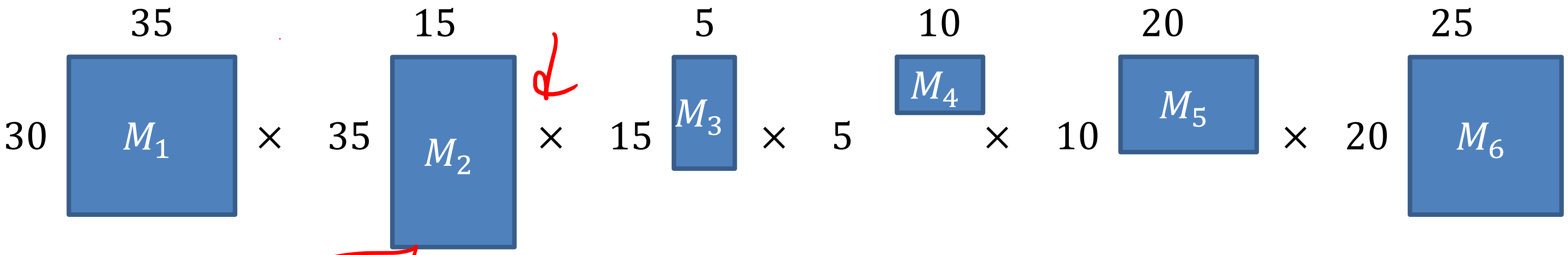
Warm Up

Any more Fall 2023 schedule questions?

3. Select a good order for solving subproblems

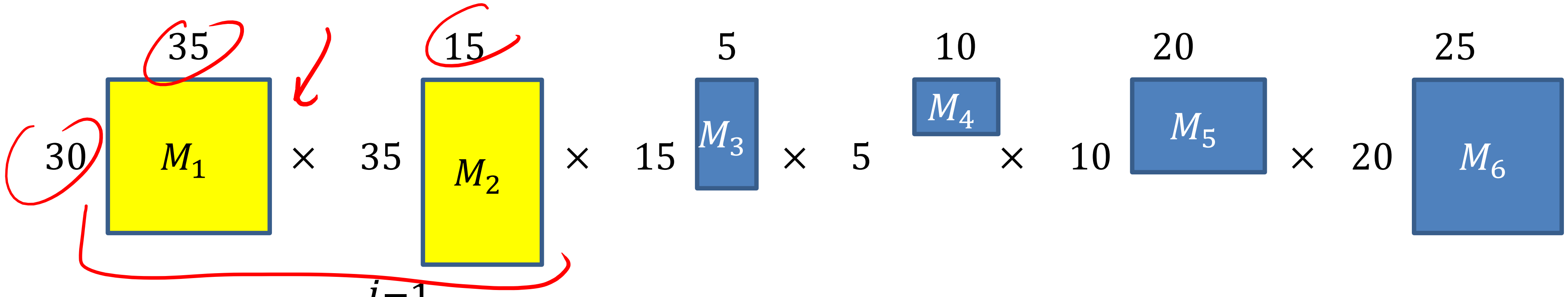


$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$ 1	2	3	4	5	6	$= i$
0						1
	0					2
		0				3
			0			4
				0		5
					0	6

3. Select a good order for solving subproblems



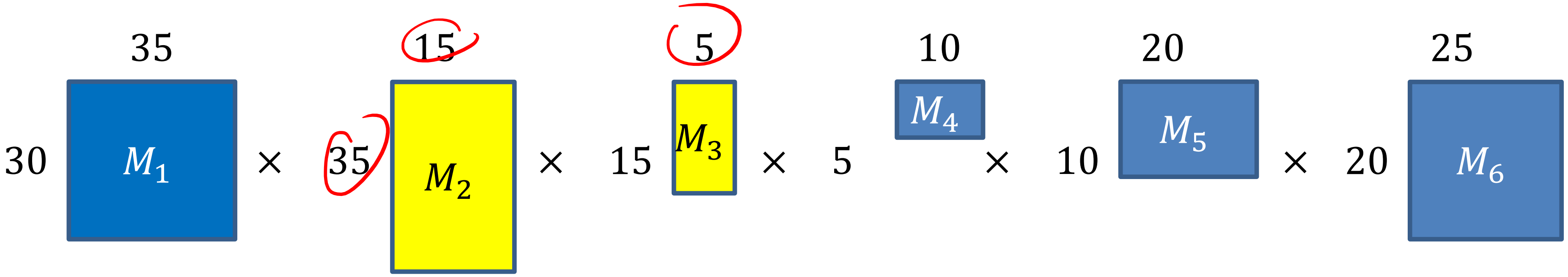
$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$= i$
	0	15750					1
		0					2
			0				3
				0			4
					0		5
						0	6

$$Best(1,2) = \min\left\{ Best(1,1) + Best(2,2) + r_1 r_2 c_2 \right.$$

3. Select a good order for solving subproblems



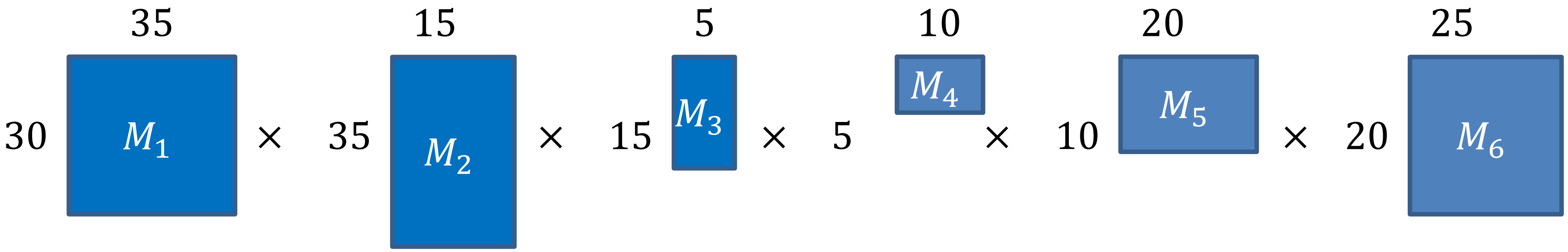
$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$i =$ \ $j =$	1	2	3	4	5	6
1	0	15750				
2		0	2625			
3			0			
4				0		
5					0	
6						0

$$Best(2,3) = \min \left\{ Best(2,2) + Best(3,3) + r_2 r_3 c_3 \right.$$

3. Select a good order for solving subproblems

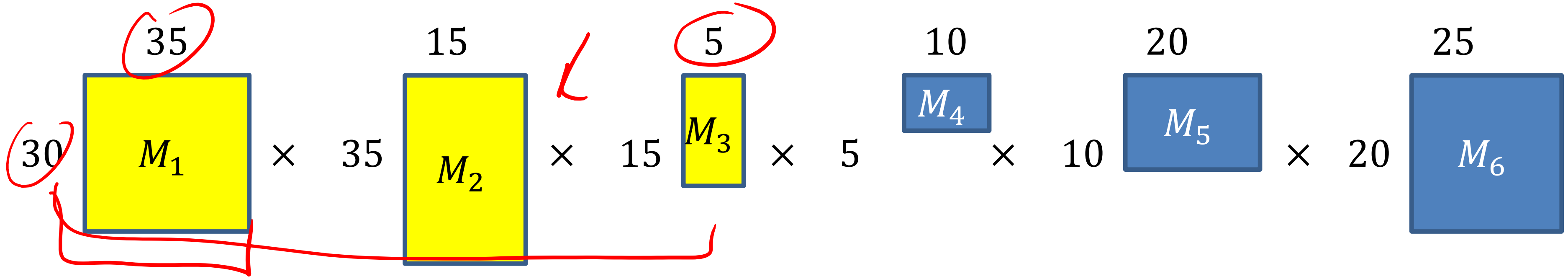


$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$= i$
	0	15750					1
		0	2625				2
			0	750			3
				0	1000		4
					0	5000	5
						0	6

3. Select a good order for solving subproblems



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

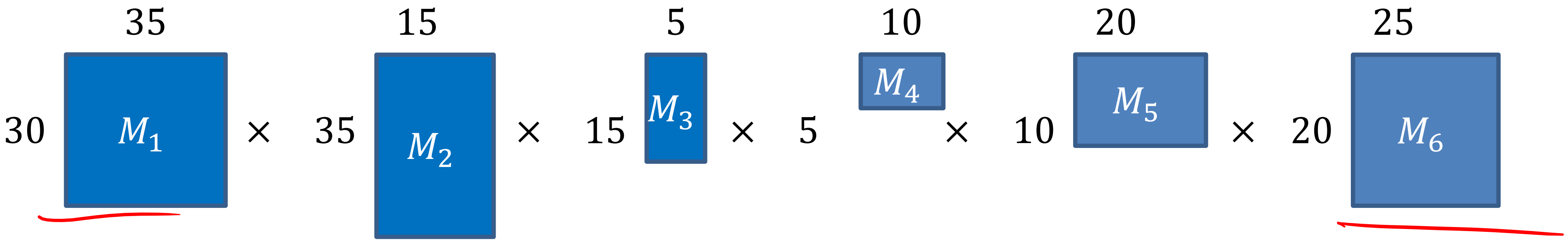
$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$= i$
	0	15750	7875				1
		0	2625				2
			0	750			3
				0	1000		4
					0	5000	5
						0	6

$$r_1 r_2 c_3 = 30 \cdot 35 \cdot 5 = 5250$$
$$r_1 r_3 c_3 = 30 \cdot 15 \cdot 5 = 2250$$

$$Best(1,3) = \min\begin{cases} \overbrace{Best(1,1)}^{0} + \overbrace{Best(2,3)}^{2625} + r_1 r_2 c_3 \\ \underbrace{Best(1,2)}_{15750} + \underbrace{Best(3,3)}_{0} + r_1 r_3 c_3 \end{cases}$$

3. Select a good order for solving subproblems



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$ 1	2	3	4	5	6	$= i$
0	15750	7875				1
	0	2625				2
		0	750			3
			0	1000		4
				0	5000	5
					0	6

To find $Best(i,j)$: Need all preceding terms of row i and column j

Conclusion: solve in order of diagonal

Matrix Chain Multiplication

	35		15		5		10		20		25
30	M_1	$\times$ 35	M_2	$\times$ 15	M_3	$\times$ 5	M_4	$\times$ 10	M_5	$\times$ 20	M_6

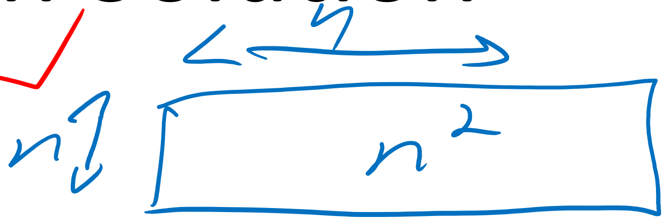
$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$ 1	2	3	4	5	6	$= i$
0	15750	7875	9375	11875	15125	1
	0	2625	4375	7125	10500	2
		0	750	2500	5375	3
			0	1000	3500	4
				0	5000	5
					0	6

$$Best(1,6) = \min \begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 & 1 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 & 2 \\ Best(1,3) + Best(4,6) + r_1 r_4 c_6 & 3 \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 & 4 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 & 5 \end{cases}$$

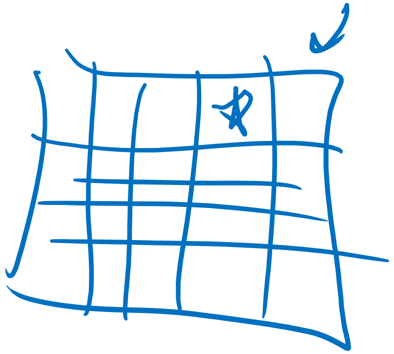
Top Down Solution




```
Mem = nxn
Def Best(i,j):
        if mem[i][j] >= 0:                    1
                return mem[i][j]
        if I == j:                            1
                mem[i][j] = 0
                return 0
        minimum = infinity                    1
        for x from i to j-1:                  x n
                left = Best(i,x)              1
                right = Best(x+1,j)           1
                answer = left + right + r_i * r_x * c_j    1
                minimum = min(minimum, answer)             1
        mem[i][j] = minimum                   1
        return minimum                        1
```

$\Theta(n^3)$

$$Best(1,n) = \min \begin{cases} Best(2,n) + r_1 r_2 c_n \\ Best(1,2) + Best(3,n) + r_1 r_3 c_n \\ Best(1,3) + Best(4,n) + r_1 r_4 c_n \\ Best(1,4) + Best(5,n) + r_1 r_5 c_n \\ \dots \\ Best(1,n-1) + r_1 r_n c_n \end{cases}$$



1) How big is the memory?

n^2

2) how much work does each location require?

n

Backtrack to find the best order

"remember" which choice of k was the minimum at each cell

$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

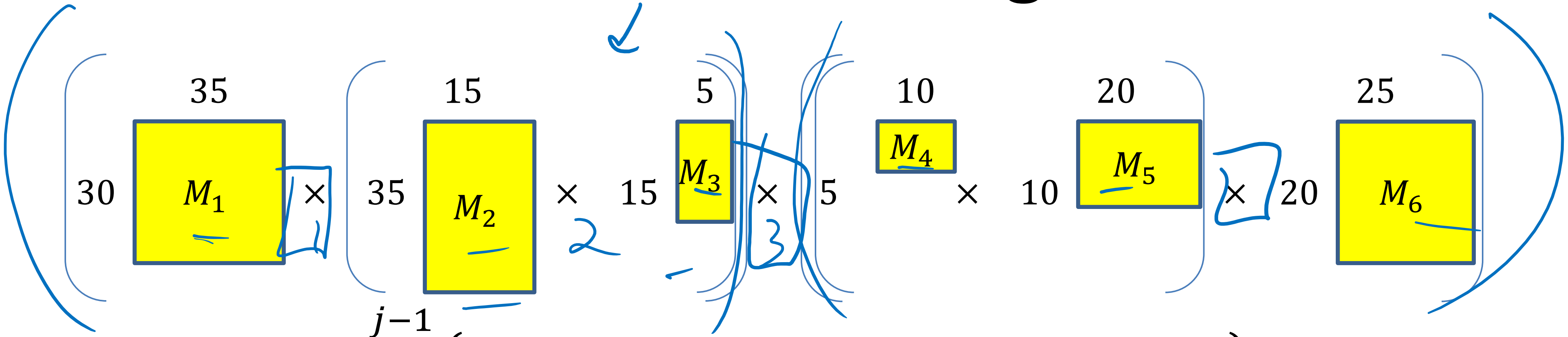
$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$= i$
	0	15750	7875 1	9375	11875	15125 3	1
		0	2625	4375	7125	10500	2
			0	750	2500	5375	3
				0	1000	3500 5	4
					0	5000	5
						0	6

$$Best(1,6) = \min \begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 \\ \boxed{Best(1,3) + Best(4,6) + r_1 r_4 c_6} \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 \end{cases}$$

10

Matrix Chaining



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	
	0	15750	7875 1	9375	11875	15125 3	1
		0	2625	4375	7125	10500	2
			0	750	2500	5375	3
				0	1000	3500 5	4
					0	5000	5
						0	6

$$Best(1,6) = \min \begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 \\ \boxed{Best(1,3) + Best(4,6) + r_1 r_4 c_6} \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 \end{cases}$$

Storing and Recovering Optimal Solution

Backtracking

- Maintain table Choice[i,j] in addition to Best table
  - Choice[i,j] = k means the best “split” was right after M_k
  - Work backwards from value for whole problem, Choice[1,n]
  - Note: Choice[i,i+1] = i because there are just 2 matrices
- From our example: 1, 3 4, 6
  - Choice[1,6] = 3. So $[M_1\ M_2\ M_3]\ [M_4\ M_5\ M_6]$
  - We then need Choice[1,3] = 1. So $[(M_1)\ (M_2\ M_3)]$
  - Also need Choice[4,6] = 5. So $[(M_4\ M_5)\ M_6]$
  - Overall: $[(M_1)\ (M_2\ M_3)]\ [(M_4\ M_5)\ M_6]$

Currency Exchange

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

Arbitrage

Currency Exchange

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

Currency Exchange

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		[illegible]	0.6906068347
THB		[illegible]	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!

Free money!

Best Currency Exchange

Best way to transfer USD to MYR:

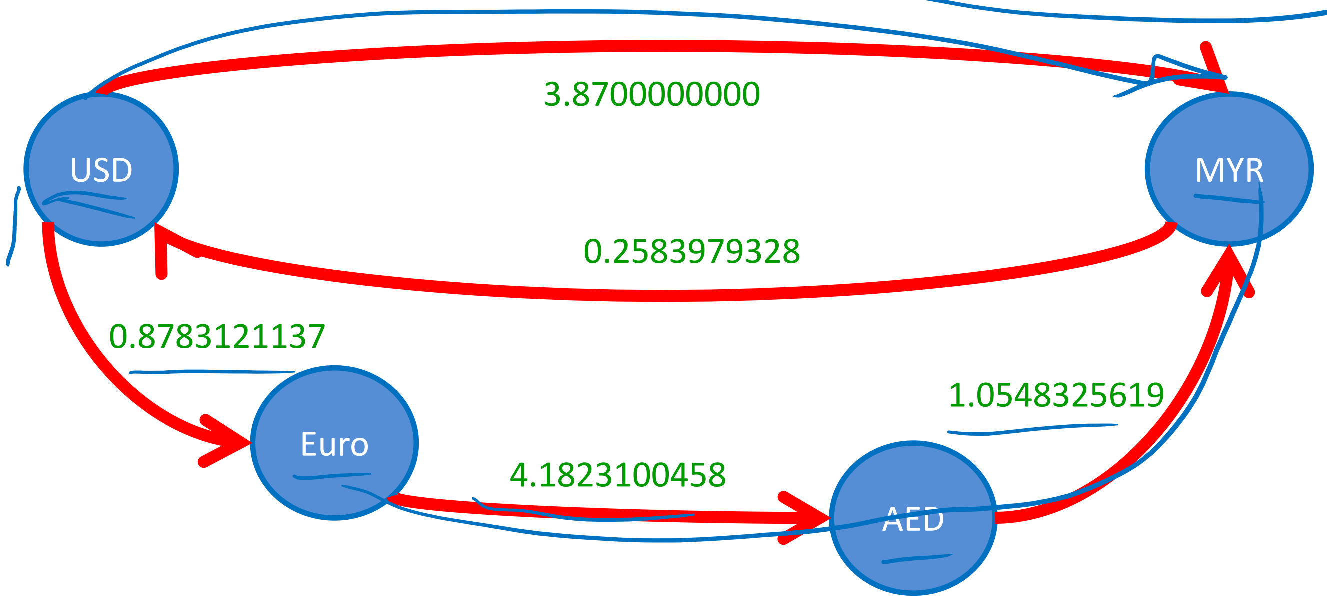
Given a graph of currencies (edges are exchange rates)

find the shortest path by product of edge weights

min $\frac{1}{}$

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

Invert edge weights to make it a minimization problem → min Σ



Best Currency Exchange

$x \cdot y \cdot z$

$\log(x \cdot y \cdot z)$

$= \log x + \log y + \log z$

Best way to transfer USD to MYR:

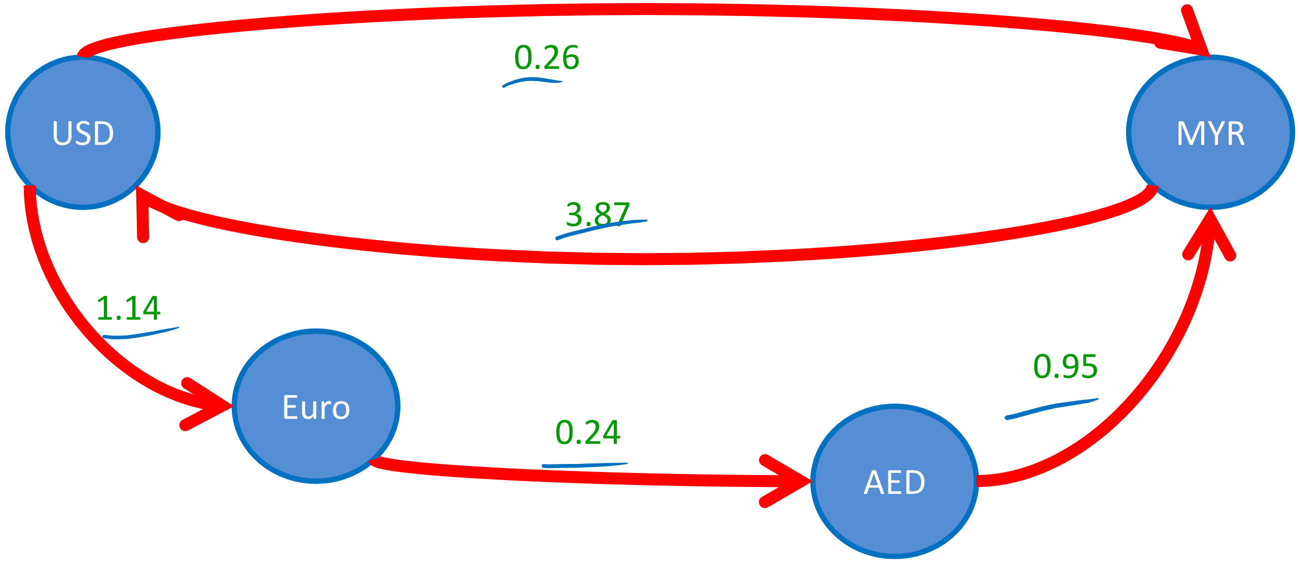
Given a graph of currencies (edges are exchange rates)

find the shortest path by product of edge weights

$$\frac{1}{x \cdot y \cdot z} = \frac{1}{x} \cdot \frac{1}{y} \cdot \frac{1}{z}$$

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

Take log of edge weights to make summation



Best Currency Exchange

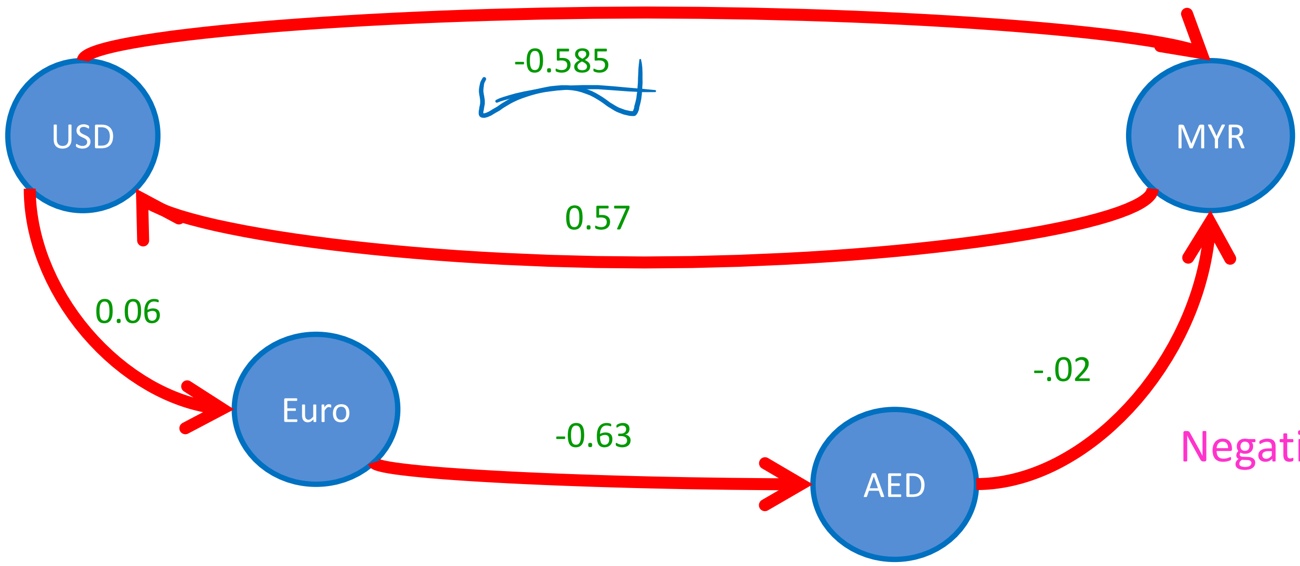
Best way to transfer USD to MYR:

Given a graph of currencies (edges are exchange rates)

find the shortest path by product of edge weights

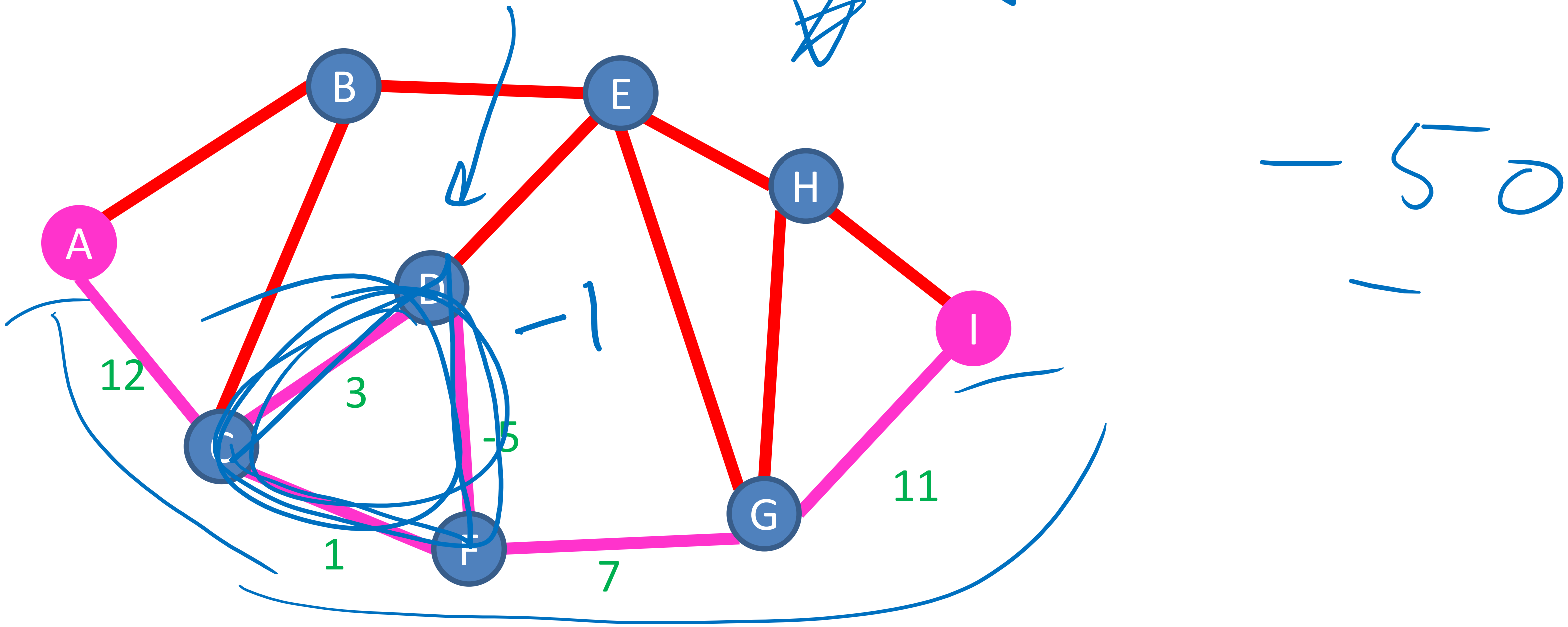
$$\underbrace{\min_{p}}\underbrace{\sum_{e\in p}}\underbrace{\log\frac{1}{w(e)}}$$

Now a shortest path problem!



Negative Edge Weights!

Problem with negative edges



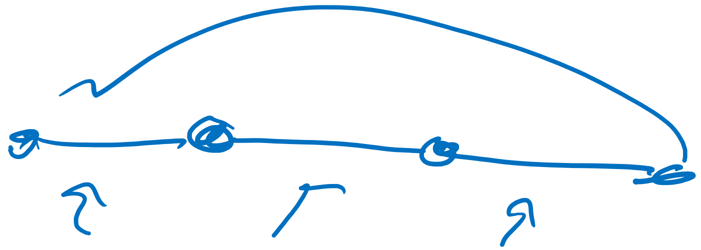
$w(C,F,D,C) = -1$

Weight if we take the cycle 0 times: 31
Weight if we take the cycle 1 time: 30
Weight if we take the cycle 2 times: 29
…

There is no shortest path from A to I!

What we need: an algorithm that finds the shortest path in graphs with negative edge weights (if one exists)

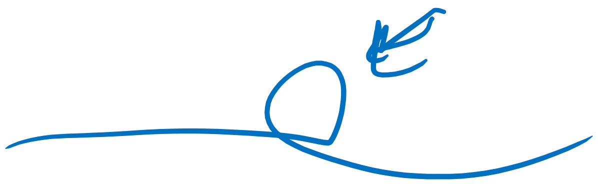
Note



Any simple path has at most $V-1$ edges

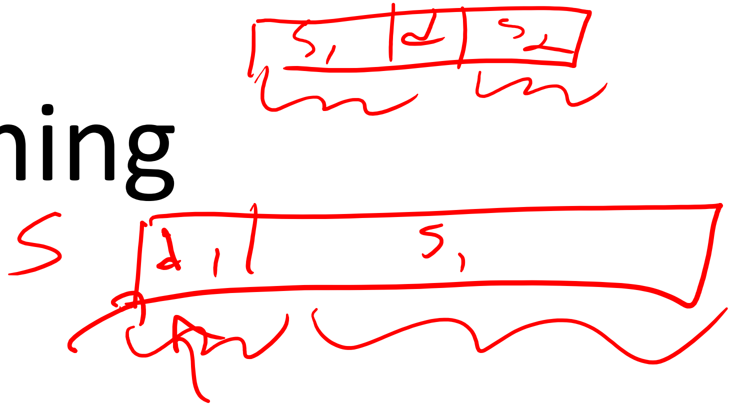
Pigeonhole Principle!

More than $V-1$ edges means some node appears twice (i.e., there is a cycle)



If there is a shortest path of more than $V-1$ edges, there is a negative weight cycle

Dynamic Programming



- Requires Optimal Substructure
  - Solution to larger problem is the (optimal) solutions to a smaller one plus one “decision”
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the “last thing” done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

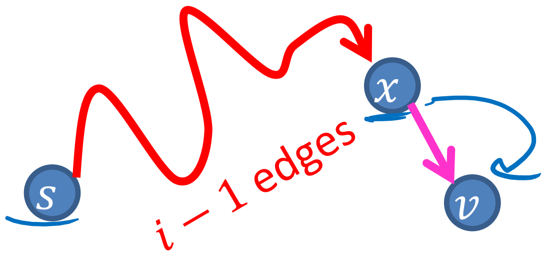
Bellman-Ford

Idea: Use Dynamic Programming!

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

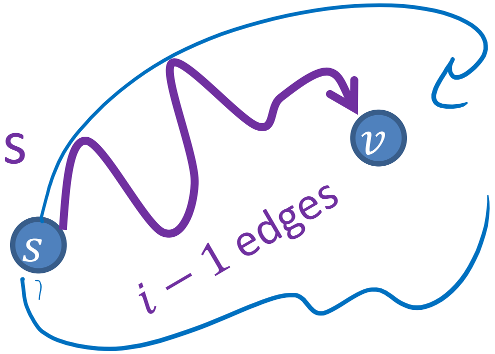
Two options:

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



$V+1$

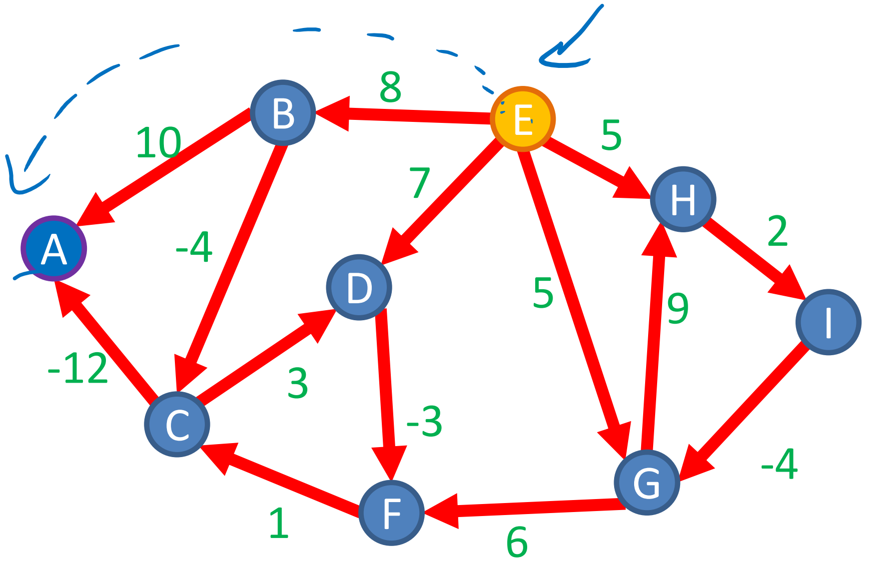
$$Short(i, v) = \min \begin{cases} \min\limits_{x} \left(Short(i-1, x) + w(x, v) \right) \\ Short(i-1, v) \end{cases}$$

Bellman Ford

Start node is E
Initialize all others to ∞

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

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



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

Bellman Ford

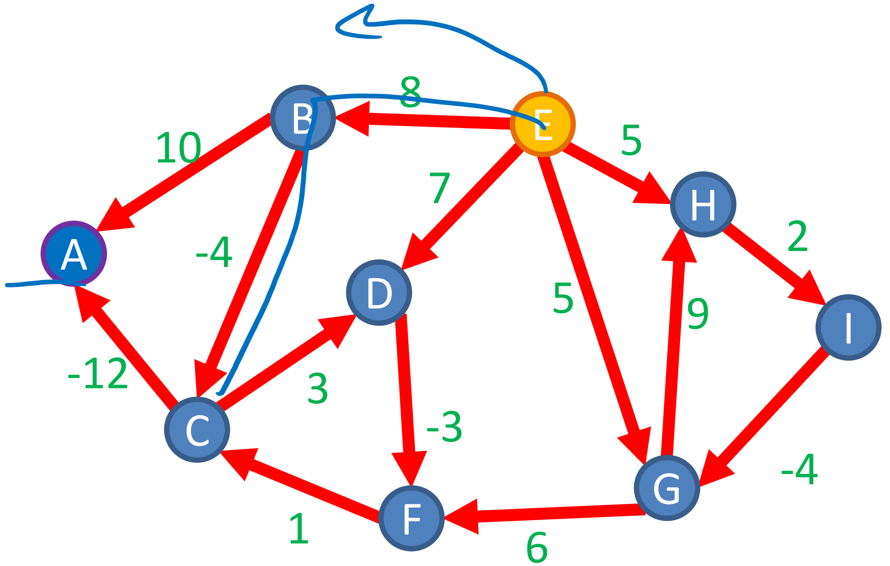
Start node is E
Initialize all others to ∞

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

B B, A ~~D~~

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

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



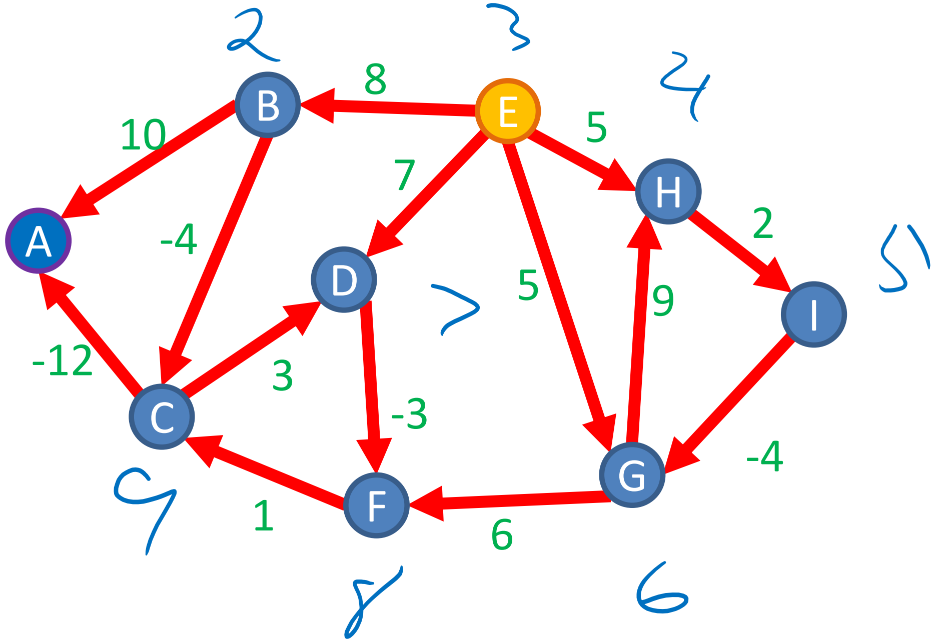
Bellman Ford

Start node is E
Initialize all others to ∞

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

$$Short(i, v) = \min \begin{cases} \min_x (Short(i-1, x) + w(x, v)) \\ Short(\underline{i-1}, v) \end{cases}$$

$i =$ \ $v =$	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									



shortest simple paths

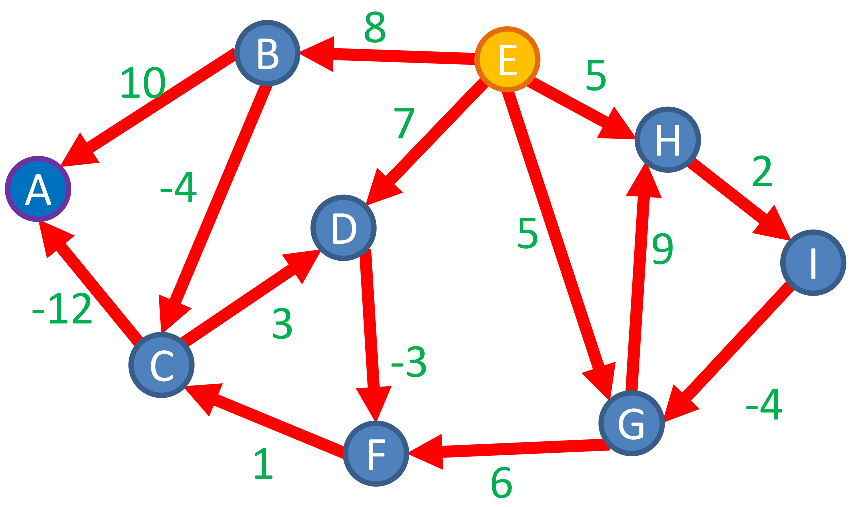
Bellman Ford

Start node is E
Initialize all others to ∞

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

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

$i =$ \ $v =$	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									



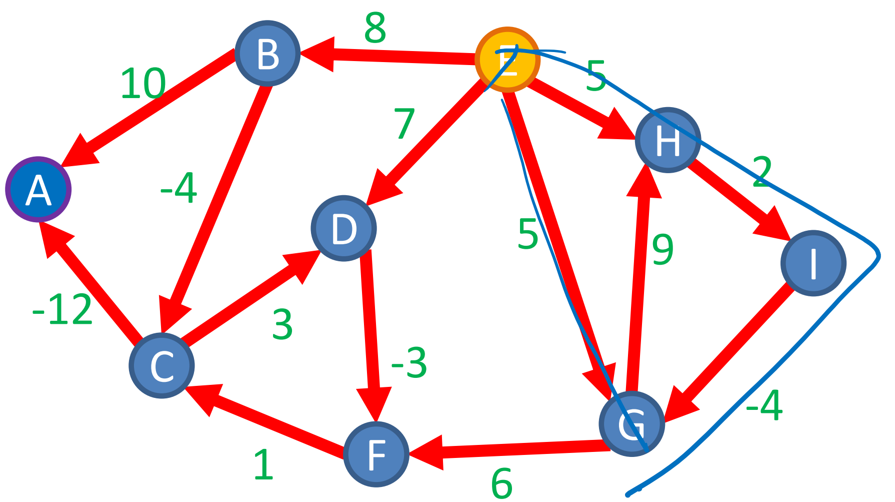
Bellman Ford

Start node is E
Initialize all others to ∞

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

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

$i =$ \ $v =$	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

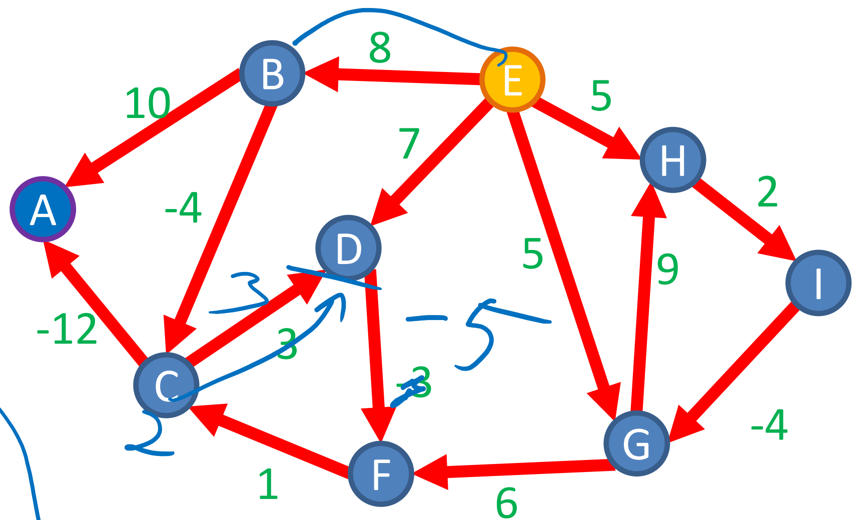


Bellman Ford: Negative Cycles

Start node is E
Initialize all others to ∞

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

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



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

If we computed row $|V|$, values change

There is a negative weight cycle!

Bellman Ford Top Down DP

```
Memory = (n+1)x(n) 2-d array of NULL
Def Bellman_Ford(graph, s, t):
        Memory[0][s] = 0
        return Short(length(graph),t)

Def Short(graph, i, v):
        if Memory[i][j] != NULL:
                return Memory[i][j]
        if i==0:
                 Memory[i][j] = infinity
                return infinity
        best = infinity
        for x from 0 to length(graph):
                new_path = Short(graph, i-1, x) + graph.weight(x,v)
                best = min(best, new_path)
        answer = min(best, Short(i-1,v))
        Memory[i][j] = answer
        return answer
```

Bellman Ford Bottom-up and Run Time

Intialize array $Short[V][V]$ V^2

Initialize $Short[0][v] = \infty$ for each vertex V

Initialize $Short[0][s] = 0$ 1

For $i = 1, \ldots, V-1$: V times

for each $e = (x, v) \in E$: E times

$Short[i][v] = \min\{$ 1

$Short[i-1][x] + w(x, v),$

$Short[i-1][v]\}$

$\Theta(V^2 + EV)$

$\Theta(EV)$

Why Use Bellman-Ford?

- Dijkstra's:
  - only works for positive edge weights
  - Run Time: $\Theta(E \log V)$
  - Not good for dynamic graphs (where edge weights are variable)
    - Must recalculate "from scratch"
- Bellman-Ford:
  - Works for negative edge weights
  - Run Time: $\Theta(E \cdot V)$
  - More efficient for dynamic graphs
    - $\Theta(E)$ time to recalculate

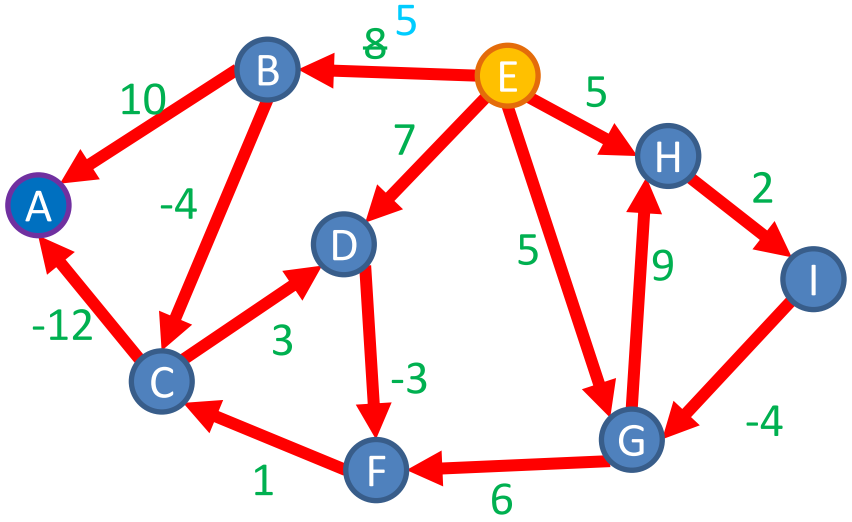
Bellman Ford

Each node will update its neighbors if edge weight changes

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

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

$i =$ \ $v =$	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



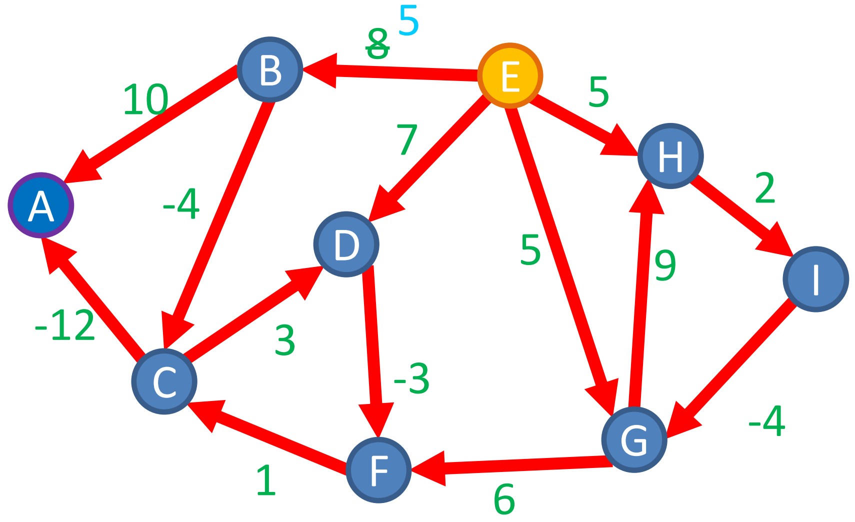
Bellman Ford

Each node will update its neighbors if edge weight changes

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

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

$i =$ \ $v =$	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	5	1	7	0	4	3	5	7
5	-8	5	1	7	0	4	3	5	7
6	-8	5	1	7	0	4	3	5	7
7	-8	5	1	7	0	4	3	5	7
8	-8	5	1	7	0	4	3	5	7



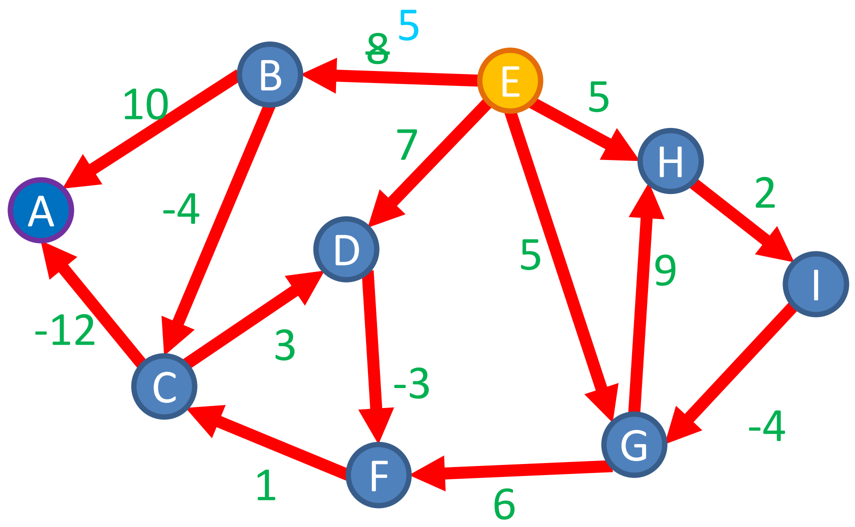
Bellman Ford

Each node will update its neighbors if edge weight changes

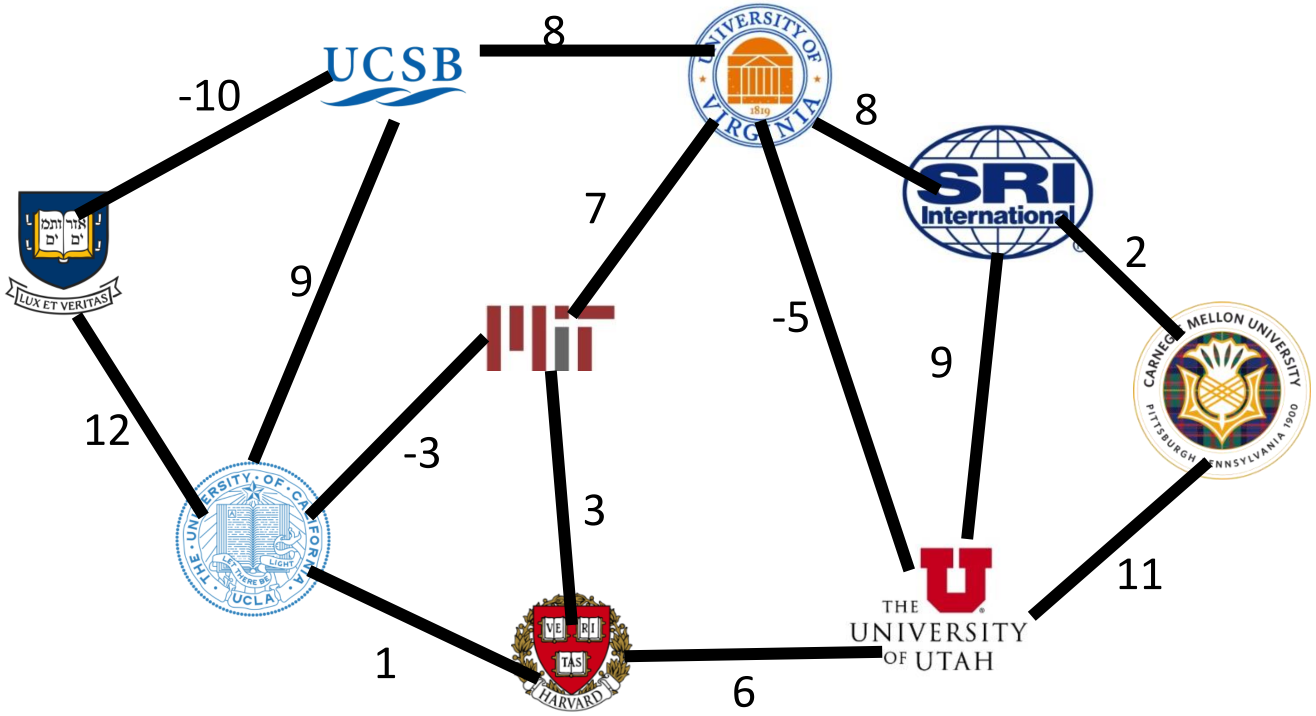
$$Short(i, v) = \text{weight of the shortest path from } s \text{ to } v \text{ using at most } i \text{ edges}$$

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

$i =$ \ $v =$	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	4	3	5	7
5	-11	5	1	4	0	4	3	5	7
6	-11	5	1	4	0	4	3	5	7
7	-11	5	1	4	0	4	3	5	7
8	-11	5	1	4	0	4	3	5	7



All-Pairs Shortest Path



Find the quickest way to get from each place to every other place

Given a graph $G = (V, E)$ for each start node $s \in V$ and destination node $v \in V$ find the least-weight path from $s \to v$

All-Pairs Shortest Path

- Can clearly be found in $O(V^2 \cdot E)$
  - Run Bellman-Ford with each node being the start

for each $s \in V$: V times

$BellmanFord(s)$ $O(V \cdot E)$

Floyd-Warshall

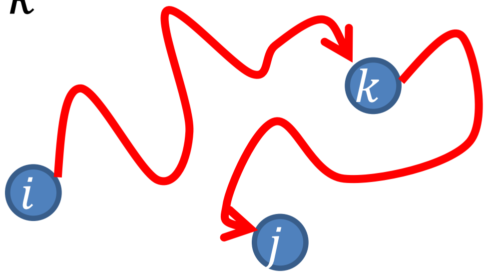
Finds all-pairs shortest paths in $\Theta(V^3)$

- Uses Dynamic Programming

1. Fix the ordering of nodes
2. $Short(i,j,k)$ is the length using only the first k nodes in that list

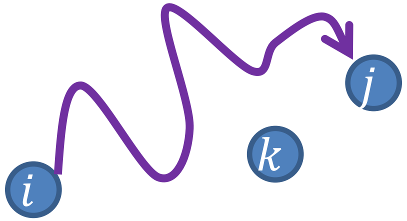
$Short(i,j,k) =$ the length of the shortest path from node i to node j using only intermediate nodes $1, \ldots, k$

Two options: Shortest path from i to j includes k



OR

Shortest path from i to j excludes k



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

Node at position k

$k-1$ is the index (first $k-1$ nodes can be used)

Floyd-Warshall Top-Down

Shortest Paths Review

- Single Source Shortest Paths
  - Dijkstra's Algorithm $\Theta(E \log V)$
    - No negative edge weights
  - Bellman-Ford $\Theta(EV)$
    - First Dynamic Programming Algorithm
    - Allows negative edge weights (finds negative weight cycles)
    - Update memory in $\Theta(E)$ time on edge weight updates
- All Pairs Shortest Paths
  - Floyd-Warshall $\Theta(V^3)$
    - Allows negative edge weights