



UNIVERSITAT DE  
BARCELONA



# Grafs IIII

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2021-2022

# Sessió

- Floyd-Warshall
- Flux Màxim
  - Algorítmic Ford-Fulkerson
- Cami/Circuit Eulerià

# All Short Paths

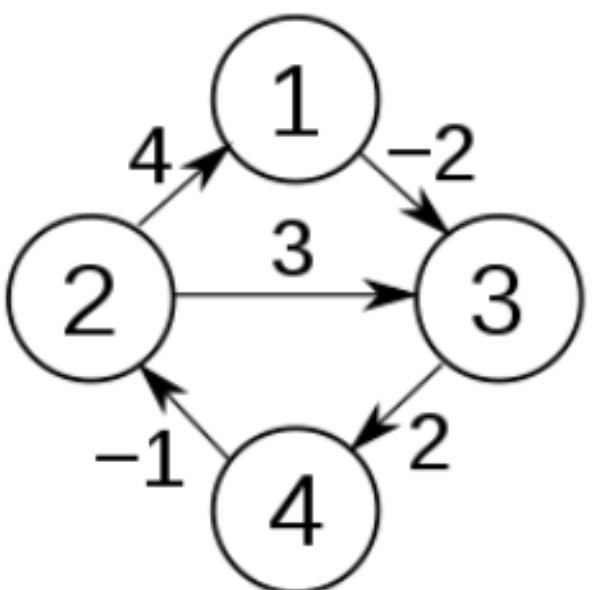
- Donat un graf **G** volem trobar el camí més curt entre totes les parelles dels seus nodes.
- Podem utilitzar Dijkstra per trobar tots els camins més curts?
  - Com ho fem?
  - Quina serà la seva complexitat?

# All Short Paths

- Donat un graf **G** volem trobar el camí més curt entre totes les parelles dels seus nodes.
- Podem utilitzar Dijkstra per trobar tots els camins més curts? **Sí!**
  - Com ho fem? **Executem Dijkstra per a cada node del nostre graf**
  - Quina serà la seva complexitat? **La complexitat de Dijkstra mitjançant (cua prioritària) és  $O(V + E \log(V))$** 
    - Per tant per trobar els camins mínims entre totes les parelles de nodes, la complexitat serà:  $O(|V|^2 + |V||E| \log(V))$

# All Short Paths - Floyd Warshall

```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 for each edge (u,v)
3     dist[u][v] ← w(u,v) // the weight of the edge (u,v)
4 for each vertex v
5     dist[v][v] ← 0
6 for k from 1 to |V|
7     for i from 1 to |V|
8         for j from 1 to |V|
9             if dist[i][j] > dist[i][k] + dist[k][j]
10                dist[i][j] ← dist[i][k] + dist[k][j]
11 end if
```

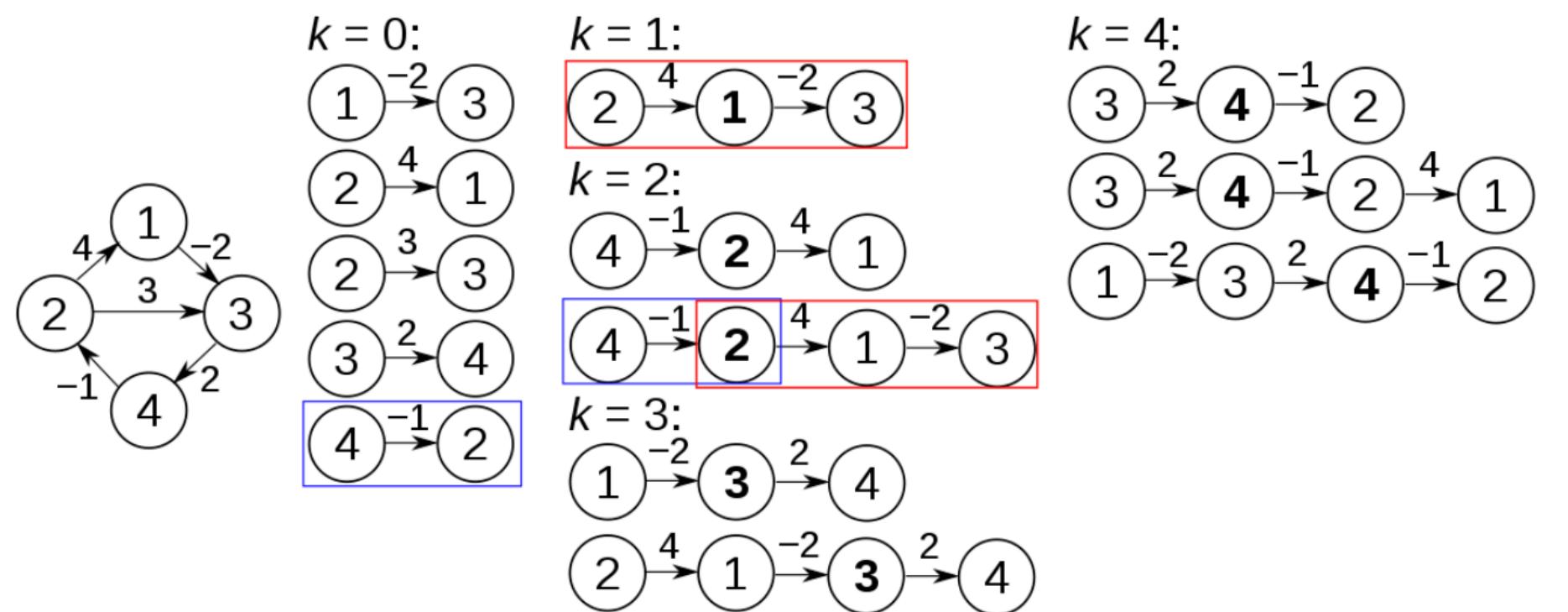


Trobeu la solució

# Exercici: All Short Paths

## Example [\[edit\]](#)

The algorithm above is executed on the graph on the left below:



Prior to the first recursion of the outer loop, labeled  $k = 0$  above, the only known paths correspond to the single edges in the graph. At  $k = 1$ , paths that go through the vertex 1 are found: in particular, the path [2,1,3] is found, replacing the path [2,3] which has fewer edges but is longer (in terms of weight). At  $k = 2$ , paths going through the vertices {1,2} are found. The red and blue boxes show how the path [4,2,1,3] is assembled from the two known paths [4,2] and [2,1,3] encountered in previous iterations, with 2 in the intersection. The path [4,2,3] is not considered, because [2,1,3] is the shortest path encountered so far from 2 to 3. At  $k = 3$ , paths going through the vertices {1,2,3} are found. Finally, at  $k = 4$ , all shortest paths are found.

The distance matrix at each iteration of  $k$ , with the updated distances in **bold**, will be:

		$j$			
$i$		1	2	3	4
$k = 0$	1	0	$\infty$	-2	$\infty$
2	4	0	3	$\infty$	
3	$\infty$	$\infty$	0	2	
4	$\infty$	-1	$\infty$	0	

		$j$			
$i$		1	2	3	4
$k = 1$	1	0	$\infty$	-2	$\infty$
2	4	0	2	$\infty$	
3	$\infty$	$\infty$	0	2	
4	$\infty$	-1	$\infty$	0	

		$j$			
$i$		1	2	3	4
$k = 2$	1	0	$\infty$	-2	$\infty$
2	4	0	2	$\infty$	
3	$\infty$	$\infty$	0	2	
4	$\infty$	-1	1	0	

		$j$			
$i$		1	2	3	4
$k = 3$	1	0	$\infty$	-2	0
2	4	0	2	$\infty$	
3	$\infty$	$\infty$	0	2	
4	3	-1	1	0	

		$j$			
$i$		1	2	3	4
$k = 4$	1	0	-1	-2	0
2	4	0	2	4	
3	5	1	0	2	
4	3	-1	1	0	

# All Short Paths - Floyd Warshall

- L'algorisme de **Floyd-Warshall** proporciona les longituds dels camins mínims entre tots els parells de vèrtexs. Però com obtenim els camins??
- Amb petites modificacions és possible crear un mètode per a reconstruir el camí real entre dos vèrtexs.

# All Short Paths - Floyd Warshall

- L'algorisme de **Floyd-Warshall** proporciona les longituds dels camins mínims entre tots els parells de vèrtexs. Però com obtenim els camins??
- Amb petites modificacions és possible crear un mètode per a reconstruir el camí real entre dos vèrtexs.

```
let dist be a |V| × |V| array of minimum distances initialized to  $\infty$  (infinity)
let next be a |V| × |V| array of vertex indices initialized to null
```

```
procedure FloydWarshallWithPathReconstruction() is
    for each edge (u, v) do
        dist[u][v] ← w(u, v) // The weight of the edge (u, v)
        next[u][v] ← v
    for each vertex v do
        dist[v][v] ← 0
        next[v][v] ← v
    for k from 1 to |V| do // standard Floyd-Warshall implementation
        for i from 1 to |V|
            for j from 1 to |V|
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    next[i][j] ← next[i][k]
```

```
procedure Path(u, v)
    if next[u][v] = null then
        return []
    path = [u]
    while u ≠ v
        u ← next[u][v]
        path.append(u)
    return path
```

# All Short Paths - Floyd Warshall

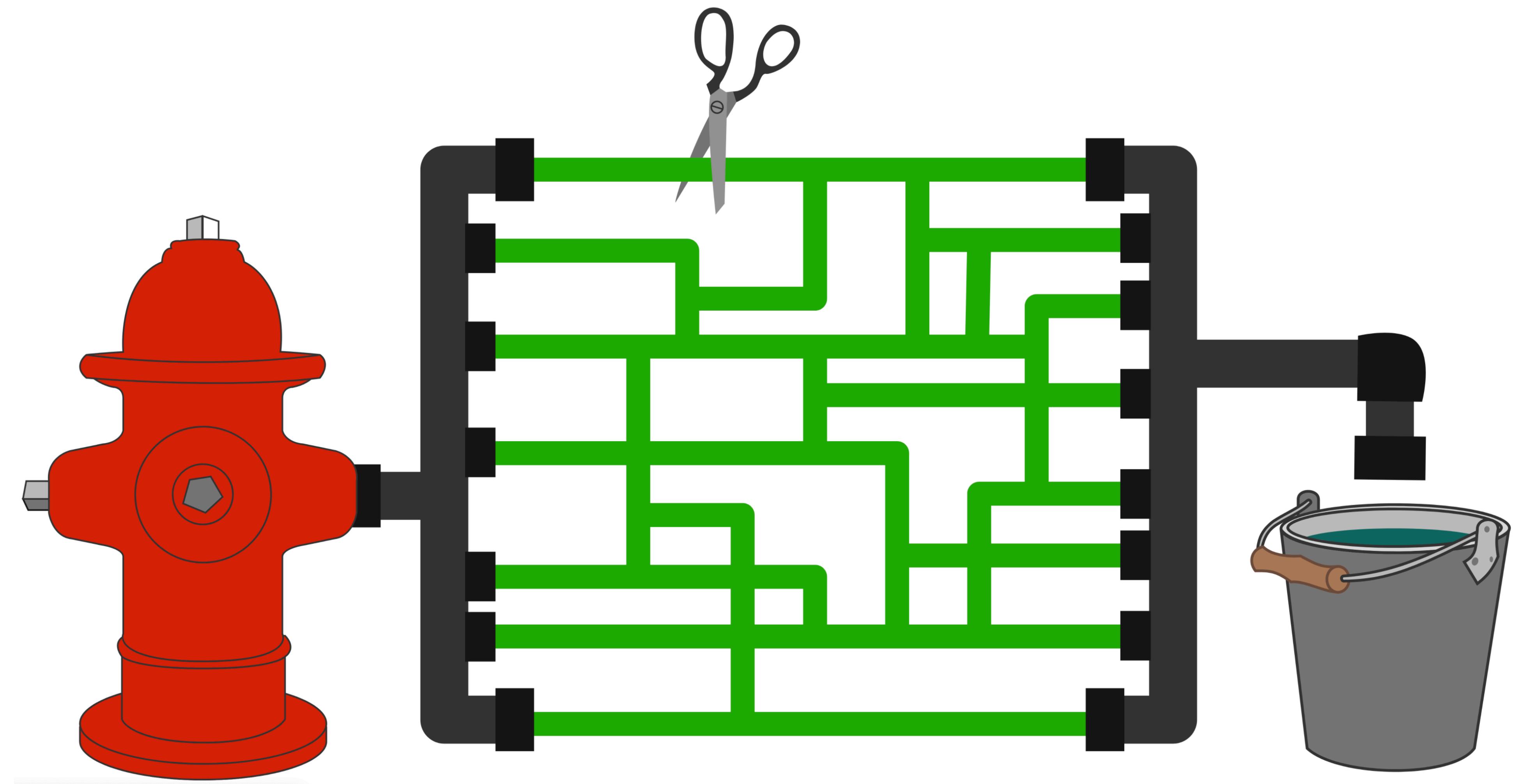
- L'algorisme de **Floyd-Warshall** proporciona les longituds dels camins mínims entre tots els parells de vèrtexs. Però com obtenim els camins??
- Amb petites modificacions és possible crear un mètode per a reconstruir el camí real entre dos vèrtexs.
- Quina és la complexitat en termes de temps?

```
let dist be a |V| × |V| array of minimum distances initialized to  $\infty$  (infinity)
let next be a |V| × |V| array of vertex indices initialized to null
```

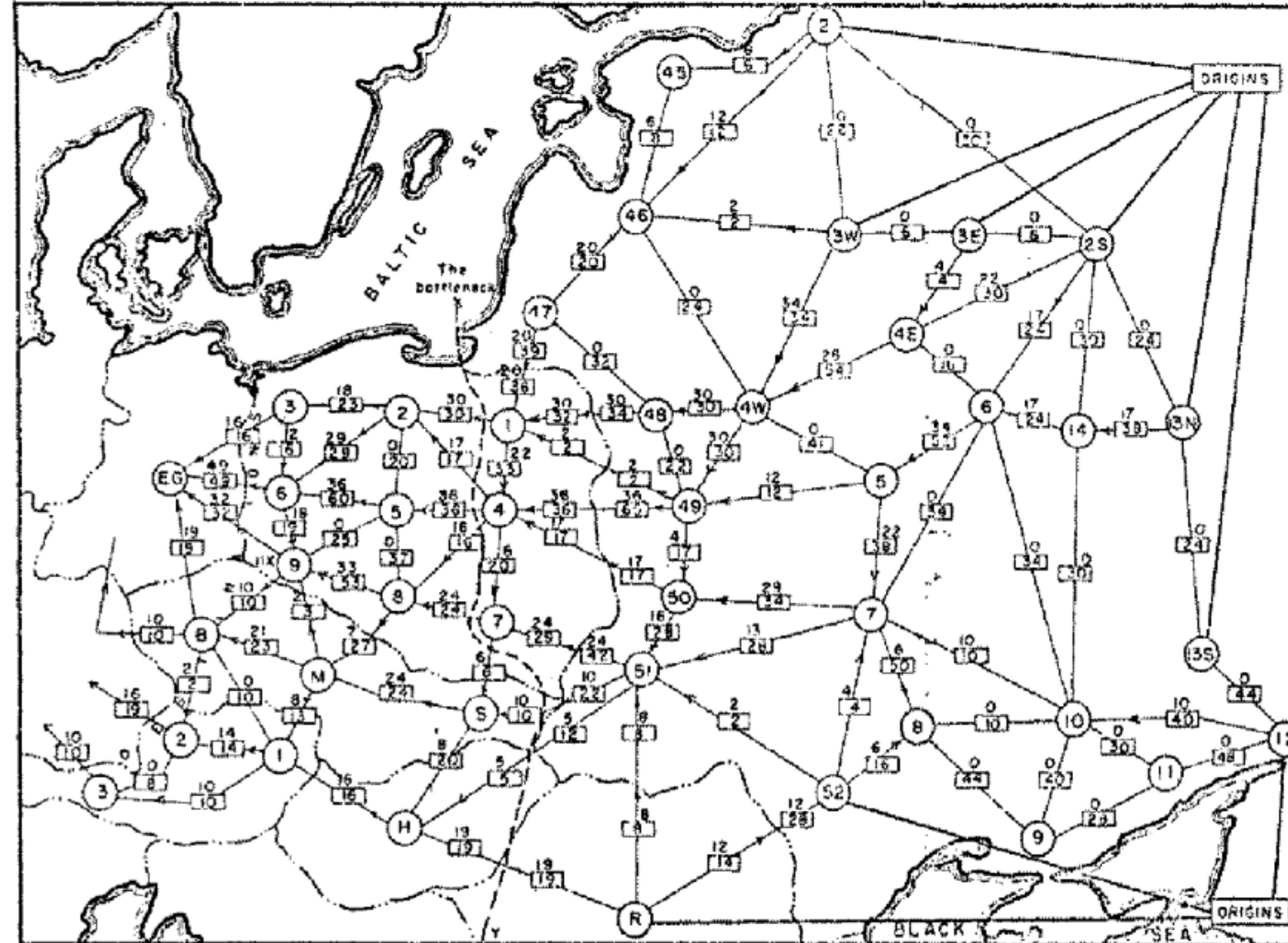
```
procedure FloydWarshallWithPathReconstruction() is
    for each edge (u, v) do
        dist[u][v] ← w(u, v) // The weight of the edge (u, v)
        next[u][v] ← v
    for each vertex v do
        dist[v][v] ← 0
        next[v][v] ← v
    for k from 1 to |V| do // standard Floyd-Warshall implementation
        for i from 1 to |V|
            for j from 1 to |V|
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    next[i][j] ← next[i][k]
```

```
procedure Path(u, v)
    if next[u][v] = null then
        return []
    path = [u]
    while u ≠ v
        u ← next[u][v]
        path.append(u)
    return path
```

# Flux màxim



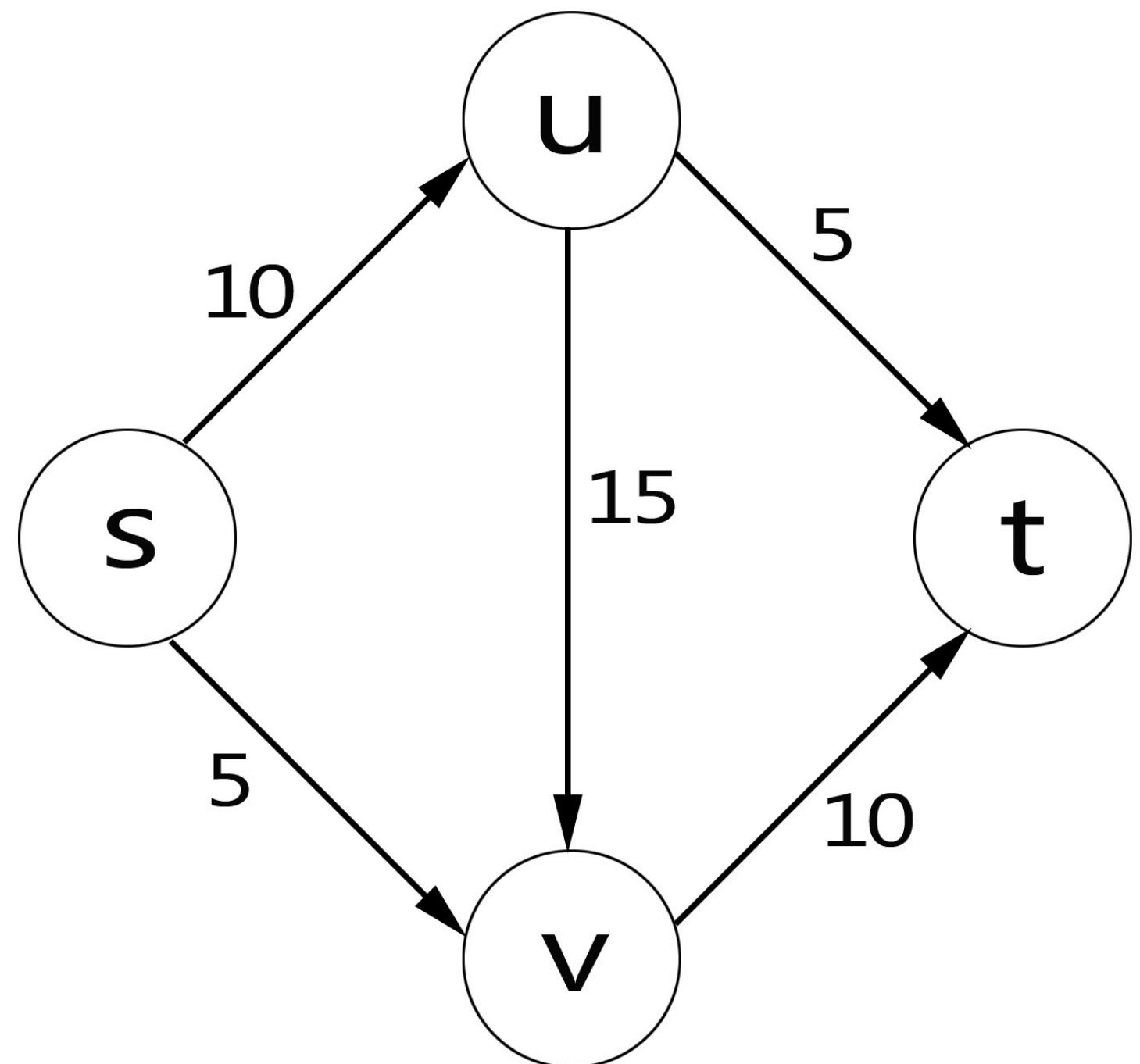
Based on a 1940 Soviet railway network, find the maximum amount of cargo that can be transported from sources in the Western Soviet Union to destinations in Eastern European countries.



Source: *On the history of the transportation and maximum flow problems*.  
Alexander Schrijver in Math Programming, 91: 3, 2002.

# El flux màxim (max-flow)

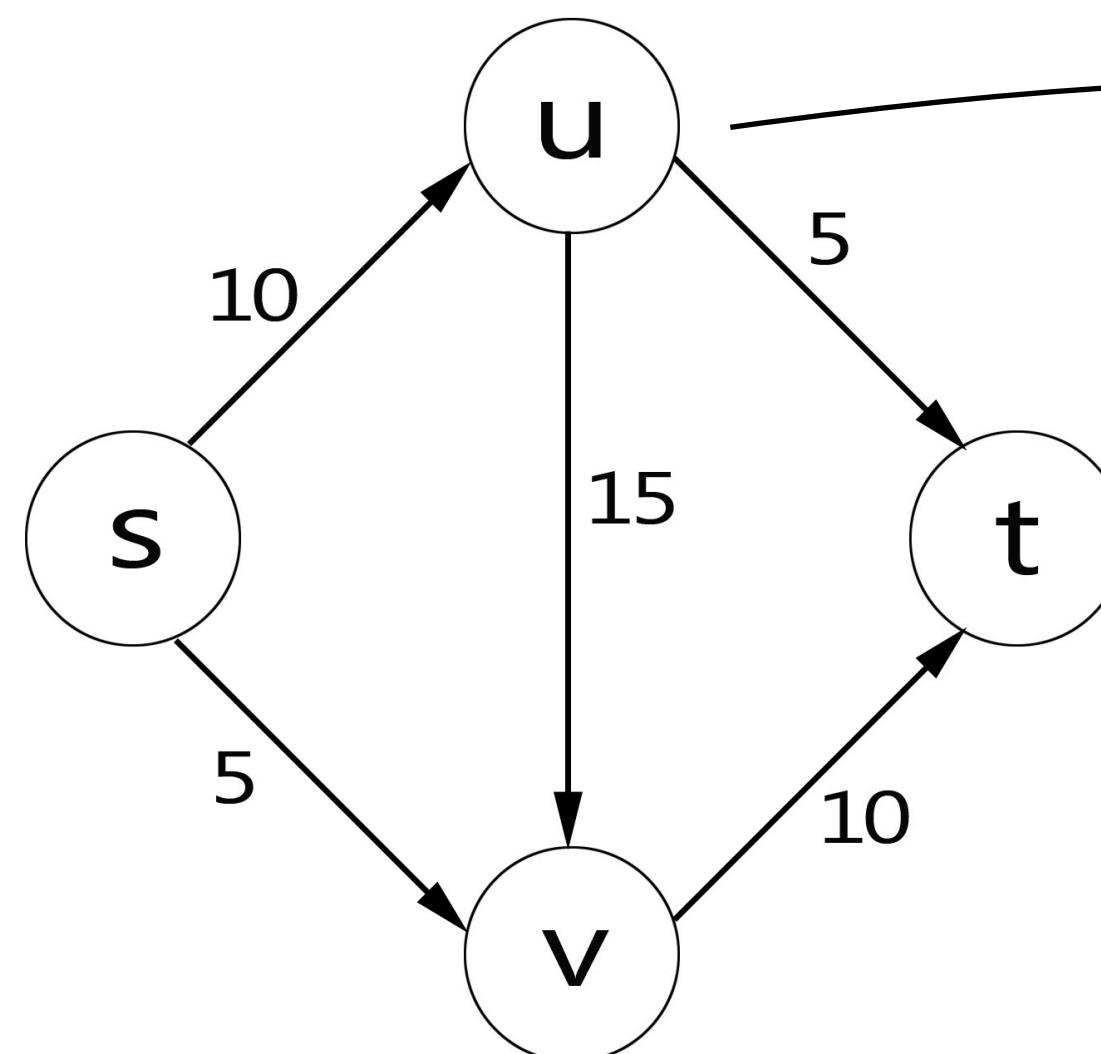
- En optimització i teoria de grafs, el problema de flux màxim serveix per trobar la quantitat màxima de flux que pot passar per una xarxa, des d'una sola **font (s)** fins a un sol **pou (t)**. El problema es pot definir com un **graf dirigit** on tenim un node inicial (**s**) i un node objectiu (**t**) amb l'objectiu de passar el **màxim flux** possible.



Una xarxa de flux, amb font **s** i un pou **t**. Els números al costat de les arestes són les capacitats.  
El flux màxim de l'exemple és 15.

# El flux màxim (max-flow)

- Cada aresta dirigida pot ser vista com un conducte per on passa el material, segons les següents restriccions:
  - Cada un dels conductes té una capacitat màxima finita ( $>= 0$ ).
  - És compleix la conservació del flux.  $\sum f_{input} = \sum f_{output}$  (per a cada node).

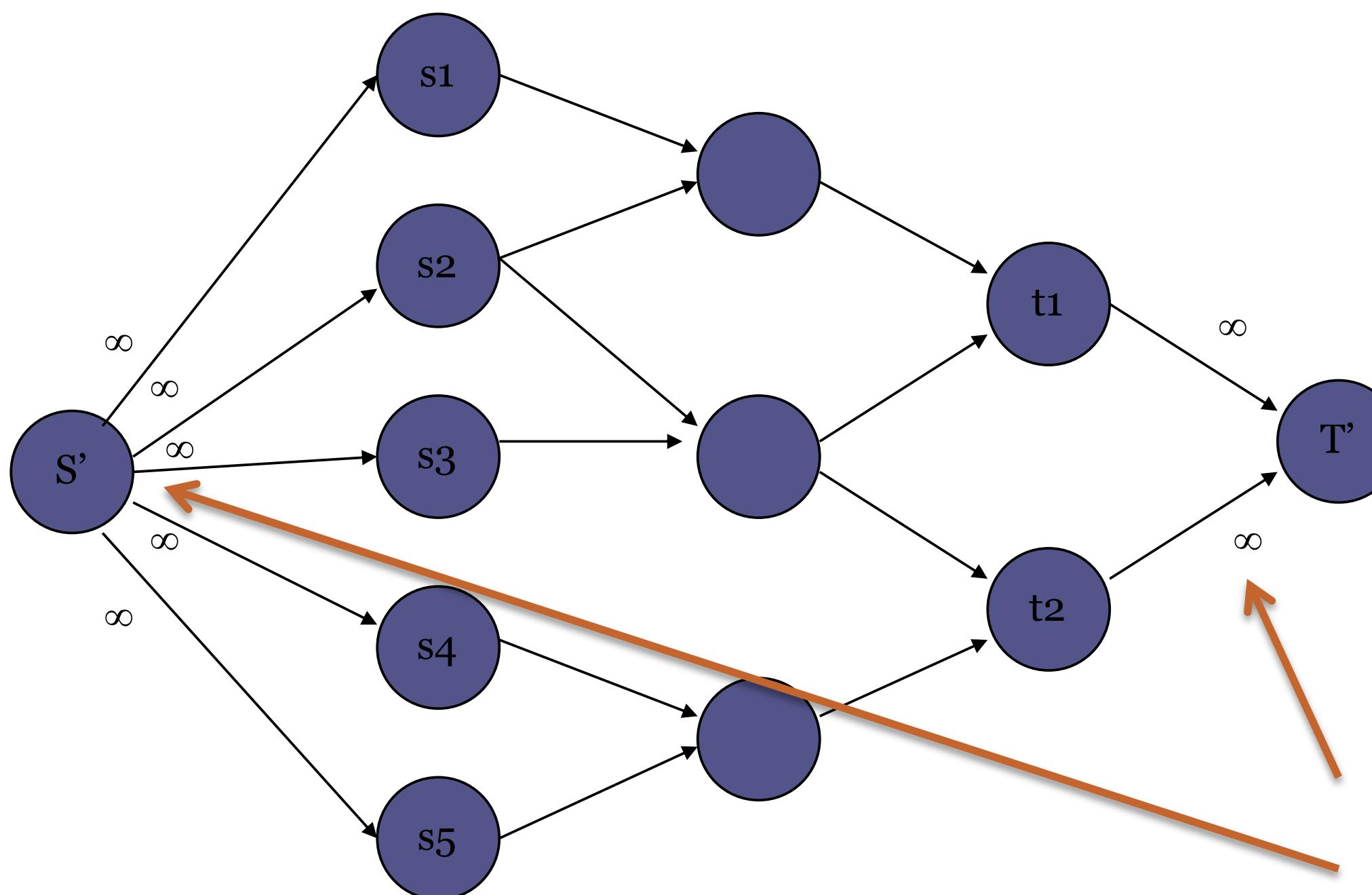


El màxim flux que pot sortir del node **u** és **10**. La qüestió es saber en quina direcció!

# El flux màxim (max-flow)

- Cada aresta dirigida pot ser vista com un conducte per on passa el material, segons les següents restriccions:
  - Cada un dels conductes té una capacitat màxima finita ( $>=0$ ).
  - És compleix la conservació del flux.  $\sum f_{input} = \sum f_{output}$  (per a cada node).
- **Problema del flux màxim?**
  - Quina és la millor taxa a la que podem portar el flux sense violar cap restricció?

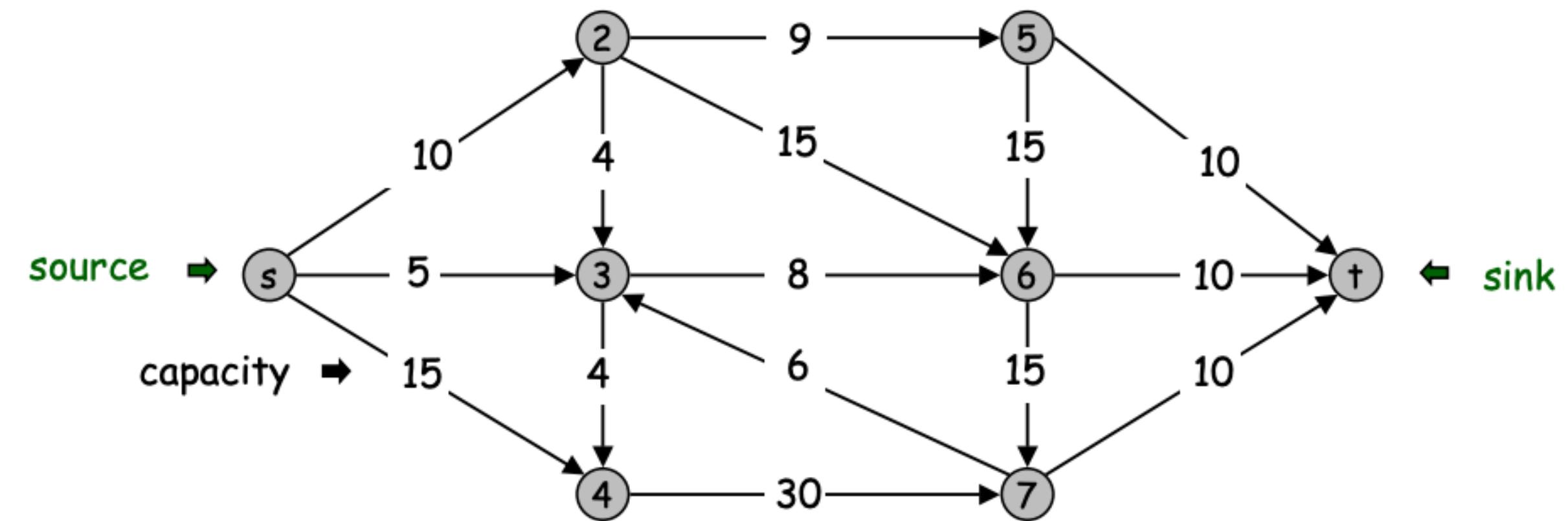
# El flux màxim (max-flow)



Quan podem enviar?

# Xarxa Max-Flow

- Xarxa Max-Flow:  $G = (V, E, s, t, u)$
- $(V, E) =$  graf dirigit sense arcs
- Node inicial  $s$ , i node destí  $t$
- $u(e) =$  capacitat de l'aresta  $e$

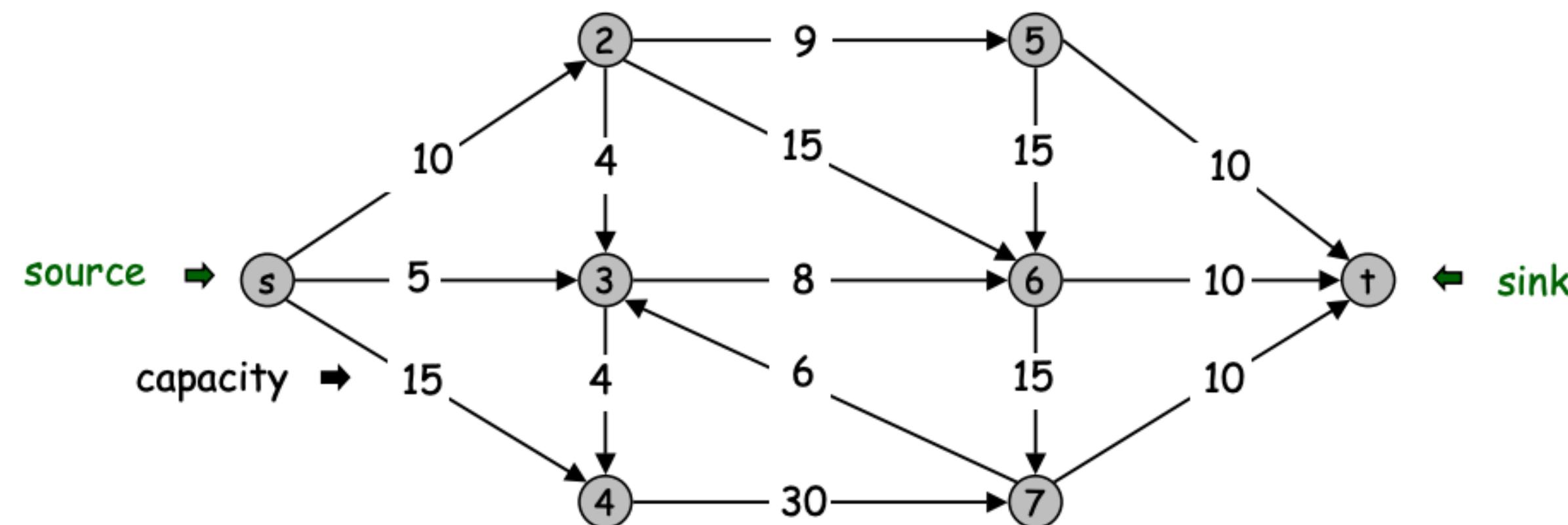


# El flux màxim (max-flow)

- El mètode iteratiu depèn de tres idees importants:
  - La xarxa residual.
  - Augment de camins.
  - Talls.
- Per tal de resoldre el problema utilitzarem el teorema:
  - **max-flow/min-cut** que caracteritza el flux màxim en termes de talls de la xarxa.

# Teorema max-flow min-cut

- El **teorema de flux màxim tall mínim** postula que en una **xarxa de flux**, la quantitat **màxima de flux** que pot passar d'una font fins a un pou és igual a la **capacitat mínima** que necessitem treure-li a la xarxa perquè no pugui passar més flux de la font al pou.



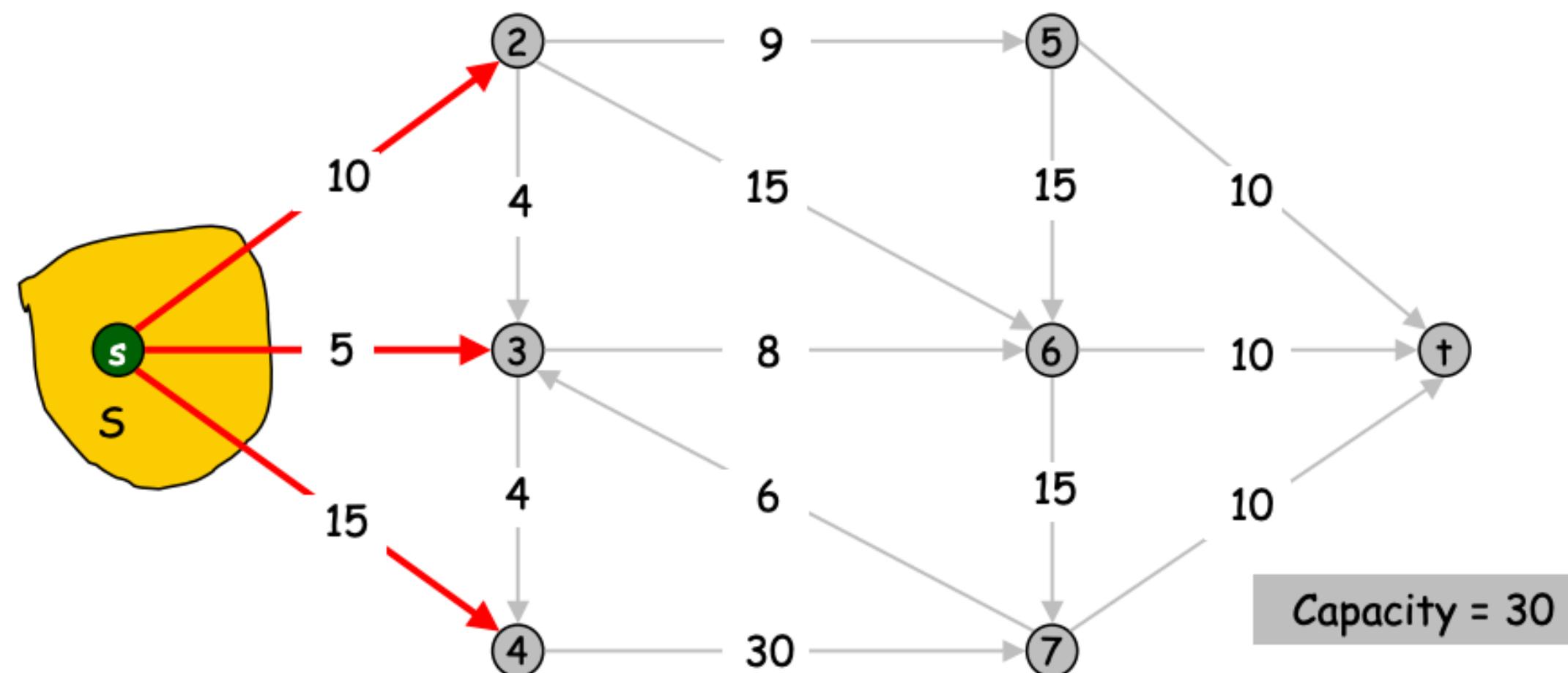
- Objectiu: Saturar la xarxa per satisfer el teorema !!!

# Problema Min-Cut

Un tall és una partició dels nodes ( $S, T$ ), tal que  $s$  està dins  $S$  i  $t$  està dins  $T$ .

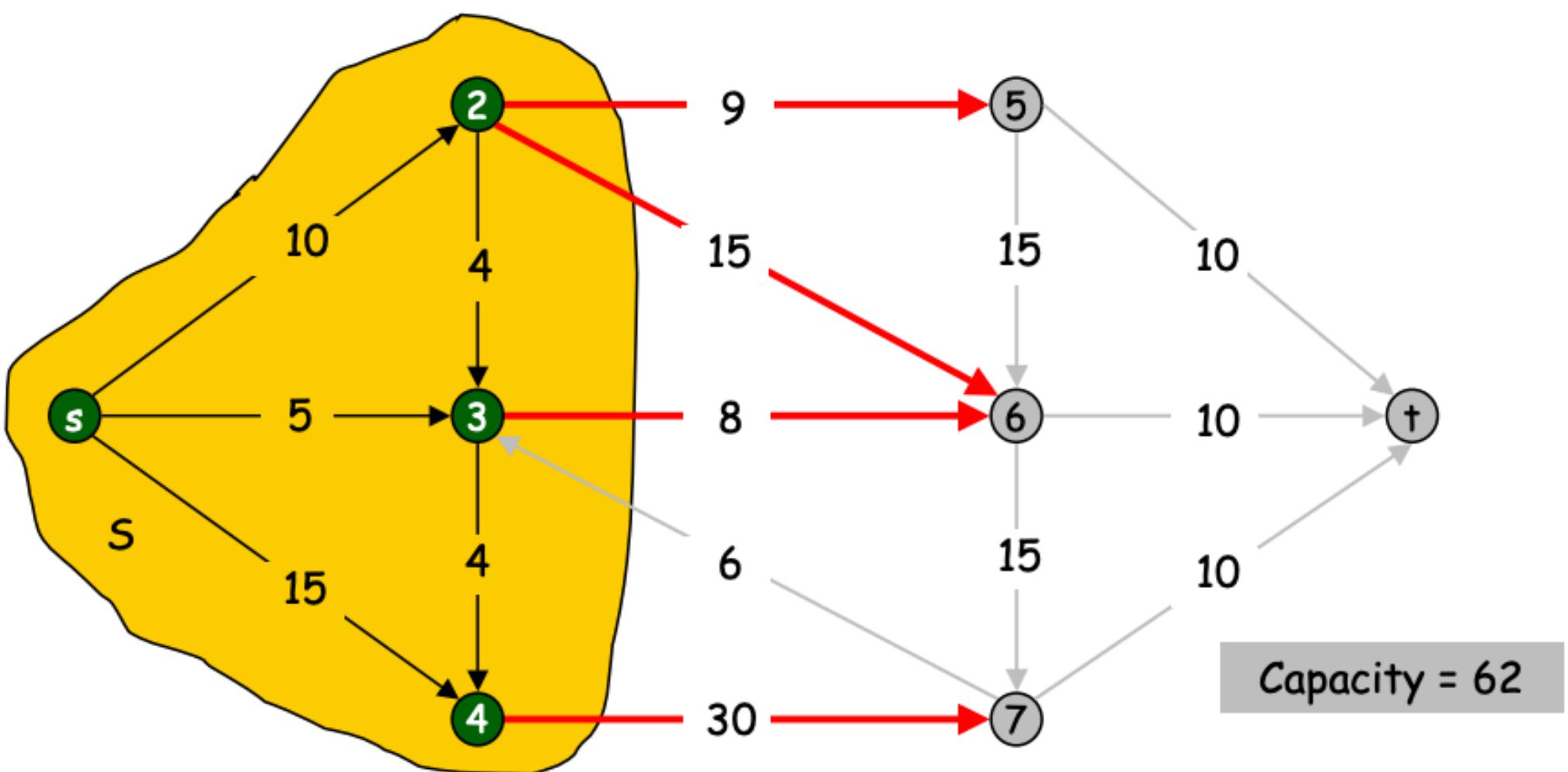
**capacitat( $S, T$ )** = Suma dels pesos que surten de  $S$

$$\sum_{e \text{ out of } S} u(e) := \sum_{\substack{(v,w) \in E \\ v \in S, w \in T}} u(v,w).$$



Un tall és una partició dels nodes ( $S, T$ ), tal que  $s$  està dins  $S$  i  $t$  està dins  $T$ .

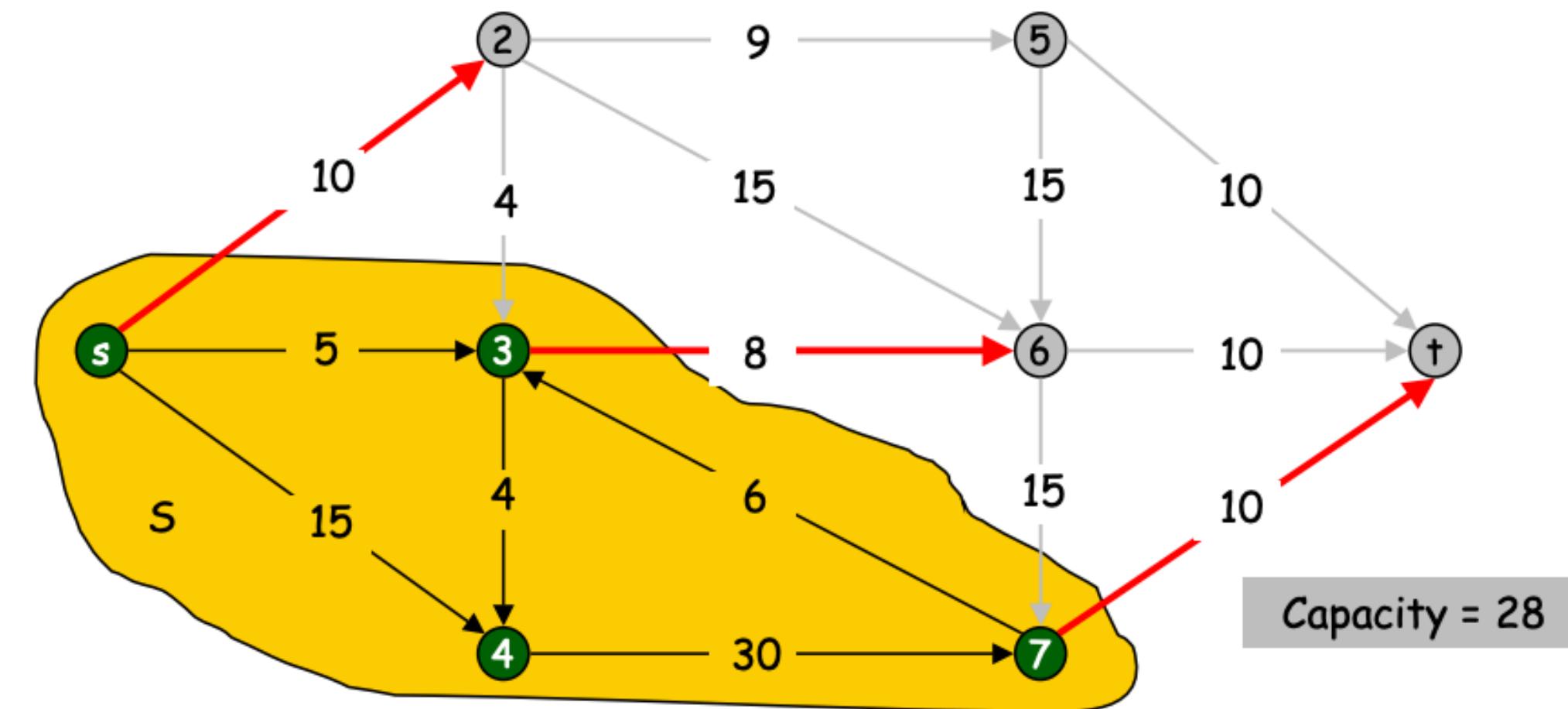
**capacitat( $S, T$ )** = Suma dels pesos que surten de  $S$



6

Un tall és una partició dels nodes ( $S, T$ ), tal que  $s$  està dins  $S$  i  $t$  està dins  $T$ .

**capacitat( $S, T$ )** = Suma dels pesos que surten de  $S$

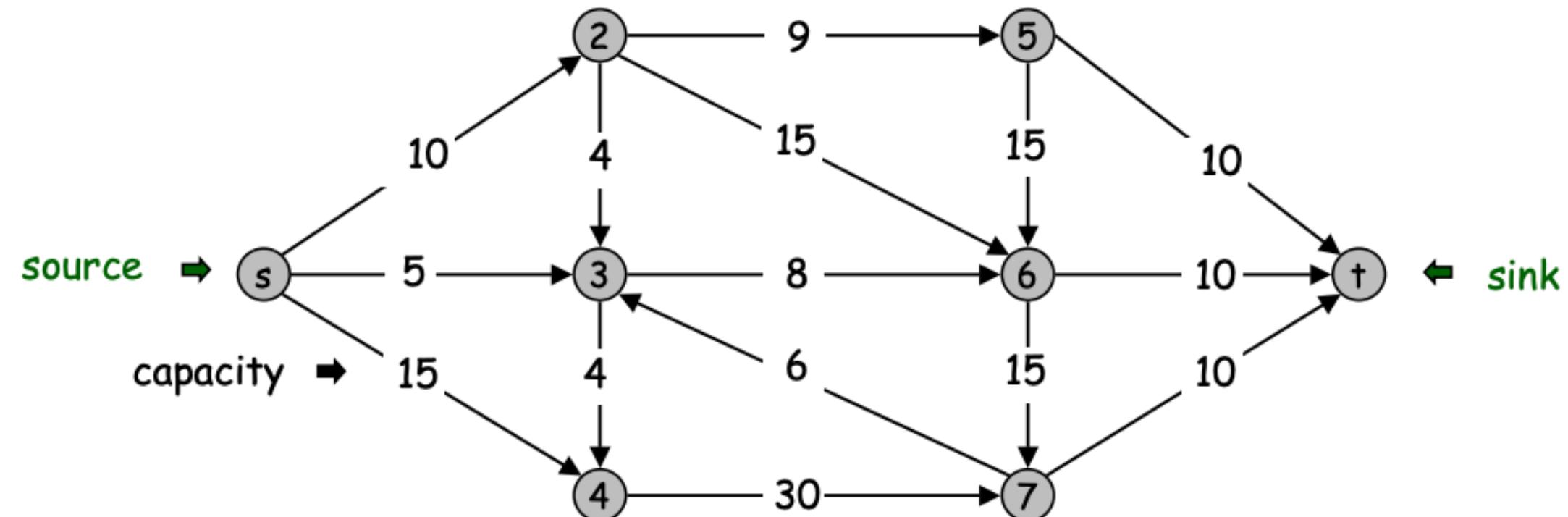


**Problema Min-cut.** Trobar el **tall s-t amb capacitat mínima**.

# Problema Max-Flow

Problema de **flux màxim**. Assigna el flux a les arestes de la següent forma:

- Igualem el flux d'entrada i de sortida de tots els nodes intermedis
- Maximitzar el flux enviat de s a t

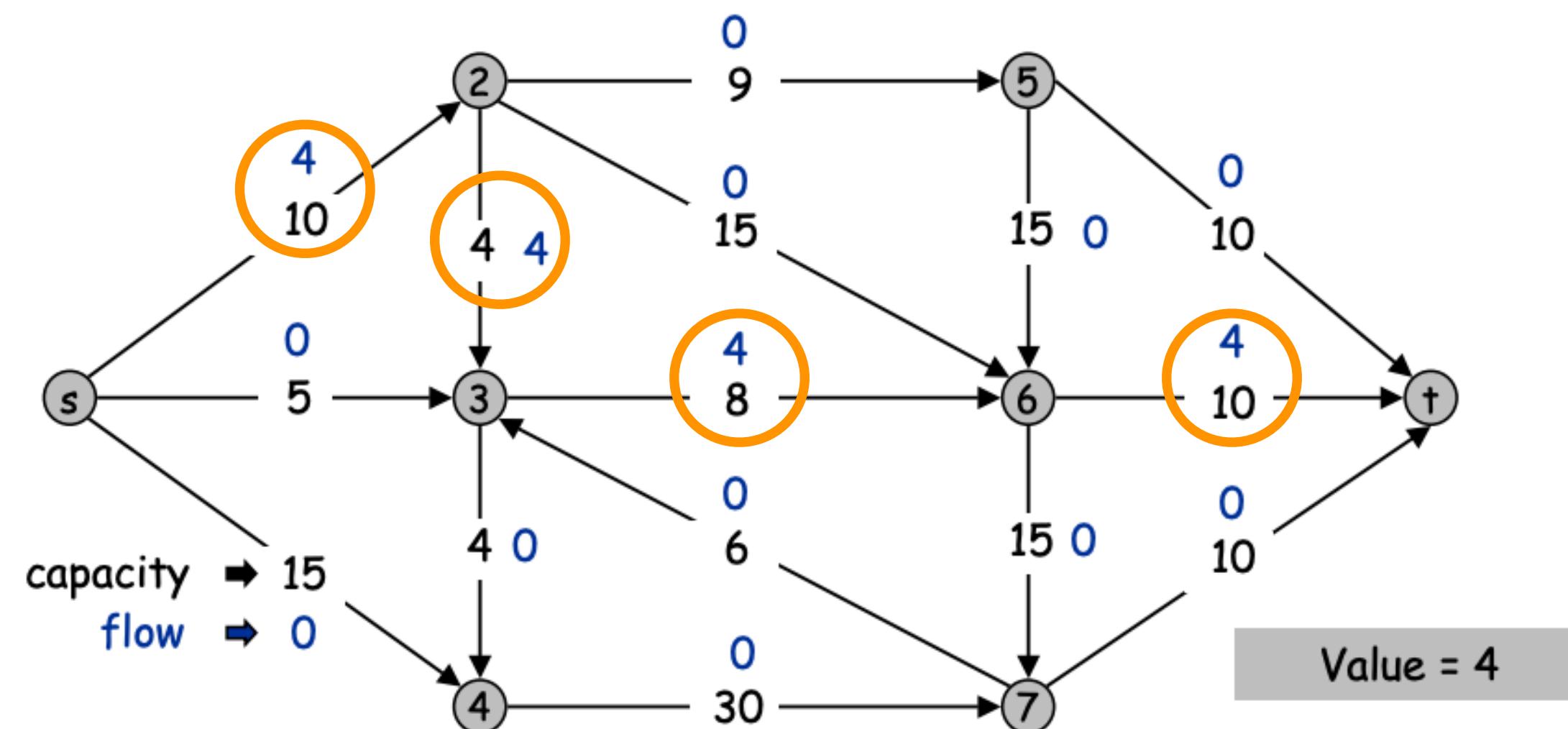


# Problema Max-Flow

Un flux  $\mathbf{f}$  és una assignació de pesos a les arestes de manera que:

- Capacitat:  $0 \leq f(e) \leq u(e)$
- Conservació del flux:
  - Flux d'entrada a  $v$  = Flux de sortida a  $v$

Excepte a  $s$  i  $t$



# Ford-Fulkerson Max Flow

## Algorithm Ford-Fulkerson

**Inputs** Given a Network  $G = (V, E)$  with flow capacity  $c$ , a source node  $s$ , and a sink node  $t$

**Output** Compute a flow  $f$  from  $s$  to  $t$  of maximum value

1.  $f(u, v) \leftarrow 0$  for all edges  $(u, v)$
2. While there is a path  $p$  from  $s$  to  $t$  in  $G_f$ , such that  $c_f(u, v) > 0$  for all edges  $(u, v) \in p$ :
  1. Find  $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
  2. For each edge  $(u, v) \in p$ 
    1.  $f(u, v) \leftarrow f(u, v) + c_f(p)$  (*Send flow along the path*)
    2.  $f(v, u) \leftarrow f(v, u) - c_f(p)$  (*The flow might be "returned" later*)

- " $\leftarrow$ " denotes **assignment**. For instance, "*largest*  $\leftarrow$  *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

# Xarxa residual

- La xarxa residual consisteix en arcs que admeten més flux. Donada una xarxa flux  $G = (V, E)$  amb inici  $\mathbf{s}$  i destinació  $\mathbf{t}$ . Sigui  $\mathbf{f}$  el flux en  $\mathbf{G}$ , i consideri un parell de vèrtexs  $u, v \in V$ , la quantitat de flux addicional que es pot abocar sobre  $u, v$  és la capacitat residual.

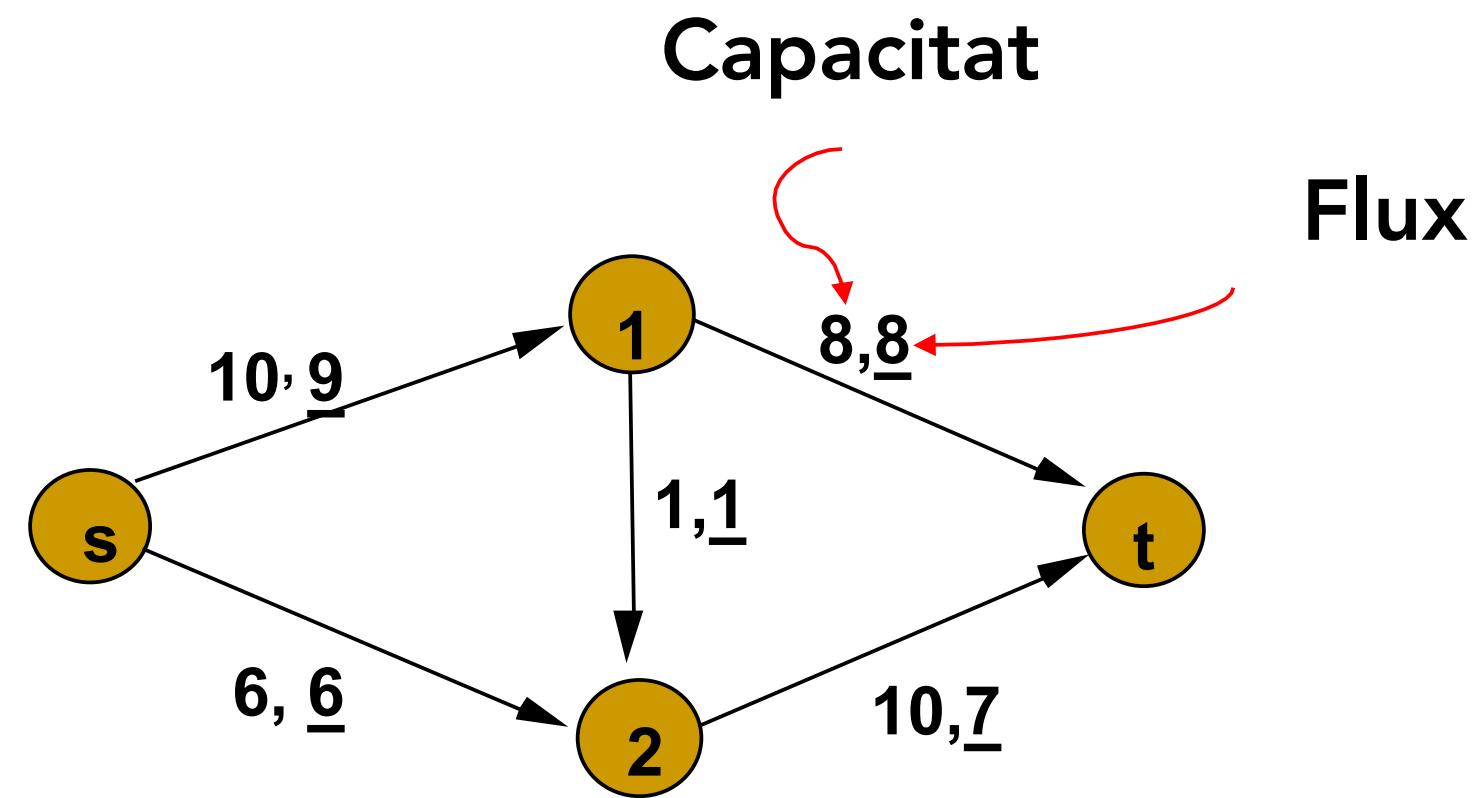
$$c_f(u,v) = c(u,v) - f(u,v)$$

Exemple:

$$c(u,v)=16, f(u,v)=10 \rightarrow c_f(u,v)=6$$

↓  
Capacitat residual connexió  
(u,v) en el pas 1

→  
Capacitat residual connexió  
(u,v) en el pas 2



Taula il·lustrant el flux i capacitat a través de diferents nodes

$$f_{s,1} = 9, c_{s,1} = 10 \text{ (Valid flow since } 10 > 9\text{)}$$

$$f_{s,2} = 6, c_{s,2} = 6 \text{ (Valid flow since } 6 \geq 6\text{)}$$

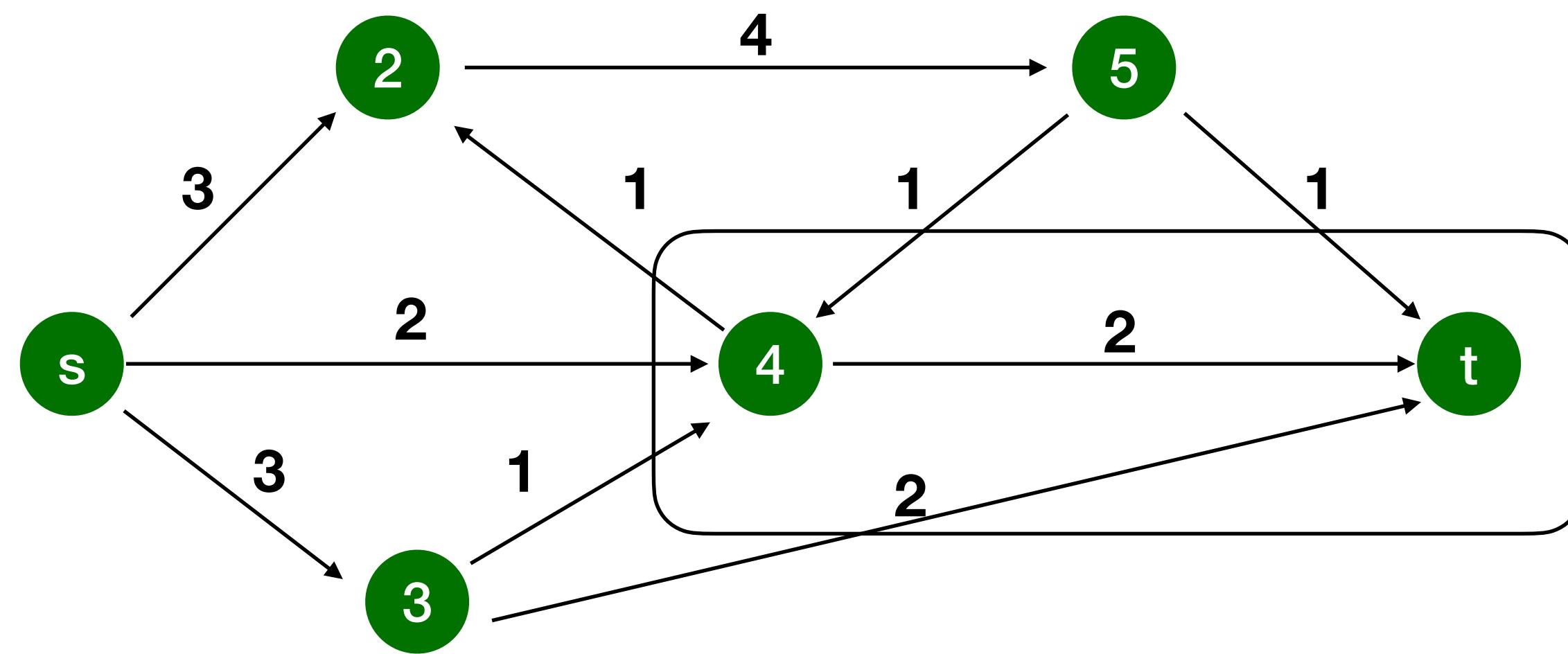
$$f_{1,2} = 1, c_{1,2} = 1 \text{ (Valid flow since } 1 \geq 1\text{)}$$

$$f_{1,t} = 8, c_{1,t} = 8 \text{ (Valid flow since } 8 \geq 8\text{)}$$

$$f_{2,t} = 7, c_{2,t} = 10 \text{ (Valid flow since } 10 > 7\text{)}$$

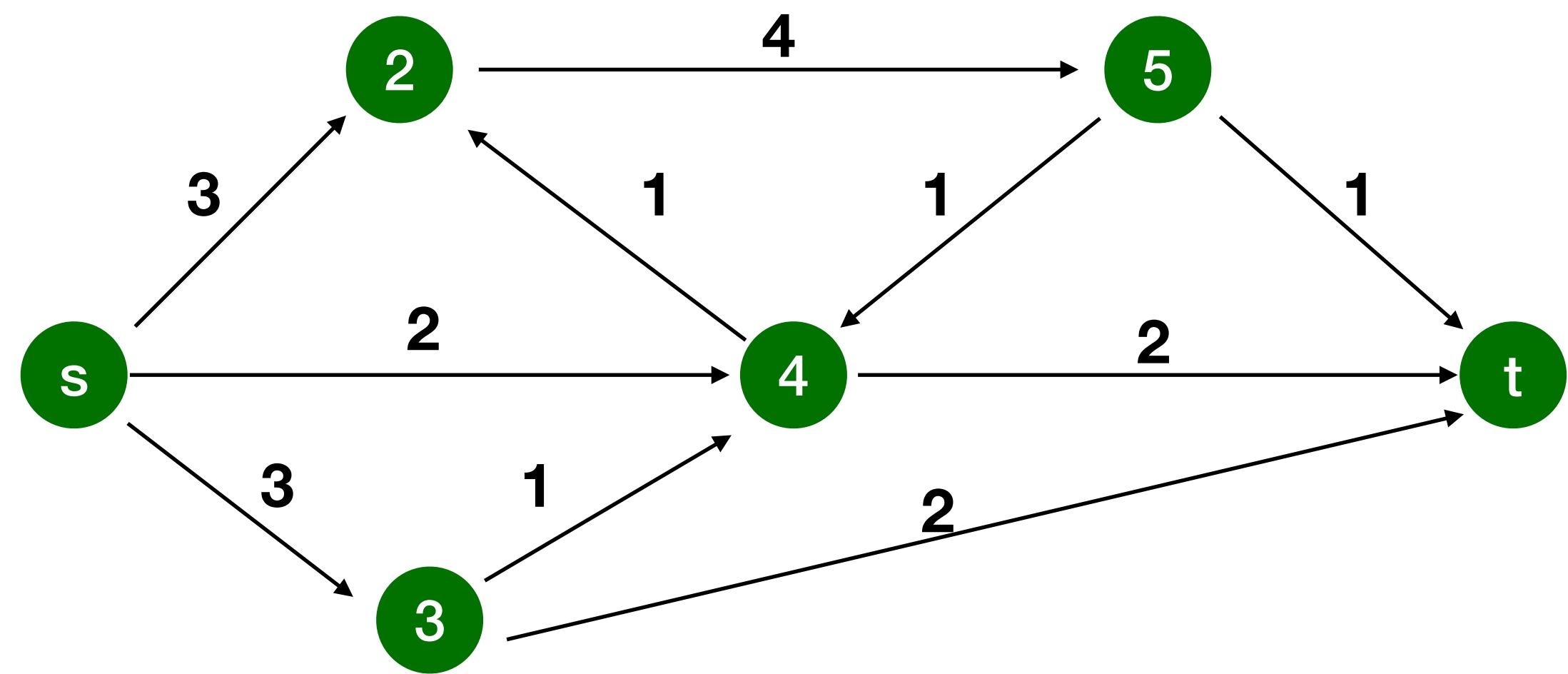
El flux a través dels nodes 1 i 2 també es conserva quan flueixen cap a ells = **flow out**.

# Ford-Fulkerson



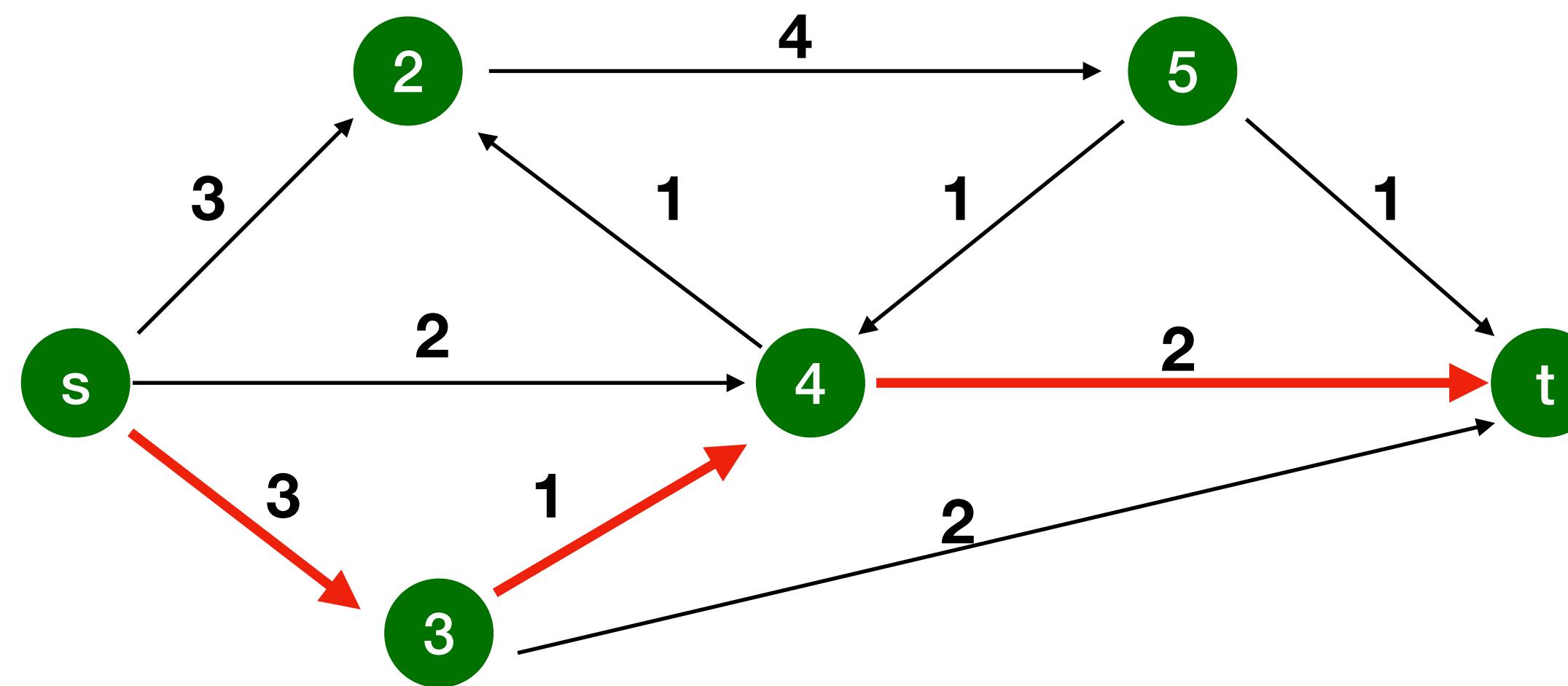
Graf original, i la xarxa residual original

# Ford-Fulkerson



Trobar qualsevol camí s-t al graf  $G(x)$

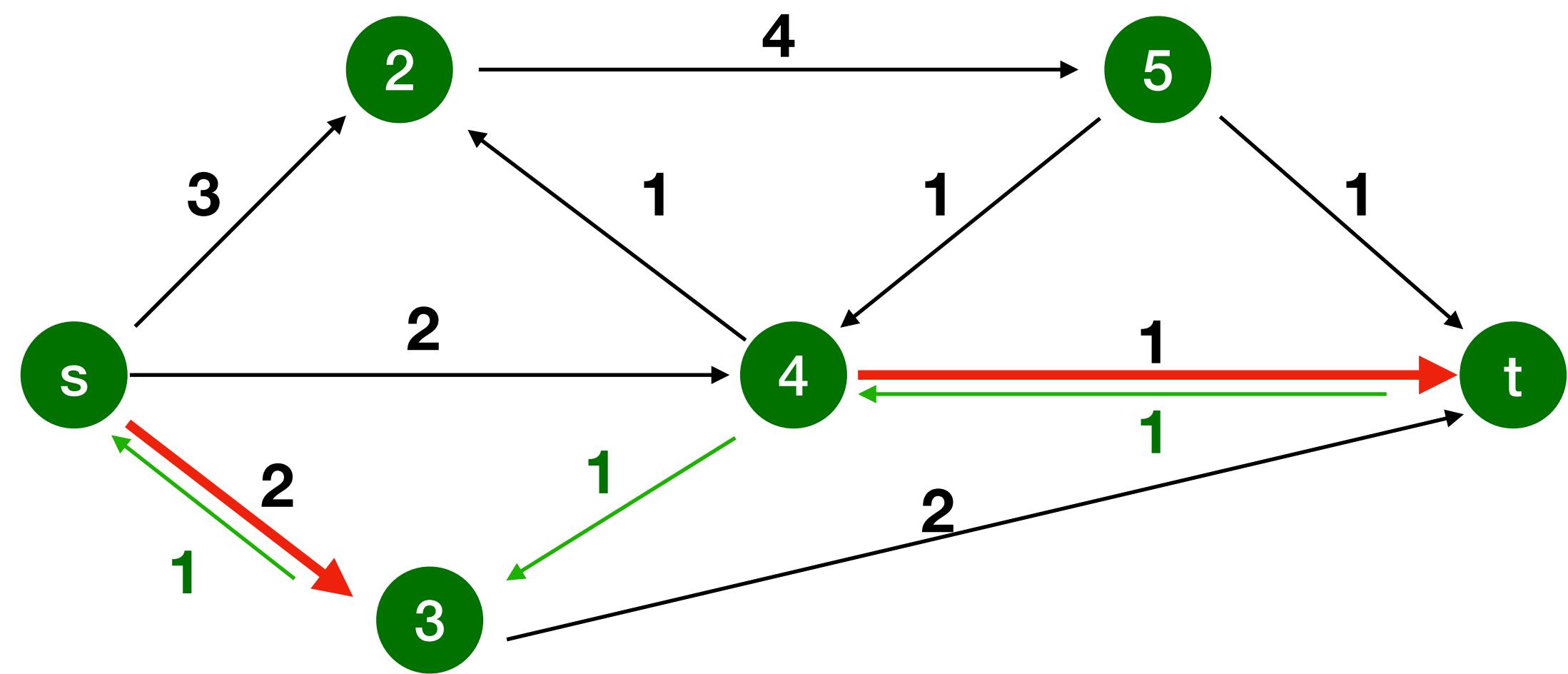
# Ford-Fulkerson



Determinar la flux màxim  $\Delta$  del camí (mínima capacitat de les arestes del camí)

Enviar  $\Delta$  unitats de flux al camí.

# Ford-Fulkerson

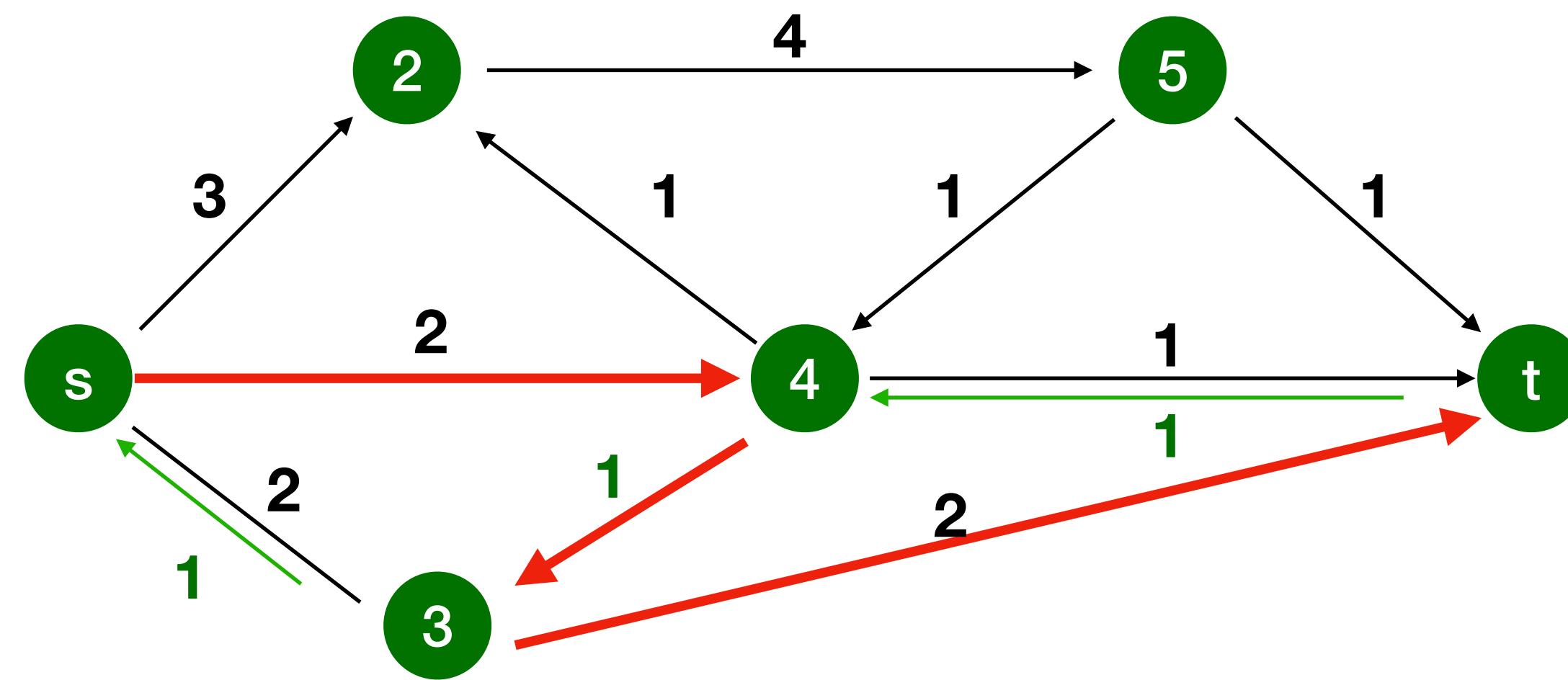


Determinar la capacitat  $\Delta$  del camí

Enviar  $\Delta$  unitats de flux al camí.

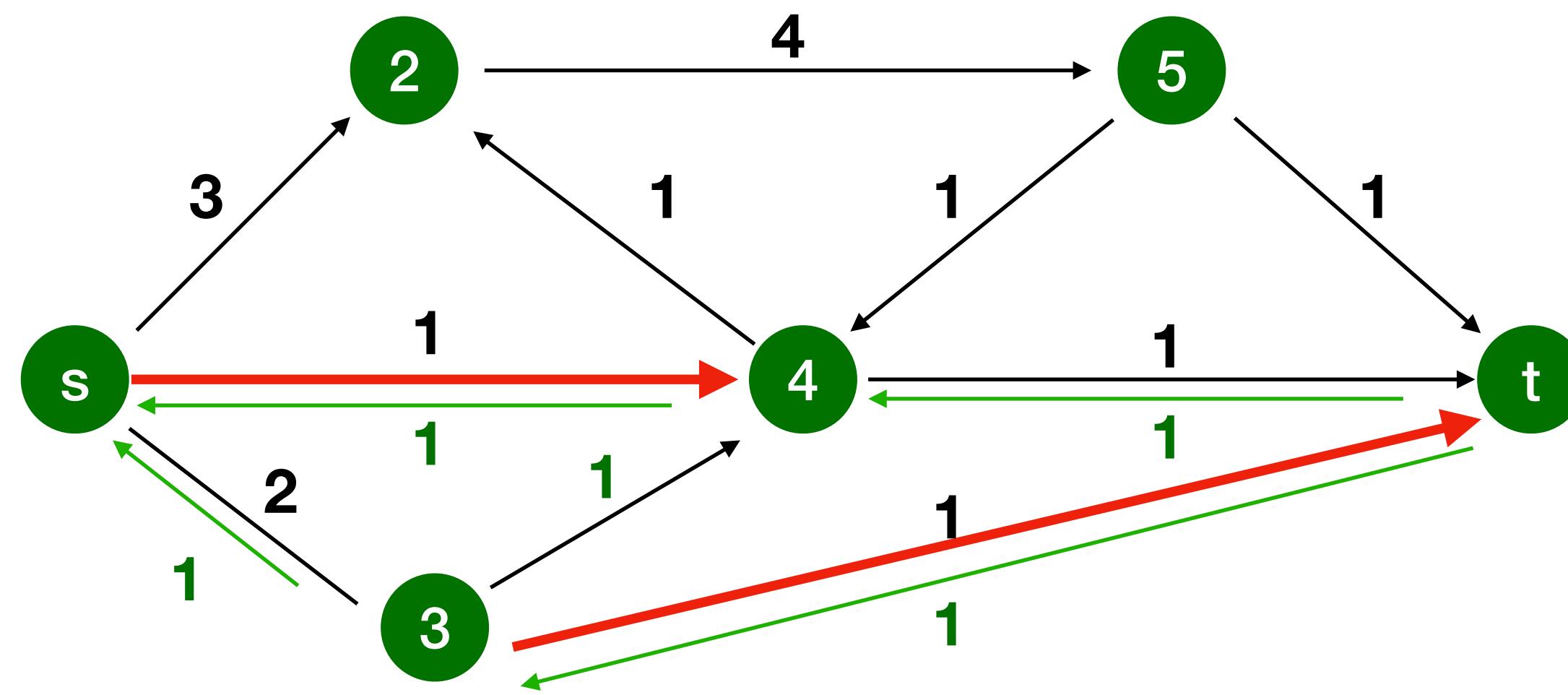
Actualitzar la capacitat residual.

# Ford-Fulkerson



Troba algun camí entre s i t

# Ford-Fulkerson



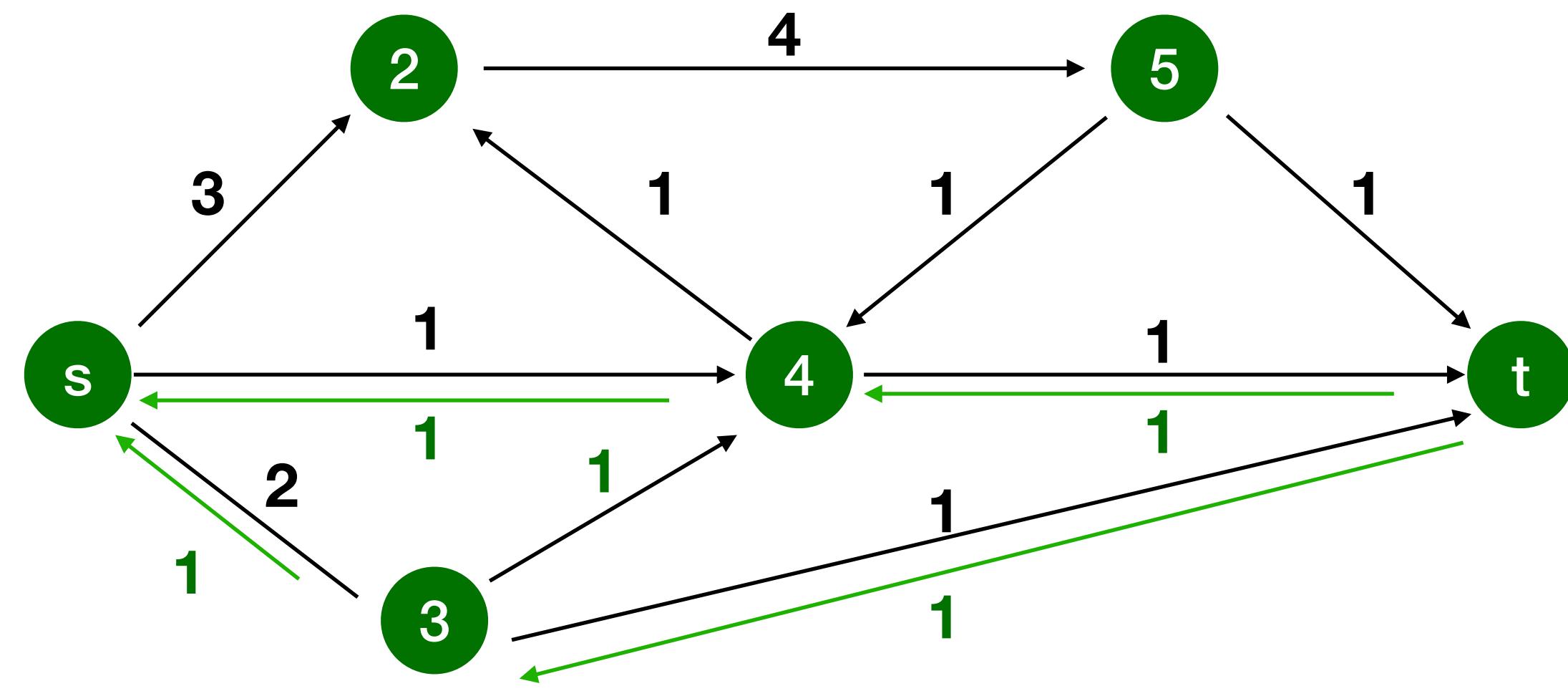
Troba algun camí entre s i t

Determinar la capacitat  $\Delta$  del camí

Enviar  $\Delta$  unitats de flux al camí.

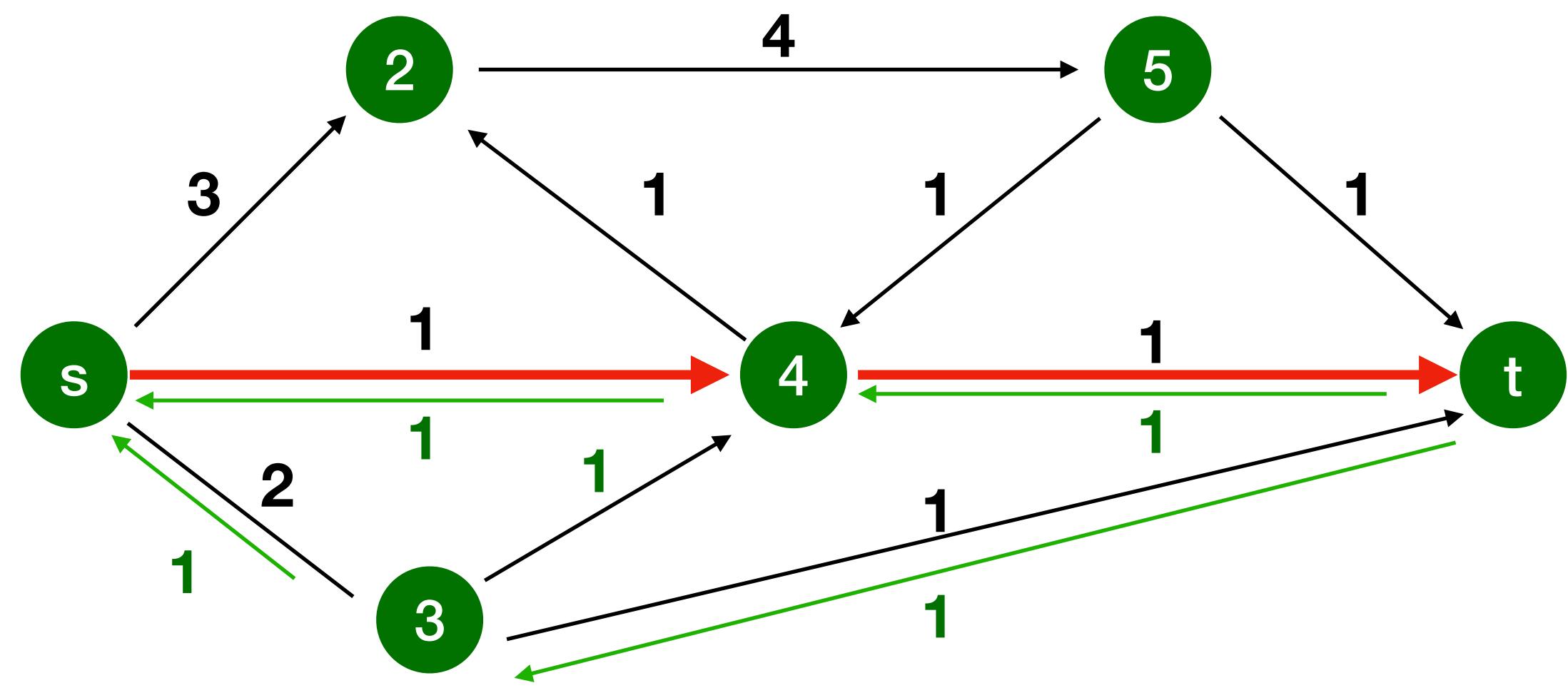
Actualitzar la capacitat residual.

# Ford-Fulkerson



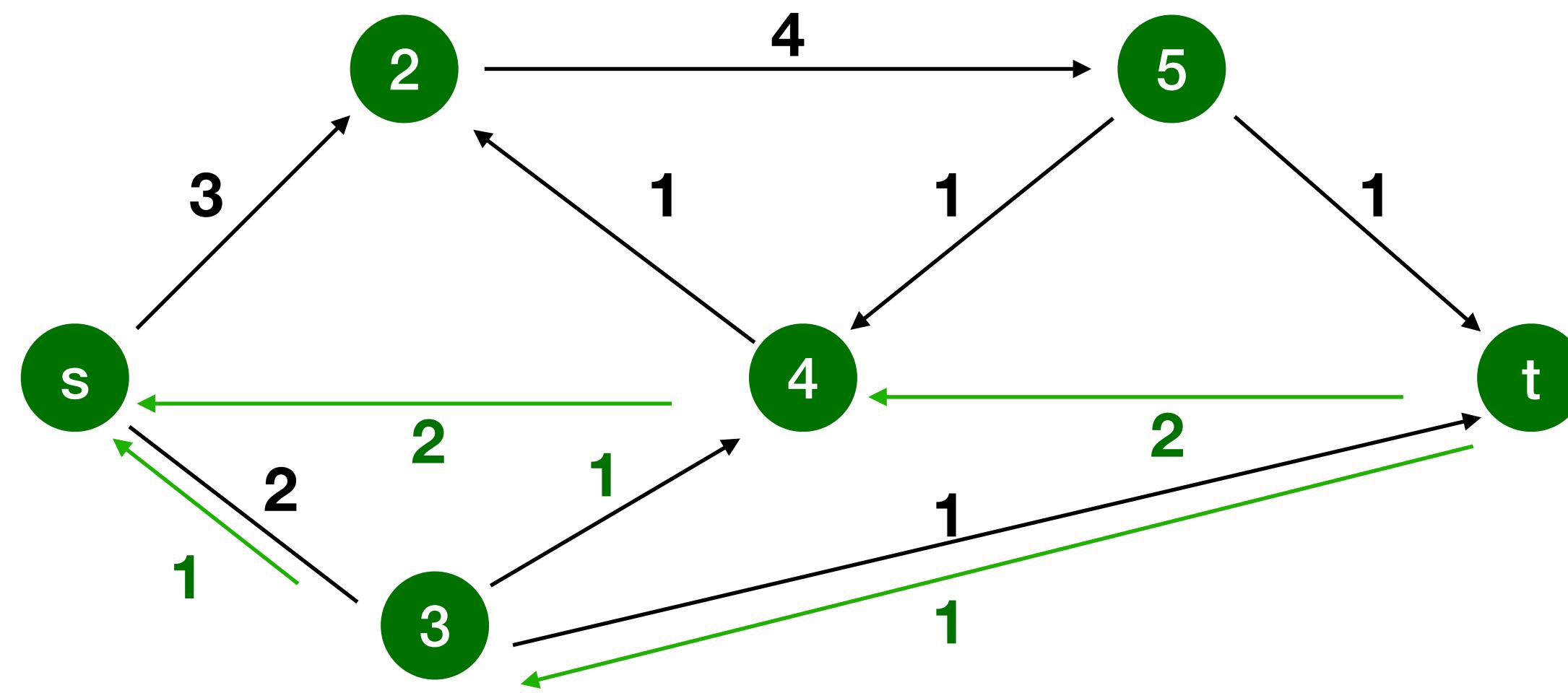
Troba algun camí entre s i t

# Ford-Fulkerson



Troba algun camí entre s i t

# Ford-Fulkerson



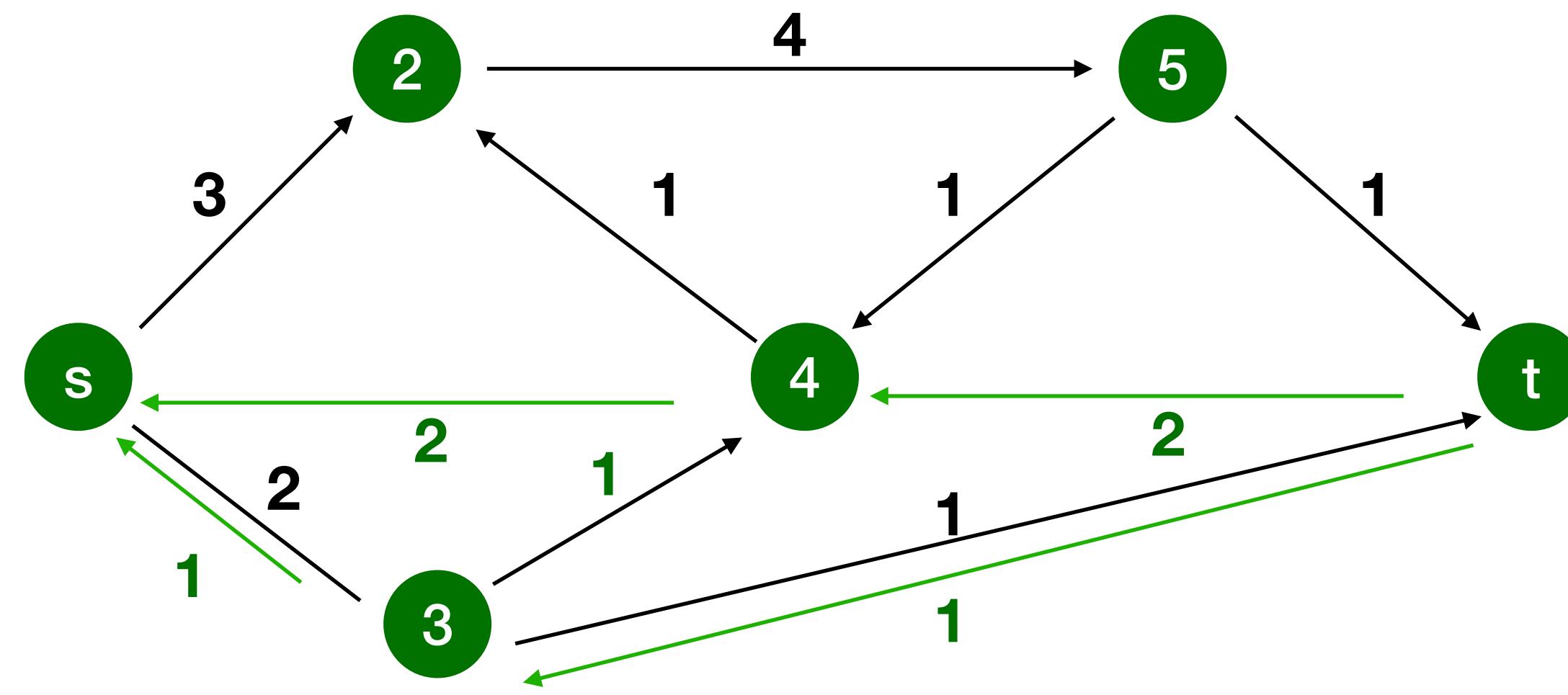
Troba algun camí entre s i t

Determinar la capacitat  $\Delta$  del camí

Enviar  $\Delta$  unitats de flux al camí.

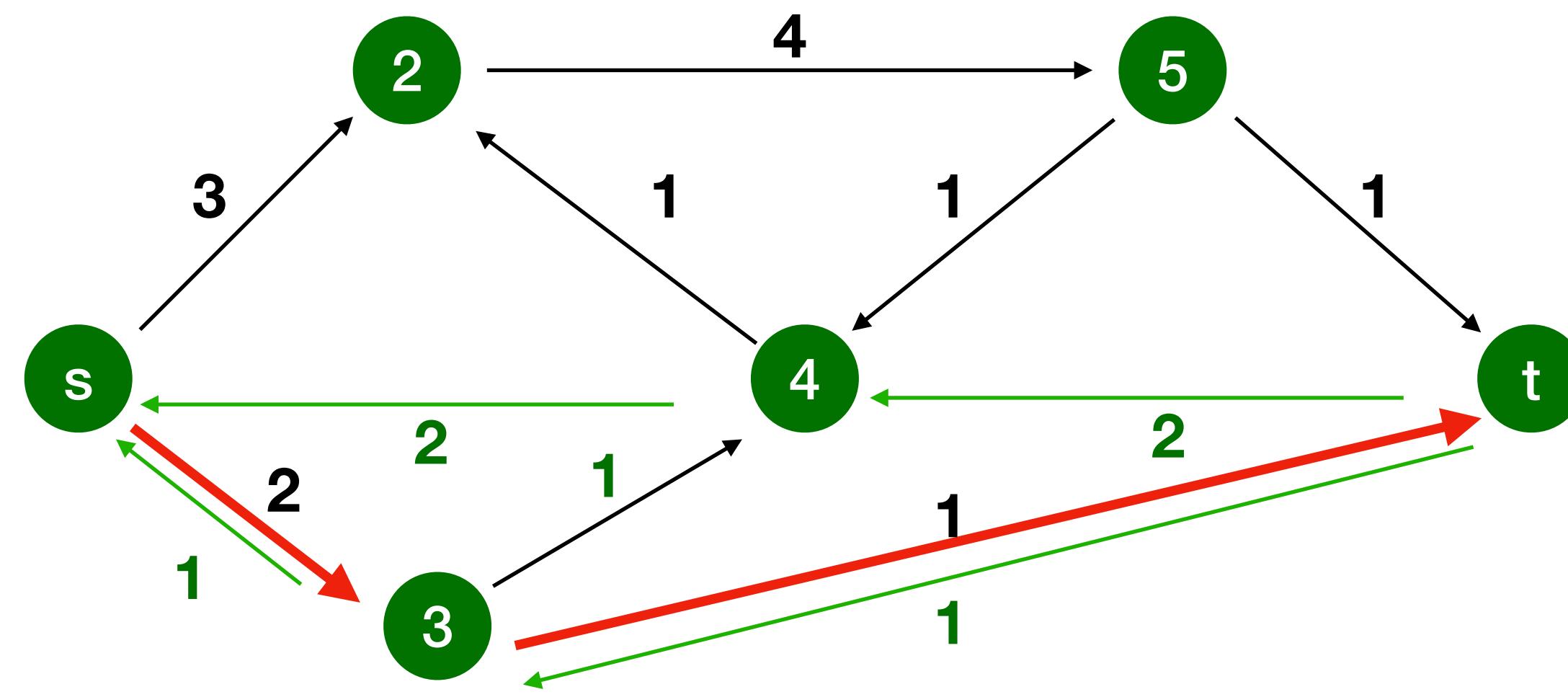
Actualitzar la capacitat residual.

# Ford-Fulkerson



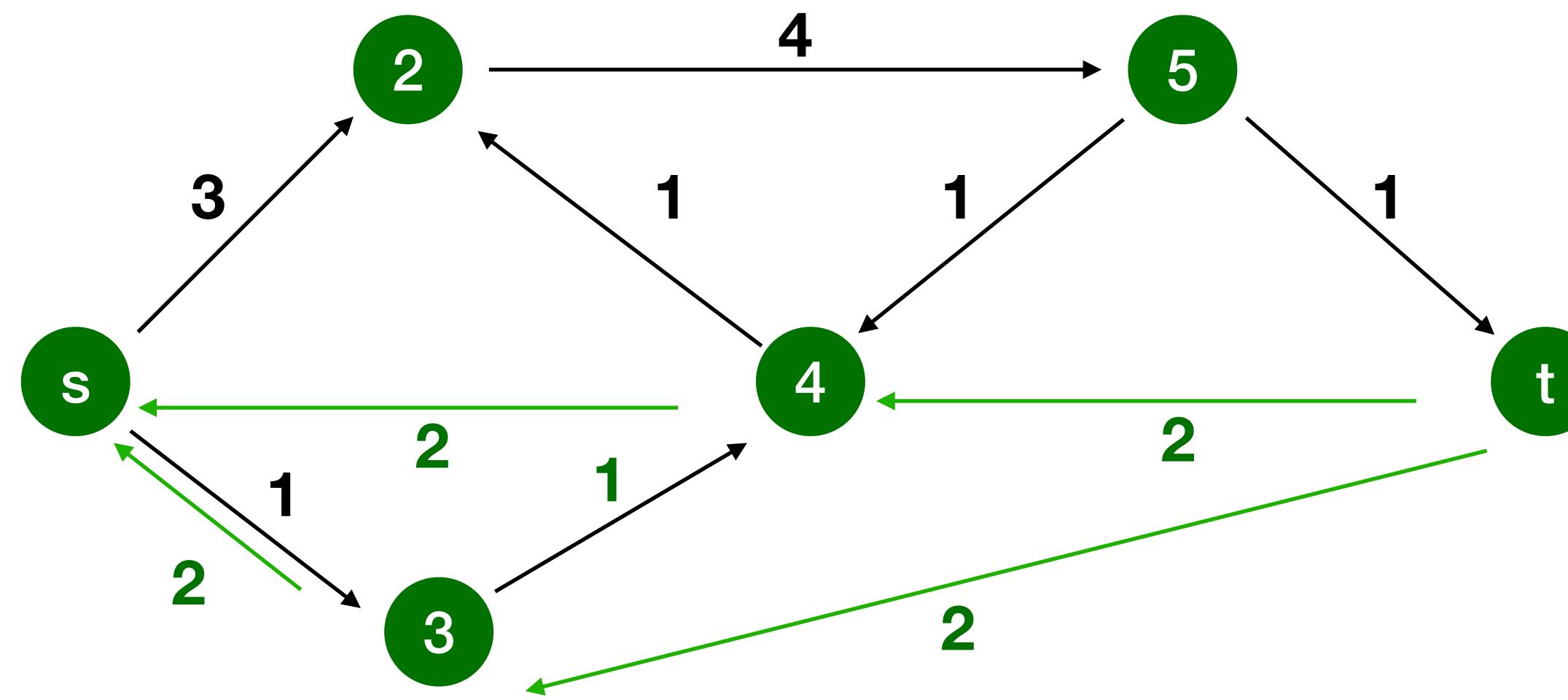
Troba algun camí entre s i t

# Ford-Fulkerson



Troba algun camí entre s i t

# Ford-Fulkerson



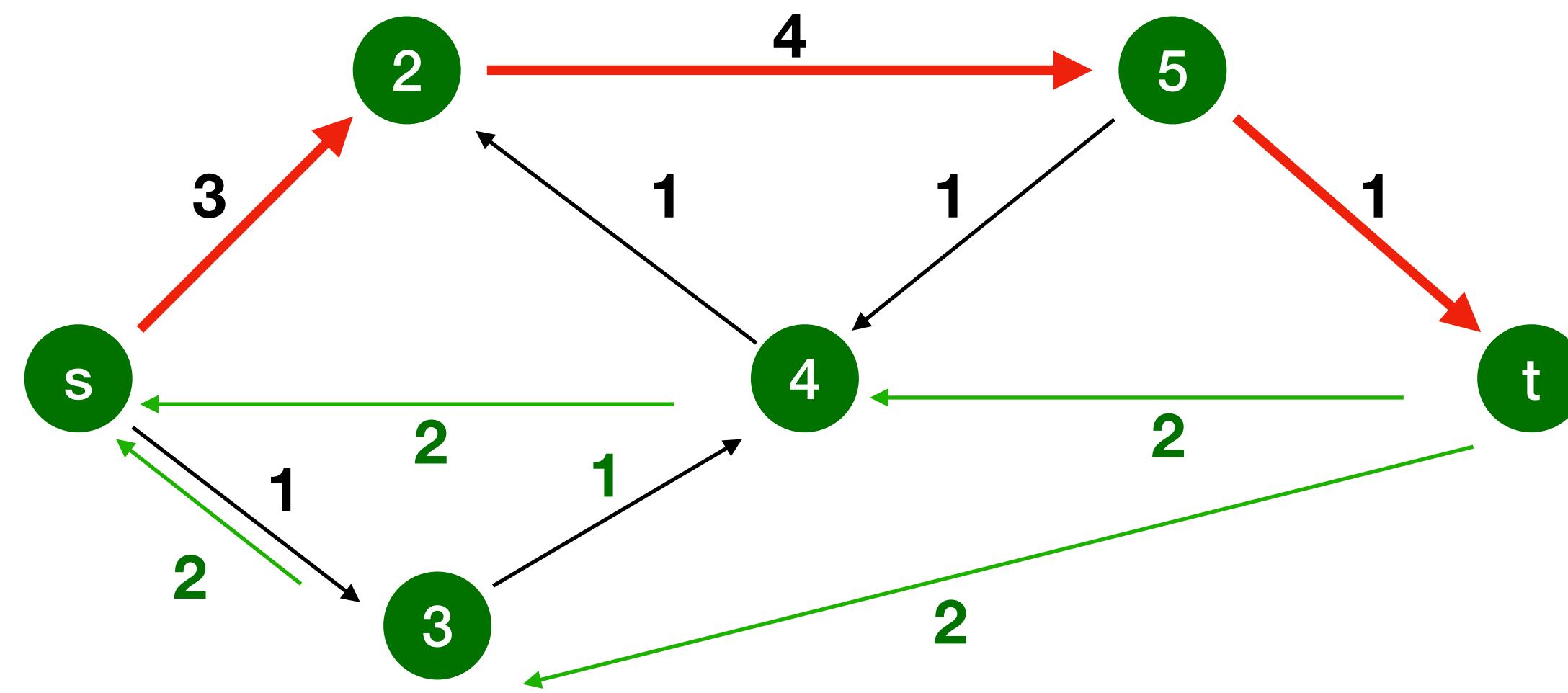
Troba algun camí entre s i t

Determinar la capacitat  $\Delta$  del camí

Enviar  $\Delta$  unitats de flux al camí.

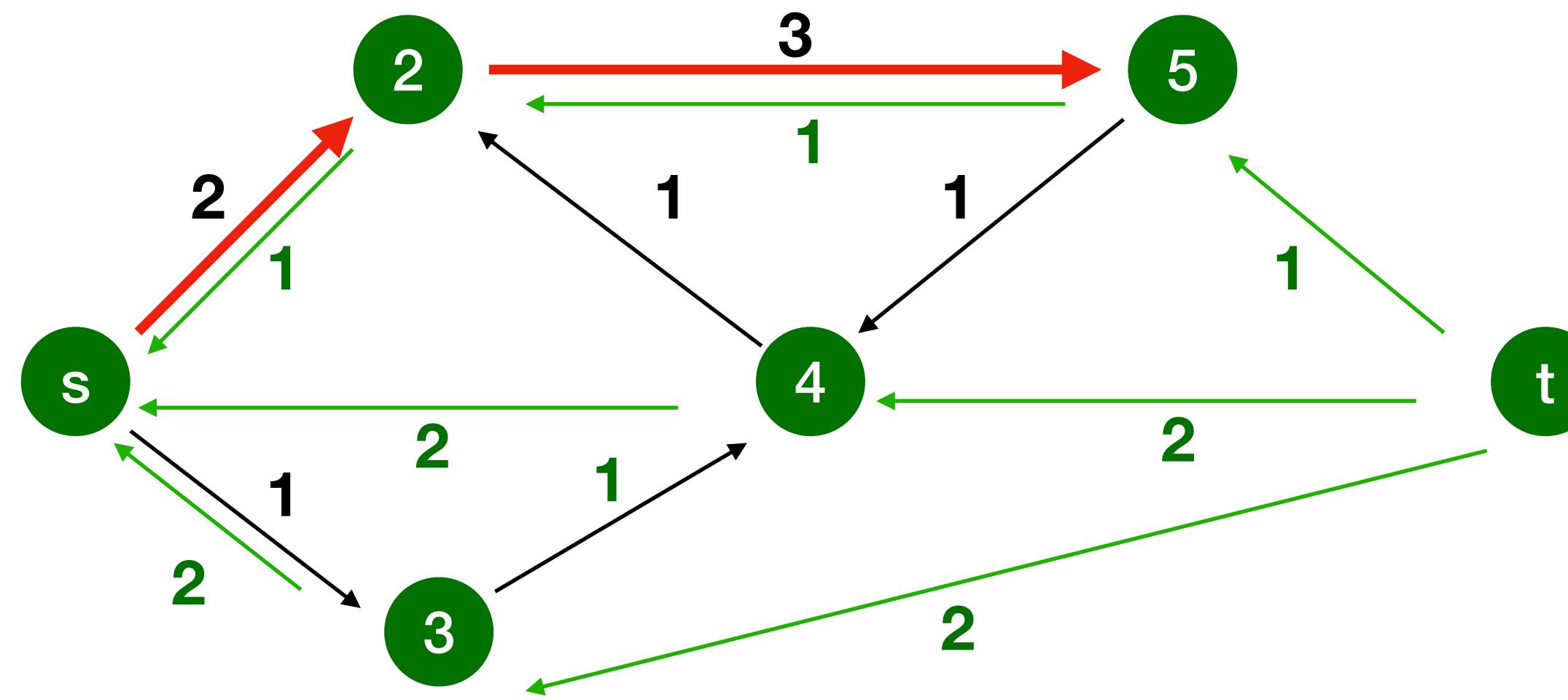
Actualitzar la capacitat residual.

# Ford-Fulkerson



Troba algun camí entre s i t

# Ford-Fulkerson



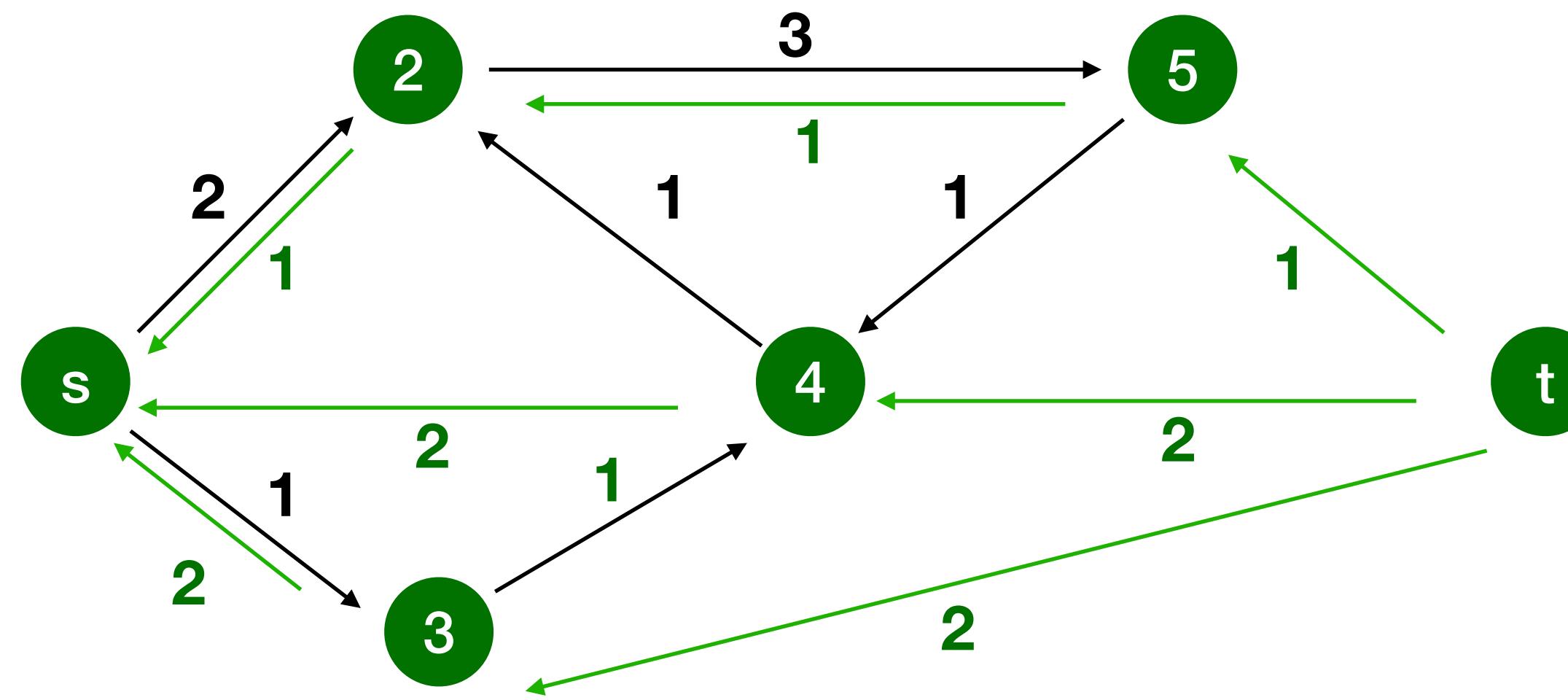
Troba algun camí entre s i t

Determinar la capacitat  $\Delta$  del camí

Enviar  $\Delta$  unitats de flux al camí.

Actualitzar la capacitat residual.

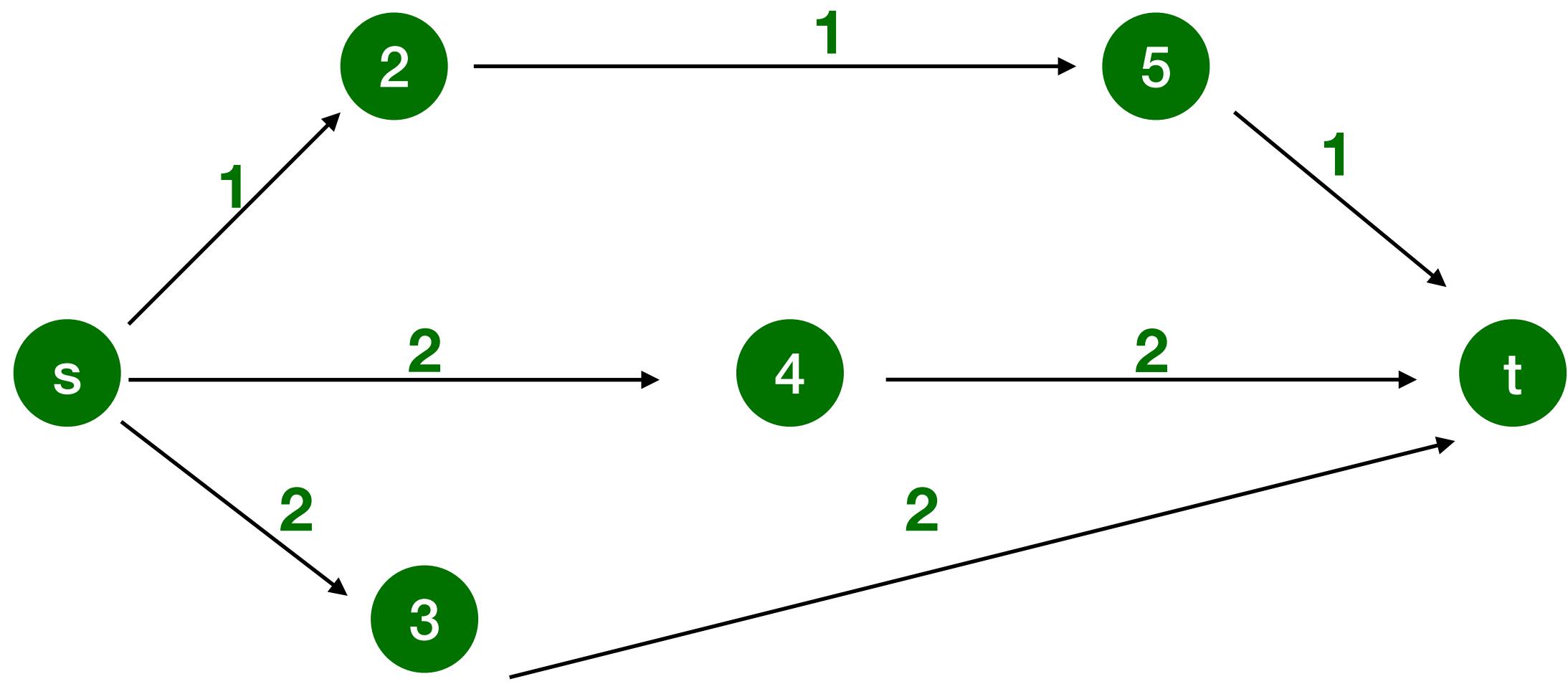
# Ford-Fulkerson



Troba algun camí entre s i t

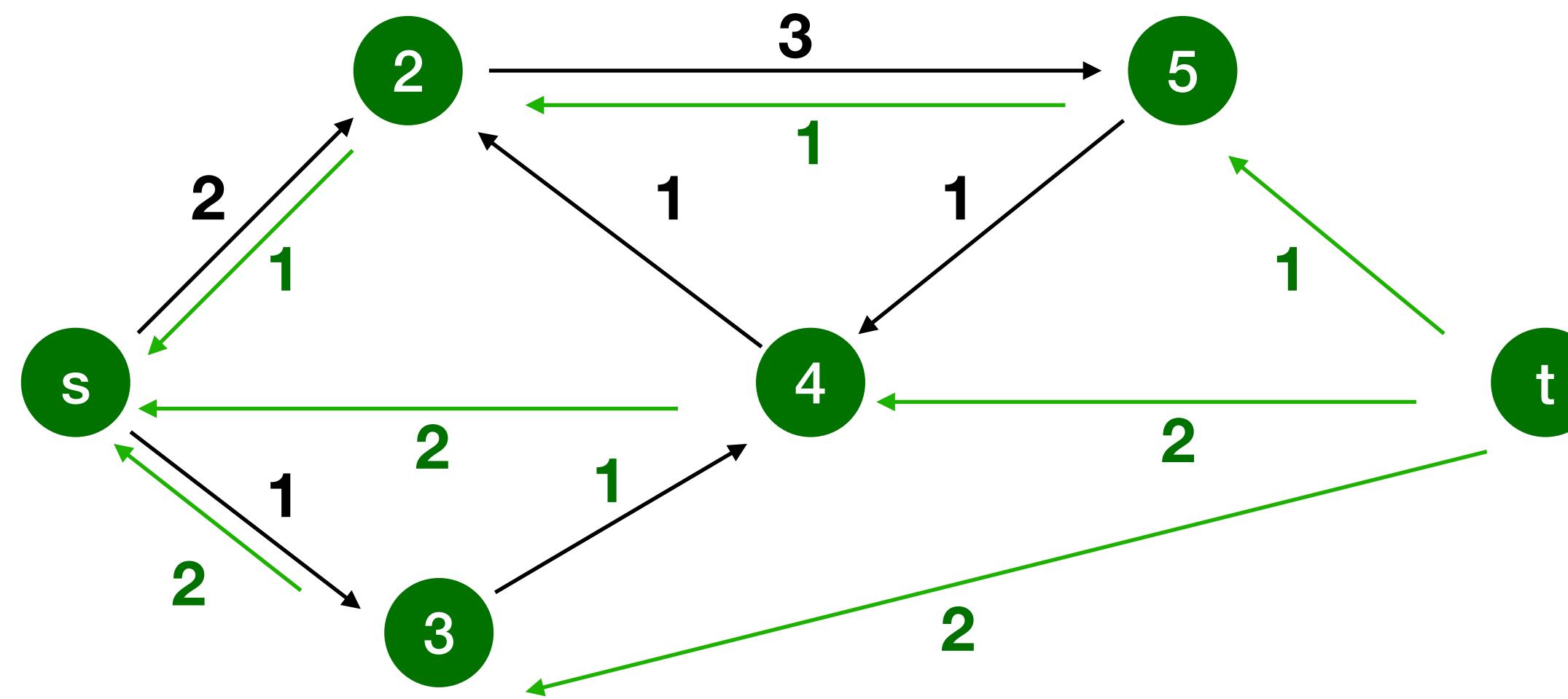
No hi ha cap camí entre s i t. S'ha trobat el flux òptim

# Ford-Fulkerson



Flux òptim

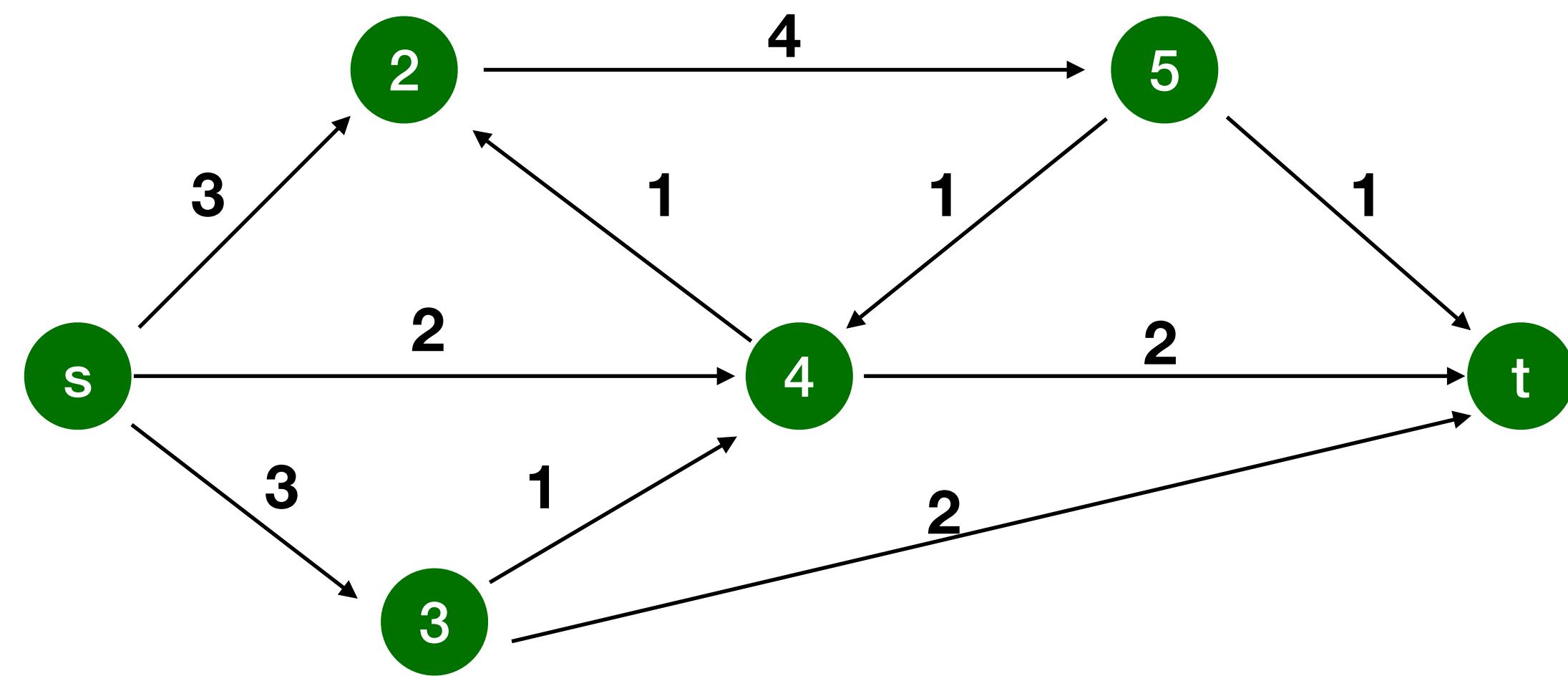
# Quin és min-cut?



Quan no es trobin més rutes al pas 2, no podreu arribar a la t a la xarxa residual.

**S** és el conjunt de nodes accessibles per **s** a la xarxa residual, **V** està format per la resta de nodes.

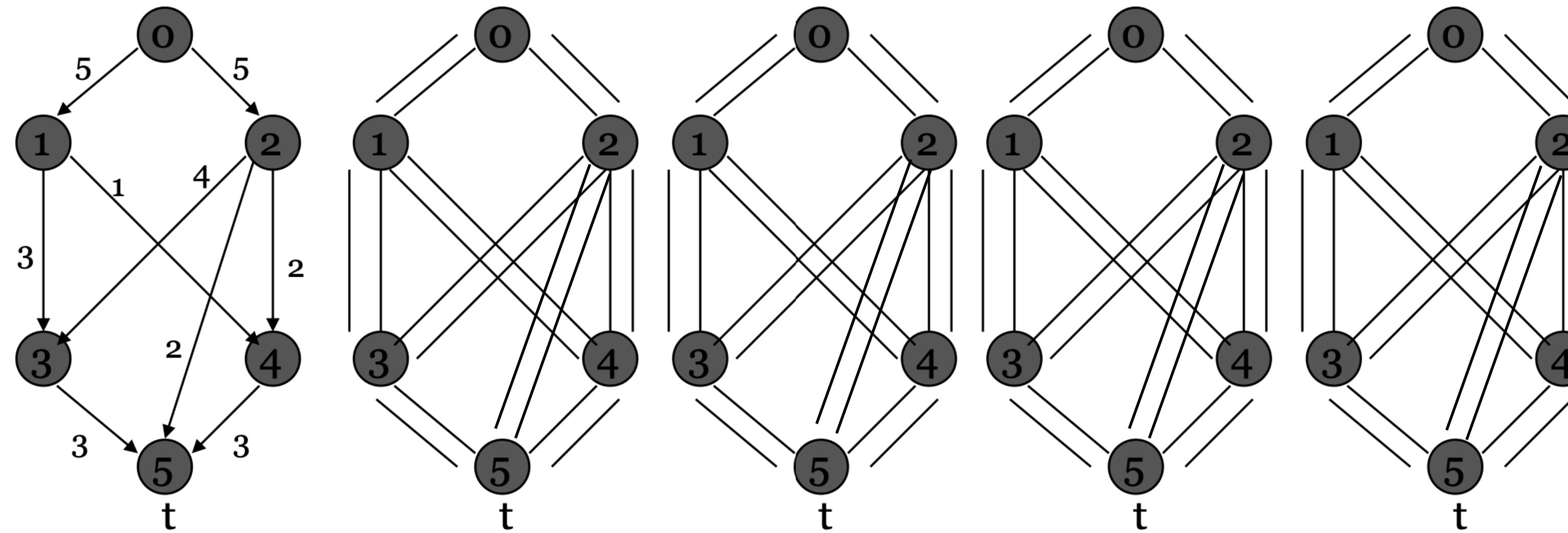
# Ford-Fulkerson



Graf original, i la xarxa residual original

# Exercici: Min-Cut Max-Flow

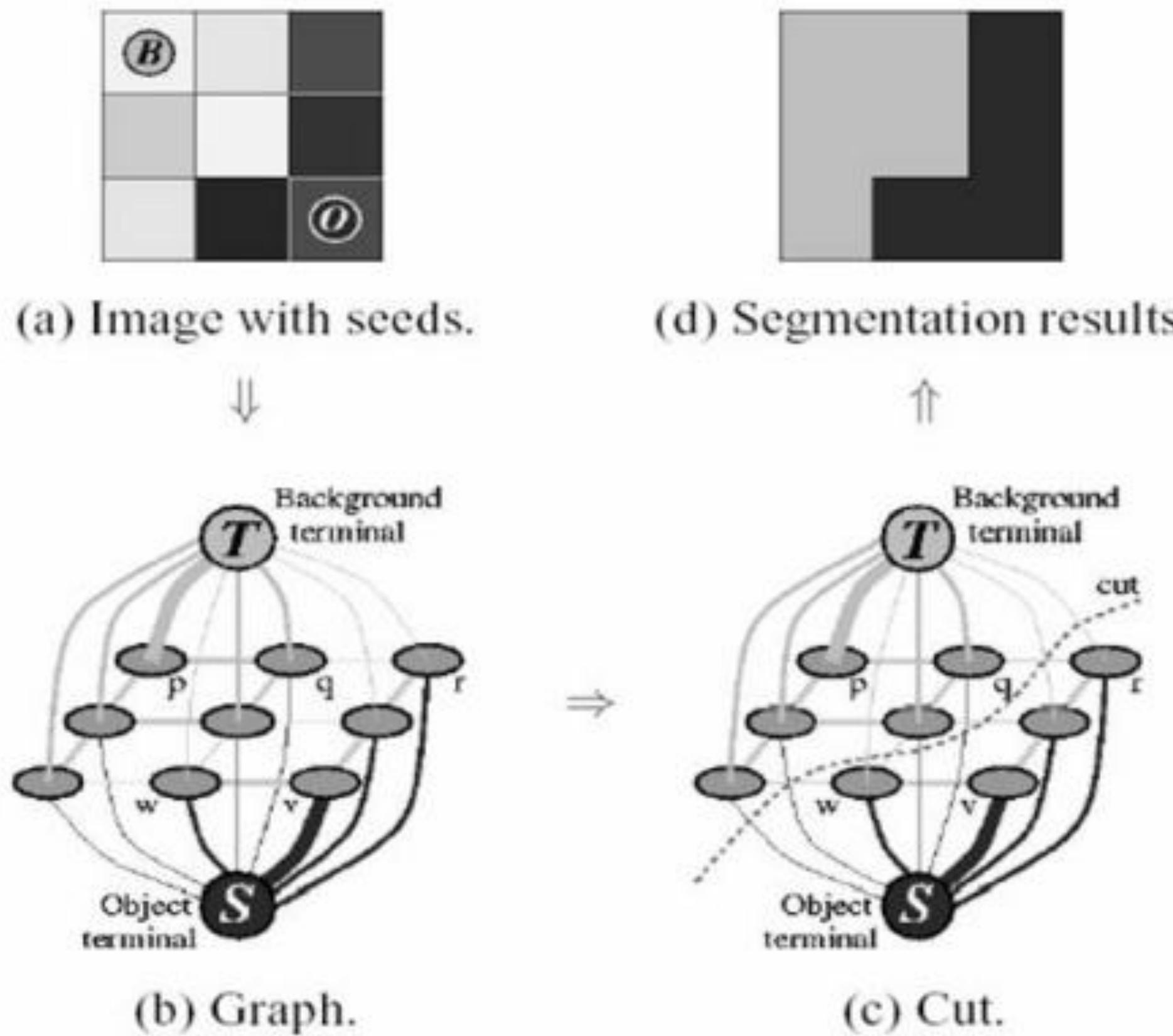
- Aplica Ford-Fulkerson: identifica **min-cut** y **max-flow**



**min-cut:**

**Max-flow:**

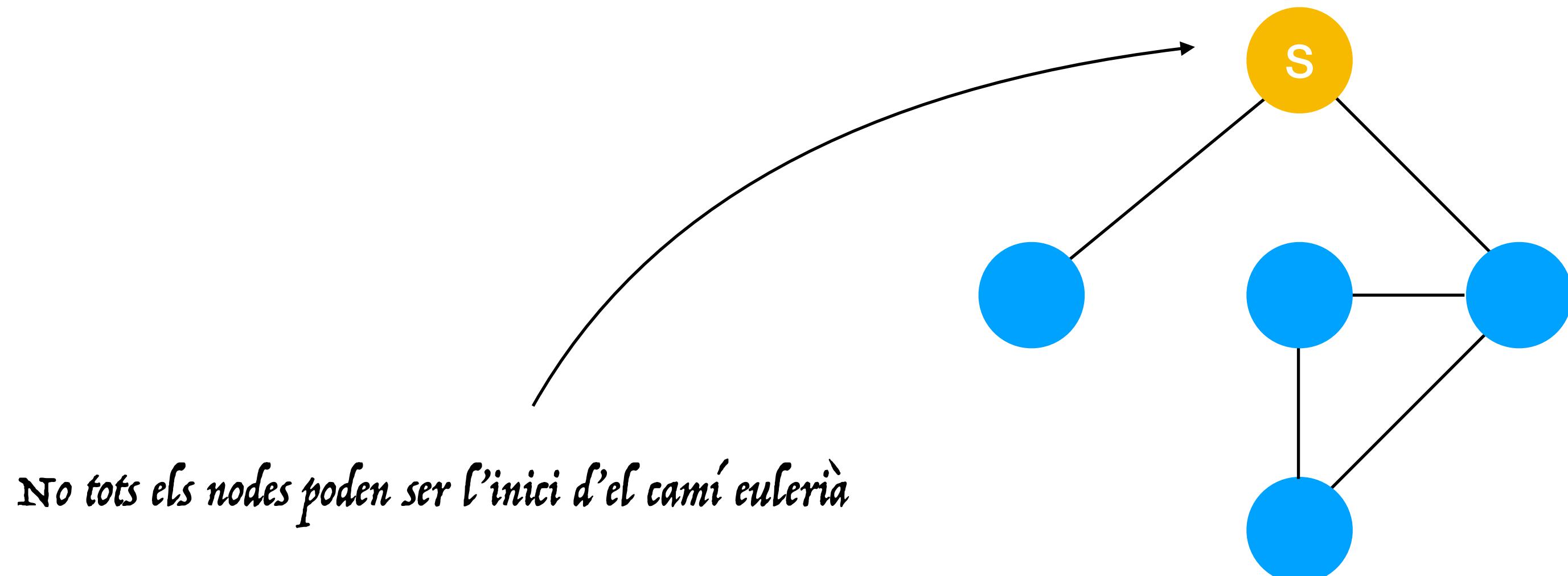
GrabCut was the method to accurately segment the foreground of an image from the background



# Camí/Circuit Eulerià

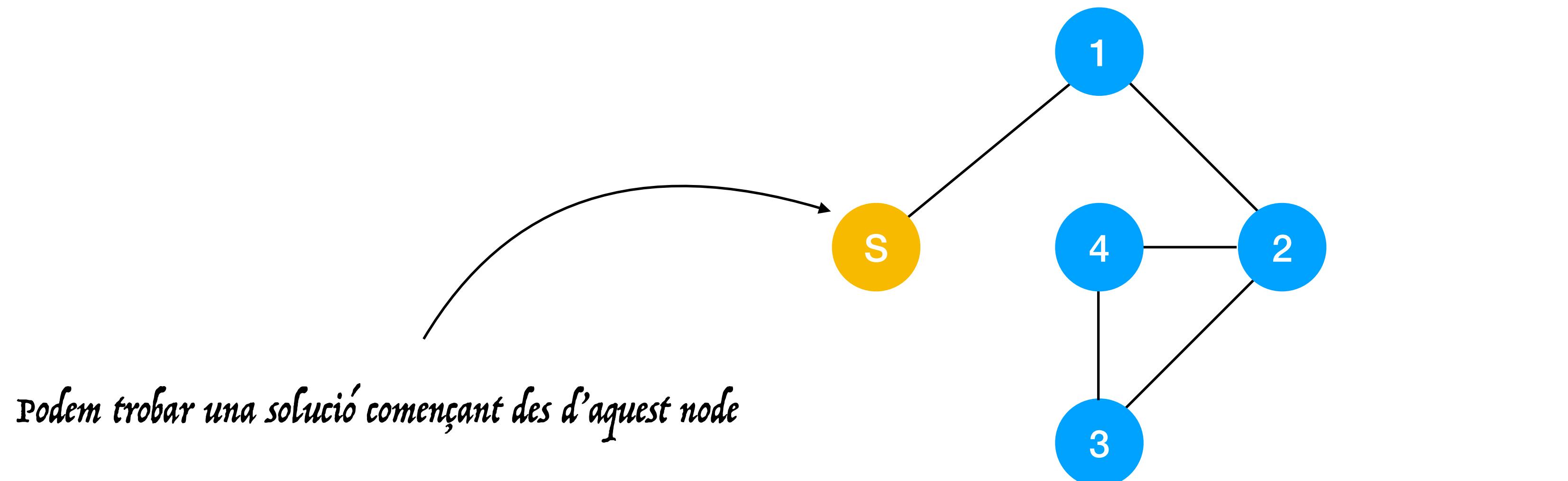
# Camí Eulerià

- Un **camí Eulerià** és aquell camí que recorre **tots les arestes** d'un graf passant una i només una vegada per a cada una d'elles.
- No tots els grafs tenen un camí Eulerià



# Camí Eulerià

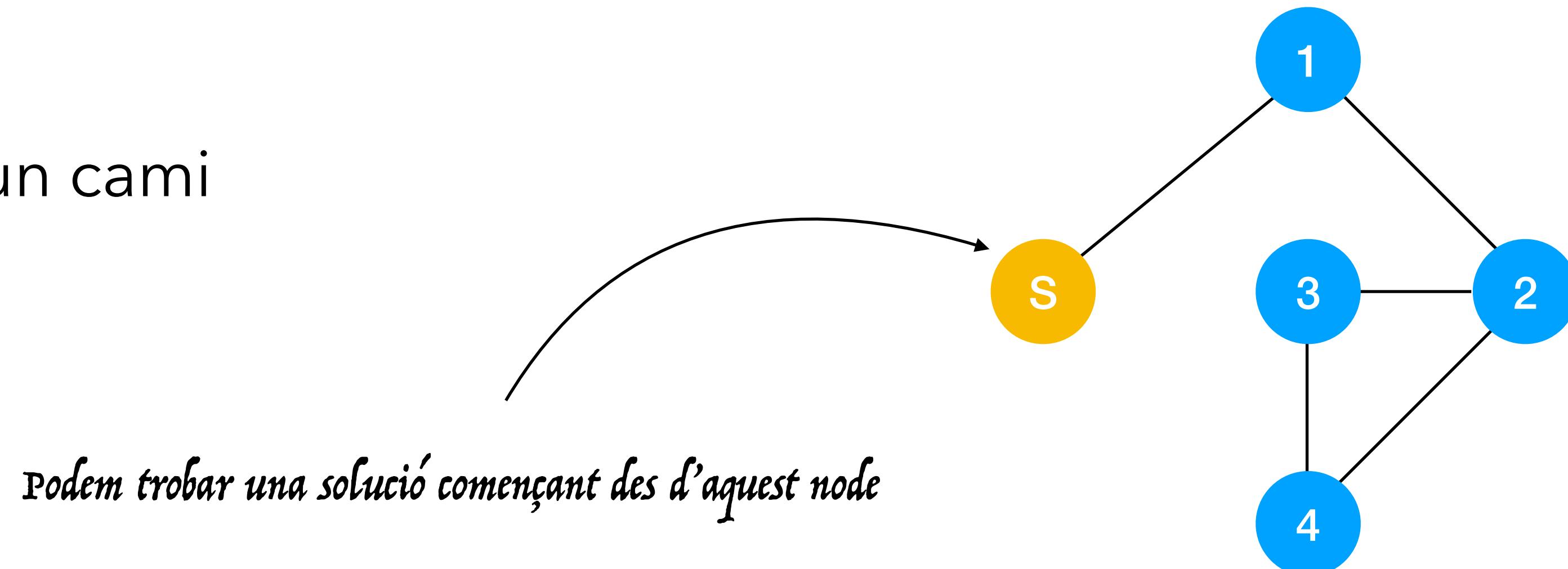
- Un **camí Eulerià** és aquell camí que recorre **tots les arestes** d'un graf passant una i només una vegada per a cada una d'elles.
- No tots els grafs tenen un camí Eulerià
- El graf de la dreta té un camí Eulerià, però per trobar-lo hem de començar des de un node en concret



# Camí Eulerià

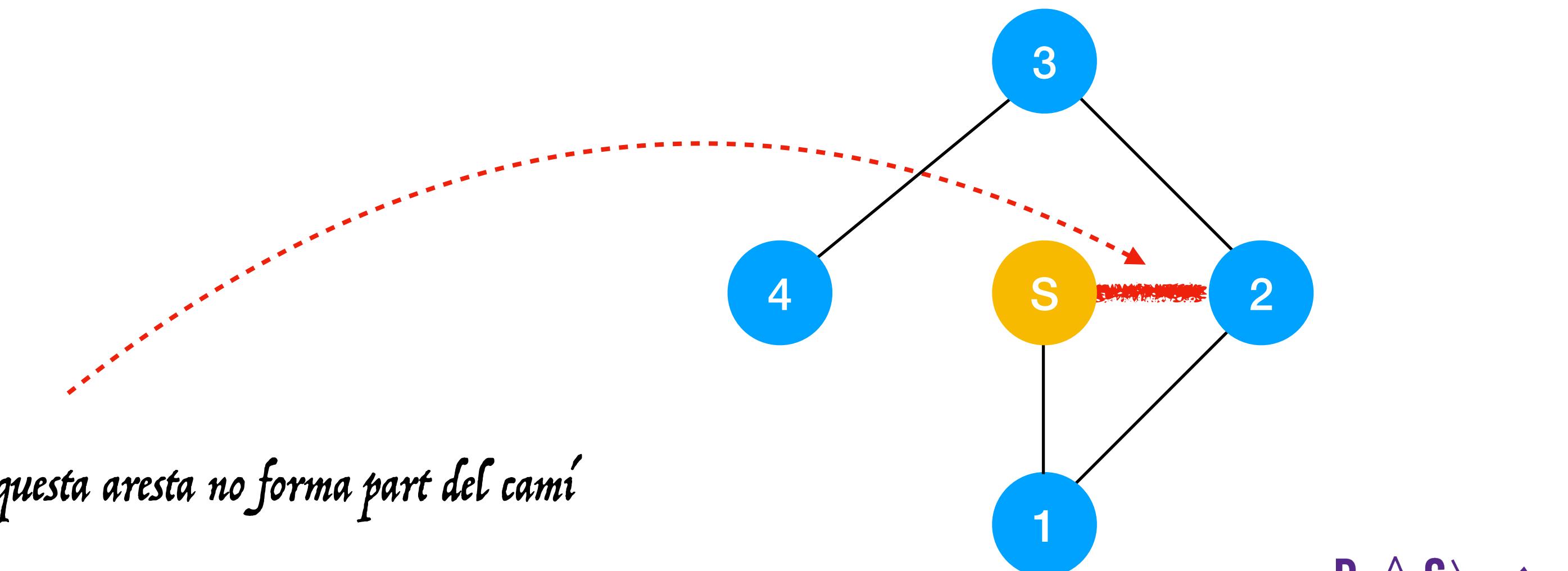
- Un **camí Eulerià** és aquell camí que recorre **tots les arestes** d'un graf passant una i només una vegada per a cada una d'elles.
- No tots els grafs tenen un camí Eulerià
- El graf de la dreta té un camí Eulerià, però per trobar-lo hem de començar des de un node en concret

- Pot haver-hi més d'un camí



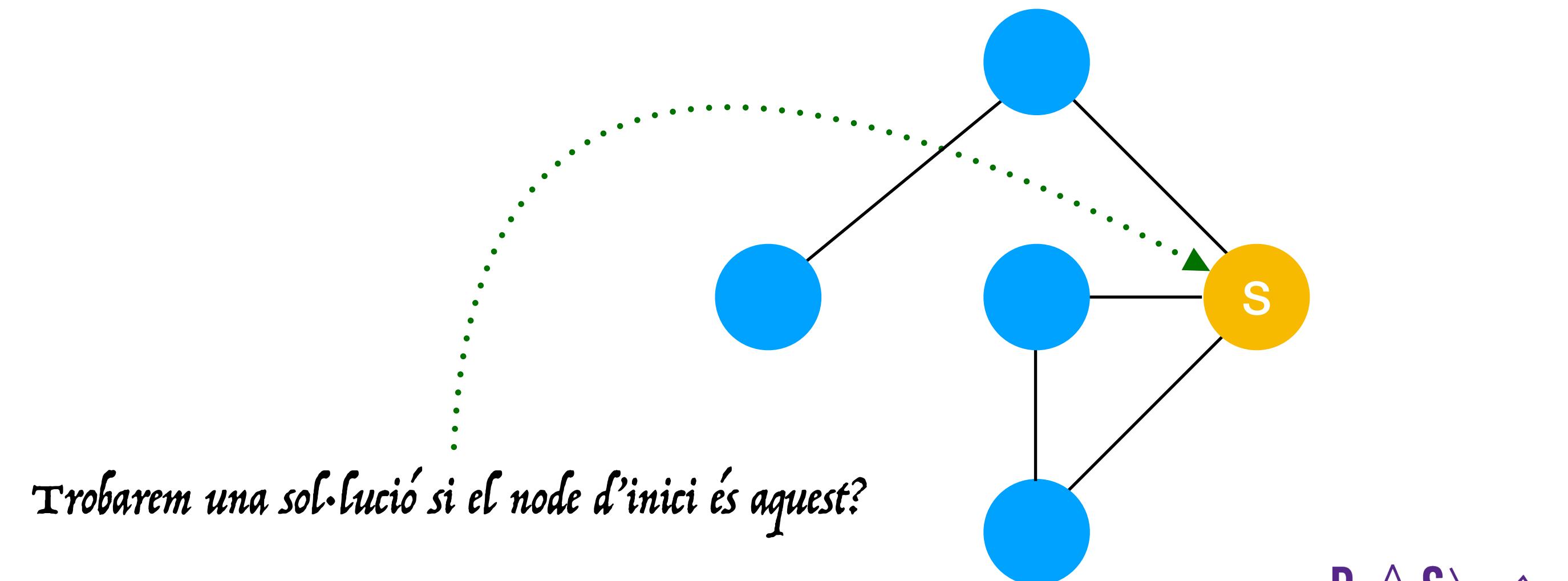
# Camí Eulerià

- Un **camí Eulerià** és aquell camí que recorre **tots les arestes** d'un graf passant una i només una vegada per a cada una d'elles.
- No tots els grafs tenen un camí Eulerià
- El graf de la dreta té un camí Eulerià, però per trobar-lo hem de començar des de un node en concret
  - Pot haver-hi més d'un camí



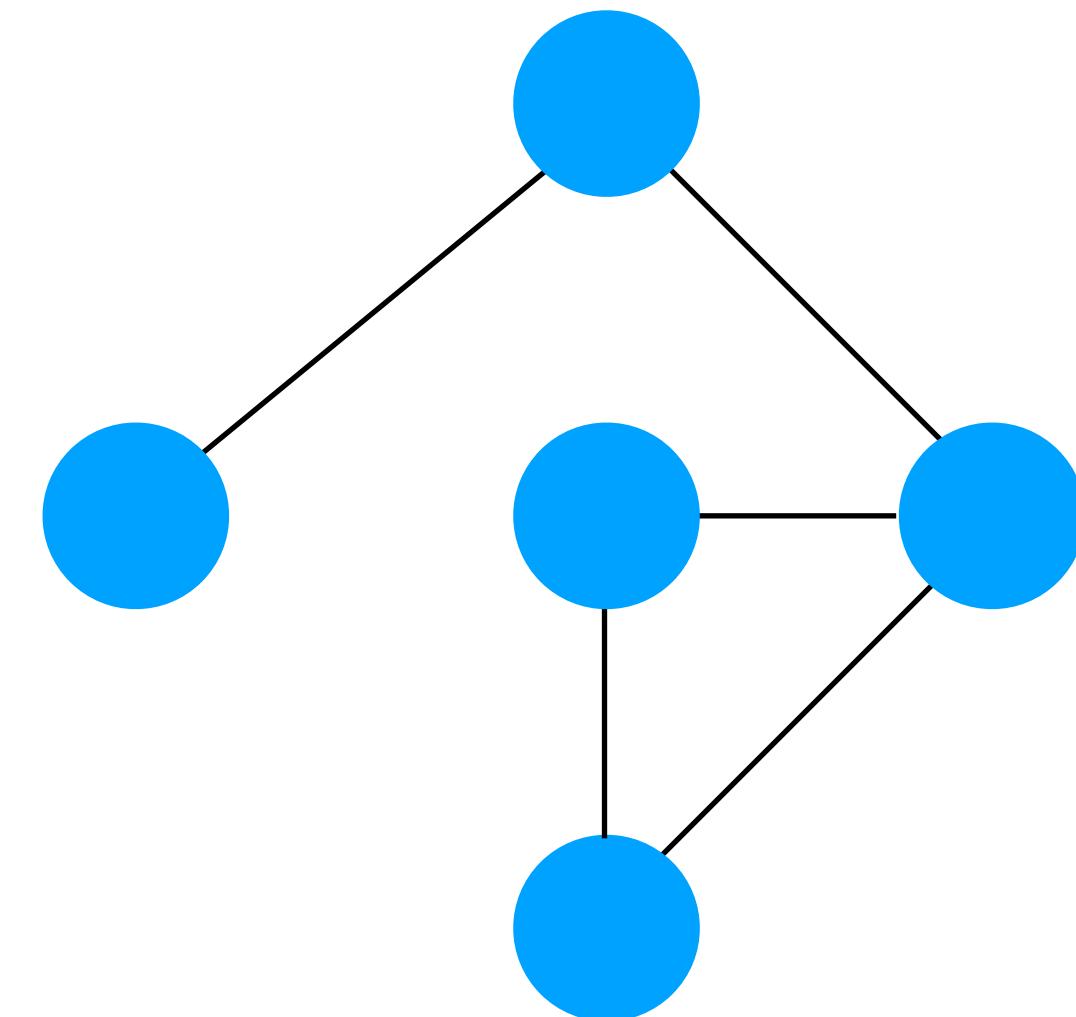
# Camí Eulerià

- Un **camí Eulerià** és aquell camí que recorre **tots les arestes** d'un graf passant una i només una vegada per a cada una d'elles.
- No tots els grafs tenen un camí Eulerià
- El graf de la dreta té un camí Eulerià, però per trobar-lo hem de començar des de un node en concret
  - Pot haver-hi més d'un camí



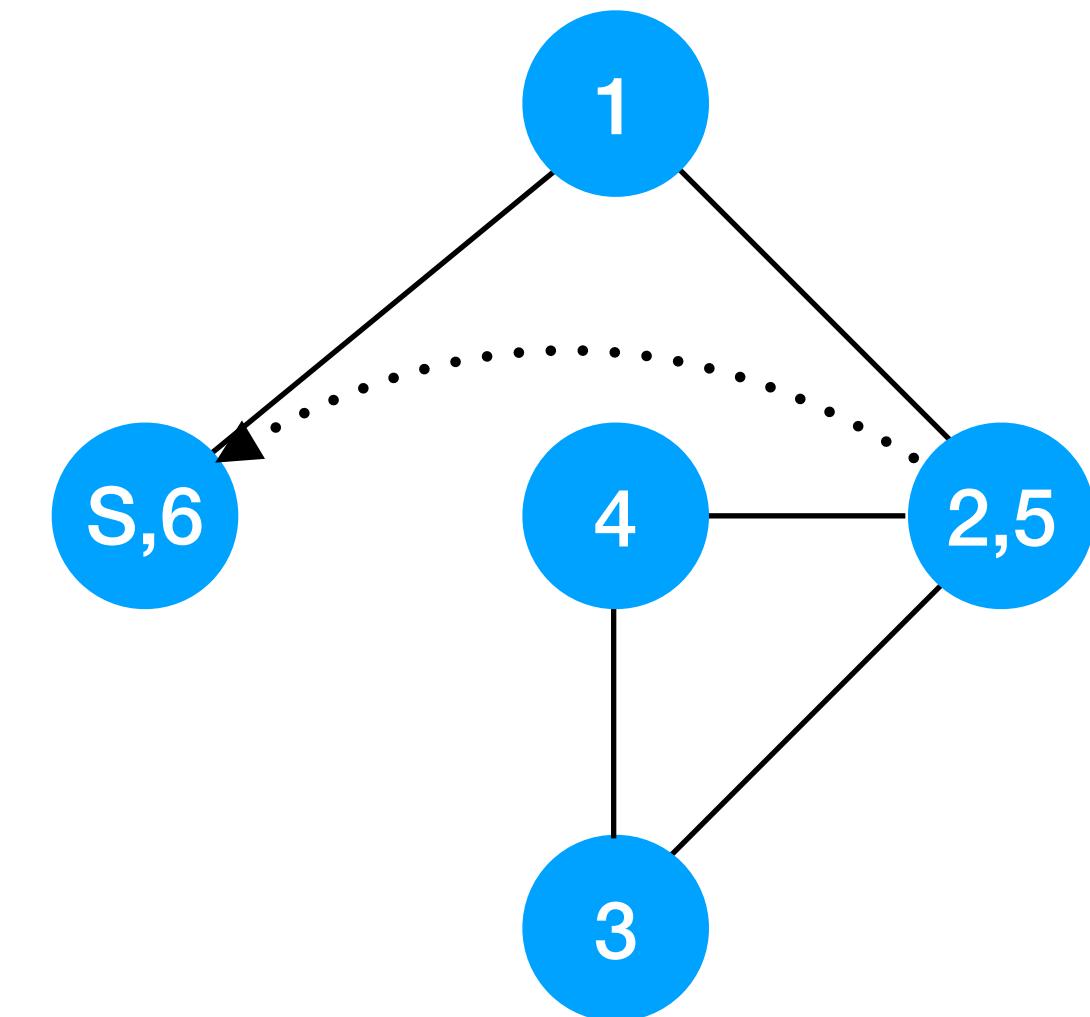
# Camí/Circuit Eulerià

- Un **circuit (o cicle) Eulerià** és aquell **camí eulerià** que comença i acaba en el mateix **node**.



Aquest graf no té un circuit Eulerià!

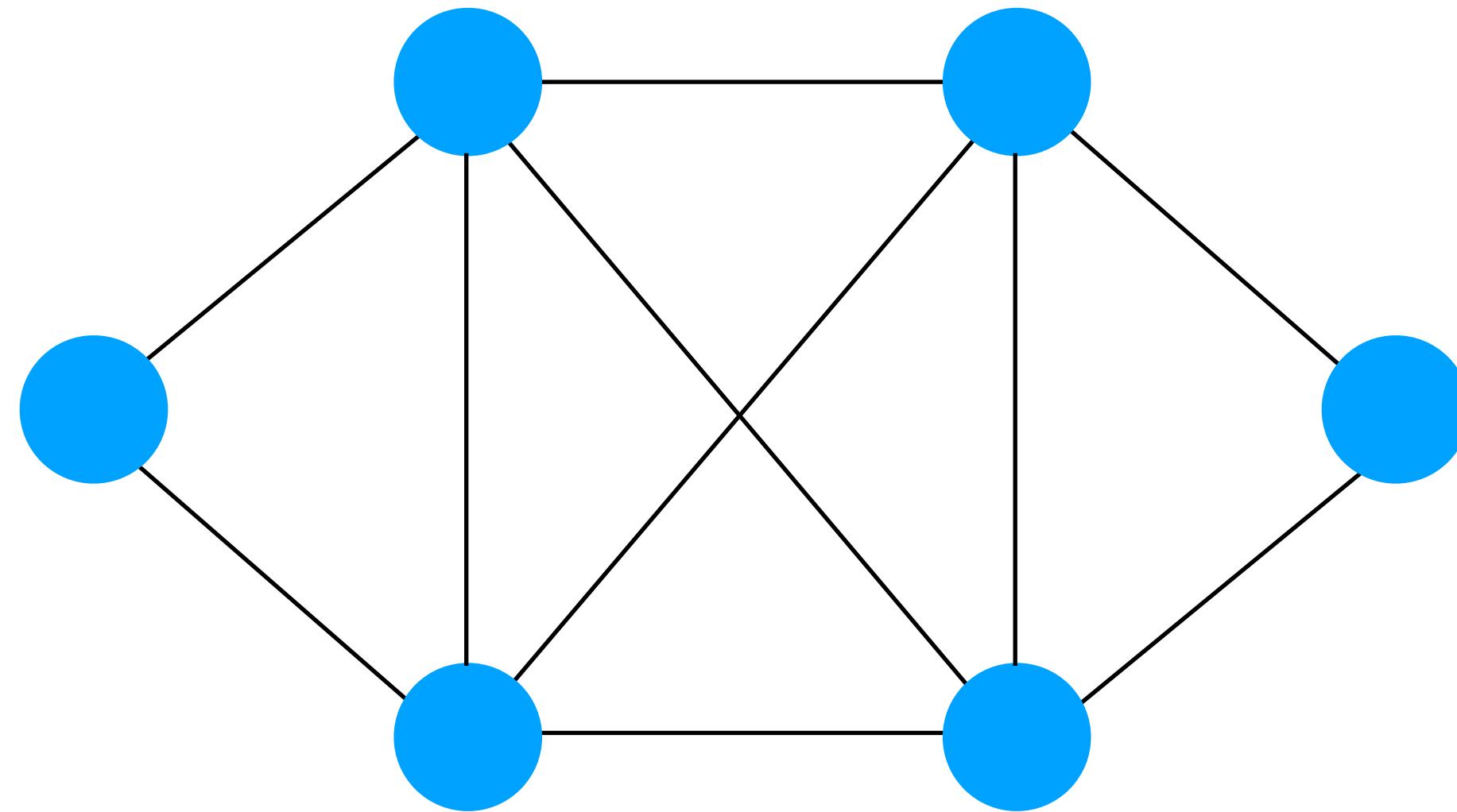
*com l'hauriem de modificar per tal que el graf tingüés un camí Euleria?*



Aquí tenim una possibilitat!

# Camí/Circuit Eulerià

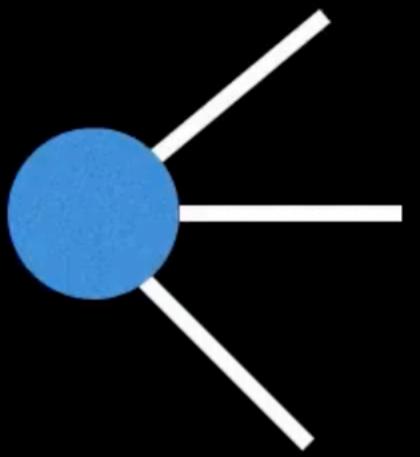
- Un **circuit (o cicle) Eulerià** és aquell **camí eulerià** que comença i acaba en el mateix **node**.



Té un circuit Eulerià aquest graf?

# Que és el grau d'un node?

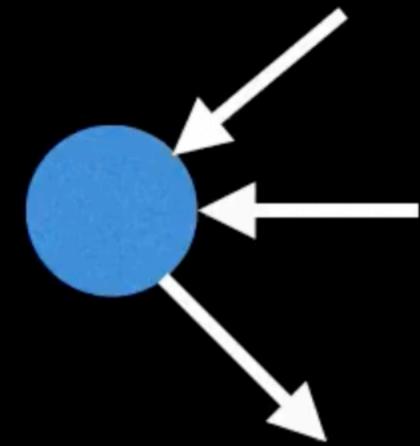
Graf no dirigit



Grau (degree) del node = 3

El grau d'un node és el nombre d'arestes incidents a ell

Graf dirigit



Grau d'entrada (In degree) = 1

Grau de sortida (Out degree) = 3

El grau d'entrada (indegree) es el nombre d'arestes d'entrada i el grau de sortida (outdegree) és el nombre d'arestes de sortida

# Quines son les condicions necessàries per al que el graf tingui un Camí/Circuit Eulerià?

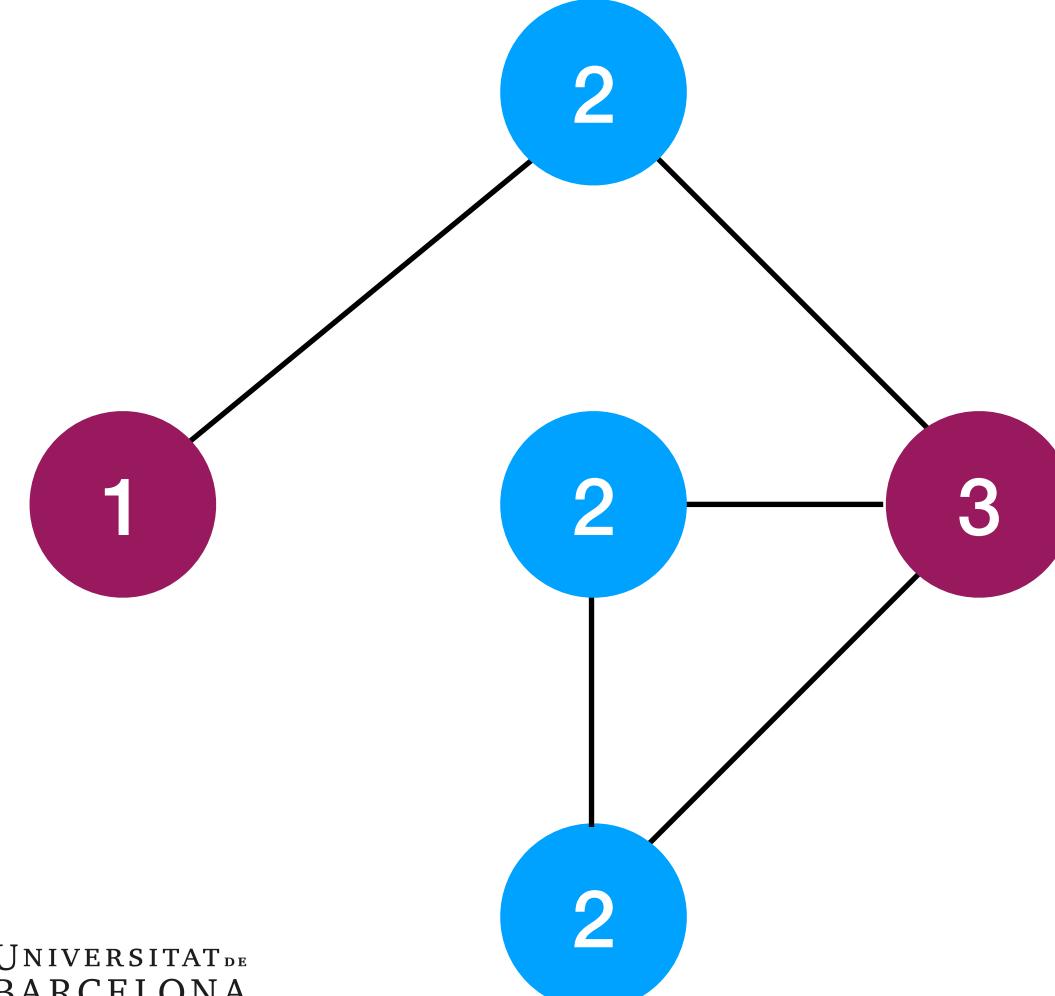
Això depén del tipus de graf que tinguem.

## Graf no dirigit

### Camí Eulerià

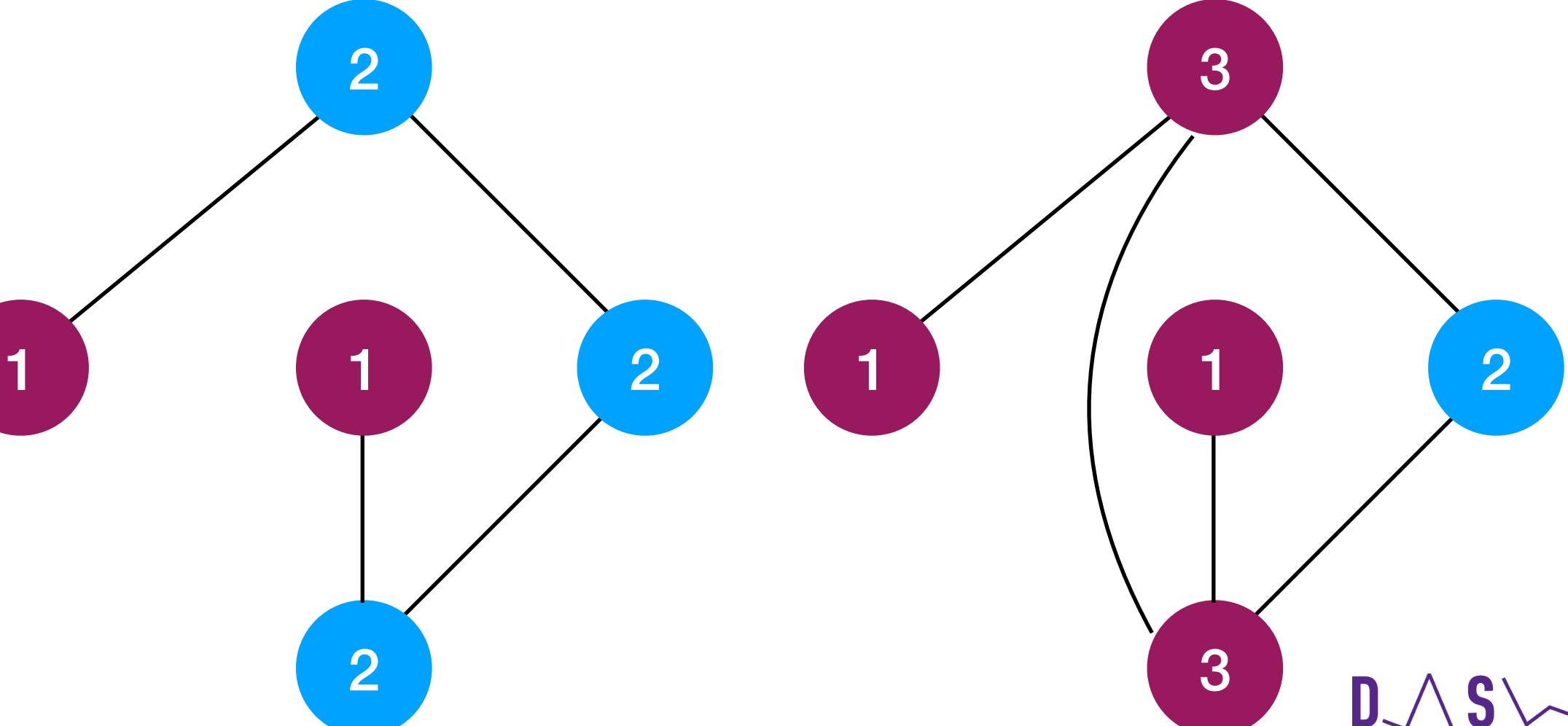
Dues possibilitats:

- 1) Tots els **vèrtex** del graf tenen un **grau parell**
- 2) Hi ha **exactament 2 vèrtex** amb **grau senar**



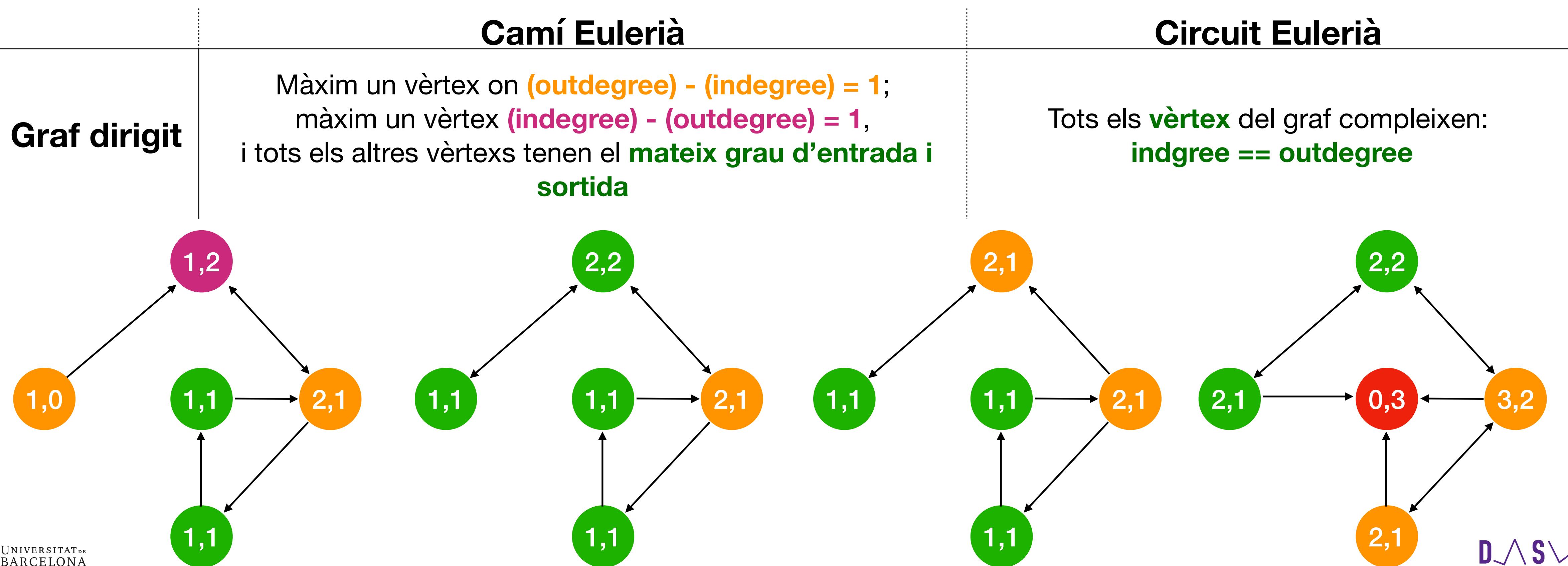
### Circuit Eulerià

Tots els **vèrtex** del graf tenen un **grau parell**

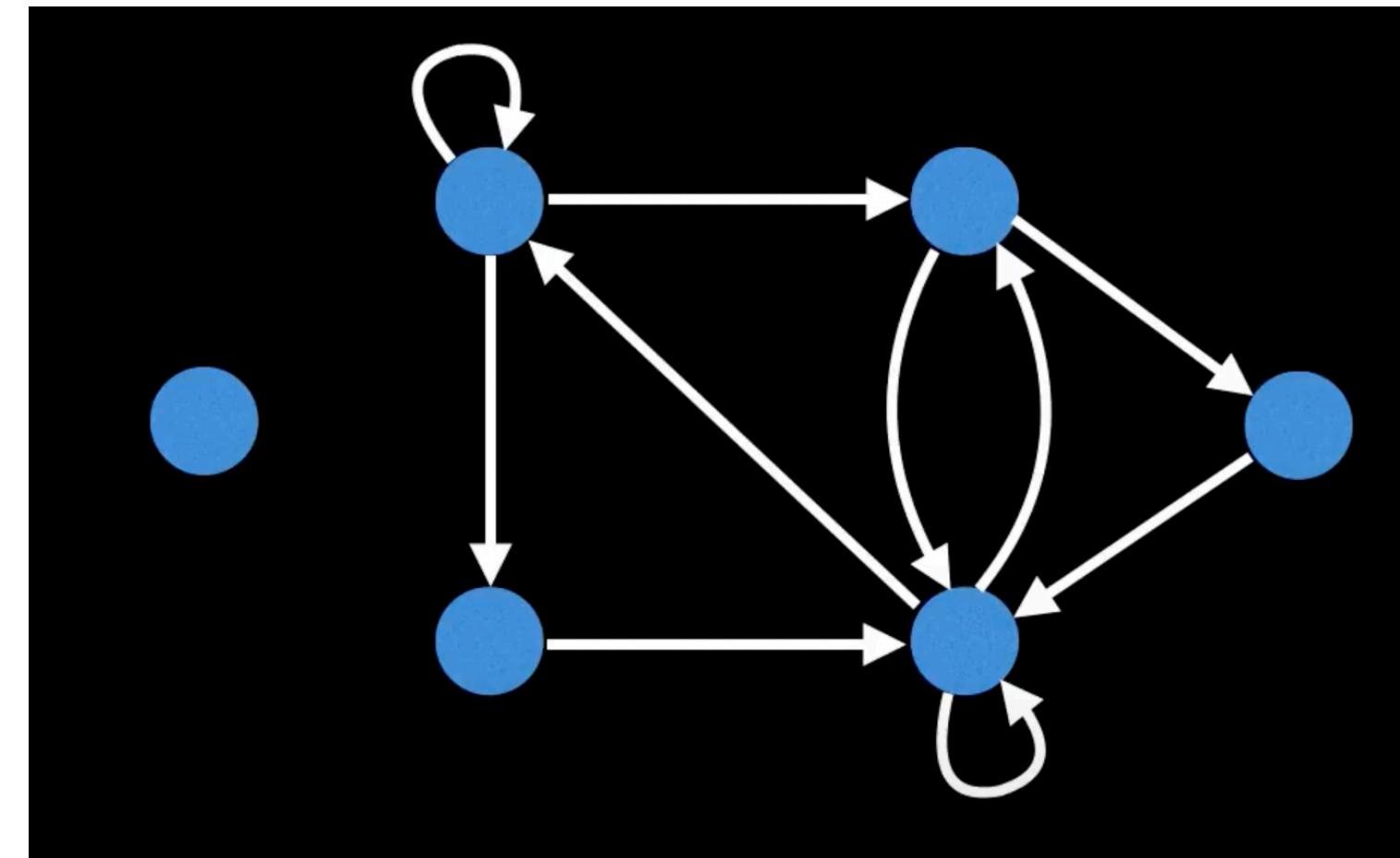
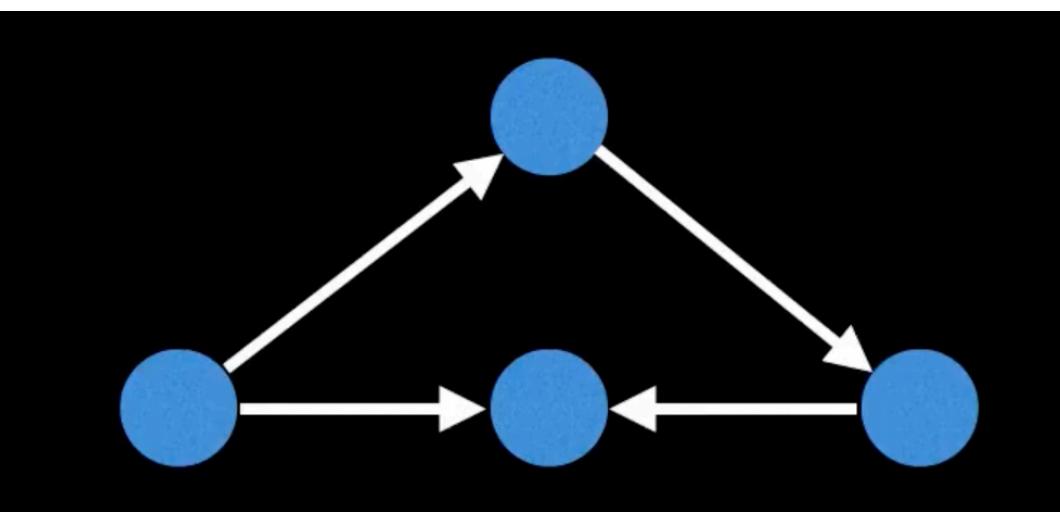
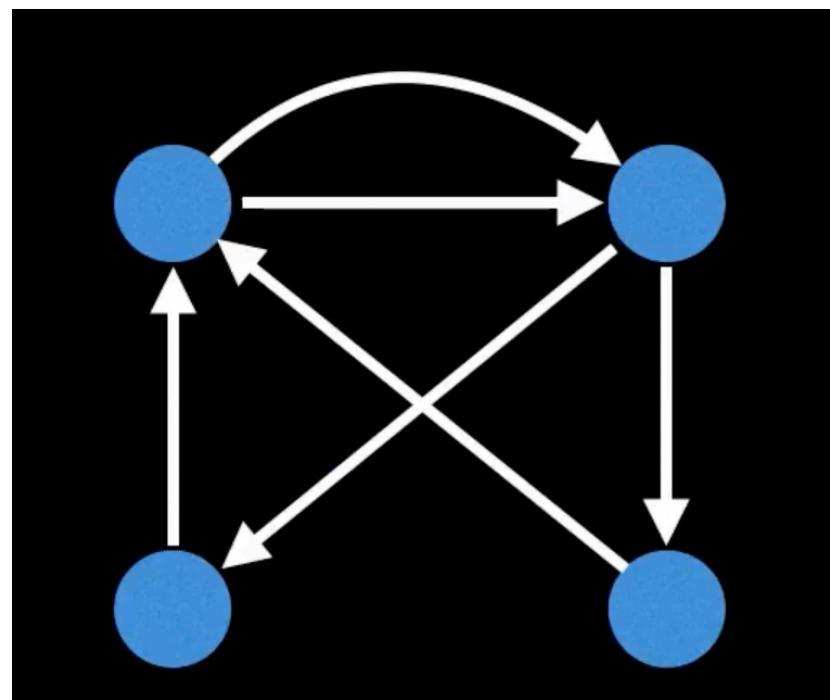
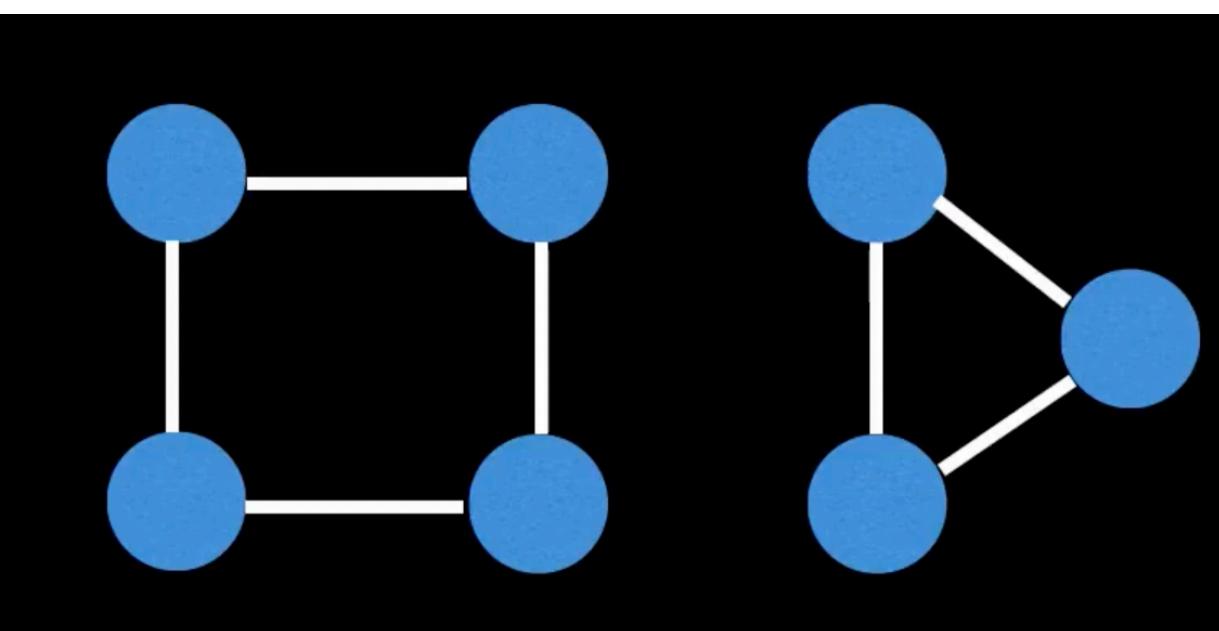
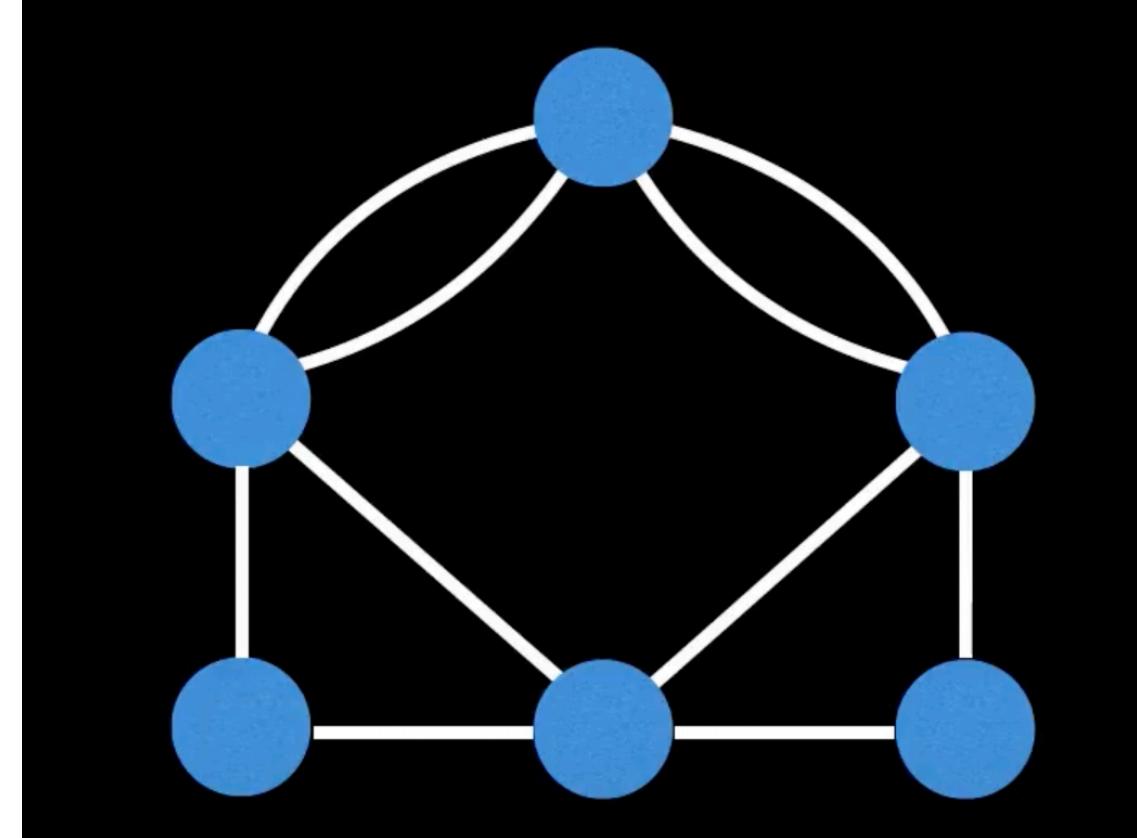
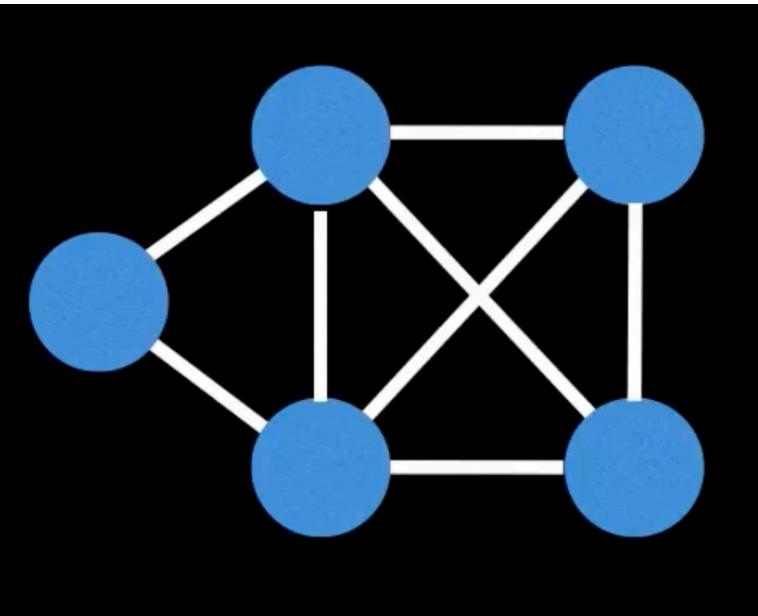
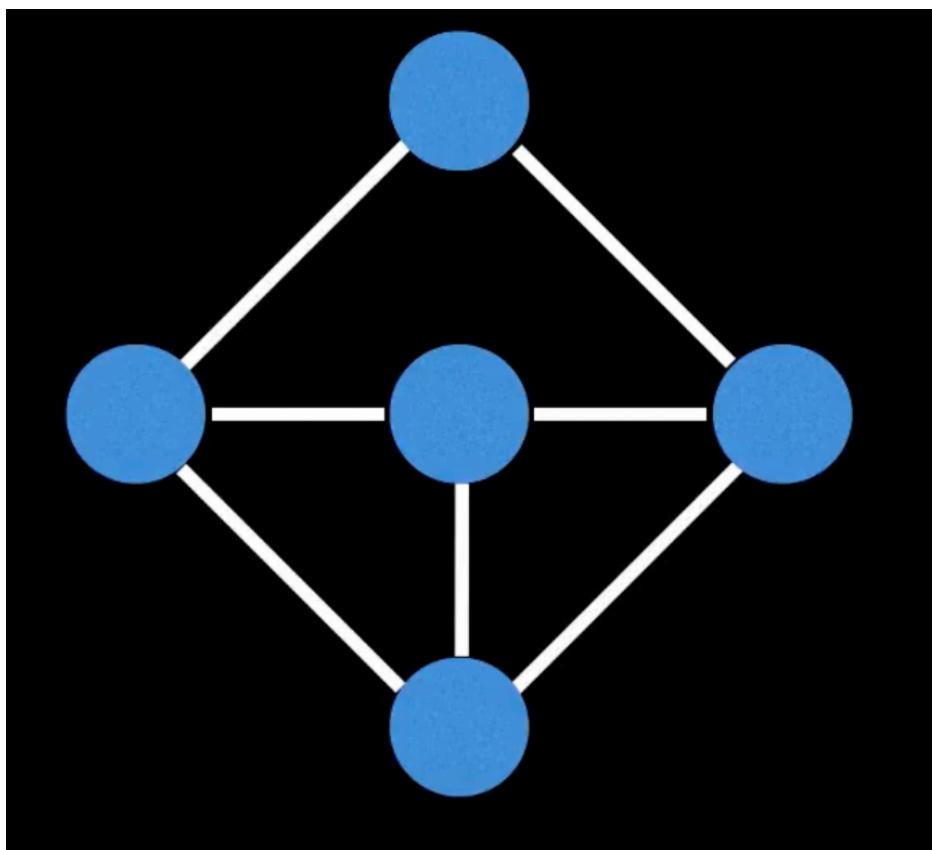


# Quines son les condicions necessàries per al que el graf tingui un Camí/Circuit Eulerià?

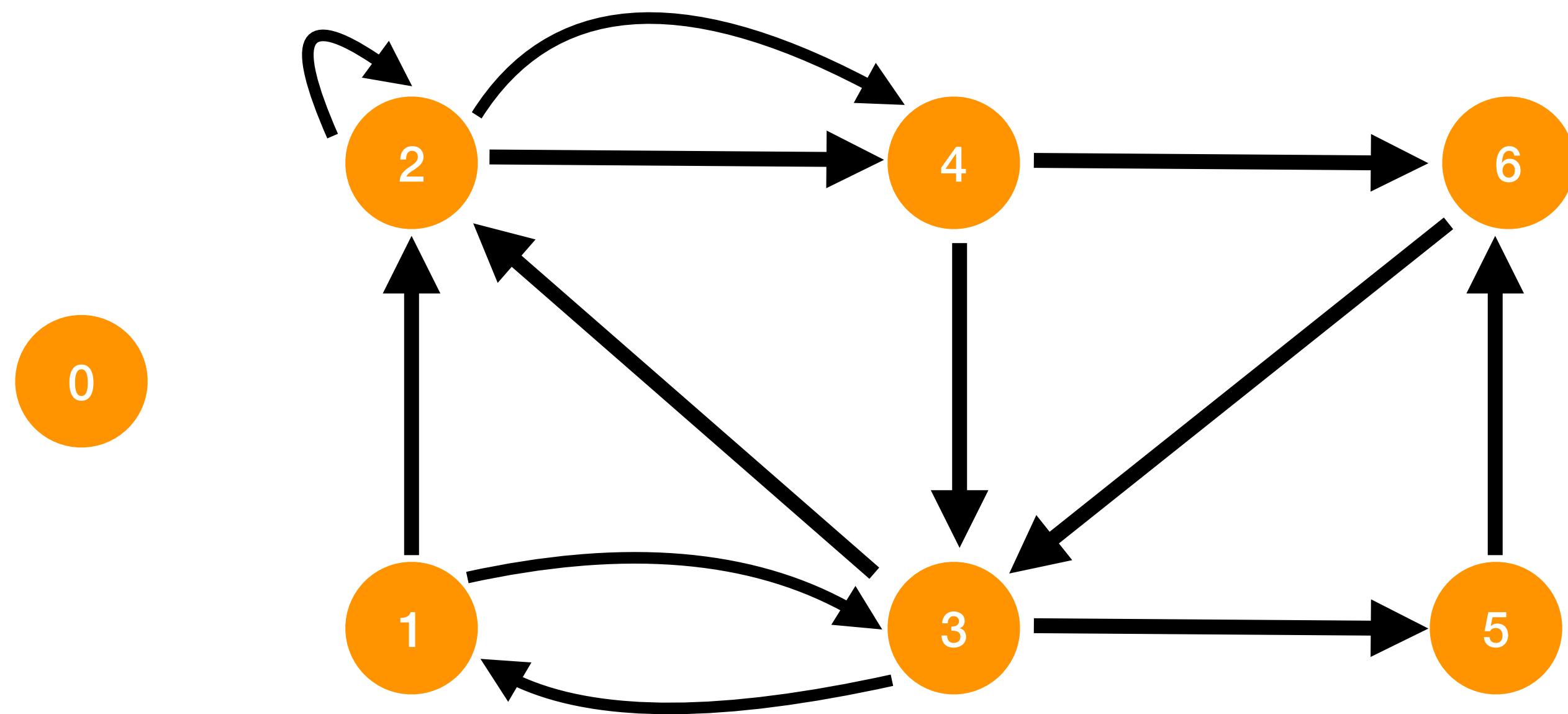
Això depén del tipus de graf que tinguem.



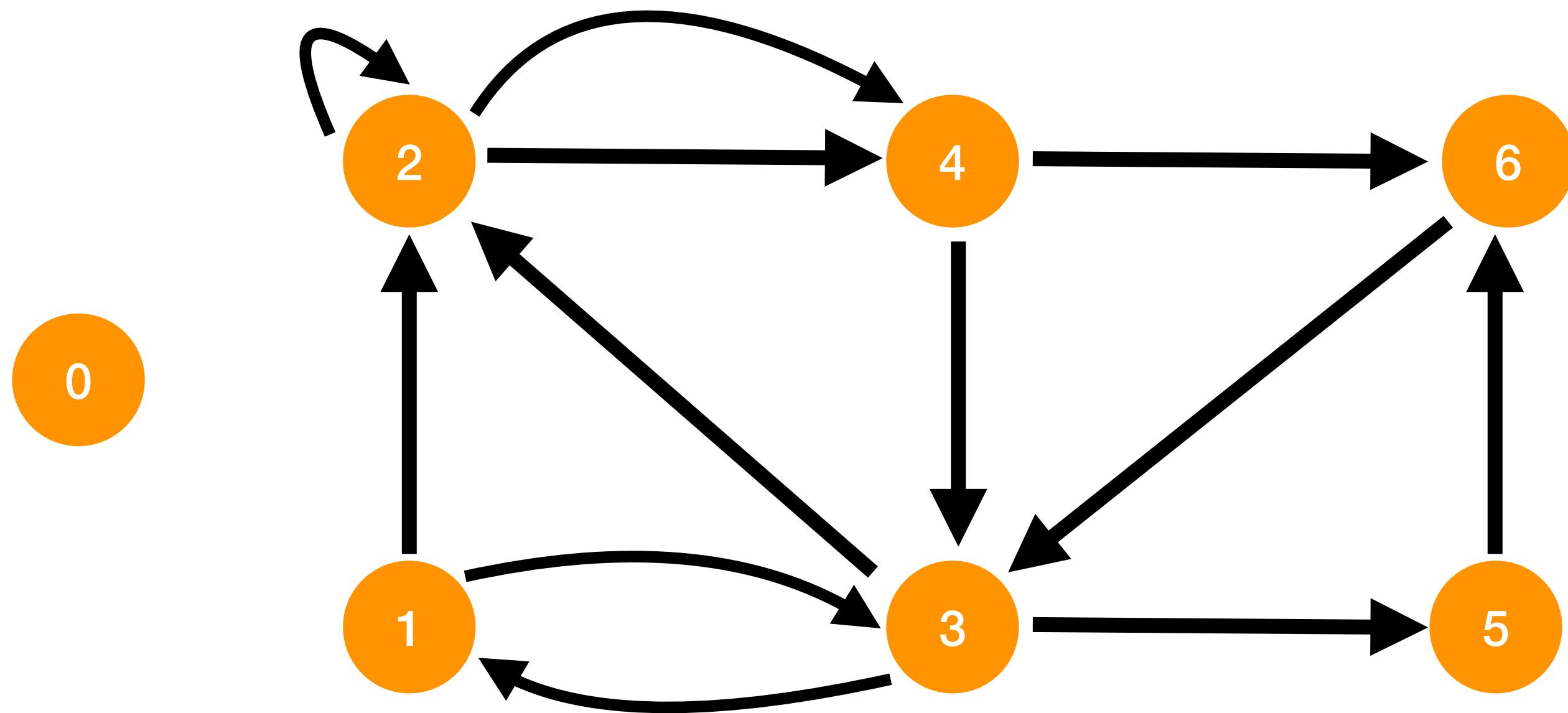
# Quin dels següents grafs tenen un camí i/o circuit eulerià?



# Exemple



# Exemple - Trobar el Camí Eulerià



**Pas 1: Determinar si existeix un camí eulerià.**

# Exemple - Trobar el Camí Eulerià

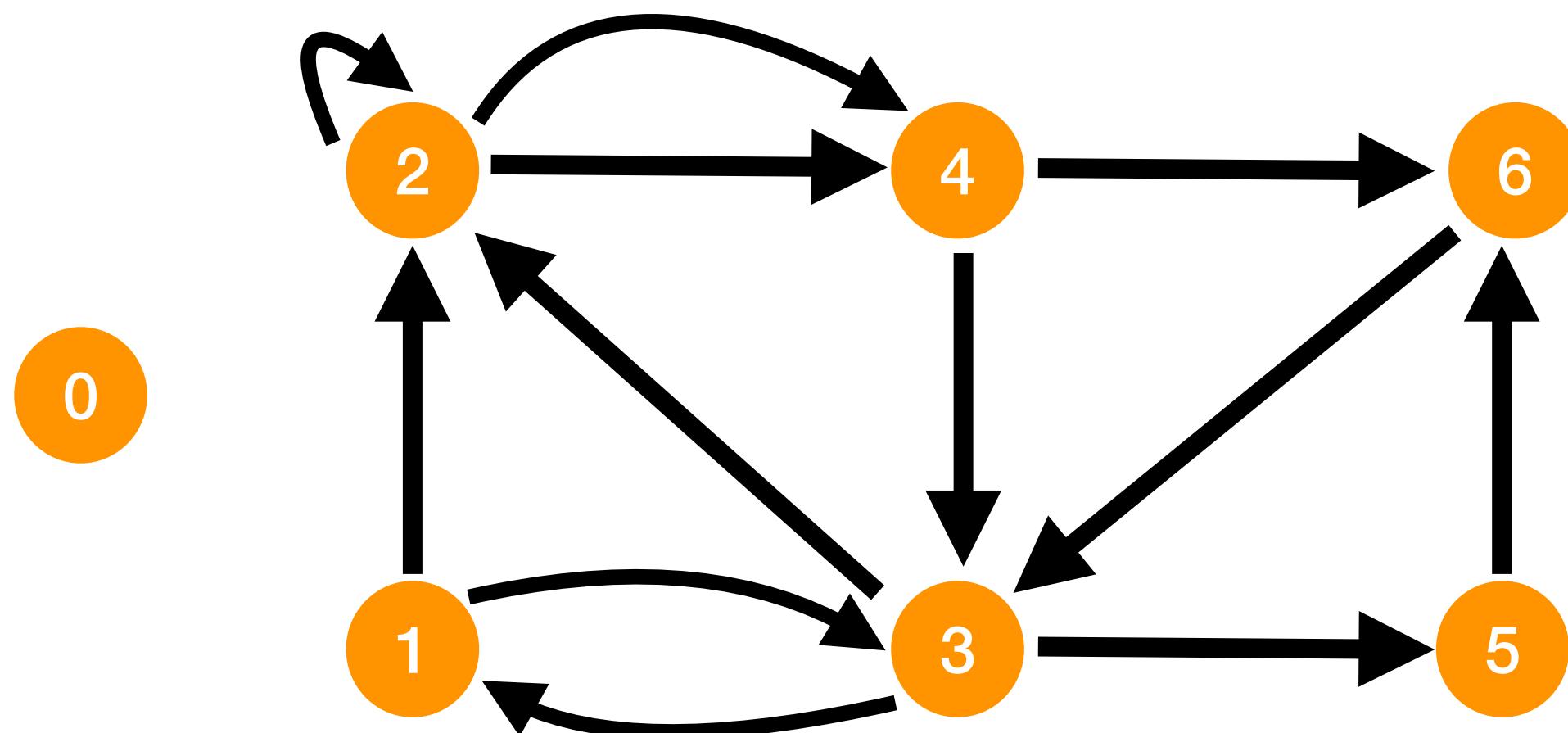
## Pas 1: Determinar si existeix un camí eulerià.

Existeix si:

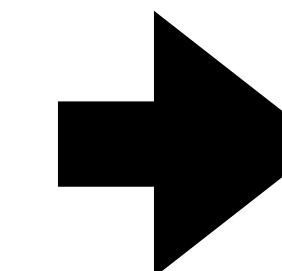
Màxim un vèrtex on **(outdegree) - (indegree) = 1**;

màxim un vèrtex **(indegree) - (outdegree) = 1**,

i tots els altres vèrtexs tenen el **mateix grau d'entrada i sortida**



Node	In	Out
0	0	0
1	2	2
2	3	3
3	3	3
4	2	2
5	1	1
6	1	2



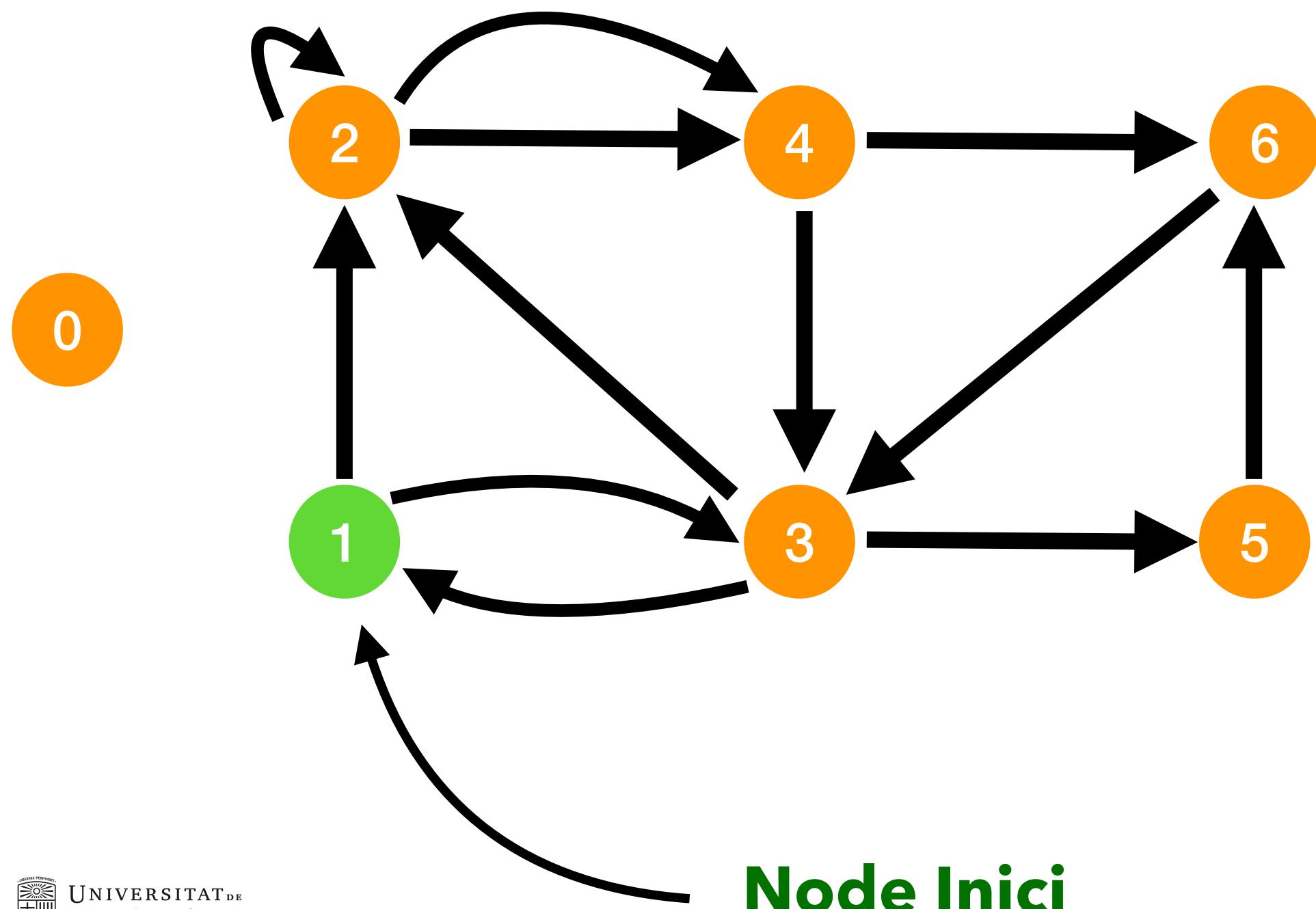
**Existeix!**

# Exemple - Trobar el Camí Eulerià

## Pas 2: Determinar node d'inici

Si hi ha 0 vèrtexs imparells, comencem en qualsevol lloc.

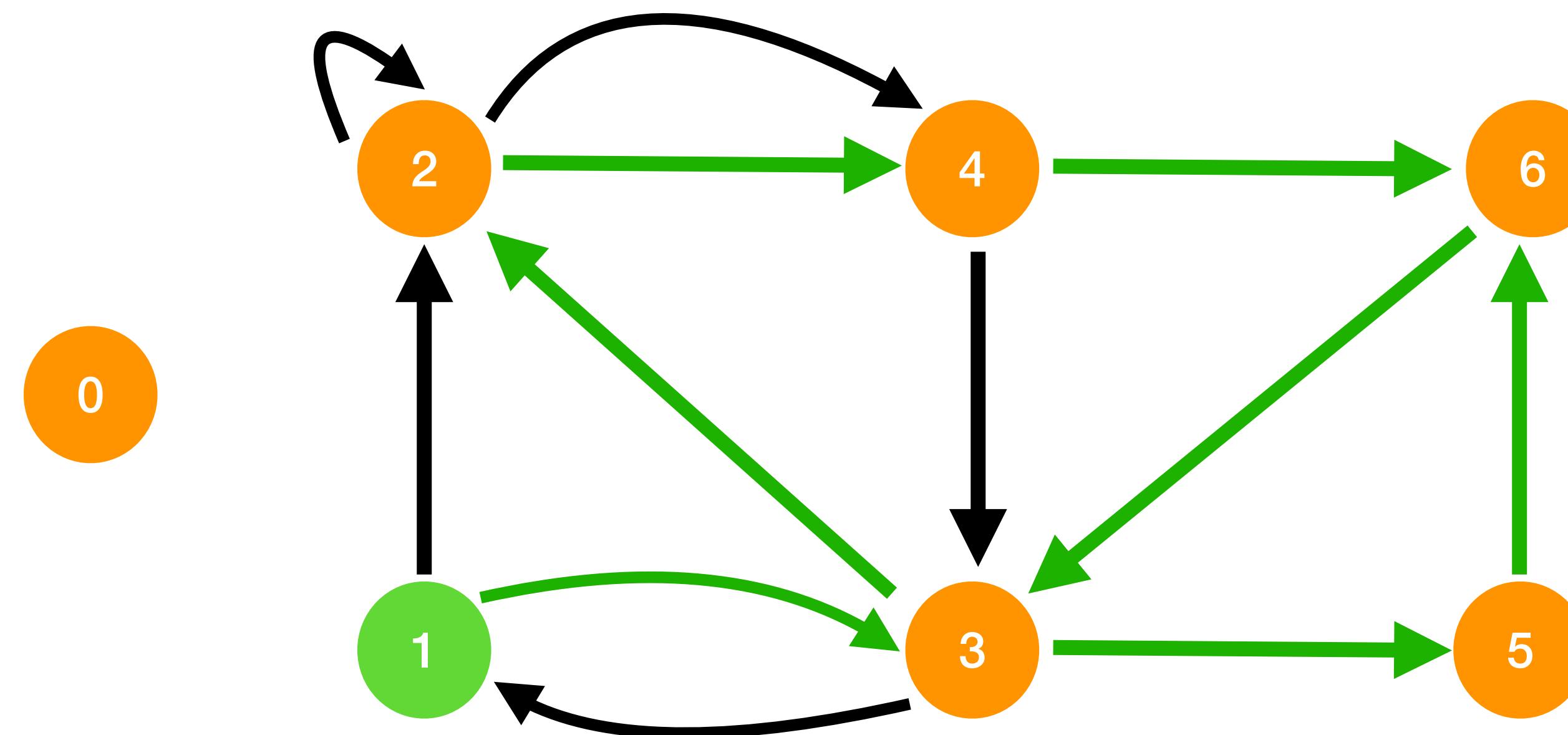
Si hi ha 2 vèrtexs senars, comencem per un d'ells.



Node	In	Out
0	0	0
1	1	2
2	3	3
3	3	3
4	2	2
5	1	1
6	2	1

# Exemple - Trobar el Camí Eulerià

## Pas 3: Trobar el camí Eulerià



### Que passa si apliquem el DFS?

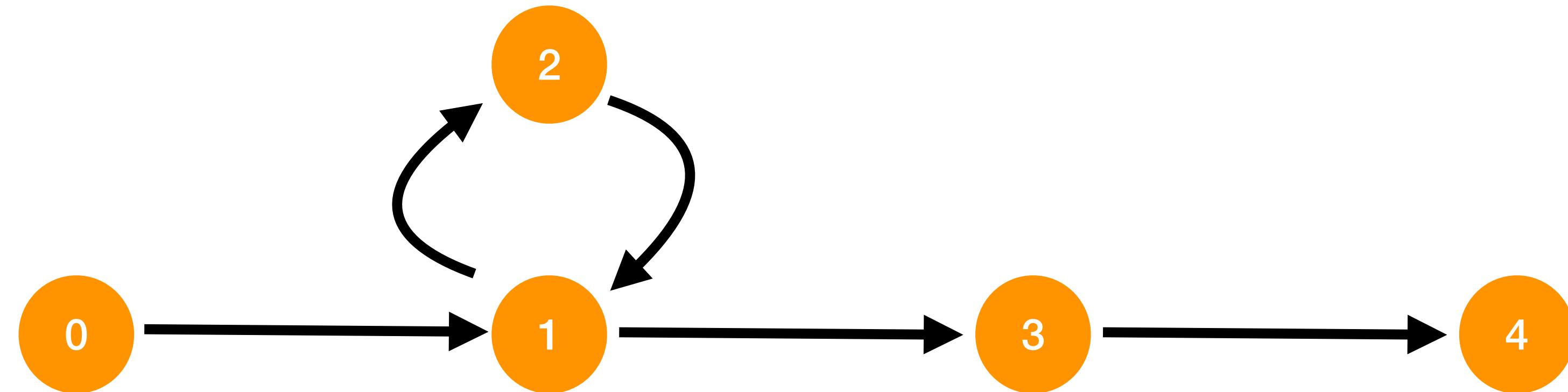
seleccionant aleatoriament les arestes durant el DFS, trobarem un camí que va del node inicial fins al node final

No trobem la solució ja que hi ha arestes que no son visitades

# Exemple - Trobar el Camí Eulerià

- Hem de modificar el DFS per tal que forcem a visitar totes les Arestes

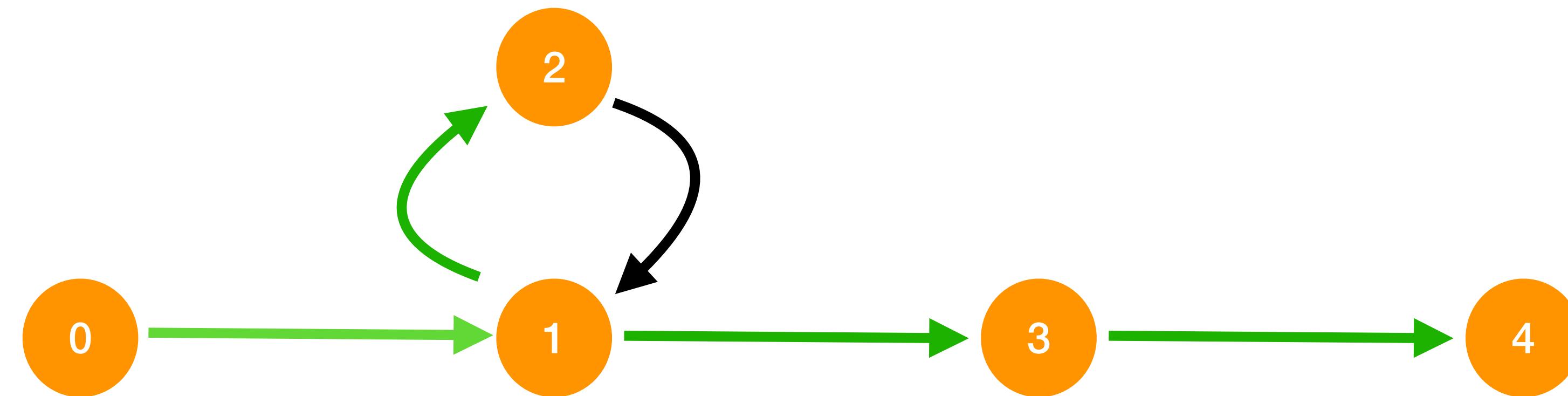
Per il·lustrar el problema, considerem el següent graf on el node d'inici es el node 0



# Exemple - Trobar el Camí Eulerià

- Hem de modificar el DFS per tal que forcem a visitar totes les Arestes

Per il·lustrar el problema, considerem el següent graf on el node d'inici es el node 0

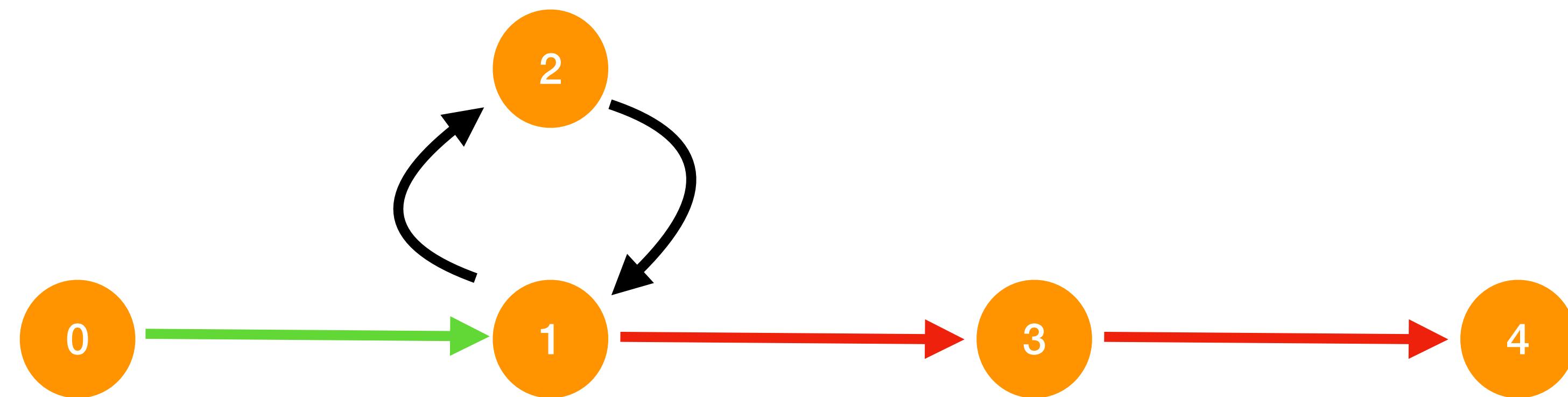


El DFS es podria donar el següent resultat: **0, 1, 3, 4, 2**

# Exemple - Trobar el Camí Eulerià

- Hem de modificar el DFS per tal que forcem a visitar totes les Arestes

Per il·lustrar el problema, considerem el següent graf on el node d'inici es el node 0

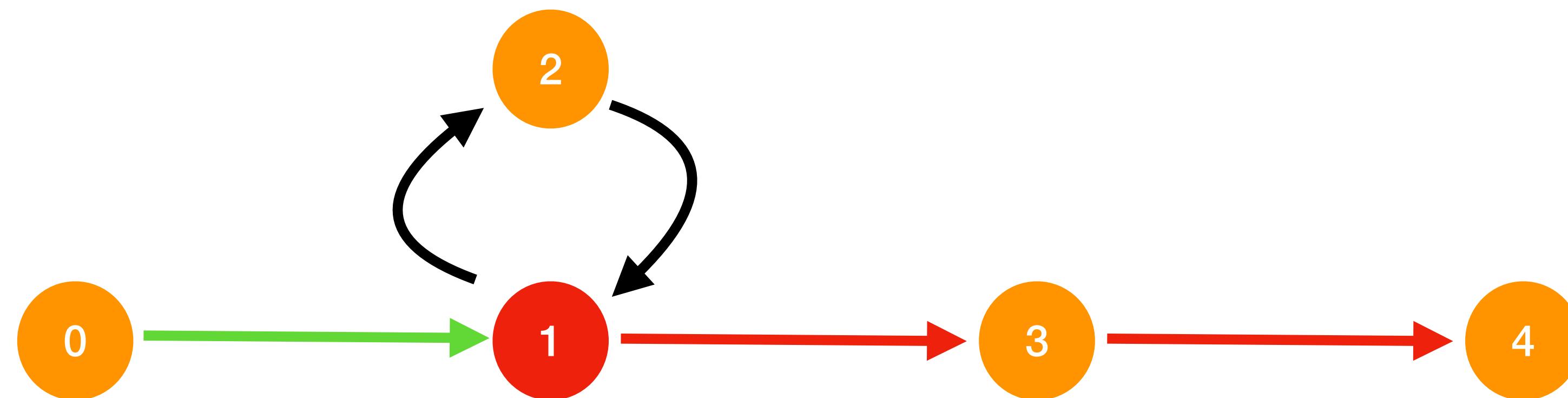


Modifiquem el DFS tal que quan fem el backtracking (no queden nodes adjacents sense visitar) afegim el node dins la solució. **El resultat seria: [4,3]**

# Exemple - Trobar el Camí Eulerià

- Hem de modificar el DFS per tal que forcem a visitar totes les Areastes

Per il·lustrar el problema, considerem el següent graf on el node d'inici es el node 0

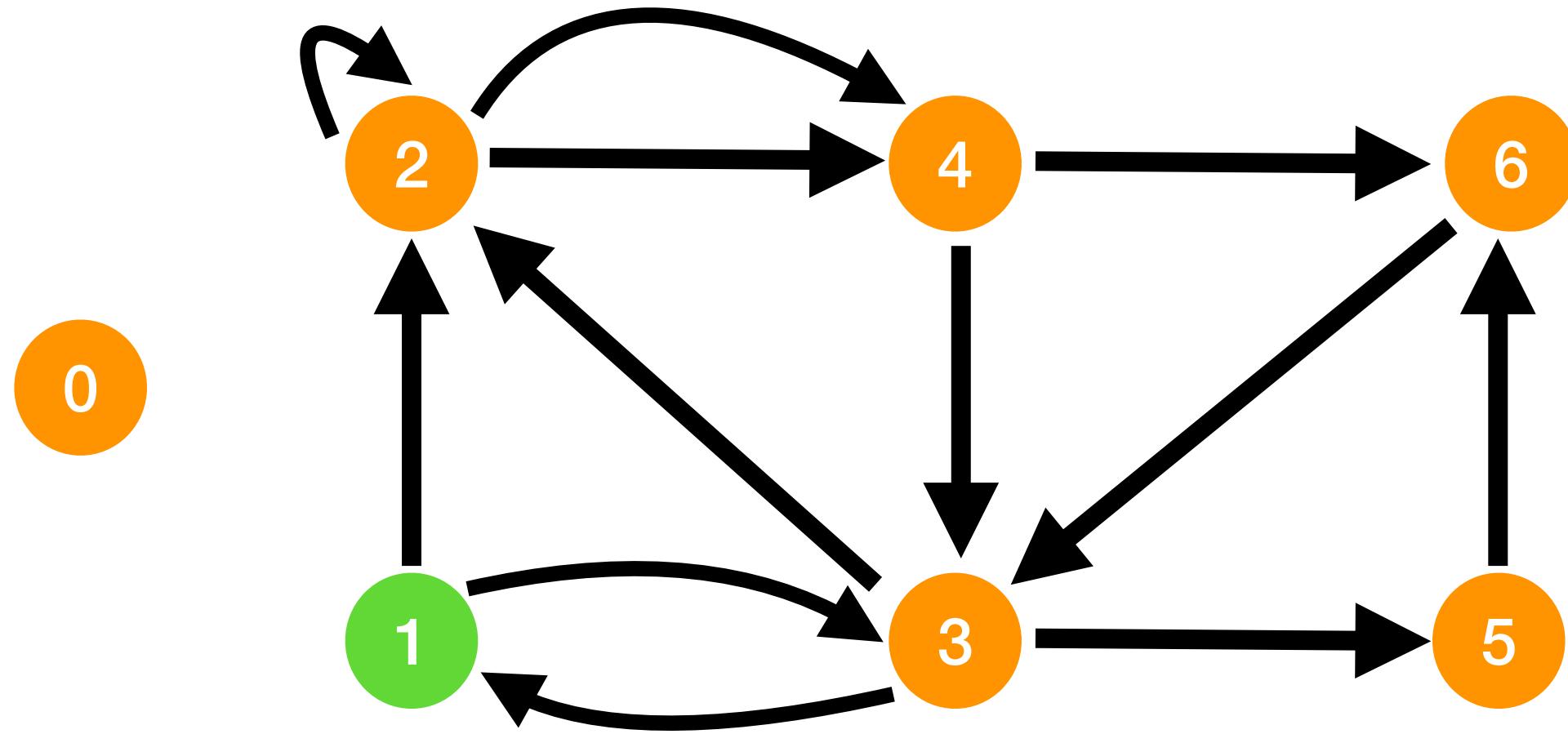


Modifiquem el DFS tal que quan fem el backtracking (no queden nodes adjacents sense visitar) afegim el node dins la solució. **El resultat seria:** [4,3]

Si al fer backtracking, ens trobem amb un node que té fills sense explorar, els explorem utilitzant el DFS i afegim els nodes a la solució com em vist abans

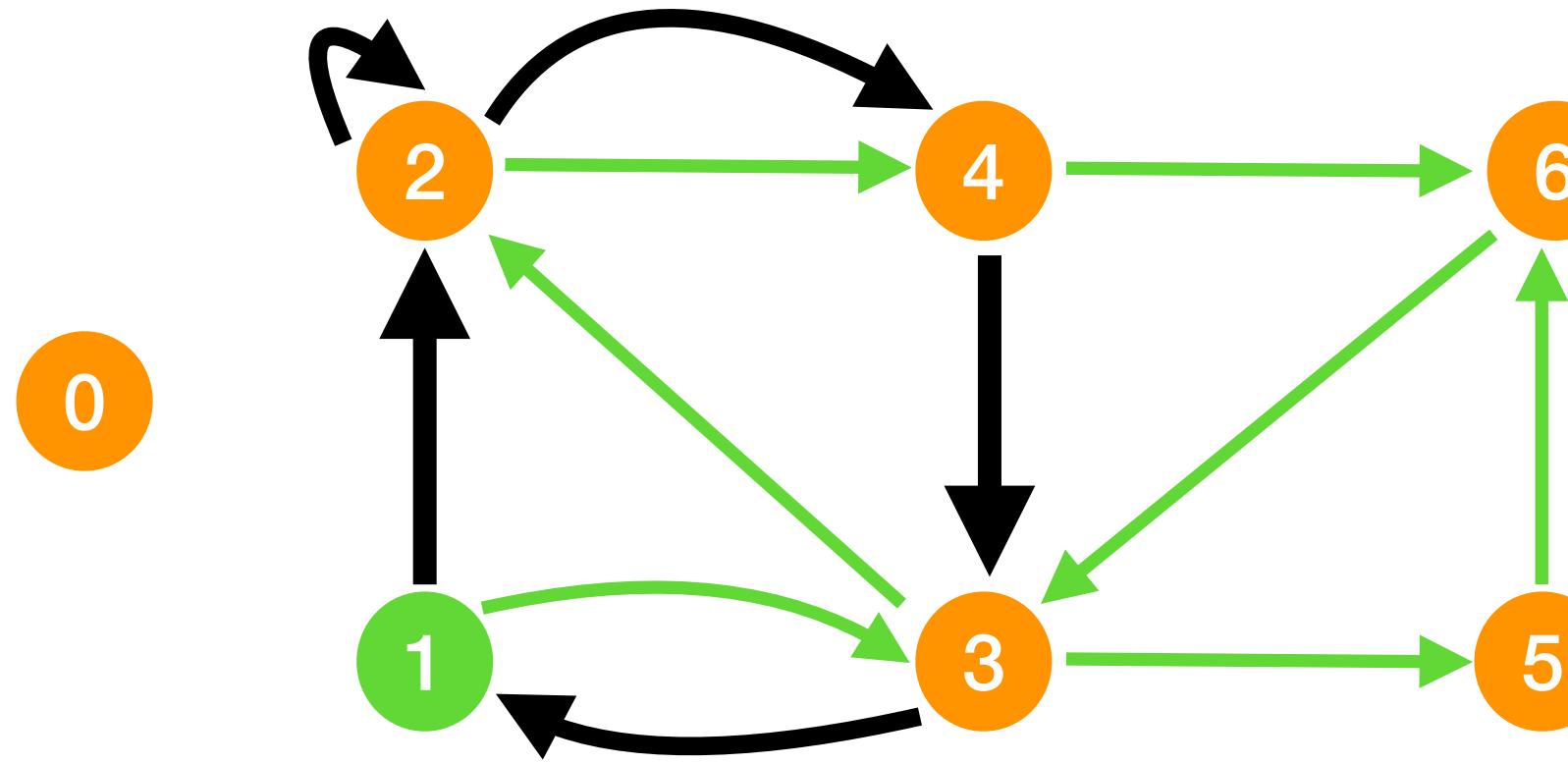
**El resultat seria:** [4,3,1,2,1,0]

# Exemple - Trobar el Camí Eulerià



Node	In	Out
0	0	0
1	1	2
2	3	3
3	3	3
4	2	2
5	1	1
6	2	1

Executem DFS i fem la següent exploració:  
1, 3, 5, 6, 3, 2, 4, 6



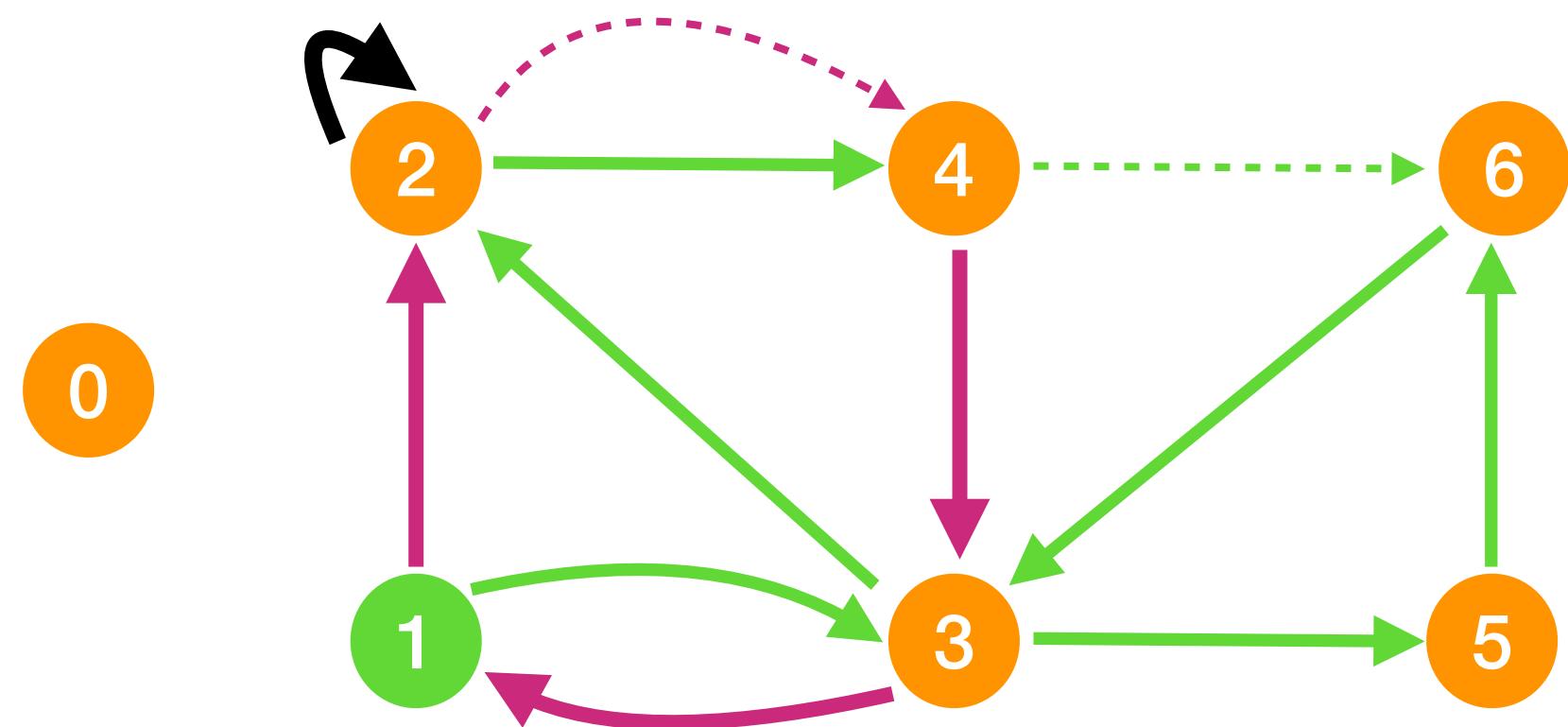
Al mateix temps que visitem els nodes, reduïm el seu out-degree

Node	Out
0	0
1	1
2	2
3	1
4	1
5	1
6	0

# Exemple - Trobar el Camí Eulerià

Executem DFS i fem la següent exploració:

1, 3, 5, 6, 3, 2, 4, 6



Quan el DFS no té més nodes per explorar (es, a dir, ens trobem un node on  $\text{out}[i]=0$ ), fem backtracking i afegim el node a la solució



Solució: [6]



Tornem al node 4, i veiem que aquest node té arestes sense visitar ( $\text{out}[i]>0$ ). Cridem el DFS a partir del node 4

Node	Out
0	0
1	1
2	2
3	1
4	1
5	1
6	0

Executem DFS (4) i fem la següent exploració:

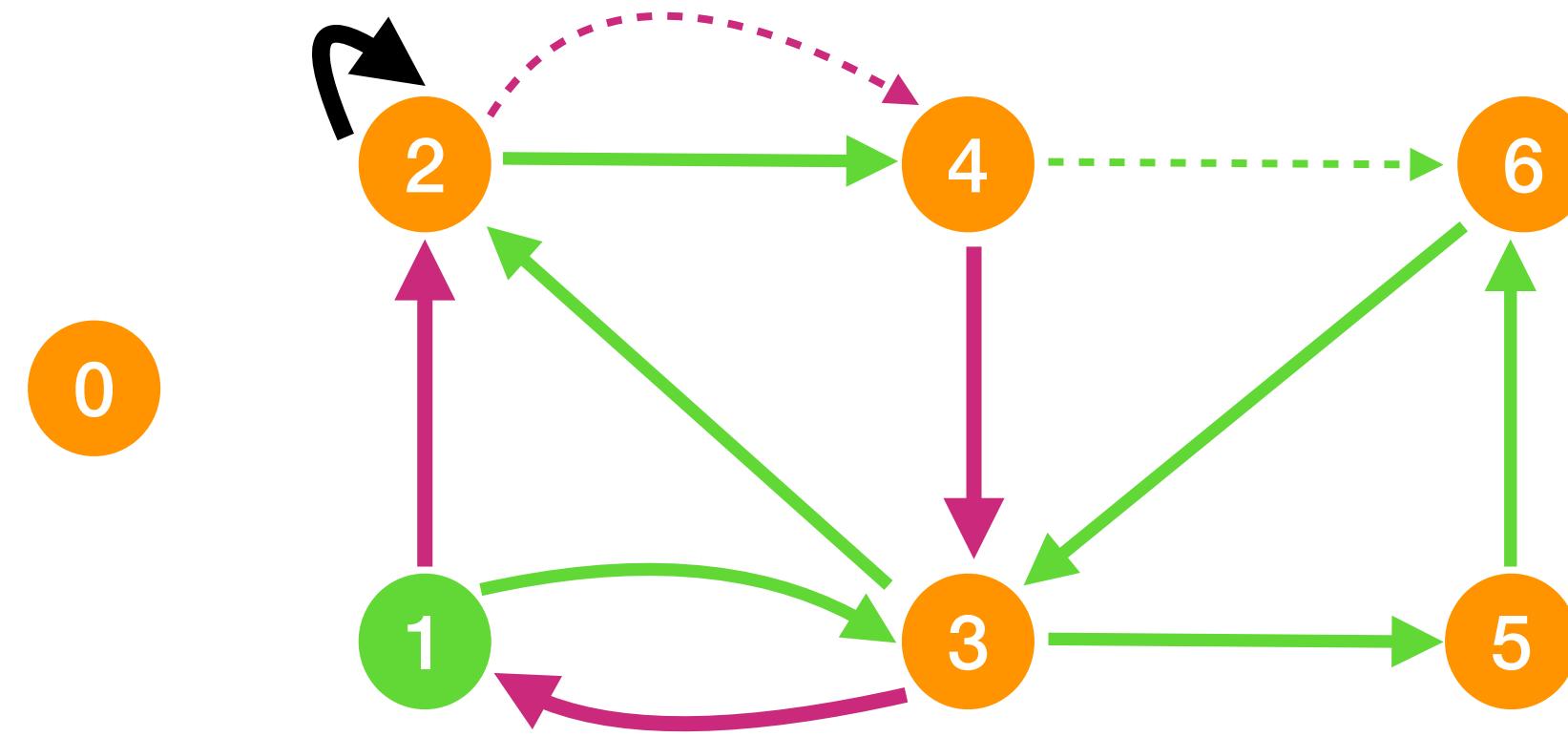
4, 3, 1, 2, 4

Quan el DFS no té més nodes per explorar (es, a dir, ens trobem un node on  $\text{out}[i]=0$ ), fem backtracking i afegim el node a la solució



Solució: [4, 6]

# Exemple - Trobar el Camí Eulerià



Node	Out
0	0
1	1
2	2
3	1
4	1
5	1
6	0

Tornem al node 2, i veiem que aquest node té arestes sense visitar ( $\text{out}[i] > 0$ ). Cridem el DFS a partir del node 2

Executem DFS (2).

No tenim nodes sense visitar, i per tant fem backtracking. Al fer backtracking afegim el node a la sol·lució

Solució: [2, 4, 6]

Continuem el backtracking de la crida del **DFS(4)**

Solució: [4, 3, 1, 2, 2, 4, 6]

Continuem el backtracking de la crida del **DFS(1)**

Solució: [1, 3, 5, 6, 3, 2, 4, 3, 1, 2, 2, 4, 6]

# Com trobar el Camí Eulerià

- Pas 1:
  - Determinar si existeix un camí eulerià.
- Pas 2:
  - Busquem un node d'inici vàlid.
    - Si hi ha 0 vèrtexs imparells, comencem en qualsevol lloc.
    - Si hi ha 2 vèrtexs senars, comenceu per un d'ells.
- Pas 3
  - Executem la versió modificada del DFS

```

# Global/class scope variables
n = number of vertices in the graph
m = number of edges in the graph
g = adjacency list representing directed graph

in  = [0, 0, ..., 0, 0] # Length n
out = [0, 0, ..., 0, 0] # Length n

path = empty integer linked list data structure

function findEulerianPath():

    countInOutDegrees()
    if not graphHasEulerianPath(): return null

    dfs(findStartNode())

    # Return eulerian path if we traversed all the
    # edges. The graph might be disconnected, in which
    # case it's impossible to have an euler path.
    if path.size() == m+1: return path
return null

```

```
function countInOutDegrees():
    for edges in g:
        for edge in edges:
            out[edge.from]++
            in[edge.to]++

function graphHasEulerianPath():
    start_nodes, end_nodes = 0, 0
    for (i = 0; i < n; i++):
        if (out[i] - in[i]) > 1 or (in[i] - out[i]) > 1:
            return false
        else if out[i] - in[i] == 1:
            start_nodes++
        else if in[i] - out[i] == 1:
            end_nodes++
    return (end_nodes == 0 and start_nodes == 0) or
           (end_nodes == 1 and start_nodes == 1)
```

```
function findStartNode():
    start = 0
    for (i = 0; i < n; i = i + 1):
        # Unique starting node
        if out[i] - in[i] == 1: return i

        # Start at any node with an outgoing edge
        if out[i] > 0: start = i
    return start

function dfs(at):
    # While the current node still has outgoing edges
    while (out[at] != 0):

        # Select the next unvisited outgoing edge
        next_edge = g[at].get(--out[at])
        dfs(next_edge.to)

        # Add current node to solution
        path.insertFirst(at)
```

# **Exercici:** Quina és la complexitat de l'algoritme anterior?