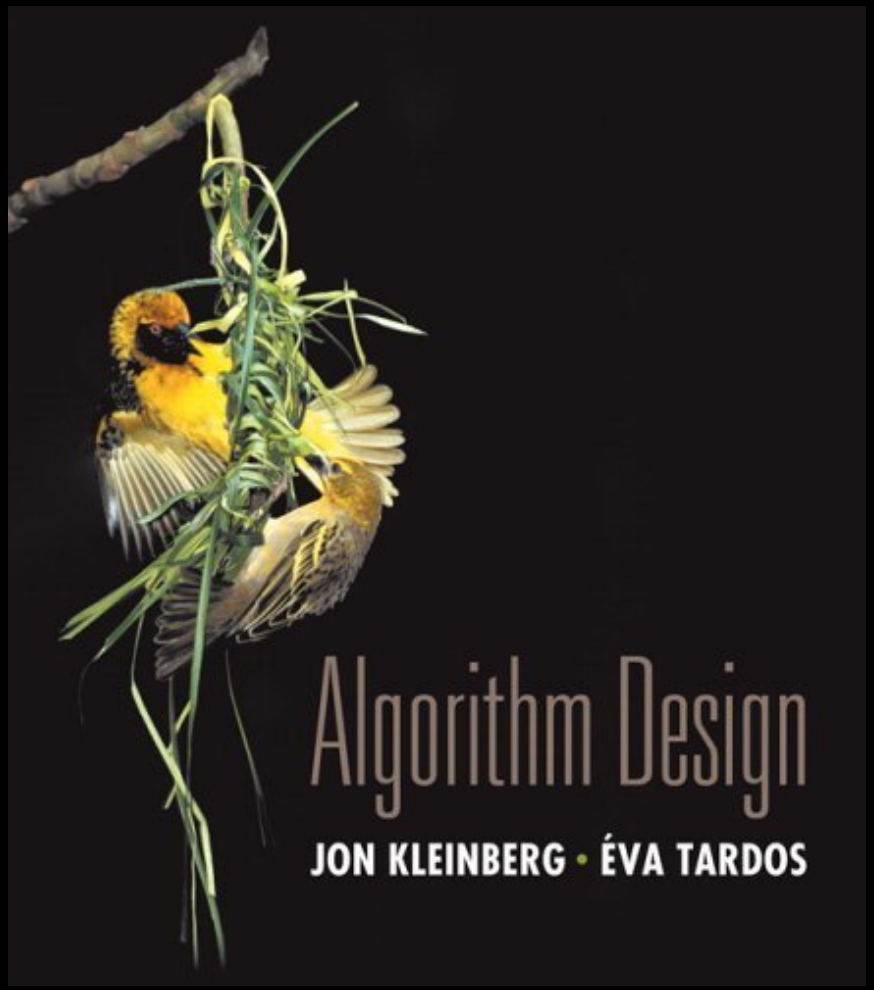


Chapter 7

Network Flow

PEARSON
Addison Wesley

Rete ferroviaria sovietica

Hanno analizzato problemi di trasporto relati alla rete ferroviaria sovietica:

- A.N. Tolstoi, Methods of finding the minimal total kilometrage in cargo-transportation planning in space, 1930, (in russo)
- T.E. Harris, F.S. Ross, Fundamentals of a Method for Evaluating Rail Net Capacities, Research Memorandum RM-1573, The RAND Corporation, Santa Monica, California, 1955.
  - Scritto per US Air Force, dapprima "classified" e poi "unclassified" il 21 maggio 1999.
  - I nodi rappresentano piccole regioni connesse con archi
  - Capacità di una connessione: tonnellate che possono essere trasportate in un giorno
  - Interesse degli US: calcolo dei "bottleneck" per disconnettere gli stati satellite

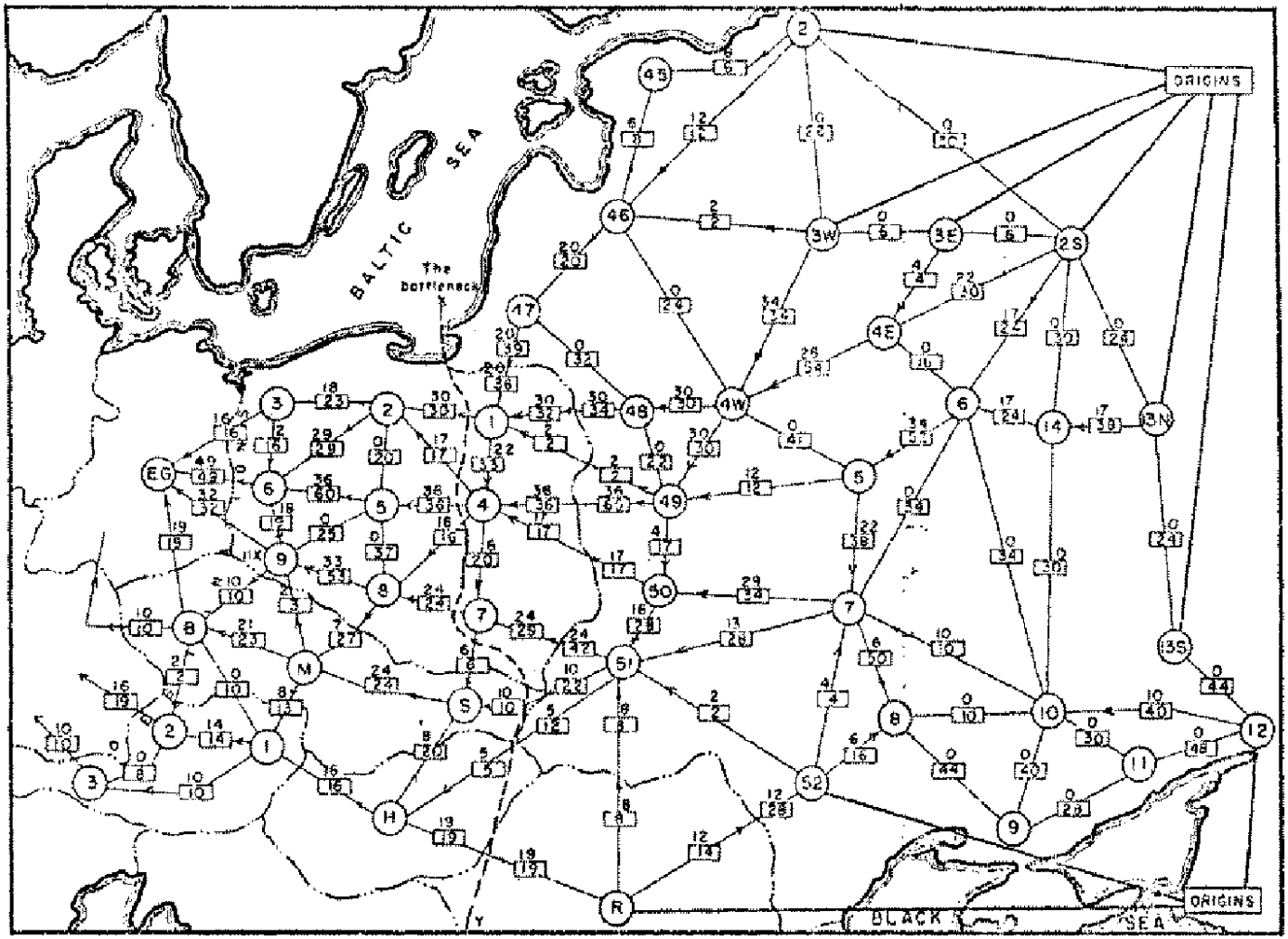
On the history of the transportation and maximum flow problems

Alexander Schrijver[1]

Abstract. We review two papers that are of historical interest for combinatorial optimization: an article of A.N. Tolstoĭ from 1930, in which the transportation problem is studied, and a negative cycle criterion is developed and applied to solve a (for that time) large-scale (10×68) transportation problem to optimality; and an, until recently secret, RAND report of T.E. Harris and F.S. Ross from 1955, that Ford and Fulkerson mention as motivation to study the maximum flow problem. The papers have in common that they both apply their methods to the Soviet railway network.

in Math Programming, 91: 3, 2002.

Rete ferroviaria sovietica, 1955



Harris e Ross: Max flow di 163.000 tonnellate dalla Russia all'Europa dell'Est e taglio di capacità 163.000 tonnellate indicato come "The bottleneck"

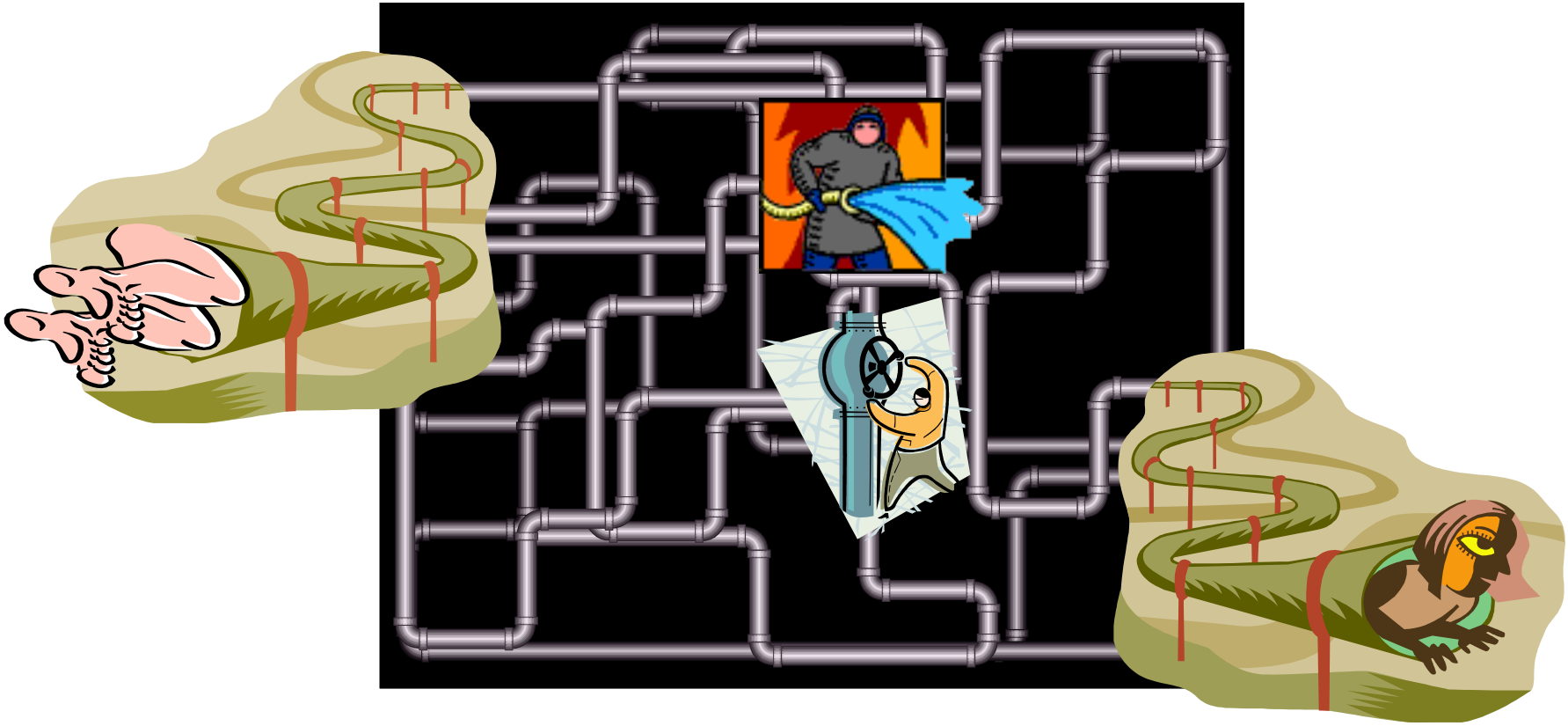
Riferimento: *On the history of the transportation and maximum flow problems.* Alexander Schrijver in Math Programming, 91: 3, 2002.

Altri esempi di network

- Liquidi attraverso tubi
- Parti tra le linee di assemblaggio
- Corrente attraverso una rete elettrica
- Informazione in reti di comunicazioni
- Trasporto di merci su strada
- …

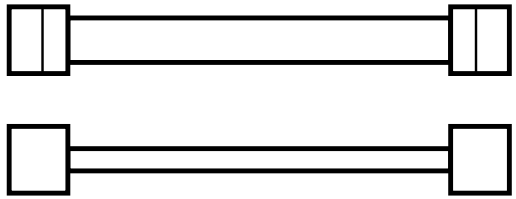
Network Flow

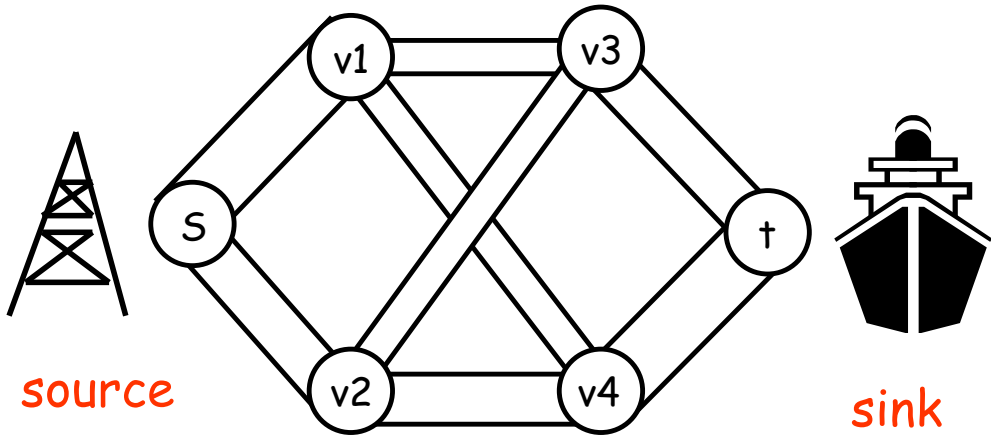
- Un network è un grafo diretto
- Archi rappresentano tubi che trasportano liquido
- Ogni arco ha una capacità (tubi di grandezze diverse)
- Un nodo sorgente
- Un nodo pozzo



Network Flow

- Un network è un grafo diretto
- Archi rappresentano tubi che trasportano liquido
- Ogni arco ha una capacità (tubi di grandezze diverse)
- Un nodo sorgente
- Un nodo pozzo

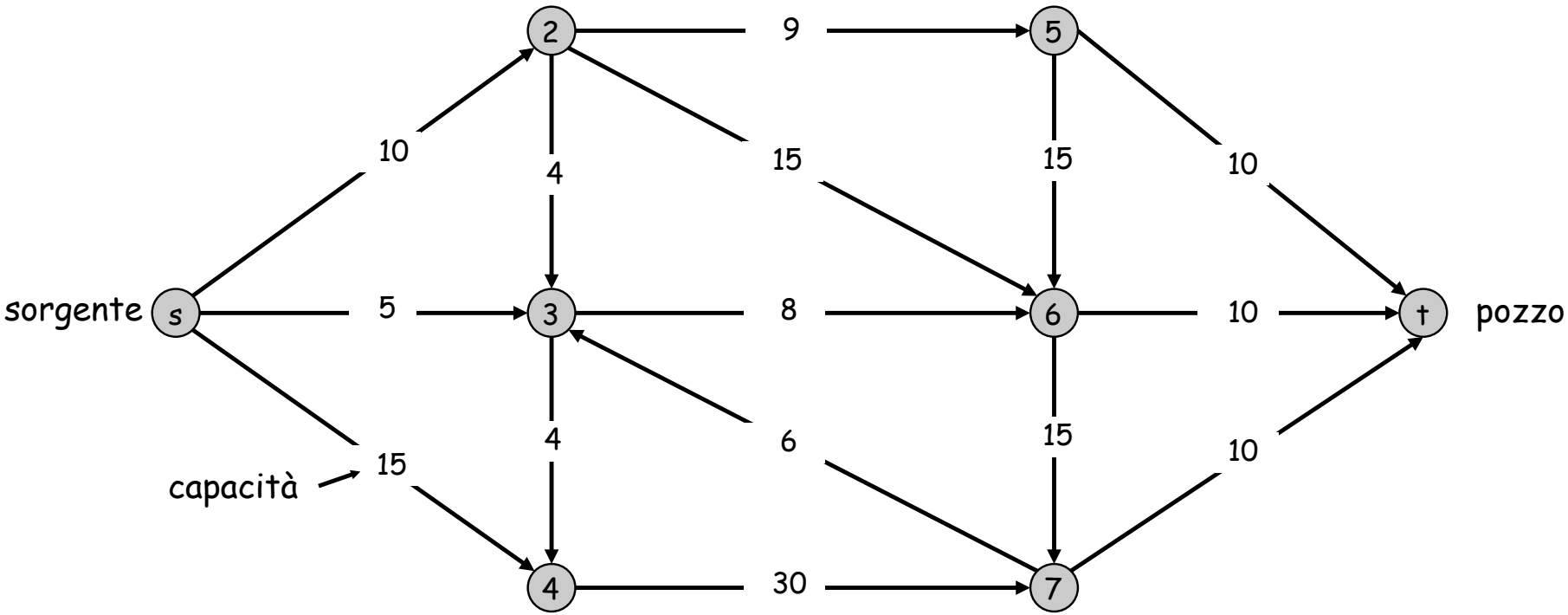




Problema Taglio Minimo

Rete dei flussi.

- Astrazione per flusso materiali sugli archi.
- $G = (V, E)$ grafo diretto, senza archi "paralleli".
- Due nodi distinti: s = sorgente, t = pozzo.
- $c(e)$ = capacità dell'arco e.



Massimo Flusso e Minimo Taglio

Massimo Flusso e Minimo Taglio.

- Due problematiche algoritmiche molto ricche.
- Importante problema nell'ottimizzazione combinatoriale.
- Bella dualità matematica.

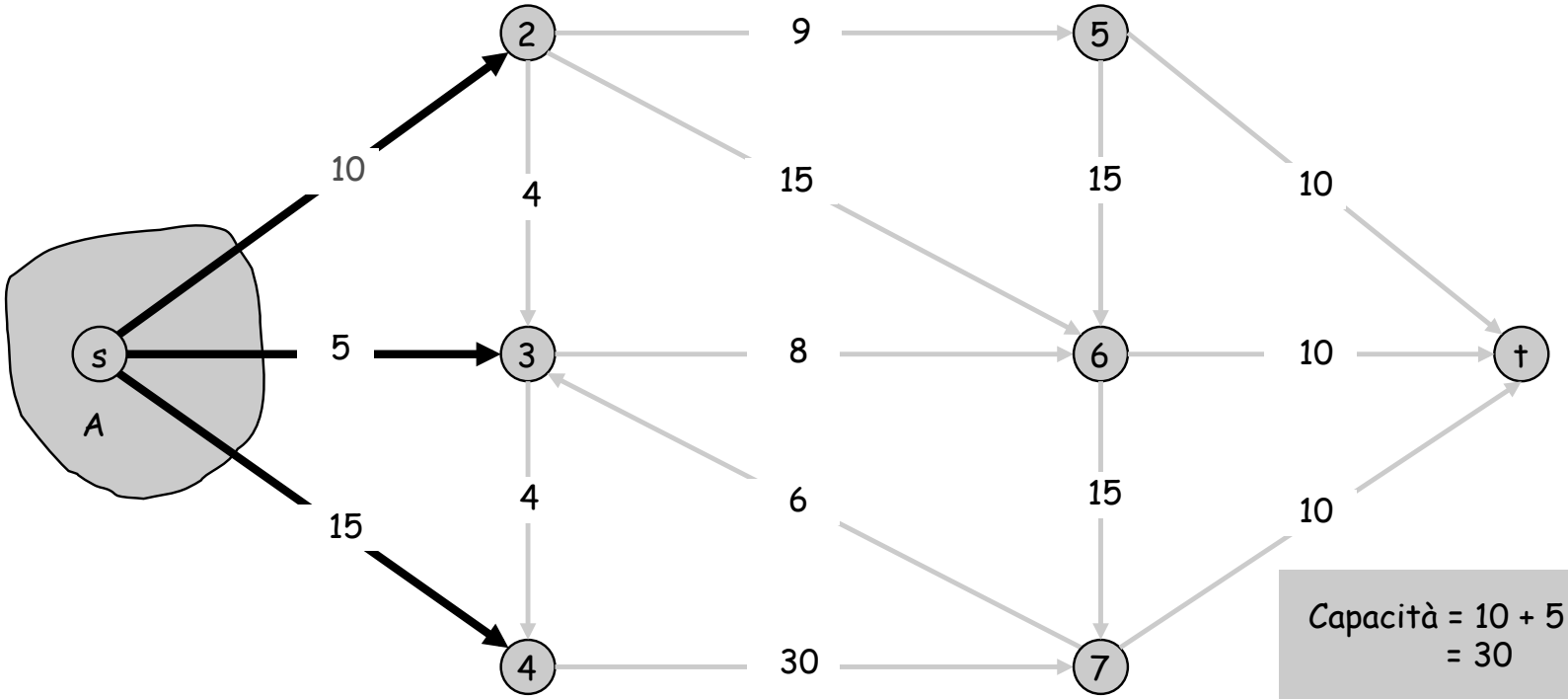
Aplicazioni non banali / riduzioni

- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more . . .

Taglio

Definizione. Un taglio s-t è una partizione (A, B) di V con $s \in A$ e $t \in B$.

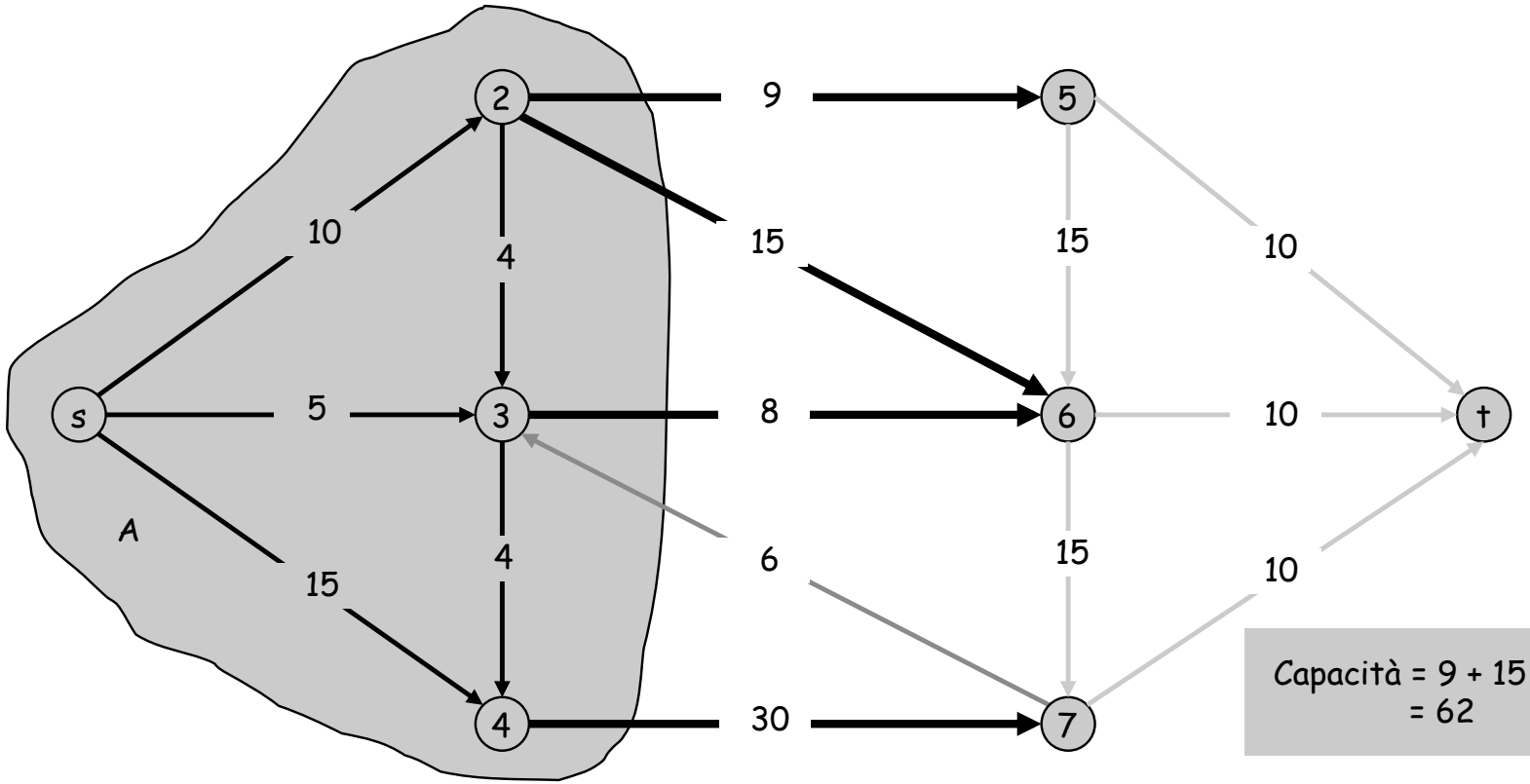
Definizione. La capacità di un taglio (A, B) è: $cap(A, B) = \sum_{e \text{ esce da } A} c(e)$



Taglio

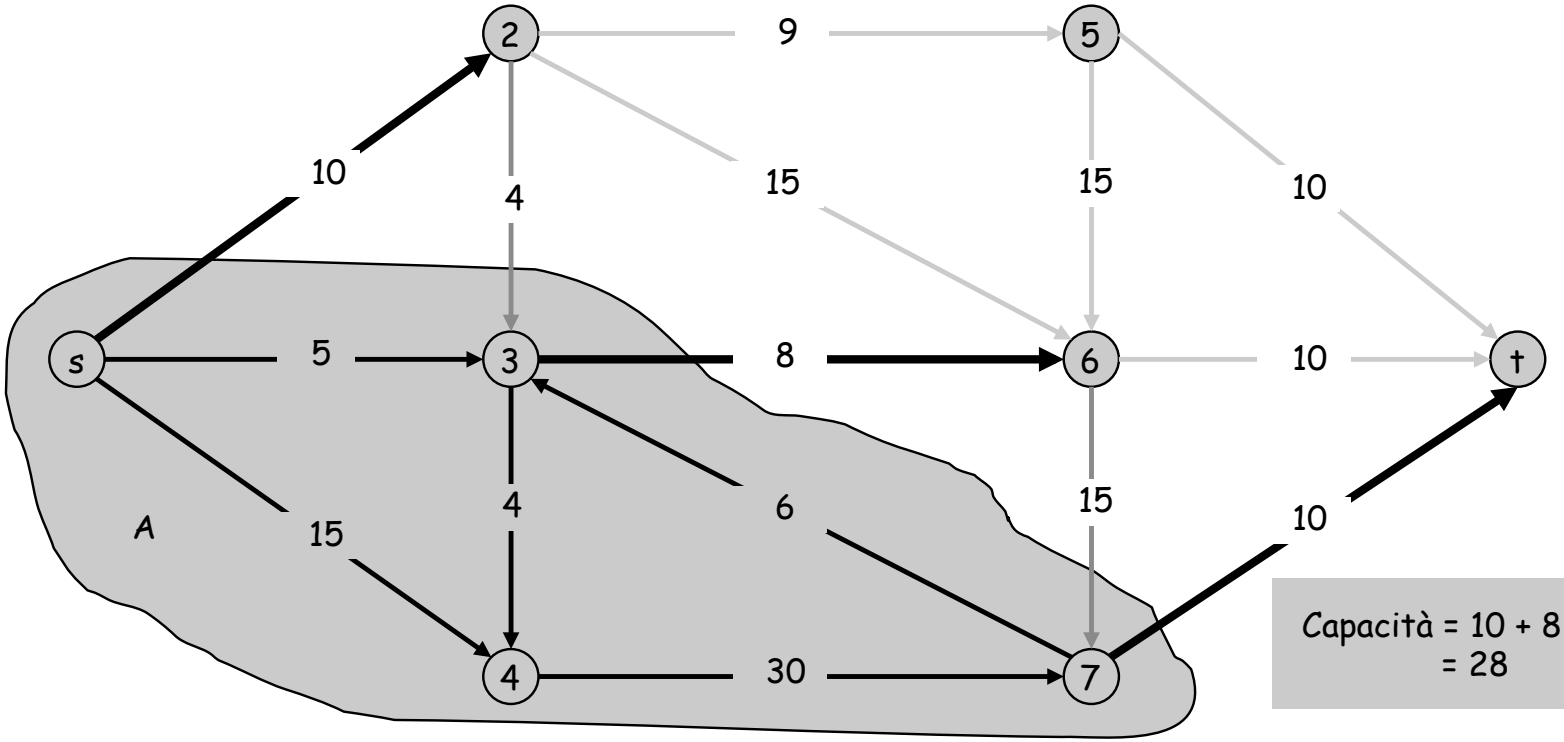
Definizione. Un taglio s-t è una partizione (A, B) di V con $s \in A$ e $t \in B$.

Definizione. La capacità di un taglio (A, B) è: $cap(A, B) = \sum_{e \text{ esce da } A} c(e)$



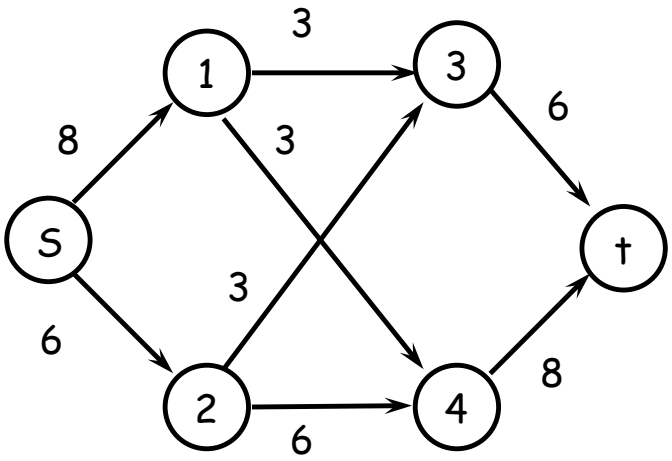
Problema Taglio Minimo

Problema taglio s-t minimo. Trovare un taglio s-t di capacità minima.

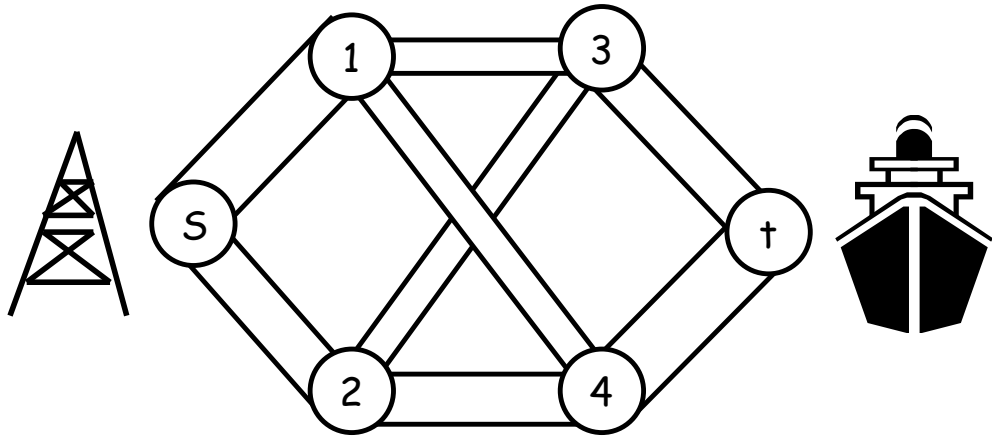


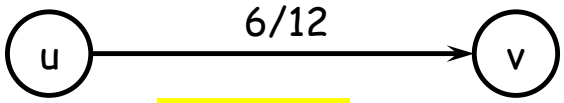
Flusso

Grafo G=(V,E)

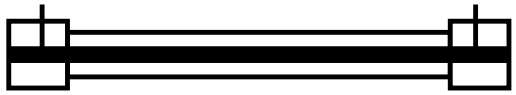


Esempio: oil pipeline

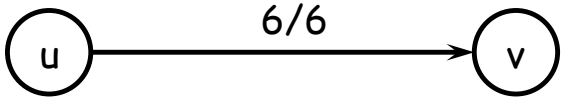




f(u,v)=6



Flusso inferiore alla capacità

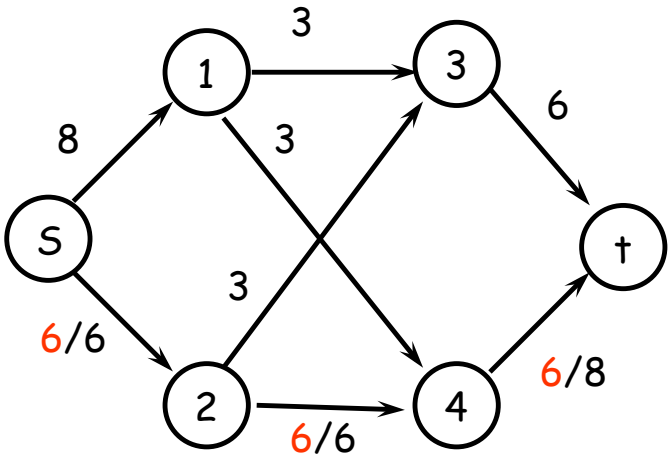


f(u,v)=6

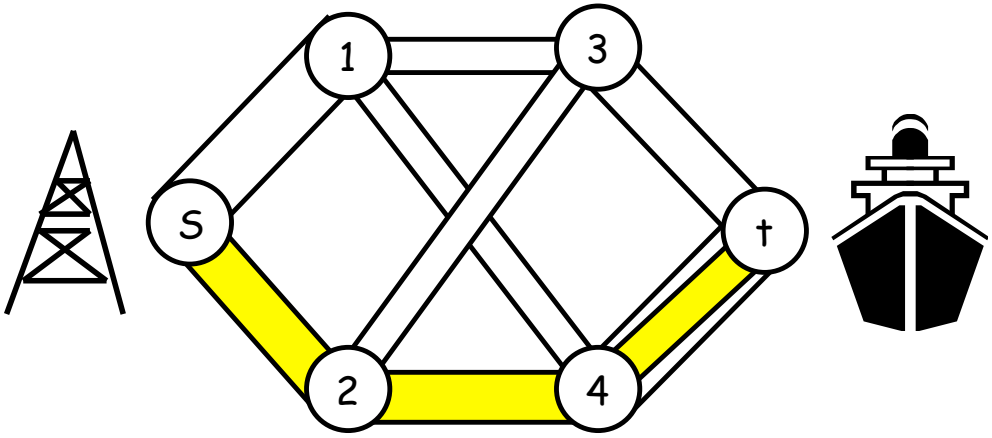
Massimo flusso

Flusso

Grafo G=(V,E)



Esempio: oil pipeline

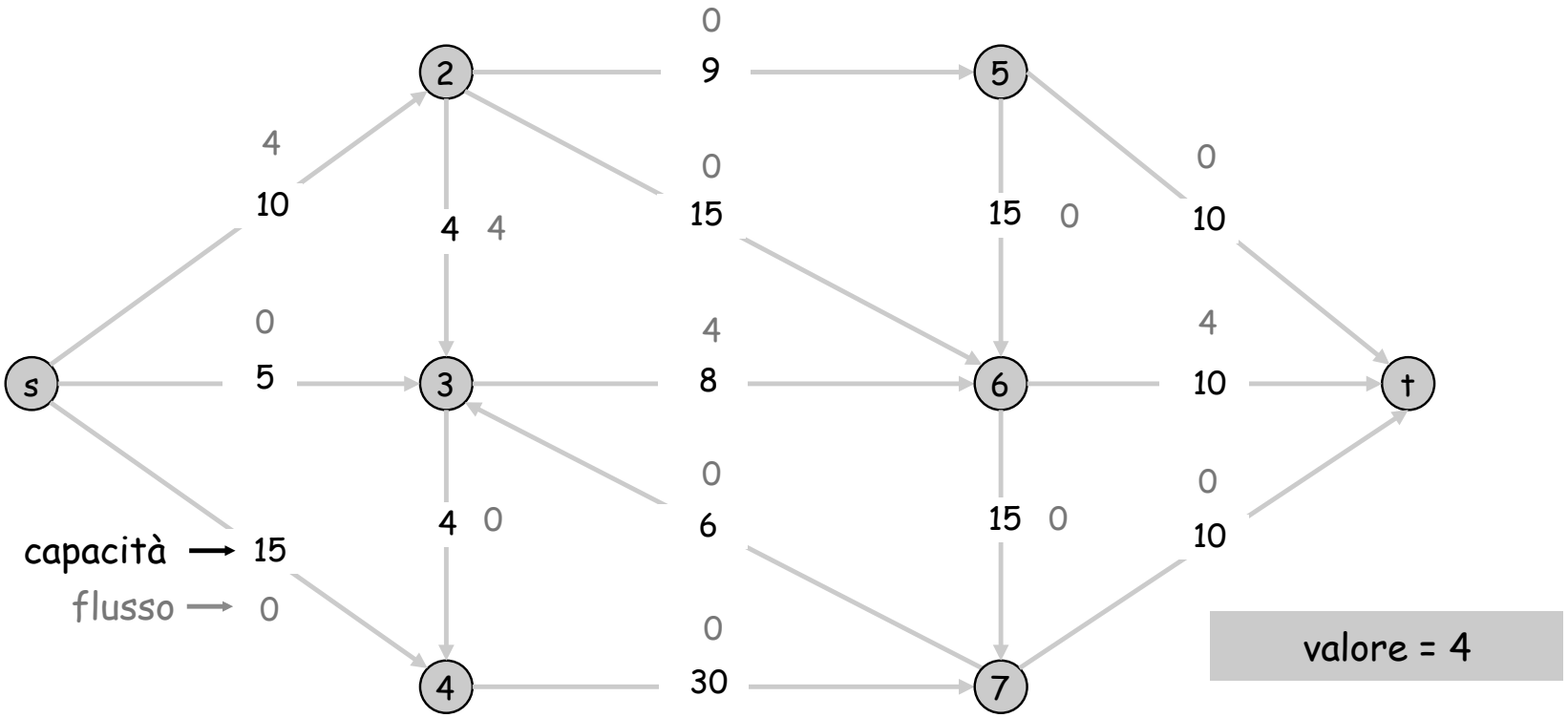


Flusso

Definizione. Un **flusso s-t** è una funzione che soddisfa:

- Per ogni $e \in E$: $0 \le f(e) \le c(e)$ (capacità)
- Per ogni $v \in V - \{s, t\}$: $\sum_{e \text{ entra in } v} f(e) = \sum_{e \text{ esce da } v} f(e)$ (conservazione)

Definizione. Il **valore** di un flusso è: $v(f) = \sum_{e \text{ esce da } s} f(e)$.

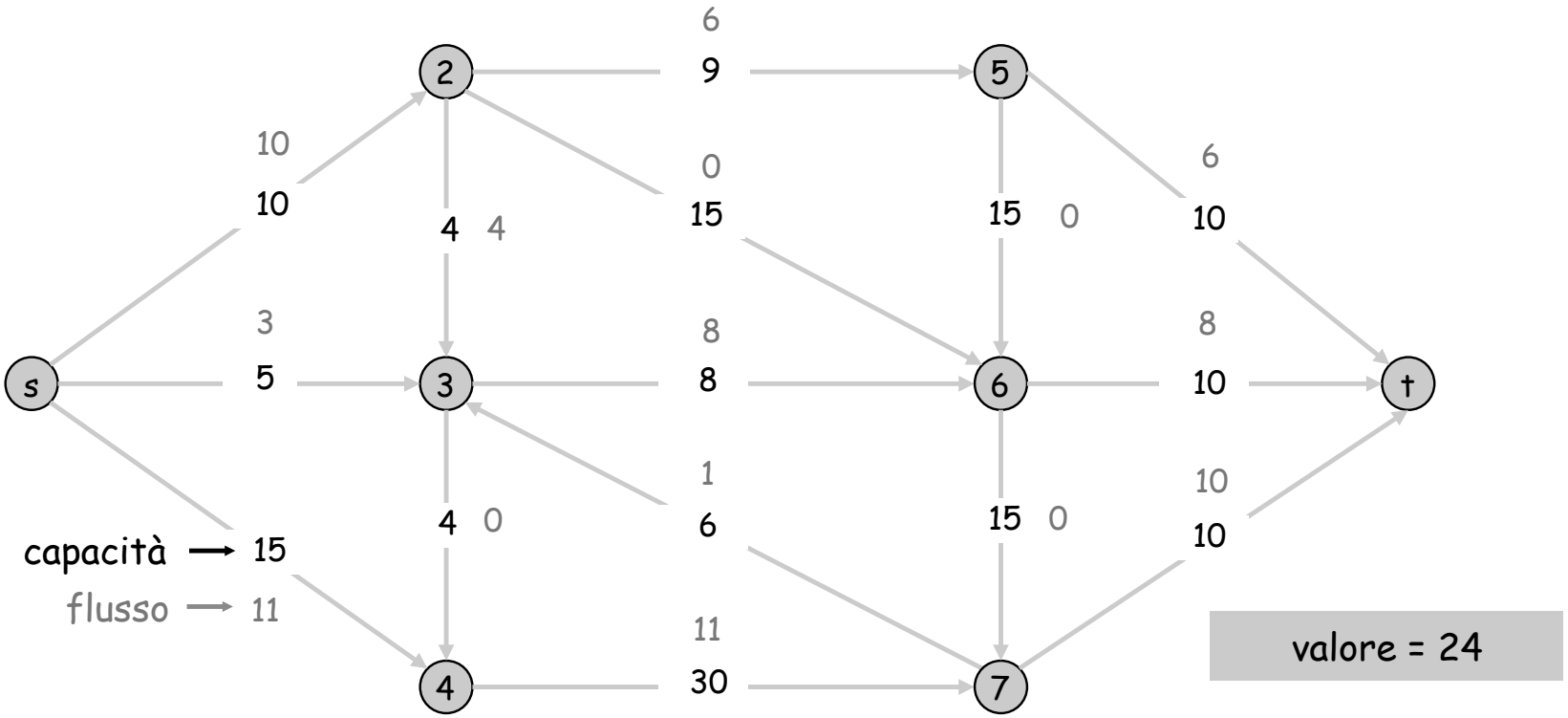


Flusso

Definizione. Un **flusso s-t** è una funzione che soddisfa:

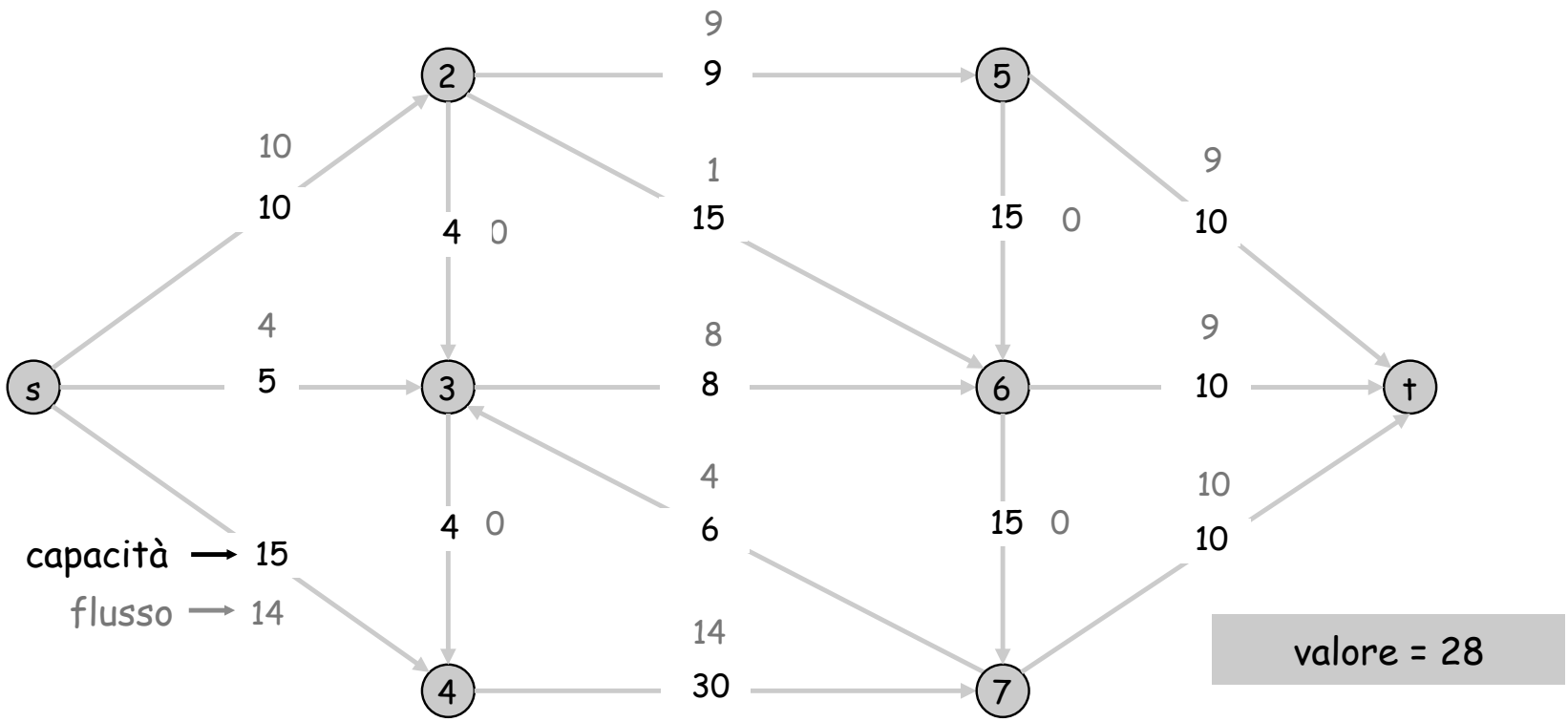
- Per ogni $e \in E$: $0 \le f(e) \le c(e)$ (capacità)
- Per ogni $v \in V - \{s, t\}$: $\sum_{e \text{ entra in } v} f(e) = \sum_{e \text{ esce da } v} f(e)$ (conservazione)

Definizione. Il **valore** di un flusso è: $v(f) = \sum_{e \text{ esce da } s} f(e)$.



Problema Flusso Massimo

Problema Flusso Massimo. Trovare flusso s-t con il massimo valore.



Flussi e Tagli

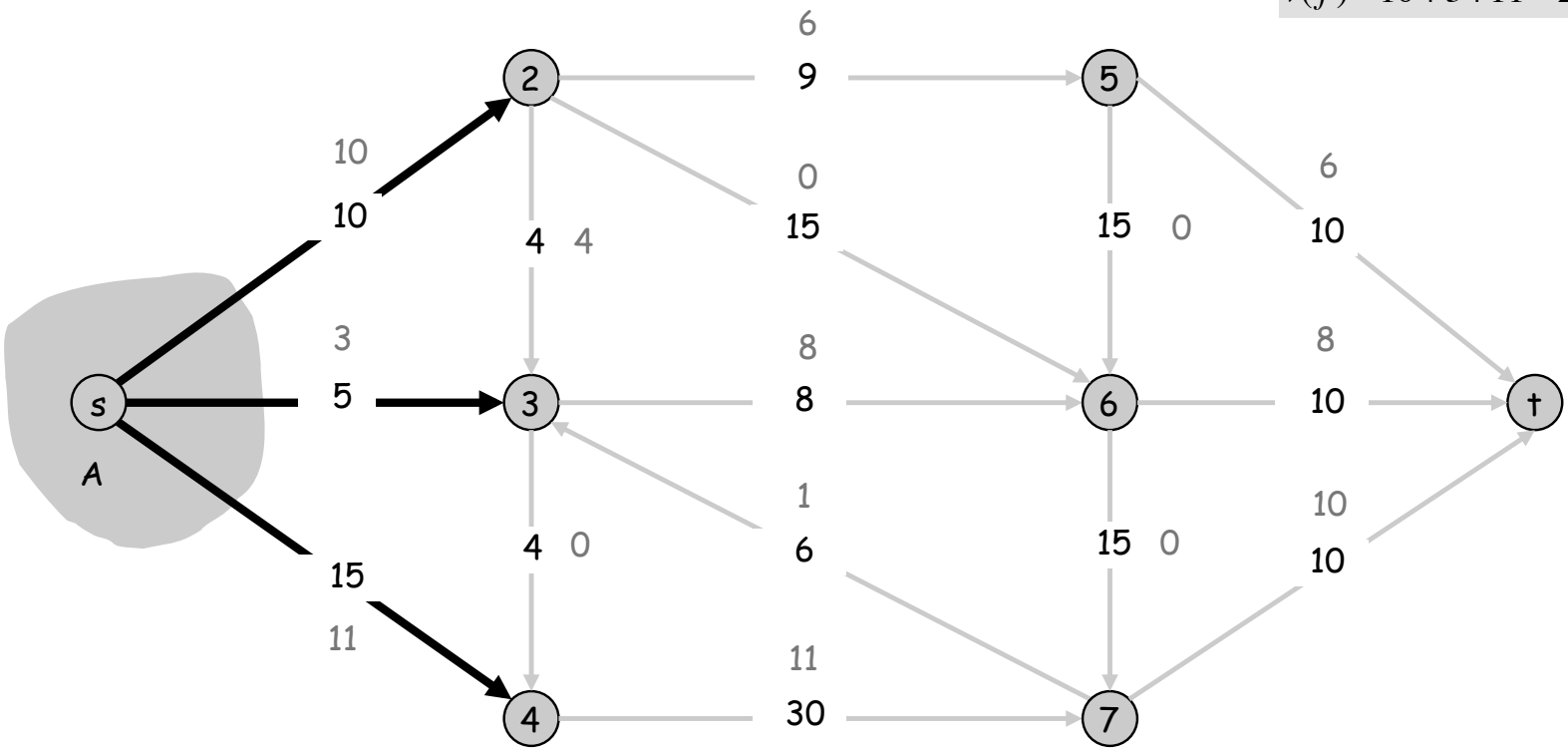
Lemma valore flusso. Sia f un flusso, e sia (A, B) un taglio s-t. Allora, il flusso netto inviato attraverso il taglio è uguale all'ammontare che lascia s.

$$\sum_{e \text{ esce da } A} f(e) \; - \sum_{e \text{ entra in } A} f(e) \; = \; v(f)$$

$$\sum_{e \text{ esce da } A} f(e) = 10 + 3 + 11 = 24$$

$$\sum_{e \text{ entra in } A} f(e) = 0$$

$$v(f) = 10 + 3 + 11 = 24$$



Flussi e Tagli

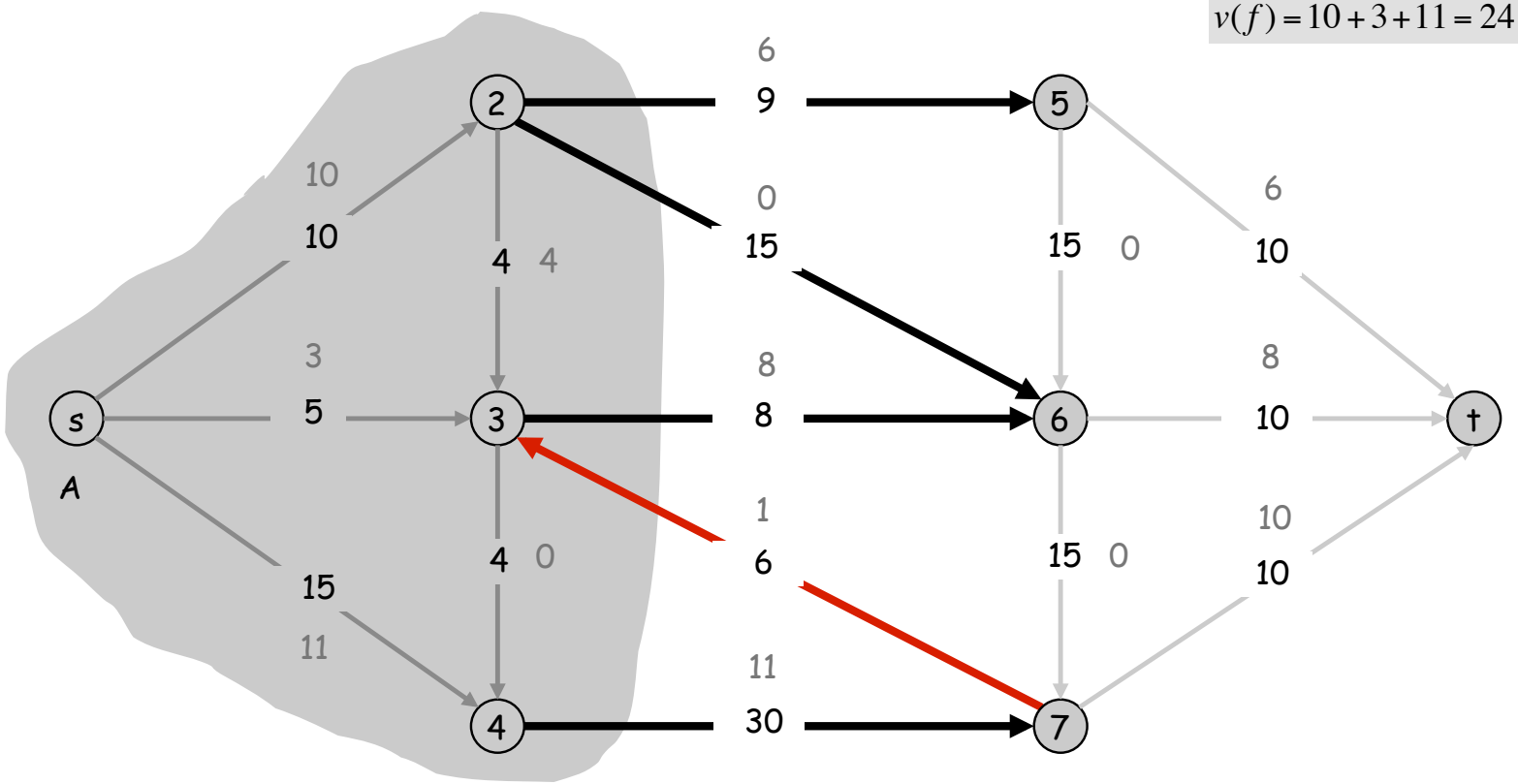
Lemma valore flusso. Sia f un flusso, e sia (A, B) un taglio s-t. Allora, il flusso netto inviato attraverso il taglio è uguale all' ammontare che lascia s.

$$\sum_{e \text{ esce da } A} f(e) \; - \sum_{e \text{ entra in } A} f(e) \; = \; v(f)$$

$$\sum_{e \text{ esce da } A} f(e) = 6 + 0 + 8 + 11 = 25$$

$$\sum_{e \text{ entra in } A} f(e) = 1$$

$$v(f) = 10 + 3 + 11 = 24$$



Flussi e Tagli

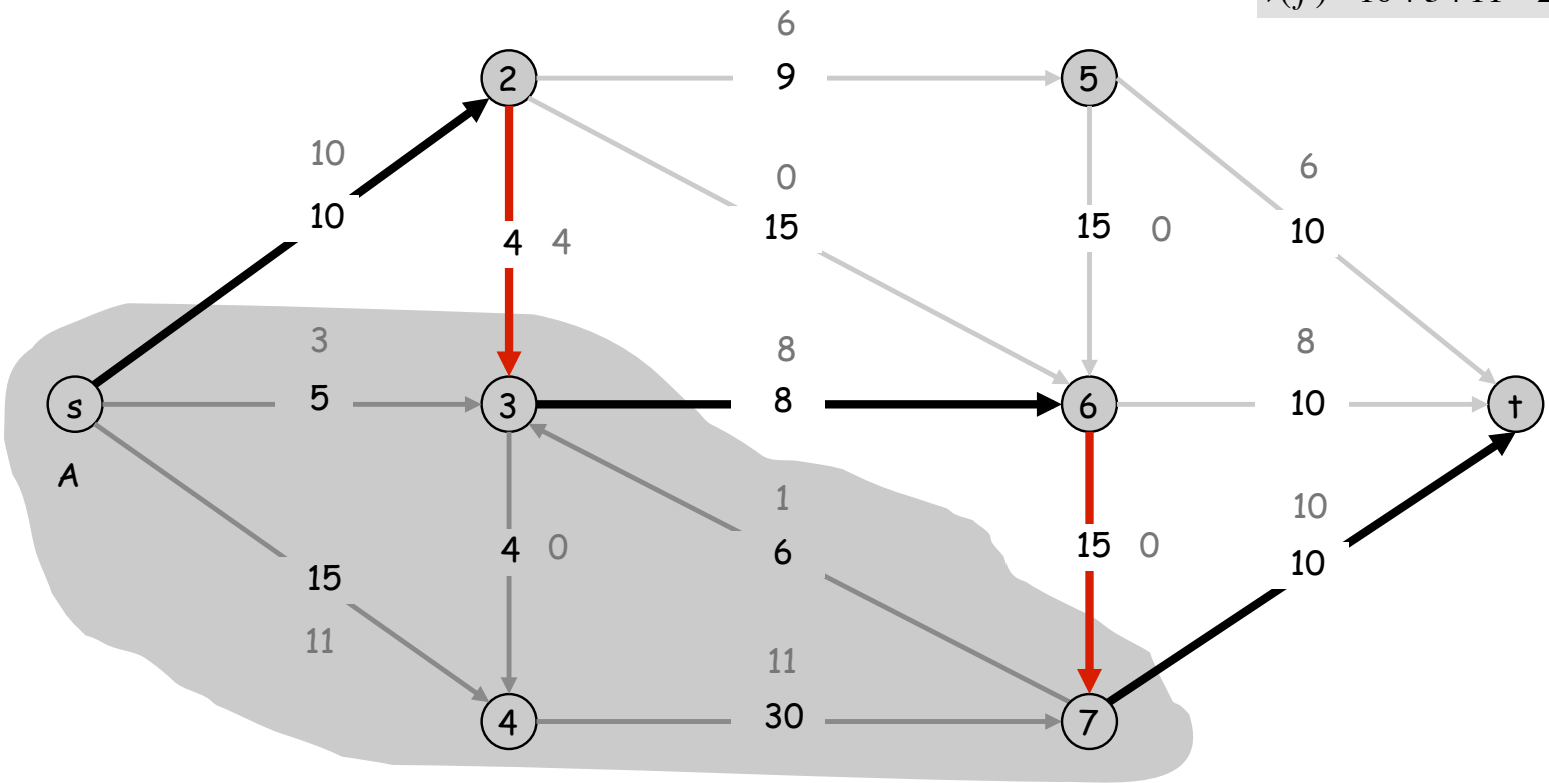
Lemma valore flusso. Sia f un flusso, e sia (A, B) un taglio s-t. Allora, il flusso netto inviato attraverso il taglio è uguale all'ammontare che lascia s.

$$\sum_{e \text{ esce da } A} f(e) \; - \sum_{e \text{ entra in } A} f(e) \; = \; v(f)$$

$$\sum_{e \text{ esce da } A} f(e) = 10 + 8 + 10 = 28$$

$$\sum_{e \text{ entra in } A} f(e) = 4 + 0 = 4$$

$$v(f) = 10 + 3 + 11 = 24$$



Flussi e Tagli

Lemma valore flusso. Sia f un flusso, e sia (A, B) un taglio s-t. Allora,

$$\sum_{e \text{ esce da } A} f(e) \; - \sum_{e \text{ entra in } A} f(e) \; = \; v(f)$$

Prova.

Ricorda: Un taglio s-t è una partizione (A, B) di V con $s \in A$ e $t \in B$.

$$v(f) \; = \sum_{e \text{ esce da } s} f(e)$$

per la conservazione del flusso, tutti i termini eccetto $v = s$ sono 0

$$= \sum_{v \in A} \left(\sum_{e \text{ esce da } v} f(e) - \sum_{e \text{ entra in } v} f(e) \right)$$

Flussi e Tagli

Lemma valore flusso. Sia f un flusso, e sia (A, B) un taglio s-t. Allora,

$$\sum_{e \text{ esce da } A} f(e) \; - \sum_{e \text{ entra in } A} f(e) \; = \; v(f)$$

Prova.

Ricorda: Un taglio s-t è una partizione (A, B) di V con $s \in A$ e $t \in B$.

$$v(f) = \sum_{e \text{ esce da } s} f(e)$$

per la conservazione del flusso, tutti i termini eccetto $v = s$ sono 0

$$= \sum_{v \in A} \left(\sum_{e \text{ esce da } v} f(e) - \sum_{e \text{ entra in } v} f(e) \right)$$

$$= \sum_{v \in A} \sum_{e \text{ esce da } v} f(e) - \sum_{v \in A} \sum_{e \text{ entra in } v} f(e)$$

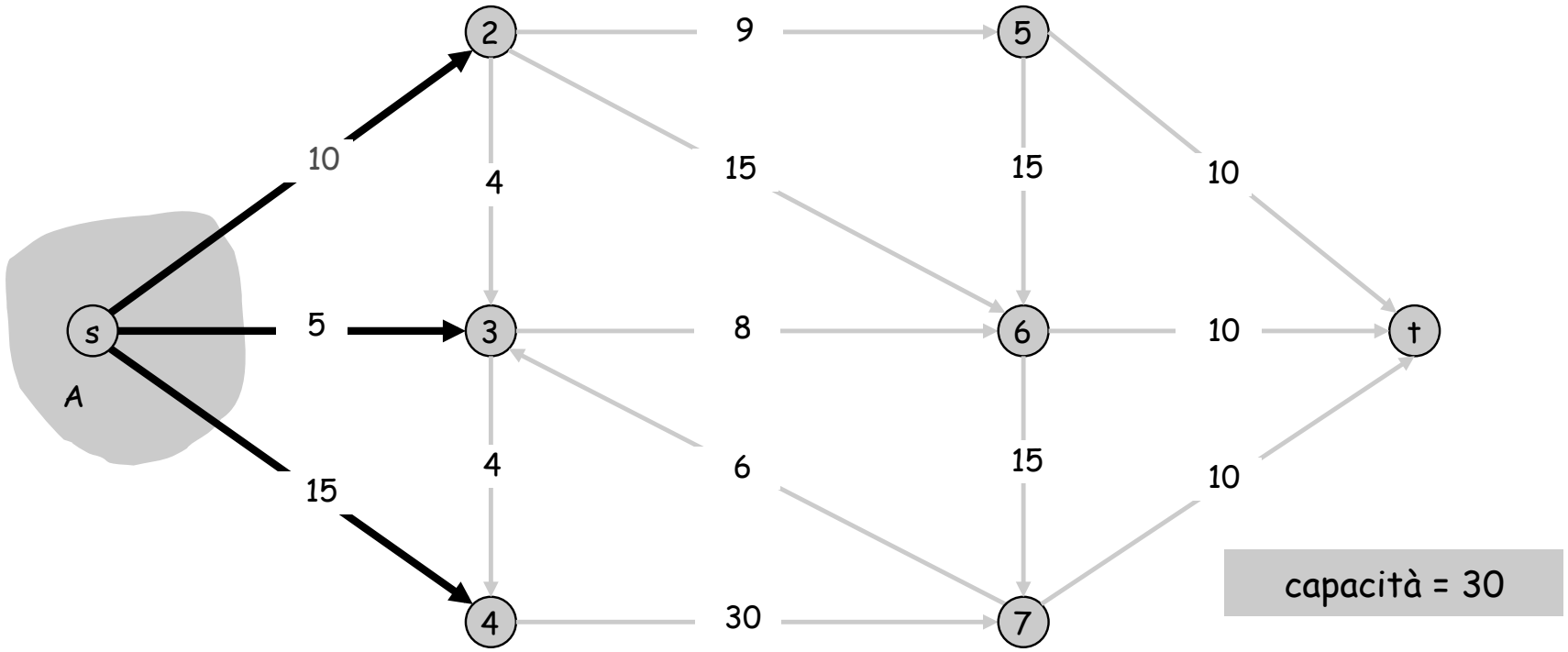
gli archi e che hanno entrambi gli endpoint in A danno contributo 0

$$= \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in A}} f(e).$$

Flussi e Tagli

Dualità debole. Sia f un flusso, e sia (A, B) un taglio s-t. Allora il valore del flusso è al massimo la capacità del taglio.

capacità taglio = 30 ⇒ valore flusso ≤ 30

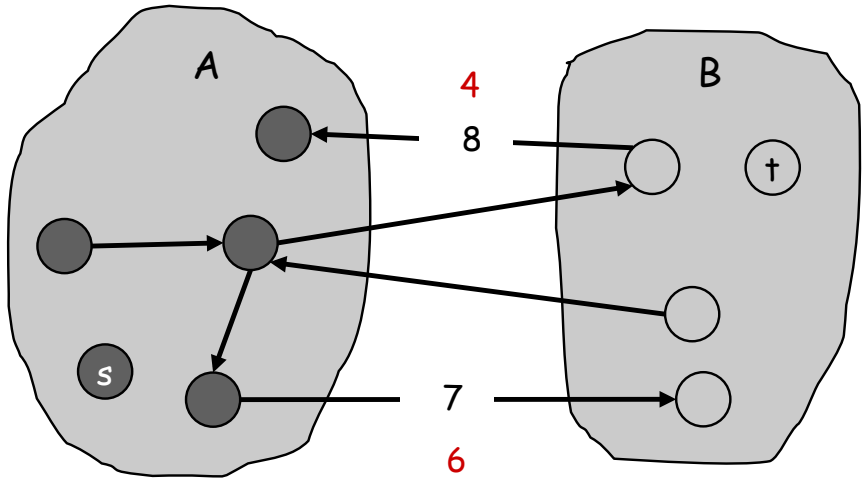


Flussi e Tagli

Dualità debole. Sia f un flusso. Allora, per ogni taglio s-t (A, B) si ha v(f) ≤ cap(A, B).

Prova.

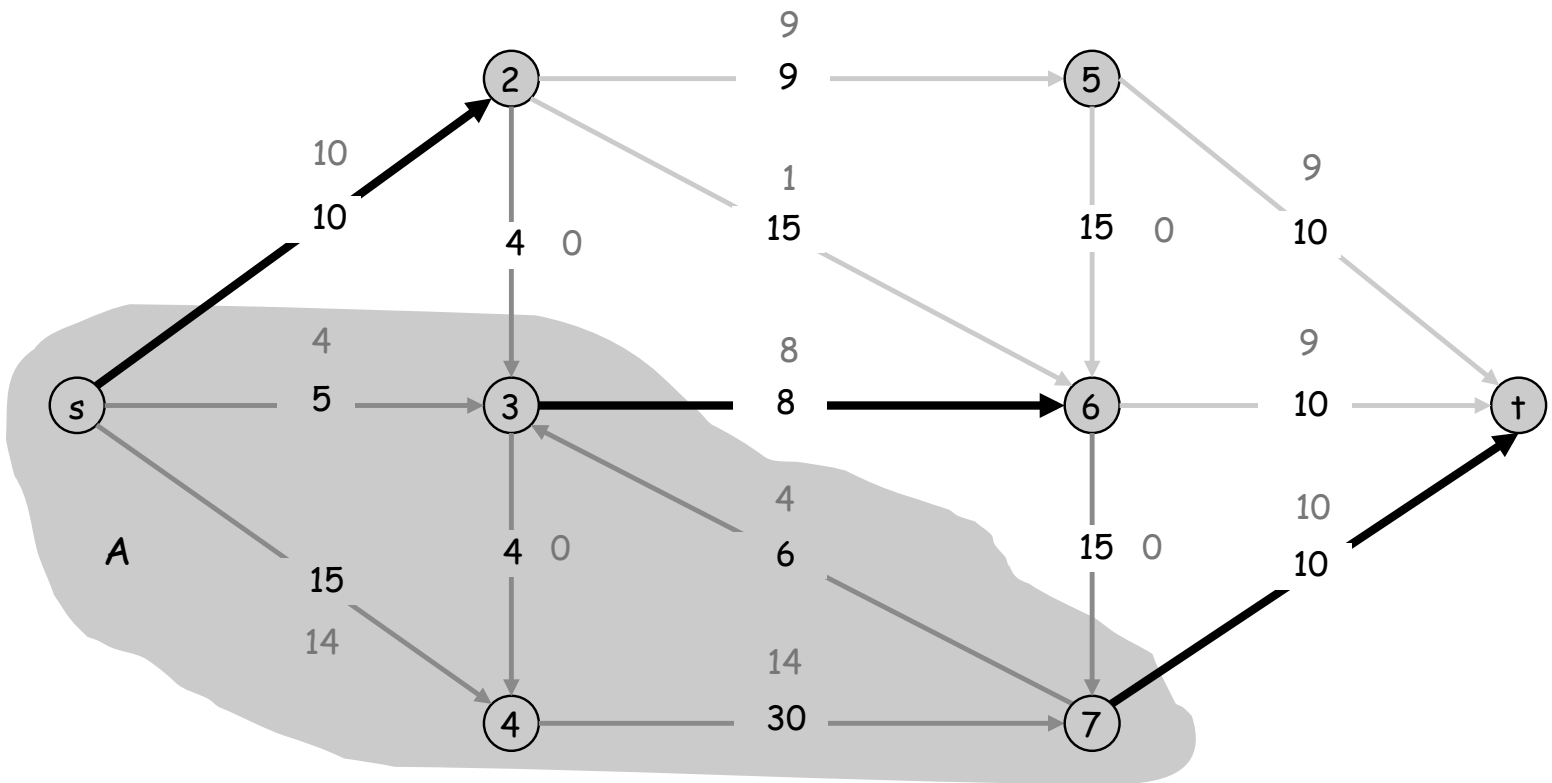
$$
\begin{aligned}
v(f) &= \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in } A} f(e) \\
&\le \sum_{e \text{ esce da } A} f(e) \\
&\le \sum_{e \text{ esce da } A} c(e) \\
&= \text{cap}(A, B) \qquad \blacksquare
\end{aligned}
$$



Certificato di Ottimalità

Corollario. Sia f un flusso, e sia (A, B) un taglio.
Se v(f) = cap(A, B), allora f è un flusso massimo e (A, B) è un taglio minimo.

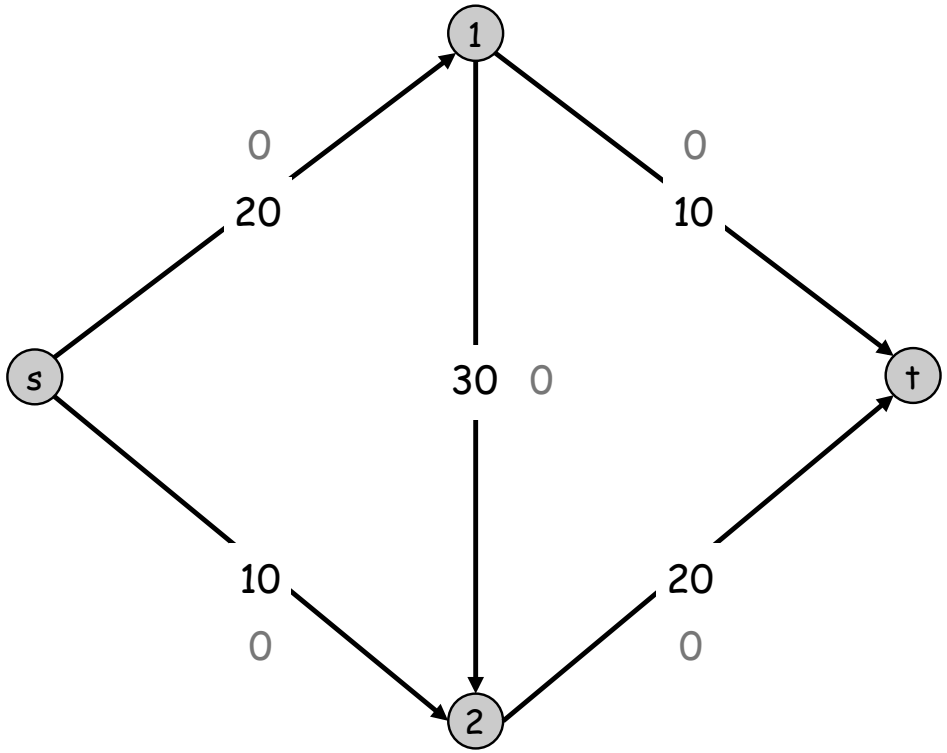
Valore del flusso = 28
Capacità taglio = 28 ⇒ valore flusso ≤ 28



Verso un Algoritmo per il Flusso Massimo

Algoritmo greedy.

- Iniziamo con f(e) = 0 per tutti gli archi $e \in E$.
- Trovare un cammino s-t, sia esso P, dove ogni arco ha $f(e) < c(e)$.
- Aumentare flusso lungo il cammino P.
- Ripetere i precedenti due passi fino a che è possibile.

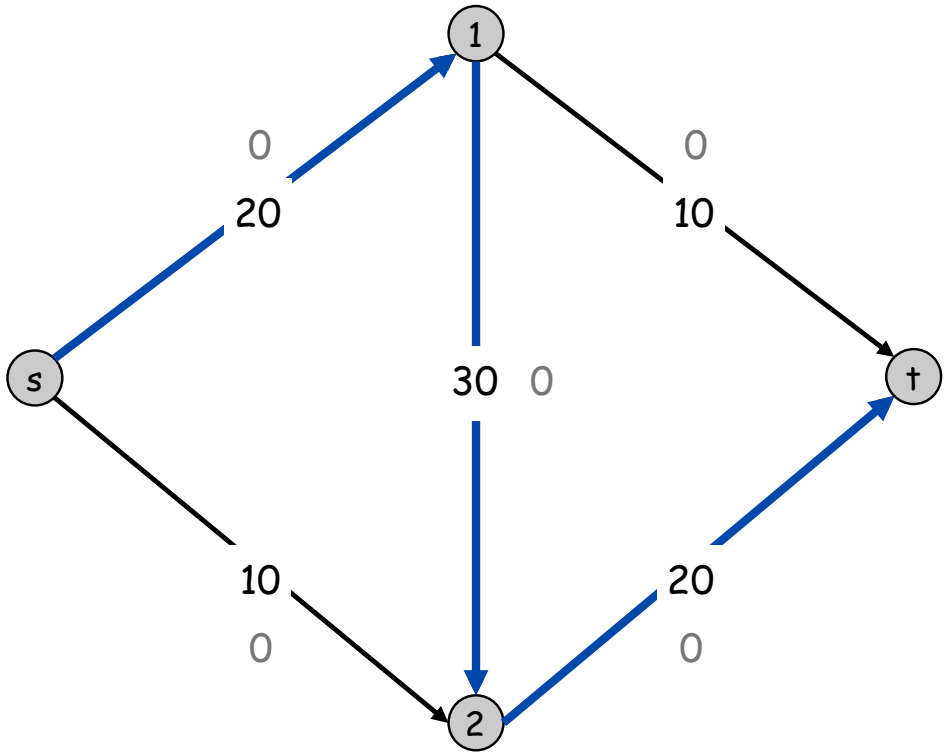


valore flusso = 0

Verso un Algoritmo per il Flusso Massimo

Algoritmo greedy.

- Iniziamo con f(e) = 0 per tutti gli archi $e \in E$.
- Trovare un cammino s-t, sia esso P, dove ogni arco ha $f(e) < c(e)$.
- Aumentare flusso lungo il cammino P.
- Ripetere i precedenti due passi fino a che è possibile.

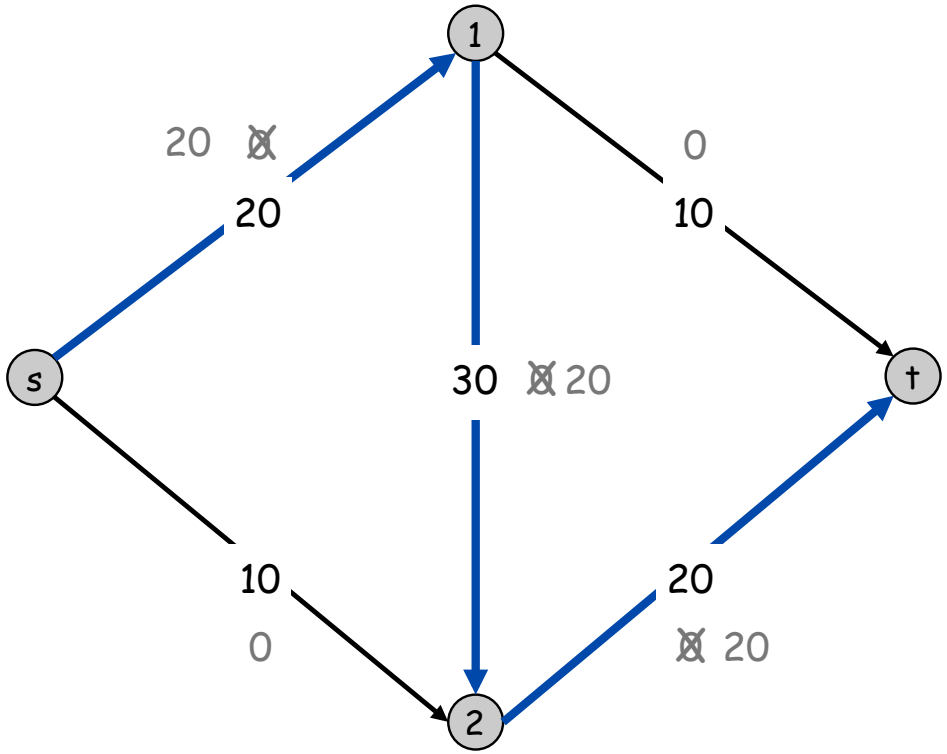


valore flusso = 0

Verso un Algoritmo per il Flusso Massimo

Algoritmo greedy.

- Iniziamo con f(e) = 0 per tutti gli archi $e \in E$.
- Trovare un cammino s-t, sia esso P, dove ogni arco ha $f(e) < c(e)$.
- Aumentare flusso lungo il cammino P.
- Ripetere i precedenti due passi fino a che è possibile.



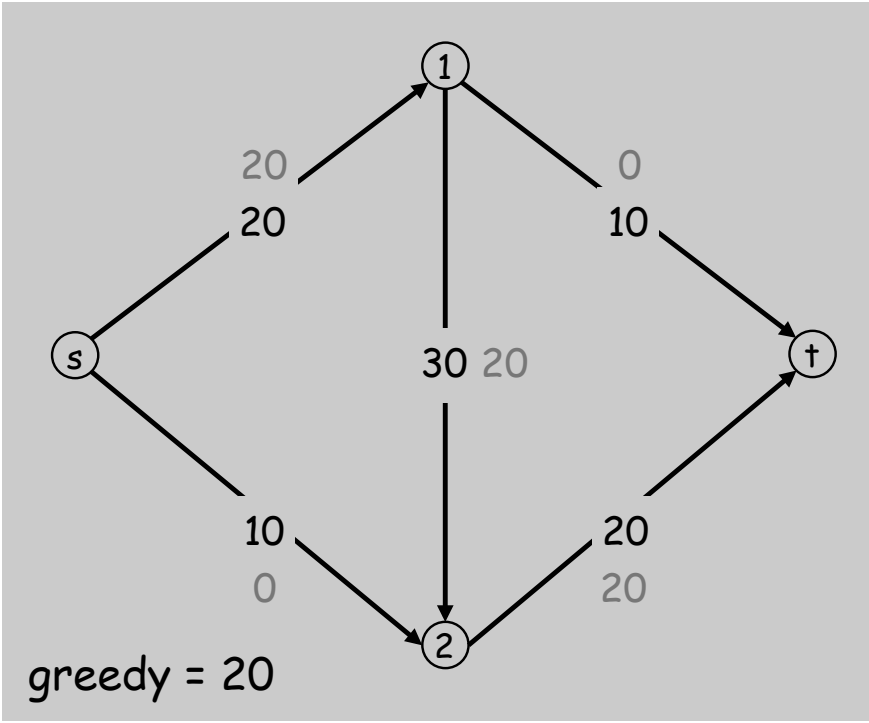
valore flusso = 20

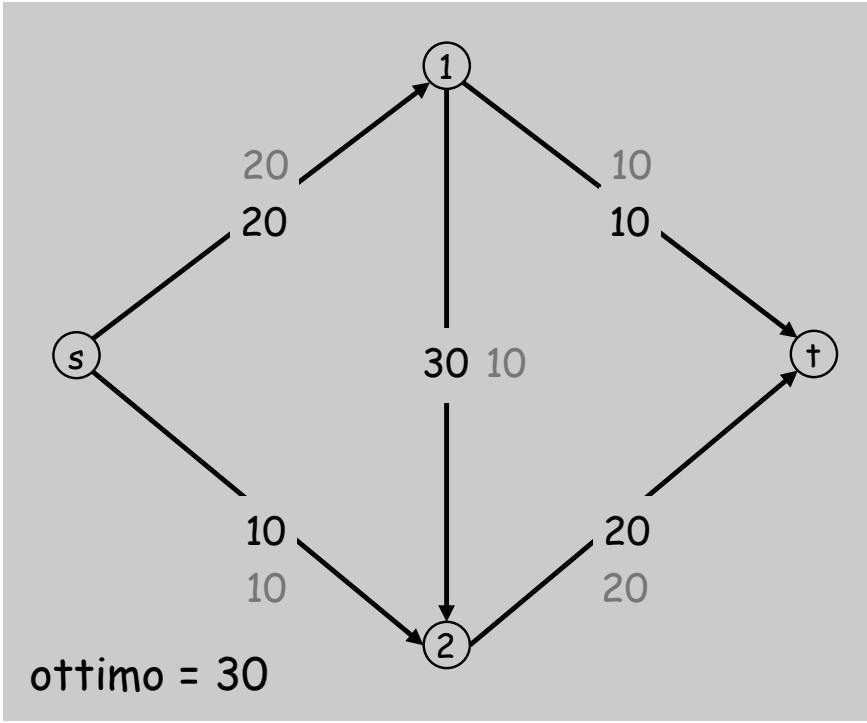
Verso un Algoritmo per il Flusso Massimo

Algoritmo greedy.

- Iniziamo con f(e) = 0 per tutti gli archi $e \in E$.
- Trovare un cammino s-t, sia esso P, dove ogni arco ha $f(e) < c(e)$.
- Aumentare flusso lungo il cammino P.
- Ripetere i precedenti due passi fino a che è possibile.

ottimalità locale $\not\Rightarrow$ ottimalità globale





Nuovi cammini per aumentare il flusso

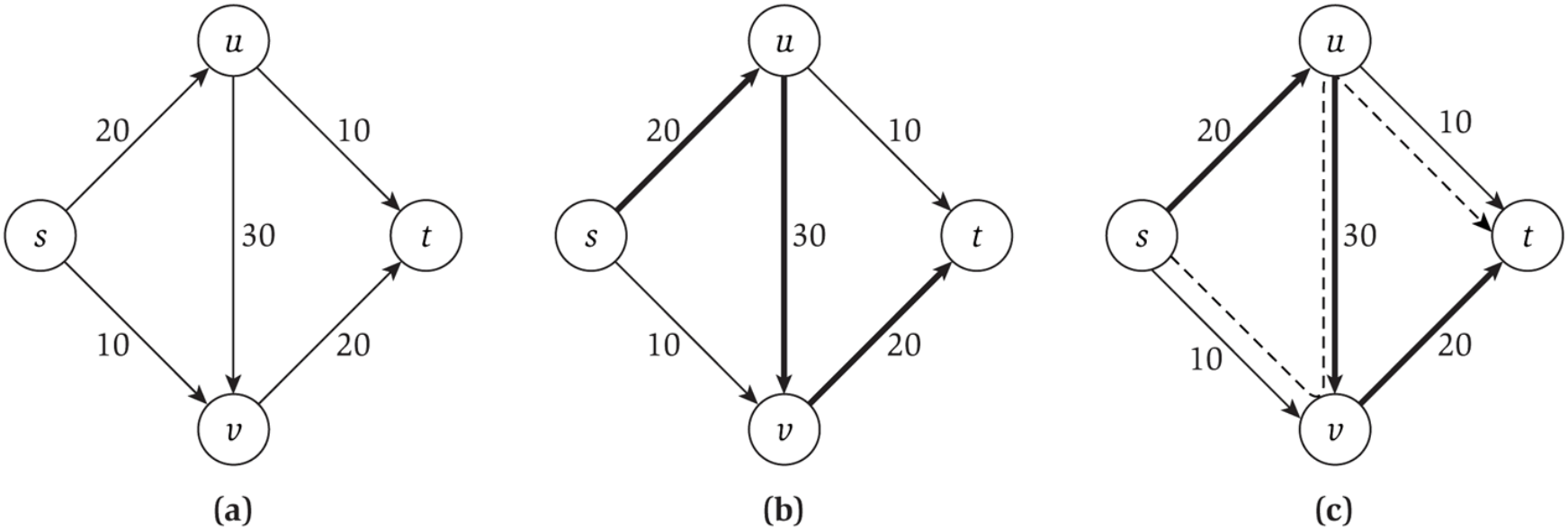
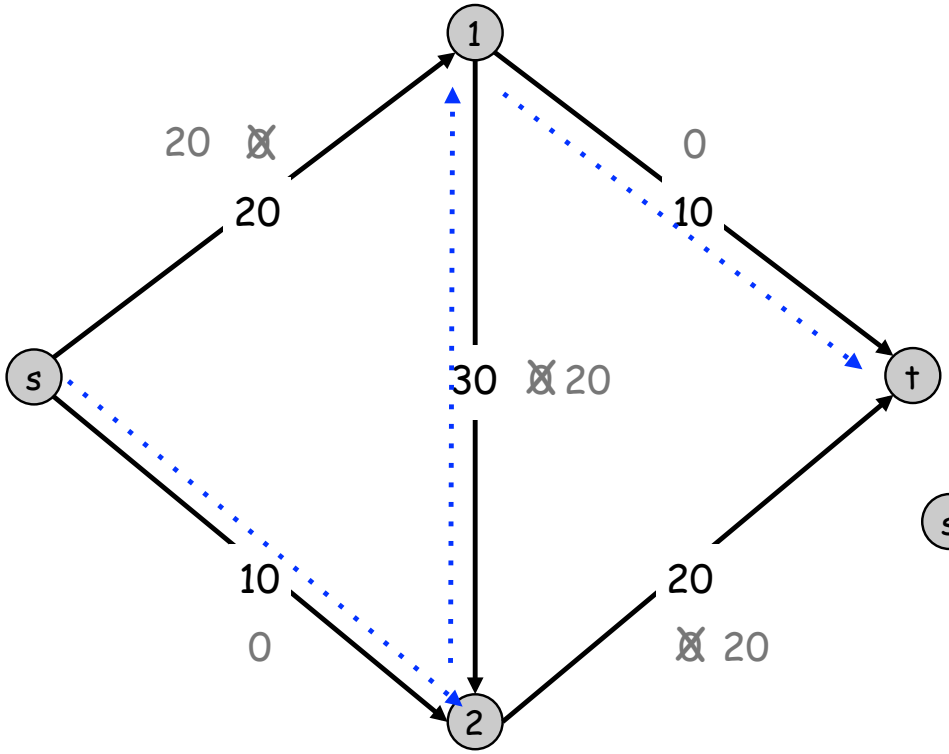
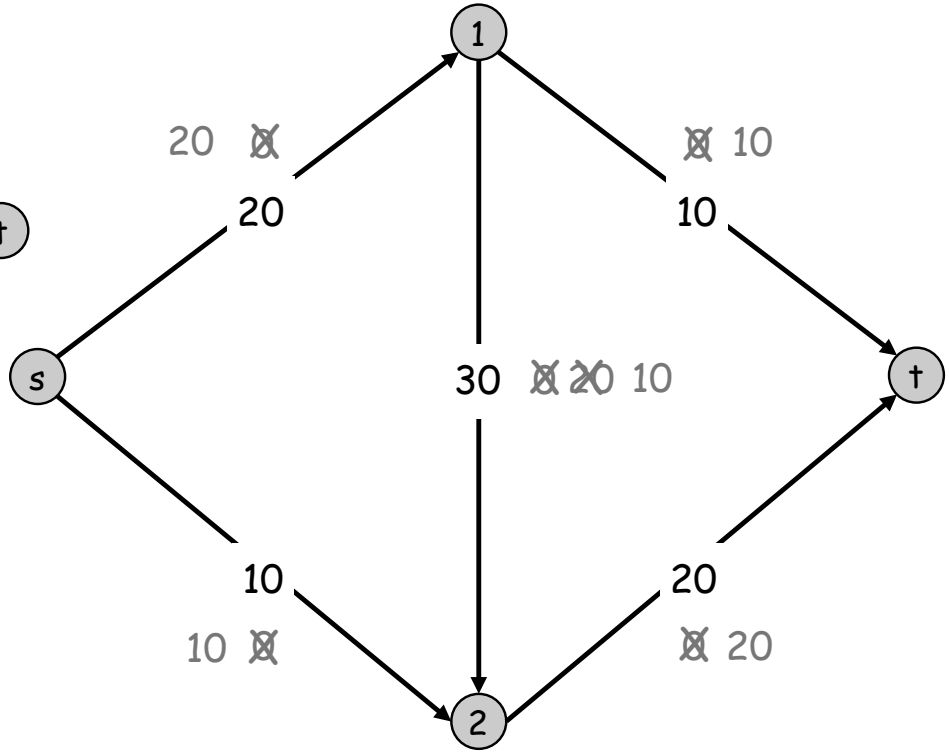


Figure 7.3 (a) The network of Figure 7.2. (b) Pushing 20 units of flow along the path s, u, v, t. (c) The new kind of augmenting path using the edge (u, v) backward.

Nuovi cammini per aumentare il flusso



valore flusso = 20

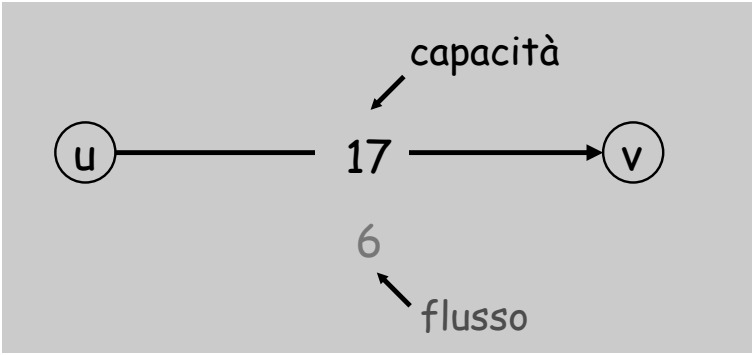


valore flusso = 30

Grafo residuale

Arco originale: $e = (u, v) \in E$.

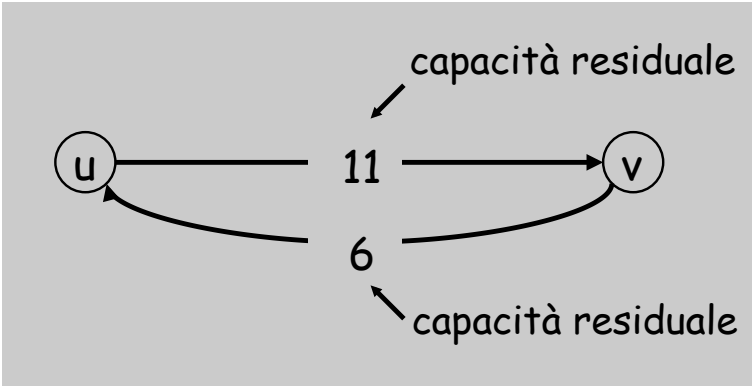
- Flusso f(e), capacità c(e).



Arco residuale.

- Flusso di "undo".
- $e = (u, v)$ $e^R = (v, u)$.
- Capacità residuale:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{se } e \in E \\ f(e) & \text{se } e^R \in E \end{cases}$$



Grafo residuale rispetto ad f: $G_f = (V, E_f)$.

- Archi residuali con capacità residuale positiva.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : c(e) > 0\}$.

Grafo residuale: esempio

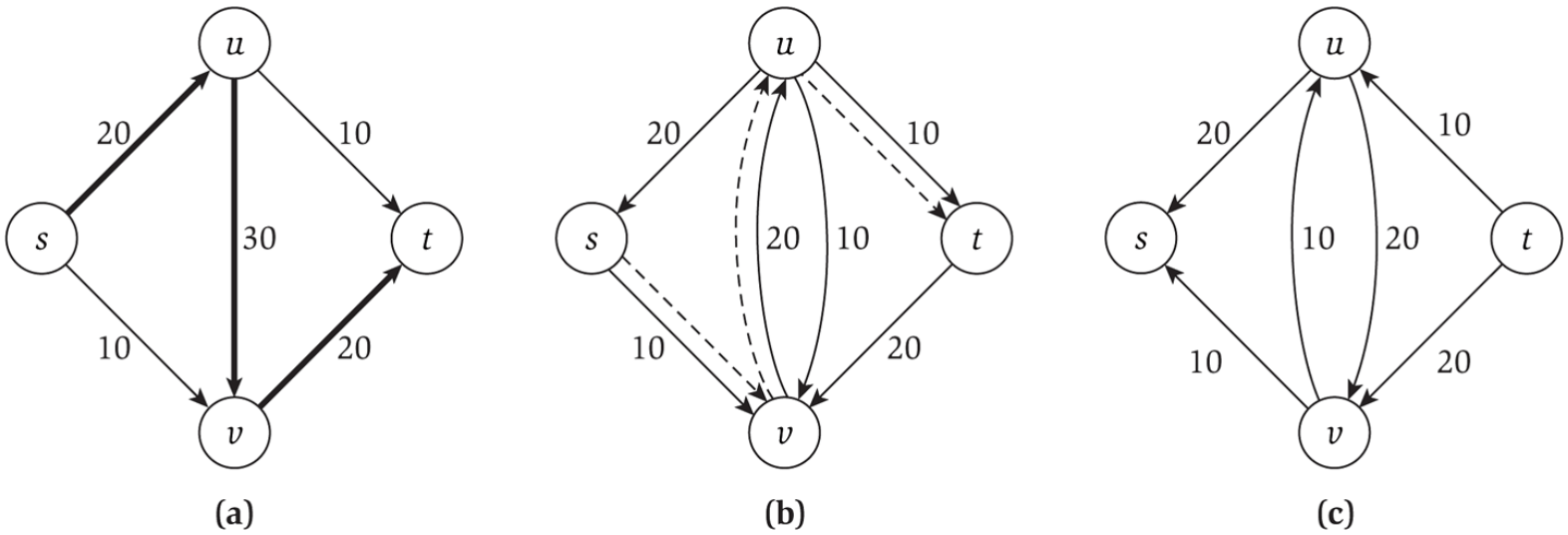
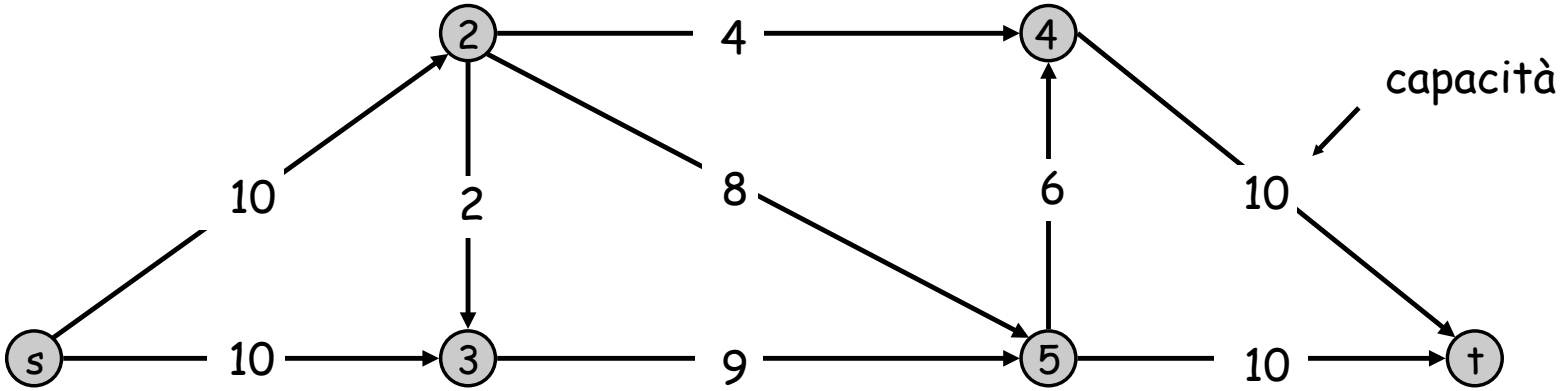


Figure 7.4 (a) The graph G with the path s, u, v, t used to push the first 20 units of flow. (b) The residual graph of the resulting flow f, with the residual capacity next to each edge. The dotted line is the new augmenting path. (c) The residual graph after pushing an additional 10 units of flow along the new augmenting path s, v, u, t.

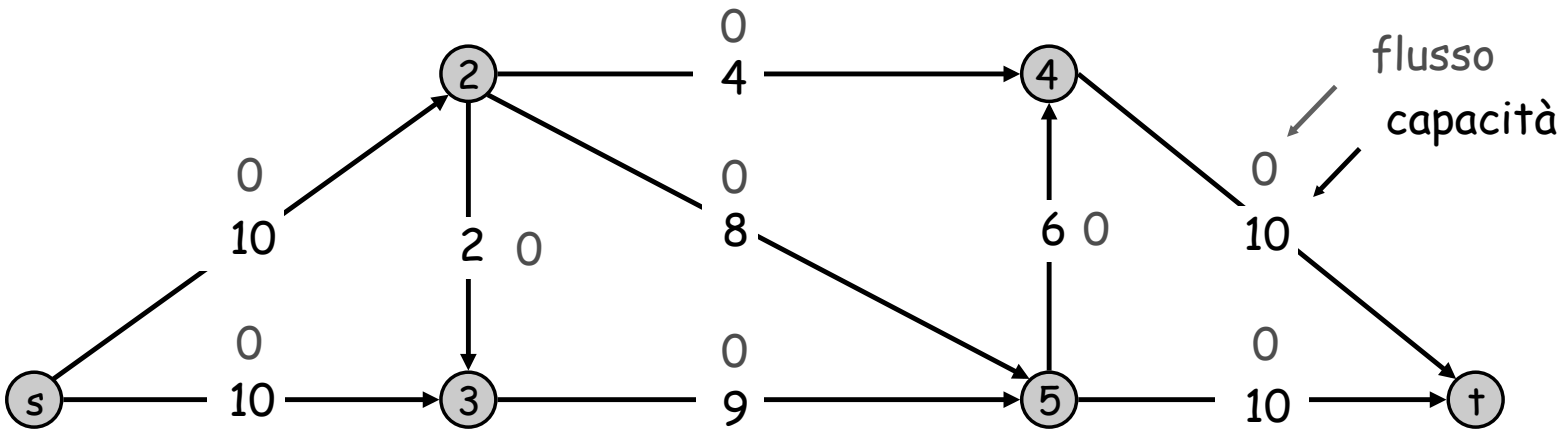
Algoritmo Ford-Fulkerson: esempio

G:



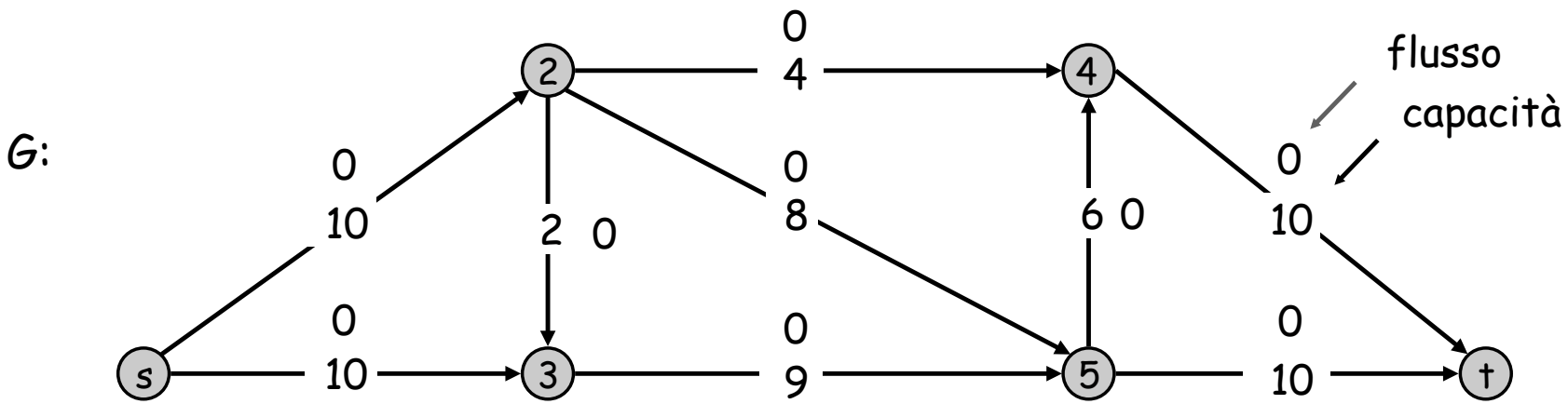
Algoritmo Ford-Fulkerson: esempio

G:

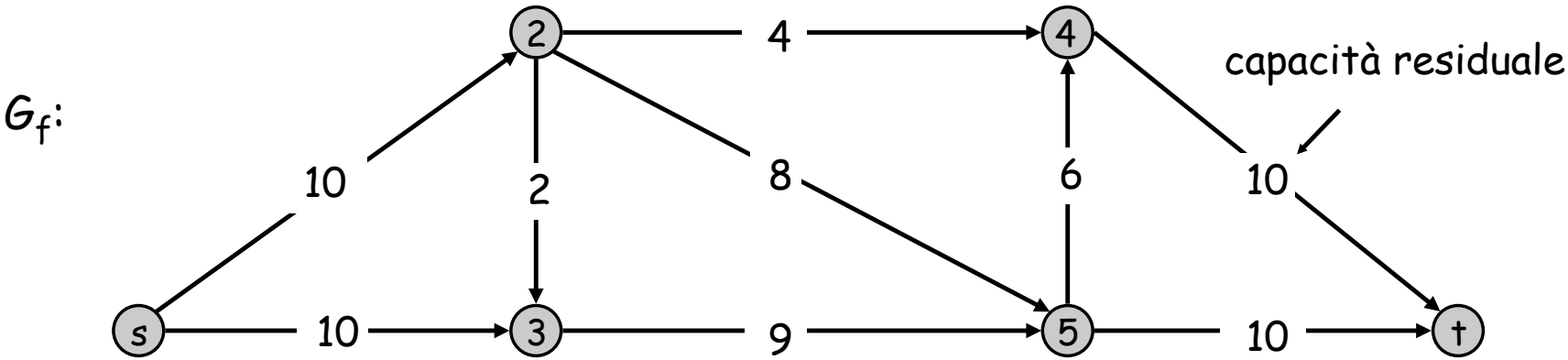


valore flusso = 0

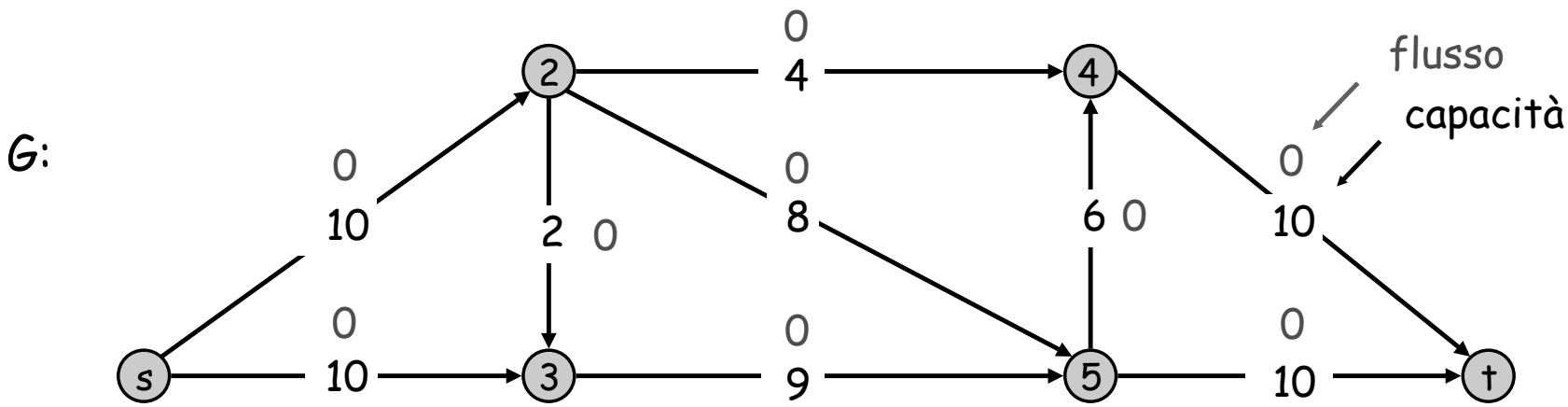
Algoritmo Ford-Fulkerson: esempio



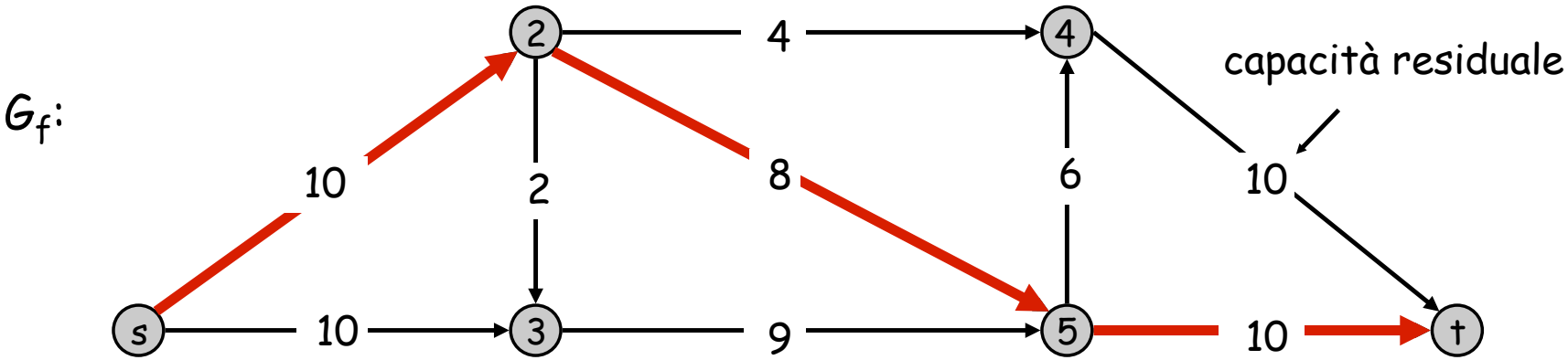
valore flusso = 0



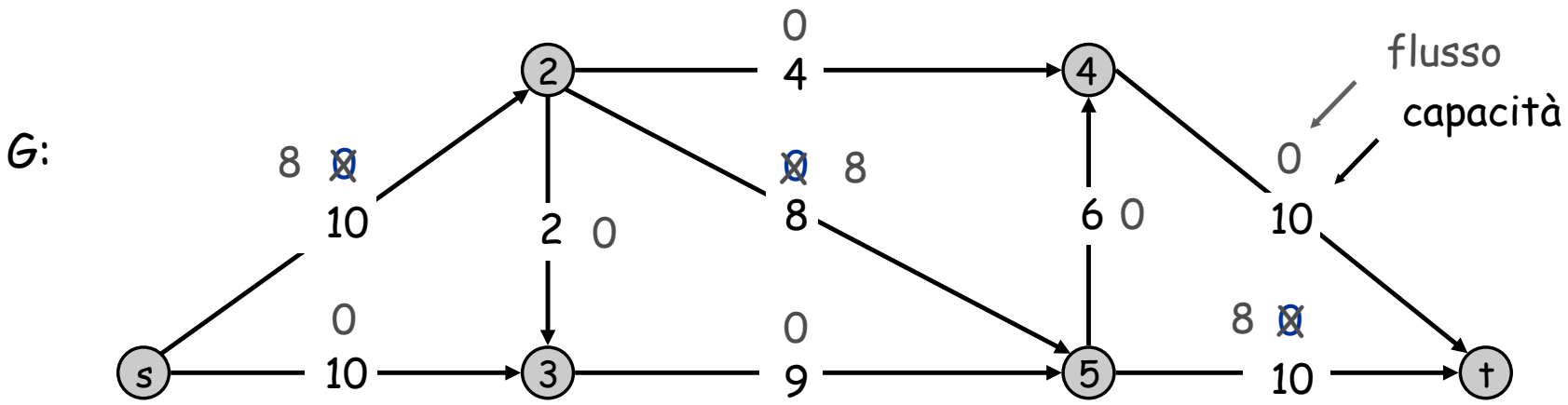
Algoritmo Ford-Fulkerson: esempio



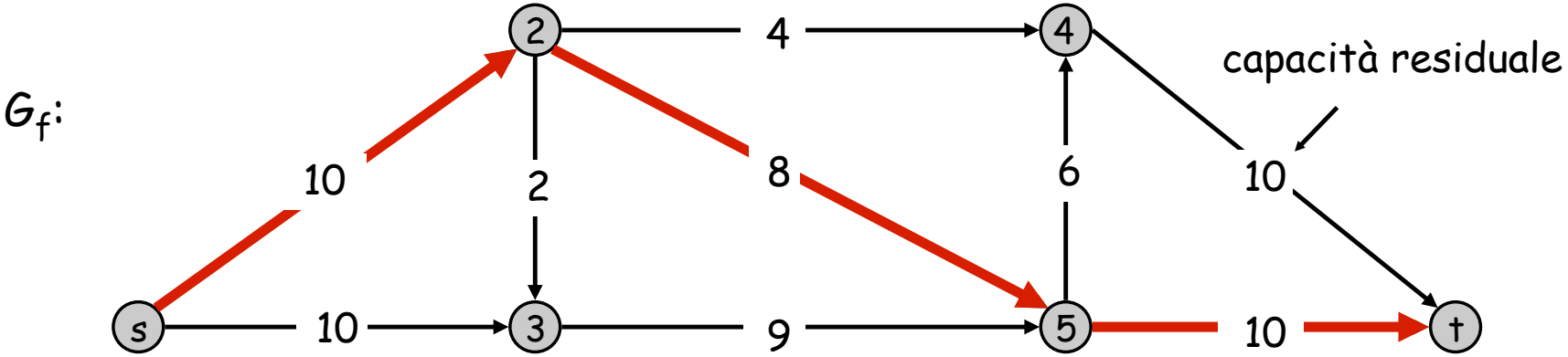
valore flusso = 0



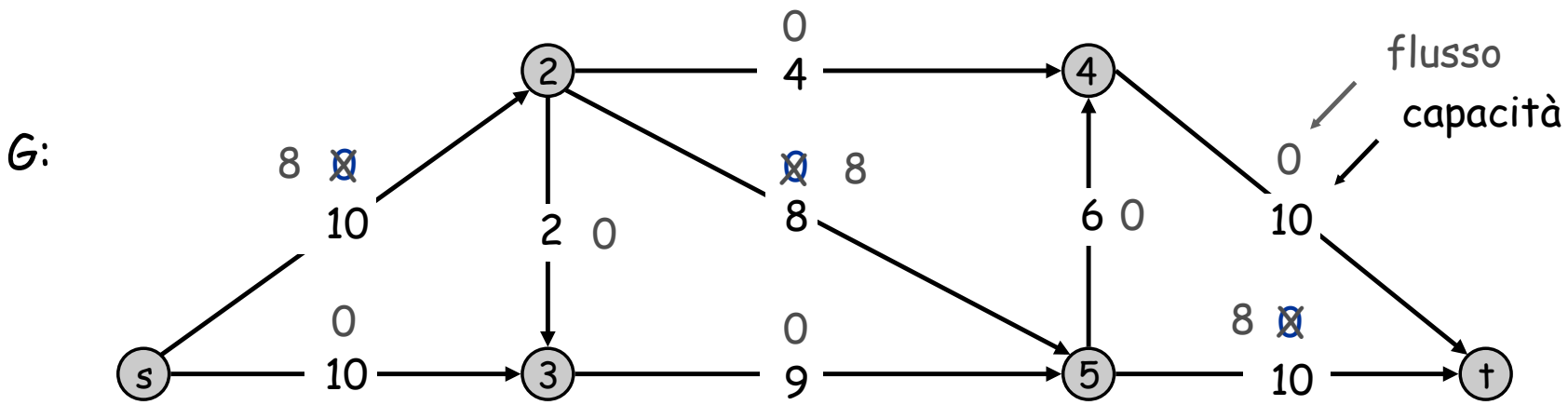
Algoritmo Ford-Fulkerson: esempio



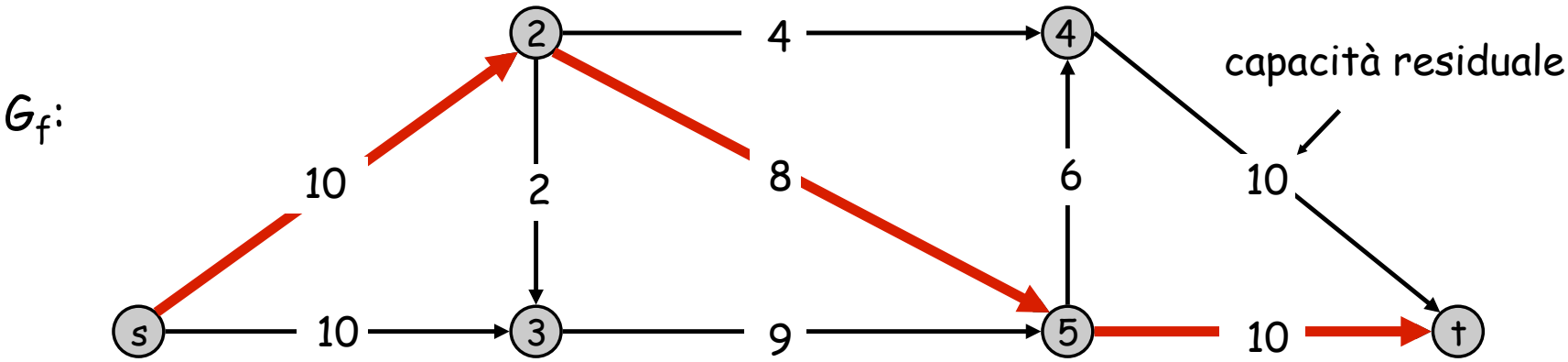
valore flusso = 0



Algoritmo Ford-Fulkerson: esempio

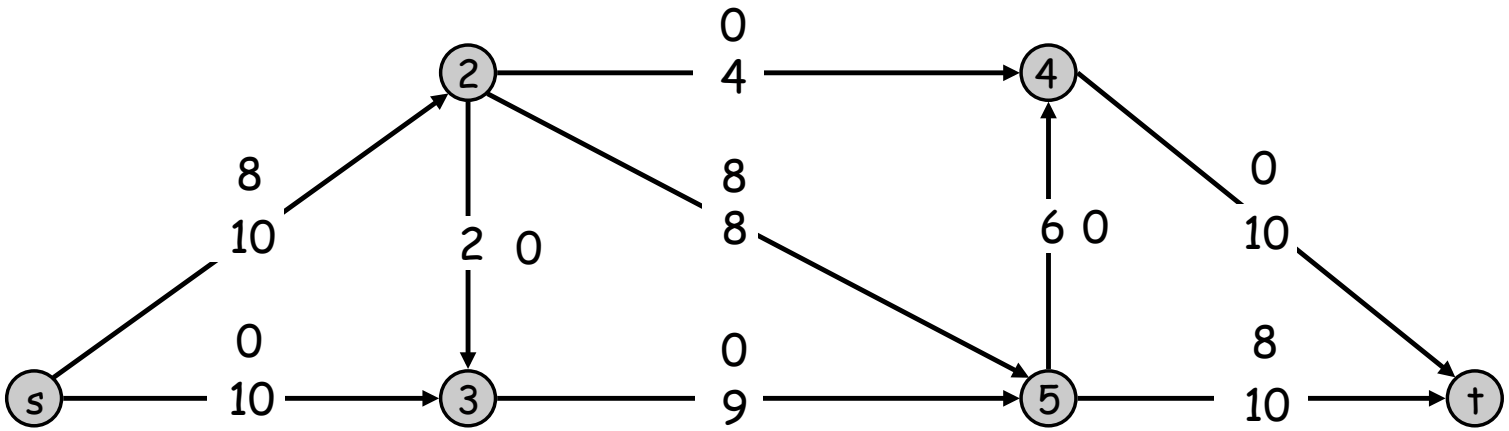


valore flusso = 8



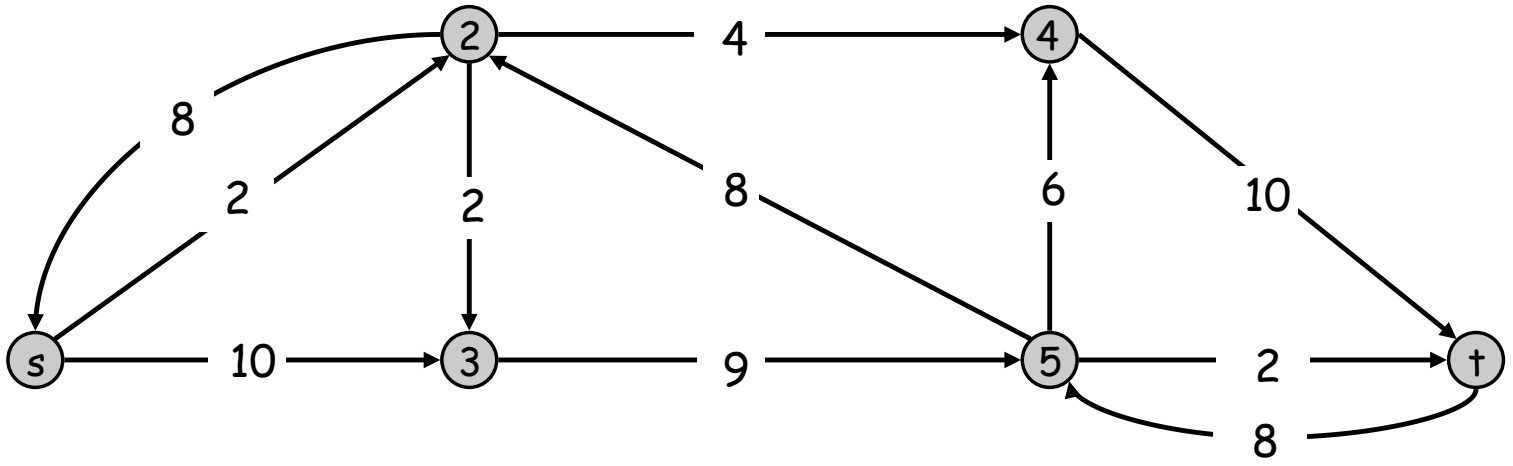
Algoritmo Ford-Fulkerson: esempio

G:

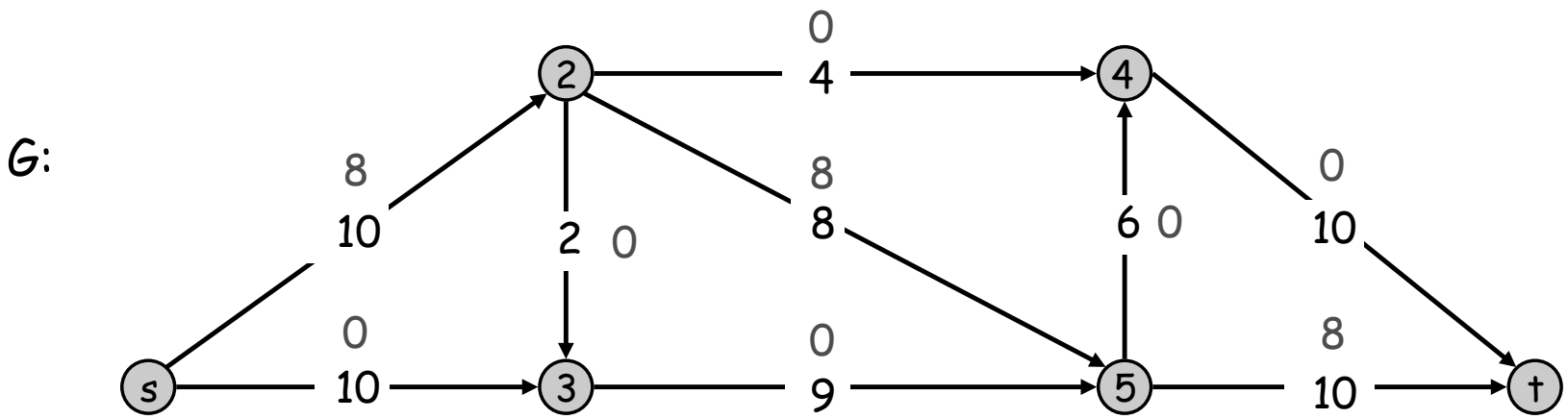


valore flusso = 8

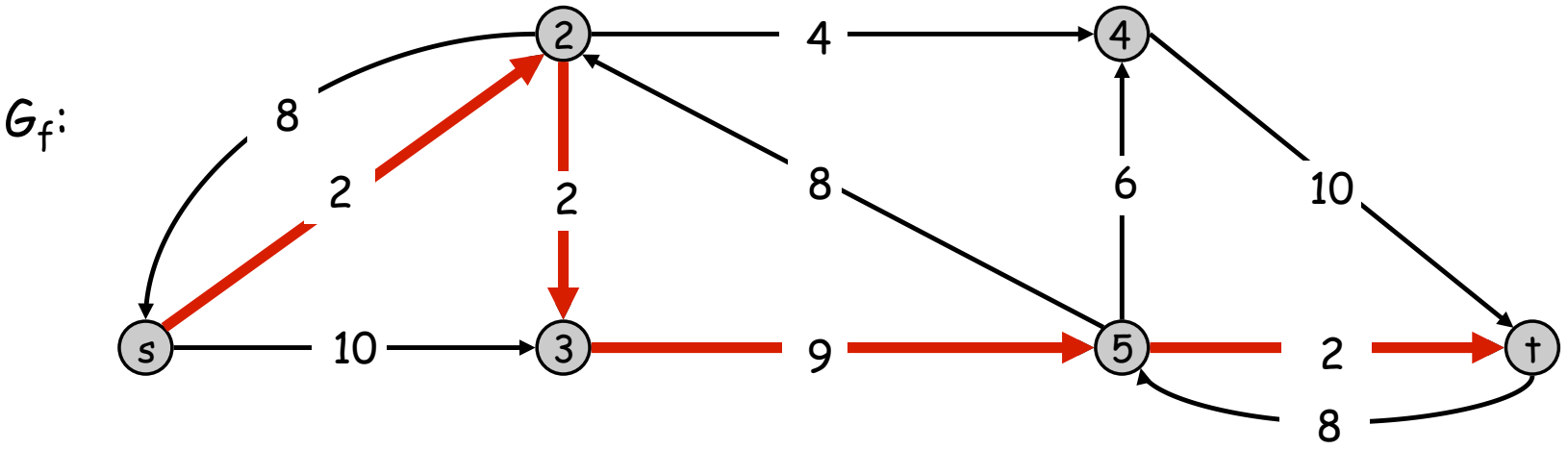
G_f:



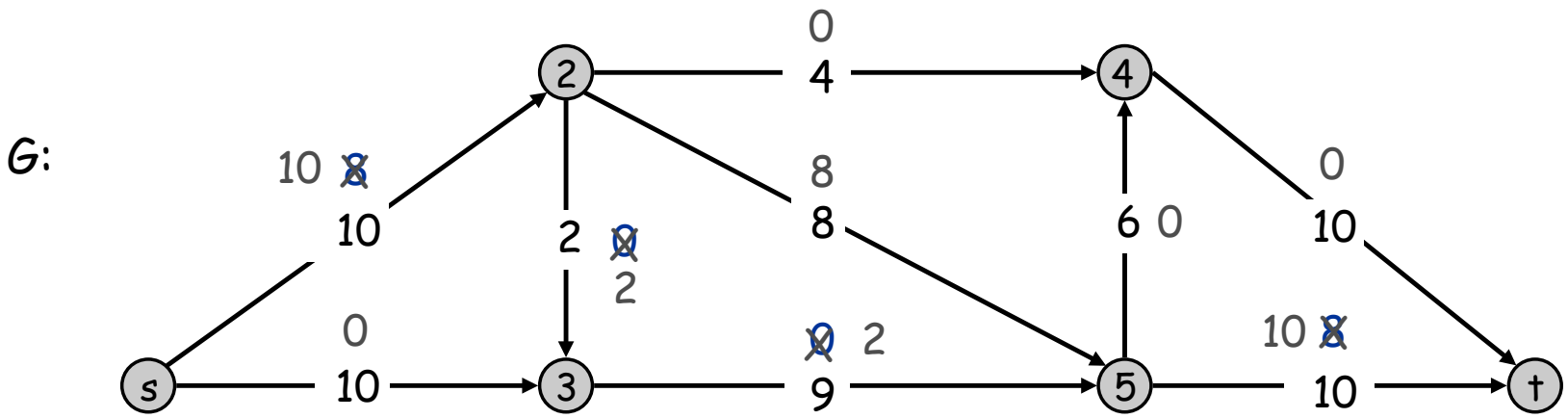
Algoritmo Ford-Fulkerson: esempio



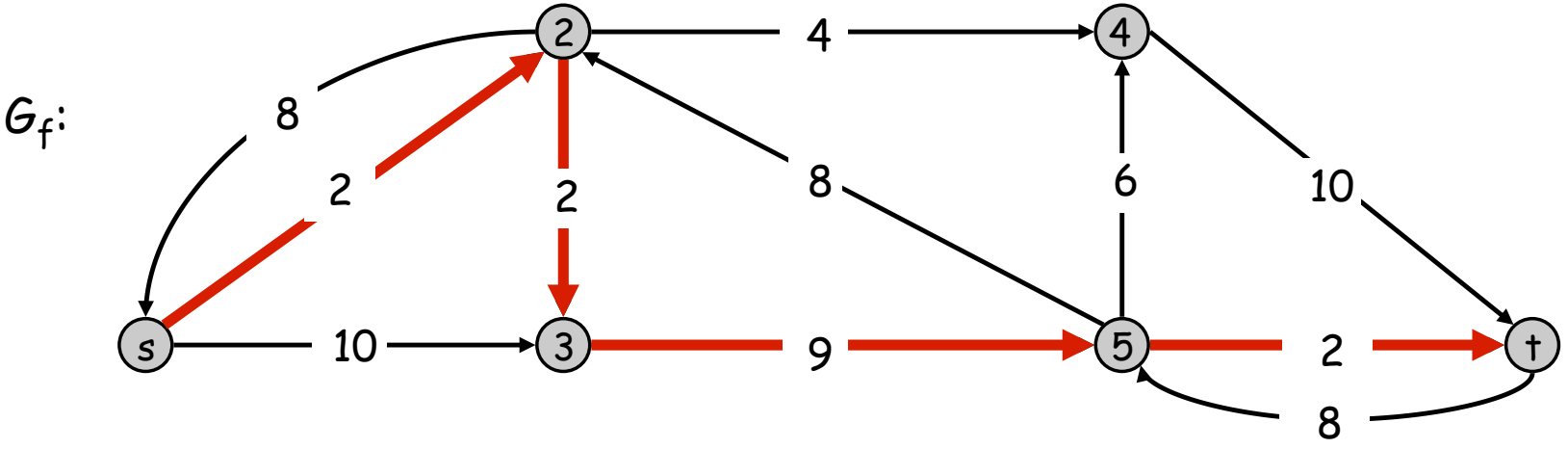
valore flusso = 8



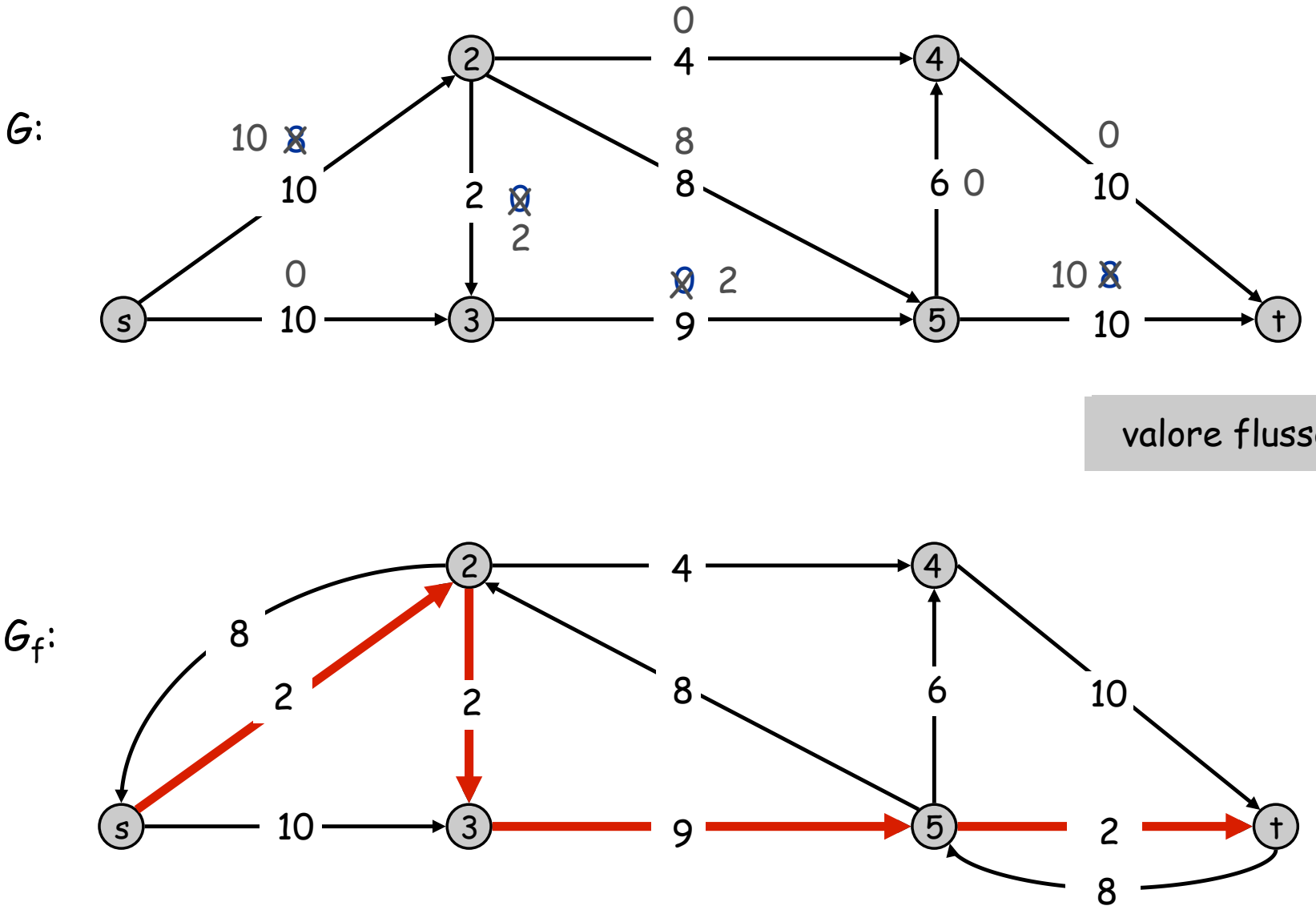
Algoritmo Ford-Fulkerson: esempio



valore flusso = 8



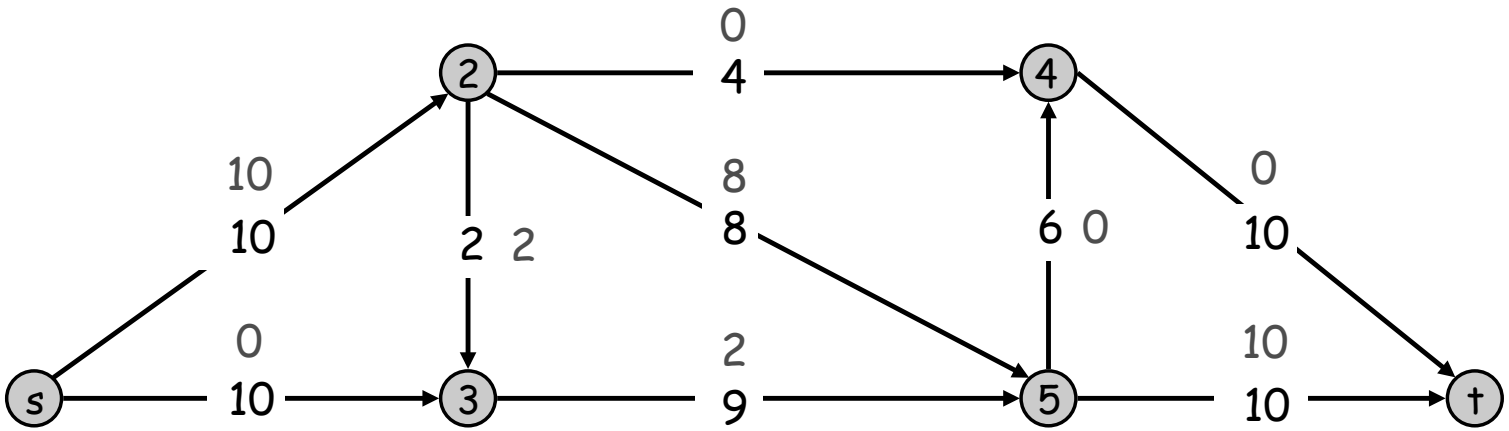
Algoritmo Ford-Fulkerson: esempio



valore flusso = 10

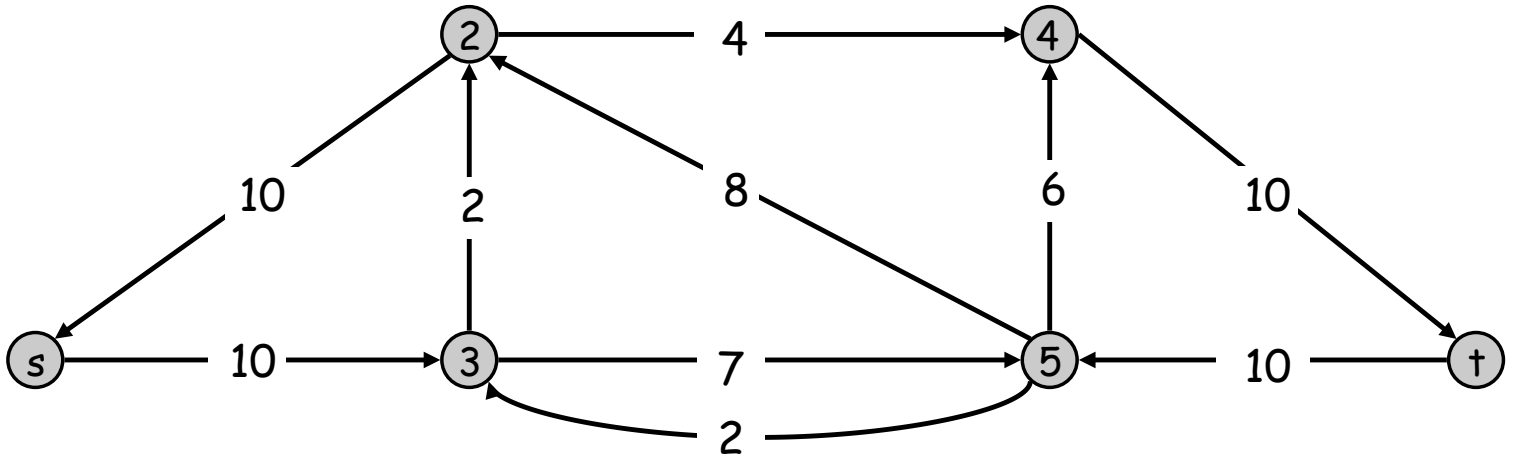
Algoritmo Ford-Fulkerson: esempio

G:



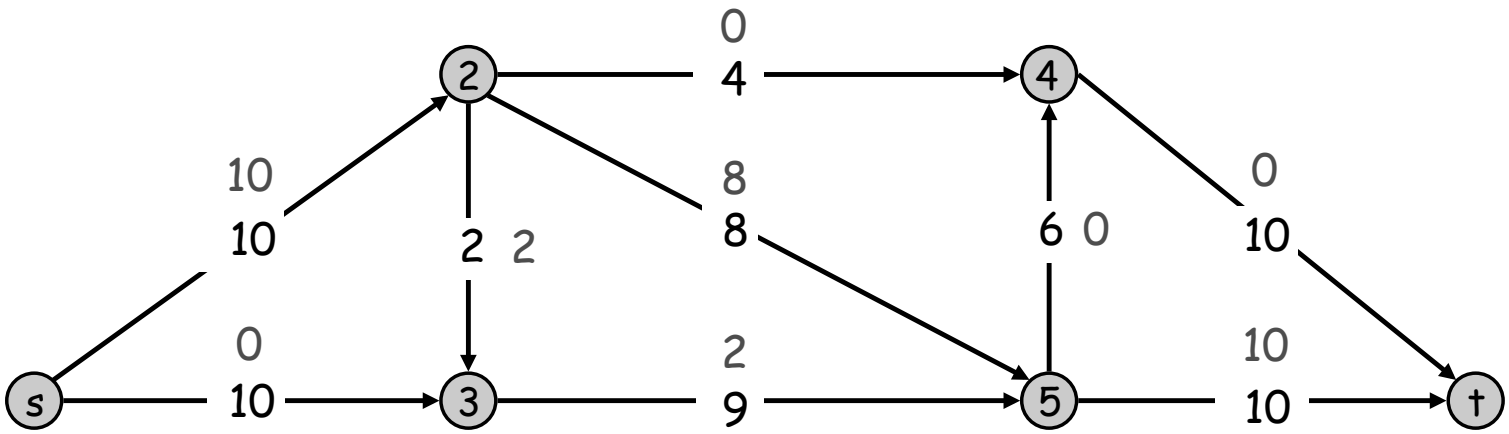
valore flusso = 10

G_f:



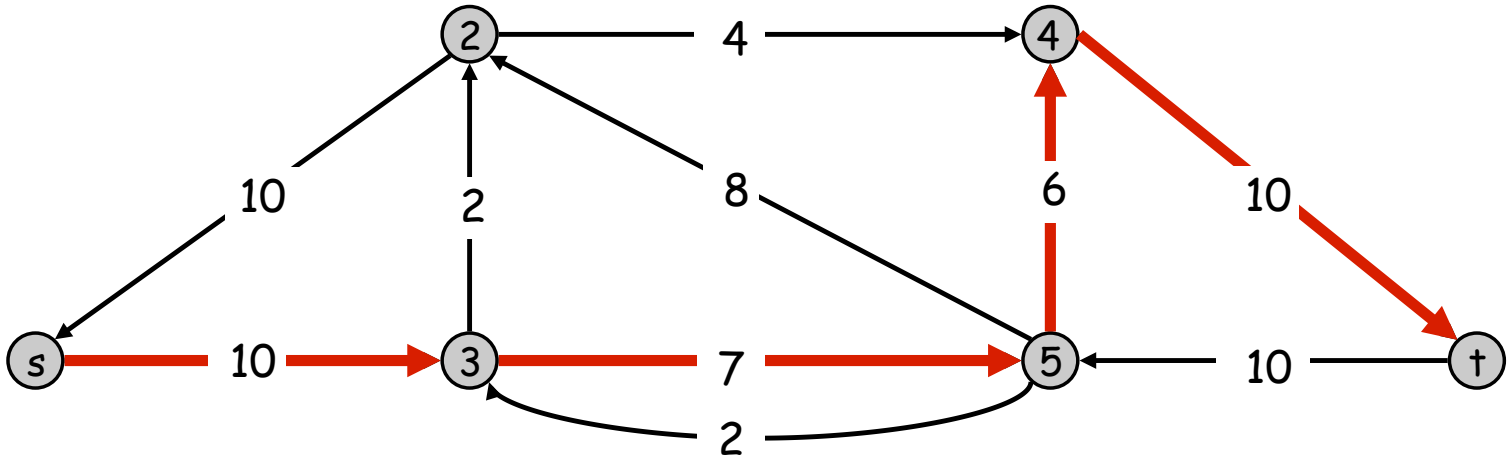
Algoritmo Ford-Fulkerson: esempio

G:



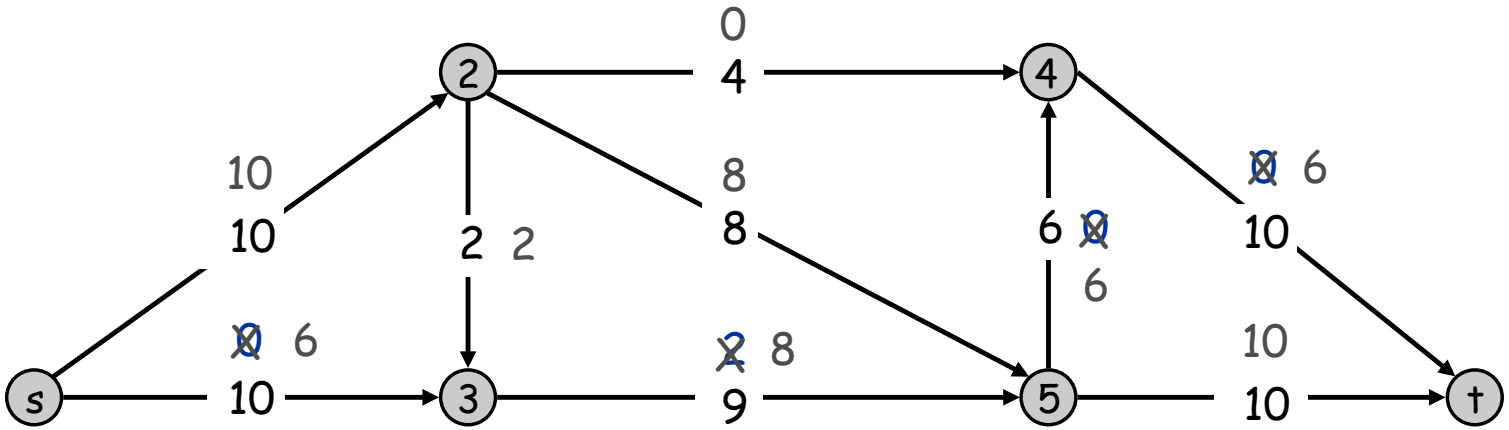
valore flusso = 10

G_f:



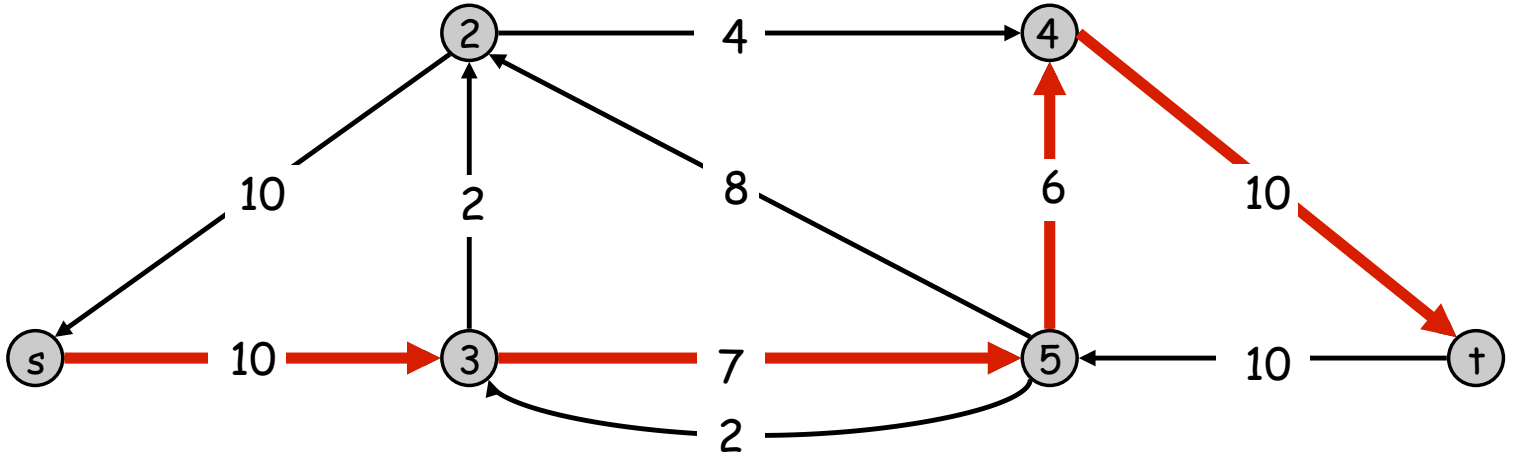
Algoritmo Ford-Fulkerson: esempio

G:



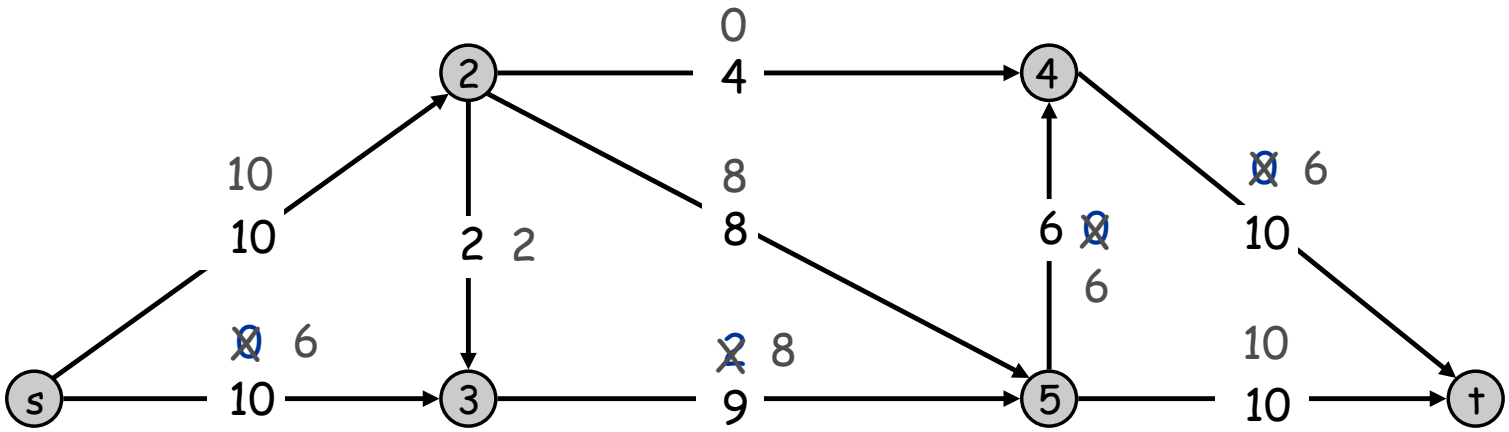
valore flusso = 10

G_f:



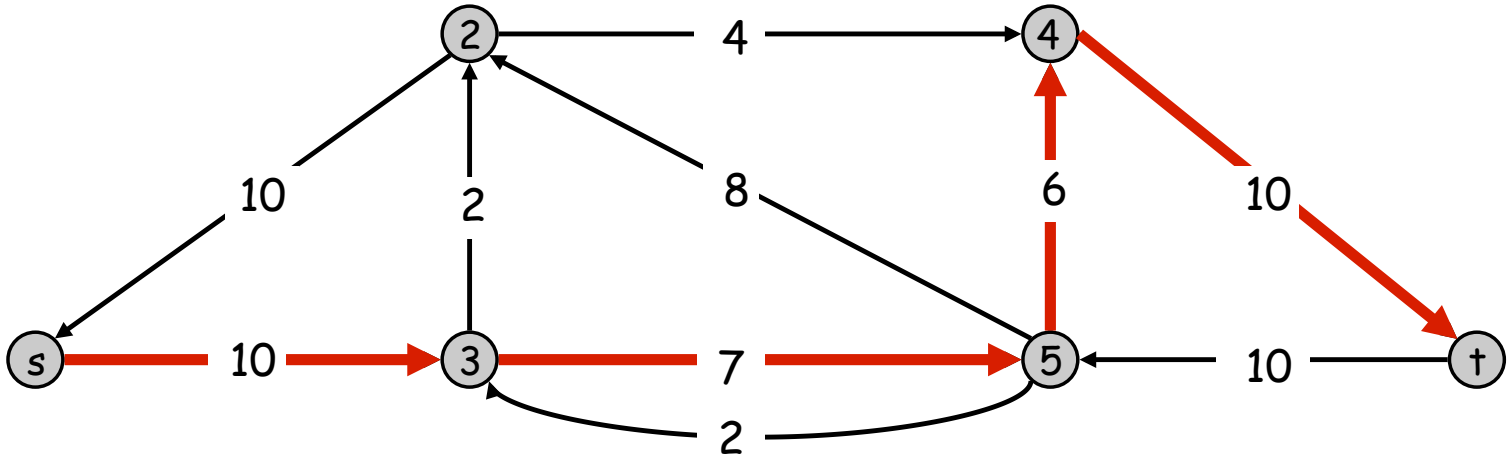
Algoritmo Ford-Fulkerson: esempio

G:



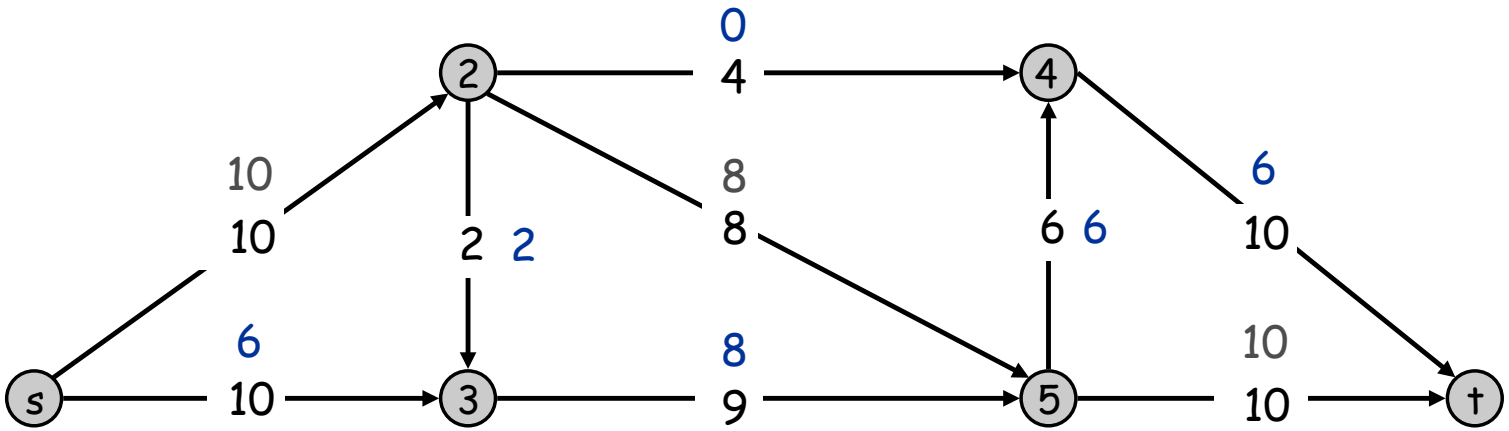
valore flusso = 16

G_f:



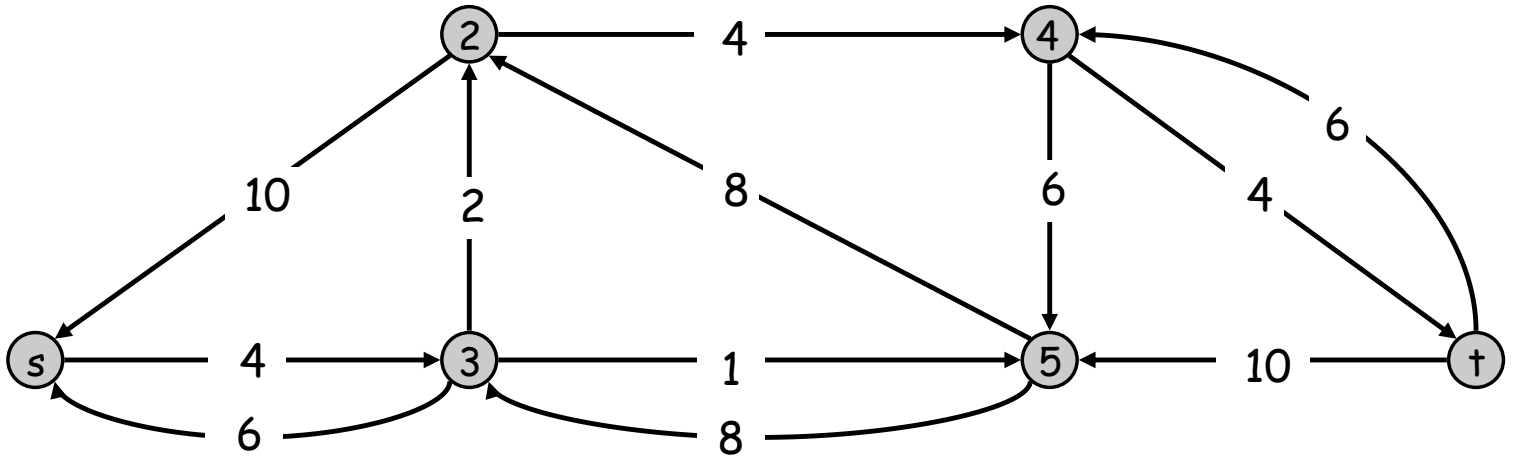
Algoritmo Ford-Fulkerson: esempio

G:



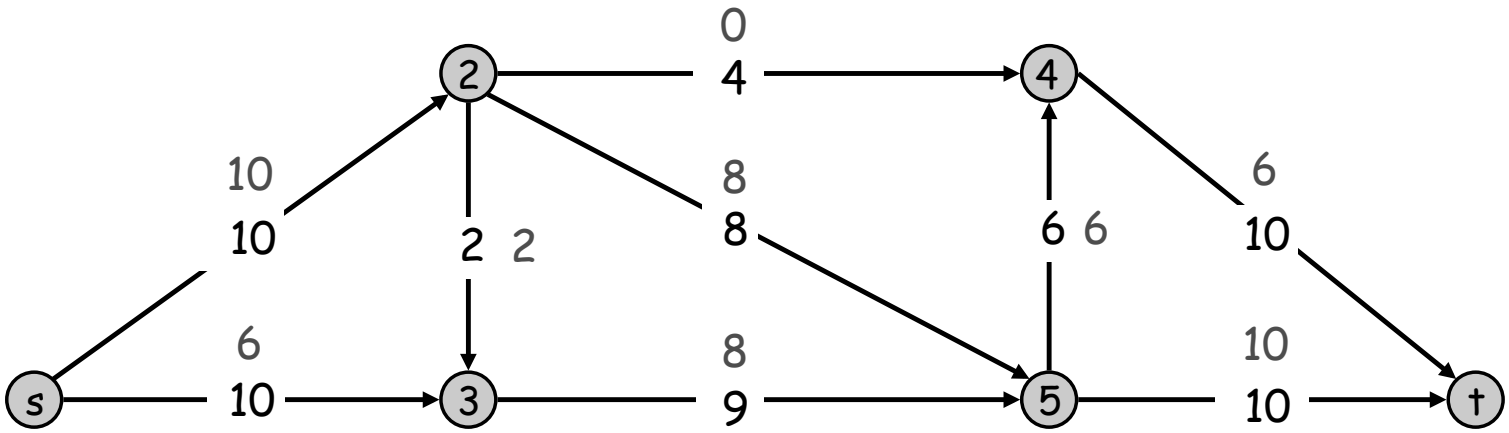
valore flusso = 16

G_f:



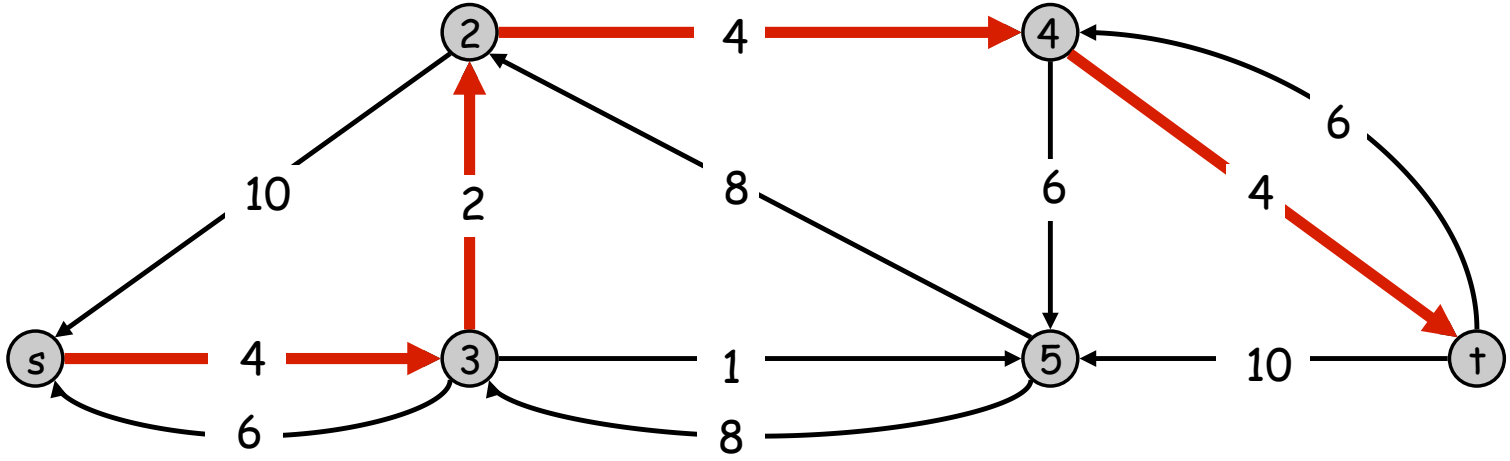
Algoritmo Ford-Fulkerson: esempio

G:



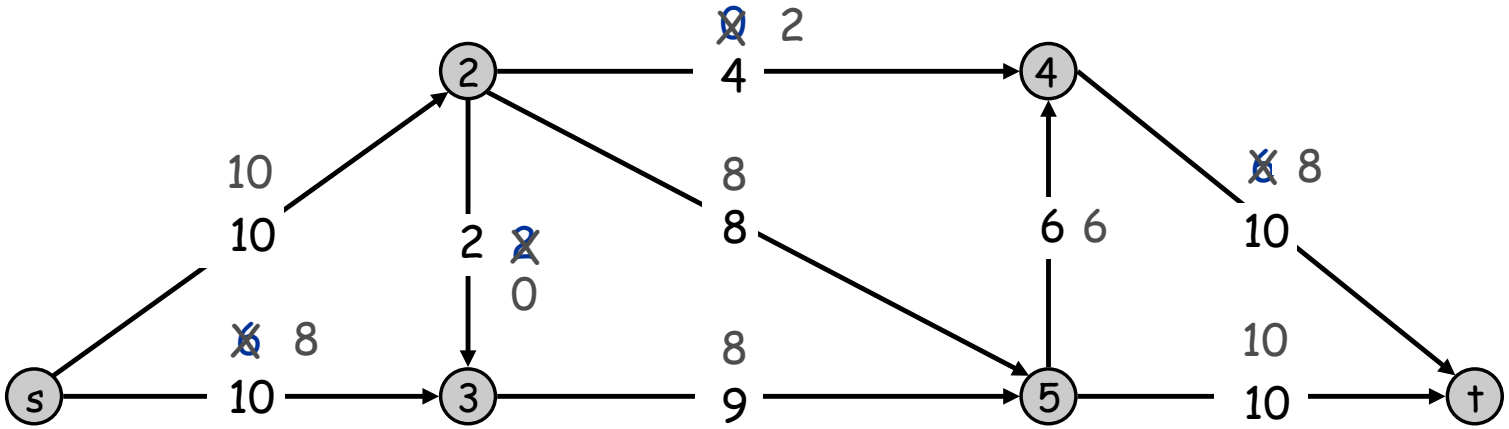
valore flusso = 16

G_f:



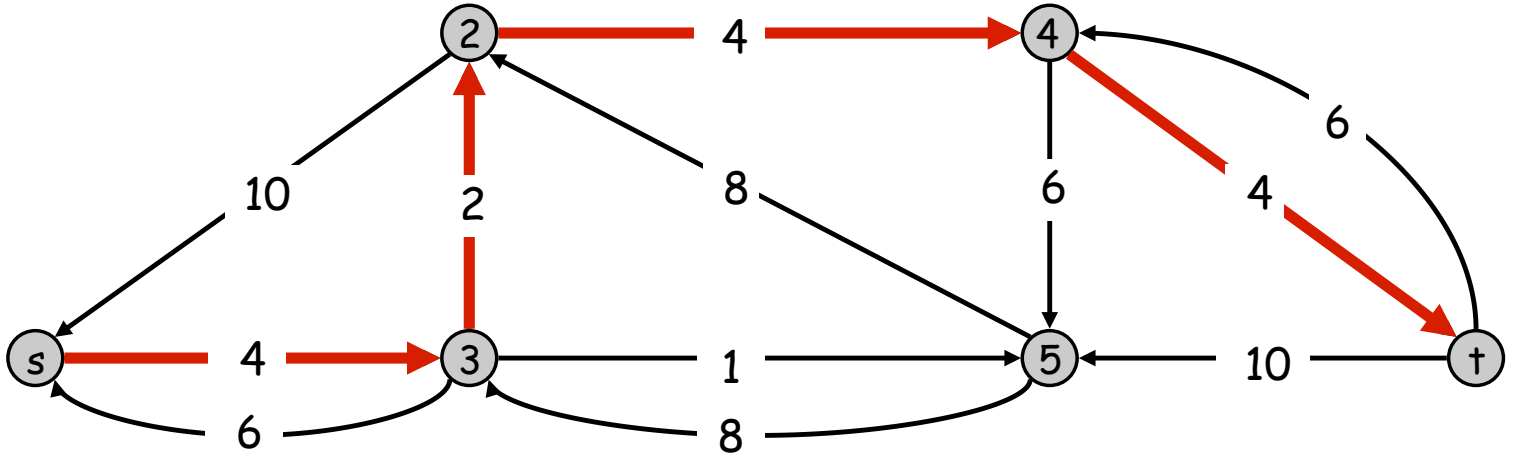
Algoritmo Ford-Fulkerson: esempio

G:



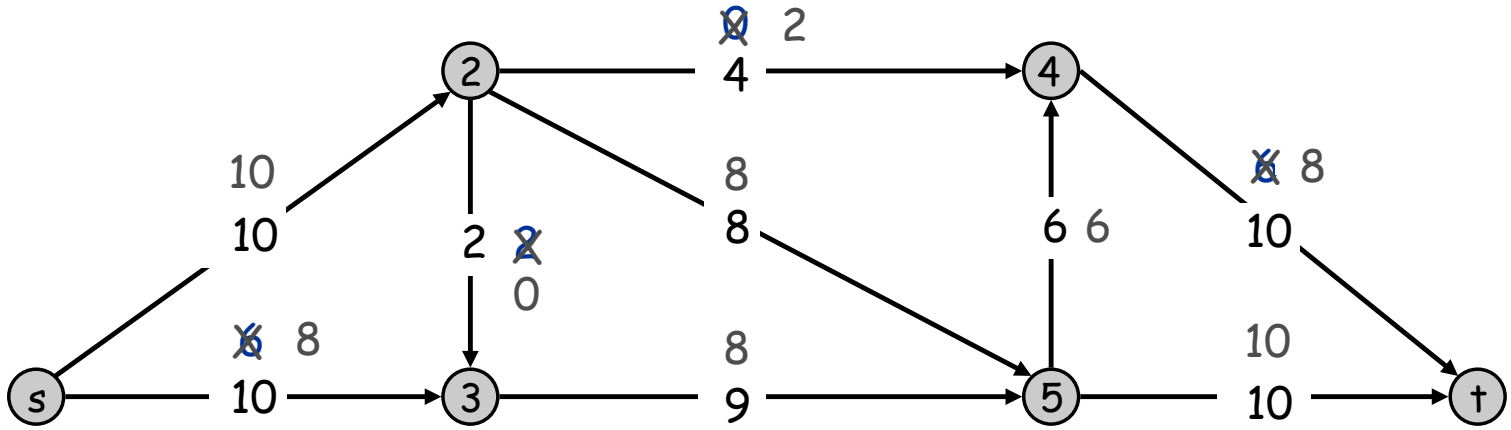
valore flusso = 16

G_f:



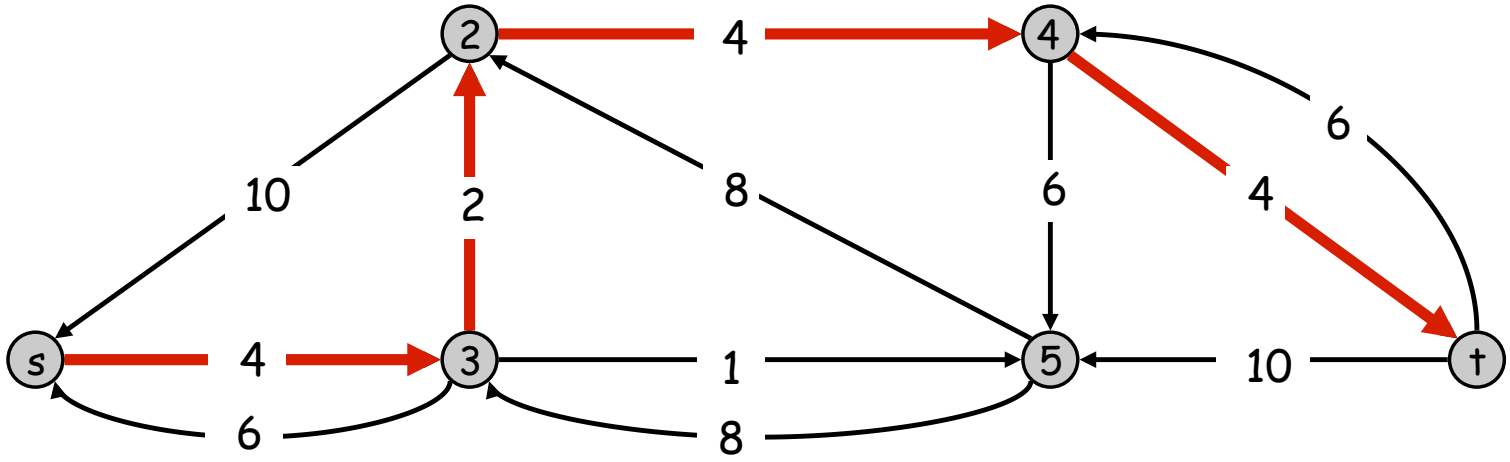
Algoritmo Ford-Fulkerson: esempio

G:



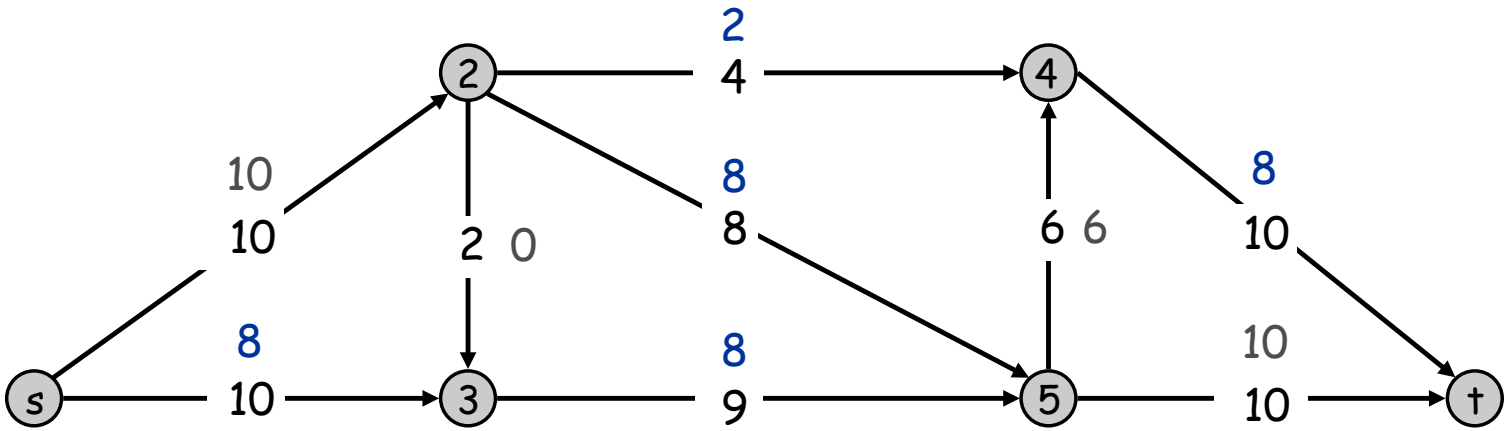
valore flusso = 18

G_f:



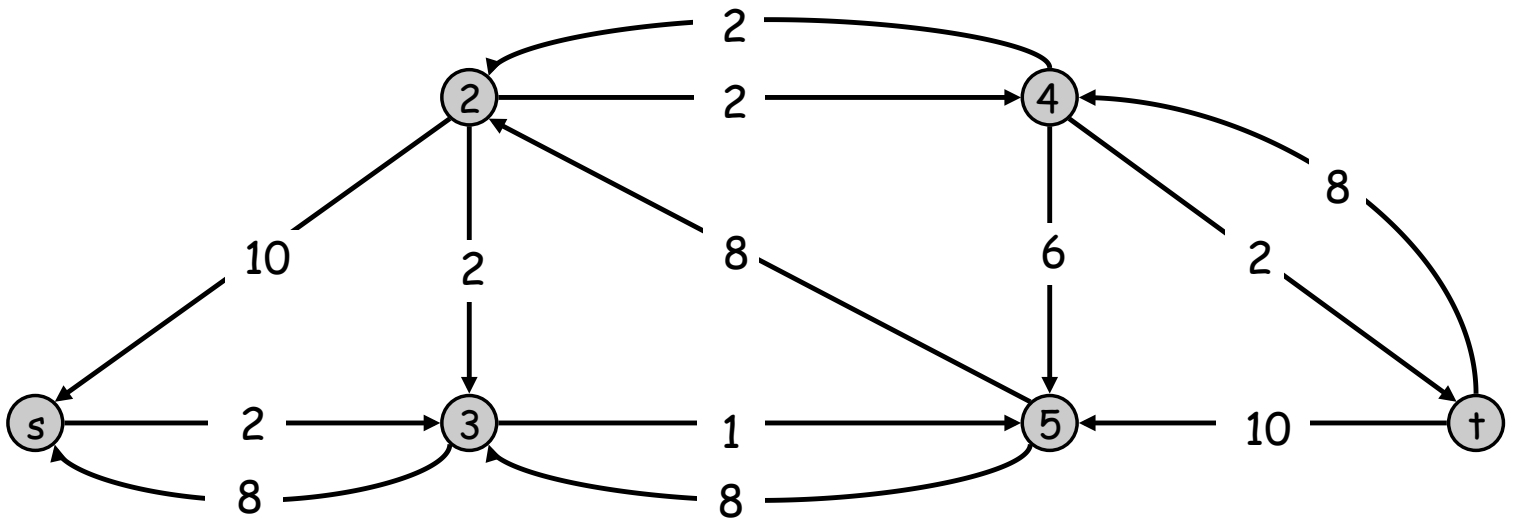
Algoritmo Ford-Fulkerson: esempio

G:



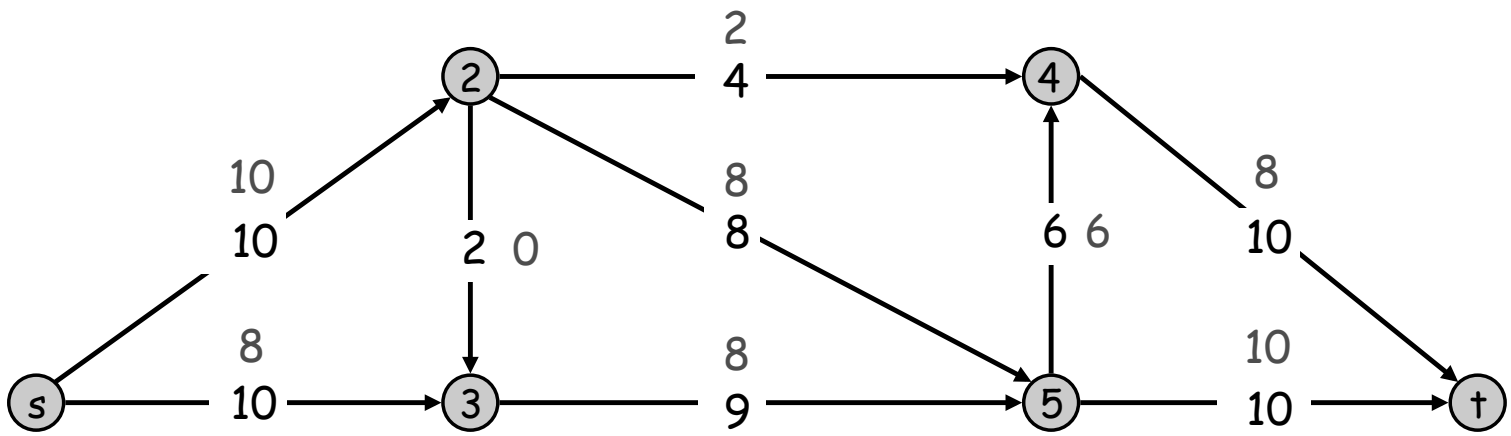
valore flusso = 18

G_f:



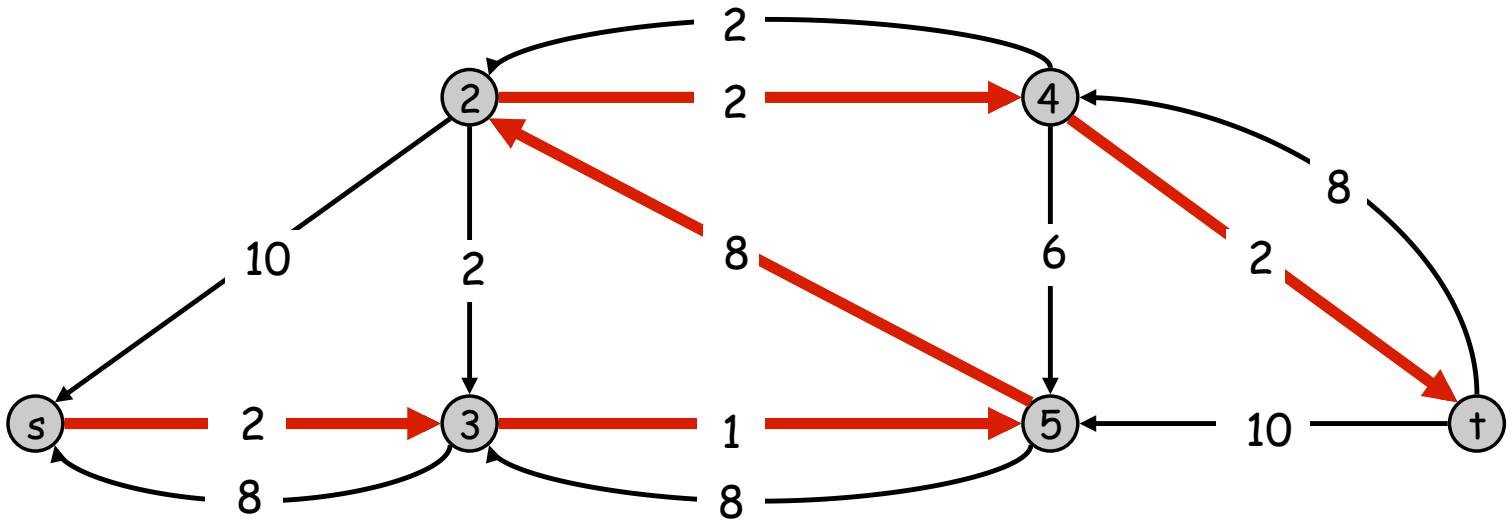
Algoritmo Ford-Fulkerson: esempio

G:



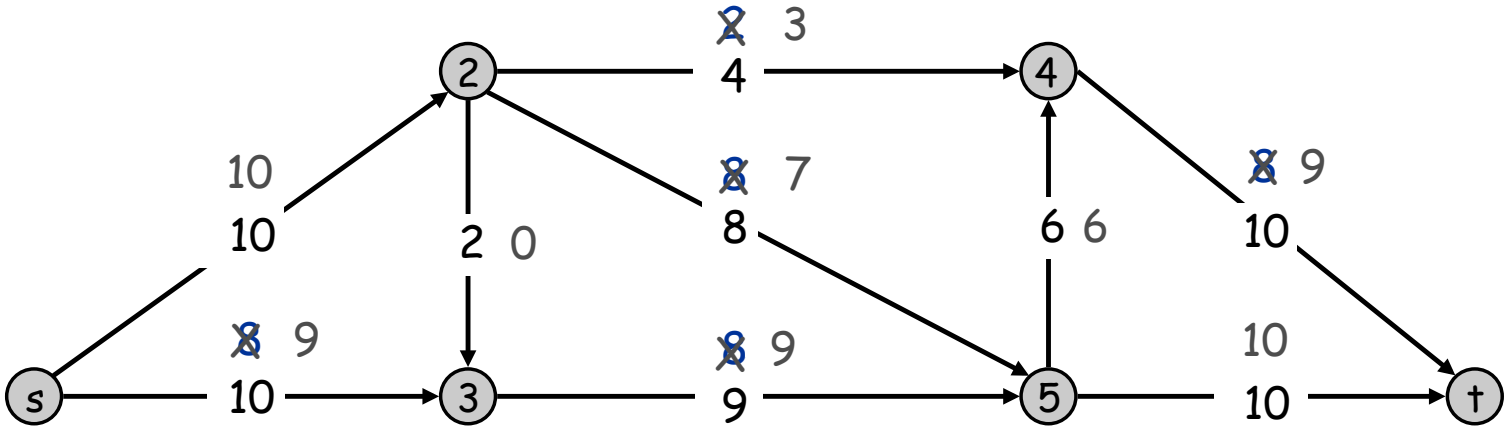
valore flusso = 18

G_f:



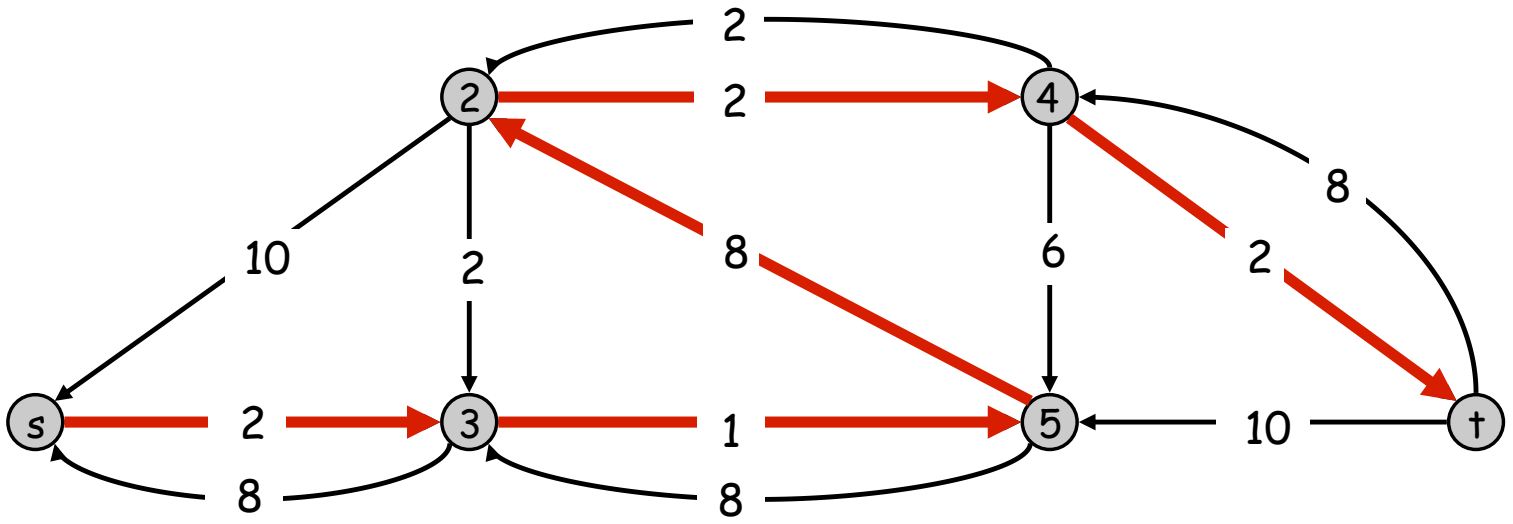
Algoritmo Ford-Fulkerson: esempio

G:

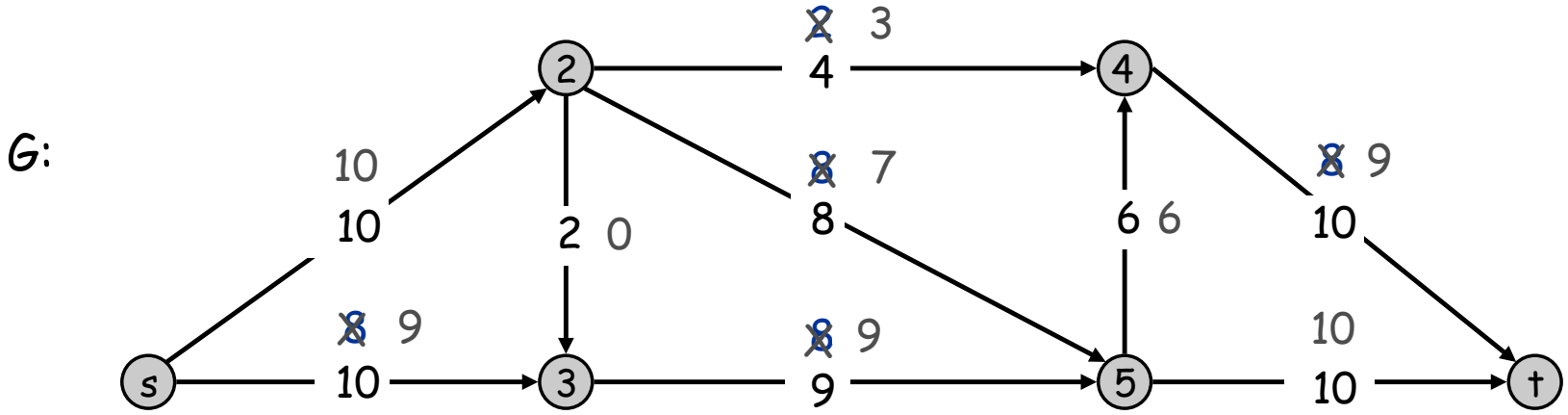


valore flusso = 18

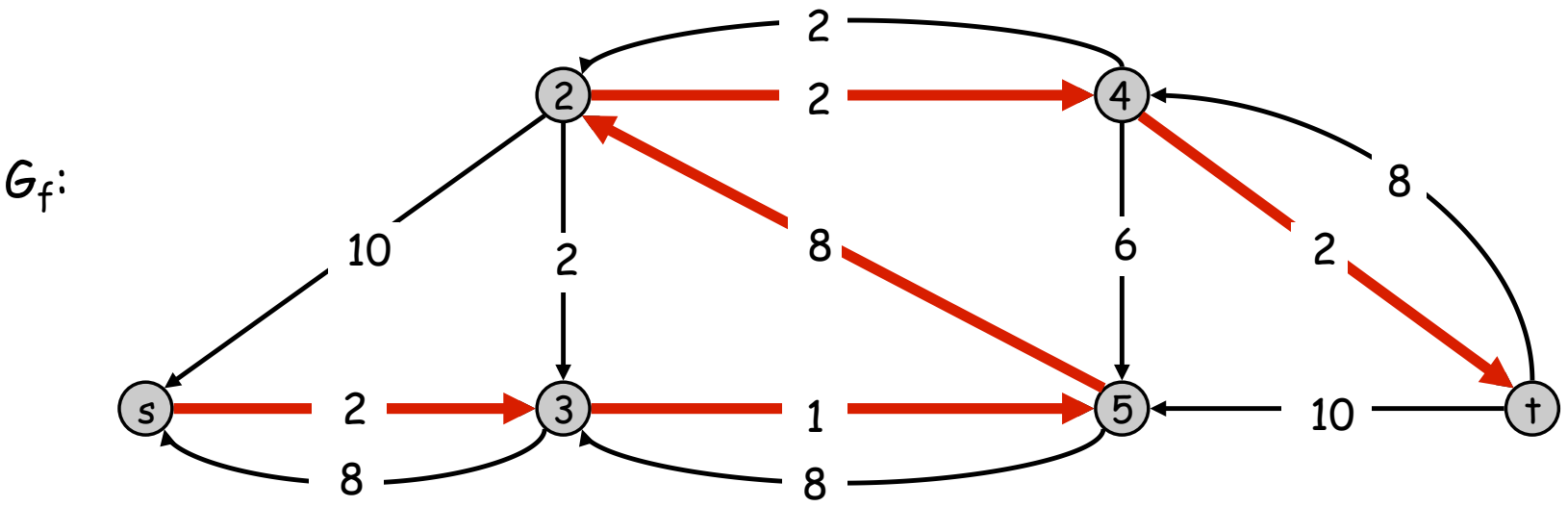
G_f:



Algoritmo Ford-Fulkerson: esempio

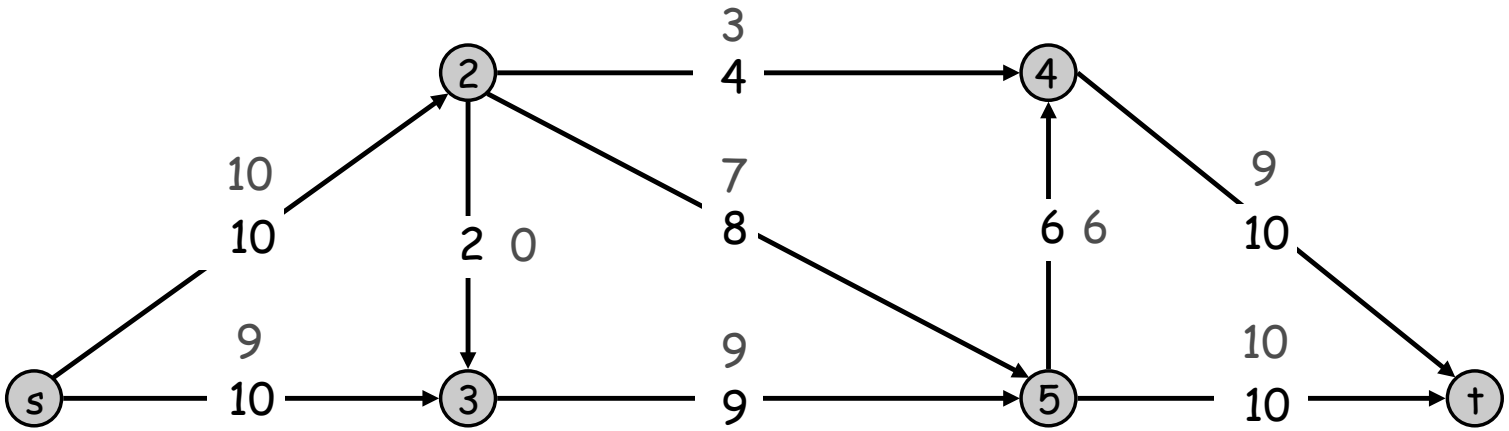


valore flusso = 19



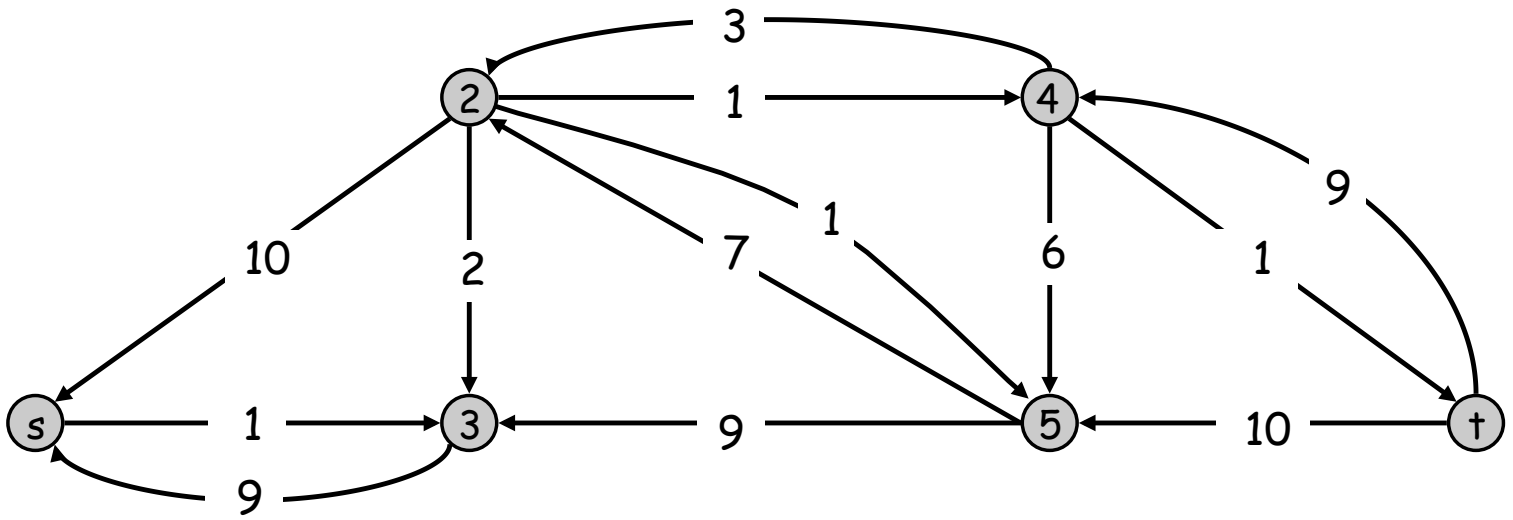
Algoritmo Ford-Fulkerson: esempio

G:



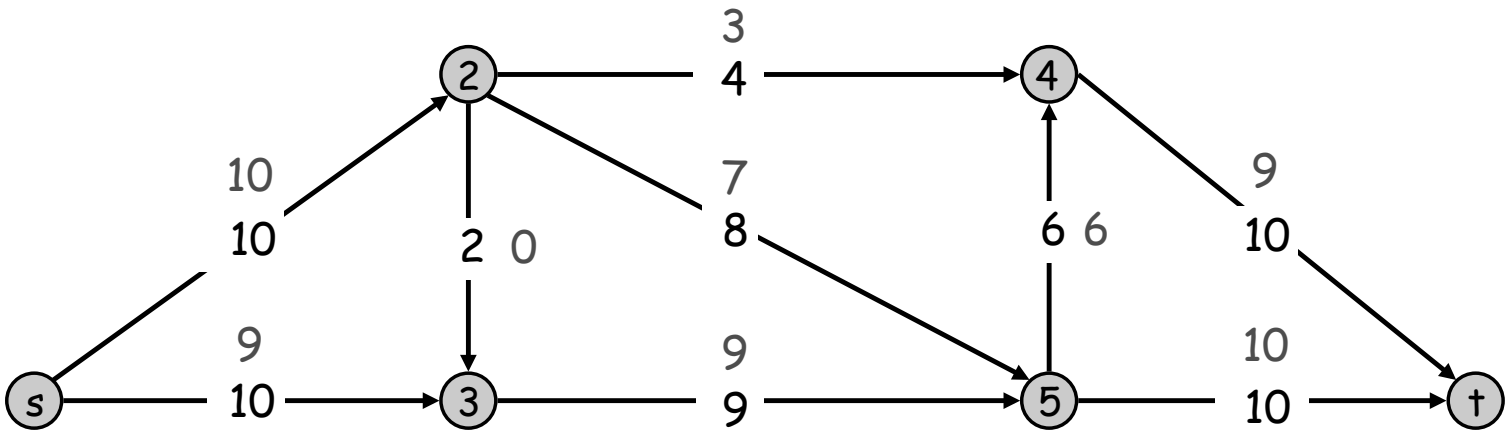
valore flusso = 19

G_f:



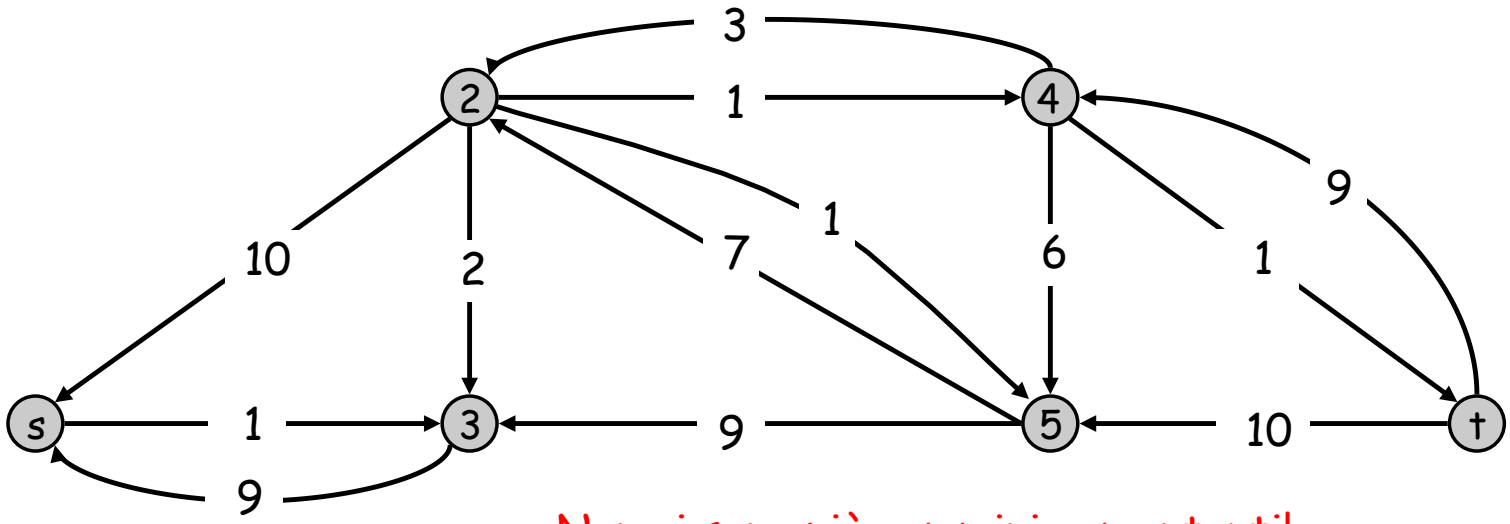
Algoritmo Ford-Fulkerson: esempio

G:



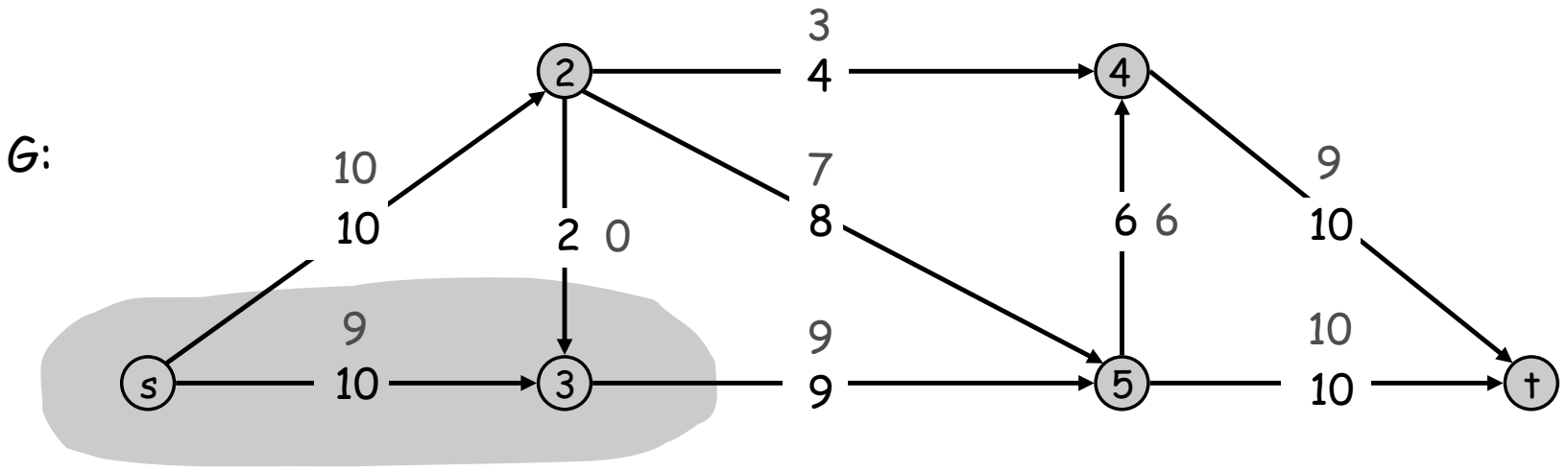
valore flusso = 19

G_f:



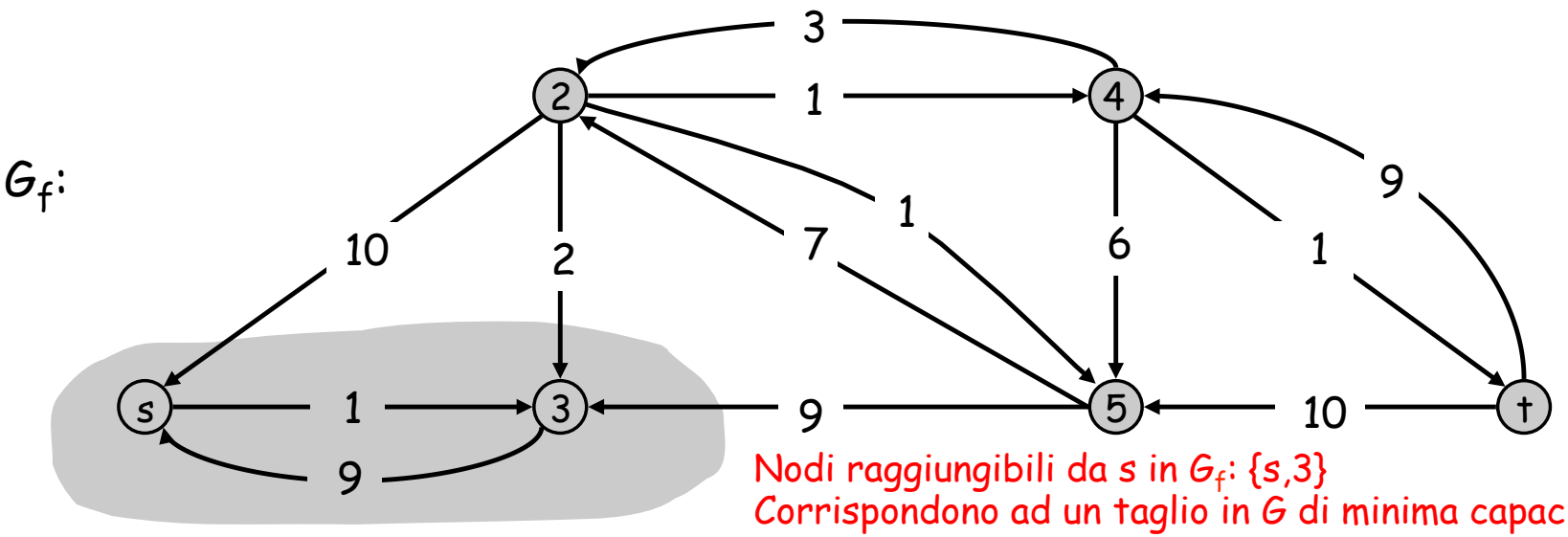
Non ci sono più cammini aumentanti!

Algoritmo Ford-Fulkerson: esempio



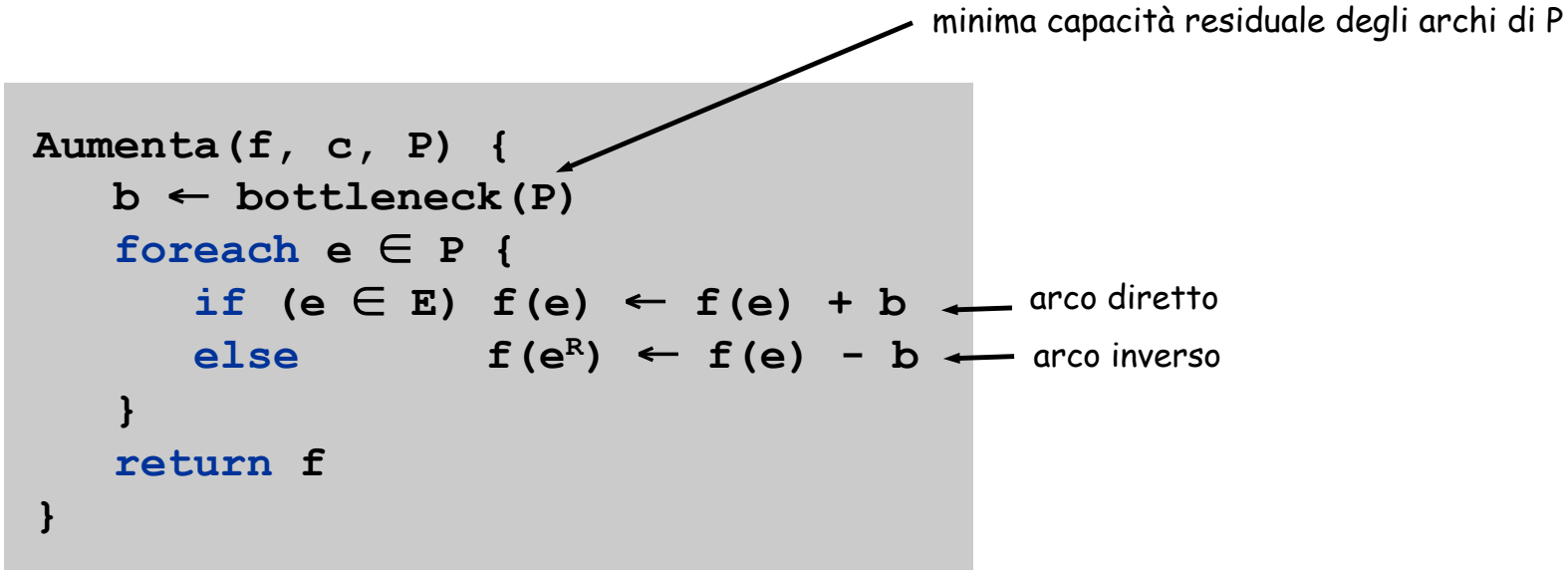
capacità taglio = 19

valore flusso = 19



Nodi raggiungibili da s in G_f: {s,3}
Corrispondono ad un taglio in G di minima capacità

Cammino aumentante: Algoritmo




```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   G_f ← grafo residuale

   while (c'è un cammino aumentante P) {
      f ← Aumenta(f, c, P)
      aggiorna G_f
   }
   return f
}
```

Teorema Max-Flow Min-Cut

Teorema del cammino aumentante. Flusso f è flusso massimo sse non ci sono cammini aumentanti.

Teorema Max-Flow Min-Cut. [Ford-Fulkerson 1956] Il valore del flusso massimo è uguale al valore del taglio minimo.

Prova. Dimostrazione di entrambi mostrando che sono equivalenti:

(i) C' è un taglio (A, B) tale che v(f) = cap(A, B).

(ii) Flusso f è un flusso massimo.

(iii) Non ci sono cammini aumentanti relativi ad f.

Teorema Max-Flow Min-Cut

Teorema del cammino aumentante. Flusso f è flusso massimo sse non ci sono cammini aumentanti.

Teorema Max-Flow Min-Cut. [Ford-Fulkerson 1956] Il valore del flusso massimo è uguale al valore del taglio minimo.

Prova. Dimostrazione di entrambi mostrando che sono equivalenti:

(i) C' è un taglio (A, B) tale che v(f) = cap(A, B).

(ii) Flusso f è un flusso massimo.

(iii) Non ci sono cammini aumentanti relativi ad f.

(i) $\Rightarrow$ (ii) Questo era il corollario al lemma della dualità debole.

Corollario. Sia f un flusso, e sia (A, B) un taglio.
Se v(f) = cap(A, B), allora f è un flusso massimo e (A, B) è un taglio minimo.

Teorema Max-Flow Min-Cut

Teorema del cammino aumentante. Flusso f è flusso massimo sse non ci sono cammini aumentanti.

Teorema Max-Flow Min-Cut. [Ford-Fulkerson 1956] Il valore del flusso massimo è uguale al valore del taglio minimo.

Prova. Dimostrazione di entrambi mostrando che sono equivalenti:

(i) C' è un taglio (A, B) tale che v(f) = cap(A, B).

(ii) Flusso f è un flusso massimo.

(iii) Non ci sono cammini aumentanti relativi ad f.

(i) $\Rightarrow$ (ii) Questo era il corollario al lemma della dualità debole.

(ii) $\Rightarrow$ (iii) Proviamo che not (iii) $\Rightarrow$ not (ii) .

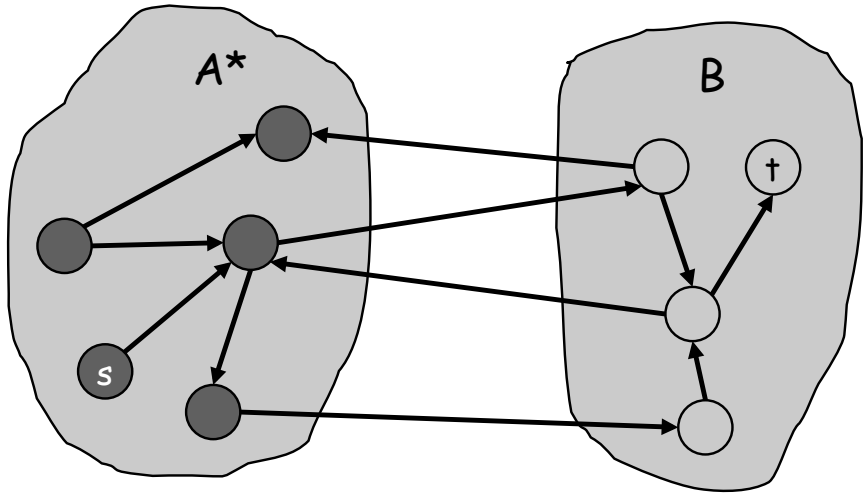
- Sia f un flusso. Se esistesse un cammino aumentante, allora potremo migliorare f inviando flusso sul cammino.

Prova del Teorema del Max-Flow Min-Cut

(iii) $\Rightarrow$ (i)

- Sia f un flusso che non ha cammini aumentanti.
- Sia A* l' insieme dei vertici raggiungibili da s nel grafo residuale.
- Per la definizione di A*, $s \in A^*$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A^*$.

$$v(f) = \sum_{e \text{ esce da } A^*} f(e) - \sum_{e \text{ entra in } A^*} f(e)$$



grafo originale

Prova del Teorema del Max-Flow Min-Cut

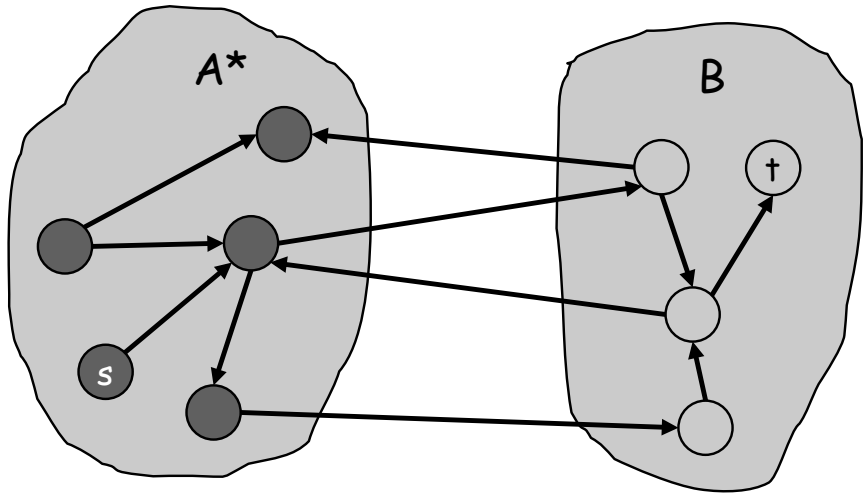
(iii) $\Rightarrow$ (i)

- Sia f un flusso che non ha cammini aumentanti.
- Sia A* l' insieme dei vertici raggiungibili da s nel grafo residuale.
- Per la definizione di A*, $s \in A^*$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A^*$.

$$v(f) = \sum_{e \text{ esce da } A^*} f(e) - \sum_{e \text{ entra in A*}} f(e)$$

Ricorda: definizione di arco residuale.
- Flusso di "undo".
- e = (u, v) e^R = (v, u).
- Capacità residuale:

$$c_f(e) = \begin{cases} c(e) - f(e) & \textbf{se } e \in E \\ f(e) & \textbf{se } e^R \in E \end{cases}$$



grafo originale

Prova del Teorema del Max-Flow Min-Cut

(iii) $\Rightarrow$ (i)

- Sia f un flusso che non ha cammini aumentanti.
- Sia A* l' insieme dei vertici raggiungibili da s nel grafo residuale.
- Per la definizione di A*, $s \in A^*$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A^*$.

$$v(f) = \sum_{e \text{ esce da } A^*} f(e) - \sum_{e \text{ entra in } A^*} f(e)$$

Sia (u,v) tale che $u \in A^*$ e $v \in B^*$.
Allora f(e) = c(e).
Se fosse f(e) < c(e) allora ci sarebbe un arco (u,v) nel grafo residuale e quindi $v \in A^*$.

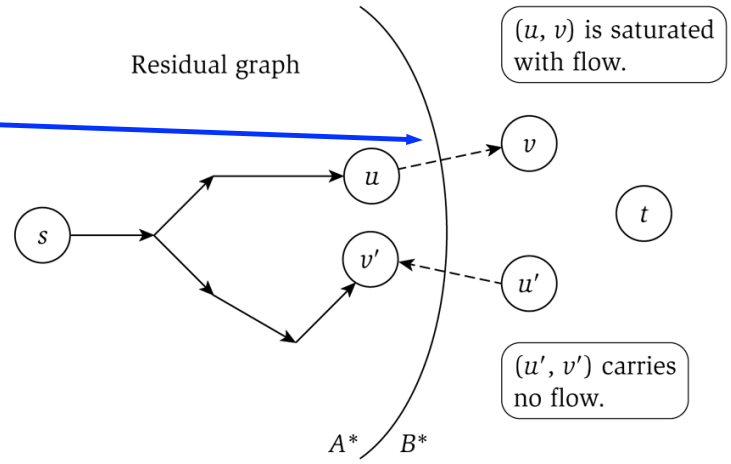


Figure 7.5 The (A^*, B^*) cut in the proof of (7.9).

Prova del Teorema del Max-Flow Min-Cut

(iii) $\Rightarrow$ (i)

- Sia f un flusso che non ha cammini aumentanti.
- Sia A* l' insieme dei vertici raggiungibili da s nel grafo residuale.
- Per la definizione di A*, $s \in A^*$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A^*$.

$$v(f) = \sum_{e \text{ esce da } A^*} f(e) - \sum_{e \text{ entra in } A^*} f(e)$$

Sia (u,v) tale che $u' \in B^*$ e $v' \in A^*$.
Allora f(e) = 0.
Se fosse f(e) > 0 allora ci sarebbe un arco (v',u') nel grafo residuale e quindi $u' \in A^*$.

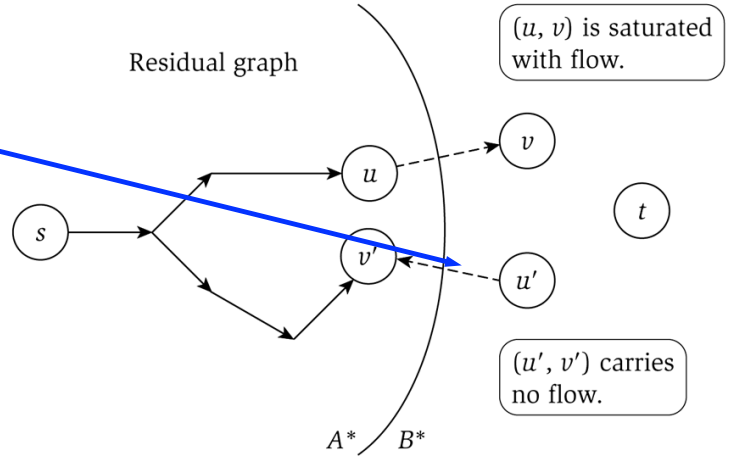


Figure 7.5 The (A^*, B^*) cut in the proof of (7.9).

Prova del Teorema del Max-Flow Min-Cut

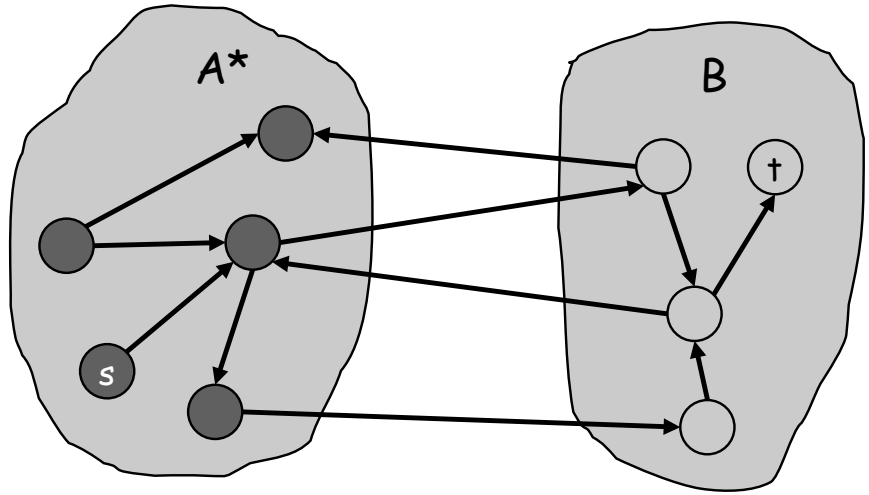
(iii) $\Rightarrow$ (i)

- Sia f un flusso che non ha cammini aumentanti.
- Sia A* l' insieme dei vertici raggiungibili da s nel grafo residuale.
- Per la definizione di A*, $s \in A$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A^*$.

$$v(f) = \sum_{e \text{ esce da } A^*} f(e) - \sum_{e \text{ entra in } A^*} f(e)$$

$$= \sum_{e \text{ esce da } A^*} c(e)$$

$$= cap(A^*, B)$$



grafo originale

Cammino aumentante: Complessità Algoritmo

complessità "bottleneck"

quante volte viene eseguito?

complessità tempo

```
Aumenta(f, c, P) {
    b ← bottleneck(P)
    foreach e ∈ P {
        if (e ∈ E) f(e) ← f(e) + b
        else       f(e^R) ← f(e) - b
    }
    return f
}
```

```
Ford-Fulkerson(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    G_f ← grafo residuale

    while (c'è un cammino aumentante P) {
        f ← Aumenta(f, c, P)
        aggiorna G_f
    }
    return f
}
```

Cammino aumentante: Complessità Algoritmo

```
Aumenta(f, c, P) {
    b ← bottleneck(P)
    foreach e ∈ P {
        if (e ∈ E) f(e) ← f(e) + b
        else       f(e^R) ← f(e) - b
    }
    return f
}
```

complessità "bottleneck"
$O(n)$

quante volte viene eseguito?
$O(n)$

complessità tempo
$O(n)$

```
Ford-Fulkerson(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    G_f ← grafo residuale

    while (c'è un cammino aumentante P) {
        f ← Aumenta(f, c, P)
        aggiorna G_f
    }
    return f
}
```

quante volte viene eseguito?

complessità

complessità "aggiorna G_f"

complessità tempo

Cammino aumentante: Complessità Algoritmo

complessità "bottleneck"
$O(n)$

quante volte viene eseguito?
$O(n)$

```
Aumenta(f, c, P) {
    b ← bottleneck(P)
    foreach e ∈ P {
        if (e ∈ E) f(e) ← f(e) + b
        else       f(e^R) ← f(e) - b
    }
    return f
}
```

complessità tempo
$O(n)$

tutti interi

```
Ford-Fulkerson(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    G_f ← grafo residuale

    while (c'è un cammino aumentante P) {
        f ← Aumenta(f, c, P)
        aggiorna G_f
    }
    return f
}
```

quante volte viene eseguito?
$\leq C$ volte, dove $C = \sum_{e \text{ esce da } s} c(e)$

complessità $O(m)$
ricerca depth-first o breadth-first in G_f

complessità "aggiorna G_f"
$O(m)$

complessità tempo
$O(mC)$

Complessità tempo

Limite superiore al valore del flusso: $C = \sum_{e \text{ esce da } s} c(e)$.

infatti $v(f) = \sum_{e \text{ esce da } s} f(e) \leq \sum_{e \text{ esce da } s} c(e)$

Assunzione. Tutte le capacità sono numeri interi tra 1 e C.

Invariante. Ogni valore del flusso f(e) e delle capacità residuali c(e) rimangono interi durante l'algoritmo.

Teorema. L'algoritmo termina in al massimo $v(f^*) \leq C$ iterazioni.
Prova. Ogni cammino aumentante incrementa il valore del flusso di almeno 1.

Teorema Integralità. Se tutte le capacità sono numeri interi, allora vi è un flusso massimo f per cui ogni valore del flusso f(e) è un intero.
Prova. L'algoritmo termina, quindi il teorema segue dall'invariante. ▪

Complessità tempo

Supponiamo che tutti i nodi abbiano almeno un arco incidente, quindi $m \geq n/2$.

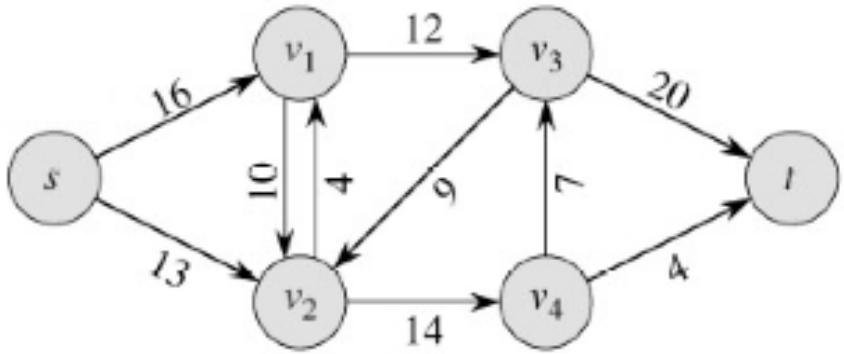
Corollario. Complessità di tempo dell'algoritmo di Ford-Fulkerson è $O(mC)$.

Prova.

- al massimo $v(f^*) \leq C$ iterazioni
- in ogni iterazione:
  - il grafo residuale ha $\leq 2m$ archi
  - grafo residuale rappresentato utilizzando la lista delle adiacenze
  - troviamo un cammino aumentante in $O(m)$ mediante ricerca depth-first oppure breadth-first nel grafo residuale.
  - `Aumenta(f,c,P)` ha complessità $O(n)$
  - aggiornamento grafo residuale in $O(m)$

Esercizio

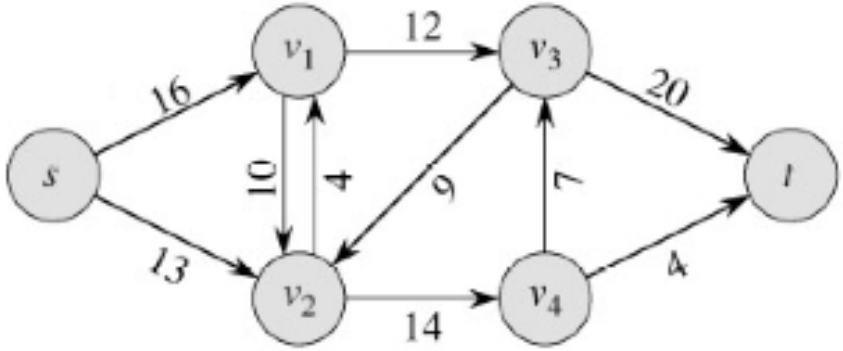
Si esegua l'algoritmo per il calcolo del flusso massimo sul grafo



- Si evidenzino per ogni singolo passo effettuato quale è l'augmenting path utilizzata, il flusso ed il grafo residuale rispetto al flusso.
- Si determini un taglio minimo.
- Si argomenti sul perchè il flusso ottenuto è massimo analizzando la sua relazione con tale taglio minimo.

Esercizio

Si esegua l'algoritmo per il calcolo del flusso massimo sul grafo



- Si evidenzino per ogni singolo passo effettuato quale è l'augmenting path utilizzata, il flusso ed il grafo residuale rispetto al flusso.
- Si determini un taglio minimo.
- Si argomenti sul perchè il flusso ottenuto è massimo analizzando la sua relazione con tale taglio minimo.

(Soluzione per il valore del flusso massimo: 23)

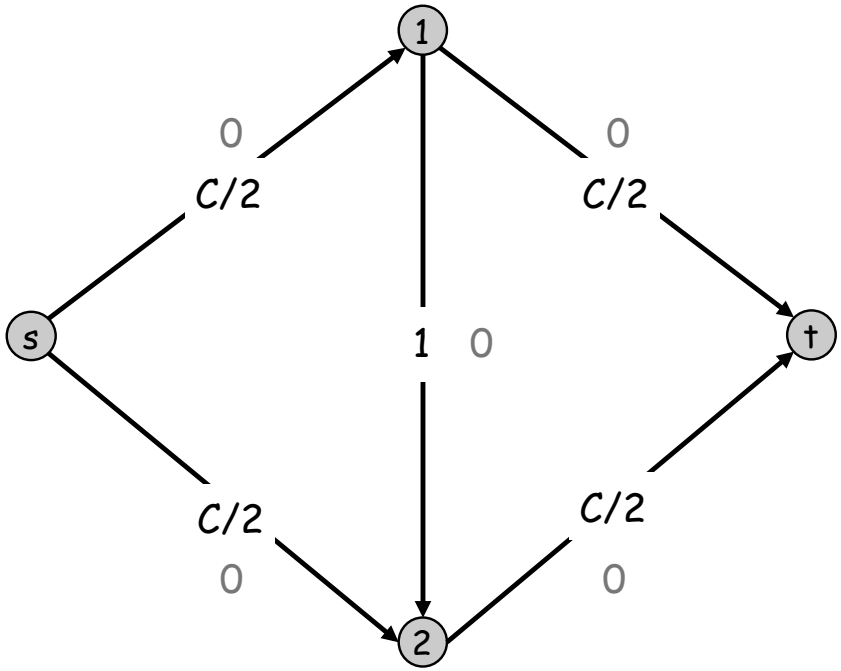
7.3 Choosing Good Augmenting Paths

Ford-Fulkerson: Numero esponenziale di passi

Domanda. L'algoritmo di Ford-Fulkerson è polinomiale nella taglia dell'input?

m, n, e log C

Risposta. No. L'algoritmo può fare al massimo C iterazioni.

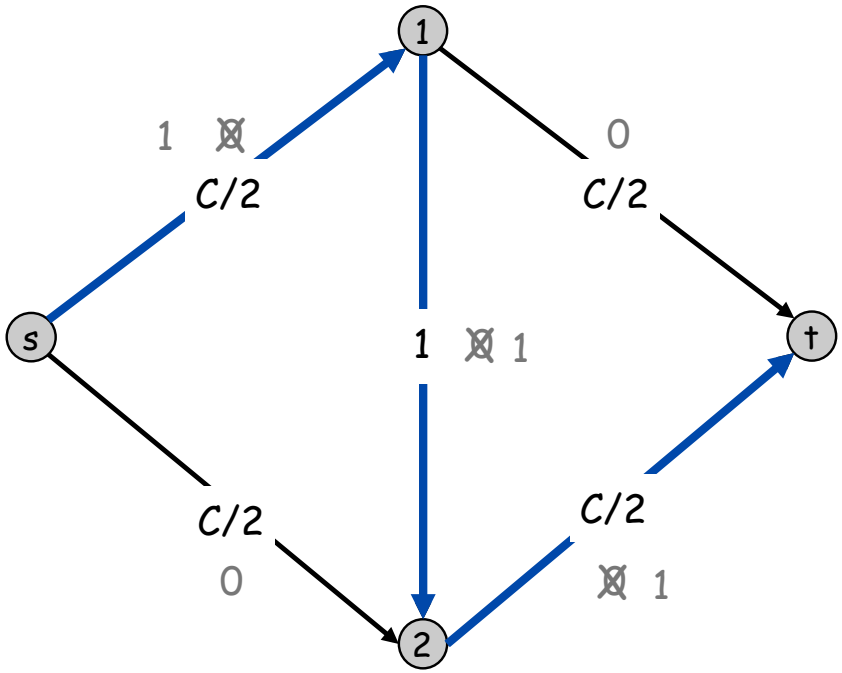


Ford-Fulkerson: Numero esponenziale di passi

Domanda. L'algoritmo di Ford-Fulkerson è polinomiale nella taglia dell'input?

m, n, e log C

Risposta. No. L'algoritmo può fare al massimo C iterazioni.

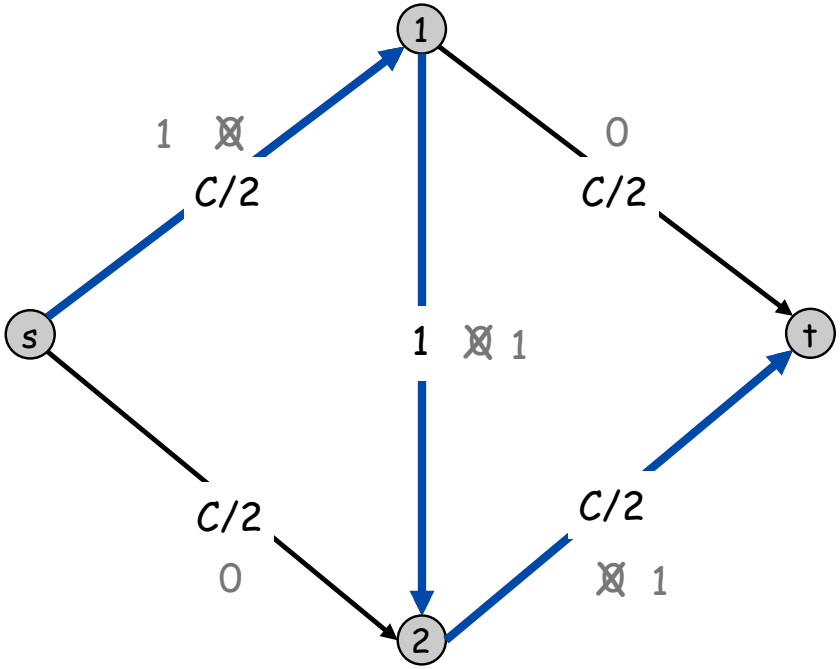


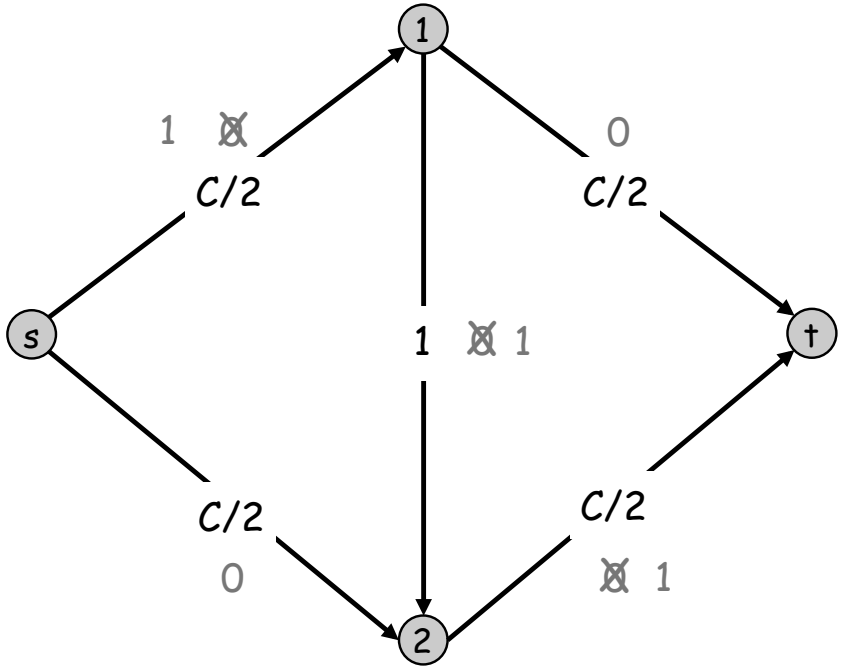
Ford-Fulkerson: Numero esponenziale di passi

Domanda. L'algoritmo di Ford-Fulkerson è polinomiale nella taglia dell'input?

m, n, e log C

Risposta. No. L'algoritmo può fare al massimo C iterazioni.



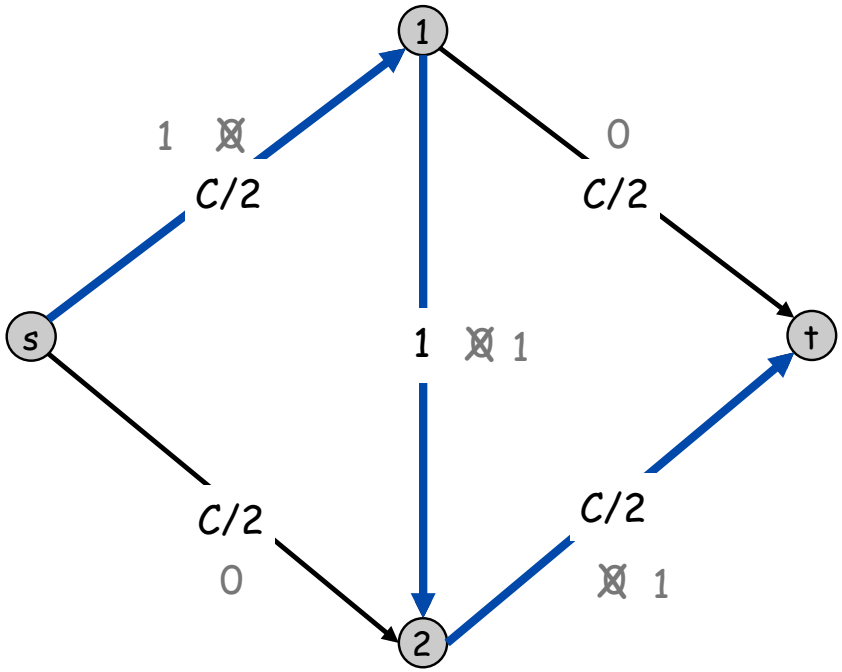


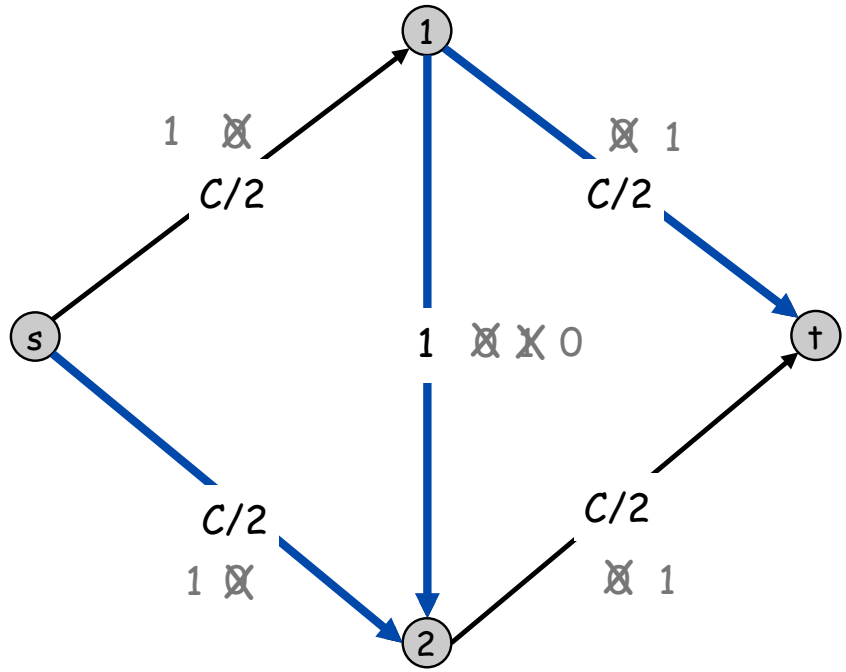
Ford-Fulkerson: Numero esponenziale di passi

Domanda. L'algoritmo di Ford-Fulkerson è polinomiale nella taglia dell'input?

m, n, e log C

Risposta. No. L'algoritmo può fare al massimo C iterazioni.





Scegliere Buoni Cammini Aumentanti

Fare attenzione quando si scelgono i cammini aumentanti.

- Alcune scelte portano ad algoritmi esponenziali.
- Buone scelte portano ad algoritmi polinomiali.
- Se le capacità fossero irrazionali, l'algoritmo potrebbe non terminare!

Obiettivo: scegliere cammini aumentanti in modo tale che:

- Possiamo trovare cammini aumentanti efficientemente.
- Poche iterazioni.

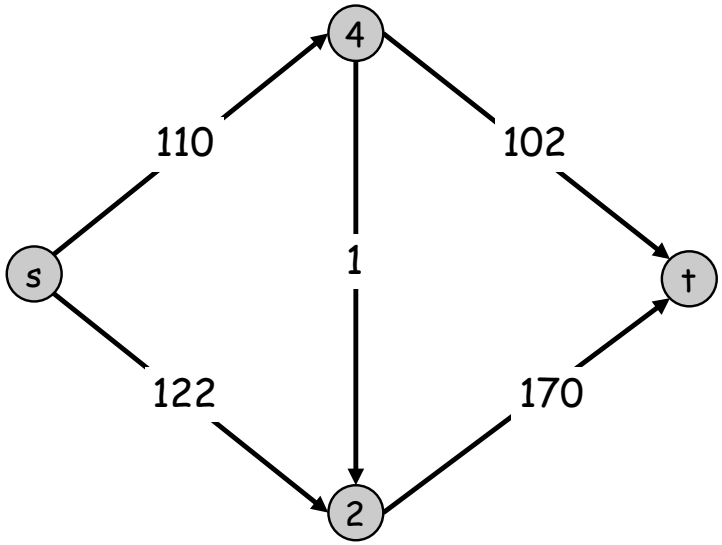
Scegliere cammini aumentanti con: [Edmonds-Karp 1972, Dinitz 1970]

- Massima capacità *bottleneck*.
- Sufficientemente grande capacità *bottleneck*.
- Minor numero di archi.

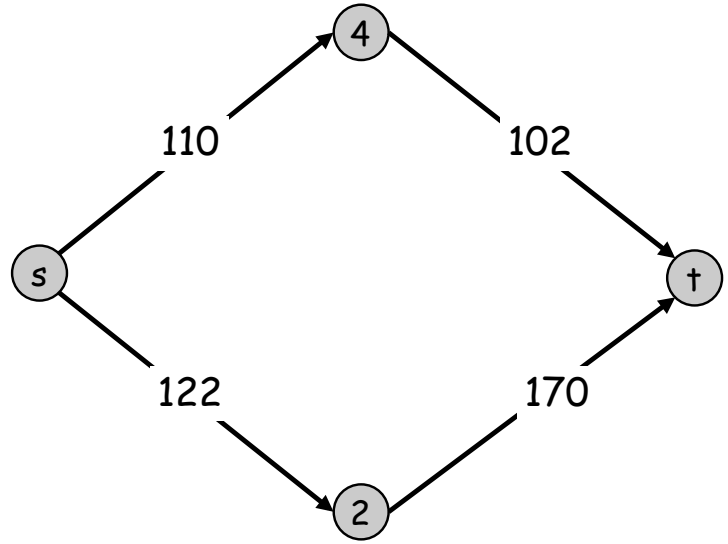
Capacity Scaling

Intuizione. La scelta di cammini con la maggior capacità bottleneck incrementa il flusso del massimo ammontare possibile.

- Non bisogna preoccuparsi di trovare il cammino con *esattamente* la massima capacità bottleneck.
- Mantenere un parametro di scaling Δ.
- Sia $G_f(\Delta)$ il sottografo del grafo residuale con i soli archi avente capacità almeno Δ.



G_f



$G_f(100)$

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← più piccola potenza di 2 che è ≤ max_{e esce da s} c(e)
   G_f ← grafo residuale

   while (Δ ≥ 1) {
      G_f(Δ) ← grafo Δ-residuale
      while (vi è un cammino aumentante P in G_f(Δ)) {
         f ← aumenta(f, c, P)
         aggiorna G_f(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

Capacity Scaling: Correttezza

Assunzione. Tutte le capacità degli archi sono interi tra 1 e $C = \sum_{e \text{ esce da } s} c(e)$.

Invariante dei valori interi. Tutti i valori dei flussi e delle capacità residuali sono interi.

Correttezza. Se l'algoritmo termina, allora f è un flusso massimo.

Prova.

- Dall'invariante dei valori interi, quando $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Dopo la fine della fase $\Delta = 1$, non vi sono cammini aumentanti. ▪

In realtà l'algoritmo Scaling-Max-Flow è solo una implementazione dell'algoritmo di Ford-Fulkerson in cui si fa una scelta guidata dei cammini aumentanti.

Capacity Scaling: complessità

```
Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    Δ ← più piccola potenza di 2 che è ≤ max_{e esce da s} c(e)
    G_f ← grafo residuale

    while (Δ ≥ 1) {
        G_f(Δ) ← grafo Δ-residuale
        while (vi è un cammino aumentante P in G_f(Δ)) {
            f ← aumenta(f, c, P)
            aggiorna G_f(Δ)
        }
        Δ ← Δ / 2
    }
    return f
}
```

quante volte viene eseguito?

quante volte viene eseguito?

Capacity Scaling: complessità

```
Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    Δ ← più piccola potenza di 2 che è ≤ max_{e esce da s} c(e)
    G_f ← grafo residuale

    while (Δ ≥ 1) {
        G_f(Δ) ← grafo Δ-residuale
        while (vi è un cammino aumentante P in G_f(Δ)) {
            f ← aumenta(f, c, P)
            aggiorna G_f(Δ)
        }
        Δ ← Δ / 2
    }
    return f
}
```

quante volte viene eseguito?
$\leq 1 + \lceil \log_2 C \rceil$ volte, dove $C = \sum_{e \text{ esce da } s} c(e)$

quante volte viene eseguito?
≤2m

numero totale di "aumenti"

Capacity Scaling: complessità

```
Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    Δ ← più piccola potenza di 2 che è ≤ max_{e esce da s} c(e)
    G_f ← grafo residuale

    while (Δ ≥ 1) {
        G_f(Δ) ← grafo Δ-residuale
        while (vi è un cammino aumentante P in G_f(Δ)) {
            f ← aumenta(f, c, P)
            aggiorna G_f(Δ)
        }
        Δ ← Δ / 2
    }
    return f
}
```

quante volte viene eseguito?
$\leq 1 + \lceil \log_2 C \rceil$ volte, dove $C = \sum_{e \text{ esce da } s} c(e)$

quante volte viene eseguito?
$\leq 2m$

numero totale di "aumenti"
$2m(1 + \lceil \log_2 C \rceil) = O(m \log C)$

Capacity Scaling: complessità

```
Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    Δ ← più piccola potenza di 2 che è ≤ max_{e esce da s} c(e)
    G_f ← grafo residuale

    while (Δ ≥ 1) {
        G_f(Δ) ← grafo Δ-residuale
        while (vi è un cammino aumentante P in G_f(Δ)) {
            f ← aumenta(f, c, P)
            aggiorna G_f(Δ)
        }
        Δ ← Δ / 2
    }
    return f
}
```

quante volte viene eseguito?
$\leq 1 + \lceil \log_2 C \rceil$ volte, dove $C = \sum_{e \text{ esce da } s} c(e)$

quante volte viene eseguito?
$\leq 2m$

numero totale di "aumenti" ?
$2m(1 + \lceil \log_2 C \rceil) = O(m \log C)$

complessità tempo?

Capacity Scaling: complessità

```
Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    Δ ← più piccola potenza di 2 che è ≤ max_{e esce da s} c(e)
    G_f ← grafo residuale

    while (Δ ≥ 1) {
        G_f(Δ) ← grafo Δ-residuale
        while (vi è un cammino aumentante P in G_f(Δ)) {
            f ← aumenta(f, c, P)
            aggiorna G_f(Δ)
        }
        Δ ← Δ / 2
    }
    return f
}
```

quante volte viene eseguito?
$\leq 1 + \lceil \log_2 C \rceil$ volte, dove $C = \sum_{e \text{ esce da } s} c(e)$

quante volte viene eseguito?
$\leq 2m$

numero totale di "aumenti" ?
$2m(1 + \lceil \log_2 C \rceil) = O(m \log C)$

complessità tempo
$O(m^2 \log C)$

Capacity Scaling: Complessità

Lemma. Il ciclo while esterno si ripete ≤ 1 + $\lceil \log_2 C \rceil$ volte, dove $C = \sum_{e \text{ esce da } s} c(e)$.

Prova. Inizialmente $\Delta < C$.

Poi, Δ decresce di un fattore 2 ad ogni iterazione.

Lemma. Sia f un flusso alla fine di un ciclo while esterno. Allora il valore del flusso massimo è $v(f_{max}) \le v(f) + m\,\Delta$. ← prova nella prossima slide

Lemma. Ci sono ≤2m "aumenti" in ogni iterazione del ciclo while esterno.

- Sia f il flusso alla fine del ciclo while esterno precedente.
- Dal Lemma precedente ⇒ $v(f_{max}) \le v(f) + m\,(2\Delta)$.
- Ogni aumento incrementa v(f) di almeno Δ. ▪

Teorema. L'algoritmo **`Scaling-Max-Flow`** calcola un flusso massimo con $2m(1 + \lceil \log_2 C \rceil) = O(m \log C)$ "aumenti" (ovvero chiamate di **`aumenta`**).
Può essere implementato con una complessità di tempo $O(m^2 \log C)$. ▪

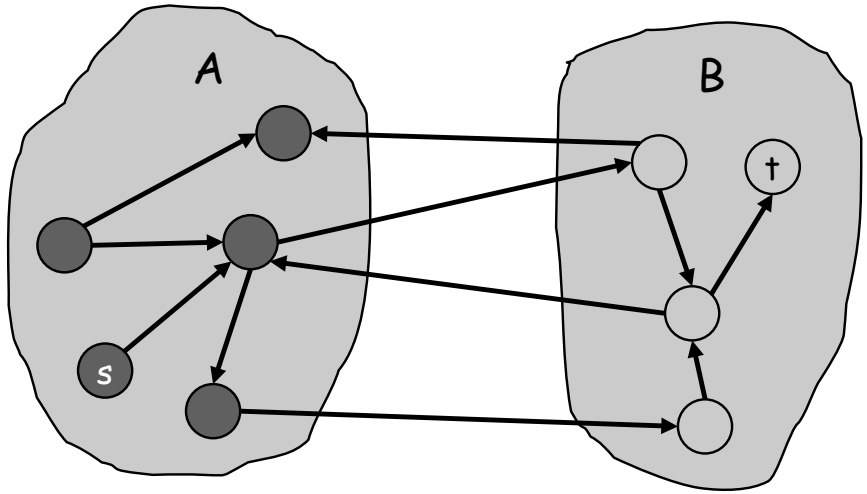
Capacity Scaling: Complessità

Lemma . Sia f un flusso alla fine di un ciclo while esterno. Allora il valore del flusso massimo f* è $v(f^*) \le v(f) + m\,\Delta$.

Prova. (quasi identica alla prova del teorema max-flow min-cut)

- Mostriamo che alla fine del ciclo, c' è un taglio (A, B) tale che
$$cap(A, B) \le v(f) + m\,\Delta.$$
- Sia A l'insieme dei nodi raggiungibili da s in $G_f(\Delta)$.
- Per la definizione di A, $s \in A$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A$.

$$v(f) = \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in } A} f(e)$$



rete originale

Capacity Scaling: Complessità

Lemma . Sia f un flusso alla fine di un ciclo while esterno. Allora il valore del flusso massimo f* è $v(f^*) \le v(f) + m\,\Delta$.

Prova. (quasi identica alla prova del teorema max-flow min-cut)

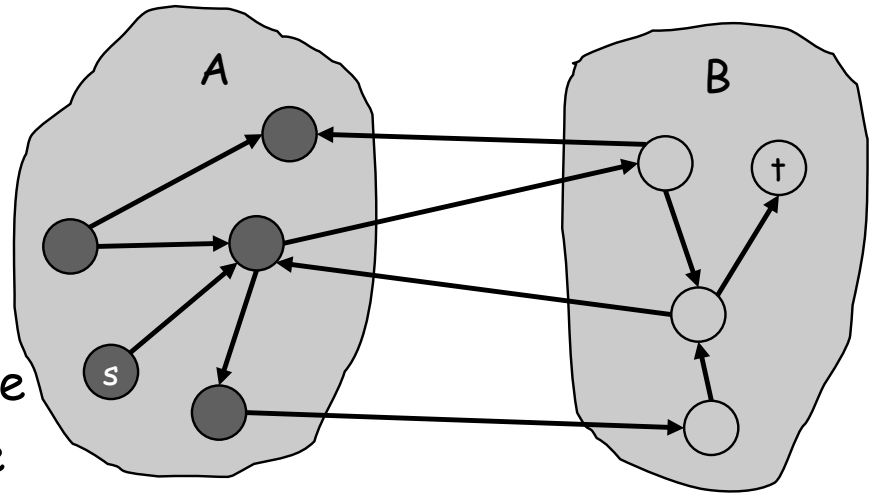
- Mostriamo che alla fine del ciclo, c' è un taglio (A, B) tale che
$$cap(A, B) \le v(f) + m\,\Delta.$$
- Sia A l'insieme dei nodi raggiungibili da s in $G_f(\Delta)$.
- Per la definizione di A, $s \in A$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A$.

$$v(f) = \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in } A} f(e)$$

Sia e=(u,v) tale che $u \in A$ e $v \in B$.

Allora $c(e) < f(e) + \Delta$.

Se fosse $c(e) \ge f(e) + \Delta$ allora ci sarebbe un arco (u,v) nel grafo residuale $G_f(\Delta)$ e quindi $v \in A$.



grafo originale

Capacity Scaling: Complessità

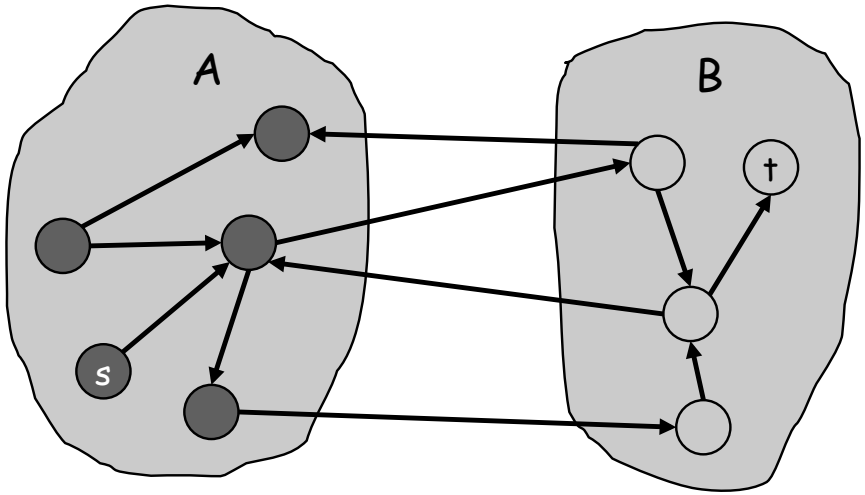
Lemma . Sia f un flusso alla fine di un ciclo while esterno. Allora il valore del flusso massimo f* è $v(f^*) \le v(f) + m\,\Delta$.

Prova. (quasi identica alla prova del teorema max-flow min-cut)

- Mostriamo che alla fine del ciclo, c' è un taglio (A, B) tale che
$$cap(A, B) \le v(f) + m\,\Delta.$$
- Sia A l'insieme dei nodi raggiungibili da s in $G_f(\Delta)$.
- Per la definizione di A, $s \in A$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A$.

$$v(f) = \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in } A} f(e)$$

Sia e' =(u',v') tale che $u' \in B$ e $v' \in A$.
Allora $f(e') < \Delta$.
Se fosse $f(e') \ge \Delta$ allora ci sarebbe un arco (v',u') nel grafo residuale $G_f(\Delta)$ e quindi $v' \in A$.



grafo originale

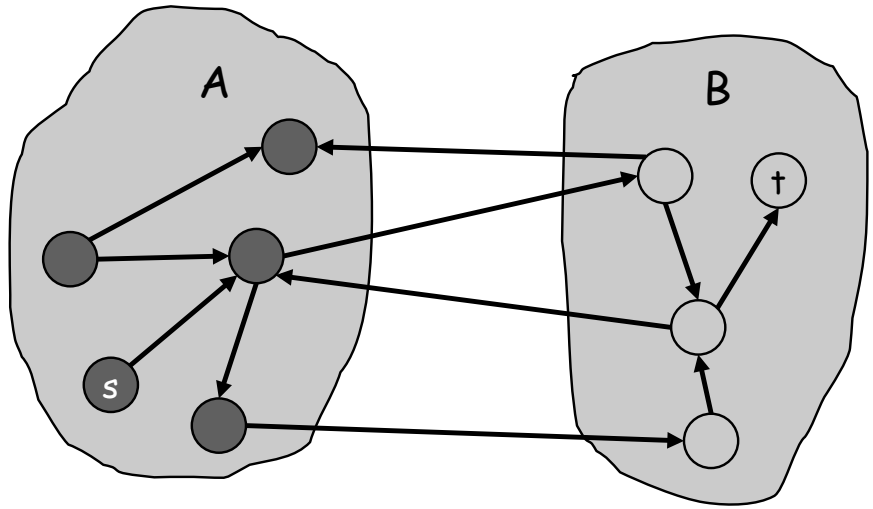
Capacity Scaling: Complessità

Lemma . Sia f un flusso alla fine di un ciclo while esterno. Allora il valore del flusso massimo f* è $v(f^*) \le v(f) + m\,\Delta$.

Prova. (quasi identica alla prova del teorema max-flow min-cut)

- Mostriamo che alla fine del ciclo, c' è un taglio (A, B) tale che
$$cap(A, B) \le v(f) + m\,\Delta.$$
- Sia A l'insieme dei nodi raggiungibili da s in $G_f(\Delta)$.
- Per la definizione di A, $s \in A$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A$.

$$\begin{aligned} v(f) &= \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in } A} f(e) \\ &\ge \sum_{e \text{ esce da } A} (c(e) - \Delta) - \sum_{e \text{ entra in } A} \Delta \\ &= \sum_{e \text{ esce da } A} c(e) - \sum_{e \text{ esce da } A} \Delta - \sum_{e \text{ entra in } A} \Delta \end{aligned}$$



grafo originale

Capacity Scaling: Complessità

Lemma . Sia f un flusso alla fine di un ciclo while esterno. Allora il valore del flusso massimo f* è $v(f^*) \le v(f) + m\,\Delta$.

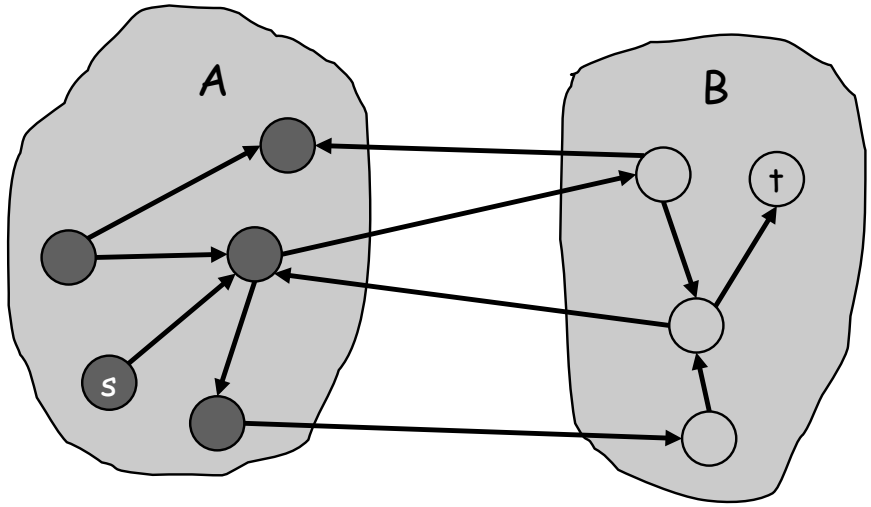
Prova. (quasi identica alla prova del teorema max-flow min-cut)

- Mostriamo che alla fine del ciclo, c'è un taglio (A, B) tale che
$$cap(A, B) \le v(f) + m\,\Delta.$$
- Sia A l'insieme dei nodi raggiungibili da s in $G_f(\Delta)$.
- Per la definizione di A, $s \in A$.
- Flusso f non ha cammini aumentanti $\Rightarrow$ $t \notin A$.

$$\begin{aligned} v(f) &= \sum_{e \text{ esce da } A} f(e) - \sum_{e \text{ entra in } A} f(e) \\ &\ge \sum_{e \text{ esce da } A} (c(e) - \Delta) - \sum_{e \text{ entra in } A} \Delta \\ &= \sum_{e \text{ esce da } A} c(e) - \sum_{e \text{ esce da } A} \Delta - \sum_{e \text{ entra in } A} \Delta \\ &\ge cap(A, B) - m\Delta \quad \blacksquare \end{aligned}$$

$$cap(A, B) = \sum_{e \text{ esce da } A} c(e)$$

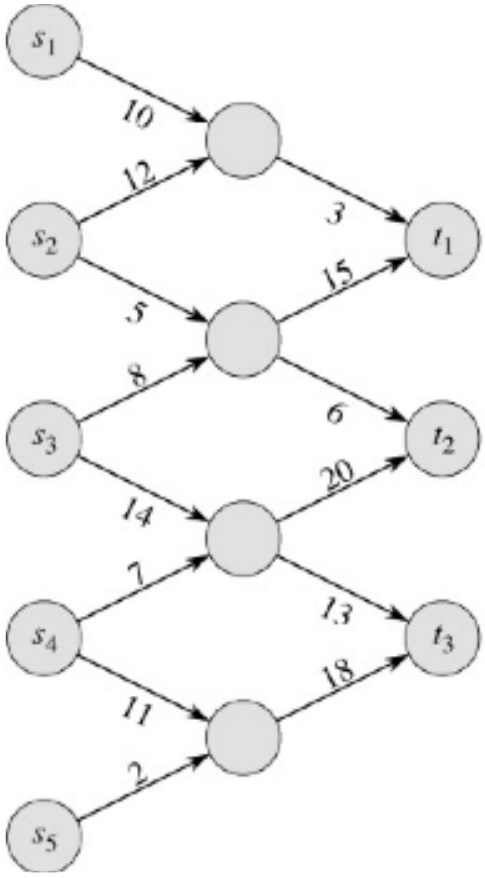
ci sono m archi



grafo originale

Rete con più sorgenti e pozzi

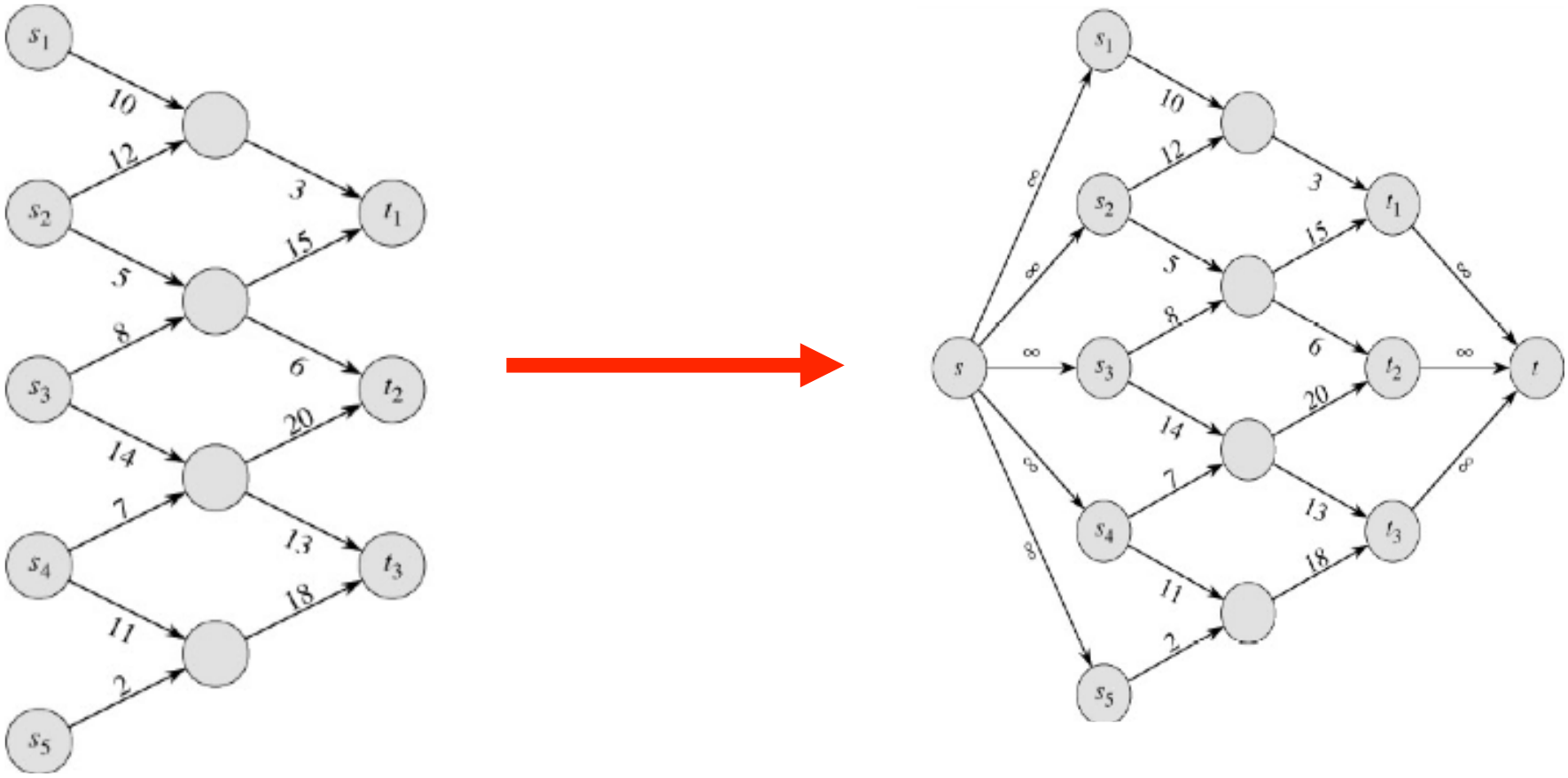
Problema: Vogliamo massimizzare il flusso totale da tutte le sorgenti a tutte le destinazioni



Rete con più sorgenti e pozzi

Problema: Vogliamo massimizzare il flusso totale da tutte le sorgenti a tutte le destinazioni

Soluzione: Riduciamo il problema al Maximum Flow creando una supersorgente ed un superpozzo



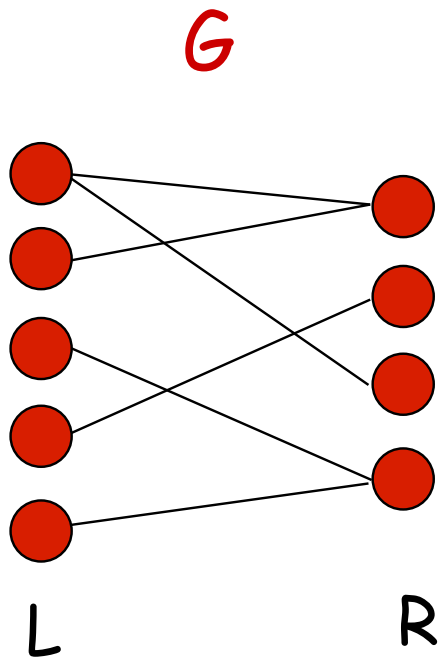
7.5 A First Application: The Bipartite Matching Problem

The Bipartite Matching Problem

Ricorda: Un grafo bipartito è un grafo *G=(V,E)* in cui V può essere diviso in due parti L ed R in modo tale che ogni arco in E collega un vertice in L con un vertice in R.

Potrebbe modellare:

- i vertici in L potrebbero rappresentare lavoratori ed i vertici in R lavori.
- gli archi connettono lavoratori a lavori che possono essere effettuati

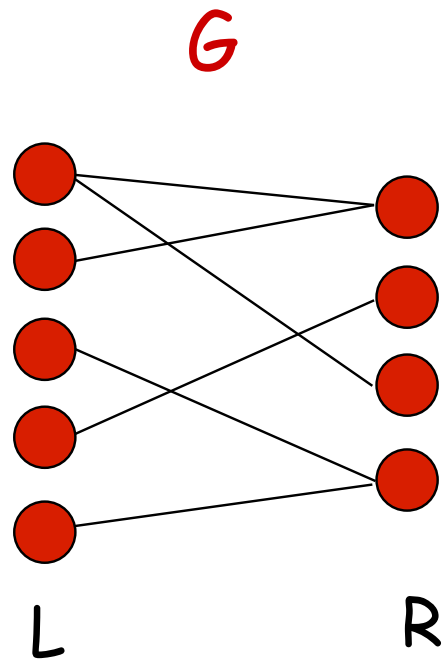


Maximum Bipartite Matching

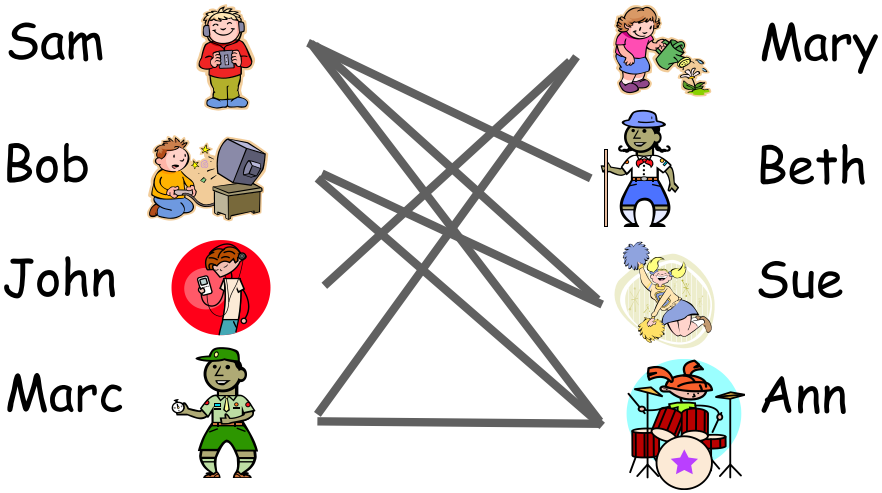
Ricorda: Un grafo bipartito è un grafo *G=(V,E)* in cui V può essere diviso in due parti L ed R in modo tale che ogni arco in E collega un vertice in L con un vertice in R.

Potrebbe modellare:

- i vertici in L potrebbero rappresentare lavoratori ed i vertici in R lavori.
- gli archi connettono lavoratori a lavori che possono essere effettuati



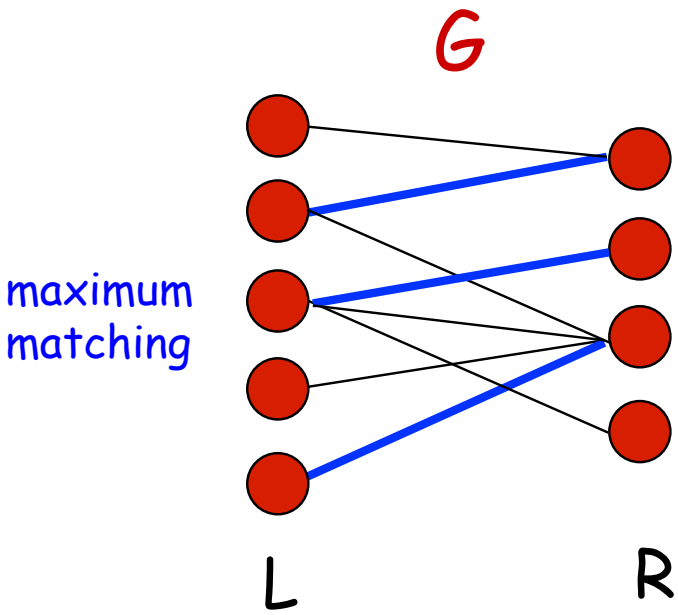
Oppure:

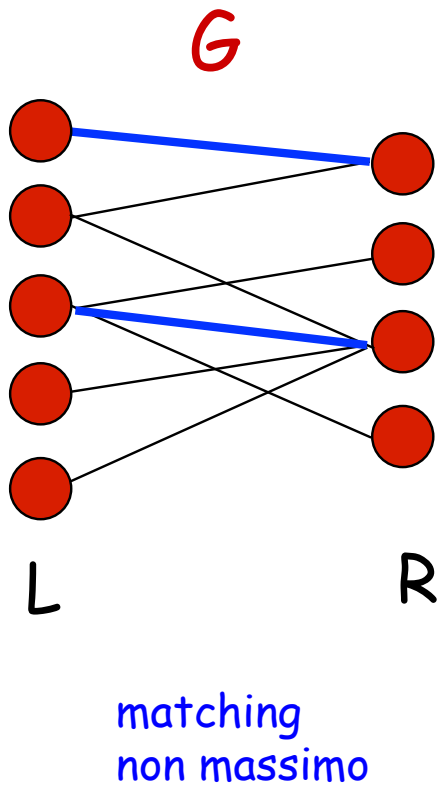


Maximum Bipartite Matching

Un matching in G è un sottoinsieme di archi M tale che ogni nodo compare in al massimo un arco in M.

Un maximum matching è un matching di massima cardinalità (massimo numero di archi).





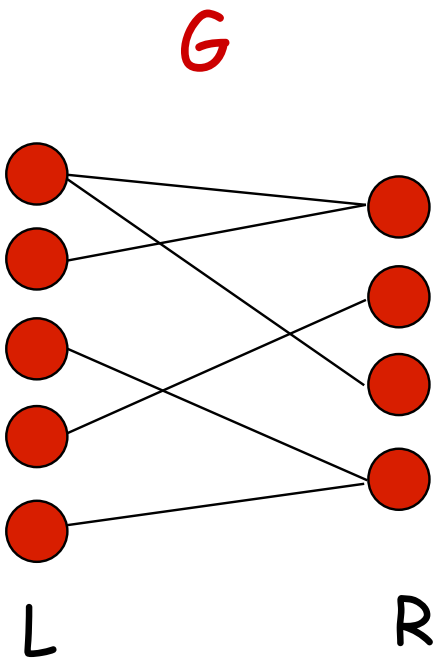
Maximum Bipartite Matching e Max-flow

Il problema del maximum bipartite matching su un grafo G si riduce al problema del max-flow su un grafo G' opportunamente costruito.

La soluzione al problema del max-flow su G' corrisponde ad una soluzione al problema del maximum bipartite matching su G.

Maximum Bipartite Matching e Max-flow

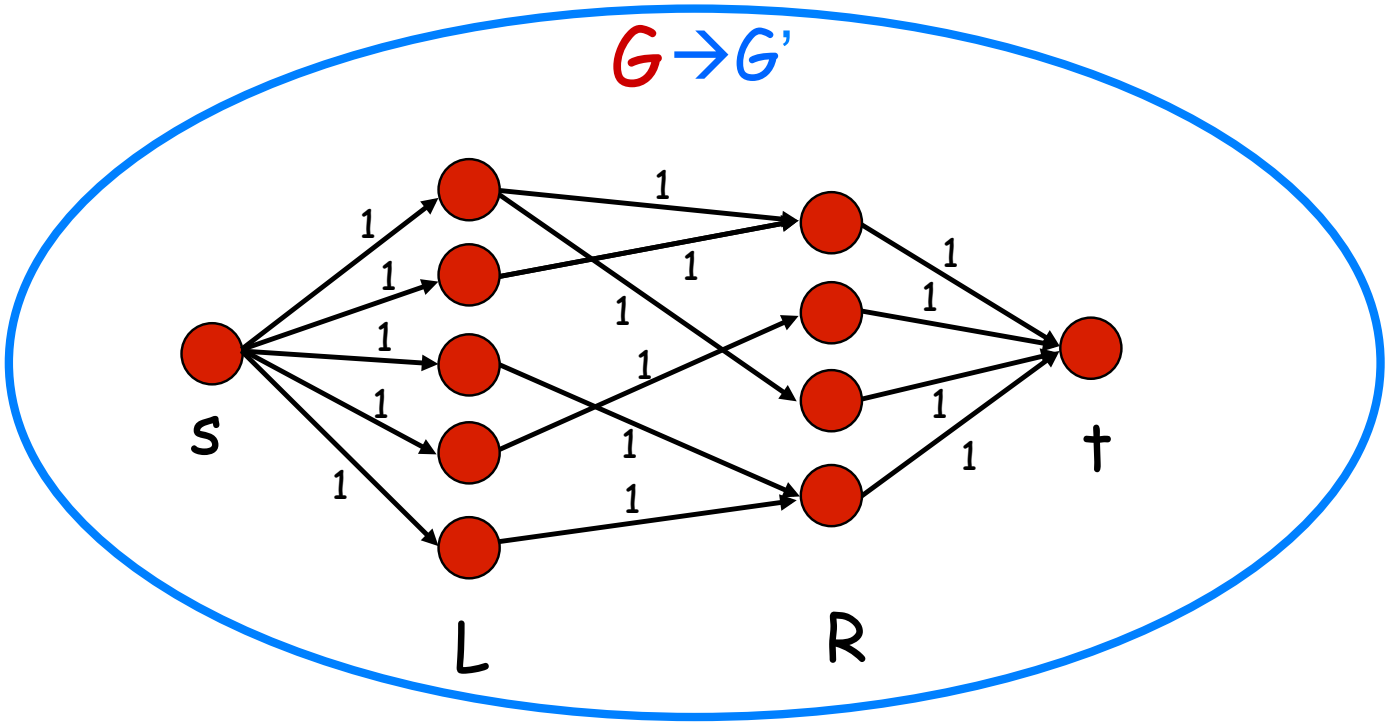
Costruzione di G' a partire da $G(V,E)$:



Maximum Bipartite Matching e Max-flow

Costruzione di G' a partire da G(V,E):

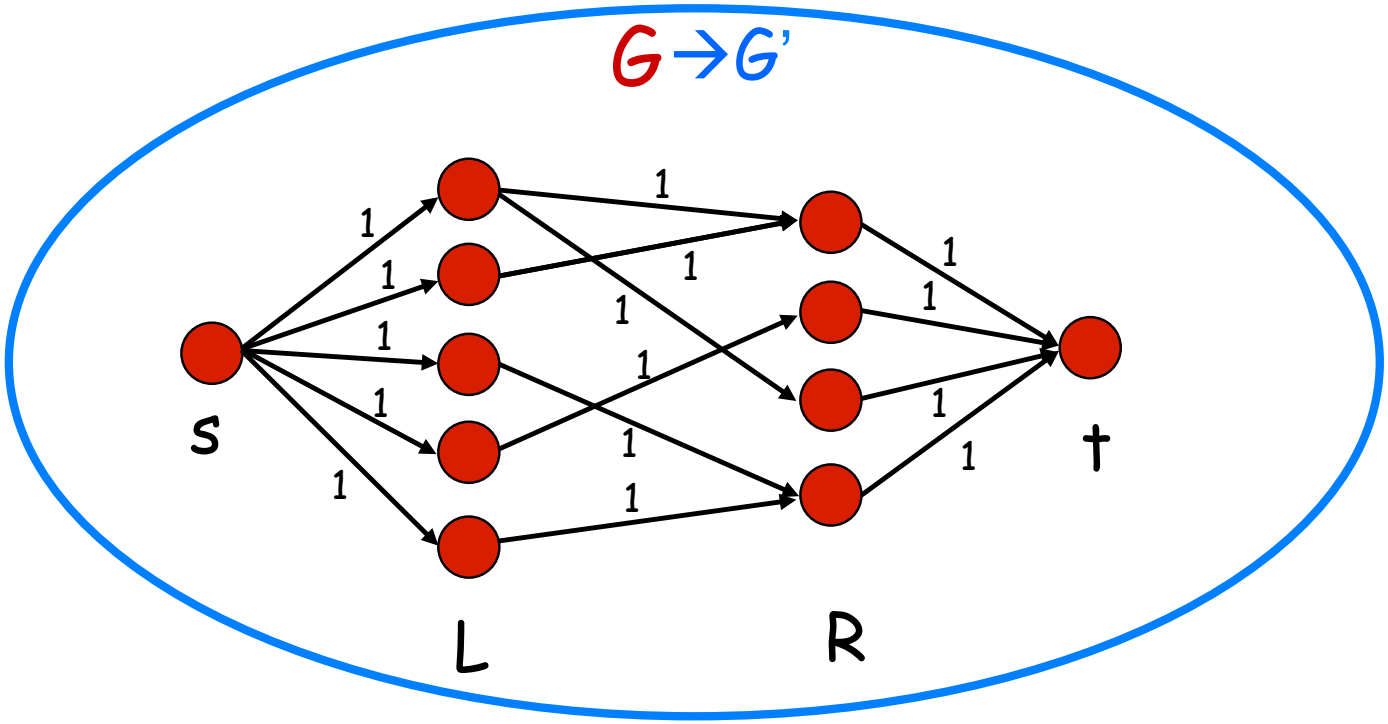
- Aggiungere un vertice sorgente s ed archi da s ad L
- Rendere diretti tutti gli archi in E: da L ad R
- Aggiungere un vertice pozzo ed archi da R a t
- Assegnare una capacità 1 a tutti gli archi



Maximum Bipartite Matching e Max-flow

Costruzione di G' a partire da G(V,E):

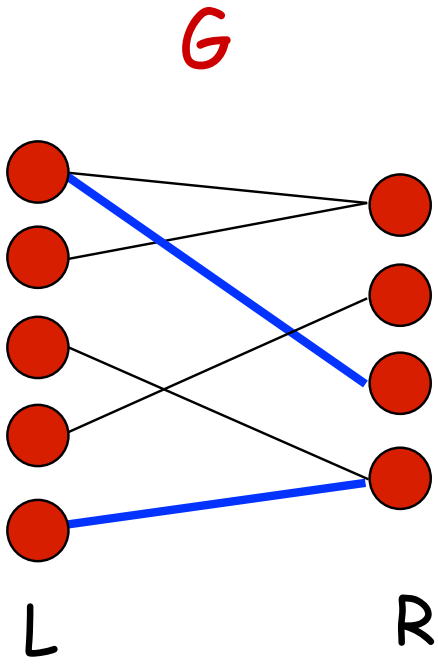
- Aggiungere un vertice sorgente s ed archi da s ad L
- Rendere diretti tutti gli archi in E: da L ad R
- Aggiungere un vertice pozzo ed archi da R a t
- Assegnare una capacità 1 a tutti gli archi



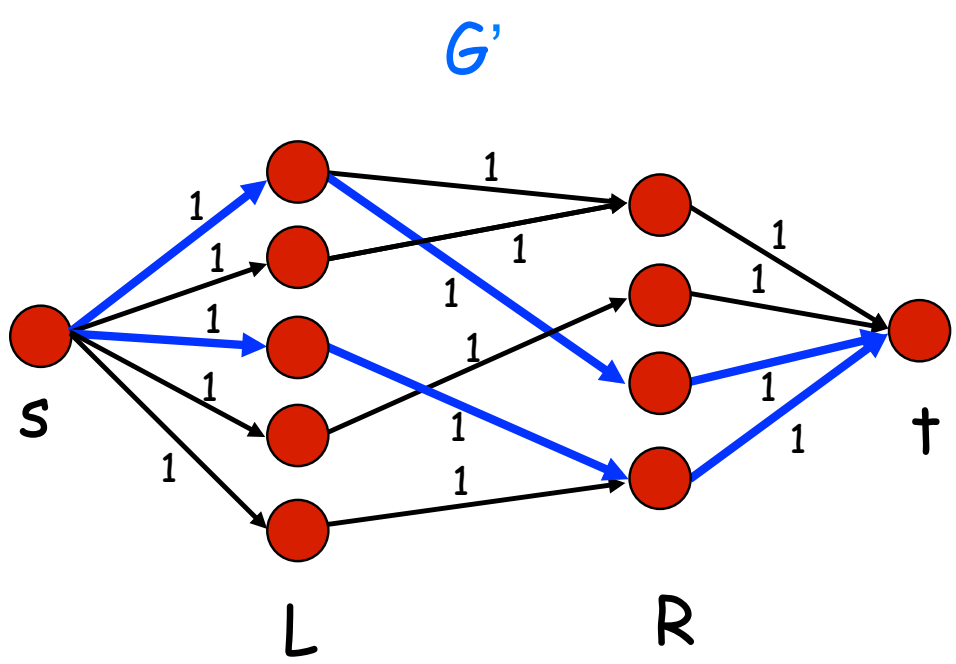
Maximum Bipartite Matching e Max-flow

Corrispondenza di matching in G con flussi in G'

- Per ogni matching di cardinalità |M| c'è un flusso di valore v(f)=|M| in G'



|M|=2

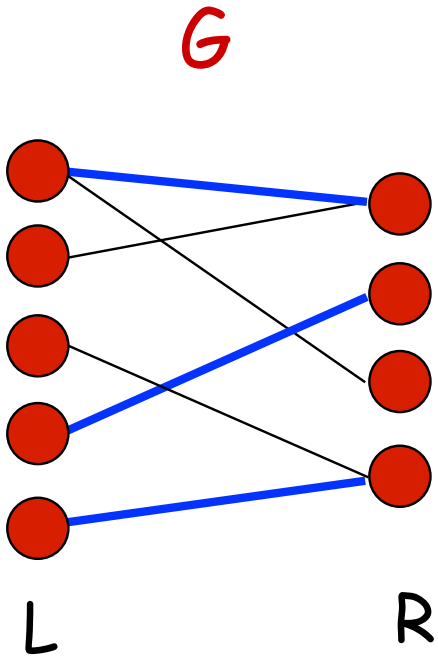


v(f)=2

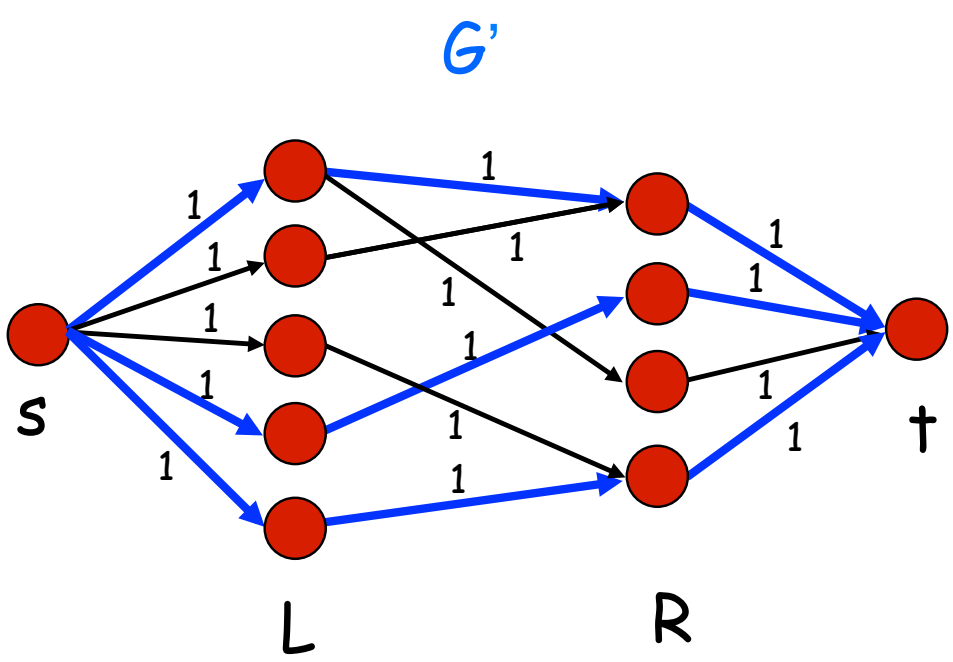
Maximum Bipartite Matching e Max-flow

Corrispondenza di matching in G con flussi in G'

- Per ogni matching di cardinalità |M| c'è un flusso di valore v(f)=|M| in G'



|M|=3



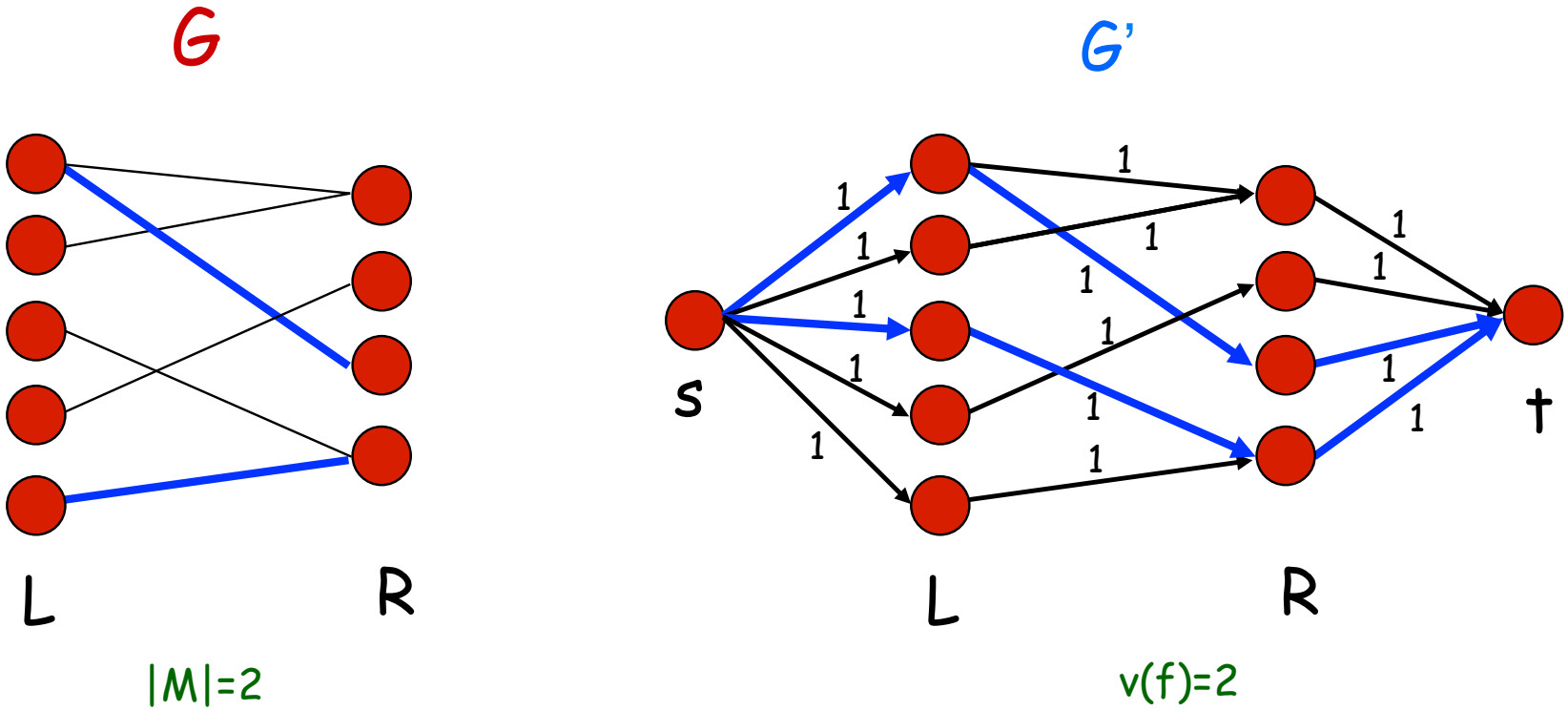
v(f)=3

Maximum Bipartite Matching e Max-flow

Corrispondenza di matching in G con flussi in G'

- Per ogni matching di cardinalità |M| c'è un flusso di valore v(f)=|M| in G'
- Per ogni flusso a valori interi in G' c'è un matching in G con |M|=v(f)

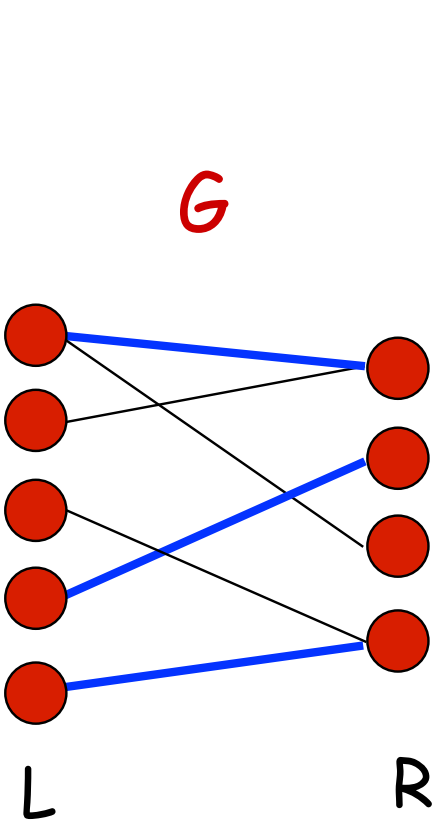
cioè f(e) è un intero per ogni arco

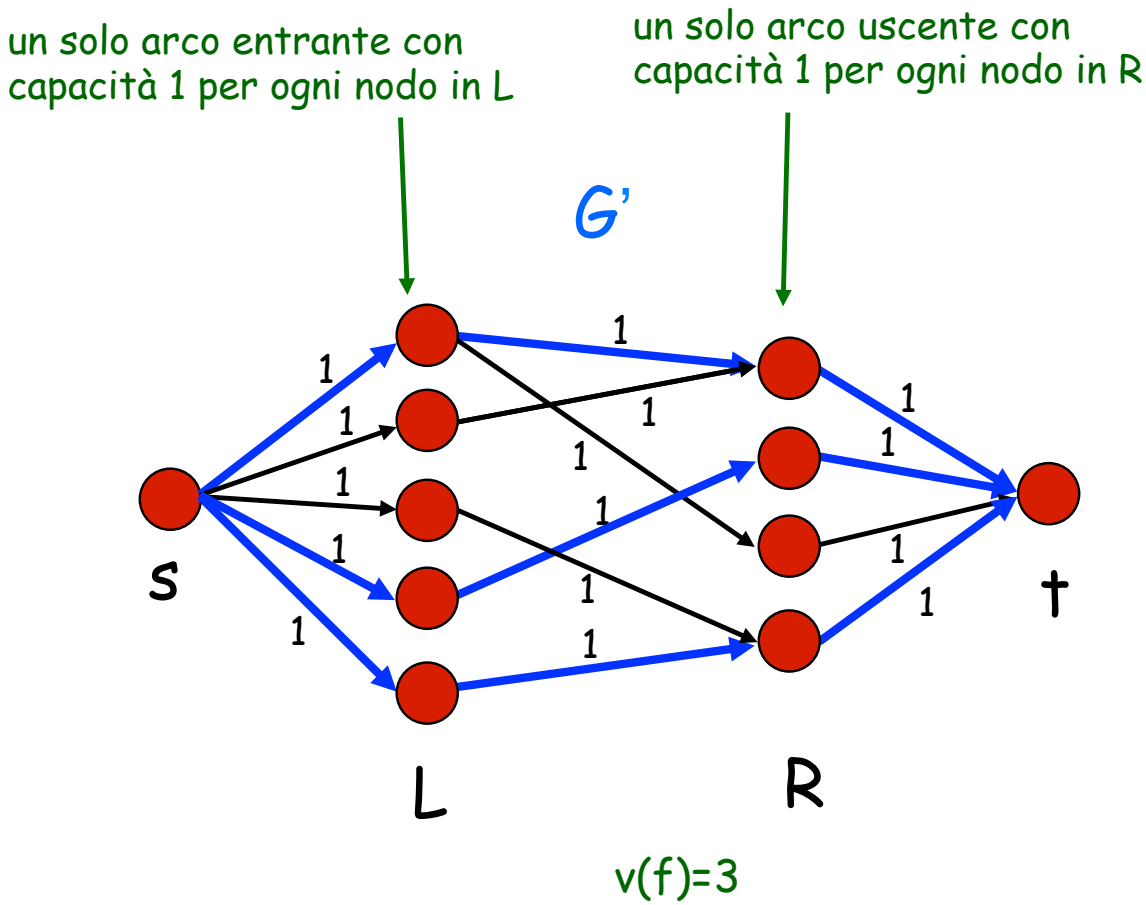


Maximum Bipartite Matching e Max-flow

Corrispondenza di matching in G con flussi in G'

- Per ogni matching di cardinalità |M| c'è un flusso di valore v(f)=|M| in G'
- Per ogni flusso a valori interi in G' c'è un matching in G con |M|=v(f)

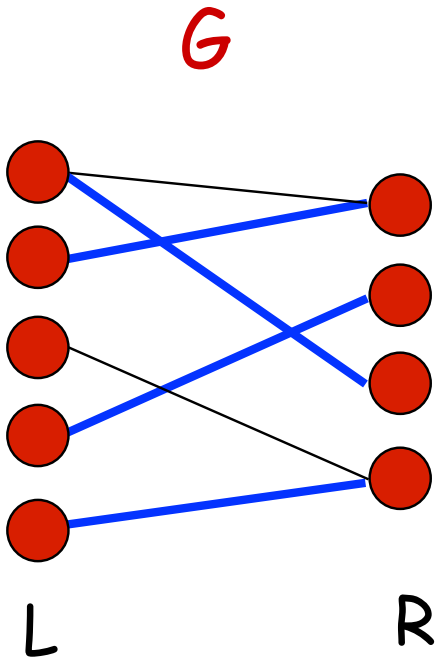


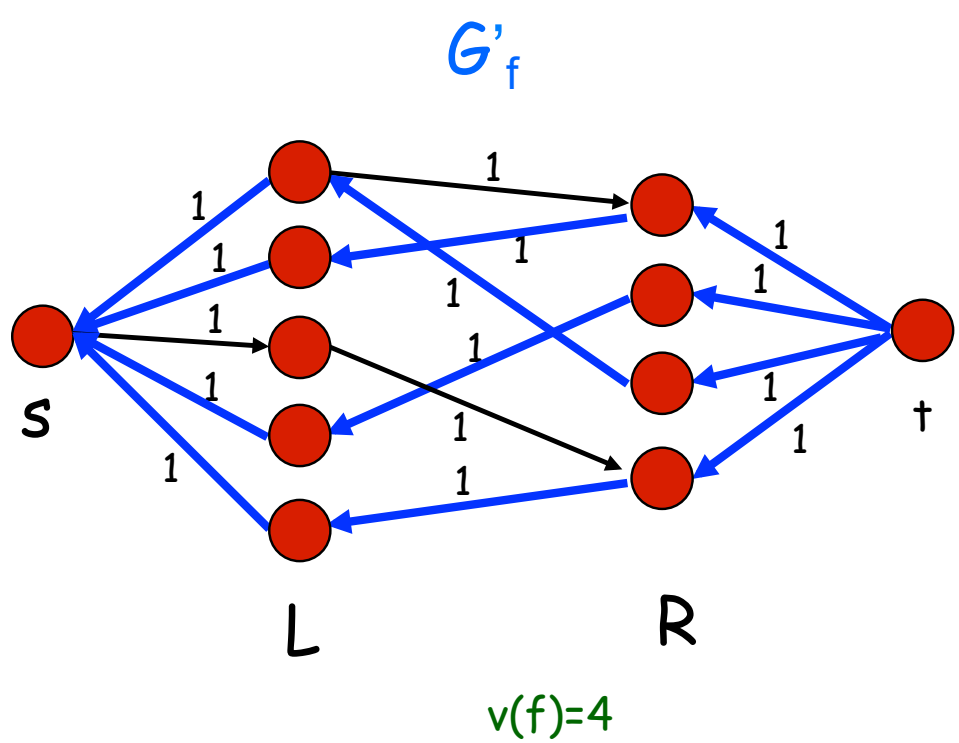


Maximum Bipartite Matching e Max-flow

Corrispondenza di matching in G con flussi in G'

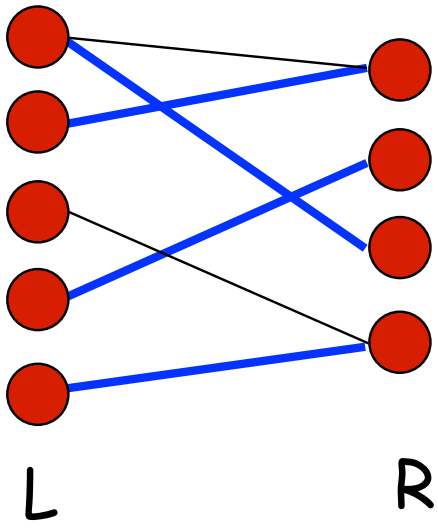
- Per ogni matching di cardinalità |M| c'è un flusso di valore v(f)=|M| in G'
- Per ogni flusso a valori interi in G' c'è un matching in G con |M|=v(f)
- Nell'algoritmo di Ford-Fulkerson su G': flusso f(e) ∈ {0,1} per ogni arco e



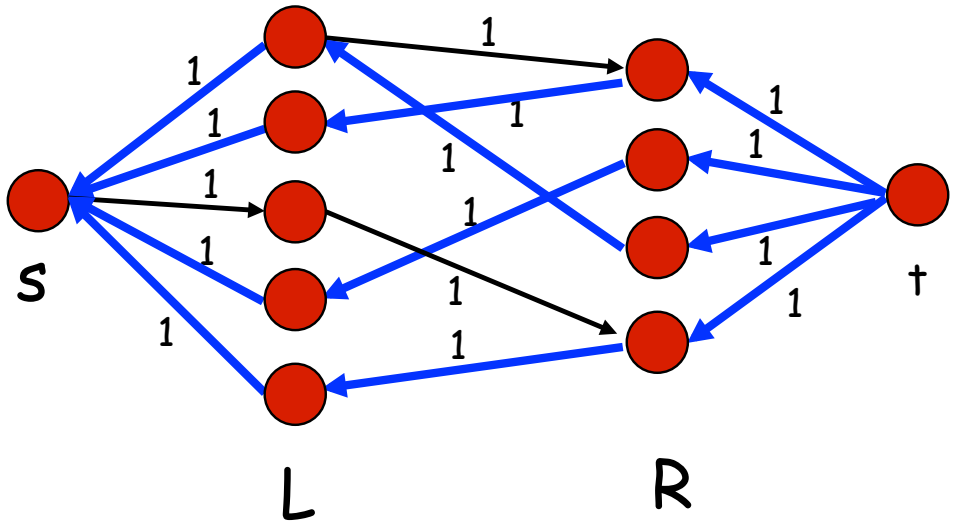


Maximum Bipartite Matching e Max-flow

Il flusso è massimo. Perchè?



|M|=4

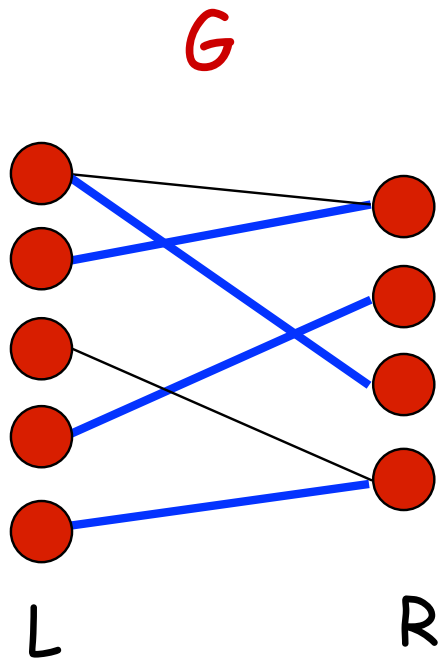


v(f)=4

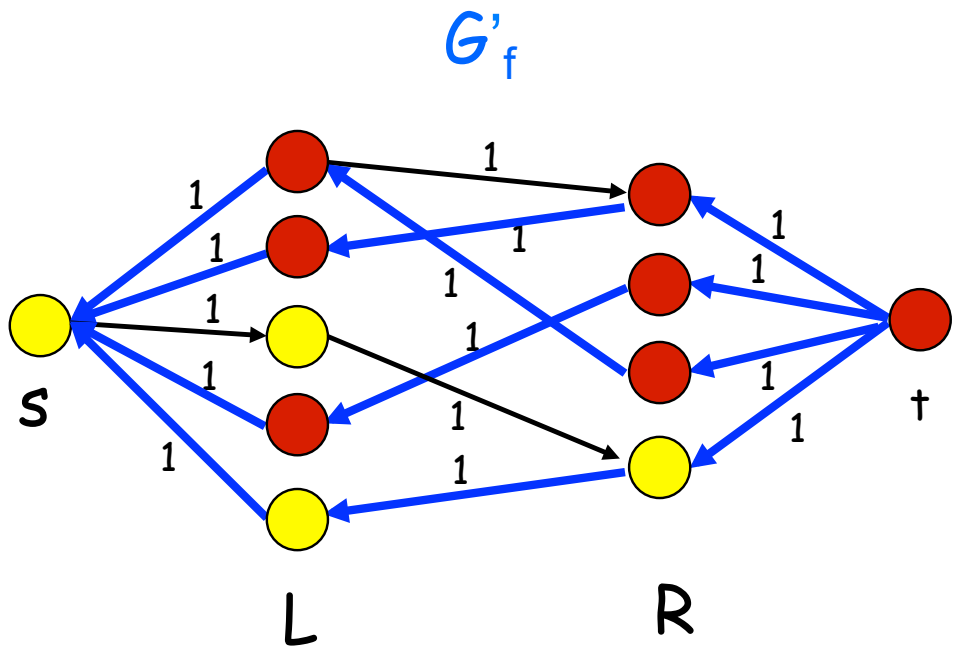
Maximum Bipartite Matching e Max-flow

Il flusso è massimo. Perchè?

- nodi raggiungibili da s nel grafo residuale G'_f = nodi gialli



|M|=4

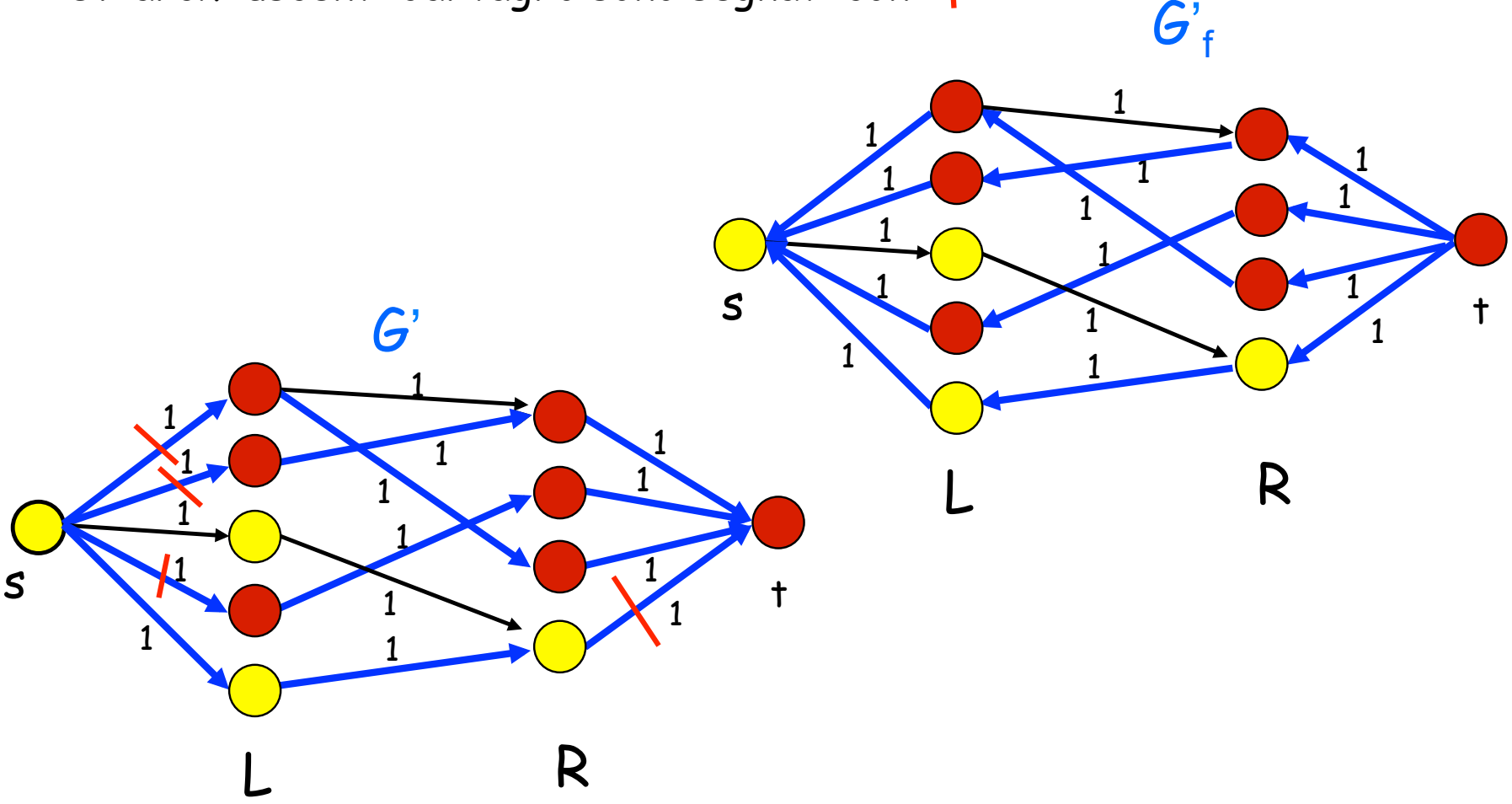


v(f)=4

Maximum Bipartite Matching e Max-flow

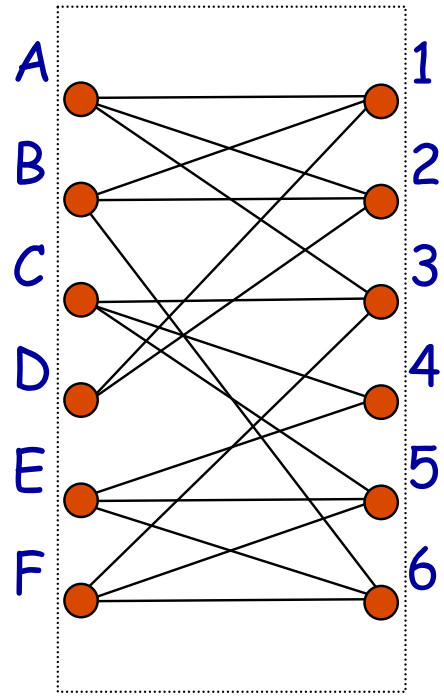
Il flusso è massimo. Perchè?

- nodi raggiungibili da s nel grafo residuale G'_f = nodi gialli
- Capacità taglio (nodi gialli, nodi rossi) in G = 4
  Gli archi uscenti dal taglio sono segnati con " | "

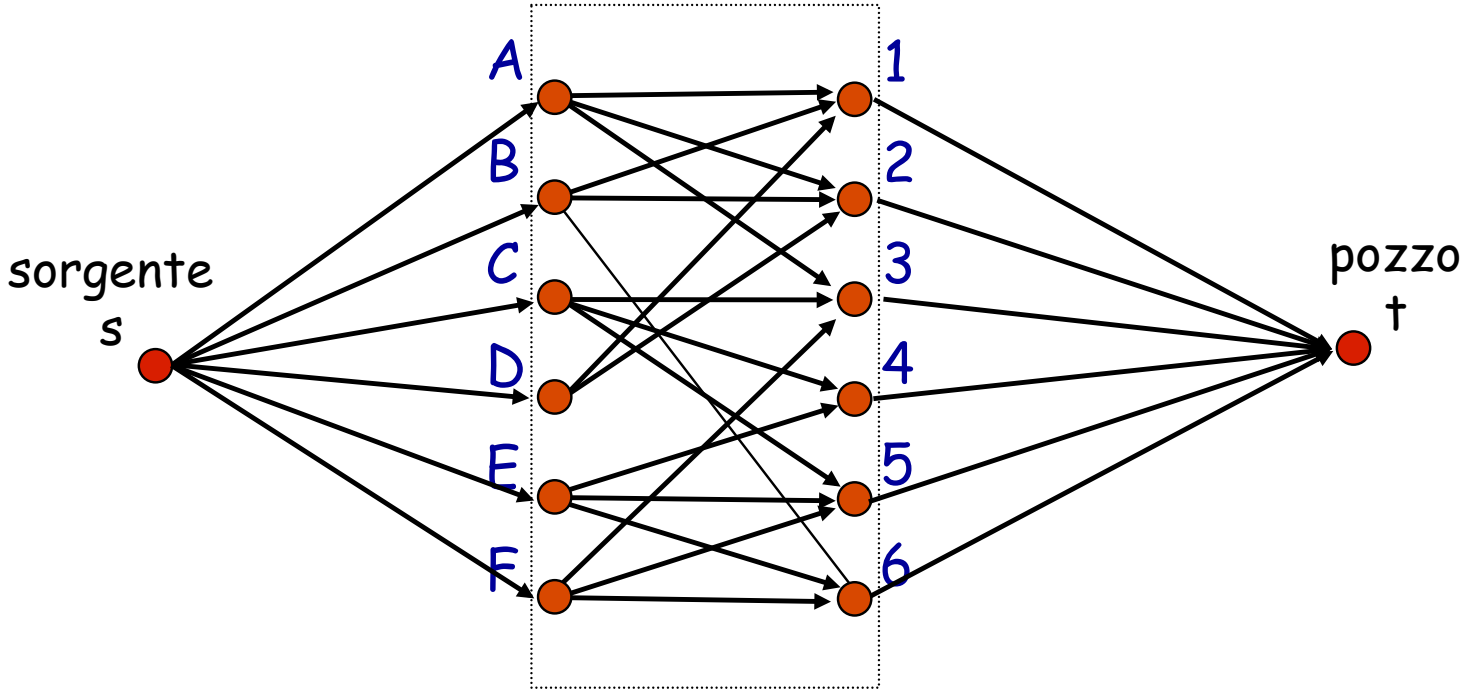


Maximum Bipartite Matching: Esercizio

Calcolare il Maximum Matching sul grafo bipartito:



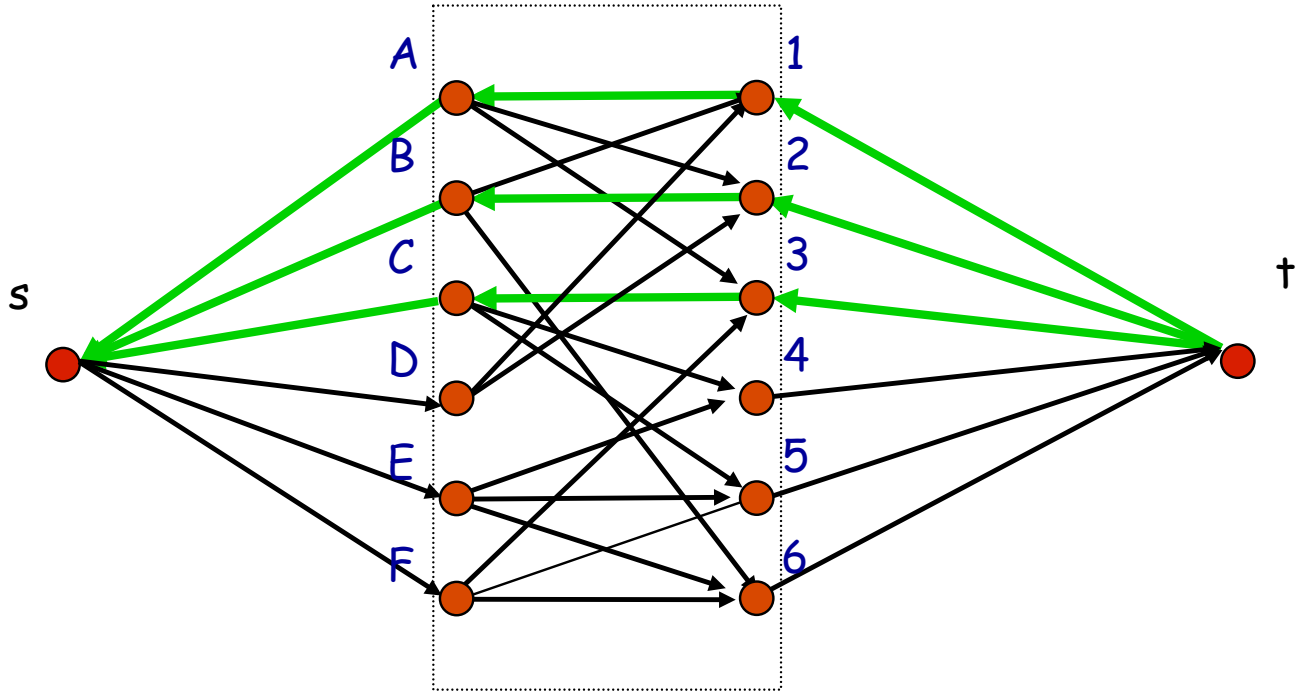
Maximum Bipartite Matching: Esercizio



La capacità di ogni arco è c(e)=1

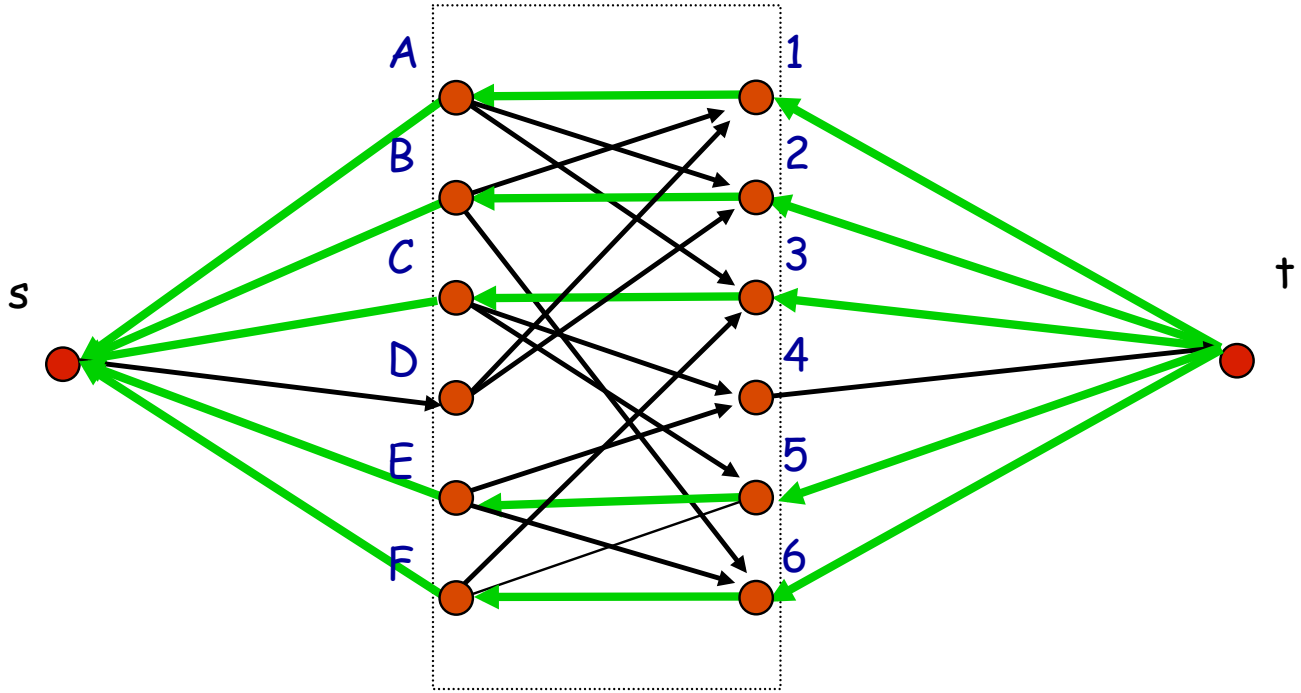
Maximum Bipartite Matching: Esercizio

Grafo residuale dopo i primi 3 cammini aumentanti:



Maximum Bipartite Matching: Esercizio

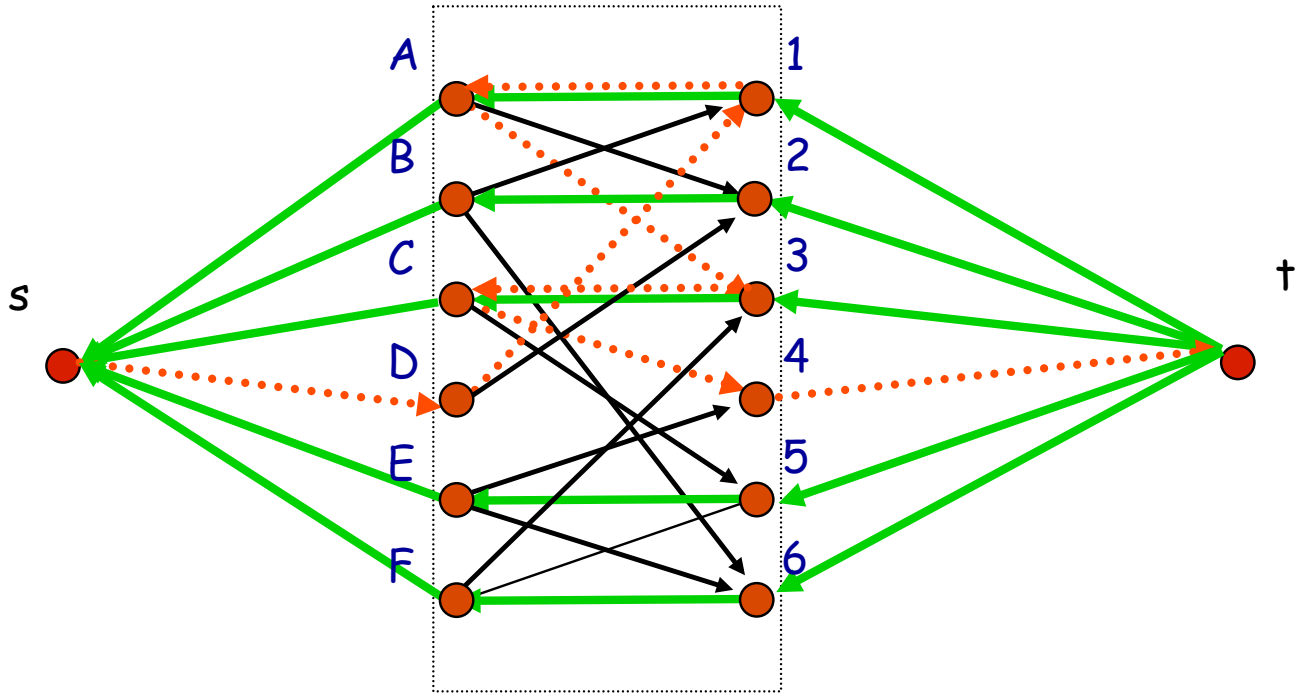
Grafo residuale dopo i primi 5 cammini aumentanti:



Maximum Bipartite Matching: Esercizio

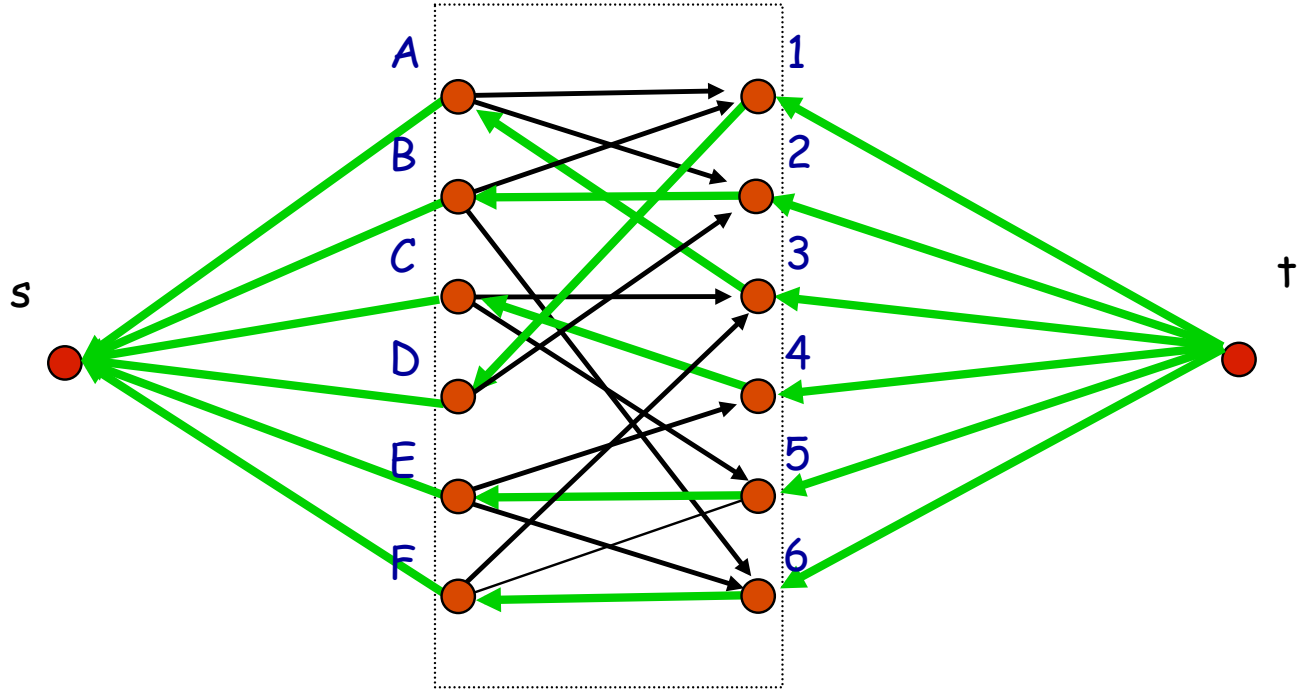
Il sesto cammino aumentante:

s-D-1-A-3-C-4-t



Maximum Bipartite Matching: Esercizio

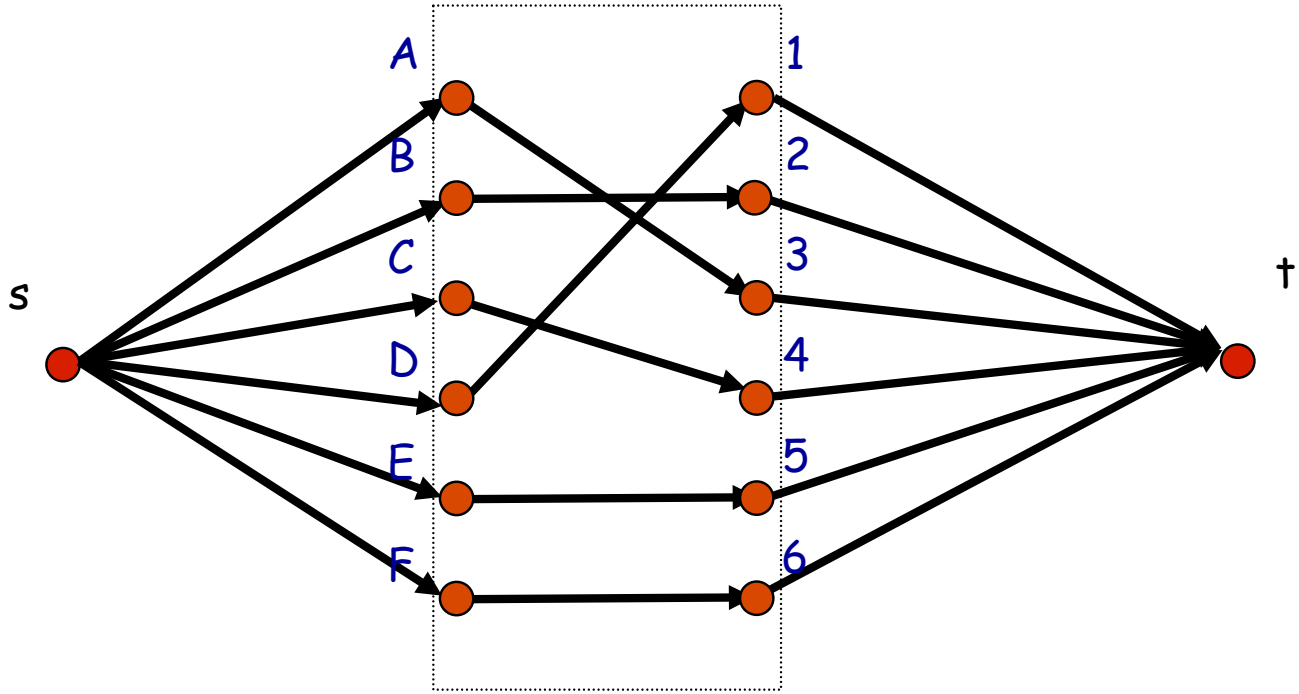
Grafo residuale dopo i primi 6 cammini aumentanti:



Non ci sono più cammini aumentanti.
Nodi raggiungibili da s nel grafo residuale = {s}.
Capacità taglio ({s},V-{s})=6.

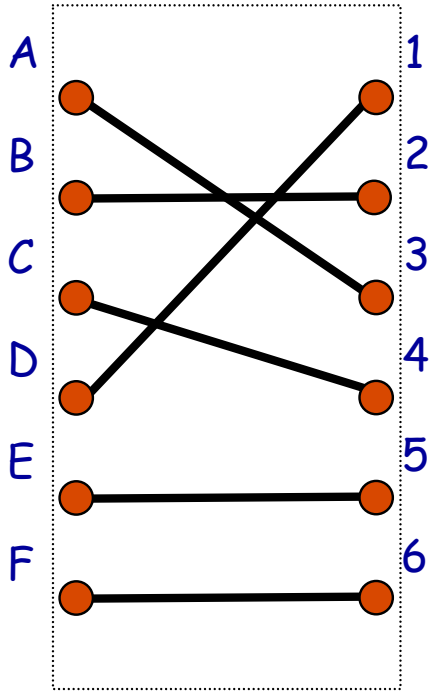
Maximum Bipartite Matching: Esercizio

Maximum flow:



Maximum Bipartite Matching: Esercizio

Maximum Matching:



Riepilogo Cap. 7, Network Flow

7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm
7.2 Maximum Flows and Minimum Cuts in a Network
7.3 Choosing Good Augmenting Paths
7.4 The Preflow-Push Maximum-Flow Algorithm (no)
7.5 A First Application: The Bipartite Matching Problem