UNIVERSITY OF PATRAS

DEPARTMENT OF BUSINESS ADMINISTRATION

FURTHER OPERATIONAL RESEARCH TECHNIQUES

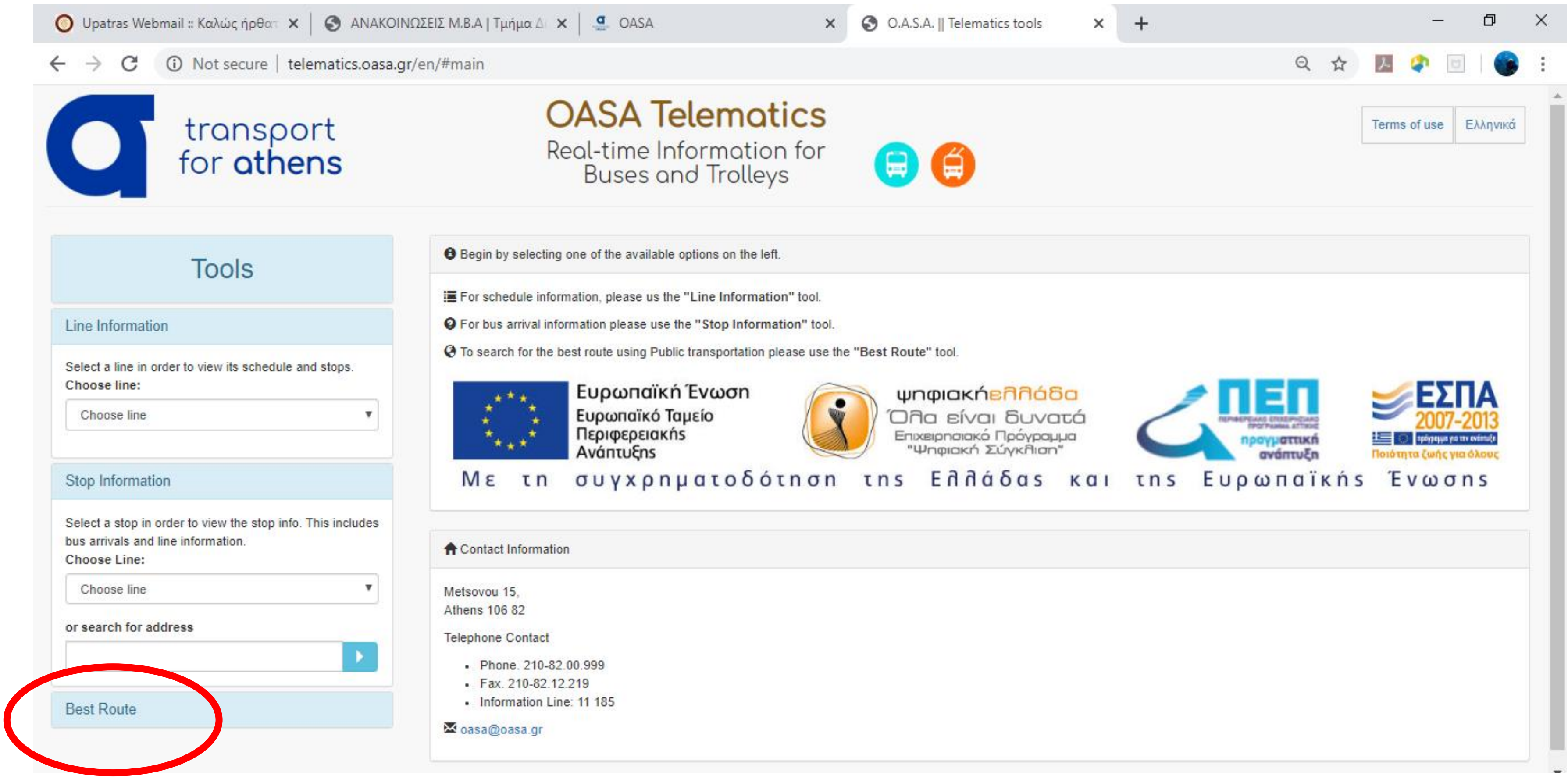
Lecture 1: NETWORK ANALYSIS-INTRODUCTION

Patras 2022

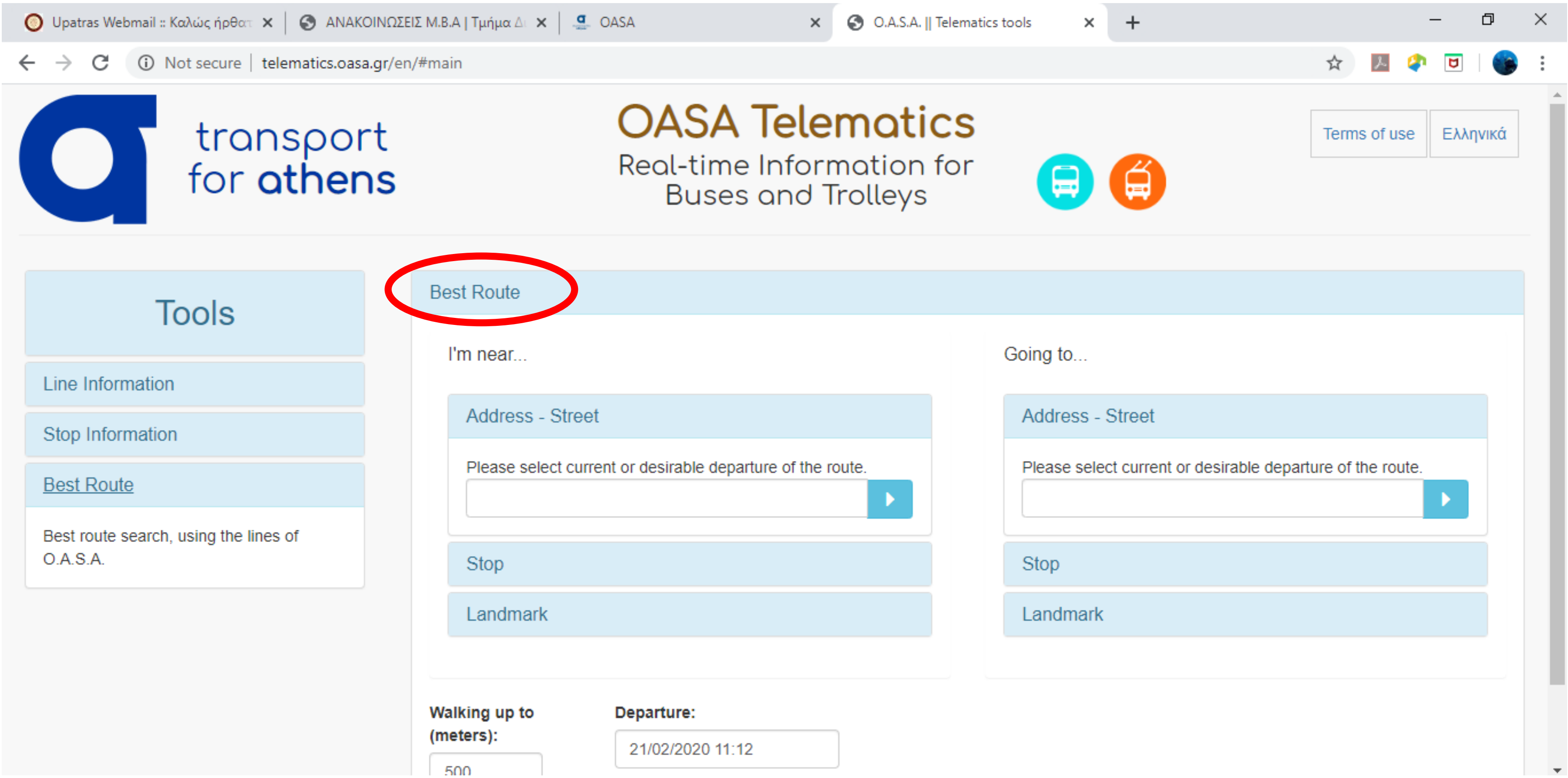
Logistics

- **Organization of the material**
  - **2 hours lecture**
  - **Exercises or workshops when necessary**
  - **1 hour tutorial**
- **Note: Important to attend lectures!**
- **Office hours: To be arranged via e-mail**
- **e-mail: I.Giannikos@upatras.gr**

Network Analysis An Introduction



Network Analysis An Introduction/2



Network Analysis An Introduction /3

- Webpage of Athens Transport (www.oasa.gr)
- Application: Optimal (best) route
- How is optimality defined?
  - Minimum distance
  - Shortest time
  - Minimum number of transits
  - etc
- (These are well known problems in Network Analysis)

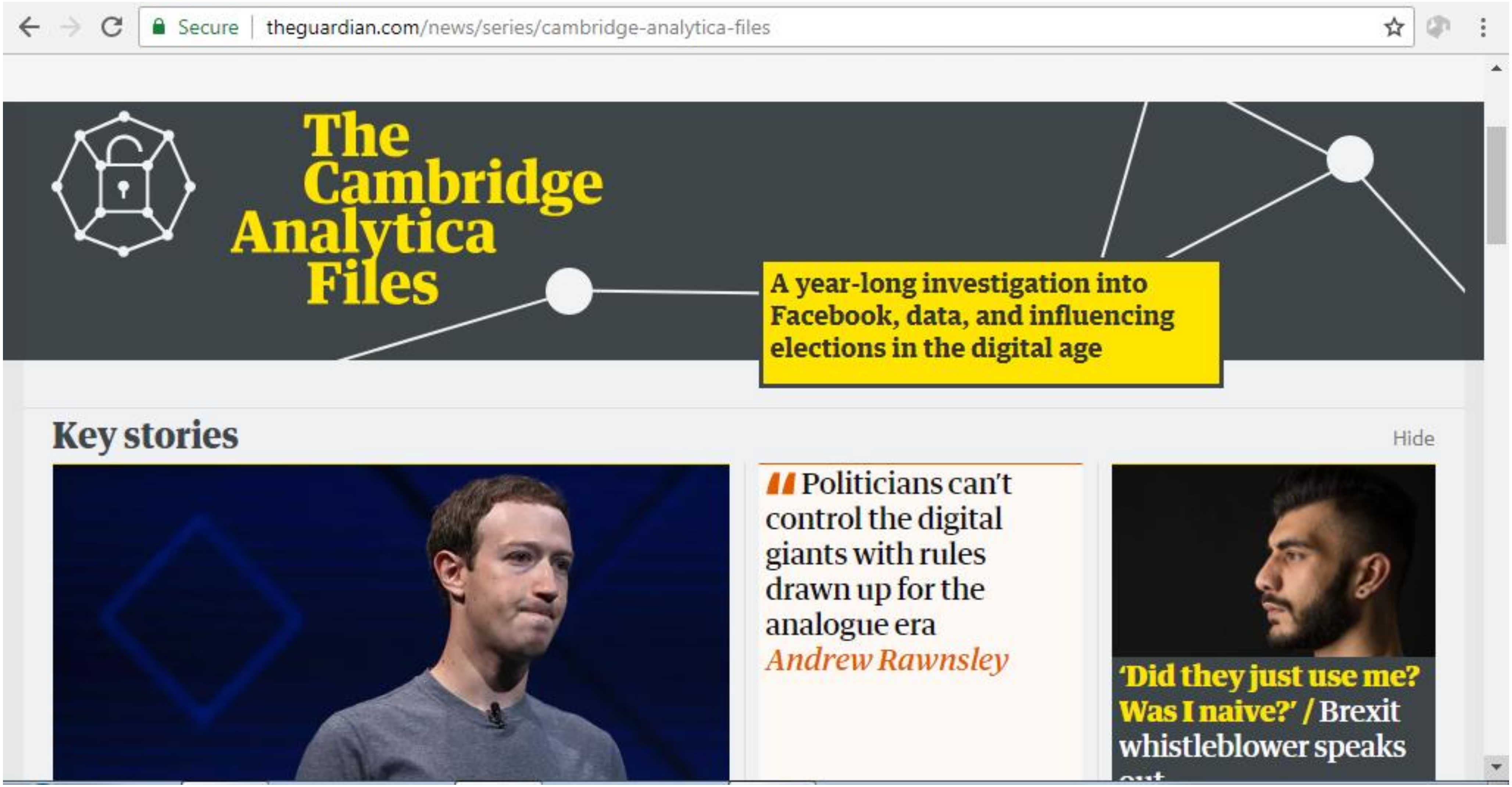
Network Analysis An Introduction /3

- **More practical applications**
  - **Transportation networks**
  - **Telecommunication networks**
  - **Scheduling**

- **Important developments**
  - **New algorithms**
  - **Technology**

- **Note**
  - **Several network analysis problems can be formulated as Linear Programming problems (LPs)**
  - **E.g. transportation problem, assignment problem**

A (recent) story

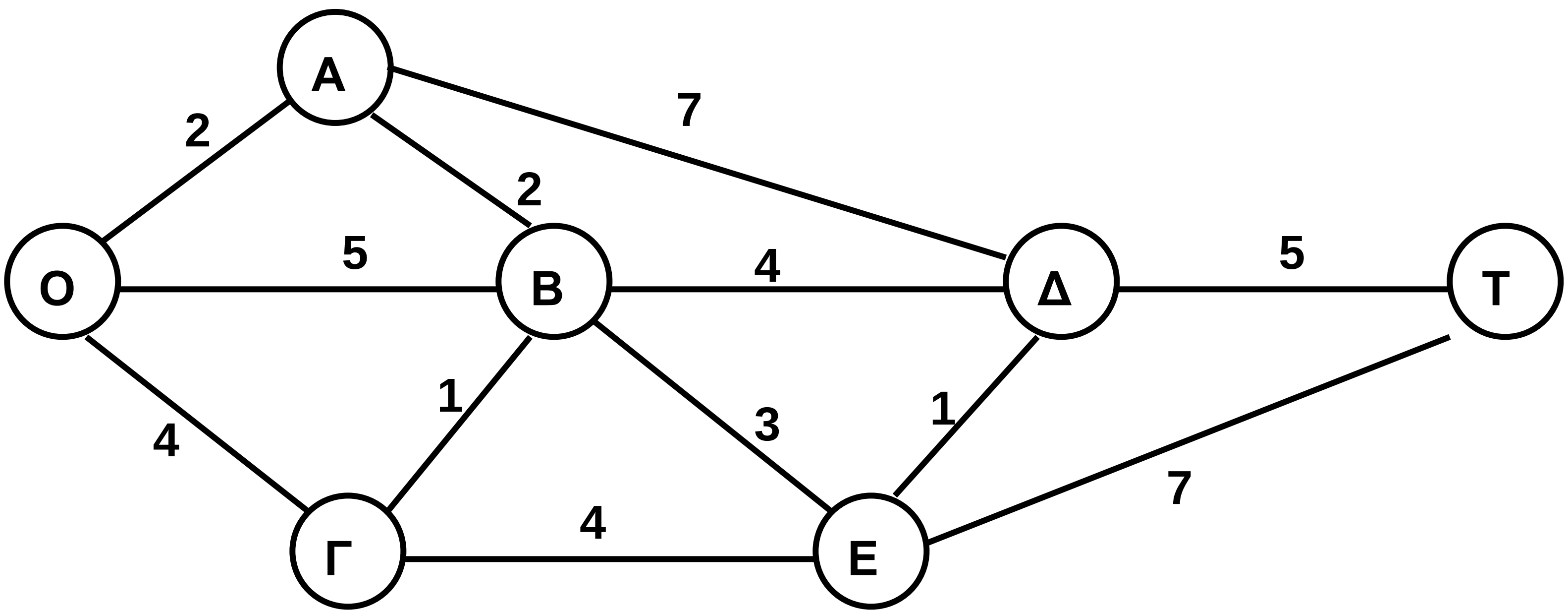


Network Analysis Problems

- Shortest path
- Maximum Flow
- Minimum Cost Flow (MCF)
  - The first two problems can be formulated as special cases of MCF
- Minimum spanning tree
- Scheduling

Example

- In a certain national park there are several kiosks connected by roads. Let O be the entrance and T the exit of the park.



Problems

- Which is the shortest route from O to T?
- Which is the shortest route from O to any other kiosk?
- If any road can accept a limited number of cars per day, what is the maximum number of cars that can travel from O to T per day?
- If all kiosks must be connected by phone lines, what is the minimum length of lines required?

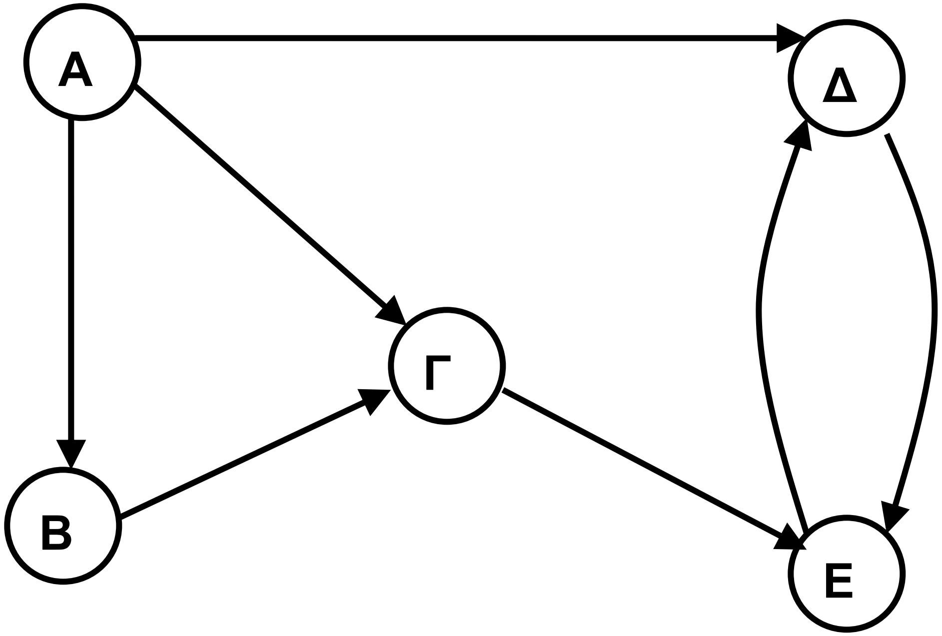
Some Definitions

- Graph
  - A set of *nodes* V and *edges* E
- *Edge* (or *arc* or *link*)
  - Directed 
  - Undirected
- Path (or route) from i to j
  - A set of edges connecting i with j
  - Directed
  - Undirected

More definitions

- Network
  - A directed graph
- Each edge is characterized by
  - Capacity (maximum flow it can accept)
  - Cost per unit flow

Example

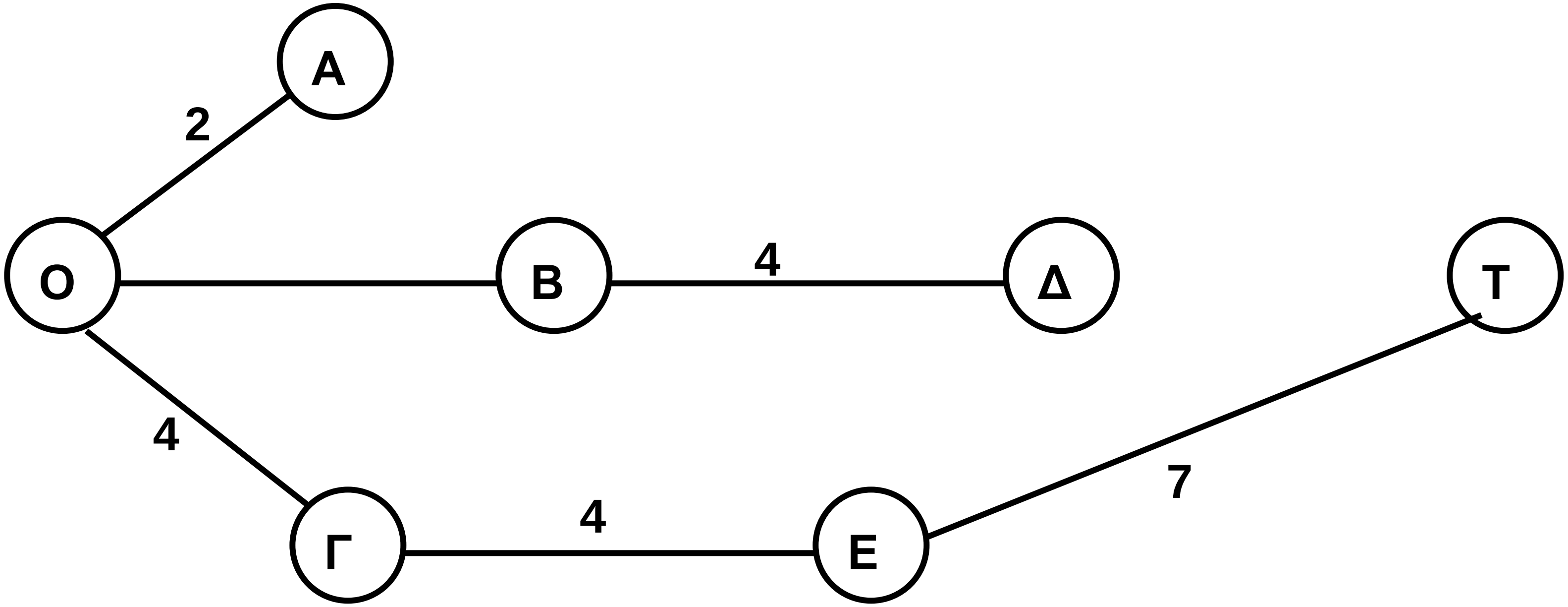


- **Path AB-BΓ-ΓE**
- **The set of edges BΓ-AΓ-AΔ is not a path!**

Even more definitions

- **Two nodes A and B are <u>connected</u> when there is a path from A to B.**
- **A graph is connected when <u>any two nodes</u> of the graph are connected**
- **A <u>cycle</u> is a path beginning and ending at the same node**
- **A <u>tree</u> is a connected graph without cycles**

Example of a tree



- **Properties of trees (proven theoretically):**
  - **A tree with n nodes has n-1 edges**
  - **Any pair of nodes in a tree is connected with a unique path**

The Shortest Path Problem

- **Assume that we have an undirected graph**
- **A node O is considered as the origin and another node T as the destination**
- **Every edge is characterized by a "distance" $d \geq 0$**
- **Problem: Find the shortest path from the origin to the destination**
- **Nodes**
  - **Permanent: nodes for which we have calculated the length of the shortest path from the origin**
  - **Non permanent: all others**

Dijkstra's Algorithm

1 Consider all nodes as *non permanent*, except the origin. Consider the origin as *permanent* node

2 Repeat until the end

   2.1 For every *permanent* node find the nearest *non permanent*

   2.2 Out of all candidates (non permanent nodes) select the nearest one to the origin and make it *permanent*

Dijkstra’s Algorithm – Formal description

- Maintain a set S of permanent nodes u for which we have calculated the length of the shortest path ***δ(u)*** from the origin s to u

- Initially it is S={s} and ***δ(s)***=0

- Find a non permanent node v such that

$$dist(v) = \min_{e=(u,v):u\in S} \{\delta(u) + length(e)\}$$

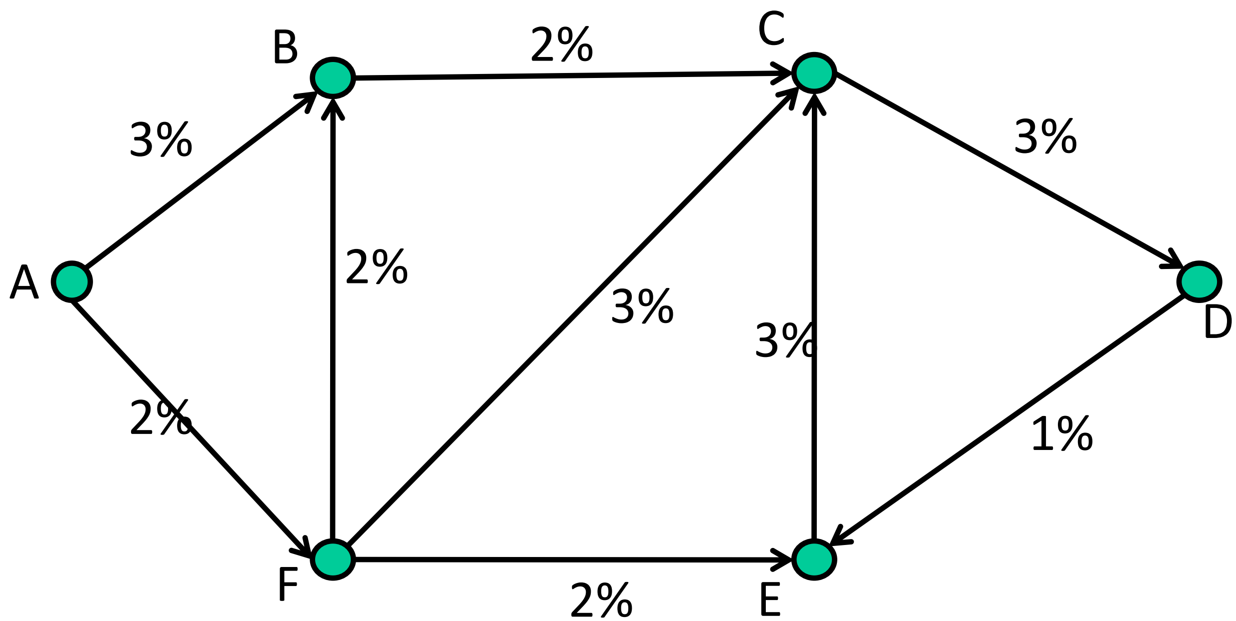
- Insert v to the set of permanent nodes and set ***δ(v)***=dist(v)

Extensions

- The length of each edge may express time, cost, etc
- The algorithm may easily be adapted for directed graphs
- The algorithm may easily find the shortest path from the origin to any other node

Other Network Applications: The Safest Path Problem

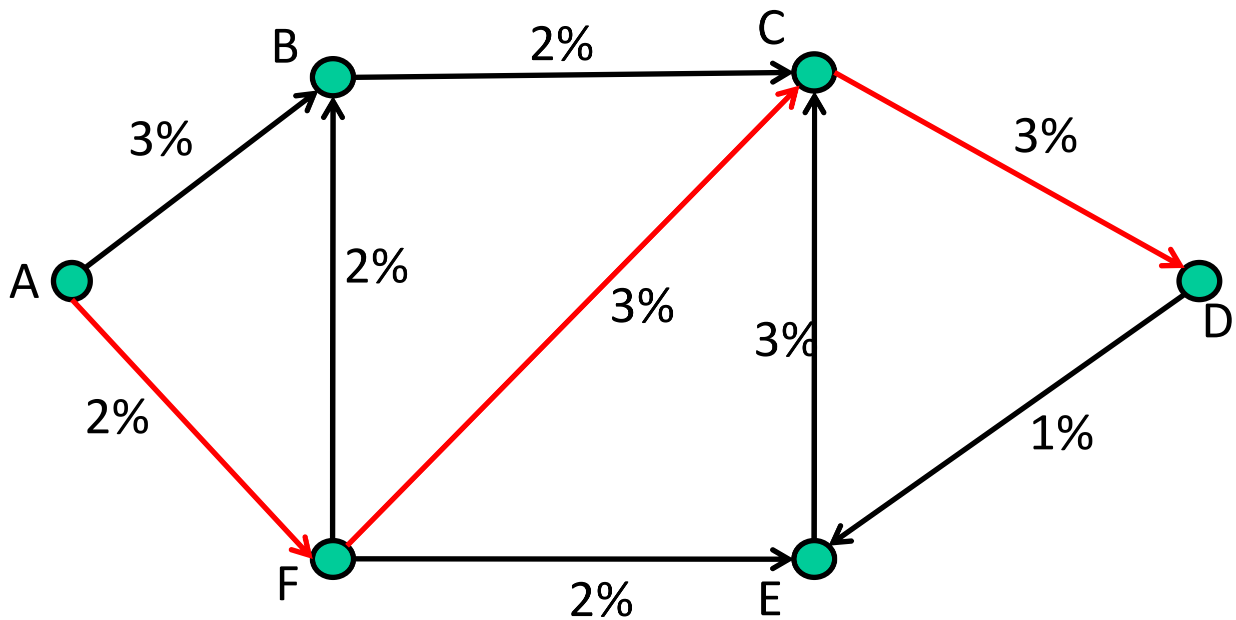
- **In the following network the nodes represent computers and the edges connections. The number next to each edge denotes the probability that the edge fails**



- **What is the safest path from node A to node D?**

The Safest Path Problem/2

- **Lets consider a specific path (e.g. path AFCD). What is the reliability of the path?**



- **Reliability: probability of no failure**
- **For path AFCD it is (0,98)·(0,97)·(0,97)=0,922082**
- ***(the product of non failure probabilities)***
- **(Hence, the probability of failure is 1-0,922082=0,077918)**

The Safest Path Problem/3

- **We wish to find the path from A to D which maximizes the reliability (safety)**
- **Generally: p(e) probability of failure along edge *e***

  q(e)=1─ p(e) probability of non-failure along edge e

  (E.g. for edge AF it is q(AF)=0,98)
- **The reliability of any path *S*, consisting of, say *k* edges, is:**

$$Q(S) = \prod_{e \in S} q(e) = q_1 \cdot q_2 \cdot \ldots \cdot q_k$$

- **<u>We wish to find the path that maximizes function *Q(S)*</u>**

(We will formulate the problem as a Shortest Path problem)

The Safest Path Problem/4

- **Since the logarithmic function is increasing, maximizing *Q(S)* is equivalent to:**

$$\max z = \ln Q(S) = ln\ (q_1 \cdot q_2 \cdot \ldots \cdot q_k) = \ln(q_1) + ln(q_2) + \ldots + \ln(q_k)$$

- **(Since the Shortest Path problem concerns minimization, we have:)**

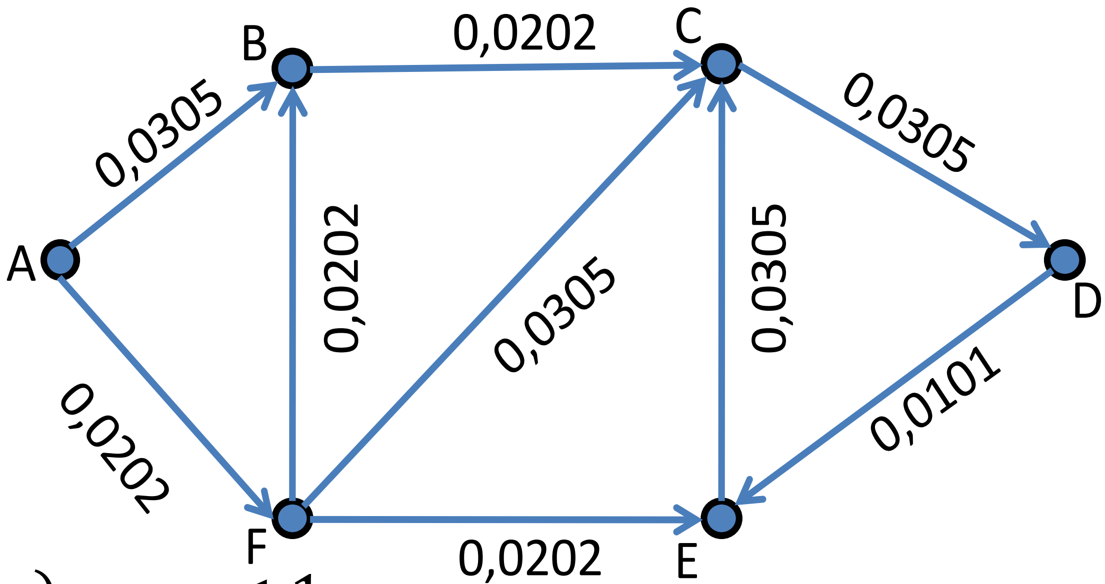
$$\max z = \min -z$$

$$\min\ -z = \min z' = -\ln(q_1) - ln(q_2) - \cdots - \ln(q_k)$$

- **If we let** $w(e) = -\ln q(e)$
- **Then the function is written:** $\min\ z' = w_1 + w_2 + \ldots + w_k$
- **(*I.e. we have a shortest path problem with weights w_1, w_2, …, w_k*)**

The Safest Path Problem/5

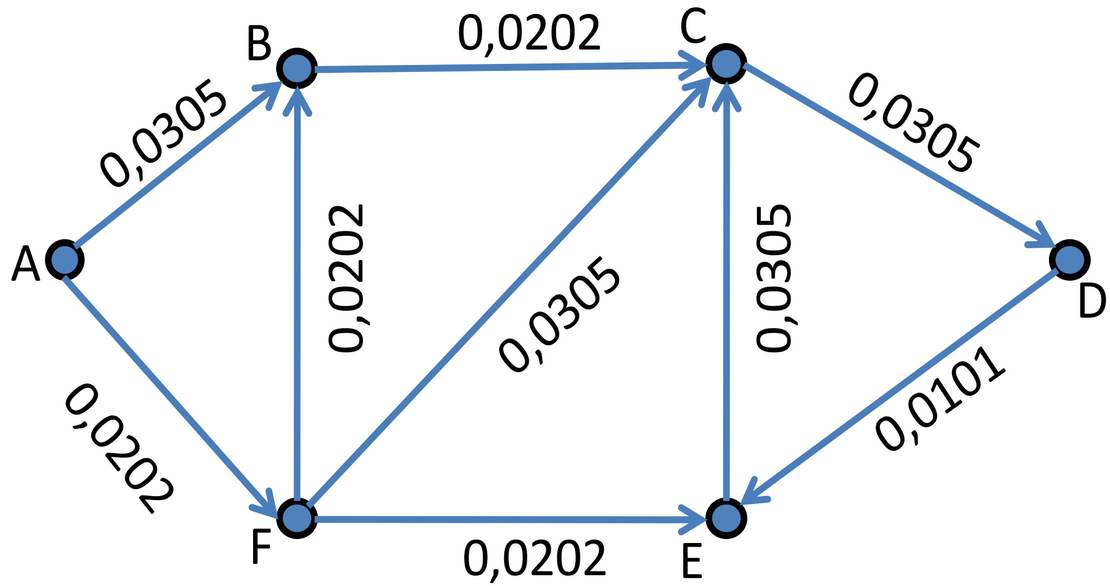
- **The new graph is:**



- **Since** $q(e) = q_e < 1$
- **It is** $\ln(q_e) < 0$
- **And, finally** $w(e) = -ln \quad q(e) > 0$
- **Since we have positive weights, we can apply Djikstra's algorithm!**

The Safest Path Problem/6

- **The safest (most reliable) path is ABCD (or AFCD)**
- ***(There are two optimal paths)***



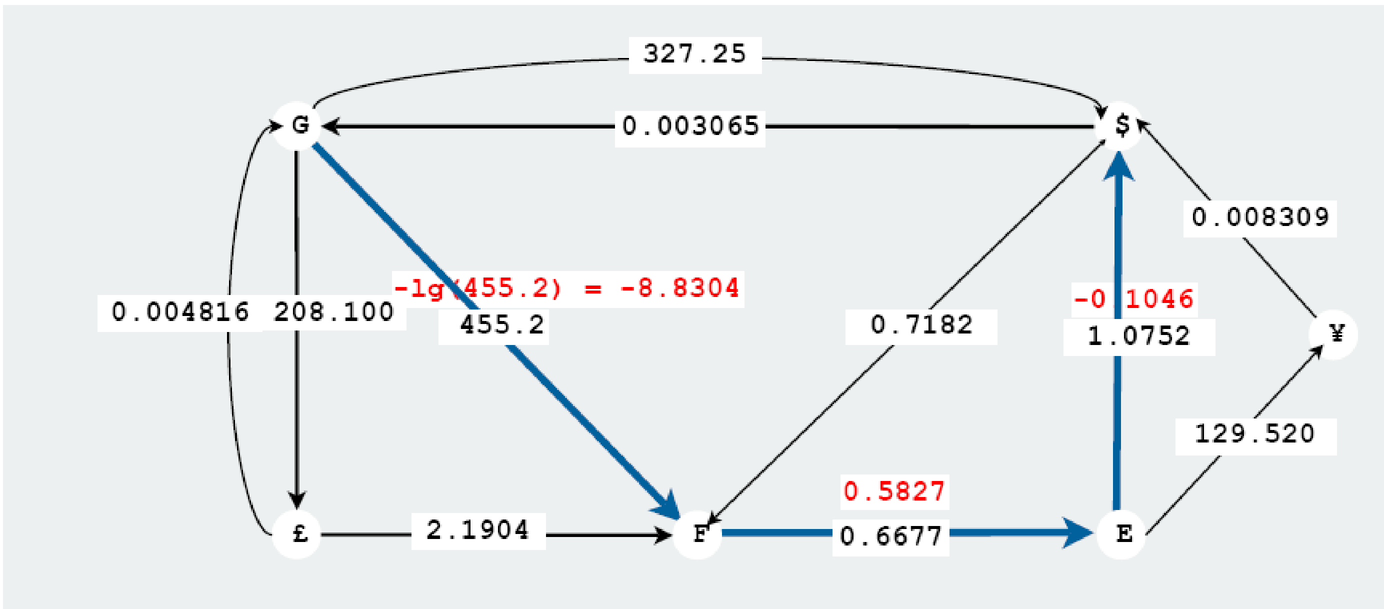
- **Total (minimum) weight 0,0305+0,0202+0,0305=0,0812**
- **Reliability** $e^{-0,0812} = 92,21\%$

Edges with negative length - Example

- Currency exchange rates
  - Given the exchange rates in the international market, what is the best way to convert 1 ounce of Gold to UD Dollars?
  - 1 oz. Gold corresponds to $327.25.
  - 1 oz. gold corresponds to £208.10 or $327.00.
  - 1 oz. gold corresponds to 455.2 Francs or 304.39 Euros or $327,28
- Graph with
  - Currencies as nodes
  - Edges: conversions of one currency to others
  - Problem: find the path which maximizes the product of rates

Contrast: Example with exchange rates

- By taking logarithms of the weights, we end up with a shortest path problem



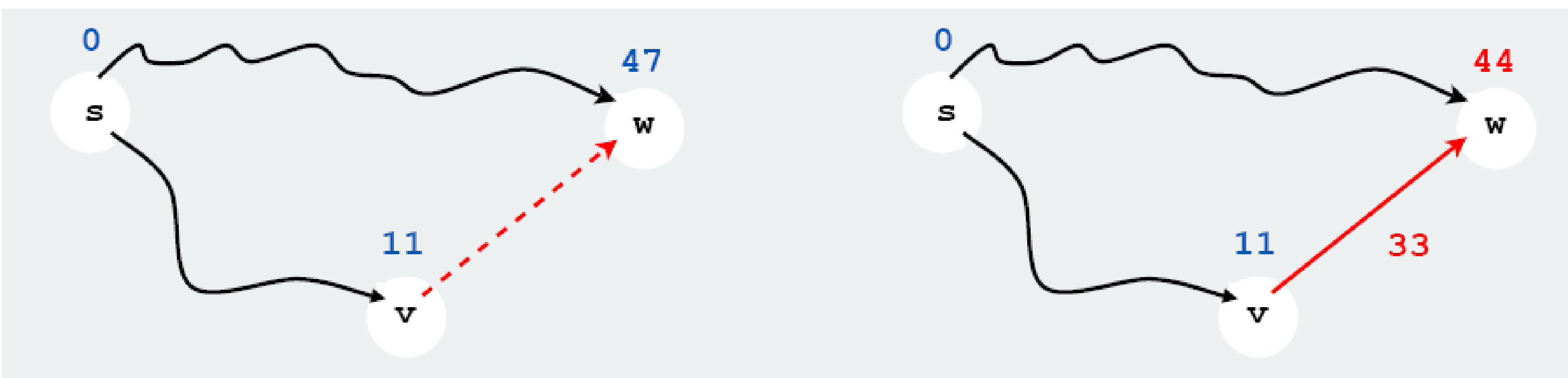
- (Some) exchange rates are greater than 1
- Problem: negative weights!

Extension – Edges with negative length

- **Dijkstra's algorithm cannot be applied in these cases. It may produce wrong solutions!**
- **A different approach is required**
- **If there exists a cycle from s to t with negative total weight, then the length of the path may become arbitrarily small (tends to $-\infty$)**
- **A special algorithm (known as the Bellman-Ford algorithm) allows for negative weights and identifies negative cycles**

Main Idea (Edge Relaxation)

- **Fundamental process in shortest paths**
- **For every v ∈ V , let δ [v] be the length of some path from s (the origin) to v**
- **Practically**
  - **Edge relaxation sets δ [w] equal to the length of the shortest path from s to v if this path goes through node w i.e. if it includes edge (v, w).**



Edge Relaxation /2

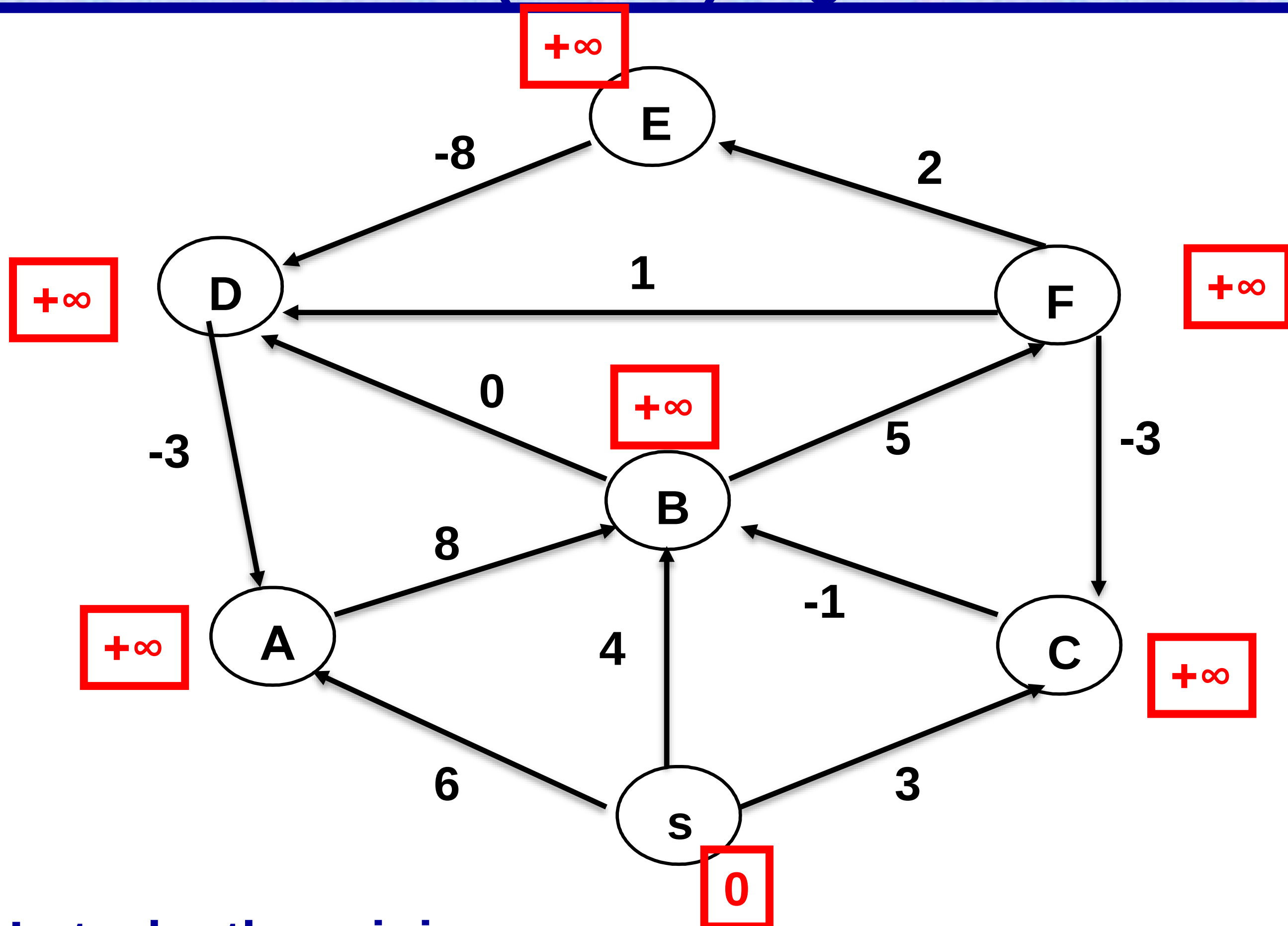
- For every $v \in V$, pred [v] is the previous node to v in the current shortest path
- Relaxing edge (v,w)
  - δ [v] the length of some path from s to v
  - δ [w] the length of some path from s to w
  - If δ [v] + length (v,w) < δ [w] then
    - Update δ [w] and set pred [w] $\leftarrow$ v

Bellman – Ford (Moore) Algorithm (Outline)

- For i=1 to |V|-1 do
  - For every edge (u,v)

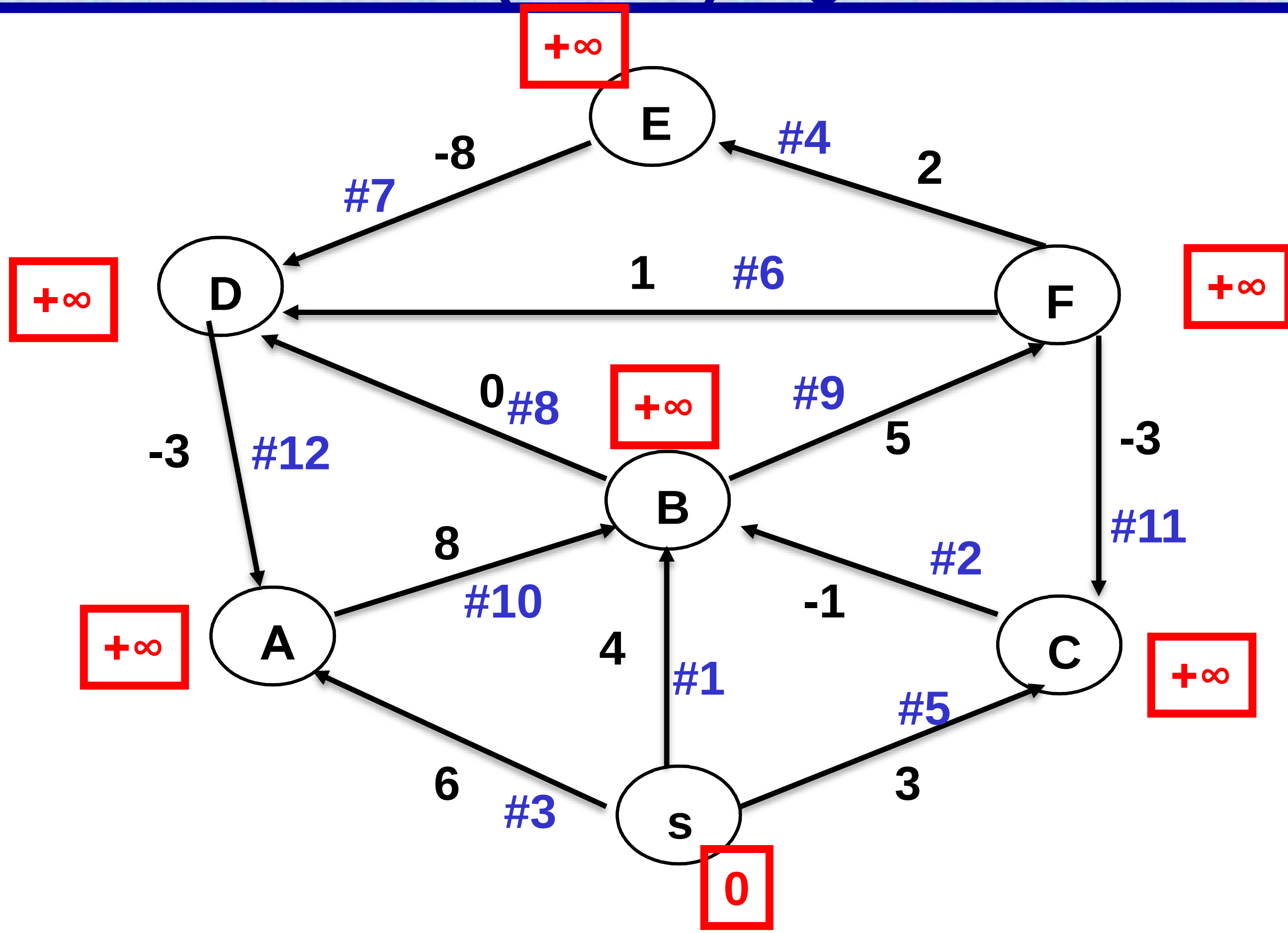
    Relax the edge (u,v)
- For every edge (u,v)
  - If the edge can be relaxed, then there exists a cycle with negative total length (the problem does not have a finite solution)
- Practically, we need to determine the order in which we will visit the edges in each iteration
- (The order does not affect the final solution)

Bellman-Ford (Moore) Algorithm – Exercise 3



- Let s be the origin
- The red number next to each node denotes the length of the current shortest path from the origin (initially it is +∞, except at s)

Bellman-Ford (Moore) Algorithm – Exercise 3



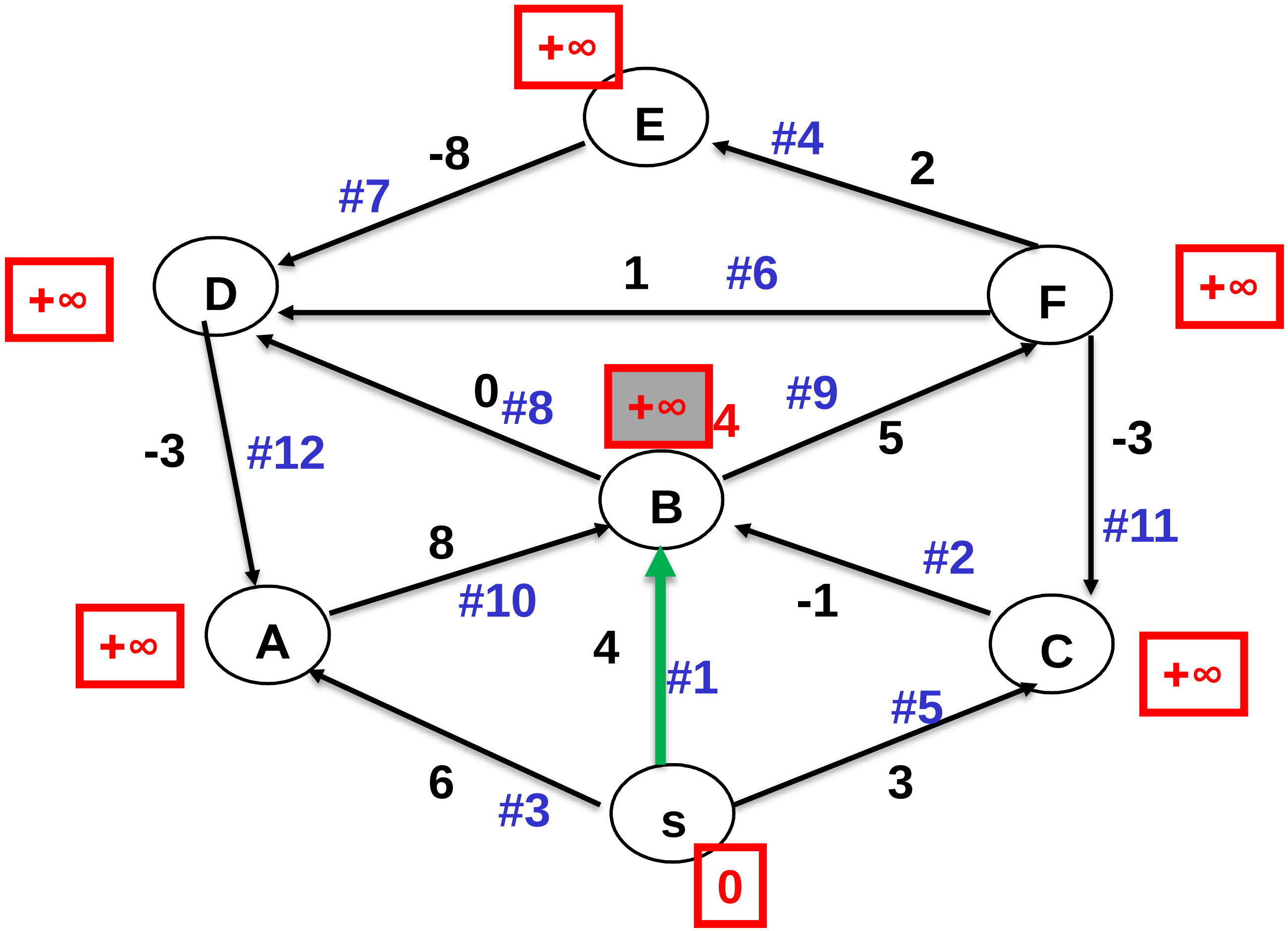
- The order in which we will visit the edges is denoted by #
- (This order does not affect the optimal solution)

Bellman-Ford (Moore) Algorithm – Implementation

- We then present the iterative steps of the algorithm
- Each time we denote the edge which is relaxed and the new length of the shortest path from the origin to the final node of the edge

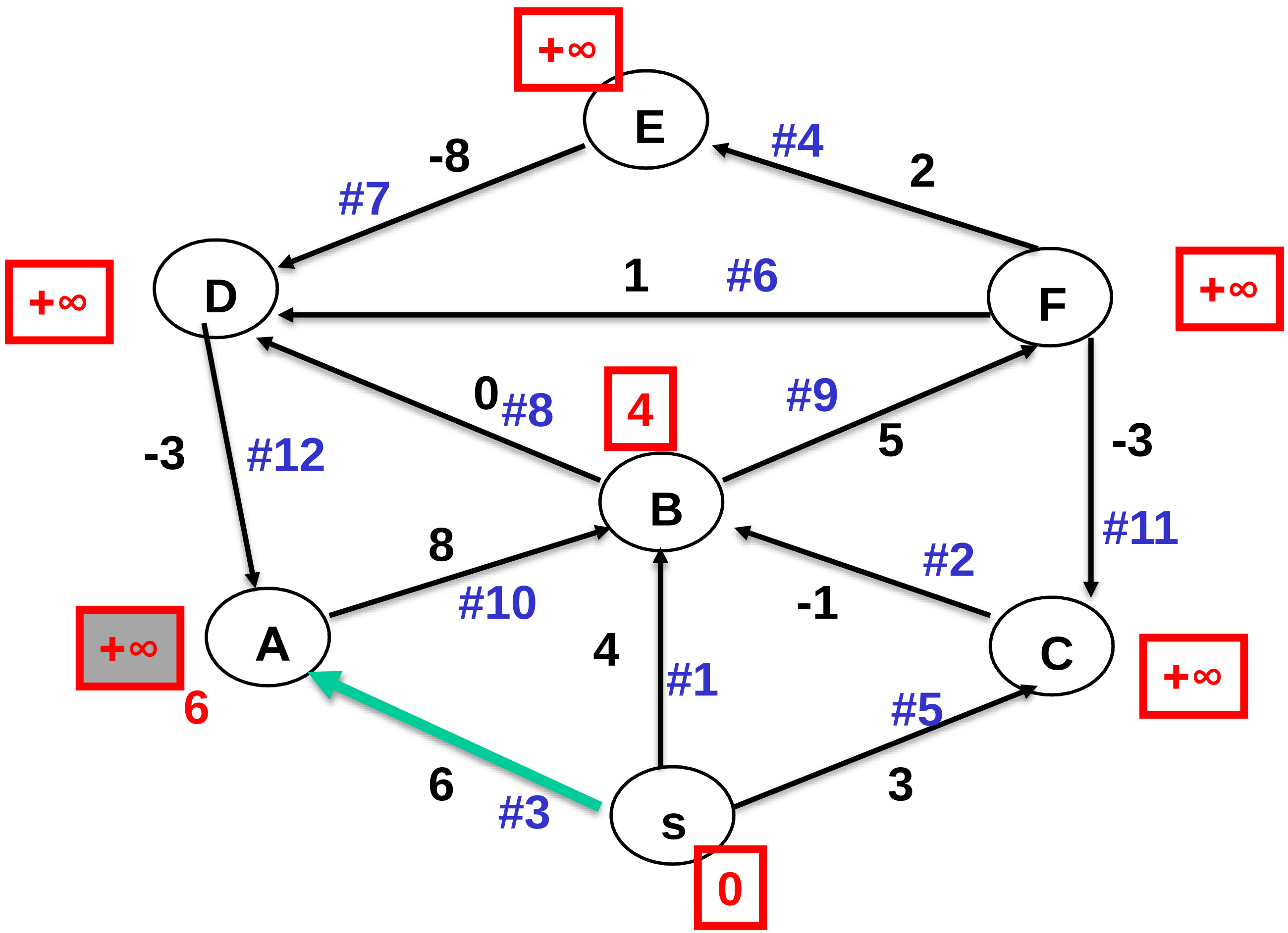
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 1, Edge #1**



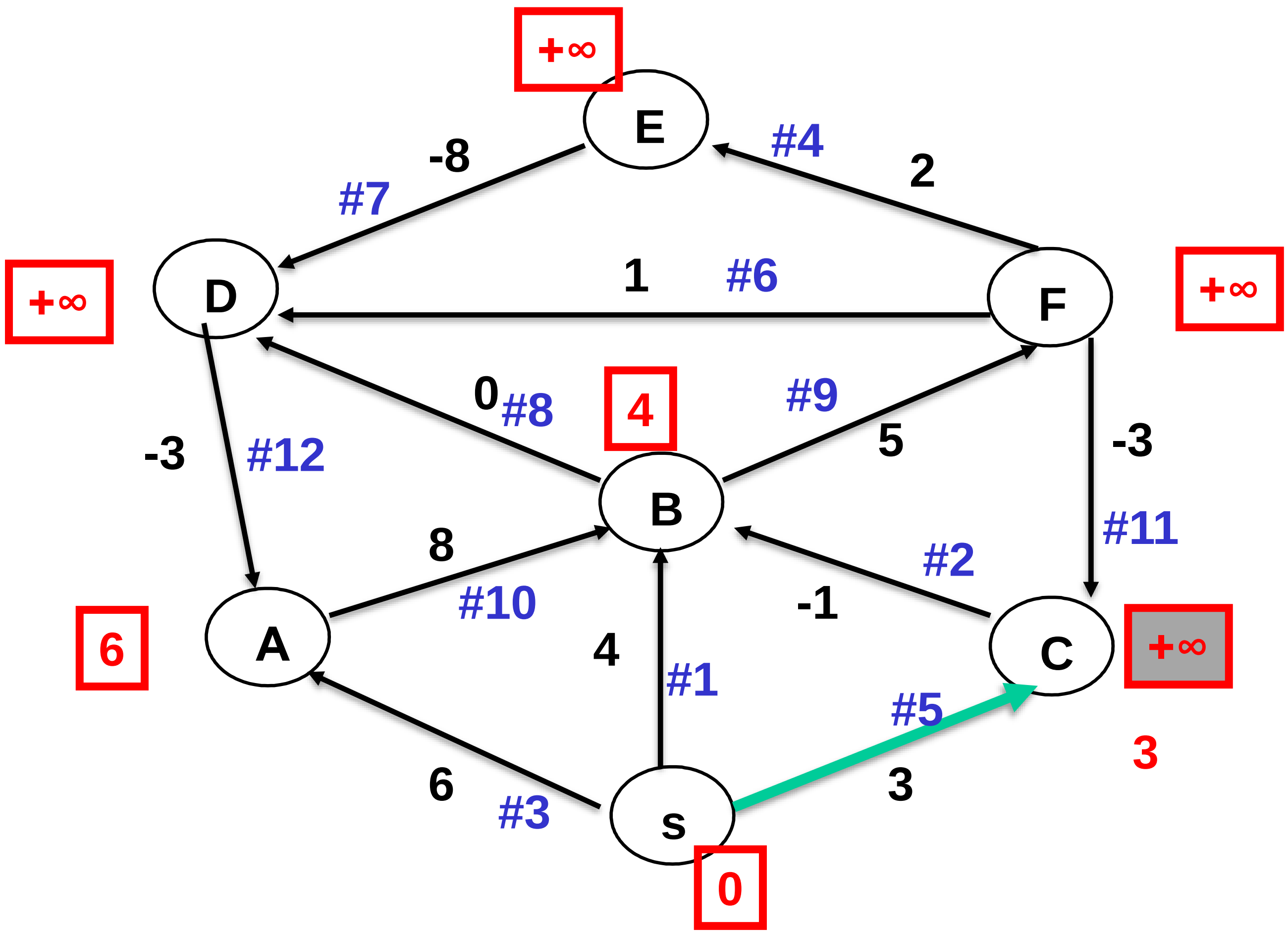
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 1, Edge #3**



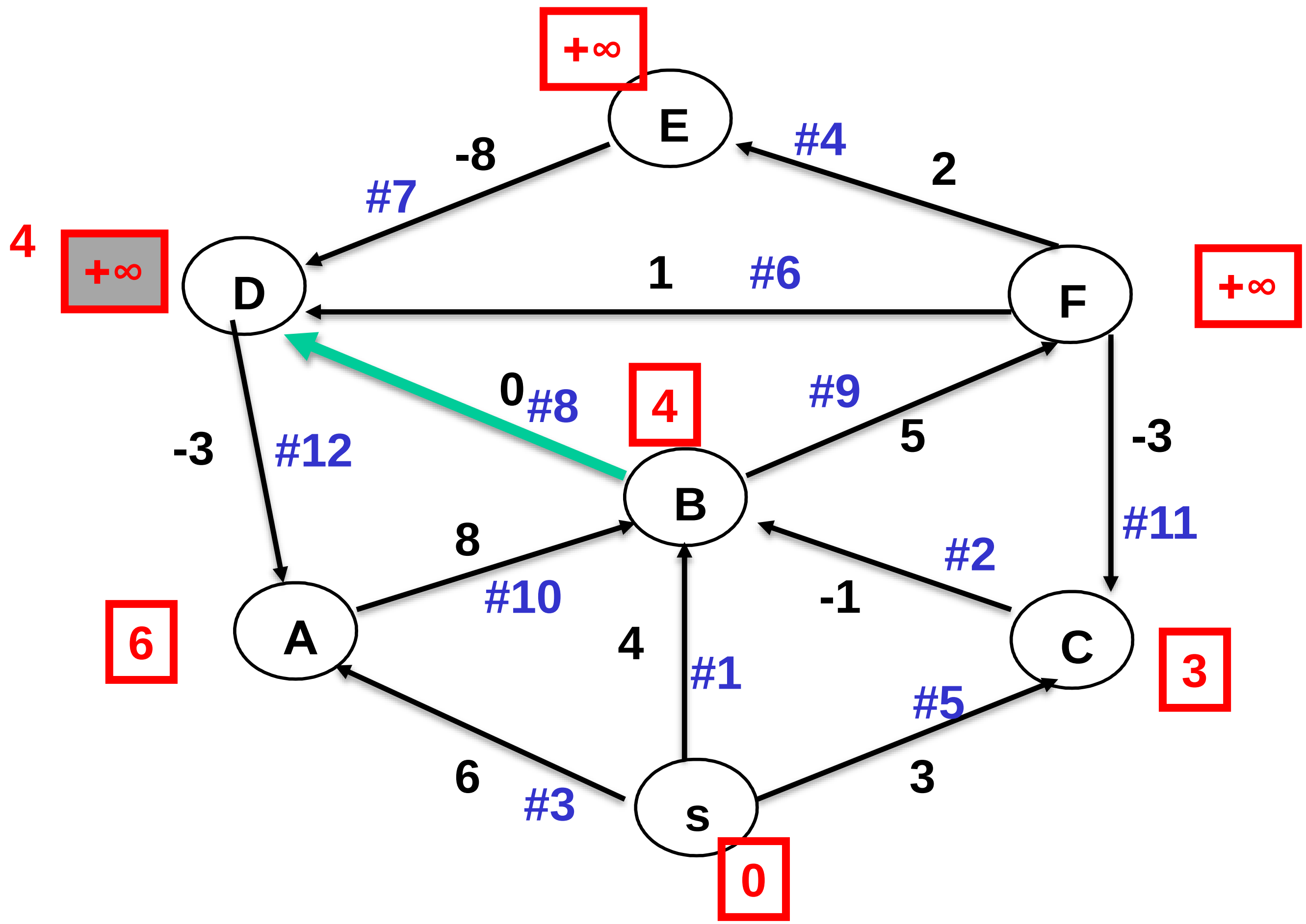
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 1, Edge #5**



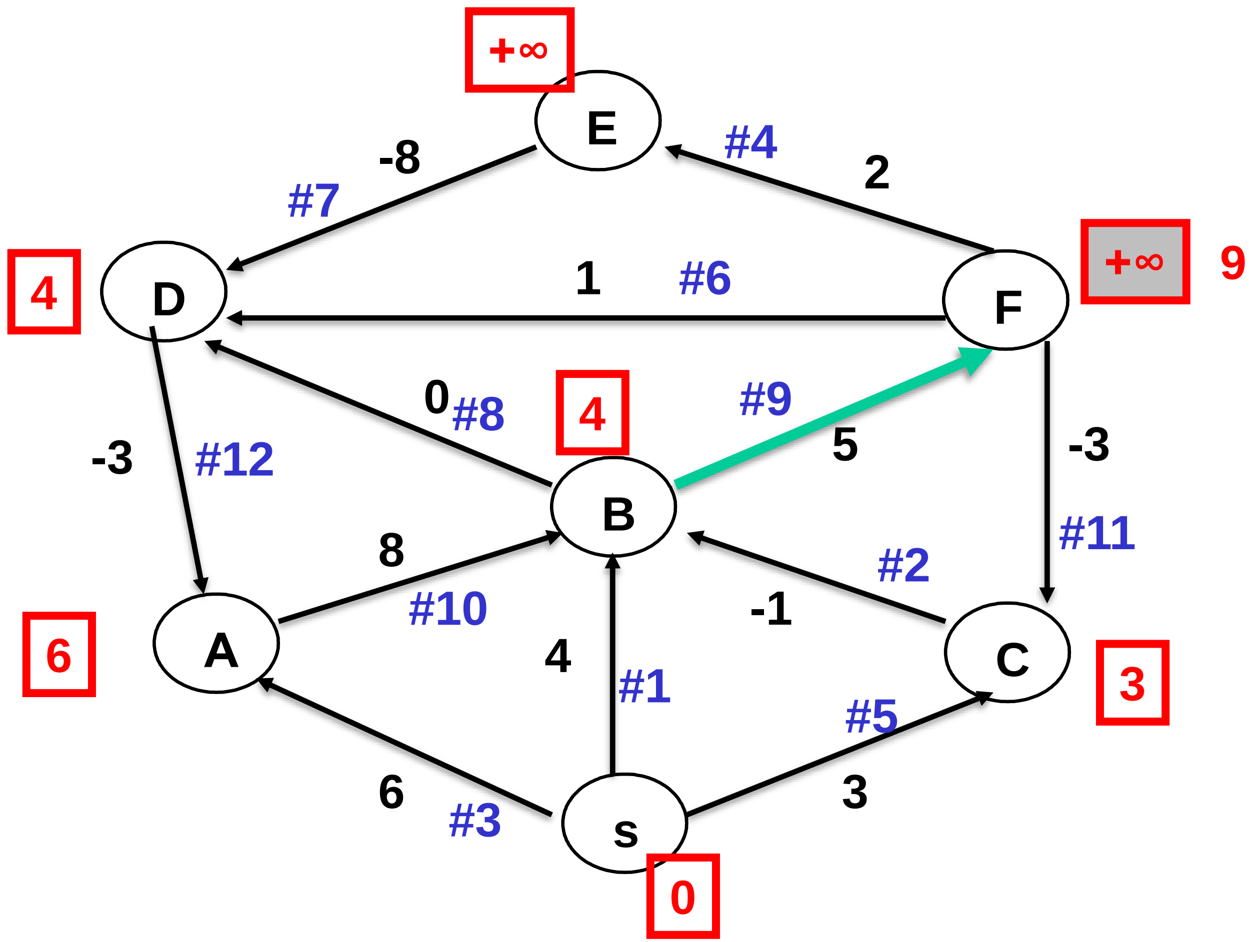
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 1, Edge #8**



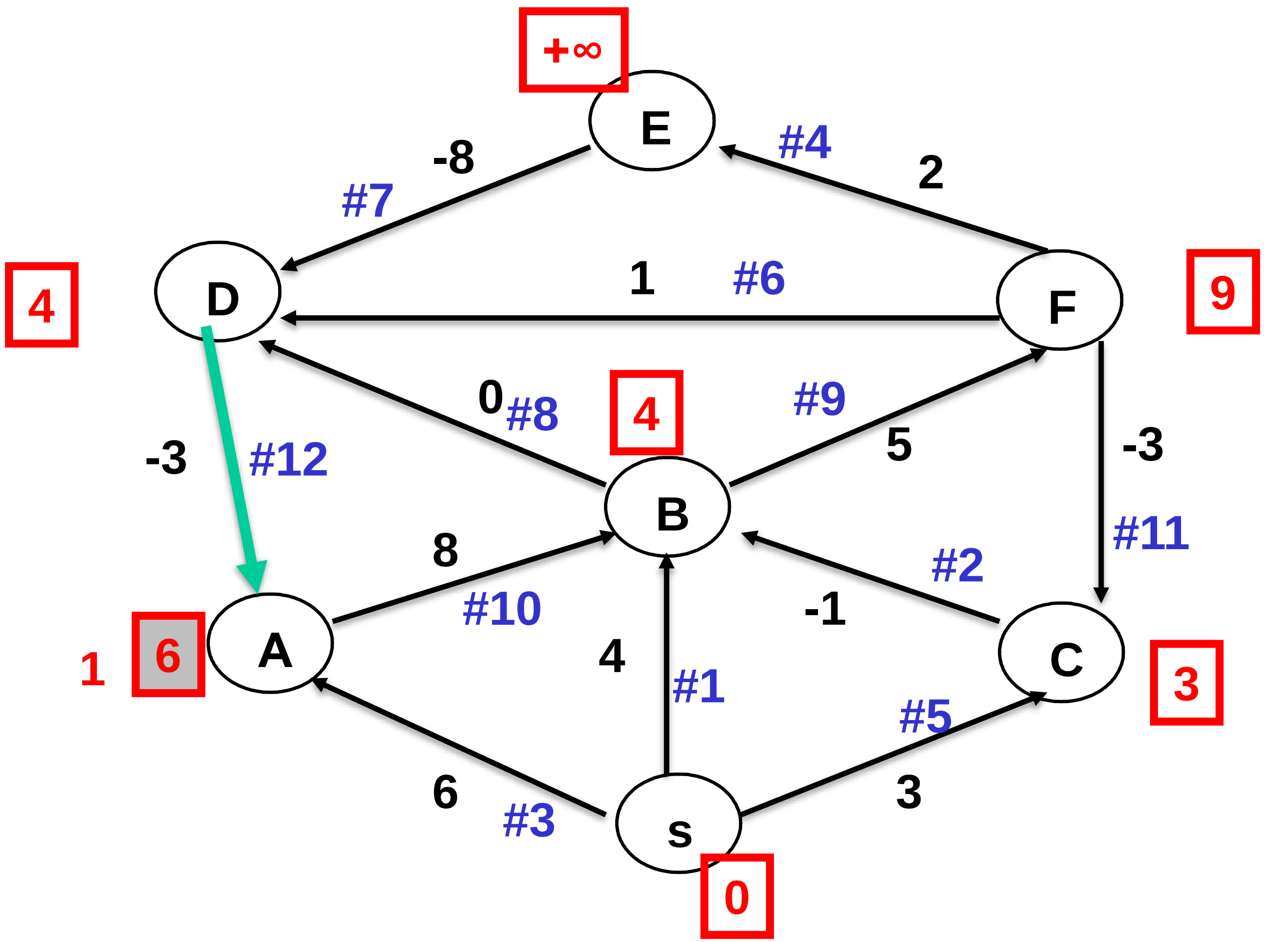
Bellman-Ford (Moore) Algorithm – Exercise 3

- Iteration 1, Edge #9



Bellman-Ford (Moore) Algorithm – Exercise 3

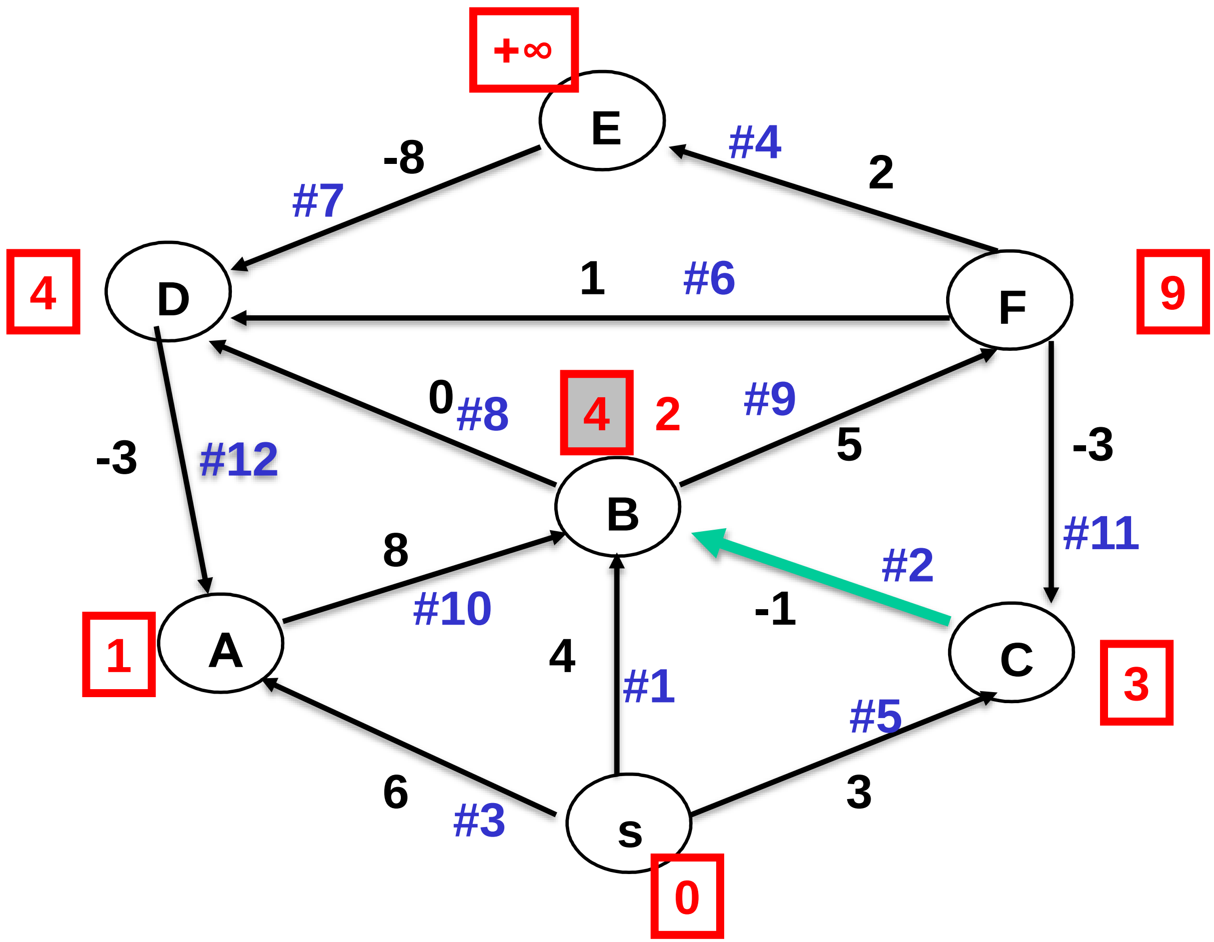
- **Iteration 1, Edge #12**



- *(This terminates the 1st iteration)*

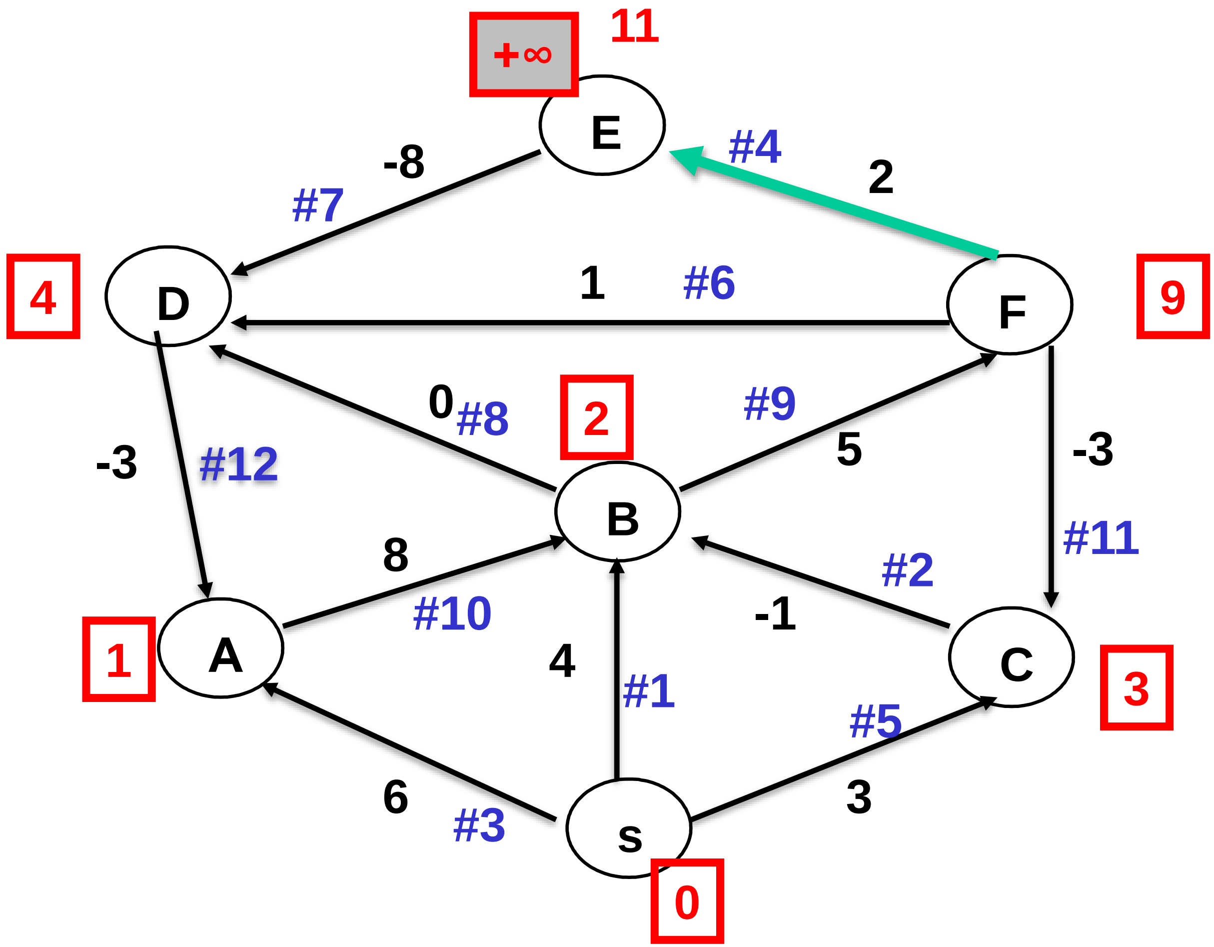
Bellman-Ford (Moore) Algorithm – Exercise 3

- Iteration 2, Edge #2



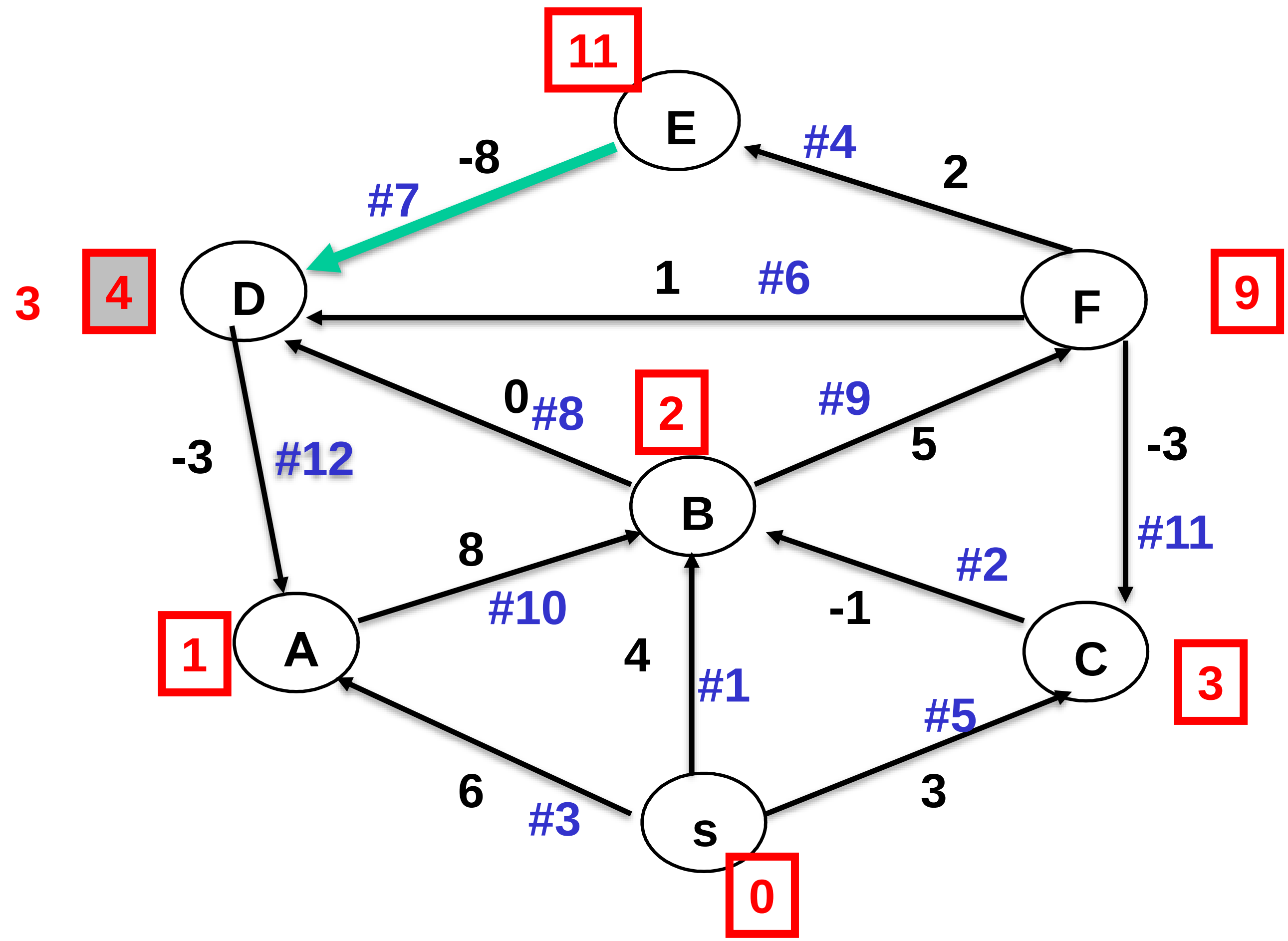
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 2, Edge #4**



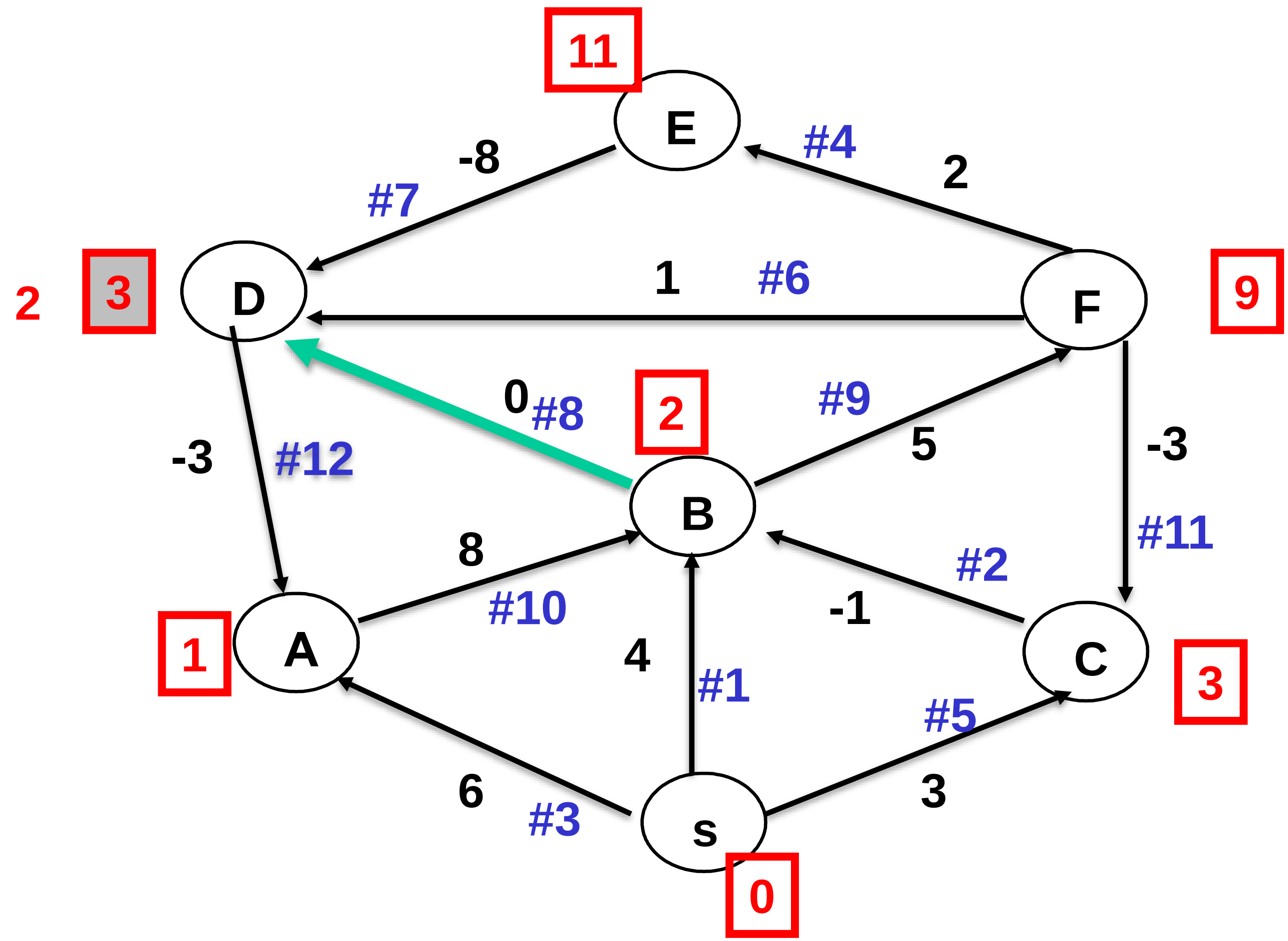
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 2, Edge #7**



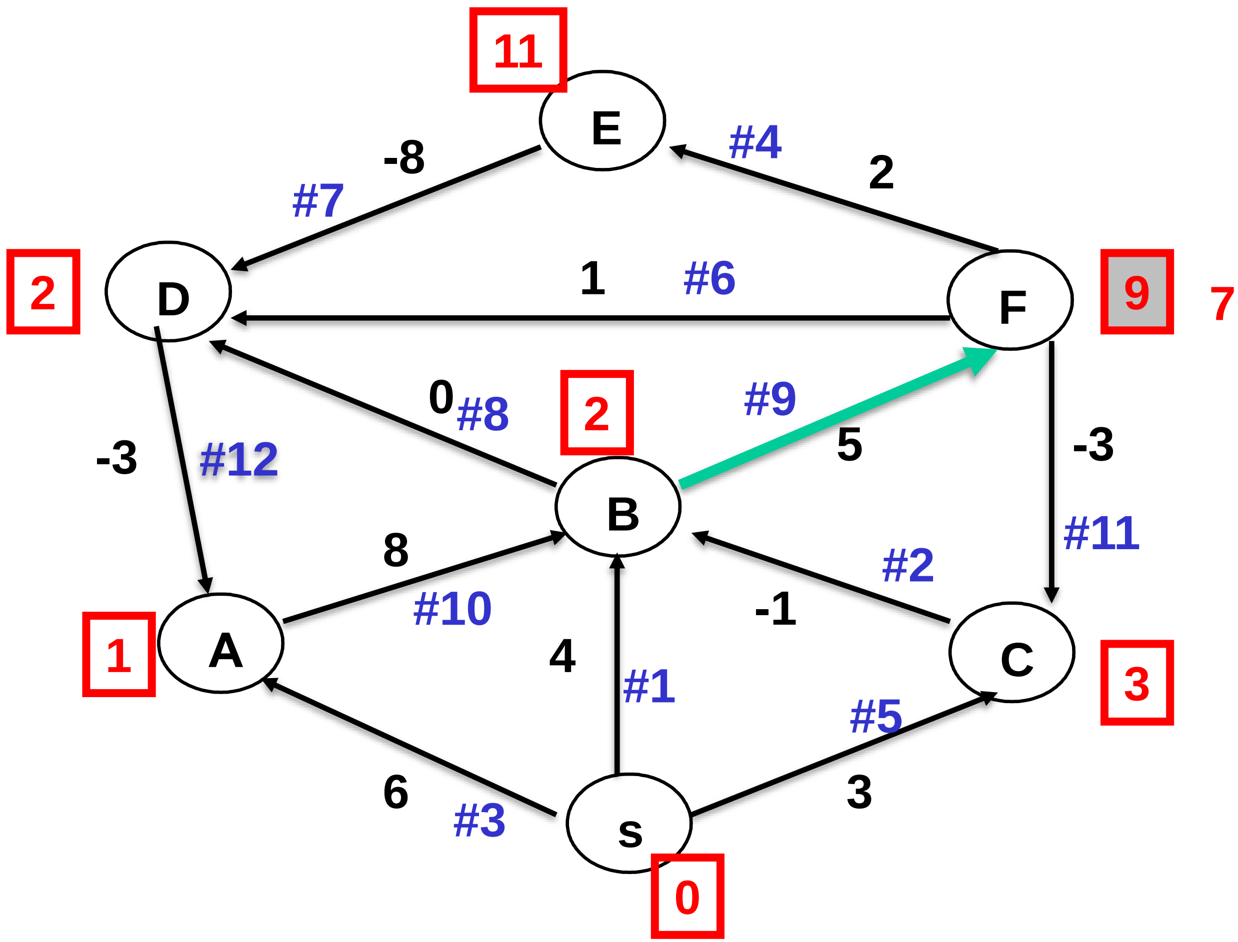
Bellman-Ford (Moore) Algorithm – Exercise 3

- Iteration 2, Edge #8



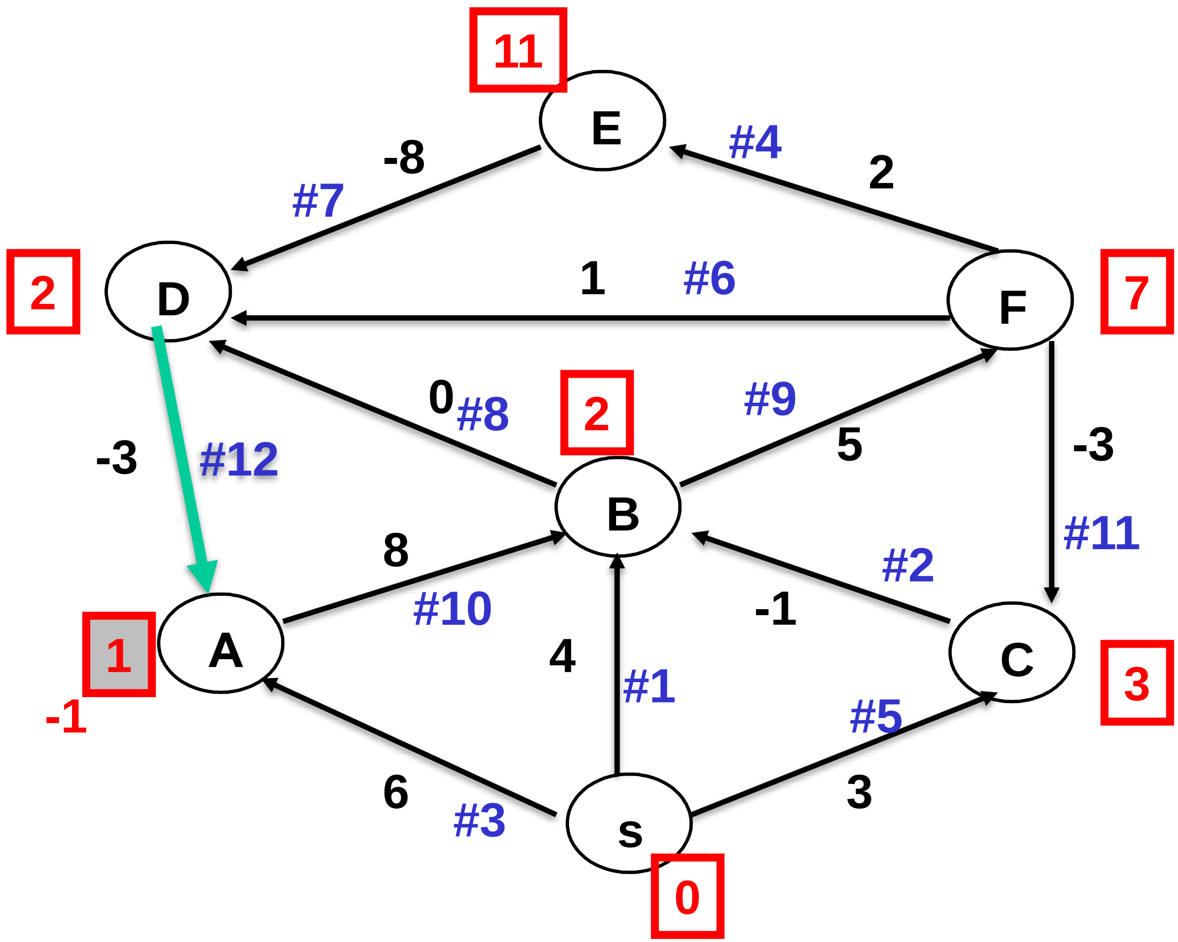
Bellman-Ford (Moore) Algorithm – Exercise 3

- Iteration 2, Edge #9



Bellman-Ford (Moore) Algorithm – Exercise 3

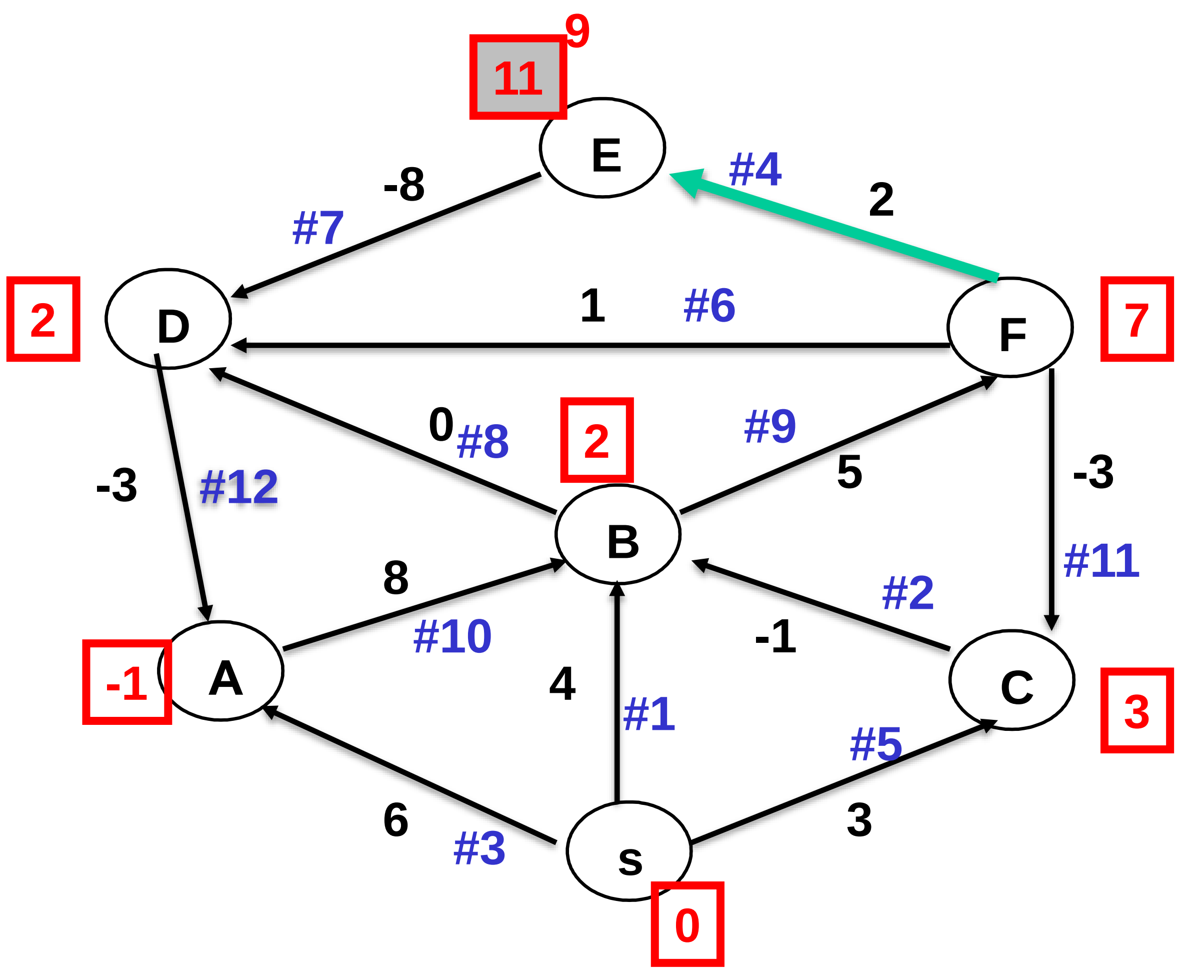
- Iteration 2, Edge #12



- *(This terminates the 2nd iteration)*

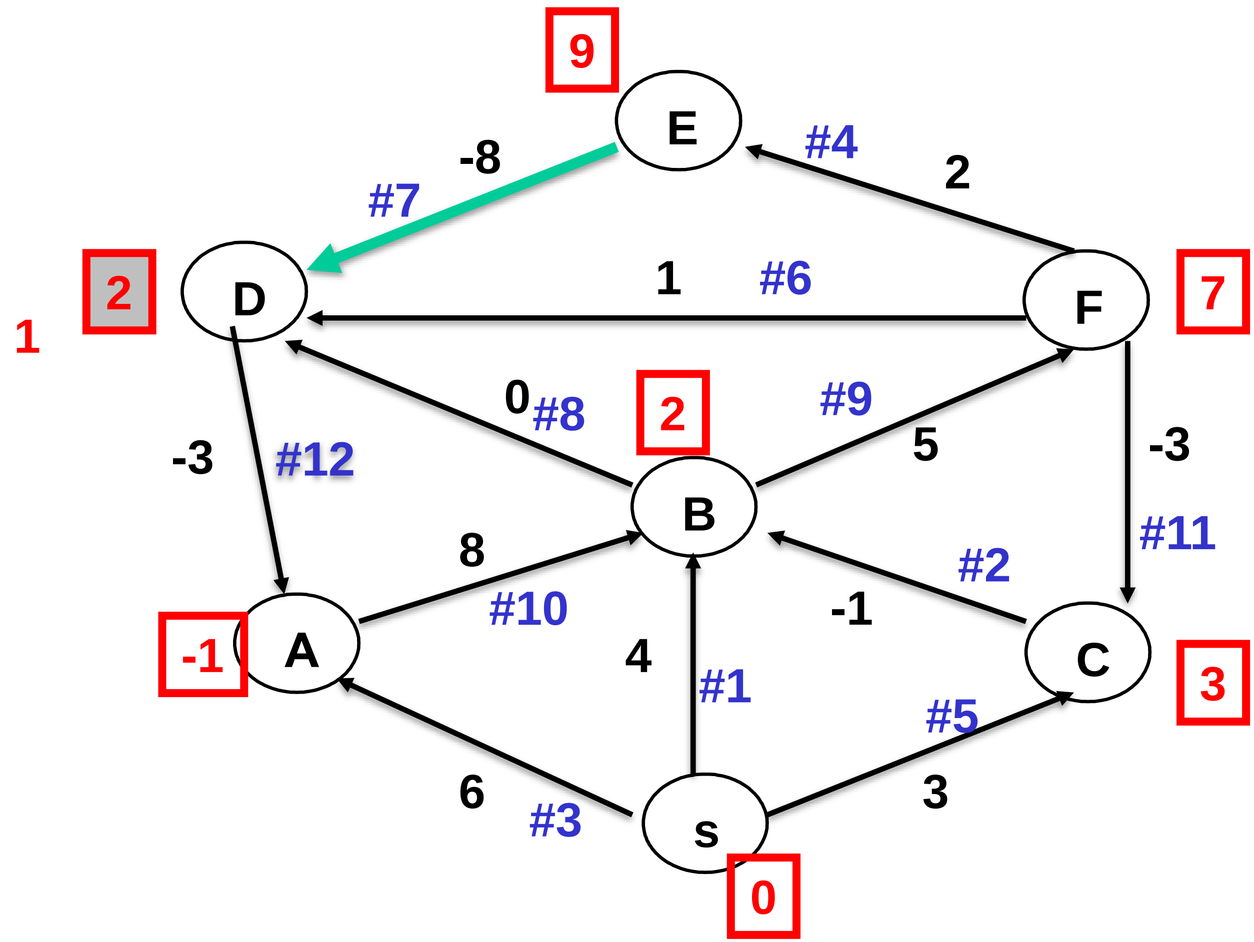
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 3, Edge #4**



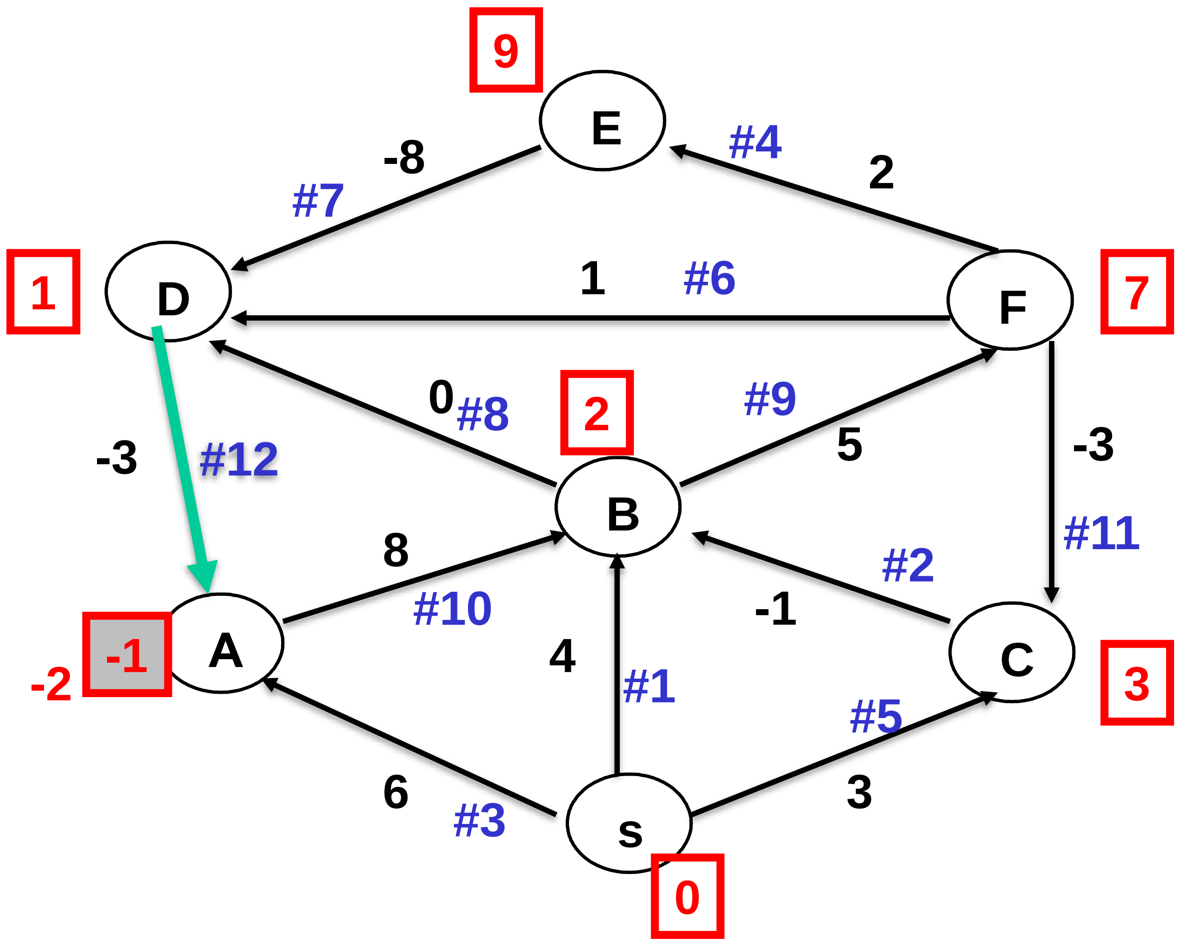
Bellman-Ford (Moore) Algorithm – Exercise 3

- **Iteration 3, Edge #7**



Bellman-Ford (Moore) Algorithm – Exercise 3

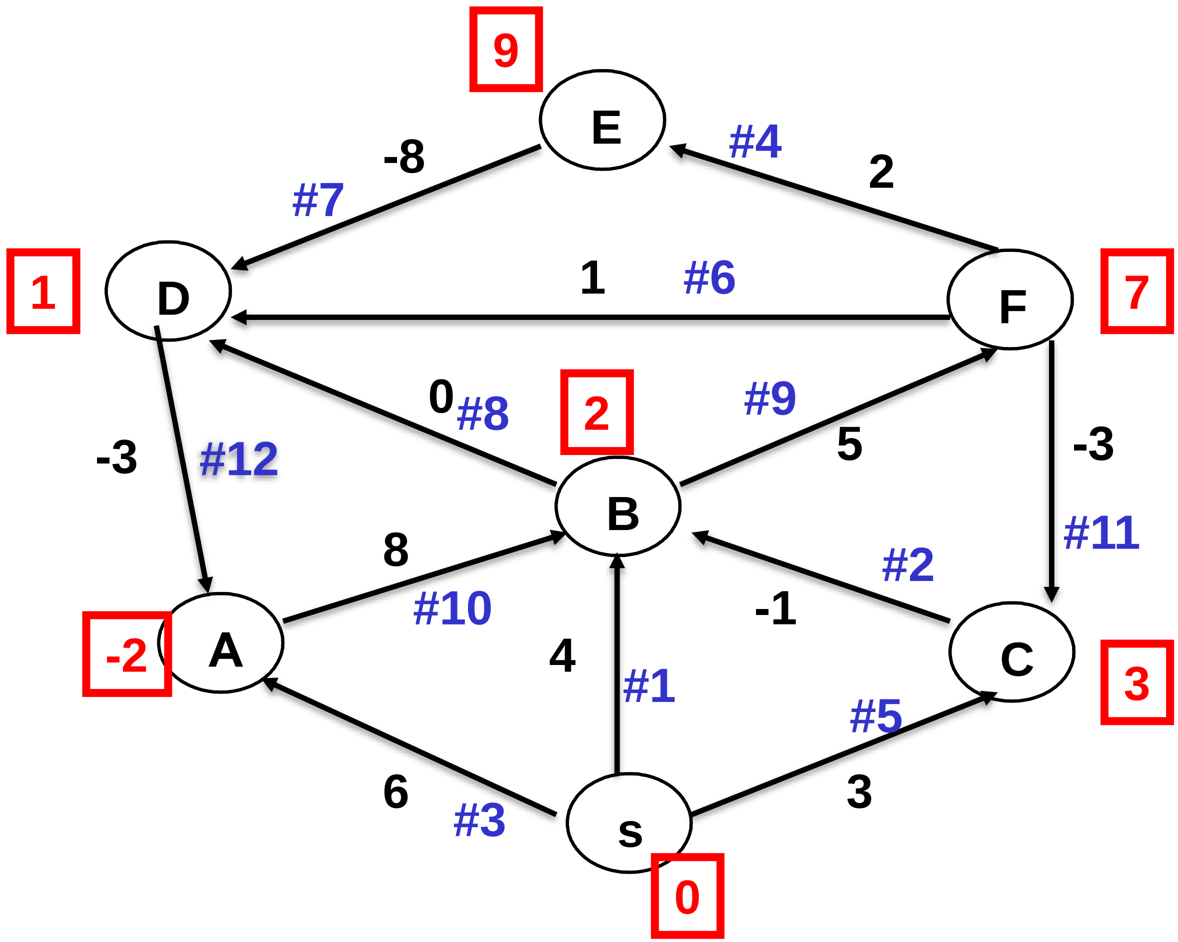
- **Iteration 3, Edge #12**



- ***(This terminates the 3d iteration)***

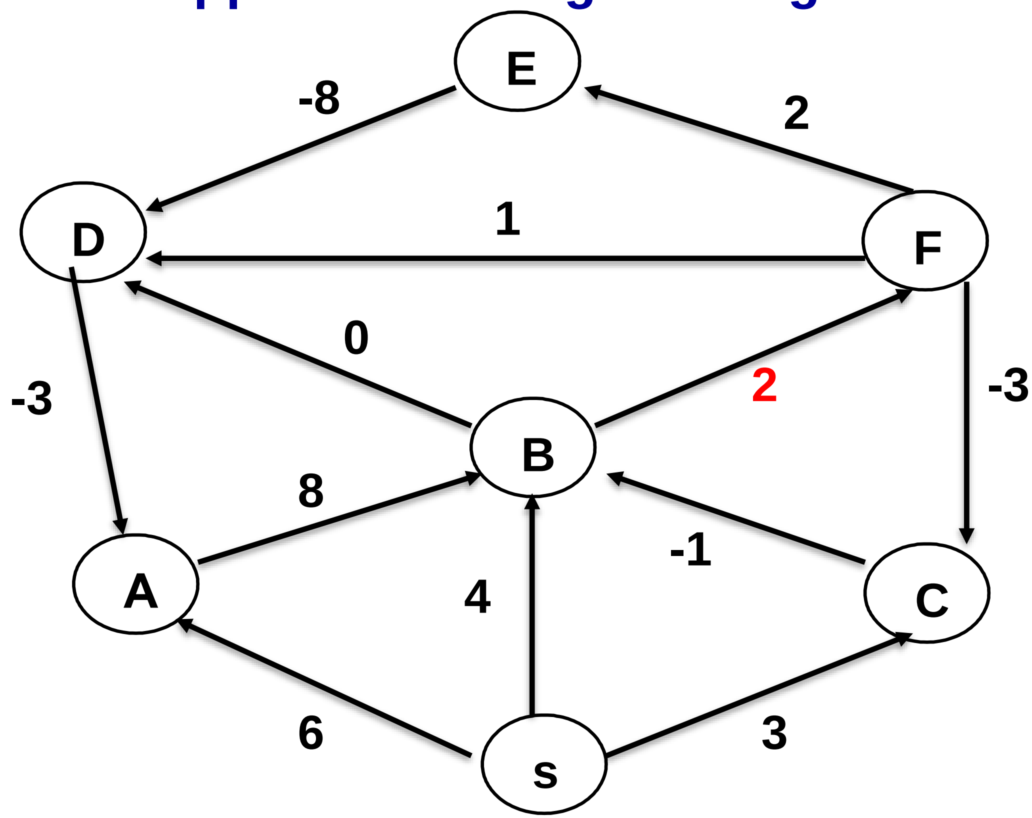
Bellman-Ford (Moore) Algorithm – Exercise 3

- No edges are relaxed at the 4th iteration
- The algorithm terminates! (The red number next to each node denotes the length of the shortest path from s to that node)

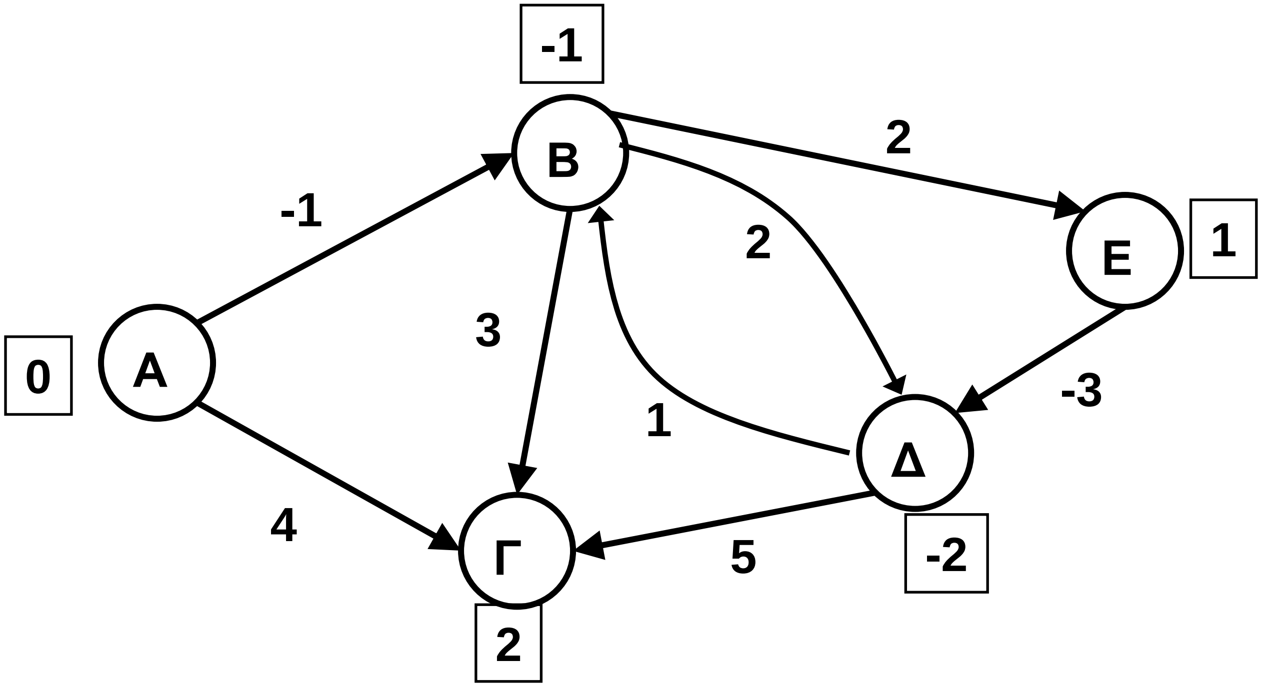


Homework

- What will happen if the weight of edge BF is equal to 2?



Bellman-Ford (Moore) Algorithm – Another Example



- **Let A be the origin**
- **The number next to each node represents the length of the shortest path from A**