Graph Optimization

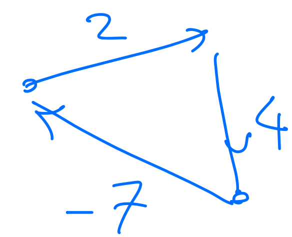
a. Dijkstra's Algorithm
b. Bellman-Ford

Why Care About Negative Edge Weights?

- Models various phenomena
  - Currency exchange (exchange rate can be + or -)
  - Chemical reactions (can be exo- or endothermic)
  - …

Bellman-Ford

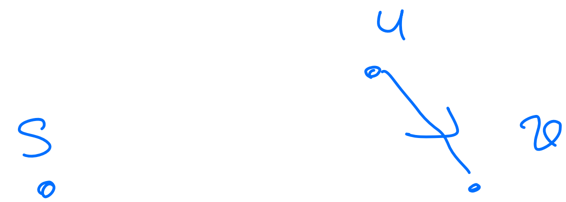
- **Input:** Directed, weighted graph $G = (V, E, \{w_e\})$, source node s
  - **Possibly negative edge lengths $w_e \in \mathbb{R}$**
  - **No negative-length cycles!**



- **Output:** Two arrays d, p
  - $d[u]$ is the length of the shortest $s \rightsquigarrow u$ path
  - $p[u]$ is the final hop on shortest $s \rightsquigarrow u$ path

Structure of Shortest Paths

- If $(u, v) \in E$, then $d(s, v) \leq d(s, u) + w(u, v)$ for every node $s \in V$

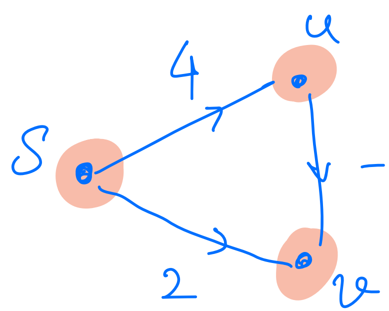


- If $(u, v) \in E$, and $d(s, v) = d(s, u) + w(u, v)$ then there is a shortest $s \rightsquigarrow v$-path ending with (u, v)

- For every *v != s*, there exists an edge $(u, v) \in E$ such that *d(s,v) = d(s,u) + w(u,v)*

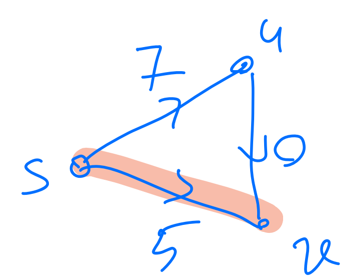
Ask the Audience

- Show that Dijkstra's Algorithm can fail in graphs with negative edge lengths (even without negative length cycles)



	s	u	v
d_0	0	∞	∞
d_1	0	4	2
d_2	0	4	2
d_3	0	4	2

- Why won't the following work?
  - Take a graph $G = (V, E, \{w(e)\})$ with negative lengths
  - Add $|\min w(e)|$ to all lengths to make them non-negative
  - Run Dijkstra's on the new graph



Ask the Audience

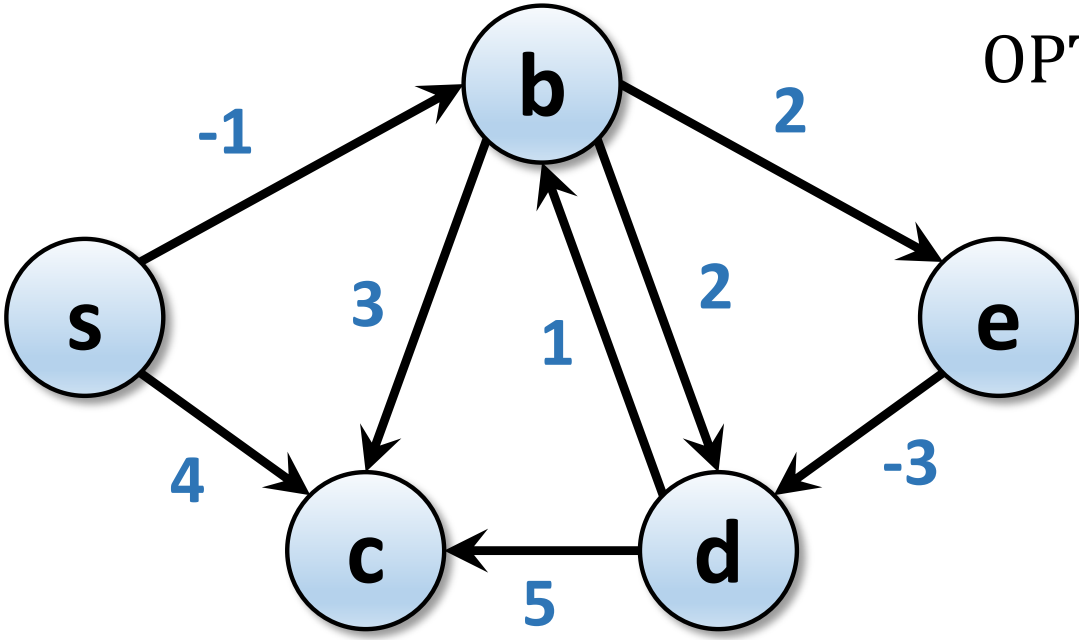
- Show that Dijkstra's Algorithm can fail in graphs with negative edge lengths (even without negative length cycles)

- Why won't the following work?
  - Take a graph $G = (V, E, \{w(e)\})$ with negative lengths
  - Add $|\min w(e)|$ to all lengths to make them non-negative
  - Run Dijkstra's on the new graph

Dynamic Programming

- **Subproblems:** Let OPT(*v*) be the length of the shortest path from *s* to *v*
- For every *v*, the shortest path makes some final hop *(u,v)*
- Case *u:* the final hop is *(u,v)*
  - *OPT(v) =* $OPT(u) + w_{u,v}$
- Recurrence: $\min_{u} OPT(u) + w_{u,v}$

Bottom-Up Implementation?



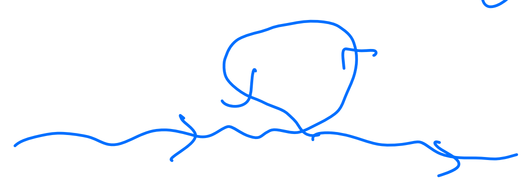
$$\mathrm{OPT}(v) = \min_{(u,v)\in E}\{\mathrm{OPT}(u) + w_{u,v}\}$$

v	s	b	c	d	e
OPT(v)	0				

Dynamic Programming Take II

- **Subproblems:** Let $\mathrm{OPT}(v, j)$ be the length of the shortest path from s to v with at most j hops $(0 \leq j \leq n-1)$

Why $j \leq n-1$? Suppose that there are $\geq n$ edges in the shortest path from s to v. This means there are at least $n+1$ vertices in the shortest path from s to v. By pigeonhole principle we visit a vertex twice at least $\rightarrow$ a cycle. Because no negative cycles, we can shortcut the cycle.



Recurrence

- **Subproblems:** $\mathrm{OPT}(v, j)$ is the length of the shortest $s \rightsquigarrow v$ path with at most j hops
- **Case u:** (u, v) is final edge on the shortest $s \rightsquigarrow v$ path with at most j hops $\leftarrow$ $OPT(v,j) = OPT(u,j-1) + w_{u,v}$
- **Recurrence:**

$$OPT(v,j) = \min\left\{ OPT(v,j-1),\ \min_{u \to v} \left(OPT(u,j-1) + w_{u,v} \right) \right\}$$

Recurrence

- **Subproblems:** $\mathrm{OPT}(v, j)$ is the length of the shortest $s \rightsquigarrow v$ path with at most j hops
- **Case u:** (u, v) is final edge on the shortest $s \rightsquigarrow v$ path with at most j hops

Recurrence:

$$\mathrm{OPT}(v, j) = \min \left\{ \mathrm{OPT}(v, j-1), \min_{(u,v)\in E} \{\mathrm{OPT}(u, j-1) + w_{u,v}\} \right\}$$

$$\mathrm{OPT}(s, 0) = 0$$

$\mathrm{OPT}(v, 0) = \infty$ for every $v \neq s$

Finding the paths

- $\mathrm{OPT}(v, j)$ is the length of the shortest $s \rightsquigarrow v$ path with at most j hops
- $P(v, j)$ is the last hop on some shortest $s \rightsquigarrow v$ path with at most j hops

Recurrence:

$$\mathrm{OPT}(v, j) = \min\left\{\mathrm{OPT}(v, j-1), \min_{(u,v) \in E}\{\mathrm{OPT}(u, j-1) + w_{u,v}\}\right\}$$

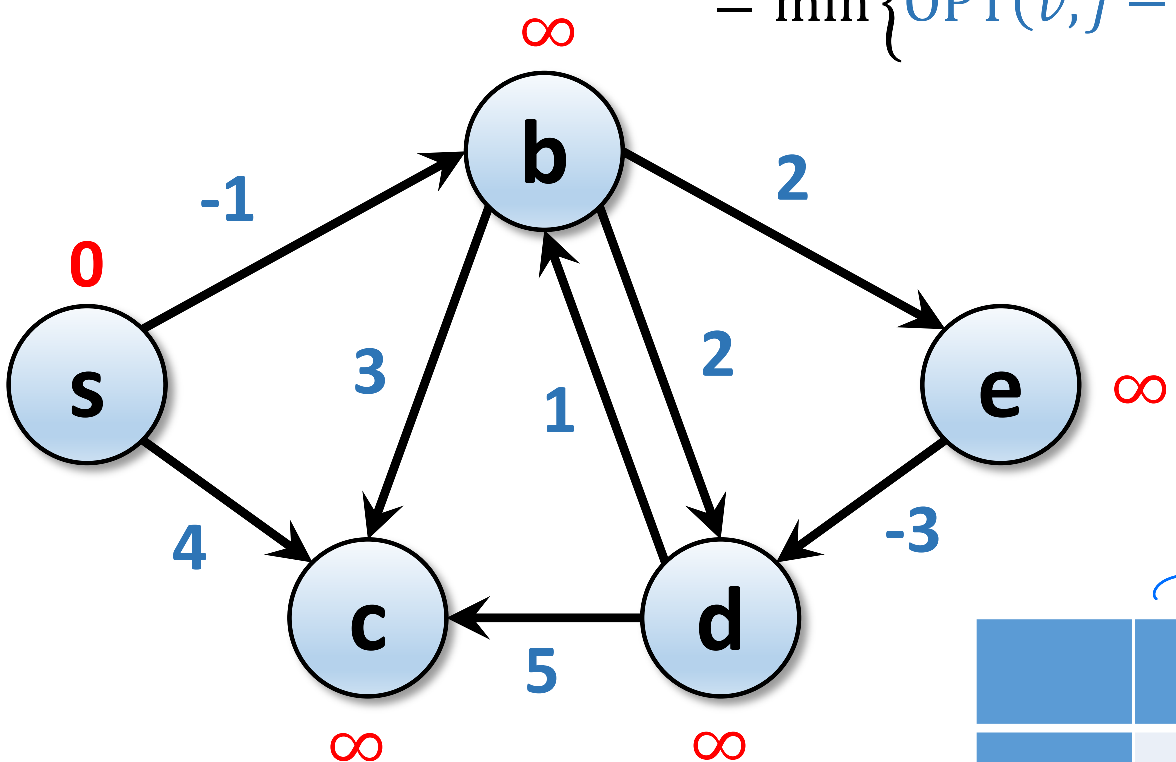
Finding $P(v, j)$: if $\mathrm{OPT}(v,j) = \mathrm{OPT}(v,j-1)$, then $P(v,j) = P(v,j-1)$

else if $\mathrm{OPT}(v,j) = \mathrm{OPT}(u,j-1) + w_{uv}$ then

$P(v,j) = u$.

Example

$$\mathrm{OPT}(v,j) = \min\left\{\mathrm{OPT}(v,j-1),\ \min_{(u,v)\in E}\left\{\mathrm{OPT}(u,j-1)+w_{u,v}\right\}\right\}$$



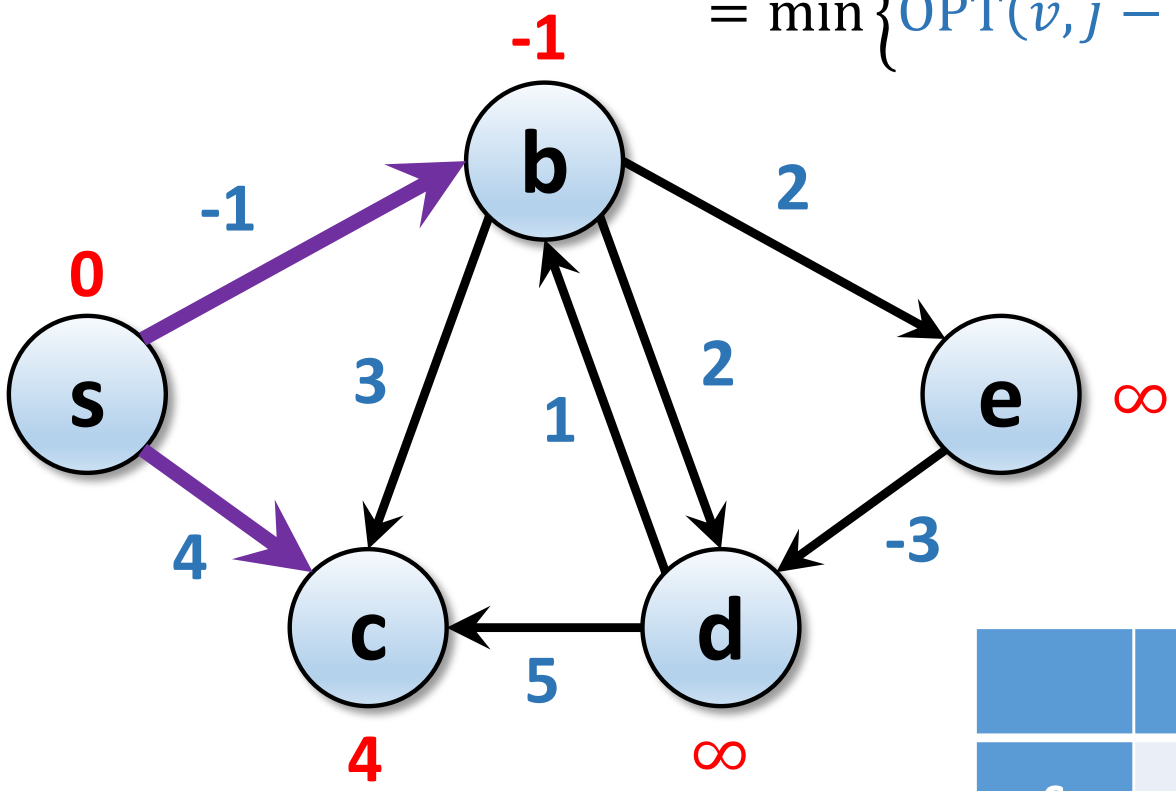
j

vertices

	0	1	2	3	4
s	0				
b	∞				
c	∞				
d	∞				
e	∞				

Example

$$\mathrm{OPT}(v,j) = \min\left\{\mathrm{OPT}(v,j-1),\ \min_{(u,v)\in E}\{\mathrm{OPT}(u,j-1)+w_{u,v}\}\right\}$$



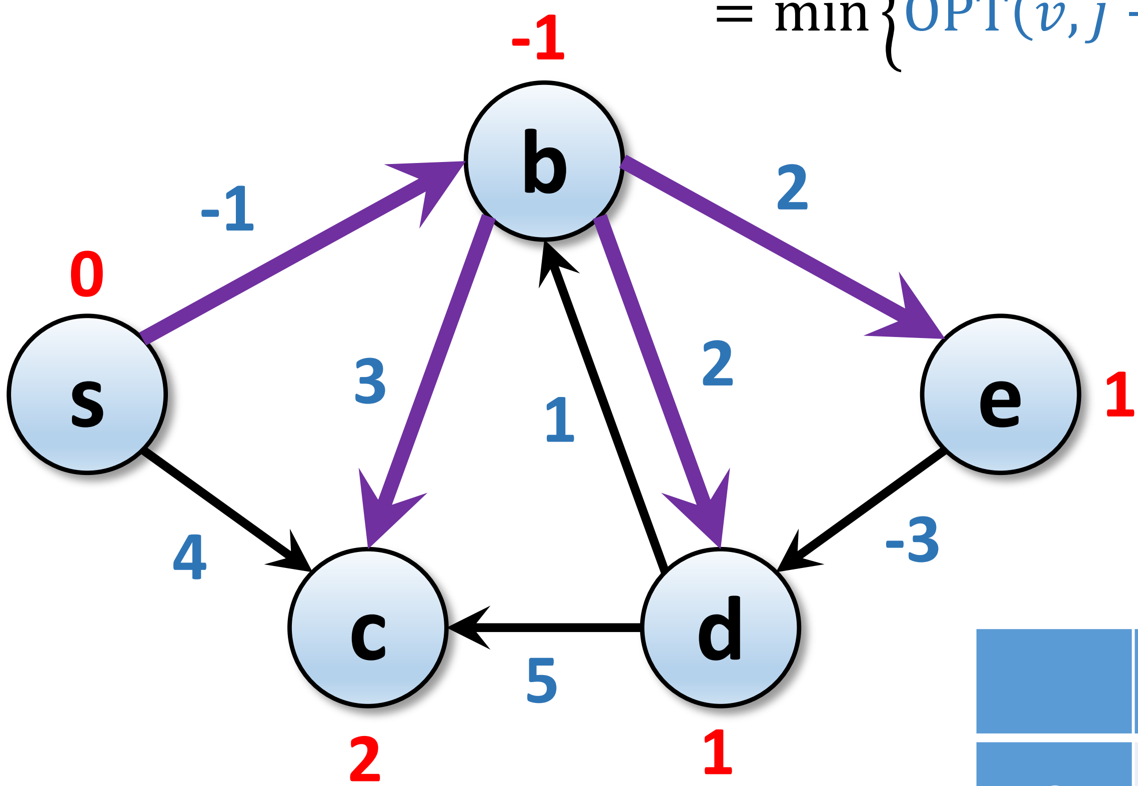
$\mathrm{OPT}(b,1) = \min\{\mathrm{OPT}(b,0),$

$\mathrm{OPT}(s,0) + w_{s,b} = -1\} = -1$

	0	1	2	3	4
s	0	0			
b	∞	-1			
c	∞	4			
d	∞	∞			
e	∞	∞			

Example

$$\mathrm{OPT}(v,j)$$
$$= \min\left\{\mathrm{OPT}(v,j-1),\ \min_{(u,v)\in E}\{\mathrm{OPT}(u,j-1)+w_{u,v}\}\right\}$$



$$\min\{\mathrm{OPT}(b,1)^{-1},\ \mathrm{OPT}(s,1)+w_{s,b},\ -1\quad \mathrm{OPT}(d,1)+w_{d,b}\quad \infty\} = -1$$

	0	1	2	3	4
s	0	0	0		
b	∞	-1	-1		
c	∞	4	2		
d	∞	∞	1		
e	∞	∞	1		

$\mathrm{OPT}(c,2) = \min\{\mathrm{OPT}(c,1)^{4},$ 2

$\mathrm{OPT}(S,1)+w_{s,c},$ 4

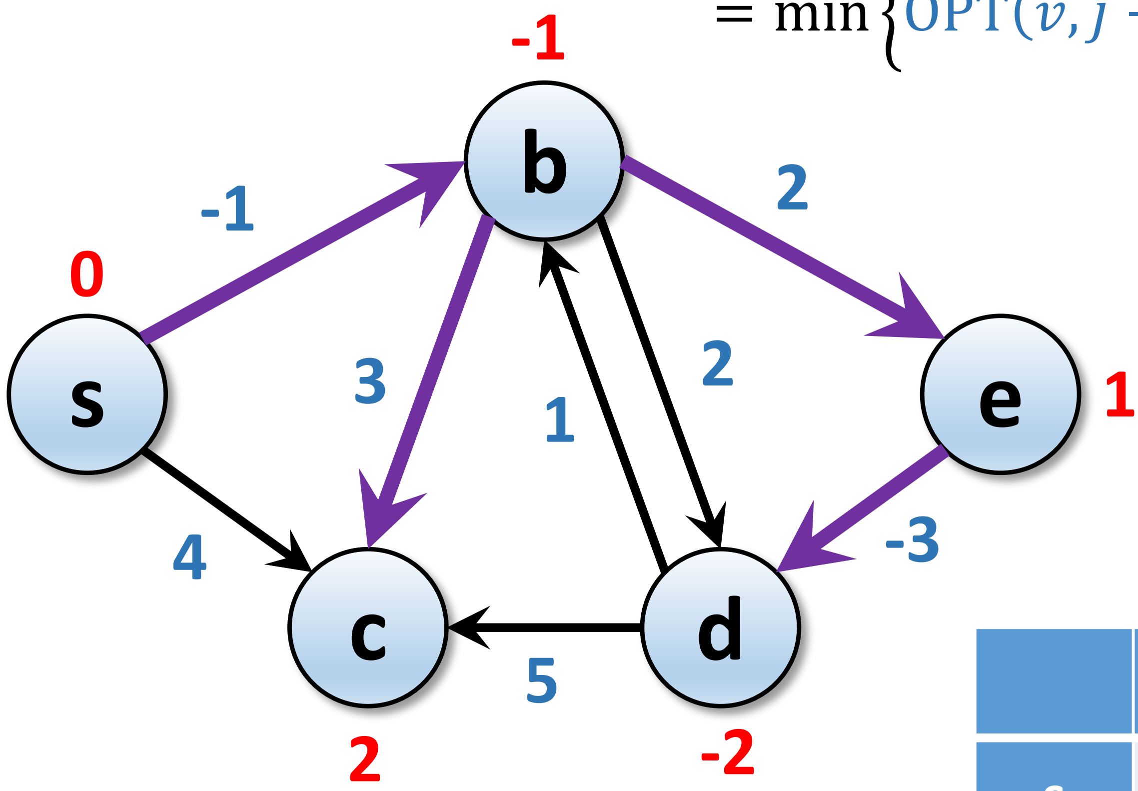
$\mathrm{OPT}(\overset{-1}{b},1)+w^{3}_{b,c},$ 2

$\mathrm{OPT}(d,1)+w_{d,c}$ ∞

$\}$

Example

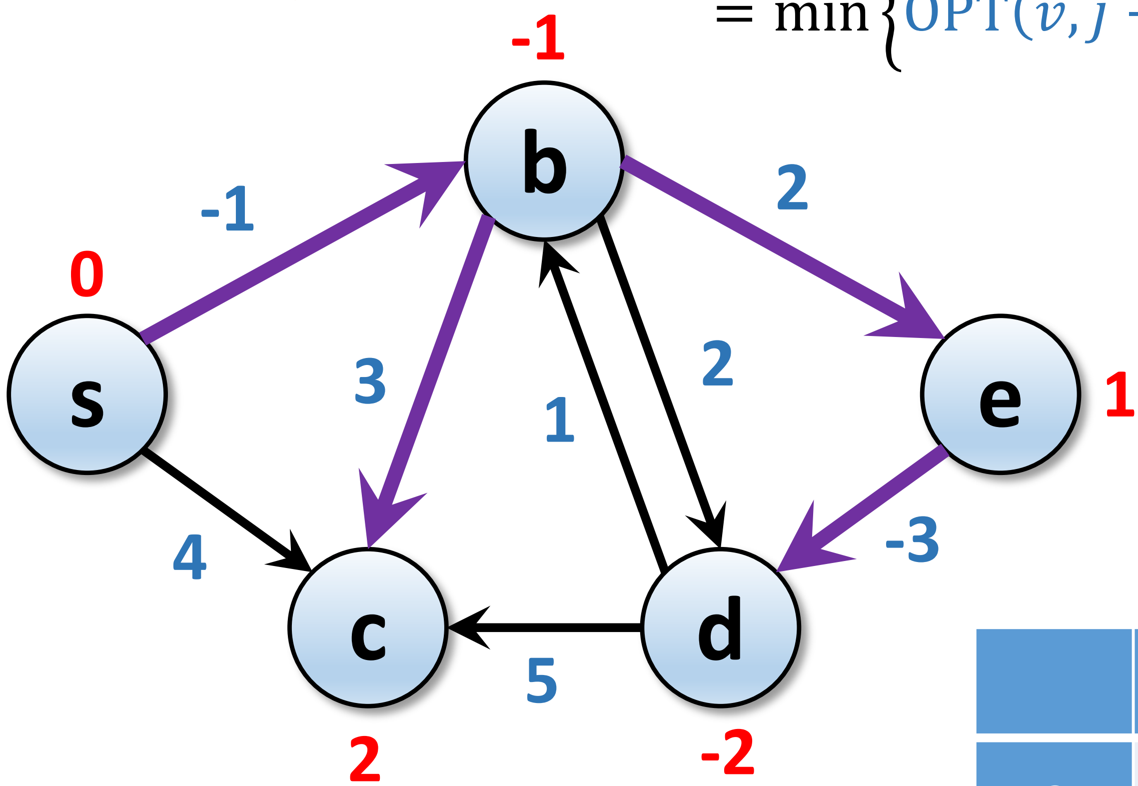
$$\mathrm{OPT}(v,j) = \min\left\{\mathrm{OPT}(v,j-1),\ \min_{(u,v)\in E}\{\mathrm{OPT}(u,j-1)+w_{u,v}\}\right\}$$



	0	1	2	3	4
s	0	0	0	0	
b	∞	-1	-1	-1	
c	∞	4	2	2	
d	∞	∞	1	-2	
e	∞	∞	1	1	

Example

$$\mathrm{OPT}(v,j) = \min\left\{\mathrm{OPT}(v,j-1), \min_{(u,v)\in E}\{\mathrm{OPT}(u,j-1)+w_{u,v}\}\right\}$$



	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1	-1	-1
c	∞	4	2	2	2
d	∞	∞	1	-2	-2
e	∞	∞	1	1	1

Implementation (Bottom Up)

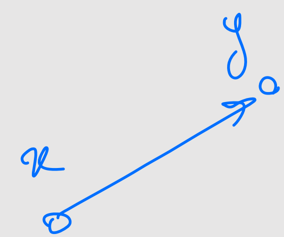
```
Shortest-Path(G, s)
    foreach node v ∈ V
        D[v,0] ← ∞
        P[v,0] ← ⊥
    D[s,0] ← 0

    for i = 1 to n-1
        foreach node v ∈ V
          D[v,i] ← D[v,i-1]
          P[v,i] ← P[v,i-1]
          foreach edge (u,v) ∈ E
              if (D[u,i-1] + w_uv < D[v,i])
                  D[v,i] ← D[u,i-1] + w_uv
                  P[v,i] ← u
```

n times →

$\Theta(m+n)$ time

can be done in $O(\deg^{-}(x))$ time by storing reverse adj lists



Time: $\Theta(n(m+n))$.

Space: $\Theta(n^2)$ + storing the graph

Optimizations

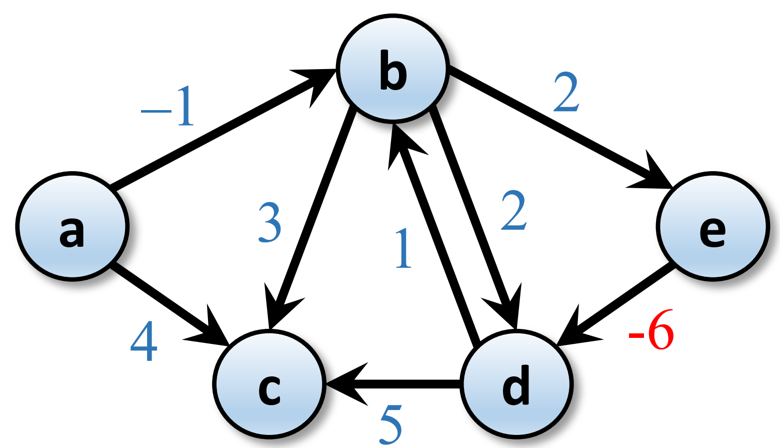
- One array $d[v]$ containing shortest path found so far
  - Once you have *OPT(v, j),* don't care about *OPT(v, j-1)*
- No need to check edges (u, v) unless $d[u]$ has changed
- Stop if no $d[v]$ has changed for a full pass through V

- **Time:** $\Theta(n(m+n))$

- **Space:** $O(n)$ + storing the graph

Negative Cycle Detection

- **Algorithm:**
  - Pick a node $s \in V$
  - Run Bellman-Ford for n iterations
  - Check if $OPT(v, n) < OPT(v, n-1)$ for some $v \in V$
    - If no, then there are no negative cycles
    - If yes, the shortest $s - v$ path contains a negative cycle



Optimized Implementation w/ Negative Cycle Detection

```
Efficient-Shortest-Path(G, s)
   foreach node v ∈ V
      D[v] ← ∞
      P[v] ← ⊥
   D[s] ← 0

   for i = 1 to n
    foreach edge (u,v) ∈ E where D[u] changed
       during last iteration
            if (D[u] + w_uv < D[v])
               D[v] ← D[u] + w_uv
               P[v] ← u
               if (i == n): return "NEG CYCLE"
     if (no D[u] changed): return (D,P)
```

Shortest Paths Summary

- **Input:** Directed, weighted graph $G = (V, E, \{w_e\})$, source node s
- **Output:** Two arrays d, p
  - $d[u]$ is the length of the shortest $s \rightsquigarrow u$ path
  - $p[u]$ is the final hop on shortest $s \rightsquigarrow u$ path

- **Non-negative lengths**: Dijkstra's Algorithm solves in $O(m \log n)$ time
- **Negative lengths:** Bellman-Ford solves in $O(nm)$ time, or finds a negative cycle

$\rightarrow \; O(m \log^8 n).$