

CS3000: Algorithms & Data

Jonathan Ullman

Lecture 12:

- Shortest Paths: Finish Dijkstra, Bellman-Ford

Feb 26, 2020

Shortest Paths: Bellman-Ford

Dijkstra Recap

- **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 $s \rightsquigarrow u$ path
 - $p(u)$ is the final hop on shortest $s \rightsquigarrow u$ path
- **Running time:** $O(m \log n)$

Ask the Audience

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

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 endo-thermic)
 - ...
- 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

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

Structure of Shortest Paths

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

Dynamic Programming

Dynamic Programming

Dynamic Programming Take II

Recurrence

- **Subproblems:** $\text{OPT}(v, j)$ 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:

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

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

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

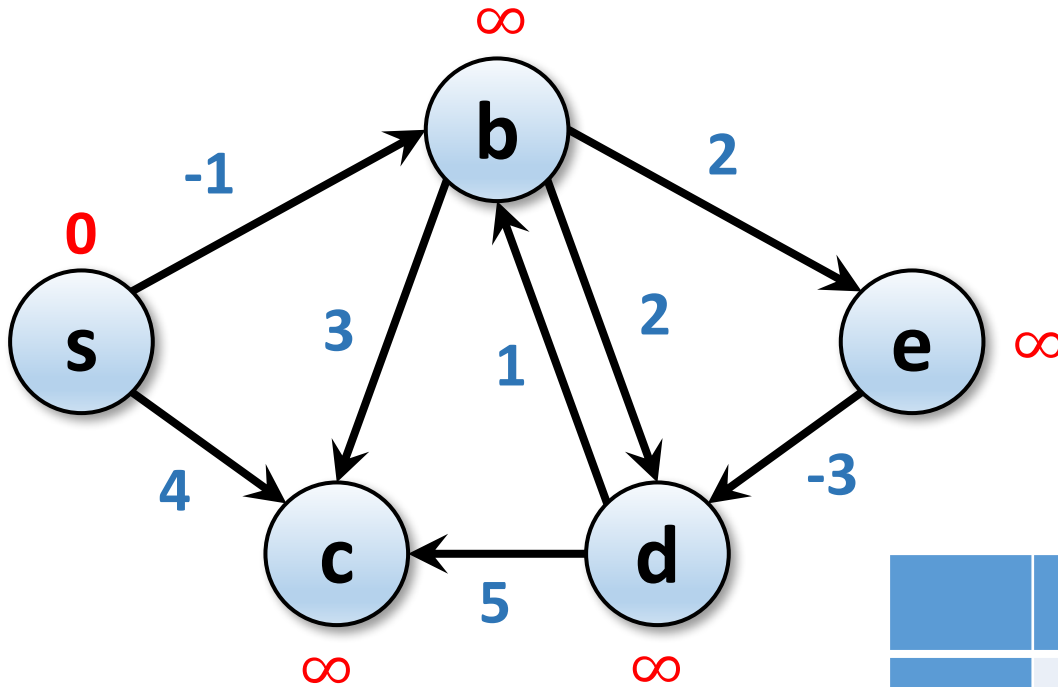
Finding the paths

- $\text{OPT}(v, j)$ 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:

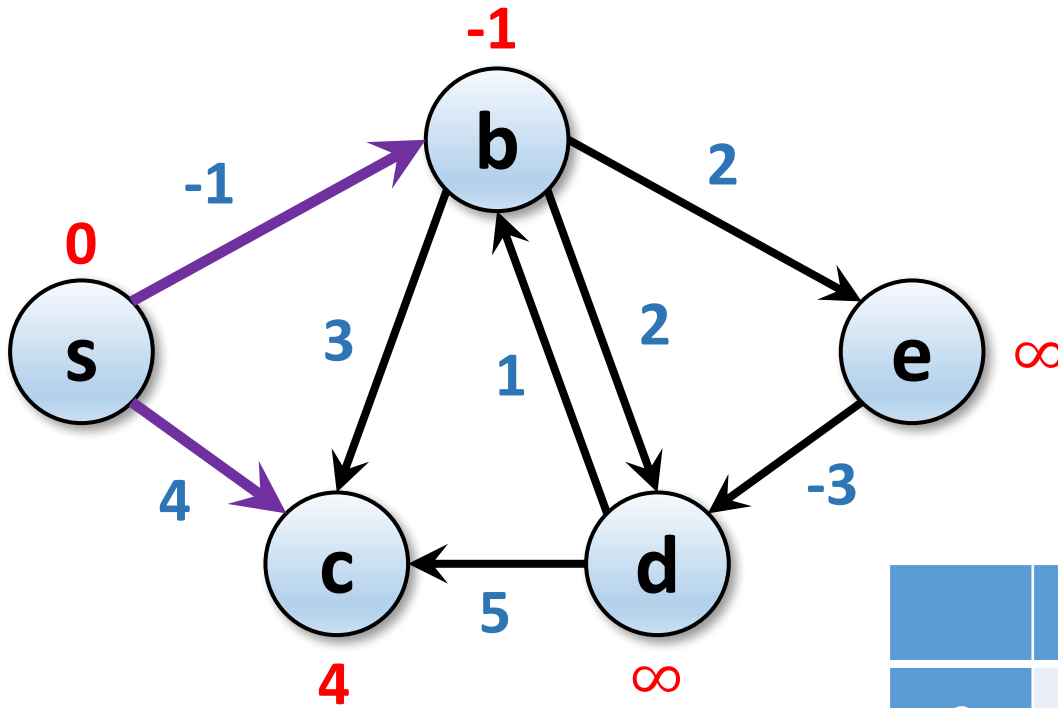
$$\text{OPT}(v, j) = \min \left\{ \text{OPT}(v, j-1), \min_{(u,v) \in E} \{ \text{OPT}(u, j-1) + \ell_{u,v} \} \right\}$$

Example



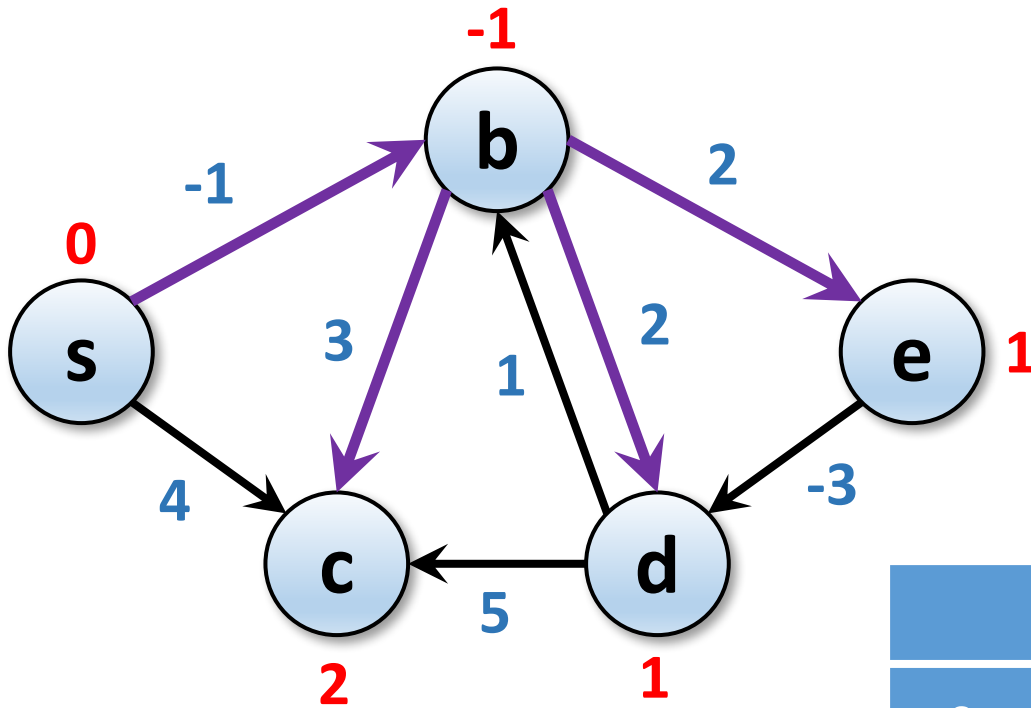
	0	1	2	3	4
s	0	0	0	0	0
b	∞				
c	∞				
d	∞				
e	∞				

Example



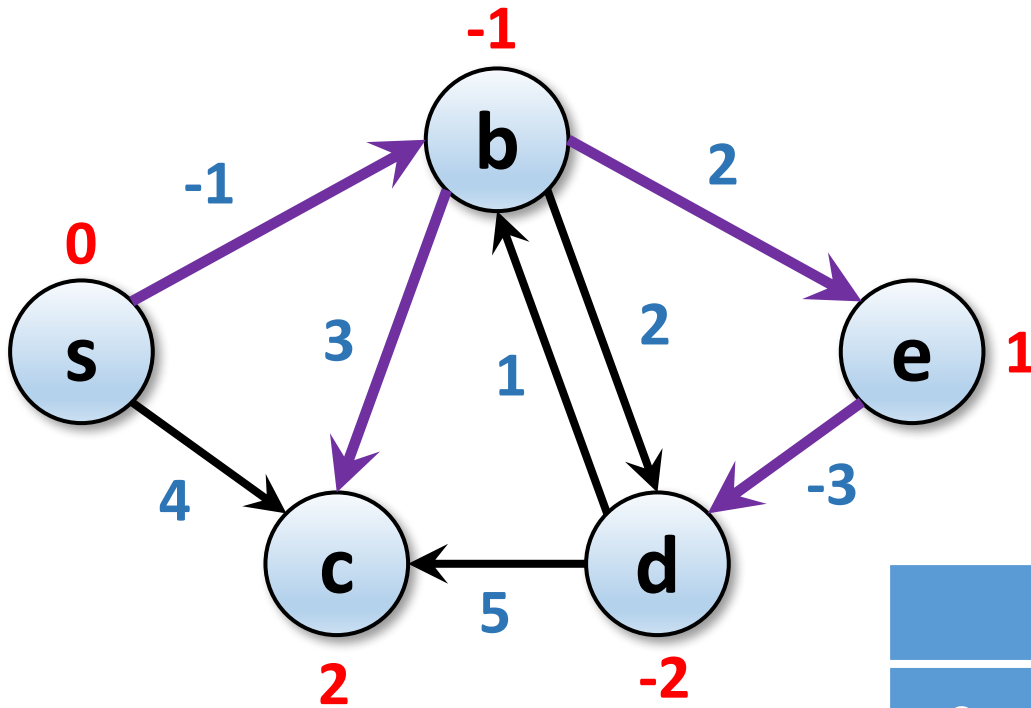
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1			
c	∞	4			
d	∞	∞			
e	∞	∞			

Example



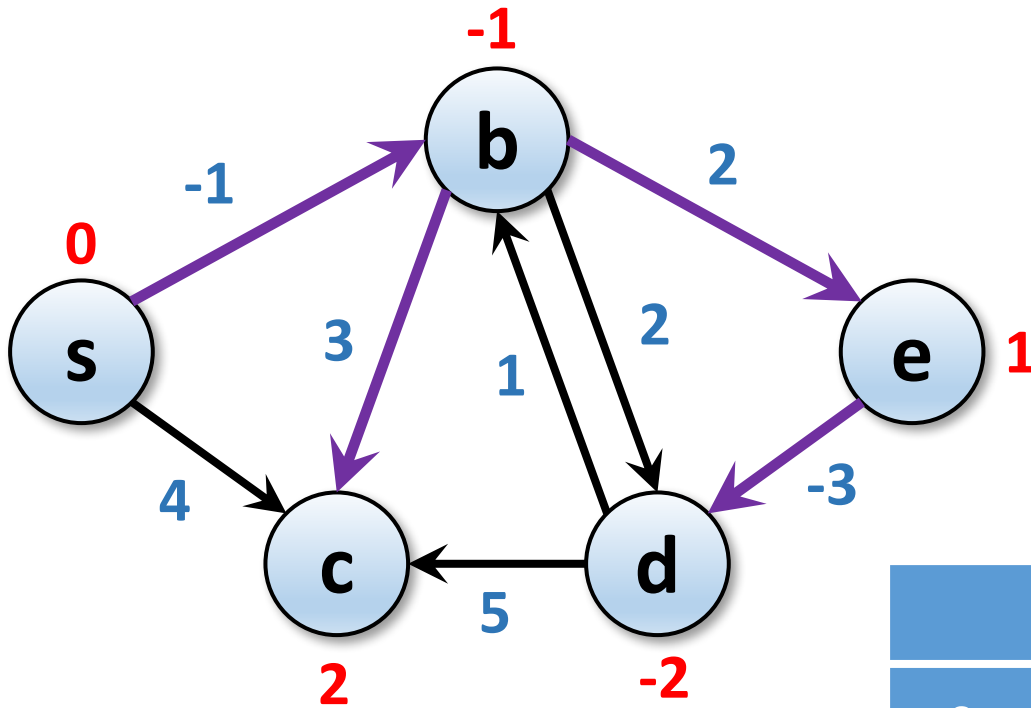
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1		
c	∞	4	2		
d	∞	∞	1		
e	∞	∞	1		

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)

```
Shortest-Path(G, s)
```

```
  foreach node  $v \in V$ 
```

```
     $M[0, v] \leftarrow \infty$ 
```

```
     $P[0, v] \leftarrow \phi$ 
```

```
   $M[0, s] \leftarrow 0$ 
```

```
  for  $i = 1$  to  $n-1$ 
```

```
    foreach node  $v \in V$ 
```

```
       $M[i, v] \leftarrow M[i-1, v]$ 
```

```
       $P[i, v] \leftarrow P[i-1, v]$ 
```

```
      foreach edge  $(v, w) \in E$ 
```

```
        if  $(M[i-1, w] + \ell_{wv} < M[i, v])$ 
```

```
           $M[i, v] \leftarrow M[i-1, w] + \ell_{wv}$ 
```

```
           $P[i, v] \leftarrow w$ 
```

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, $M[v] \leq OPT(v, i)$

Implementation II

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

  for i = 1 to n-1
    foreach node w ∈ V
      if (M[w] changed in the last iteration)
        foreach edge (w,v) ∈ E
          if (M[w] +  $l_{wv}$  < M[v])
            M[v] ← M[w] +  $l_{wv}$ 
            P[v] ← w
    if (no M[w] changed): return M
```

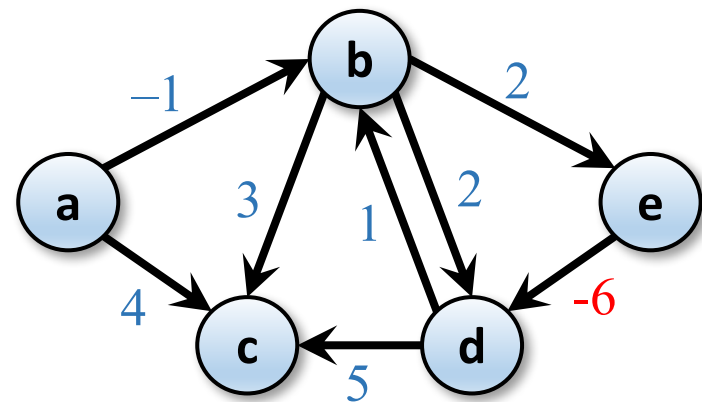
Negative Cycle Detection

- **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:**

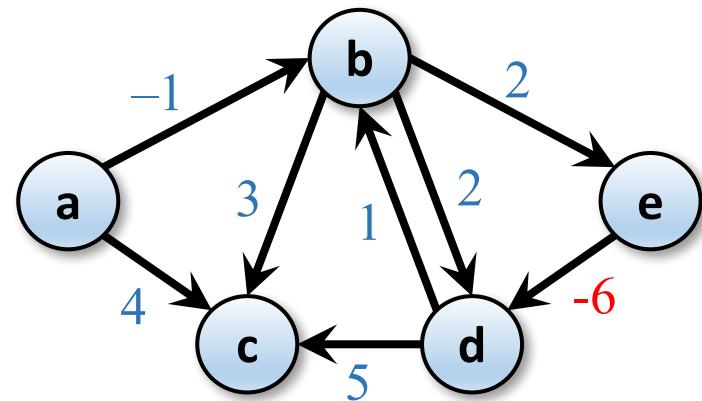
- Pick a node $a \in V$
- Run Bellman-Ford for n iterations
- Check if $OPT(v, n) \neq OPT(v, n - 1)$ for some $v \in V$
 - If no, then there are no negative cycles
 - If yes, the shortest $a - v$ path contains a negative cycle



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:**
- **Informal Version:**
 - **Maintain a set S of explored nodes**
 - **Maintain an upper bound on distance**
 - If u is explored, then we know $d(u)$ (**Key Invariant**)
 - If u is explored, and (u, v) is an edge, then we know $d(v) \leq d(u) + \ell(u, v)$
 - **Explore the “closest” unexplored node**
 - **Repeat until we’re done**

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** ($\ell_e \geq 0$): Dijkstra's Algorithm can solve in $O(m \log n)$ time
- **Negative lengths:** Bellman-Ford solves in $O(nm)$ time, or finds a negative-length cycle