CS3000: Algorithms & Data
Jonathan Ullman

Lecture 12:

- Shortest Paths: Finish Dijkstra, Bellman-Ford

Feb 26, 2020

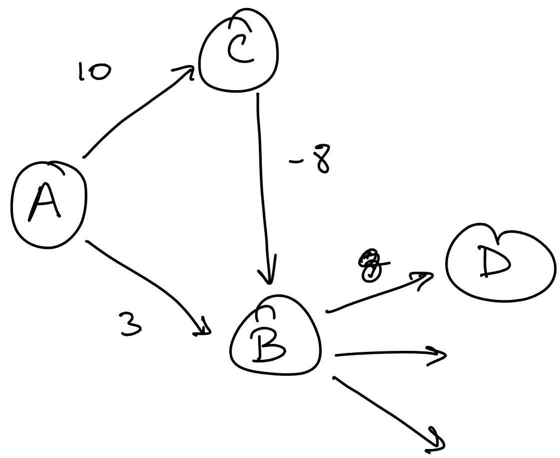
Shortest Paths: Bellman-Ford

Dijkstra Recap

- **Input:** Directed, weighted graph $G = (V, E, \{\ell_e\})$, source node s
  - **Non-negative edge lengths** $\ell_e \geq 0$

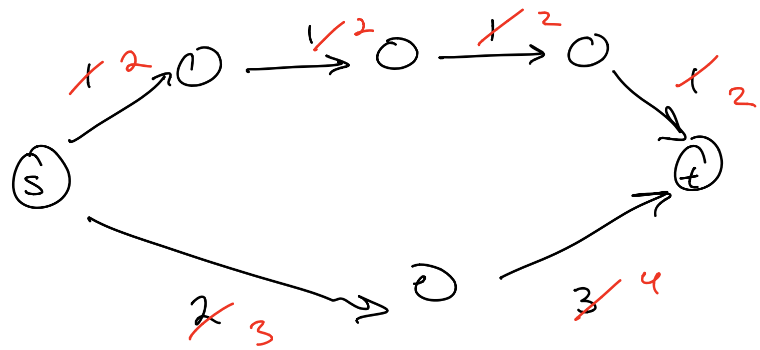
- **Output:** Two arrays d, p
  - $d(u)$ is the length of the shortest $s \rightsquigarrow u$ path
  - $p(u)$ is the final hop on shortest $s \rightsquigarrow u$ path

- **Running time:** $O(m \log n)$



A	B	C	D
0 ✓	3	10	∞
0 ✓	3 ✓	10	11
0 ✓	2 ✓	10 ✓	11

← Invariant breaks down
Explored B, but dont know its distance yet


s
t
1 2
1 2
1 2
1 2
2 3
3 4

Ask the Audience

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

Why Care About Negative Lengths?

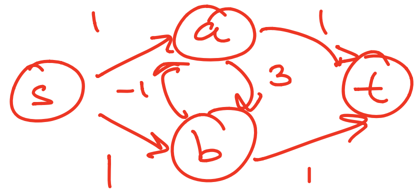
- Models various phenomena
  - Transactions (credits and debits)
  - Currency exchange (log exchange rate can be + or -)
  - Chemical reactions (can be exo- or endo-thermic)
  - …

- 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!**

(Might not be a shortest path)

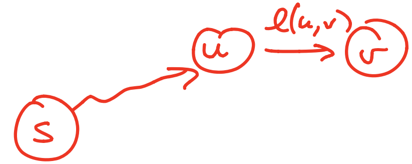


- **Output:** Two arrays d, p
  - $d(u)$ is the length of the shortest $s \rightsquigarrow u$ path
  - $p(u)$ is the final hop on shortest $s \rightsquigarrow u$ path

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 "the" shortest path from $s \rightsquigarrow v$ ends with the edge $(u \to v)$ then $d(s,v) = d(s,u) + \ell(u \to v)$

$$d(s,v) = \min_{(u,v) \in E} d(s,u) + \ell(u,v)$$

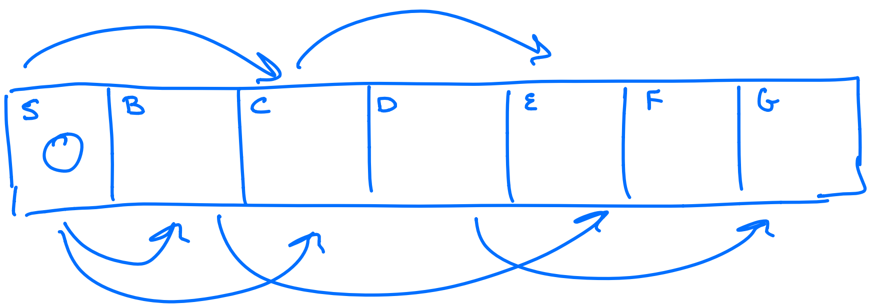
- If $(u, v) \in E$, and $d(s, v) = d(s, u) + \ell(u, v)$ then there is a shortest $s \rightsquigarrow v$-path ending with (u, v)

Dynamic Programming

$OPT(v)$ = length of the shortest path from $s \rightsquigarrow v$

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

$$OPT(s) = 0$$



If the graph cant be topologically ordered then we cannot do bottom-up dynamic programming

Dynamic Programming

Dynamic Programming Take II

$OPT(v, j)$ = the length of the shortest $s \rightsquigarrow v$ path using $\leq j$ hops.

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

$$OPT(s, j) = 0 \quad \forall j$$

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

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:

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

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

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

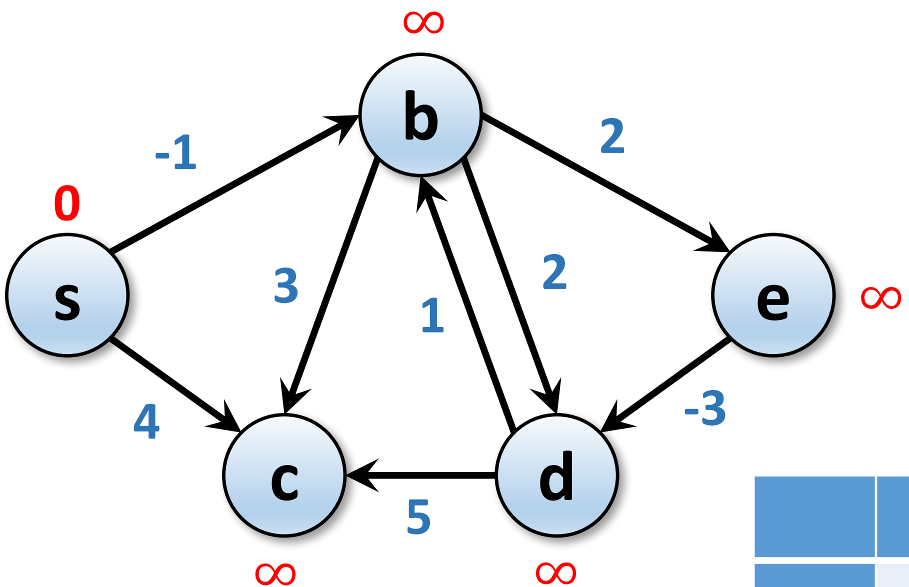
Finding the paths

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

Recurrence:

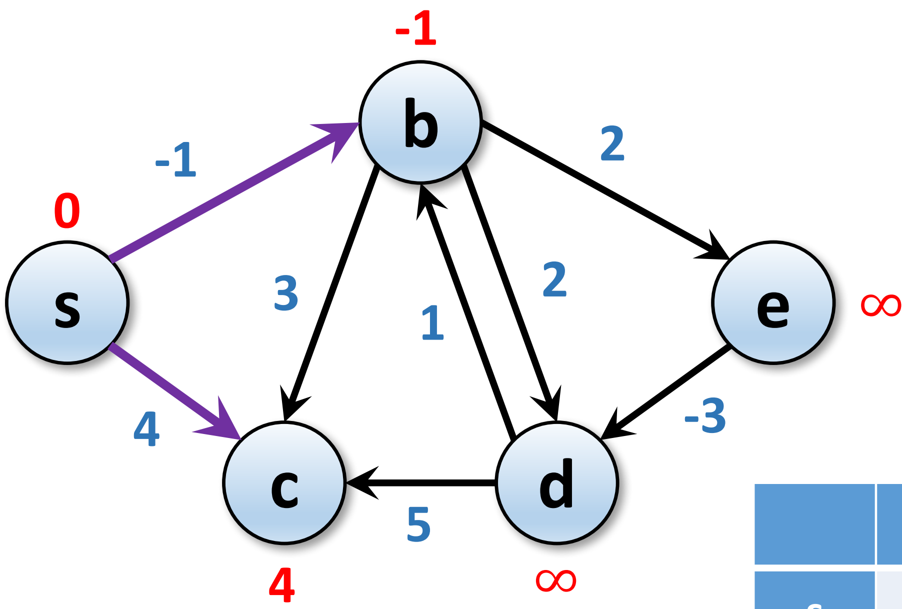
$$\mathrm{OPT}(v, j) = \min\left\{\mathrm{OPT}(v, i-1), \min_{(u,v)\in E}\left\{\mathrm{OPT}(u, i-1) + \ell_{u,v}\right\}\right\}$$

Example



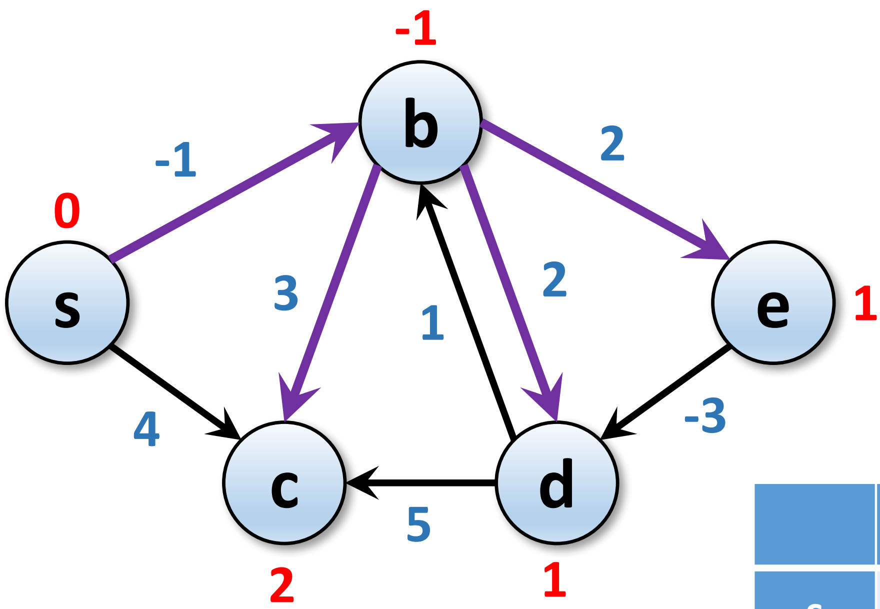
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1		
c	∞	4			
d	∞	∞			
e	∞	∞			

Example



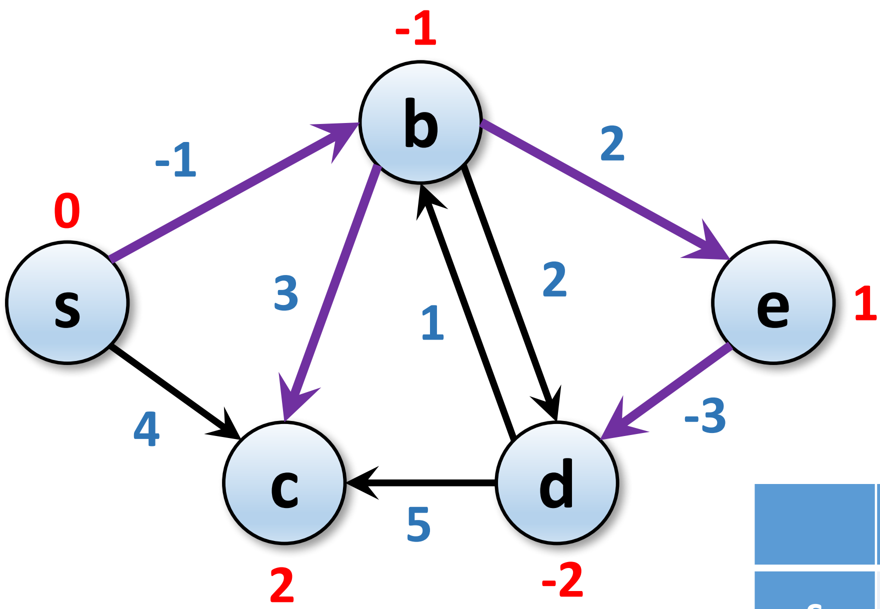
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1		
c	∞	4	2		
d	∞	∞	1		
e	∞	∞	1		

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	

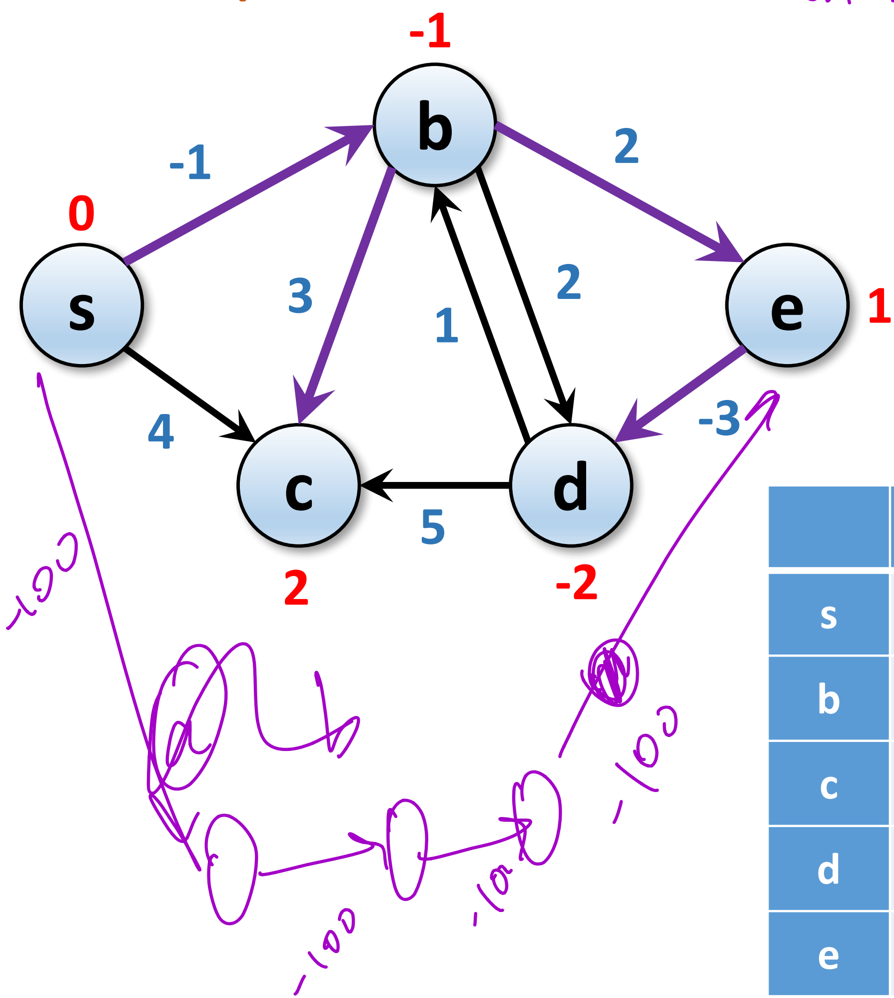
Example



	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

Example

Every shortest path has at most n-1 edges



Why stop at j=n-1?

	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
      M[0,v] ← ∞
      P[0,v] ← ϕ
   M[0,s] ← 0

   for i = 1 to n-1                          ← n columns
      foreach node v ∈ V
        M[i,v] ← M[i-1,v]
        P[i,v] ← P[i-1,v]
        foreach edge (v, w) ∈ E              ← m edges per column
           if (M[i-1,w] + ℓwv < M[i,v])
              M[i,v] ← M[i-1,w] + ℓwv
              P[i,v] ← w
```

Worst-case running time is $O(nm)$

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] + ℓwv < M[v])
                 M[v] ← M[w] + ℓwv
                 P[v] ← w
        if (no M[w] changed): return M
```

Running time is $O(m \cdot \text{diameter})$

most hops in any shortest path

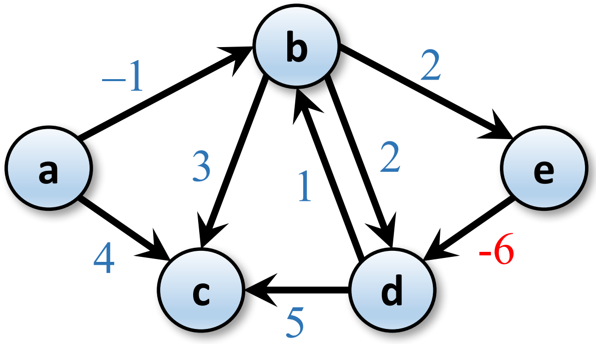
Negative Cycle Detection

- **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

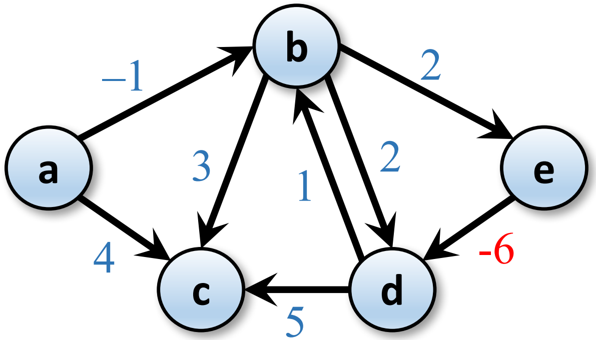
Negative Cycle Detection

- **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