COMP3251
Lecture 10: Bellman-Ford Algorithm
(Chapter 4.6)

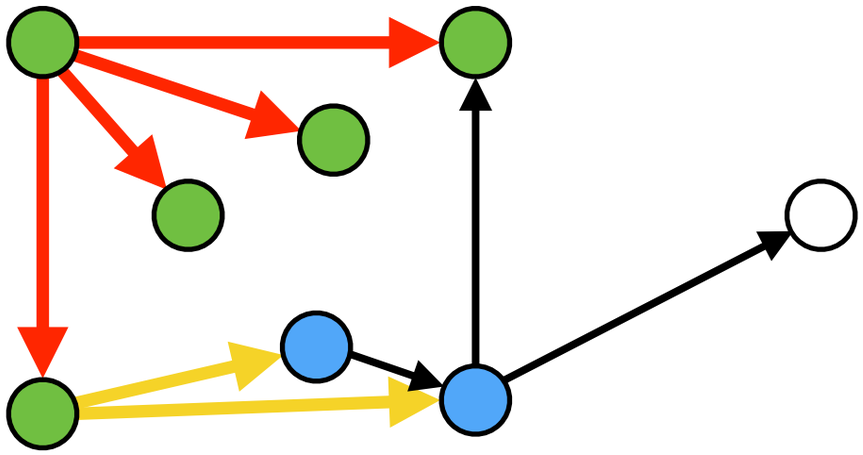
Recap: Single-Source Shortest Path

Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Recap: Single-Source Shortest Path

Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths

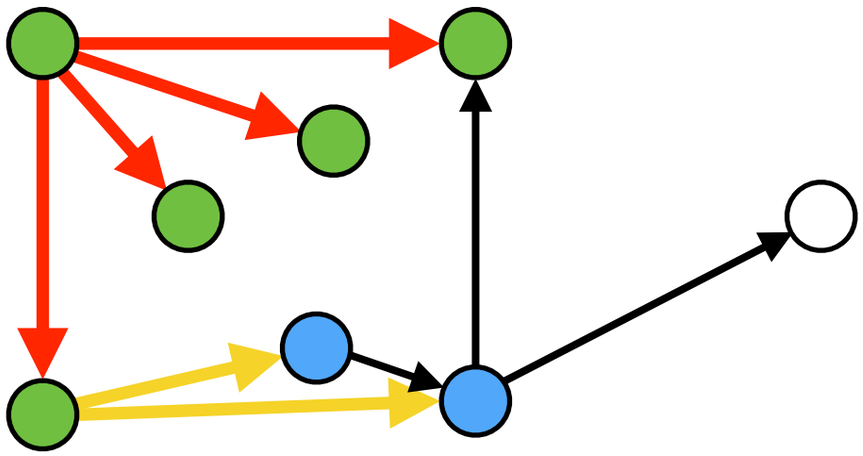


Breadth First Search

Recap: Single-Source Shortest Path

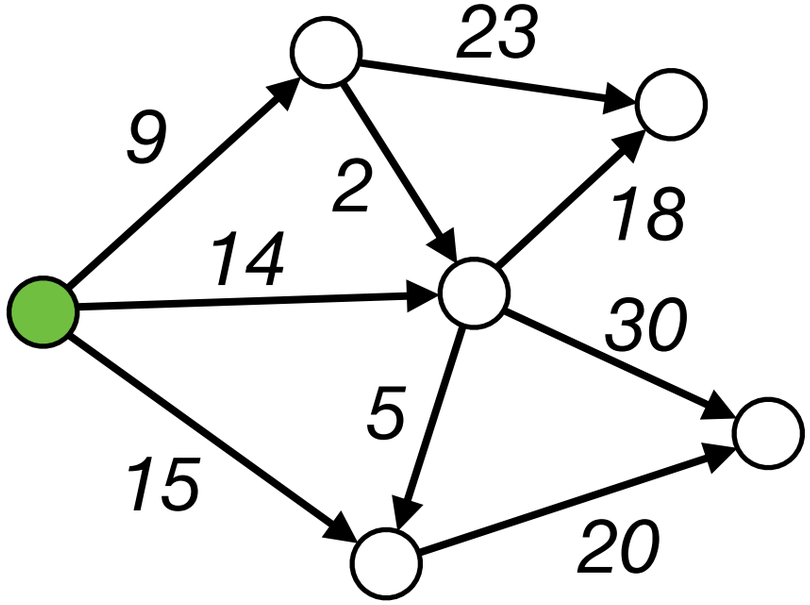
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

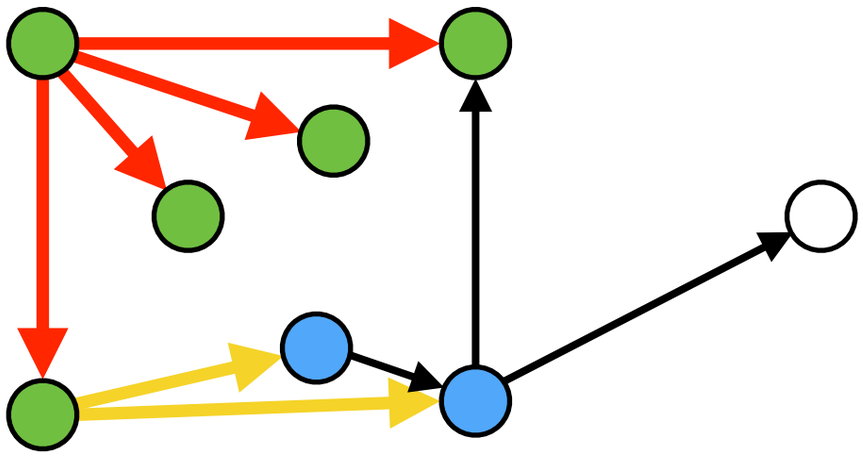


Dijkstra Algorithm

Recap: Single-Source Shortest Path

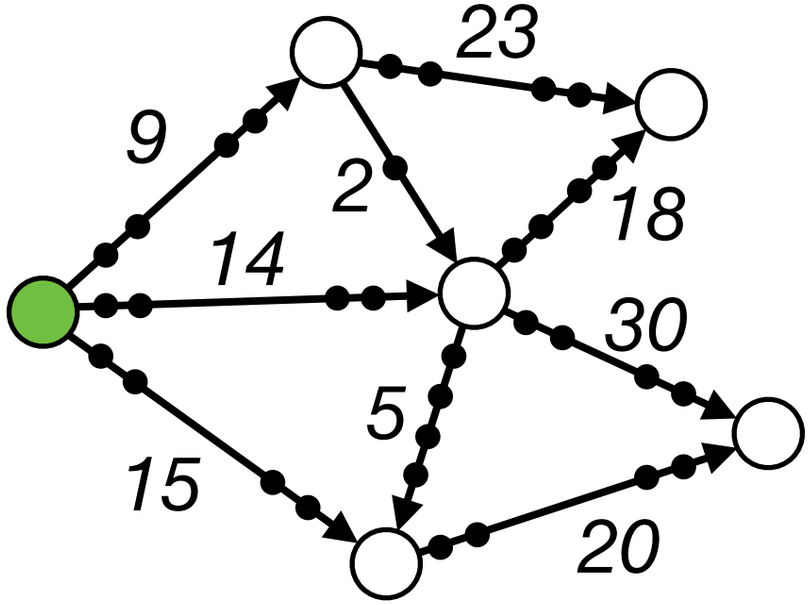
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

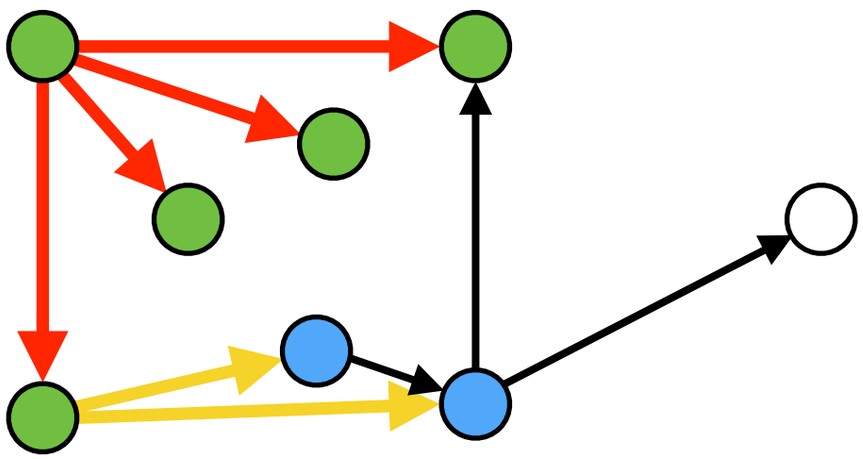


Dijkstra Algorithm

Recap: Single-Source Shortest Path

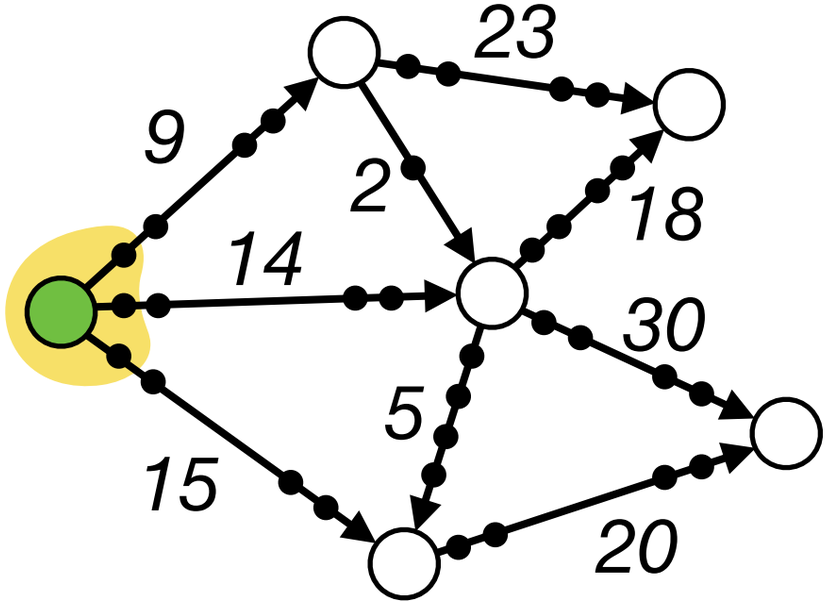
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

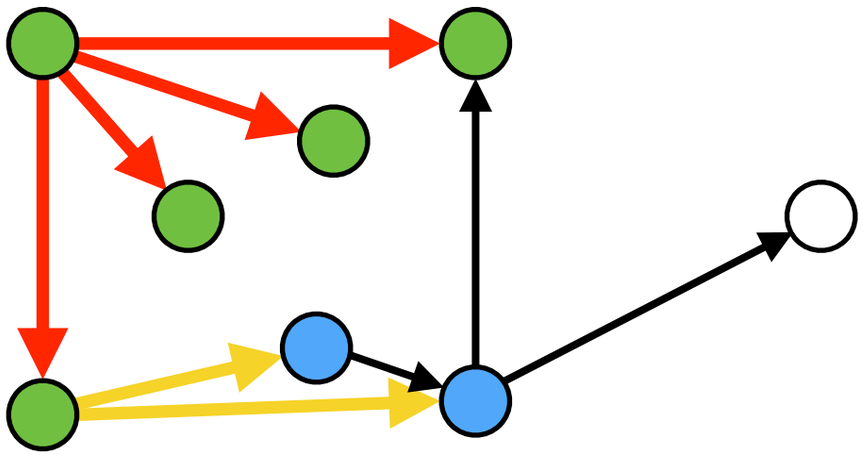


Dijkstra Algorithm

Recap: Single-Source Shortest Path

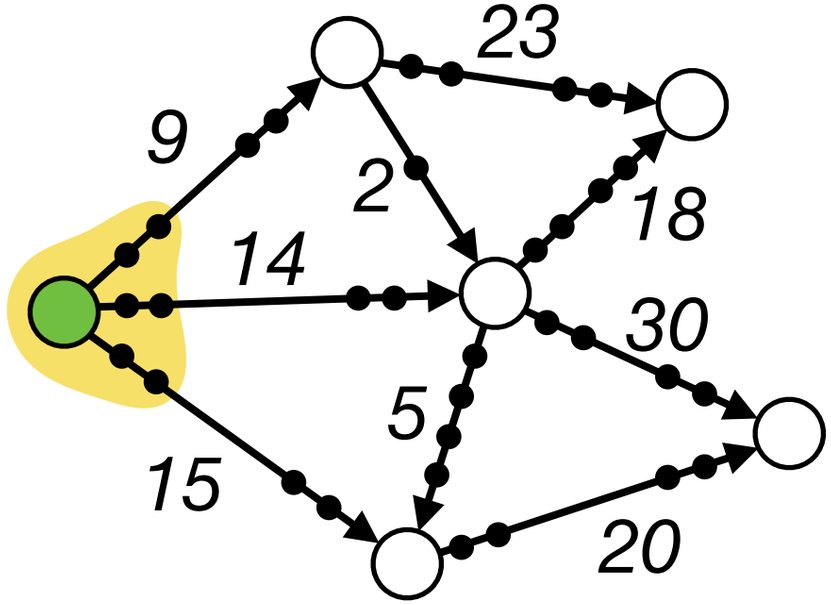
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

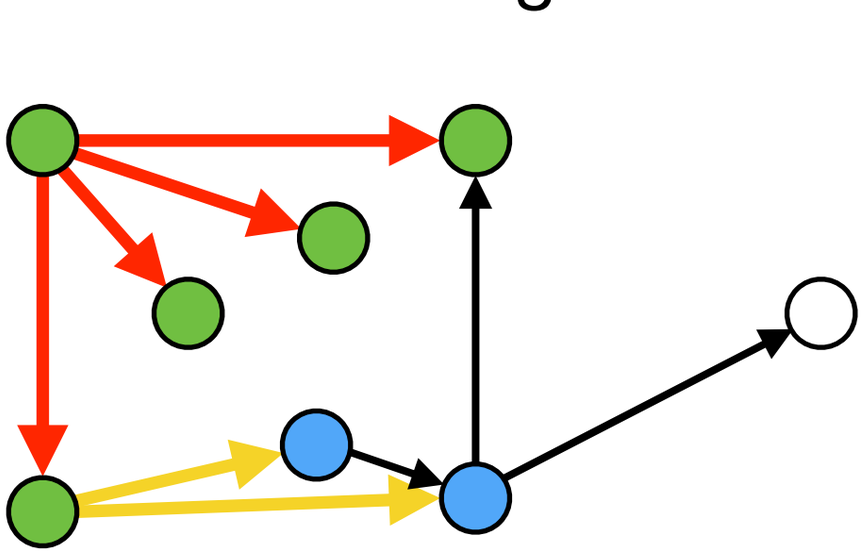


Dijkstra Algorithm

Recap: Single-Source Shortest Path

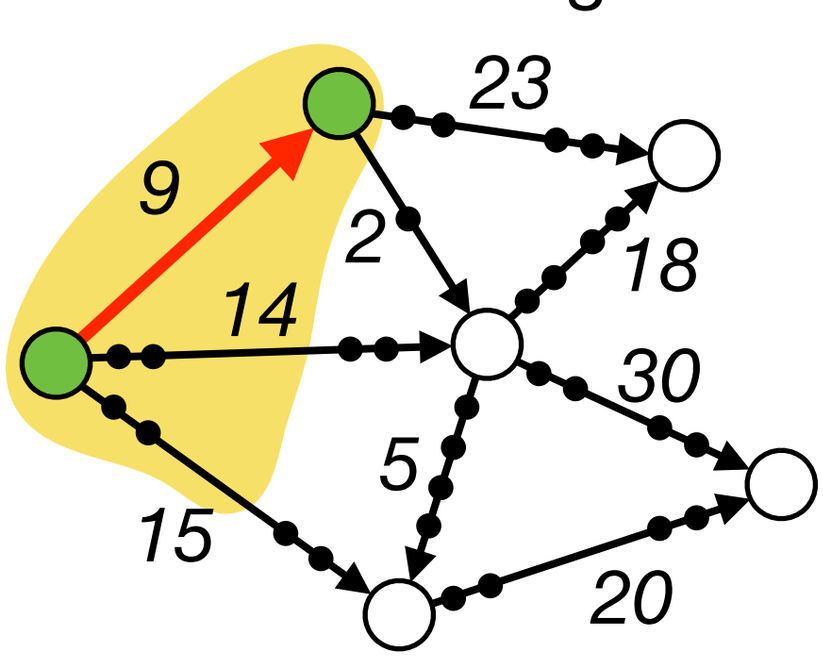
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

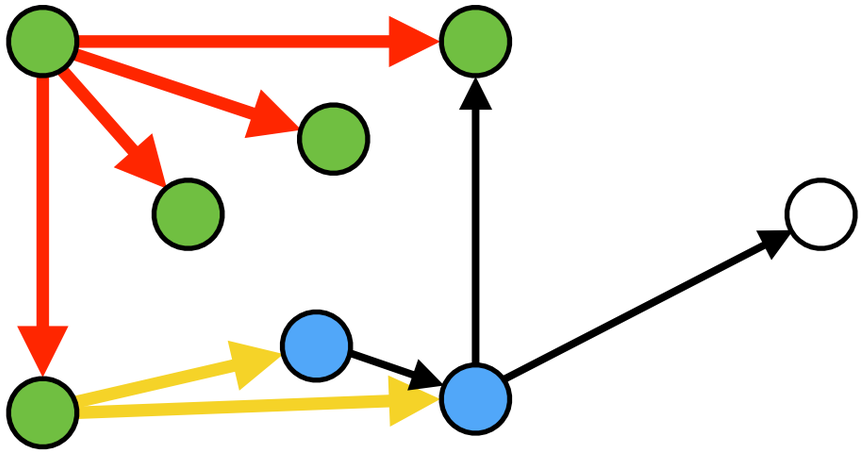


Dijkstra Algorithm

Recap: Single-Source Shortest Path

Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

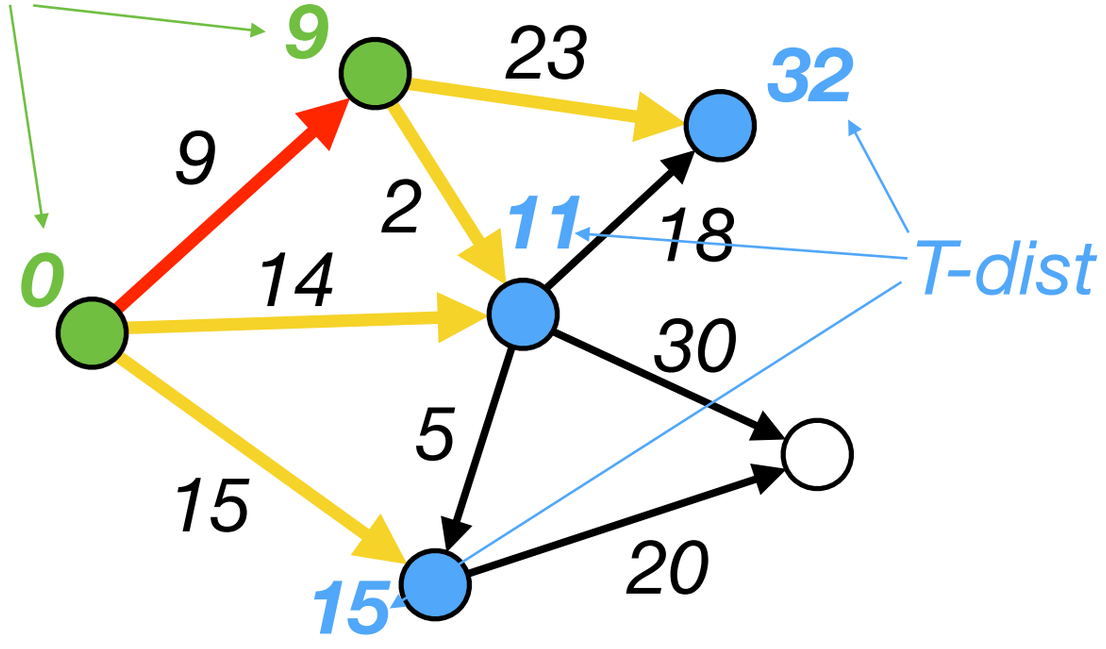
Unit Lengths



Breadth First Search

distance

Positive Lengths

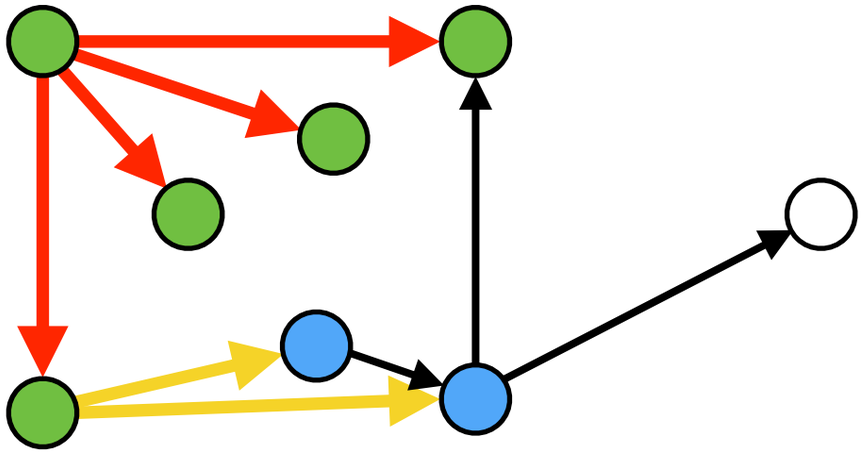


Dijkstra Algorithm

Recap: Single-Source Shortest Path

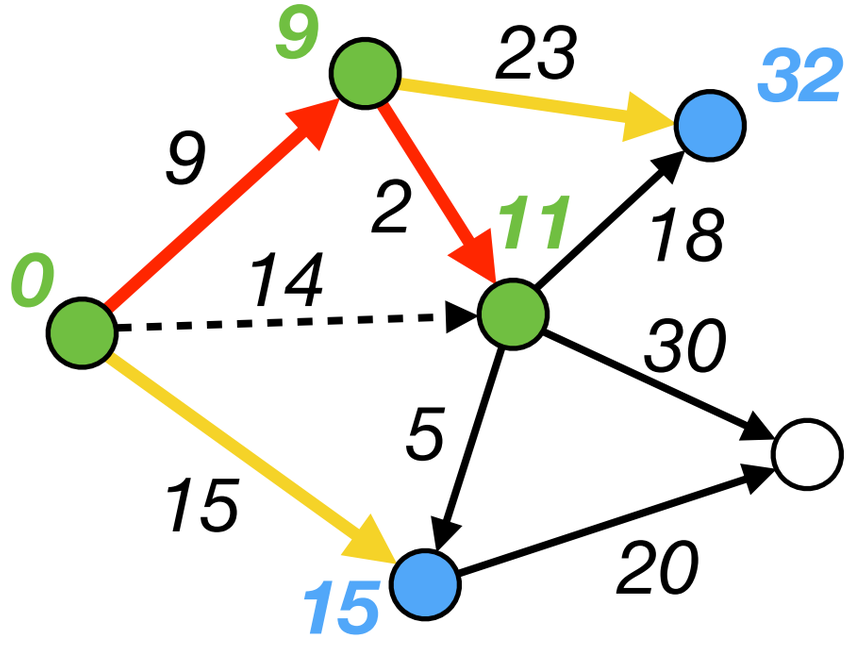
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

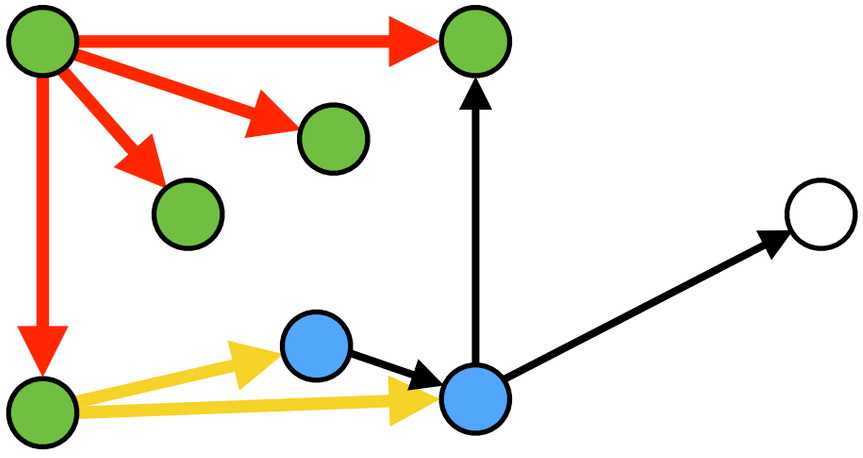


Dijkstra Algorithm

Recap: Single-Source Shortest Path

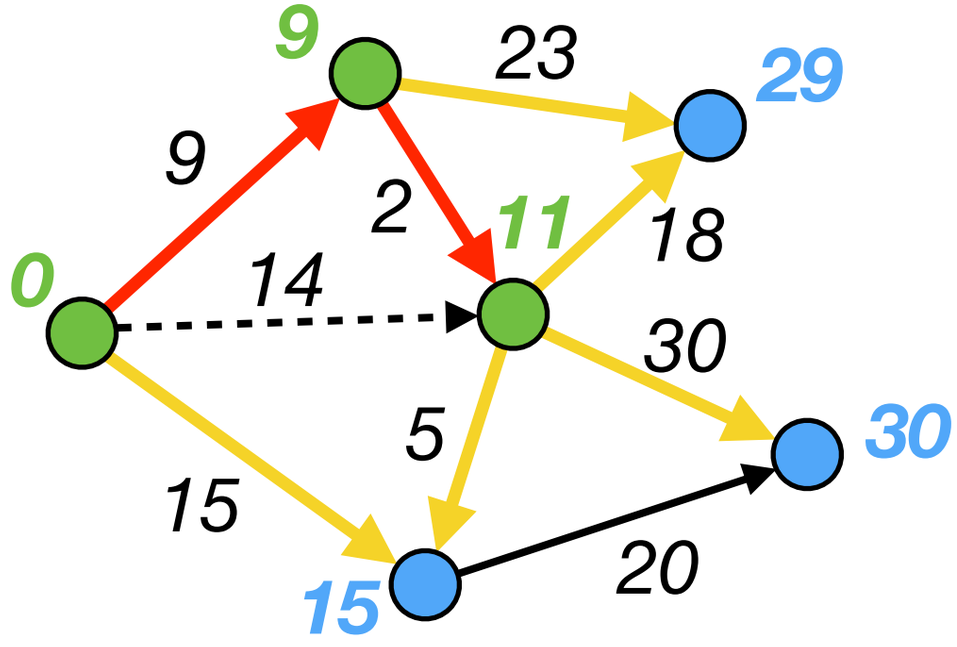
Given a directed graph with edge lengths and a source vertex *s*, find the length of the shortest path from *s* to *v* for every vertex *v*.

Unit Lengths



Breadth First Search

Positive Lengths

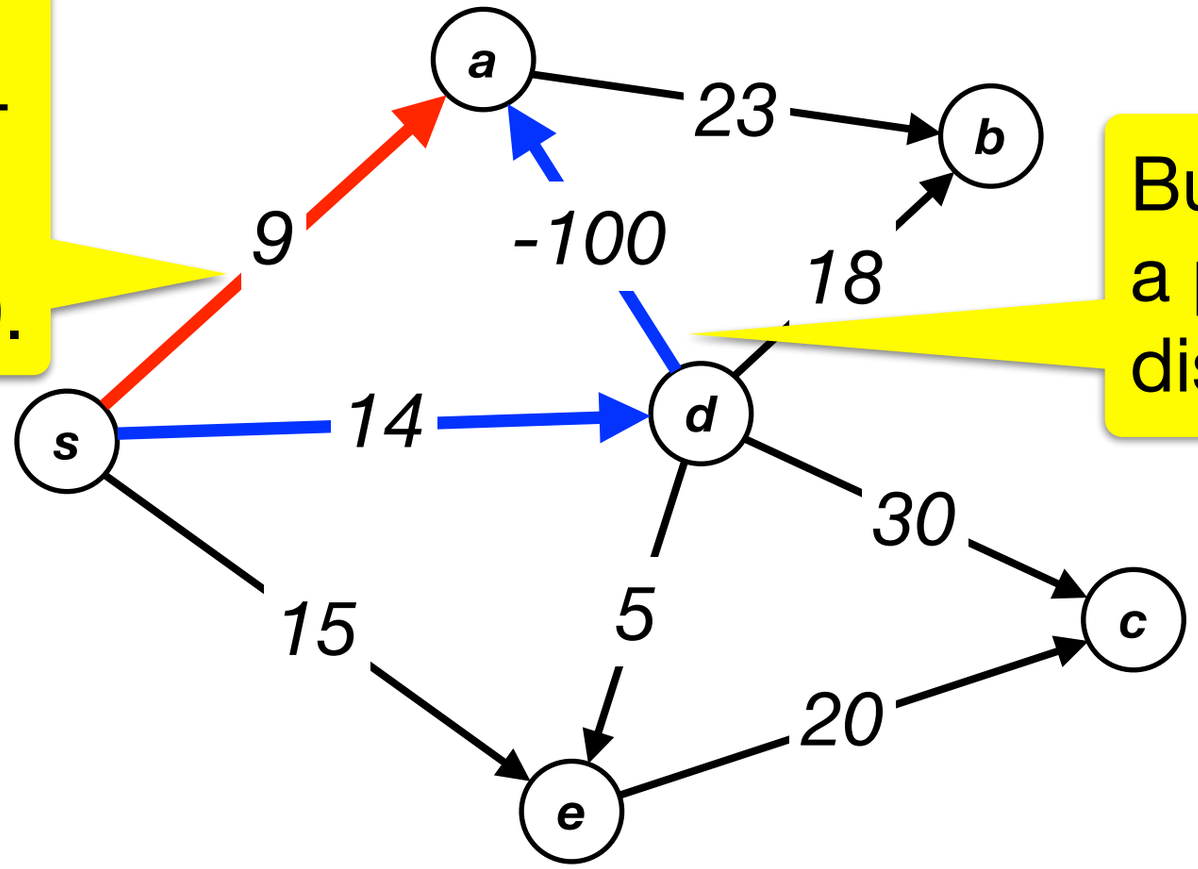


Dijkstra Algorithm

Today: Negative Edge Lengths

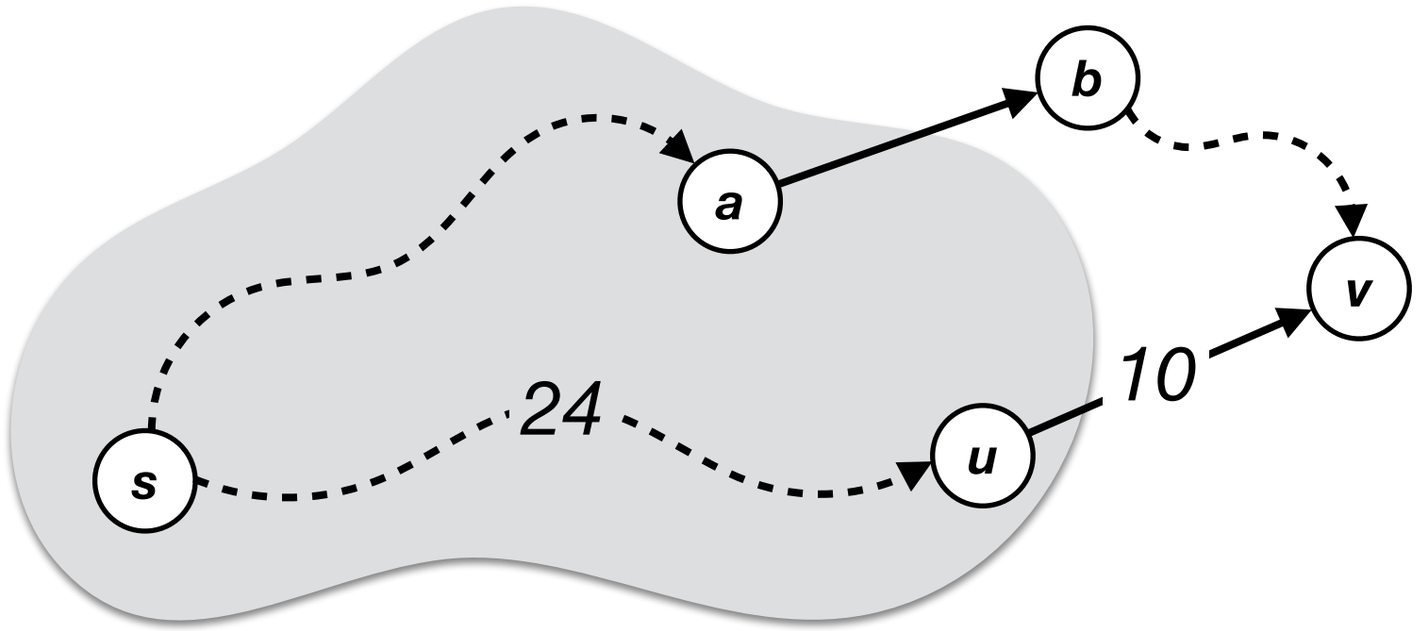
- Negative lengthes are natural in some applications.
- The correctness of Dijkstra depends on the fact that no edge has negative weight.

Dijkstra will add this edge to SPT at step 1, and assert dist(*a*) = 9.

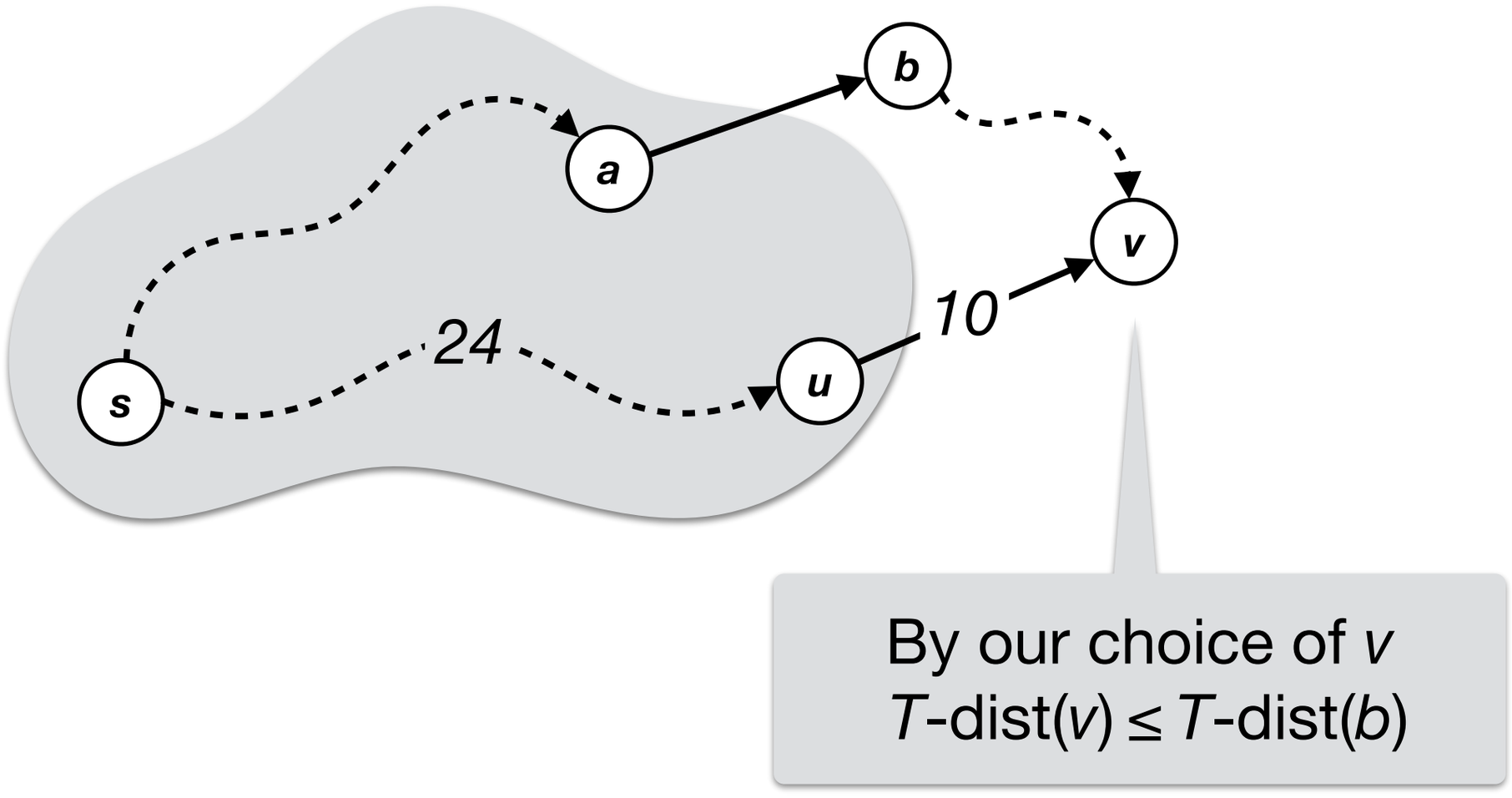


But there is a path with distance -86

What goes wrong in the proof of Dijkstra's correctness with negative edges?

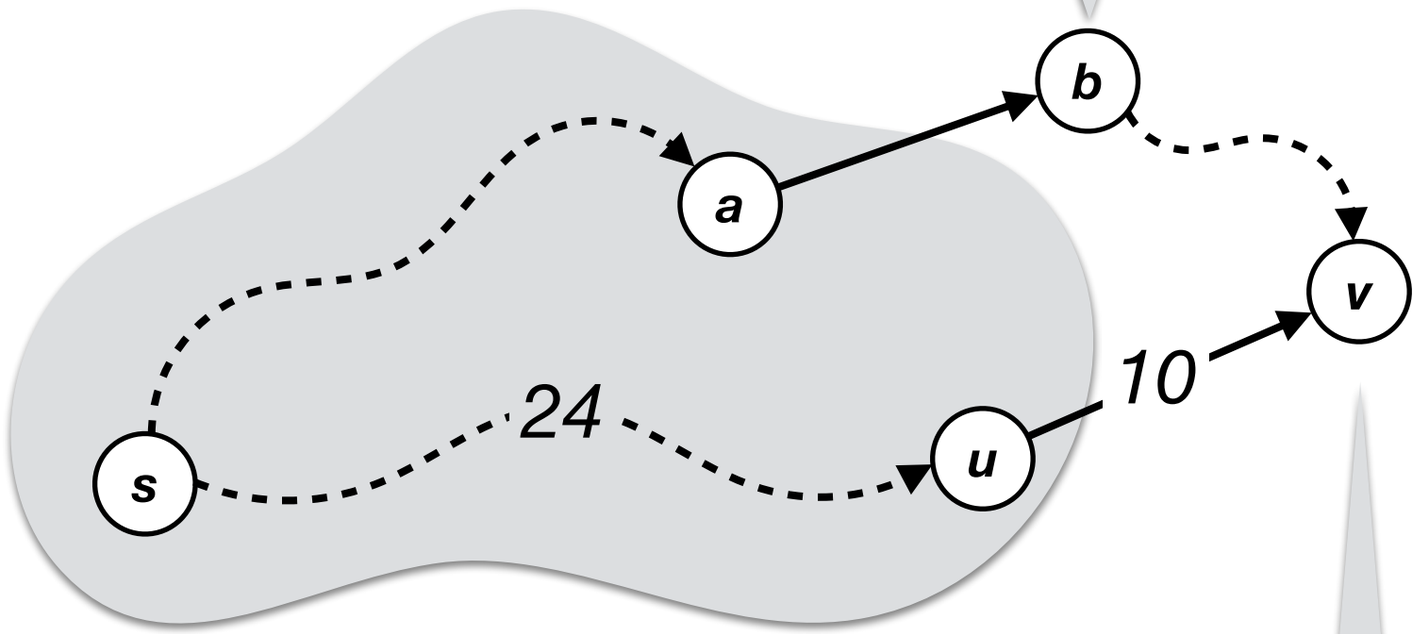


What goes wrong in the proof of Dijkstra's correctness with negative edges?



What goes wrong in the proof of Dijkstra's correctness with negative edges?

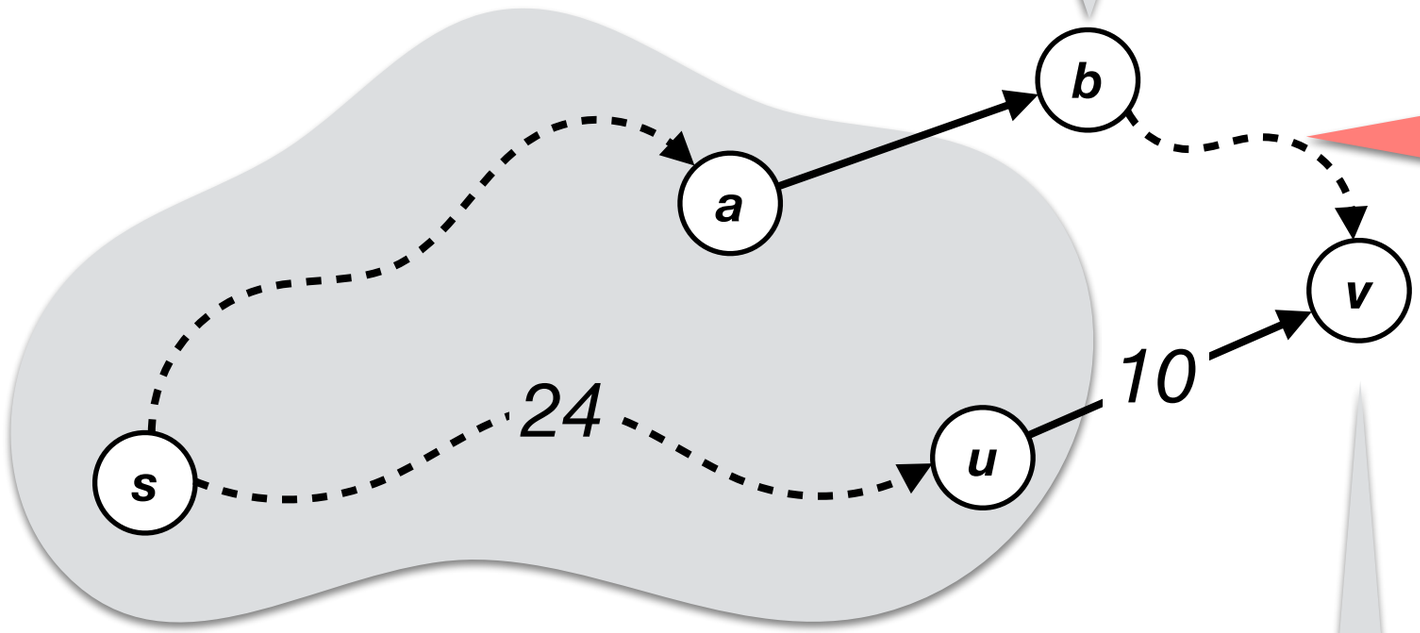
Since going through u is the shortest T-path from s to v, any alternative path from s to v must go through some vertex $b \notin T$, and T-dist(b) ≥ T-dist(v).



By our choice of v
T-dist(v) ≤ T-dist(b)

What goes wrong in the proof of Dijkstra's correctness with negative edges?

Since going through u is the shortest T-path from s to v, any alternative path from s to v must go through some vertex $b \notin T$, and T-dist(b) ≥ T-dist(v).



The length of this part from b to v is positive. (all edges have positive length.)

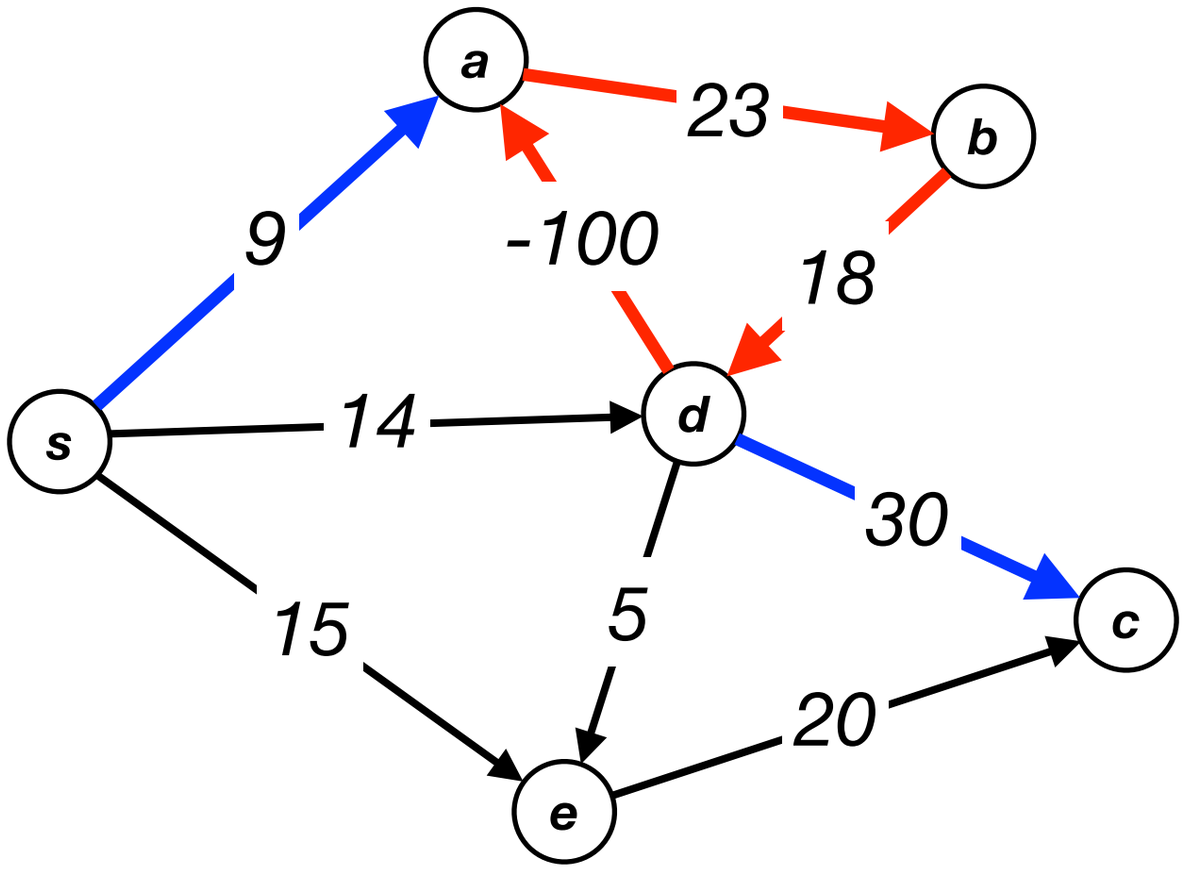
The red part is no longer true.

By our choice of v
T-dist(v) ≤ T-dist(b)

Negative cycles

Note: "shortest path" is not well defined if the input graph has negative cycles.

- From *s* to *c*, we can traverse the red cycle as many times as we want to get a path with distance as small as we want.



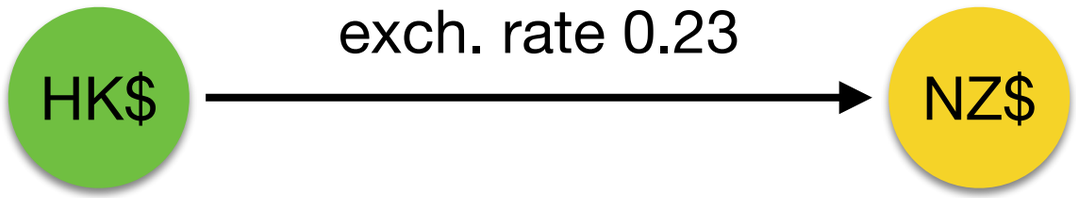
This lecture: Bellman-Ford algorithm

For graphs that have negative edges, **but do not have negative cycle**, we can use the Bellman-Ford algorithm to solve the single source shortest path problem.

Negative Lengths??

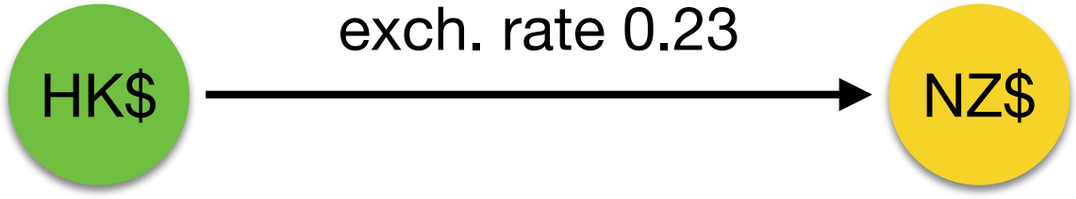
Negative Lengths??

- Suppose Zhiyi will go to New Zealand in reading week and needs some New Zealand dollars



Negative Lengths??

- Suppose Zhiyi will go to New Zealand in reading week and needs some New Zealand dollars

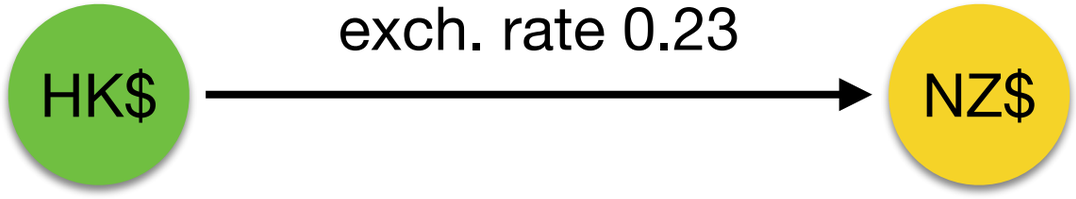


- Or:

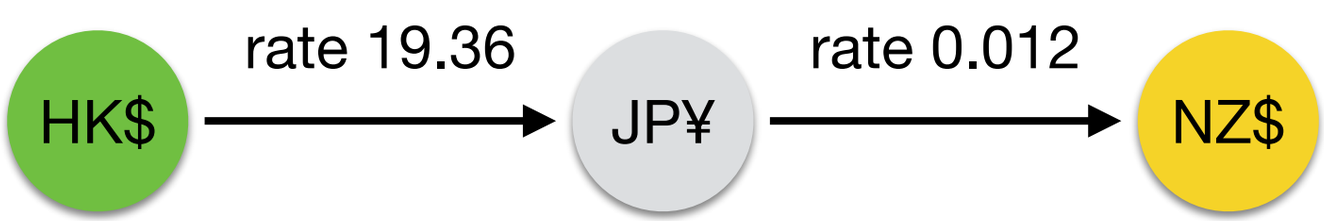
with $19.36 \times 0.012 = 0.23232$

Negative Lengths??

- Suppose Zhiyi will go to New Zealand in reading week and needs some New Zealand dollars

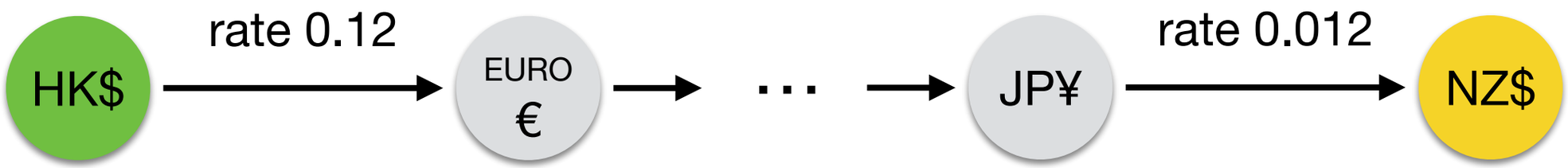


- Or:



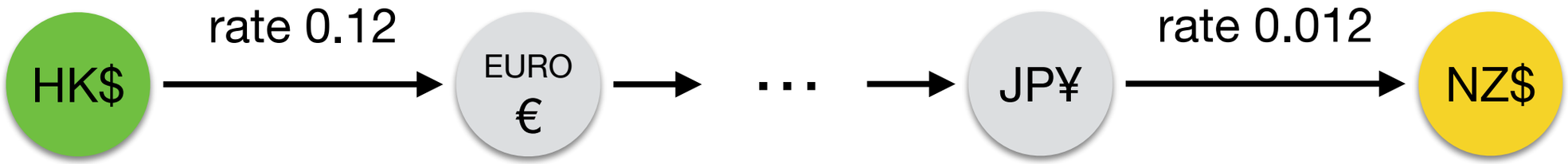
with $19.36 \times 0.012 = 0.23232$

- Or:



with $0.12 \times \ldots \times 0.012 = 0.236$

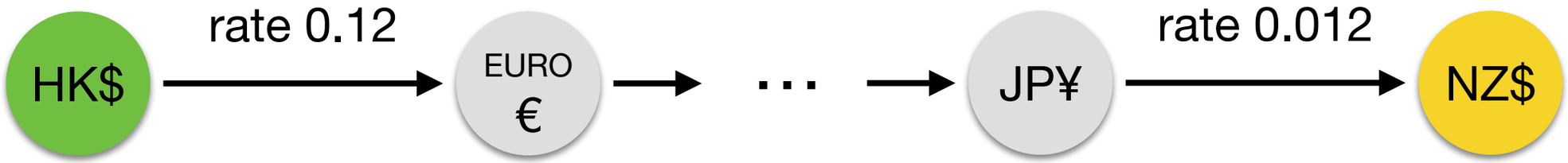
“Reduction” to Shortest Path



with $0.12 \times \ldots \times 0.012 = 0.236$

- This is a path with (1) **currencies** as **vertices**, (2) **exchanges** as **edges**, and (3) **exchange rates** as **“edge weights”**

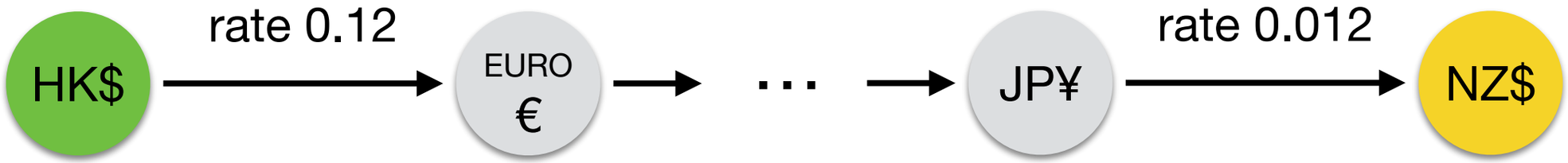
“Reduction” to Shortest Path



with $0.12 \times \ldots \times 0.012 = 0.236$

- This is a path with (1) **currencies** as **vertices**, (2) **exchanges** as **edges**, and (3) **exchange rates** as **“edge weights”**
- **Goal: maximizing** the **product** of rates along the path

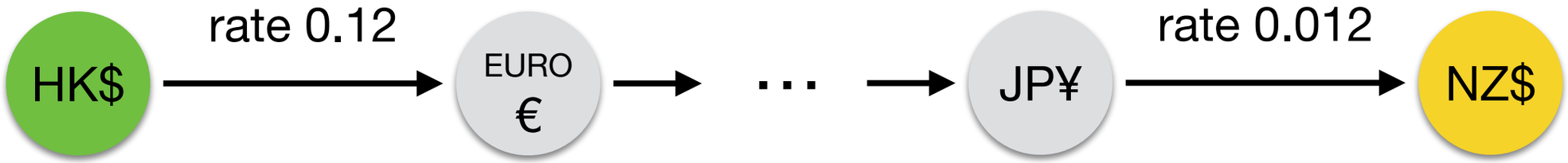
“Reduction” to Shortest Path



with $0.12 \times \ldots \times 0.012 = 0.236$

- This is a path with (1) **currencies** as **vertices**, (2) **exchanges** as **edges**, and (3) **exchange rates** as **“edge weights”**
- **Goal: maximizing** the **product** of rates along the path
- Compared to **minimizing** the **sum** of edge lengths

“Reduction” to Shortest Path

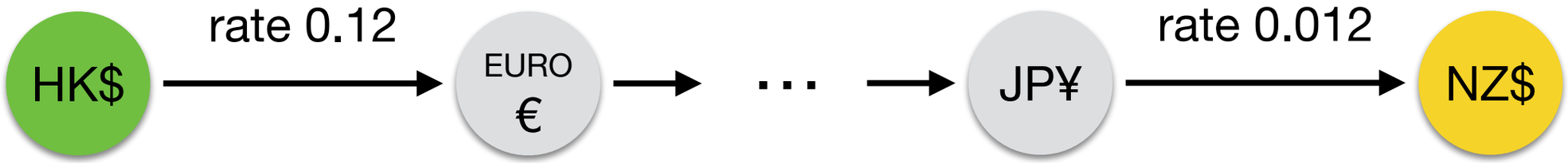


with $0.12 \times \ldots \times 0.012 = 0.236$

- This is a path with (1) **currencies** as **vertices**, (2) **exchanges** as **edges**, and (3) **exchange rates** as **“edge weights”**
- **Goal: maximizing** the **product** of rates along the path
- Compared to **minimizing** the **sum** of edge lengths
- **Product vs. Sum**

$$\log(abc) = \log(a) + \log(b) + \log(c)$$

“Reduction” to Shortest Path



with $0.12 \times \ldots \times 0.012 = 0.236$

- This is a path with (1) **currencies** as **vertices**, (2) **exchanges** as **edges**, and (3) **exchange rates** as **“edge weights”**
- **Goal: maximizing** the **product** of rates along the path
- Compared to **minimizing** the **sum** of edge lengths
- **Product vs. Sum**

$$\log(abc) = \log(a) + \log(b) + \log(c)$$

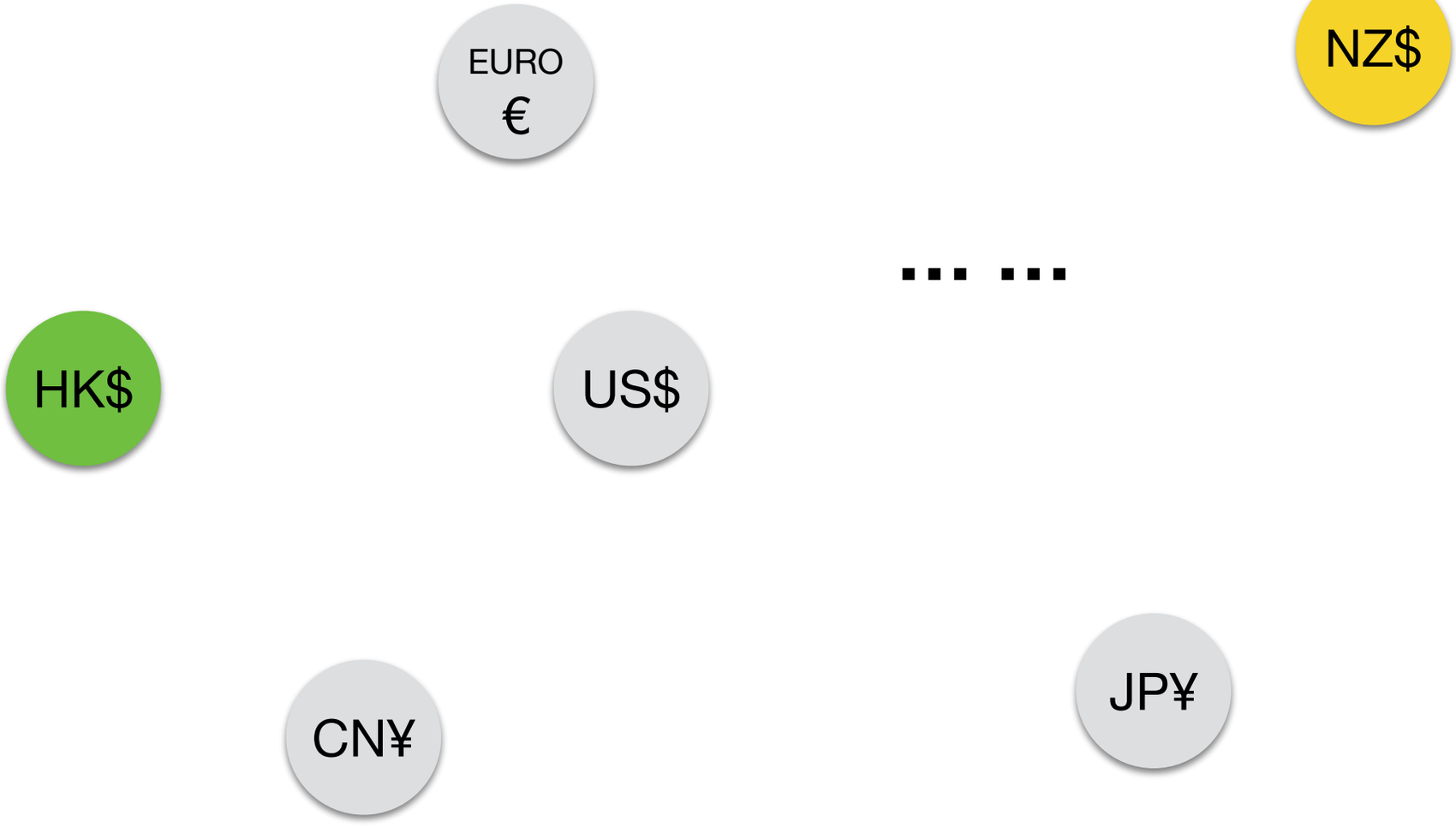
- **Maximization vs. Minimization**

$$\max\ a + b + c \quad \Leftrightarrow \quad \min\ (-a) + (-b) + (-c)$$

Putting together

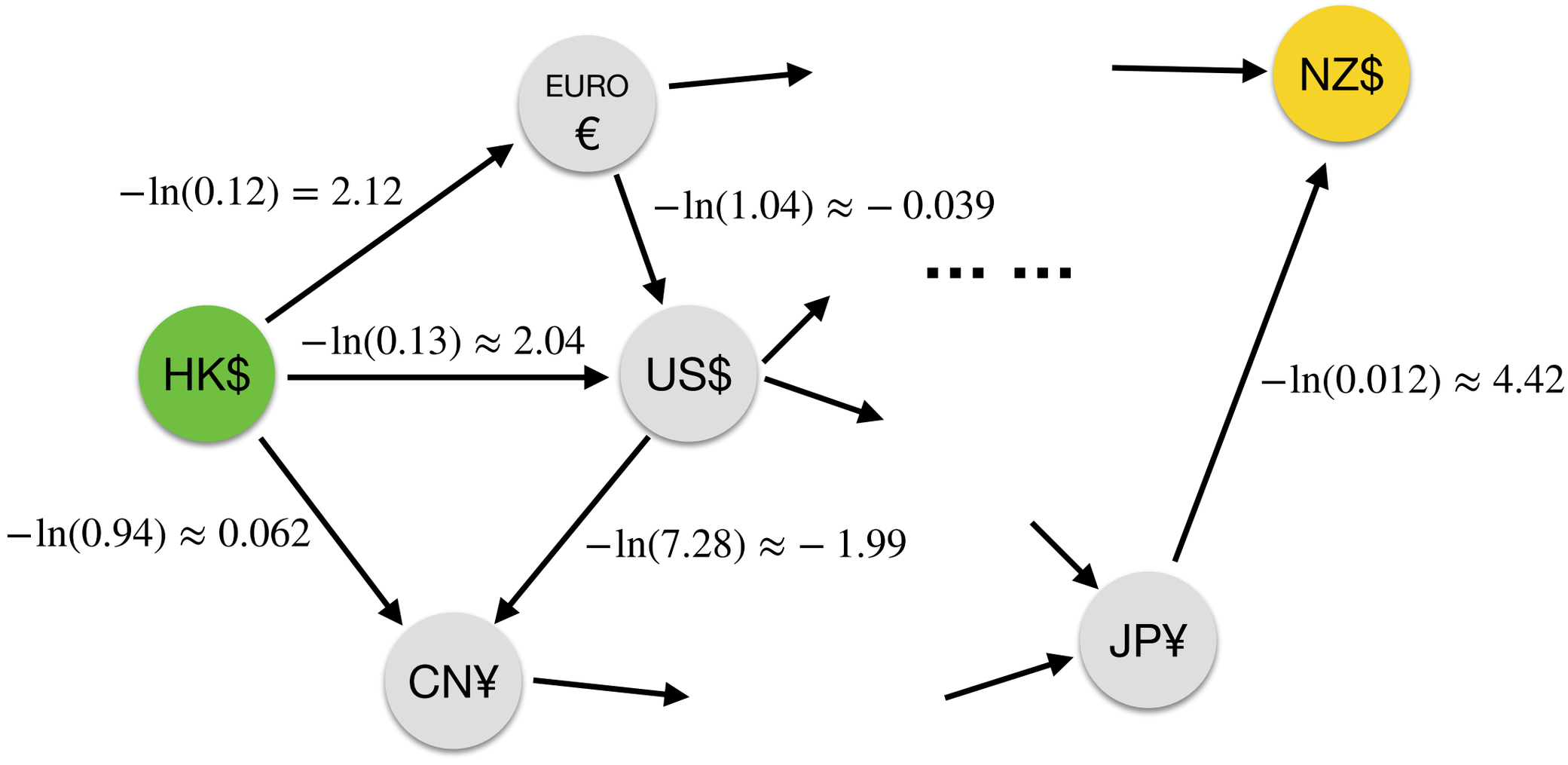
Putting together

- Let there be a vertex for each currency



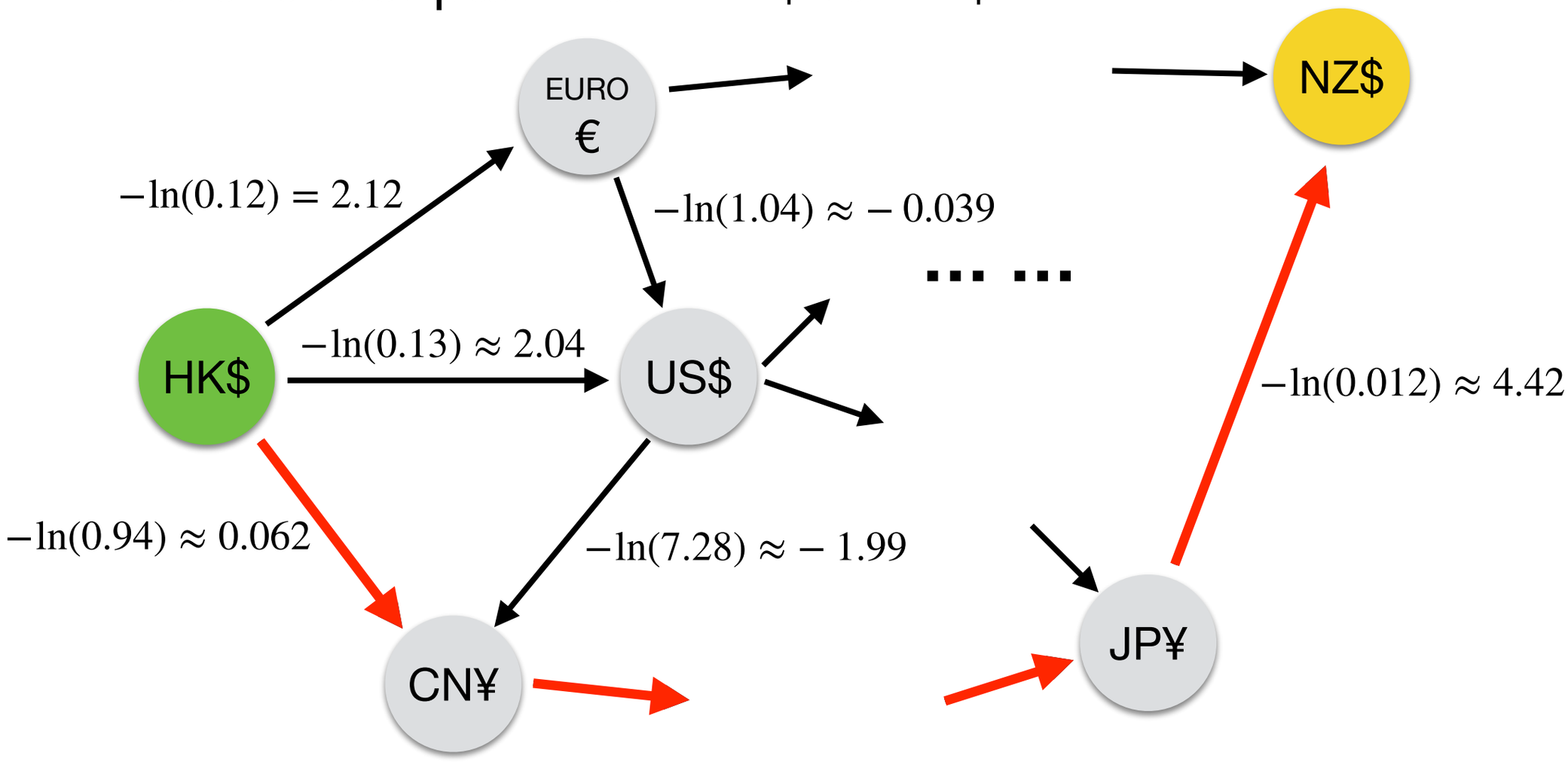
Putting together

- Let there be a vertex for each currency
- Let there be a directed edge between each pair of currencies with the **negative logarithm** of exchange rate as **"length"**



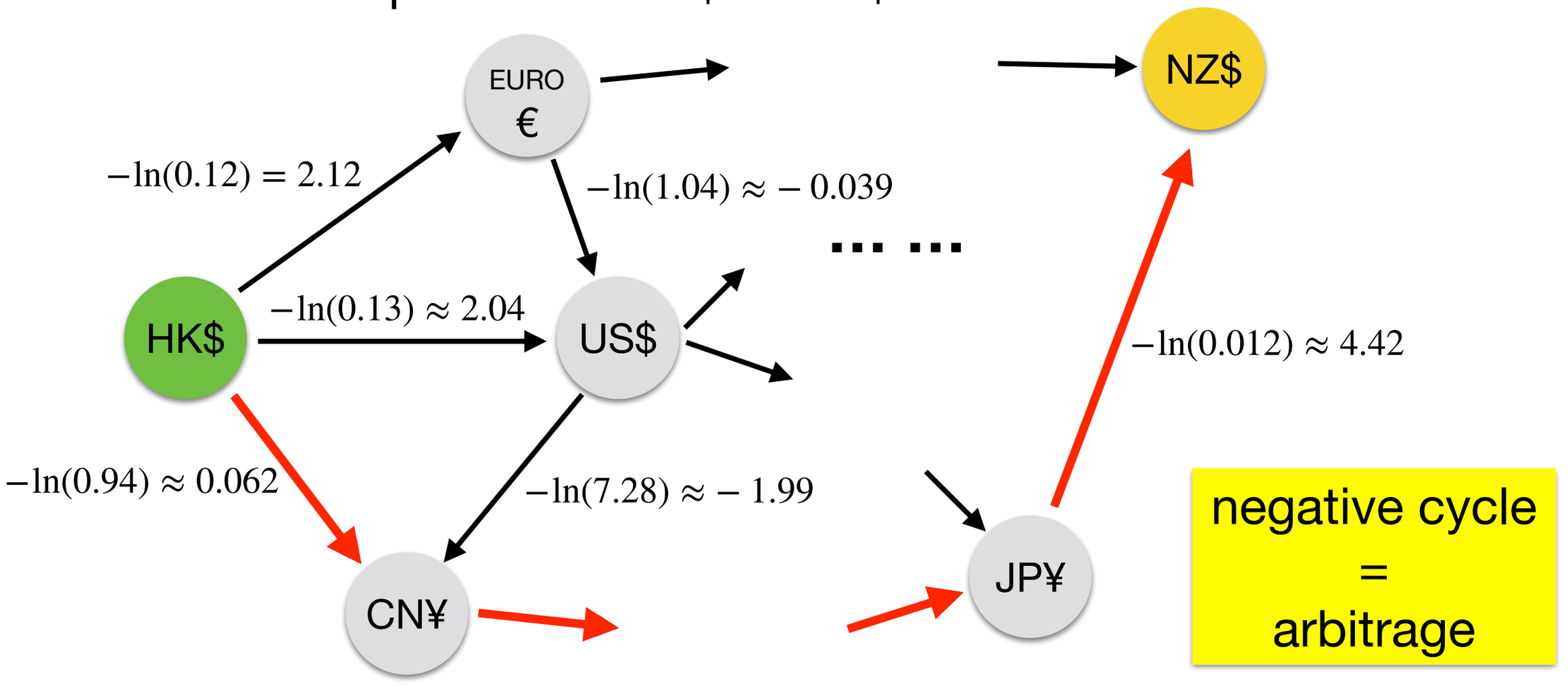
Putting together

- Let there be a vertex for each currency
- Let there be a directed edge between each pair of currencies with the **negative logarithm** of exchange rate as **"length"**
- Find shortest path from HK$ to NZ$



Putting together

- Let there be a vertex for each currency
- Let there be a directed edge between each pair of currencies with the **negative logarithm** of exchange rate as **"length"**
- Find shortest path from HK$ to NZ$



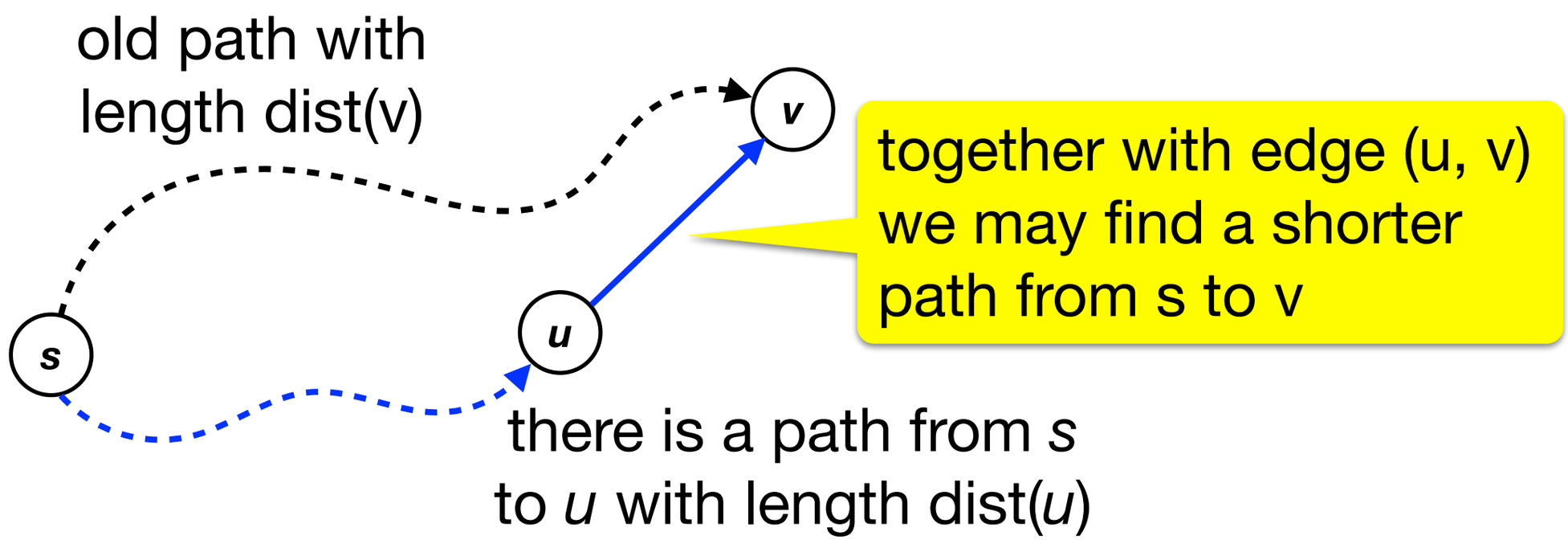
Another view on Dijkstra

Basic step: update *(u, v)* for an edge *(u, v)* $\in$ *E* as follows

$$\text{dist}(v) = \min \{ \text{dist}(v), \text{dist}(u) + L(u, v) \}$$

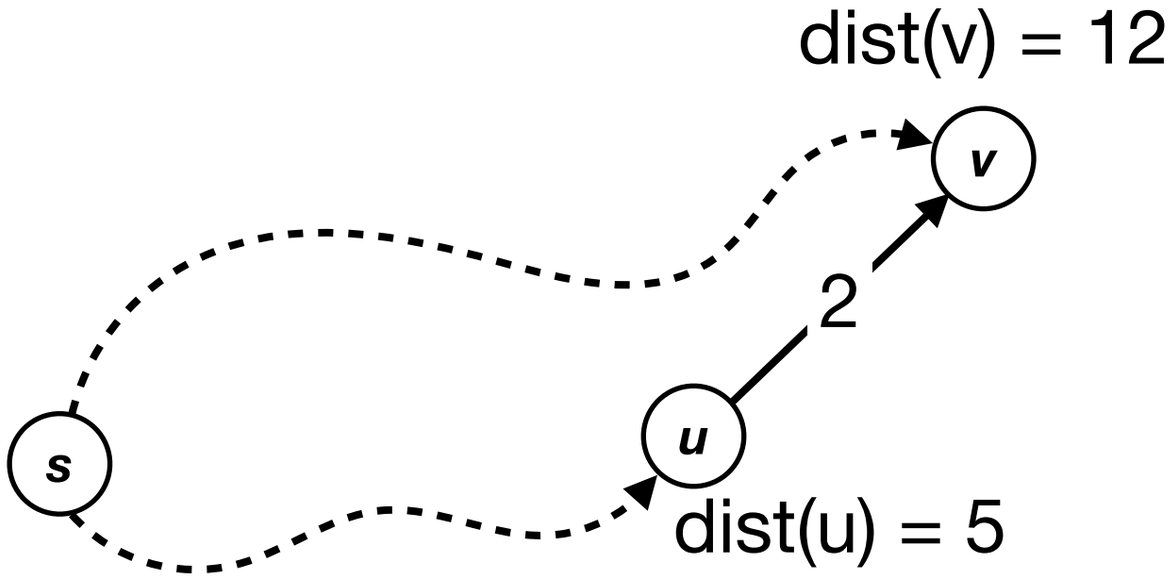
Intuition:

- dist(*v*) is the length of the shortest path we have found so far from *s* to *v*.
- Whenever we find a shorter path, we update dist(*v*).



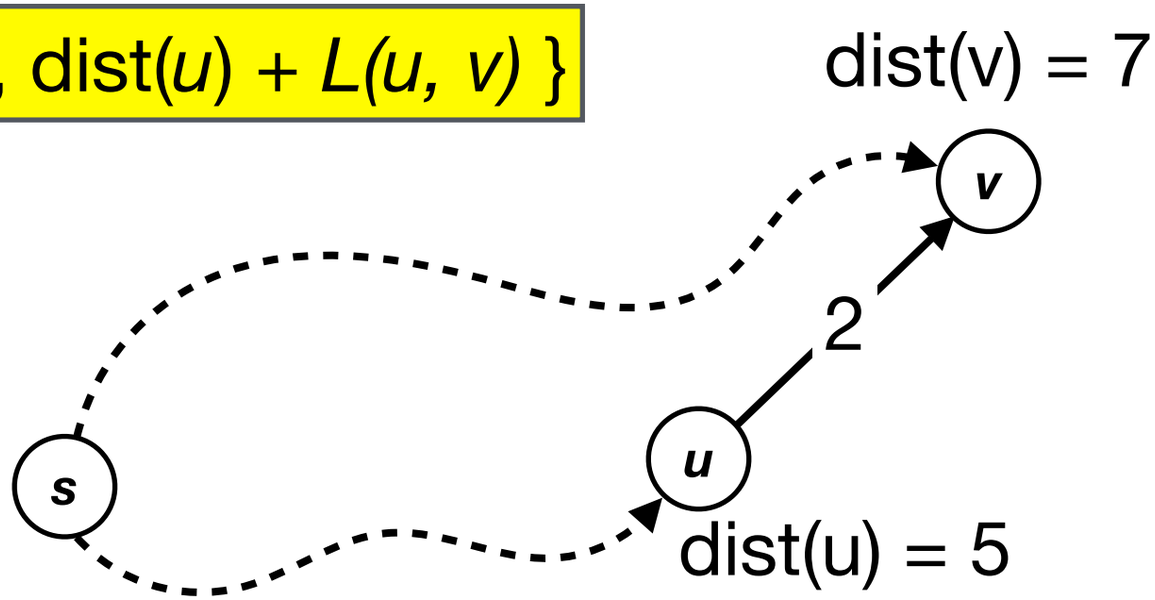
An Example

before update:



dist(*v*) = min { dist(*v*), dist(*u*) + *L(u, v)* }

after update:



Bellman-Ford algorithm

- **Without** negative weights, Dijkstra updates edges in a clever order so that the algorithm updates each edge only once.
- **With** negative weights, Dijkstra's clever order no longer works.
- Bellman-Ford returns to the "stupid" way and updates each edge multiple times:

```
1) initialize dist(s) = 0, and dist(x) = ∞ for other vertices x;
2) do
3)     for each edge (u, v) in E :
4)         dist(v) = min { dist(v), dist(u) + L(u, v) }
5) until there is no more update
```

Bellman-Ford algorithm

- **Without** negative weights, Dijkstra updates edges in a clever order so that the algorithm updates each edge only once.
- **With** negative weights, Dijkstra's clever order no longer works.
- Bellman-Ford returns to the "stupid" way and updates each edge multiple times:

```
1) initialize dist(s) = 0, and dist(x) = ∞ for other vertices x;
2) for i from 1 to |V| - 1 :
3)     for each edge (u, v) in E :
4)         dist(v) = min { dist(v), dist(u) + L(u, v) }
```

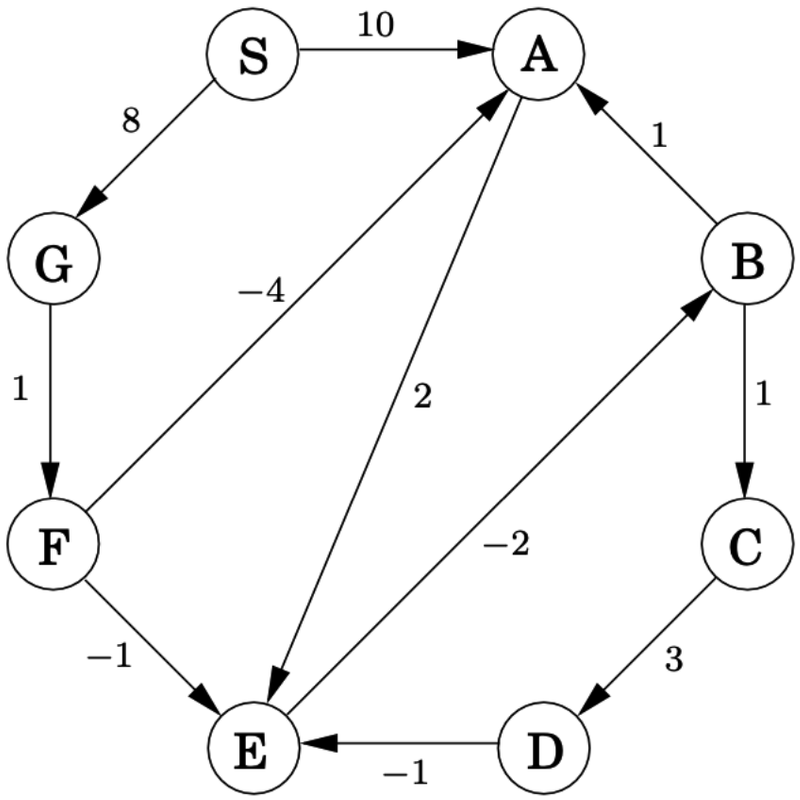
Bellman-Ford algorithm

- **Without** negative weights, Dijkstra updates edges in a clever order so that the algorithm updates each edge only once.
- **With** negative weights, Dijkstra's clever order no longer works.
- Bellman-Ford returns to the "stupid" way and updates each edge multiple times:

1) **initialize** dist(s) = *0*, and dist(x) = ∞ for other vertices x;
2) **for** i from *1* to $|V| - 1$ **:**
3) **for** each edge (u, v) in E **:**
4) dist(v) = min { dist(v), dist(u) + $L(u, v)$ }

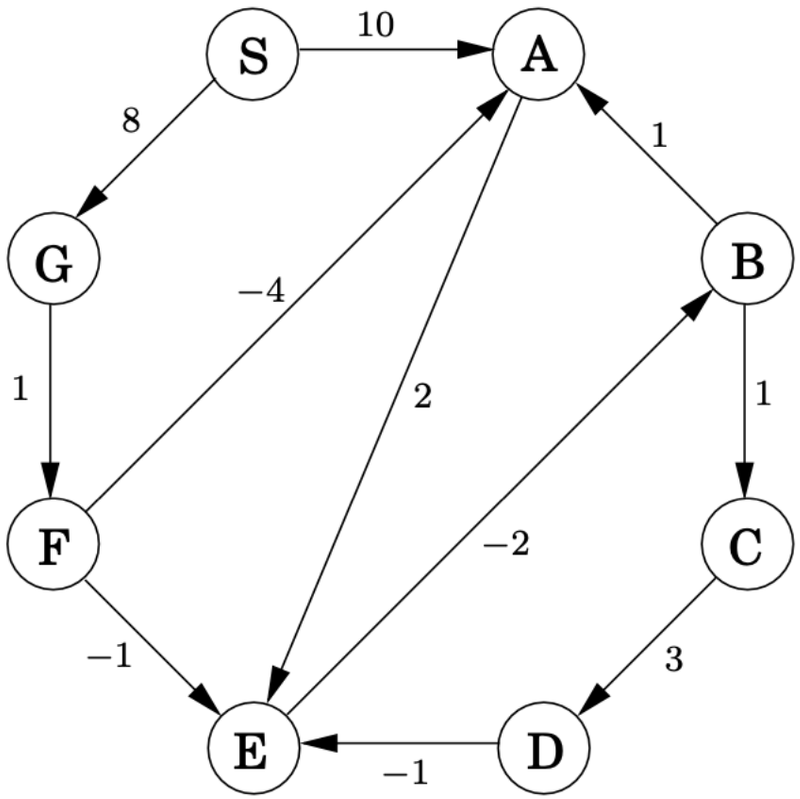
Running time: $O(|V||E|) = O(|V|^3)$

Sample Run



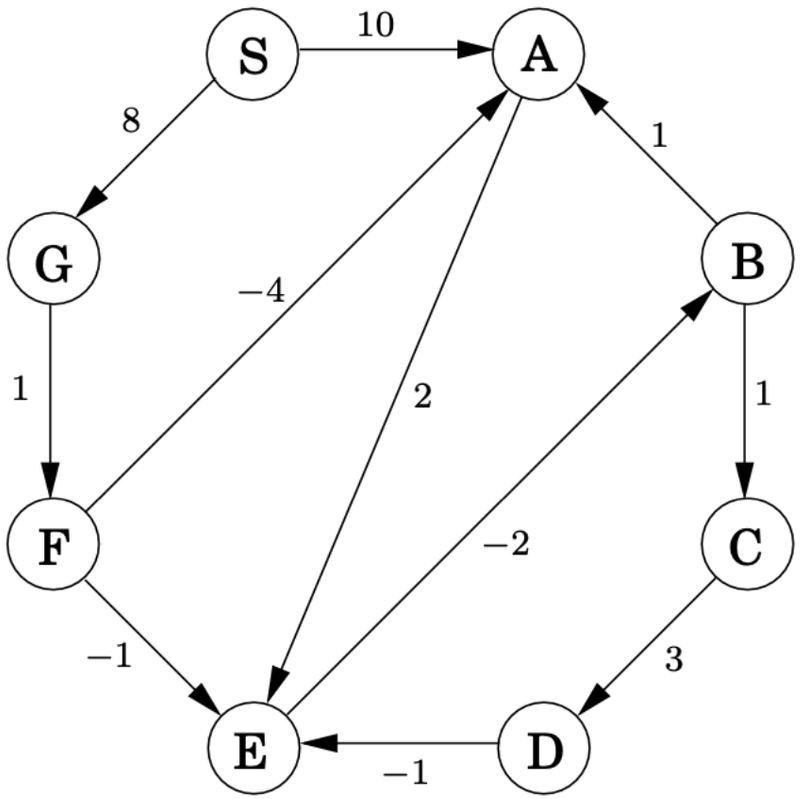
	0	1	2	3	4	5	6	7
S	0							
A	∞							
B	∞							
C	∞							
D	∞							
E	∞							
F	∞							
G	∞							

Sample Run



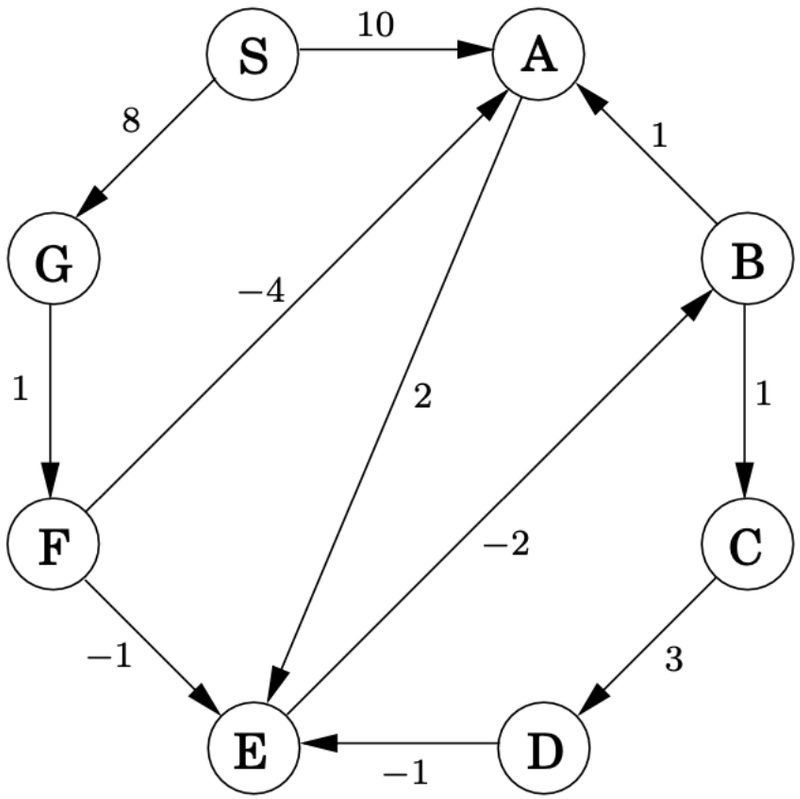
	0	1	2	3	4	5	6	7
S	0	0						
A	∞	**10**						
B	∞	∞						
C	∞	∞						
D	∞	∞						
E	∞	∞						
F	∞	∞						
G	∞	**8**						

Sample Run



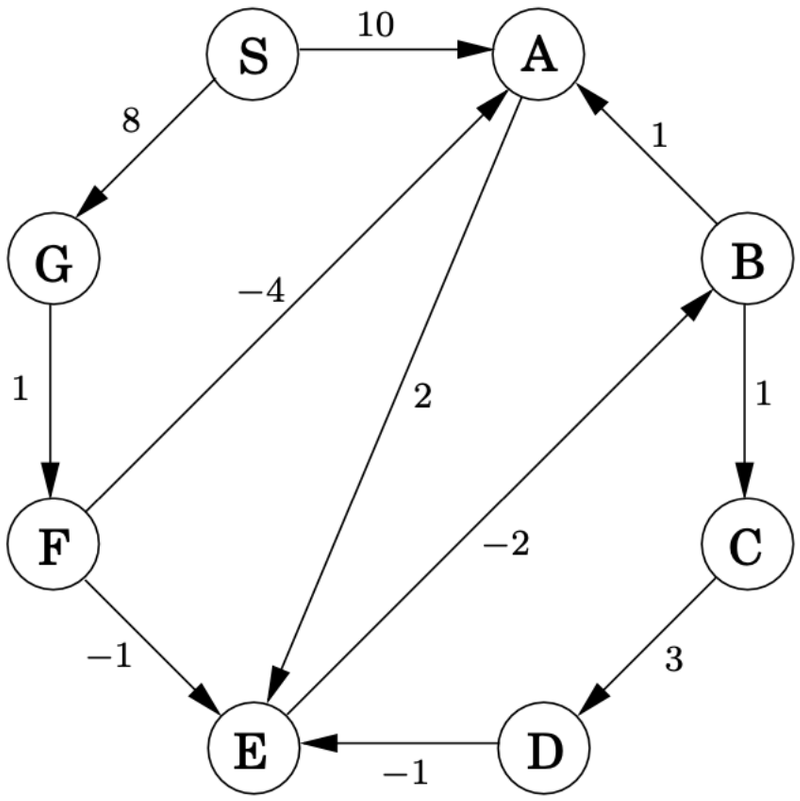
	0	1	2	3	4	5	6	7
S	0	0	0					
A	∞	10	10					
B	∞	∞	∞					
C	∞	∞	∞					
D	∞	∞	∞					
E	∞	∞	**12**					
F	∞	∞	**9**					
G	∞	8	8					

Sample Run



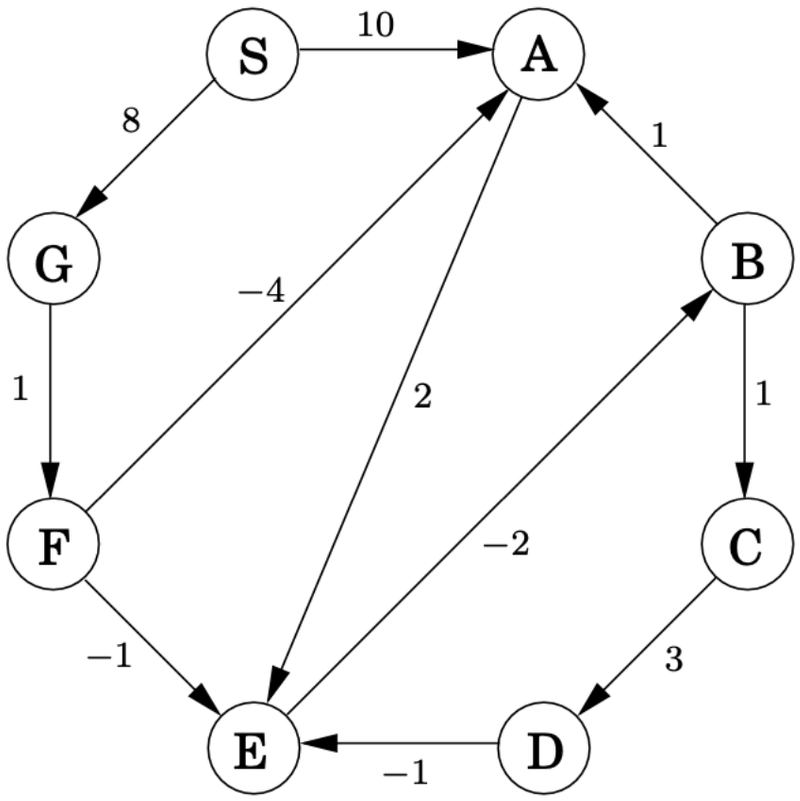
	0	1	2	3	4	5	6	7
S	0	0	0	0				
A	∞	10	10	**5**				
B	∞	∞	∞	**10**				
C	∞	∞	∞	∞				
D	∞	∞	∞	∞				
E	∞	∞	12	**8**				
F	∞	∞	9	9				
G	∞	8	8	8				

Sample Run



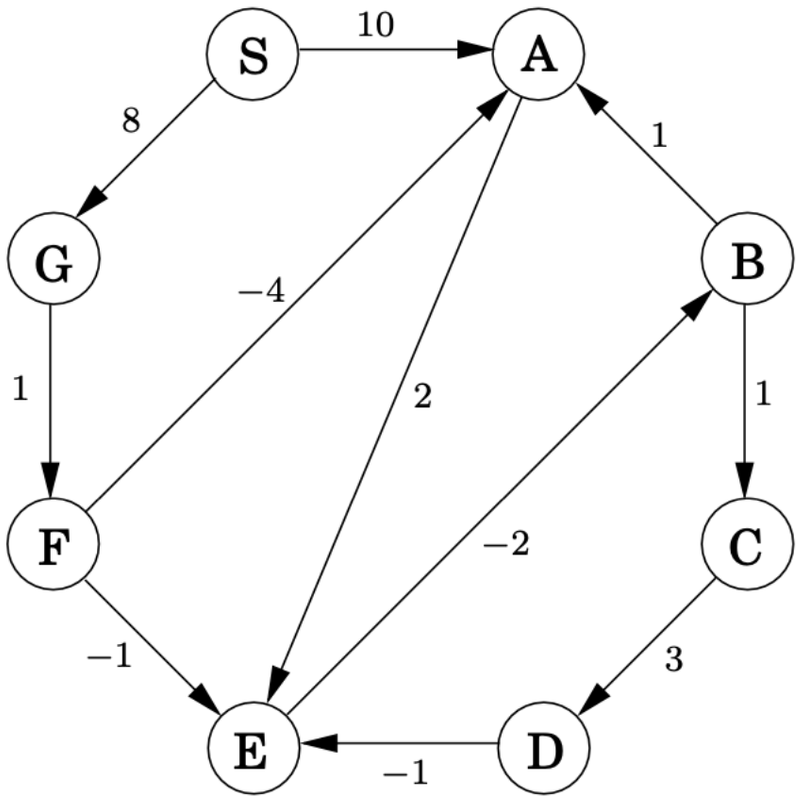
	0	1	2	3	4	5	6	7
S	0	0	0	0	0			
A	∞	10	10	5	5			
B	∞	∞	∞	10	**6**			
C	∞	∞	∞	∞	**11**			
D	∞	∞	∞	∞	∞			
E	∞	∞	12	8	**7**			
F	∞	∞	9	9	9			
G	∞	8	8	8	8			

Sample Run



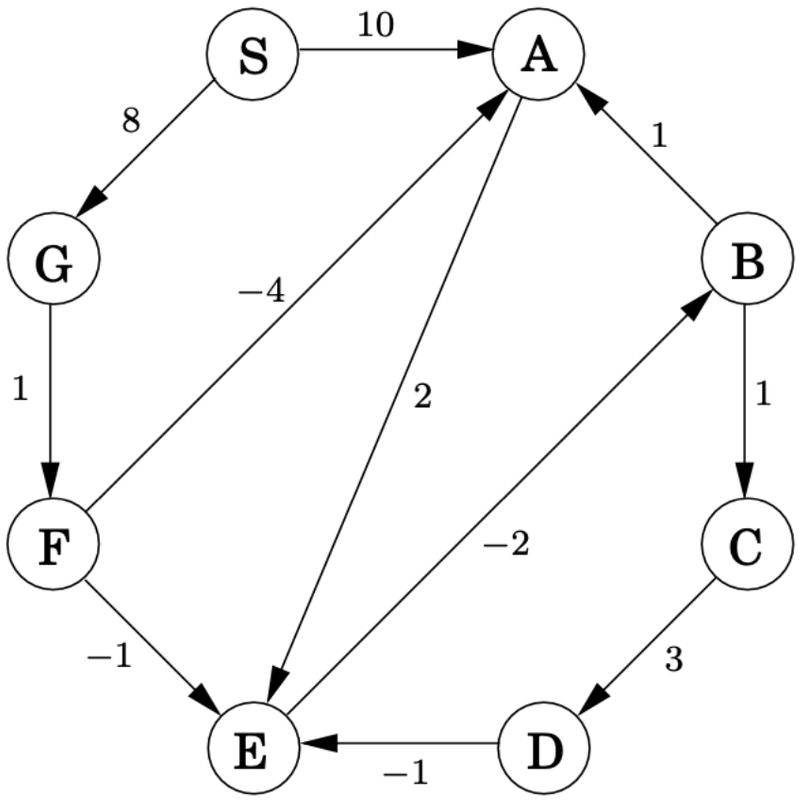
	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0		
A	∞	10	10	5	5	5		
B	∞	∞	∞	10	6	**5**		
C	∞	∞	∞	∞	11	**7**		
D	∞	∞	∞	∞	∞	**14**		
E	∞	∞	12	8	7	7		
F	∞	∞	9	9	9	9		
G	∞	8	8	8	8	8		

Sample Run



	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	
A	∞	10	10	5	5	5	5	
B	∞	∞	∞	10	6	5	5	
C	∞	∞	∞	∞	11	7	**6**	
D	∞	∞	∞	∞	∞	14	**10**	
E	∞	∞	12	8	7	7	7	
F	∞	∞	9	9	9	9	9	
G	∞	8	8	8	8	8	8	

Sample Run



	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	**9**
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Correctness of Bellman-Ford

Suppose that $G = (V, E)$ is a weighted directed graph without negative cycles.

Observation 1: For every vertex v, there is a shortest path from s to v with at most $|V| - 1$ edges.

Why?

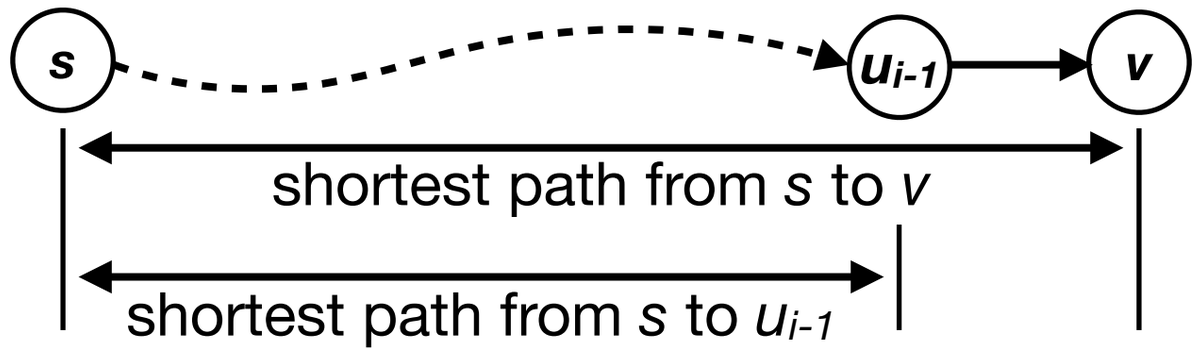
- We can assume without loss of generality that there is a shortest path that does not contain any cycle.
- All the vertices on this shortest path must be distinct; otherwise there is a cycle.
- Thus, there are at most $|V|$ vertices, and thus at most $|V| - 1$ edges on this path.

Correctness of Bellman-Ford (cont'd)

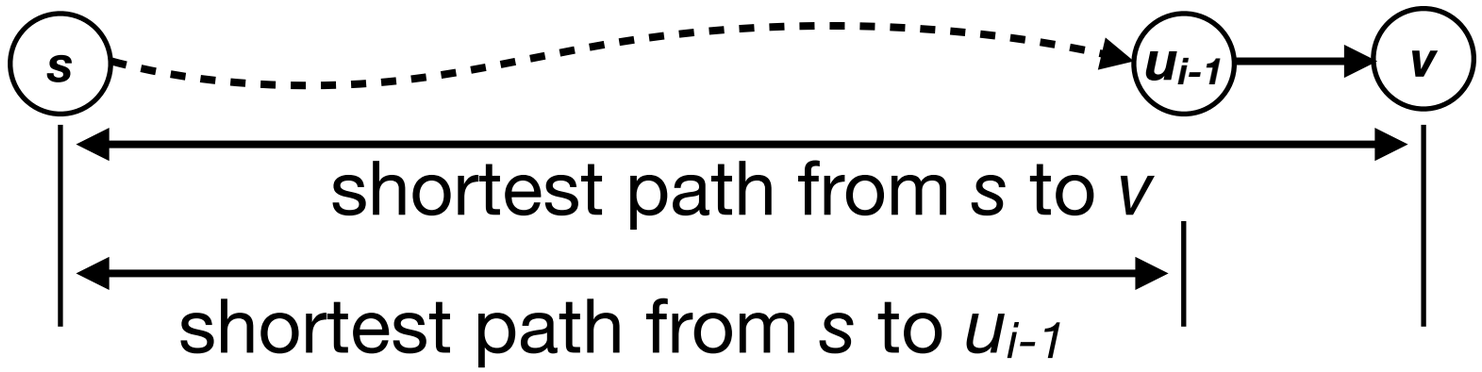
Fact: dist[*v*] is no larger than the length of the shortest path from *s* to *v* through out the algorithm.

Observation 2: For any vertex *v*, if there is a shortest path from *s* to *u* with *i* edges, then dist[*v*] = *d(s, v)* after the *i*-th iteration.

- Prove by mathematical induction.
- **Base case:** obviously true for *i = 0* because only *s* has a shortest path with no edge.
- **Inductive hypothesis:** suppose it is true for iteration *i - 1*.
- Consider a shortest path of length *i*: $P = (s, u_1, \ldots, u_{i-1}, v)$.



Correctness of Bellman-Ford (cont'd)



- Thus, there is a shortest path from s to u_{i-1} with $i - 1$ edges.
- By the inductive hypothesis, we have dist[u_{i-1}] = $d(s, u_{i-1})$ after iteration i-1.
- When the algorithm update edge (u_{i-1}, v) at iteration i,

$$\text{dist}(v) = \min \{ \text{dist}(v), \text{dist}(u_{i-1}) + L(u_{i-1}, v) \} = d(s, v)$$

- Note that dist[v] will remain the same hereafter because there are no shorter paths from s to v.

Correctness of Bellman-Ford (cont'd)

Observation 1: For every vertex *v*, there is a shortest path from *s* to *v* with at most $|V| - 1$ edges.

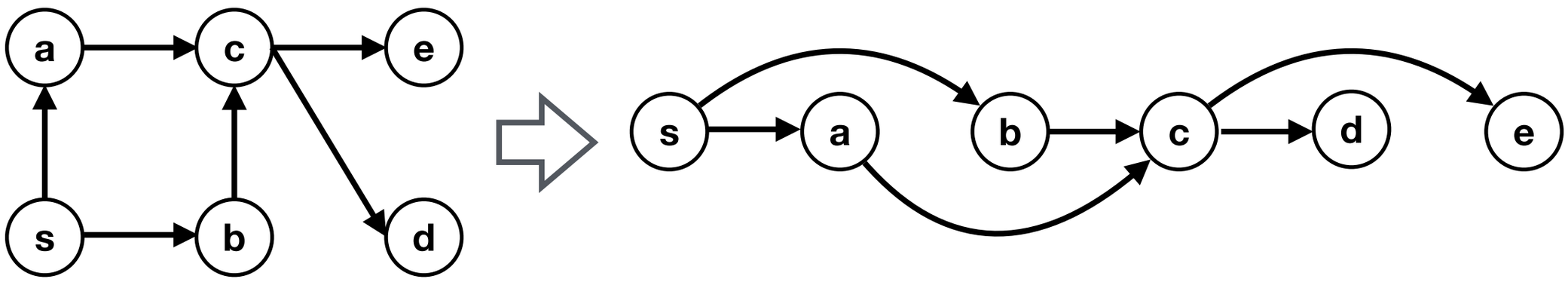
Observation 2: For any vertex *v*, if there is a shortest path from *s* to *v* with *i* edges, then dist[*v*] = *d(s, v)* after the *i*-th iteration.

- When Bellman-Ford terminates, it has iterated $|V| - 1$ times.
- By the above observations, dist[*v*] = *d(s, v)* when Bellman-Ford terminates.

Optional Topic:
Shortest Path in DAGs (Chapter 4.7)

Shortest Path in DAGs

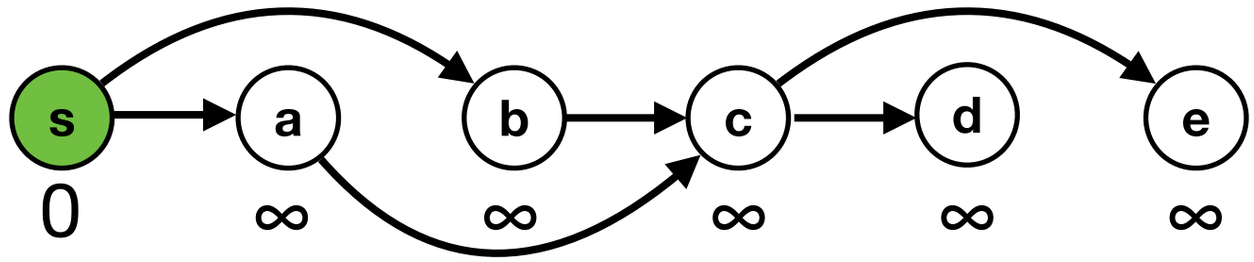
- Given a Directed Acyclic Graph (DAG) *G* with possibly negative-length edges, we can solve the single-source shortest path problem in $O(|V| + |E|)$ time.
- **Note:** Since *G* is a DAG, it does not have any cycle, and thus it does not have any negative cycle.
- **Recall:** We can linearize G in a line such that every edge is from left to right by running a DFS in $O(|V| + |E|)$ *time*.



Shortest Path in DAGs

1) **initialize** dist(s) = 0, and dist(*u*) = ∞ for all other $u \in V$
2) Linearize G;
3) **for** each $u \in V$ in linearized order **:**
4) **for** all edges *(u, v)*, update (u, v) as follows **:**

$$\text{dist}(v) = \min \{ \text{dist}(v), \text{dist}(u) + L(u, v) \}$$

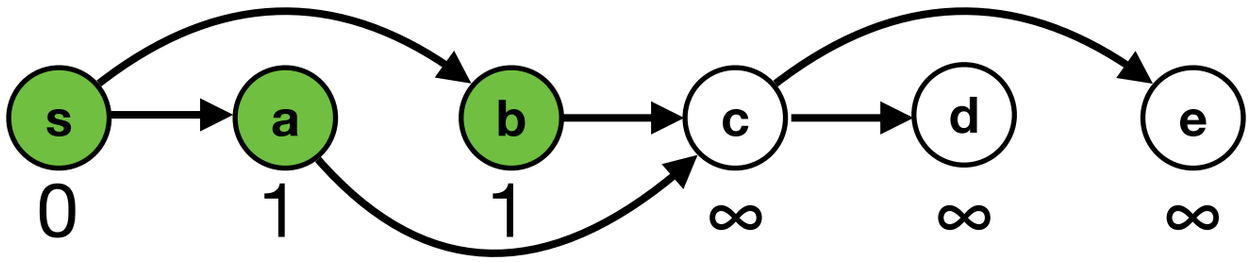


Initially, dist(s) is correct.

Shortest Path in DAGs

1) **initialize** dist(s) = 0, and dist(*u*) = ∞ for all other $u \in V$
2) Linearize G;
3) **for** each $u \in V$ in linearized order **:**
4) **for** all edges *(u, v)*, update (u, v) as follows **:**

$$\text{dist}(v) = \min \{ \text{dist}(v), \text{dist}(u) + L(u, v) \}$$

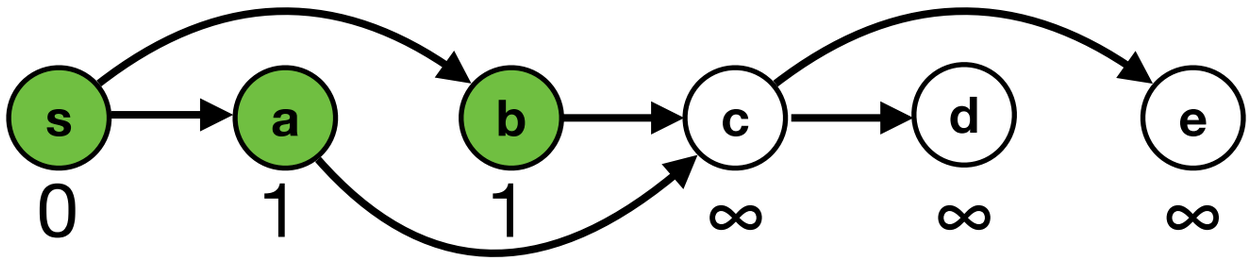


After the 1st iteration of the first for loop (step 3), the 1st vertex following *s* has correct dist(.) value.

Shortest Path in DAGs

1) **initialize** dist(s) = 0, and dist(*u*) = ∞ for all other $u \in V$
2) Linearize G;
3) **for** each $u \in V$ in linearized order **:**
4) **for** all edges *(u, v)*, update (u, v) as follows **:**

dist(*v*) = min { dist(*v*), dist(*u*) + *L(u, v)* }



After the *i*-th iteration of the first for loop (step 3), the *i*-th vertex following *s* has correct dist(.) value.