CMP448: Algorithms

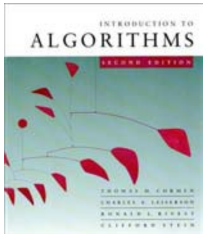
Lecture 14: Bellman-Ford Algorithm

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2013

Agenda

- Bellman-Ford Algorithm
- DAG Shortest Paths
- Linear Programming and Difference Constraints
- VLSI Layout Compaction

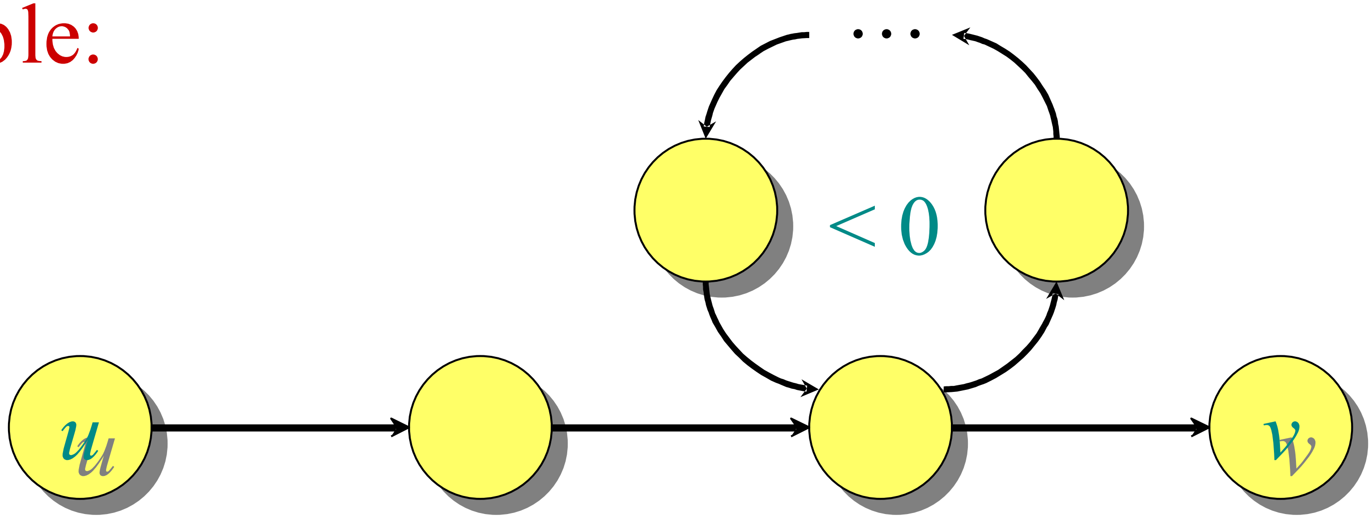
Acknowledgment
A lot of slides adapted from the slides of Erik Demaine and Charles Leiserson.

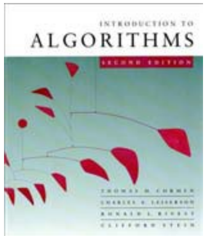


Negative-weight cycles

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

Example:

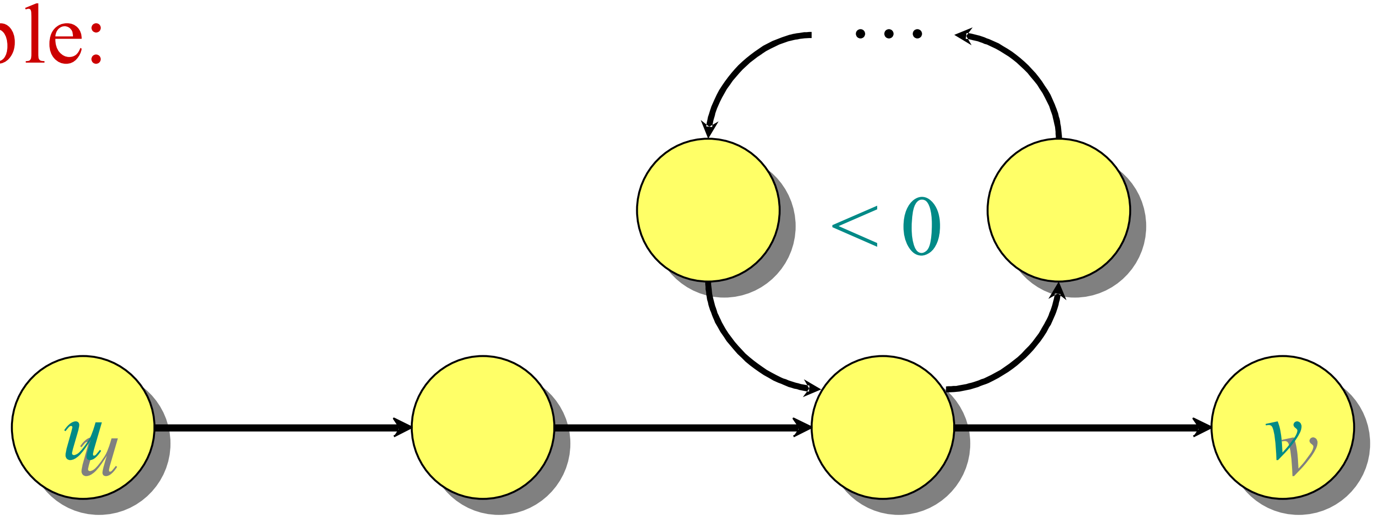




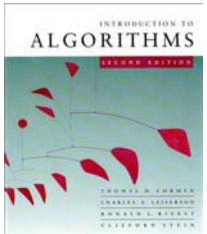
Negative-weight cycles

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

Example:

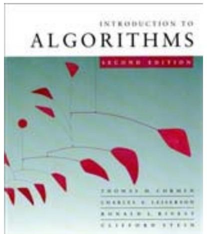


Bellman-Ford algorithm: Finds all shortest-path lengths from a *source* $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.



Bellman-Ford algorithm

$$\left.\begin{array}{l} d[s] \leftarrow 0 \\ \text{for each } v \in V - \{s\} \\ \quad \text{do } d[v] \leftarrow \infty \end{array}\right\} \text{ initialization}$$



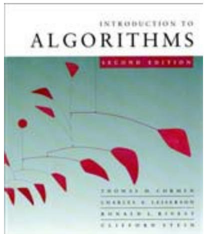
Bellman-Ford algorithm

$d[s] \leftarrow 0$
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



Bellman-Ford algorithm

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

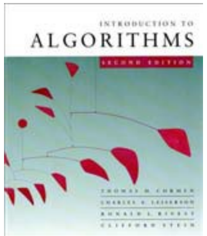
initialization

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 $d[v] \leftarrow d[u] + w(u, v)$

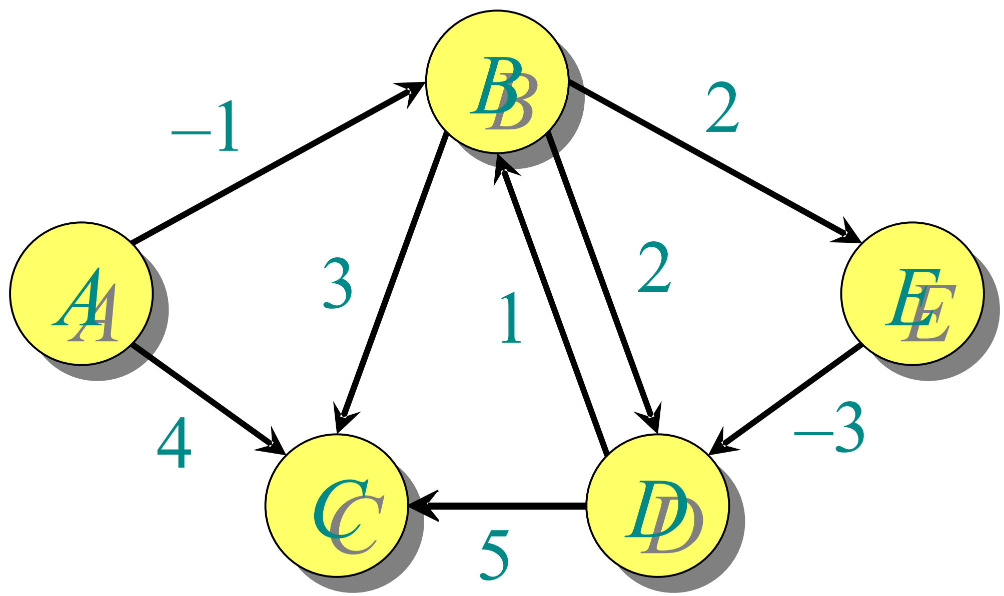
relaxation step

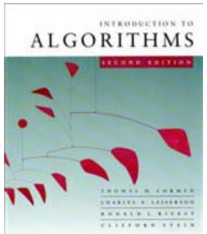
for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
        then report that a negative-weight cycle exists

At the end, $d[v] = \delta(s, v)$, if no negative-weight cycles.
Time $= O(VE)$.

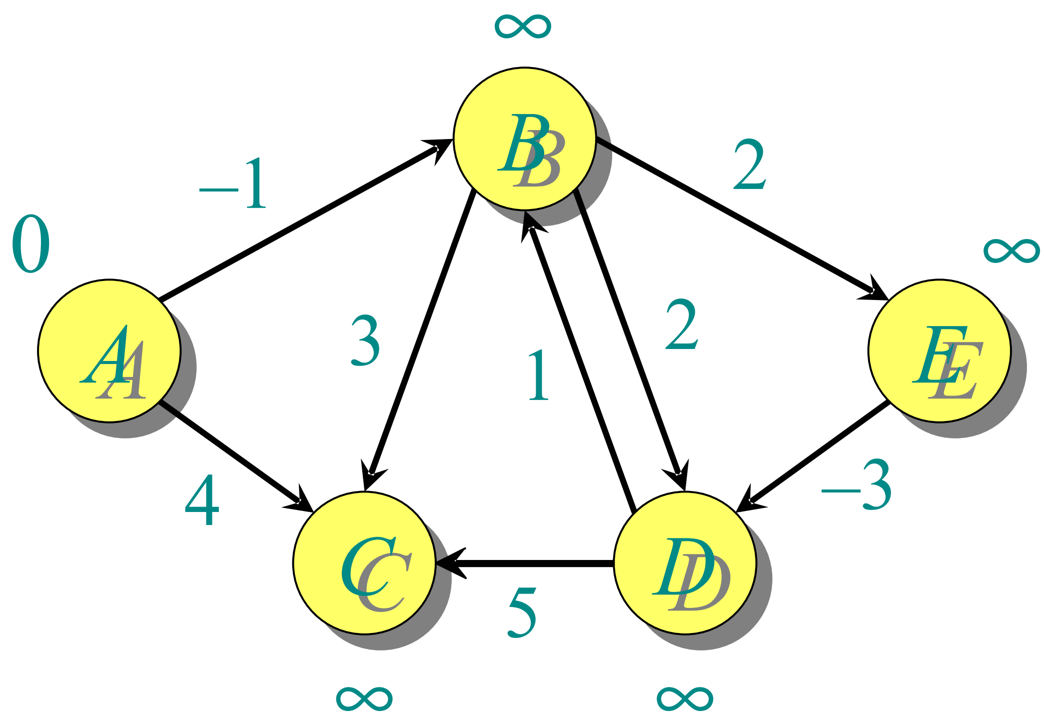


Example of Bellman-Ford

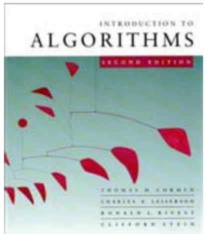




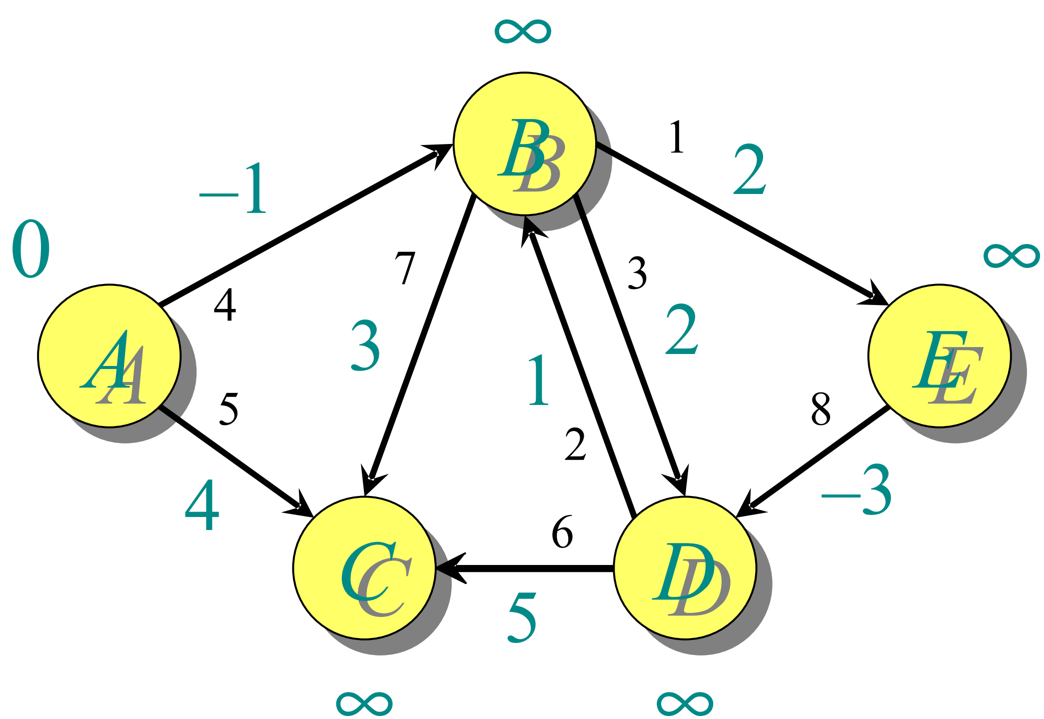
Example of Bellman-Ford



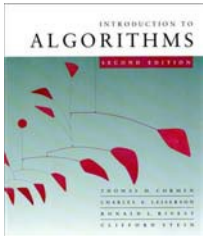
Initialization.



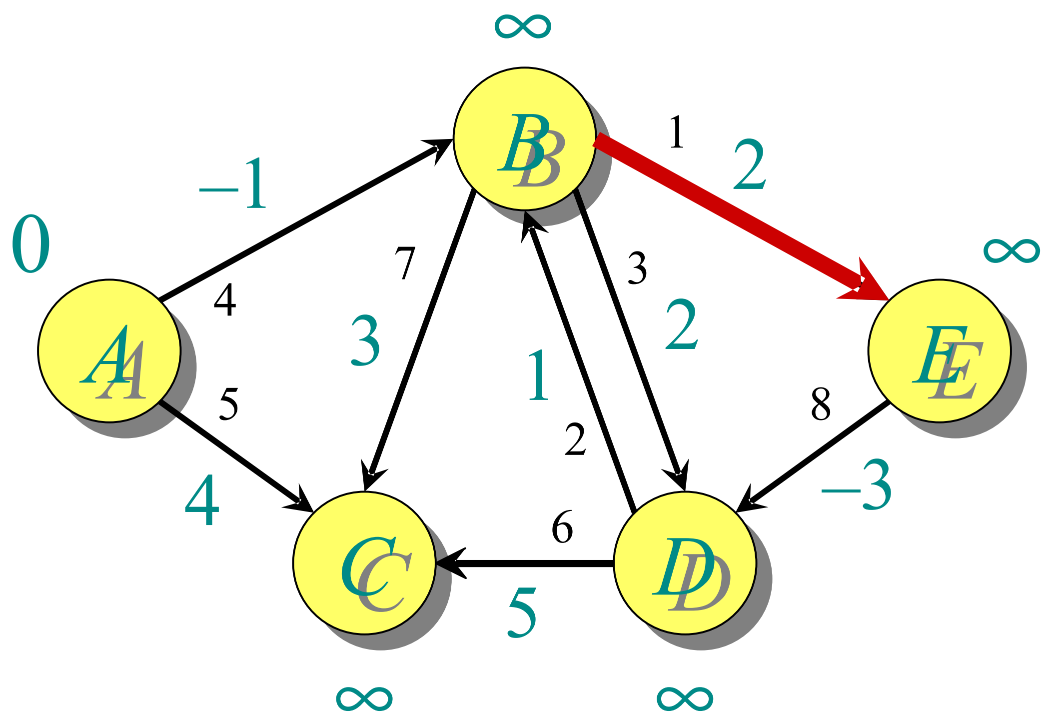
Example of Bellman-Ford

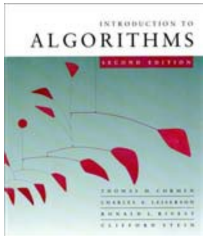


Order of edge relaxation.

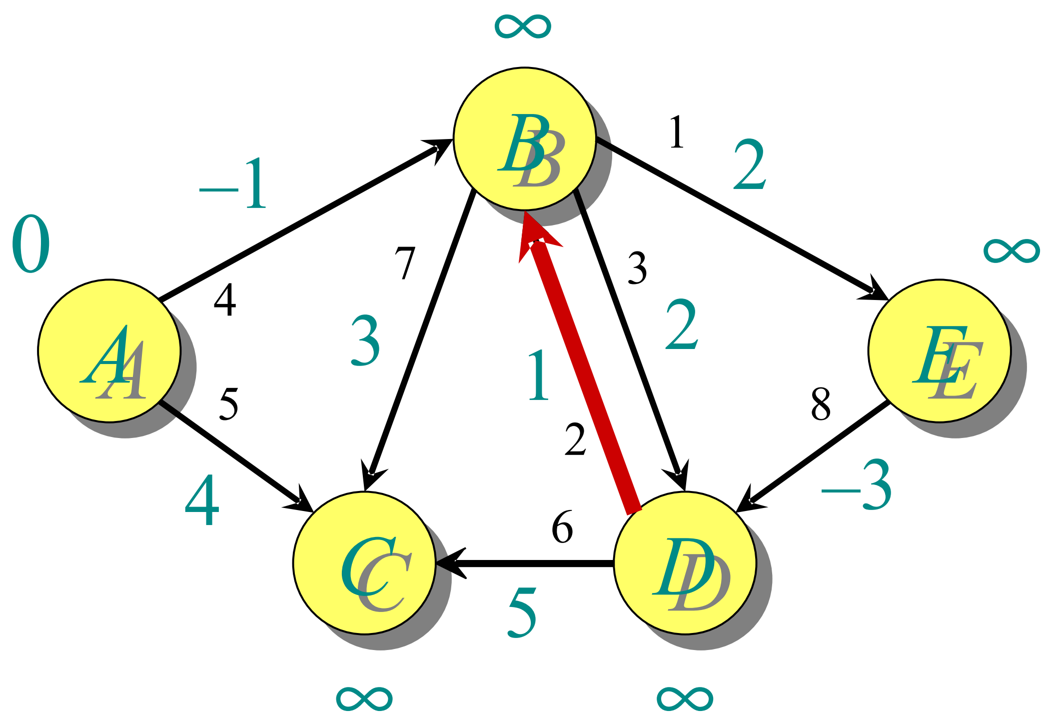


Example of Bellman-Ford

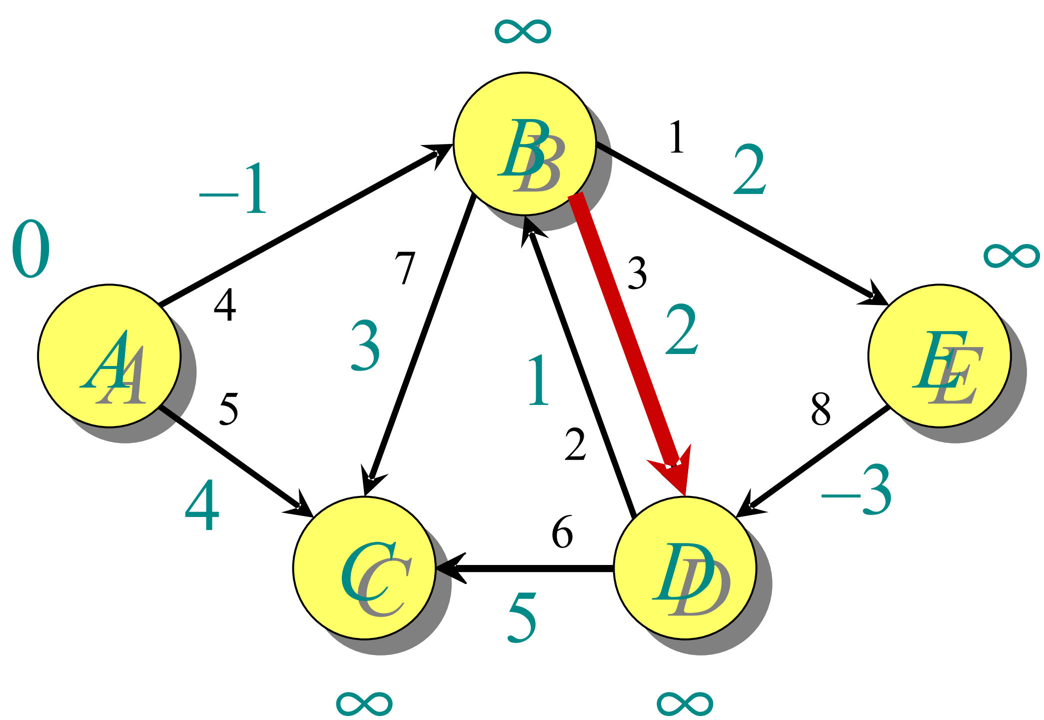




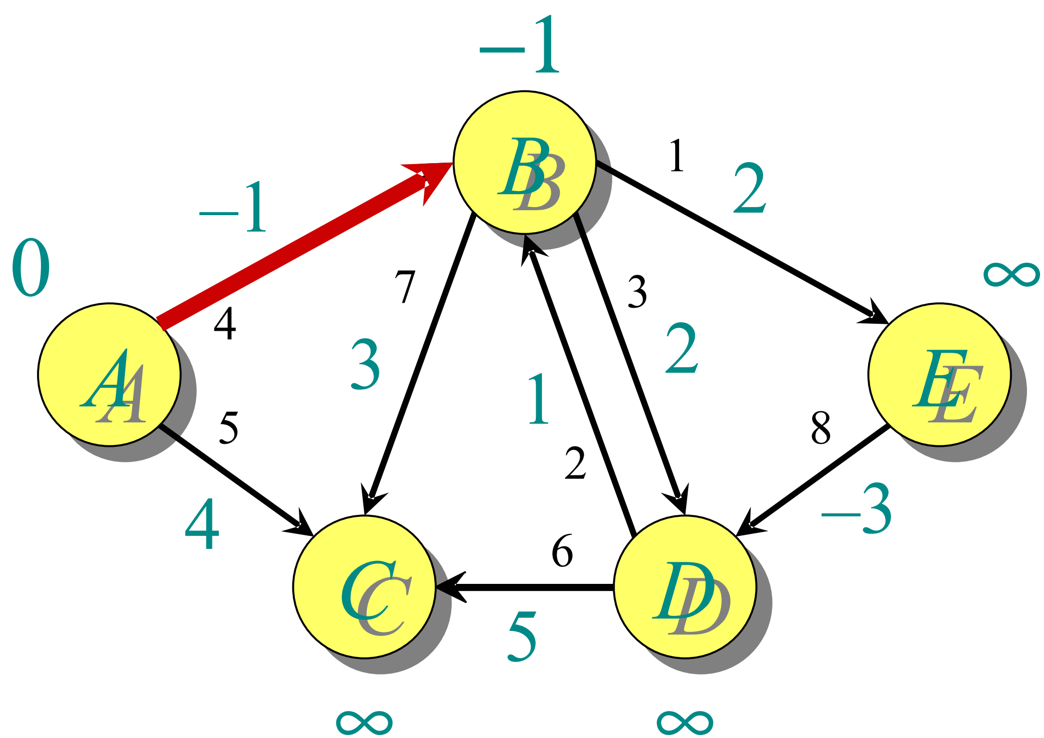
Example of Bellman-Ford

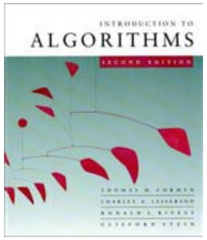


Example of Bellman-Ford

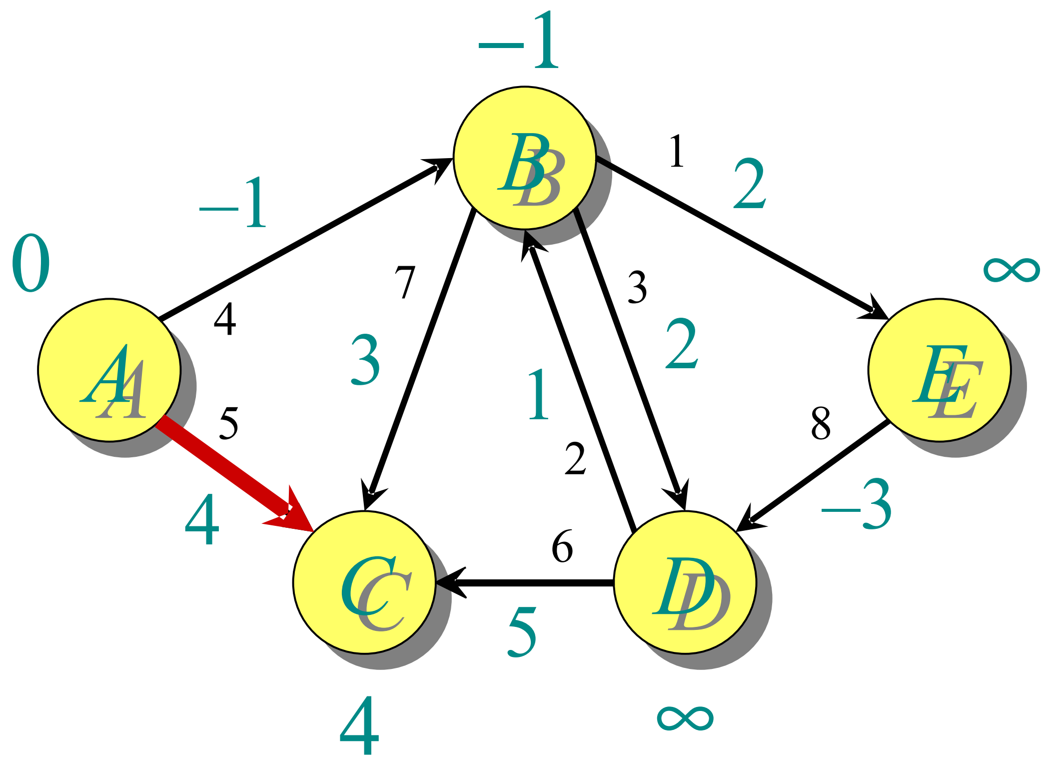


Example of Bellman-Ford

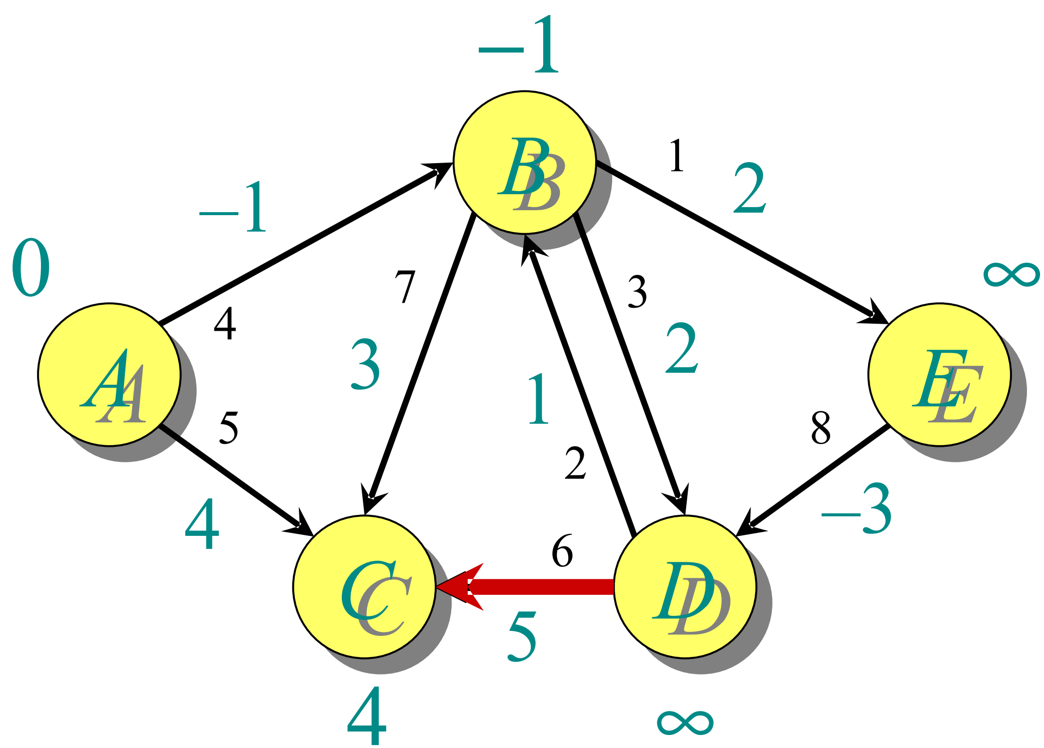




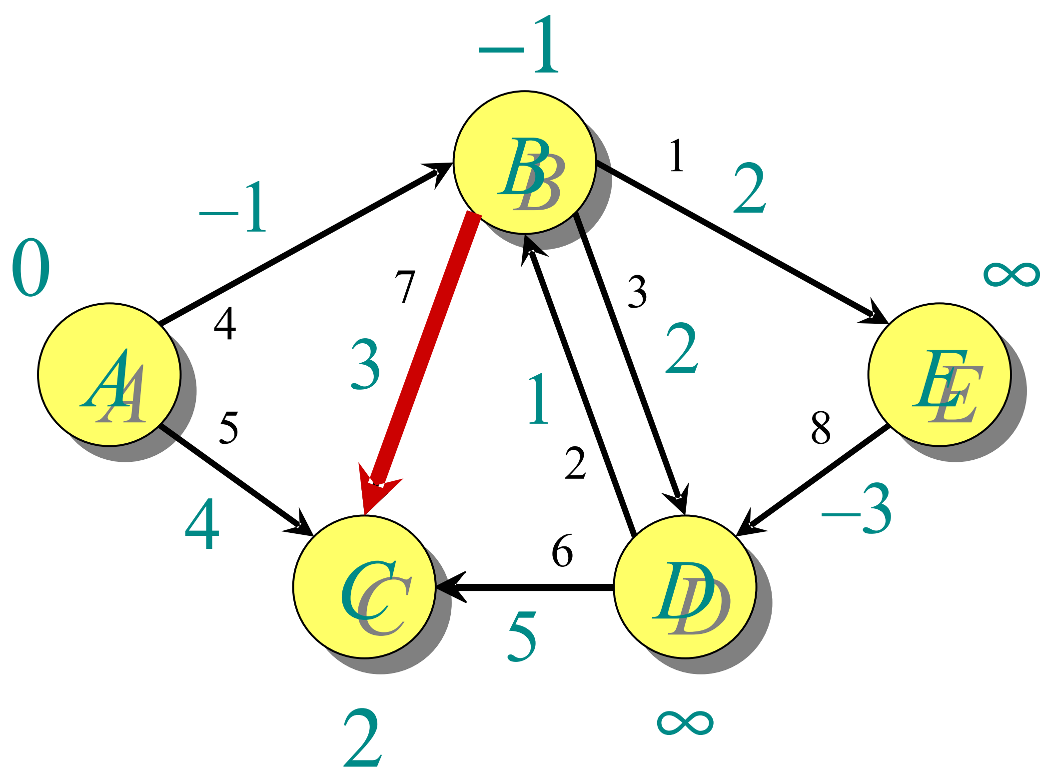
Example of Bellman-Ford

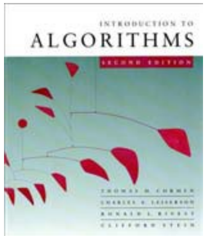


Example of Bellman-Ford

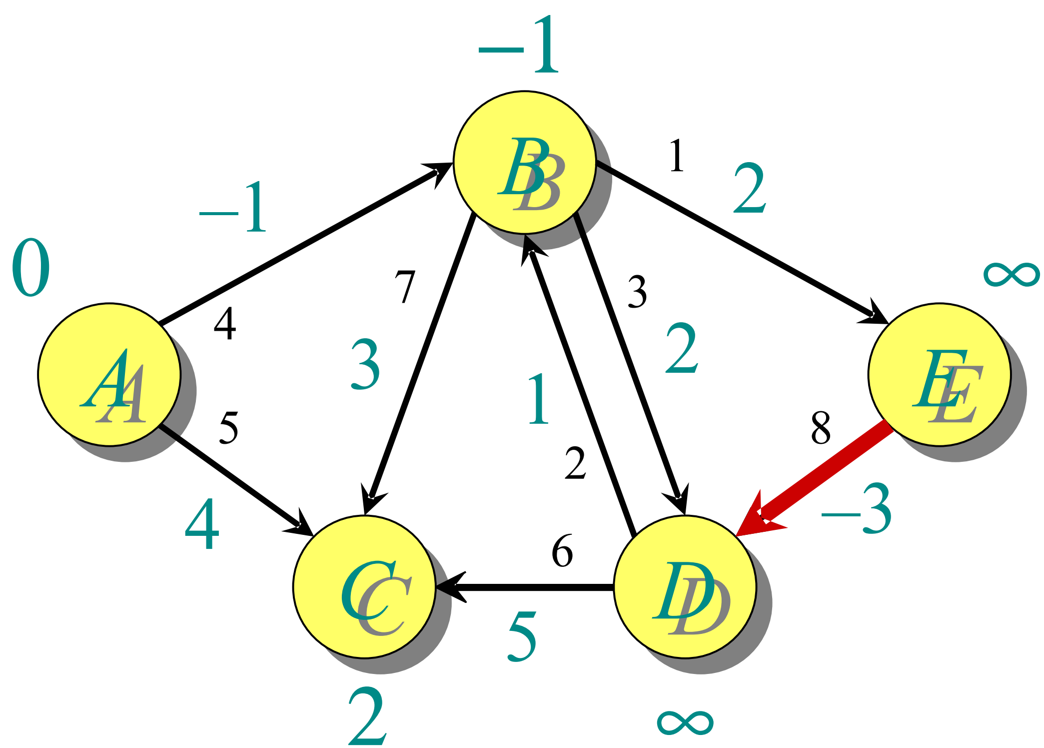


Example of Bellman-Ford

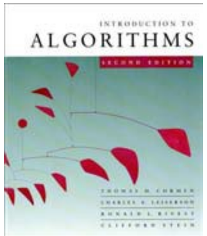




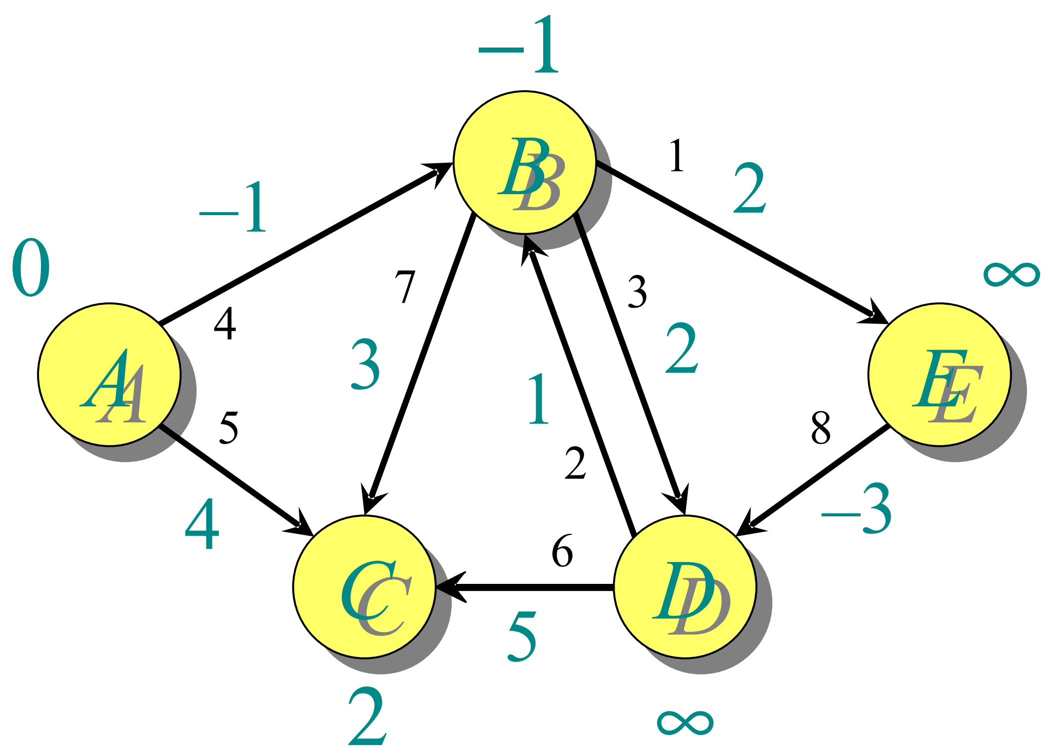
Example of Bellman-Ford



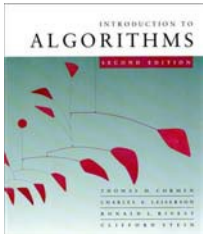
 Introduction to Algorithms November 3, 2004 L15.15



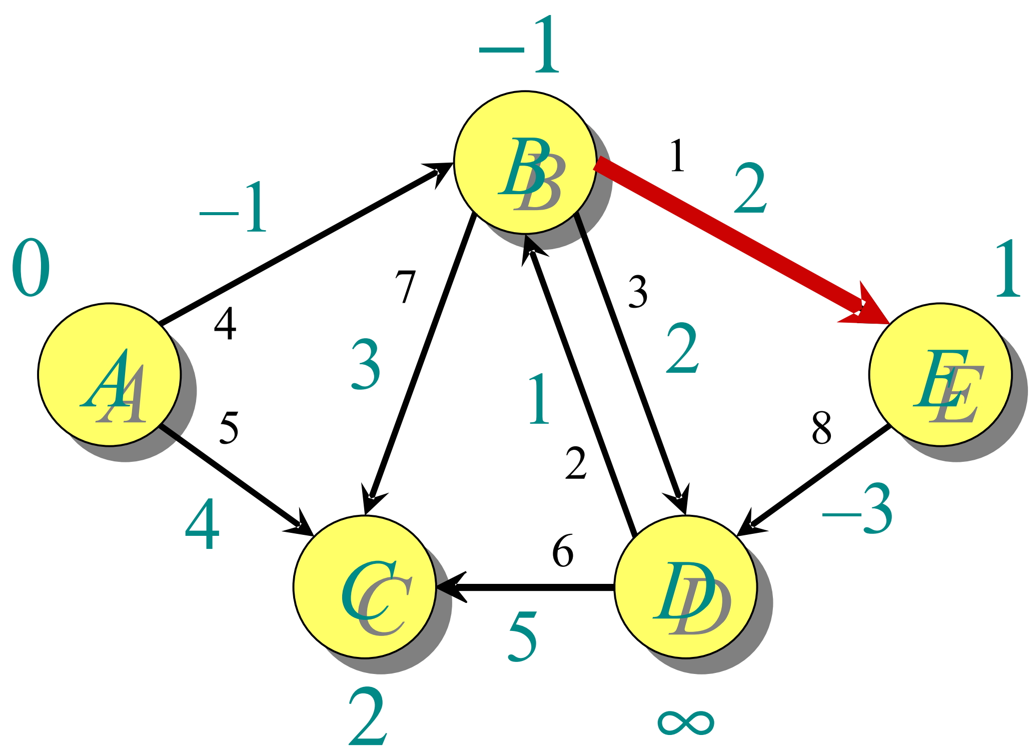
Example of Bellman-Ford

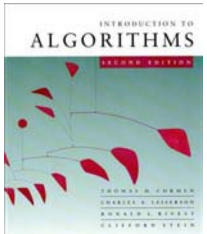


End of pass 1.

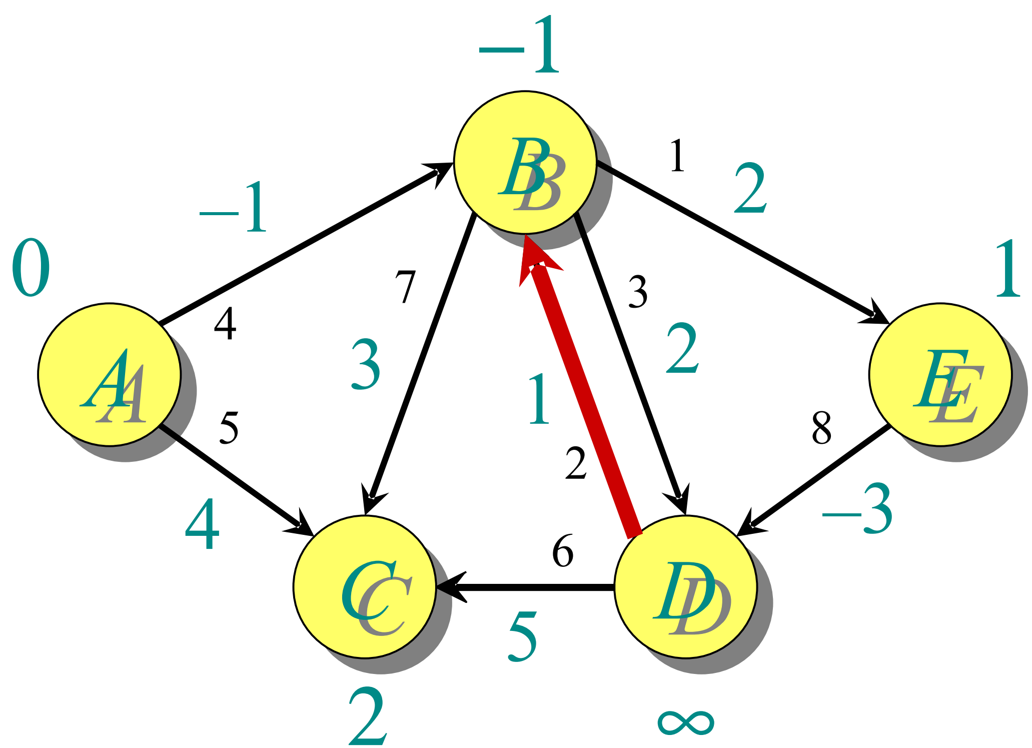


Example of Bellman-Ford

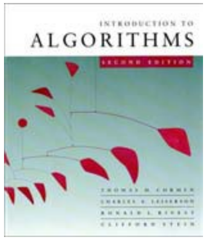




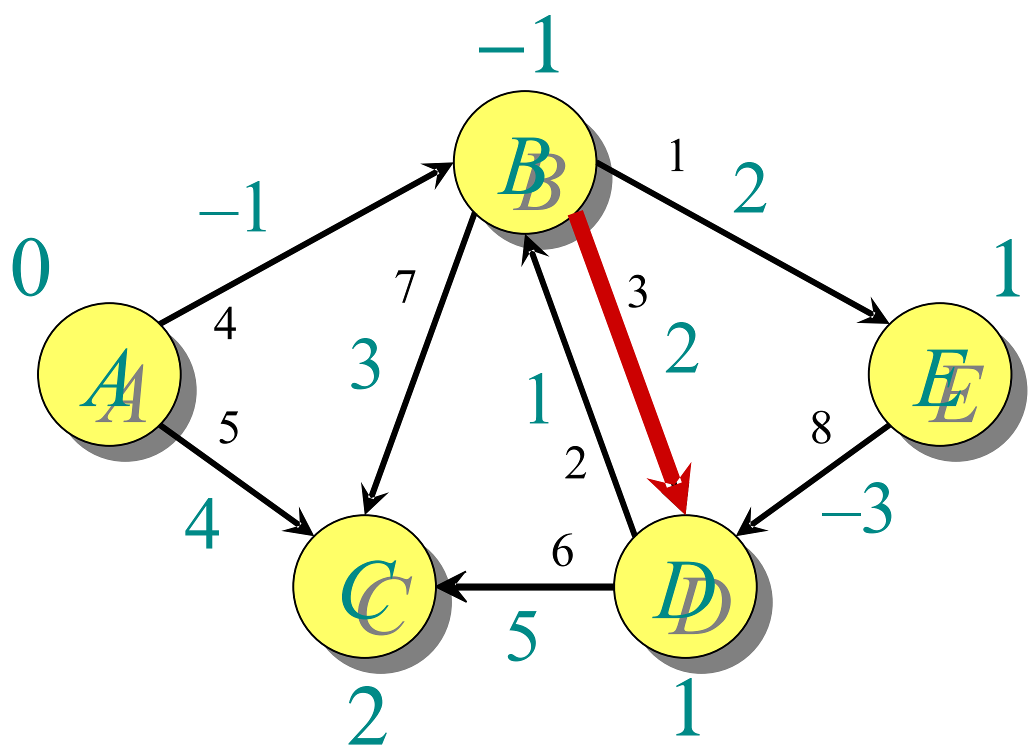
Example of Bellman-Ford



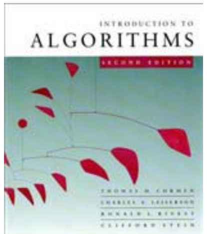
 Introduction to Algorithms November 3, 2004 L15.18



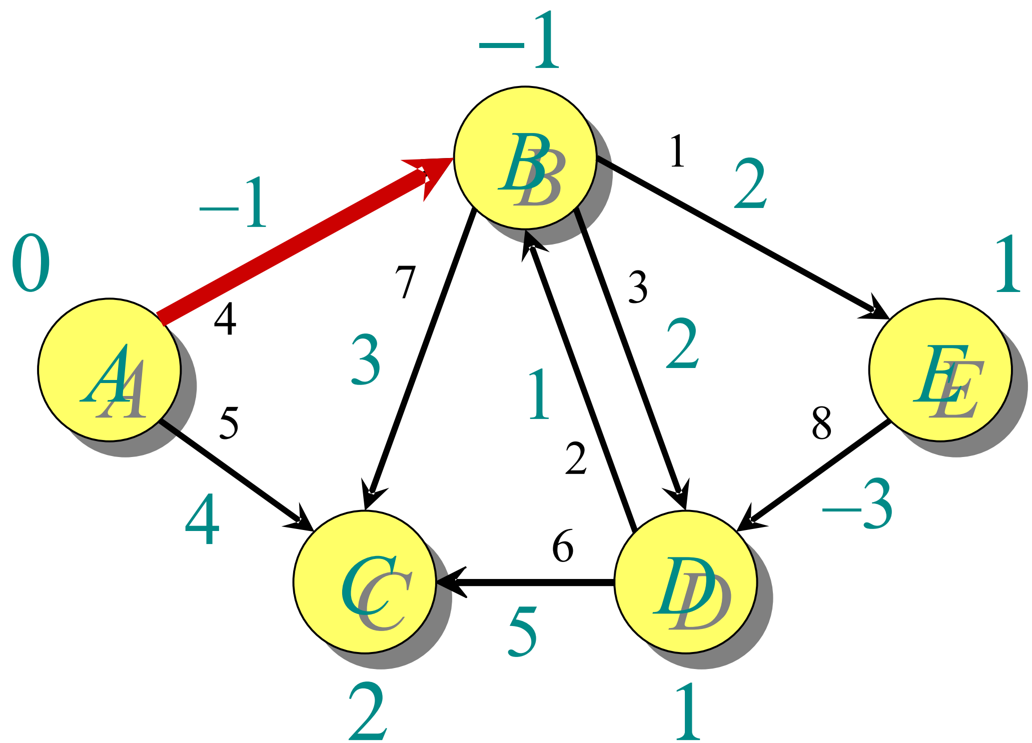
Example of Bellman-Ford

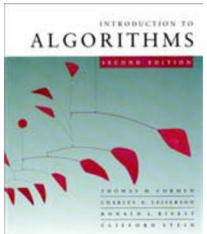


 Introduction to Algorithms November 3, 2004 L15.19

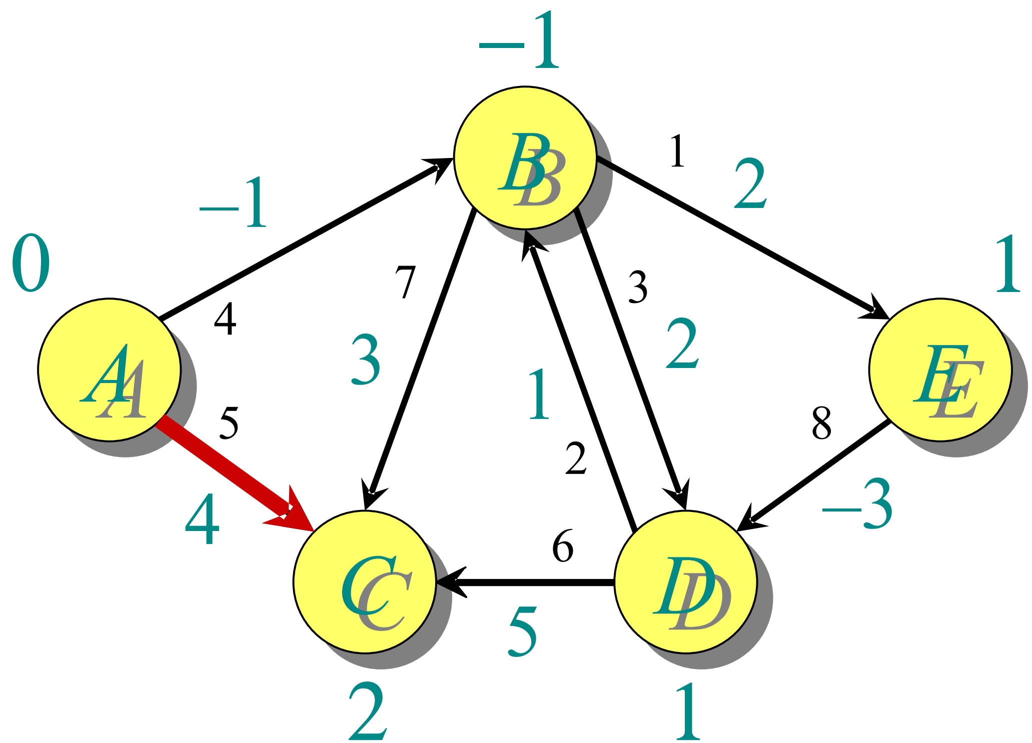


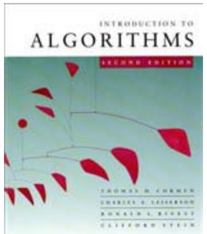
Example of Bellman-Ford



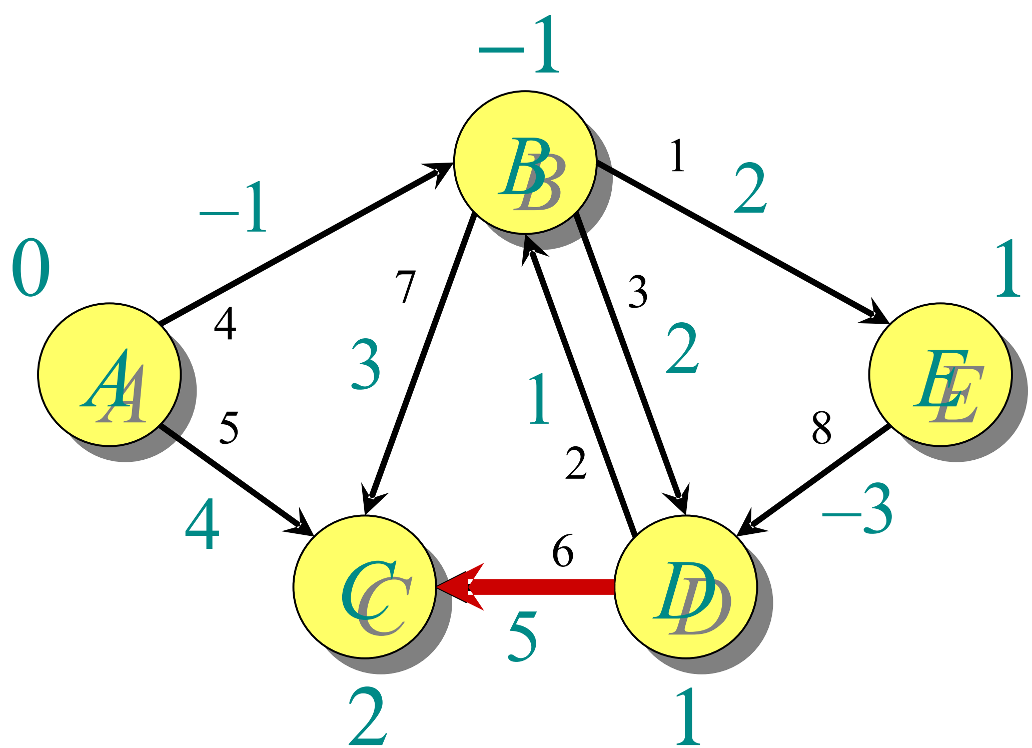


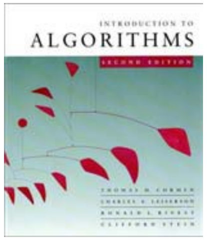
Example of Bellman-Ford



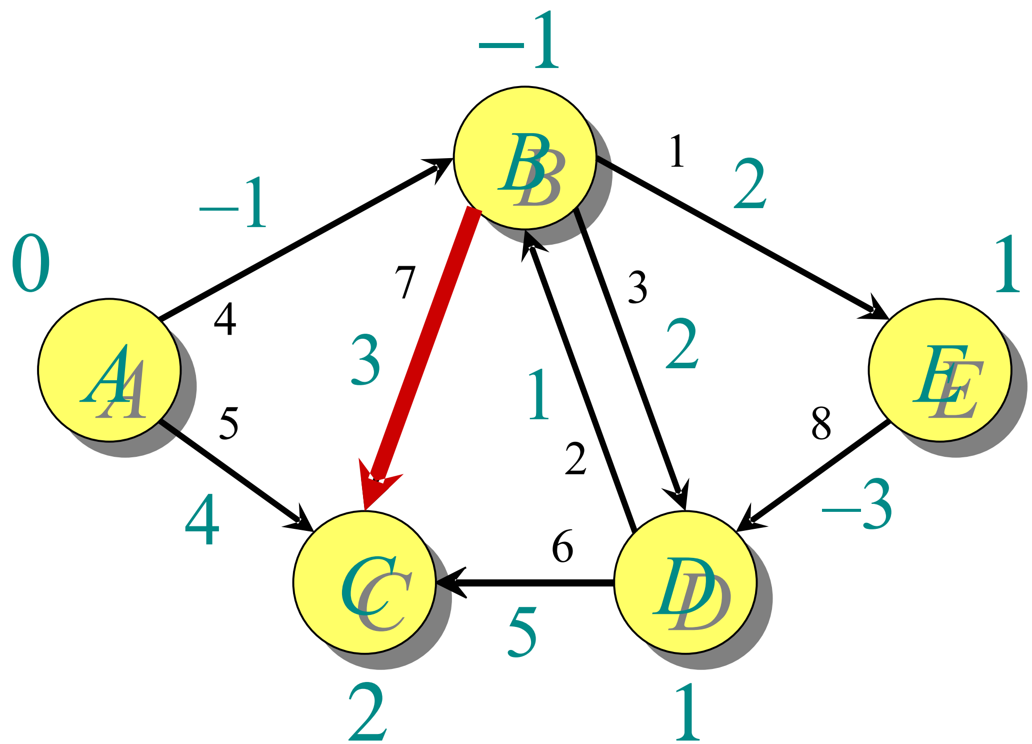


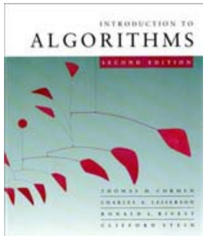
Example of Bellman-Ford



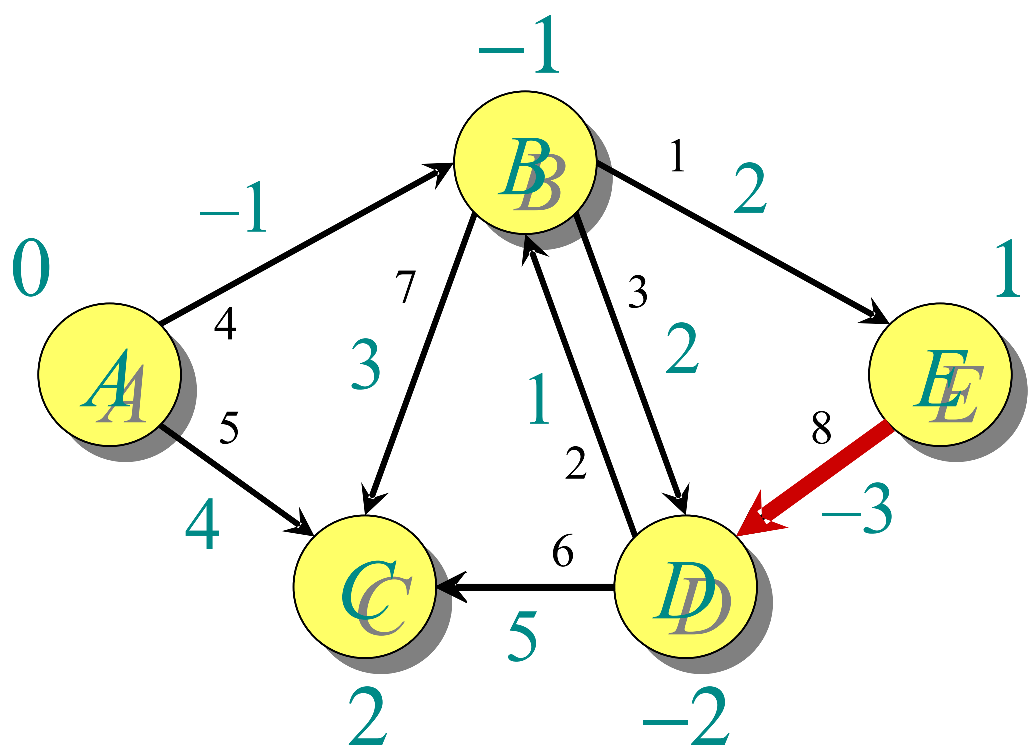


Example of Bellman-Ford

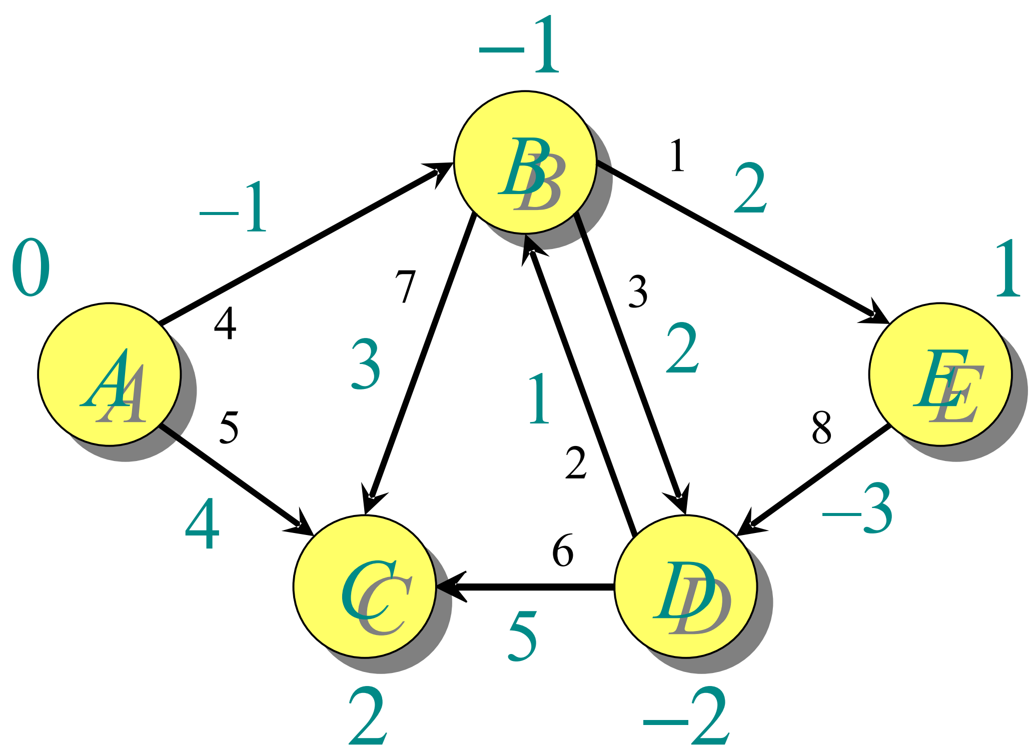




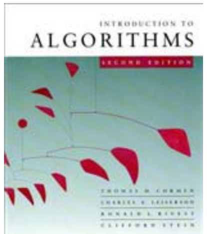
Example of Bellman-Ford



Example of Bellman-Ford

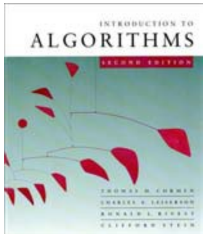


End of pass 2 (and 3 and 4).



Correctness

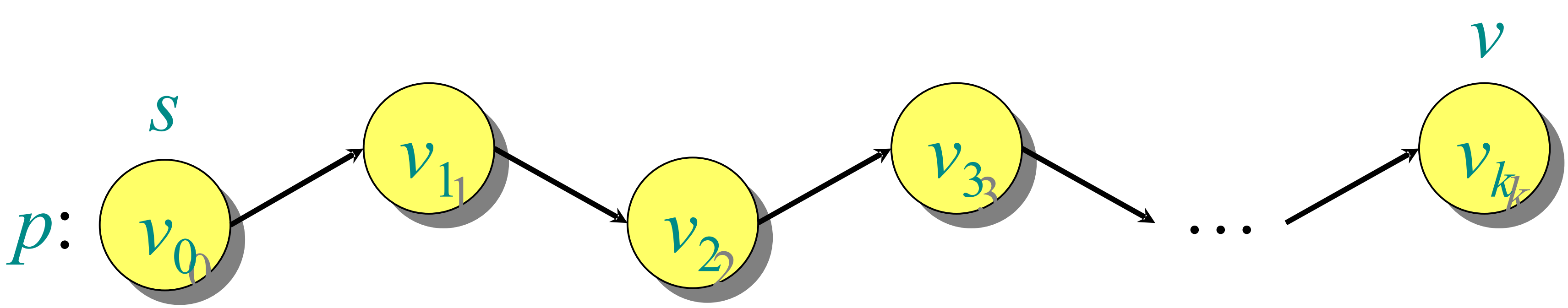
Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.



Correctness

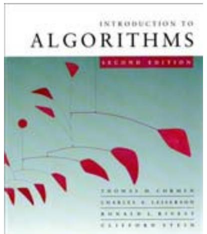
Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.

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

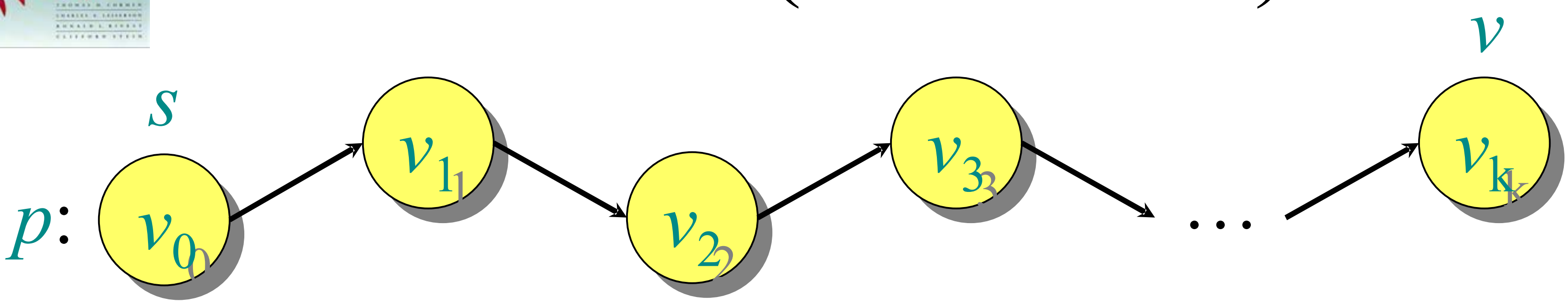


Since p is a shortest path, we have

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



Correctness (continued)



Initially, $d[v_0] = 0 = \delta(s, v_0)$, and $d[v_0]$ is unchanged by subsequent relaxations (because of the lemma $d[v] \ge \delta(s,v)$)

- 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 $\le |V| - 1$ 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. $\square$

DAG Shortest Paths

Bellman-Ford takes time $O(VE)$. Can we do better on a DAG?
Yes!

How?
We can apply *topological sorting* on the DAG, and *relax* every edge only *once*.

DAG Shortest Paths

$\Theta(E+V)$

$\Theta(V)$

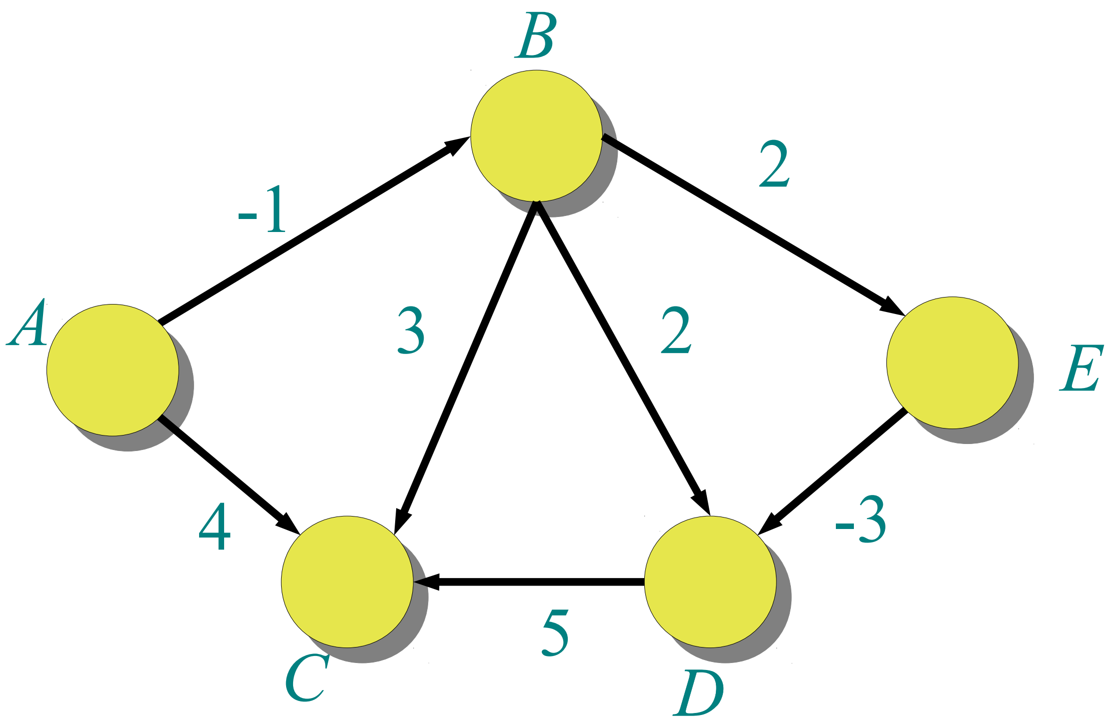
$\Theta(E)$

```
Topological-Sort(G)

for v in V
    d[v] = ∞
d[s] = 0

for each vertex u in topologically sorted order
        for each vertex v in Adj[u]
            if d[v] > d[u] + w(u,v)
                d[v] = d[u] + w(u,v)
```

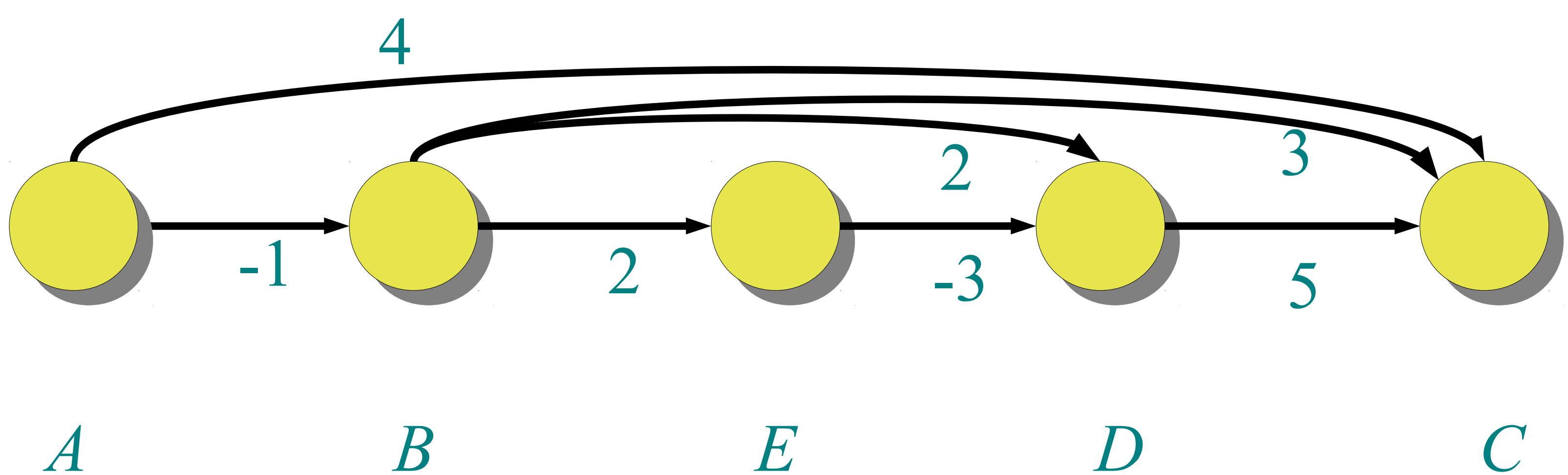
Total Time = $\Theta(E+V)$

Proof. Exercise.

DAG Shortest Paths

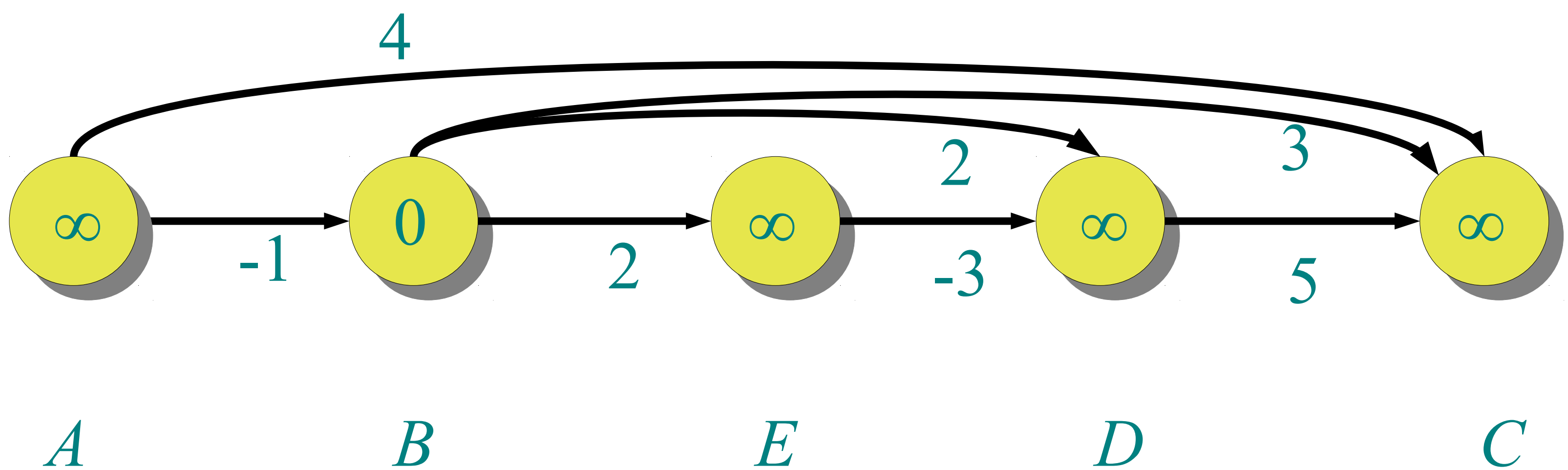


DAG Shortest Paths



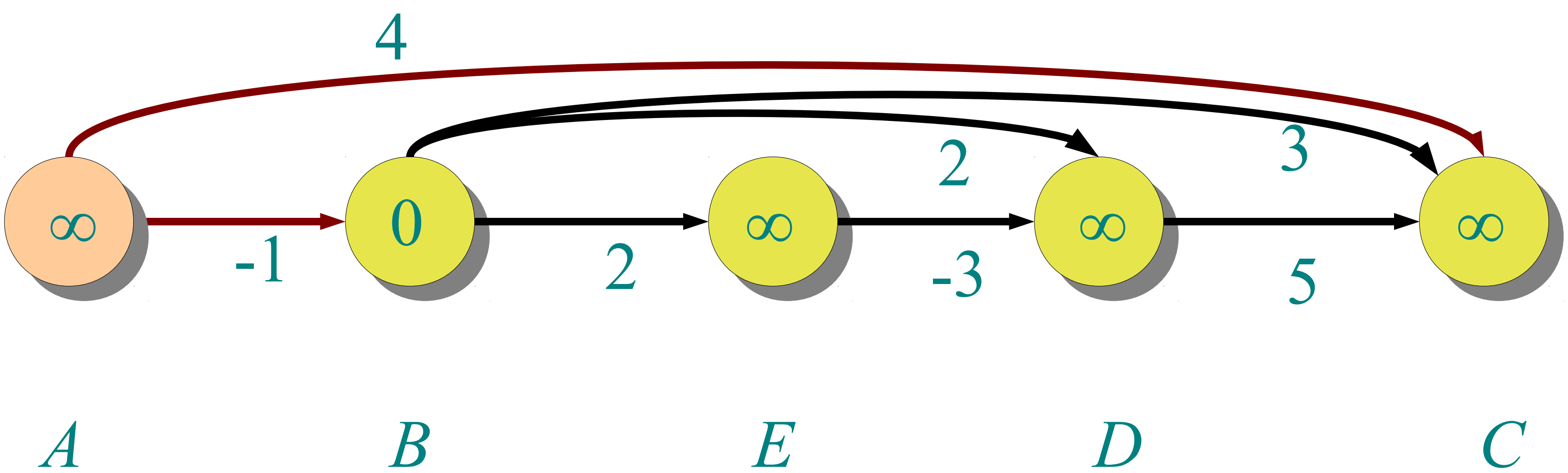
Topological Sorting

DAG Shortest Paths



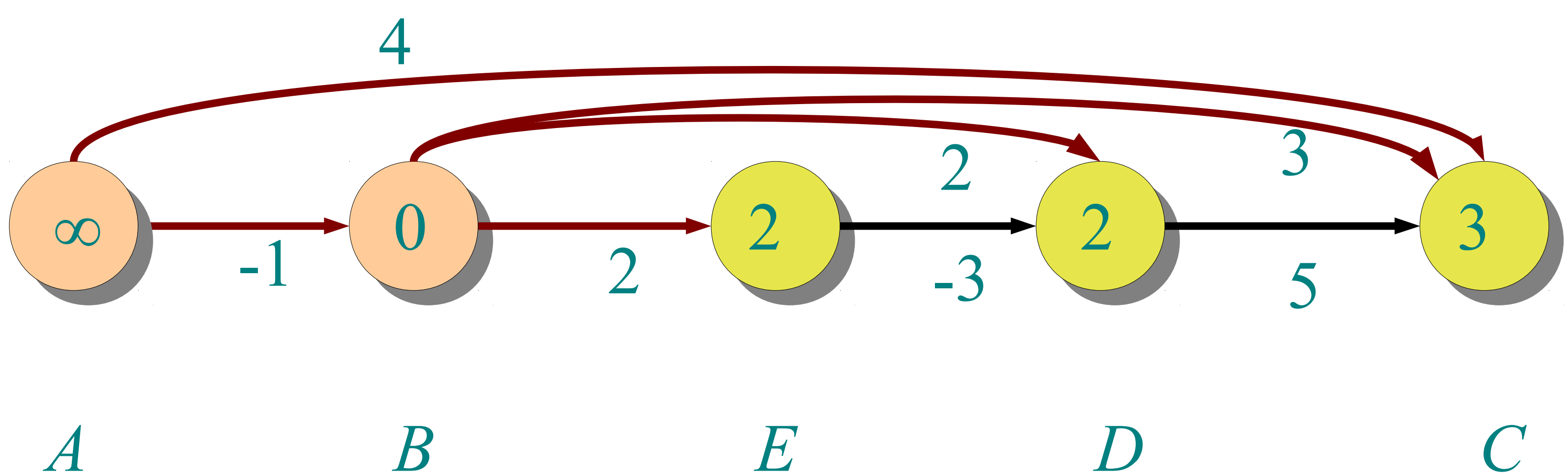
Initialize with *B* as a source

DAG Shortest Paths



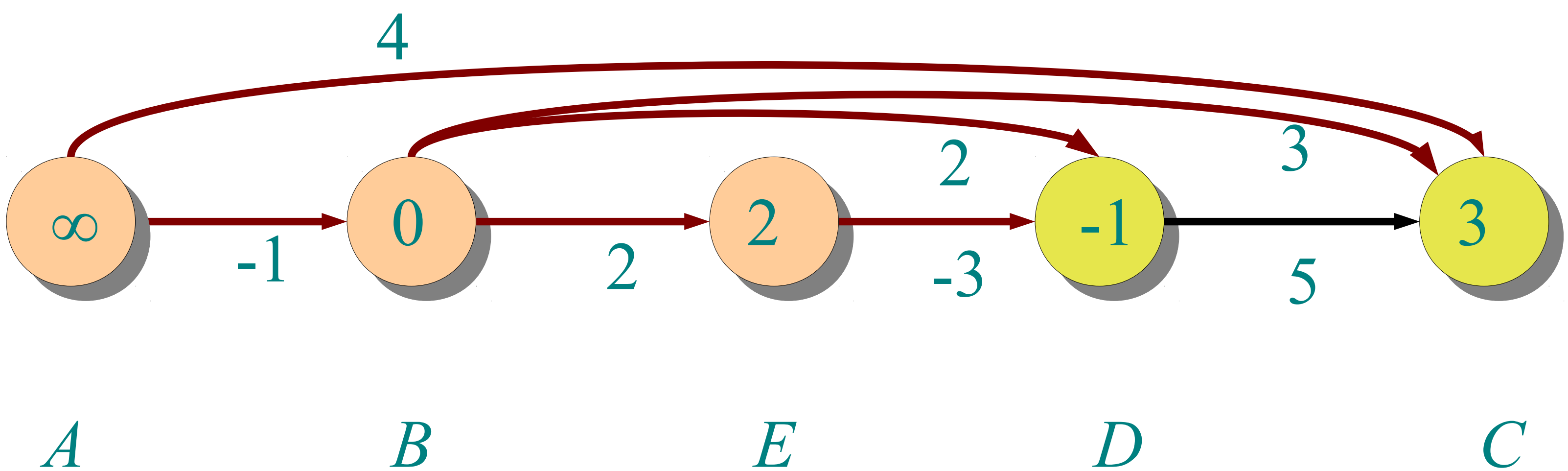
Process *A*

DAG Shortest Paths



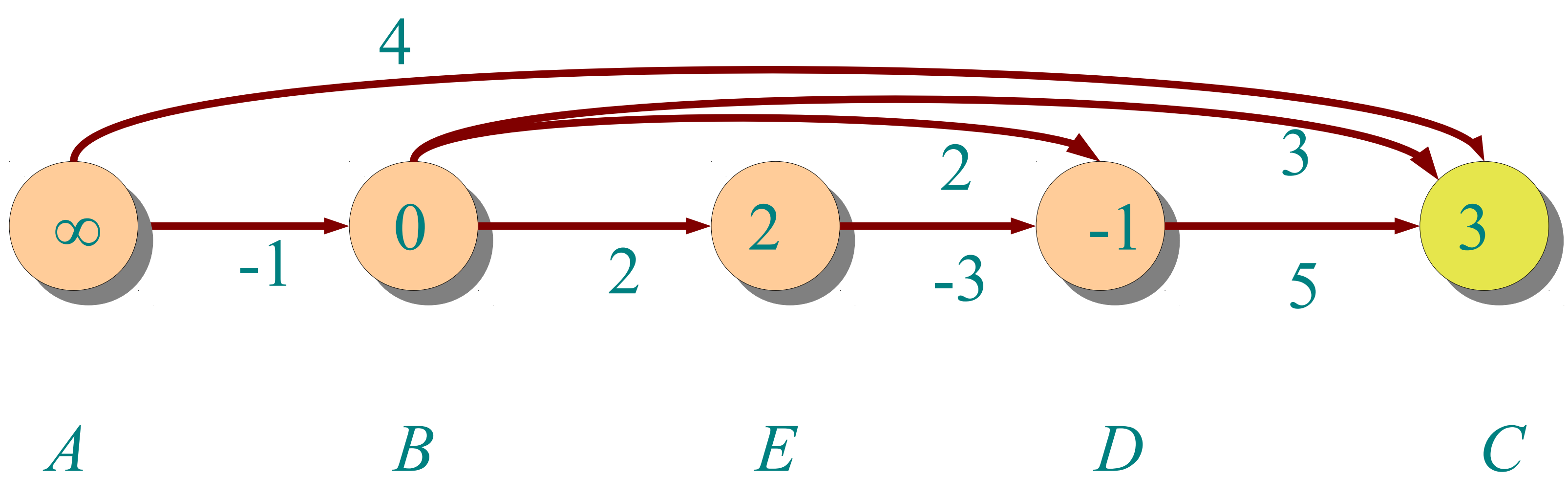
Process *B*

DAG Shortest Paths



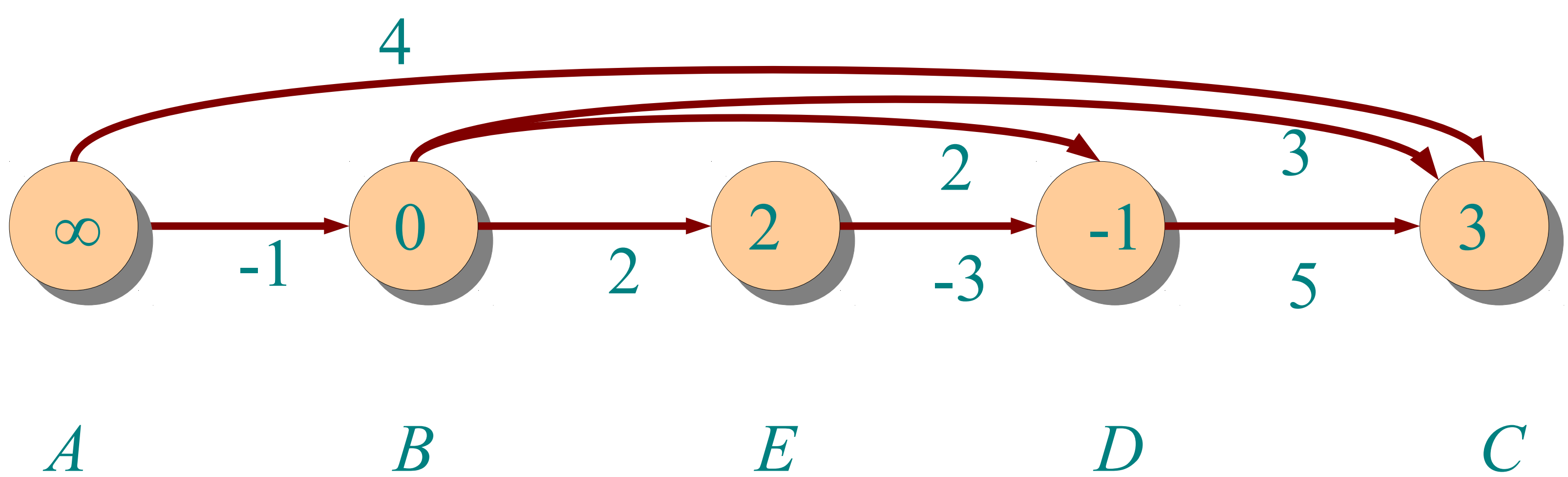
Process *E*

DAG Shortest Paths

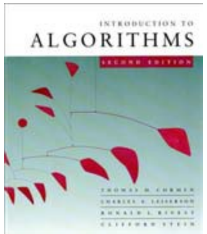


Process *D*

DAG Shortest Paths

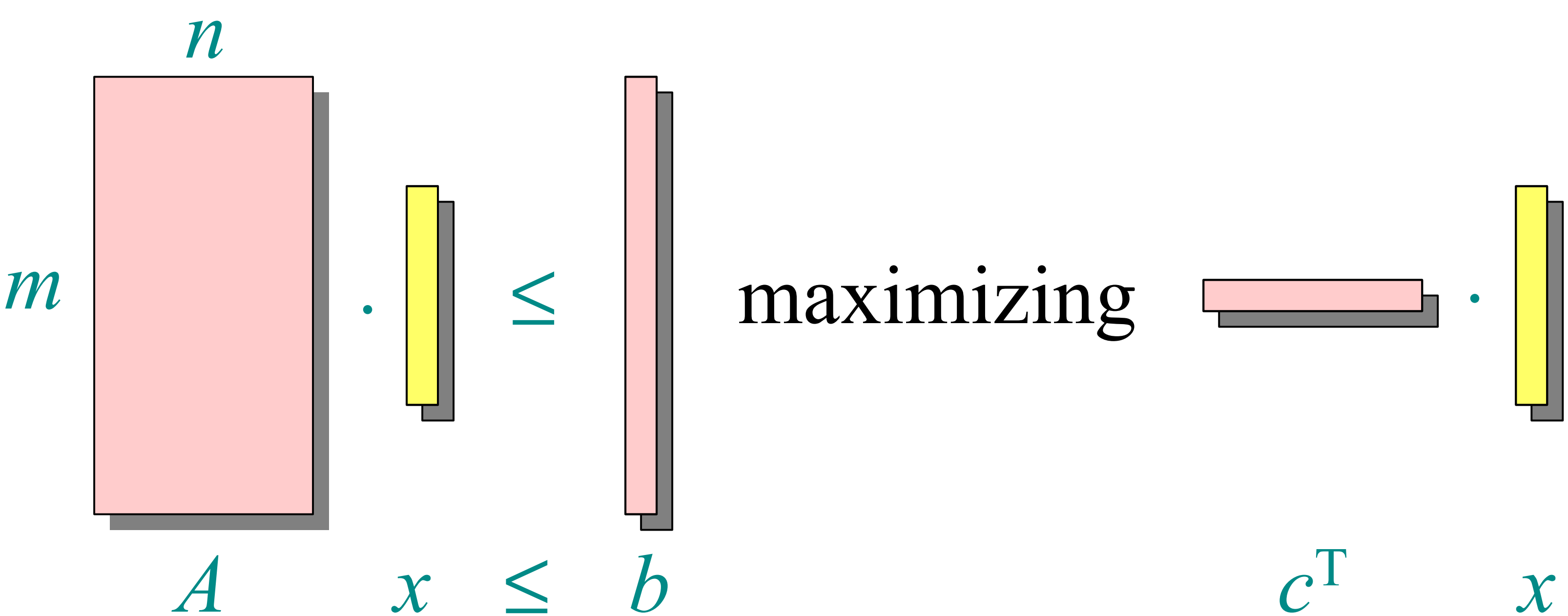


Process *C*

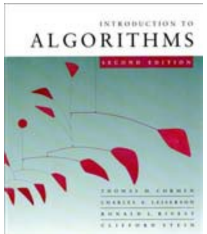


Linear programming

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 $Ax \le b$, or determine that no such solution exists.



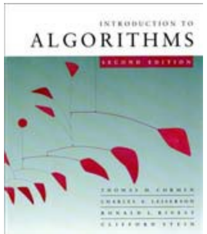
 Introduction to Algorithms November 3, 2004 L15.30



Linear-programming algorithms

Algorithms for the general problem

- Simplex methods — practical, but worst-case exponential time.
- Interior-point methods — polynomial time and competes with simplex.



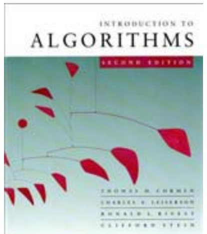
Linear-programming algorithms

Algorithms for the general problem

- Simplex methods — practical, but worst-case exponential time.
- 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.

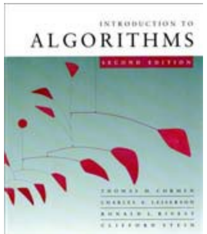


Solving a system of difference constraints

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

Example:

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



Solving a system of difference constraints

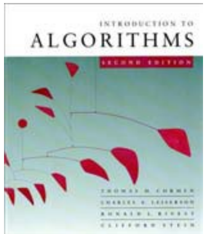
Linear programming where each row of A contains exactly one 1, one –1, and the rest 0's.

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:

$$\begin{aligned} x_1 &= 3 \\ x_2 &= 0 \\ x_3 &= 2 \end{aligned}$$



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:

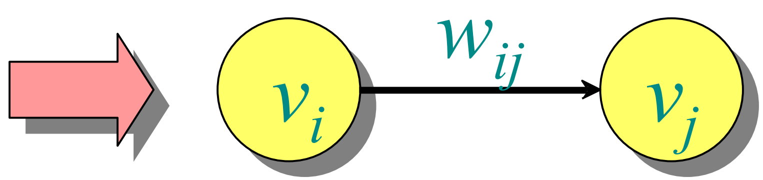
$$\left.\begin{aligned} x_1 - x_2 &\le 3 \\ x_2 - x_3 &\le -2 \\ x_1 - x_3 &\le 2 \end{aligned}\right\} \quad x_j - x_i \le w_{ij}$$

Solution:

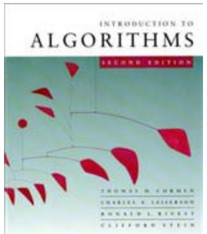
$$\begin{aligned} x_1 &= 3 \\ x_2 &= 0 \\ x_3 &= 2 \end{aligned}$$

Constraint graph:

$x_j - x_i \le w_{ij}$

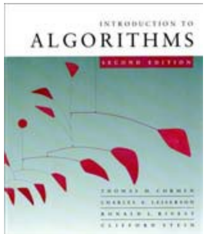


(The "A" matrix has dimensions $|E| \times |V|$.)



Unsatisfiable constraints

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

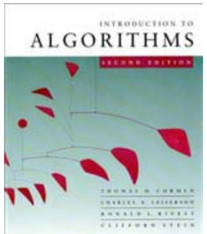


Unsatisfiable constraints

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

$$
\begin{aligned}
x_2 - x_1 &\le w_{12} \\
x_3 - x_2 &\le w_{23} \\
&\;\vdots \\
x_k - x_{k-1} &\le w_{k-1,\,k} \\
x_1 - x_k &\le w_{k1}
\end{aligned}
$$



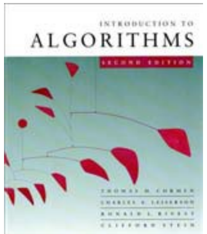
Unsatisfiable constraints

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

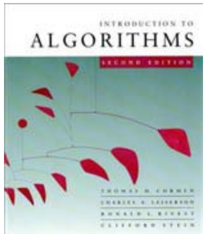
$$
\begin{array}{rcl}
x_2 - x_1 & \le & w_{12} \\
x_3 - x_2 & \le & w_{23} \\
& \vdots & \\
x_k - x_{k-1} & \le & w_{k-1,\,k} \\
x_1 - x_k & \le & w_{k1} \\
\hline
0 & \le & \text{weight of cycle} \\
& < & 0
\end{array}
$$

Therefore, no values for the x_i can satisfy the constraints. $\square$



Satisfying the constraints

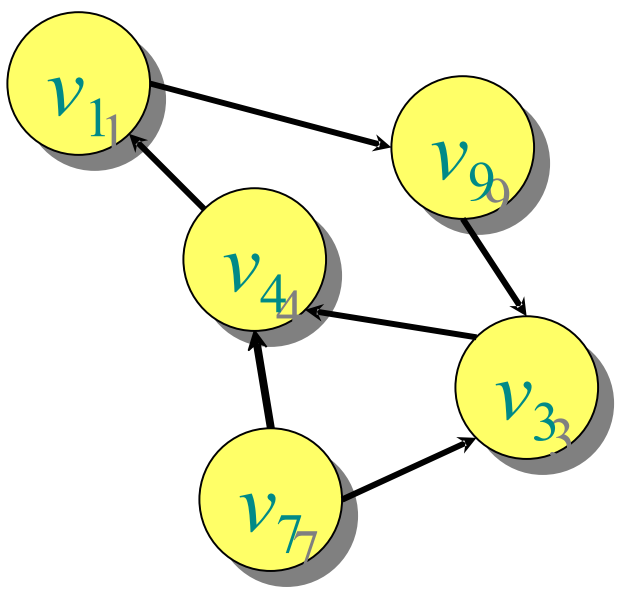
Theorem. Suppose no negative-weight cycle exists in the constraint graph. Then, the constraints are satisfiable.

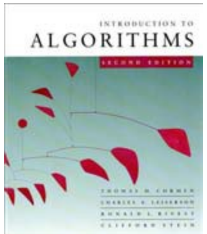


Satisfying the constraints

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 $v_i \in V$.

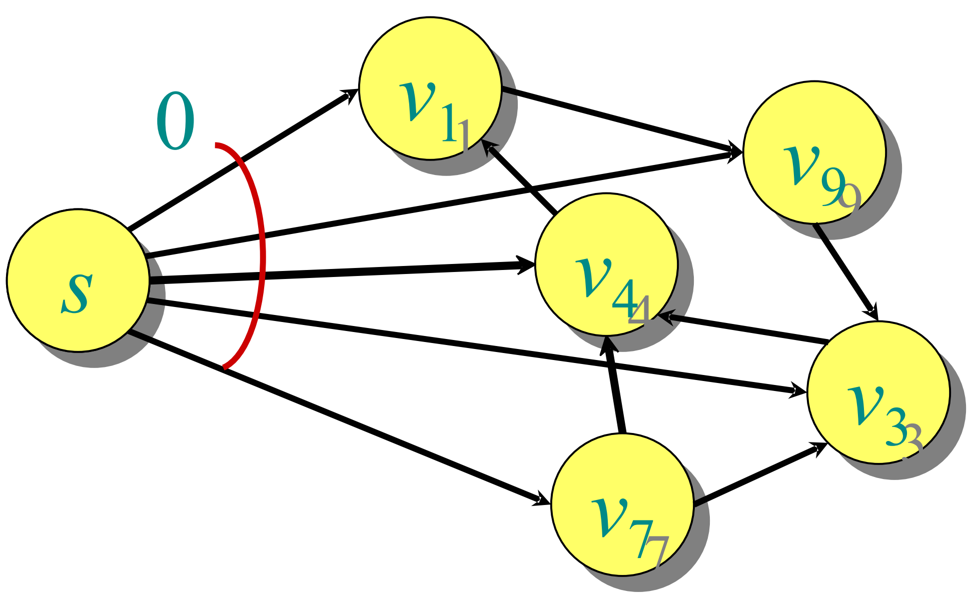




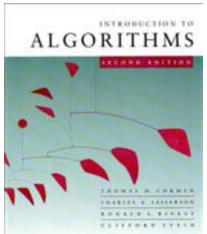
Satisfying the constraints

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 $v_i \in V$.



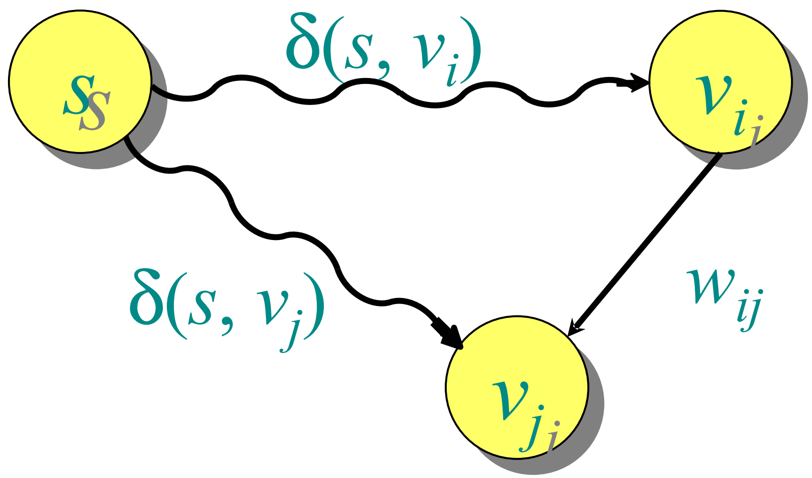
Note:
No negative-weight cycles introduced $\Rightarrow$ shortest paths exist.



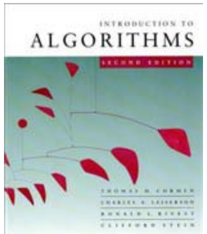
Proof (continued)

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

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

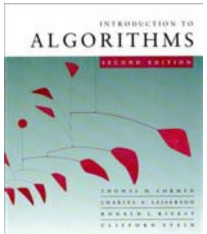


The triangle inequality gives us $\delta(s,v_j) \le \delta(s, v_i) + w_{ij}$. Since $x_i = \delta(s, v_i)$ and $x_j = \delta(s, v_j)$, the constraint $x_j - x_i \le w_{ij}$ is satisfied. ◻



Bellman-Ford and linear programming

Corollary. The Bellman-Ford algorithm can solve a system of m difference constraints on n variables in $O(n^2 + nm)$ time $\square$



Bellman-Ford and linear programming

Corollary. The Bellman-Ford algorithm can solve a system of m difference constraints on n variables in $O(n^2 + nm)$ time

Proof.
The augmeted constraint graph has $n+1$ vertices (variables) and $m+n$ edges. why?
Therefore, Bellman-Ford takes time

$$O(VE) = O((n+1)(m+n)) = O(n^2 + nm)$$

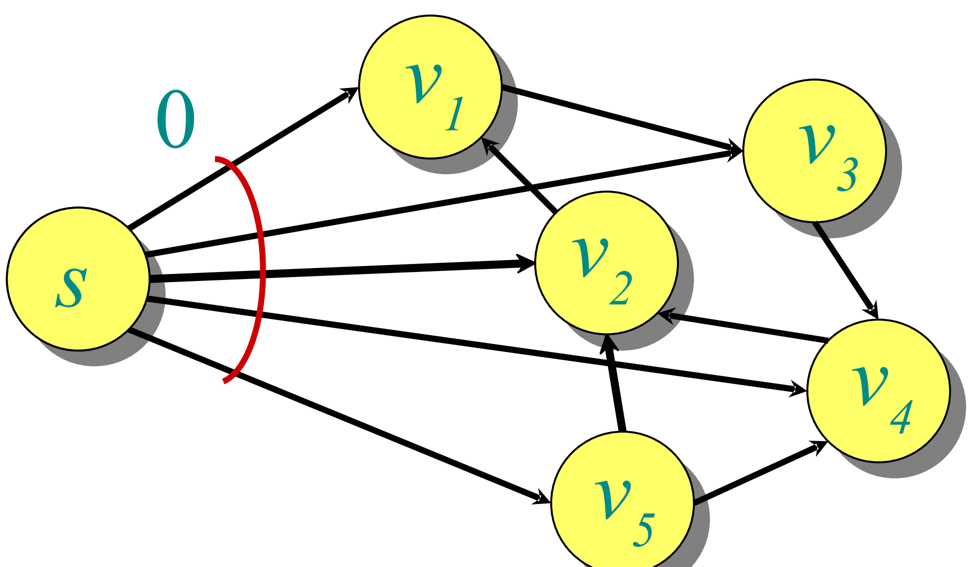
Can we make it take time $O(mn)$ instead? Yes!

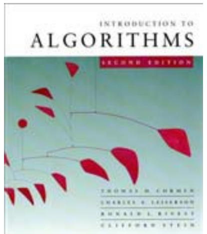
Bellman-Ford and Linear Programming

Notice that the *first* time any edge (s, v_i) is updated, $d[v_i]$ is set to 0 instead of ∞, and it does *not* increase afterwards. *Why?*

So we can do away with s altogether, and initialize $d[v_i] = 0$

Therefore, we will have n vertices and m edges, and Bellman-Ford will take time $O(mn)$





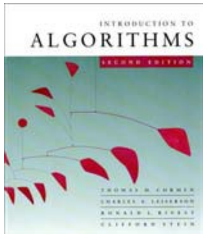
Bellman-Ford and linear programming

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

Single-source shortest paths is a simple LP problem.

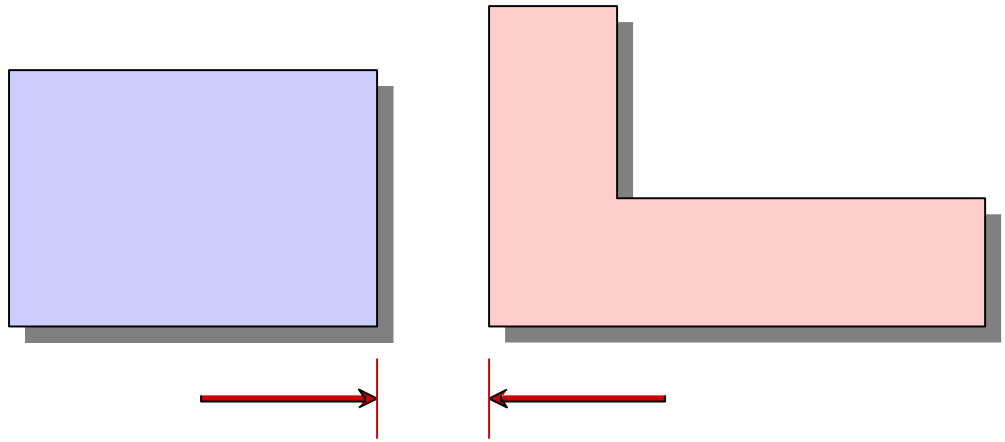
In fact, Bellman-Ford maximizes $x_1 + x_2 + \cdots + x_n$ subject to the constraints $x_j - x_i \le w_{ij}$ and $x_i \le 0$ (exercise).

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



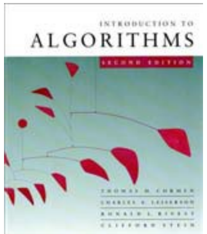
Application to VLSI layout compaction

Integrated -circuit features:

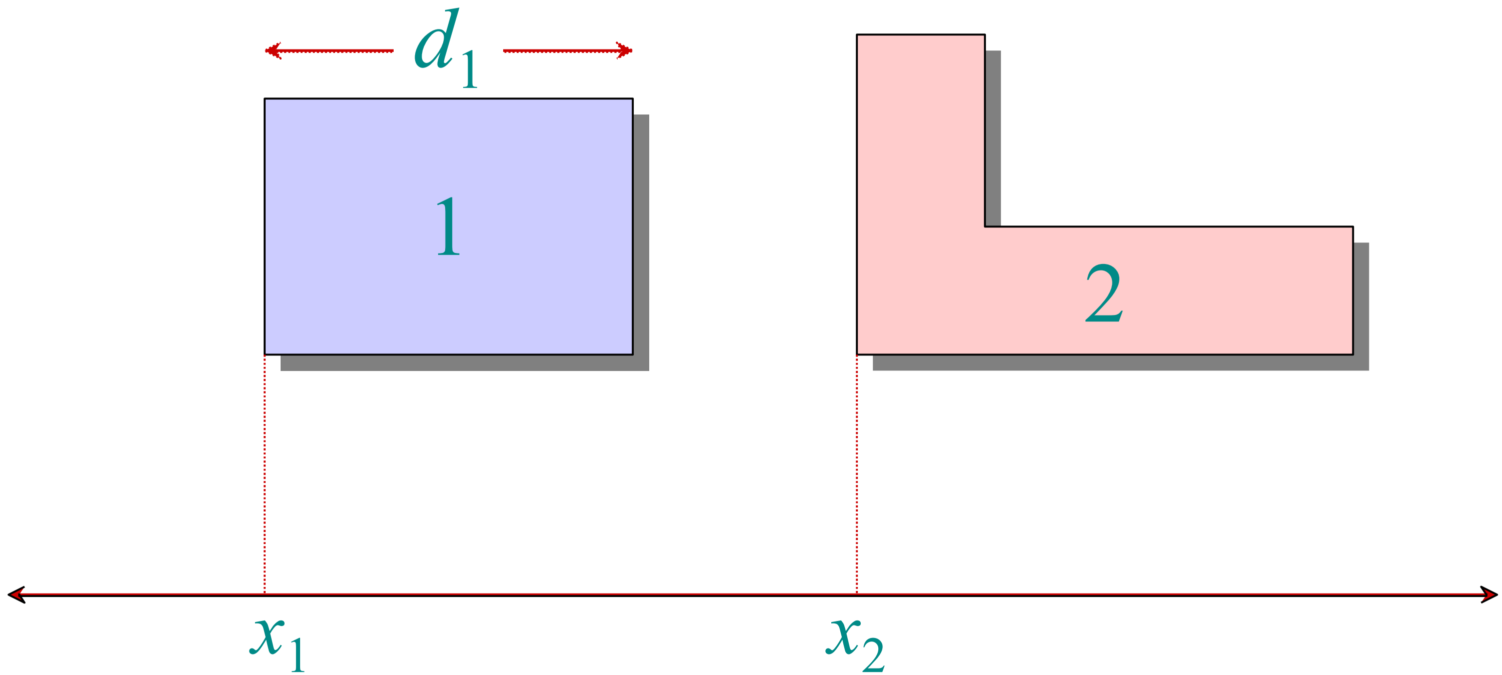


minimum separation λ

Problem: Compact (in one dimension) the space between the features of a VLSI layout without bringing any features too close together.



VLSI layout compaction



Constraint: $x_2 - x_1 \geq d_1 + \lambda$

Bellman-Ford minimizes $\max_i\{x_i\} - \min_i\{x_i\}$, which compacts the layout in the x-dimension.

Recap

- Bellman-Ford Algorithm
- DAG Shortest Paths
- Linear Programming and Difference Constraints
- VLSI Layout Compaction