CSE
565

LECTURES 15

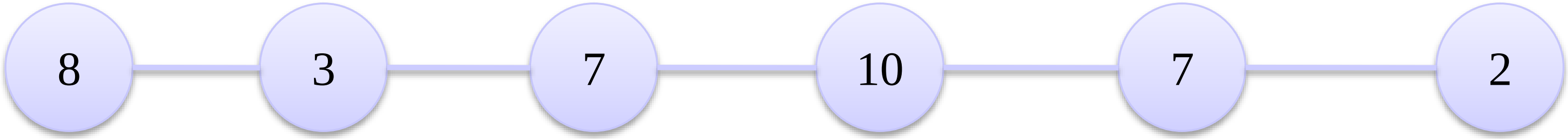
Dynamic Programming

- RNA Secondary Structure
- Shortest Paths: Bellman-Ford

Sofya Raskhodnikova

Review

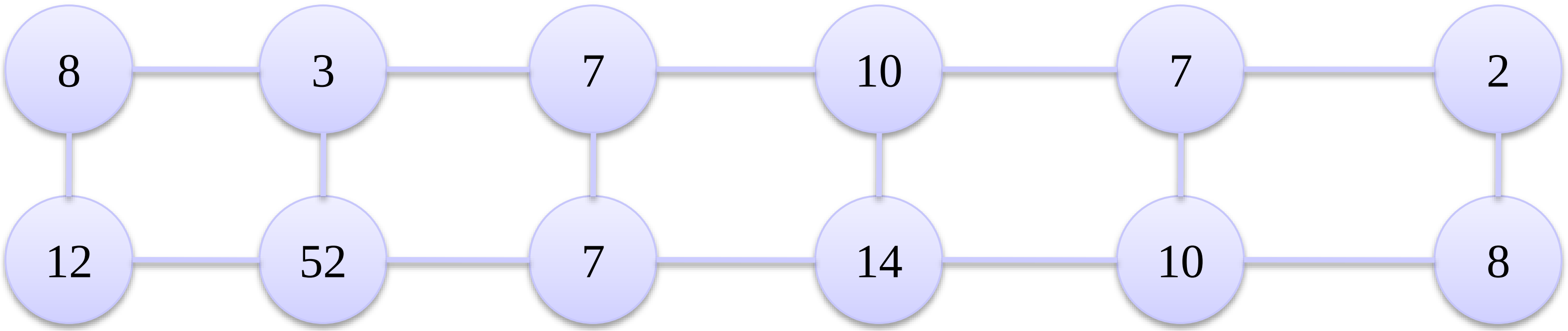
- Weighted independent set on the chain
  - Input: a chain graph of length n with values $v_1,..,v_n$
  - Goal: find a heaviest independent set



- Let OPT(j) = ???
- Write down a recursive formula for OPT
  - How many different subproblems?

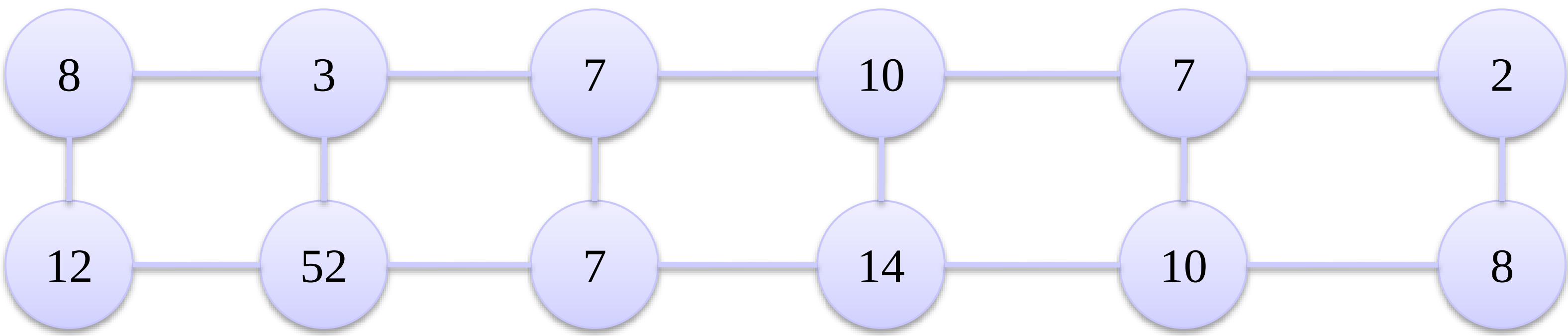
Exercise

- Do the same with a 2 x n grid graph

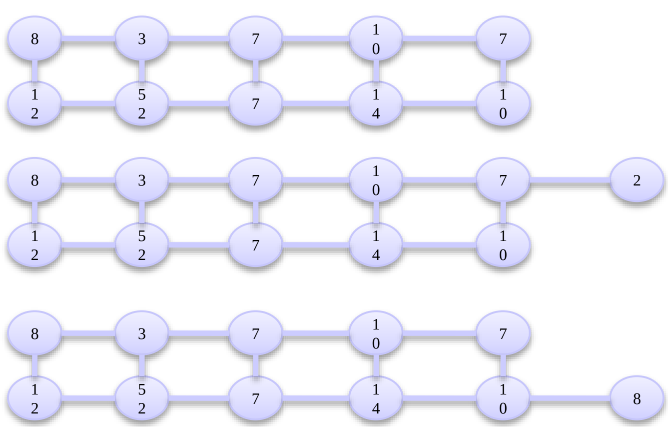


Exercise

- Do the same with a 2 x n grid graph

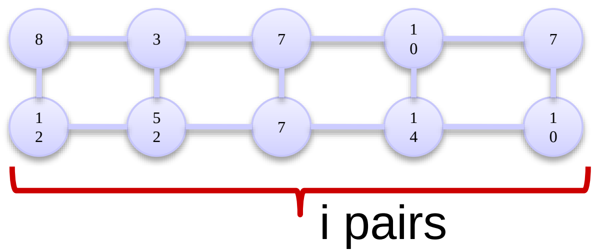


- Three types of subproblems:
  - grid(i)
  - gridTop(i)
  - gridBottom(i)

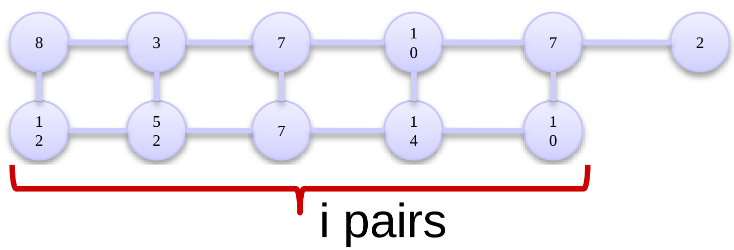


Exercise

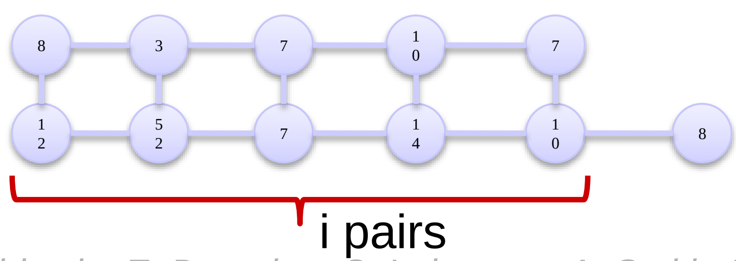
– grid(i): maximum independent set in the subgraph consisting of only the first i pairs of nodes



– gridTop(i): maximum independent set in the subgraph consisting of the first i pairs of nodes plus the top node of the (i+1)-st pair



– gridBottom(i): maximum independent set in the subgraph consisting of the first i pairs of nodes plus the bottom node of the (i+1)-st pair



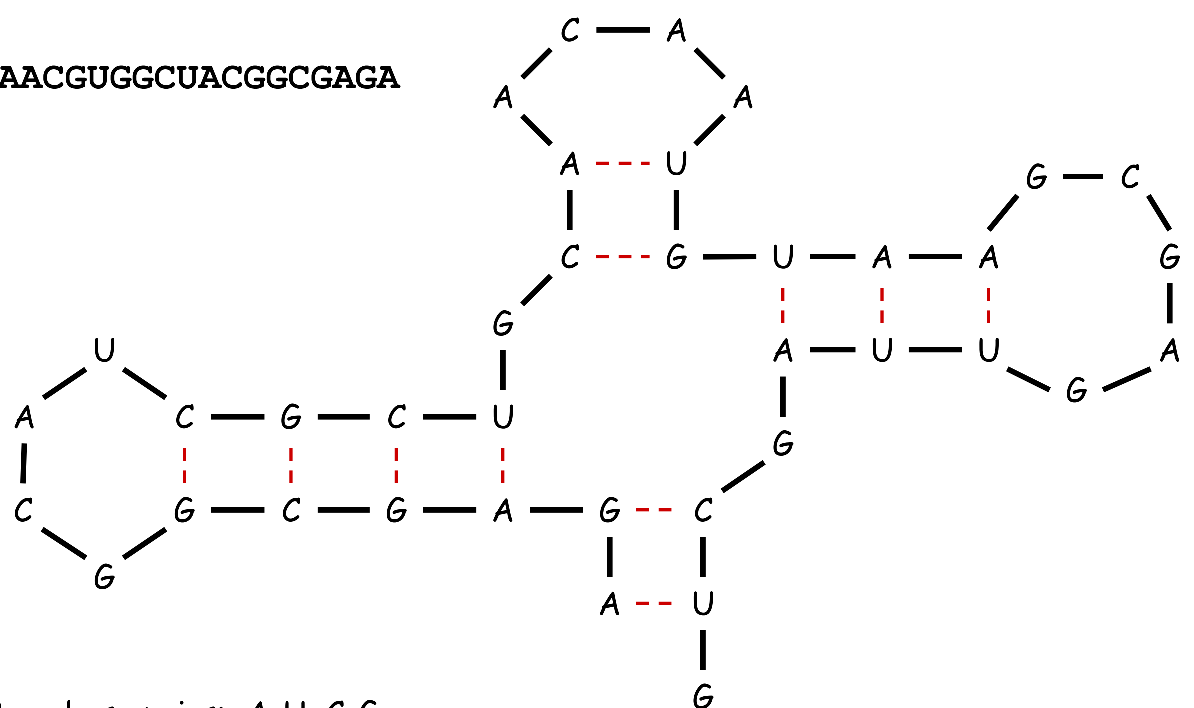
RNA Secondary Structure

RNA Secondary Structure

RNA. String B = $b_1 b_2 \ldots b_n$ over alphabet { A, C, G, U }.

Secondary structure. RNA is single-stranded so it tends to loop back and form base pairs with itself. This structure is essential for understanding behavior of molecule.

Ex: **GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA**



complementary base pairs: A-U, C-G

RNA Secondary Structure

Secondary structure. A set of pairs $S = \{ (b_i, b_j) \}$ that satisfy:

- [Watson-Crick.] S is a matching and each pair in S is a Watson-Crick complement: A-U, U-A, C-G, or G-C.
- [No sharp turns.] The ends of each pair are separated by at least 4 intervening bases. If $(b_i, b_j) \in S$, then $i < j - 4$.
- [Non-crossing.] If (b_i, b_j) and (b_k, b_l) are two pairs in S, then we cannot have $i < k < j < l$.

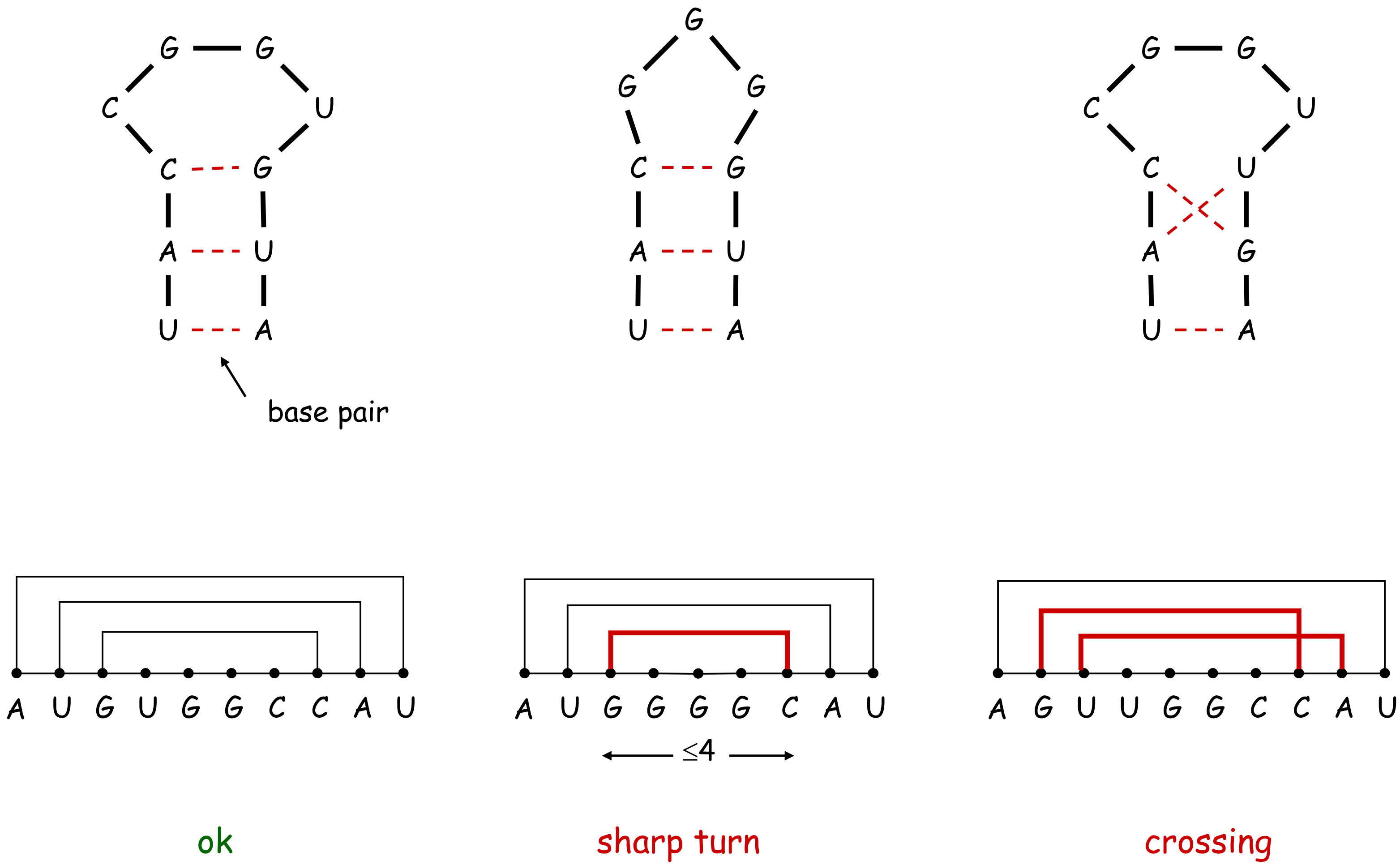
Free energy. Usual hypothesis is that an RNA molecule will form the secondary structure with the optimum total free energy.

approximate by number of base pairs

Goal. Given an RNA molecule $B = b_1 b_2 \ldots b_n$, find a secondary structure S that maximizes the number of base pairs.

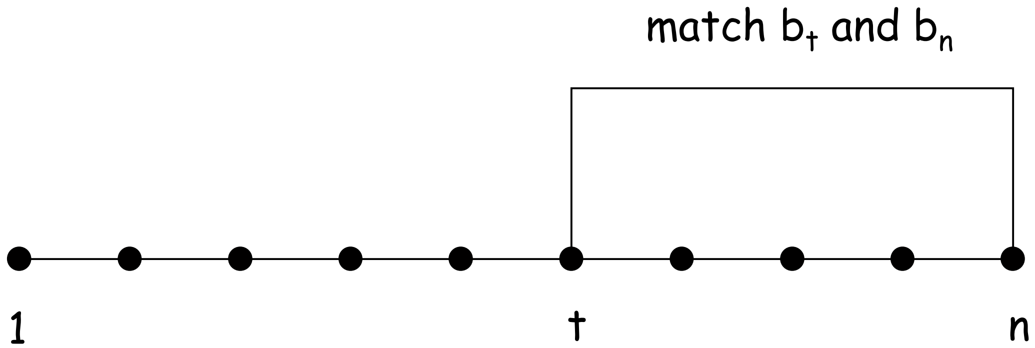
RNA Secondary Structure: Examples

Examples.



RNA Secondary Structure: Subproblems

First attempt. $OPT(j)$ = maximum number of base pairs in a secondary structure of the substring $b_1 b_2 \ldots b_j$.



Difficulty. Results in two sub-problems.

- Finding secondary structure in: $b_1 b_2 \ldots b_{t-1}$. ← $OPT(t-1)$
- Finding secondary structure in: $b_{t+1} b_{t+2} \ldots b_{n-1}$. ← need more sub-problems

Dynamic Programming Over Intervals

Notation. $OPT(i, j)$ = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

- Case 1. $i \geq j - 4$.
  - $OPT(i, j) = 0$ by no-sharp turns condition.

- Case 2. Base b_j is not involved in a pair.
  - $OPT(i, j) = OPT(i, j-1)$

- Case 3. Base b_j pairs with b_t for some $i \leq t < j - 4$.
  - non-crossing constraint decouples resulting sub-problems
  - $OPT(i, j) = 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}$

take max over t such that $i \leq t < j-4$ and
b_t and b_j are Watson-Crick complements

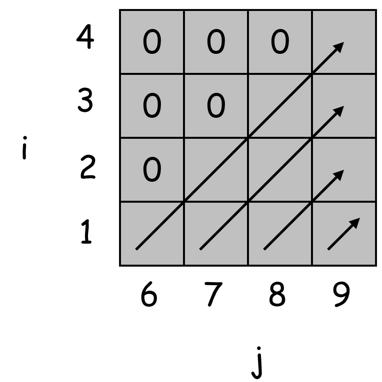
Bottom Up Dynamic Programming Over Intervals

Q. In what order should we solve the subproblems?

A. Do shortest intervals first.

```
RNA(b1,…,bn) {
   for k = 5, 6, …, n-1
      for i = 1, 2, …, n-k
         j = i + k
         Compute M[i, j]

   return M[1, n]
}
```

using recurrence



Running time. $O(n^3)$.

Dynamic Programming Summary

Recipe.

- Decide which subproblems to use (define OPT(???)).
- Recursively define value of optimal solution.
- Compute value of optimal solution (bottom up or via memoization).
- Construct optimal solution from computed information.

If it fails, try again.

Dynamic programming techniques.

- Binary choice: weighted interval scheduling.
- Multi-way choice: segmented least squares.
- Adding a new variable: knapsack.
- Dynamic programming over intervals: RNA secondary structure.

Viterbi algorithm for Hidden Markov Models also uses DP to optimize a maximum likelihood tradeoff between parsimony and accuracy

parsing algorithm for context-free grammars has similar structure

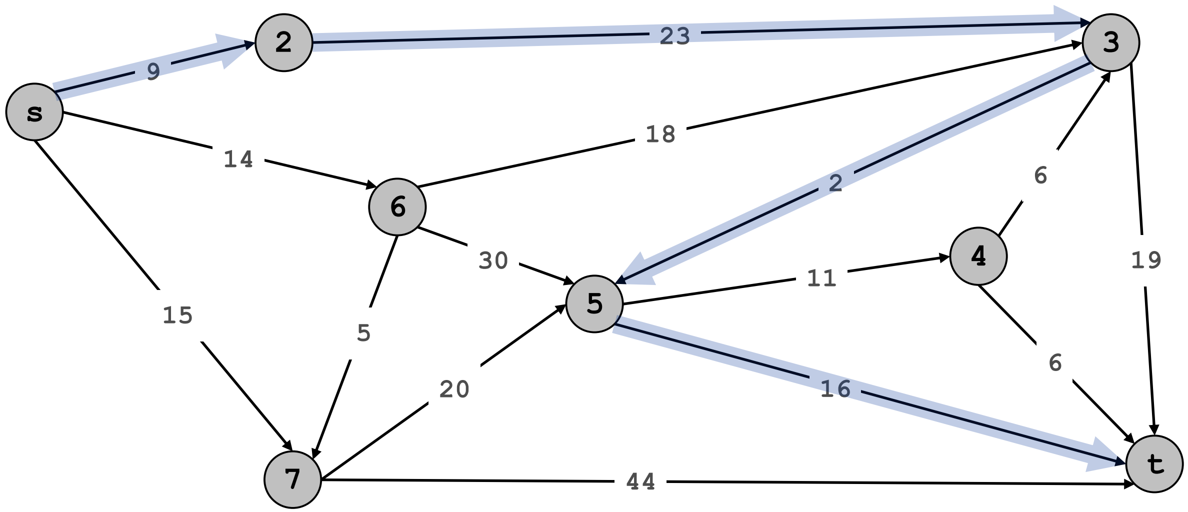
Top-down vs. bottom-up: different people have different intuition.

Bellman-Ford: Shortest paths via dynamic programming

For consistency with book: shortest paths from all vertices to a destination t

Single-source Shortest Path Problem

- **Input:**
  - Directed graph G = (V, E).
  - Source node s, destination node t.
  - for each edge e, length $\ell(e)$ = length of e.
  - length path = sum of edge lengths
- **Find:** shortest directed path from s to t.



Length of path (s,2,3,5,t)
is 9 + 23 + 2 + 16 = 50.

When is there a shortest path?

Under which conditins do shortest paths

- Always exist?
- Sometimes exist and sometimes not exist?
- Never exist?

- In a directed graph with nonnegative edge lengths?

- In a directed graph with negative edges lengths?

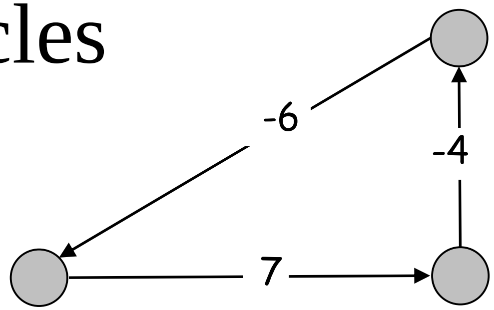
When is there a shortest path?

- Edge weights nonnegative:
  – shortest path always exists

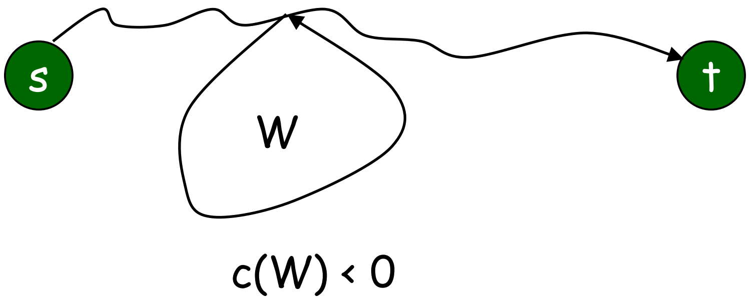
- Negative weight edges:
  – shortest path **may** exist or not
  – If *no negative cycles in G*, then shortest path exists
  – If *negative cycles in G*, then no shortest path

Shortest Paths: Negative-Cost Cycles

Negative cost cycles



Observation. If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise, there exists one that is simple.



Shortest Paths: Dynamic Programming

Def. OPT(i, v) = length of shortest v-t path P using at most i edges.

- Case 1: P uses at most i-1 edges.
  - OPT(i, v) = OPT(i-1, v)

- Case 2: P uses exactly i edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best w-t path using at most i-1 edges

$$OPT(i,v) = \begin{cases} 0 & \text{if } \; i = 0 \\ \min \left\{ OPT(i-1,\ v),\ \min\limits_{(v,w) \in E} \{ OPT(i-1,\ w) + c_{vw} \} \right\} & \text{otherwise} \end{cases}$$

Remark. By previous observation, if no negative cycles, then OPT(n-1, v) = length of shortest v-t path.

Shortest Paths: Implementation

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] ← ∞
   M[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
         M[i, v] ← M[i-1, v]
      foreach edge (v, w) ∈ E
         M[i, v] ← min { M[i, v], M[i-1, w] + c_vw }
}
```

Analysis. $\Theta(mn)$ time, $\Theta(n^2)$ space.

Finding the shortest paths. Maintain a "successor" for each table entry.

Shortest Paths: Improvements

- Maintain one array M[v] = length of shortest v-t path found so far.
- No need to check edges of the form (v, w) unless M[w] changed in previous iteration.

Theorem. Throughout the algorithm,

- M[v] is length of some v-t path, and
- For every i: after i rounds of updates, the value M[v] is no larger than the length of shortest v-t path using $\leq$ i edges.

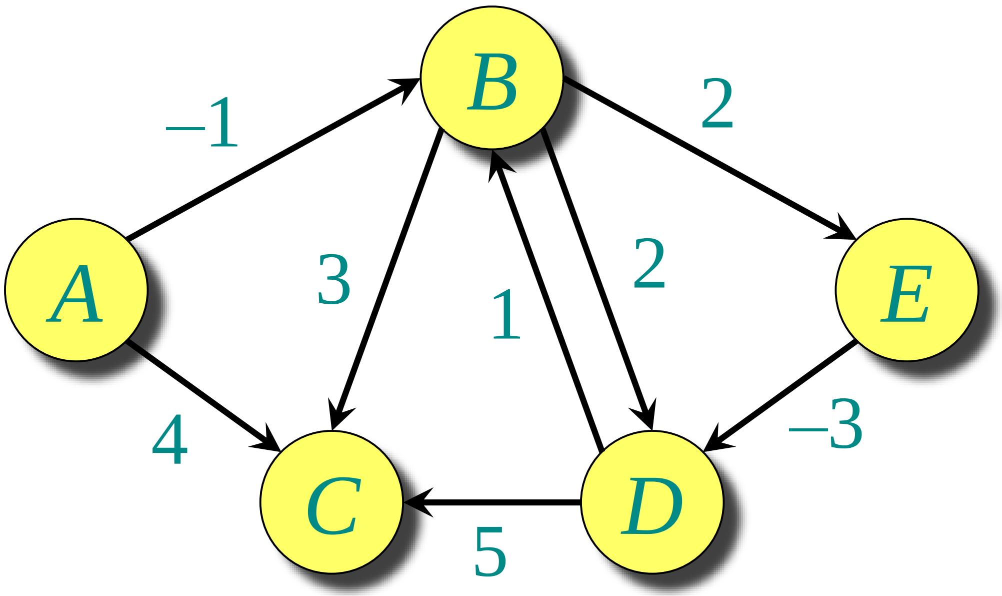
Space and time complexity.

- Memory: O(m + n).
- Running time: O(mn) worst case, but substantially faster in practice.

Belman-Ford: Efficient Implementation

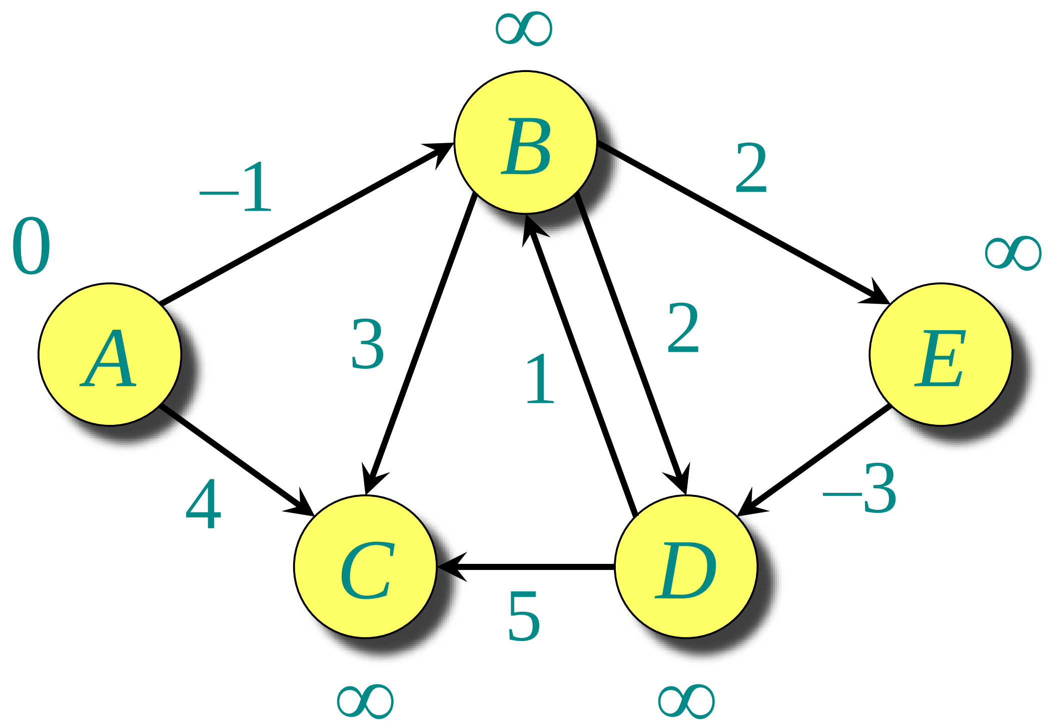
```
Bellman-Ford-Shortest-Path(G, s, t) {
   foreach node v ∈ V {
      M[v] ← ∞
      successor[v] ← φ
   }

   M[t] = 0
   for i = 1 to n-1 {
      foreach node w ∈ V {
      if (M[w] has been updated in previous iteration) {
         foreach node v such that (v, w) ∈ E {
            if (M[v] > M[w] + c_vw) {
               M[v] ← M[w] + c_vw
               successor[v] ← w
            }
         }
      }
      If no M[w] value changed in iteration i, stop.
   }
}
```

Example of Bellman-Ford



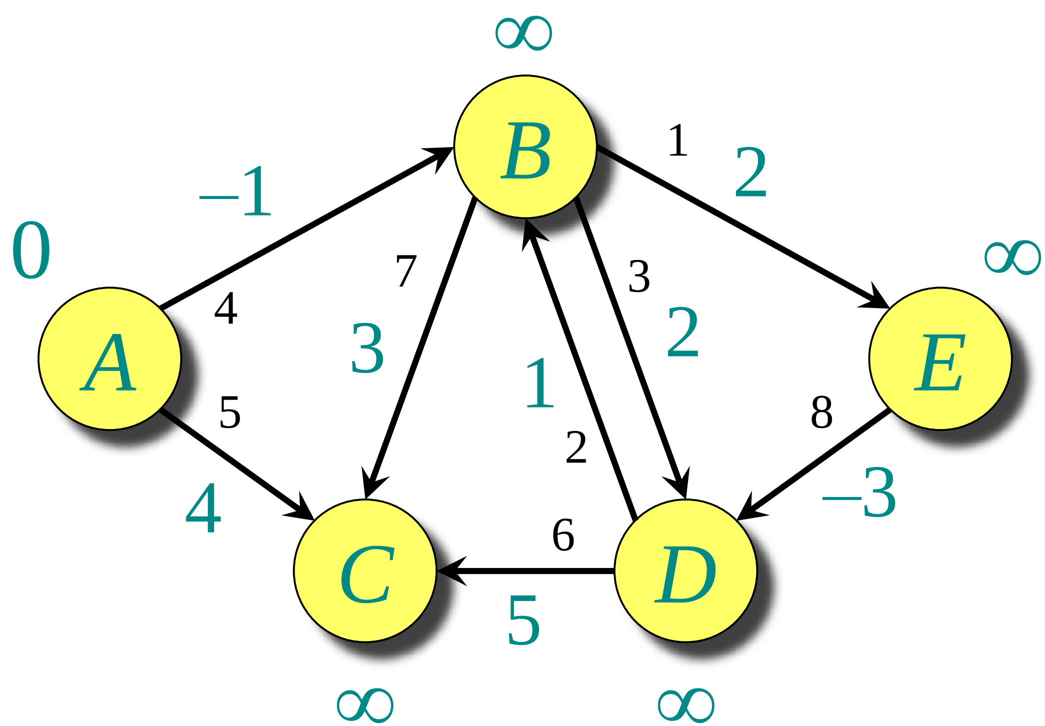
The demonstration is for a sligtly different version of the algorithm (see CLRS) that computes distances from the sourse node rather than distances to the destination node.

Example of Bellman-Ford



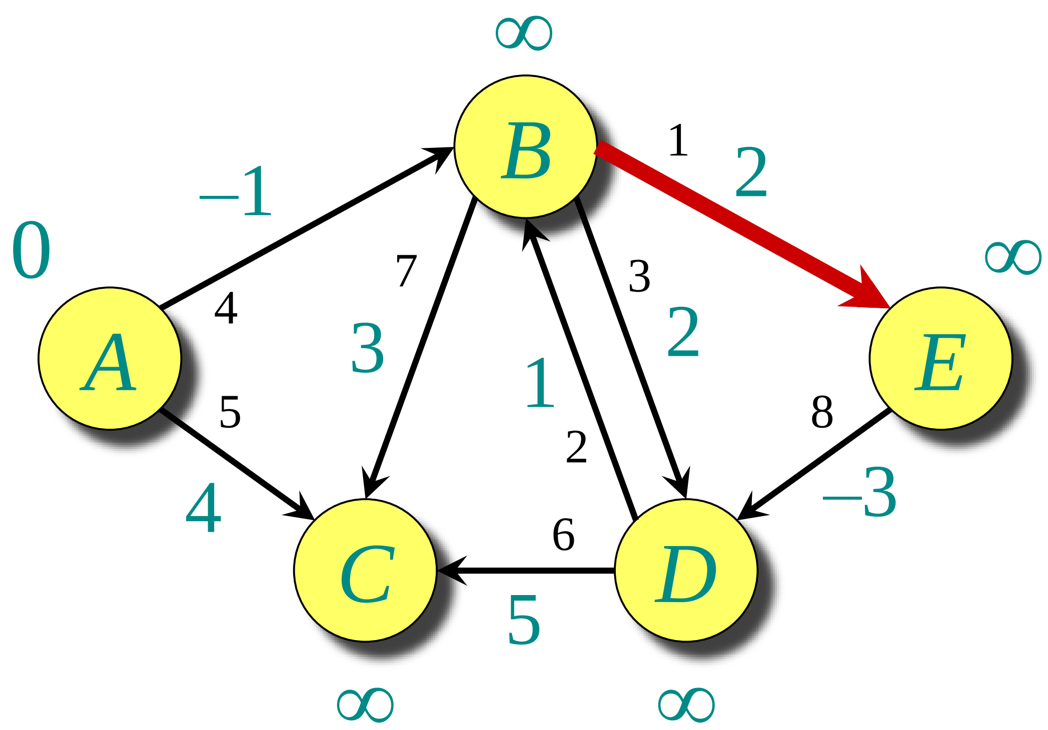
Initialization.

Example of Bellman-Ford

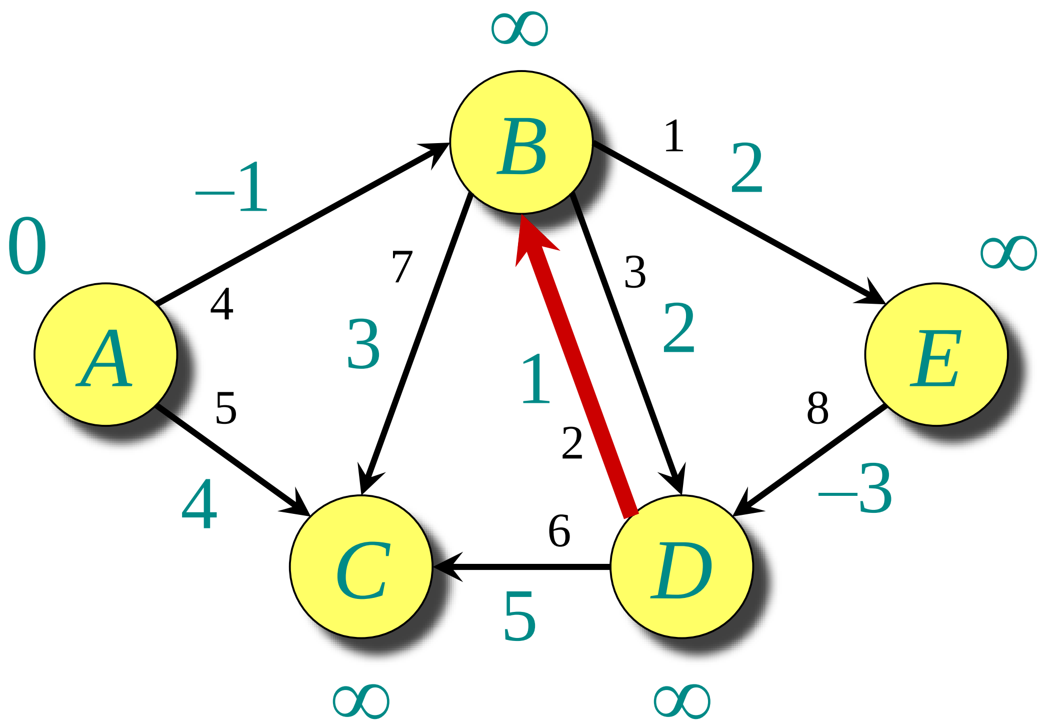


Order of edge relaxation.

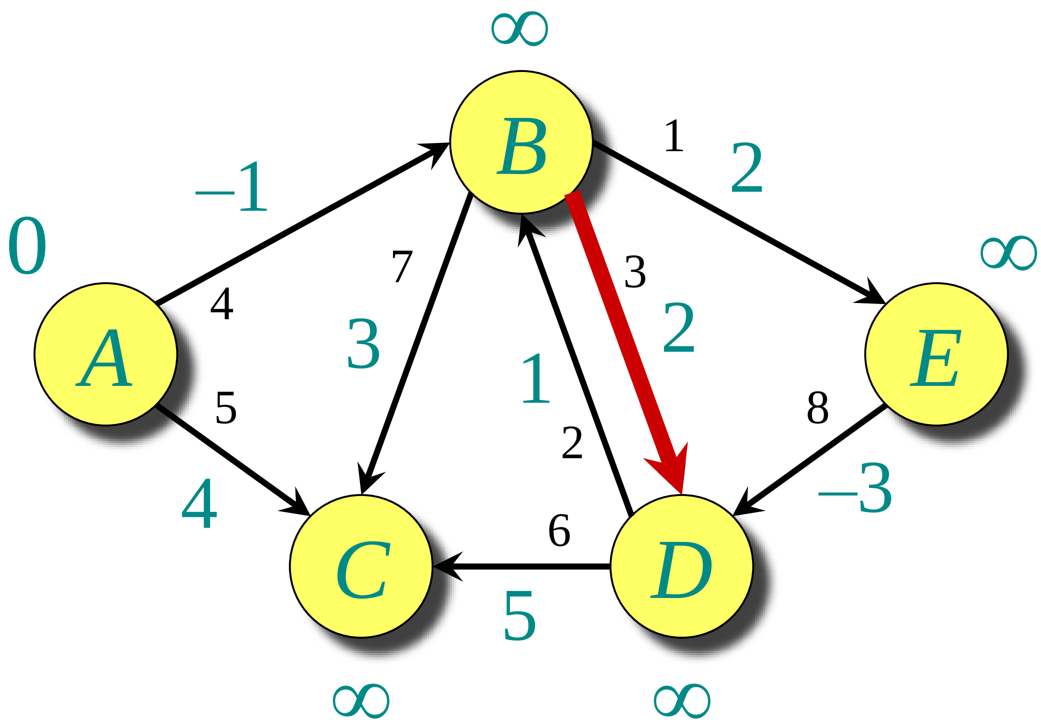
Example of Bellman-Ford



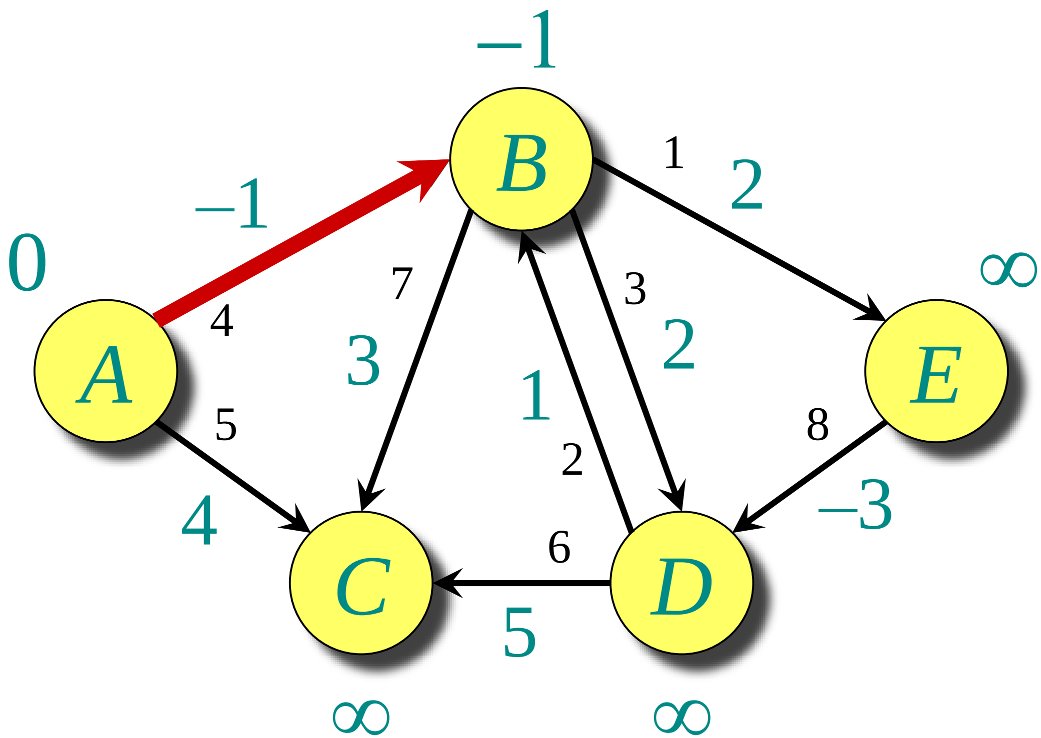
Example of Bellman-Ford



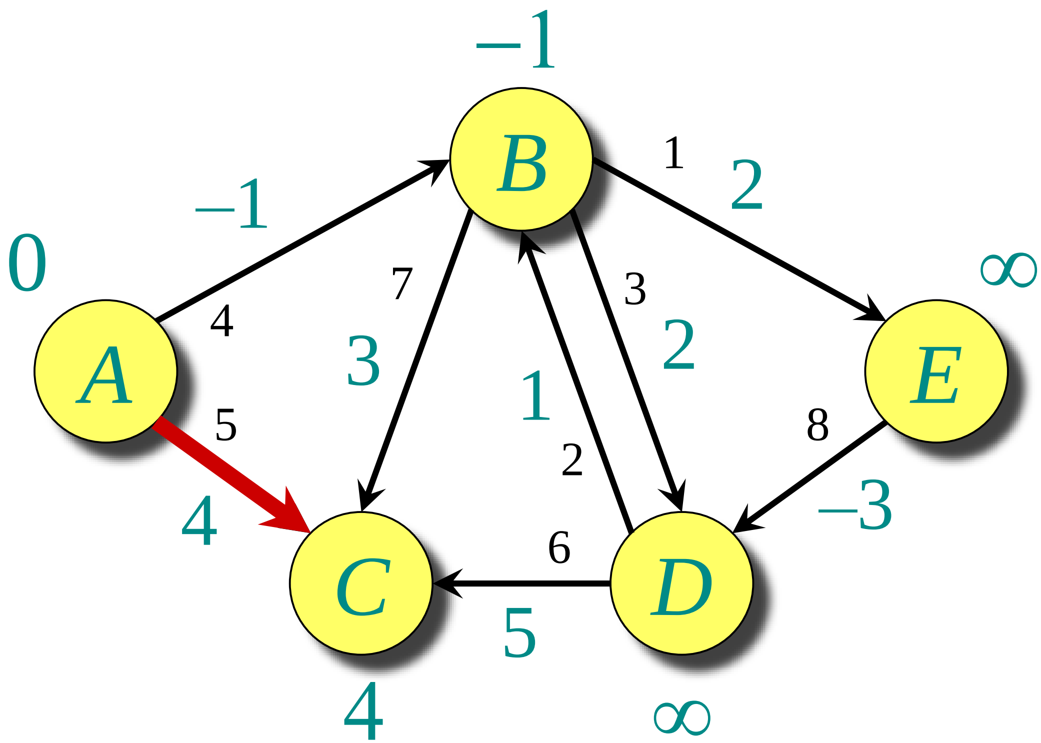
Example of Bellman-Ford



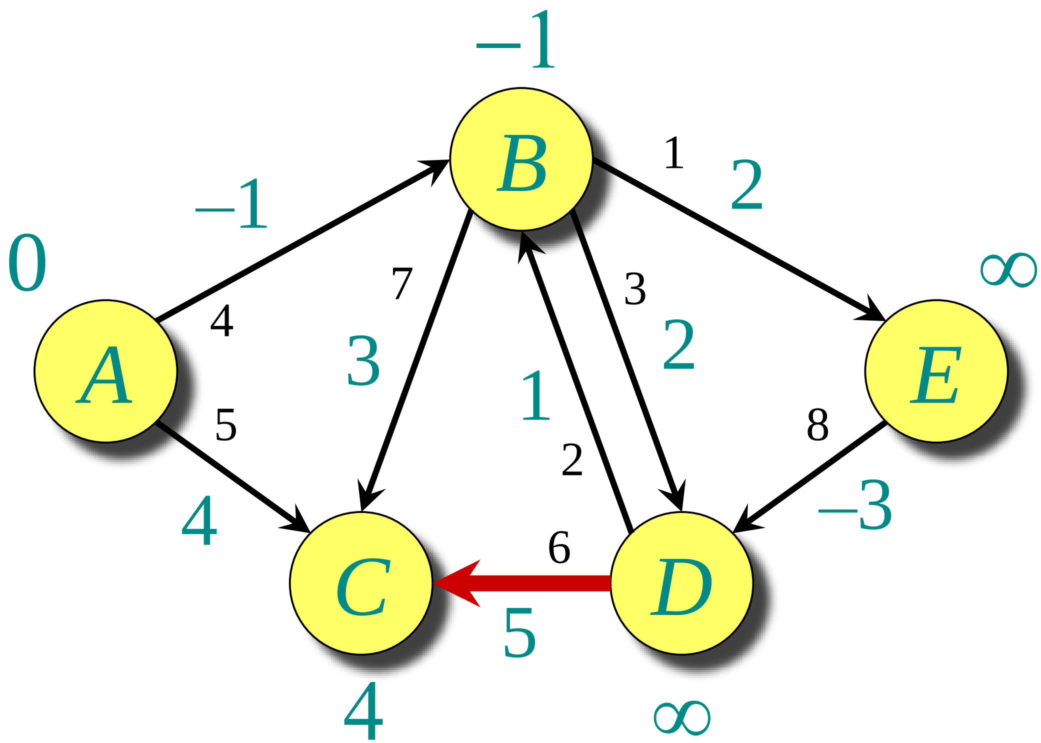
Example of Bellman-Ford



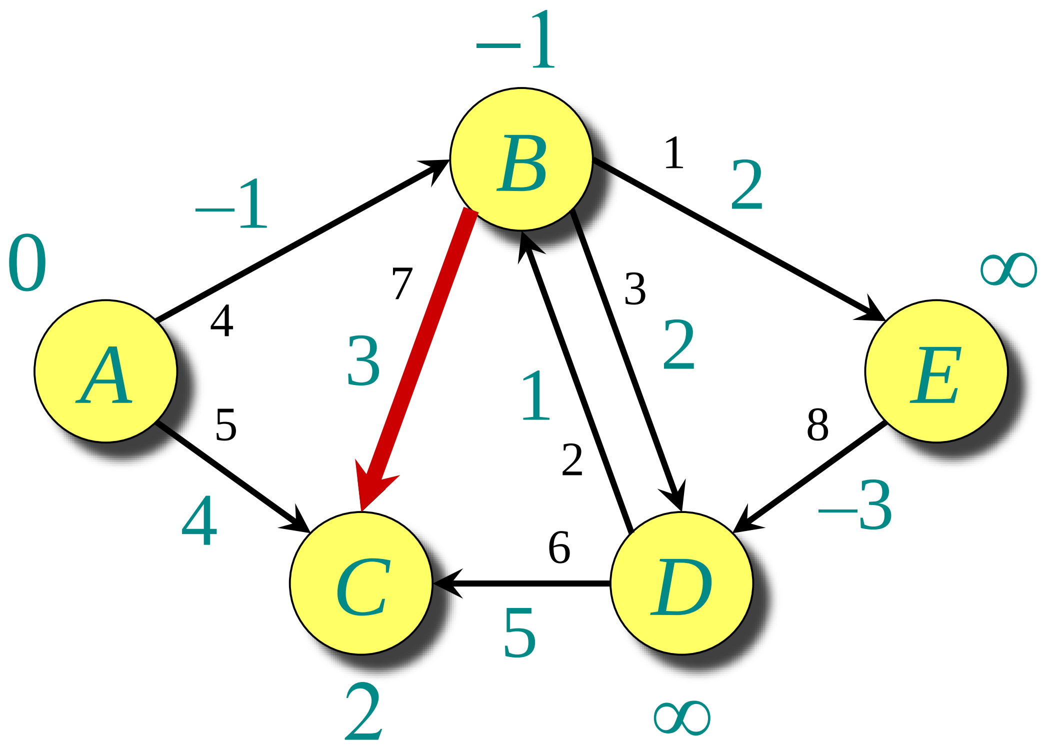
Example of Bellman-Ford



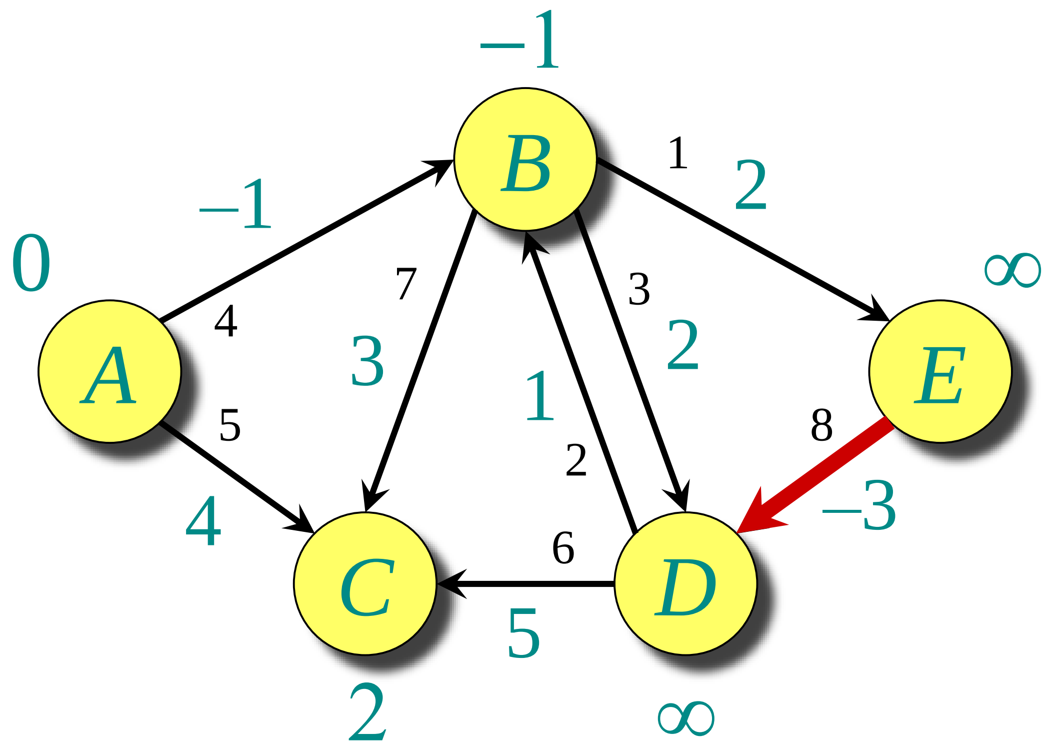
Example of Bellman-Ford



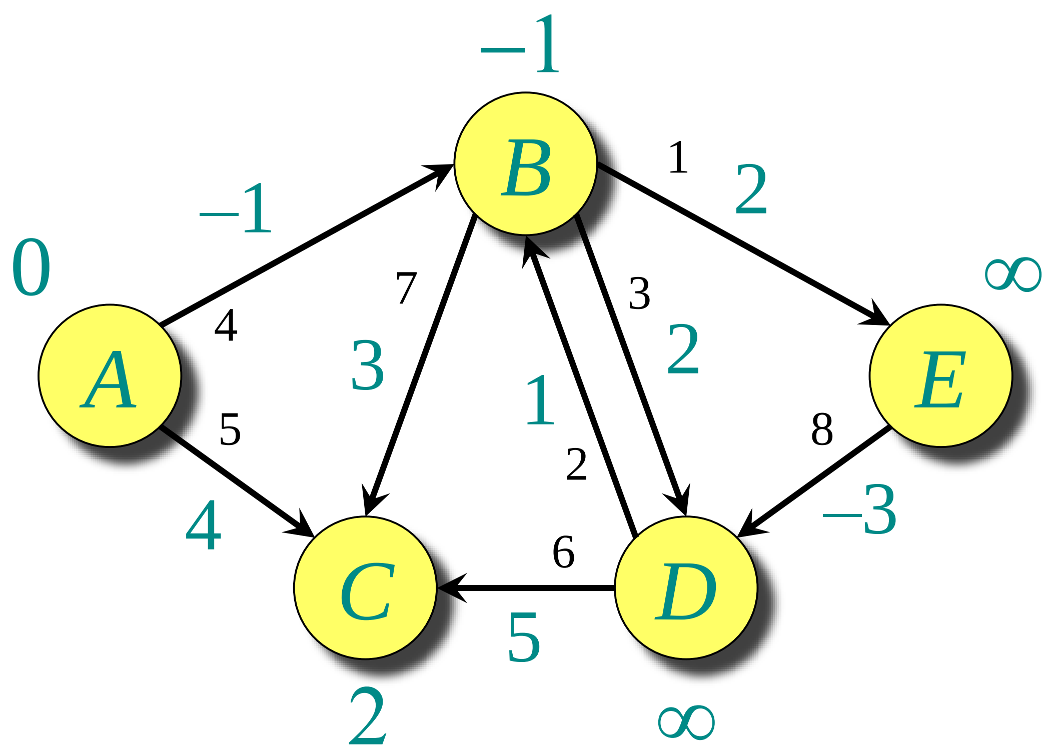
Example of Bellman-Ford



Example of Bellman-Ford

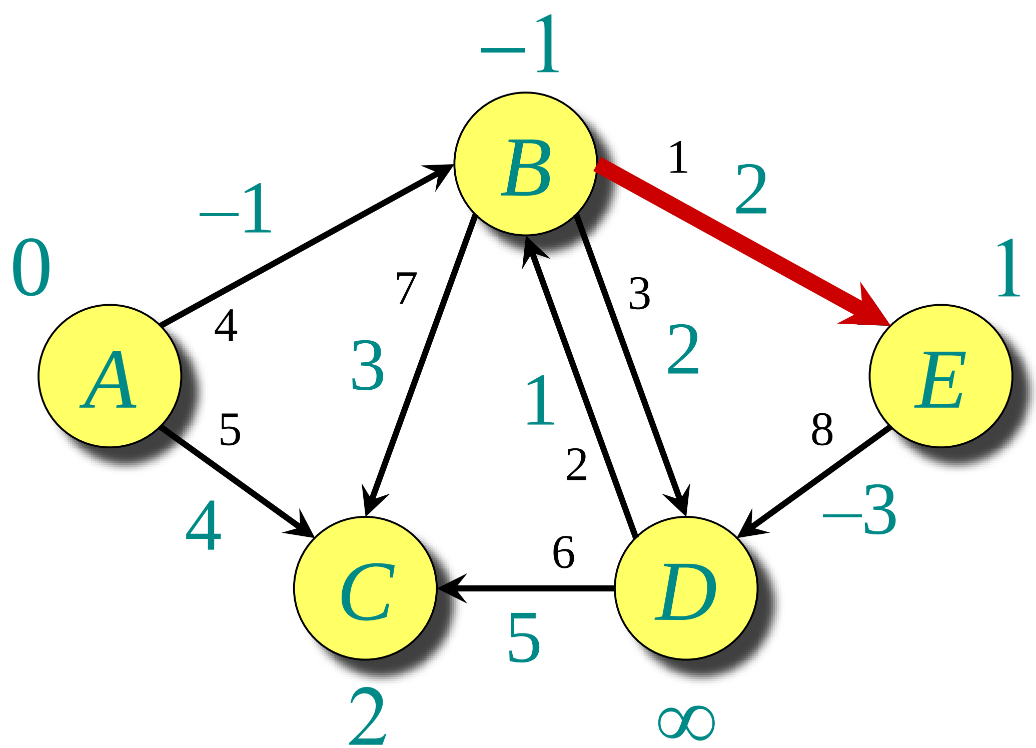


Example of Bellman-Ford

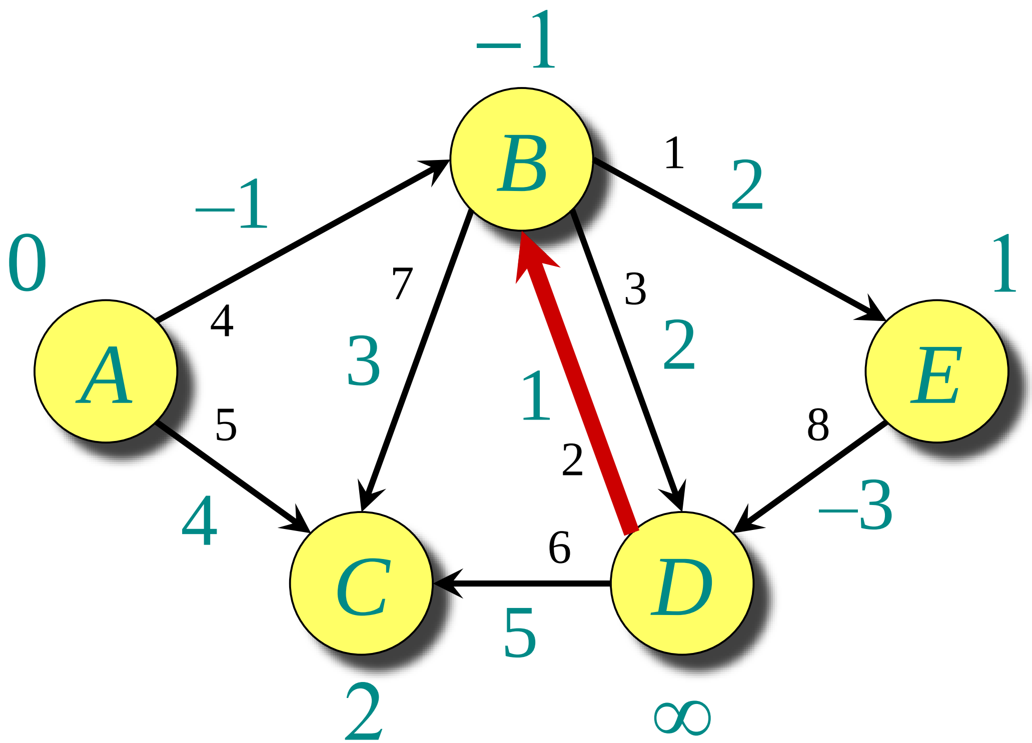


End of pass 1.

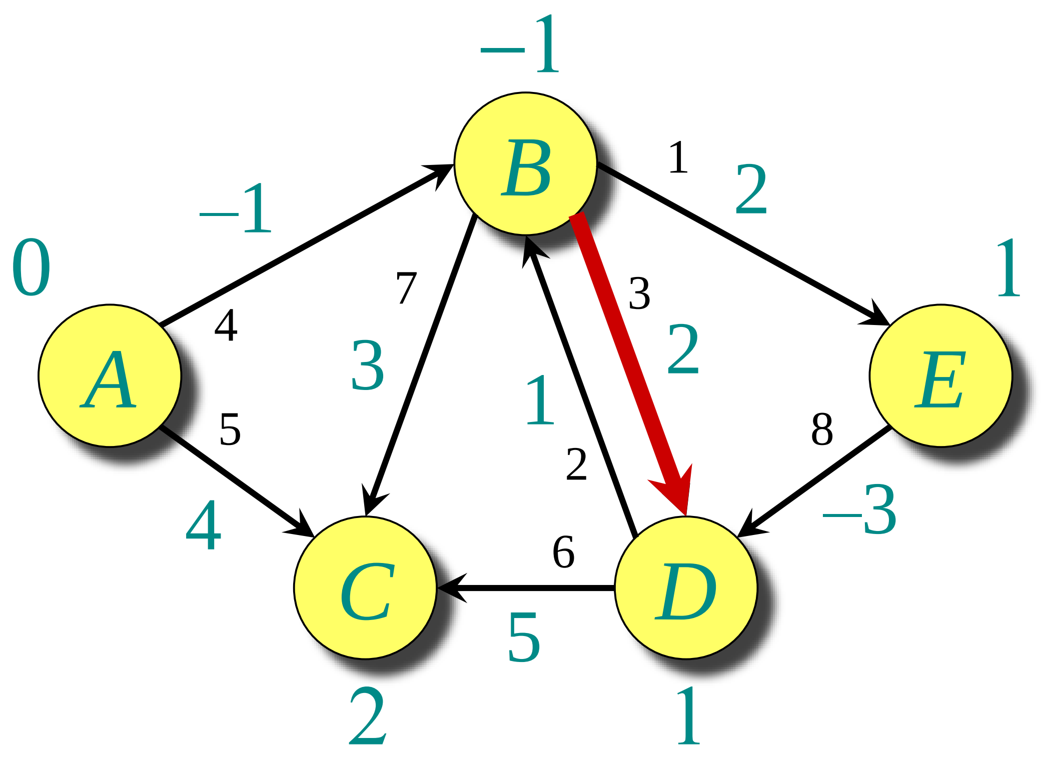
Example of Bellman-Ford



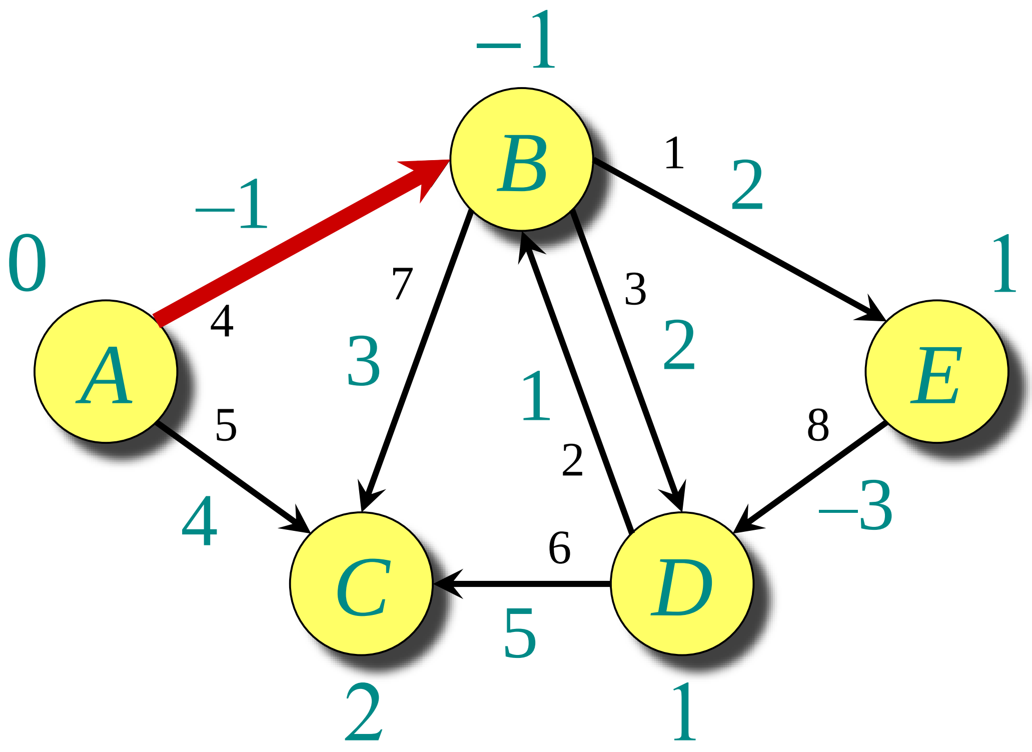
Example of Bellman-Ford



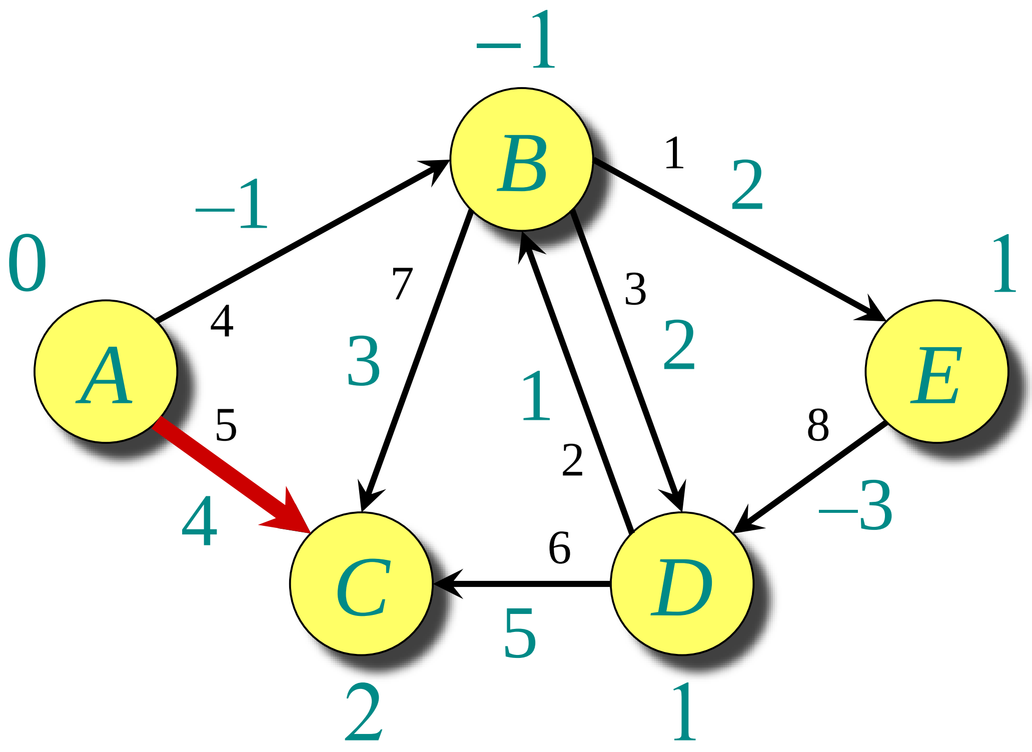
Example of Bellman-Ford



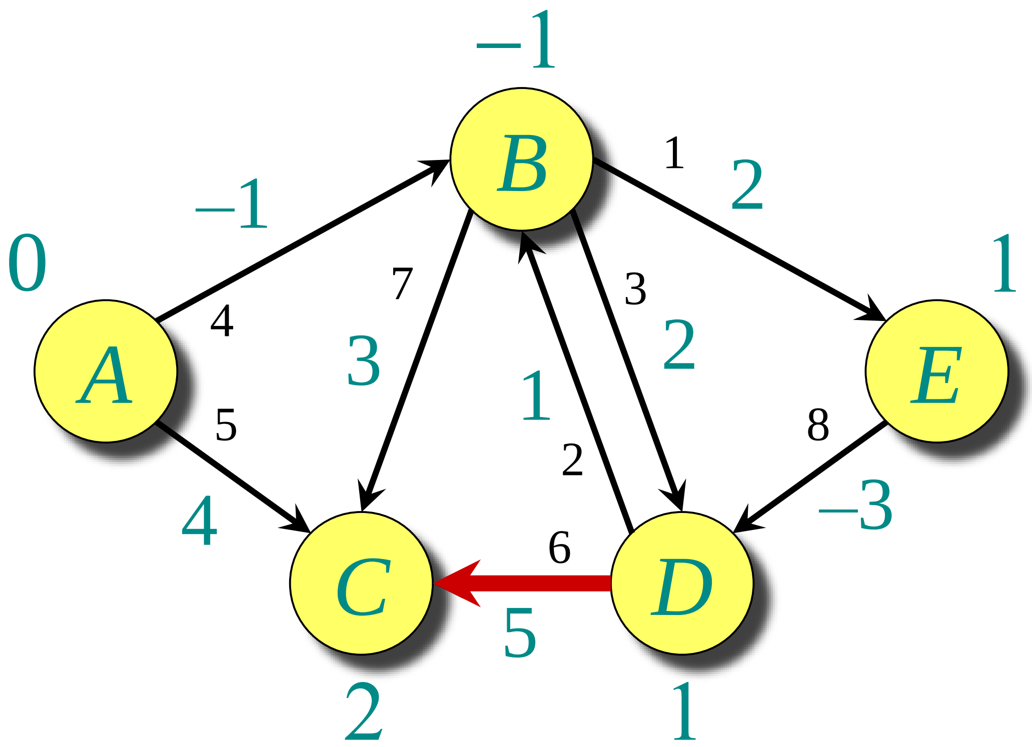
Example of Bellman-Ford



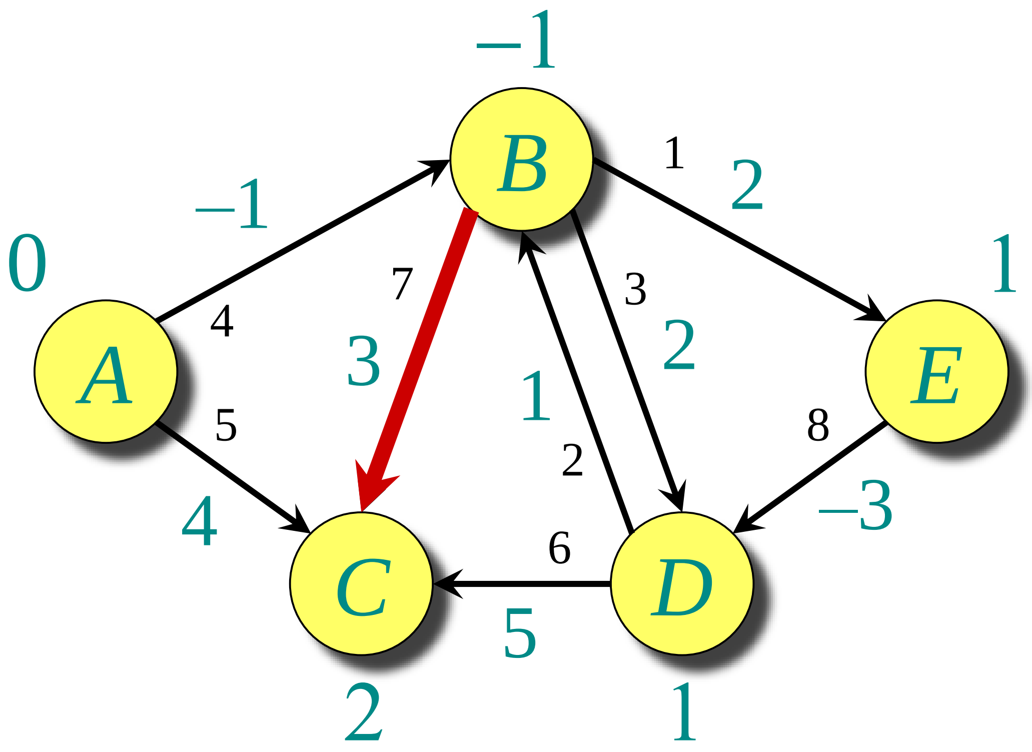
Example of Bellman-Ford



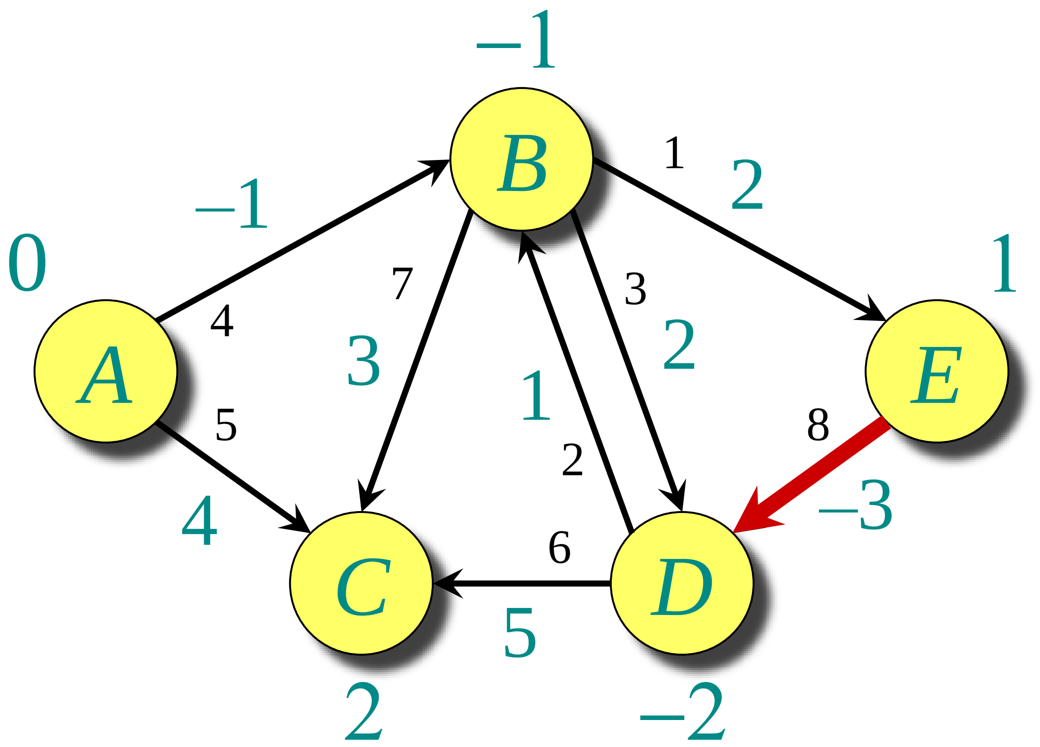
Example of Bellman-Ford



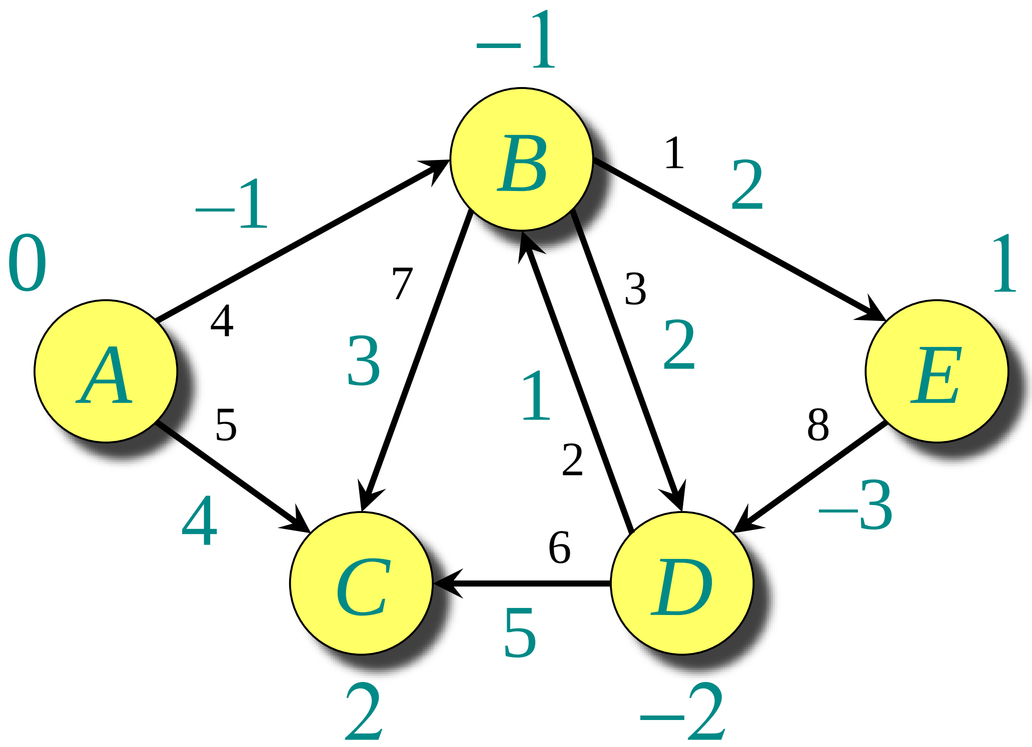
Example of Bellman-Ford



Example of Bellman-Ford



Example of Bellman-Ford



End of pass 2 (and 3 and 4).