CS3000: Algorithms & Data
Jonathan Ullman

Lecture 15:
- Bellman-Ford

Oct 30, 2018

Schedule

- Midterm II Fri 11/16
- No class Thanksgiving week
- Final 12/11

Bellman-Ford

Dijkstra Recap

Single-source shortest paths

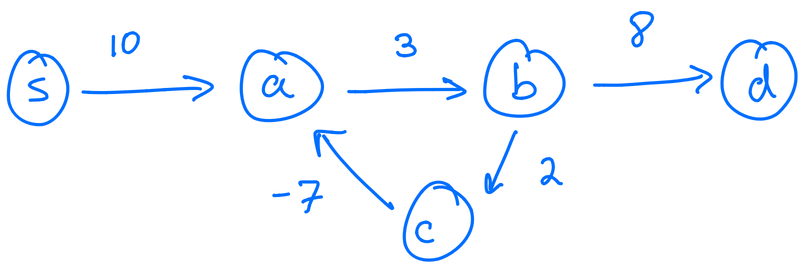
- **Input:** Directed, weighted graph $G = (V, E, \{\ell_e\})$, source node s
  - **Non-negative edge lengths** $\ell_e \geq 0$
- **Output:** Two arrays d, p
  - $d[u]$ is the length of the shortest $s \leadsto u$ path
  - $p[u]$ is the final hop on shortest $s \leadsto u$ path

    ↳ edges in the shortest path tree
- **Running time:** $O(m \log n)$
  - Implement using **binary heaps**

Ask the Audience

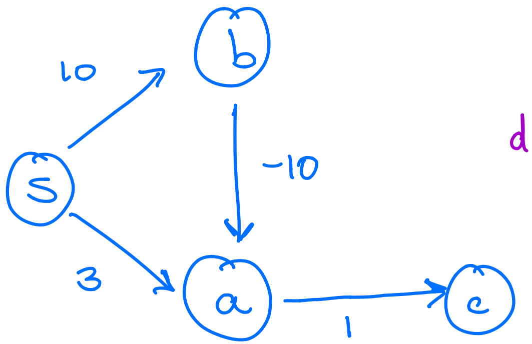
- Show that Dijkstra's Algorithm can fail in graphs with negative edge lengths

① Negative length cycles — shortest path may be undefined



Ask the Audience

- Show that Dijkstra’s Algorithm can fail in graphs with negative edge lengths, and no negative-length cycles.



	s	a	b	c
d(u)	$\boxed{0}$	∞	∞	∞
	0	$\boxed{3}$	10	∞
	0	3	10	$\boxed{4}$
	0	3	$\boxed{10}$	4
	0	0	10	4

Have explored all nodes, but distances are wrong

Why Care About Negative Lengths?

- Models various phenomena
  - Transactions (credits and debits)
  - Currency exchange (log(exchange rate) can be + or -)
  - Chemical reactions (can be exo or endothermic)

- Leads to interesting algorithms
  - Variants of Bellman-Ford are used in internet routing

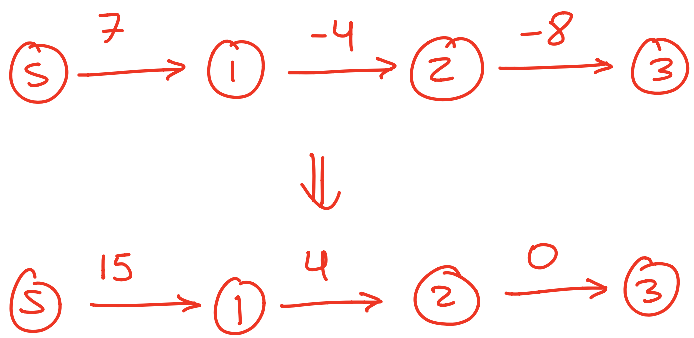
Bellman-Ford

- **Input:** Directed, weighted graph $G = (V, E, \{\ell_e\})$, source node s
  - **Possibly negative edge lengths $\ell_e \in \mathbb{R}$**
  - **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

Ask the Audience

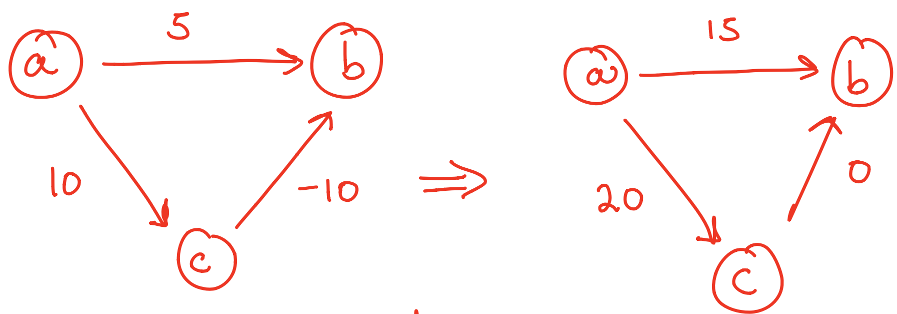
- Suppose we try the following algorithm
    - Take a graph $G = (V, E, \{\ell(e)\})$ with negative lengths
    - Add C to all lengths to make them non-negative
    - Run Dijkstra on the new graph
- Why wont this work?



Ask the Audience

- Suppose we try the following algorithm
  - Take a graph $G = (V, E, \{\ell(e)\})$ with negative lengths
  - Add C to all lengths to make them non-negative
  - Run Dijkstra on the new graph
- Why wont this work?

source



add +10

shortest (a,b) path

a → c → b length 0

a → b length 15

Structure of Shortest Paths

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

$s \xrightarrow{d(s,u)} u \xrightarrow{\ell(u,v)} v$

$\exists$ a path P, $\ell(P) = d(s,u) + \ell(u,v)$

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

shortest path P^* makes some final hop (u,v), $\ell(P^*) = d(s,u) + \ell(u,v)$

$s \xrightarrow{d(s,u)} u \xrightarrow{\ell(u,v)} v$

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

$s \xrightarrow{d(s,u)} u \xrightarrow{\ell(u,v)} v$ $\exists\ P = (s \rightsquigarrow u \rightarrow v)$ with length

$\ell(P) = d(s,u) + \ell(u,v)$

Dynamic Programming

$OPT(v) = d(s, v)$

- **Subproblems:** Let $\mathrm{OPT}(v)$ be the length of the shortest path from s to v

- If the shortest path from s to v, has final hop (u, v),

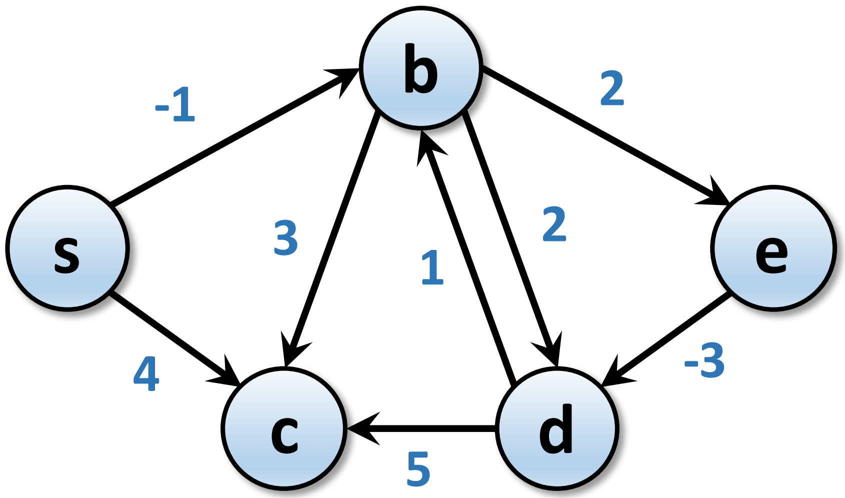
  then $d(s, v) = d(s, u) + \ell(u, v)$

  $OPT(v) = OPT(u) + \ell(u, v)$

- Recurrence: $OPT(v) = \min\limits_{(u,v) \in E} OPT(u) + \ell(u, v)$

- Base Cases: $OPT(s) = 0$

Bottom-Up Implementation?



Bottom up impenentations

⇕

Topologically ordering the "DP Table"

If the graph has cycles, then there is no good order

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

- Fact: If there are no negative-length cycles, then there exists a shortest path with $\leq n-1$ hops

  Pf: If P has $\geq n$ hops it contains a cycle. Removing that cycle cannot increase the length

- $\therefore$ $\mathrm{OPT}(v, n-1) = d(s, v)$

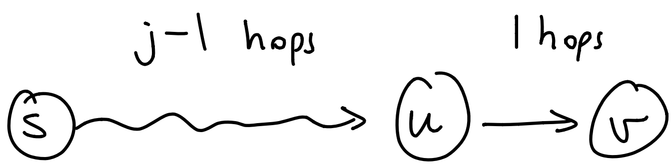
- # of subproblems $= \underbrace{n}_{v \in V} \times \underbrace{n}_{0 \leq j \leq n-1} = n^2$

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

- If P uses $\le j-1$ hops then $\mathrm{OPT}(v,j) = \mathrm{OPT}(v,j-1)$

- If P uses exactly j hops, and its final hop is (u,v) then $\mathrm{OPT}(v,j) = \mathrm{OPT}(u,j-1) + \ell(u,v)$



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

- Recurrence:

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

- Base Cases:

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

$$\mathrm{OPT}(v, 0) = \infty \quad \forall v \neq s$$

Recurrence

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

Recurrence:

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

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

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

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:

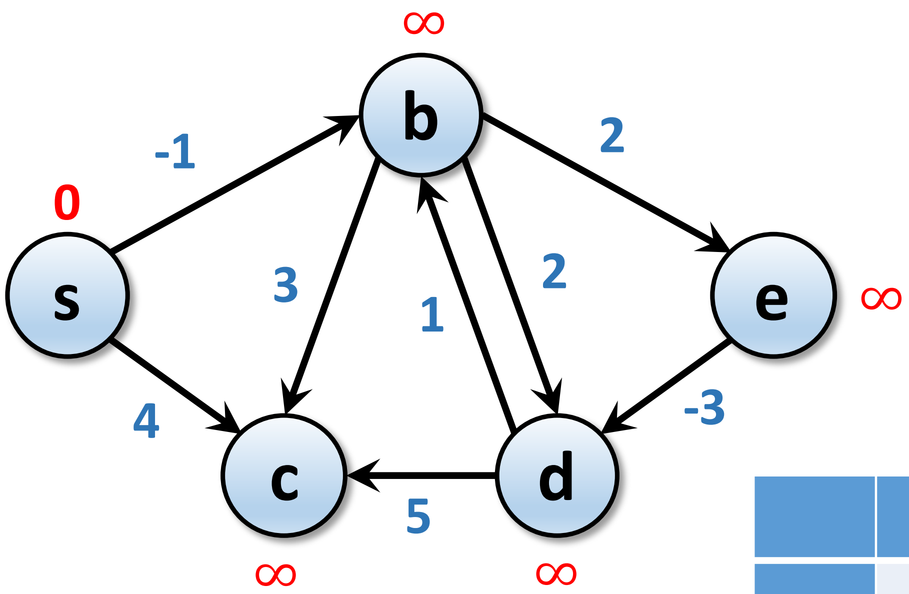
$$\mathrm{OPT}(v, j) = \min\left\{\mathrm{OPT}(v, j-1), \min_{(u,v)\in E}\{\mathrm{OPT}(u, j-1) + \ell_{u,v}\}\right\}$$

$P(v, j) \leftarrow P(v, j-1)$

If u is the min

$P(v, j) \leftarrow u$

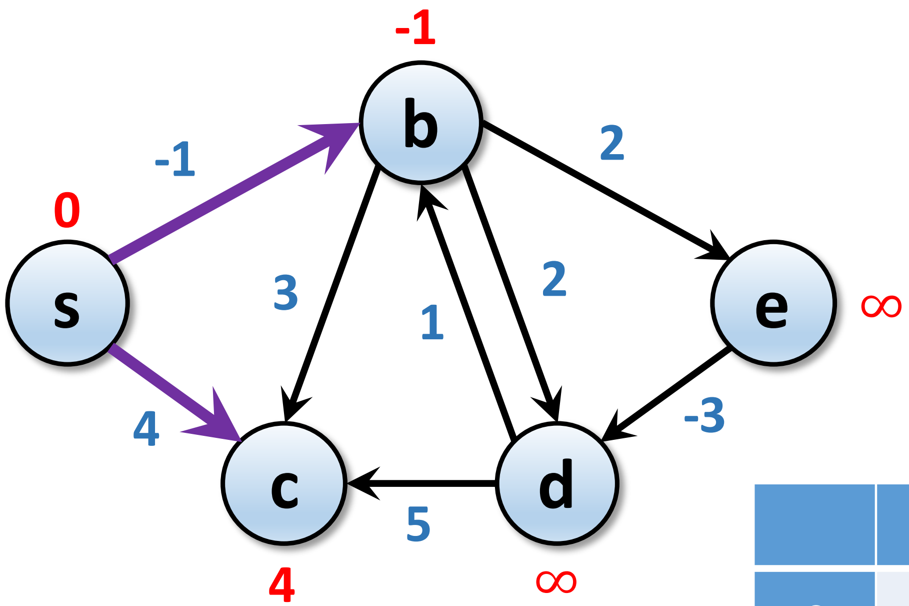
Example



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

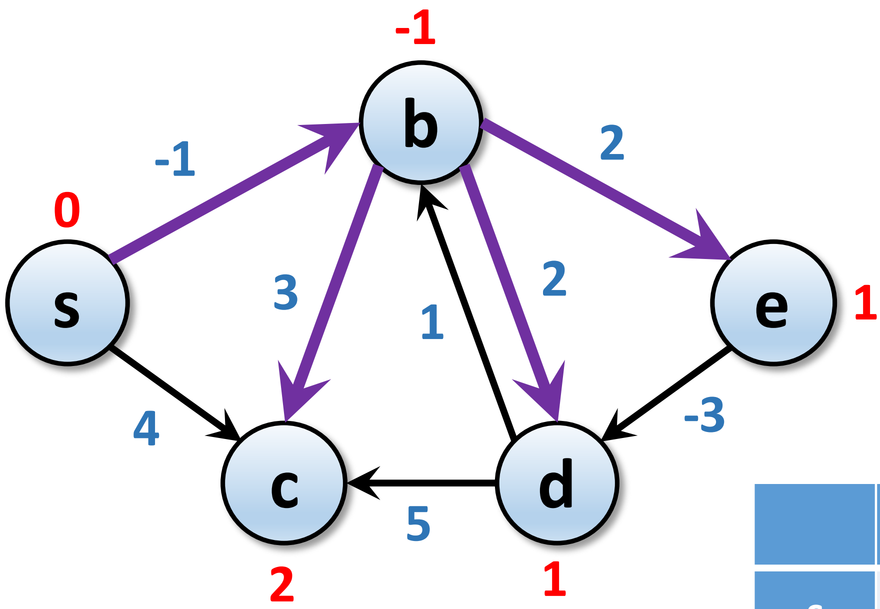
Example

$$OPT(v,1) = \min\left\{ \begin{array}{c} OPT(v,0), \\ \min\limits_{(u,v)\in E} OPT(u,0)+\ell(u,v) \end{array} \right\}$$



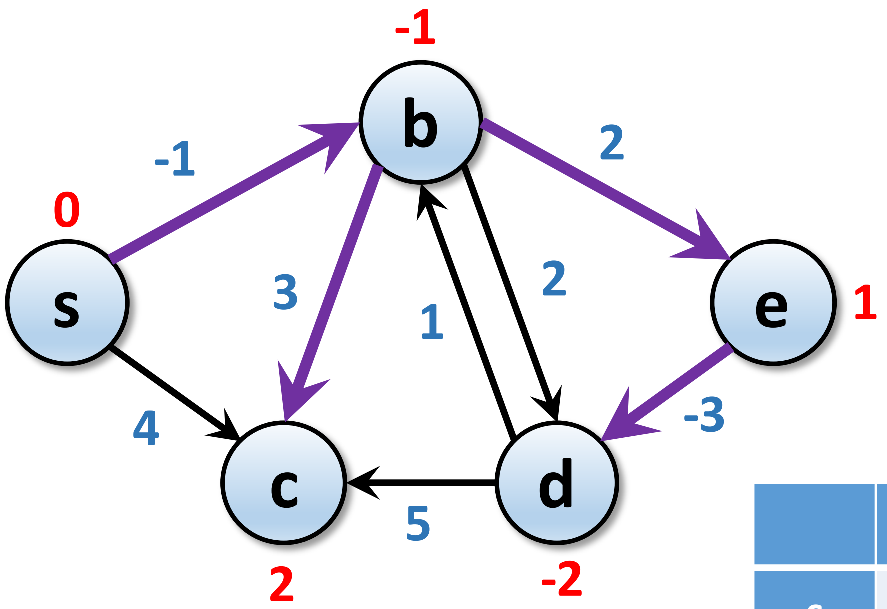
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1			
c	∞	4			
d	∞	∞			
e	∞	∞			

Example



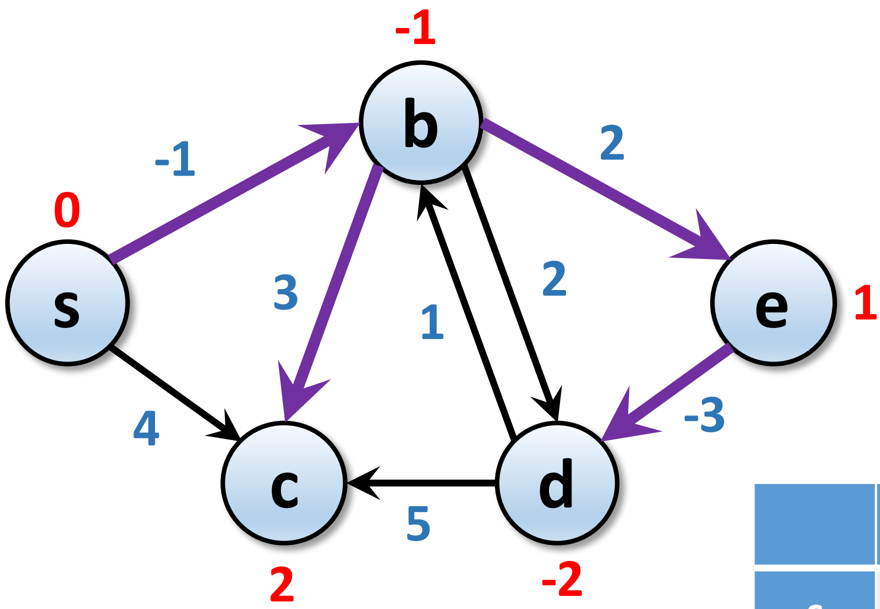
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1		
c	∞	4	2		
d	∞	∞	1		
e	∞	∞	1		

Example



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

Example



	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 DP)

```
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] + ℓuv < D[v,i])
                  D[v,i] ← D[u,i-1] + ℓuv
                  P[v,i] ← u
```

$O(n)$

$\rightarrow$ n-1 iterations (one per column)

$O(1)$

$O(1)$

$O(\text{indeg}(v))$

$\sum_{v \in V} O(\text{indeg}(v)) = O(m)$

Total Time: $O(nm)$

Space: $O(n^2)$

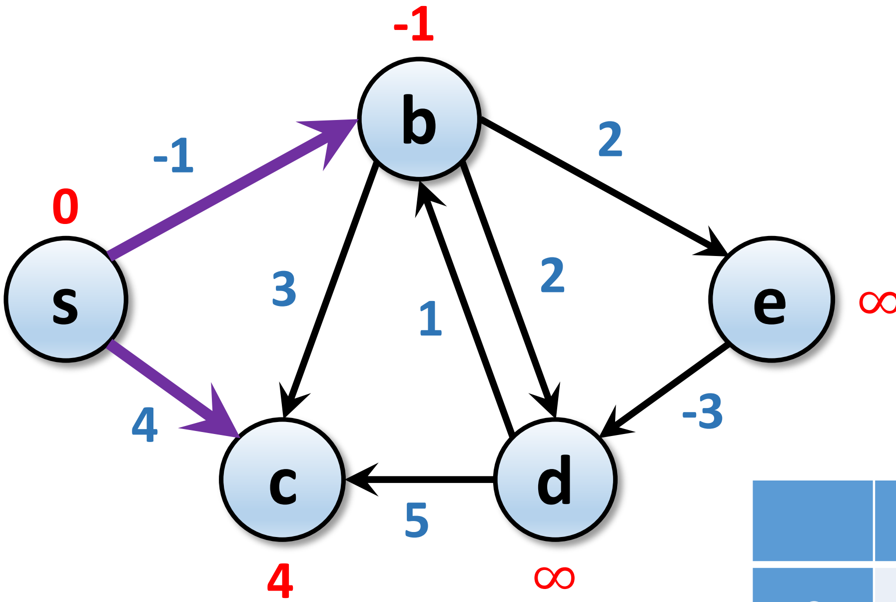
Optimizations

- One array $d[v]$ containing shortest path found so far
- 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

- **Theorem:**
  - Throughout the algorithm $M[v]$ is the length of some $s - v$ path
  - After i passes through the nodes, $M[v] \leq OPT(v, i)$

Example

$$OPT(v,1) = \min \begin{Bmatrix} OPT(v,0), \\ \min\limits_{(u,v)\in E} OPT(u,0) + \ell(u,v) \end{Bmatrix}$$



only d[b] changed

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

Implementation II

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

   for i = 1 to n
      foreach node u ∈ V
        if (D[u] changed in the last iteration)
           foreach edge (u,v) ∈ E
           if (D[u] + ℓ_uv < D[v])
               D[v] ← D[u] + ℓ_uv
               P[v] ← u
        if (no D[u] changed): return (D,P)
```

Running Time: $O(nm)$ [but typically much faster]

Space: $O(n+m)$

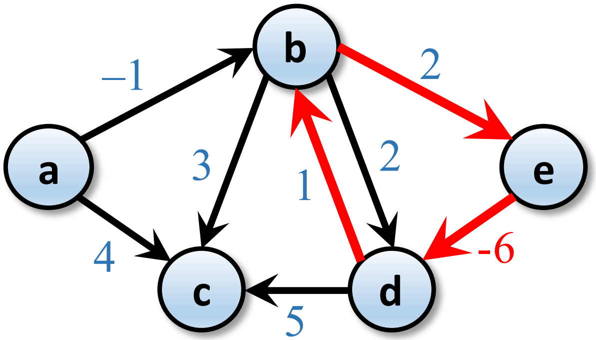
Negative Cycle Detection

- **Claim 1:** if $OPT(v, n) = OPT(v, n-1)$ then there are no negative cycles reachable from s

- **Claim 2:** if $OPT(v, n) < OPT(v, n-1)$ then any shortest $s - v$ path contains a negative cycle

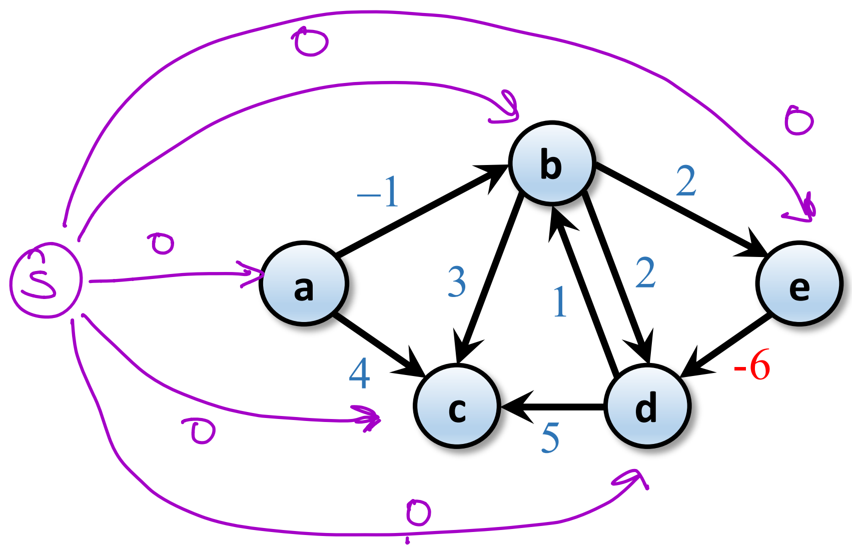
Negative Cycle Detection

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



Negative Cycle Detection

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



Shortest Paths Summary

- **Input:** Directed, weighted graph $G = (V, E, \{\ell_e\})$, 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 $O(n + m)$ space, or finds a negative cycle