Shortest Path II: Bellman-Ford

Instructor: Meng-Fen Chiang

COMPSCI: WEEK 11.4

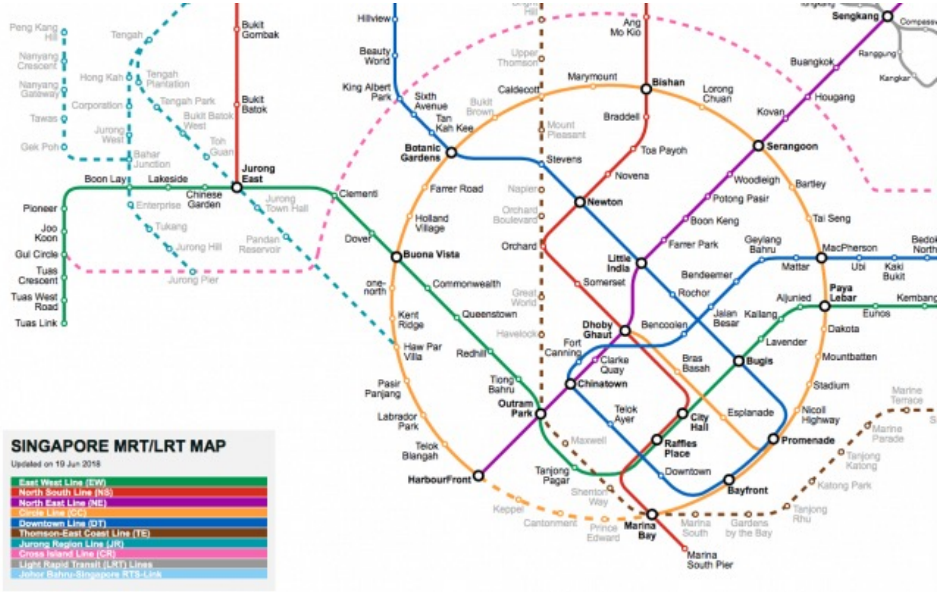
THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz, Tanya Gvozdeva, and Kaiqi Zhao

OUTLINE

- Algorithms on Weighted Graphs
  - Dijkstra
  - **Bellman-Ford**
  - Floyd-Warshall

- Time Complexity Analysis



Bellman-Ford Algorithm

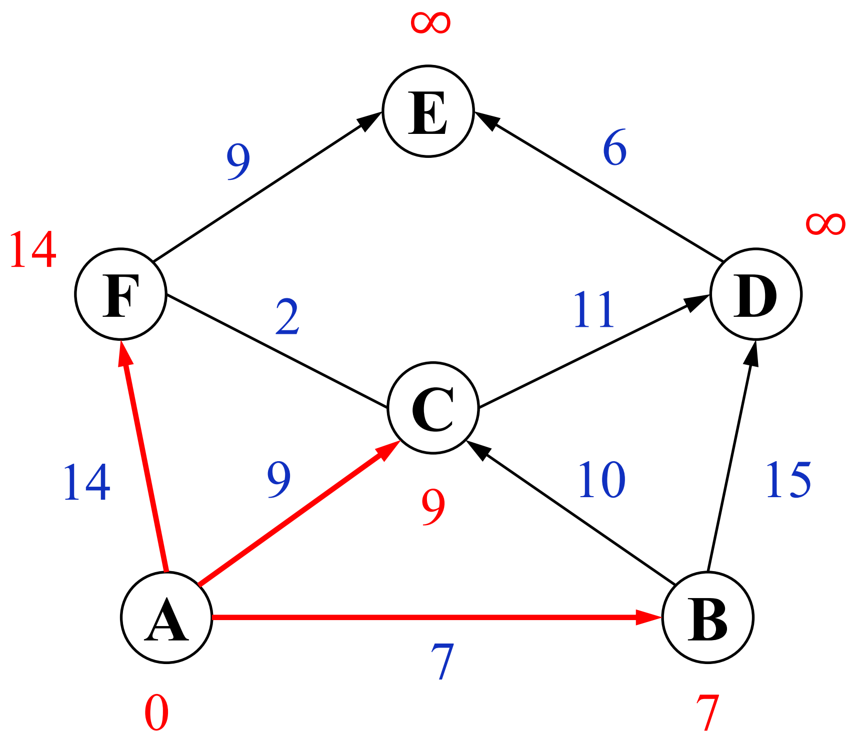
- Bellman-Ford can solve SSSP as well
- Slower than Dijkstra but can handle negative weights
- Bellman-ford performs at most n iterations, where n is the number of nodes/vertices

Bellman-Ford Algorithm

Algorithm 1 Bellman-Ford algorithm.

function BELLMANFORD(weighted digraph(G, c); node $s \in V(G)$)
    array dist$[0..n-1]$
    for $u \in V(G)$ **do**
        $dist[u] \leftarrow \infty$
    $dist[s] \leftarrow 0$
    for i **from** 0 **to** $n-1$ **do**
        for $x \in V(G)$ **do**
            for $v \in V(G)$ **do**
                $dist[v] \leftarrow \min\{dist[v], dist[x] + c[x, v]\}$
    return $dist$

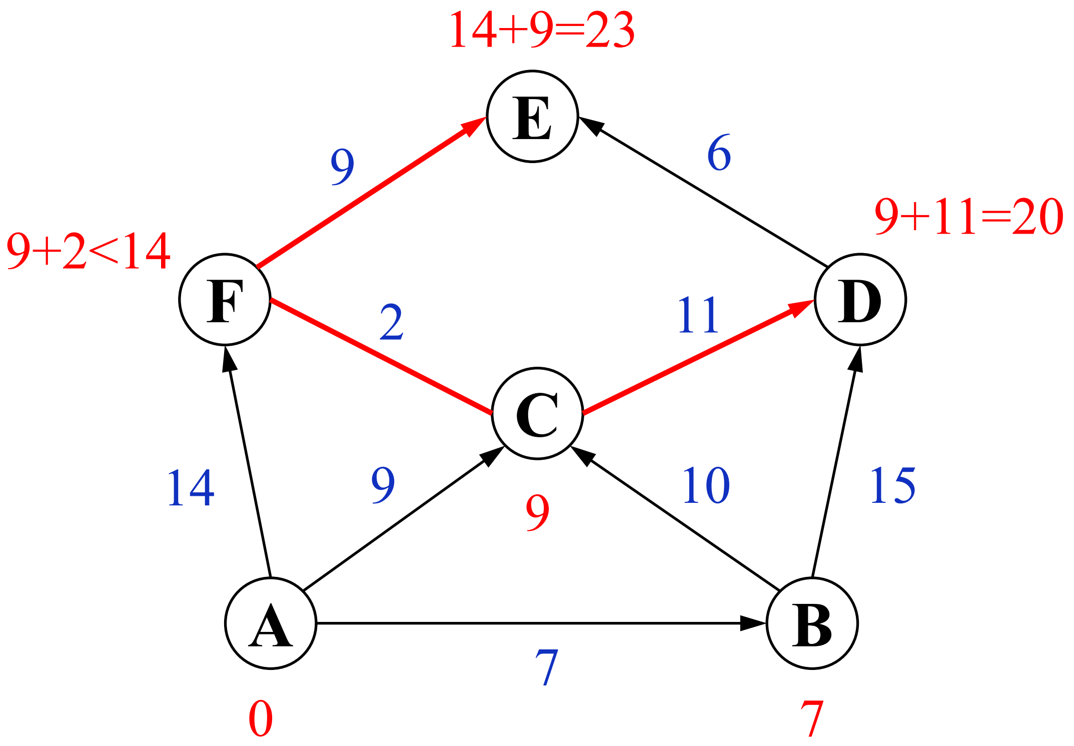
Bellman-Ford algorithm



Start at A

$i = 0$
$i = 1$

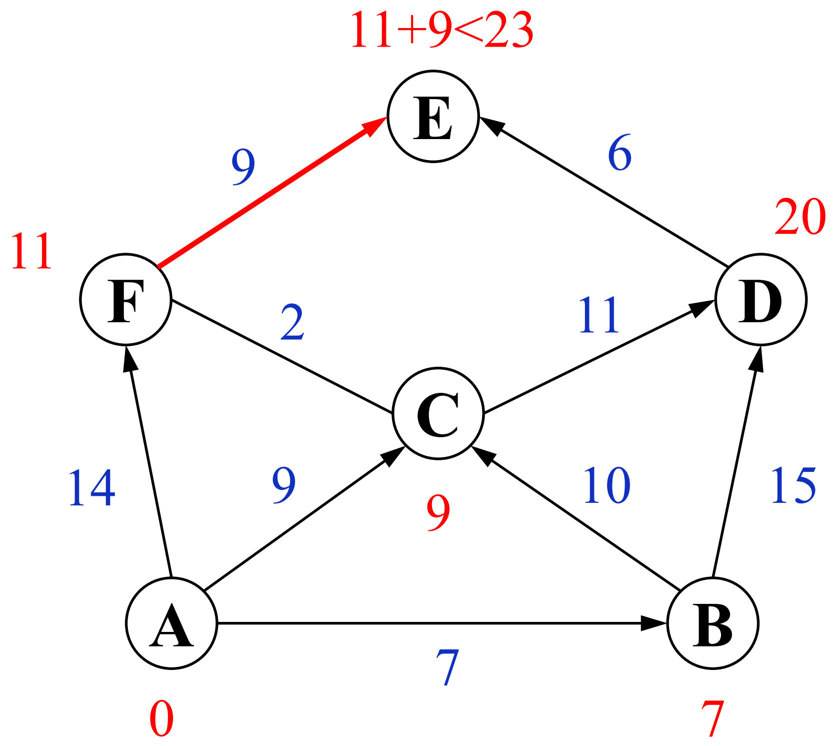
Bellman-Ford algorithm



Start at A

$i = 2$

Bellman-Ford algorithm



Start at A

$$i = 3$$
$$i = 4$$
$$\vdots$$
$$i = n - 1$$

Bellman-Ford Algorithm

- Slower than Dijkstra’s algorithm when all arcs are nonnegative.

- Similar idea as in Dijkstra’s: to find the single-source shortest paths(SSSP) under progressively relaxing restrictions.
  - Dijkstra’s: one node a time based on their current distance estimate.
  - Bellman-Ford: all nodes at “level” 0,1,…,n−1 in turn.
    - Level of a node v− the minimum possible number of arcs in a minimum weight path to that node from the source s.

Bellman-Ford Algorithm

- **Theorem**. If a graph G contains no negative weight cycles, then after the ith iteration of the outer for-loop, the element $dist[v]$ contains the minimum weight of a path to v for all nodes v with level at most i.

Why Bellman-Ford algorithm Works

Just as for Dijkstra's, the update ensures $dist[v]$ never increases.

Induction by the level i of the nodes:

- **Base case**: i=0; the result is true due to initialization: $dist[s] = 0; dist[v] = \infty;\ v \in V\backslash s$.
- **Induction hypothesis**: $dist[v]$; $v \in V$, are true for $i-1$.
- **Induction step** for a node v at level i:
- Due to no negative weight cycles, a min-weight s-to-v path, γ, has i arcs.
- If y is the last node before v and γ_1 the subpath to y, then $dist[y] \le |\gamma_1|$ by the induction hypothesis.
- Thus by the update rule: $dist[v] \le dist[y] + c(y, v) \le |\gamma_1| + c(y, v) \le |\gamma|$

  as required at level i.

update formula **IH**

Bellman-Ford Algorithm

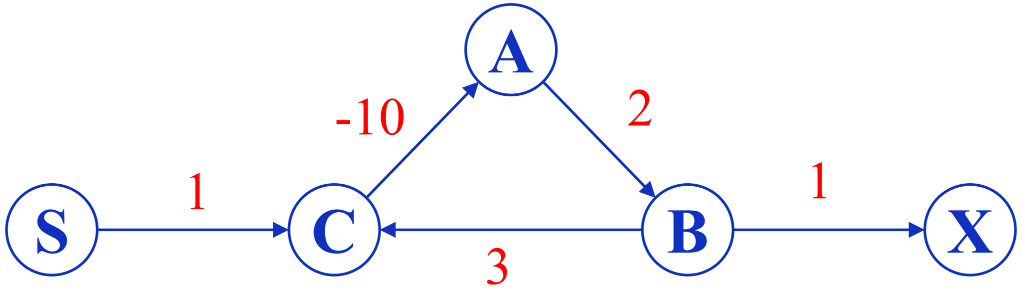
- **Fact**. This (non-greedy) algorithm handles negative weight arcs but not negative weight cycles.

- Runs in time $O(nm)$ since the two inner-most for loops can be replaced with: for(x, v)$\in E(V)$.

- Can be modified to detect negative weight cycle.

Cycles of Negative Weights

- SSSP problem makes no sense if we allow digraphs with cycles of negative total weight.



Path from S to X

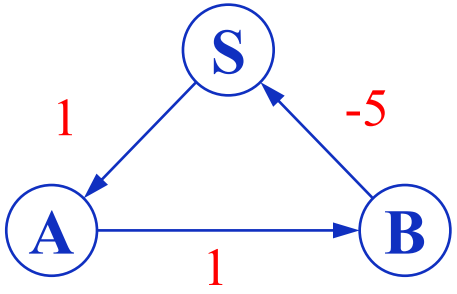
S-C-A-B-X : $1 + (-10) + 2 + 1 = -6$

S-C-A-B-C-A-B-X : $1 + (-10) + 2 + 3 + (-10) + 2 + 1 = -11$

Cycles of Negative Weights (Contd.)

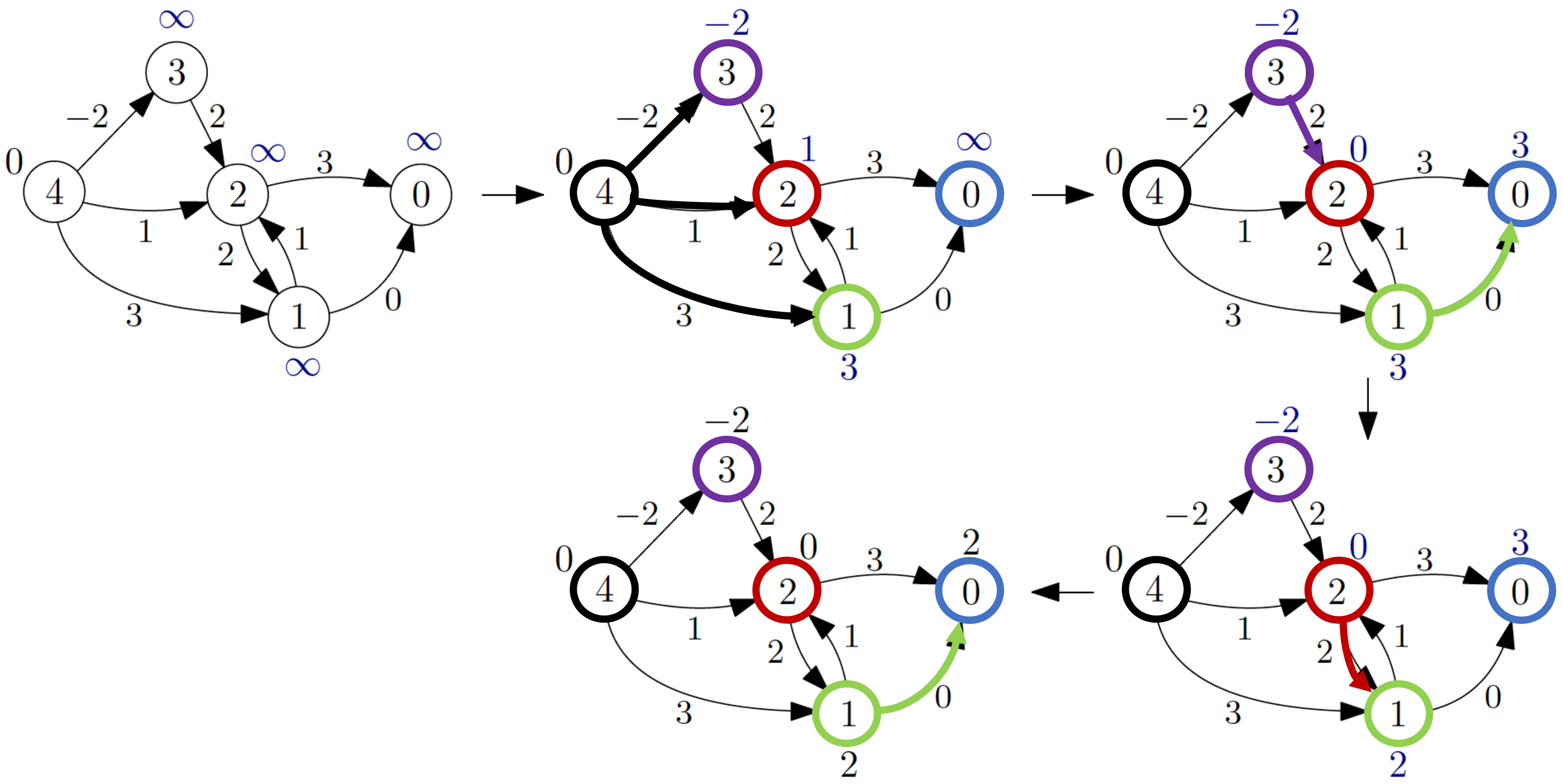
- Suppose the input to the Bellman–Ford algorithm is a digraph with a negative weight cycle. How could the algorithm detect this, so it can exit gracefully with an error message?

Run outer *for* loop for one more iteration. If $dist[v]$ changes for some vertex v in the last iteration, then the graph has a negative weight cycle.

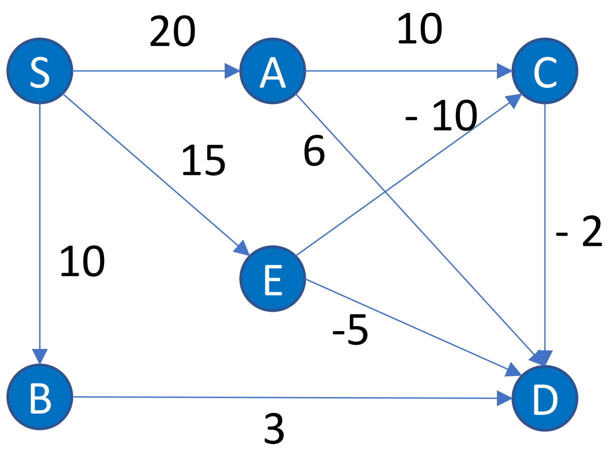


1st iteration : $dist[s] = -3$
2nd iteration : $dist[s] = -6$
3rd iteration : $dist[s] = -9$

Example. An application of Bellman–Ford algorithm with starting node 4 when the nodes are processed in the order from 0 to 4.

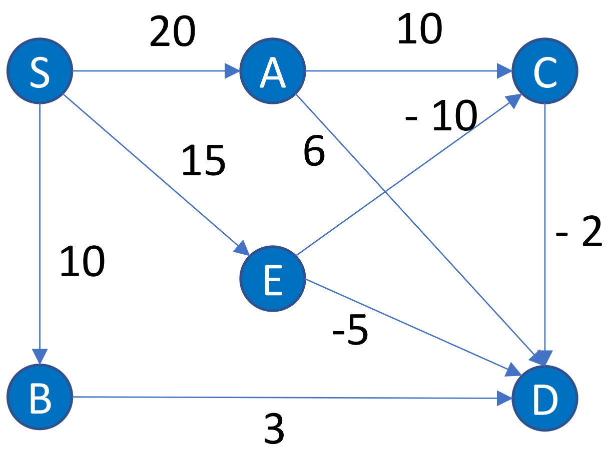


Example: Bellman-Ford at Work



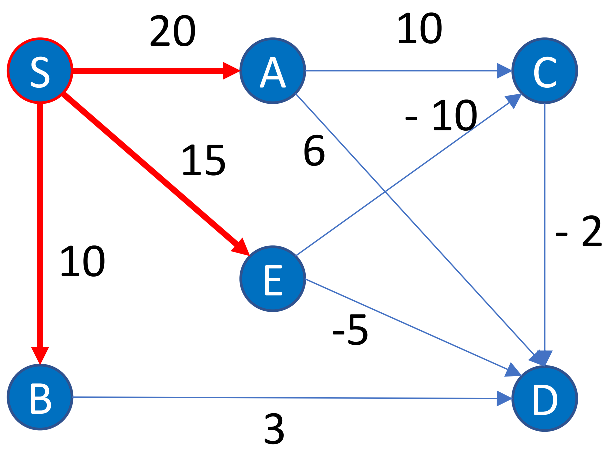
We have 6 vertices which means that at most we will do 5 iterations

Example: Bellman-Ford at Work



S	0, S
A	∞
B	∞
C	∞
D	∞
E	∞

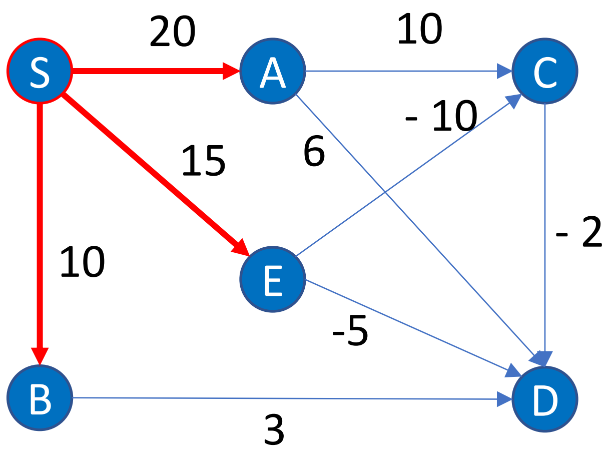
Example: Bellman-Ford at Work



S	0, S
A	∞
B	∞
C	∞
D	∞
E	∞

1st Iteration

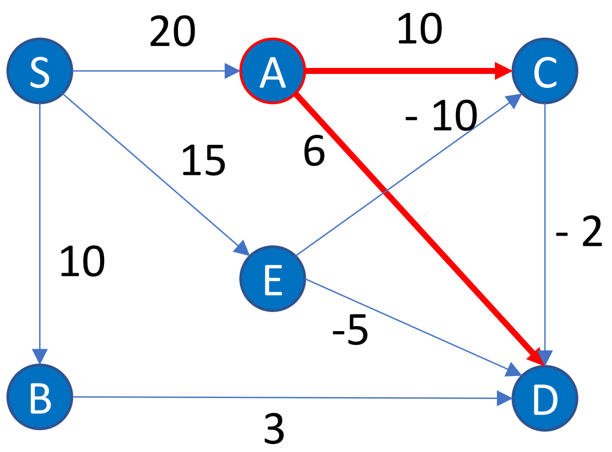
Example: Bellman-Ford at Work



S	0, S
A	20, S
B	10, S
C	∞
D	∞
E	15, S

1st Iteration

Example: Bellman-Ford at Work

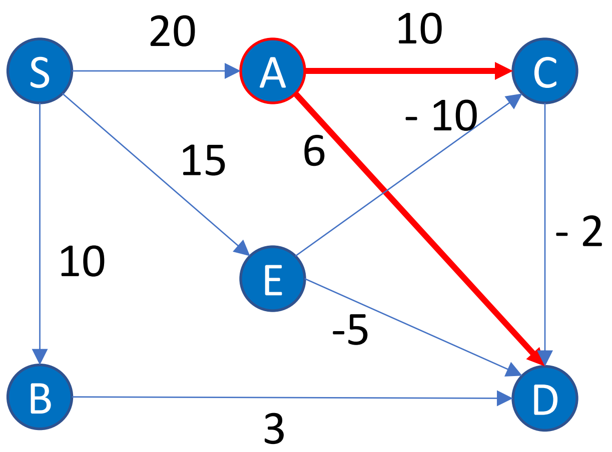


S	0, S
A	20, S
B	10, S
C	∞
D	∞
E	15, S

1st Iteration

From S we can get to A with a cost of 20
From A we can get to C with a cost of 10
So we can get from A to C with a total cost of 30

Example: Bellman-Ford at Work

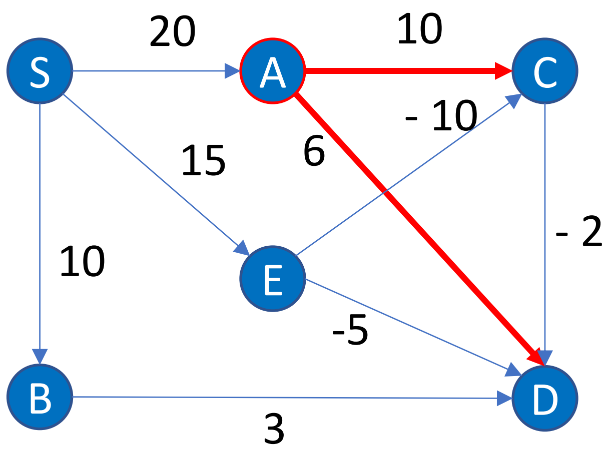


S	0, S
A	20, S
B	10, S
C	30, A
D	∞
E	15, S

1st Iteration

From S we can get to A with a cost of 20
From A we can get to C with a cost of 10
So we can get from A to C with a total cost of 30

Example: Bellman-Ford at Work

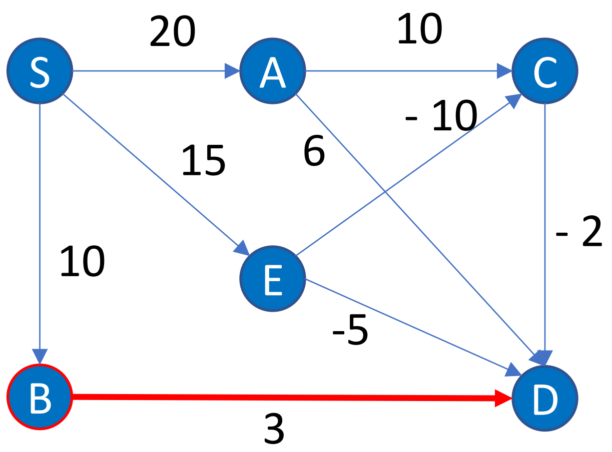


S	0, S
A	20, S
B	10, S
C	30, A
D	26, A
E	15, S

1st Iteration

Similarly, we can reach D from S through A with a total cost of 26

Example: Bellman-Ford at Work

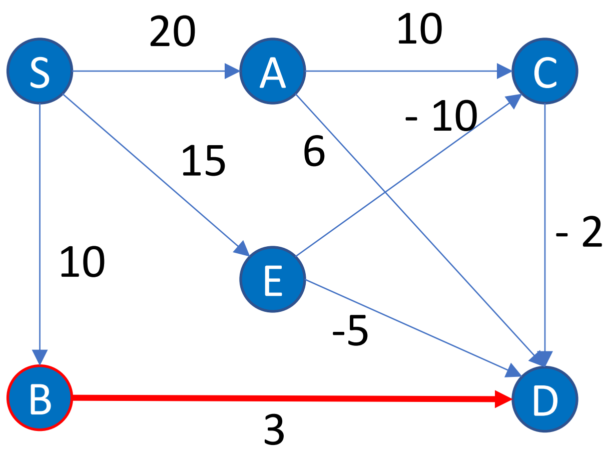


S	0, S
A	20, S
B	10, S
C	30, A
D	26, A
E	15, S

1st Iteration

We know that we can reach B from S with a cost of 10.
From B we can reach D with a cost 3.
So via B, the total cost is 13 which is less than the current total cost from S to D via A

Example: Bellman-Ford at Work

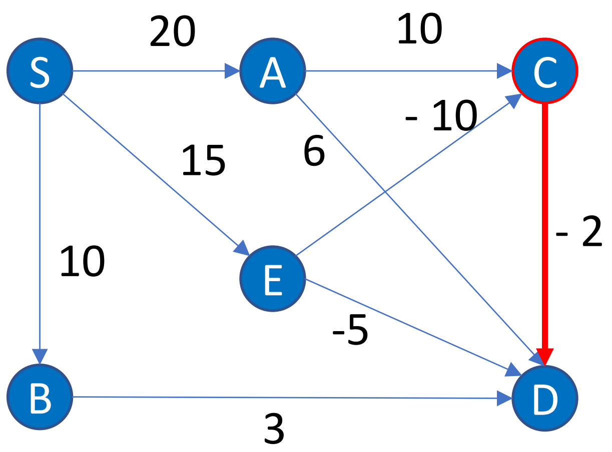


S	0, S
A	20, S
B	10, S
C	30, A
D	13, B
E	15, S

1st Iteration

We update D entry with the new total cost

Example: Bellman-Ford at Work

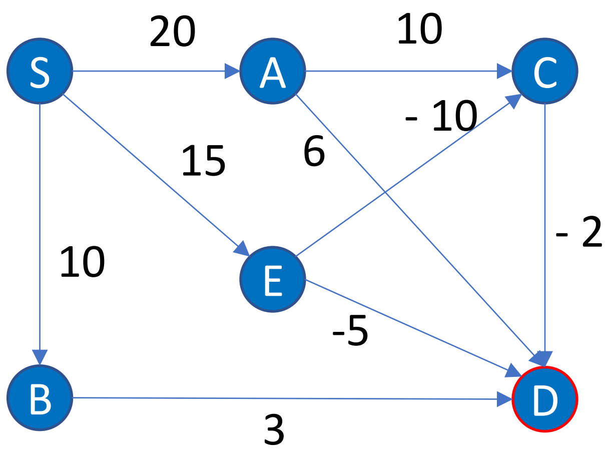


S	0, S
A	20, S
B	10, S
C	30, A
D	13, B
E	15, S

1st Iteration

From S we can reach C with cost 30
Via C, we can reach D from S with a total cost of 28 (30 – 2)
But because the current total cost to D (13) is less than this new value via C we do not update it

Example: Bellman-Ford at Work

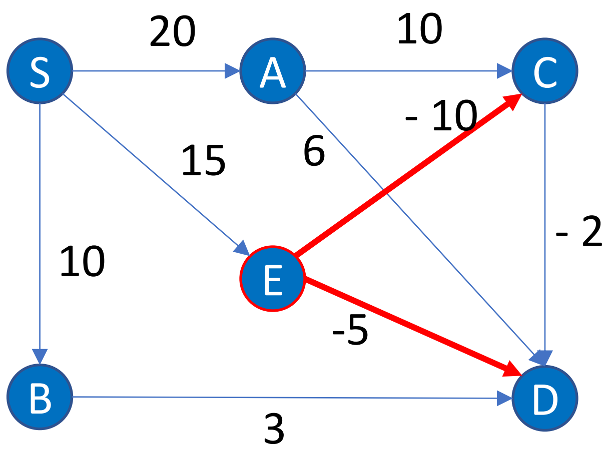


S	0, S
A	20, S
B	10, S
C	30, A
D	13, B
E	15, S

1st Iteration

D is a sink so we just skip it

Example: Bellman-Ford at Work

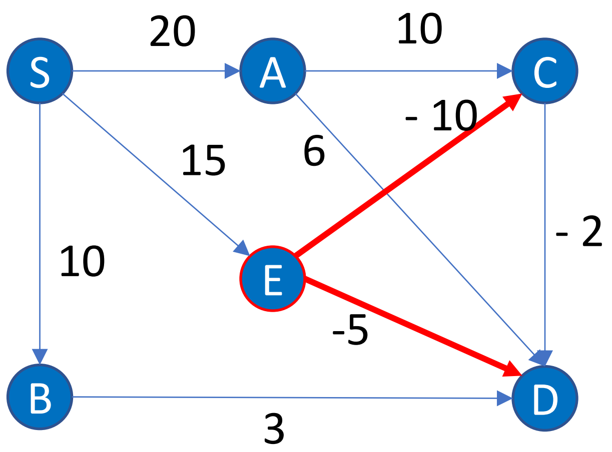


S	0, S
A	20, S
B	10, S
C	30, A
D	13, B
E	15, S

1st Iteration

From E we can reach C with cost -10 and D with cost -5

Example: Bellman-Ford at Work

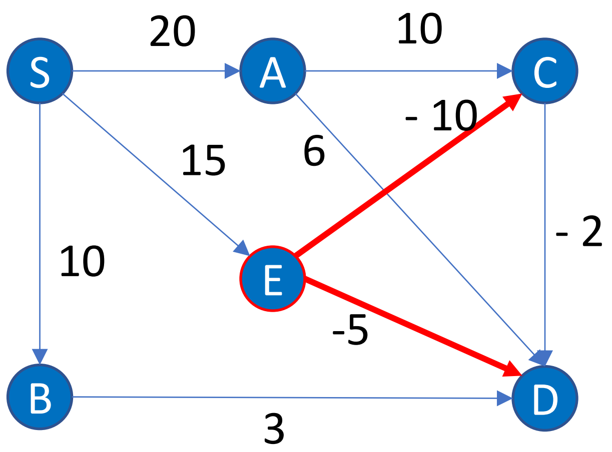


S	0, S
A	20, S
B	10, S
C	30, A
D	13, B
E	15, S

1st Iteration

This means that the total cost from S to C via E is 5 so we can update the value for C in the table

Example: Bellman-Ford at Work

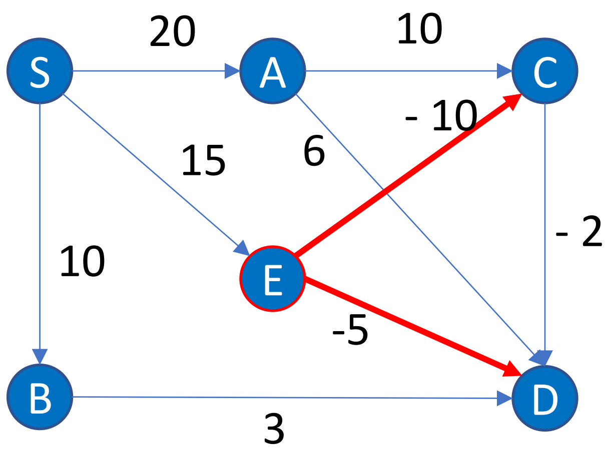


S	0, S
A	20, S
B	10, S
C	5, E
D	13, B
E	15, S

1st Iteration

This means that the total cost from S to C via E is 5 so we can update the value in the table

Example: Bellman-Ford at Work

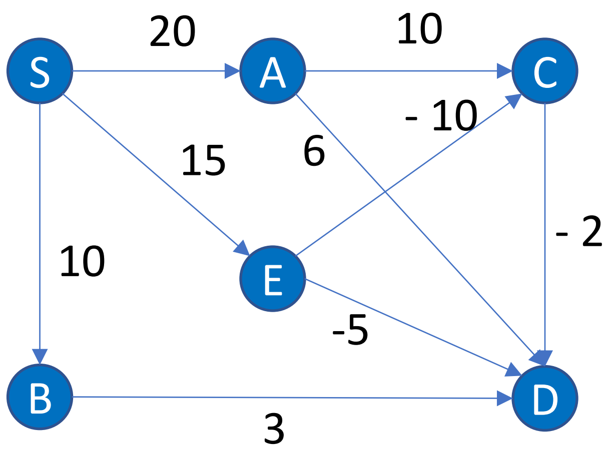


S	0, S
A	20, S
B	10, S
C	5, E
D	10, E
E	15, S

1st Iteration

The total cost from S to D via E is 10 so we can also update D.

Example: Bellman-Ford at Work

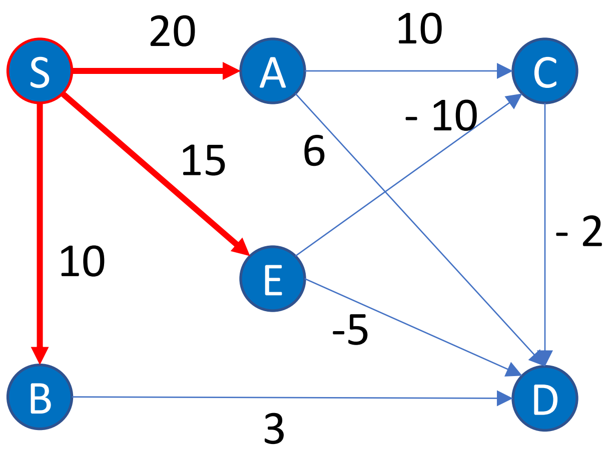


S	0, S
A	20, S
B	10, S
C	5, E
D	10, E
E	15, S

1st Iteration

Our first iteration is now concluded. We can move to our second iteration.

Example: Bellman-Ford at Work

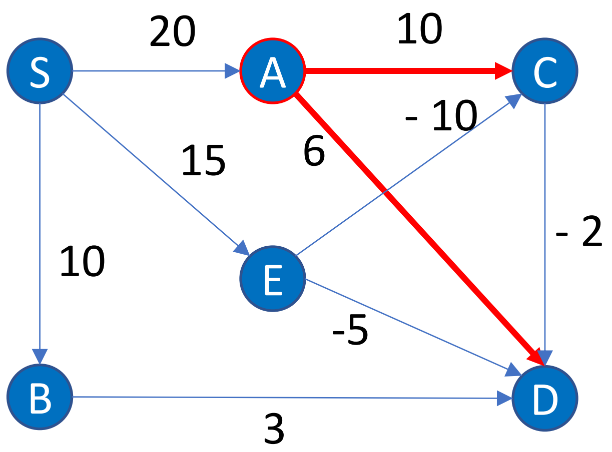


S	0, S
A	20, S
B	10, S
C	5, E
D	10, E
E	15, S

2nd Iteration

We start again from S and we see that we cannot improve the costs for A, B and E.

Example: Bellman-Ford at Work

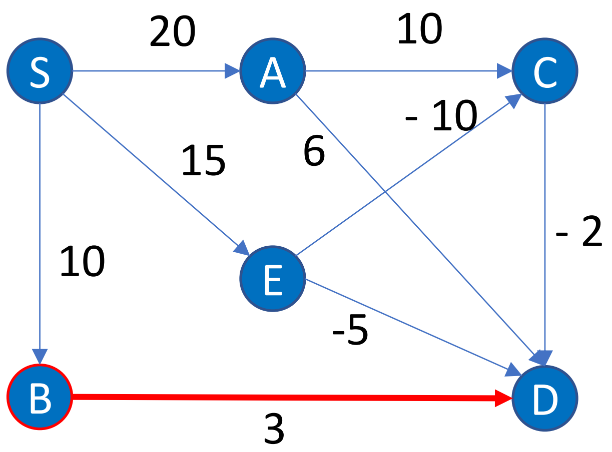


S	0, S
A	20, S
B	10, S
C	5, E
D	10, E
E	15, S

2nd Iteration

We select A and we can see that we can reach C and D. But again we cannot do better than what we have already in the table

Example: Bellman-Ford at Work

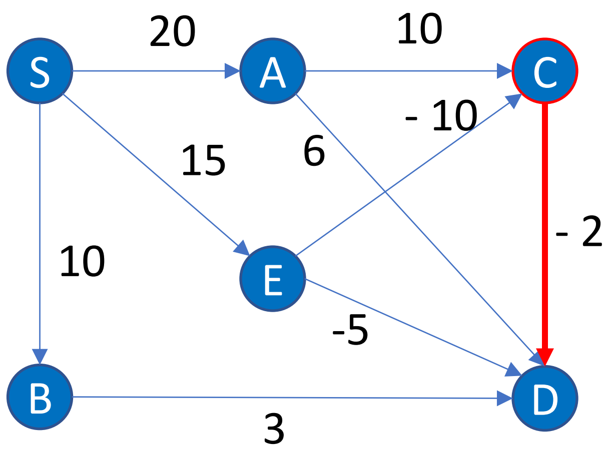


S	0, S
A	20, S
B	10, S
C	5, E
D	10, E
E	15, S

2nd Iteration

We select B and from B we can reach D. But again we cannot do better than what is in the table.

Example: Bellman-Ford at Work

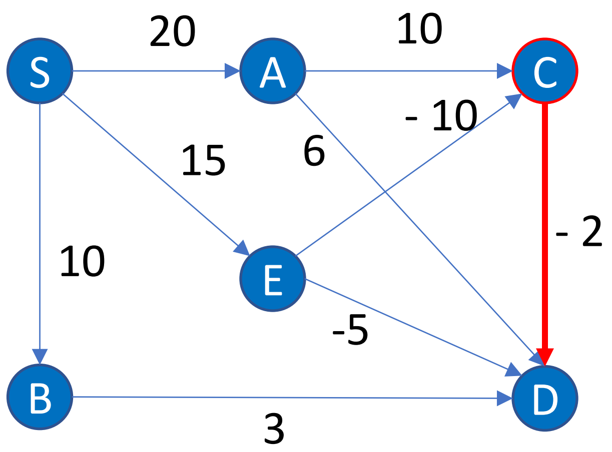


S	0, S
A	20, S
B	10, S
C	5, E
D	10, E
E	15, S

2nd Iteration

From C we can reach D with a cost -2.
The cost from S to C is 5. So the total cost from S to D through C is 3.
This is better than what we have in the table so we update D cost .

Example: Bellman-Ford at Work

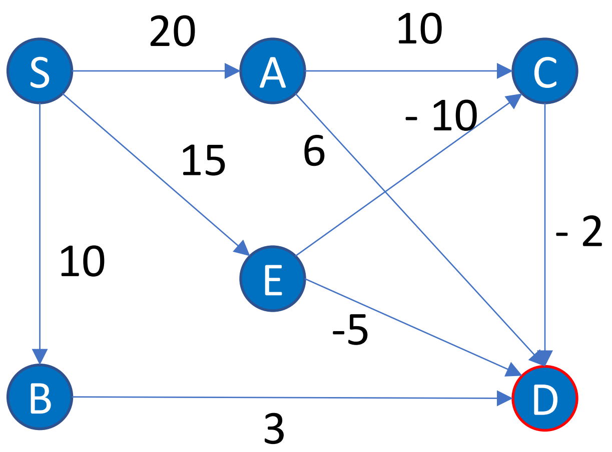


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

2nd Iteration

From C we can reach D with a cost -2.
The cost from S to C is 5. So the total cost from S to D through C is 3.
This is better than what we have in the table so we update the cost of D.

Example: Bellman-Ford at Work

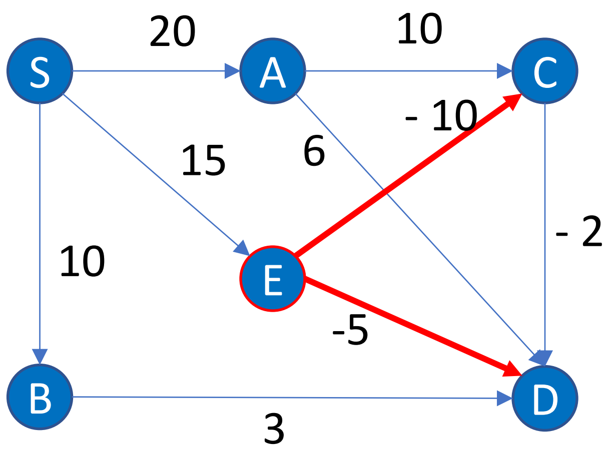


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

2nd Iteration

We move on to D. But again we cannot reach other nodes from D.

Example: Bellman-Ford at Work

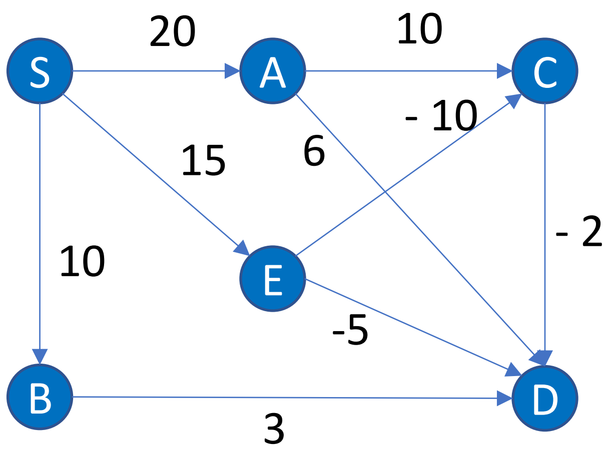


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

2nd Iteration

We move on to E. From E we can reach C and D.
But again we cannot do better than what in the table so no update needed

Example: Bellman-Ford at Work

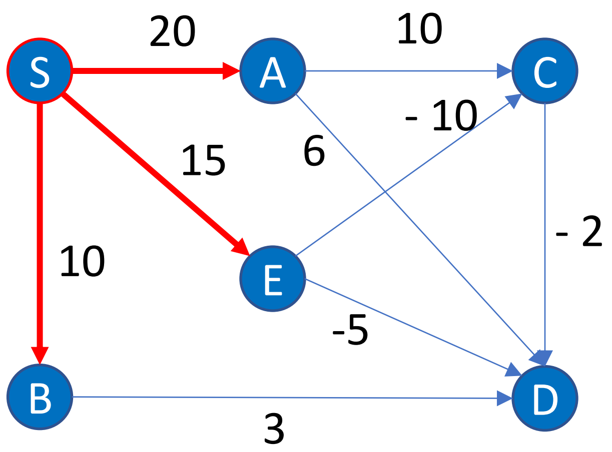


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

2nd Iteration

This concludes our second iteration. We move on to the third.

Example: Bellman-Ford at Work

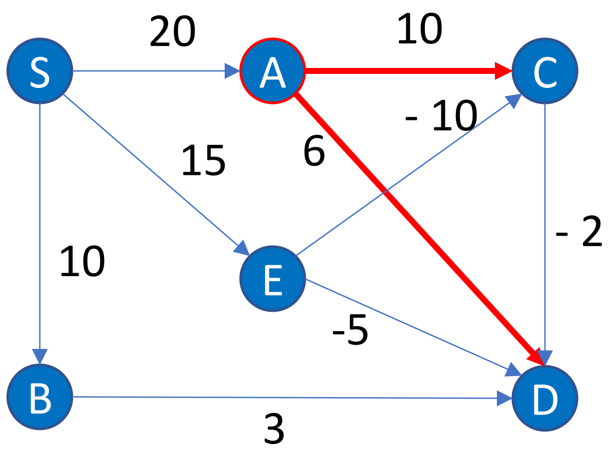


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

3rd Iteration

We start the iteration again from S. And again we cannot do better than what in the table.

Example: Bellman-Ford at Work

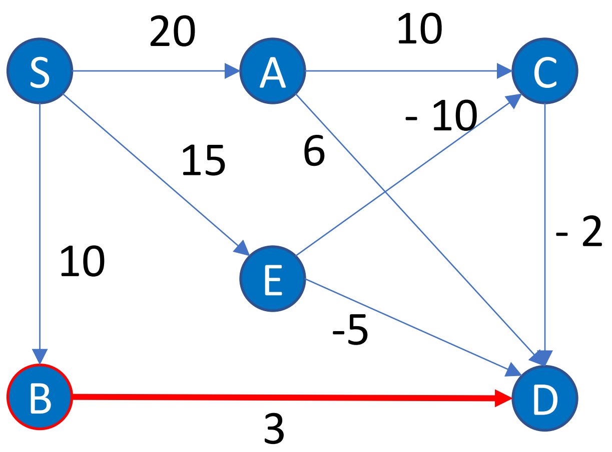


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

3rd Iteration

We move on to A and also in this case we cannot do better so we move on

Example: Bellman-Ford at Work

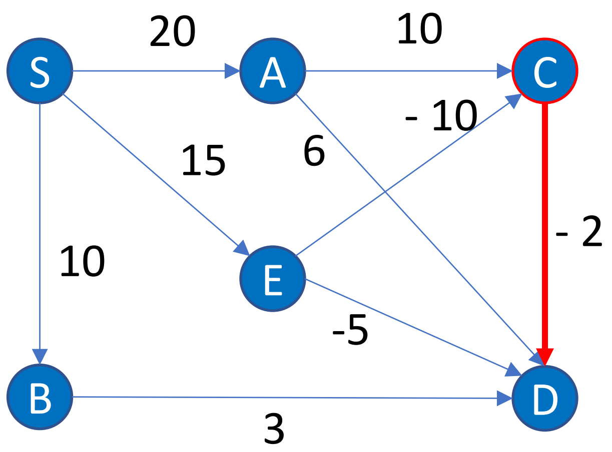


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

3rd Iteration

We select B and also here we cannot do better.

Example: Bellman-Ford at Work

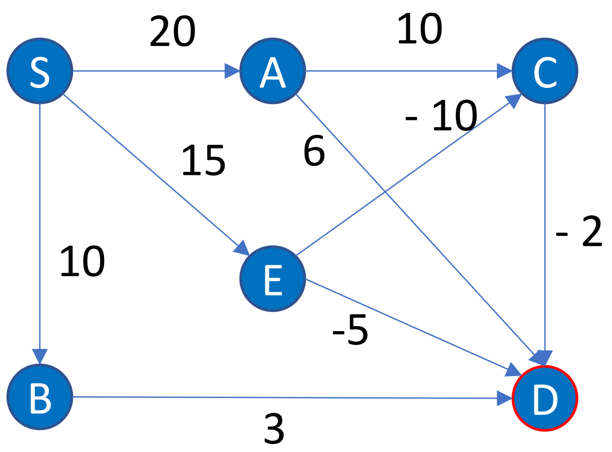


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

3rd Iteration

We select C and also in this case we cannot improve so we move on

Example: Bellman-Ford at Work

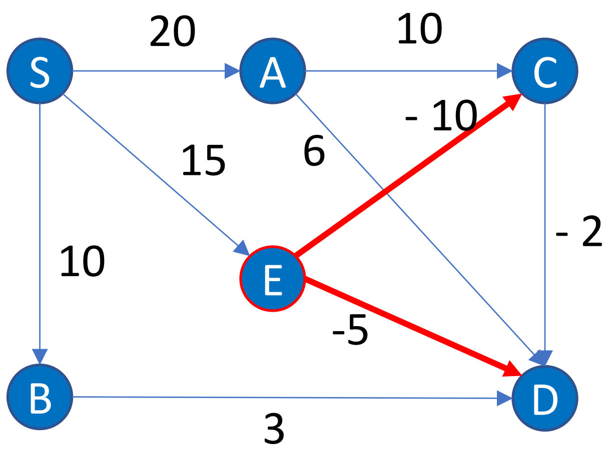


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

3^{rd} Iteration

We select D but we cannot reach other nodes. So we move on

Example: Bellman-Ford at Work

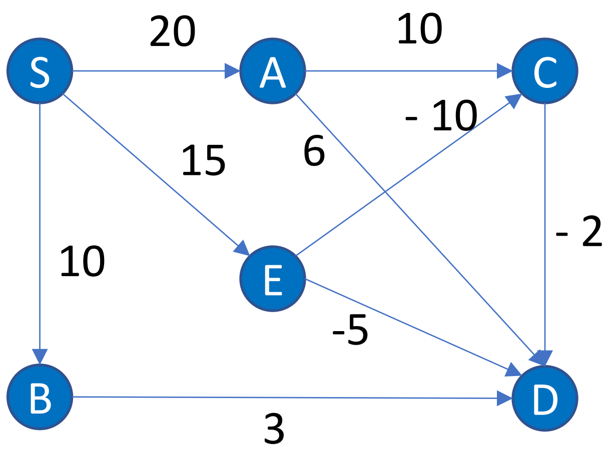


S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

3rd Iteration

Finally we select E. Again we cannot improve.
Because during this last iteration, our table has not changed, so we can **stop** here.

Example: Bellman-Ford at Work



S	0, S
A	20, S
B	10, S
C	5, E
D	3, C
E	15, S

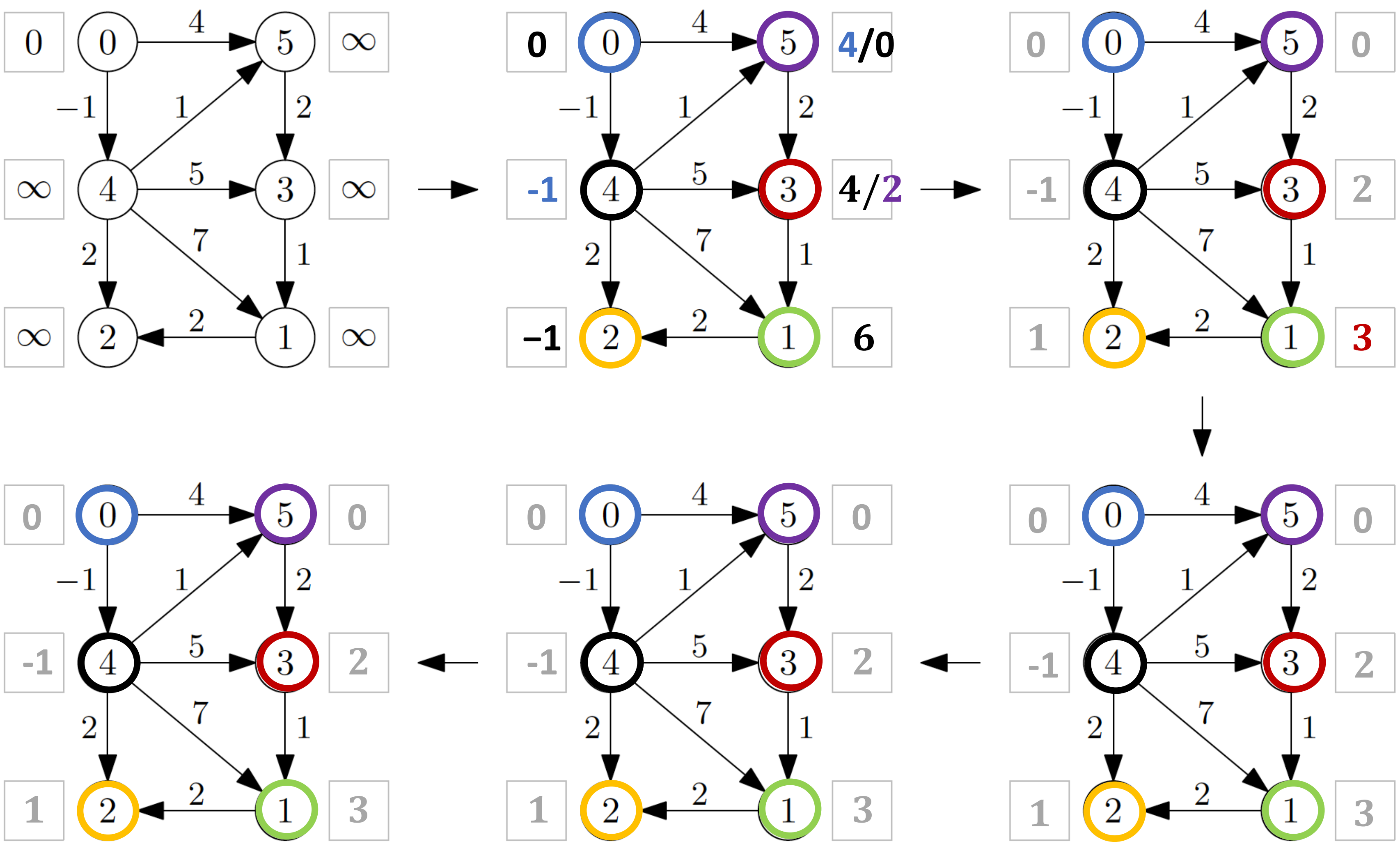
3rd Iteration

The costs in the table represent the best total costs from S to any other nodes in the digraph

Time Complexity: Bellman-Ford (Contd.)

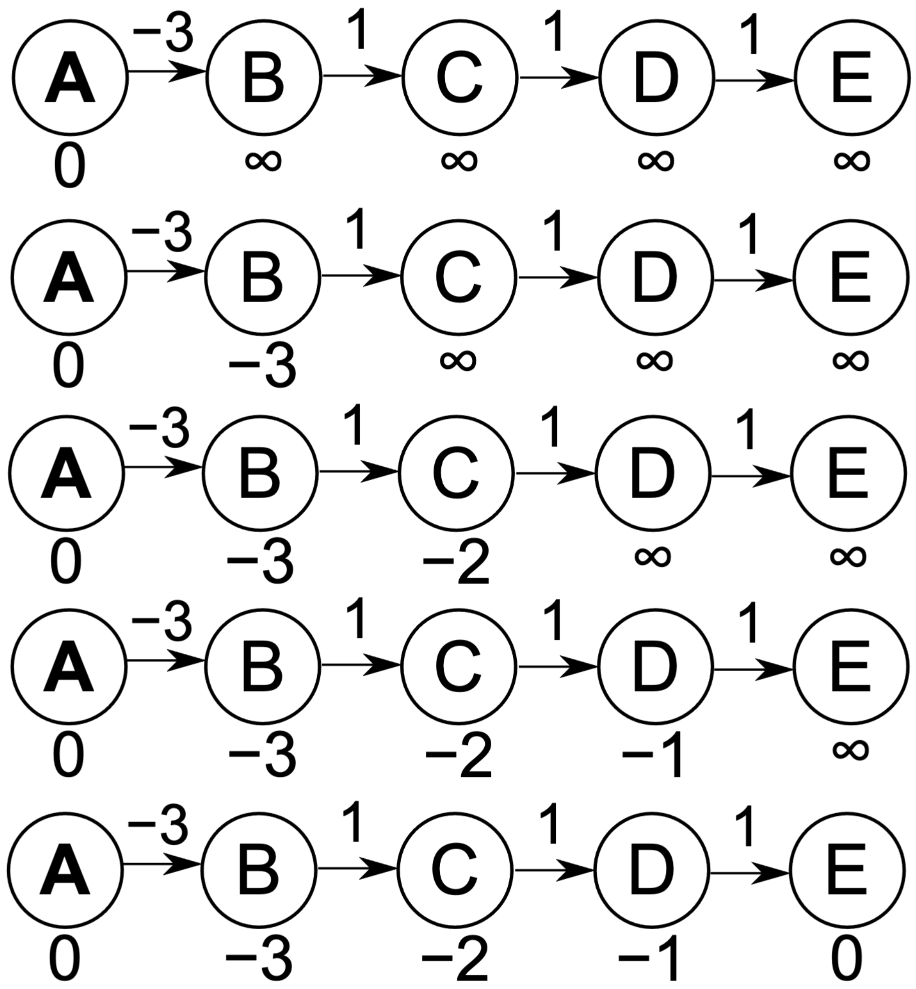
- For sparse graphs and adjacency lists: $\Theta(n)\Theta(m) = \Theta(nm)$
- For dense graphs we have $\Theta(m) = O(n^2)$, so Bellman-Ford is $\Theta(n)O(n^2) = O(n^3)$
- With an adjacency matrix: $\Theta(n^3)$.
- Conclusion: Dijkstra is faster but doesn't give the right answers when we have negative weight edges/arcs

Example. Execute the Bellman–Ford algorithm on the graph below with starting vertex 0. Process nodes in the order <u>from 0 to 5</u>.



- In this example graph, assuming that **A** is the source and edges are processed in the worst order, from right to left, it requires the full |V|−1 or 4 iterations for the distance estimates to converge.

- Conversely, if the edges are processed in the best order, from left to right, the algorithm converges in a single iteration. *Source – Wikipedia*



SUMMARY

- Algorithms on Weighted Graphs
  - Dijkstra
  - **Bellman-Ford**
  - Floyd-Warshall
- Time Complexity Analysis