THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING &
MATHEMATICAL SCIENCES

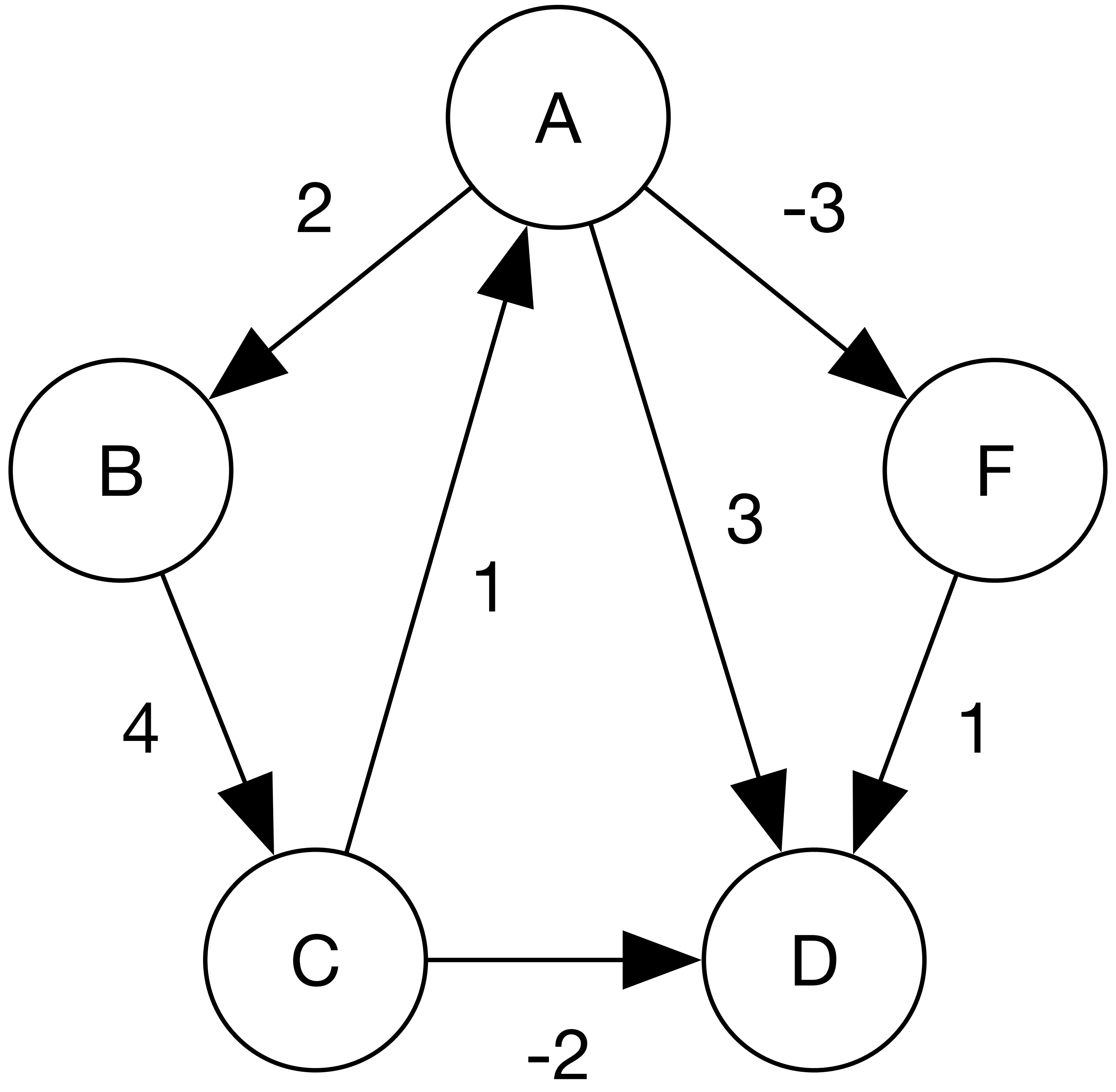
Shortest Path

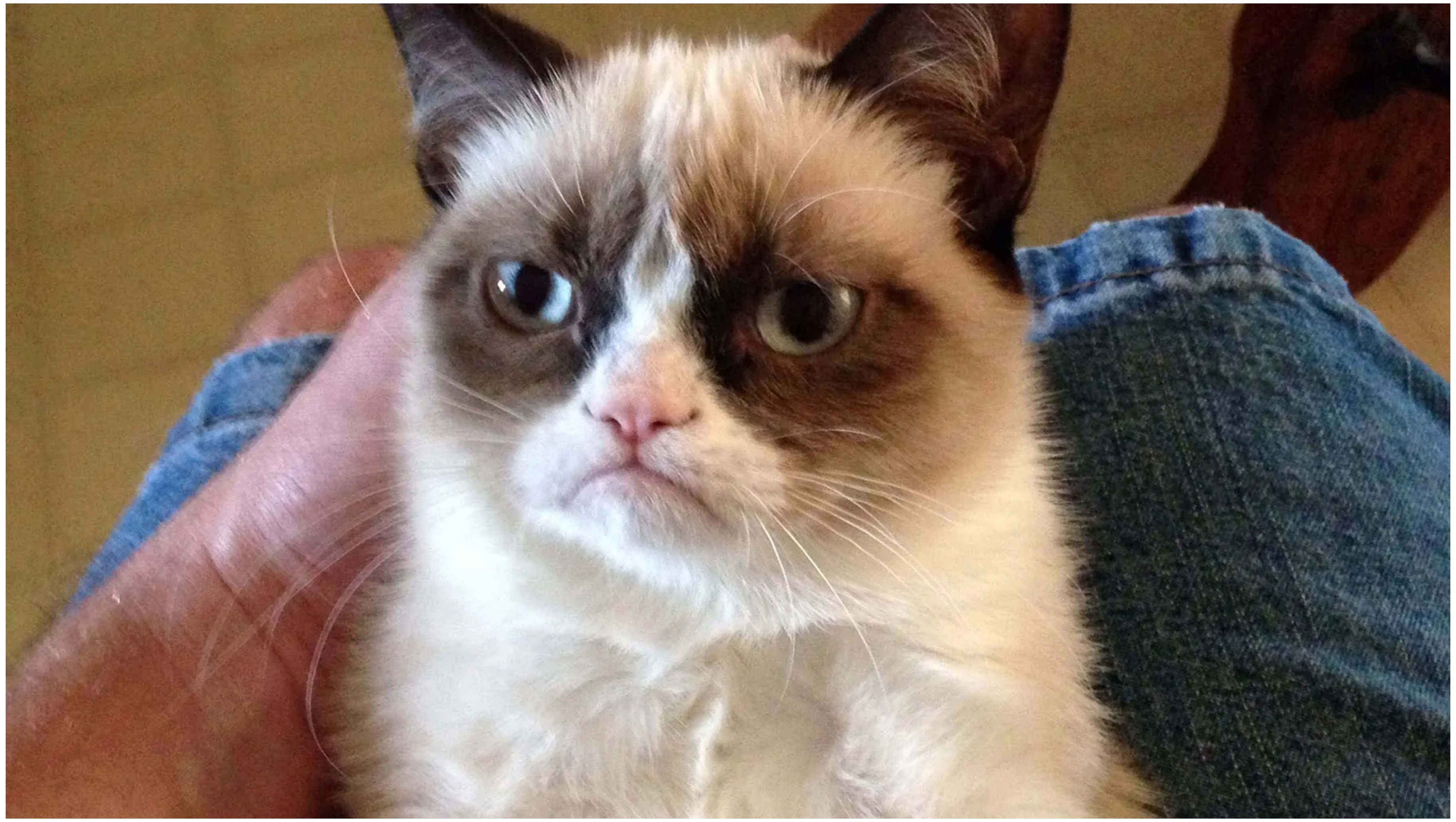
Bellman-Ford Algorithm

CS 124 / Department of Computer Science

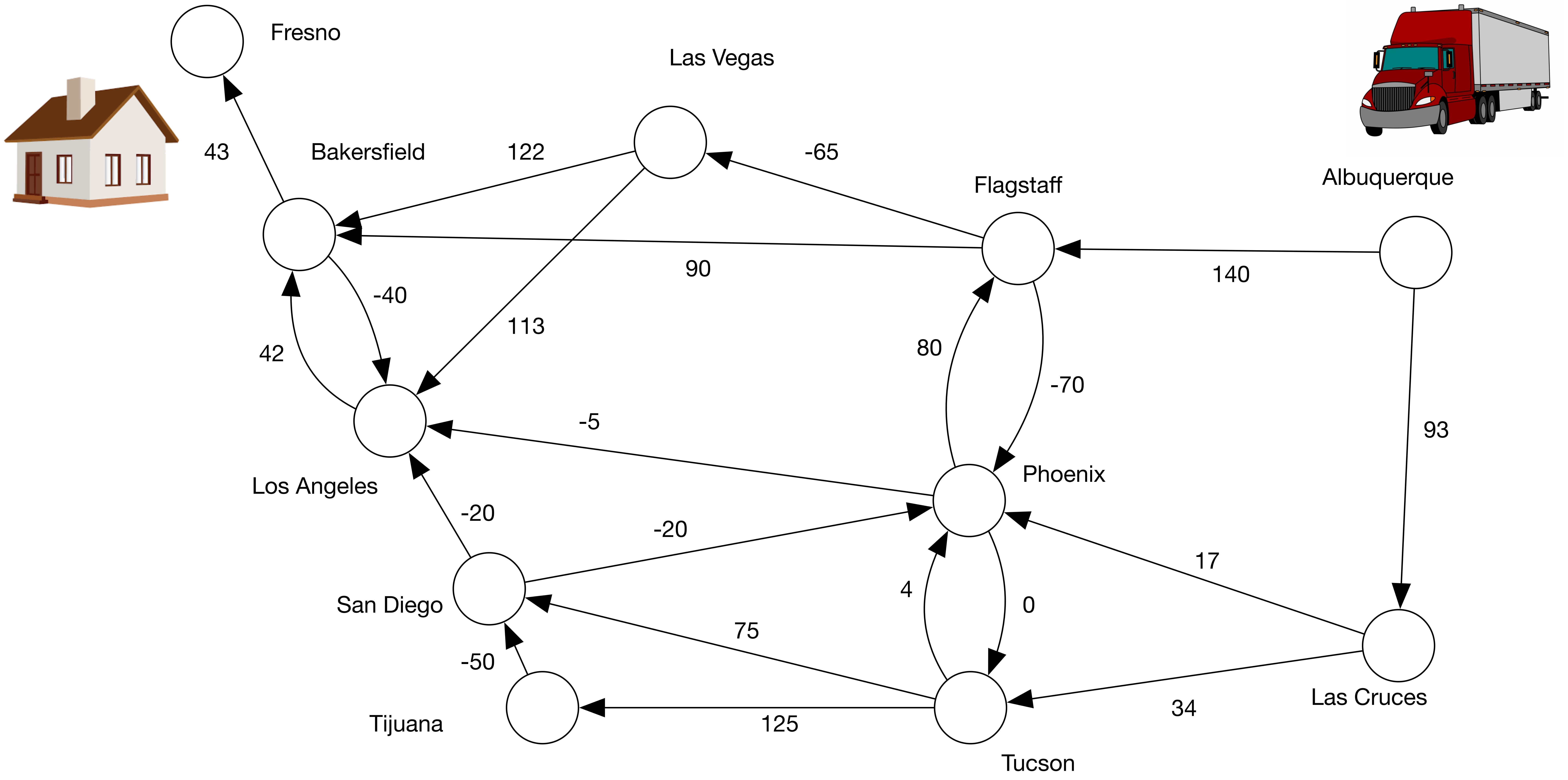
Shortest path

Dijkstra's algorithm doesn't work if there's an edge in the graph with negative weight.





Lowest-cost path with negative weights



What can be done?

What can be done?

Bellman-Ford algorithm

Bellman-Ford

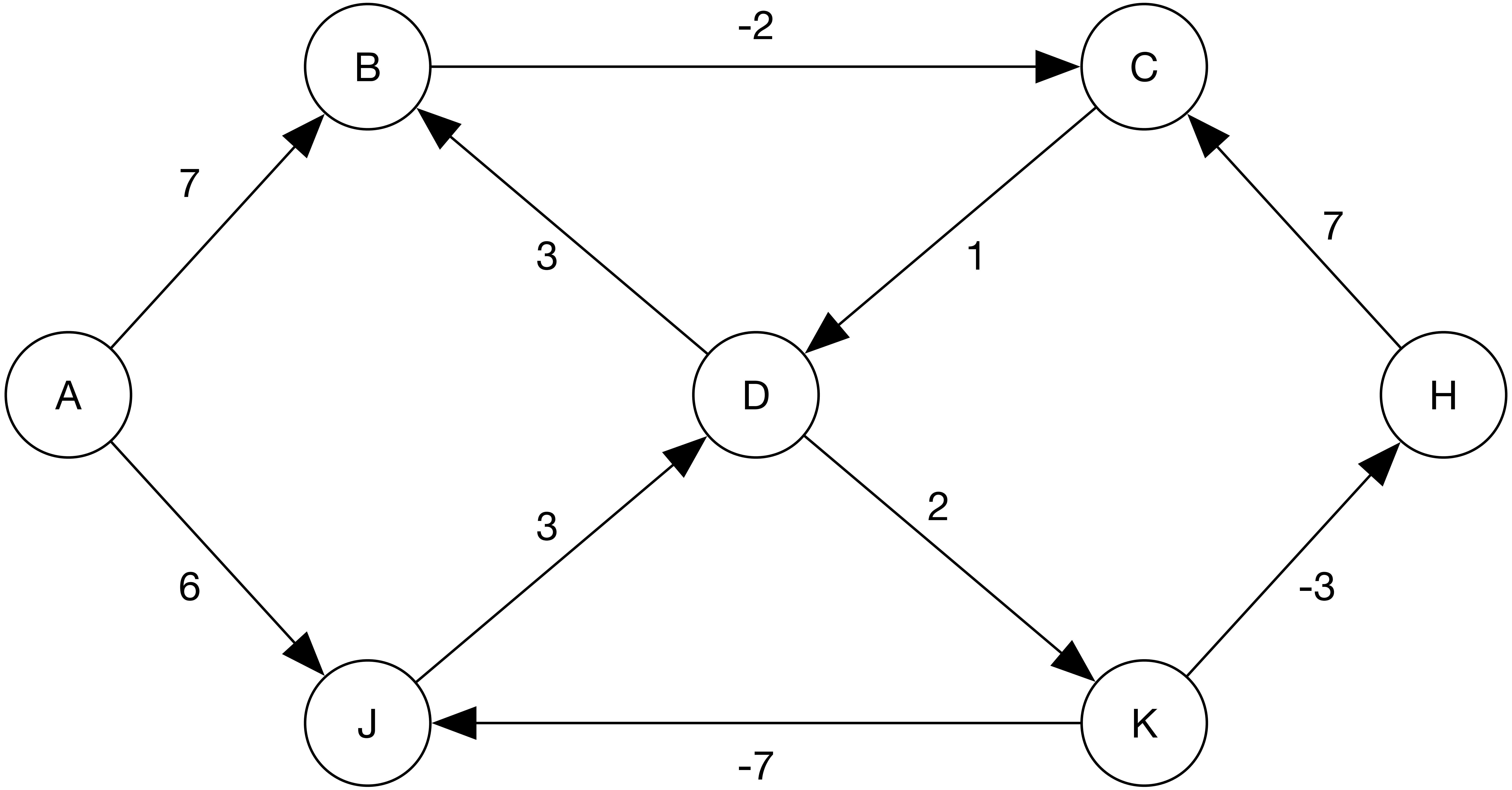
Given some graph, G = (V, E), and some starting node S ∈ V, the Bellman-Ford algorithm will find the shortest paths (or paths with minimum weight) from S to all other nodes in V.

Note that G must not contain any negative weight cycles.

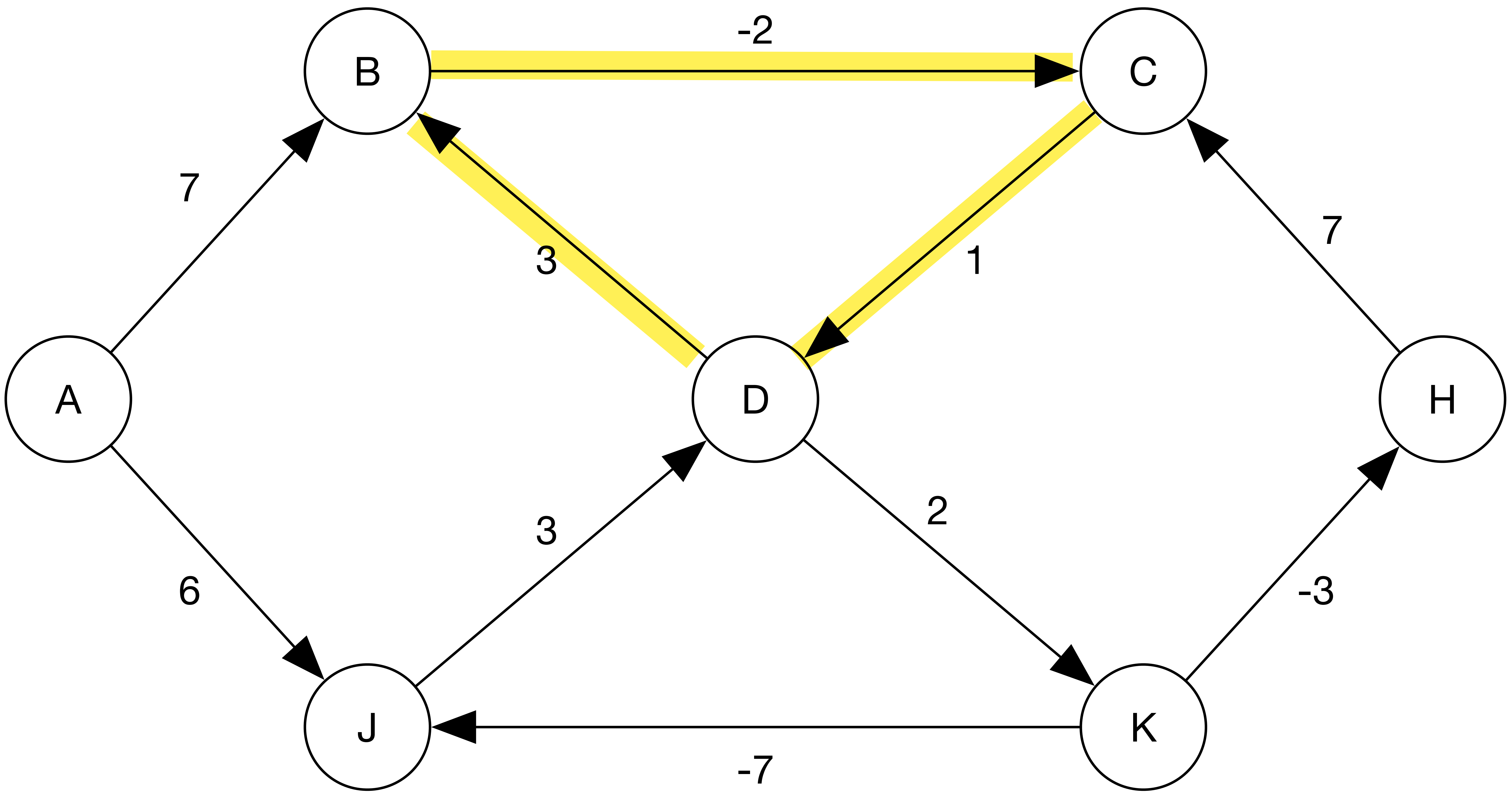
Shortest Path / Lowest Cost Path

Algorithm	Dijkstra	Bellman - Ford
Worst-case complexity	$O((\lvert V\rvert + \lvert E\rvert) \log \lvert V\rvert)$	$O(\lvert V\rvert \times \lvert E\rvert)$
Restrictions	Edge weights must be non-negative	No negative cycles

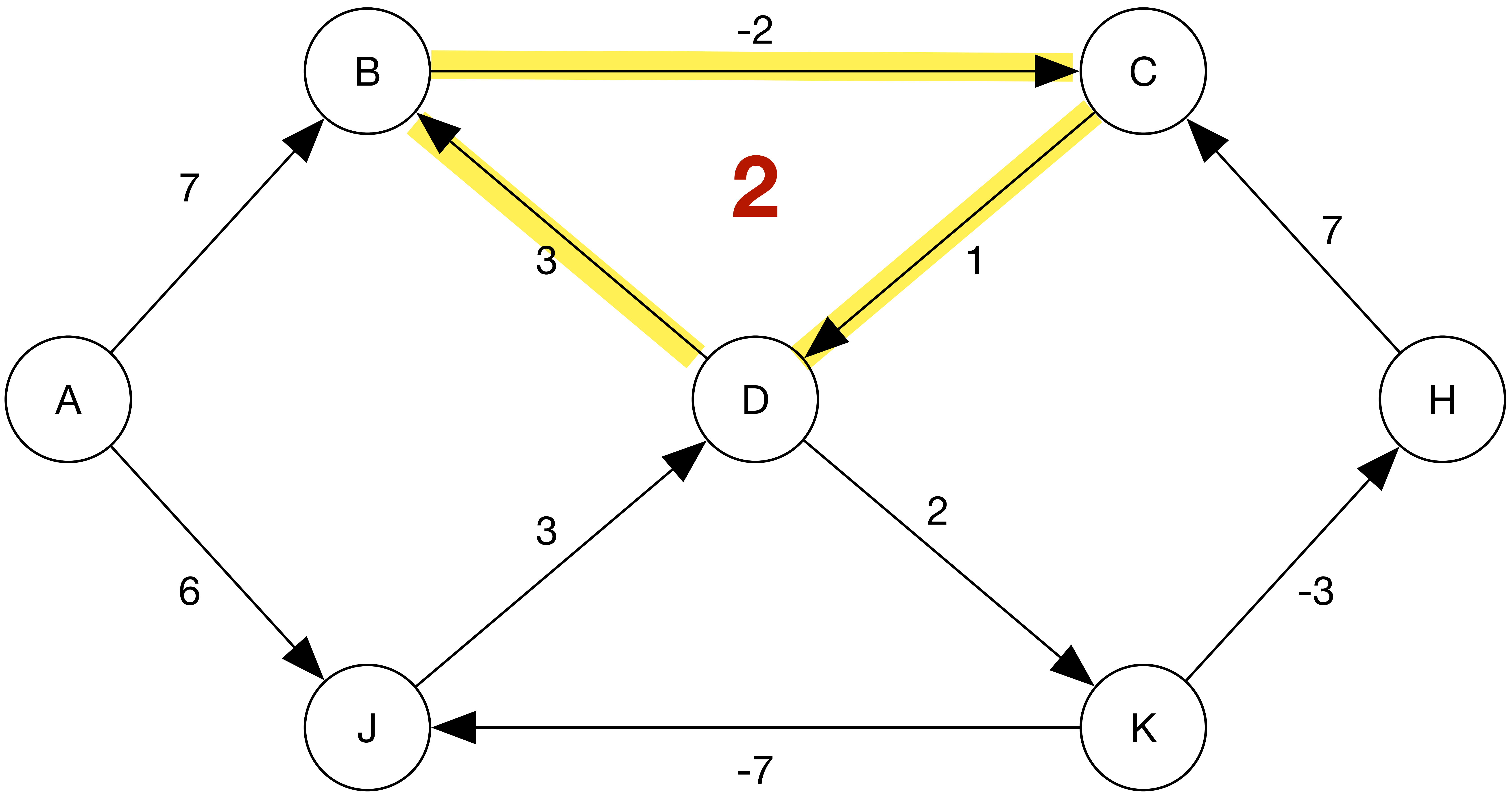
What is a negative cycle?



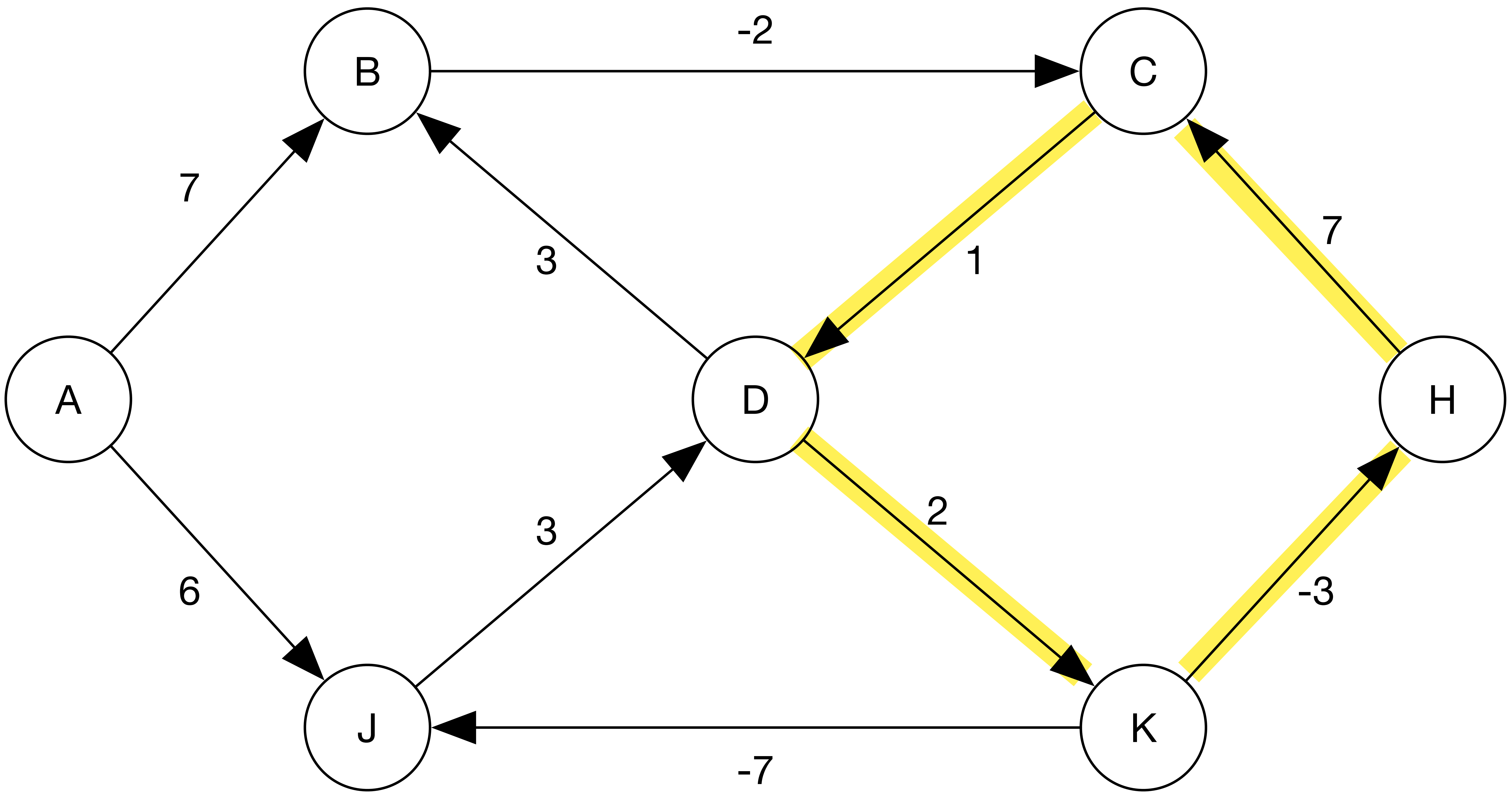
What is a negative cycle?



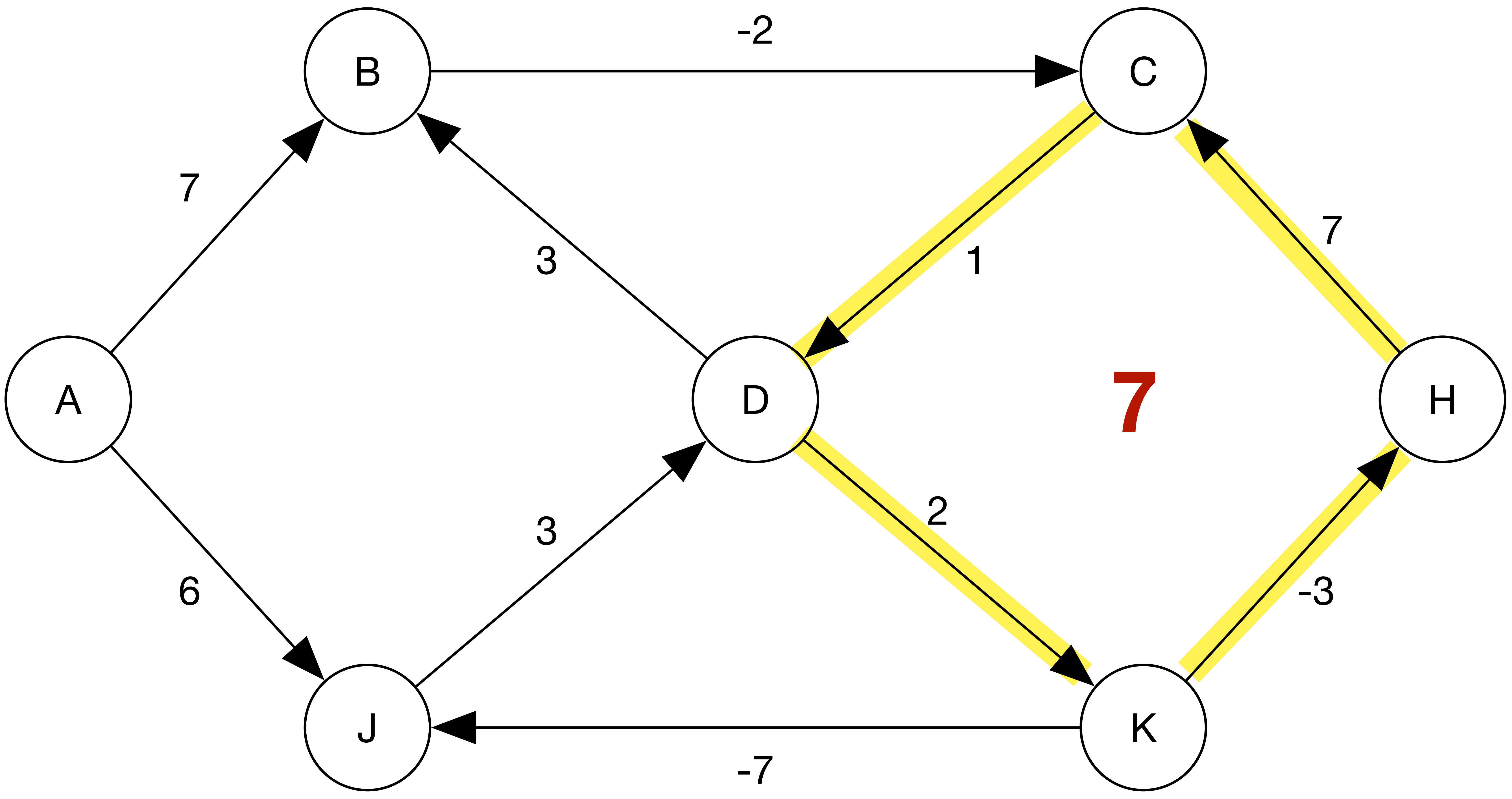
What is a negative cycle?



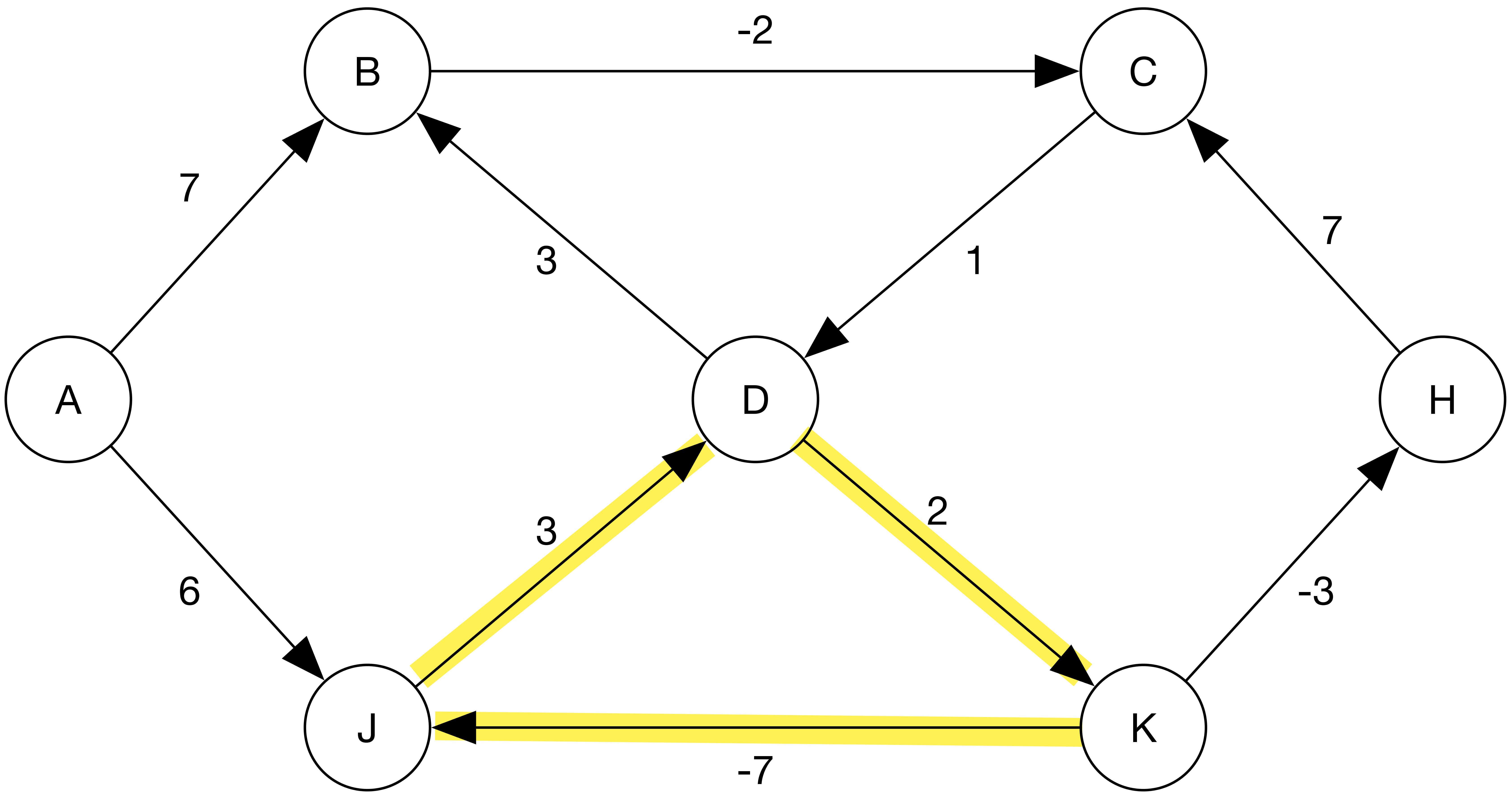
What is a negative cycle?



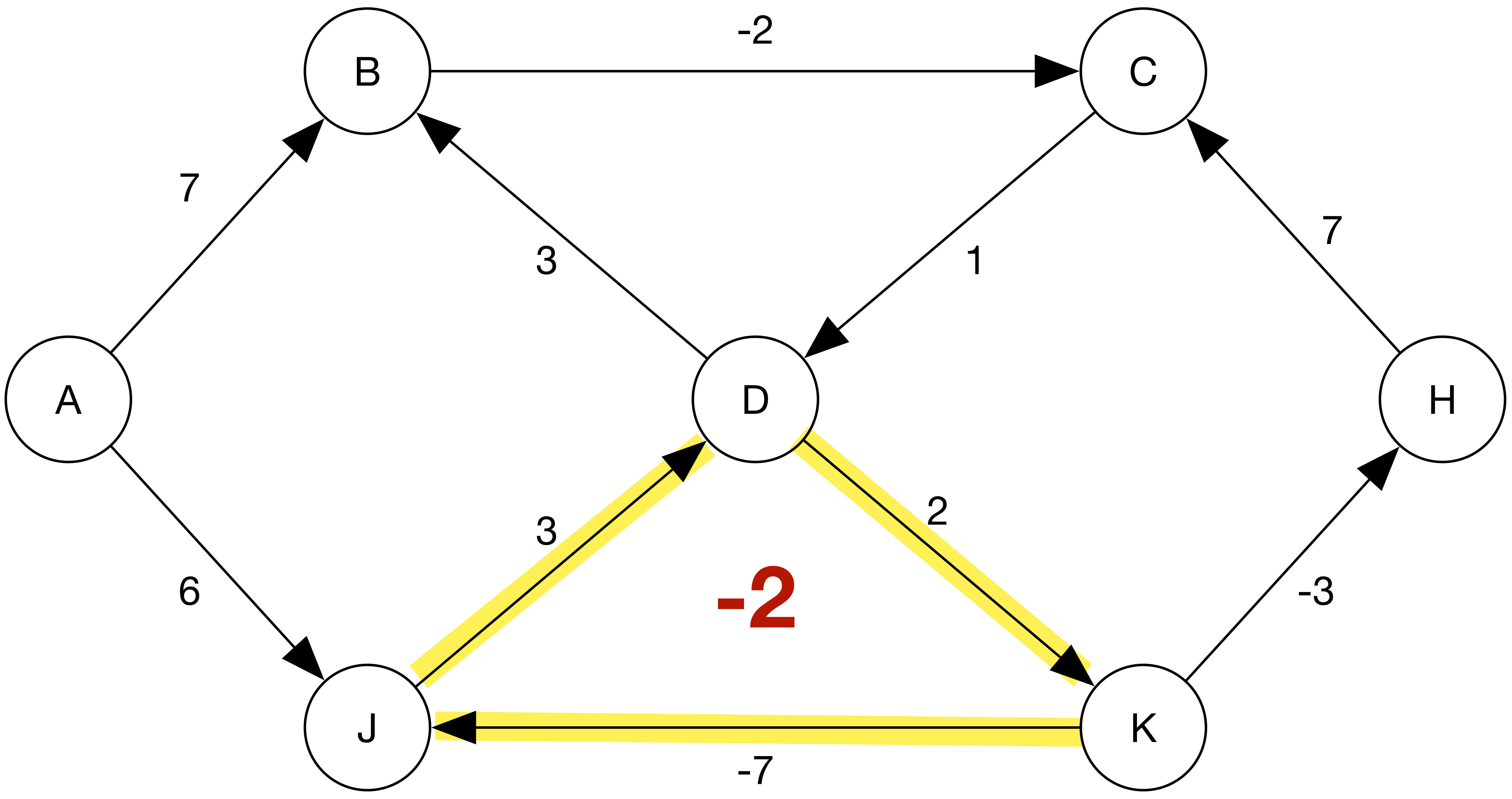
What is a negative cycle?



What is a negative cycle?



What is a negative cycle?



Bellman-Ford: pseudocode

```
function bellman_ford(G, S)  // G is the graph; S is the starting node
    for each node V in G
        arrived_from[V] = null
        if V = S
            distance[V] = 0
        else
            distance[V] = infinite

   repeat |V| - 1 times or until no distances are updated
        for each edge (U, V) in E
            distance = distance[U] + weight of edge
            if distance < distance[V]  // We've found a shorter distance
                distance[V] = distance
                arrived_from[V] = U

   for each edge (U, V) in E
       if distance[V] > distance[U] + weight of edge
           return "ERROR: negative weight cycle"
```

Bellman-Ford: pseudocode

```
function bellman_ford(G, S)  // G is the graph; S is the starting node
    for each node V in G
        arrived_from[V] = null
        if V = S
            distance[V] = 0
        else
            distance[V] = infinite

   repeat |V| - 1 times or until no distances are updated
        for each edge (U, V) in E
            distance = distance[U] + weight of edge
            if distance < distance[V]  // We've found a shorter distance
                distance[V] = distance
                arrived_from[V] = U

   for each edge (U, V) in E
       if distance[V] > distance[U] + weight of edge
           return "ERROR: negative weight cycle"
```

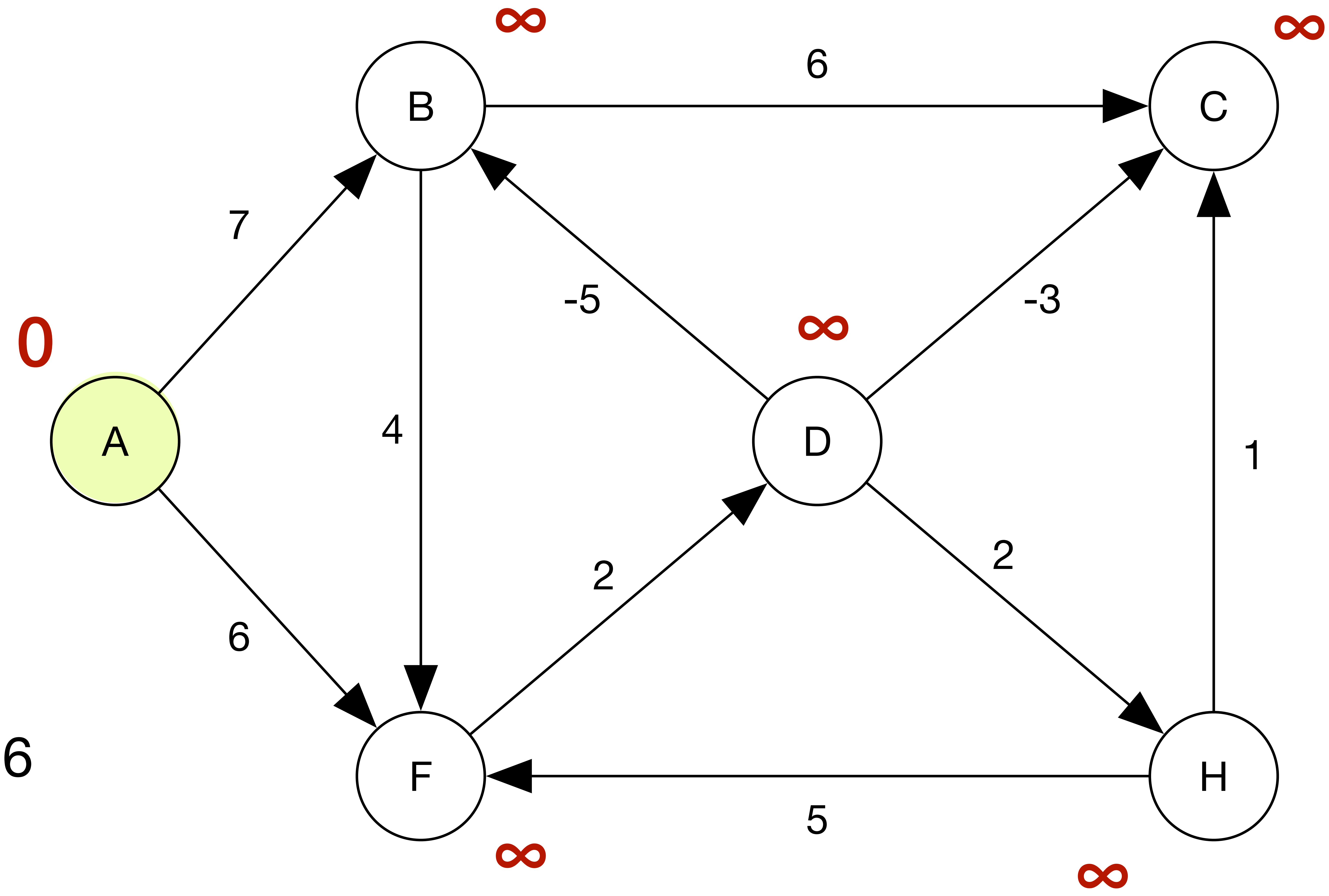
Bellman-Ford: pseudocode

```
function bellman_ford(G, S)  // G is the graph; S is the starting node
    for each node V in G
        arrived_from[V] = null
        if V = S
            distance[V] = 0
        else
            distance[V] = infinite

   repeat |V| - 1 times or until no distances are updated
        for each edge (U, V) in E
            distance = distance[U] + weight of edge
            if distance < distance[V]  // We've found a shorter distance
                distance[V] = distance
                arrived_from[V] = U

   for each edge (U, V) in E
       if distance[V] > distance[U] + weight of edge
           return "ERROR: negative weight cycle"
```

Bellman-Ford

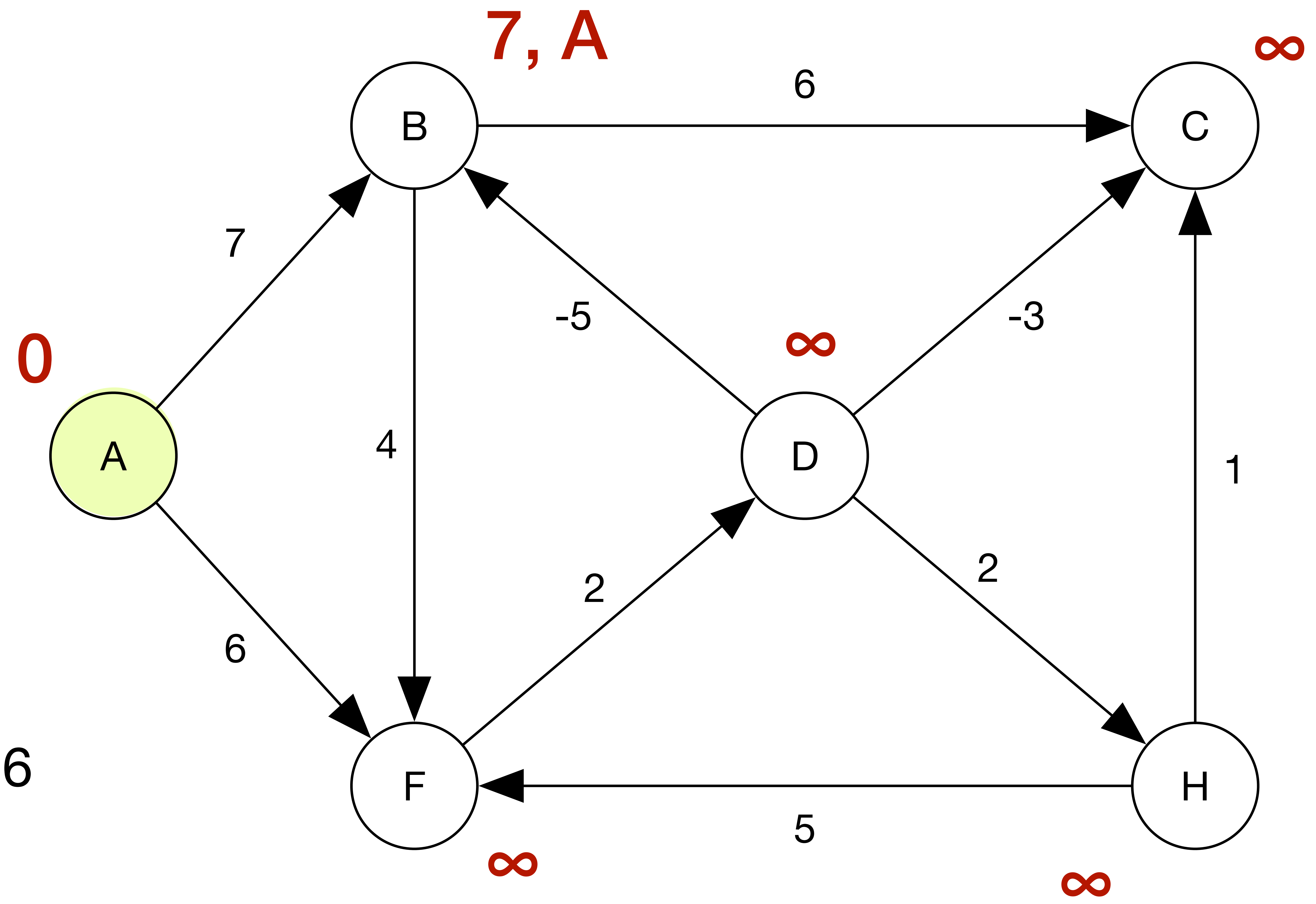


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

First iteration

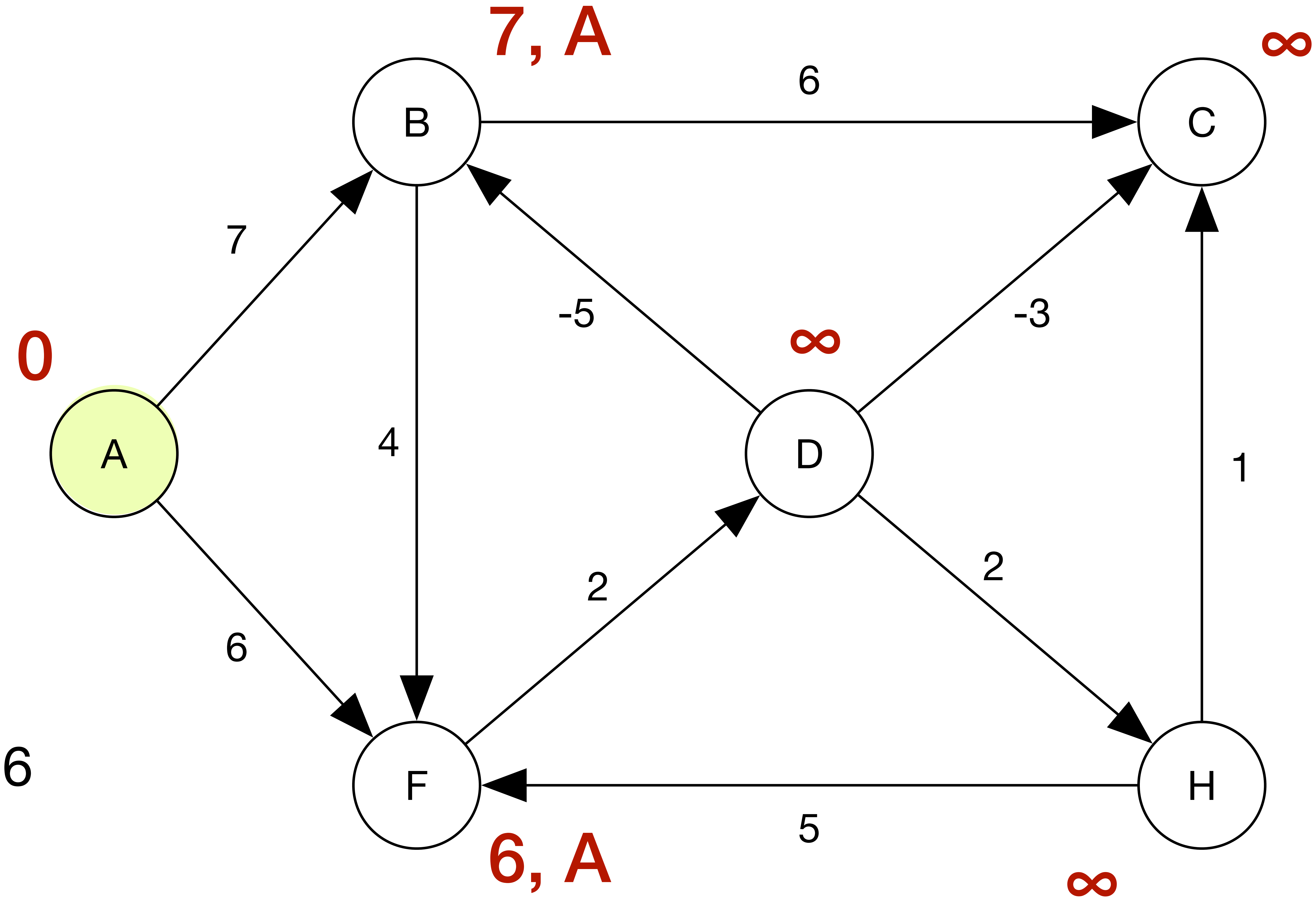


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

First iteration

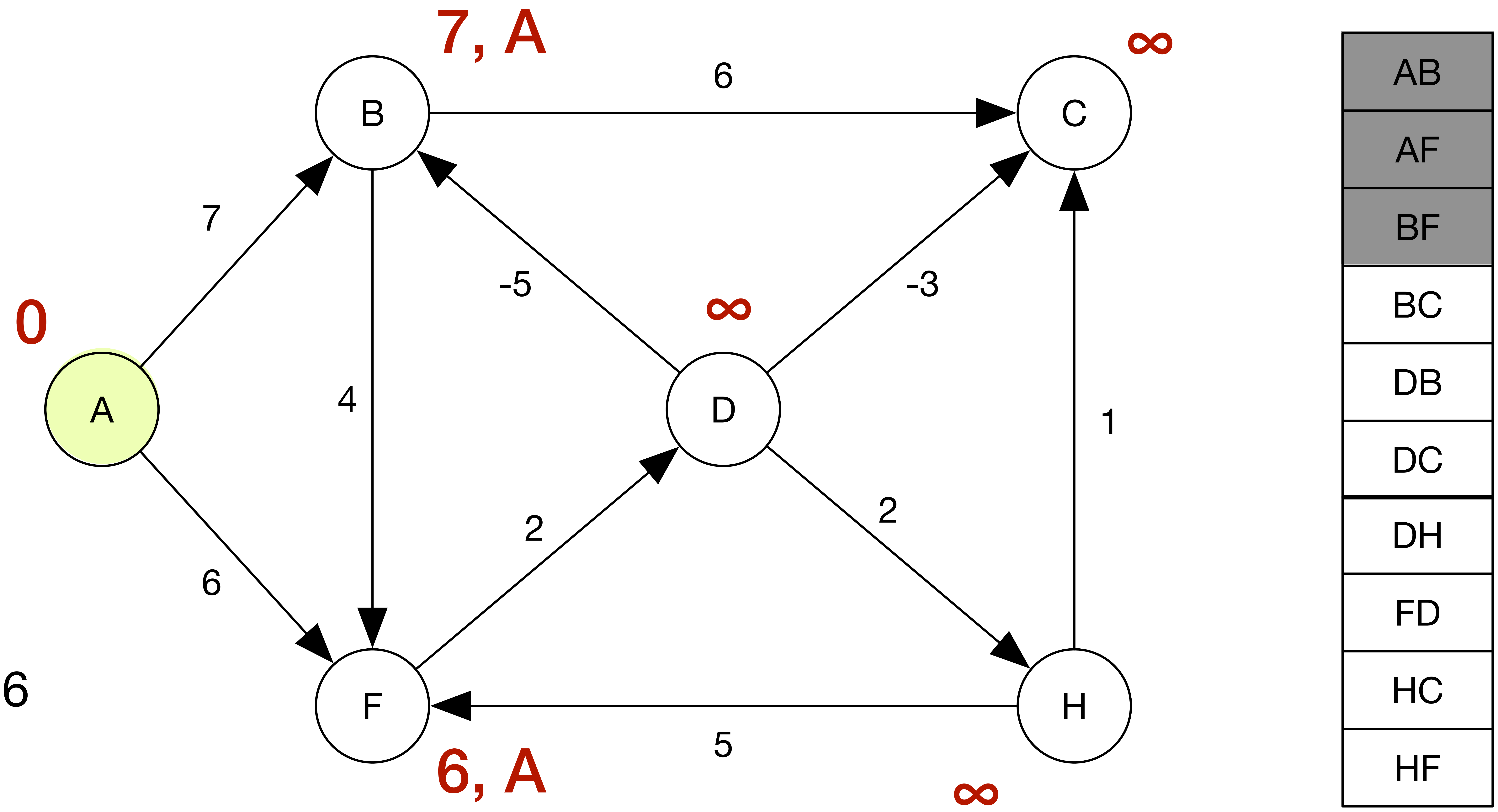


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

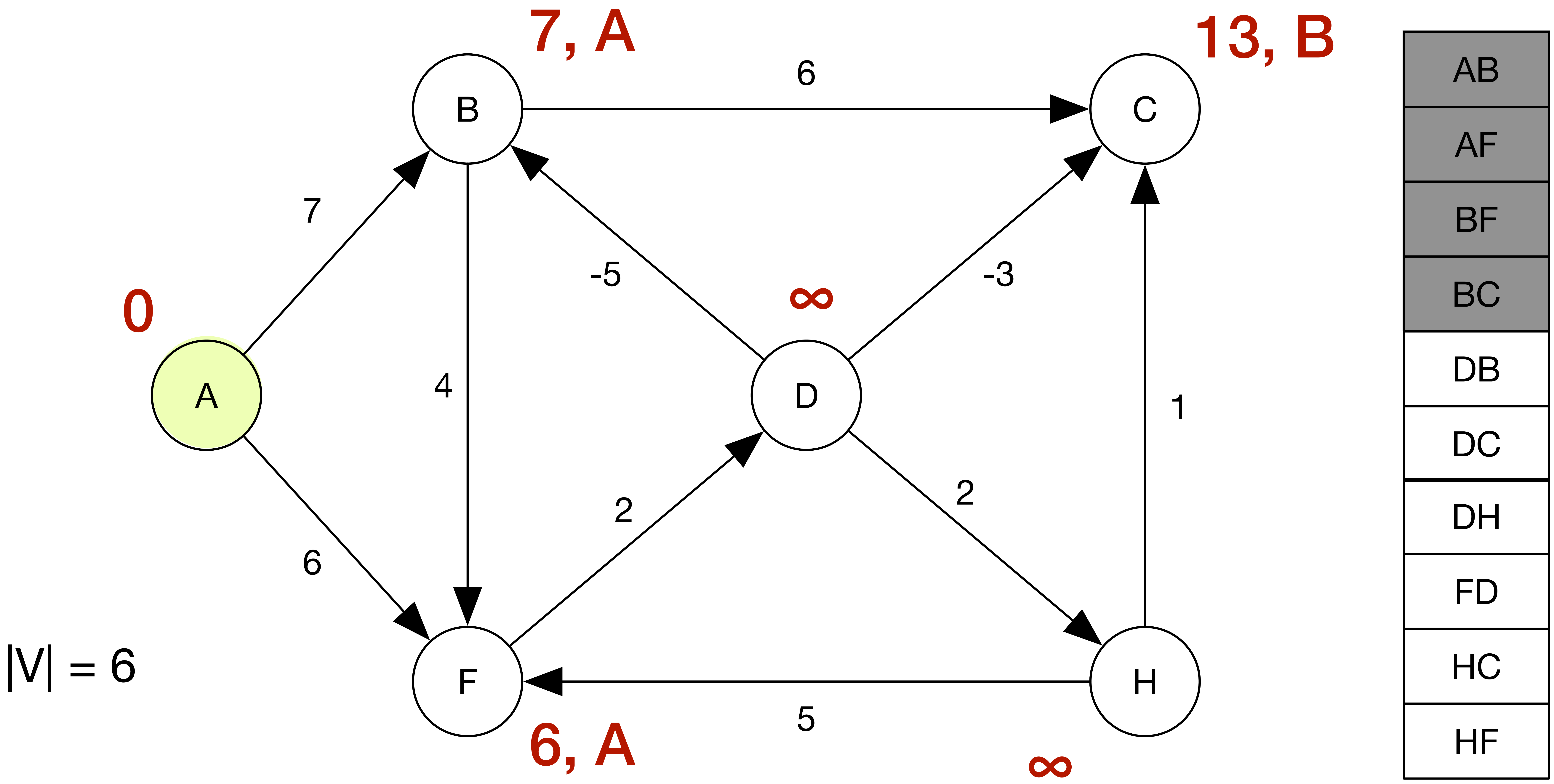
First iteration



|V| = 6

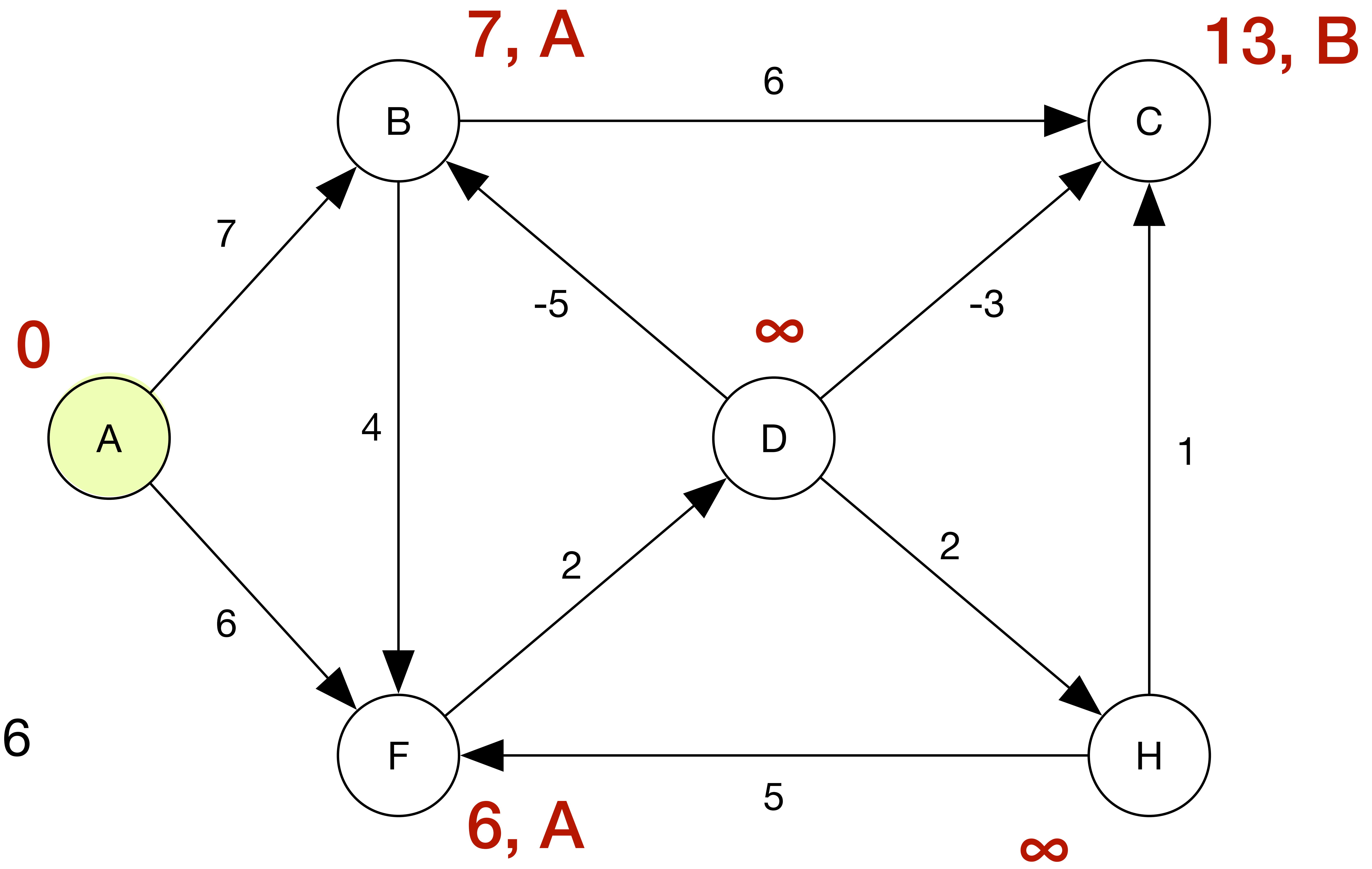
Bellman-Ford

First iteration



Bellman-Ford

First iteration

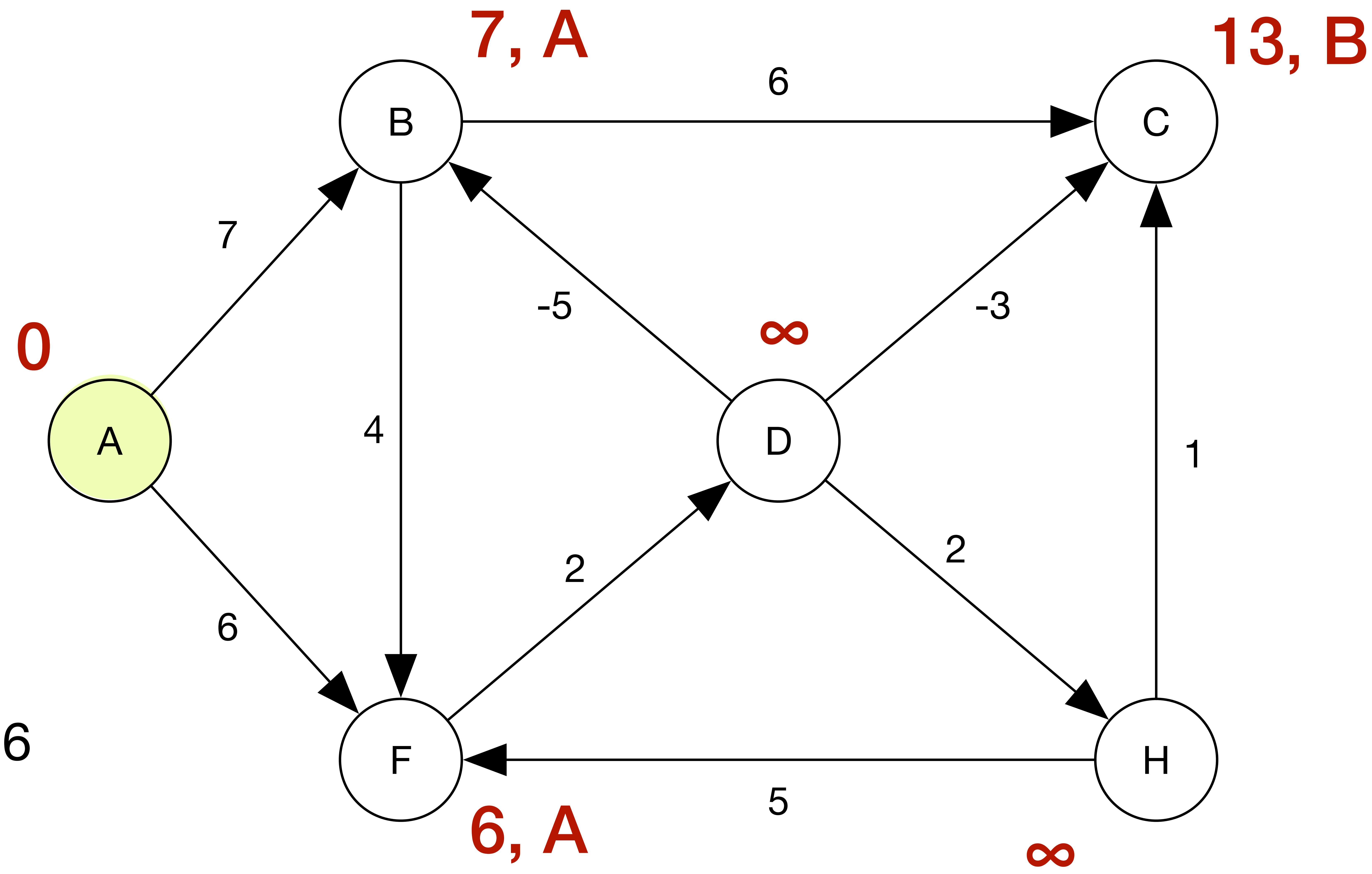


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

First iteration

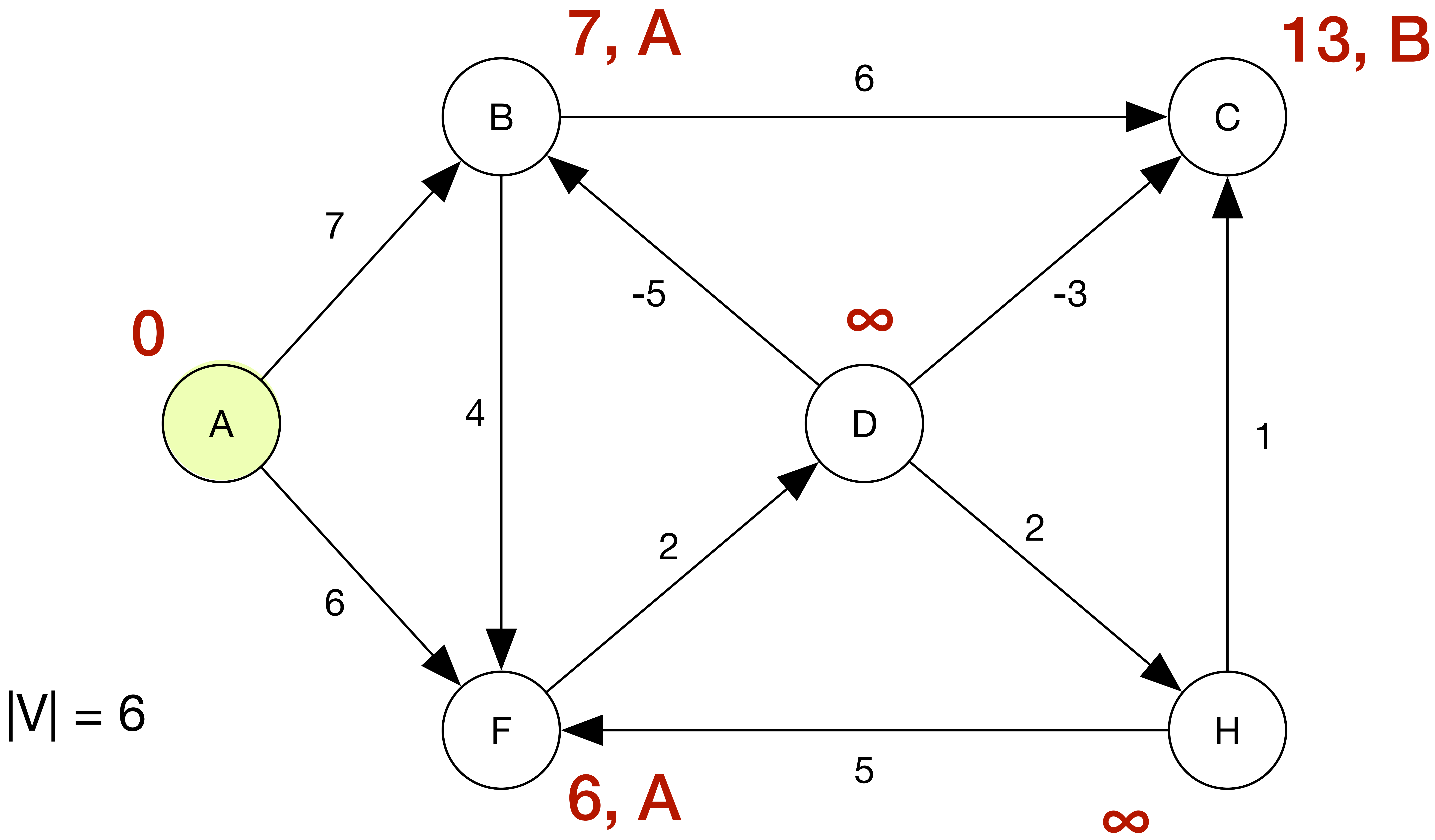


AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

$|V| = 6$

Bellman-Ford

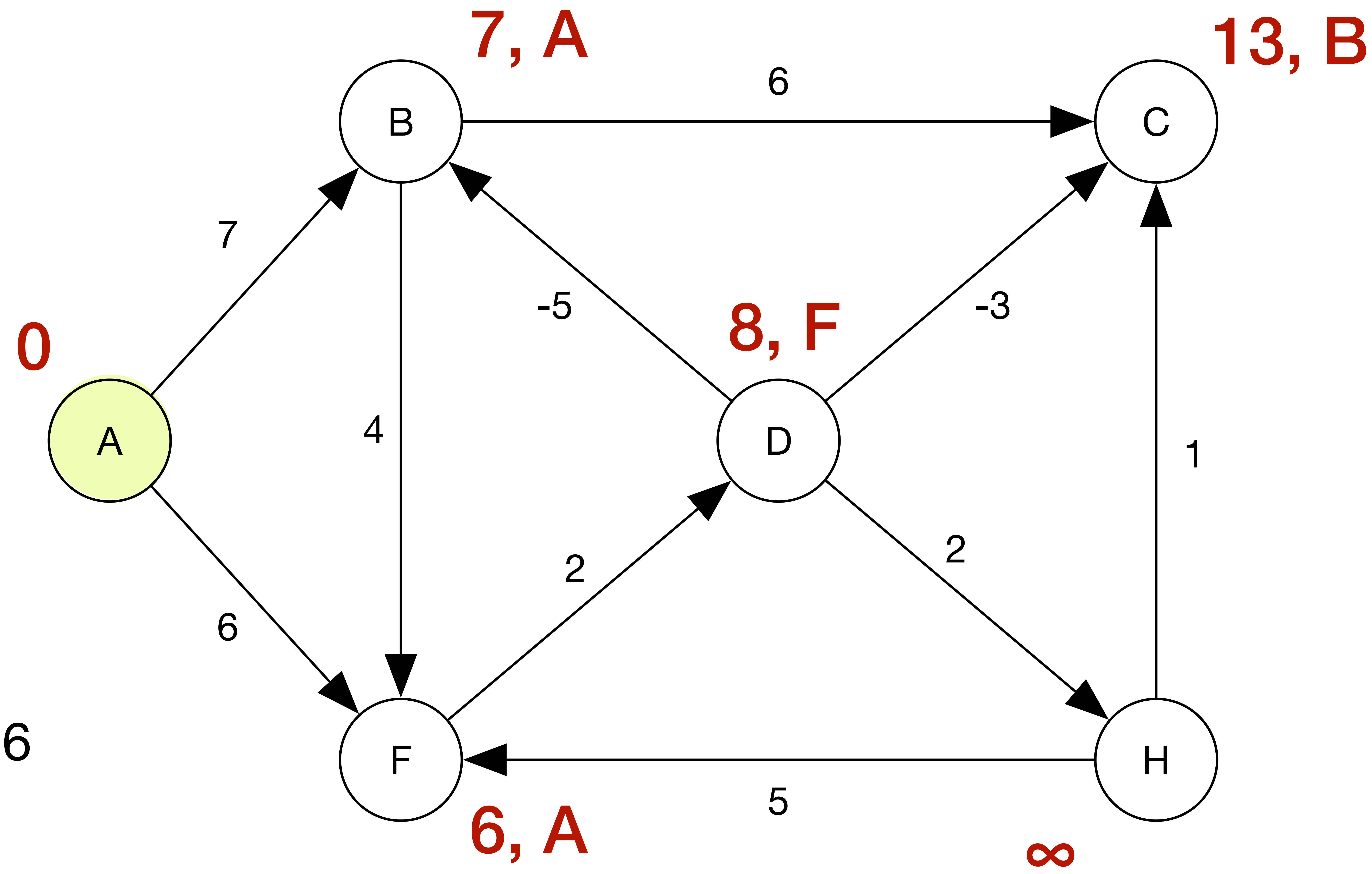
First iteration



AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

First iteration

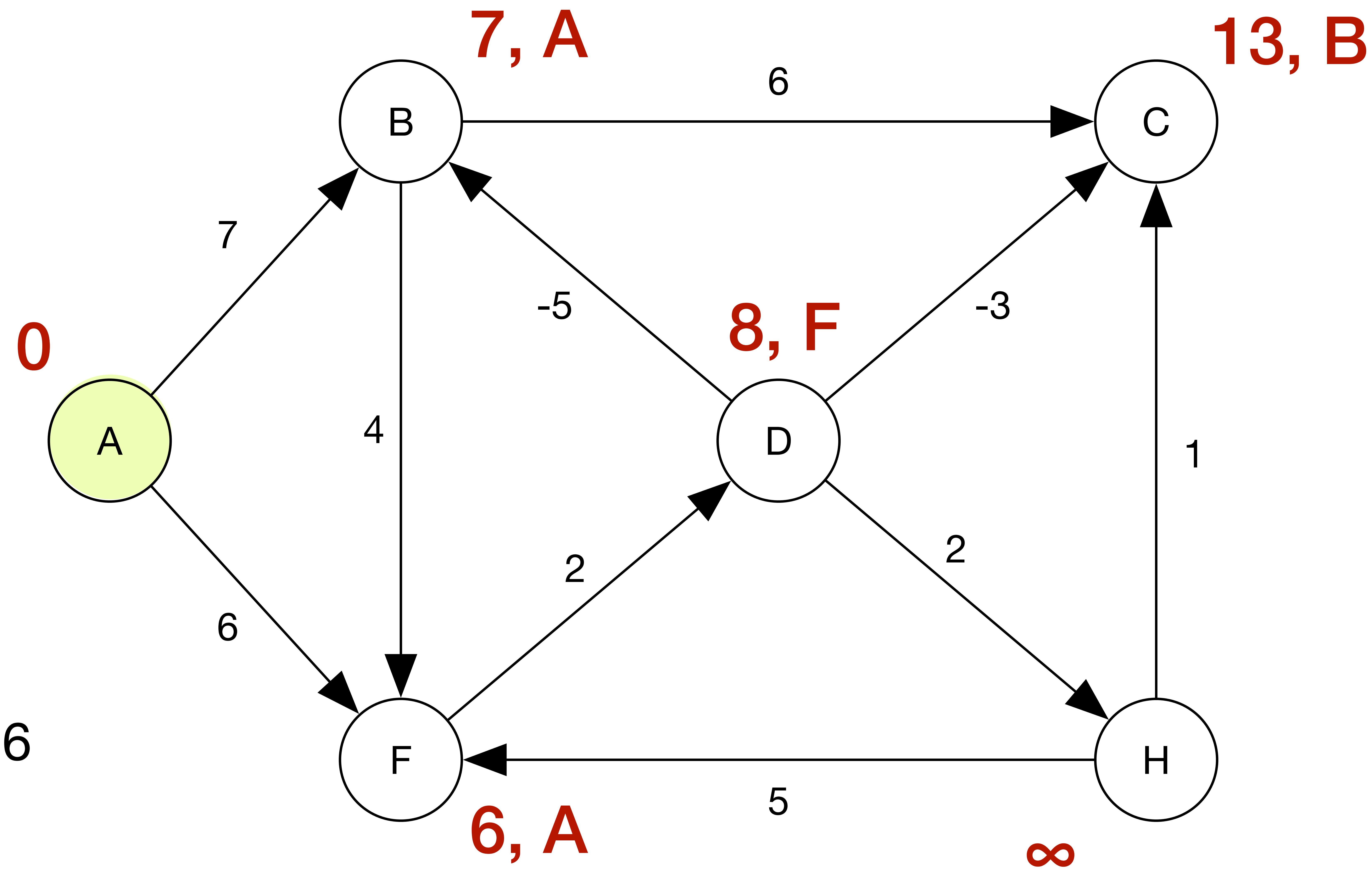


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

First iteration

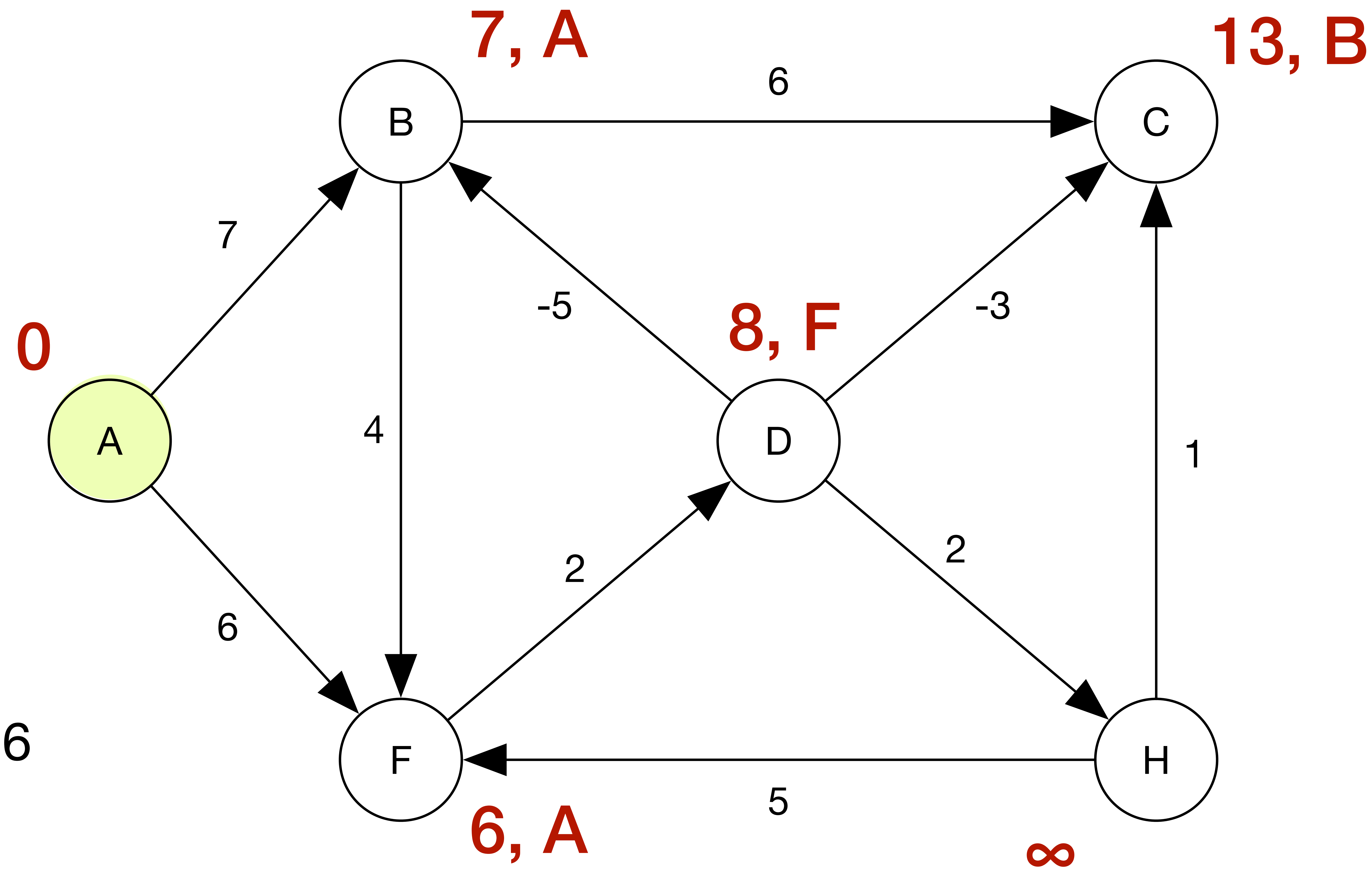


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

First iteration

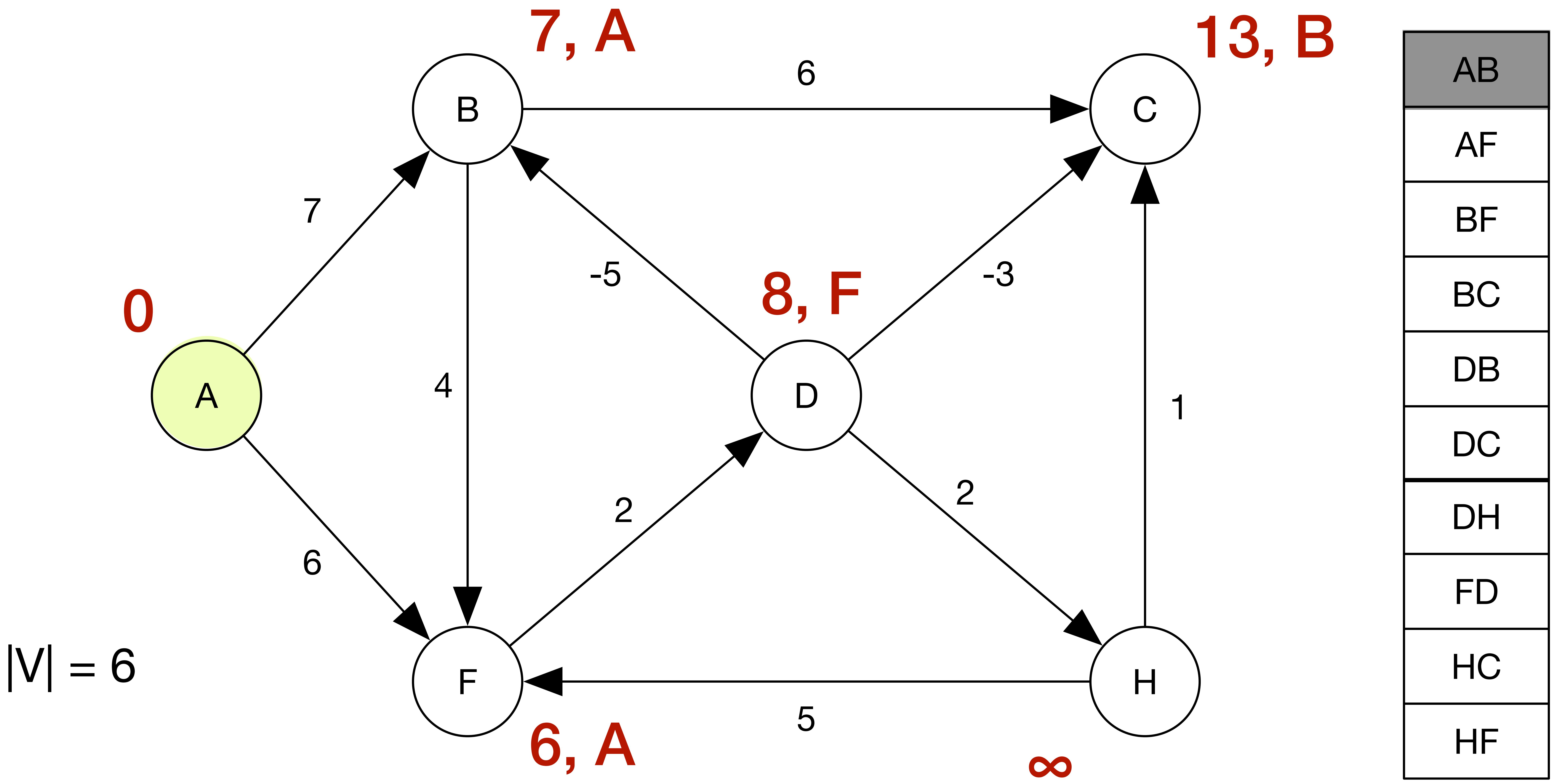


AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

$|V| = 6$

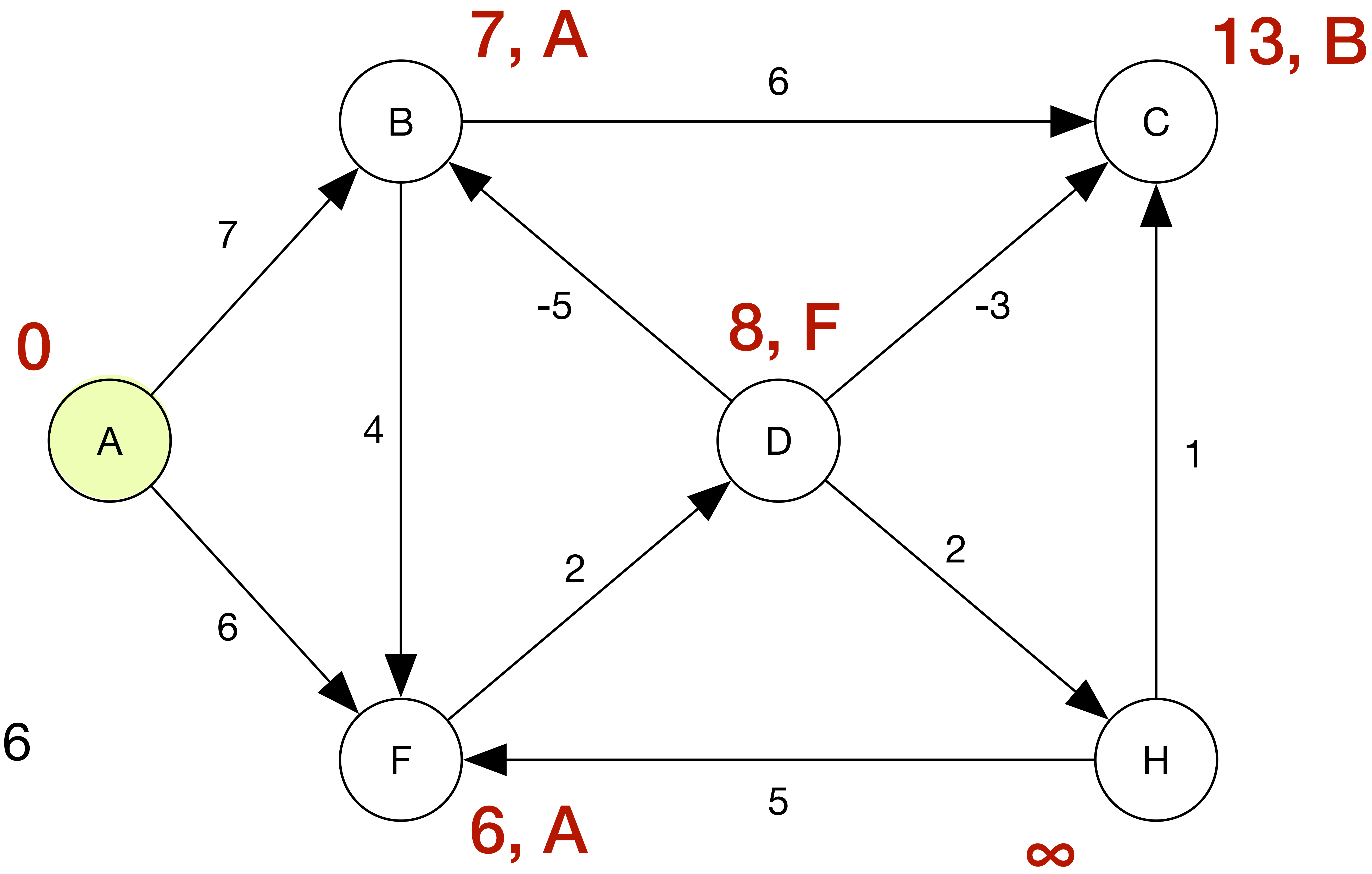
Bellman-Ford

Second iteration



Bellman-Ford

Second iteration

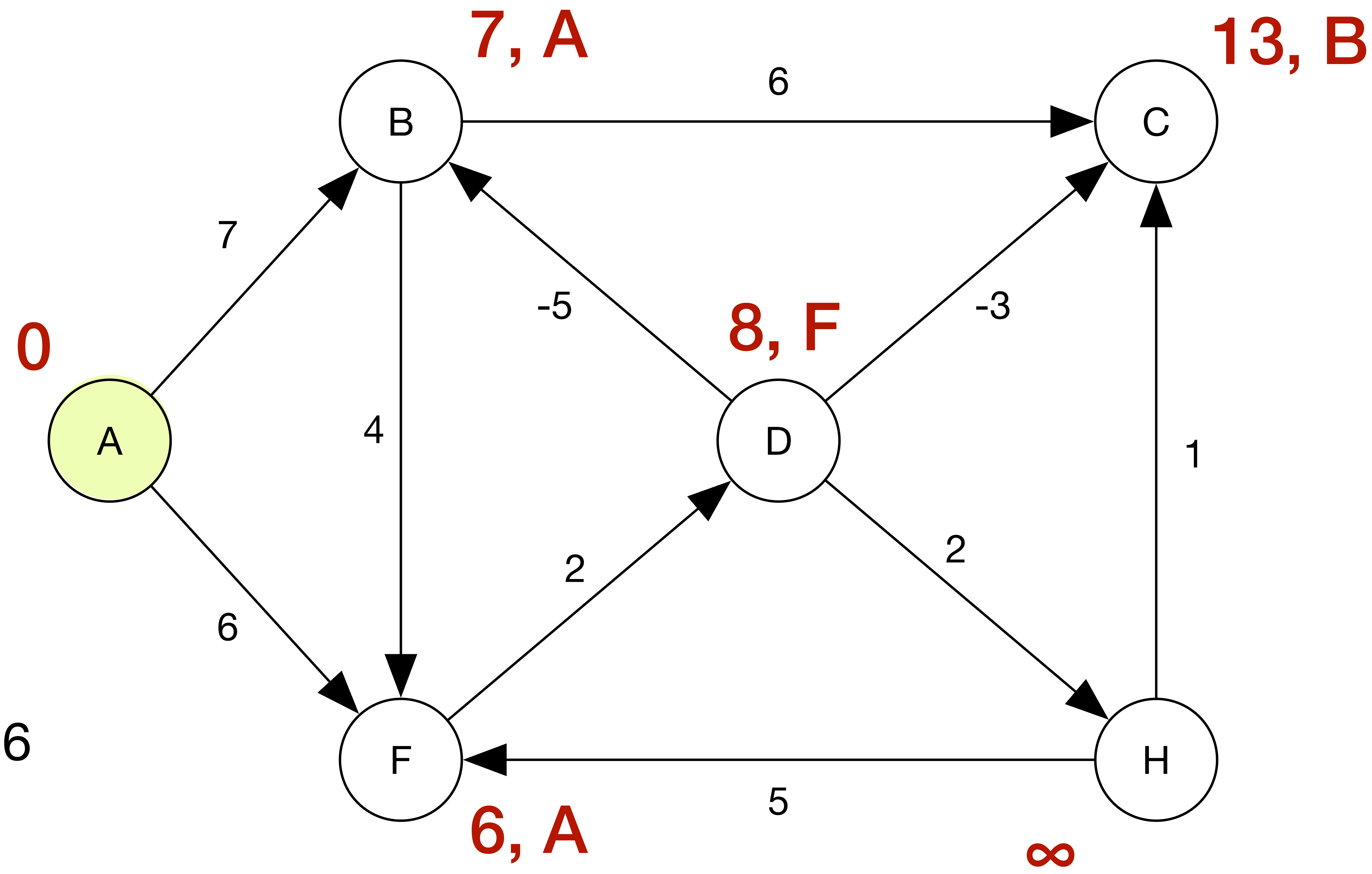


AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

|V| = 6

Bellman-Ford

Second iteration

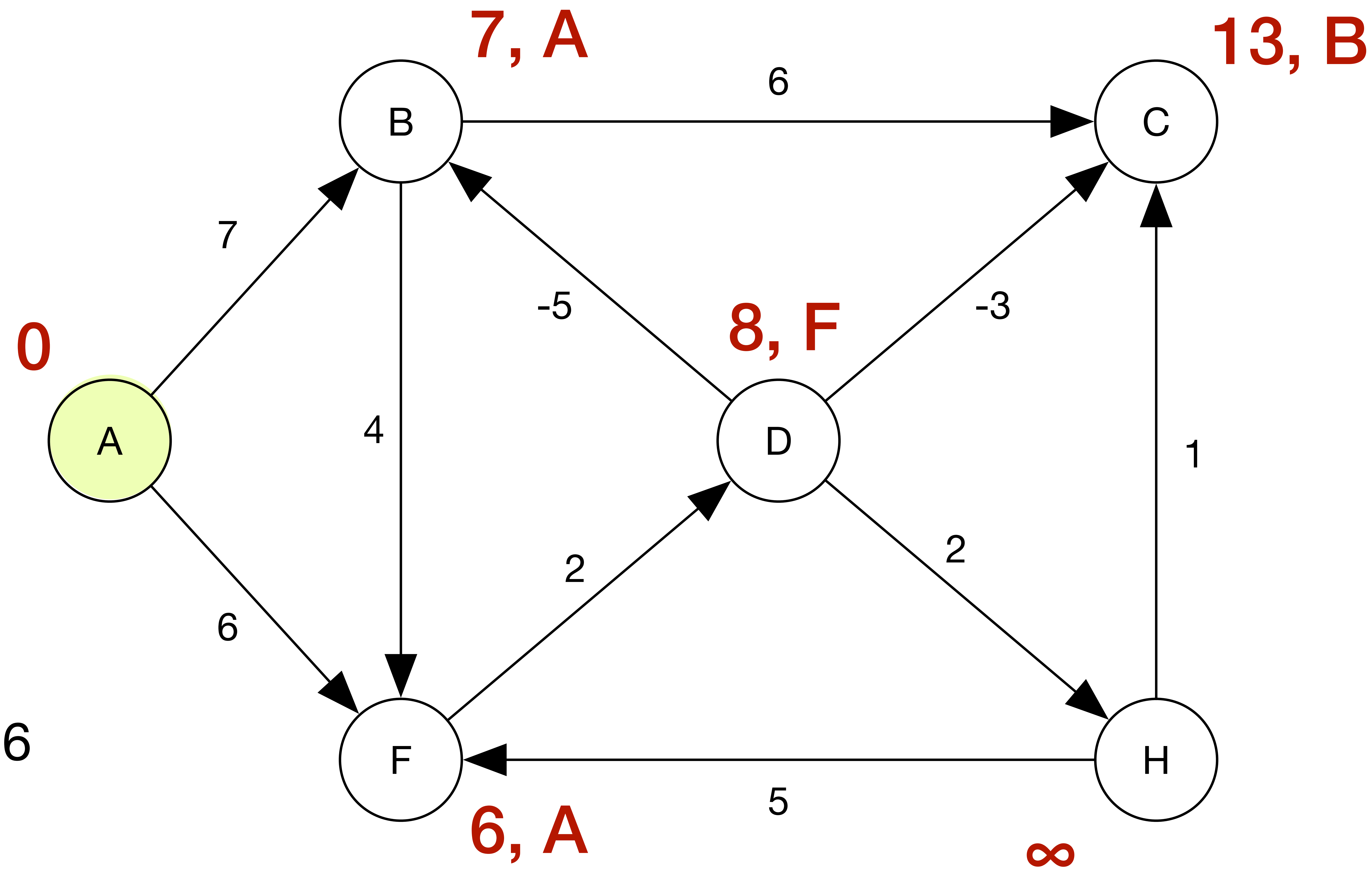


AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

$|V| = 6$

Bellman-Ford

Second iteration

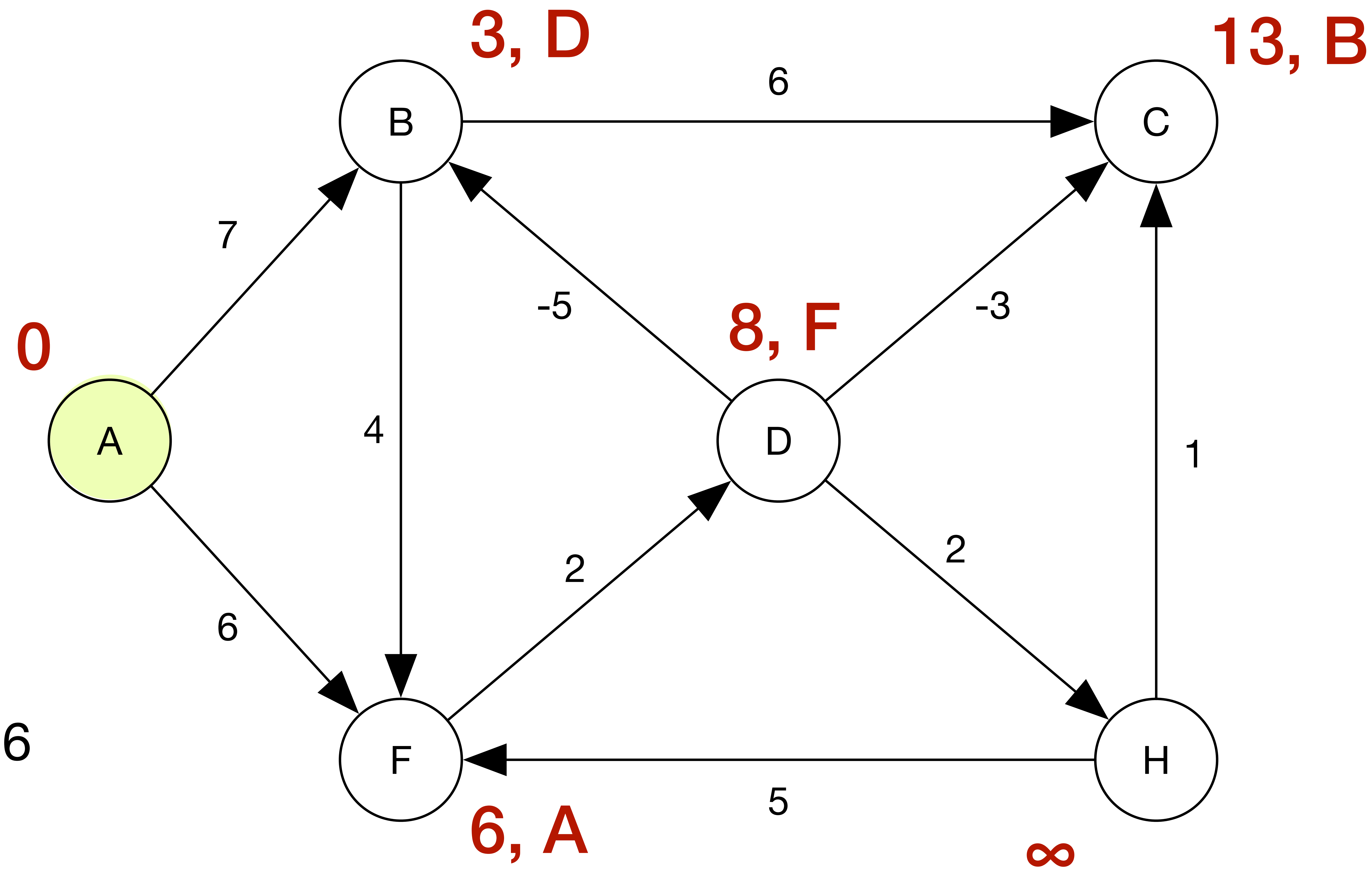


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Second iteration

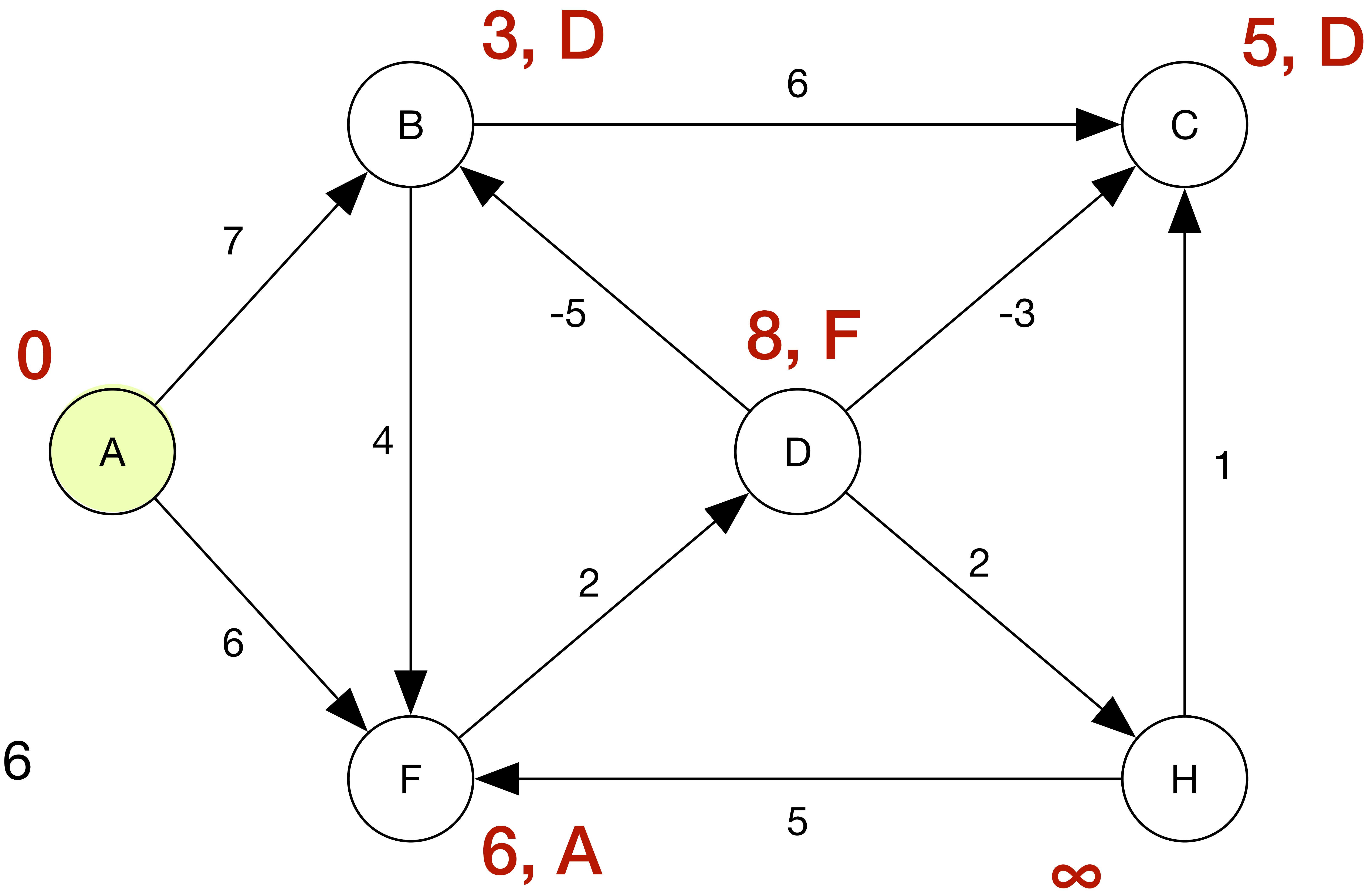


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Second iteration

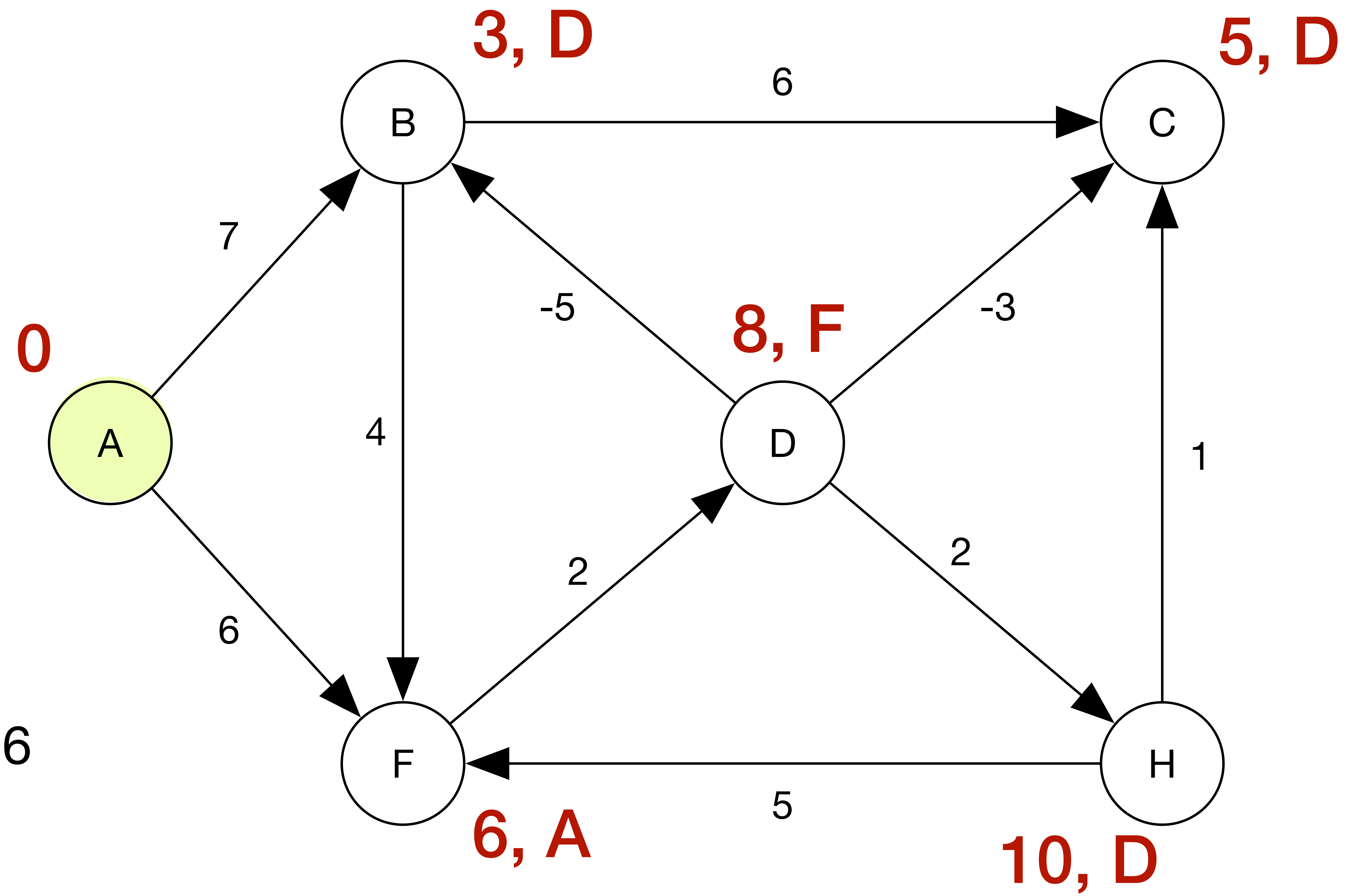


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Second iteration

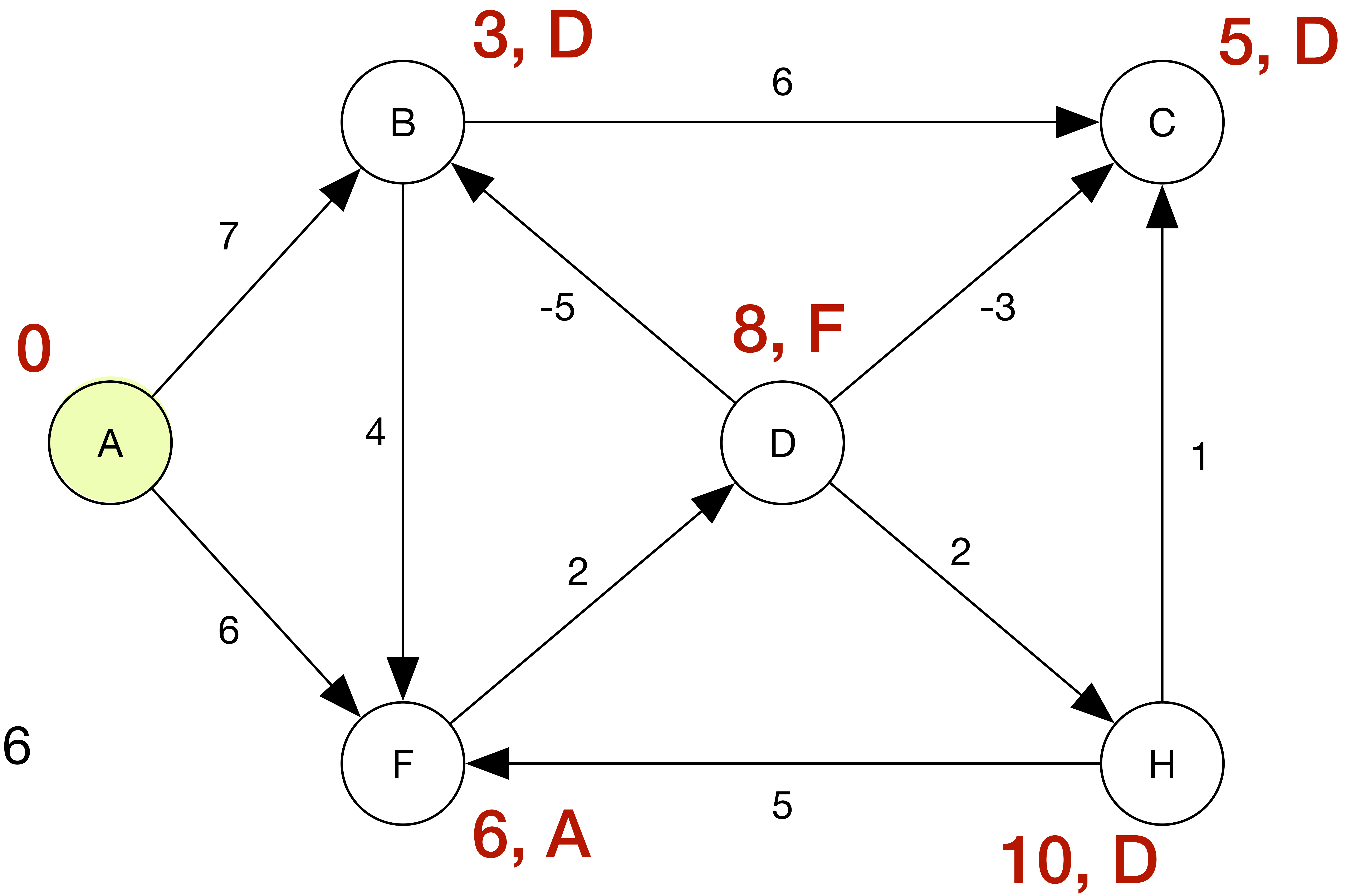


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Second iteration

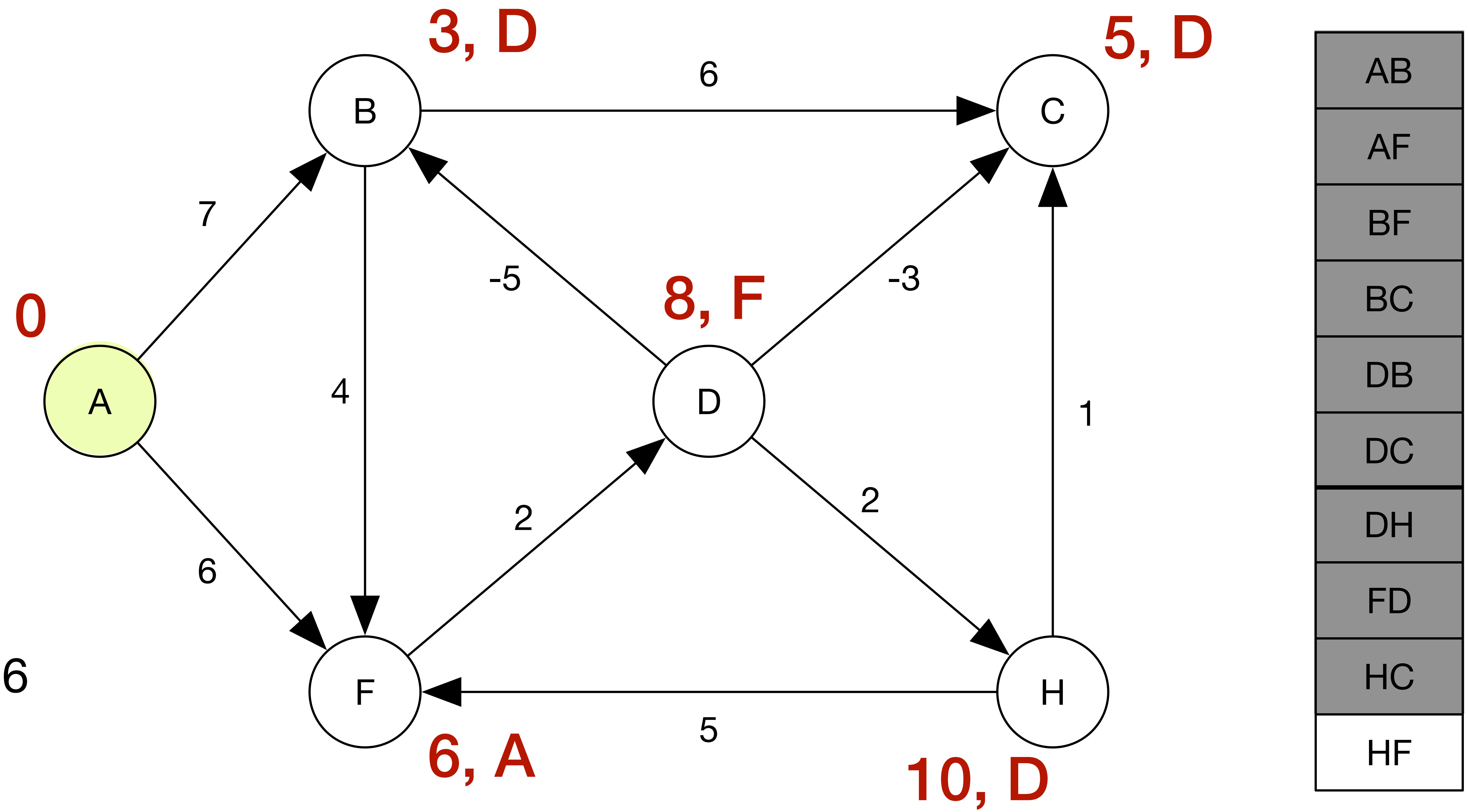


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

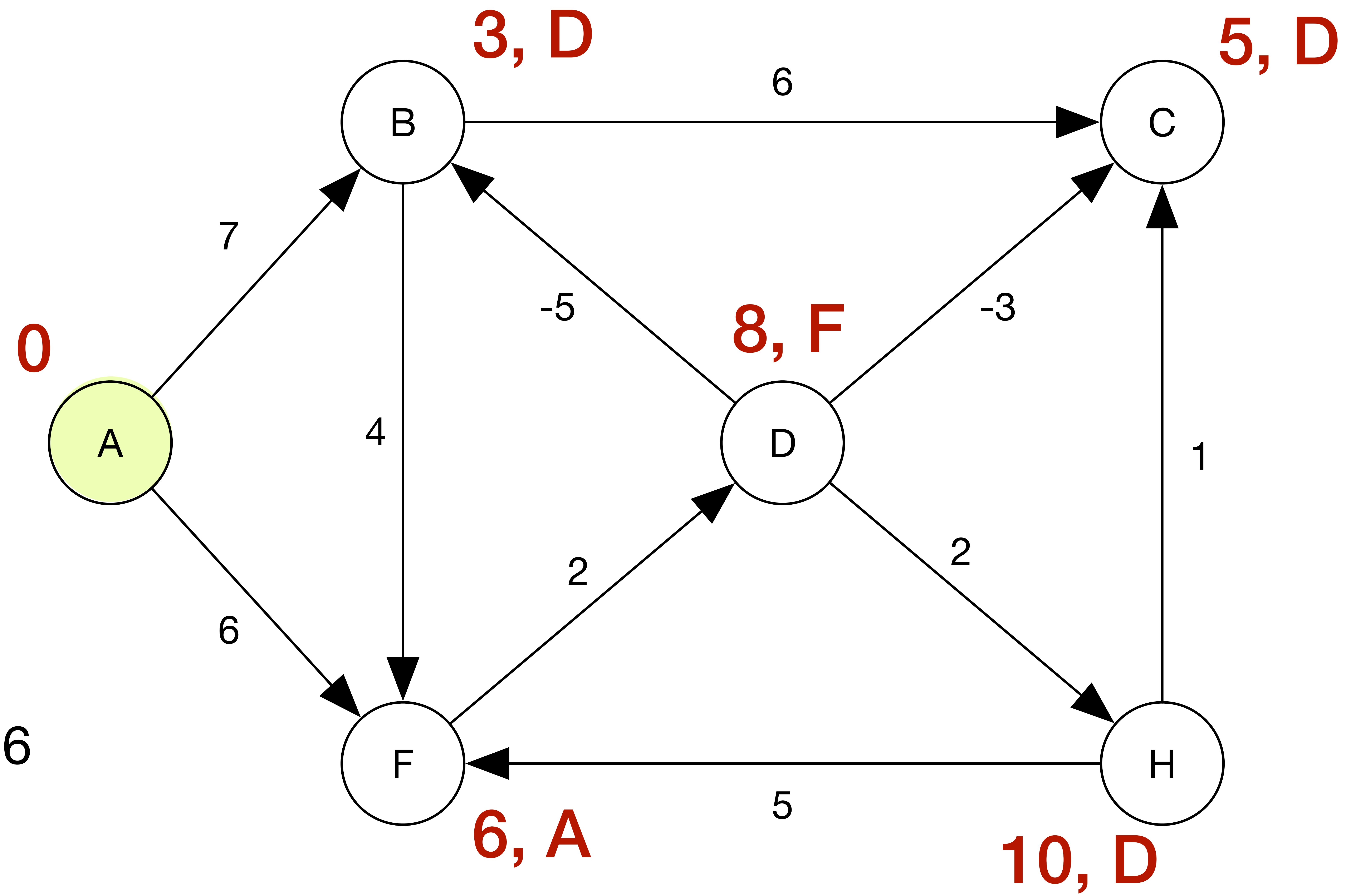
Second iteration



|V| = 6

Bellman-Ford

Second iteration

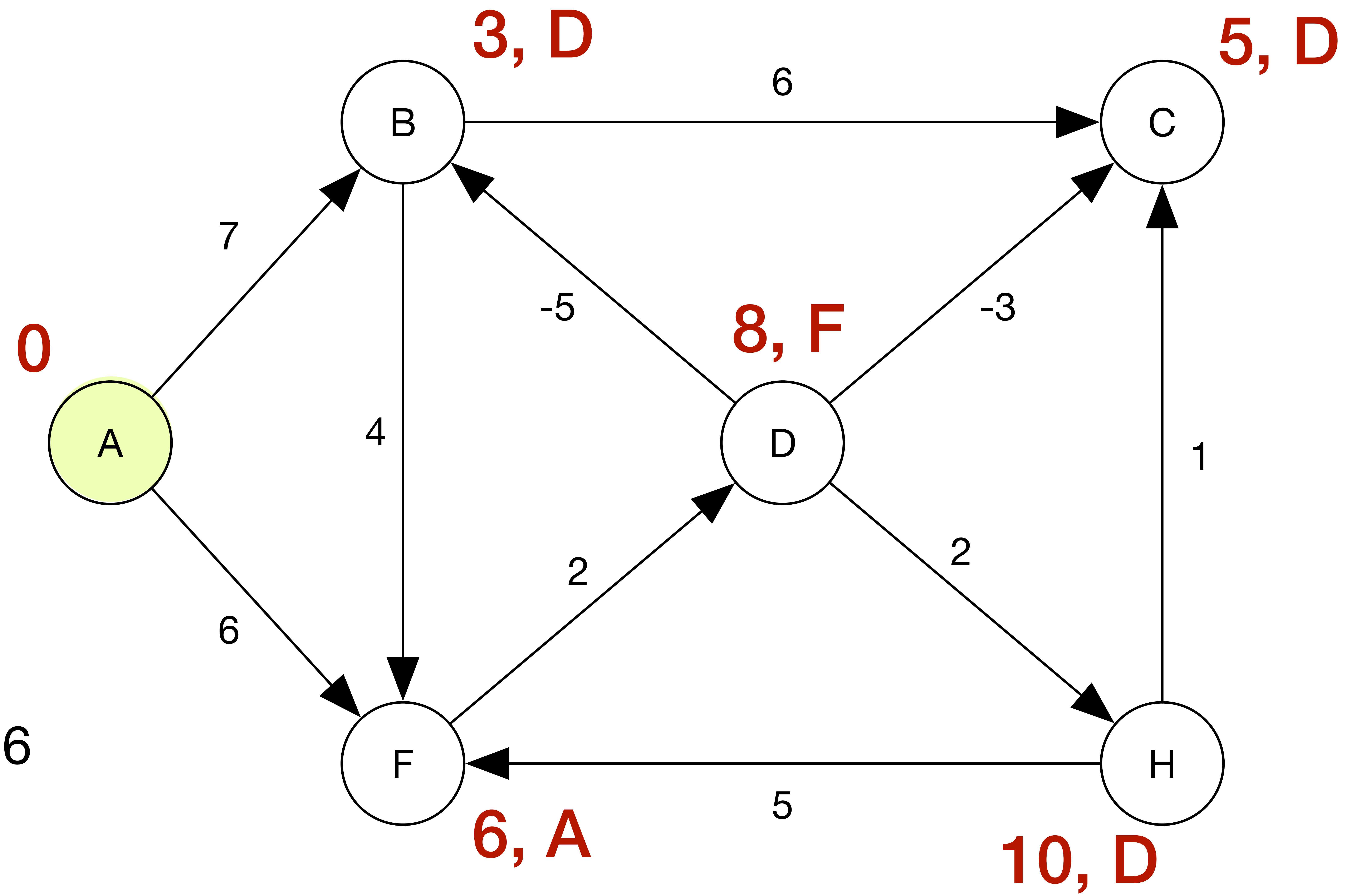


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Third iteration

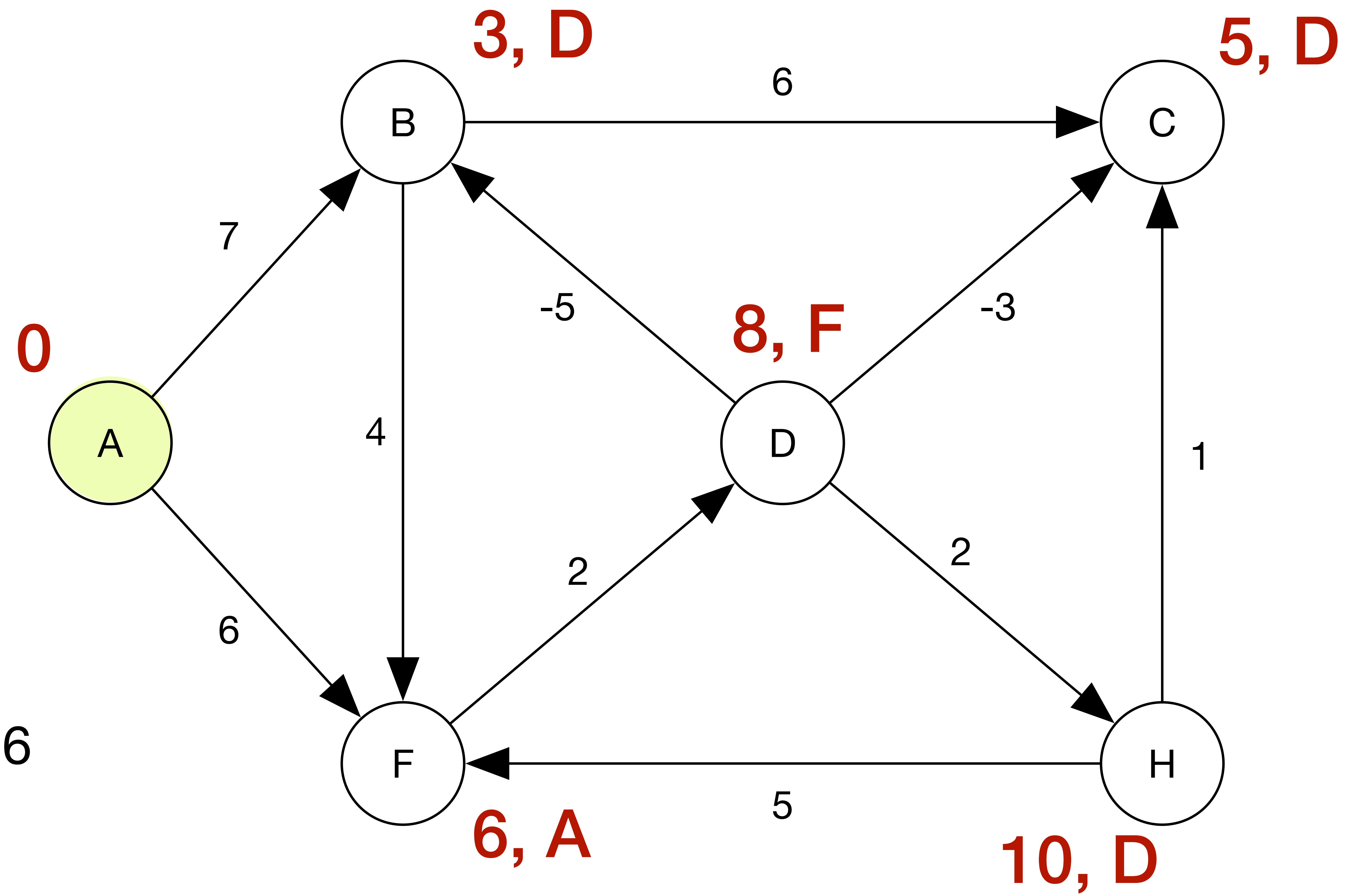


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Third iteration

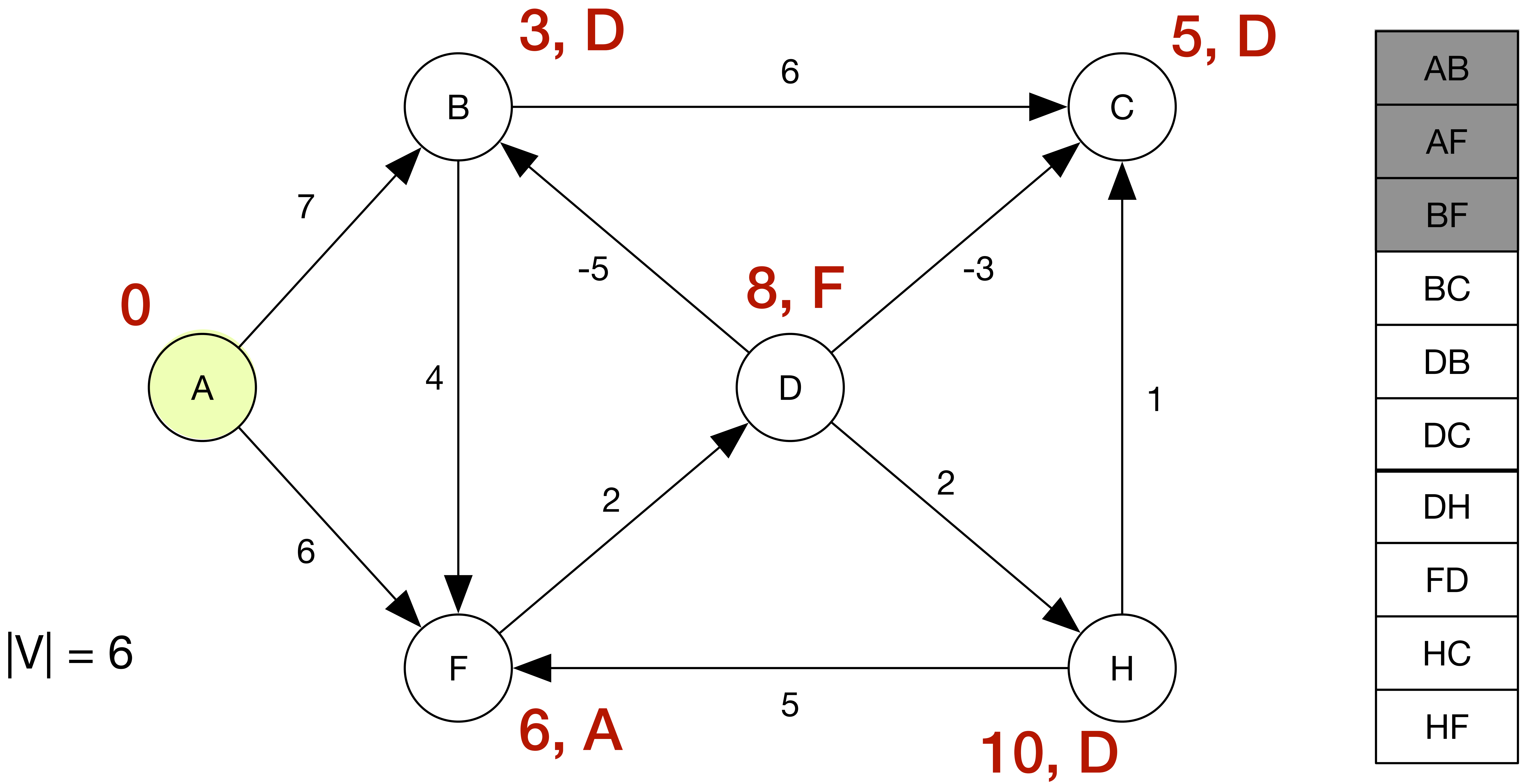


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

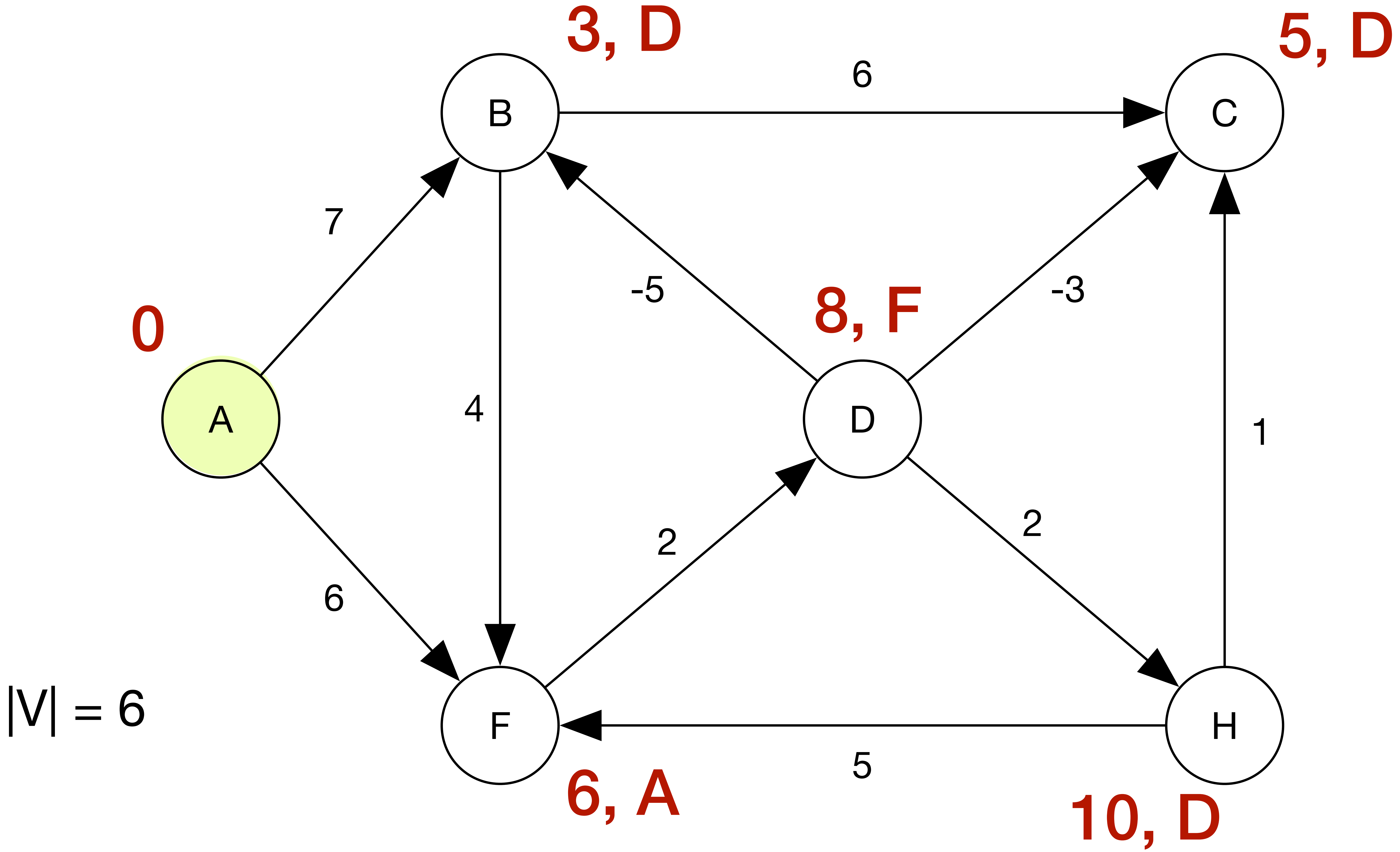
Bellman-Ford

Third iteration



Bellman-Ford

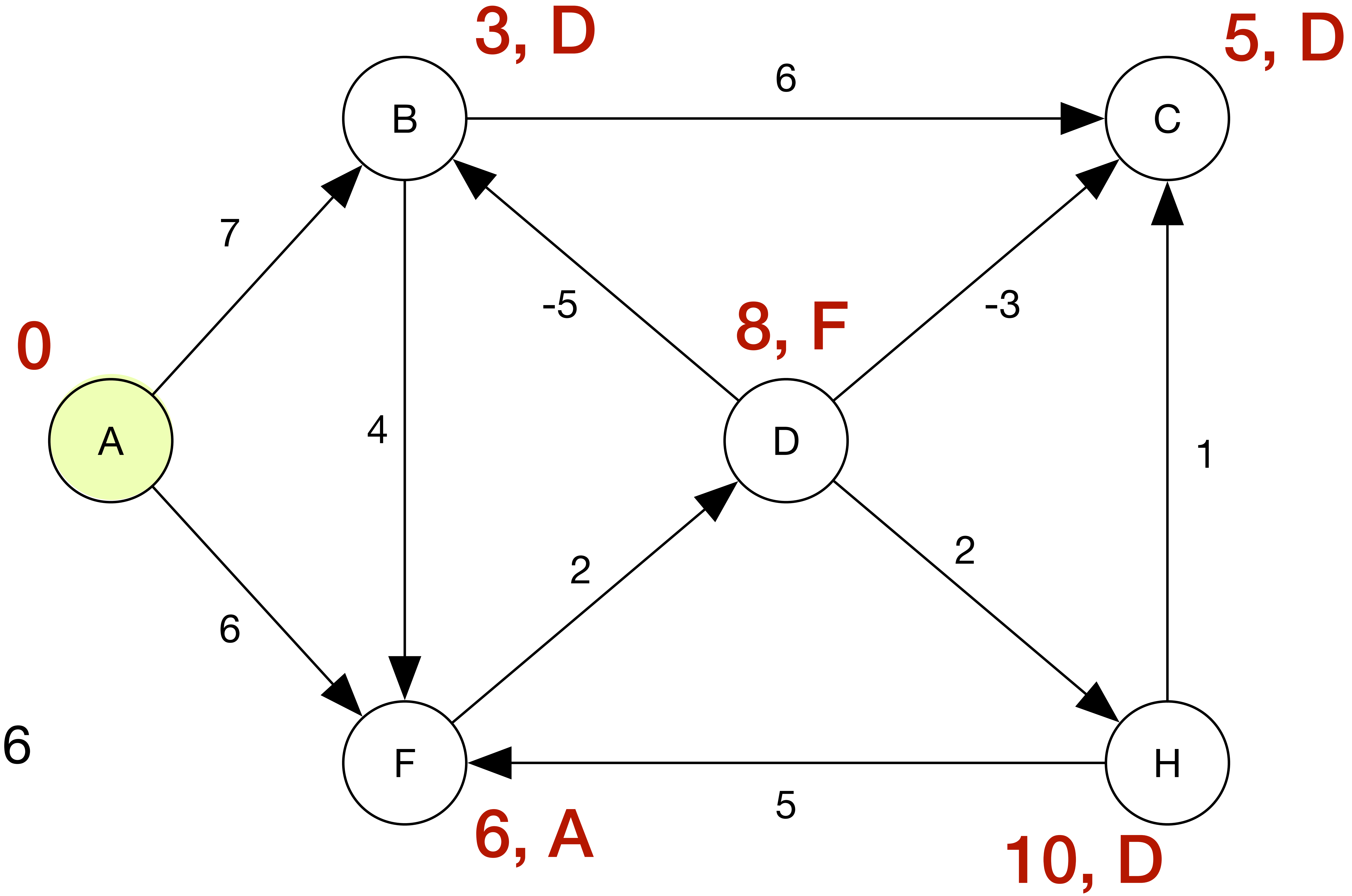
Third iteration



AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Third iteration

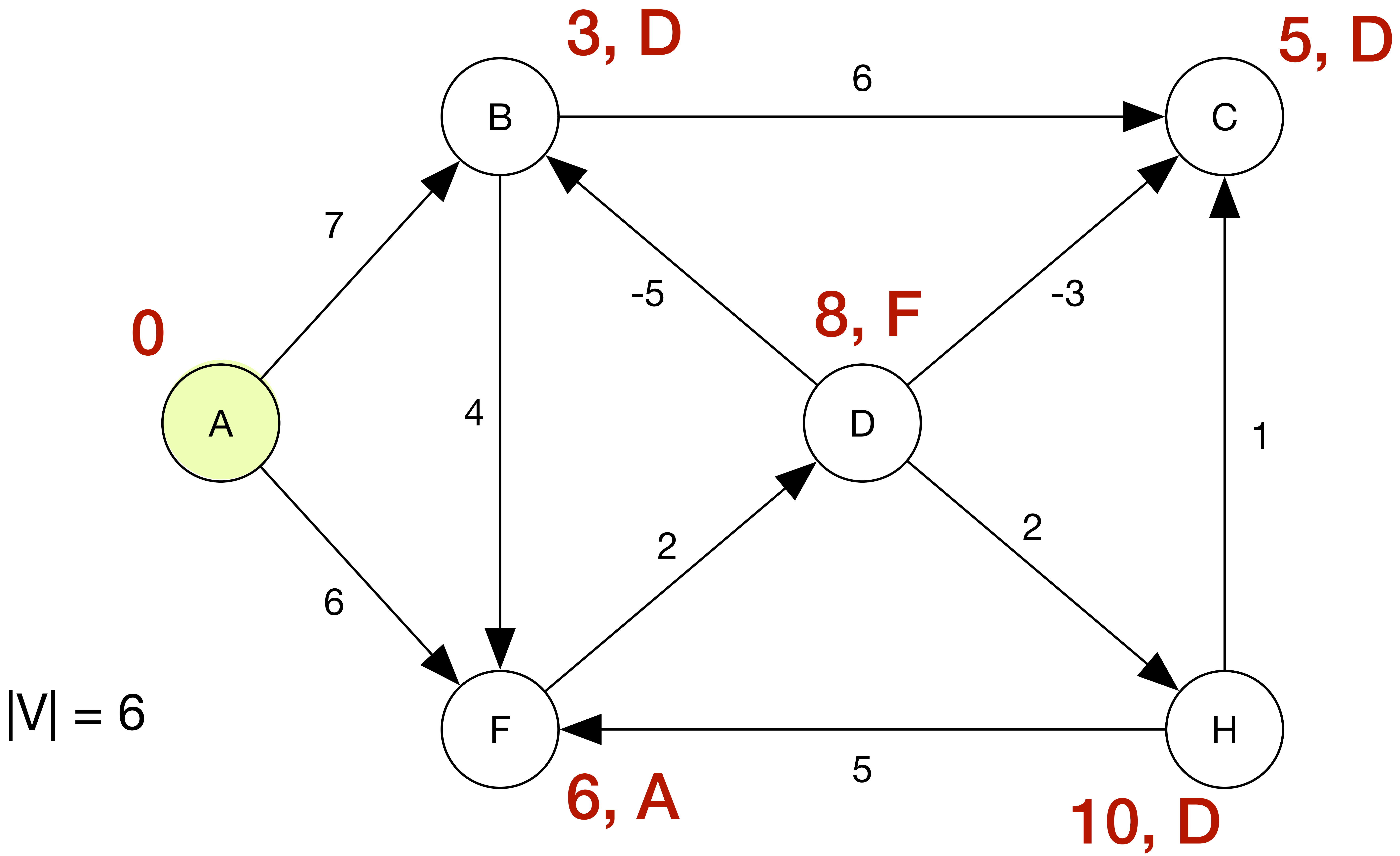


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

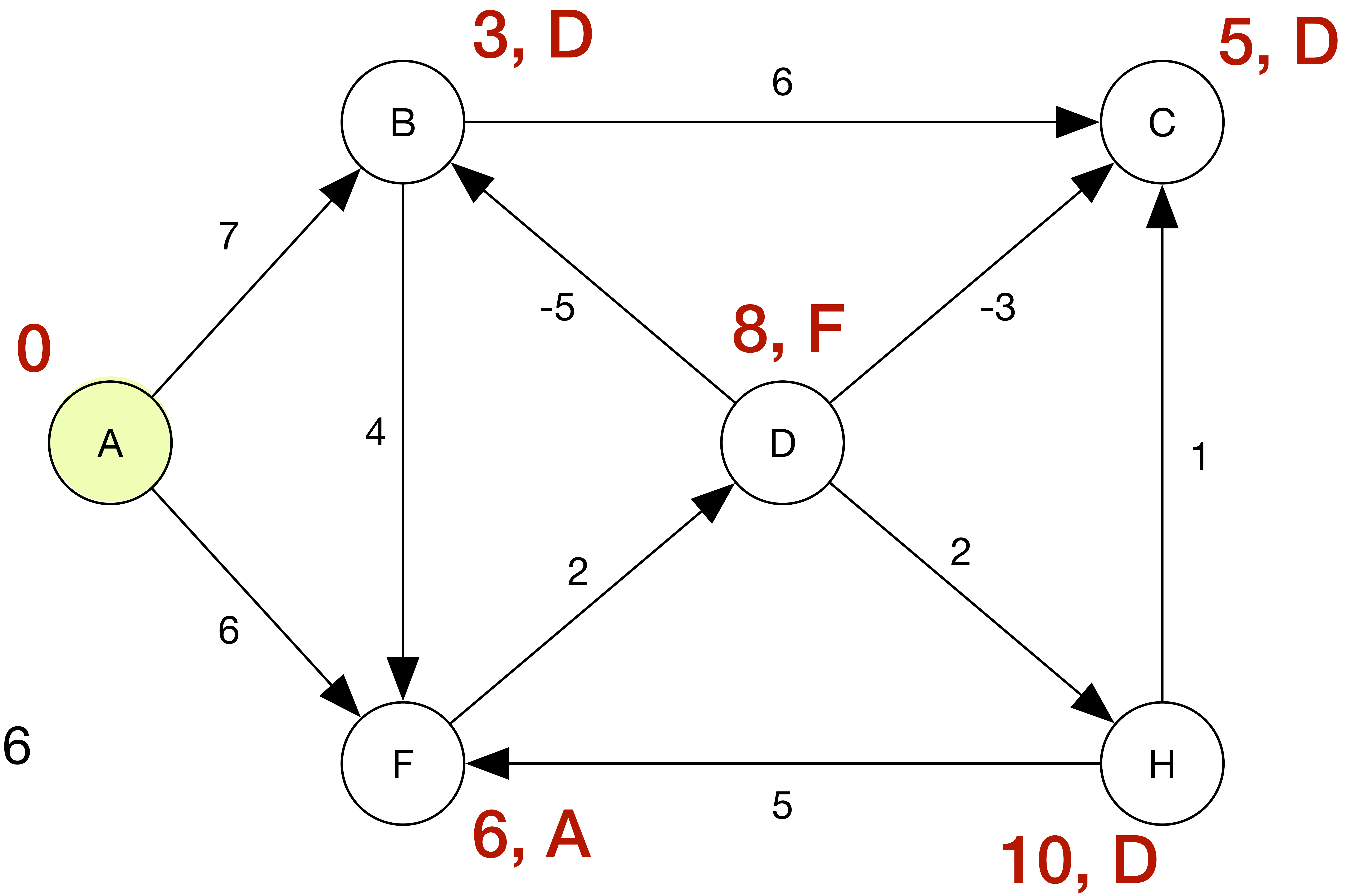
Third iteration



AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Third iteration

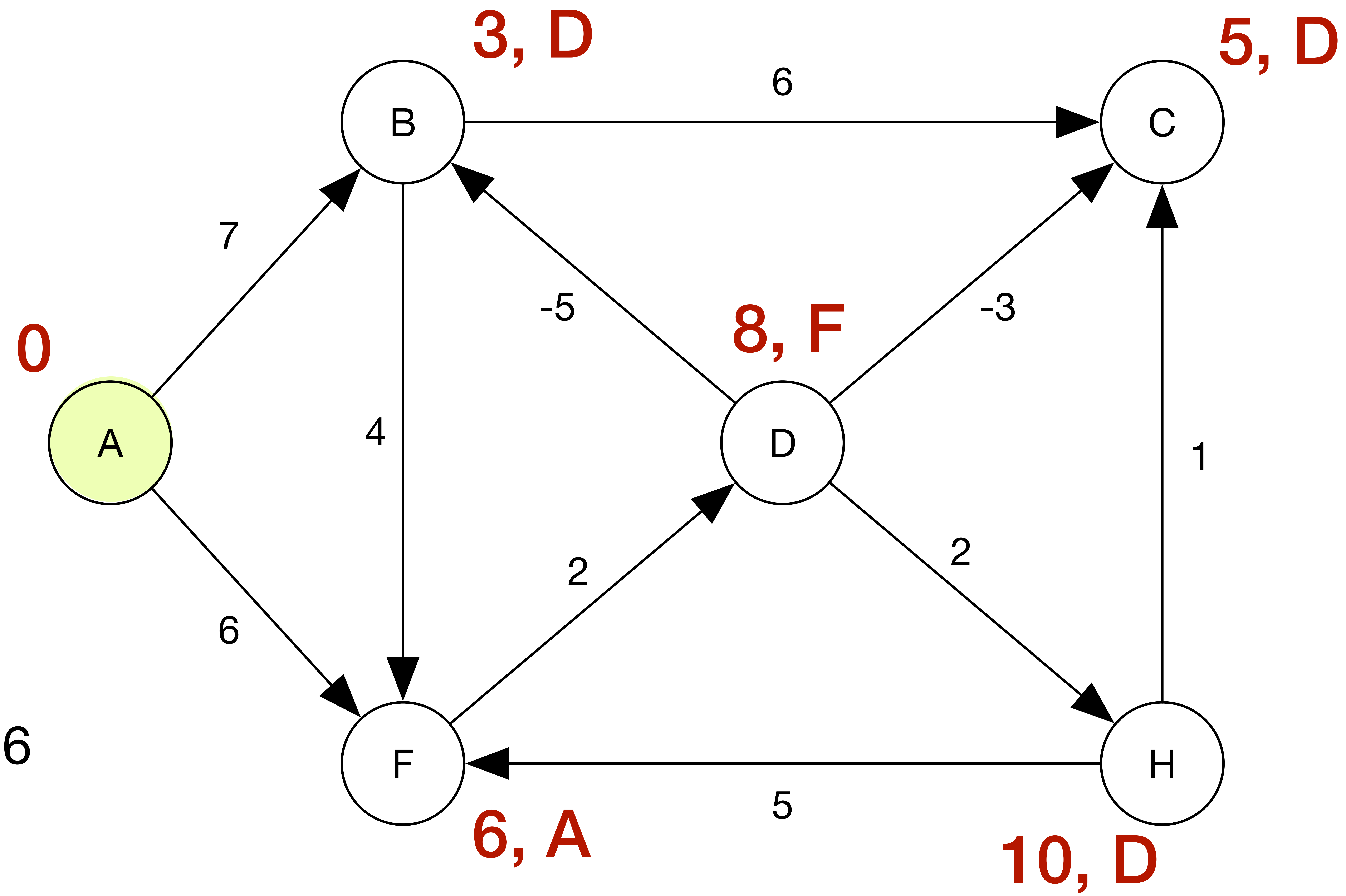


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Third iteration

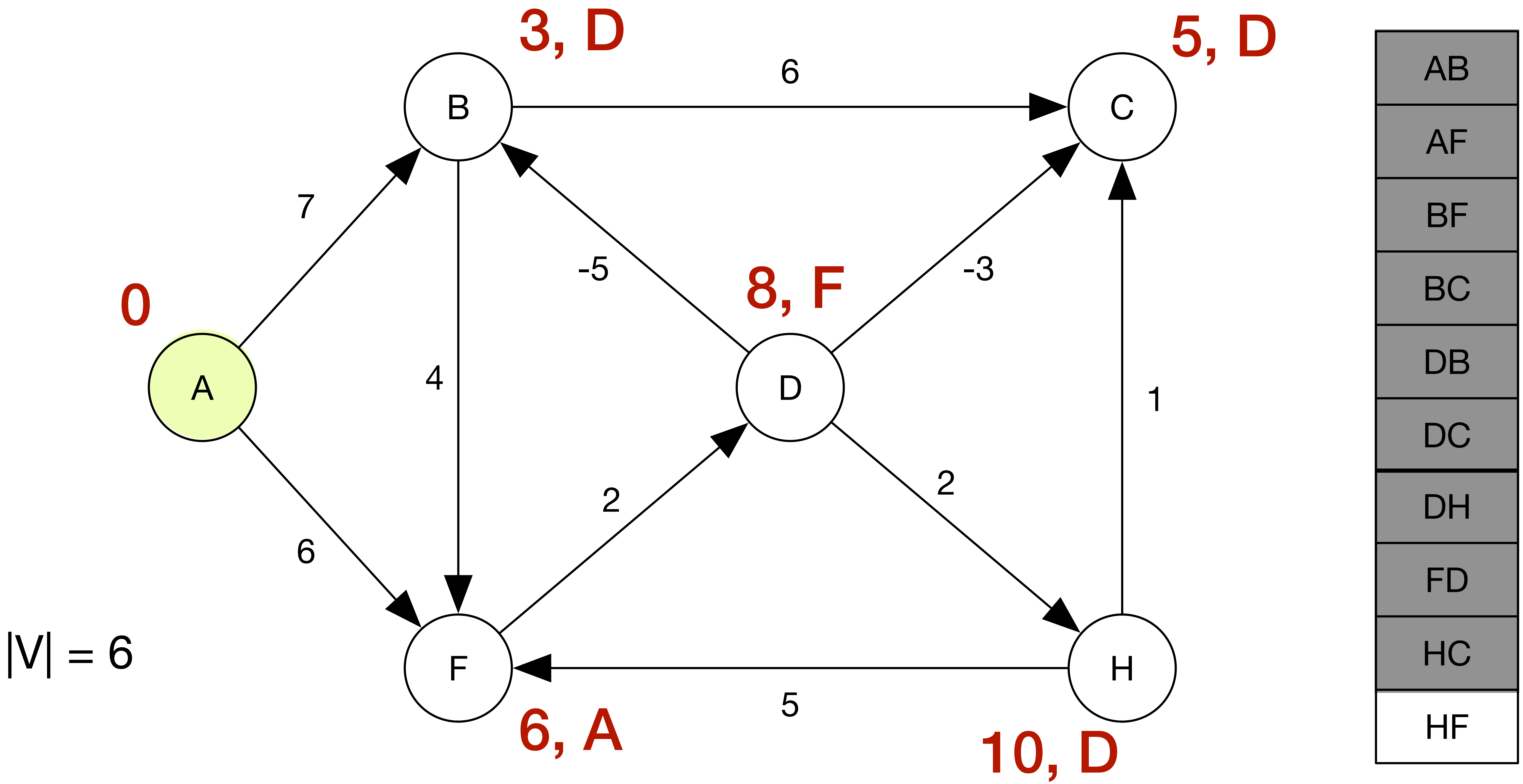


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

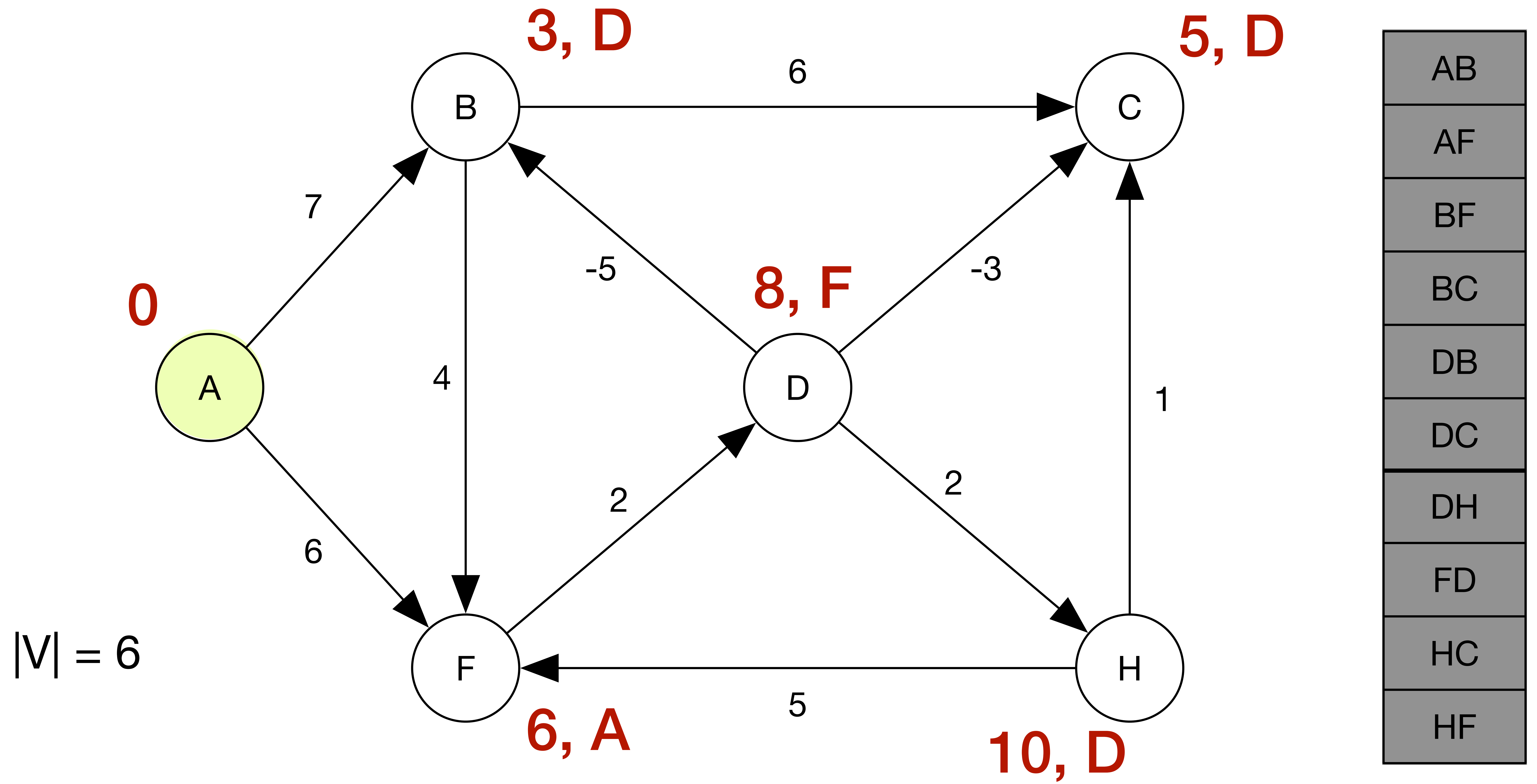
Bellman-Ford

Third iteration



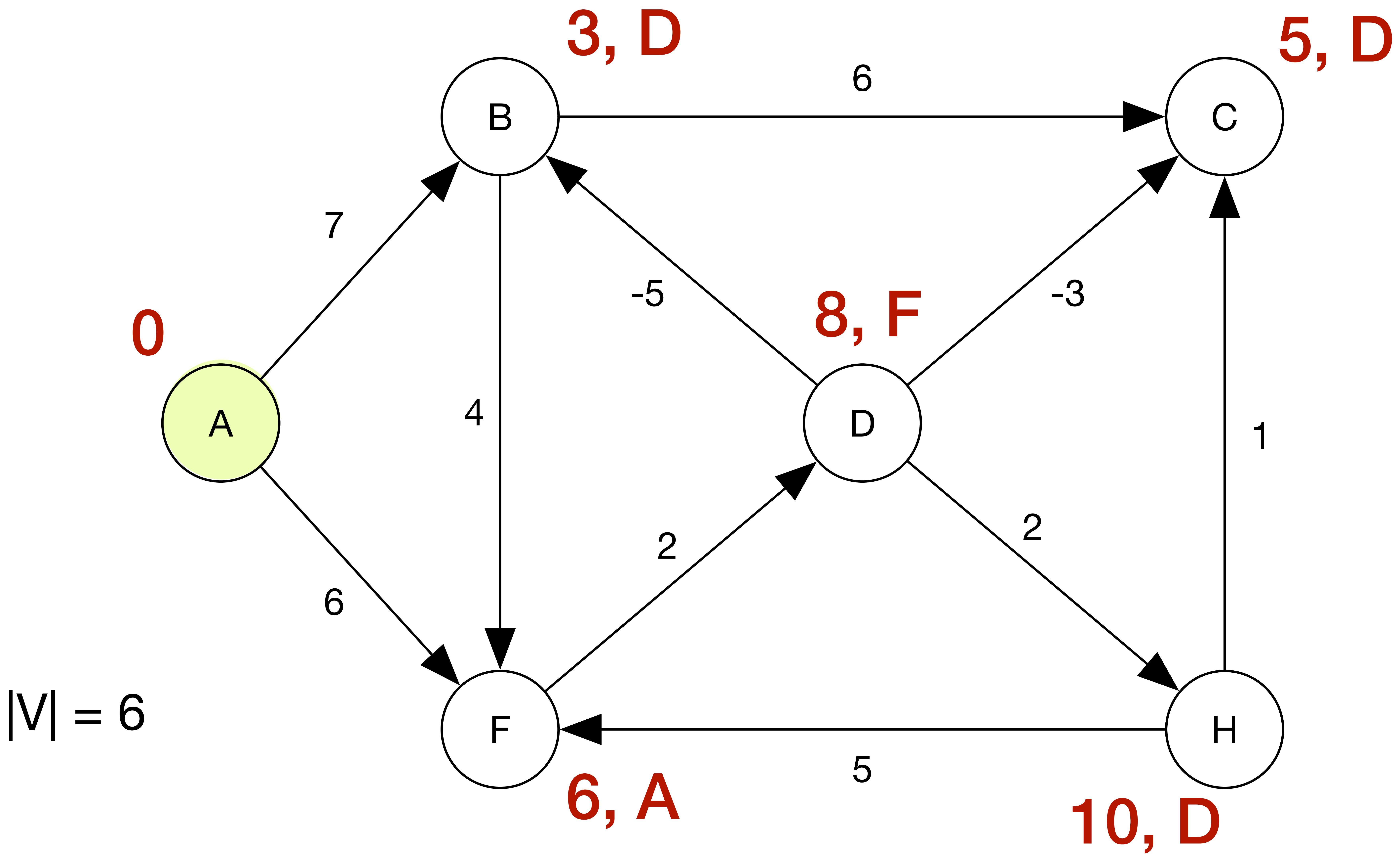
Bellman-Ford

Third iteration



Bellman-Ford

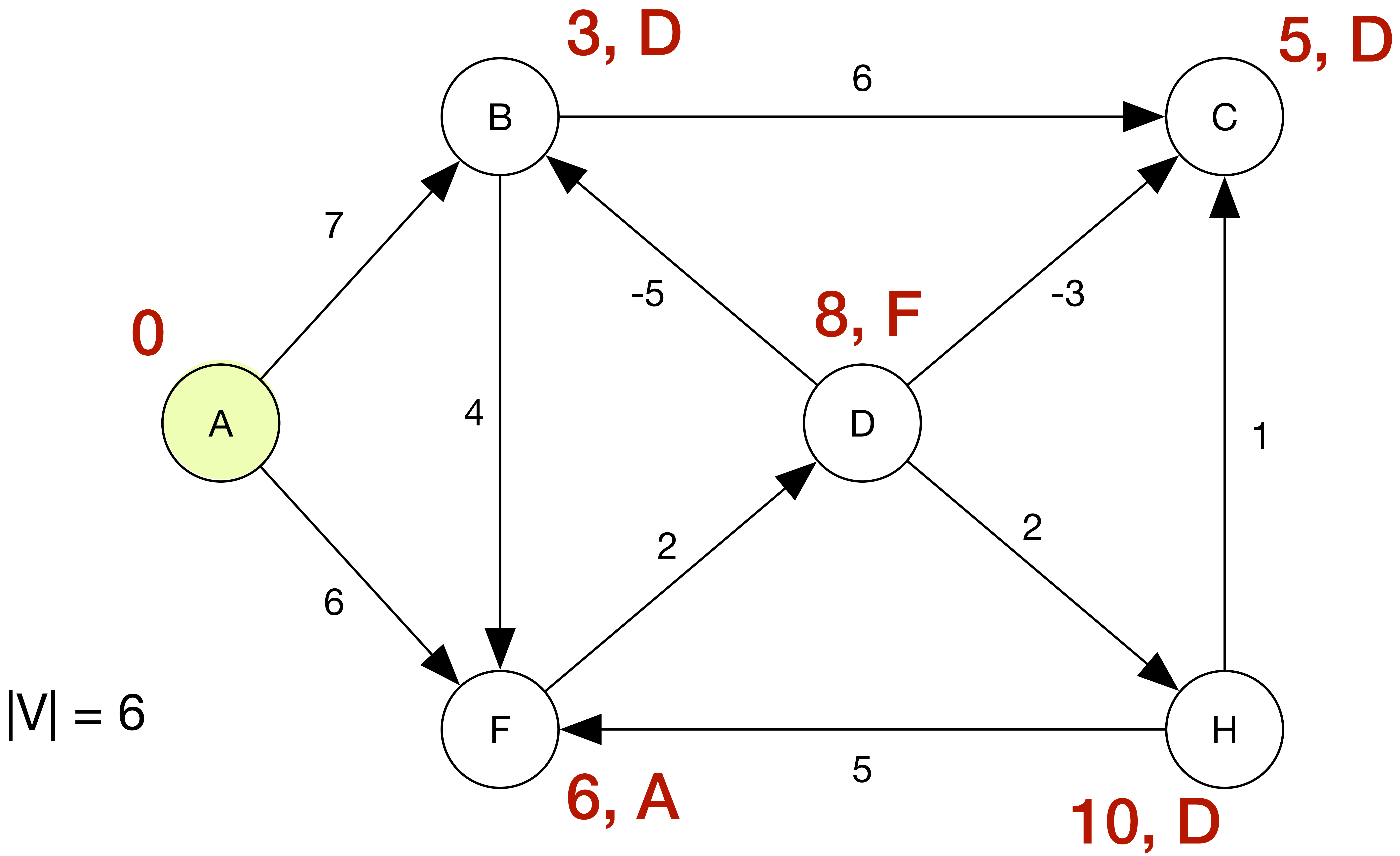
Check for negative cycles



AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

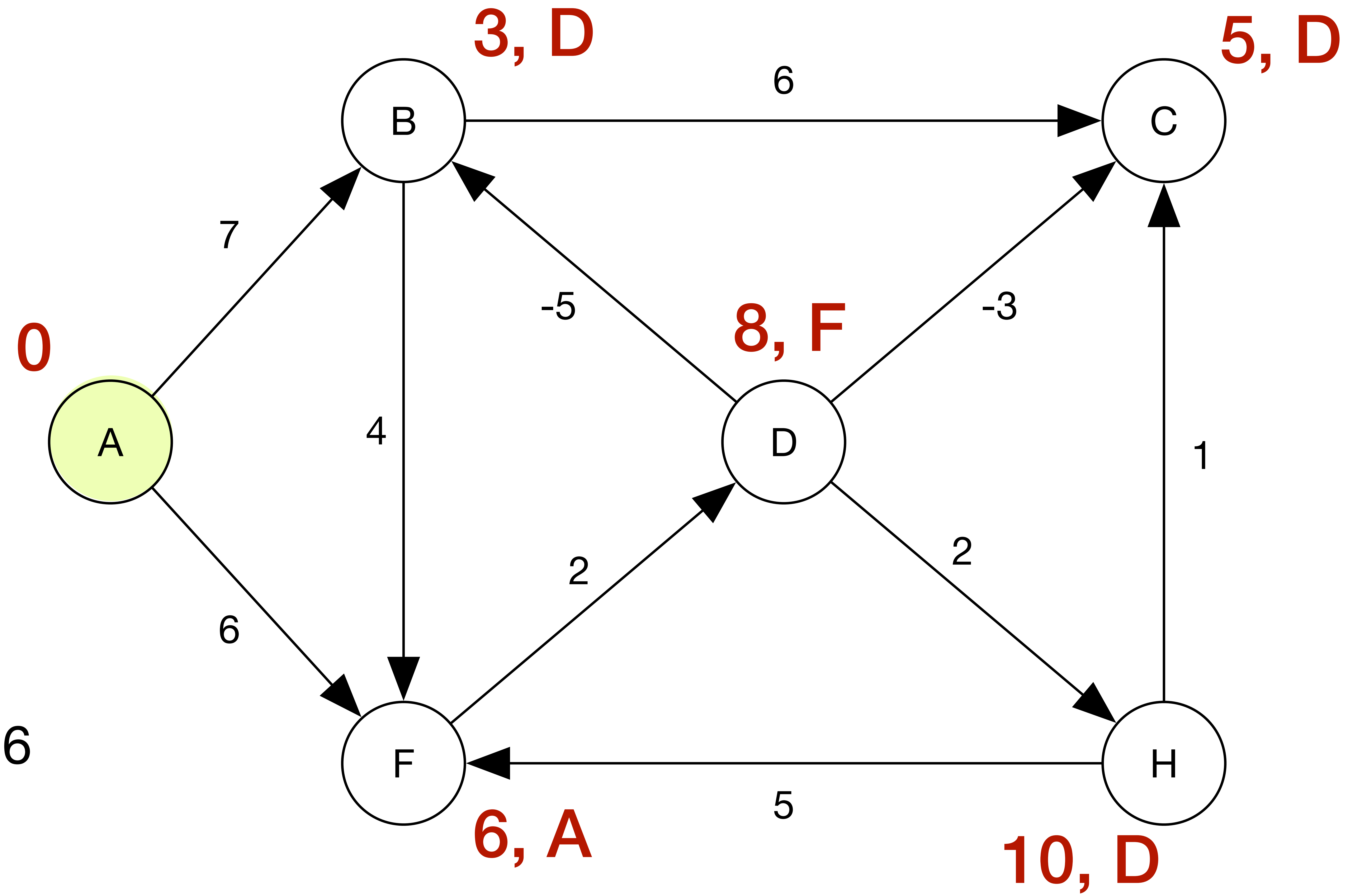
Check for negative cycles



AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Check for negative cycles

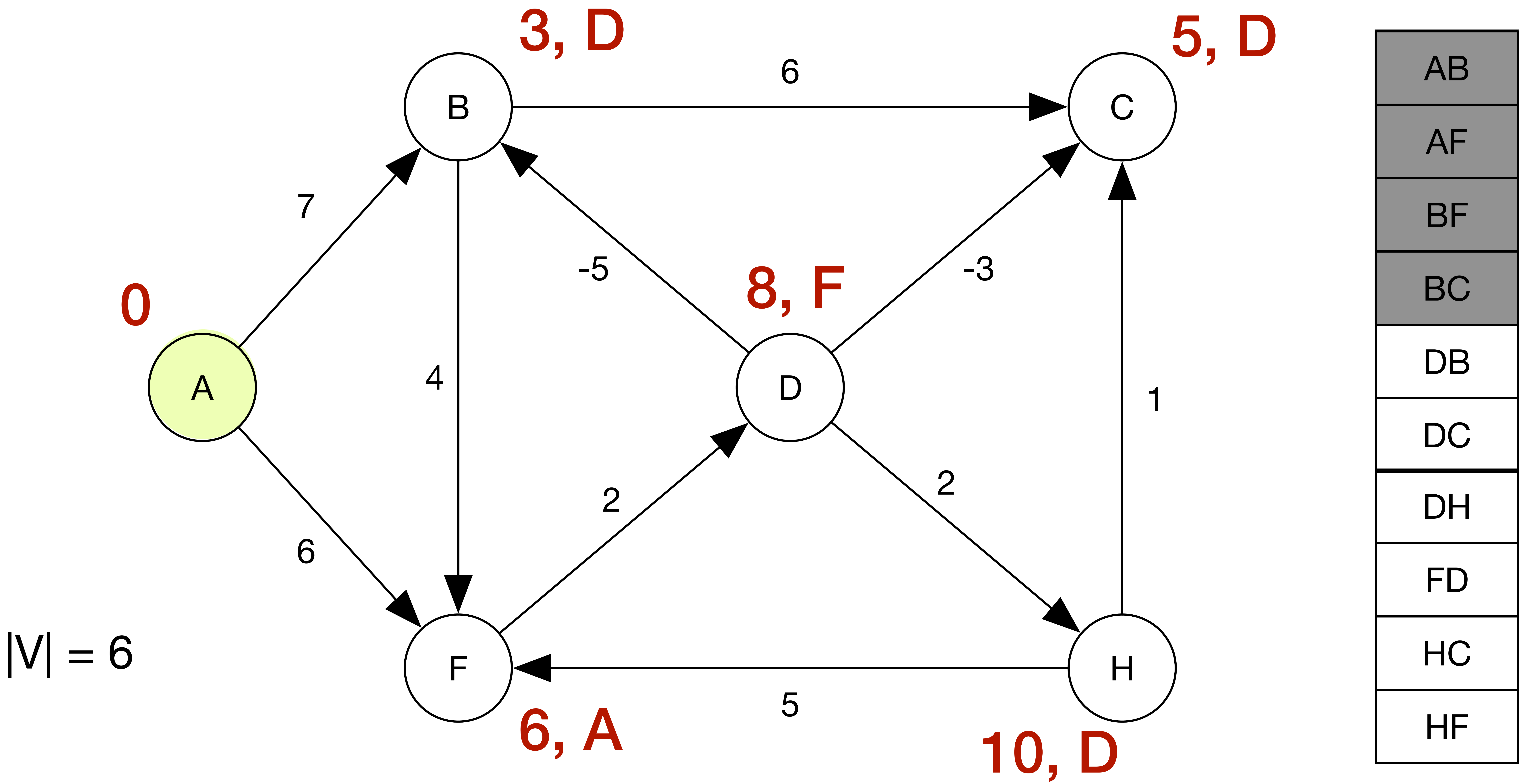


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

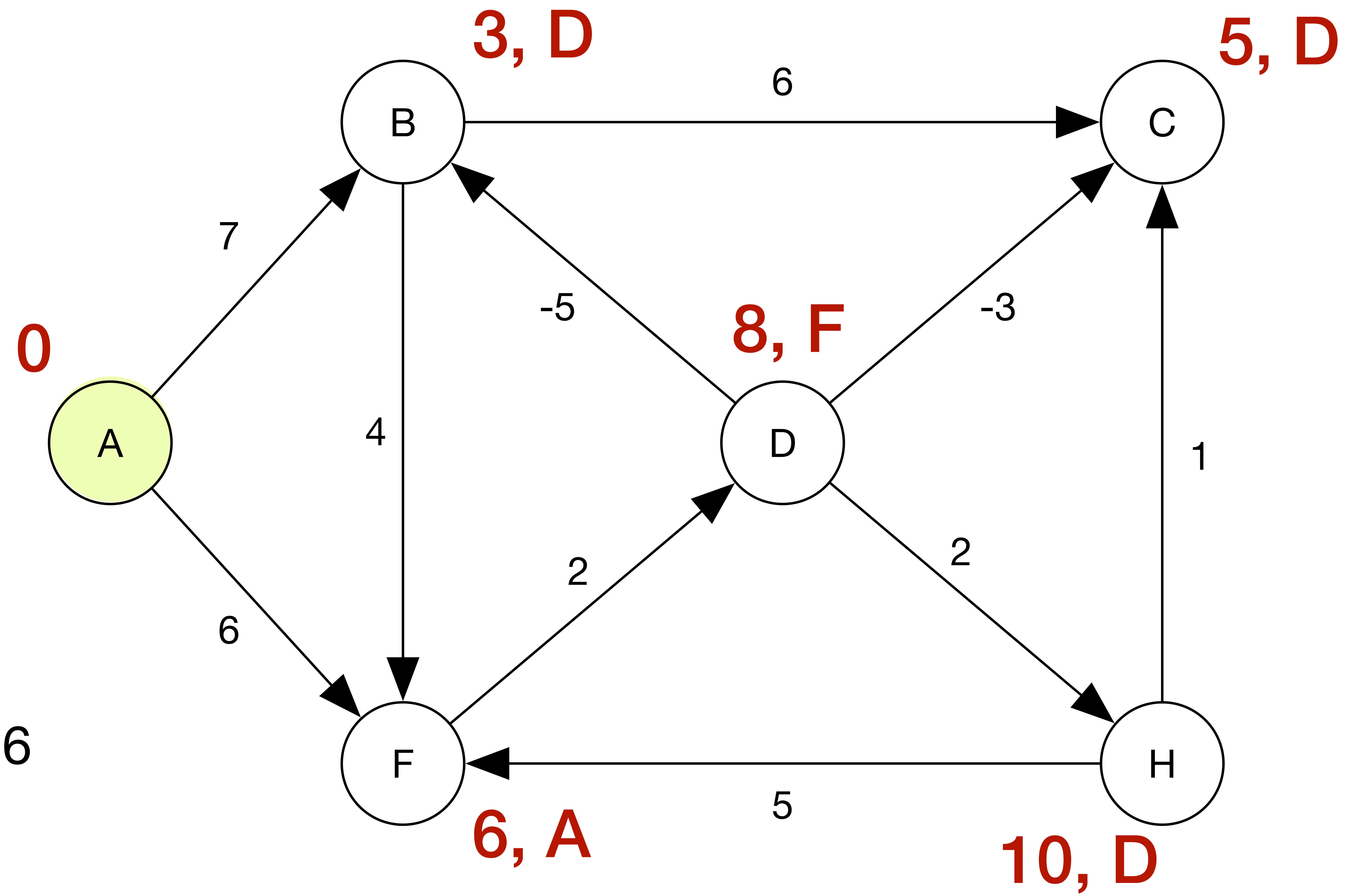
Bellman-Ford

Check for negative cycles



Bellman-Ford

Check for negative cycles

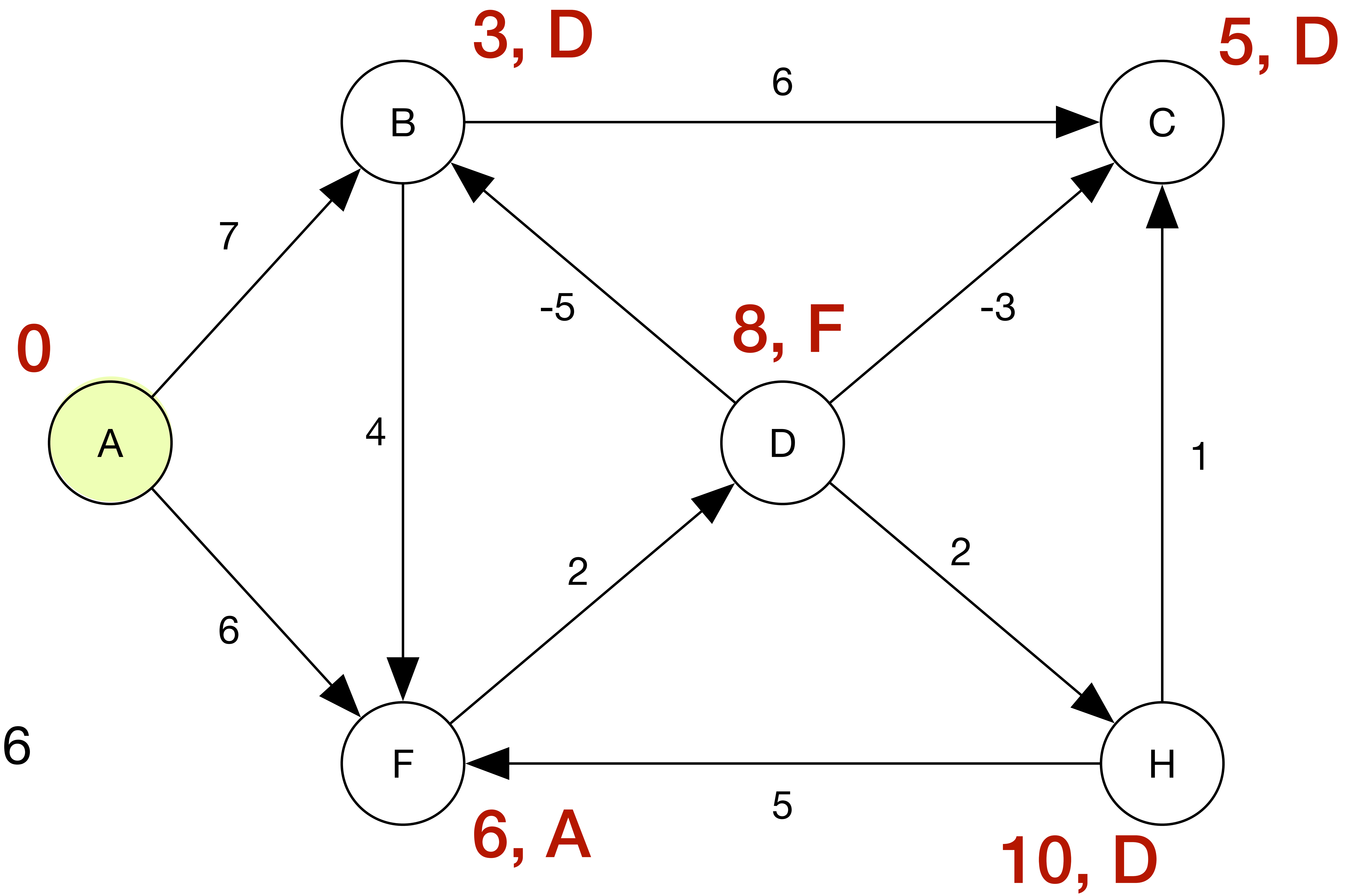


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Check for negative cycles

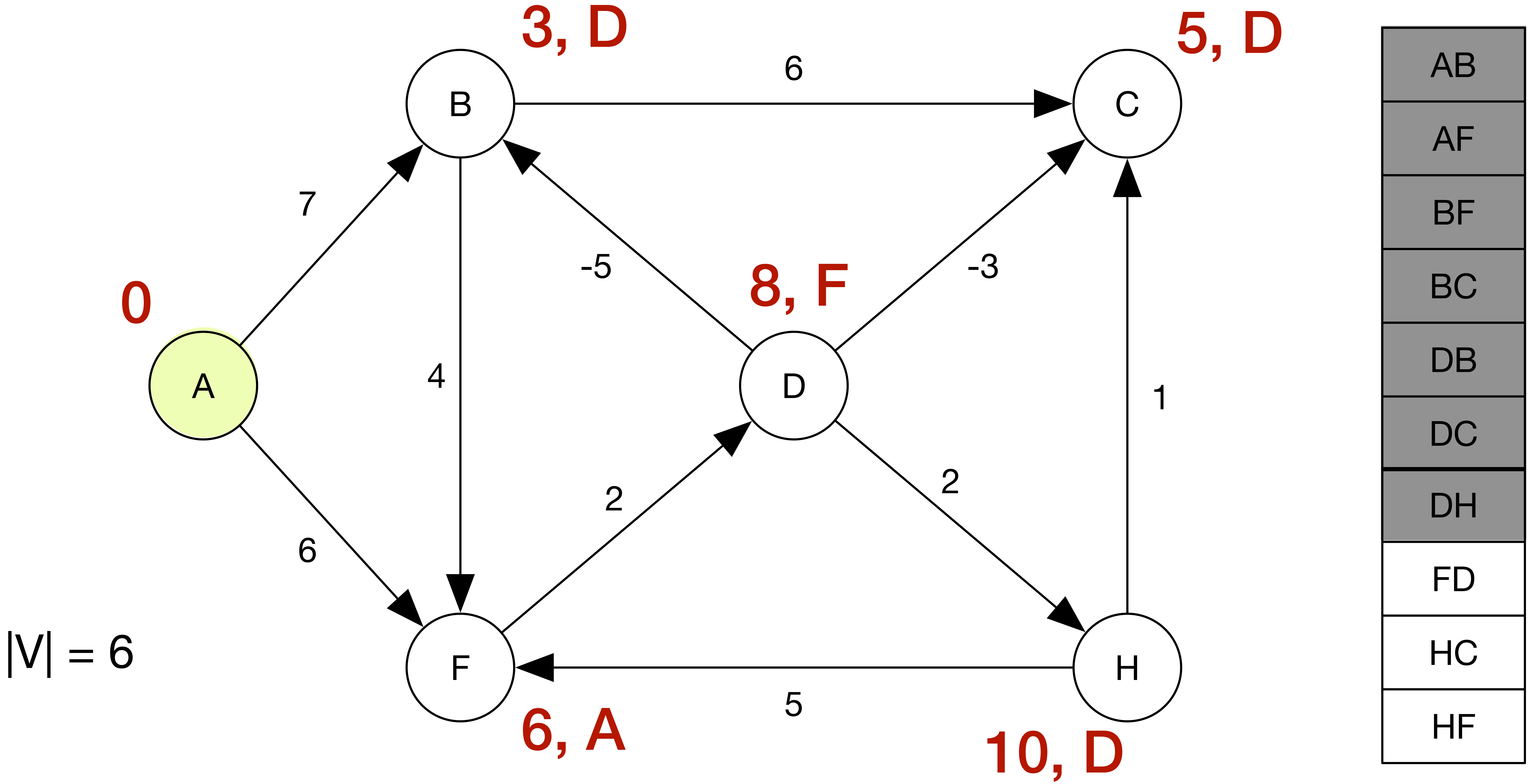


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

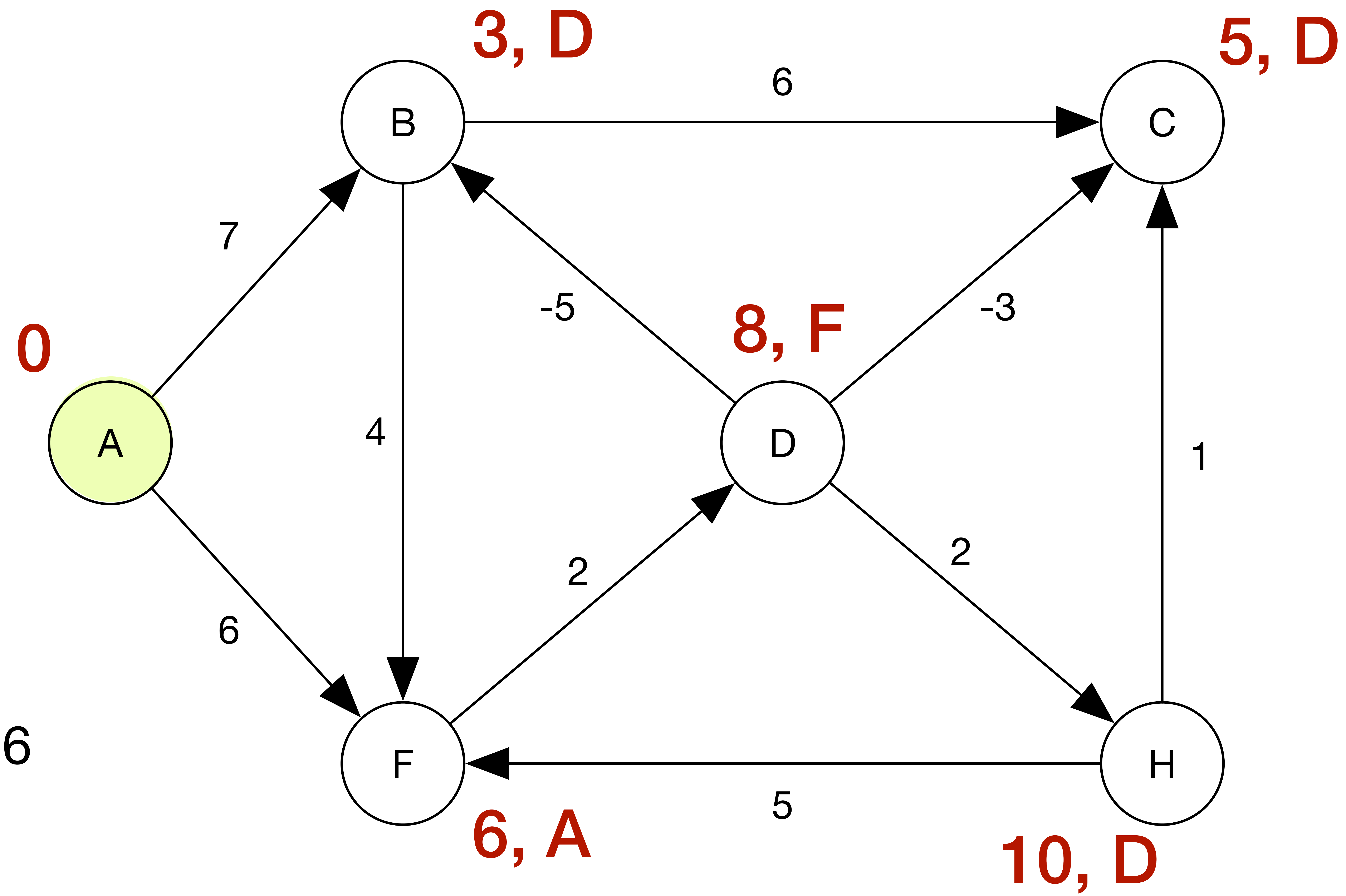
Bellman-Ford

Check for negative cycles



Bellman-Ford

Check for negative cycles

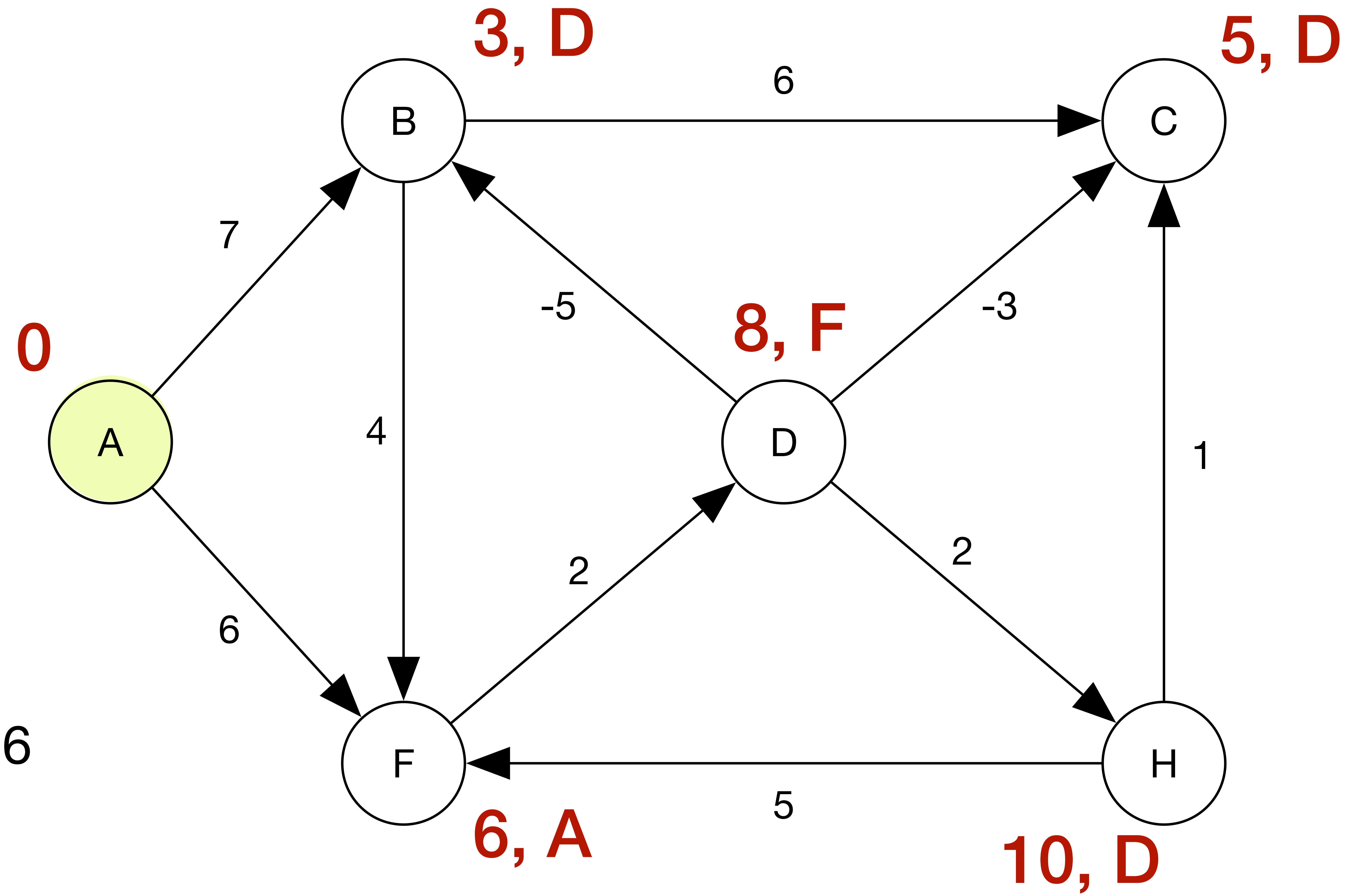


$|V| = 6$

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

Check for negative cycles

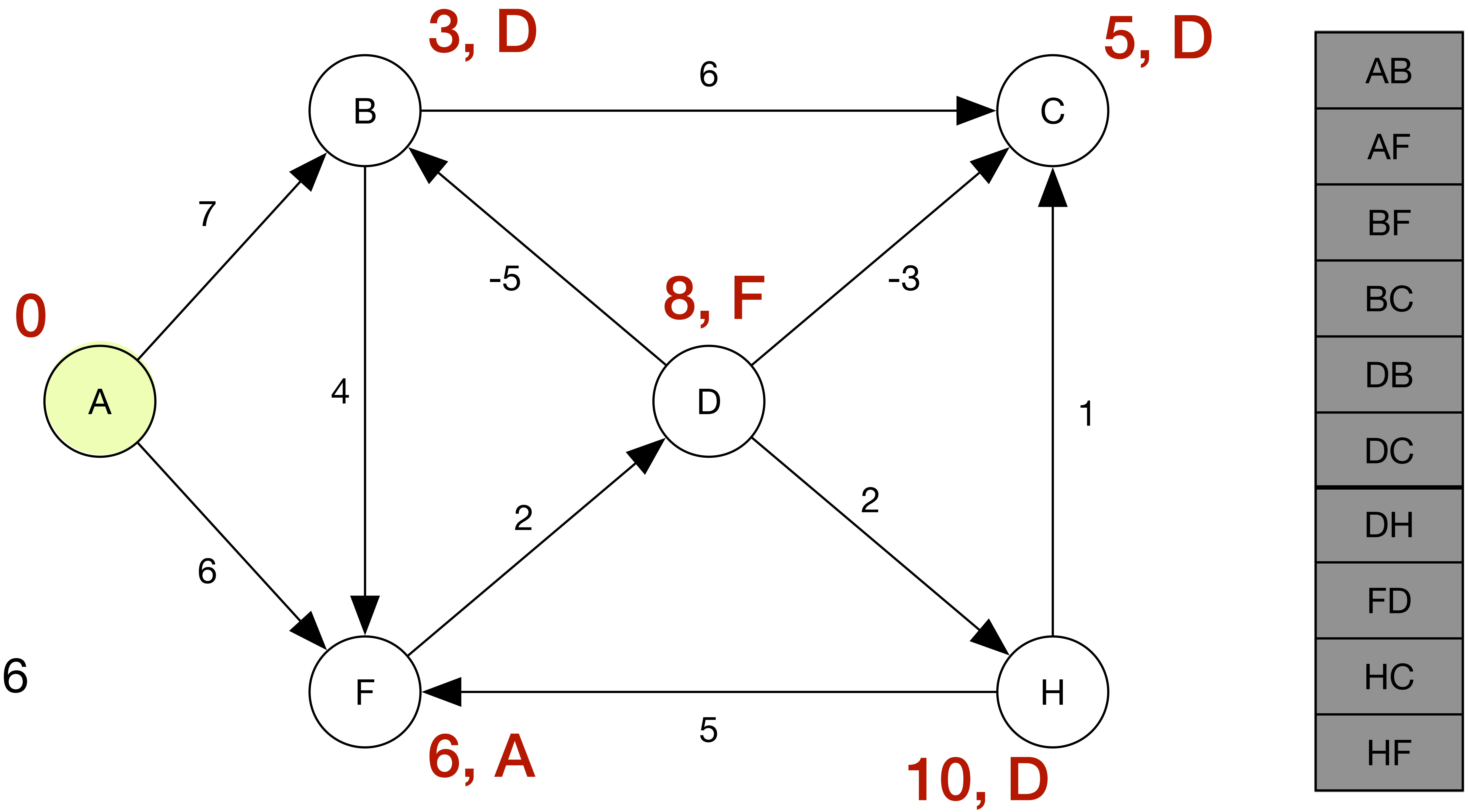


|V| = 6

AB
AF
BF
BC
DB
DC
DH
FD
HC
HF

Bellman-Ford

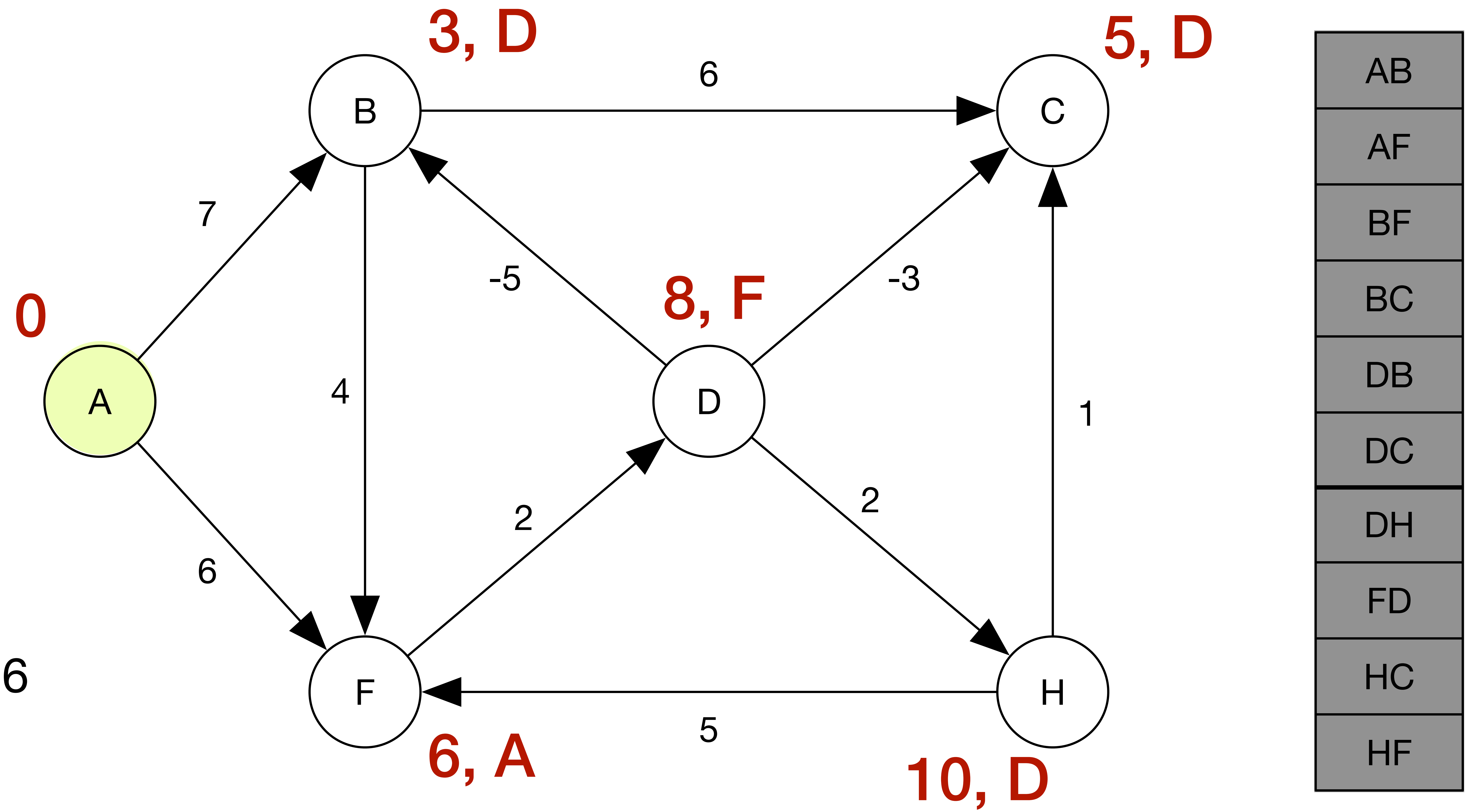
Check for negative cycles



|V| = 6

Bellman-Ford

Done!



|V| = 6

Bellman-Ford

Why do we need up to $|V| - 1$ iterations?

Bellman-Ford

Why do we need up to |V| - 1 iterations? If we have |V| nodes, then the shortest path can have no more than |V| - 1 edges. When we iterate, at the *k*th iteration, we know we've covered all shortest paths up to length *k*. To consider all possible shortest paths, we need $k = |V| - 1$.

Bellman-Ford

Why do we need up to |V| - 1 iterations? If we have |V| nodes, then the shortest path can have no more than |V| - 1 edges. When we iterate, at the *k*th iteration, we know we've covered all shortest paths up to length *k*. To consider all possible shortest paths, we need *k* = |V| - 1.

How does the check for negative cycles work?

Bellman-Ford

Why do we need up to |V| - 1 iterations? If we have |V| nodes, then the shortest path can have no more than |V| - 1 edges. When we iterate, at the *k*th iteration, we know we've covered all shortest paths up to length *k*. To consider all possible shortest paths, we need *k* = |V| - 1.

How does the check for negative cycles work? Once we've processed the graph with |V| - 1 iterations, we check weights. For each edge (U, V) if the distance to V is greater than the sum of the distance to U plus the edge weight from U -> V, then we know we have a negative cycle.

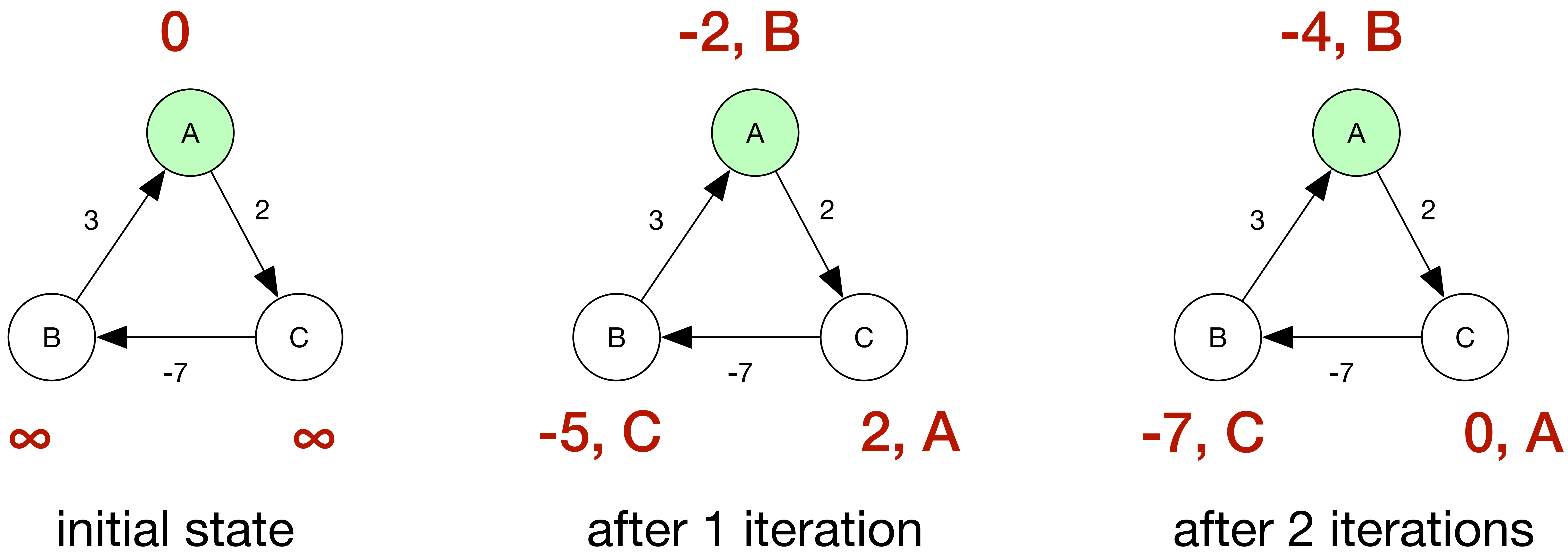
Bellman-Ford

Why do we need up to |V| - 1 iterations? If we have |V| nodes, then the shortest path can have no more than |V| - 1 edges. When we iterate, at the *k*th iteration, we know we've covered all shortest paths up to length *k*. To consider all possible shortest paths, we need *k* = |V| - 1.

How does the check for negative cycles work? Once we've processed the graph with |V| - 1 iterations, we check weights. For each edge (U, V) if the distance to V is greater than the sum of the distance to U plus the edge weight from U -> V, then we know we have a negative cycle.

Bellman-Ford

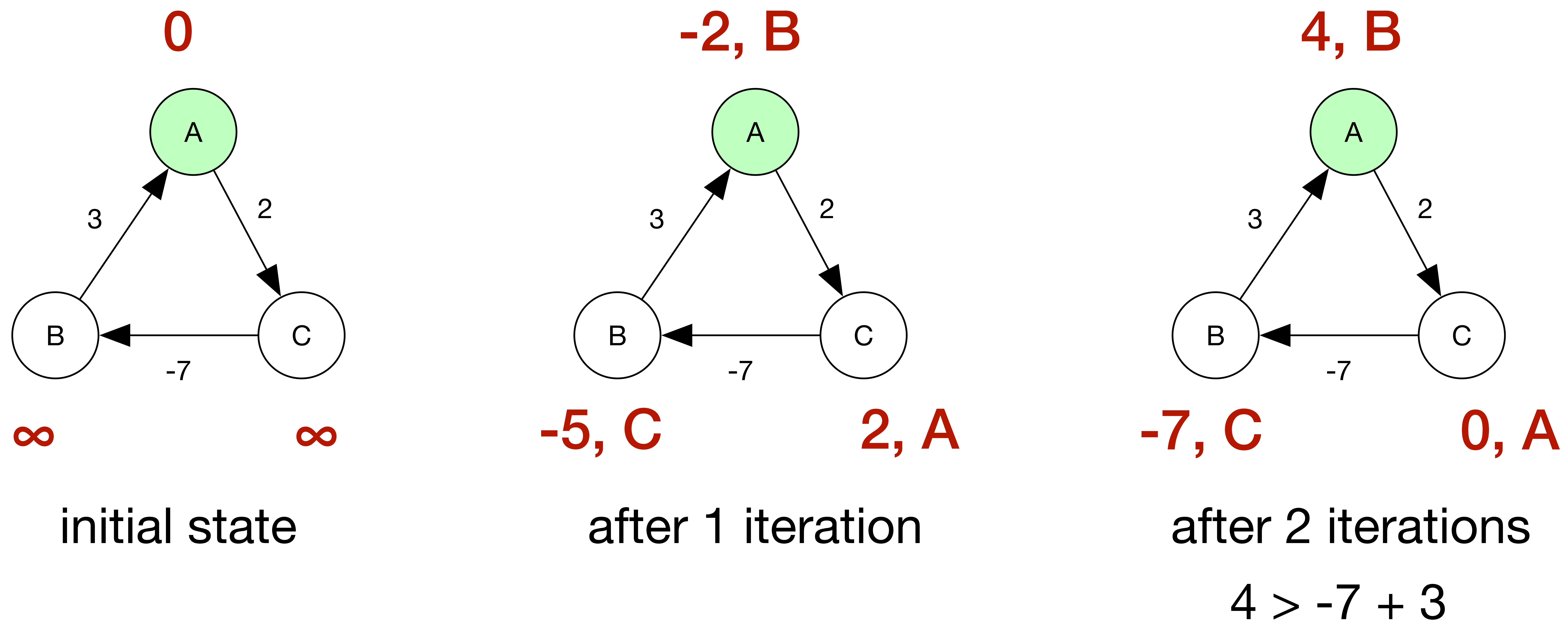
Negative cycle detection



Bellman-Ford

Negative cycle detection

```
for edge (U, V),
if distance[V] > distance[U] + weight of edge
then we have a negative cycle
```



Bellman-Ford

Why do we need up to |V| - 1 iterations? If we have |V| nodes, then the shortest path can have no more than |V| - 1 edges. When we iterate, at the *k*th iteration, we know we've covered all shortest paths up to length *k*. To consider all possible shortest paths, we need *k* = |V| - 1.

How does the check for negative cycles work? Once we've processed the graph with |V| - 1 iterations, we check weights. For each edge (U, V) if the distance to V is greater than the sum of the distance to U plus the edge weight from U -> V, then we know we have a negative cycle.

Shortest Path

Algorithm	Dijkstra	Bellman - Ford
Worst-case complexity	O((|V| + |E|) log |V|)	O(|V| x |E|)
Restrictions	Edge weights must be non-negative	No negative cycles