CSE 202
Algorithms II

Week 10

Baris ARSLAN
Department of Computer Engineering
Spring 2024

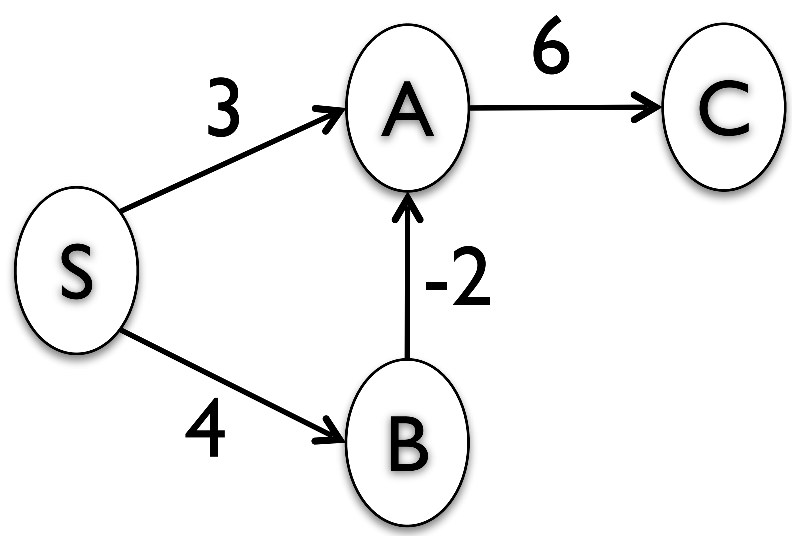
Dijkstra's SPA

Figure 4.8 Dijkstra's shortest-path algorithm.

```
procedure dijkstra(G, l, s)
Input:     Graph G = (V, E), directed or undirected;
           positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:    For all vertices u reachable from s, dist(u) is set
           to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue(V)  (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
      if dist(v) > dist(u) + l(u, v):
         dist(v) = dist(u) + l(u, v)
         prev(v) = u
         decreasekey(H, v)
```

Negative Edges

Figure 4.12 Dijkstra's algorithm will not work if there are negative edges.

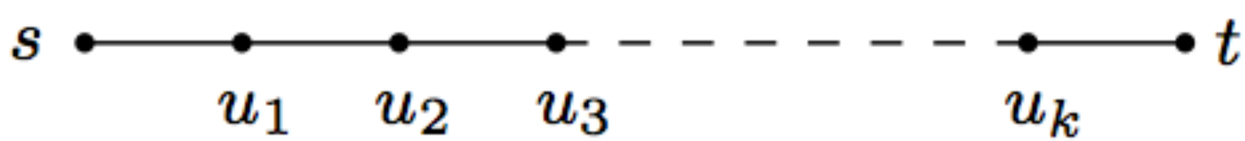


Bellman-Ford Algorithm

- Dijkstra's algorithm can be thought of simply as a sequence of `update`'s.

$$\underline{\texttt{procedure update}}\texttt{(}(u,v) \in E\texttt{)}$$
$$\texttt{dist}(v) = \min\{\texttt{dist}(v), \texttt{dist}(u) + l(u,v)\}$$

- Any sequence of updates from source to destination (part of the shortest path) can have at most |V|-1 edges.



- But, we don't know all shortest paths beforehand

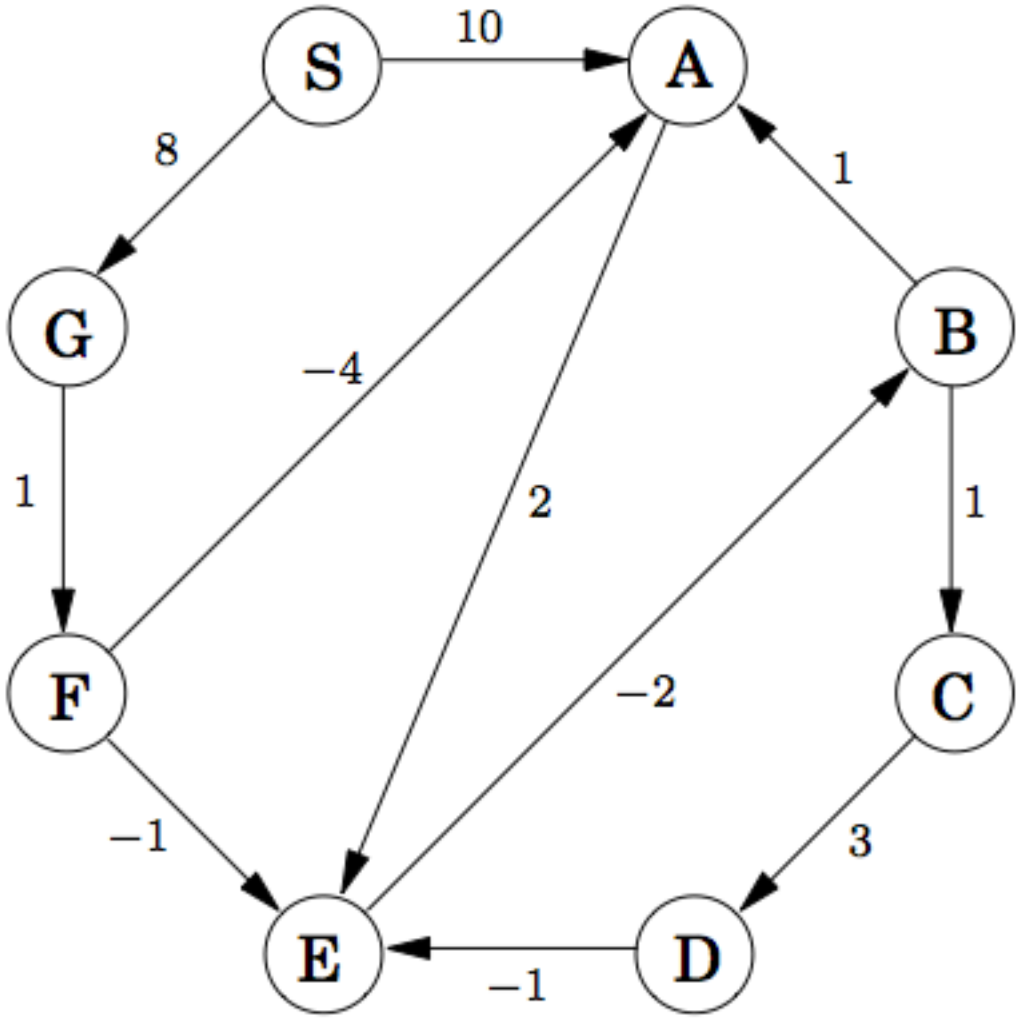
→ Bellman-Ford Algorithm:
Simply update all edges |V|-1 times (O(|V||E|))

Bellman-Ford Algorithm

Figure 4.13 The Bellman-Ford algorithm for single-source shortest paths in general graphs.

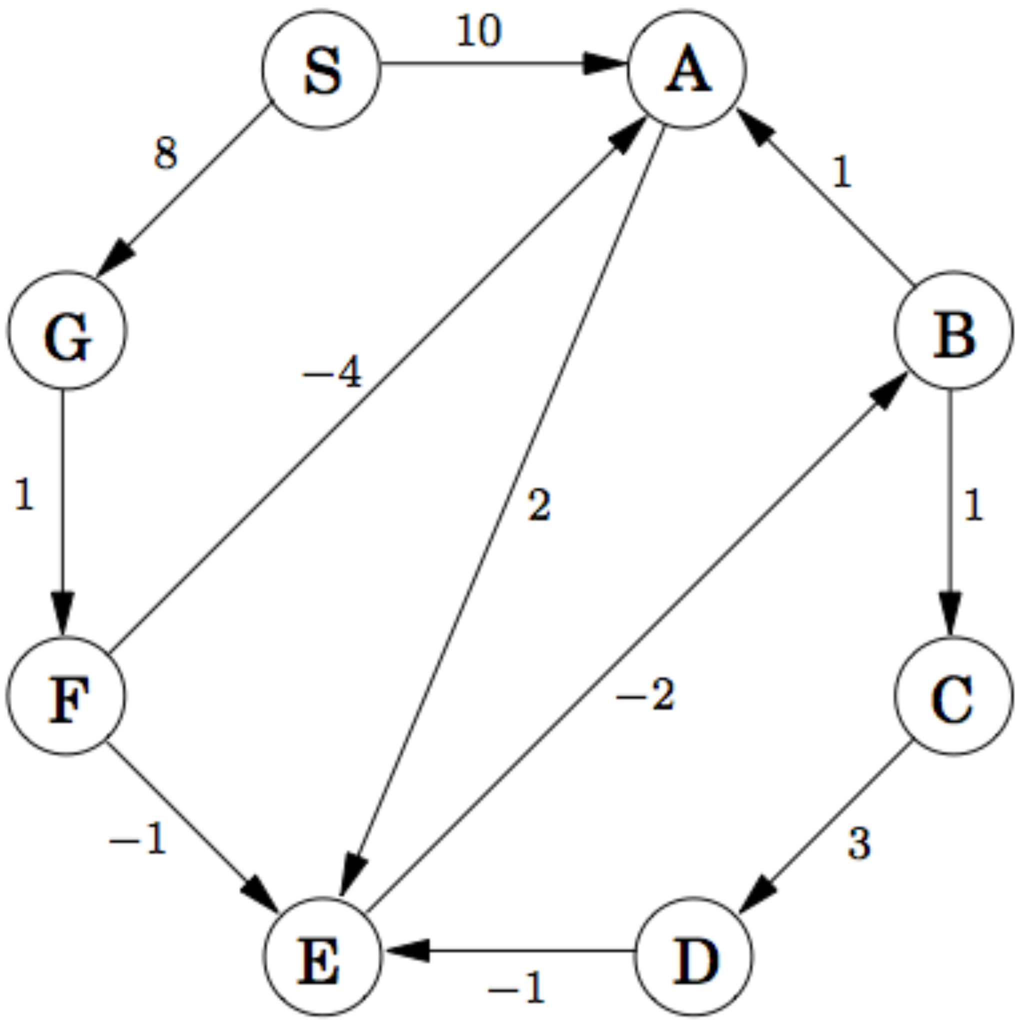
```
procedure shortest-paths(G, l, s)
Input:    Directed graph G = (V, E);
          edge lengths {l_e : e ∈ E} with no negative cycles;
          vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
repeat |V| - 1 times:
   for all e ∈ E:
      update(e)
```

Bellman-Ford Algorithm



There are 8 Nodes, Bellman-Ford needs 7 iterations

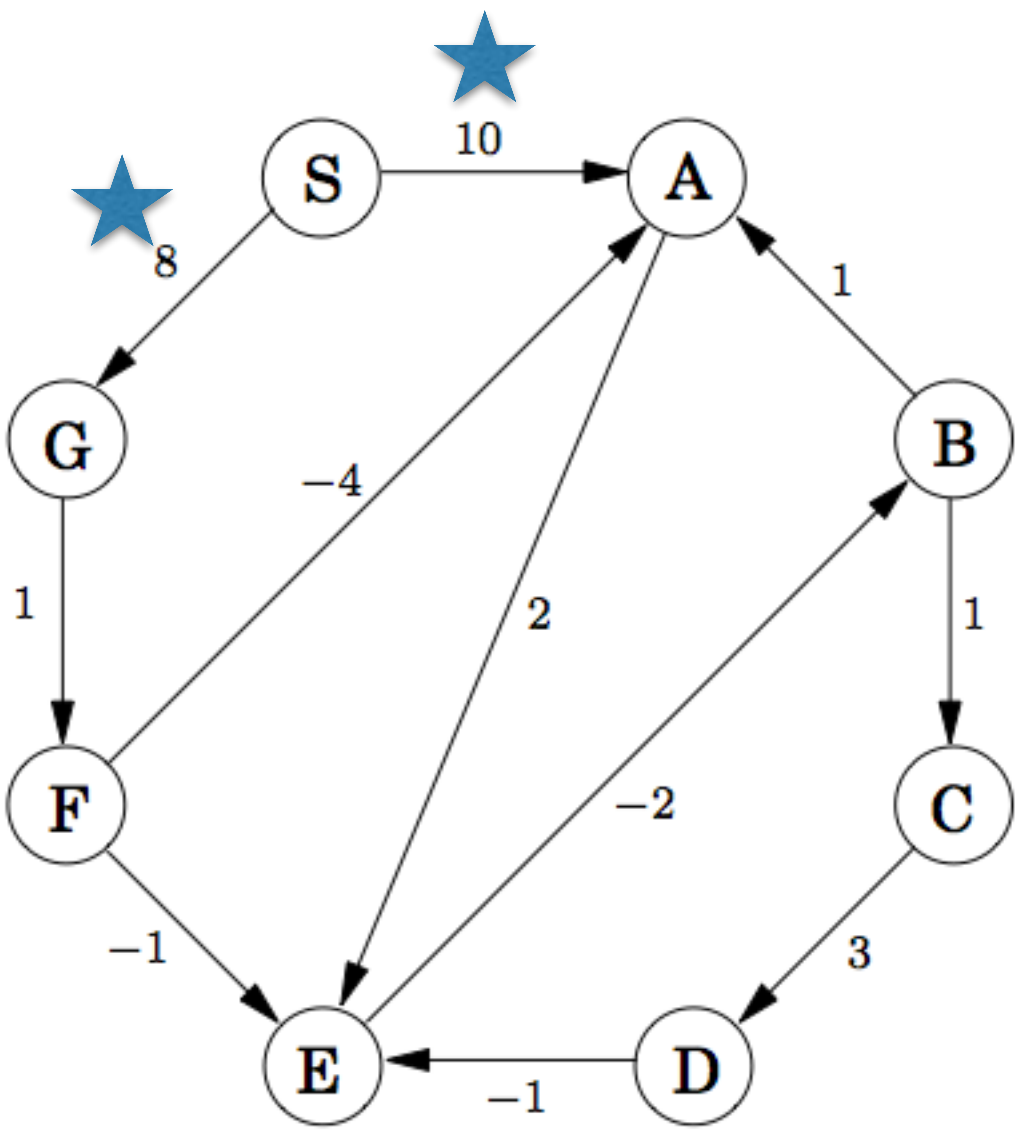
Bellman-Ford Algorithm



Initialization

Node	0
S	0
A	∞
B	∞
C	∞
D	∞
E	∞
F	∞
G	∞

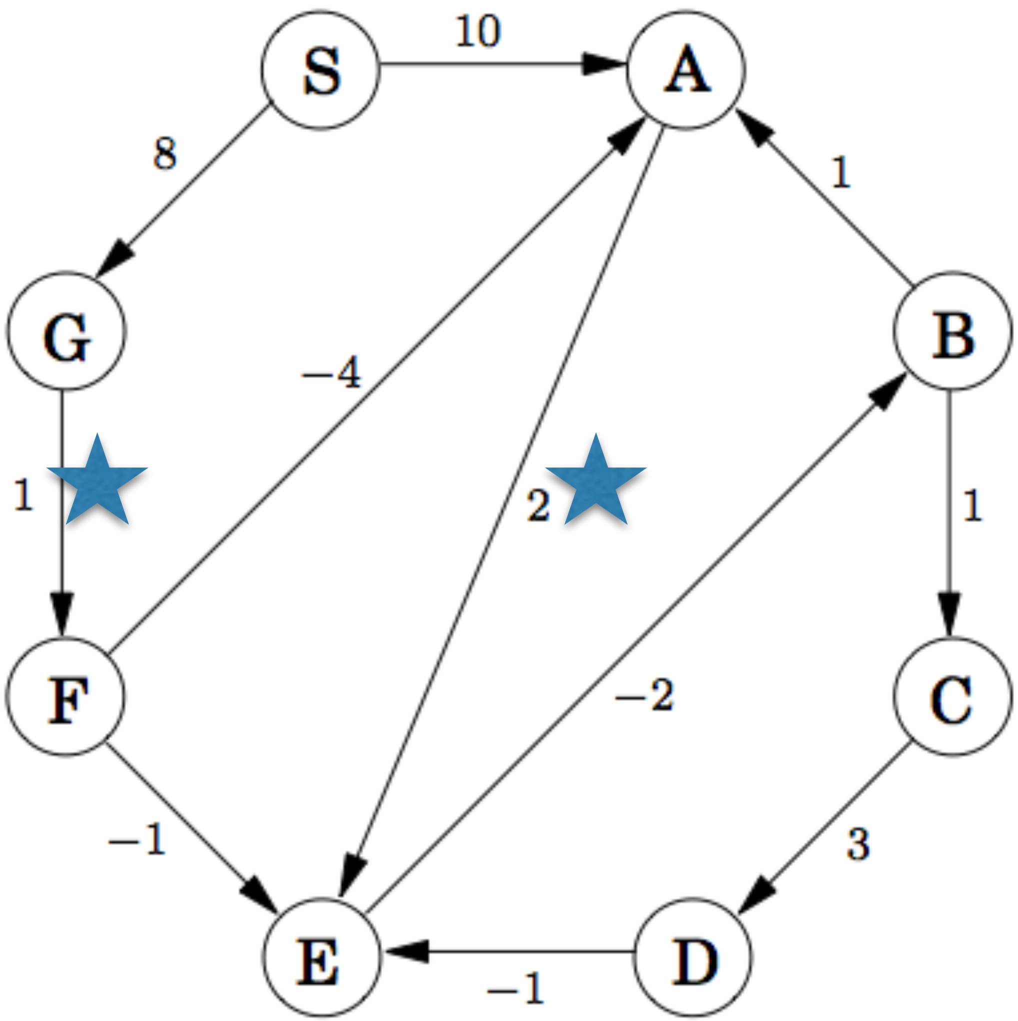
Bellman-Ford Algorithm



Iteration: 1

Node	0	1
S	0	0
A	∞	10
B	∞	∞
C	∞	∞
D	∞	∞
E	∞	∞
F	∞	∞
G	∞	8

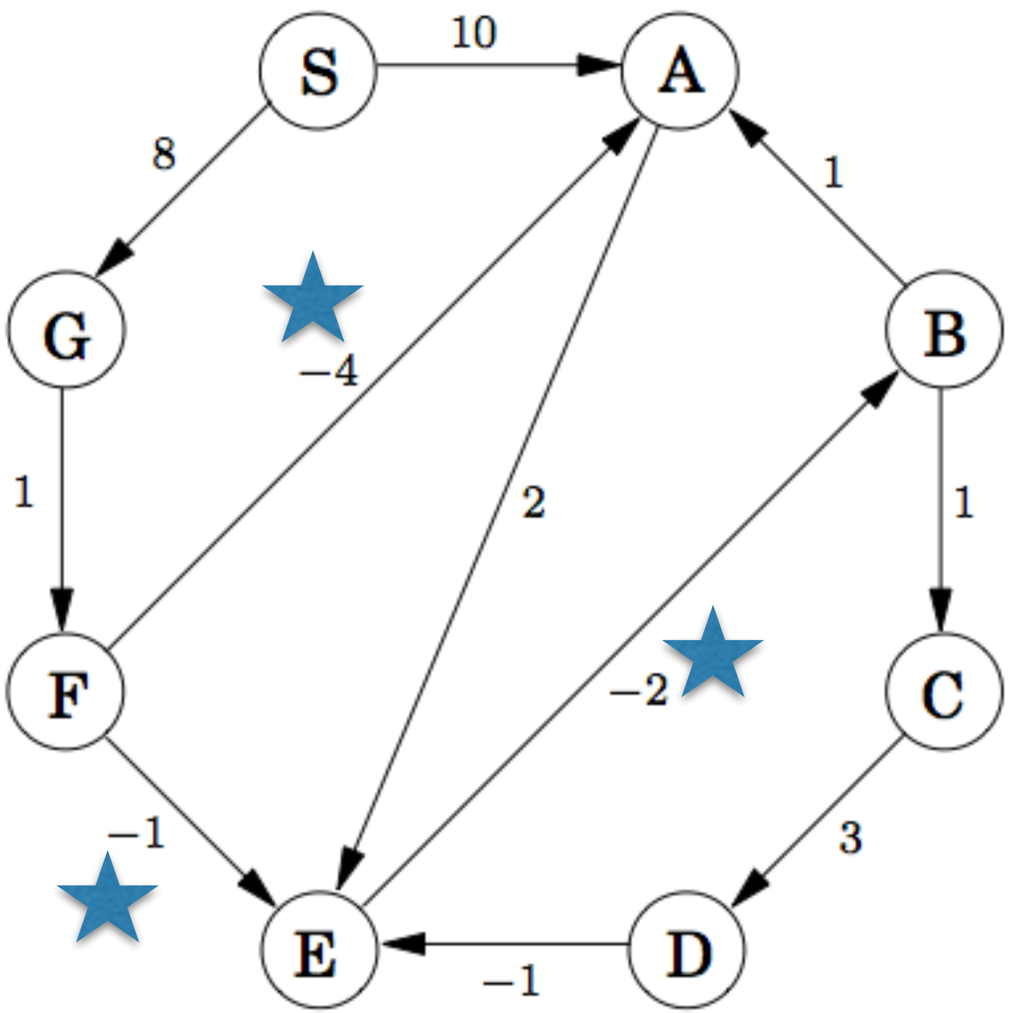
Bellman-Ford Algorithm



Iteration: 2

	I		
Node	**0**	**1**	**2**
S	0	0	0
A	∞	10	10
B	∞	∞	∞
C	∞	∞	∞
D	∞	∞	∞
E	∞	∞	12
F	∞	∞	9
G	∞	8	8

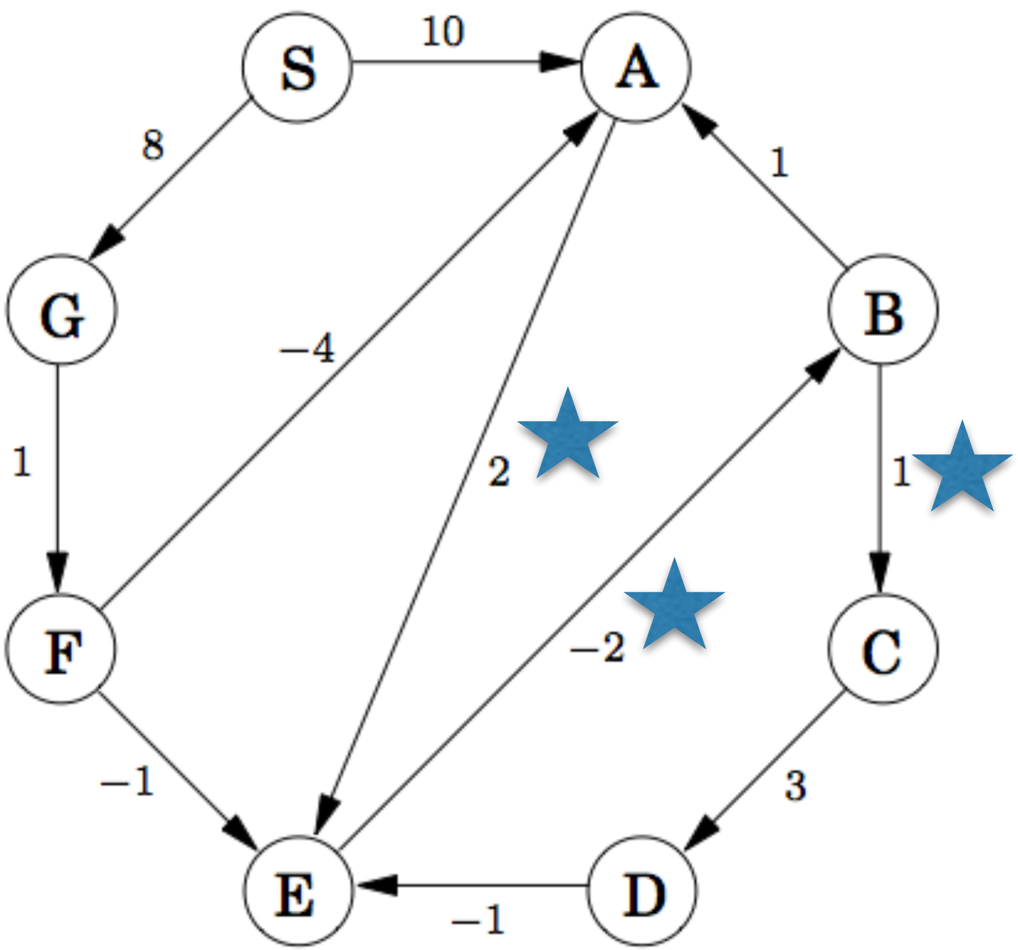
Bellman-Ford Algorithm



Iteration: 3

	Itera			
Node	0	1	2	3
S	0	0	0	0
A	∞	10	10	5
B	∞	∞	∞	10
C	∞	∞	∞	∞
D	∞	∞	∞	∞
E	∞	∞	12	8
F	∞	∞	9	9
G	∞	8	8	8

Bellman-Ford Algorithm

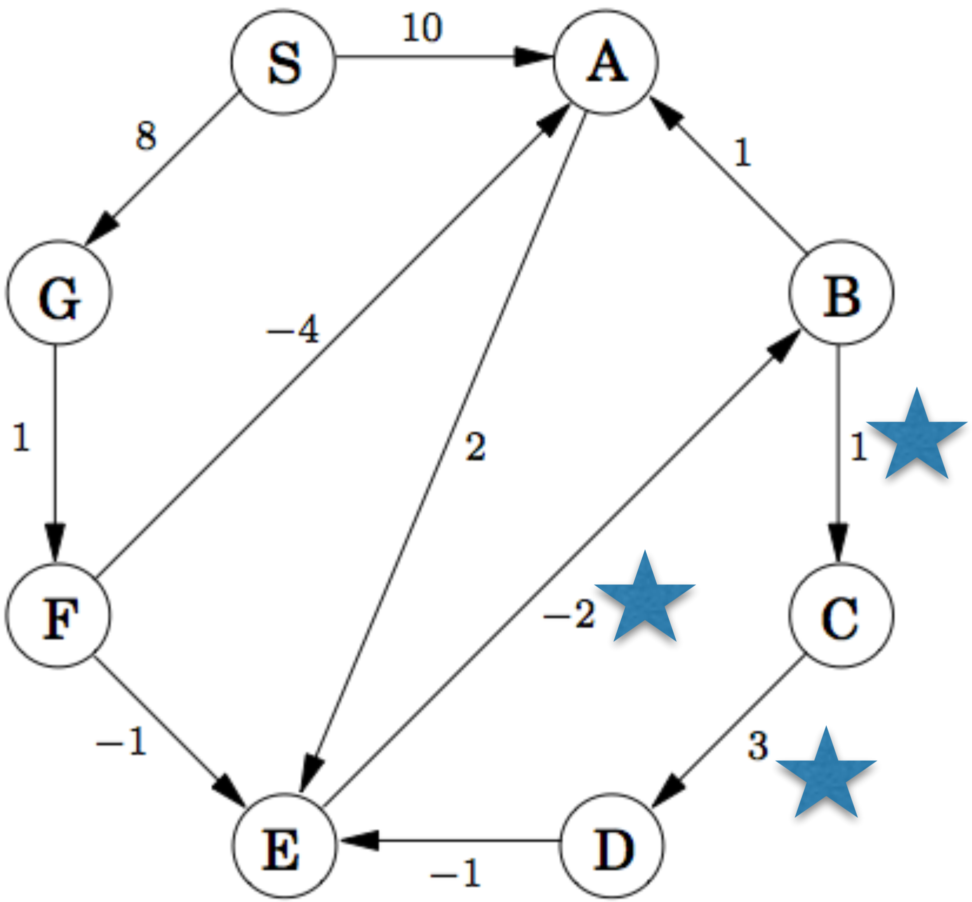


Iteration: 4

	Iteration				
Node	0	1	2	3	4
S	0	0	0	0	0
A	∞	10	10	5	5
B	∞	∞	∞	10	6
C	∞	∞	∞	∞	11
D	∞	∞	∞	∞	∞
E	∞	∞	12	8	7
F	∞	∞	9	9	9
G	∞	8	8	8	8

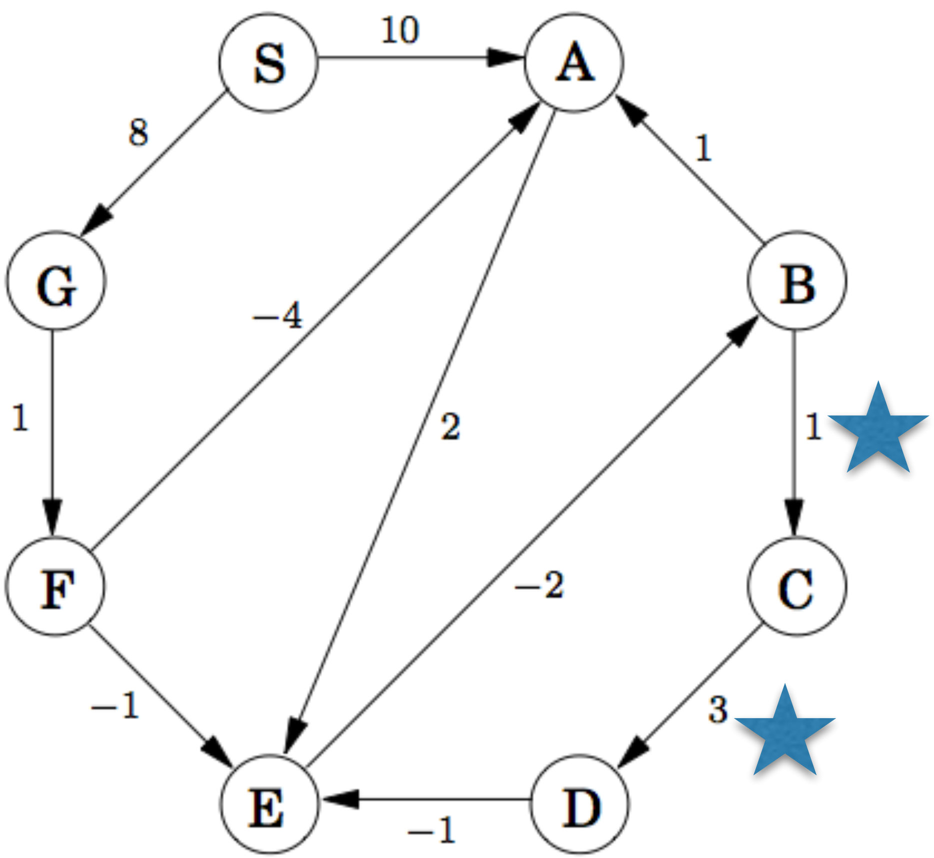
Bellman-Ford Algorithm

Iteration: 5



	Iteration					
Node	0	1	2	3	4	5
S	0	0	0	0	0	0
A	∞	10	10	5	5	5
B	∞	∞	∞	10	6	5
C	∞	∞	∞	∞	11	7
D	∞	∞	∞	∞	∞	14
E	∞	∞	12	8	7	7
F	∞	∞	9	9	9	9
G	∞	8	8	8	8	8

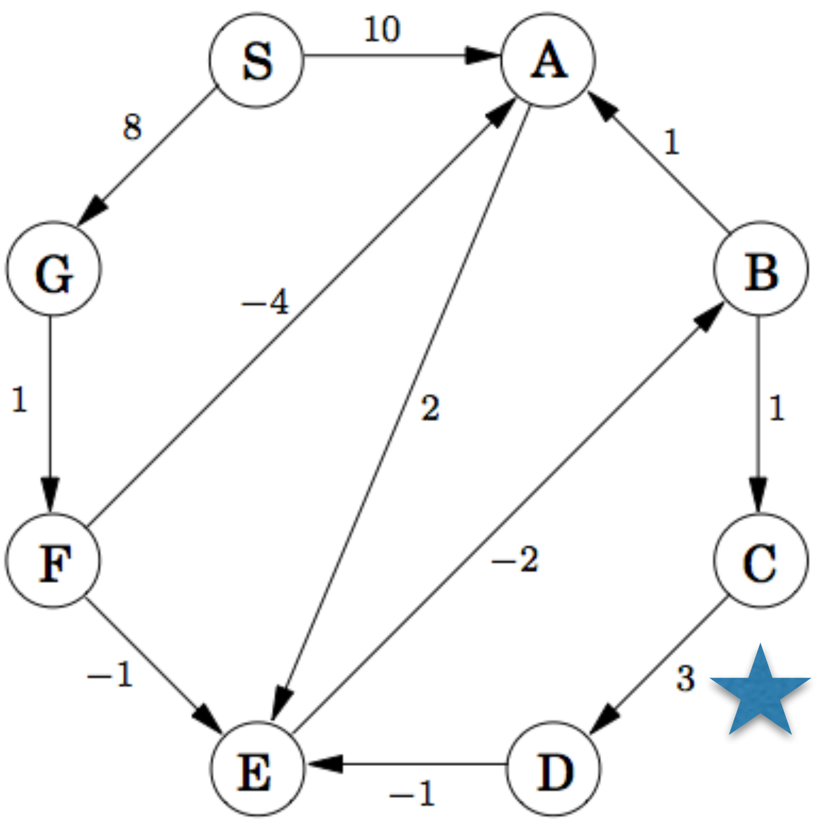
Bellman-Ford Algorithm



Iteration: 6

	Iteration						
Node	**0**	**1**	**2**	**3**	**4**	**5**	**6**
S	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5
B	∞	∞	∞	10	6	5	5
C	∞	∞	∞	∞	11	7	6
D	∞	∞	∞	∞	∞	14	10
E	∞	∞	12	8	7	7	7
F	∞	∞	9	9	9	9	9
G	∞	8	8	8	8	8	8

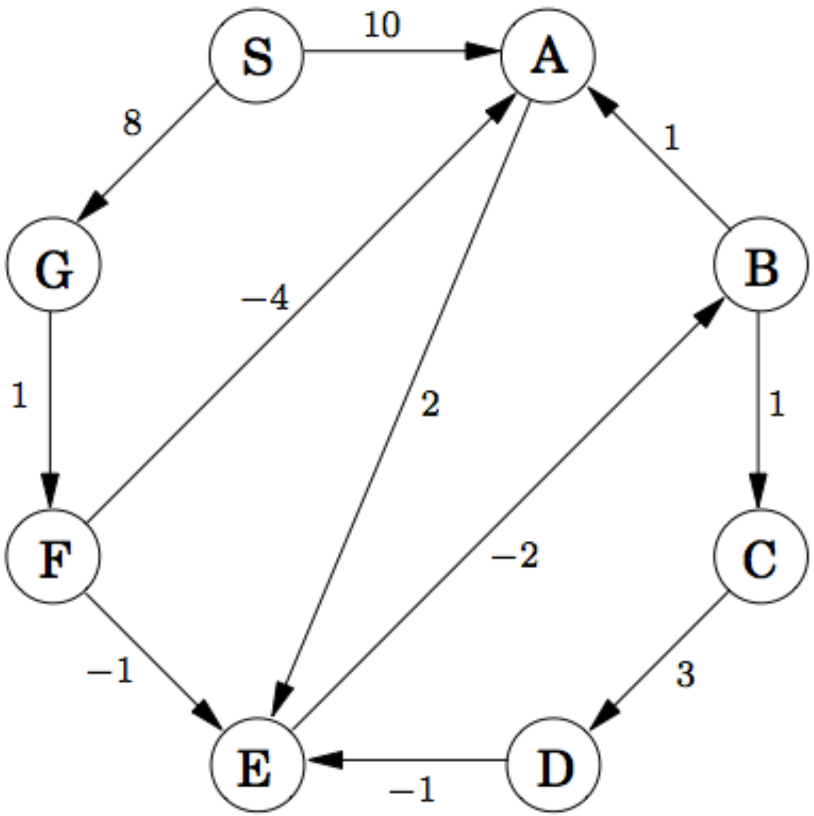
Bellman-Ford Algorithm



Iteration: 7

	Iteration							
Node	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Bellman-Ford Algorithm

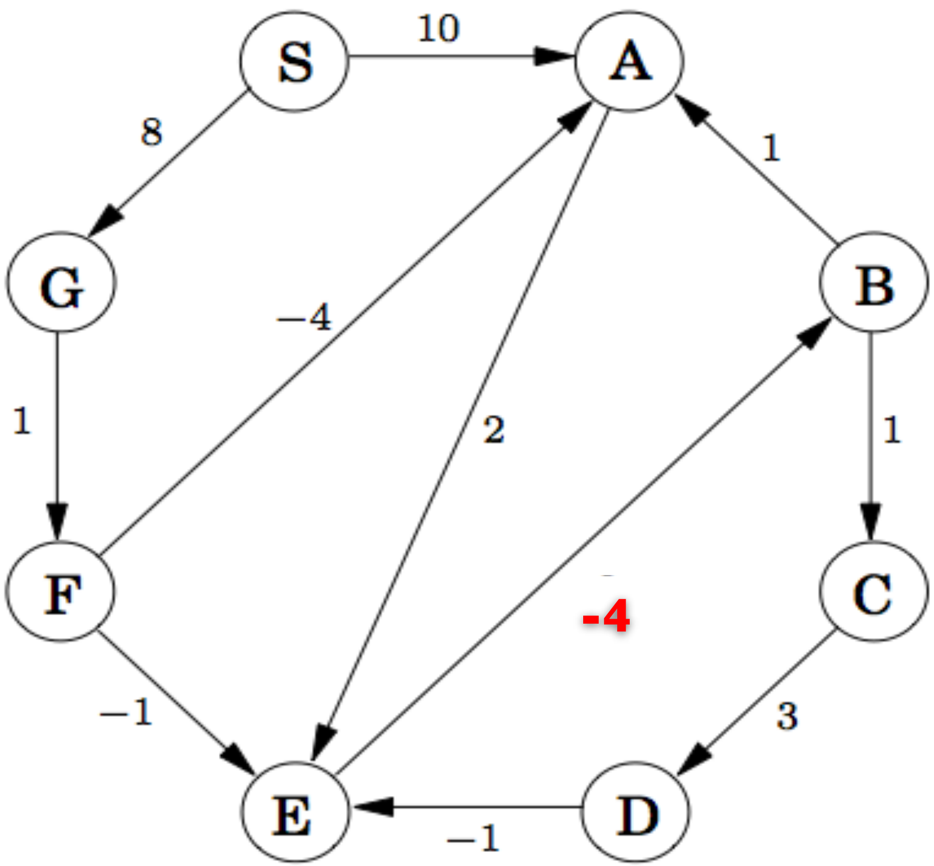


Done!

	Iteration							
Node	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Negative Cycles

- If there is a negative cycle (a loop with negative length) in graph, shortest path problem is ill-defined



What is the length of shortest path from A to E?

Negative Cycles

- Easy to detect negative cycle in Bellman-Ford algorithm.
  - With negative cycles, each update round will indefinitely continue to make changes to the distance values
  - **Do one more iteration after |V|-1 iterations and check if there is any change**

Shortest Paths in DAGs

- If the graph is DAG (no cycle), shortest paths can be found in linear time
- Topologically sort the DAG and perform the updates in the sorted order

Figure 4.15 **A single-source shortest-path algorithm for directed acyclic graphs.**

```
procedure dag-shortest-paths(G, l, s)
Input:    Dag G = (V, E);
          edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
Linearize G
for each u ∈ V, in linearized order:
   for all edges (u, v) ∈ E:
      update(u, v)
```