

Progetto e Ottimizzazione di Reti

A. A. 2006-2007

Docente

Fabrizio Rossi

rossi@di.univaq.it

Orario

Martedì 15-17 aula 2.5

Mercoledì 11.30-13.30 aula 2.5

Giovedì 11.30-13.30 aula 2.5

Orario di ricevimento

Mercoledì 17-19

Prerequisiti

Teoria della Dualità

Algoritmi per la Programmazione Lineare

[Ricerca Operativa]

Formulazioni di PLI

Tecniche di bounding e algoritmi per la PLI

[Ottimizzazione Combinatoria]

Algoritmi di visita su grafi

Algoritmi per i problemi di cammino minimo

[Algoritmi e strutture dati]

Programma

Parte 1

Problemi di flusso massimo [Capitolo 3, par. 1-4 del testo 1]

Teorema Max-Flow Min-Cut

Algoritmo di Ford e Fulkerson

Algoritmo di Edmonds e Karp

Algoritmo Preflow-push (cenni)

Problemi di taglio minimo su grafi non orientati [Capitolo 3, par 5 del testo 1]

Algoritmo Node Identification

Algoritmo Random Contraction

Parte 2

Problemi di taglio multiterminale [Capitolo 3, par 5 del testo 1]

Algoritmo di Gomory-Hu (senza dimostrazioni)

Problemi di cammino minimo con pesi qualsiasi [slide]

Algoritmi Label-Correcting

Algoritmo di Bellman e Ford

Problemi di flusso a costo minimo [Capitolo 4, par 1 e 2 del testo 1]

Algoritmo del circuito aumentante

Simpleso su reti caso senza capacità

Simpleso su reti caso con capacità

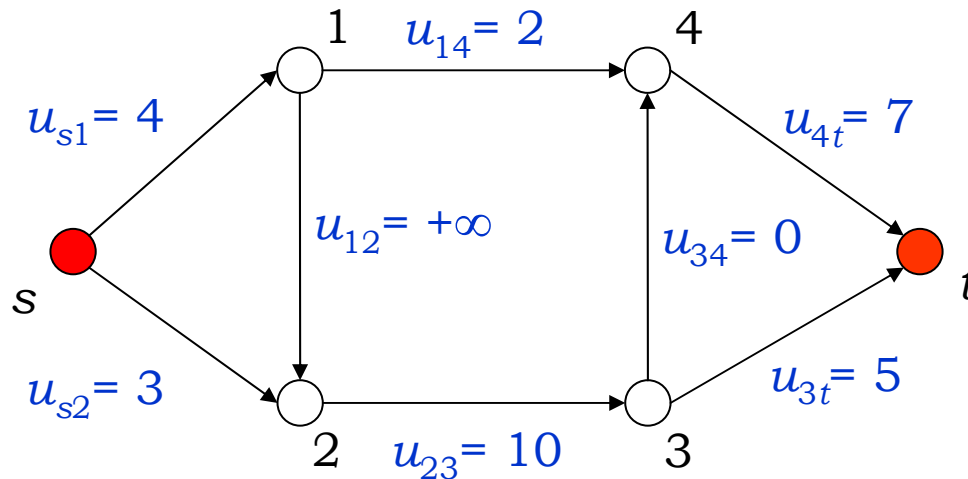
Testi di riferimento

[1] Cook, Cunningham, Pulleyblank, Schrijver Combinatorial Optimization

[2] Ahuja, Magnanti, Orlin Network Flows

Notazione

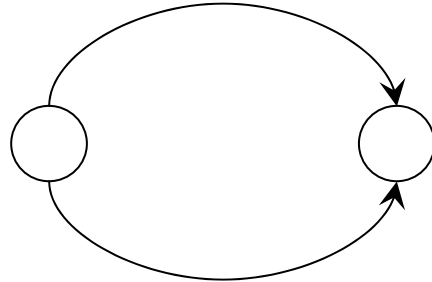
Grafo orientato $G = (N, A)$, con due nodi “speciali”:
il nodo s [nodo **sorgente**] e il nodo t [nodo **pozzo**]



Ad ogni arco (i, j) è associata una **capacità**
 $u_{ij} \in [0, +\infty)$, **INTERA**

Assunzioni

1. Il grafo **NON** contiene un **cammino orientato** dal nodo s al nodo t fatto esclusivamente da archi aventi capacità infinita.
2. Il grafo **NON** contiene archi “paralleli”



3. Al grafo si possono sempre aggiungere archi aventi capacità 0

Problema

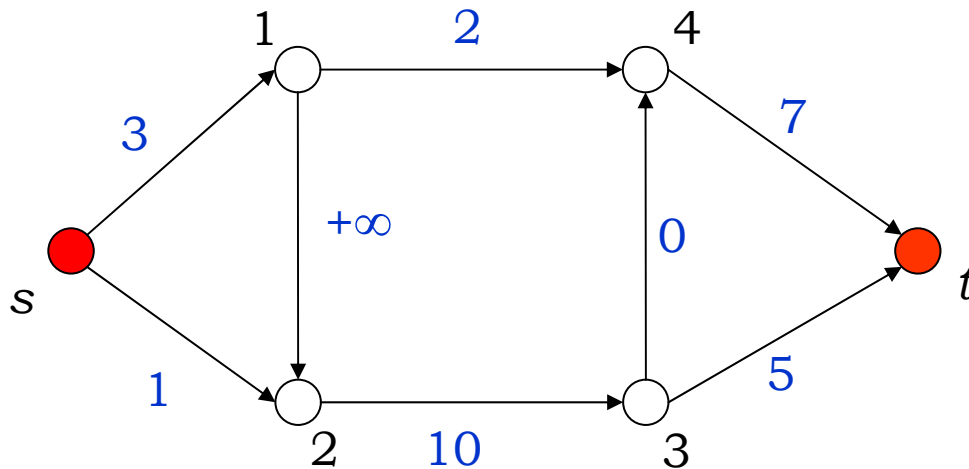
Path packing

Dati un grafo orientato $G = (N, A)$ e un vettore capacità $\mathbf{u} \in \mathbb{Z}_+^{|A|}$, individuare una famiglia di cammini orientati (semplici) $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$, non necessariamente distinti, tale che:

1. Ogni arco $(i, j) \in A$ è utilizzato da al più u_{ij} cammini
2. k sia massimo

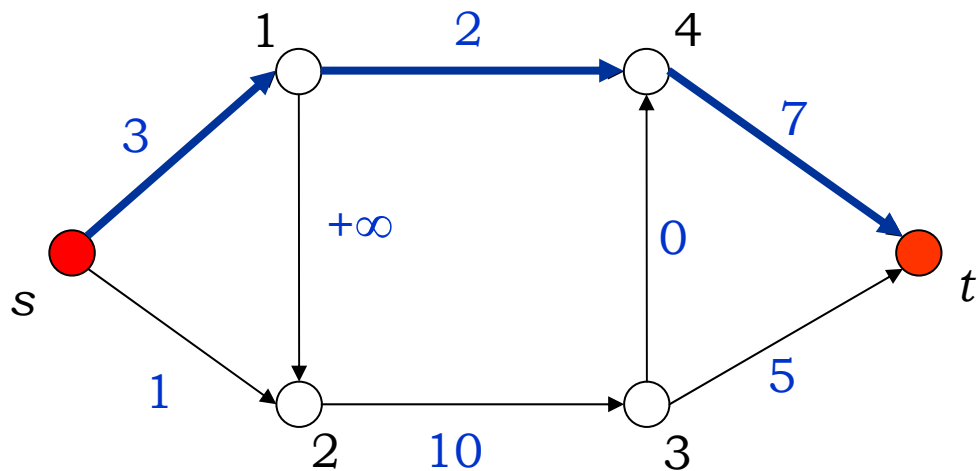
Esempio

Consideriamo il grafo di figura, avente le capacità descritte sugli archi:



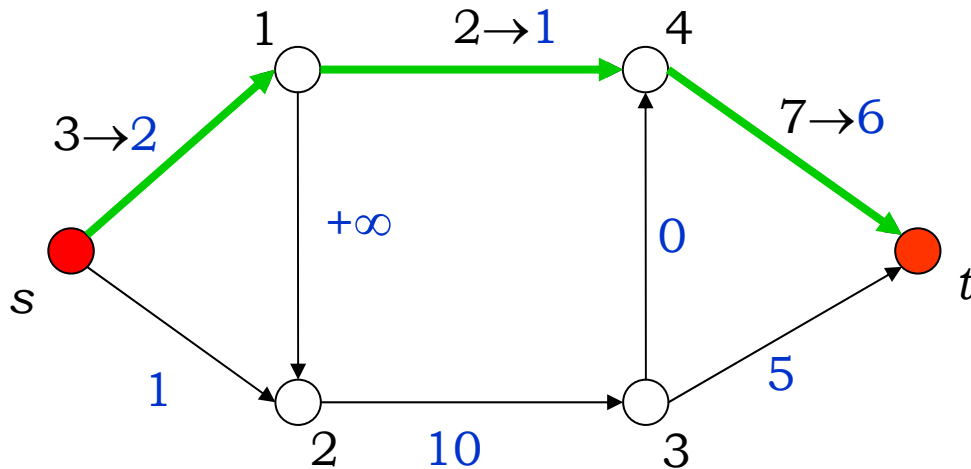
Esempio

Il primo (s,t) cammino che consideriamo è il cammino $P_1 = \{s, 1, 4, t\}$



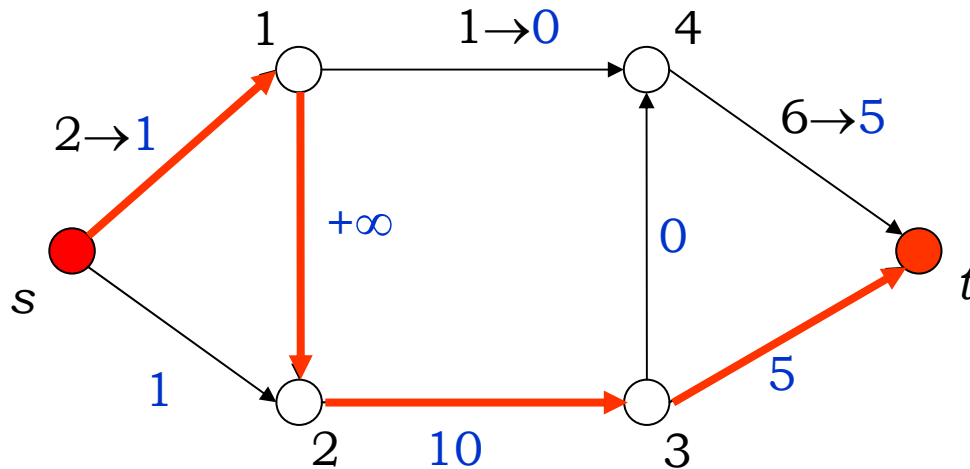
Esempio

Il secondo cammino utilizza gli stessi archi di P_1 : $P_2 = \{s, 1, 4, t\}$



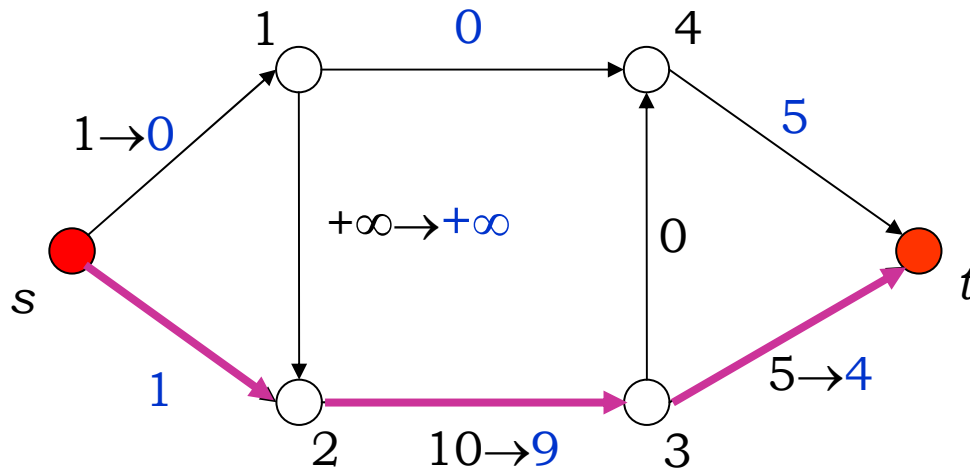
Esempio

Dal momento che la capacità dell'arco $(1, 4)$ è completamente utilizzata, non possiamo scegliere un cammino che utilizzi gli stessi archi di P_1 e P_2 . Pertanto, scelgo $P_3 = \{s, 1, 2, 3, t\}$



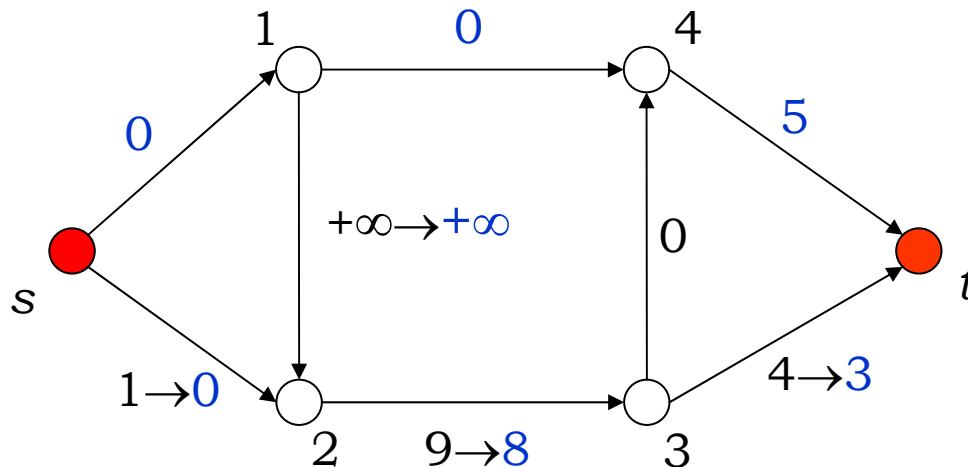
Esempio

A questo punto, anche la capacità dell'arco $(s, 1)$ è completamente utilizzata. Pertanto il cammino $P_4 = \{s, 2, 3, t\}$ utilizza l'arco $(s, 2)$



Esempio

Dal nodo s non è più possibile aggiungere cammini senza violare il vincolo di capacità sugli archi $(s, 1)$ e $(s, 2)$.
La famiglia di cammini $\{P_1, P_2, P_3, P_4\}$, con $k=4$ è una soluzione ammissibile del problema di “path packing”.



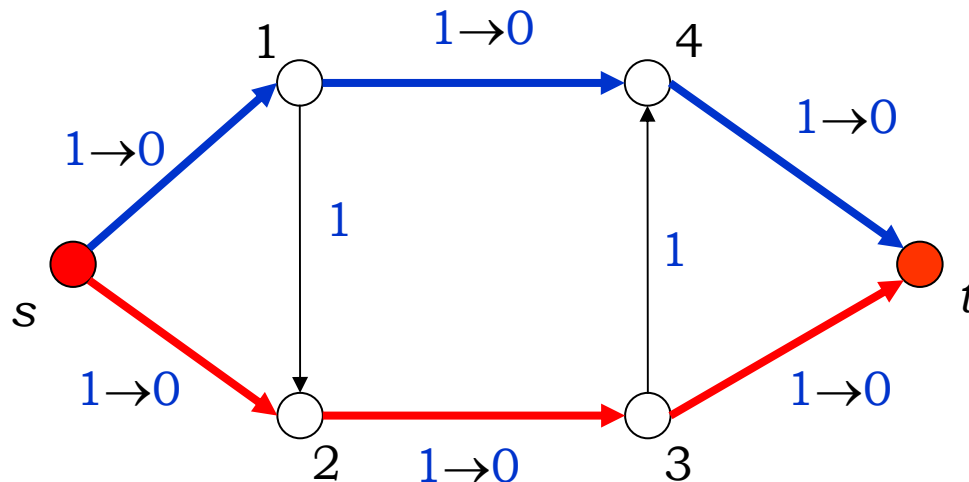
$$\begin{aligned} P_1 &= \{s, 1, 4, t\} \\ P_2 &= \{s, 1, 4, t\} \\ P_3 &= \{s, 1, 2, 3, t\} \\ P_4 &= \{s, 2, 3, t\} \end{aligned}$$

Osservazioni

1. È possibile certificare l'ottimalità della soluzione che abbiamo trovato?
2. Esiste una formulazione di PLI per il problema di path packing?
3. Esistono algoritmi a complessità polinomiale per il problema di path packing?

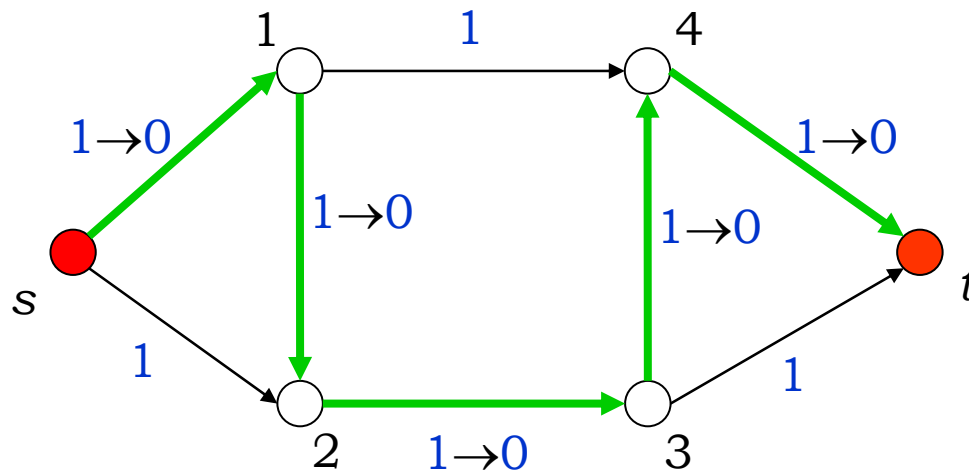
Osservazione 1

In generale, se scelgo cammini in modo greedy posso non trovare la soluzione ottima. Difatti, il seguente grafo contiene due (s, t) cammini:



Osservazione 1

ma, se scelgo come primo cammino il cammino $P = \{s, 1, 2, 3, 4, t\}$ non riesco a trovare ulteriori cammini da s a t



Osservazione 2: formulazione

Associamo ad ogni arco (i, j) una variabile
INTERA x_{ij} con il seguente significato:

x_{ij} = Numero di volte che l'arco (i, j) viene
utilizzato dai cammini di \mathcal{P}

Vincoli

Osservazione

Per ogni nodo $v \neq s, t$ si ha che ogni cammino
 P_i entra ed esce da v esattamente lo stesso
numero di volte

Formulazione

Vincoli di bilanciamento

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = 0 \quad \forall i \in N \setminus \{s, t\}$$

Vincoli di capacità

$$0 \leq x_{ij} \leq u_{ij}, \forall (i, j) \in A$$

Stipula di interezza

$$x_{ij} \text{ intera}, \forall (i, j) \in A$$

Flusso

Per il nodo sorgente s si ha, invece:

$$k = \sum_{j:(s,j) \in A} x_{sj} - \sum_{j:(j,s) \in A} x_{js}$$

Un vettore $x \in \mathbb{Z}_+^{|A|}$ che soddisfa tutti i vincoli di bilanciamento si definisce **(s,t)-flusso** o, semplicemente, **flusso**.

Se il vettore x soddisfa anche i vincoli di capacità il flusso si dice **ammissibile**

Il termine

$$f_x(v) = \sum_{j:(v,j) \in A} x_{vj} - \sum_{j:(j,v) \in A} x_{jv}$$

si dice **flusso netto** in v .

$f_x(s)$ è il **valore** del flusso x in G

Teorema di decomposizione

Ad una famiglia di cammini $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$ ammissibile è sempre possibile associare un vettore di flusso x ammissibile.

È vero il viceversa?

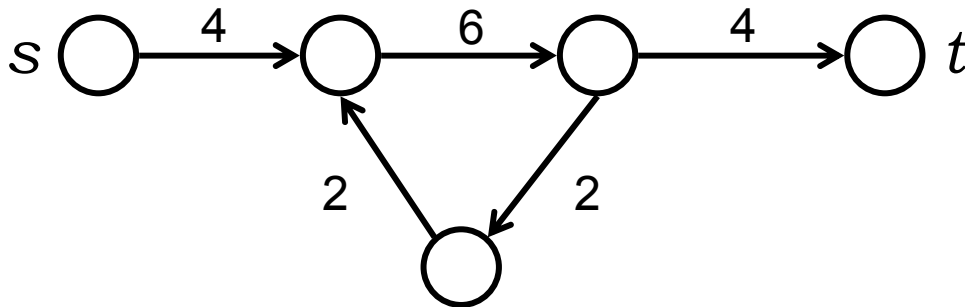
Teorema di decomposizione

In un grafo $G=(N,A)$ esiste una famiglia $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$ di k (s,t) -cammini ammissibile se e solo se esiste un (s,t) -flusso ammissibile di valore k

Dimostrazione

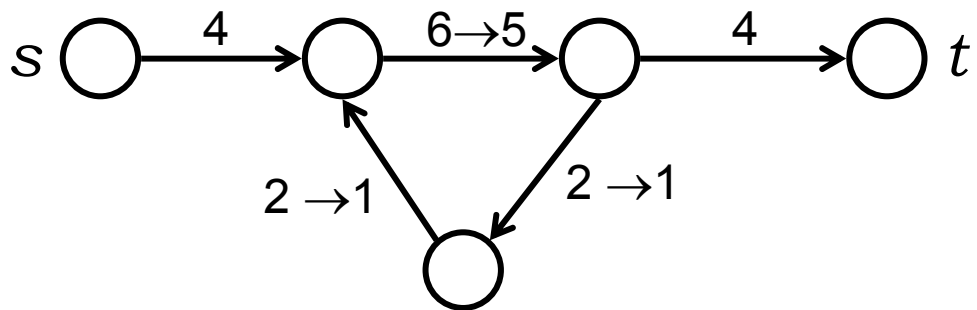
Sia x un flusso ammissibile “aciclico”, ovvero tale che non esiste un ciclo orientato C avente $x_{ij} > 0$ per tutti gli archi $(i,j) \in C$.

Difatti, se x contiene un ciclo orientato C con questa proprietà, basta diminuire x_{ij} di una unità per tutti gli archi $(i,j) \in C$.



Dimostrazione

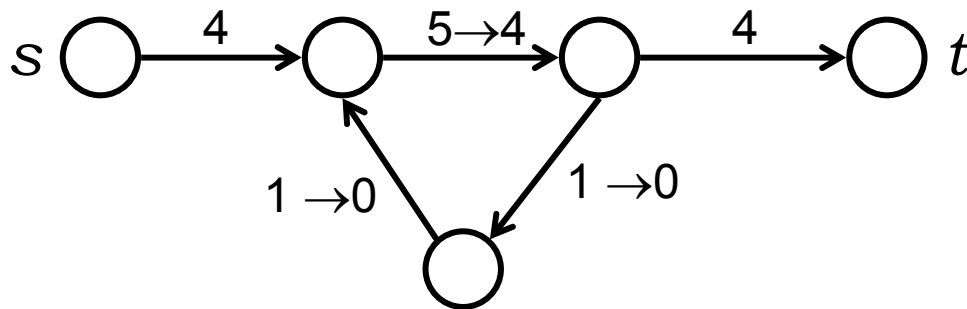
In questo modo si ottiene ancora un flusso ammissibile di valore k . A questo punto basta ripetere la procedura finché il ciclo C non esiste più.



Dimostrazione

In questo modo si ottiene ancora un flusso ammissibile di valore k . A questo punto basta ripetere la procedura finché il ciclo C non esiste più.

Questa semplice procedura consente sempre di ottenere un flusso x aciclico senza alterarne il valore k .



Dimostrazione

A questo punto, se $k \geq 1$, esiste un arco (v, t) avente $x_{vt} \geq 1$.

Se $v \neq s$, dai vincoli di bilanciamento segue che esiste almeno un arco avente $x_{wv} \geq 1$.

Ripetendo il ragionamento per il nodo w , se $w \neq s$ allora esiste un arco (p, w) con $x_{pw} \geq 1$. Se continuiamo a ripetere questa procedura, essendo x aciclico, ci si arresta con un (s, t) -cammino semplice fatto di archi (i, j) aventi $x_{ij} \geq 1$.

È, quindi, sufficiente decrementare di una unità ogni componente del vettore x corrispondente ad ogni arco dell' (s, t) -cammino per ottenere un nuovo flusso (intero) ammissibile di valore $k-1$.

Ripetendo la procedura finché $k = 0$, si ottengono i k cammini della famiglia \mathcal{P}



Problema del massimo flusso

$$\max f_x(s)$$

s.t.

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = 0 \quad \forall i \in N \setminus \{s, t\}$$

$$0 \leq x_{ij} \leq u_{ij}, \forall (i, j) \in A$$

$$x_{ij} \text{ intero}, \forall (i, j) \in A$$

Taglio di un grafo

Definizione

Dato un grafo $G=(N, A)$, un insieme $\delta(R)=\{vw: (v,w) \in A, v \in R, w \notin R\}$ per qualche $R \subseteq V$ si dice **taglio**

Un **(s,t)-taglio** è un taglio per cui $s \in R, t \notin R$

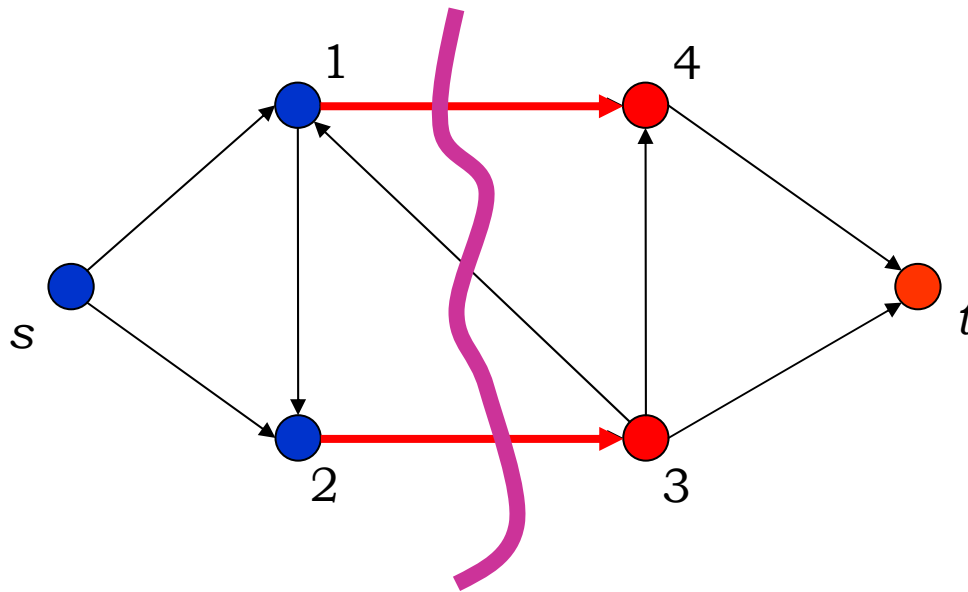
Si definisce capacità del **(s,t)-taglio** la quantità

$$\sum_{i \in R, j \notin R} u_{ij} = u(\delta(R))$$

Esempio

$$R = \{s, 1, 2\}$$

$$\bar{R} = \{3, 4, t\}$$

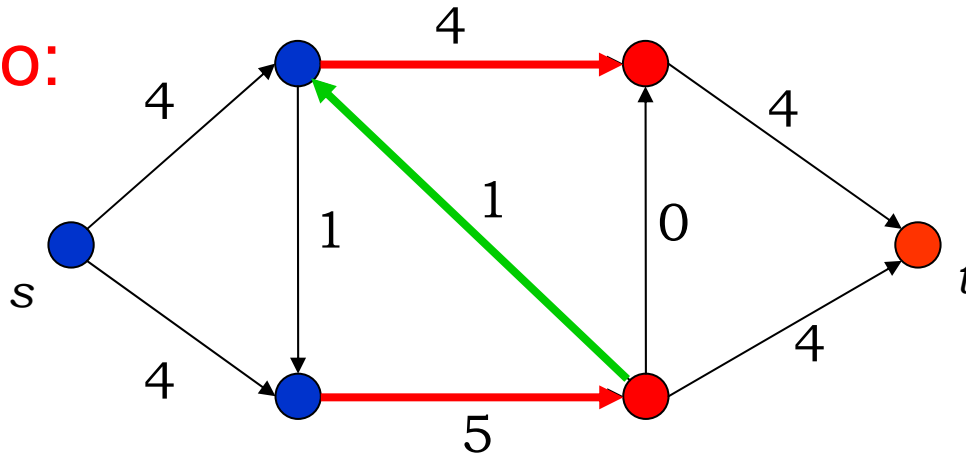


Teorema 1

Per ogni (s,t) -taglio $\delta(R)$ e per ogni (s,t) -flusso x ,
si ha:

$$x(\delta(R)) - x(\delta(\bar{R})) = f_x(s) \quad \blacksquare$$

Esempio:



Dimostrazione

Consideriamo un taglio R e sommiamo, per tutti i nodi $v \in R, v \neq s$ i vincoli di bilanciamento.

Si ottiene un'equazione del tipo:

$$\text{LHS} = 0$$

Il termine LHS è così formato:

1. Per ogni arco (v, w) tale che $v, w \in R, v \neq s$ la variabile x_{vw} **NON** è contenuta in LHS (ha coefficiente 0 nella somma).
2. Per ogni arco (v, w) tale che $v, w \notin R$, la variabile x_{vw} **NON** è contenuta in LHS (gli estremi dell'arco non appartengono a R).
3. Per ogni arco (v, w) tale che $v \in R, w \notin R$ la variabile x_{vw} compare nel LHS con coefficiente + 1

Dimostrazione

4. Per ogni arco (v, w) tale che $v \notin R, w \in R$ la variabile x_{vw} compare nel LHS con coefficiente -1 .
5. Per ogni arco (s, v) tale che $v \in R$ la variabile x_{sv} compare nel LHS con coefficiente -1
6. Per ogni arco (v, s) tale che $v \in R$ la variabile x_{vs} compare nel LHS con coefficiente 1 .

Raggruppando le variabili che soddisfano le condizioni 3 e 4 si ottiene il termine:

$$x(\delta(R)) - x(\delta(\bar{R}))$$

Le variabili che soddisfano le condizioni 5 e 6 valgono complessivamente $-f_x(s)$. Pertanto,

$$\text{LHS} = x(\delta(R)) - x(\delta(\bar{R})) - f_x(s)$$



Corollario (dualità debole)

Per ogni (s,t) -taglio $\delta(R)$ e per ogni (s,t) -flusso x , si ha:

$$f_x(s) \leq u(\delta(R))$$

Dimostrazione

Dal teorema 1 si ha che:

$$x(\delta(R)) - x(\delta(\bar{R})) = f_x(s)$$

Ora, per definizione $x(\delta(R)) \leq u(\delta(R))$. Inoltre, $x(\delta(\bar{R})) \geq 0$

Pertanto, $f_x(s) \leq u(\delta(R))$.



Conseguenza

Il corollario di dualità debole fornisce un bound per il valore del massimo flusso.

Pertanto, se identifichiamo in G un flusso x avente valore pari alla capacità u di un taglio R , abbiamo individuato la soluzione ottima del problema di massimo flusso.

Il teorema Max-Flow Min-Cut, afferma che questa possibilità si verifica per ogni grafo G , che ammette un flusso massimo finito.

Teorema Max-flow Min-cut

Se $G=(N,A)$ ammette un (s,t) -flusso massimo,
allora

$$\max\{f_x(s) : x \text{ è un } (s,t)\text{-flusso ammissibile}\} = \\ = \min\{u(\delta(R)) : \delta(R) \text{ è un } (s,t)\text{-taglio}\}$$



[Ford e Fulkerson, Kotzig 1956]

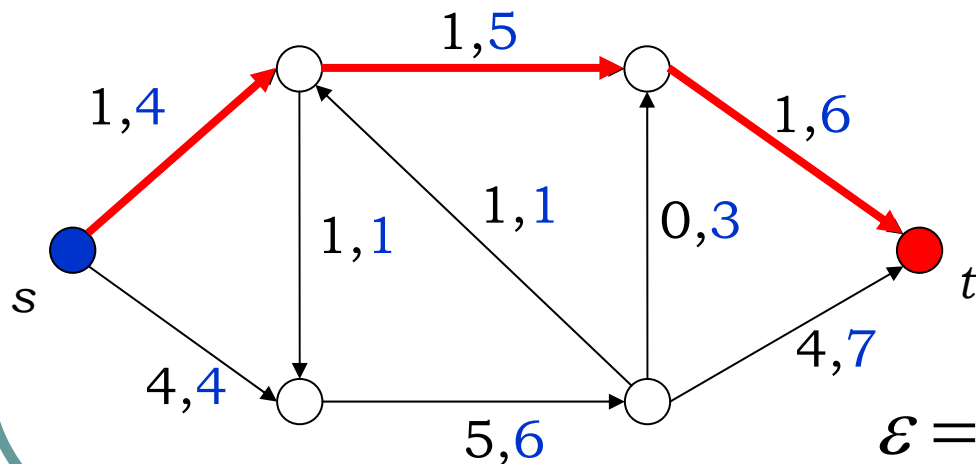
Per dimostrare questo teorema dobbiamo introdurre il
concetto di “**cammino aumentante**”

Cammini aumentanti

Osservazione

Dato un grafo $G = (N, A)$ e un flusso x , se esiste un (s, t) -cammino P tale che $x_{ij} < u_{ij}$ per ogni arco $(i, j) \in P$, allora posso aumentare il flusso di un valore pari a:

$$\varepsilon = \min \{u_{ij} - x_{ij}, (i, j) \in P\}$$

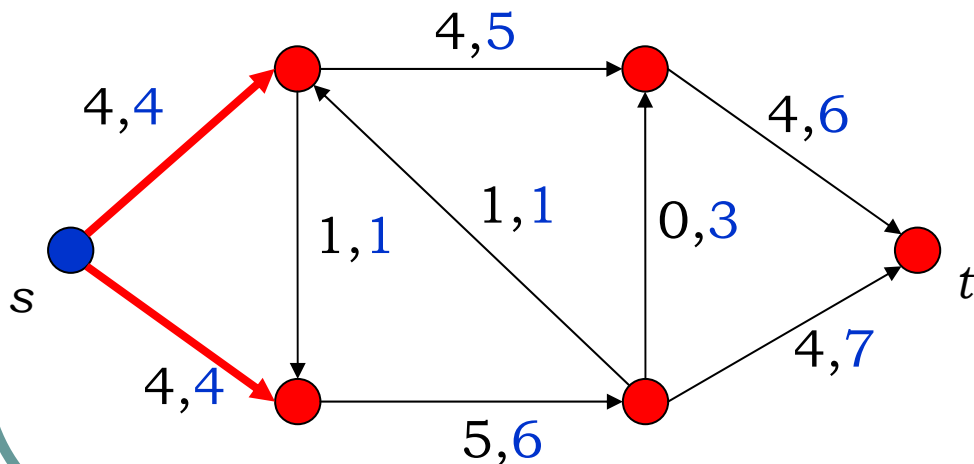


$$\varepsilon = \min\{4 - 1, 5 - 1, 6 - 1\} = 3$$

Cammini aumentanti

Osservazione (cont.)

Il nuovo flusso è ottimo. Difatti, esiste un (s,t) -taglio (individuato dai nodi blu e rossi) di capacità 8, pari al valore del massimo flusso.



Un possibile algoritmo

inizializzazione: $x = 0$;

do {

cerca in G un (s, t) -cammino P tale che $x_{ij} < u_{ij}$
per ogni arco $(i, j) \in P$;

aumenta lungo P il flusso x del valore
 $\varepsilon = \min \{u_{ij} - x_{ij}, (i, j) \in P\}$

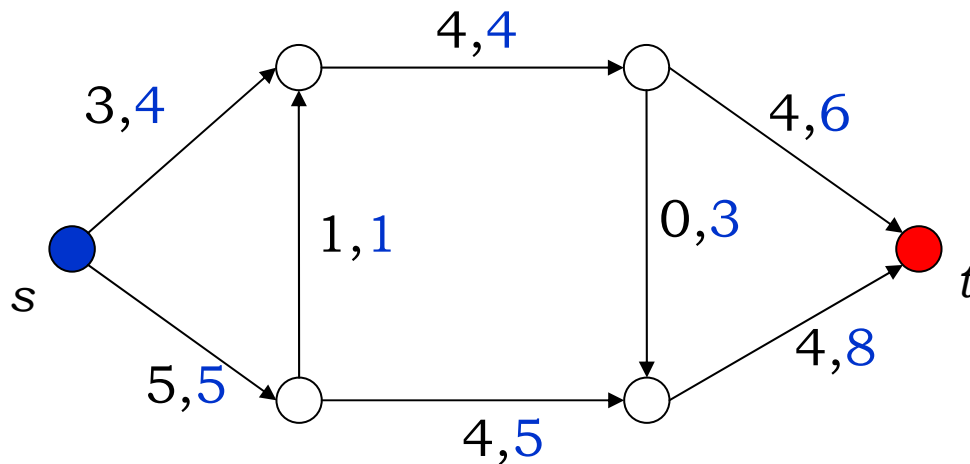
} **while** ($P \neq \emptyset$);

Questo algoritmo

1. Termina?
2. Trova la soluzione ottima?
3. Qual è la sua complessità?

Cammini aumentanti

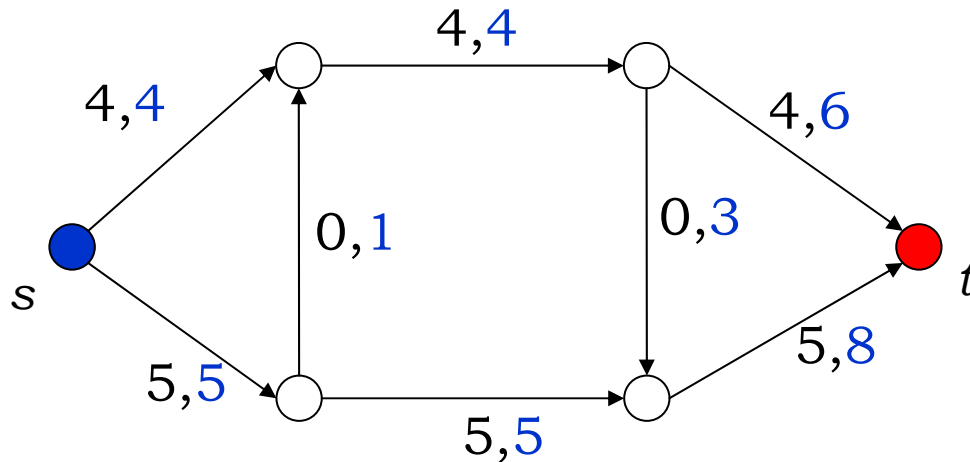
Consideriamo il grafo di figura:



Non esistono (s, t) -cammini P tali che $x_{ij} < u_{ij}$ per ogni arco $(i, j) \in P$, ma il flusso non è ottimo.

Cammini aumentanti

Questo è il flusso ottimo! [Perché?]



Come si fa ad ottenerlo dal flusso precedente?

Cammini aumentanti

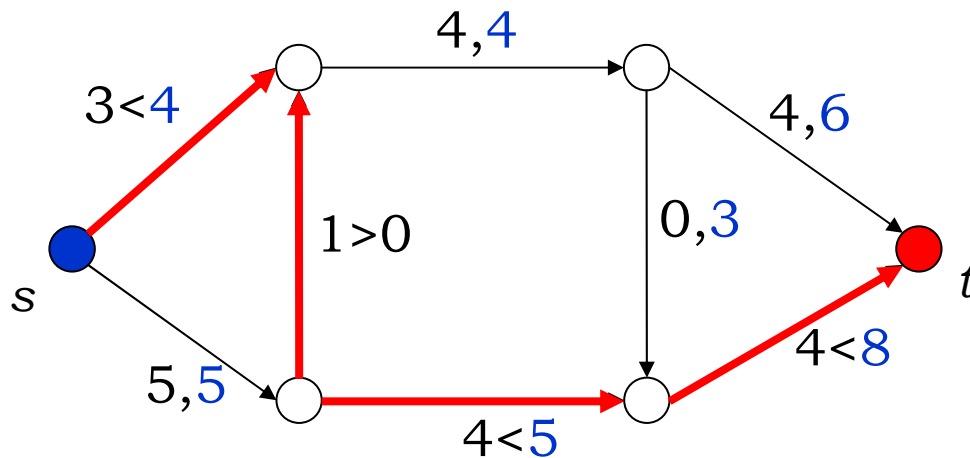
Sia P un cammino (**NON** ben orientato) da s a t . Un arco di P si dice “in avanti” (**forward**) se ha verso concorde con la direzione $s \rightarrow t$, “all’indietro” (**reverse**) viceversa.

Definizione

Un (s, t) -cammino P tale che ogni arco (i, j) forward ha $x_{ij} < u_{ij}$ e ogni arco (i, j) reverse ha $x_{ji} > 0$ si dice **cammino aumentante**.

Cammini aumentanti

Il cammino di figura è un **cammino aumentante**:



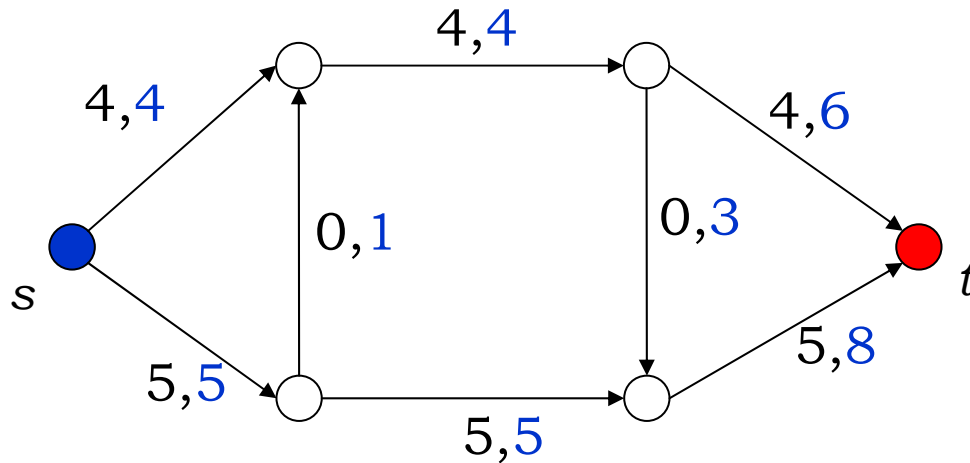
Sugli archi del cammino possiamo aumentare il flusso della quantità $\min \{ \varepsilon_1, \varepsilon_2 \}$ ove

$$\varepsilon_1 = \min \{ u_{ij} - x_{ij} : (i,j) \in P \text{ e } (i,j) \text{ è un arco forward} \}$$

$$\varepsilon_2 = \min \{ x_{ij} : (i,j) \in P \text{ e } (i,j) \text{ è un arco reverse} \}$$

Cammini aumentanti

Aumentando il flusso lungo P si ottiene:



Definizione

Un (s, v) -cammino P tale che $v \neq t$ per ogni arco (i, j) forward ha $x_{ij} < u_{ij}$ e per ogni arco (i, j) reverse ha $x_{ji} > 0$ si dice **cammino incrementante**.

Dimostrazione teorema Max-flow Min-cut

Dalla proprietà di dualità debole sappiamo che è sufficiente dimostrare che in G esiste un flusso x e un taglio $\delta(R)$ tali che $f_x(s) = u(\delta(R))$.

Sia x un flusso avente valore massimo. Costruiamo il taglio $\delta(R)$ definendo R come segue:

$R = \{v \in N: \text{esiste un cammino incrementante } (s, v)\}$.

Per definizione, $t \notin R$. Difatti, se t appartenesse ad R , il cammino sarebbe aumentante, contraddicendo la massimalità di x .

Per ogni arco $(i, j) \in \delta(R)$, si ha che $x_{ij} = u_{ij}$. Difatti, se x_{ij} fosse minore di u_{ij} , si avrebbe $j \in R$. Quindi, $x(\delta(R)) = u(\delta(R))$.

Per ogni arco $(i, j) \in \delta(\bar{R})$, si ha che $x_{ij} = 0$. Difatti, se x_{ij} fosse maggiore di 0, si avrebbe $j \in R$. Quindi, $x(\delta(\bar{R})) = 0$.

Pertanto, dal Teorema 1 si ha che:

$$f_x(s) = x(\delta(R)) - x(\delta(\bar{R})) = u(\delta(R))$$



Conseguenze del teorema MFMC

Teorema 2

Un flusso ammissibile x è ottimo se e solo se non esistono in G cammini aumentanti rispetto x .

Dimostrazione

x massimo \Rightarrow non esistono cammini aumentanti

non esistono cammini aumentanti $\Rightarrow x$ massimo

Se non esistono cammini aumentanti, utilizzando la costruzione del teorema MFMC, possiamo determinare un taglio $\delta(R)$ con la proprietà

$$f_x(s) = u(\delta(R)).$$

Dalla proprietà di dualità debole segue che x è massimo. ■

Corollario 1

Se x è un (s, t) -flusso ammissibile e $\delta(R)$ è un (s, t) -taglio, allora x è massimo e $\delta(R)$ è minimo se e solo se $x_{ij} = u_{ij}$ per ogni $(i, j) \in \delta(R)$ e $x_{ij} = 0$ per ogni $(i, j) \in \delta(\bar{R})$ ■

Algoritmo del cammino aumentante

inizializzazione: $x = 0$;

```
do {  
  cerca in  $G$  un  $(s, t)$ -cammino  $P$  aumentante  
  rispetto  $x$ ;  
  aumenta lungo  $P$  il flusso  $x$  del valore  $\min \{\varepsilon_1, \varepsilon_2\}$ ;  
} while ( $P \neq \emptyset$ ); [Algoritmo di Ford e Fulkerson]
```

Questo algoritmo

1. Termina?

2. Trova la soluzione ottima?

SI: se l'algoritmo termina, dal teorema 2 sappiamo che il flusso è ottimo.

3. Qual è la sua complessità?

Una struttura per cercare cammini aumentanti

Per cercare cammini aumentanti abbiamo bisogno di una opportuna struttura dati.

A partire da G , definiamo un grafo ausiliario $G(x)$ con le seguenti caratteristiche:

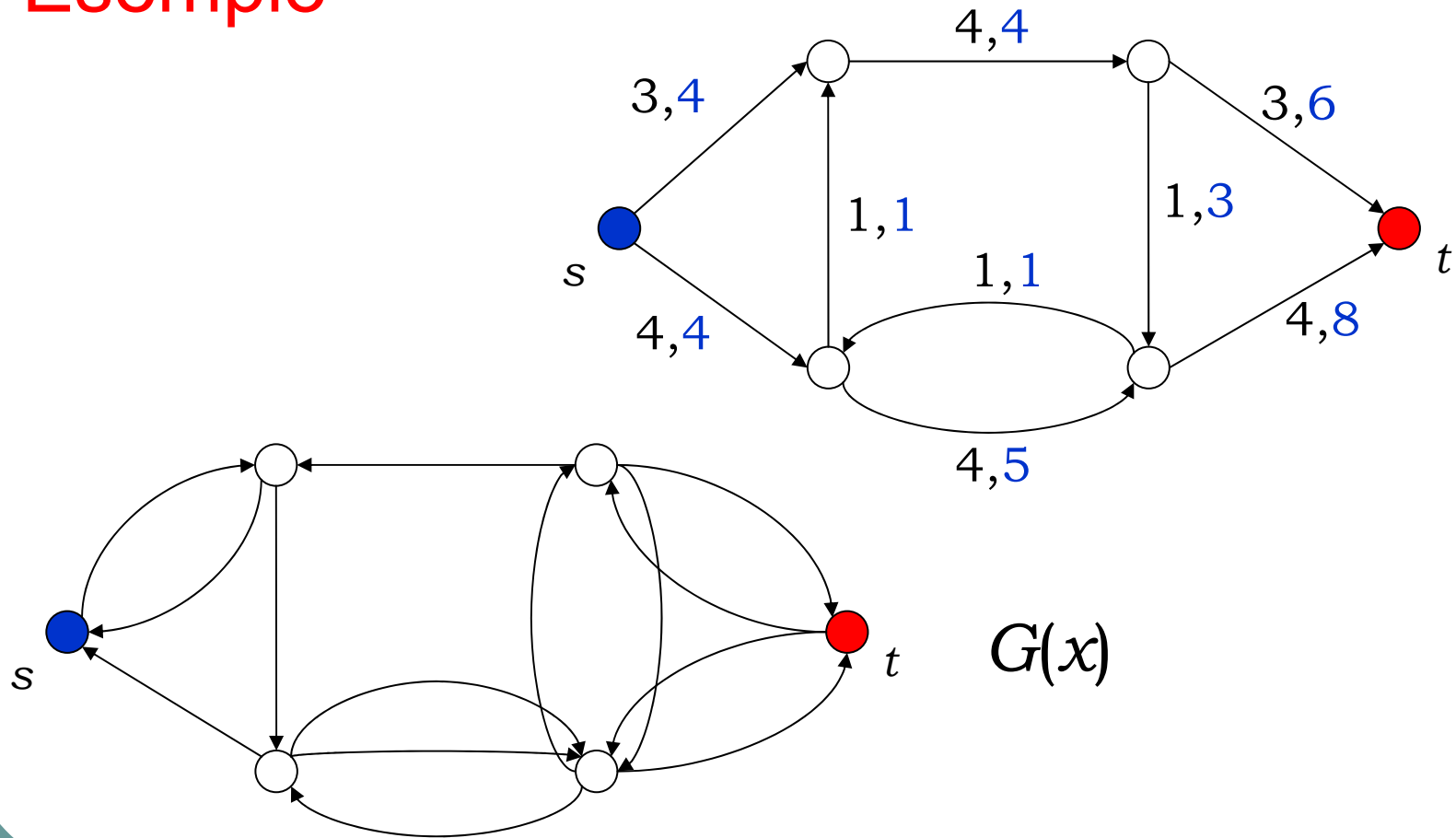
- $N(G(x)) = N$
- L'arco (i,j) appartiene ad $A(G(x))$ se e solo se l'arco (i,j) appartiene ad A e $x_{ij} < u_{ij}$ oppure (j,i) appartiene ad A e $x_{ji} > 0$.

Osservazione

Il grafo $G(x)$ non è un grafo **semplice**.

Una struttura per cercare cammini aumentanti

Esempio



Terminazione e complessità

Un (s, t) -cammino su $G(x)$ corrisponde ad un cammino aumentante su G . Pertanto, un cammino aumentante su G si può determinare in $O(m)$, “semplicemente” visitando $G(x)$.

Se u è intero e il grafo ammette un flusso finito, allora un bound sul numero di iterazioni dell’algoritmo è dato da k , dove k è il valore del massimo flusso.

Un bound banale per k si ottiene considerando un taglio con $R = \{s\}$. Se indichiamo con U la capacità massima degli archi, $u(\delta(R)) \leq nU$.

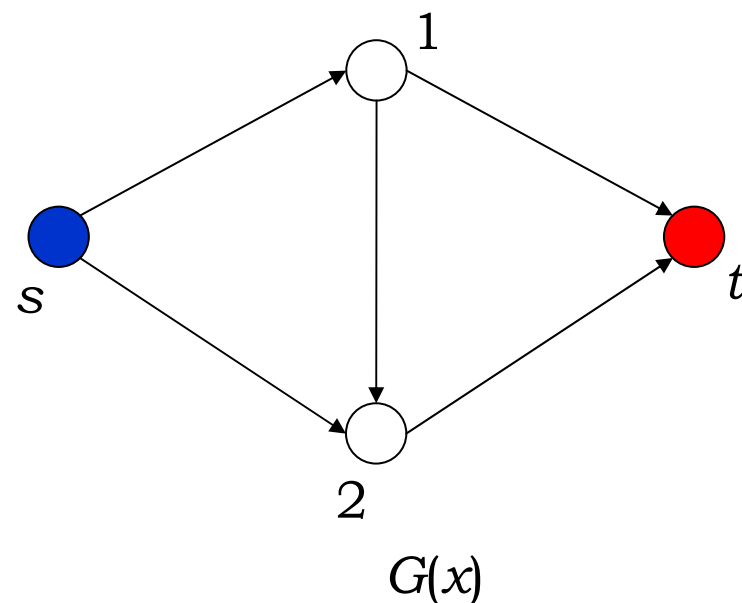
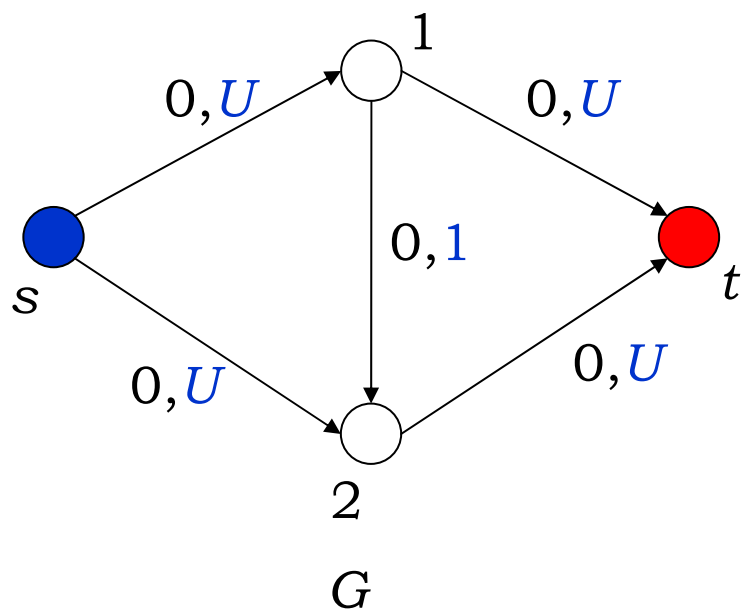
Pertanto, l’algoritmo basato sul cammino aumentante termina in al più $O(nmU)$ iterazioni se G ammette un flusso diverso da $+\infty$.

Anche se u è razionale, si può costruire un opportuno problema “scalato” e dimostrare la convergenza in un numero finito di passi.

Esempio

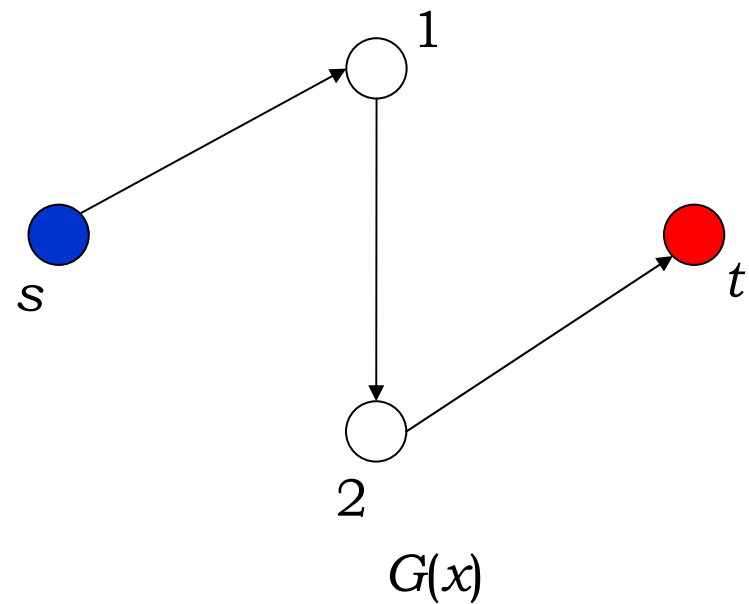
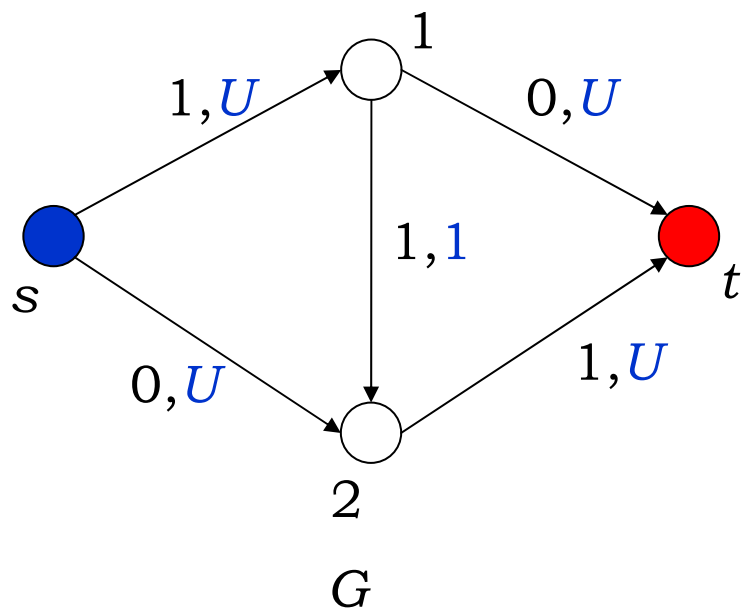
La complessità “pratica” dipende dalla scelta dei cammini aumentanti.

Consideriamo il seguente grafo:



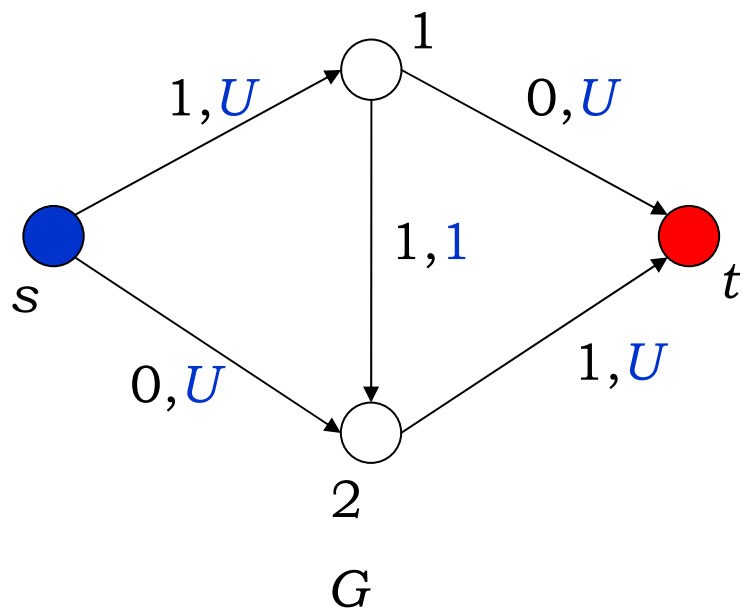
Esempio

Cammino aumentante su $G: \{s, 1, 2, t\}$. $\varepsilon = 1$

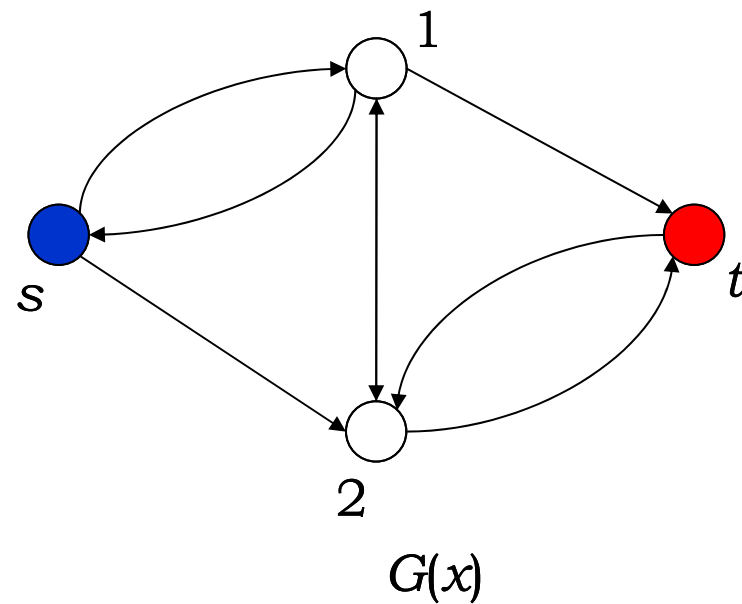


Esempio

Flusso di valore 1

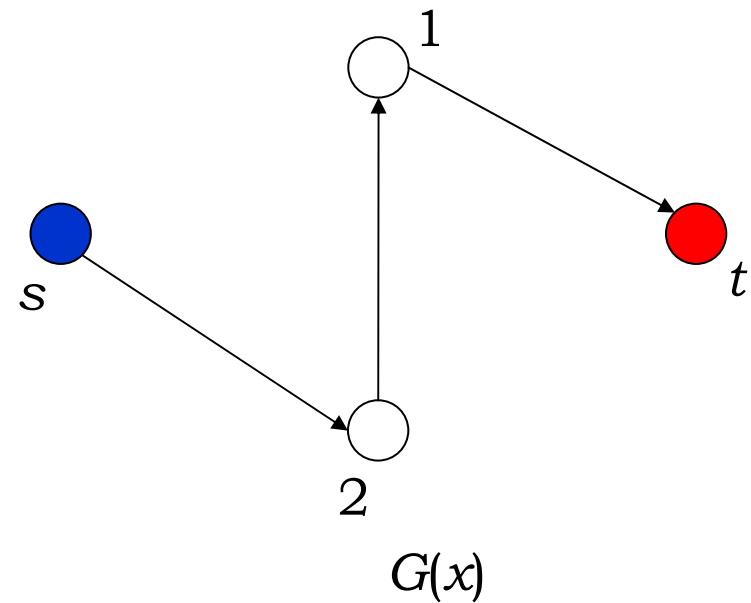
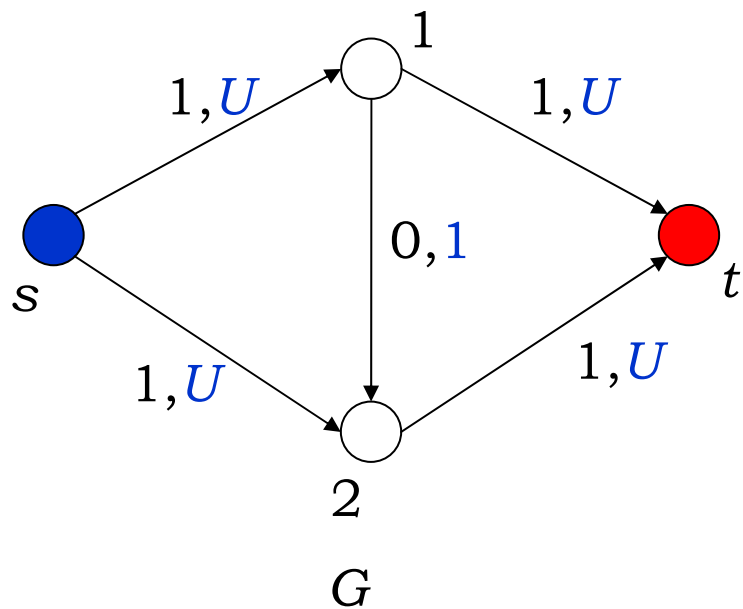


Grafo ausiliario



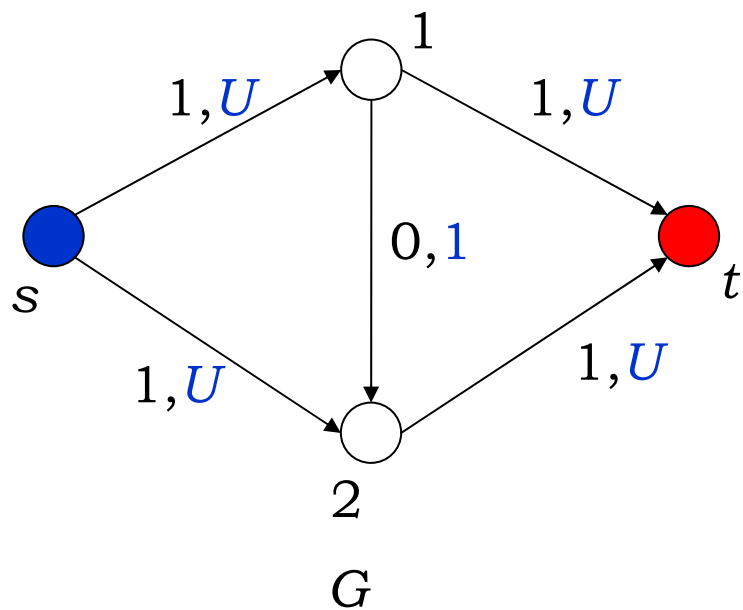
Esempio

Cammino aumentante su G : $\{s, 2, 1, t\}$, $\varepsilon=1$

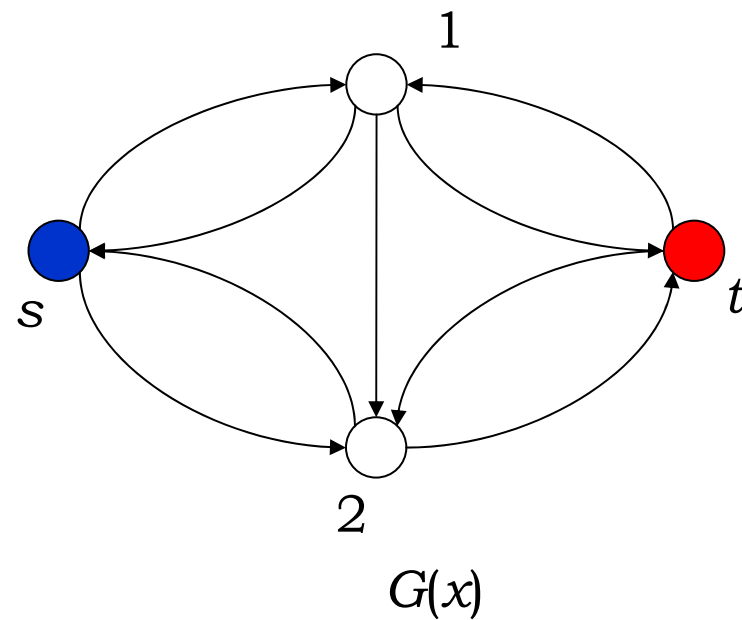


Esempio

Flusso di valore 2



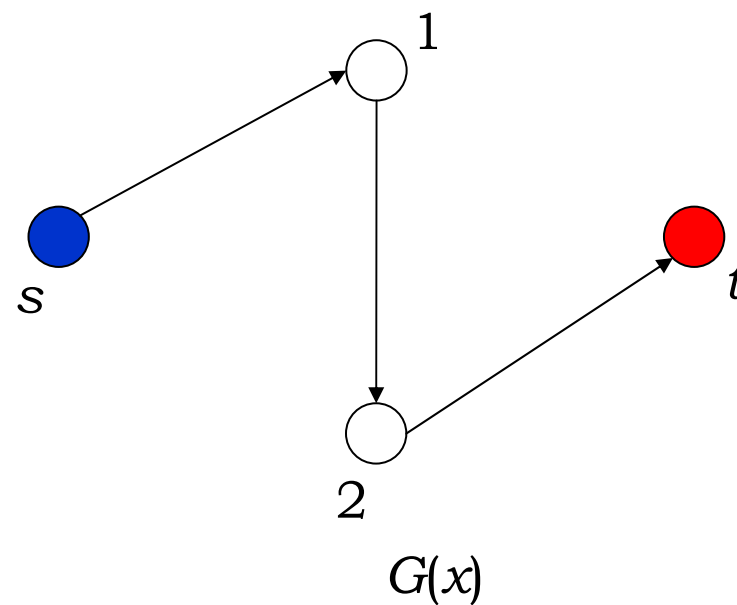
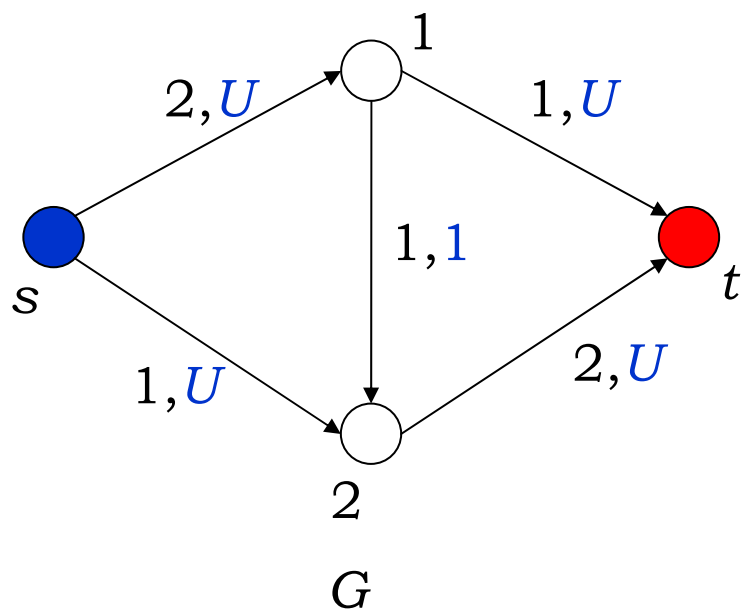
Grafo ausiliario



Esempio

Cammino aumentante su $G: \{s, 1, 2, t\}$. $\varepsilon = 1$

Flusso di valore 3



Esempio

Ripetendo questa scelta dei cammini aumentanti, l'algoritmo termina esattamente dopo $2U$ iterazioni!

Osservazione

Se il valore di U non fosse finito, l'algoritmo sul grafo precedente potrebbe NON terminare!

Una semplice criterio di scelta del cammino aumentante ad ogni iterazione elimina questa difficoltà e determina una complessità polinomiale dell'algoritmo di Ford e Fulkerson [cioè, non dipendente dalla capacità U]

Dimostremo che è sufficiente scegliere ad ogni iterazione il cammino su $G(x)$ che minimizza il numero di archi utilizzati (in altre parole, il **cammino aumentante più corto**).

[Edmonds e Karp]

Shortest augmenting path

Ad una generica iterazione dell'algoritmo si passa da un flusso x ad un flusso x' attraverso un cammino aumentante $P = \{v_0, v_1, \dots, v_k\}$.

Sia $d_x(v, w)$ la lunghezza del cammino da v a w avente il minor numero di archi.

Se P è il più corto cammino aumentante valgono le seguenti proprietà:

1. $d_x(s, v_i) = i$
2. $d_x(v_i, t) = k - i$

Inoltre, se un arco (v, w) di $G(x')$ non appartiene a $G(x)$, allora esiste un indice i tale che $v=v_i, w=v_{i-1}$. In altre parole, l'arco (w, v) era un arco del cammino aumentante più corto su $G(x)$.

Shortest augmenting path

Lemma 1

Per ogni $v \in N$, $d_{x'}(s, v) \geq d_x(s, v)$ e $d_{x'}(v, t) \geq d_x(v, t)$

Dimostrazione

Supponiamo che esista un nodo v tale che

$d_{x'}(s, v) < d_x(s, v)$ e scegliamo v in modo che $d_{x'}(s, v)$ sia la più piccola possibile.

Essendo $v \neq s$, $d_{x'}(s, v) > 0$.

Sia P' un cammino (s, v) in $G(x')$ e w il penultimo nodo di P' . Si ha:

$$d_x(s, v) > d_{x'}(s, v) = d_{x'}(s, w) + 1 \geq d_x(s, w) + 1$$

Dimostrazione

Se $d_x(s, v) > d_x(s, w) + 1$, allora l'arco (w, v) non appartiene a $G(x)$ (altrimenti $d_x(s, v) = d_x(s, w) + 1$).

Se l'arco (w, v) (che appartiene a $G(x')$) non appartiene a $G(x)$ si ha che esiste un indice i per cui $w = v_i$ e $v = v_{i-1}$.

Pertanto $d_x(s, v) = i - 1 > d_x(s, w) + 1 = i + 1$, e si ha una contraddizione.

Con un argomento simile si dimostra la seconda parte del lemma ■

Shortest augmenting path

Lemma 2

Durante l'esecuzione dell'algoritmo di Edmonds e Karp, un arco (i,j) scompare (e compare) in $G(x)$ al più $n/2$ volte.

Dimostrazione

Se un arco (i,j) “scompare” dalla rete ausiliaria, significa che esso è su un cammino aumentante e che il corrispondente arco in G si satura oppure si svuota. Pertanto, nella rete ausiliaria successiva compare l'arco (j,i) . Sia x_f il flusso al momento della “scomparsa dell'arco”. Supponiamo che ad una successiva iterazione l'arco (i,j) ricompaia in $G(x_h)$. Ciò significa che il cammino aumentante che ha generato x_h contiene l'arco (j,i) .

Shortest augmenting path

Allora, se x_g è il flusso a partire dal quale si è generato x_h , si ha [Lemma 1]

$$d_g(s, i) = d_g(s, j) + 1 \geq d_f(s, j) + 1 = d_f(s, i) + 2$$

Pertanto, nel passare dal flusso x_f al flusso x_h , $d(s, u)$ è aumentata almeno di 2. Poiché il massimo valore che può assumere $d(s, u)$ è n , un arco può scomparire e riapparire al più $n/2$ volte. ■

Shortest augmenting path

Lemma 3

L'algoritmo di Edmonds e Karp ha complessità $O(nm^2)$.

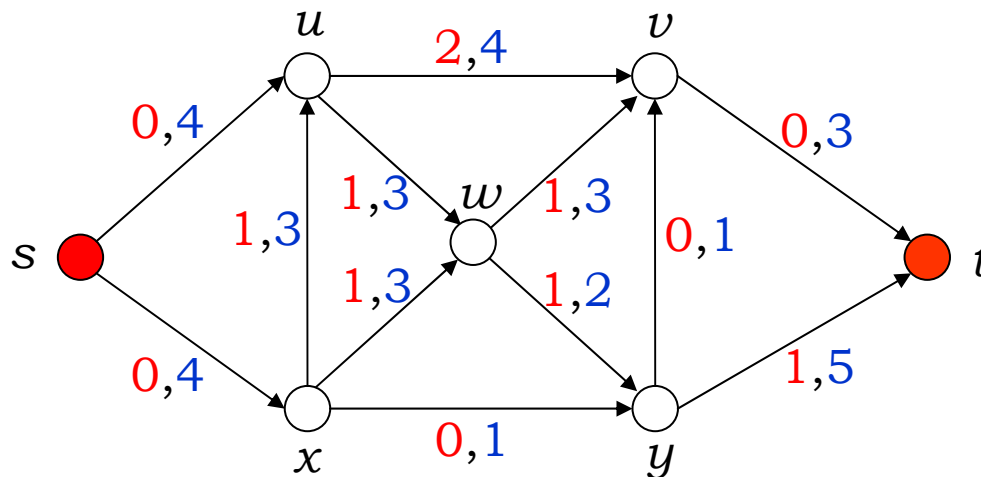
Dimostrazione

Ogni arco può “scompare” al più $n/2$ volte durante l'esecuzione dell'algoritmo [Lemma 2]. Ogni volta che effettuiamo un aumento di flusso, scompare almeno un arco. Pertanto, durante l'esecuzione, si hanno al più $mn/2$ “sparizioni”. Ogni operazione di aumento richiede $O(m)$ e, quindi, la complessità dell'algoritmo è $O(nm^2)$.



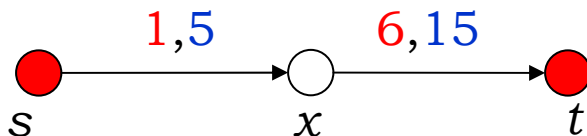
Problema di flusso con richiesta minima

Le etichette di colore rosso sulla seguente rete, rappresentano una quantità di flusso minima che deve essere trasportata dall'arco (l_{ij}, u_{ij})



Osservazione

Un problema di flusso con richiesta minima positiva sugli archi potrebbe non essere ammissibile.

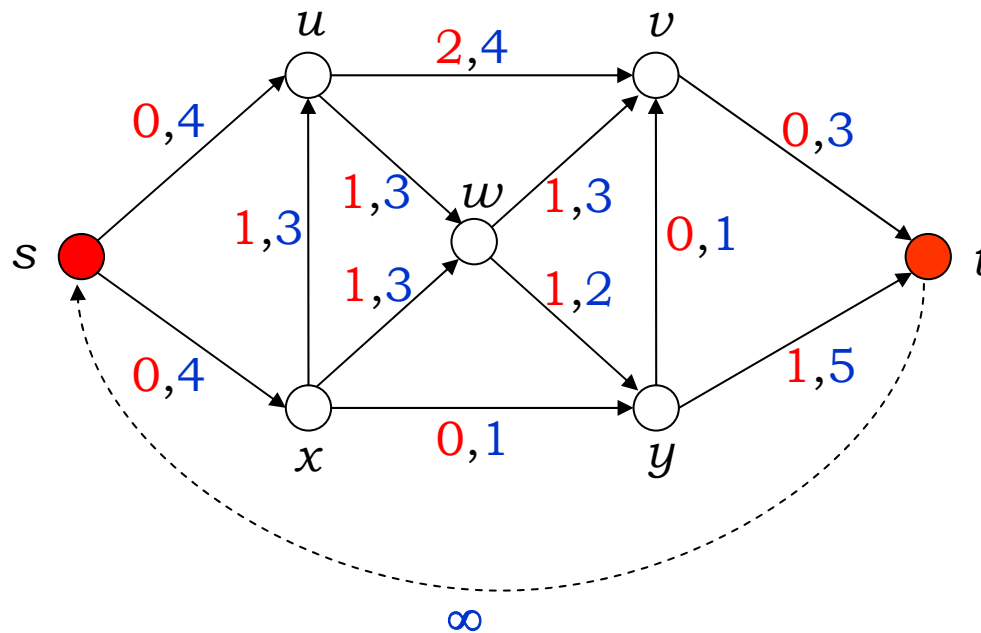


Prima di calcolare il massimo flusso sulla rete, vediamo se esiste un flusso ammissibile.

Trasformiamo il problema di flusso in un **problema di circolazione**, aggiungendo un arco (t, s) di capacità infinita.

1. Ammissibilità

Il problema di flusso ammette una soluzione ammissibile se e solo se il problema di circolazione è ammissibile



1. Ammissibilità

Nel problema di circolazione, in ogni nodo il flusso entrante è uguale al flusso uscente:

$$s: x_{su} + x_{sx} - x_{ts} = 0$$

$$u: x_{uw} + x_{uw} - x_{su} - x_{xu} = 0$$

$$v: x_{vt} - x_{uv} - x_{wv} - x_{yv} = 0$$

$$w: x_{wv} + x_{wy} - x_{uw} - x_{xw} = 0$$

$$x: x_{xu} + x_{xw} + x_{xy} - x_{sx} = 0$$

$$y: x_{yv} + x_{yt} - x_{wy} - x_{xy} = 0$$

$$t: x_{ts} - x_{vt} - x_{yt} = 0$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \quad \forall \text{ arco } (i,j)$$

Se operiamo la sostituzione $x_{ij} = x'_{ij} + l_{ij}$ si ottiene:

1. Ammissibilità

$$s: x'_{su} + x'_{sx} - x'_{ts} = b(s) = 0$$

$$u: x'_{uv} + x'_{uw} - x'_{su} - x'_{xu} = b(u) = -2$$

$$v: x'_{vt} - x'_{uv} - x'_{wv} - x'_{yv} = b(v) = 3$$

$$w: x'_{wv} + x'_{wy} - x'_{uw} - x'_{xw} = b(w) = 0$$

$$x: x'_{xu} + x'_{xw} + x'_{xy} - x'_{sx} = b(x) = -2$$

$$y: x'_{yv} + x'_{yt} - x'_{wy} - x'_{xy} = b(y) = 0$$

$$t: x'_{ts} - x'_{vt} - x'_{yt} = b(t) = 1$$

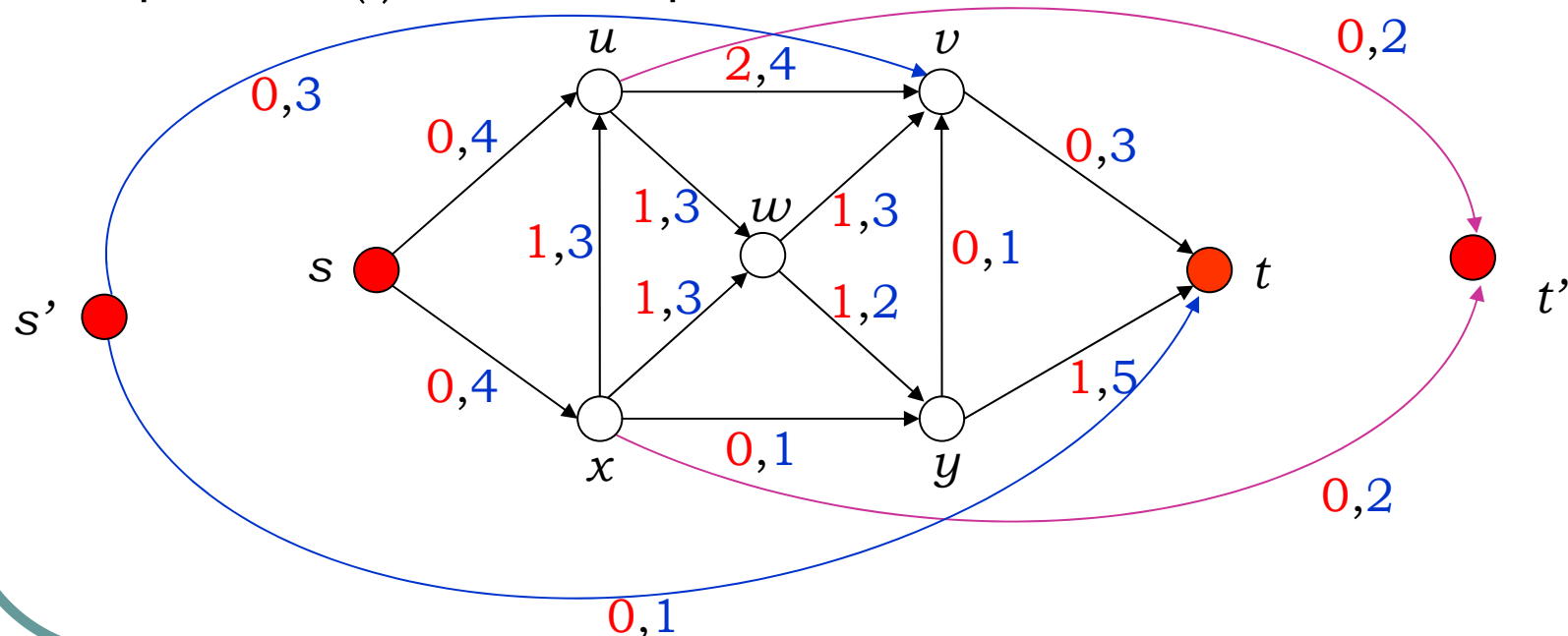
$$0 \leq x'_{ij} \leq l_{ij} - u_{ij} \quad \forall \text{ arco } (i,j)$$

Questo è un problema di circolazione equivalente con “particolari” vincoli di bilanciamento (rhs non tutti uguali a zero).

1. Ammissibilità

Quindi, introduciamo due nodi s' e t' .

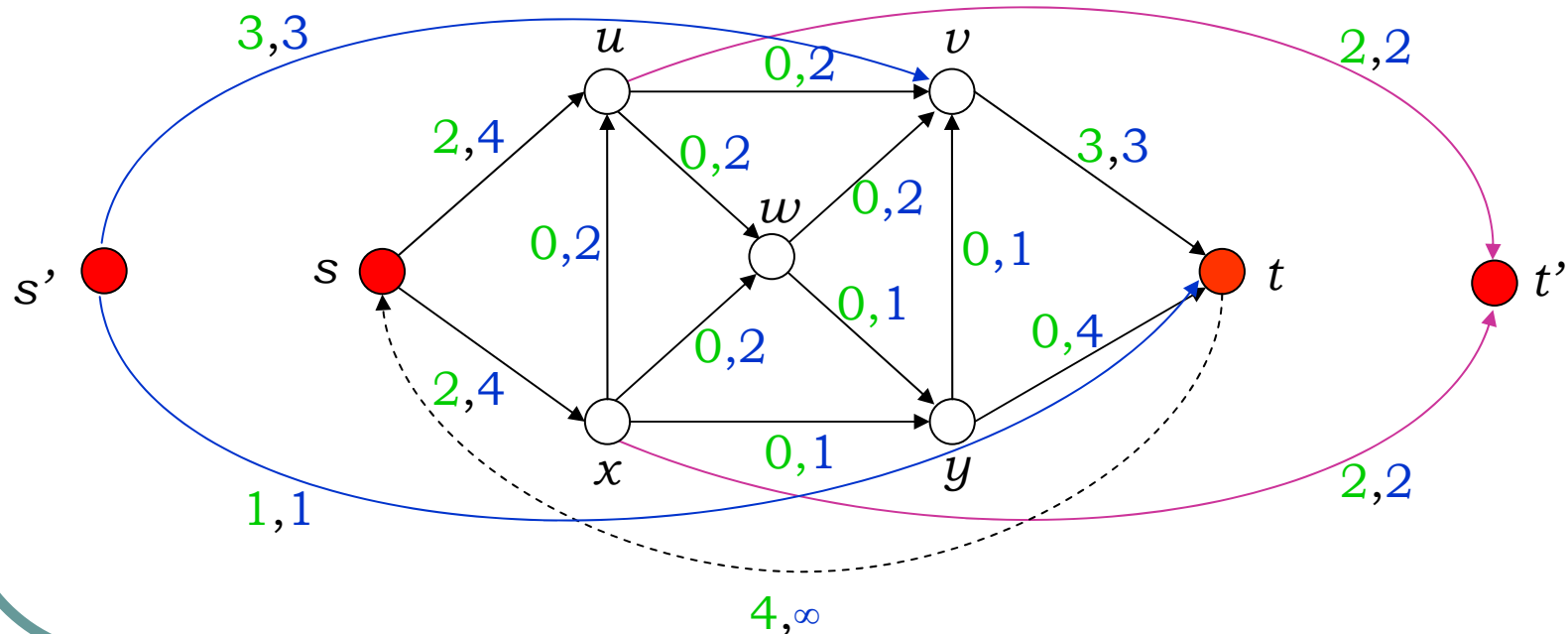
1. Per ogni nodo i con $b(i) > 0$ aggiungiamo un arco (s', i) con capacità $b(i)$ e richiesta pari a 0
2. Per ogni nodo i con $b(i) < 0$ aggiungiamo un arco (i, t') con capacità $-b(i)$ and richiesta pari a 0



1. Ammissibilità

Se il flusso massimo satura gli archi (s', i) o, equivalentemente, gli archi (i, t) allora il problema iniziale è ammissibile, altrimenti non lo è.

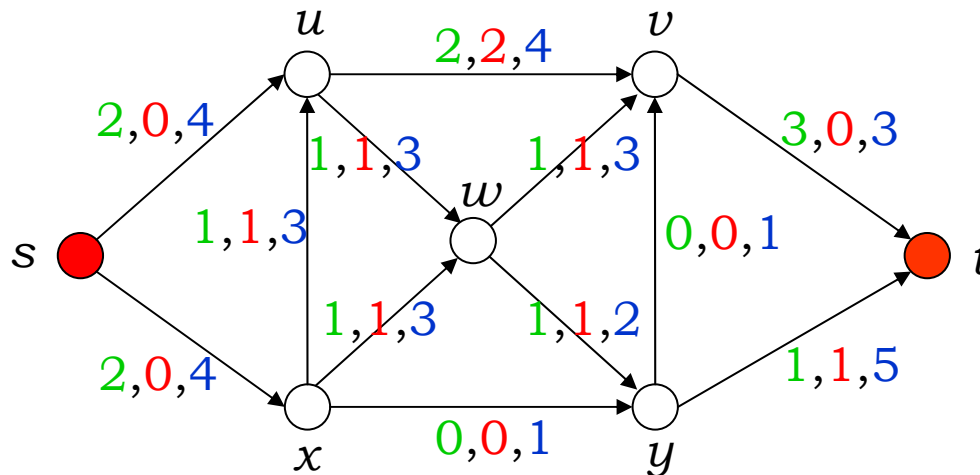
flusso, capacità →



1. Ammissibilità

Per ottenere una soluzione ammissibile del problema iniziale si eliminano gli archi aggiunti e si ripristinano le variabili originali $x_{ij} = x'_{ij} + l_{ij}$.

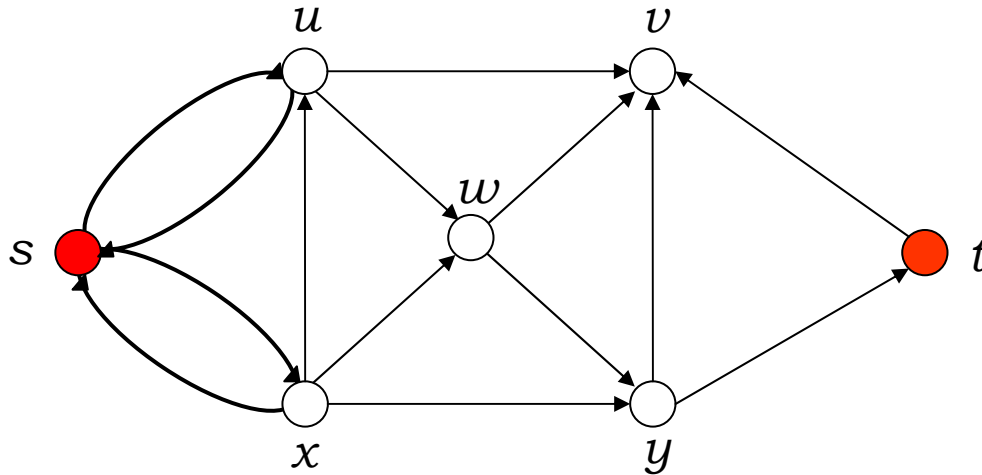
flusso, richiesta, capacità



2. Ottimalità

A partire dal flusso ammissibile trovato, determiniamo il flusso massimo.

Costruiamo il grafo ausiliario mettendo un arco “forward” se $x_{ij} < u_{ij}$ e un arco “reverse” se $x_{ij} > l_{ij}$ e cerchiamo un cammino aumentante.

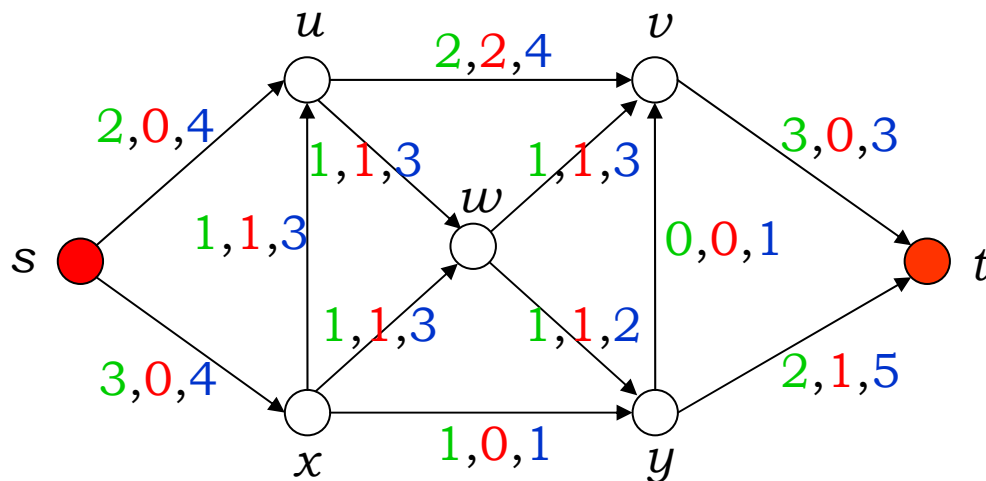


Ottimalità

Ad esempio un cammino aumentante è $P: \{s, x, y, t\}$.

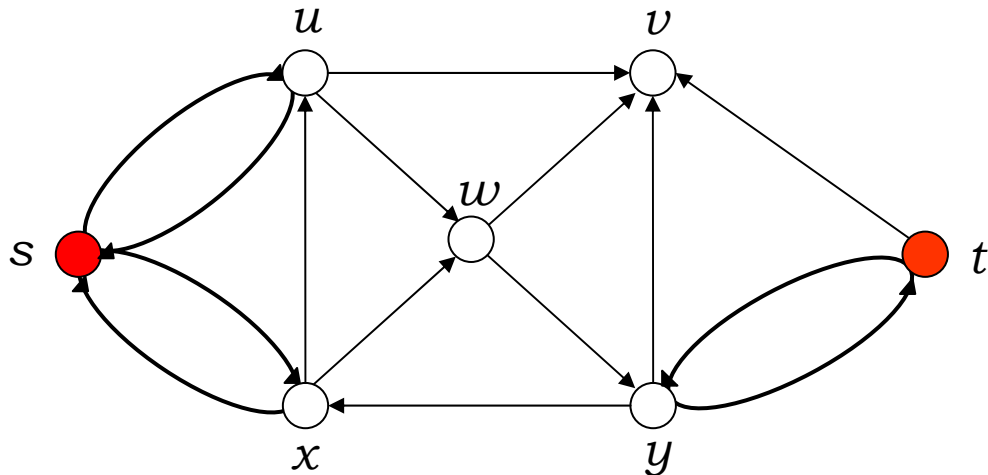
Il massimo flusso inviabile su tale cammino è il minimo valore tra $(u_{ij} - x_{ij})$ per ogni arco “forward” e $(x_{ij} - l_{ij})$ per ogni arco “reverse” appartenenti al cammino.

In questo caso: $\min\{2, 1, 4\} = 1$.



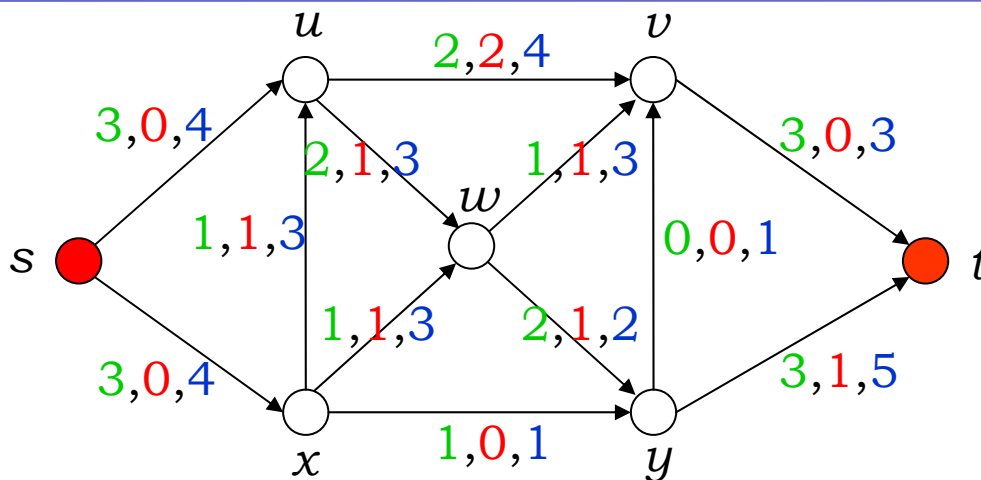
Ottimalità

Il nuovo grafo residuale è:

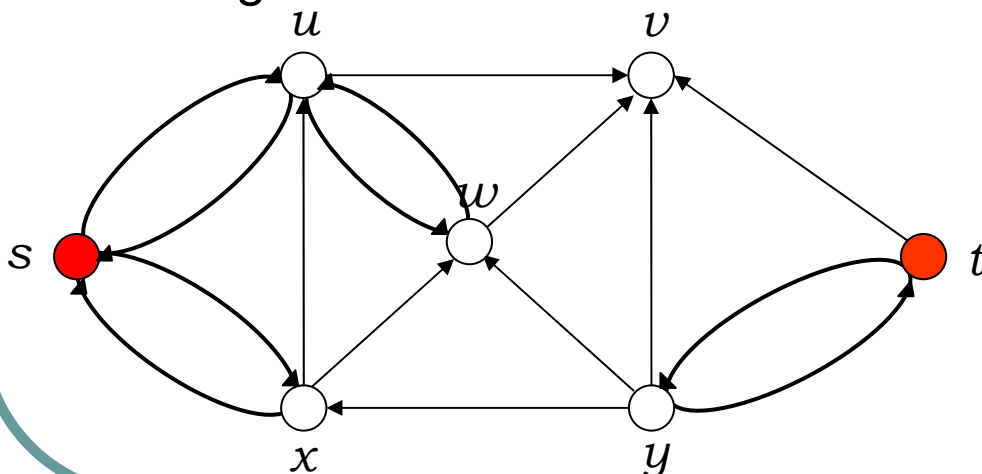


Un cammino aumentante è $P: \{s, u, w, y, t\}$, sempre di valore 1.
Aggiorniamo il flusso:

Ottimalità



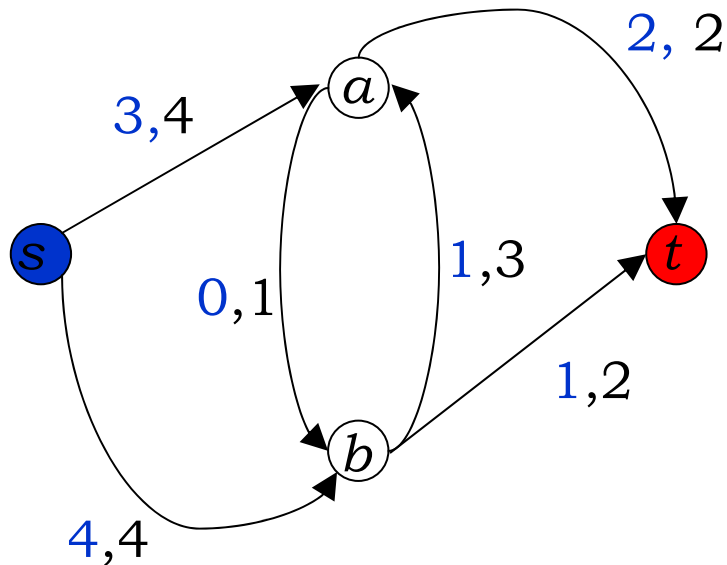
Il nuovo grafo residuale è:



Sul grafo residuale non esiste un cammino aumentante, quindi il flusso trovato è ottimo.

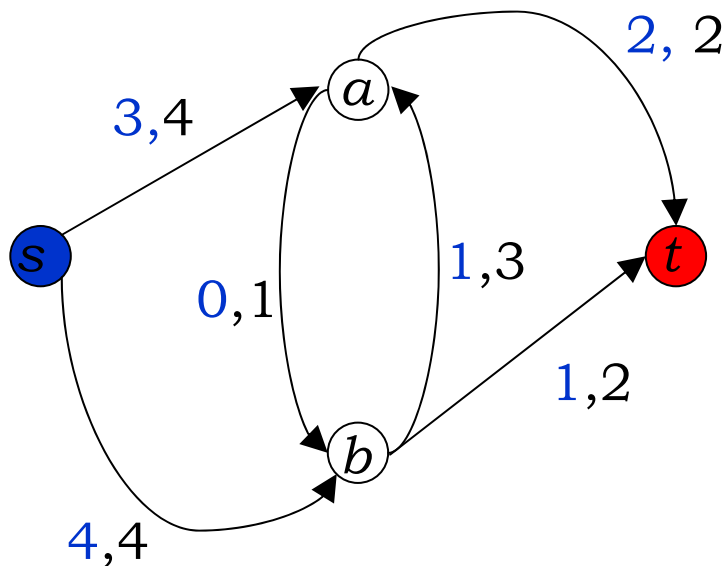
Un nuovo algoritmo

Consideriamo il seguente grafo e la seguente distribuzione di flusso: (in nero la capacità, in blu il flusso)



Un nuovo algoritmo

Il flusso **NON** è ammissibile perché non sono soddisfatti i vincoli di bilanciamento ai nodi (flusso in entrata > flusso in uscita)



$$e_x(a) = 3 + 1 - 2 - 0 = 2 > 0$$

$$e_x(b) = 4 + 0 - 1 - 1 = 2 > 0$$

Tuttavia, sono soddisfatti i vincoli di capacità sugli archi.

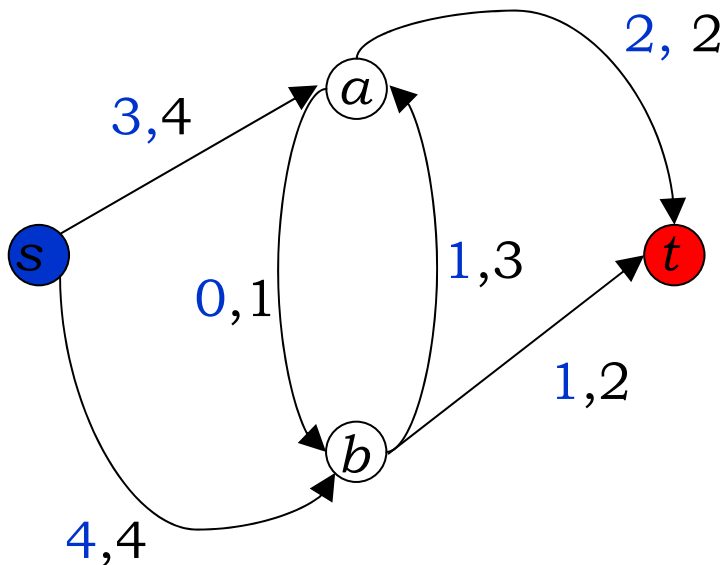
$e_x(i)$ = "eccesso" di flusso nel nodo i

Quasi-flusso

Un vettore $x \in \mathbb{Z}_+^{|A|}$ tale che:

1. $e_x(n) \geq 0$ per ogni nodo $n \in N \setminus \{s, t\}$
2. $0 \leq x_{ij} \leq u_{ij}$ per ogni arco $(i, j) \in A$

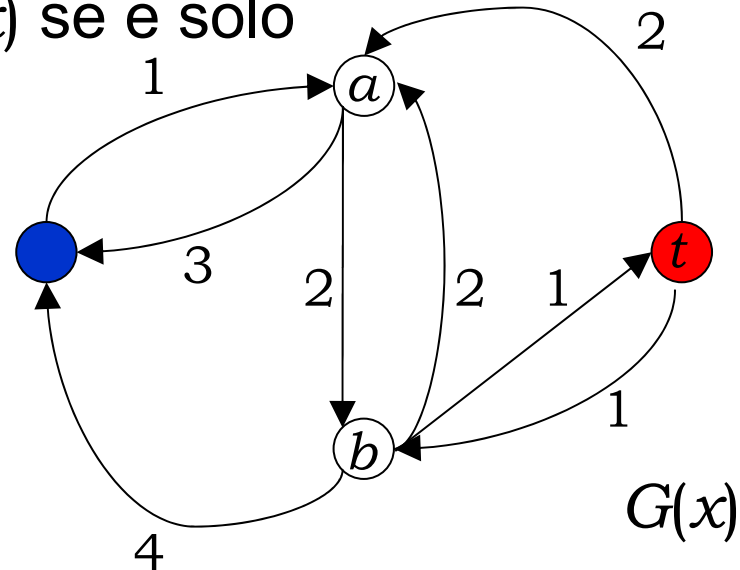
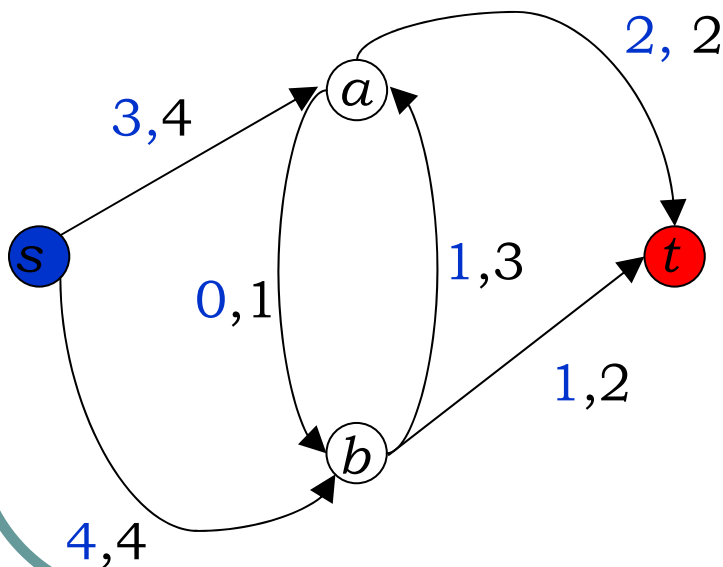
si dice **quasi-flusso**.



Ad un quasi-flusso può essere associata una rete residuale $G(x)$ con le seguenti caratteristiche:

Rete residuale

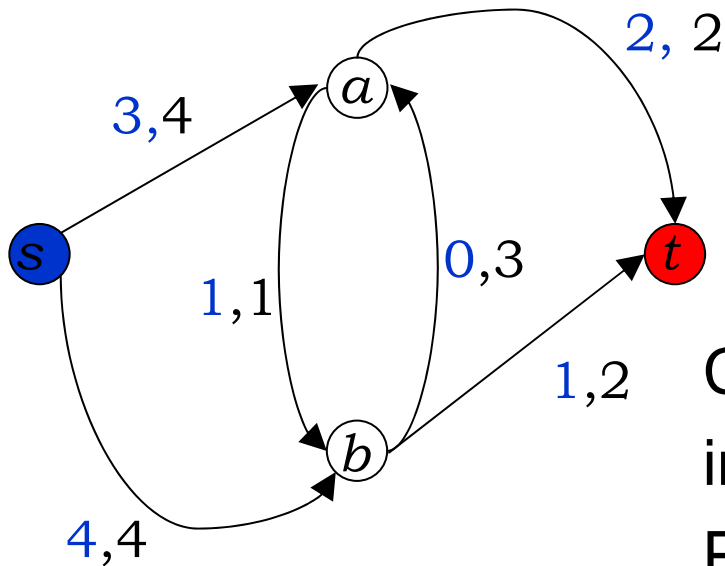
1. Esiste un arco (i,j) in $G(x)$ se e solo se $x_{ji} > 0$ oppure $x_{ij} < u_{ij}$ (eliminiamo eventuali archi paralleli)



2. L'etichetta dell'arco (i,j) in $G(x)$ è pari a $u'_{ij} = u_{ij} - x_{ij} + x_{ji}$

L'operazione "push"

La rete residuale ci dice che possiamo "spingere" 2 unità di flusso da a verso b , ottenendo:



Cosa succede all'eccesso di flusso in a e in b ?

Prima: $e_x(a) = 2$; $e_x(b) = 2$

Dopo: $e_x(a) = 0$; $e_x(b) = 4$

L'operazione “push”

L'operazione push ha modificato l'eccesso di flusso nei nodi a e b , ma non ha causato violazione dei vincoli di capacità sugli archi.

In particolare, siamo passati da un quasi-flusso ammissibile ad un nuovo quasi-flusso ammissibile (ovvero tale che $e_x(n) \geq 0$ per ogni nodo $n \in N \setminus \{s, t\}$)

Inoltre, l'eccesso di flusso su a è stato azzerato.

Osservazioni

Se tutti i nodi di un quasi-flusso (tranne s e t) hanno $e_x(i) = 0$, allora il quasi-flusso è un flusso ammissibile.

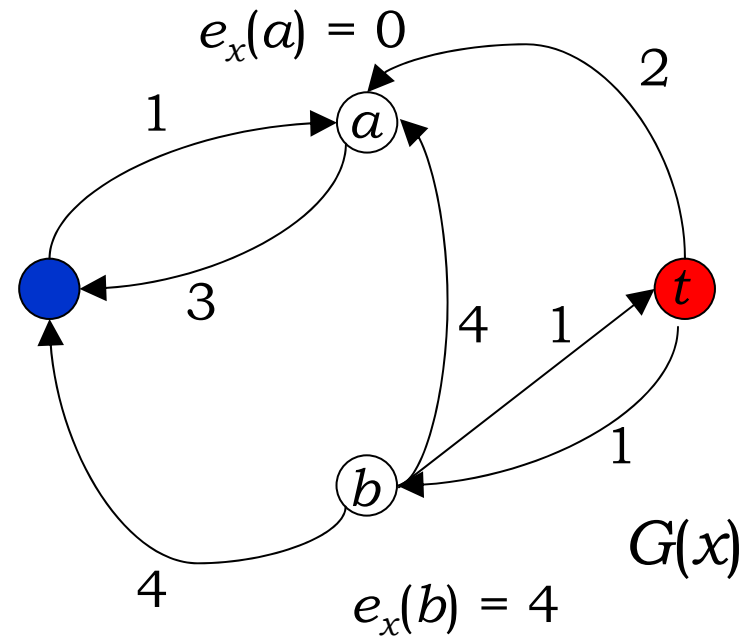
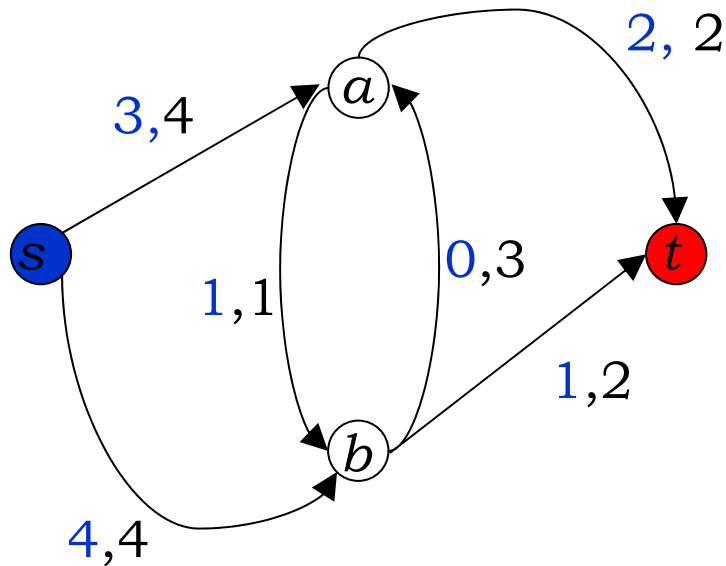
Dato un flusso ammissibile su G , s ha un eccesso di flusso negativo e t ha un eccesso di flusso positivo.

Per garantire l'ammissibilità del quasi-flusso, si deve effettuare un'operazione push su un arco (i, j) spingendo un valore di flusso pari a $\min \{u'_{ij}, e_x(i)\}$

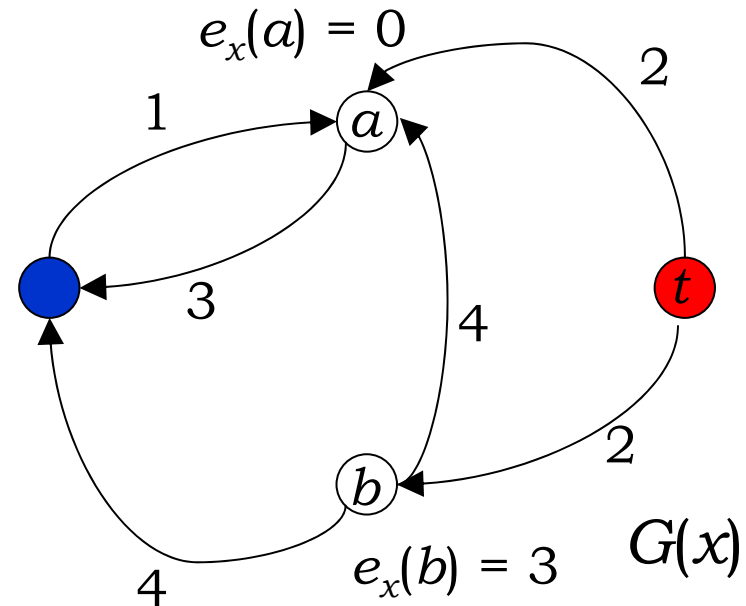
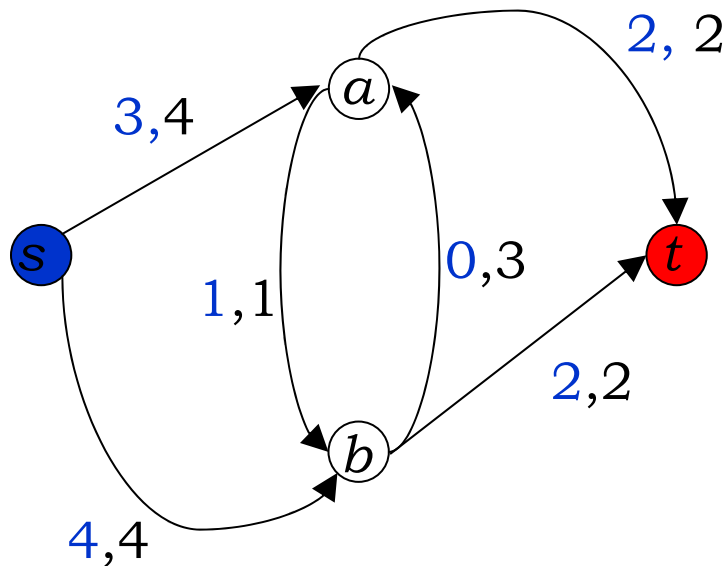
Per decrementare l'eccesso di flusso si cerca di spingere flusso verso il pozzo finché è possibile. Altrimenti si invia flusso verso la sorgente.

L'operazione "push"

In questo caso scegliamo l'arco (b, t) ($1 = \min \{e_x(b), u'_{bt}\}$)



L'operazione "push"

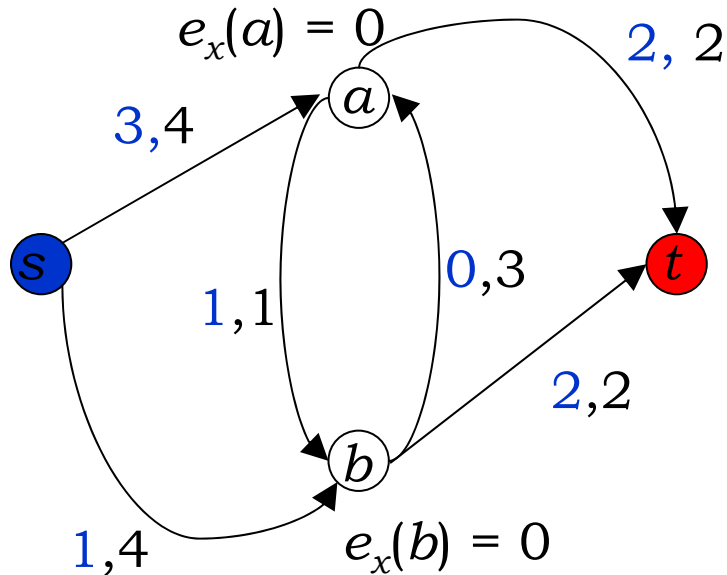


Ora abbiamo solo due scelte possibili per decrementare $e_x(b)$: l'arco (b, a) e l'arco (b, s) . Tuttavia, se scegliamo l'arco (b, a) incrementiamo $e_x(a)$.

Quindi, scegliamo (b, s) ($3 = \min \{e_x(b), u'_{bs}\}$)

L'operazione "push"

Abbiamo ottenuto un flusso ammissibile.



È ottimo?

Come si possono formalizzare le scelte effettuate in precedenza?

Etichettatura

Definizioni

- Un nodo i di G si dice attivo se $e_x(i) > 0$
- Un vettore $\mathbf{d} \in (\mathbb{Z}_+ \cup \{+\infty\})^{|A|}$ è un'etichettatura valida rispetto ad un quasi-flusso x se:
 1. $d(s) = n, d(t) = 0$
 2. Per ogni arco (i,j) di $G(x)$, $d(i) \leq d(j) + 1$

Osservazione

Indichiamo con $d_x(i,t)$ il cammino minimo (in termini di numero di archi) da i a t su $G(x)$.

Se scegliamo come etichette $d(i) = d_x(i,t)$, tutte le condizioni precedenti, tranne $d(s) = n$, sono soddisfatte.

Inizializzazione

Dato un grafo $G = (N, A)$ è sempre possibile individuare un quasi-flusso ammissibile e una etichettatura valida, ponendo:

(Inizializzazione)

1. $x_{si} = u_{si}$, per ogni arco (s, i) uscente da s
2. $x_{ij} = 0$ per tutti gli altri archi di A
3. $d(s) = n$, $d(i) = 0$ per tutti gli altri nodi di N

Teorema

Se x è un quasi-flusso ammissibile e d è un'etichettatura valida per x , allora esiste un (s, t) -taglio $\delta(R)$ tale che $x_{ij} = u_{ij}$ per ogni $(i, j) \in \delta(R)$ e $x_{ij} = 0$ per ogni $(i, j) \in \delta(R)$

Push-relabel

Conseguenza

Se x è un flusso ammissibile e ammette un'etichettatura valida, allora x è un massimo flusso.

Problema

Come costruire un flusso ammissibile e un'etichettatura valida a partire da un quasi-flusso ammissibile e da un'etichettatura valida?

Idea

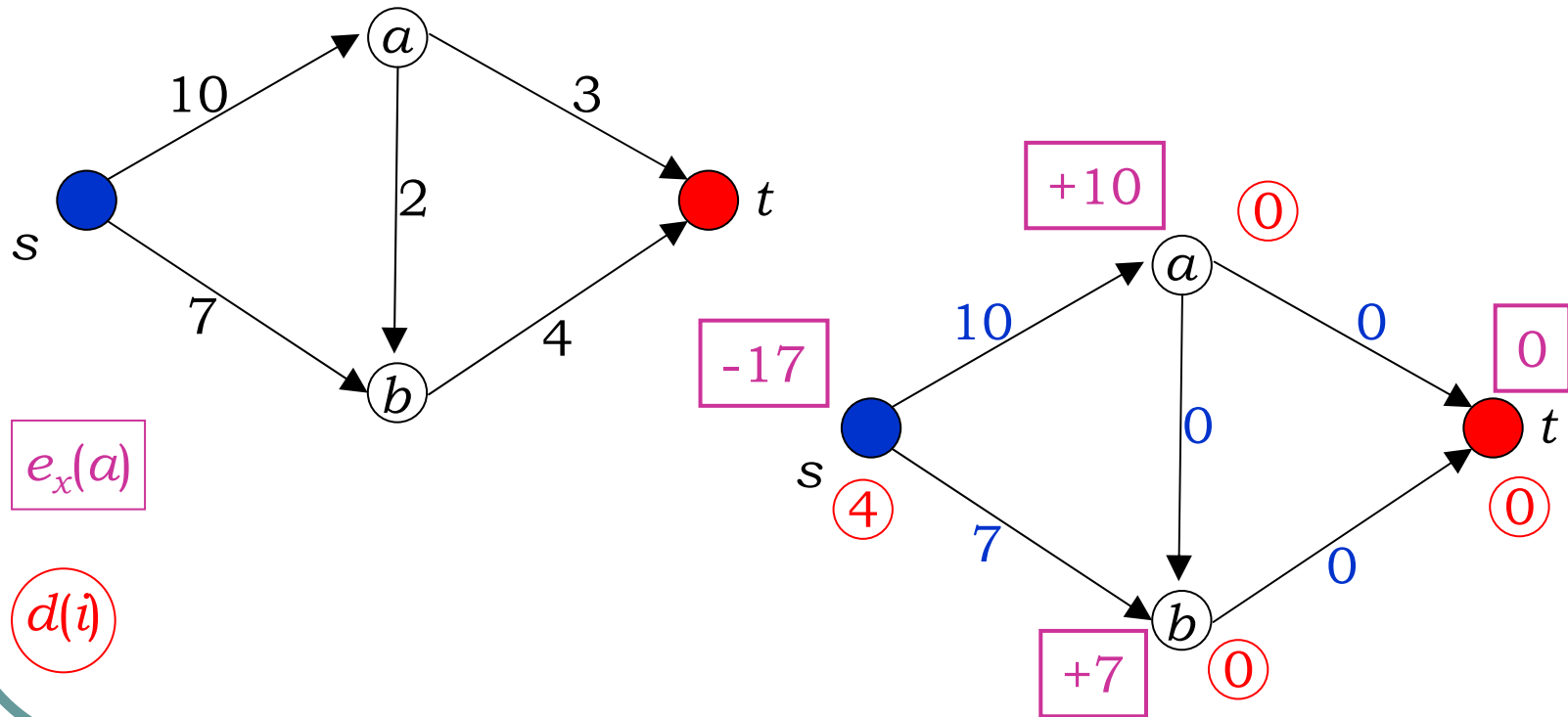
Spingo flusso da un nodo attivo lungo un arco (i,j) con la proprietà che $d(i) = d(j) + 1$

Un arco (i,j) con questa proprietà si dice **ammissibile**

Push-relabel

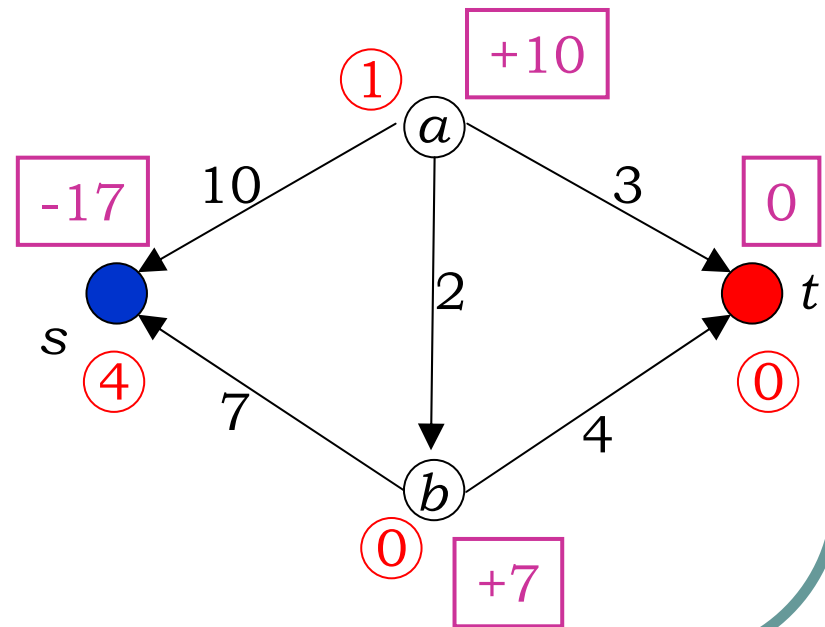
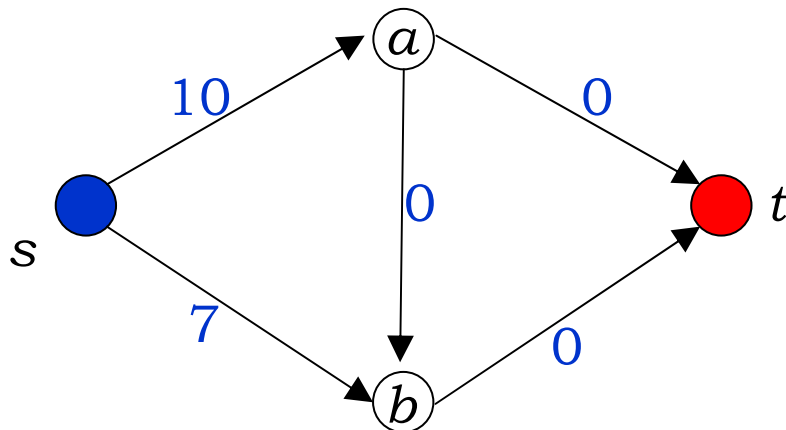
Osservazione

Possono non esistere archi ammissibili



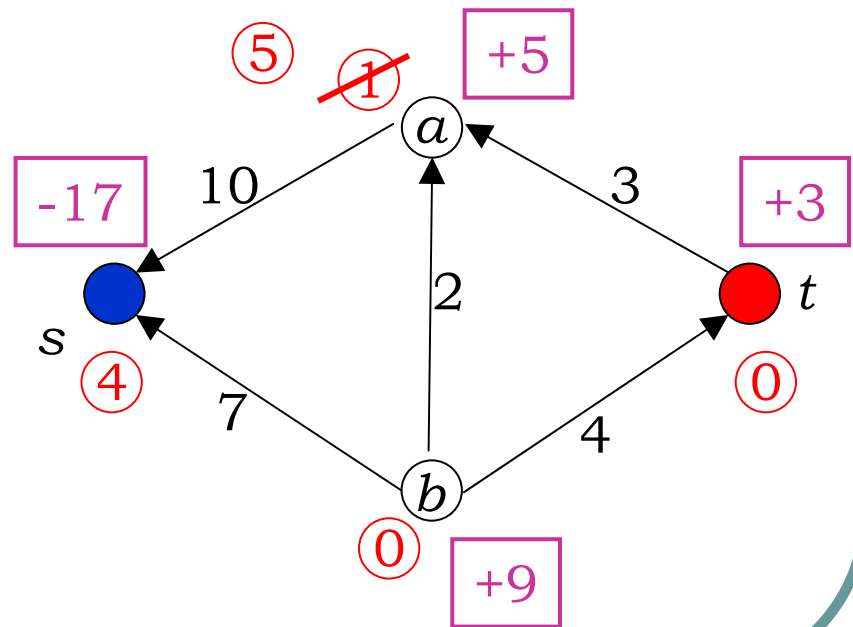
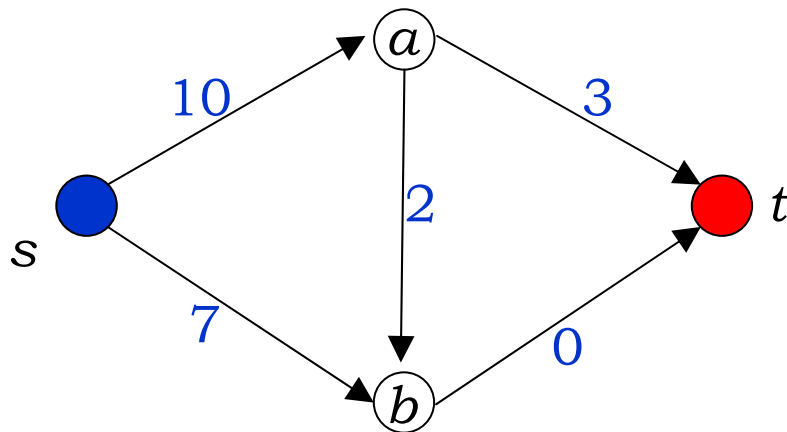
Push-relabel

Tuttavia, selezionando un nodo attivo i e ponendo $d(i) = \min \{d(j) + 1\}$, per (i,j) arco di $G(x)$, ottengo una nuova etichettatura valida e almeno un arco ammissibile (**relabel**)



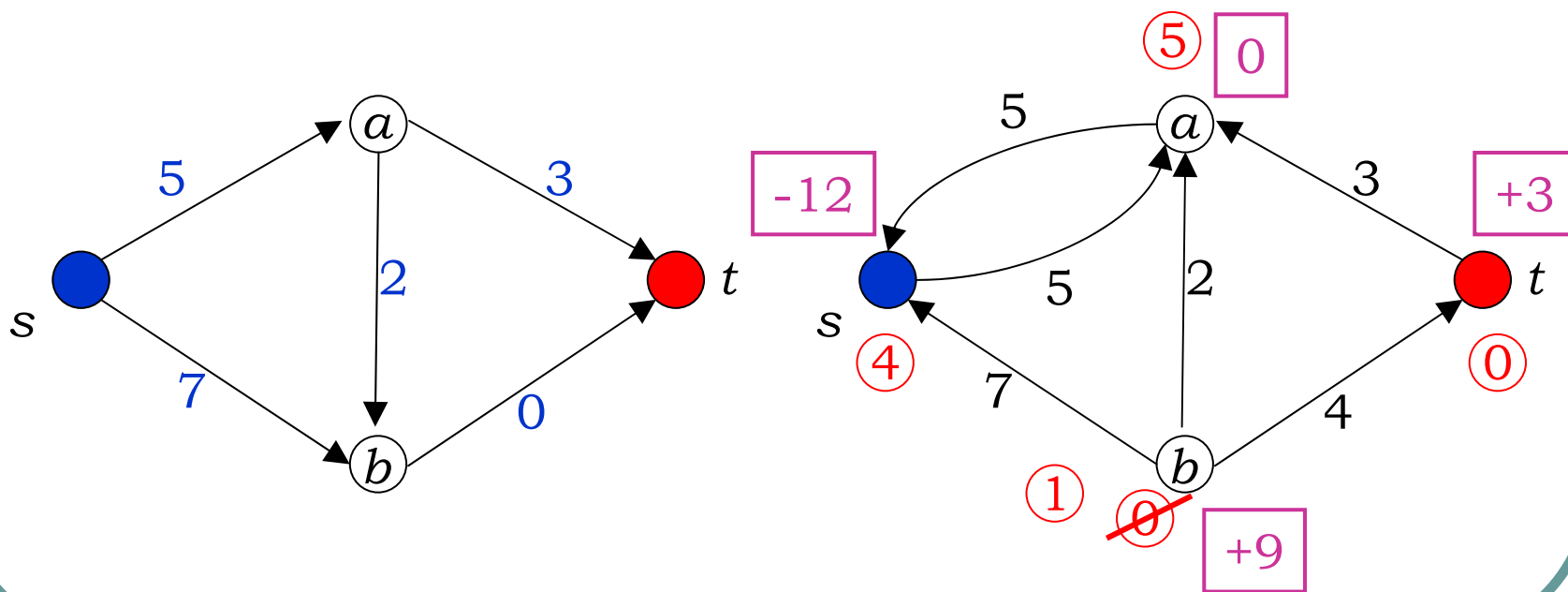
Push-relabel

A questo punto posso effettuare 2 push dal nodo a , una di valore 3 sull'arco (a,t) e una di valore 2 sull'arco (a,b) . a è ancora attivo \Rightarrow relabel



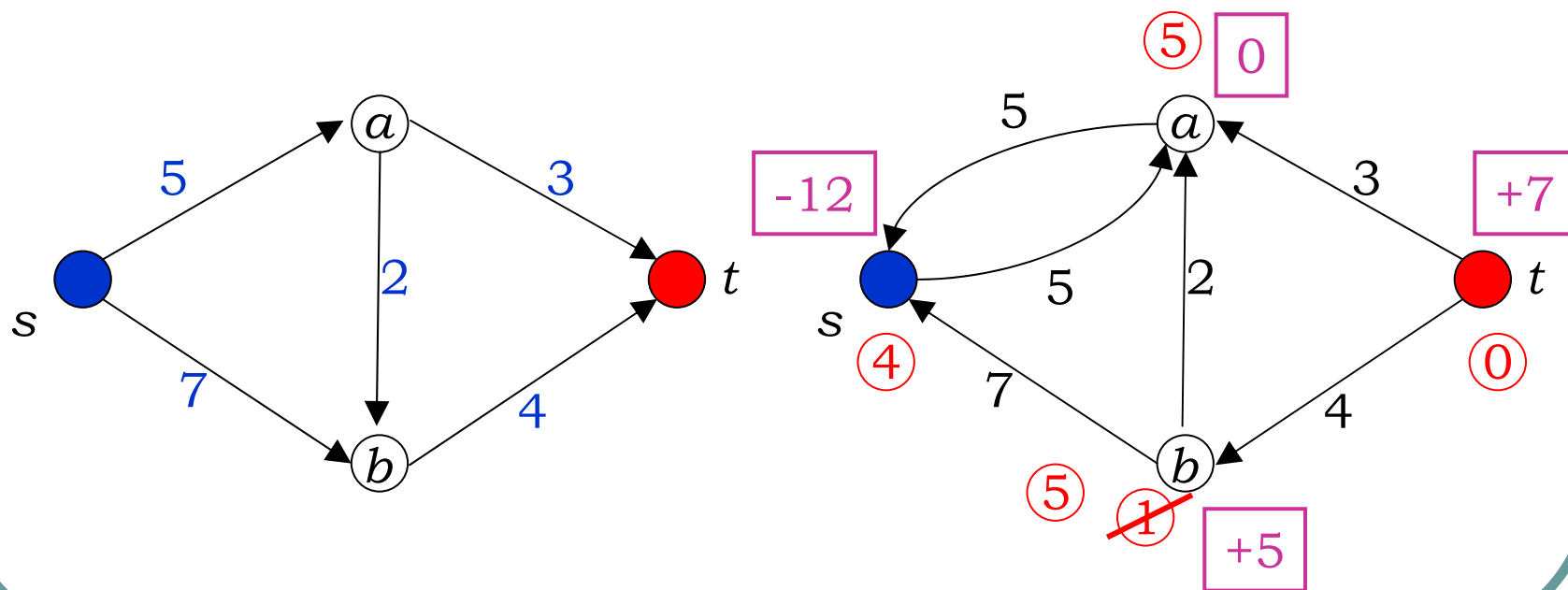
Push-relabel

Dopo una push di 5 sull'arco (a, s) si ottiene, con b unico nodo attivo \Rightarrow relabel. Dopo il relabel effettuo una push di 4 su (b, t)



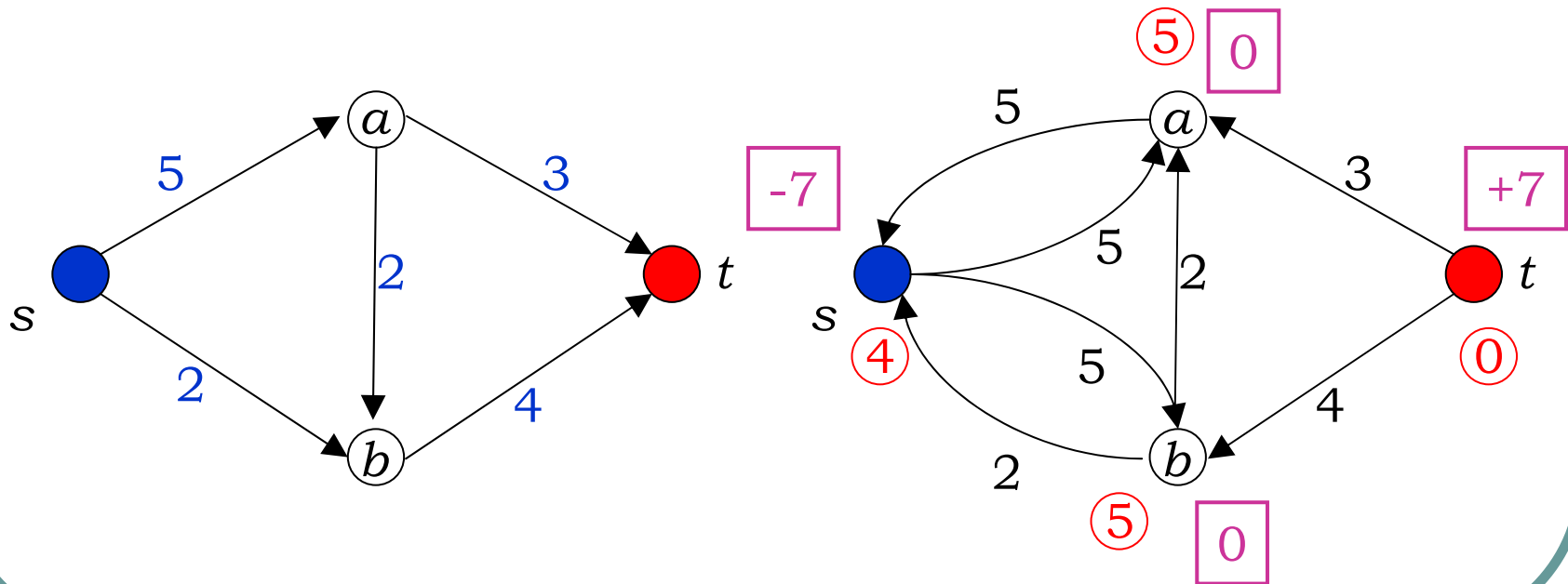
Push-relabel

Adesso b è ancora attivo ma non c'è un arco attivo \Rightarrow relabel. A questo punto si effettua una push di valore 5 su (b,s) .



Push-relabel

Ora: 1) non ci sono nodi attivi
2) l'etichettatura è valida
quindi la soluzione è ottima!



Push-relabel

Algoritmo push-relabel

x quasi-flusso, d etichette

Inizializza x e d ;

while (x non è un flusso)

 scegli un nodo attivo i su $G(x)$;

 while (esiste un arco (i, j) ammissibile)

 push (i, j)

 if (i è attivo)

 relabel i

Problema

Un complesso programma di calcolo, costituito da 3 moduli, deve essere eseguito su un calcolatore con 2 processori.

In tabella sono riportati i costi di assegnazione dei moduli ai processori $P1$ e $P2$:

	M_1	M_2	M_3
C_{P1}	20	23	8
C_{P2}	15	14	19

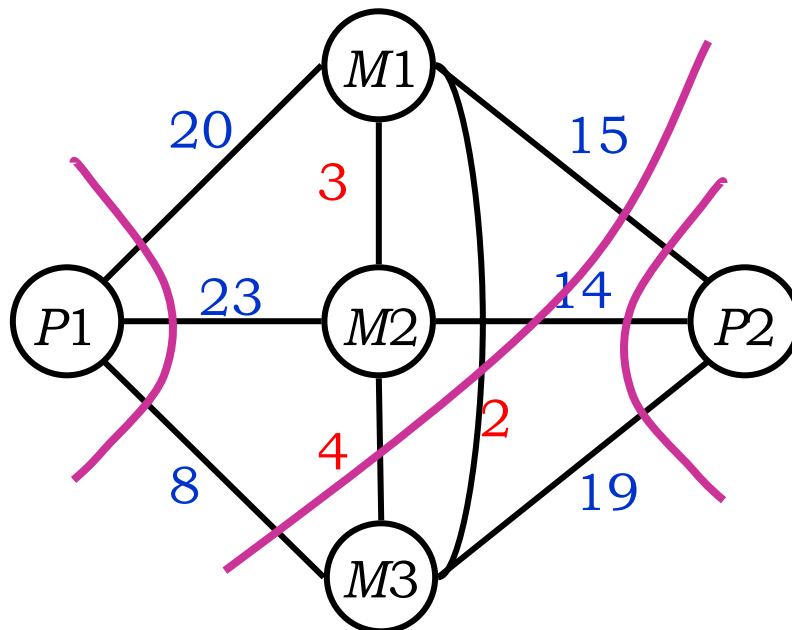
Problema

In tabella sono invece riportati i costi di intercomunicazione c_{ij} tra processori, qualora due moduli vengano assegnati a due processori diversi:

	M_1	M_2	M_3
M_1	0	3	2
M_2	3	0	4
M_3	2	4	0

Problema

Consideriamo il seguente grafo:



un (s, t) -taglio su G corrisponde ad un assegnamento di moduli a processori e il suo costo è pari al costo di assegnazione più il costo di intercomunicazione. Come si determina un (s, t) -taglio minimo se G è simmetrico? E il taglio “complessivamente” minimo di G ?

Tagli minimi su grafi simmetrici

Problema del taglio minimo

Dati

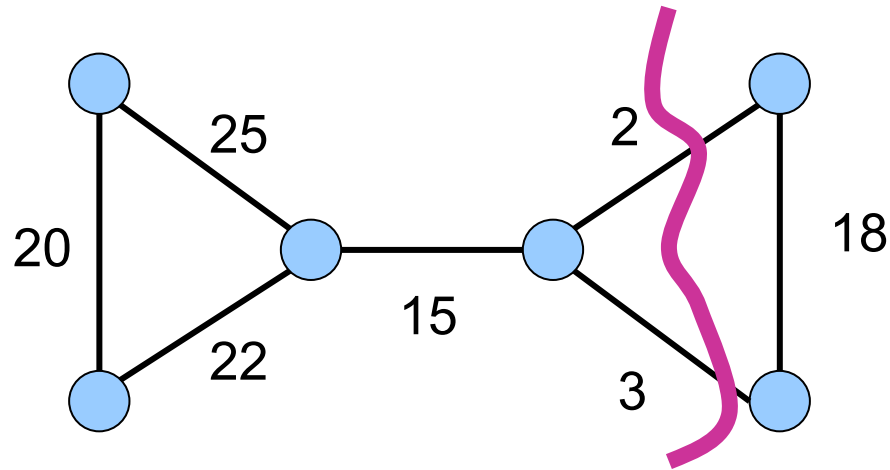
$G=(V, E)$ grafo non orientato, connesso

Vettore capacità $\mathbf{u} \in \mathcal{R}_+^{|E|}$

Determinare

Un insieme di vertici $\emptyset \subset S \subset V$, tale che $u(\delta(S))$ sia minimo

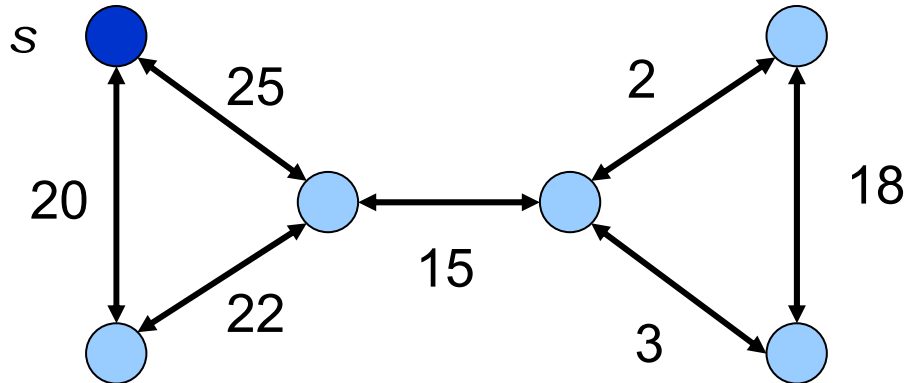
Esempio



Osservazione

Sappiamo risolvere il problema del taglio minimo utilizzando l'algoritmo di Ford e Fulkerson?

Esempio

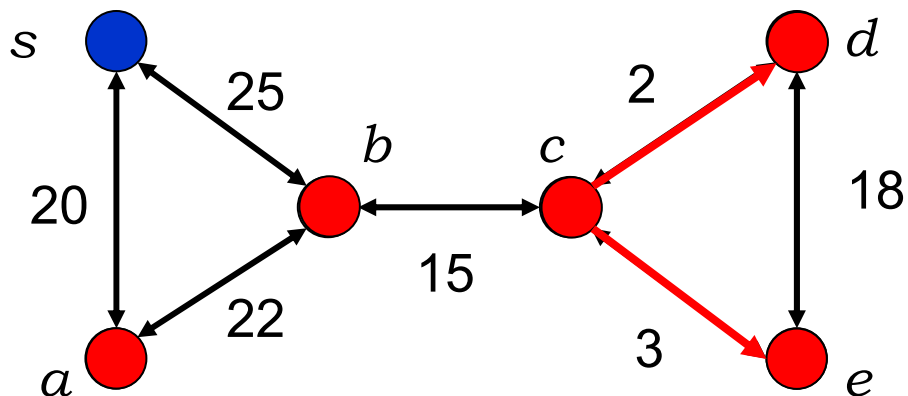


1. Sostituisco ogni arco non orientato con una coppia di archi orientati aventi la stessa capacità dell'arco originario
2. Scelgo un nodo di G , sia esso il nodo s .
3. Risolvo $(n-1)$ istanze del problema di massimo (s, v) -flusso, ove v è un nodo di $G \neq S$.
4. Il taglio minimo tra gli $n-1$ tagli individuati è il taglio minimo.

[**Complessità:** $O(nk)$, con k complessità di un algoritmo per il max-flow.

Se $k = nm^2$, la complessità è $O(n^2m^2)$]

Esempio



(s, a) : Flusso di valore 42

(s, b) : Flusso di valore 45

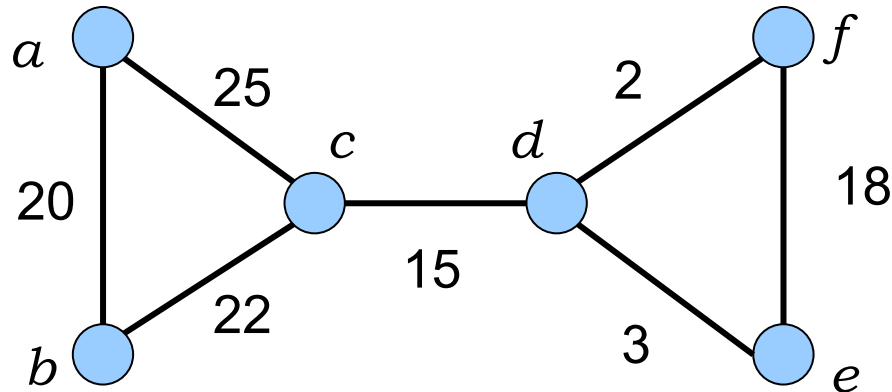
(s, c) : Flusso di valore 15

(s, d) : Flusso di valore 5

(s, e) : Flusso di valore 5

Taglio minimo in corrispondenza del
flusso (s, e) : $S = \{s, a, b, c\}$

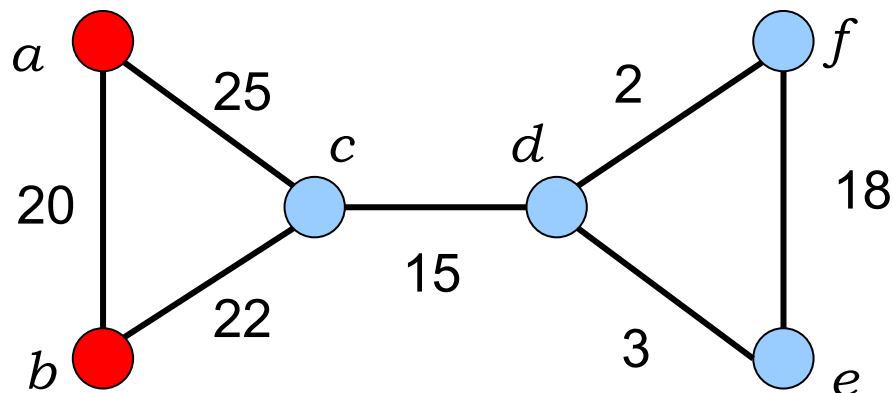
L'operazione di "node identification"



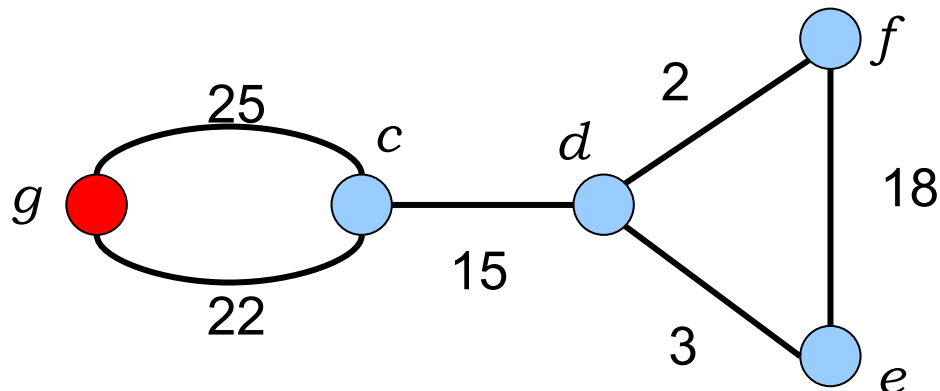
Se u e v sono due vertici distinti, G_{uv} è il grafo che si ottiene ponendo

1. $V(G_{uv}) = V \setminus \{u, v\} \cup \{x\}$
2. Un arco ij di G , rimane in G_{uv} se sia i che j sono distinti da u e da v . Se $j = u$ (ovvero $j = v$), l'arco iu (ovvero iv) diventa ix , se $i = u$ (ovvero $i = v$), l'arco uj (ovvero vj) diventa l'arco xj .

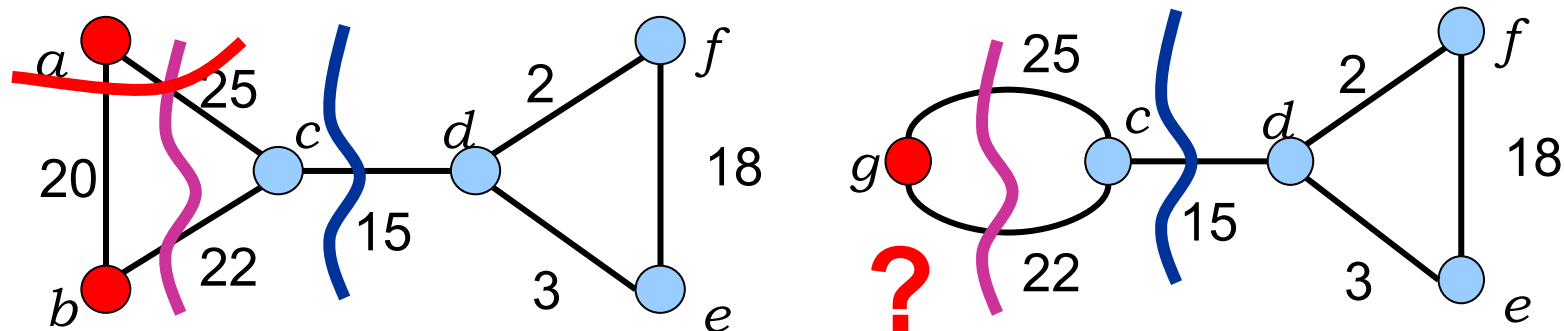
Esempio



Identifichiamo a e b , generando il nuovo vertice g :



Esempio



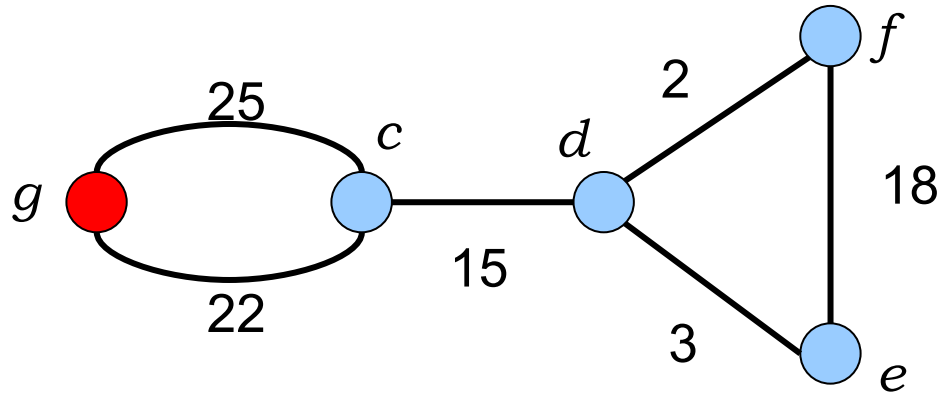
Osservazioni

1. G_{ab} non è un grafo semplice

2. Un taglio di G_{ab} è un taglio in G .

3. Un taglio in G che **NON** separa a e b , è un taglio di G_{ab}

Esempio

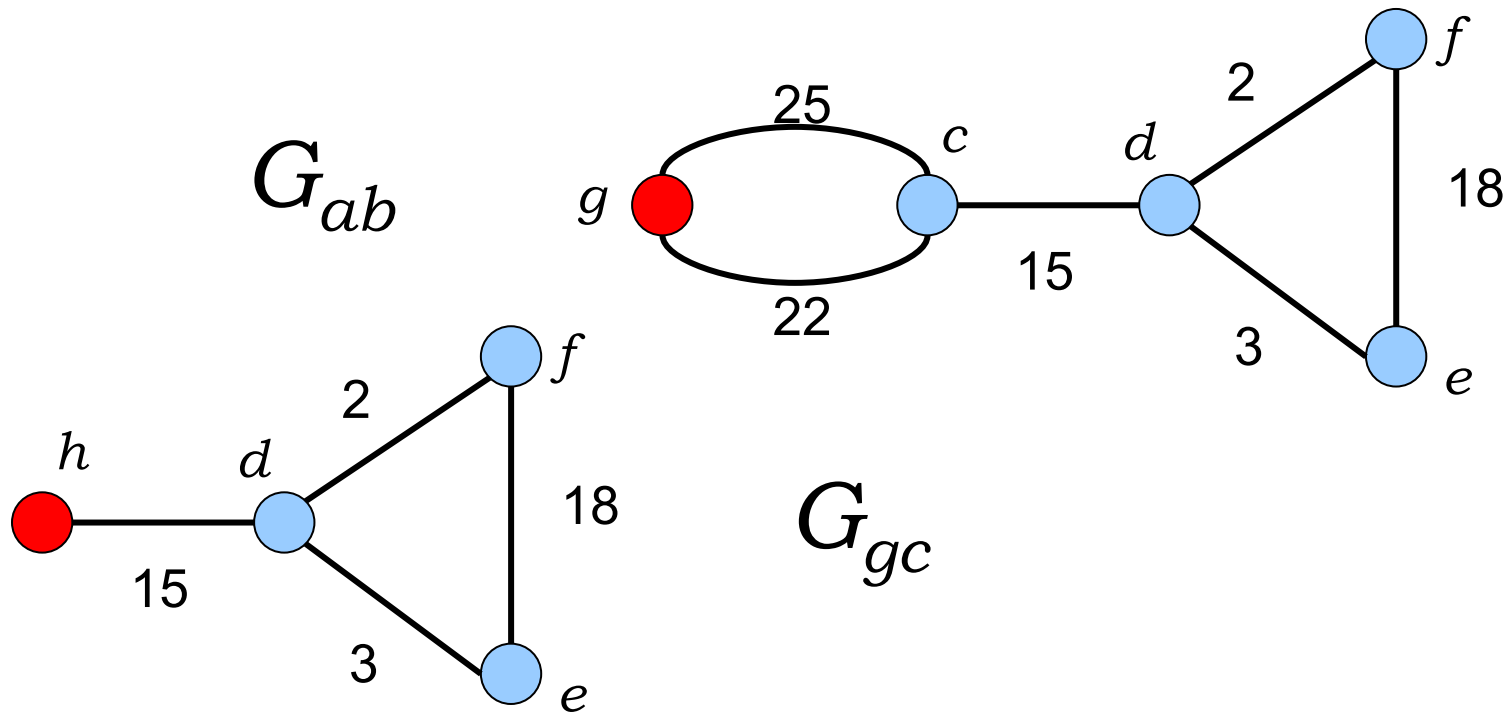


Conseguenza delle oss. 2 e 3:

Sia $\lambda(G)$ il taglio minimo in G , e $\lambda(G, a, b)$ il (a, b) -taglio minimo (ovvero, il taglio minimo che separa a e b).
Si ha:

$$\lambda(G) = \min \{ \lambda(G_{ab}), \lambda(G, a, b) \} = \min \{ 42, \lambda(G_{ab}) \}$$

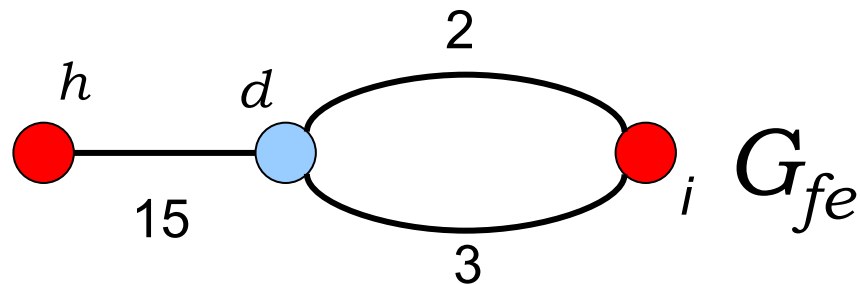
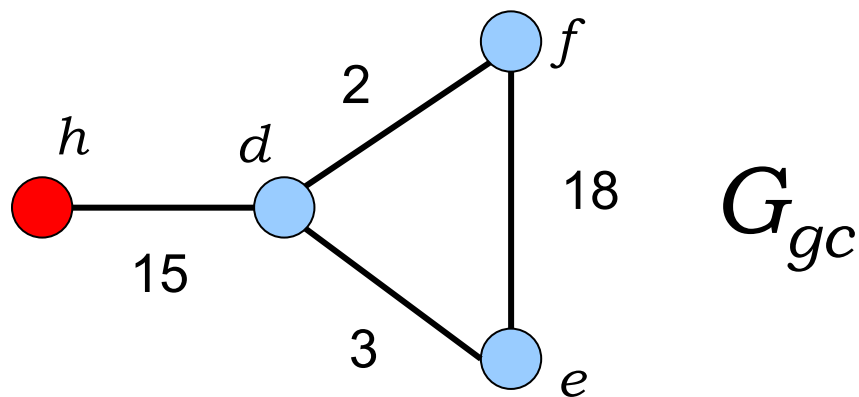
Esempio



Identificando g e c , si ottiene h e si ha:

$$\lambda(G) = \min \{42, \lambda(G_{ab})\} = \min \{42, \min \{ \lambda(G_{gc}), \lambda(G, g, c) \} \} = \min \{42, 47, \lambda(G_{gc})\}$$

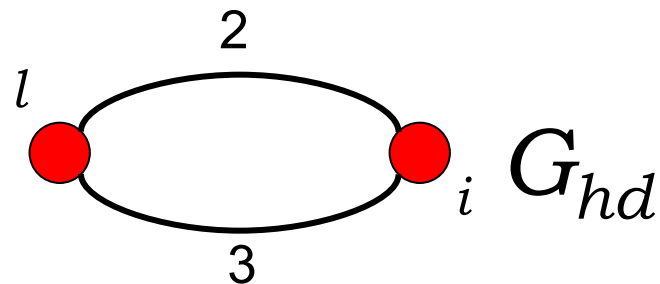
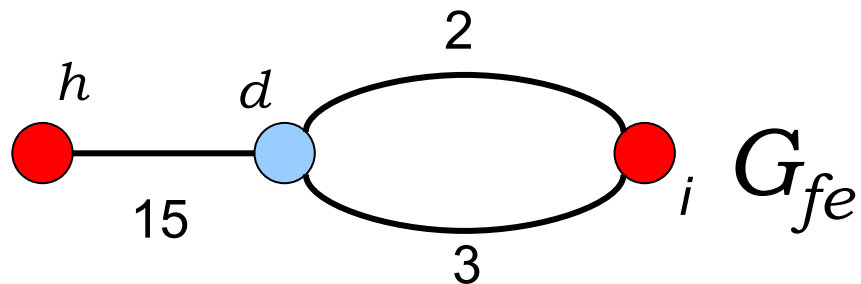
Esempio



Identificando f ed e , si ottiene i e si ha:

$$\lambda(G) = \min \{42, 47, \lambda(G_{gc})\} = \min \{42, 47, 20, \lambda(G_{fe})\}$$

Esempio



Infine, identificando h ed d , si ottiene l e si ha:

$$\lambda(G) = \min \{42, 47, 20, 15, \lambda(G_{hd})\} = \\ \min \{42, 47, 20, 15, 5\} = 5.$$

Esempio

Tenendo conto dei tagli “persi” durante le $n - 1$ operazioni di identificazione, abbiamo messo a punto un algoritmo che ha complessità $O(nk)$, con k complessità di un algoritmo per il max-flow.

Finora, l'unico vantaggio di questo approccio è quello di risolvere problemi di max-flow di dimensioni decrescenti.

Idea

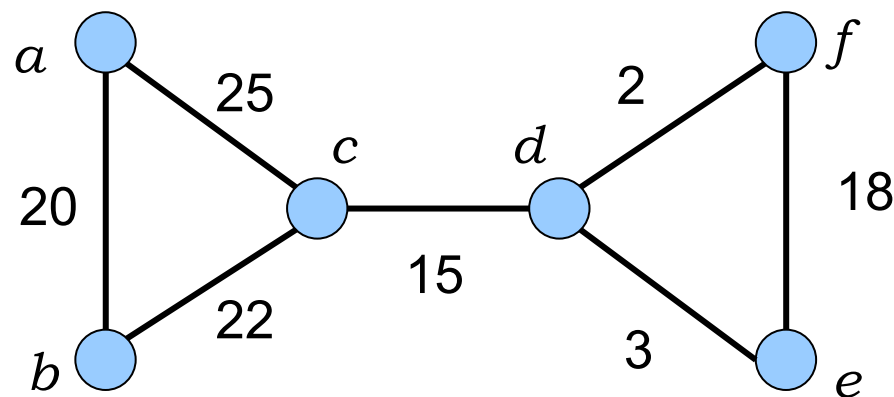
Scegliere i vertici da identificare in modo che sia facile individuare il minimo taglio che li separa.

Legal ordering

Sia v_1, v_2, \dots, v_n un ordinamento dei vertici di G
e sia $V_i = \{v_1, v_2, \dots, v_i\}$. Se

$$u(\delta(V_{i-1}) \cap \delta(v_i)) \geq u(\delta(V_{i-1}) \cap \delta(v_j)) \text{ per } 2 \leq i \leq j \leq n$$

si dice che v_1, v_2, \dots, v_n è un “legal ordering”.



Esempio:

a, c, b, d, e, f

Trovare un “legal ordering”

Inizializzazione

Associa un'etichetta $e_i = 0$, ad ogni $i \in V$.

Scegli un nodo u di G e poni $v_1 = u$, $V^{\text{ORD}} = \{v_1\}$, $k=1$

Passo k

Aggiorna le etichette dei nodi adiacenti a v_k ,
ponendo $e_i = e_i + u(iv_k)$, per ogni i adiacente a v_k .

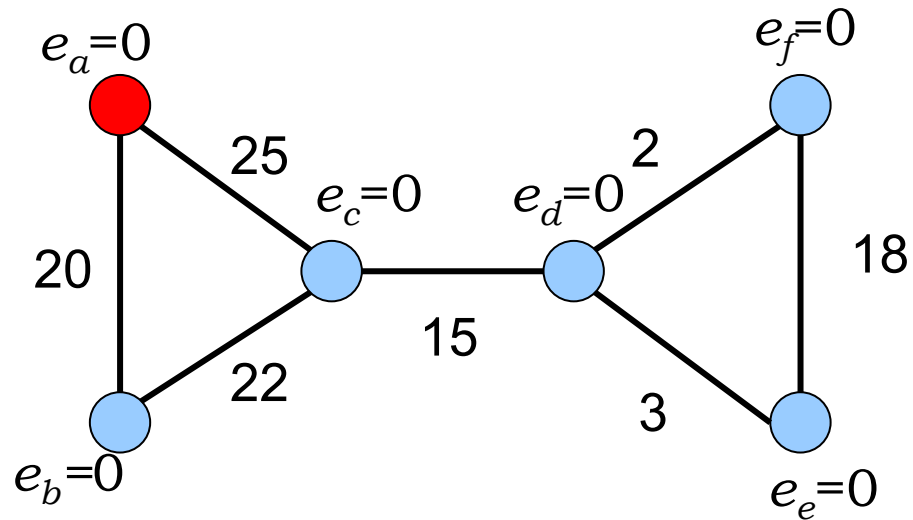
Seleziona il nodo avente etichetta massima tra i
nodi non appartenenti a V^{ORD} , sia esso il nodo v .

Poni $v_{k+1} = v$, $V^{\text{ORD}} = V^{\text{ORD}} \cup \{v_{k+1}\}$, $k = k + 1$.

Ripeti finché $k < n$

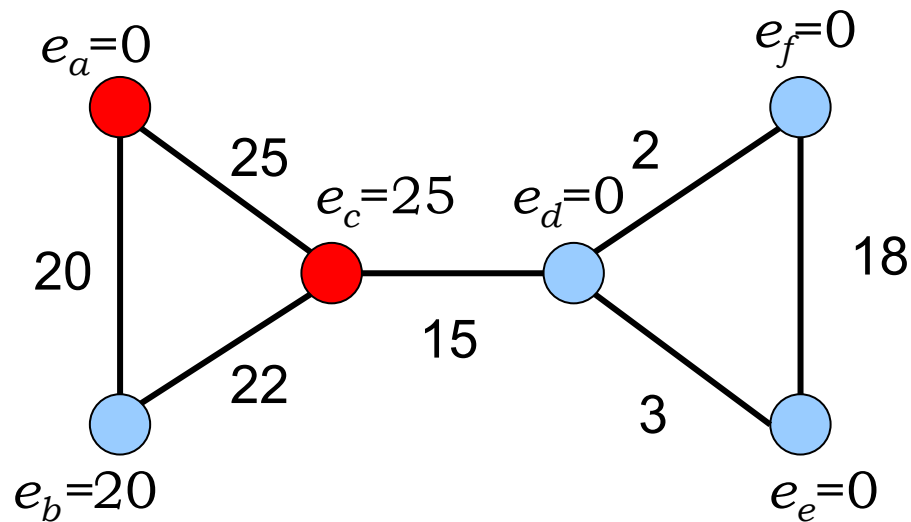
Esempio

$$V^{\text{ORD}} = \{a\}$$



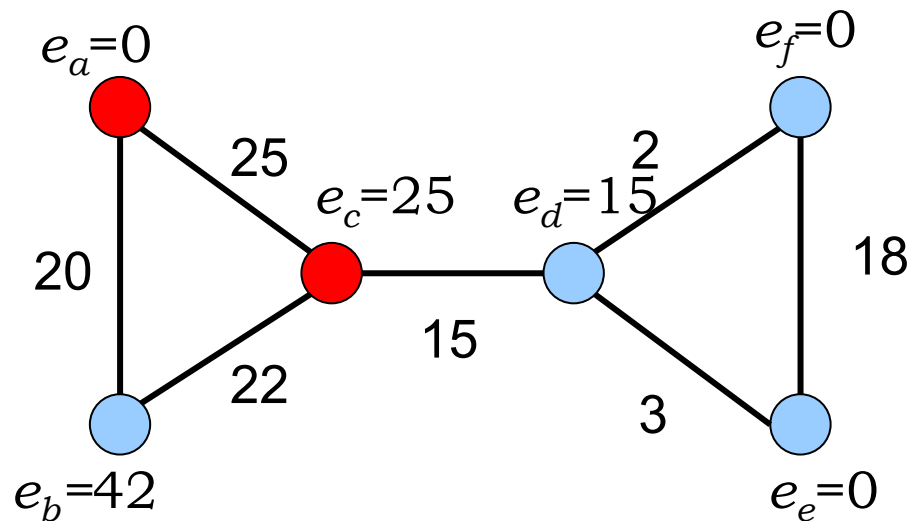
Esempio

$$V^{\text{ORD}} = \{a\}, v_1 = \{c\}$$



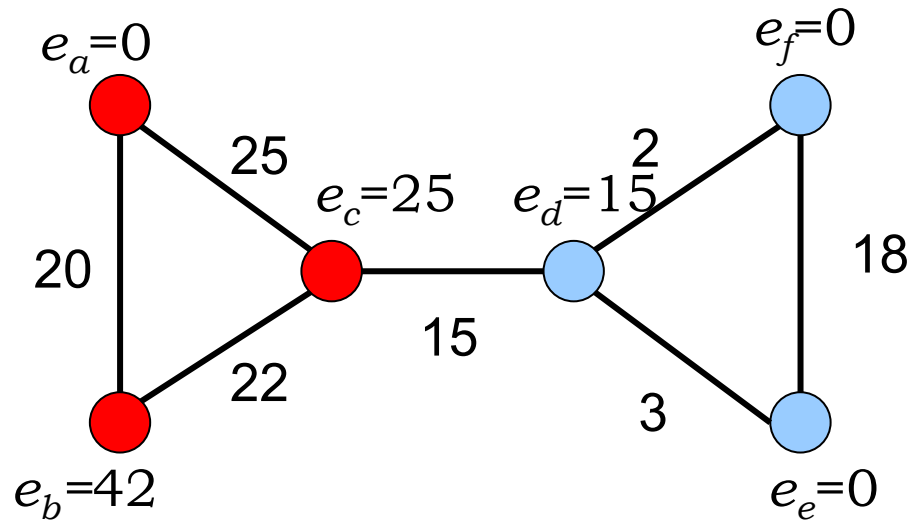
Esempio

$$V^{\text{ORD}} = \{a, c\}, v_2 = \{b\}$$



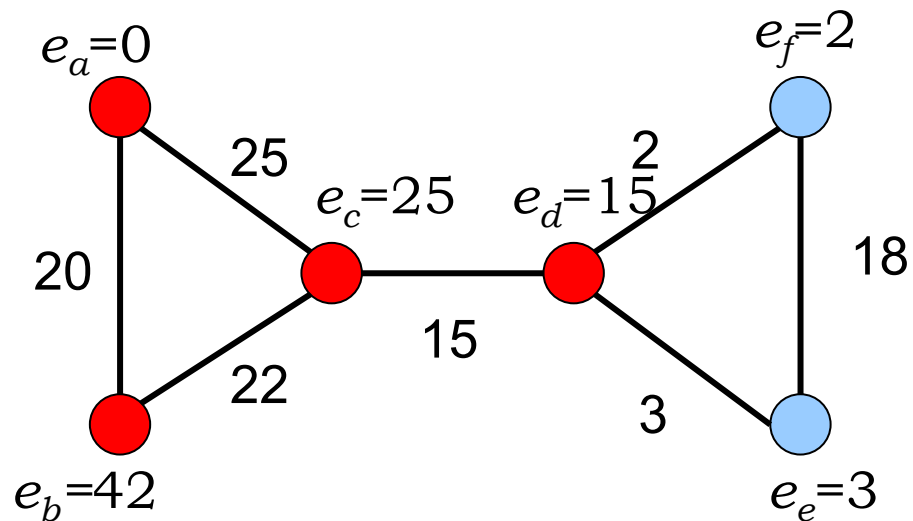
Esempio

$V^{\text{ORD}} = \{a, c, b\}$, $v_3 = \{d\}$



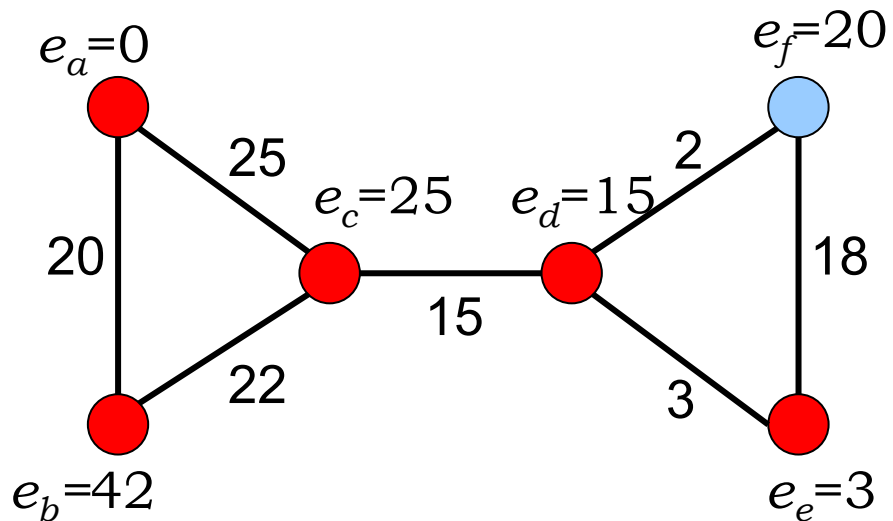
Esempio

$V^{\text{ORD}} = \{a, c, b, d\}$, $v_3 = \{e\}$



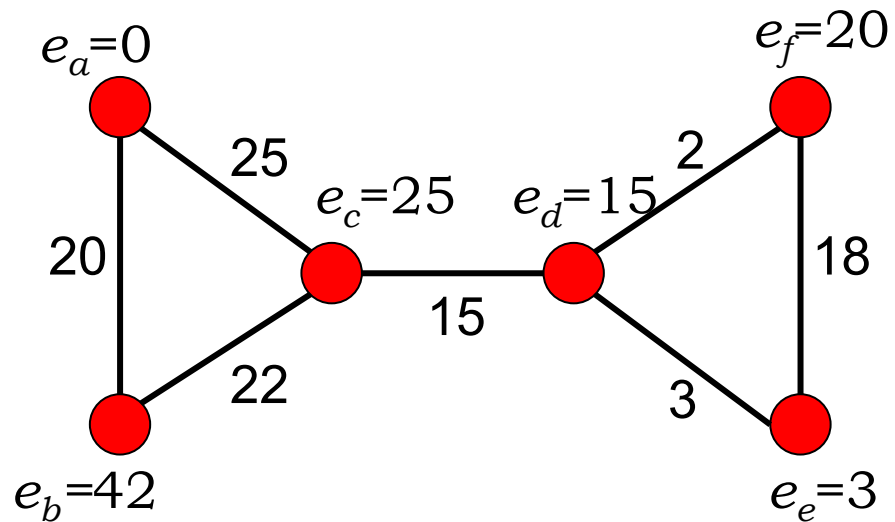
Esempio

$$V^{\text{ORD}} = \{a, c, b, d, e\}, v_3 = \{f\}$$



Esempio

$V^{\text{ORD}} = \{a, c, b, d, e, f\}$



Proprietà

1. L'algoritmo descritto trova un legal ordering in $O(n^2)$.
2. Se V^{ORD} è un legal ordering, allora $\delta(v_n)$ è il (v_{n-1}, v_n) -taglio minimo di G [da dimostrare]

$$\lambda(G) = \min \left\{ \lambda(G_{v_{n-1}v_n}), \lambda(G, v_{n-1}, v_n) \right\}$$

Ricordiamo che:

$$\lambda(G) = \min \left\{ \lambda(G_{v_{n-1}v_n}), \delta(v_n) \right\}$$

Quindi, il seguente algoritmo termina con il taglio minimo di G :

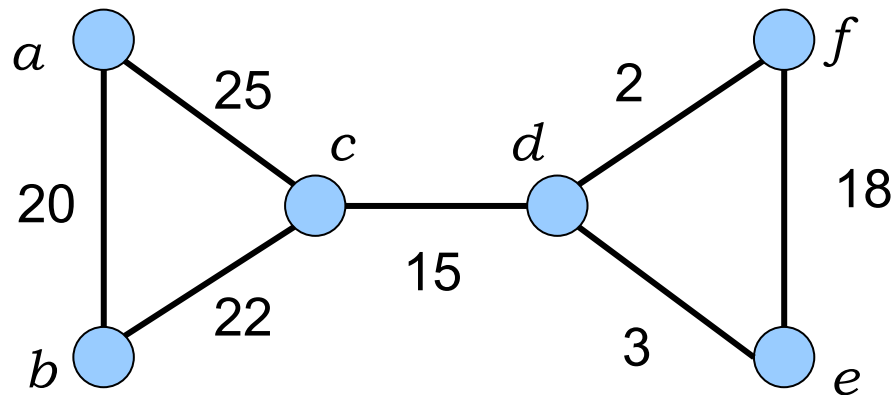
Algoritmo Min Cut

inizializzazione: $M = +\infty$, $A = \emptyset$

```
while  $G$  ha più di 2 nodi {  
  trova un legal ordering di  
   $G : \{v_1, v_2, \dots, v_n\}$   
  if  $u(\delta(v_n)) < M$  then  $M = u(\delta(v_n))$  e  $A = \delta(v_n)$ ;  
  identifica  $v_{n-1}$  e  $v_n$ ;  
  poni  $G = G_{v_{n-1}v_n}$  ;  
} endwhile;
```

Esempio (continua)

$V^{\text{ORD}} = \{a, c, b, d, e, f\} \Rightarrow u(\delta(f)) = 20$
 $M = 20; A = \{df, ef\}$

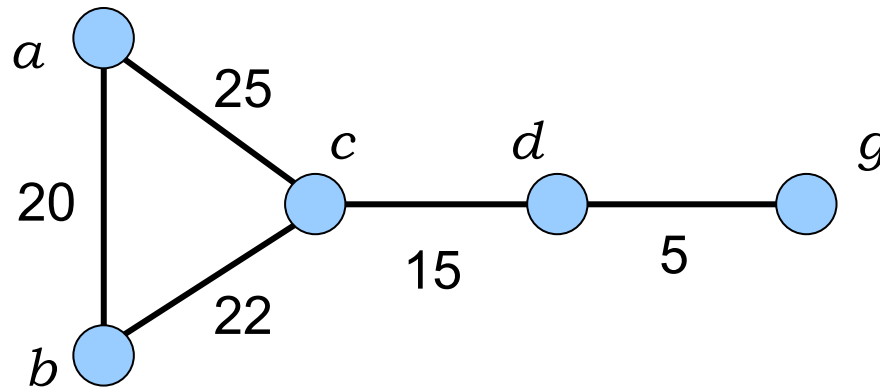


Esempio (continua)

Dopo aver identificato f ed e , ottengo G_{ef} che ammette il legal ordering:

$V^{\text{ORD}} = \{g, d, c, a, b\} \Rightarrow u(\delta(b)) = 44$, quindi non aggiorno M ed A .

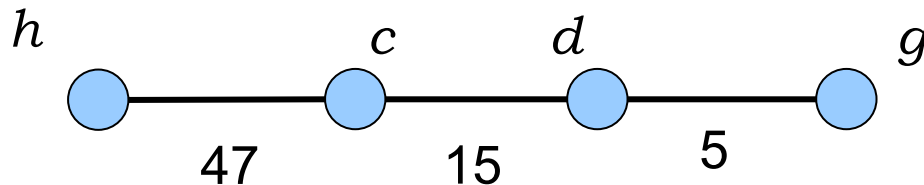
$M = 20$; $A = \{df, ef\}$



Esempio (continua)

Dopo aver identificato a e b , ottengo G_{ab} che ammette il legal ordering:

$V^{\text{ORD}} = \{h, c, d, g\} \Rightarrow u(\delta(g)) = 5$. Quindi,
 $M = 5$; $A = \{dg\} = \{df, de\}$



A questo punto, identifico d e g e ripeto il passo centrale dell'algoritmo ...

Osservazione

L'algoritmo Min-cut ha complessità $O(n^3)$.

Teorema

Teorema

Se V^{ORD} è un legal ordering, allora $\delta(v_n)$ è il (v_{n-1}, v_n) -taglio minimo di G .

Lemma

Se i, j, h sono nodi di V , allora

$$\lambda(G, i, j) \geq \min \{ \lambda(G, j, h), \lambda(G, i, h) \}$$

Dimostrazione Lemma

Consideriamo il minimo (i, j) -taglio $\delta(S)$ e supponiamo che $i \in S$. Se anche $h \in S$, allora $\delta(S)$ è anche un (j, h) -taglio e $u(\delta(S)) \geq \lambda(G, j, h)$. Altrimenti, $\delta(S)$ è un (i, h) -taglio e $u(\delta(S)) \geq \lambda(G, i, h)$ ■

Dimostrazione teorema

$\delta(v_n)$ è un (v_{n-1}, v_n) -taglio. Dobbiamo dimostrare che è minimo (ovvero che $u(\delta(v_n)) \leq \lambda(G, v_{n-1}, v_n)$).

Dimostrazione per induzione: se $n = 2$ il teorema è vero.

Supponiamo che esista lo spigolo $e = v_{n-1}v_n$ in G e sia $G' = G \setminus e$.

Il legal ordering v_1, v_2, \dots, v_n di G è anche un legal ordering di G' . Quindi:

$u(\delta(v_n)) = u(\delta'(v_n)) + u_e$ e, per l'ipotesi induttiva,

$\lambda(G', v_{n-1}, v_n) + u_e = \lambda(G, v_{n-1}, v_n)$.

Dimostrazione teorema

Invece, se v_{n-1} e v_n non sono adiacenti in G , considero il nodo v_{n-2} e dimostro che:

1. $u(\delta(v_n)) \leq \lambda(G, v_{n-2}, v_n)$
2. $u(\delta(v_n)) \leq \lambda(G, v_{n-2}, v_{n-1})$.

Poiché dal lemma precedente si ha che

$$\lambda(G, v_{n-1}, v_n) \geq \min \{ \lambda(G, v_{n-2}, v_n), \lambda(G, v_{n-2}, v_{n-1}) \} \\ \geq u(\delta(v_n)),$$

se i punti 1 e 2 sono veri il teorema è dimostrato.

Dimostrazione teorema

Caso 1

Consideriamo $G' = G \setminus v_{n-1}$

La sequenza $v_1, v_2, \dots, v_{n-2}, v_n$ è un legal ordering di G' .

Ora $u(\delta(v_n)) = u(\delta'(v_n))$ e per ipotesi induttiva

$$u(\delta'(v_n)) = \lambda(G', v_{n-2}, v_n) \leq \lambda(G, v_{n-2}, v_n),$$

ovvero $u(\delta(v_n)) \leq \lambda(G, v_{n-2}, v_n)$.

Caso 2

Consideriamo $G' = G \setminus v_n$

La sequenza v_1, v_2, \dots, v_{n-1} è un legal ordering di G' .

Per definizione di legal ordering $u(\delta(v_n)) \leq u(\delta(v_{n-1}))$, ma

$u(\delta(v_{n-1})) = u(\delta'(v_{n-1}))$ e per ipotesi induttiva

$$u(\delta'(v_{n-1})) = \lambda(G', v_{n-2}, v_{n-1}) \leq \lambda(G, v_{n-2}, v_{n-1}),$$

ovvero $u(\delta(v_n)) \leq \lambda(G, v_{n-2}, v_{n-1})$. ■

Un algoritmo probabilistico

```
while  $G$  ha più di 2 nodi {  
  scegli un arco  $ij$  di  $G$  con probabilità  
   $u_{ij}/u(E)$ ;  
   $G = G_{ij}$   
}
```

Il risultato dell'algoritmo è l'unico taglio di G .

Teorema

Sia A il taglio minimo di G . L'algoritmo di "random contraction" restituisce A con probabilità $2/n(n-1)$

Dimostrazione

Se nessun arco di A viene scelto durante l'esecuzione, allora l'algoritmo restituisce esattamente A .

Supponiamo di aver eseguito i passi dell'algoritmo, e di aver contratto i archi, nessuno dei quali appartiene ad A . Sia $G'=(V', E')$ il grafo corrente. Ovviamente, $|V'| = n-i$. Essendo A il minimo taglio di G , esso è anche il taglio minimo di G' .

Il valore del taglio minimo A è al più pari alla media della capacità dei tagli del tipo $\delta'(v)$, ovvero:

$$u(A) \leq \sum_{v \in V'} u(\delta'(v)) / (n-i) = 2u(E') / (n-i)$$

Dimostrazione

Pertanto, la probabilità p che un arco di A venga scelto al passo $i+1$ è:

$$\frac{u(A)}{u(E')} \leq \frac{2u(E')}{(n-i)u(E')} = \frac{2}{n-i}$$

La probabilità dell'evento complementare (ovvero, che **NESSUN** arco di A venga scelto al passo $i+1$) vale:

$$1 - \frac{2}{n-i} = \frac{(n-i-2)}{(n-i)}$$

Dimostrazione

Pertanto, la probabilità che durante l'esecuzione dell'algoritmo non venga scelto nessun arco di A vale:

$$\frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} = \frac{2}{n(n-1)}$$




Corollario

Sia A un taglio minimo di G e k un intero positivo. La probabilità che l'algoritmo di "random contraction" **NON** restituisca A in una di kn^2 esecuzioni è al più e^{-2k} .

Dimostrazione

$$\left(1 - \frac{2}{n(n-1)}\right)^{kn^2} \leq \left(1 - \frac{2}{n^2}\right)^{kn^2} \leq \left(e^{-\frac{2}{n^2}}\right)^{kn^2} = e^{-2k}$$

$$1 - x \leq e^{-x}$$


■