UNIVERSITAT DE BARCELONA

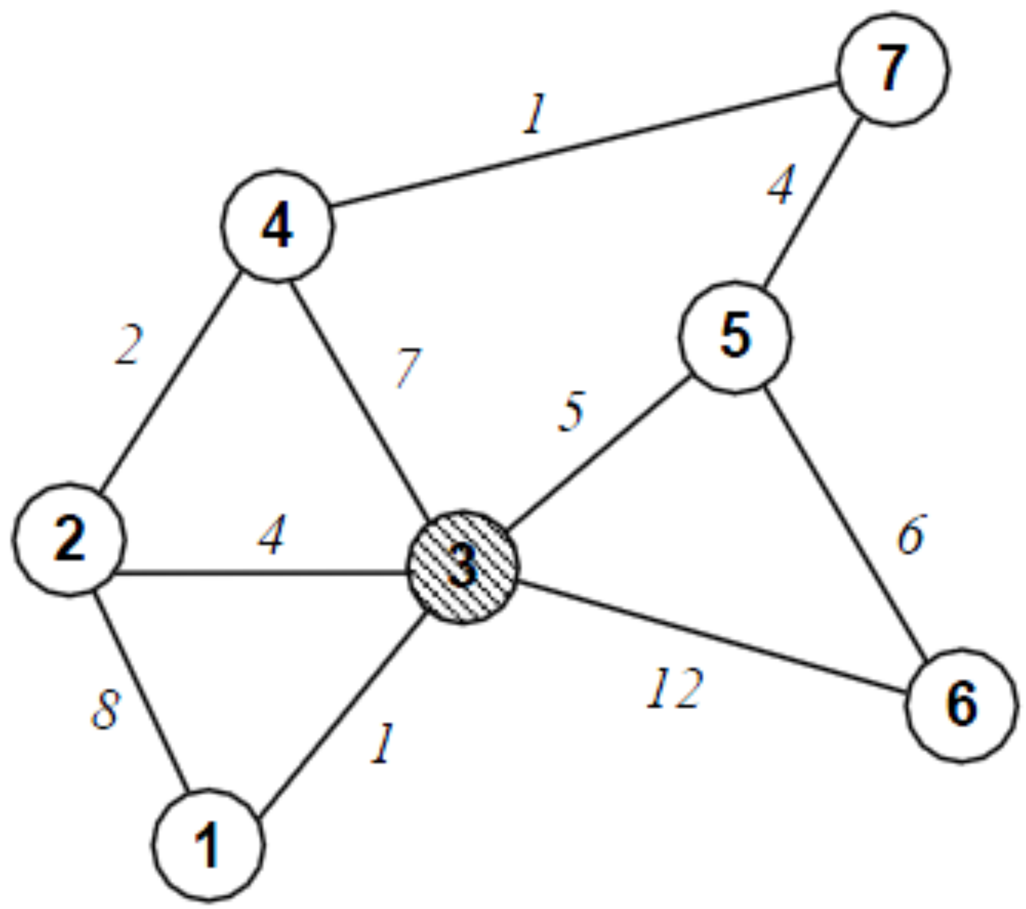
Grafs I

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2019-2020

Dijkstra

- Algorisme de **Dijkstra:** exemple graf **NO** dirigit



Dijkstra

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u:           // only v that are still in Q
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Dijkstra (with HEAP)

```
function Dijkstra(Graph, source):
    dist[source] ← 0                           // Initialization

    create vertex priority queue Q

    for each vertex v in Graph:
        if v ≠ source
            dist[v] ← INFINITY                 // Unknown distance from source to v
        prev[v] ← UNDEFINED                    // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                      // The main loop
        u ← Q.extract_min()                    // Remove and return best vertex
        for each neighbor v of u:              // only v that are still in Q
            alt ← dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist, prev
```

Dijkstra

- Complexitat
  - For any data structure for the vertex set Q, the running time is in

$$O(|E| \cdot T_{\mathrm{dk}} + |V| \cdot T_{\mathrm{em}}),$$

where and are the complexities of the *decrease-key* and *extract-minimum* operations in Q, respectively.

Implementation	extractMin	Insert/decreasekey	Total
Array			
Binary Heap			

Dijkstra

- Complexitat
  - For any data structure for the vertex set Q, the running time is in

$$O(|E| \cdot T_{\mathrm{dk}} + |V| \cdot T_{\mathrm{em}}),$$

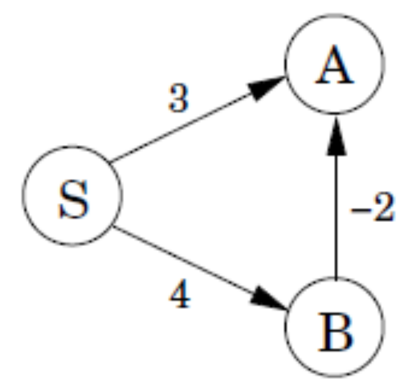
where and are the complexities of the *decrease-key* and *extract-minimum* operations in Q, respectively.

Implementation	extractMin	Insert/decreasekey	Total
Array	O(|V|)	O(1)	O(|V|^2)
Binary Heap	O(log|V|)	O(log|V|)	O(*(|V| + |E|)log(|V|)*)

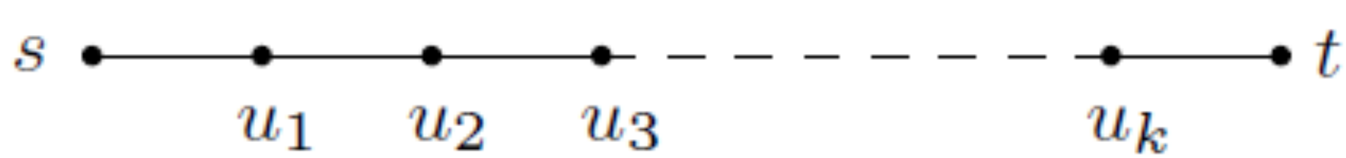
Dijkstra

- **Dijkstra** amb pesos negatius:

<u>procedure update</u>$((u,v) \in E)$
$\mathtt{dist}(v) = \min\{\mathtt{dist}(v), \mathtt{dist}(u) + l(u,v)\}$



- Amb Dijkstra sempre arribem de s a t amb camí mínim independentment de l'ordre dels pesos de les arestes si aquests són positius. **No amb negatius!**



Bellman-Ford

Bellman-Ford

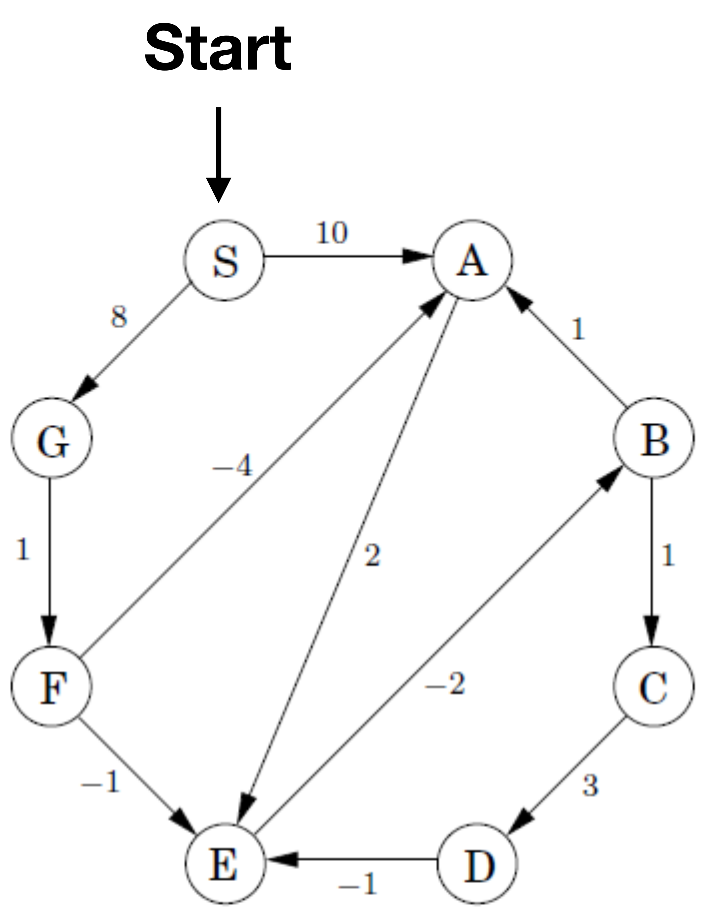
- **Solució?** Canviem l'algorisme perquè es calculin les distàncies simultàneament → actualitzar totes les arestes |V|-1 vegades (O(|V|·|E|))
  - Bellman-Ford

```
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)
```

```
procedure update((u, v) ∈ E)
dist(v) = min{dist(v), dist(u) + l(u, v)}
```

Bellman-Ford

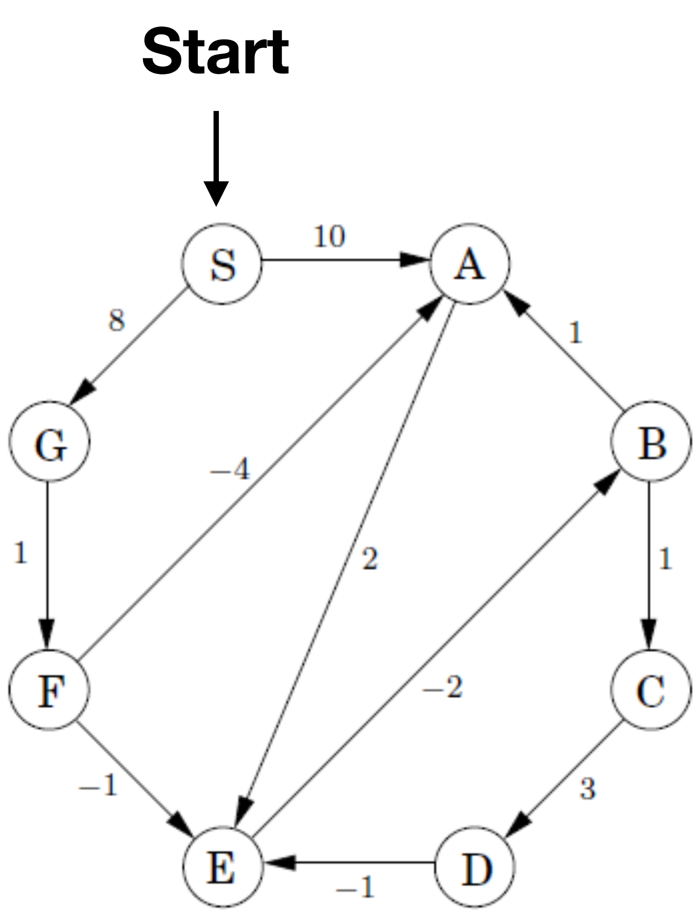
- Implementació: si en una iteració cap aresta e s'actualitza → finalitzar



	Iteration							
Node	0	1	2	3	4	5	6	7
S								
A								
B								
C								
D								
E								
F								
G								

Bellman-Ford

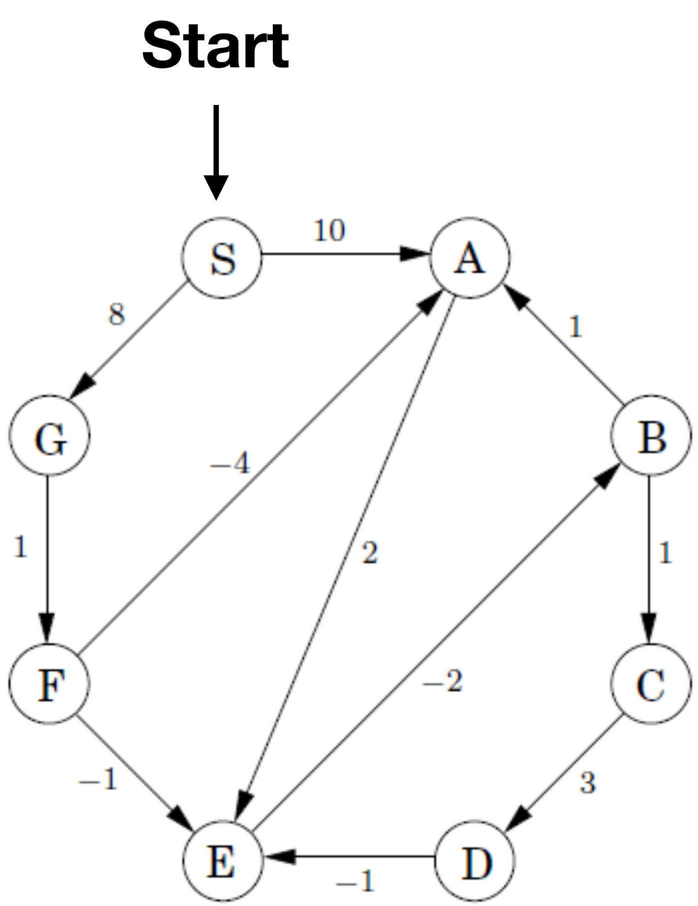
- Implementació: si en una iteració cap aresta e s’actualitza → finalitzar



	Iteration							
Node	0	1	2	3	4	5	6	7
S	0							
A	∞							
B	∞							
C	∞							
D	∞							
E	∞							
F	∞							
G	∞							

Bellman-Ford

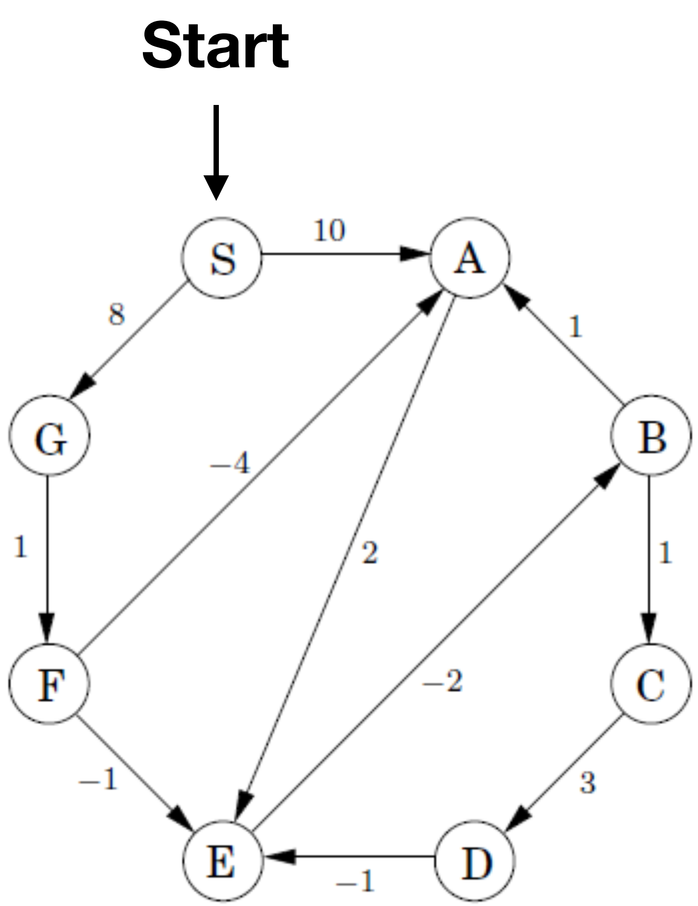
- Implementació: si en una iteració cap aresta e s'actualitza → finalitzar



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

Bellman-Ford

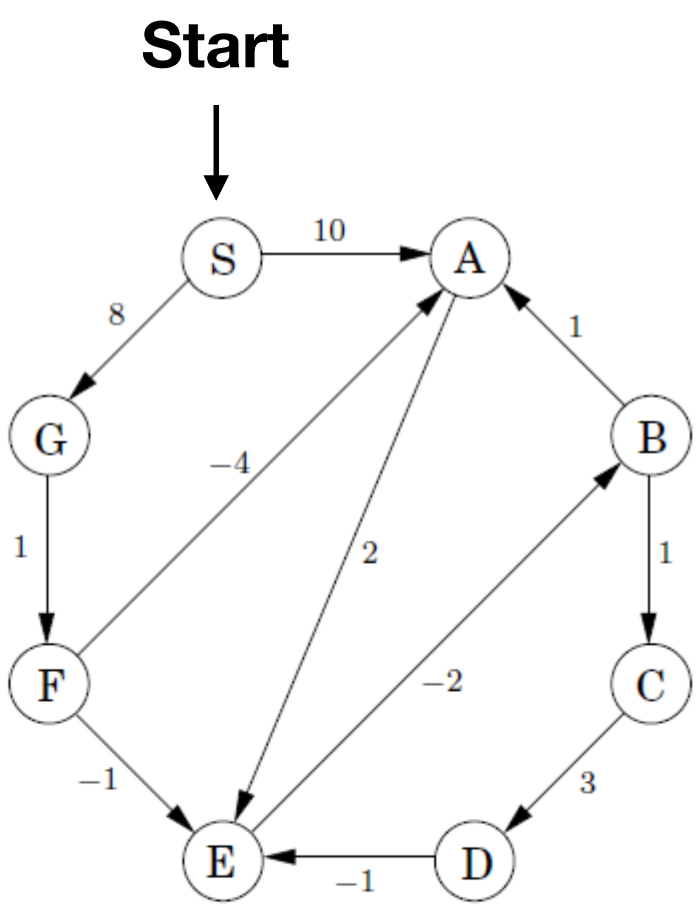
- Implementació: si en una iteració cap aresta e s'actualitza → finalitzar



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

Bellman-Ford

- Implementació: si en una iteració cap aresta e s'actualitza → finalitzar



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

Complexitat

```
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)
```

?

Complexitat

```
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)
```

$$O(|V| \cdot |E|)$$

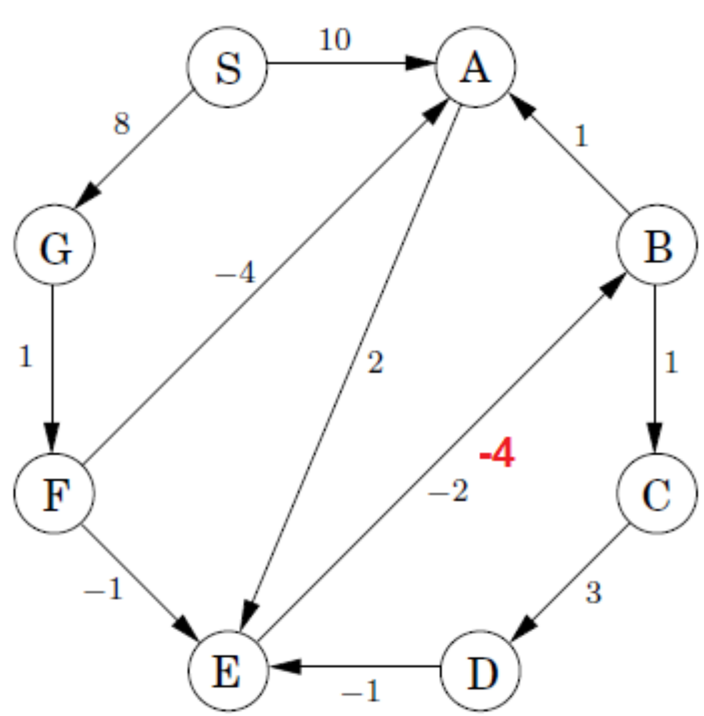
Problema:

Podem trobar **cicles amb cost negatiu** quant tenim arestes amb costos negatius.

Bellman-Ford

- **Cicles Negatius**

$$A \to E \to B \to A$$



Els podem trobar amb l'algorisme Bellman-Ford

El camí mínim té com a màxim longitud |V|-1
Podem detectar cicles si fem una iteració extra |V|

-> Hi ha cicle negatiu si a la iteració |V| alguna aresta és actualitzada

Per a un **gràfic general ponderat**, es poden calcular distàncies de font més curta en temps O(VE) utilitzant l'algoritme de **Bellman-Ford**.

Per a un graf **sense pesos negatius**, podem fer millor i calcular distàncies de la font més curta en temps O(E + VLogV) utilitzant l'algorisme de **Dijkstra**.

Podem millorar encara el gràfic acíclic dirigit (DAG)?

Camí més curt

- Hi ha dos tipus de grafs que no tenen **cicles negatius**: sense pesos negatius sense cicles
- El primer és directe. Per resoldre el camí mínim en acíclic grafs dirigits negatius:
  - Linealitzar usant DFS
    - **Temps lineal!**
- Si posem els negatius dels pesos podem trobar els camins de longitud màxima

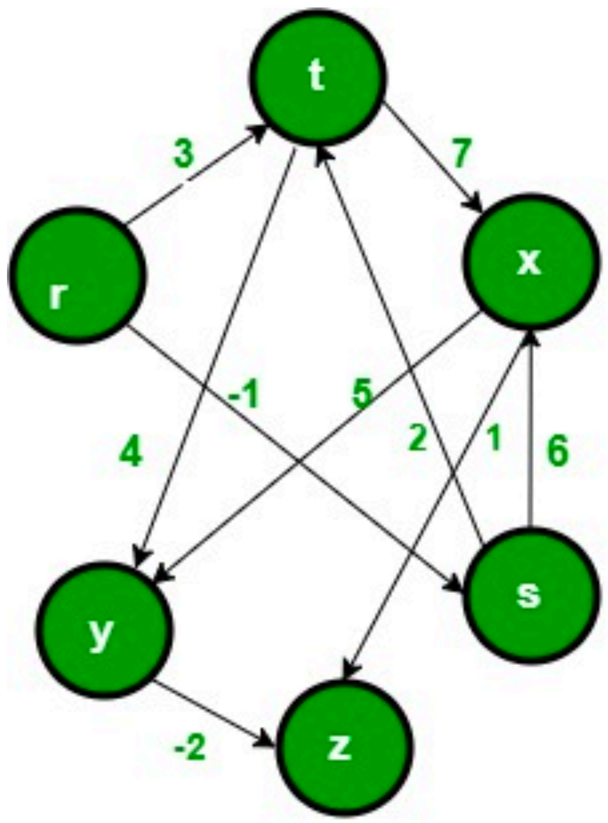
Grafs amb cicles

- Linealitzar usant DFS

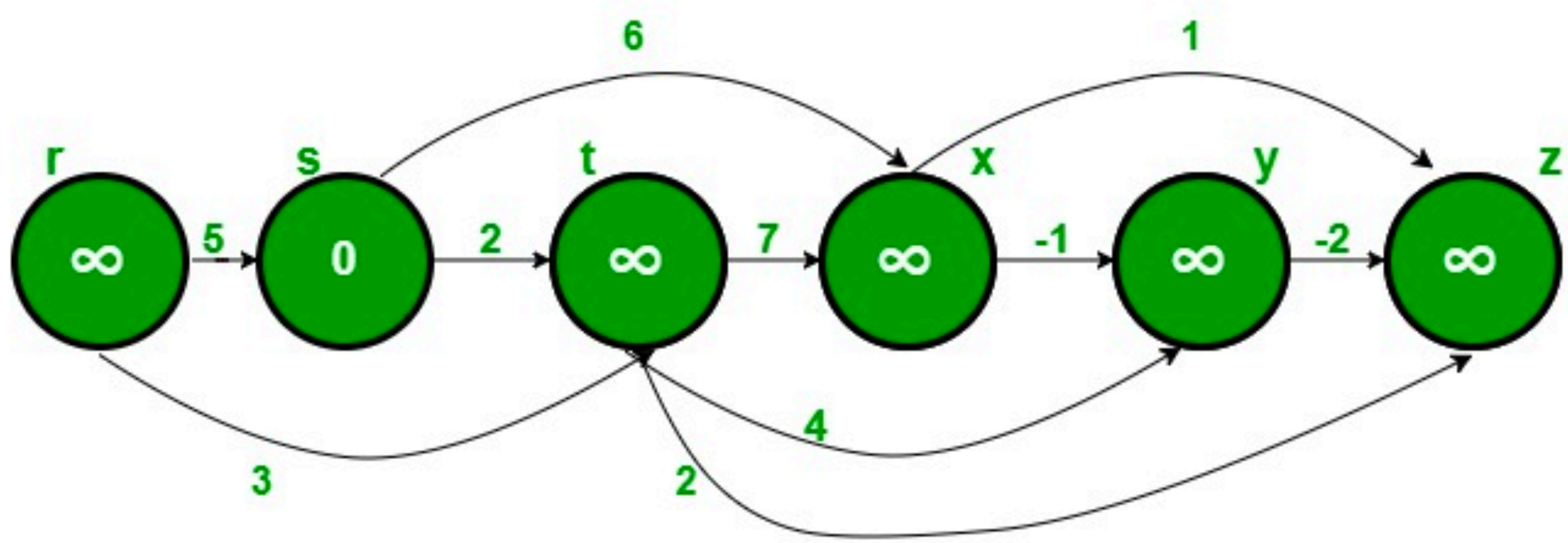
```
procedure dag-shortest-paths(G, l, s)
Input:    Dag G = (V, E);
          edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

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)
```

Graf amb cicles

- Pas 1: linearitzar el graf (topological SORT amb DFS)

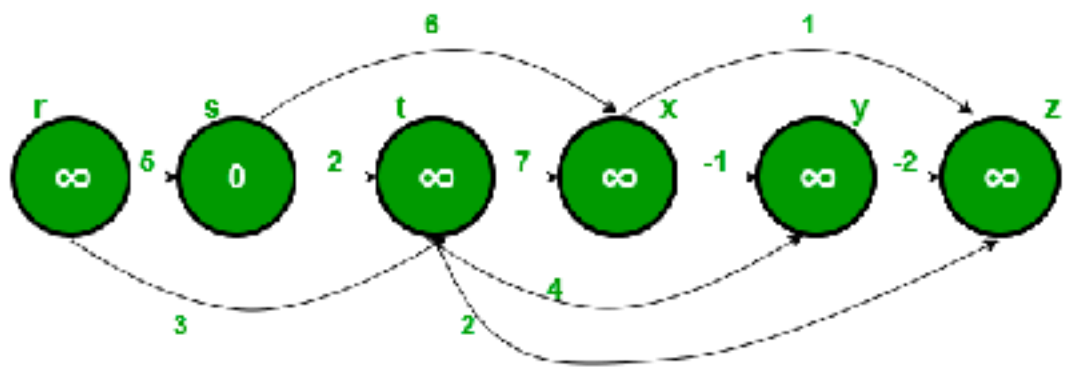


(a)

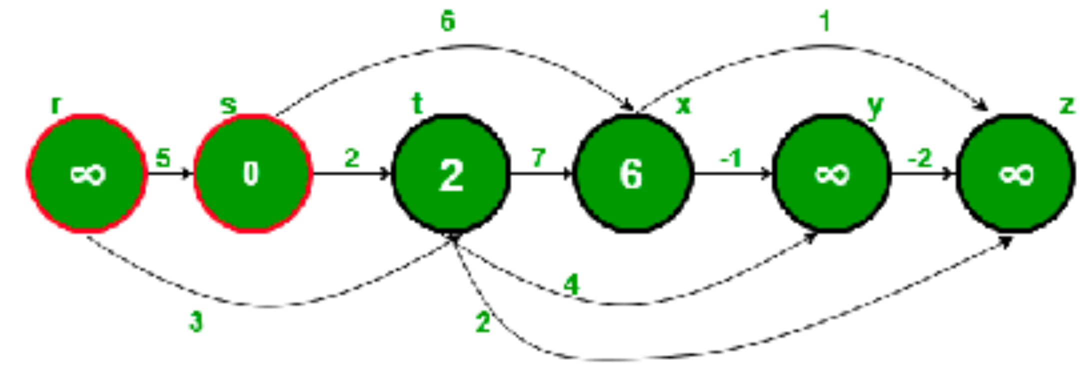


(b)

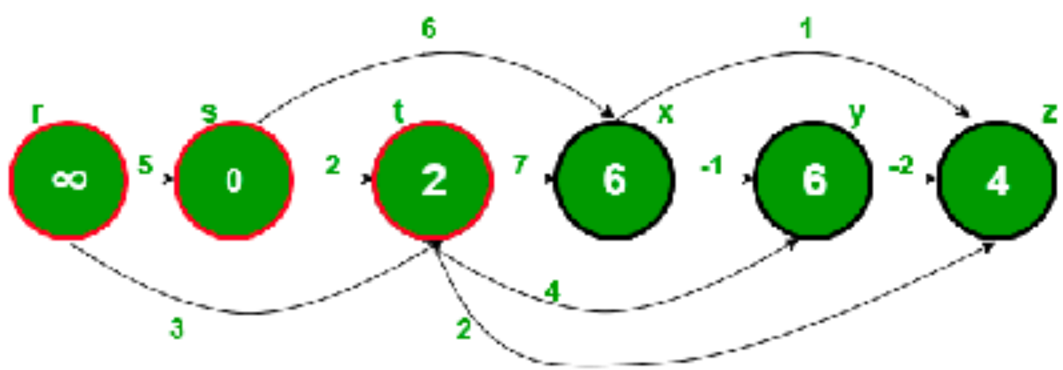
Graf amb cicles



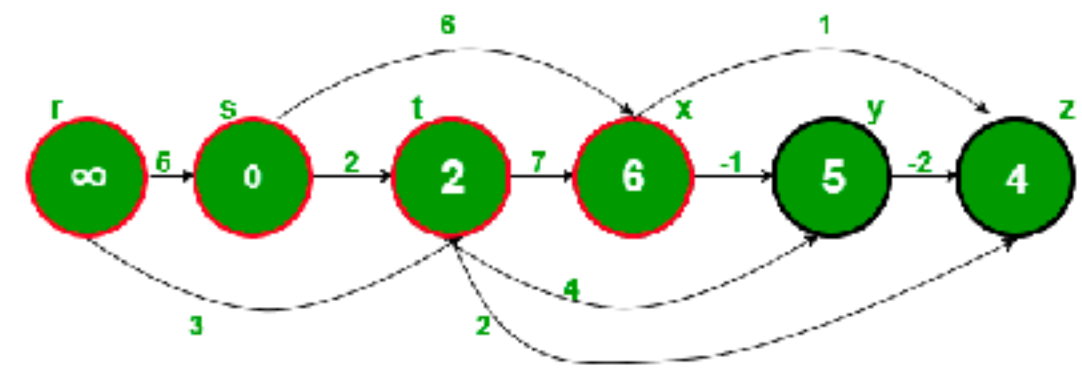
(c)



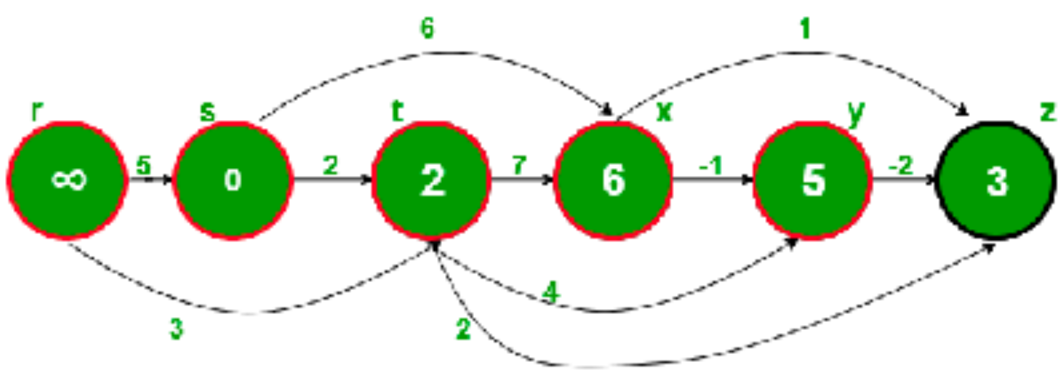
(d)



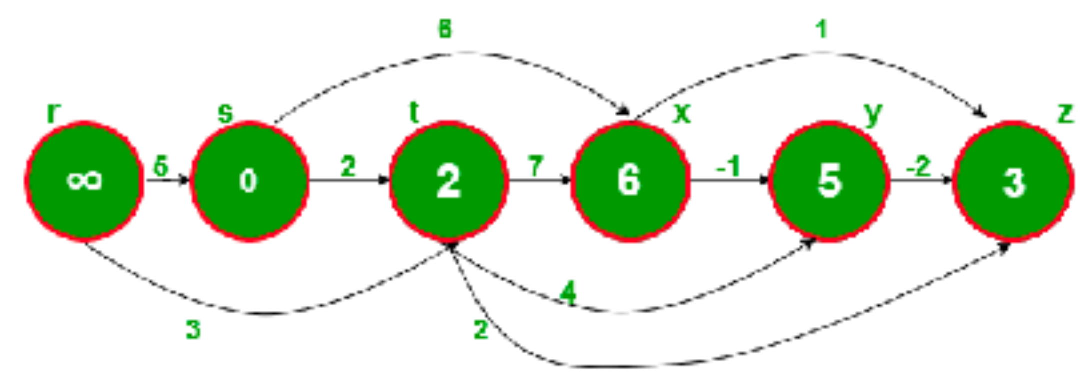
(e)



(f)

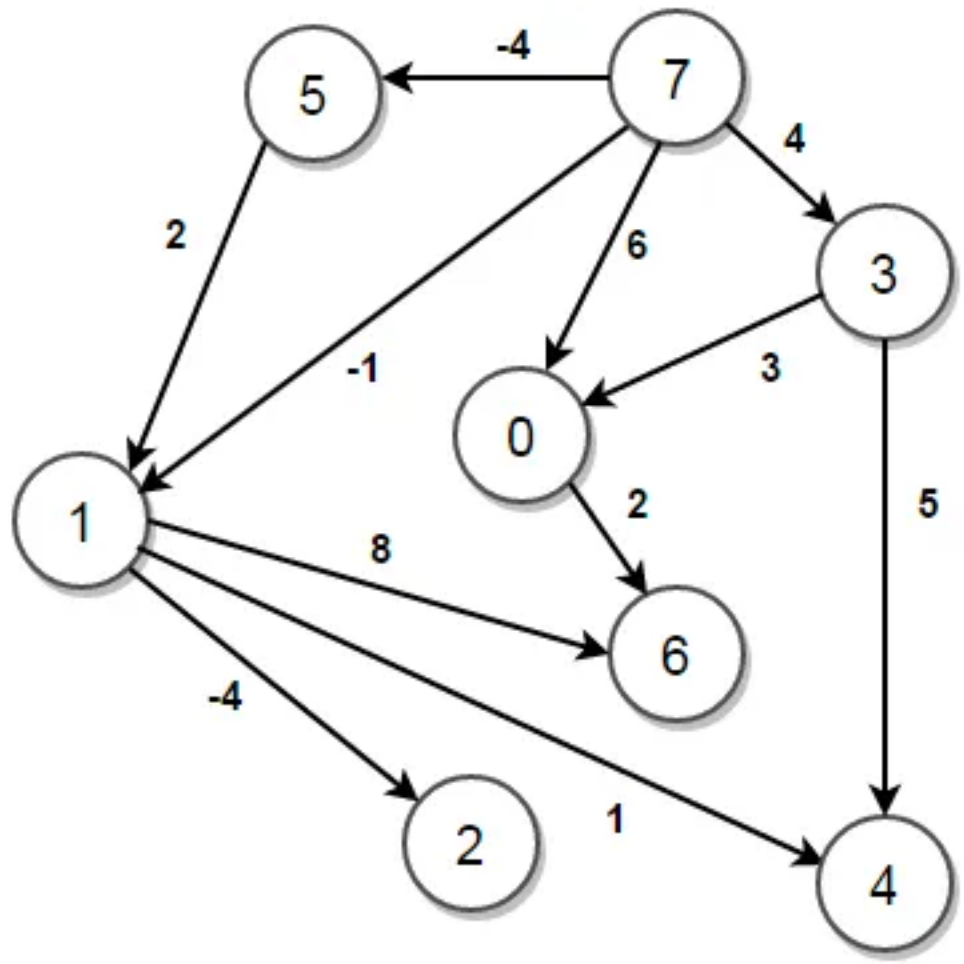


(g)

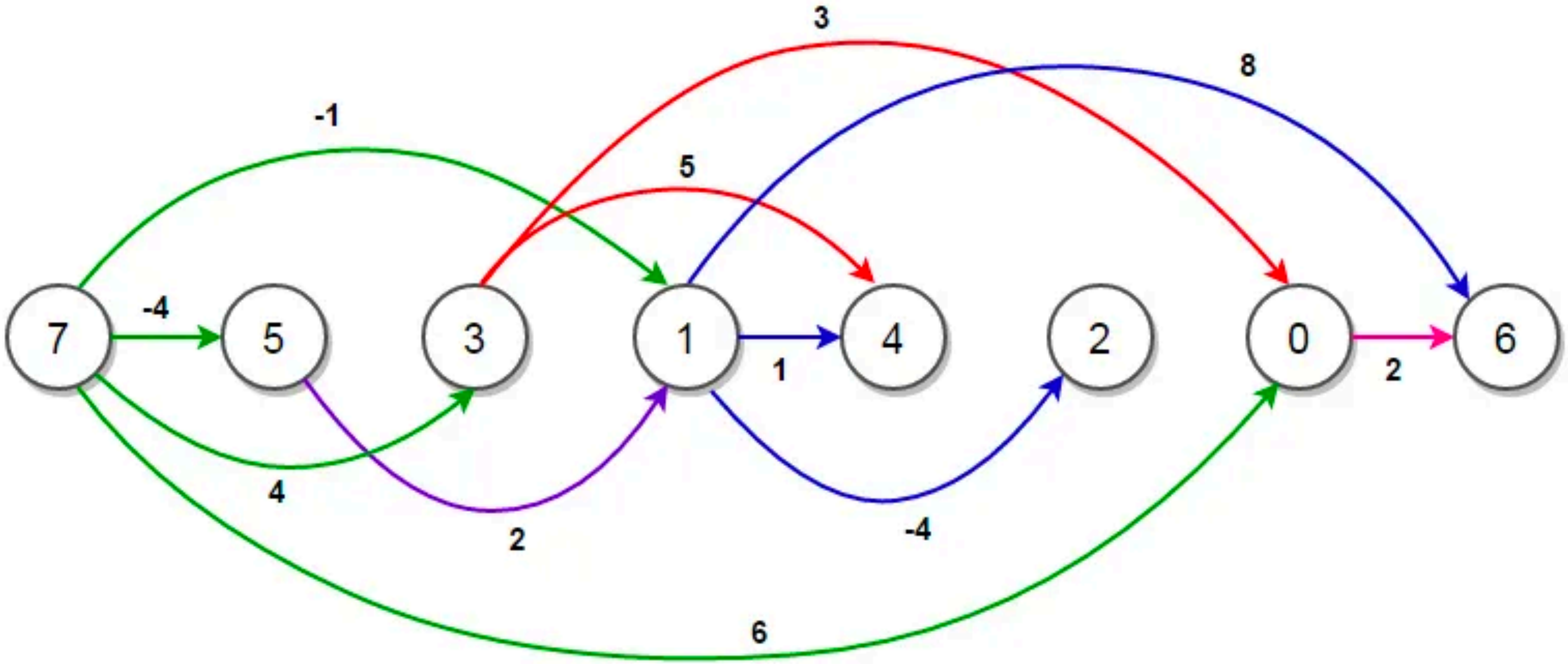


(h)

Start



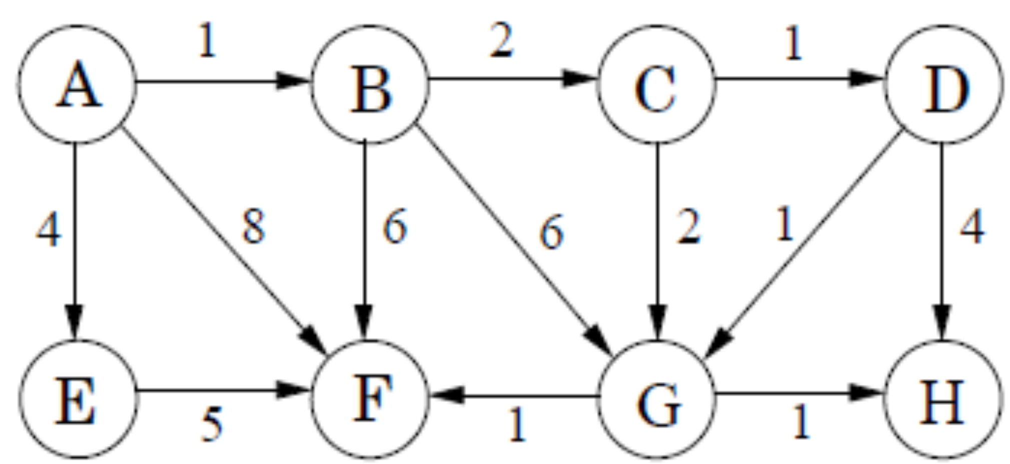
Topological order



Topological Order

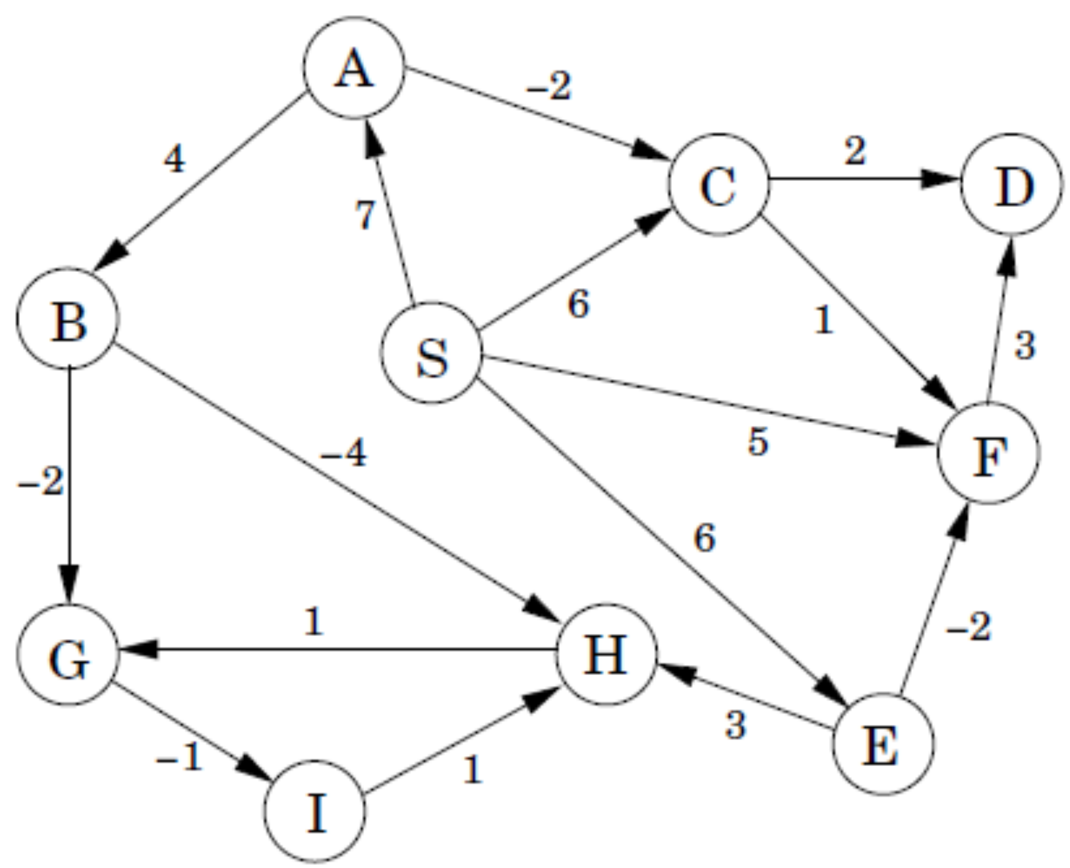
Exercici: Dijkstra

- Començant al node A: dibuixa la taula de distàncies immediates a tots els nodes a cada iteració.
- Mostra l’arbre de camins mínims



Exercici: Bellman-Ford

- Començant al node A: dibuixa la taula de distàncies immediates a tots els nodes a cada iteració.



Fins ara hem vist com calcular el camí més curt des d'un node **u** donat.

Com calculem el camí més curts de tots el nodes?