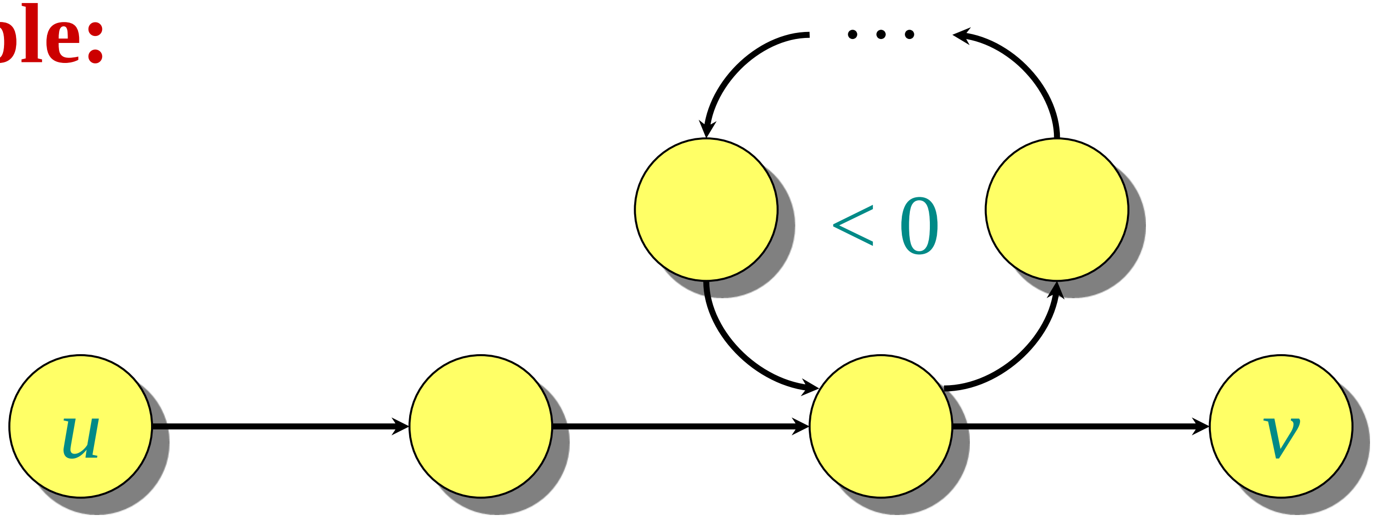
Shortest Paths II: Bellman-Ford, Topological Sort, DAG Shortest Paths, Linear Programming, Difference Constraints

Lecture 15

Negative-weight cycles

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:



Bellman-Ford algorithm: Finds all shortest-path lengths from a ***source*** $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

Bellman-Ford algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$

} initialization

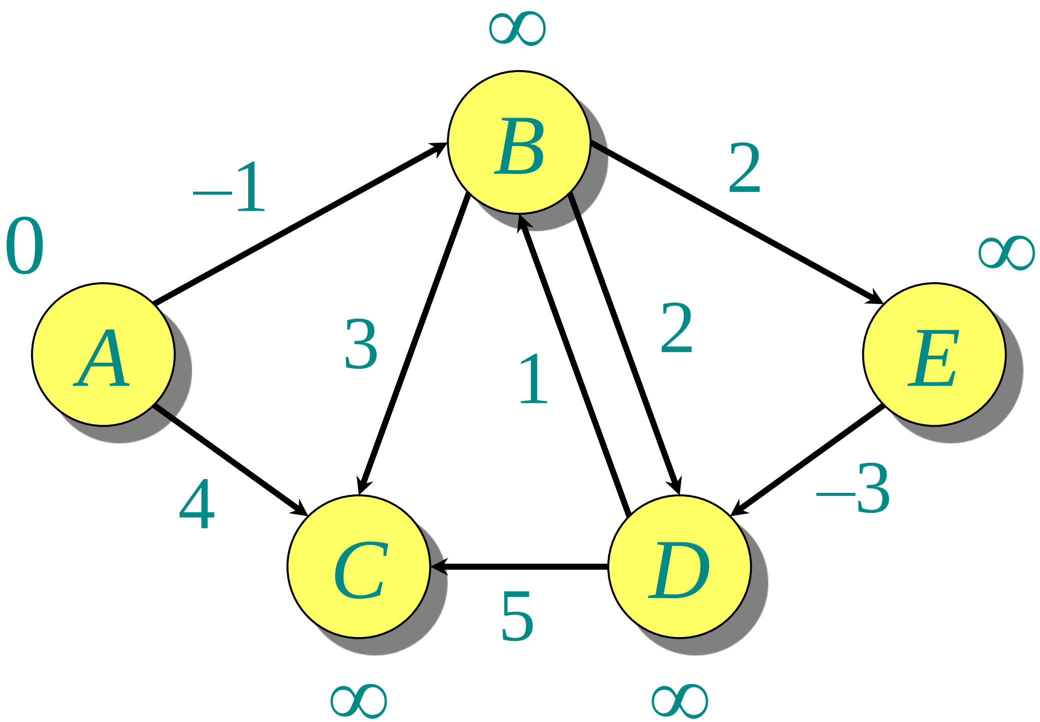
for $i \leftarrow 1$ **to** $|V| - 1$
    do for each edge $(u, v) \in E$
        do if $d[v] > d[u] + w(u, v)$
            then $d[v] \leftarrow d[u] + w(u, v)$

} ***relaxation step***

for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
        then report that a negative-weight cycle exists

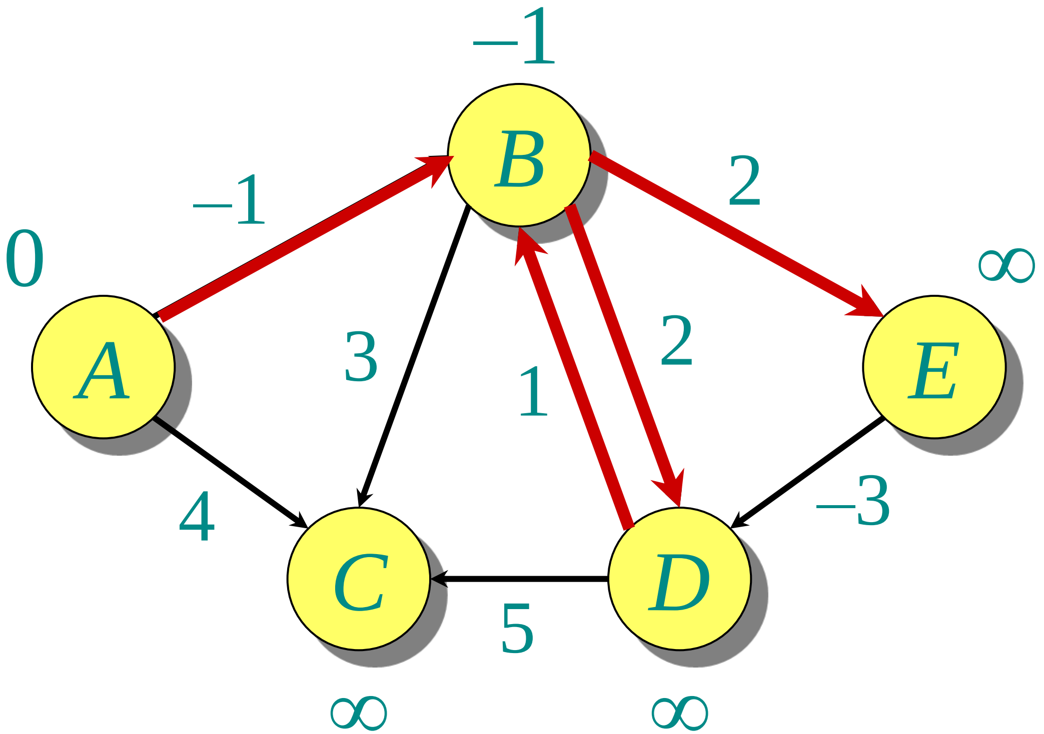
At the end, $d[v] = \delta(s, v)$. Time $= O(VE)$.

Example of Bellman-Ford



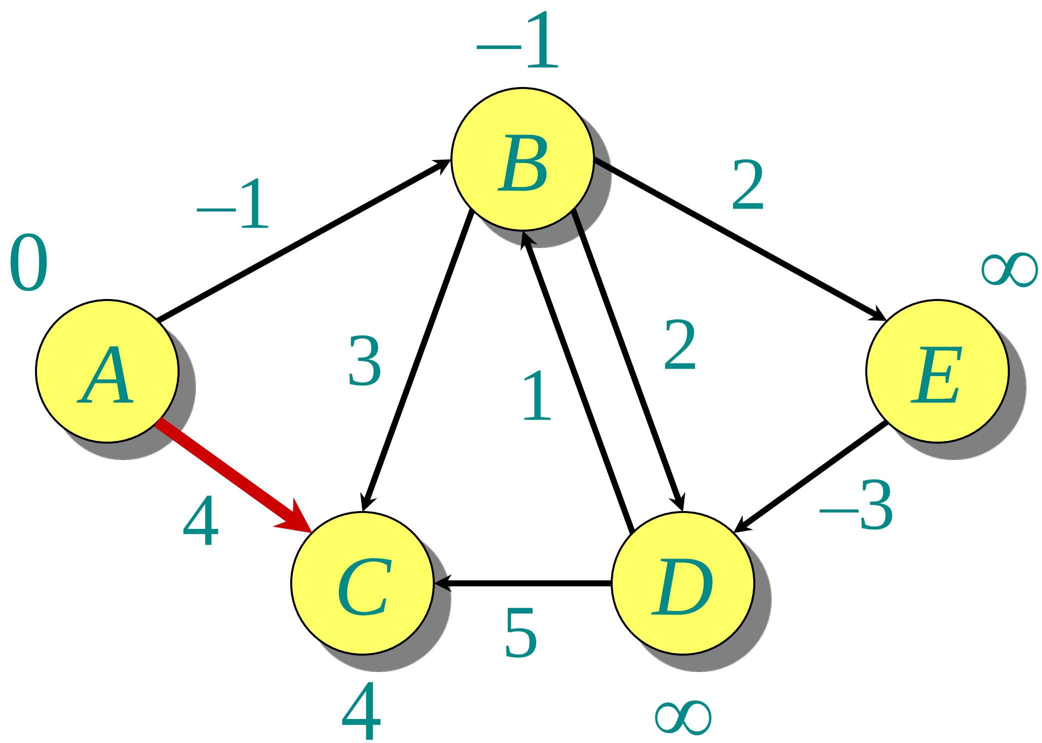
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞

Example of Bellman-Ford



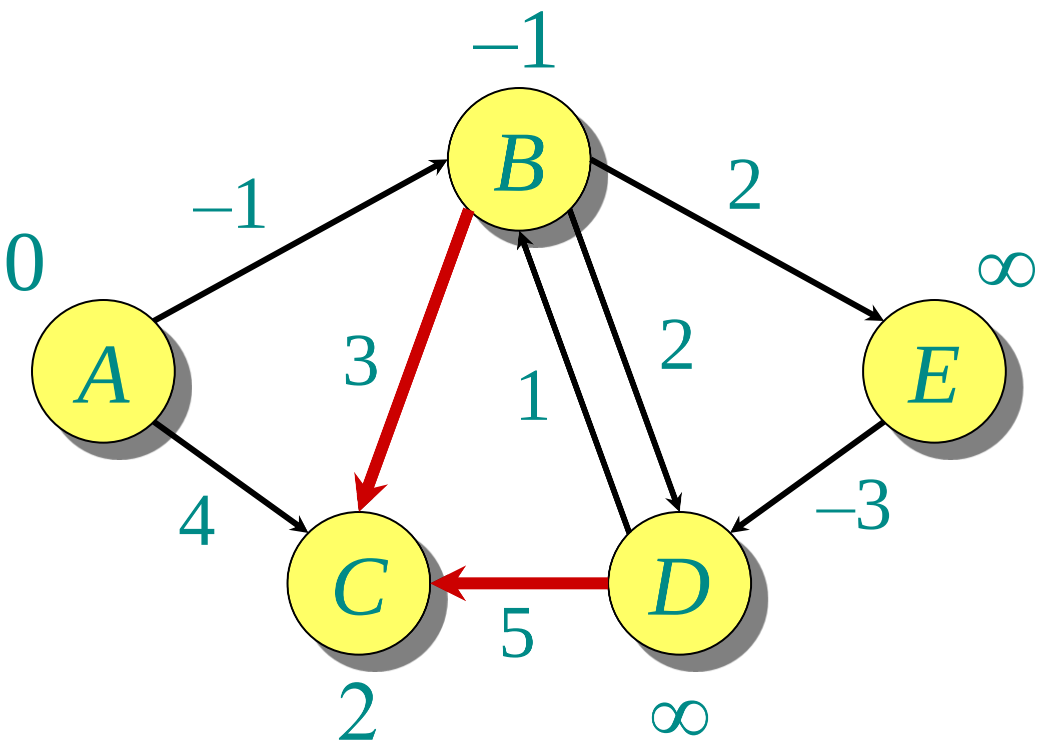
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞

Example of Bellman-Ford



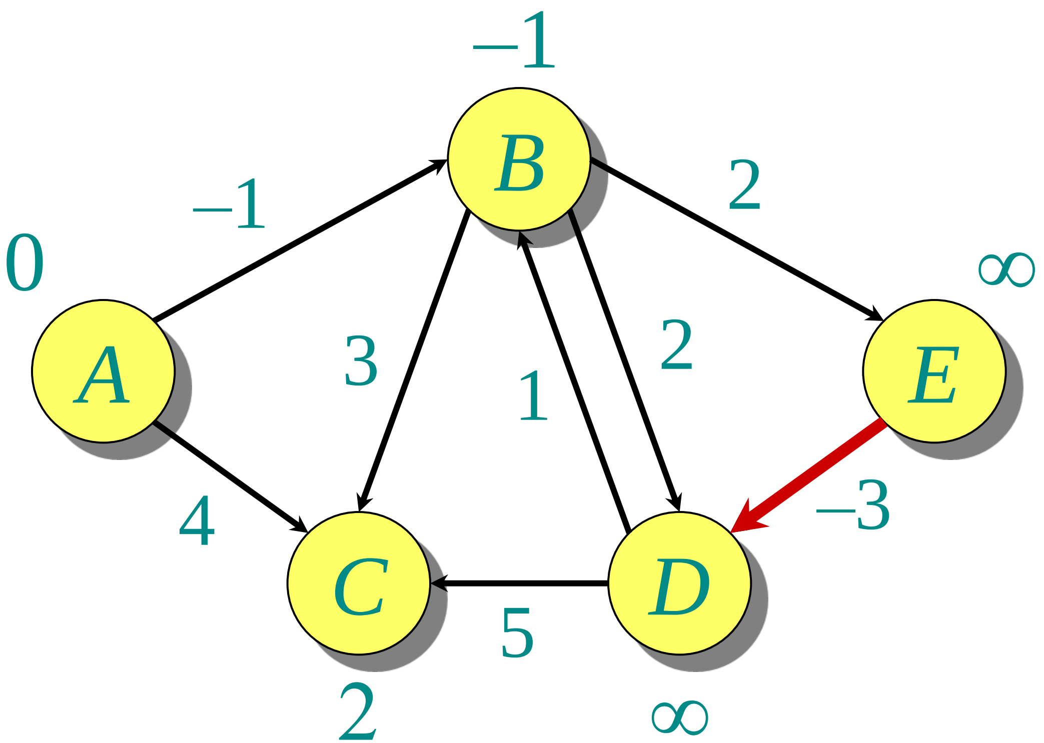
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞

Example of Bellman-Ford



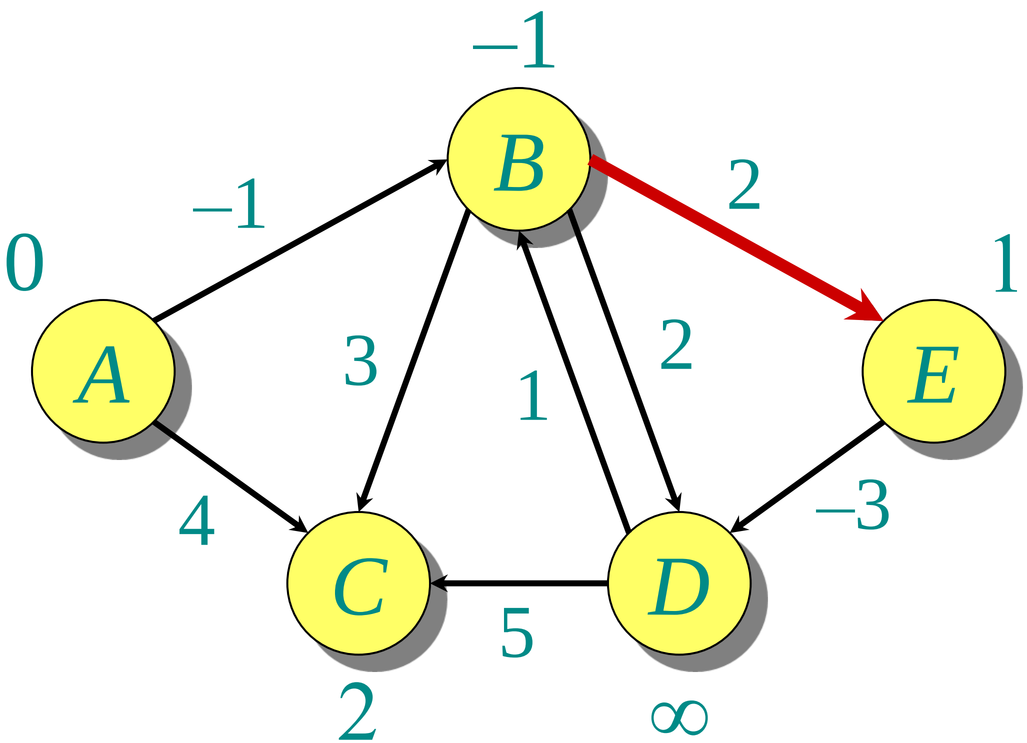
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞

Example of Bellman-Ford



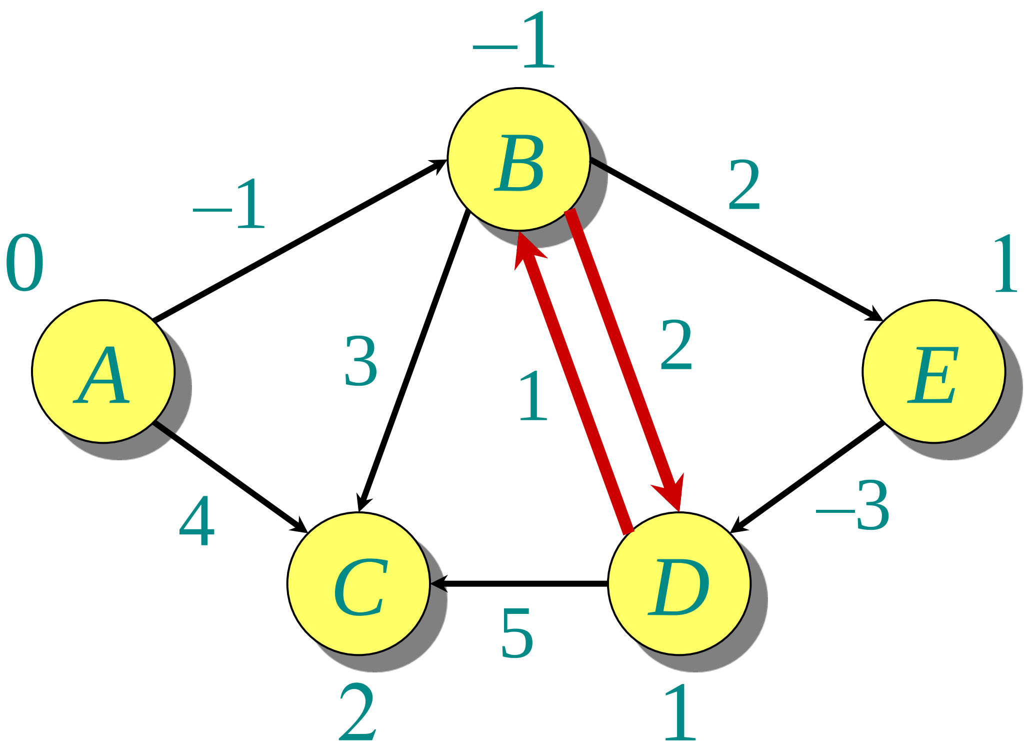
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞

Example of Bellman-Ford



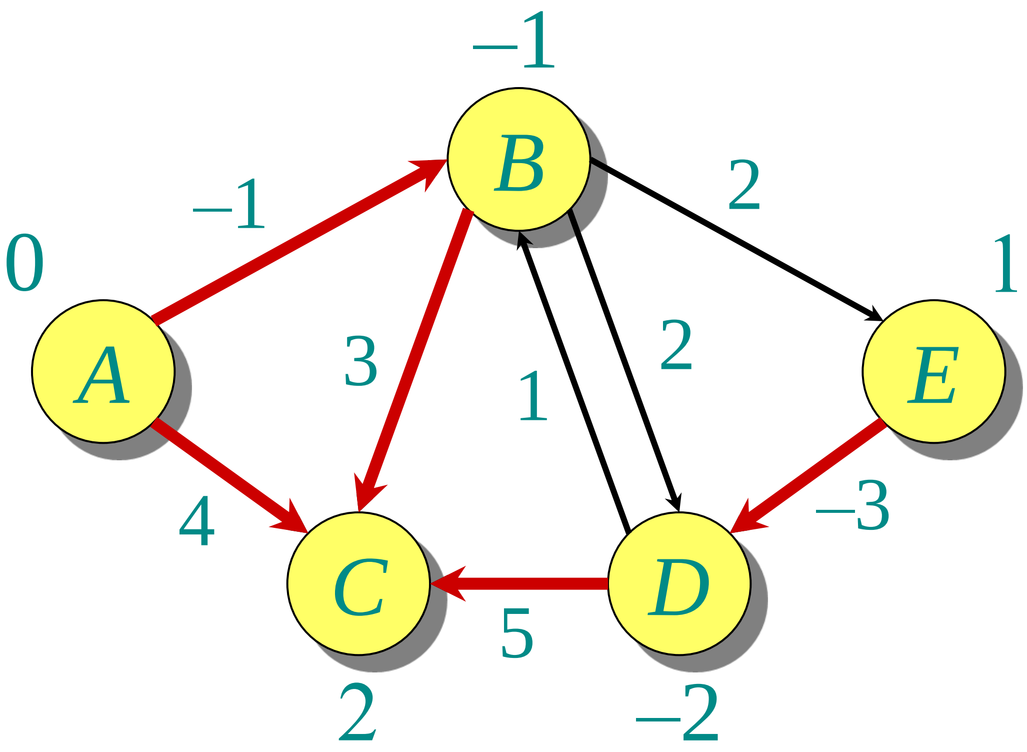
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1

Example of Bellman-Ford



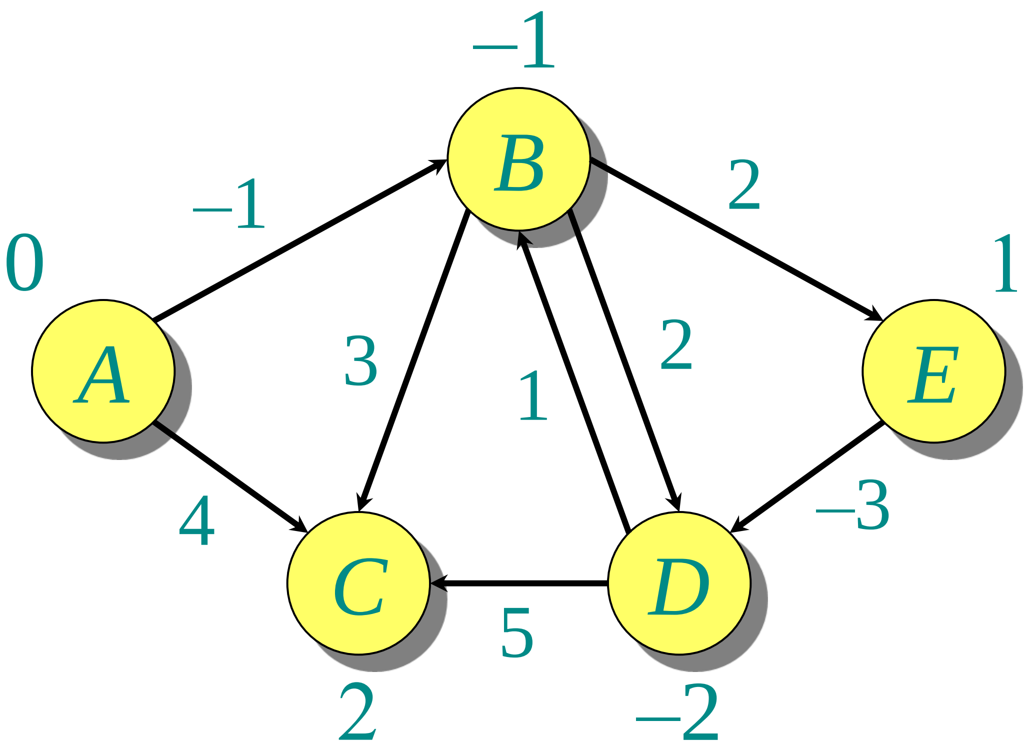
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1

Example of Bellman-Ford



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

Example of Bellman-Ford



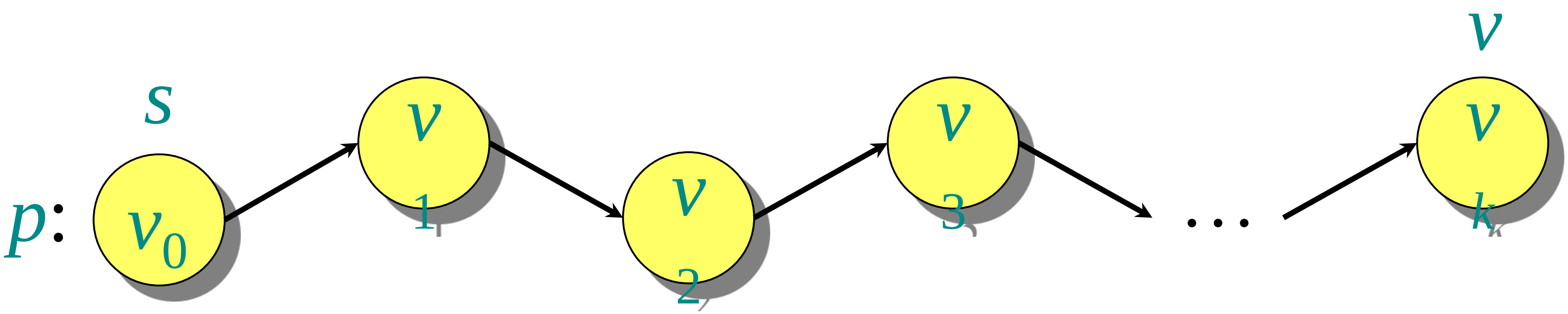
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

Note: Values decrease monotonically.

Correctness

Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.

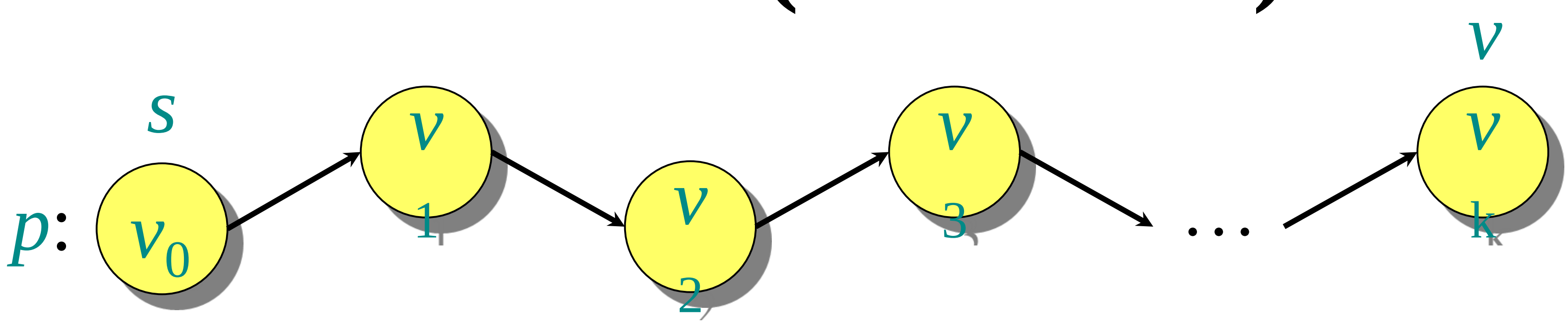
Proof. Let $v \in V$ be any vertex, and consider a shortest path p from s to v with the minimum number of edges.



Since p is a shortest path, we have

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i) .$$

Correctness (continued)



Initially, $d[v_0] = 0 = \delta(s, v_0)$, and $d[s]$ is unchanged by subsequent relaxations (because of the lemma from Lecture 17 that $d[v] \geq \delta(s, v)$).

- After 1 pass through E, we have $d[v_1] = \delta(s, v_1)$.
- After 2 passes through E, we have $d[v_2] = \delta(s, v_2)$.

  ⋮

- After k passes through E, we have $d[v_k] = \delta(s, v_k)$.

Since G contains no negative-weight cycles, p is simple. Longest simple path has $\leq |V| - 1$ edges. □

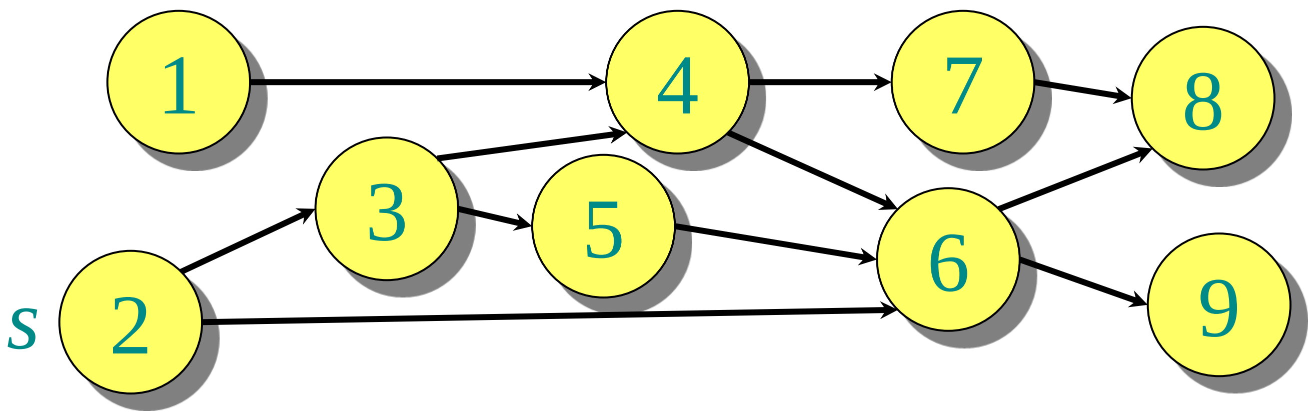
Detection of negative-weight cycles

Corollary. If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle in G reachable from s. ▫

DAG shortest paths

If the graph is a ***directed acyclic graph*** (***DAG***), we first ***topologically sort*** the vertices.

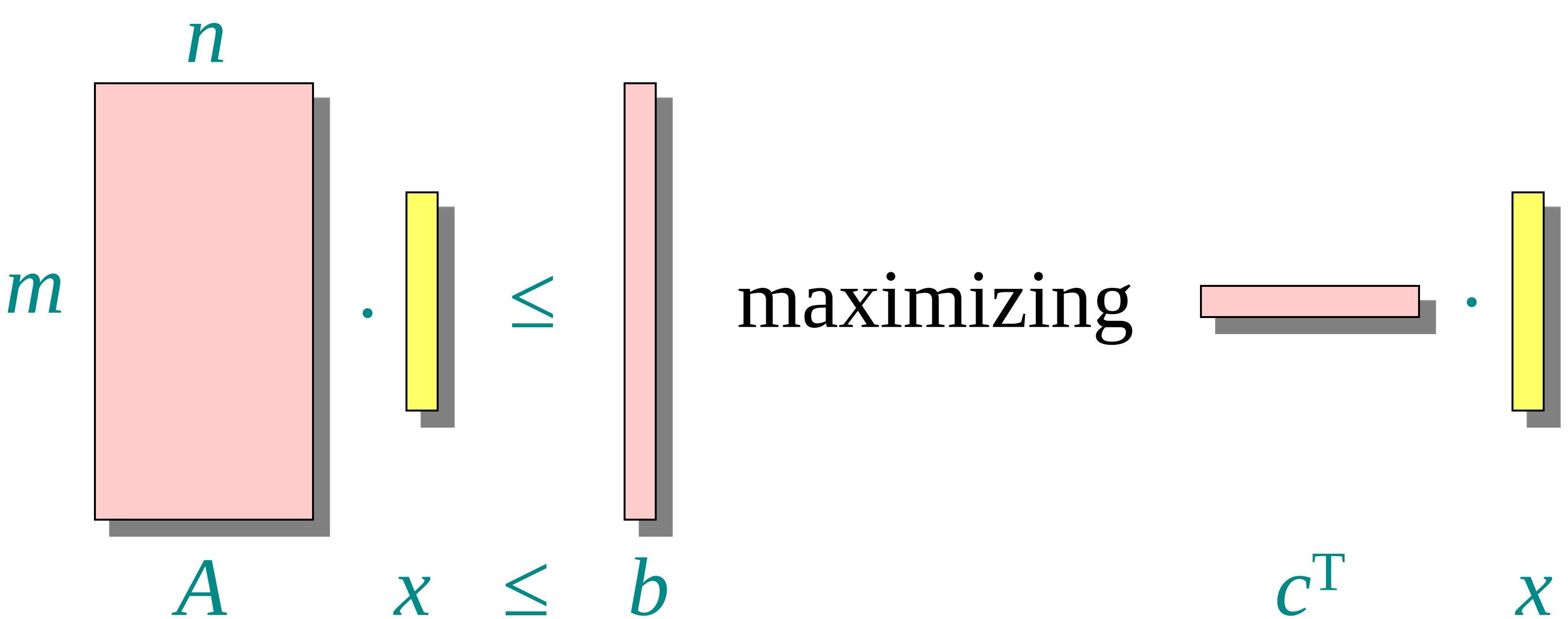
- Determine $f : V \rightarrow \{1, 2, \ldots, |V|\}$ such that $(u, v) \in E$ $\Rightarrow f(u) < f(v)$.
- $O(V + E)$ time using depth-first search.



Walk through the vertices $u \in V$ in this order, relaxing the edges in $Adj[u]$, thereby obtaining the shortest paths from s in a total of $O(V + E)$ time.

Linear programming

Let A be an $m \times n$ matrix, b be an m-vector, and c be an n-vector. Find an n-vector x that maximizes $c^T x$ subject to $Ax \leq b$, or determine that no such solution exists.



Linear-programming algorithms

Algorithms for the general problem

- Simplex methods — practical, but worst-case exponential time.
- Ellipsoid algorithm — polynomial time, but slow in practice.
- Interior-point methods — polynomial time and competes with simplex.

Feasibility problem: No optimization criterion. Just find x such that $Ax \leq b$.

- In general, just as hard as ordinary LP.

Solving a system of difference constraints

Linear programming where each row of A contains exactly one 1, one –1, and the rest 0's.

Example:

$$\left.\begin{aligned} x_1 - x_2 &\le 3 \\ x_2 - x_3 &\le -2 \\ x_1 - x_3 &\le 2 \end{aligned}\right\} \quad x_j - x_i \le w_{ij}$$

Solution:

$$x_1 = 3$$
$$x_2 = 0$$
$$x_3 = 2$$

Constraint graph:

$x_j - x_i \le w_{ij}$ ⇒

(The "A" matrix has dimensions $|E| \times |V|$.)

Unsatisfiable constraints

Theorem. If the constraint graph contains a negative-weight cycle, then the system of differences is unsatisfiable.

Proof. Suppose that the negative-weight cycle is $v_1 \to v_2 \to \cdots \to v_k \to v_1$. Then, we have

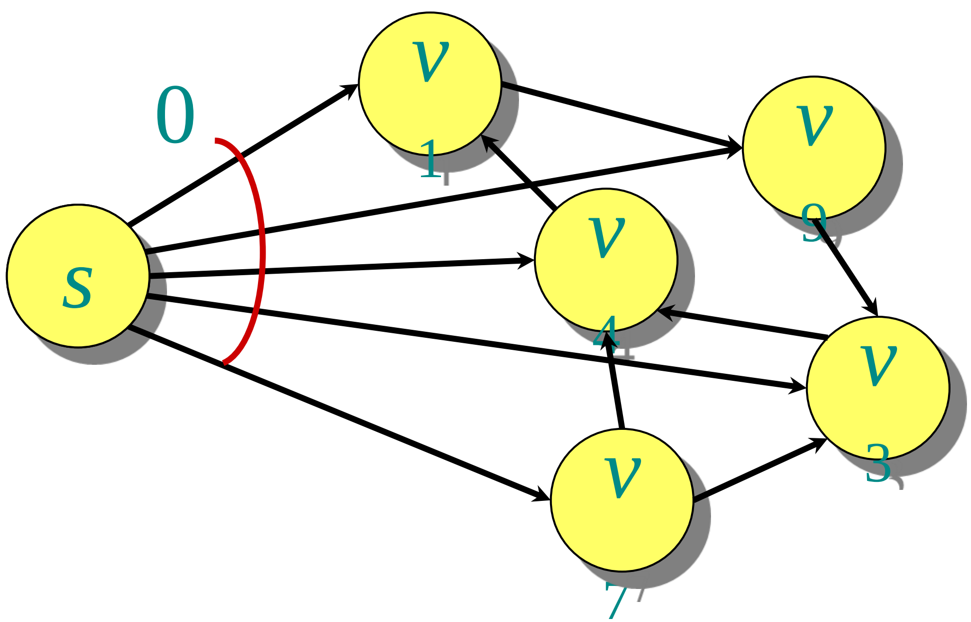
$$\begin{aligned}
x_2 - x_1 &\le w_{12} \\
x_3 - x_2 &\le w_{23} \\
&\vdots \\
x_k - x_{k-1} &\le w_{k-1,\,k} \\
x_1 - x_k &\le w_{k1} \\
\hline
0 &\le \text{weight of cycle} \\
&< 0
\end{aligned}$$

Therefore, no values for the x_i can satisfy the constraints. □

Satisfying the constraints

Theorem. Suppose no negative-weight cycle exists in the constraint graph. Then, the constraints are satisfiable.

Proof. Add a new vertex s to V with a 0-weight edge to each vertex $v_i \in V$.

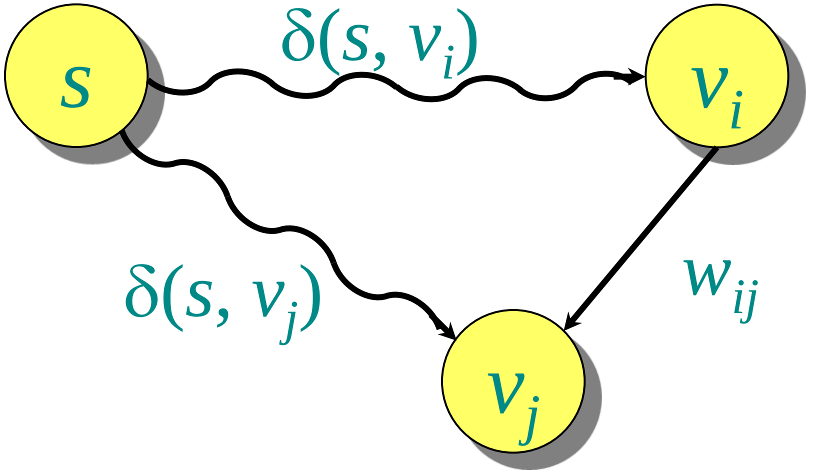


Note:
No negative-weight cycles introduced ⇒ shortest paths exist.

Proof (continued)

Claim: The assignment $x_i = \delta(s, v_i)$ solves the constraints.

Consider any constraint $x_j - x_i \leq w_{ij}$, and consider the shortest paths from s to v_j and v_i:



The triangle inequality gives us $\delta(s,v_j) \leq \delta(s, v_i) + w_{ij}$. Since $x_i = \delta(s, v_i)$ and $x_j = \delta(s, v_j)$, the constraint $x_j - x_i \leq w_{ij}$ is satisfied. ◻

Bellman-Ford and linear programming

Corollary. The Bellman-Ford algorithm can solve a system of m difference constraints on n variables in $O(mn)$ time. ☐

Single-source shortest paths is a simple LP problem.

In fact, Bellman-Ford maximizes $x_1 + x_2 + \cdots + x_n$ subject to the constraints $x_j - x_i \le w_{ij}$ and $x_i \le 0$ (exercise).

Bellman-Ford also minimizes $\max_i\{x_i\} - \min_i\{x_i\}$ (exercise).