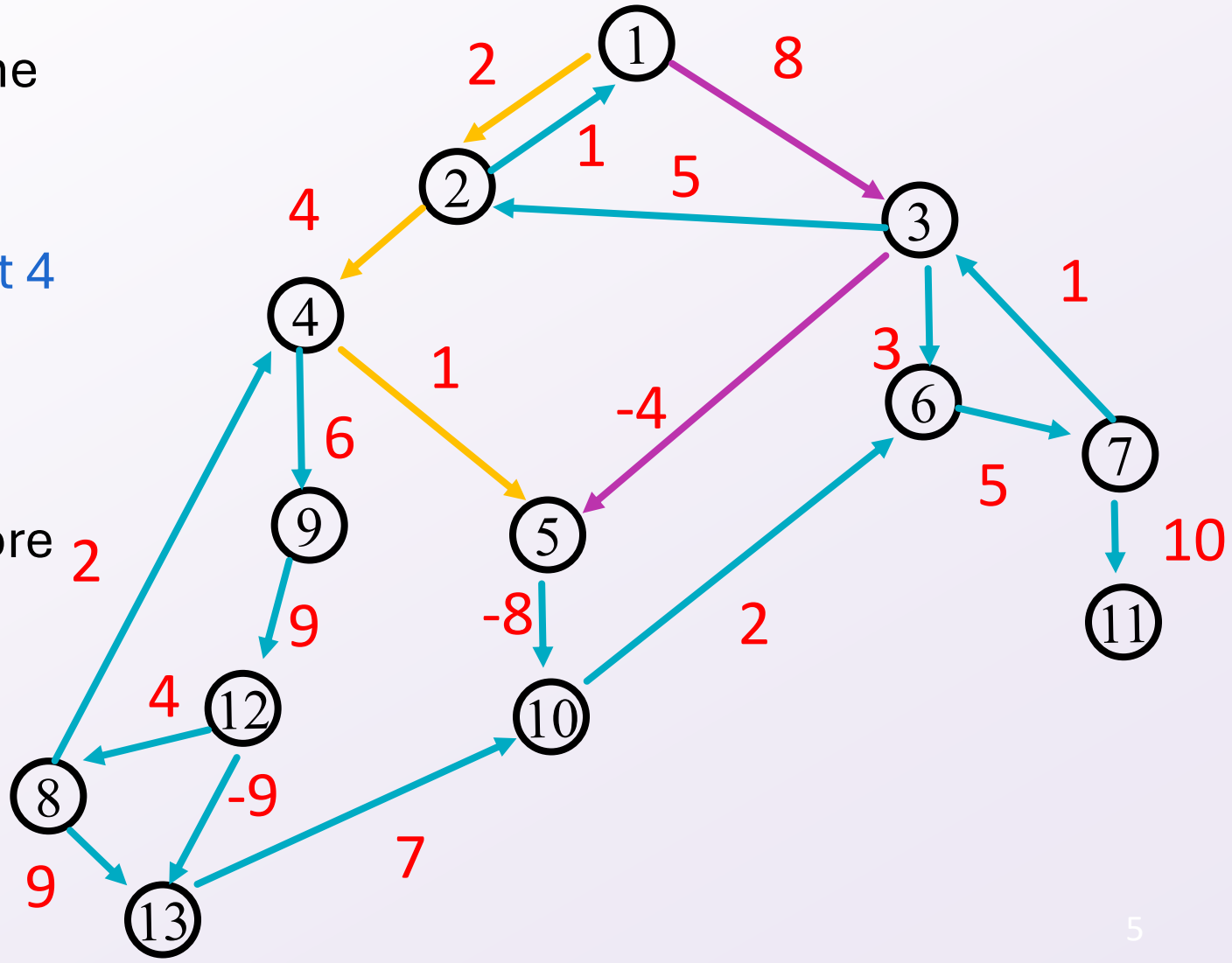
Directed Graph with Negative Weights

Dijkstra's algorithm would not find the shortest path from 1 to 5

Shortest path is 1,3,5 which has cost 4

The path 1,2,4,5 has cost 7

Dijkstra's will "lock in" *that* path before processing node 3



Negative Cycles

There's an issue when a graph has negative cost cycles

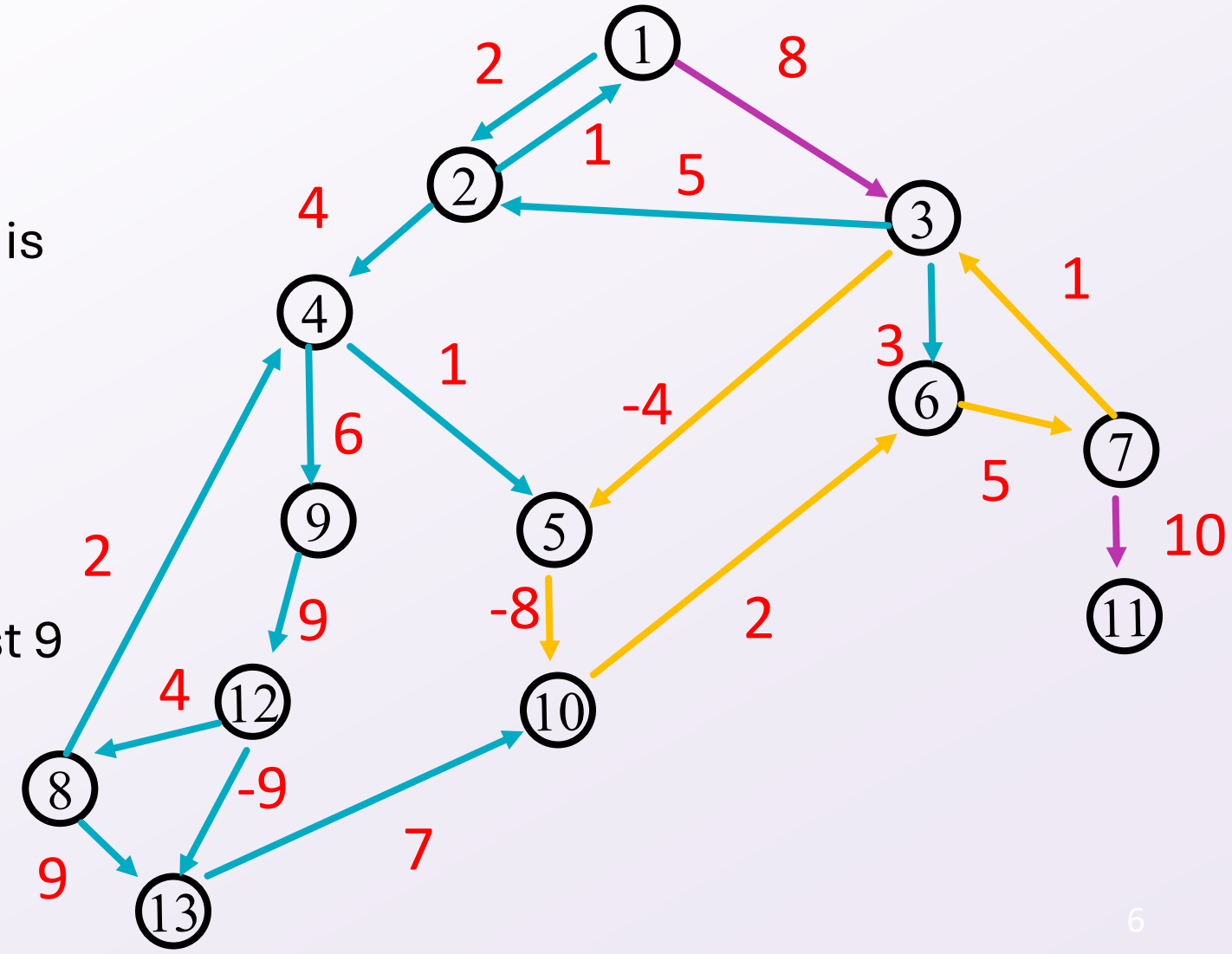
The shortest simple path to node 11 is 1,3,5,10,6,7,11 which has cost 13

The cycle 3,5,8,10,6,7,3 has cost -4

The (non-simple) path 1,3,5,8,10,6,7,3,5,10,6,7,11 has cost 9

Taking the cycle twice gives cost 5

No shortest path exists!



Motivating Example – Currency Exchanges

1 Dollar = 0.8783121137 Euro

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
HKD	Hong Kong Dollar	7.7563973683	0.1289258341
MXN	Mexican Peso	17.3168505322	0.0577472213
ZAR	South African Rand	14.7201431400	0.0679341220

1 Dollar = 3.87 Ringgit

Dollars to Ringgit

1 Dollar = 0.8783121137 Euro

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 Euro= 4.1823100458 Dirham

1 Dirham= 1.0548325619 Ringgit

1 Dollar= 0.8783121137 * 4.1823100458 * 1.0548325619 Ringgit

= **3.87479406049 Ringgit**

Directly: 1 Dollar = 3.87 Ringgit

How to get free money!

1 Dollar = 3.87479406049 Ringgit

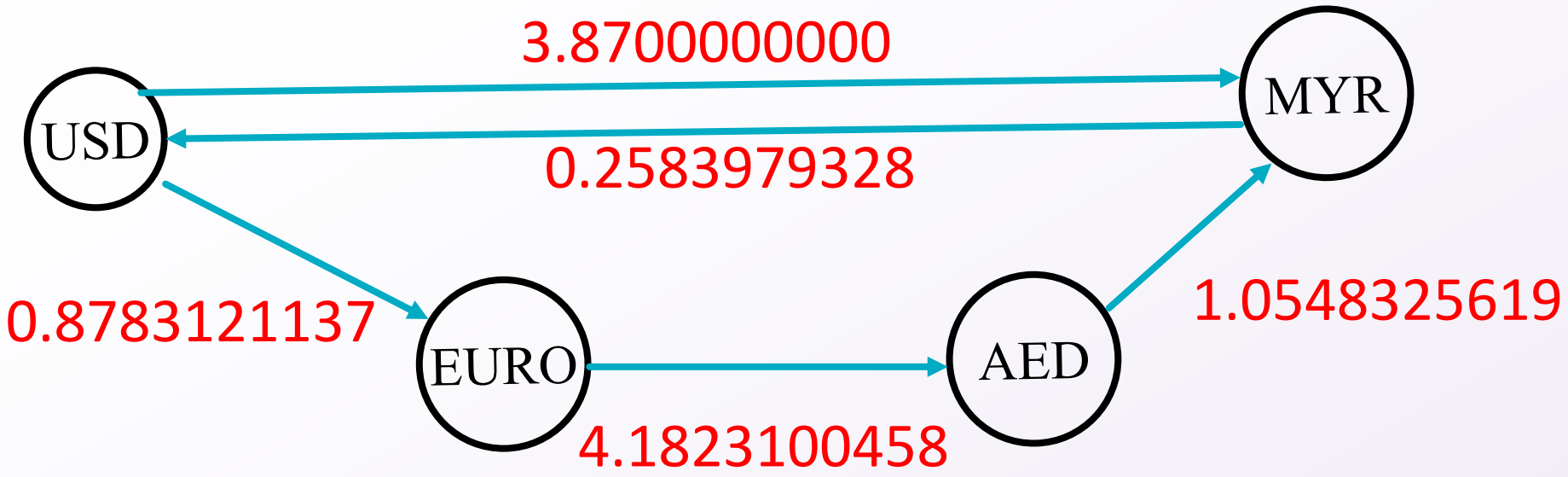
Currency code ▲▼	Currency name ▲▼	Units per USD	USD per Unit
USD	US Dollar	1.0000000000	1.0000000000
EUR	Euro	0.8783121137	1.1385474303
GBP	British Pound	0.6956087704	1.4375896950
INR	Indian Rupee	66.1909310706	0.0151078098
AUD	Australian Dollar	1.3050318080	0.7662648480
CAD	Canadian Dollar	1.2997506294	0.7693783541
SGD	Singapore Dollar	1.3478961522	0.7418969172
CHF	Swiss Fra	451582	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			0.6906068347
THB			0.0284895968
HUF	Hungarian F	275.7012427385	0.0036271146
AED	Emirati Dirha	3.6730000000	0.2722570106
HKD	Hong Kong Dollar	7.7563973683	0.1289258341
MXN	Mexican Peso	17.3168505322	0.0577472213
ZAR	South African Rand	14.7201431400	0.0679341220

1 Ringgit = 0.2583979328 Dollar

1 Dollar = 3.87479406049 * 0.2583979328 Dollar
= 1.00123877526 Dollar

Free Money!

Currency Exchange as a Graph

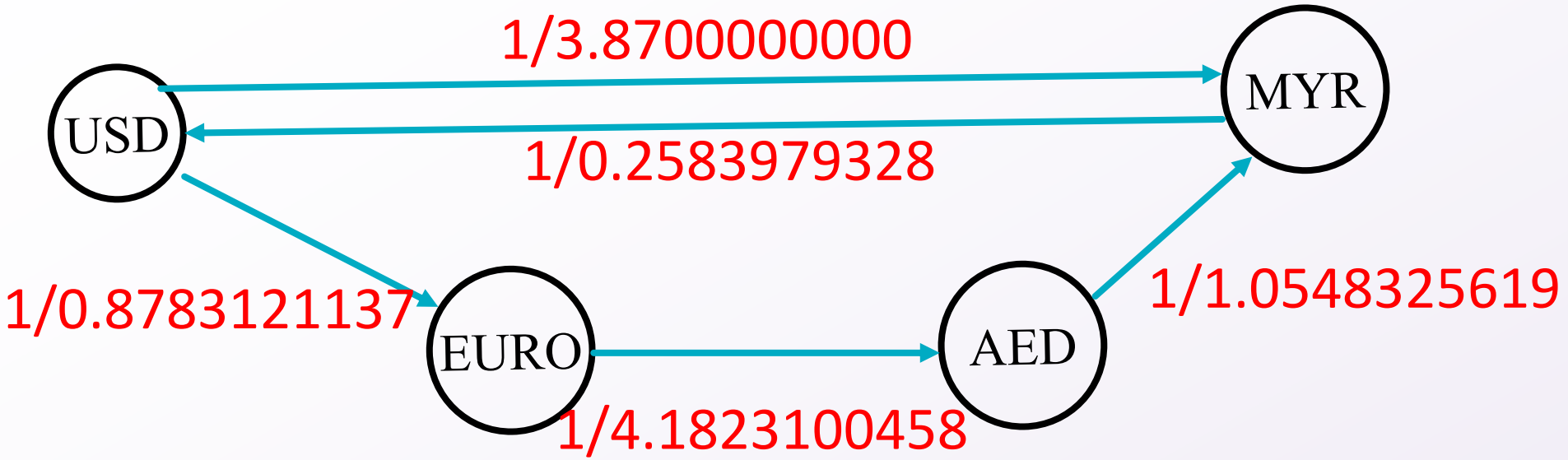


To convert 1 USD to MYR gives $1 \cdot 3.87 = 3.87$

To convert 1 USD to EURO to AED to MYR gives $1 \cdot 0.878\ldots \cdot 4.182\ldots \cdot 1.054\ldots = 3.874\ldots$

This is just finding the maximum path by product of edge weights!

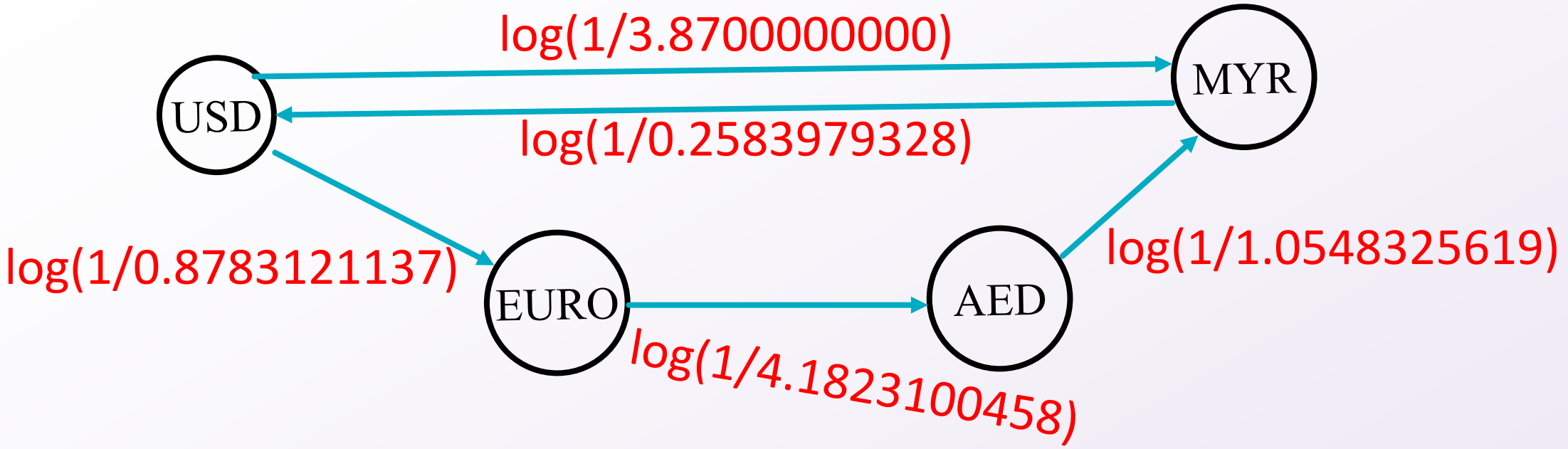
Converting to Shortest Path – max to min



If we take the reciprocal of each weight then we turn it into a minimization problem!

To maximize $a \cdot b \cdot c$ we can minimize $\frac{1}{a \cdot b \cdot c}$

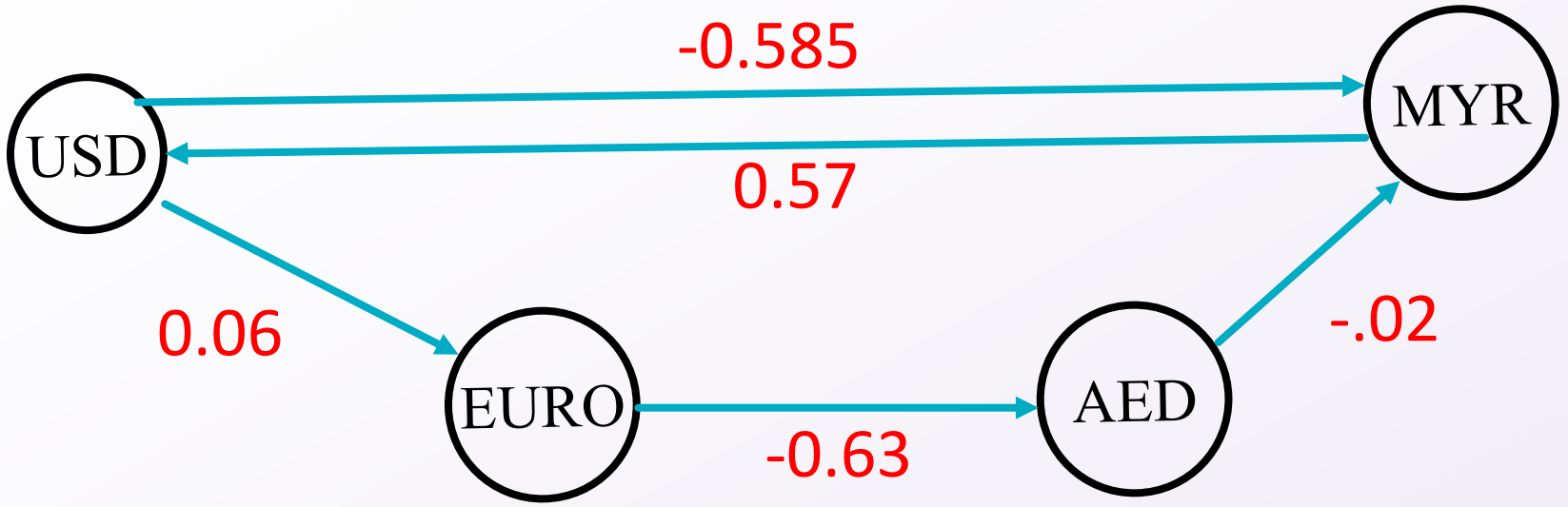
Converting to Shortest Path – product to sum



To convert the product into sums we can take the log of each weight!

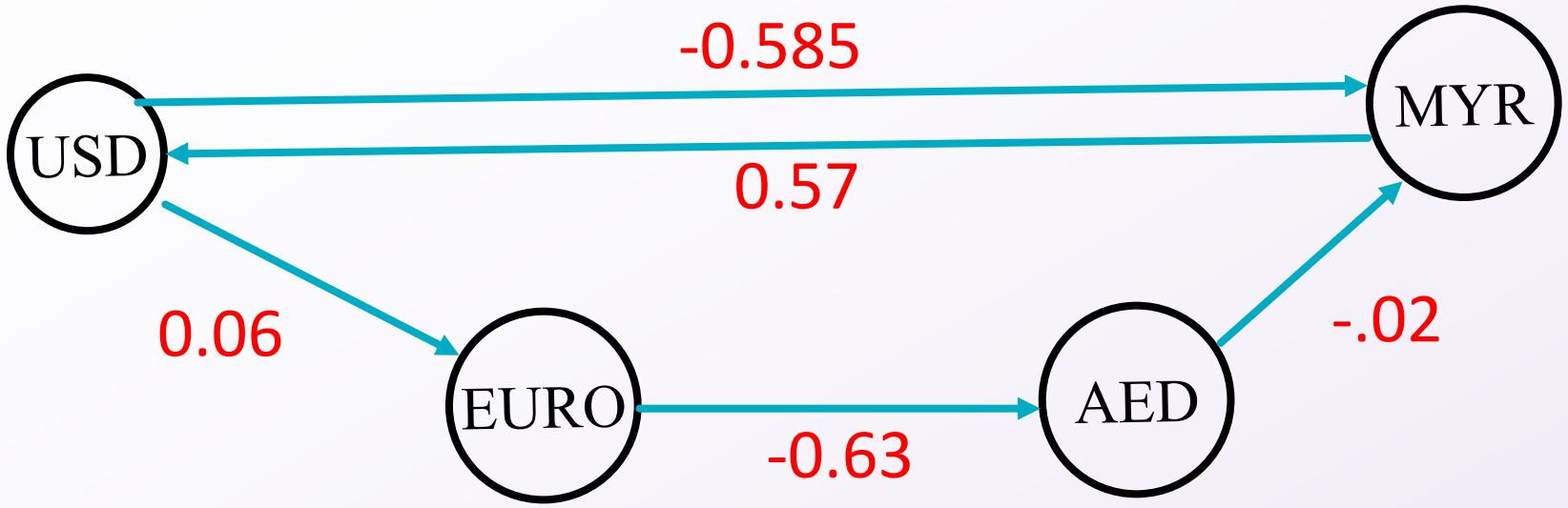
To minimize $a \cdot b \cdot c$ we can minimize $\log(a \cdot b \cdot c) = \log a + \log b + \log c$

Best Exchange = Shortest path



To find the best way to convert USD to MYR, find the shortest path from USD to MYR!

Free Money = Negative Cycles!



To get free money we need a cycle such the product of the weights is greater than 1.

Repeating our transformation: we need $\frac{1}{product} < 1$, so we need $\log \frac{1}{product} < 0$

So a negative cycle is an opportunity to make free money! 0.06-0.63-0.02+0.57=-0.02

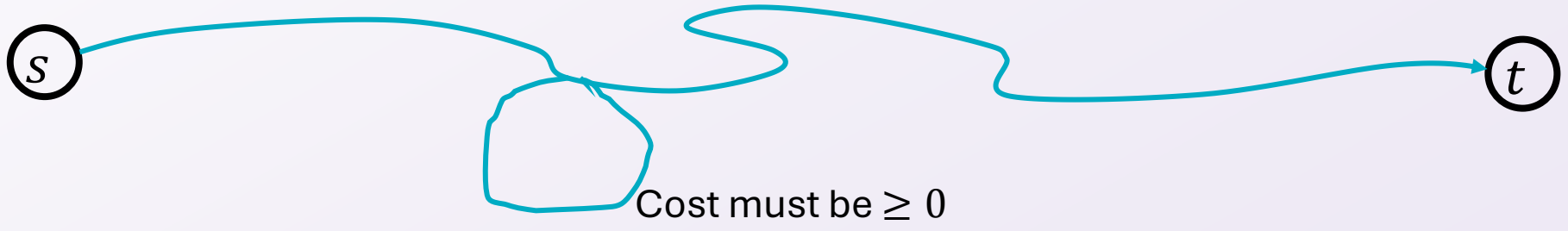
Observations

Claim: A simple path has at most $|V| - 1$ edges

Justication: Pigeon-hole principle. If we have $\geq |V|$ edges then we have used at least one node at least twice

Claim: If a graph has no negative weight cycles then any shortest path must be simple

Justication: If some shortest path was not simple then there is a repeated node. The cycle involving that repeated node must have weight ≥ 0. Removing that cycle from the path can't make it worse



Single-source shortest paths, with negative edge weights

Given: an (un)directed graph $G = (V, E)$ with each edge e having a weight $w(e)$ and a vertex s

Find: (length of) shortest paths from s to each vertex in G, **or else indicate that there is a negative-cost cycle**

Called the Bellman-Ford algorithm

(The original DP algorithm!)

(Also, the original shortest path algorithm!)

Bellman Ford – Step 1

1. Formulate the answer with a recursive structure

   What are the options for the last choice?

   For each such option, what does the subproblem look like? How do we use it?

2. Choose a memory structure.

   Figure out the possible values of all parameters in the recursive calls.

   How many subproblems (options for last choice) are there?

   What are the parameters needed to identify each?

   How many different values could there be per parameter?

3. Specify an order of evaluation.

   Want to guarantee that the necessary subproblem solutions are in memory when you need them.

   With this step: a “Bottom-up” (iterative) algorithm

   Without this step: a “Top-down” (recursive) algorithm

4. See if there’s a way to save space

   Is it possible to reuse some memory locations?

Identifying Recursive Structure – False Start

Consider the shortest path from s to t

This shortest path is composed of:

- The shortest path from s to the second-to-last node (call it u)
- The edge (u, t)

$OPT(t)$ = The cost of the shortest path from s to t

$$OPT(t) = \begin{cases} 0 & \text{if } s = t \\ \min_{u \in V} \{OPT(u) + w(u,t)\} & \text{o. w.} \end{cases}$$

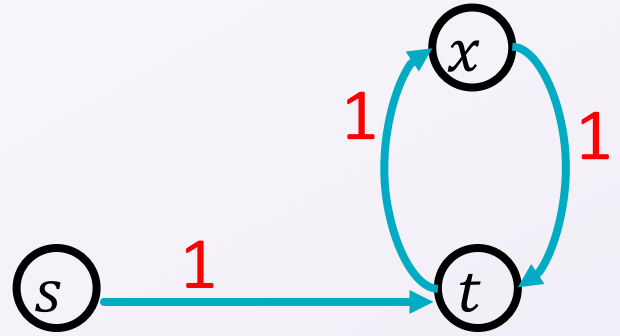
Where $w(u,t)$ is the weight of the edge from u to t if it exists and ∞ if not.

So…What's wrong with this?

$OPT(t) =$ The cost of the shortest path from s to t

$$OPT(t) = \begin{cases} 0 & \text{if } s = t \\ \min\limits_{u \in V}\{OPT(u) + w(u,t)\} & \text{o.w.} \end{cases}$$

Where $w(u,t)$ is the weight of the edge from u to t if it exists and ∞ if not.



$$OPT(t) = \min\begin{cases} \boldsymbol{OPT(x) + 1} \\ OPT(s) + 1 \end{cases}$$

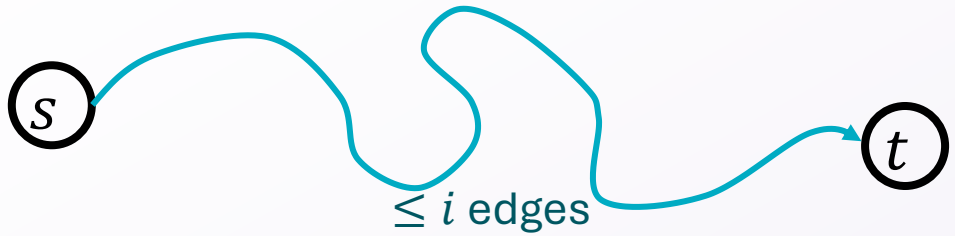
$$OPT(x) = \min\begin{cases} \boldsymbol{OPT(t) + 1} \\ OPT(s) + \infty \end{cases}$$

$$OPT(s) = 0$$

We never reach a base case!

Identifying Recursive Structure – Correctly!

Suppose the shortest path from s to t has $\boldsymbol{i}$ **or fewer edges**



$OPT(i,t)$ = the weight of the shortest path from s to t with **at most** i edges

This shortest path will be one of these:

Option 1: the shortest path from s to some u with $\boldsymbol{i-1}$ **or fewer edges**, plus the edge (u,t)

$$\min_{u \in V}\{OPT(i-1,u) + w(u,t)\}$$

Option 2: the same as shortest path from s to t with $\boldsymbol{i-1}$ **or fewer edges**

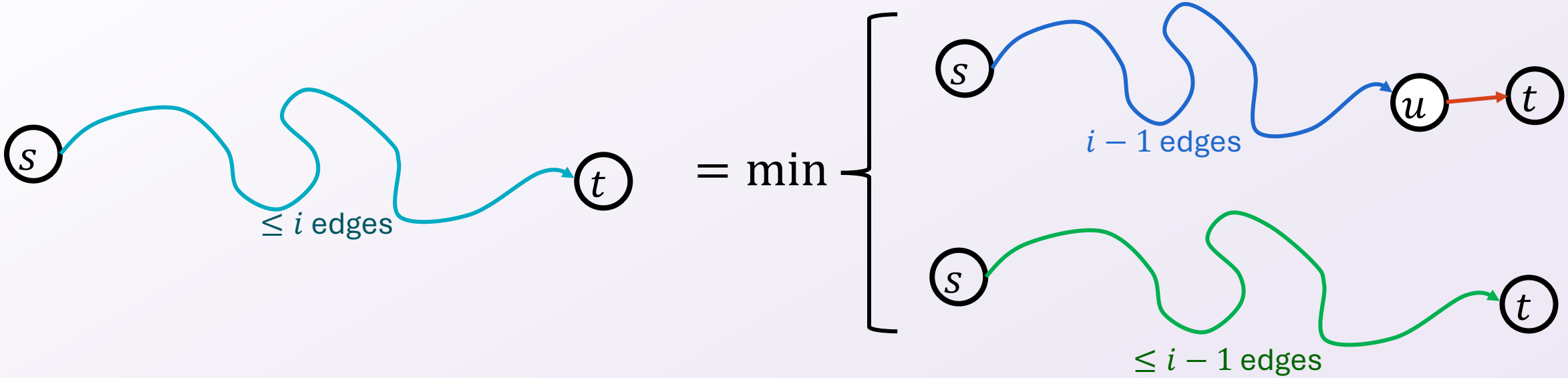
$$OPT(i-1,t)$$

Final Recursive Structure

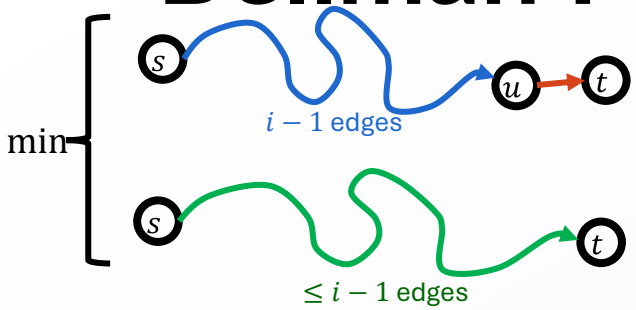
$OPT(i,t)$ = the weight of the shortest path from s to t with at most i edges

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min\limits_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

Where $w(u,t)$ is the weight of the edge from u to t if it exists and ∞ if not.



Bellman Ford – Step 2



1. Formulate the answer with a recursive structure
   - What are the options for the last choice?
   - For each such option, what does the subproblem look like? How do we use it?
2. Choose a memory structure.
   - Figure out the possible values of all parameters in the recursive calls.
   - How many subproblems (options for last choice) are there?
   - What are the parameters needed to identify each?
   - How many different values could there be per parameter?
3. Specify an order of evaluation.
   - Want to guarantee that the necessary subproblem solutions are in memory when you need them.
   - With this step: a “Bottom-up” (iterative) algorithm
   - Without this step: a “Top-down” (recursive) algorithm
4. See if there’s a way to save space
   - Is it possible to reuse some memory locations?

Identifying the Memory Structure

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min\limits_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

How many parameters?

2

What does each represent?

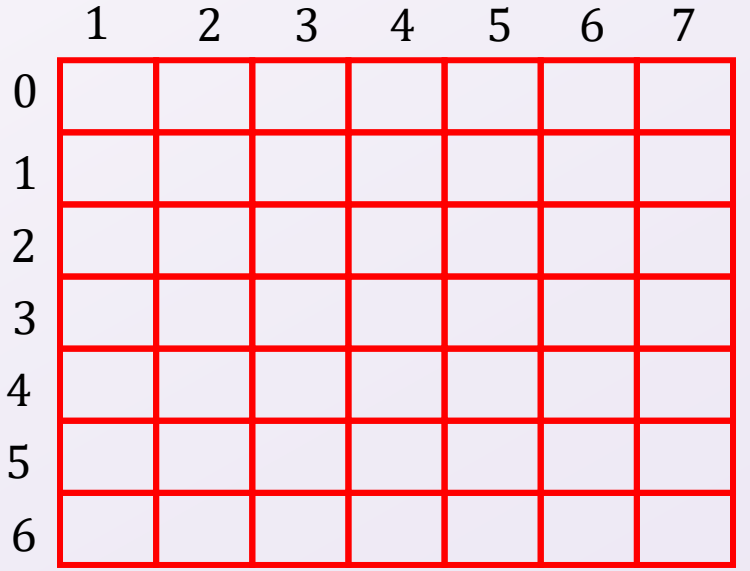
i: the length of the path

t: a node

How many different values?

i: $|V|$ (from length 0 up to $|V| - 1$ if the path is simple)

t: $|V|$ (one value per node)



Top-Down Bellman-Ford

This algorithm correctly finds shortest paths when there are no negative-cost cycles
How can we check for negative cost cycles?

BF(i, t):

```
if OPT[i][t] not blank:    // Check if we've solved this already
        return OPT[i][t]
if i == 0:    // Check if this is a base case
        solution = 0 ? t == s : ∞
        OPT[i][t] = solution    // Always save your solution before returning
        return solution
solution = ∞
for each u ∈ V:
        solution = min(solution, BF(i − 1, u) +w(u, t)) // solve each subproblem, pick which to use
solution = min(solution, BF(i − 1,t)) // solve each subproblem, pick which to use
OPT[i][t] = solution    // Always save your solution before returning
return solution
```

Checking for Negative Cycles

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

How many parameters?

2

What does each represent?

i: the length of the path

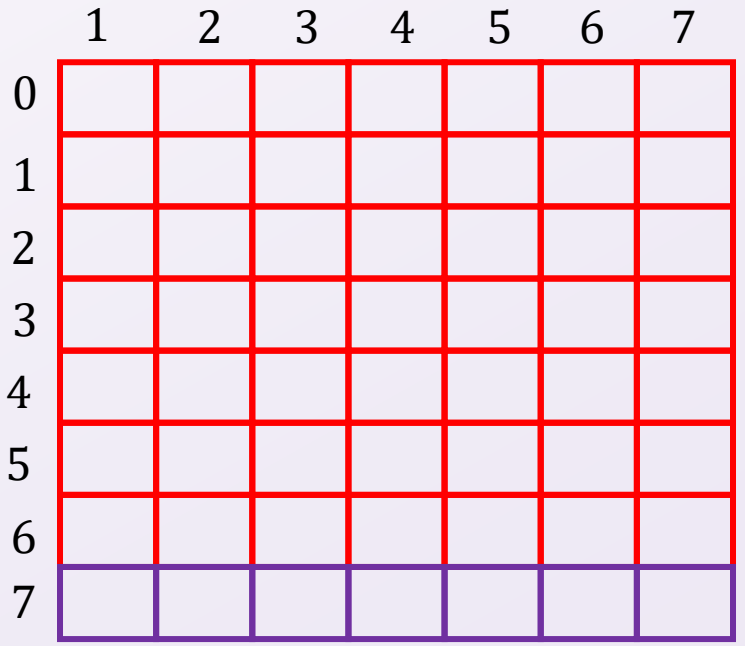
t: a node

How many different values?

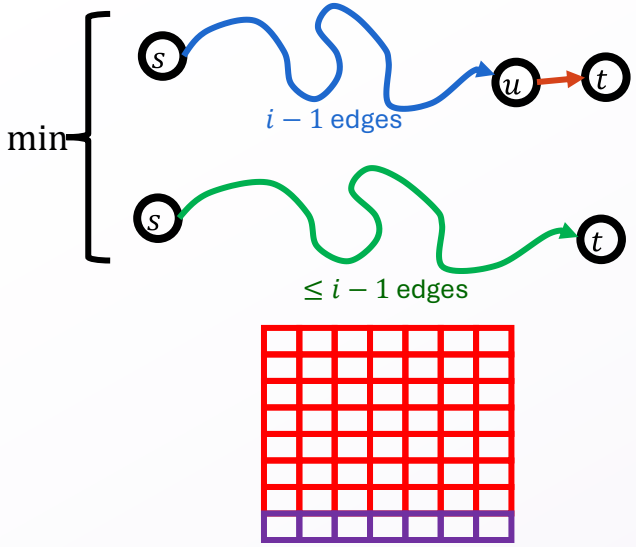
$\boldsymbol{i}$: $|\boldsymbol{V}|$**+1**

- **a path of $|V|$ edges is not simple, so if any $|V|$-edge path is shorter than one with fewer edges, there must be a negative cycle!**

t: $|V|$ (one value per node)



Bellman Ford – Step 3



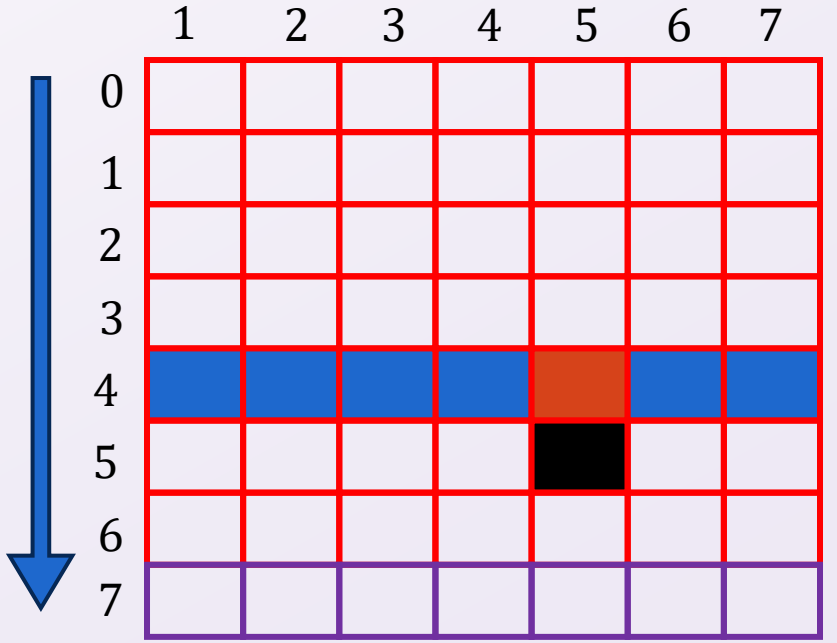
1. Formulate the answer with a recursive structure
   - What are the options for the last choice?
   - For each such option, what does the subproblem look like? How do we use it?
2. Choose a memory structure.
   - Figure out the possible values of all parameters in the recursive calls.
   - How many subproblems (options for last choice) are there?
   - What are the parameters needed to identify each?
   - How many different values could there be per parameter?
3. Specify an order of evaluation.
   - Want to guarantee that the necessary subproblem solutions are in memory when you need them.
   - With this step: a “Bottom-up” (iterative) algorithm
   - Without this step: a “Top-down” (recursive) algorithm
4. See if there’s a way to save space
   - Is it possible to reuse some memory locations?

Order of Evaluations

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min\limits_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

Each cell depends on every value in the previous row

Solve in order of i



Bottom-Up Bellman-Ford

BF(s, t):

OPT$[0][s]$ = 0 // Solve and save base cases

for $u \in V \setminus \{s\}$:

OPT$[0][u]$ = ∞ // Solve and save base cases

for $i = 0$ up to $|V|$:

for $u \in V$:

for $v \in \text{neighbors}(u)$:

$OPT[i][u] = \min(OPT[i][u], OPT[i-1][v])$ // solve and pick

$OPT[i][u] = \min(OPT[i][u], OPT[i-1][u])$ // solve and pick

for $u \in V$:

if $OPT[|V|][u] < OPT[|V|-1][u]$: // check for negative cycles

return "negative cycle"

return OPT$[s][t]$ // return the final answer

Bottom-Up Bellman-Ford

BF(s, t):

OPT$[0][s]$ = 0 // Solve and save base cases $\Theta(1)$

for $u \in V \setminus \{s\}$:

OPT$[0][u] = \infty$ // Solve and save base cases $\Theta(|V|)$

for $i = 0$ up to $|V|$:

for $u \in V$: $\Theta(|V||E|)$

for $v \in \text{neighbors}(u)$:

$OPT[i][u] = \min(OPT[i][u], OPT[i-1][v])$ // solve and pick

$OPT[i][u] = \min(OPT[i][u], OPT[i-1][u])$ // solve and pick

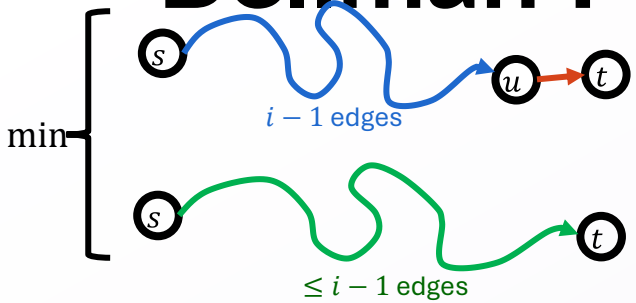
for $u \in V$:

if $OPT[|V|][u] < OPT[|V|-1][u]$: // check for negative cycles $\Theta(|V|)$

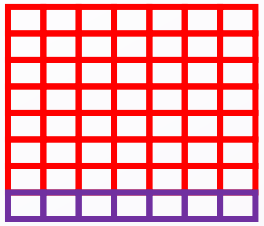
return "negative cycle"

return OPT$[s][t]$ // return the final answer $\Theta(1)$

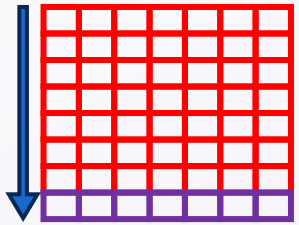
Bellman Ford – Step 4



1. Formulate the answer with a recursive structure
   - What are the options for the last choice?
   - For each such option, what does the subproblem look like? How do we use it?
2. Choose a memory structure.
   - Figure out the possible values of all parameters in the recursive calls.
   - How many subproblems (options for last choice) are there?
   - What are the parameters needed to identify each?
   - How many different values could there be per parameter?



3. Specify an order of evaluation.
   - Want to guarantee that the necessary subproblem solutions are in memory when you need them.
   - With this step: a “Bottom-up” (iterative) algorithm
   - Without this step: a “Top-down” (recursive) algorithm



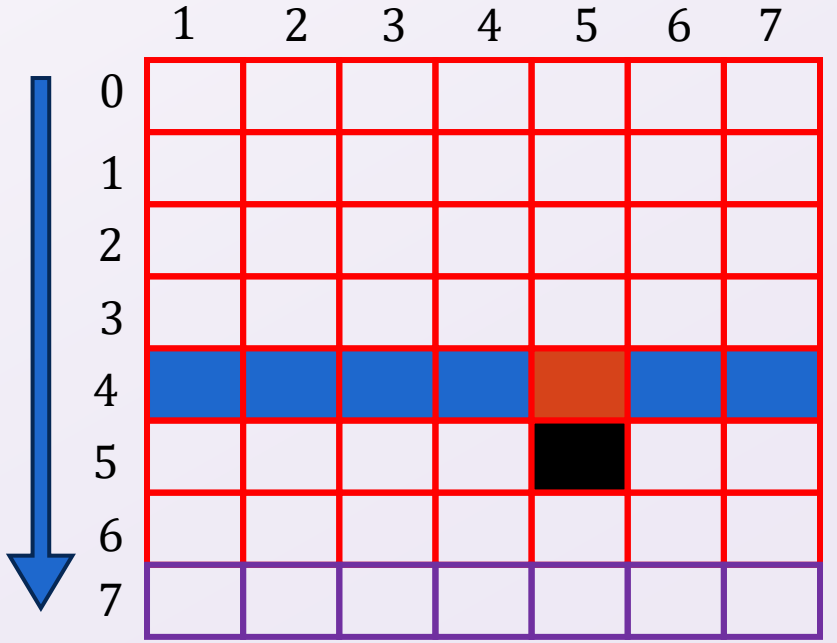
4. See if there’s a way to save space
   - Is it possible to reuse some memory locations?

Order of Evaluations

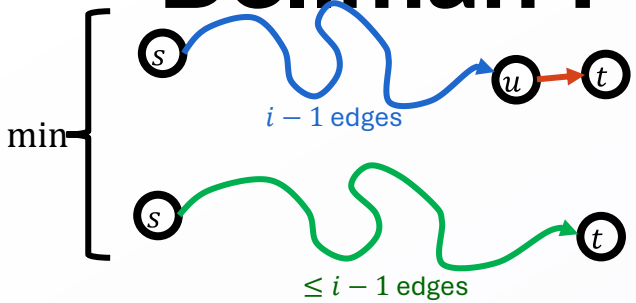
$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min\limits_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

Each cell depends *only* on values in the previous row

We only need two rows!



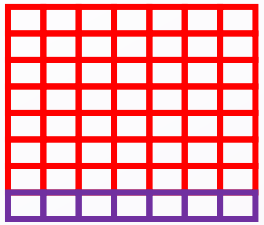
Bellman Ford– Four Steps



1. Formulate the answer with a recursive structure

   What are the options for the last choice?

   For each such option, what does the subproblem look like? How do we use it?



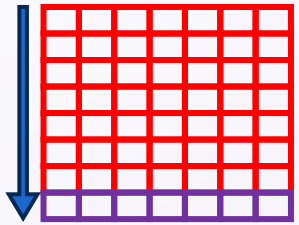
2. Choose a memory structure.

   Figure out the possible values of all parameters in the recursive calls.

   How many subproblems (options for last choice) are there?

   What are the parameters needed to identify each?

   How many different values could there be per parameter?

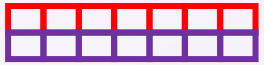


3. Specify an order of evaluation.

   Want to guarantee that the necessary subproblem solutions are in memory when you need them.

   With this step: a “Bottom-up” (iterative) algorithm

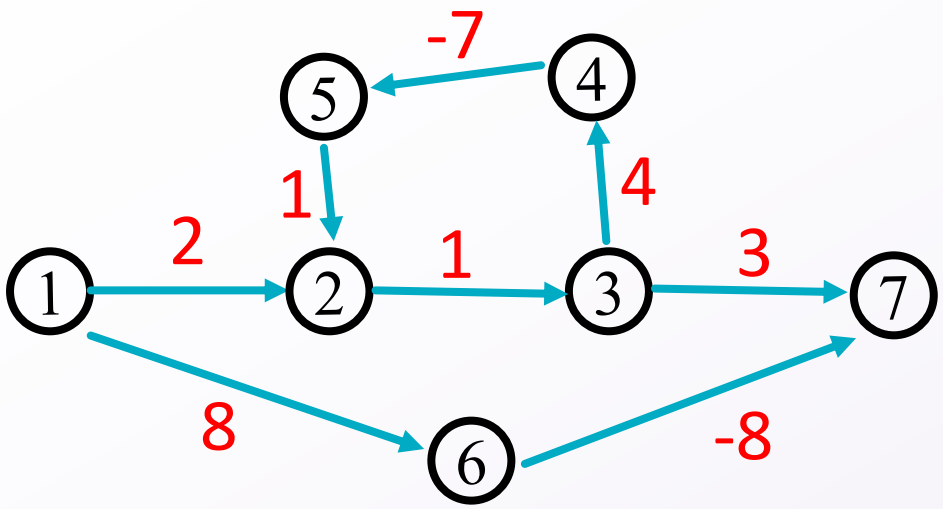
   Without this step: a “Top-down” (recursive) algorithm



4. See if there’s a way to save space

   Is it possible to reuse some memory locations?

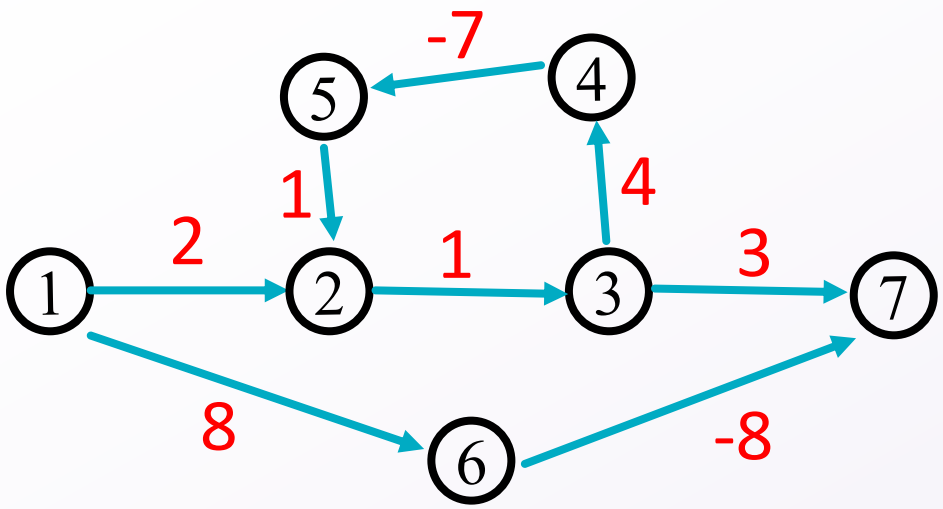
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1							
2							
3							
4							
5							
6							
7							

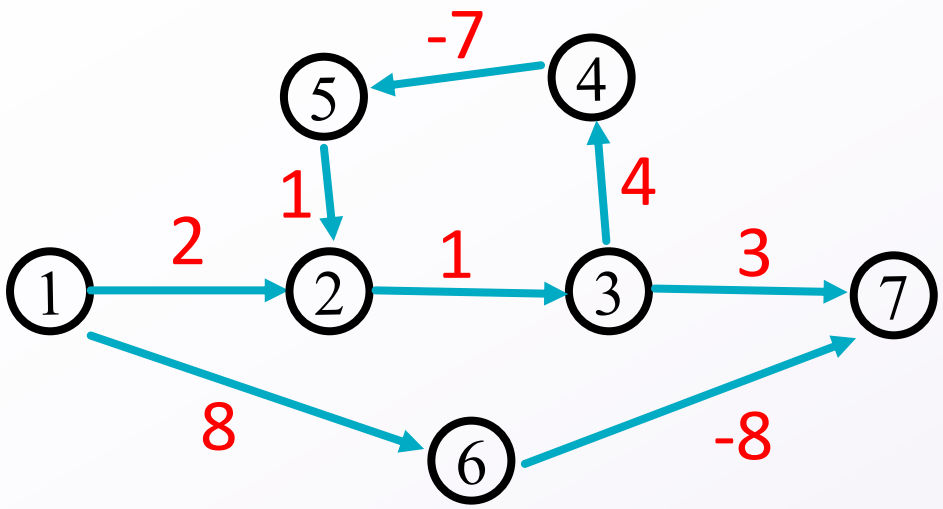
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2							
3							
4							
5							
6							
7							

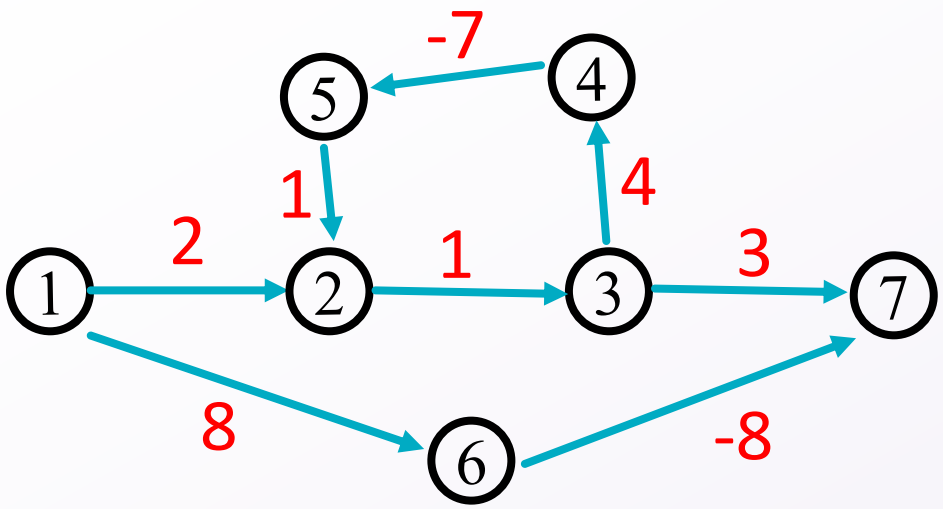
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3							
4							
5							
6							
7							

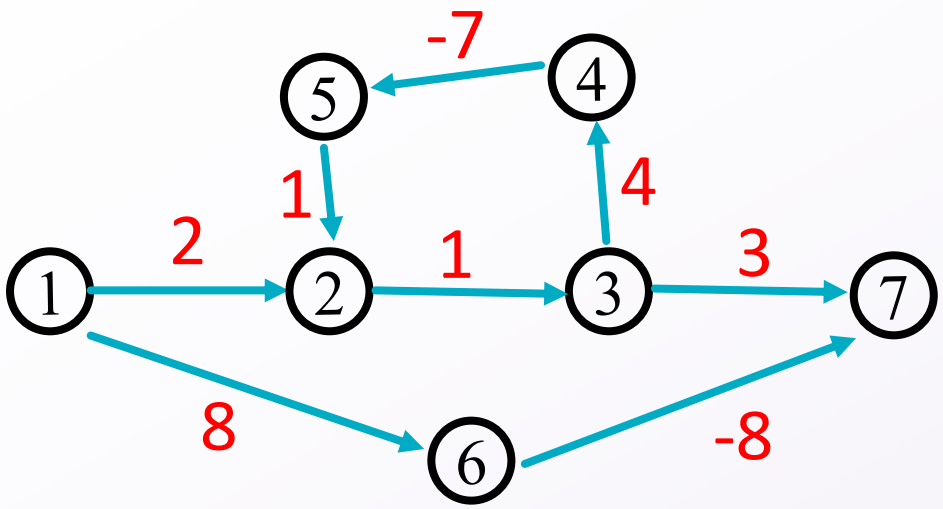
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4							
5							
6							
7							

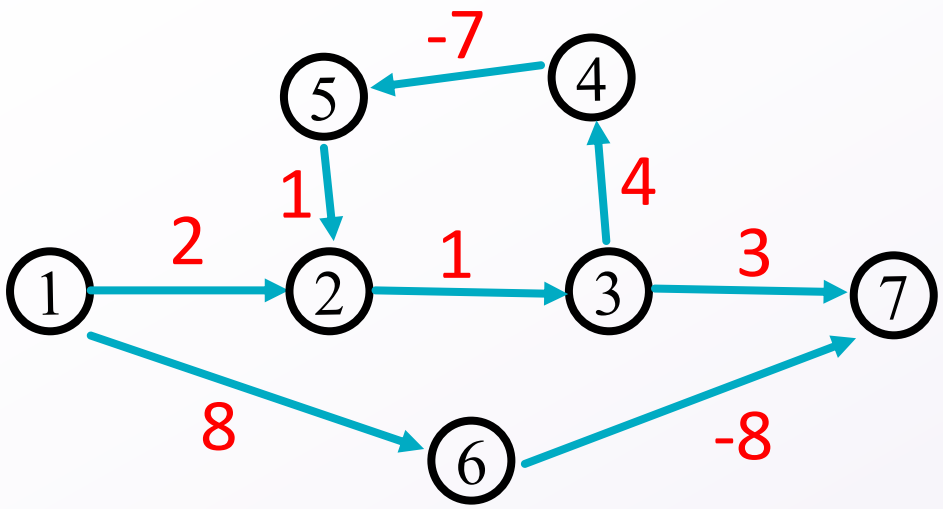
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	0	8	0
5							
6							
7							

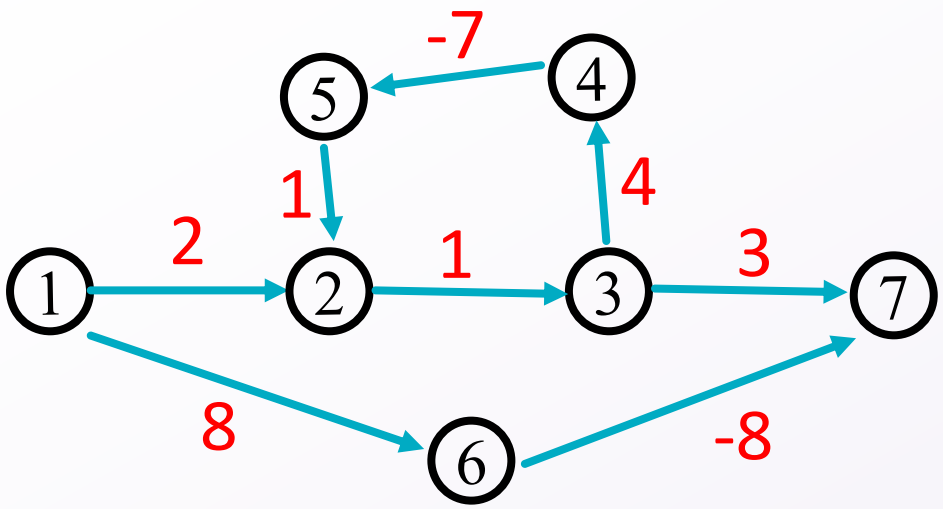
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	0	8	0
5	0	1	3	7	0	8	0
6							
7							

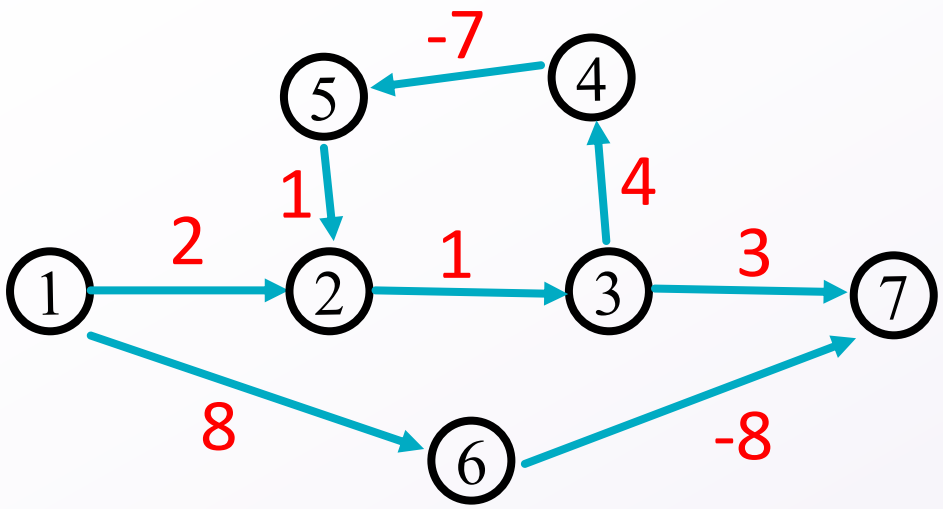
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	0	8	0
5	0	1	3	7	0	8	0
6	0	1	2	7	0	8	0
7							

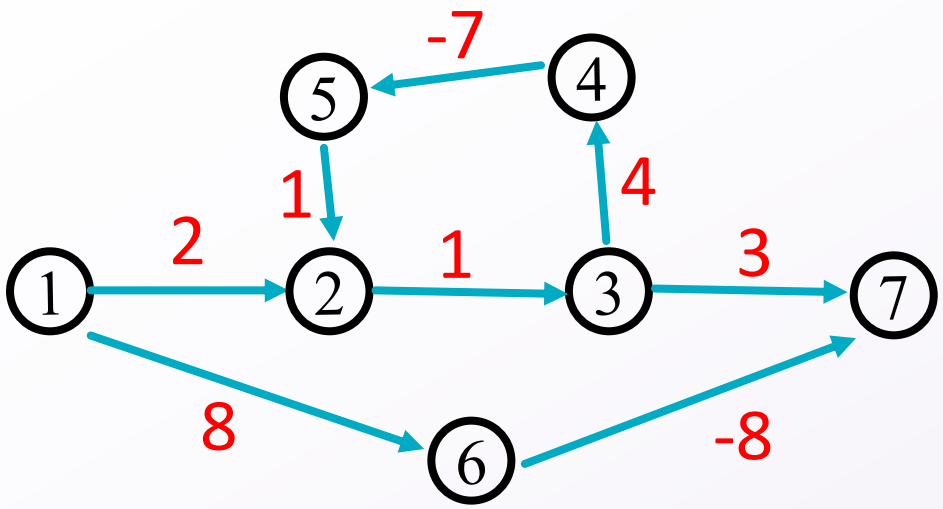
Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	0	8	0
5	0	1	3	7	0	8	0
6	0	1	2	7	0	8	0
7	0	1	2	3	0	8	0

Example Execution



$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	0	8	0
5	0	1	3	7	0	8	0
6	0	1	2	7	0	8	0
7	0	1	2	3	0	8	0

Negative Cycle Found!

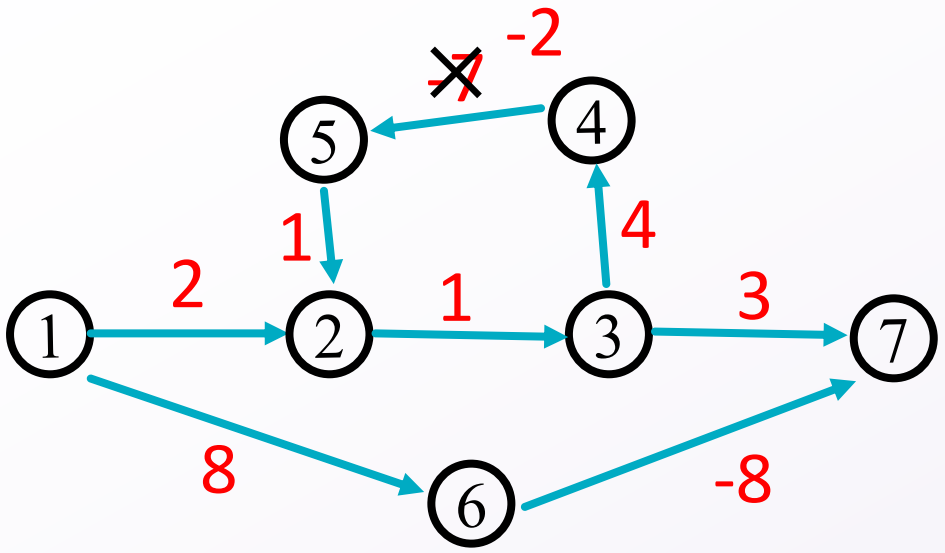
Bellman-Ford vs Dijkstras (1/2)

	Bellman-Ford	Dijkstras
Input	Any Weighted Graph	Graph with non-negative edge weights
Worst Case running time n =# nodes m =#edges	$O(nm)$	$O(n \log m)$
Conclusion	Use whenever you have negative weights?	Use whenever you don't have negative weights?

Bellman-Ford vs Dijkstras (2/2)

	Bellman-Ford	Dijkstras
Input	Any Weighted Graph	Graph with non-negative edge weights
Worst Case running time n =# nodes m =#edges	$O(nm)$	$O(n \log m)$
Dynamic graph (edges change)	Easy to do!	Need to re-solve from scratch!
Conclusion	Use whenever you have negative weights, or when you're graph changes over time	Use whenever you don't have negative weights and when your graph doesn't change (so you just need to solve once)

Changing an Edge



When 1 edge changes, we don't necessarily need to update the whole table!

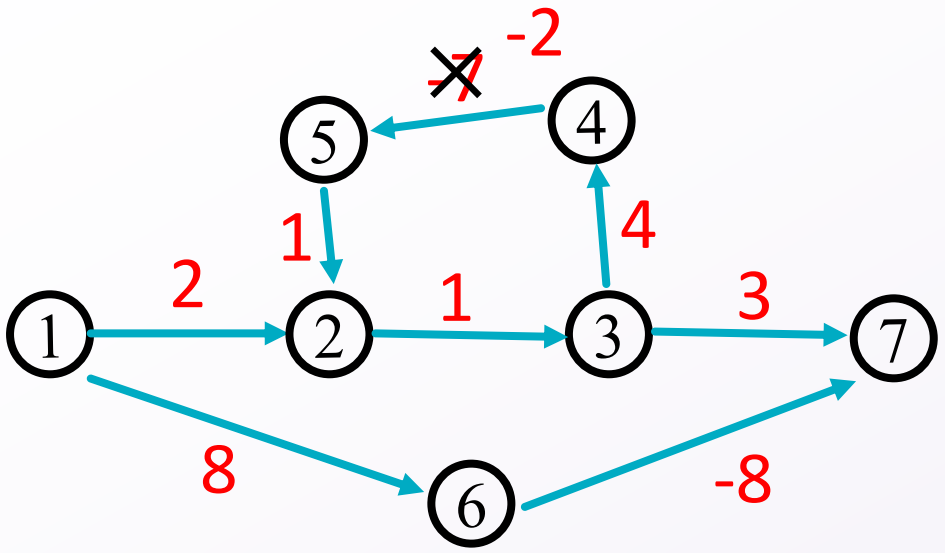
We only need to change cells where this edge is the one that changed (and any that depend on that edge)

Start with cells where $t = 5$

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	0	8	0
5	0	1	3	7	0	8	0
6	0	1	2	7	0	8	0
7	0	1	2	3	0	8	0

Updating the Table



When 1 edge changes, we don't necessarily need to update the whole table!

We only need to change cells where this edge is the one that changed (and any that depend on that edge)

Start with cells where $t = 5$

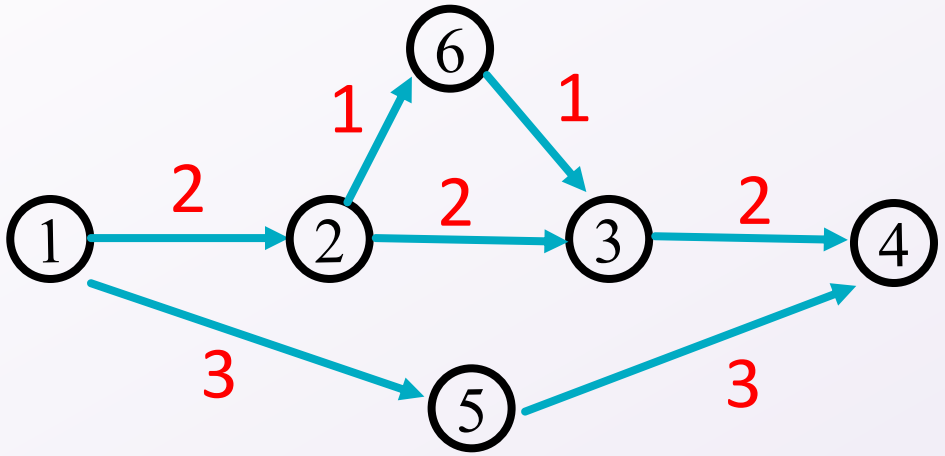
	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	2	∞	∞	∞	8	∞
2	0	2	3	∞	∞	8	0
3	0	2	3	7	∞	8	0
4	0	2	3	7	**5**	8	0
5	0	**6**	3	7	0	8	0
6	0	1	**7**	7	0	8	0
7	0	1	2	**7**	0	8	0

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min\begin{cases} \min\limits_{u \in V}\{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

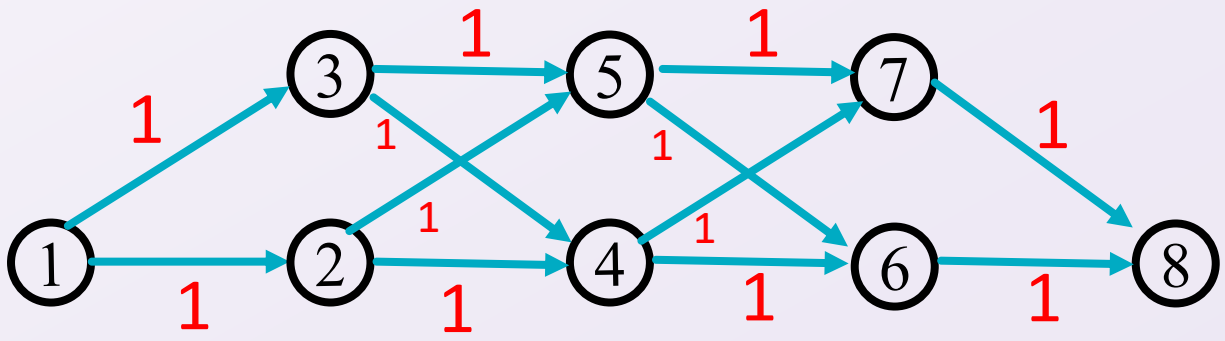
Counting Shortest Paths

If the shortest path from s to t has cost c, find the total number of s to t paths that have cost c

3 shortest paths from 1 to 4



8 shortest paths from 1 to 8



Counting Shortest Paths - Idea

If a shortest path from s to t uses edge (u, t) then then we can add (u, t) to every shortest path to u to produce a shortest path to t.

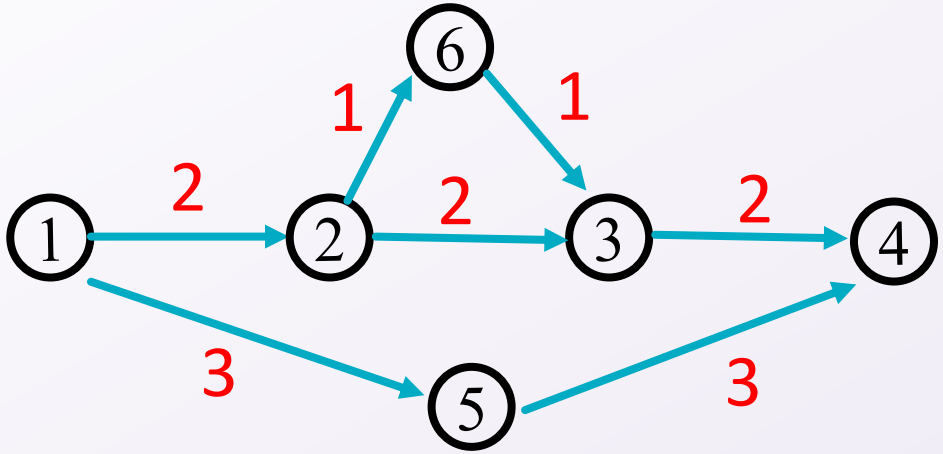
$(3,4)$ is in a shortest path to 4

$(5,6)$ is in a shortest path to 4

The number of shortest paths to 4 is at least:

The number of shortest paths to 3

plus the number of shortest paths to 5



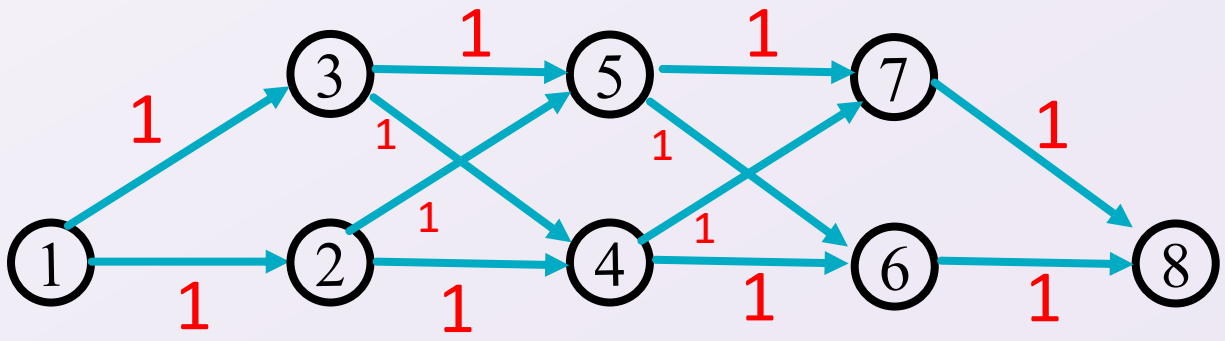
$(7,8)$ is in a shortest path to 8

$(6,8)$ is in a shortest path to 8

The number of shortest paths to 8 is at least

The number of shortest paths to 7

plus the number of shortest paths to 6



Recursive Structure – Bellman-Ford

$OPT(i,t)$ = the weight of the shortest path from s to t with at most i edges

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

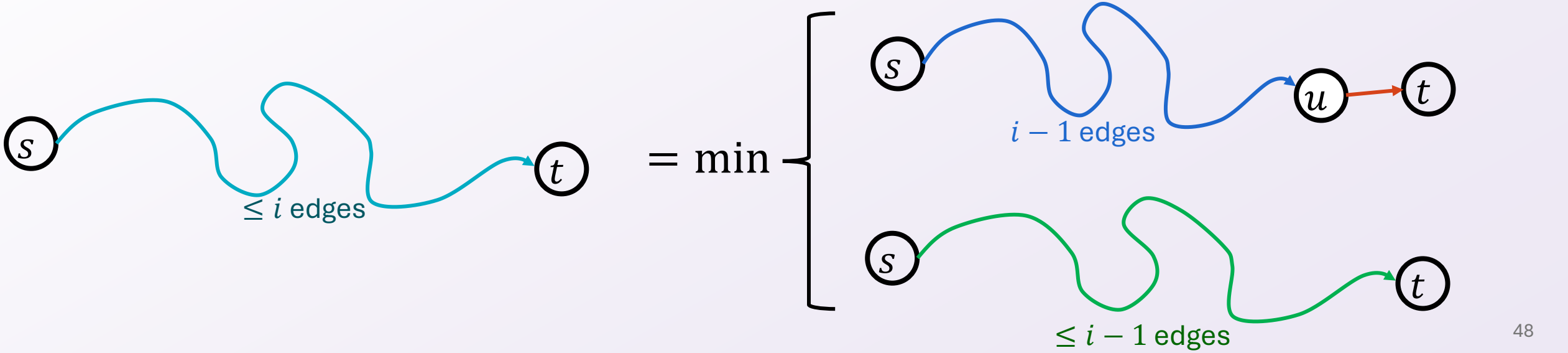
Where $w(u,t)$ is the weight of the edge from u to t if it exists and ∞ if not.

$OPT(i-1,u)$ keeps the shortest path from s to u using at most $i-1$ edges

Every shortest path to t using $\leq i$ edges will look like $OPT(i-1,u) + w(u,t)$.

Idea: Add together the count of shortest paths to u where:

$$OPT(i-1,u) + w(u,t) = OPT(i,t)$$



Recursive Structure – # Shortest Paths

$OPT(i,t)$ = the weight of the shortest path from s to t with at most i edges
$COUNT(i,t)$ = the number of shortest paths from s to t with at most i edges

$$OPT(i,t) = \begin{cases} 0 \text{ if } i = 0 \text{ and } s = t \\ \infty \text{ if } i = 0 \text{ and } s \neq t \\ \min \begin{cases} \min_{u \in V} \{OPT(i-1,u) + w(u,t)\} \\ OPT(i-1,t) \end{cases} \end{cases}$$

$$COUNT(i,t) = \begin{cases} 1 \text{ if } i = 0 \text{ and } s = t \\ 0 \text{ if } i = 0 \text{ and } s \neq t \\ \sum_{u \text{ s.t. } OPT(i-1)+w(u,t)=OPT(i,t)} COUNT(i-1,u) \end{cases}$$

If edge (u,t) is used in a shortest path to t

Then every shortest path to u can be used in a shortest path to t

Top-Down # Shortest Paths

numShortest(i, t):

```
if COUNT[i][t] not blank:    // Check if we've solved this already
        return COUNT[i][t]
if i == 0:    // Check if this is a base case
        solution = 1 ? t == s : ∞
        COUNT[i][t] = solution    // Always save your solution before returning
        return solution
solution = 0
pathlength = BF(i, t) // Call Bellman-Ford to find the length of the shortest path
for each u ∈ V: // For each node in the graph
        if BF(i − 1, u) + w(u, t) == pathlength: // If (u, t) is on a shortest path
                solution += numShortest(i − 1, u) // Include number of shortest paths to u
COUNT[i][t] = solution    // Always save your solution before returning
return solution
```

Final reminders

HW5 released, due Friday @ 11:59pm.

Quiz feedback released, regrades requests needed by tomorrow @ 11:59pm

I have OH now-12:30pm:

- Meet at front of classroom, we'll walk over together
- CSE (Allen) 434 if you're coming later

Glenn has online OH 12–1pm