CS3000: Algorithms & Data
Paul Hand

Lecture 18:

- Bellman-Ford Algorithm

Apr 1, 2019

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:** $O(m \log n)$
  - Implement using **binary heaps**

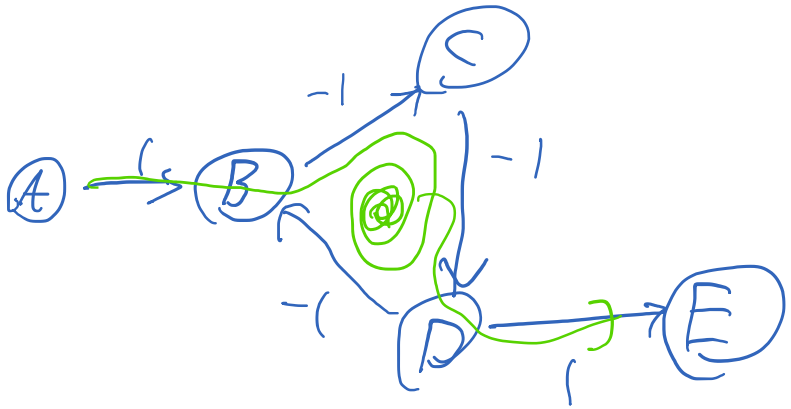
What About Negative Lengths?

- 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!**



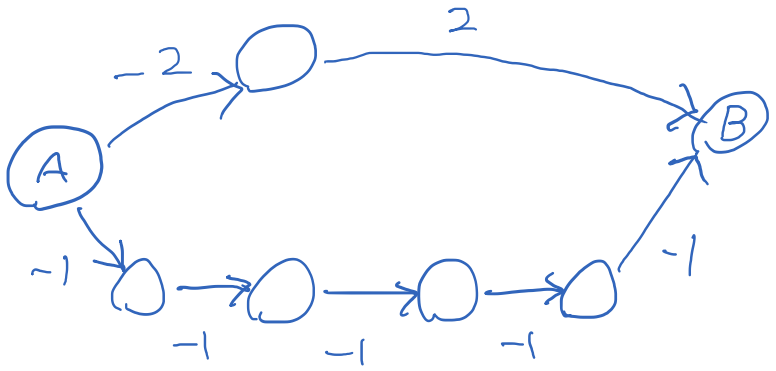
- **Output:** Two arrays d, p
  - $d[u]$ is the length of the shortest $s \rightsquigarrow u$ path
  - $p[u]$ is the final hop on shortest $s \rightsquigarrow u$ path

Could get arb. negative lengths

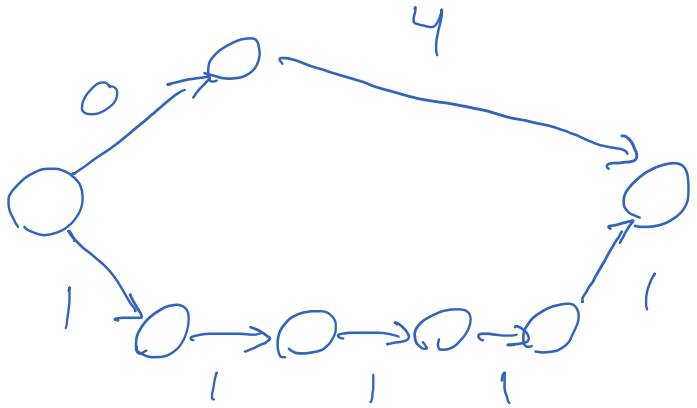
Activity

$C = -\min_e \ell(e)$

- Suppose we try the following algorithm
  - Take a graph $G = (V, E, \{\ell(e)\})$ with negative lengths
  - Add some valueC to all lengths to make them non-negative
  - Run Dijkstra on the new graph
- Activity: Come up with a graph where this fails



Best Path is lower



Best Path is upper one

Structure of Shortest Paths

$d(s,v)$ = length of shortest path from $s \to v$

- If $(u, v) \in E$, then $d(s, v) \leq d(s, u) + \ell(u, v)$ for every node $s \in V$
- For every v, there exists an edge $(u, v) \in E$ such that $d(s, v) = d(s, u) + \ell(u, v)$
- 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

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

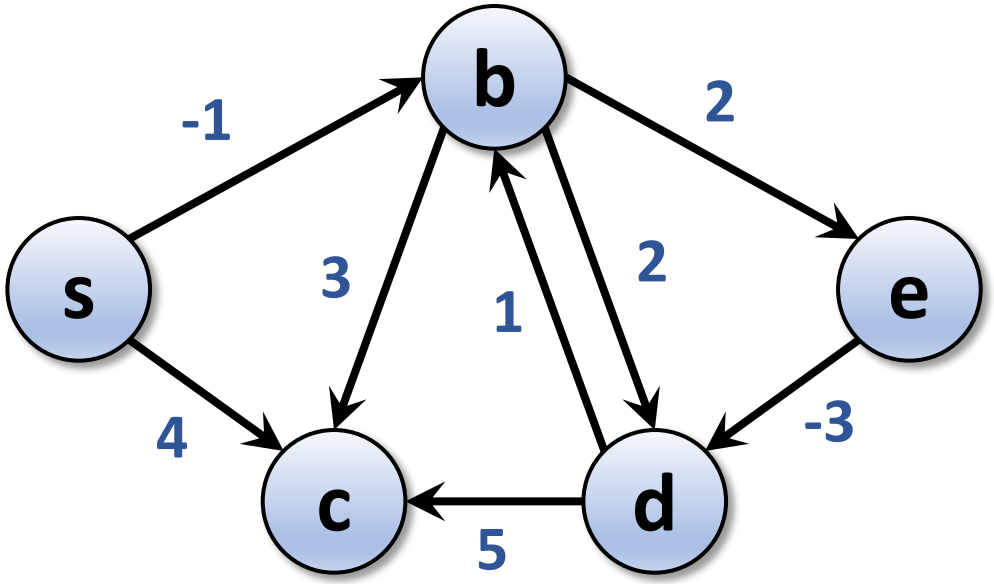
$OPT(s) = 0$ (no cycles of total neg length)

Cases:

u is preceding node

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

Bottom-Up Implementation?



Unclear which node can be filled in first

v	s	b	c	d	e
OPT(v)	0				

Dynamic Programming Take II

adding variable

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

Cases: u precedes v

Have at most $j-1$ steps to get to u

$$OPT(v, j) = \min_{u} \left[OPT(u, j-1) + \ell(u, v) \right]$$

Recurrence

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

was able to reach with fewer hops

Recurrence:

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

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

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

Compute OPT(v, n)

vertices

Finding the paths

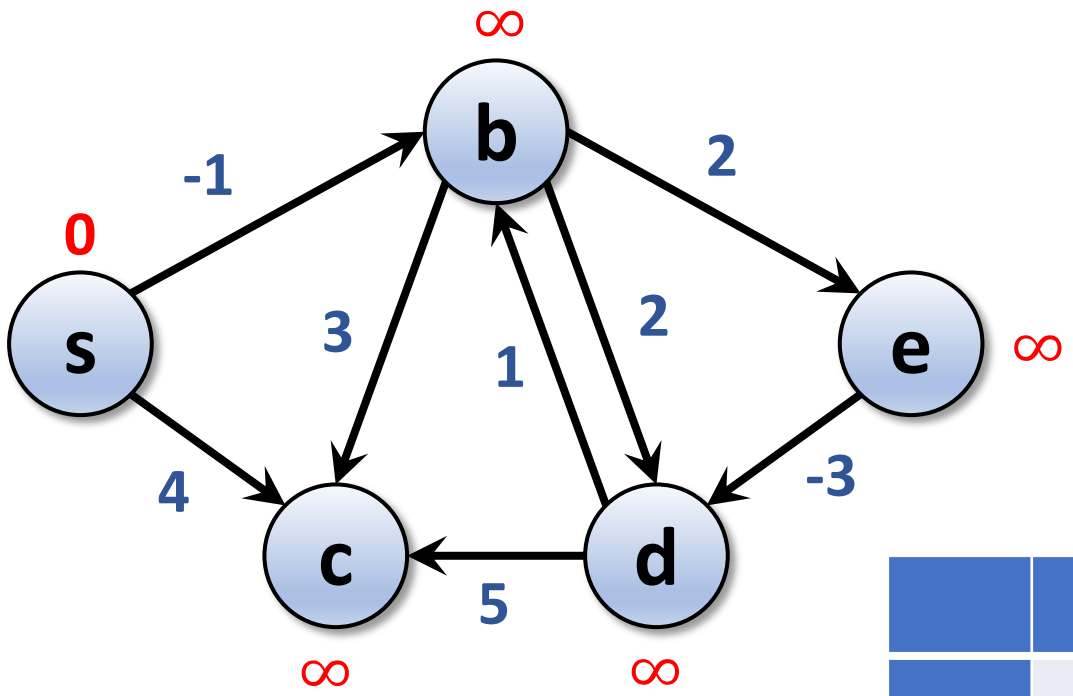
- $\mathrm{OPT}(v, j)$ is the length of the shortest $s \rightsquigarrow v$ path with at most j hops
- $P(v, j)$ is the last hop on some shortest $s \rightsquigarrow v$ path with at most j hops

Recurrence:

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

value of u that minimizes this will be parent of v if this min is smaller than

Example

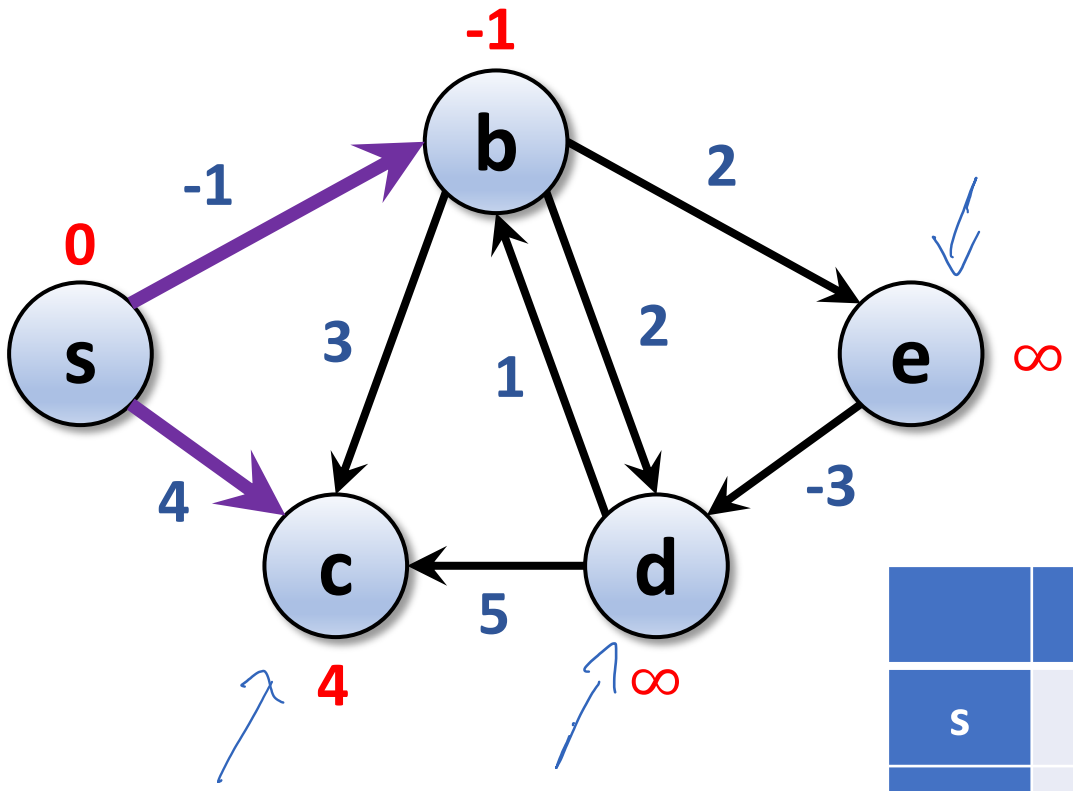


j →

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

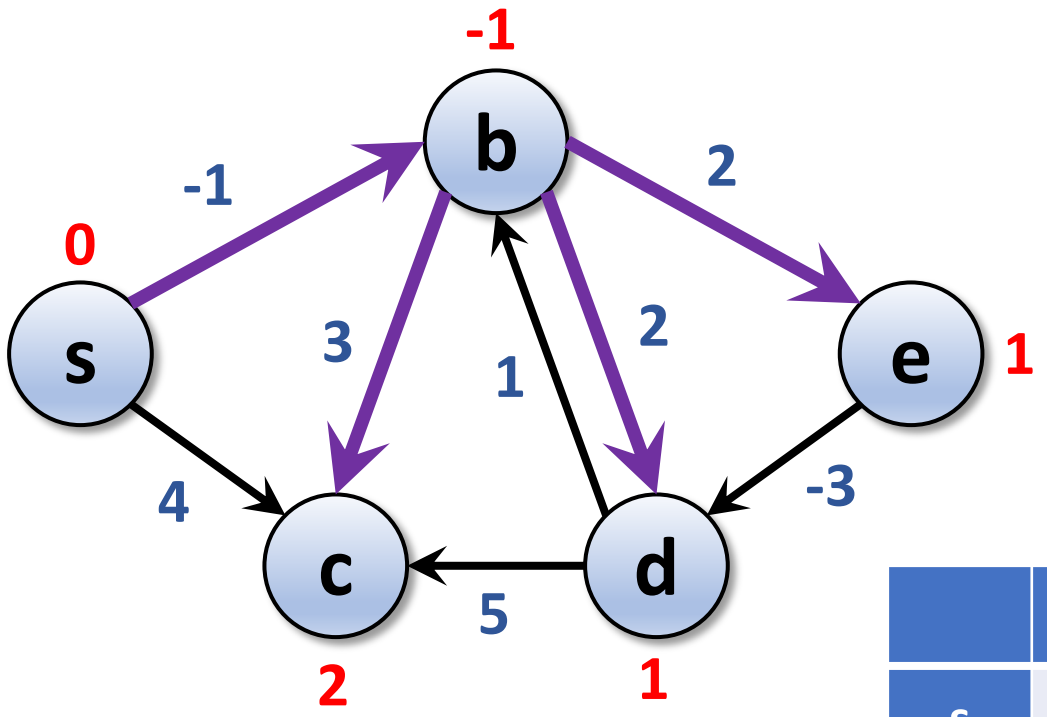
nodes

Example



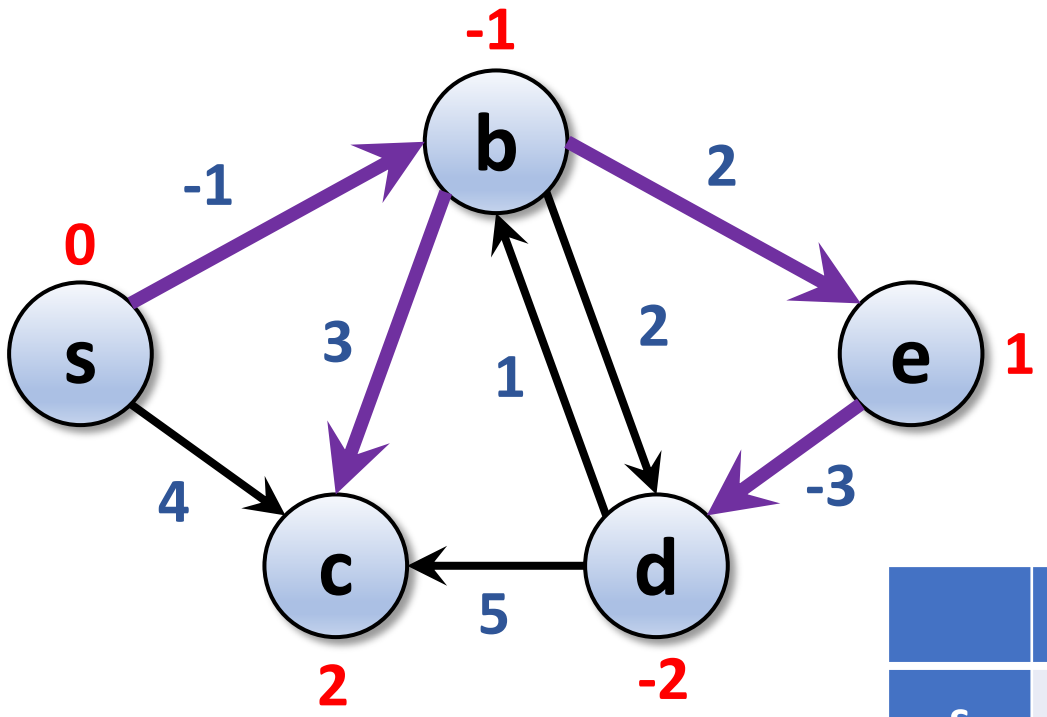
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1		
c	∞	4	2		
d	∞	∞	1		
e	∞	∞	1		

Example



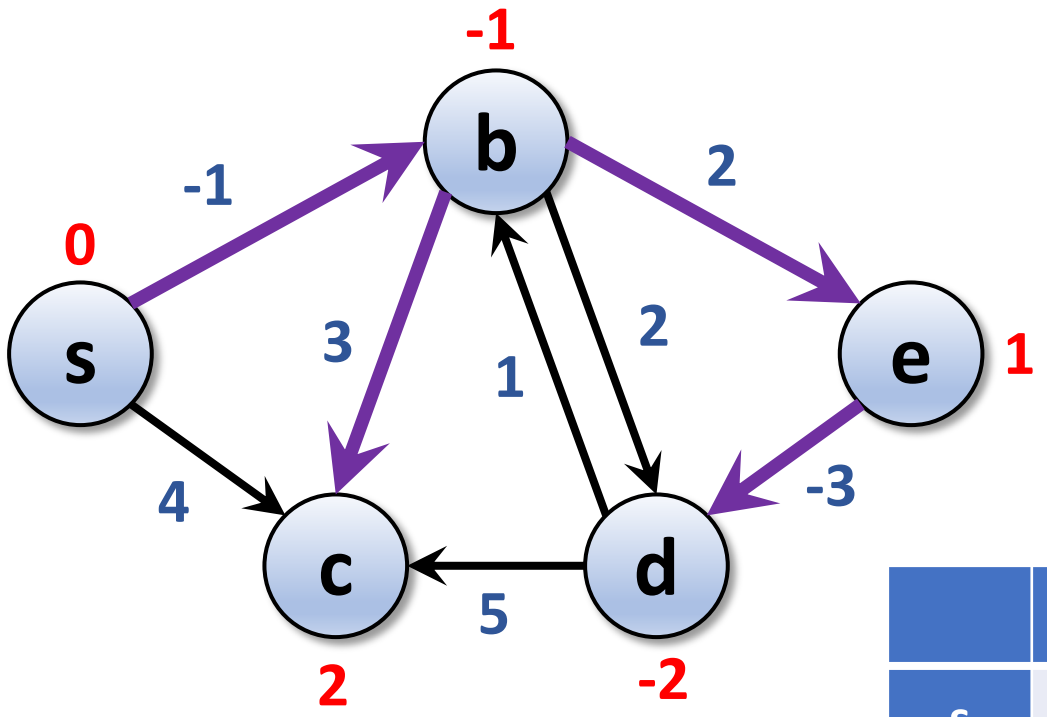
	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1	-1	
c	∞	4	2	2	
d	∞	∞	1	-2	
e	∞	∞	1	1	

Example



	0	1	2	3	4
s	0	0	0	0	0
b	∞	-1	-1	-1	
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
```

base cases

no path has more than n-1 length

O(n) iterations

O(n) iterations

copying data

indegree(V) — edge incoming to V

nodes w/ an edge into V

Running time: $O(nm)$

Space: $O(n^2)$

$\sum_{i=1}^{n-1} \sum_{v \in V} \text{indegree}(v)$

$= m$ (# edges)

$O(nm)$

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

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(mn) but O(m) in practice
Space: O(n)

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

what is cost paid for negative edges?

log n factor → n factor

if after n updates, the d continues to update, then there is a neg cycle