

Alessandro Zanin

Appunti di

Analisi e Progetto di Algoritmi (APA)

Prof. M. Pelillo



Sommario

I GRAFI	4
CAMMINO.....	5
ISOMORFISMO	8
RAPPRESENTAZIONE DI GRAFI	10
LEMMA DELLA STRETTA DI MANO	13
GRAFI AUTOCOMPLEMENTARI	15
GRAFI CONNESSI	17
ALBERI LIBERI	21
CLIQUE.....	24
MINIMUM SPANNING TREE (MST).....	26
CLUSTERING.....	26
COLLEZIONE DI INSIEMI DISGIUNTI	29
ALGORITMO DI KRUSKAL	29
ALGORITMO DI PRIM	30
ALGORITMI GREEDY (GOLOSI o MIOPI)	33
CAMMINI MINIMI	35
ALGORITMO DI DIJKSTRA.....	38
TEOREMA DI CORRETTEZZA DI DIJKSTRA	41
ALGORITMO DI BELLMAN-FORD.....	42
CAMMINI MINIMI TRA TUTTE LE COPPIE DI VERTICI.....	44
MOLTIPLICAZIONI TRA MATRICI	44
METODO DELLA QUADRATURA RIPETUTA	47
ALGORITMO DI FLOYD-WARSHALL	48
RETI DI FLUSSO	50
SOMMATORIA IMPLICITA	53
METODO DI FORD-FULKERSON.....	54
ALGORITMO DI FORD-FULKERSON.....	59
ABBINAMENTO MASSIMO	65
NP-COMPLETEZZA	69
RIDUCIBILITA' POLINOMIALE.....	71
TEOREMA DI COOK	73
SAT	74
3_SAT_FNC.....	75
COSA FARE DAVANTI A UN PROBLEMA NPCompleto	79
INDICE ANALITICO	80



Note dell'autore:

La presente dispensa non è altro che il frutto della comprensione a lezione, integrata con materiale di anni precedenti e parti del libro di testo.

Con buona probabilità sarà piena di errori di battitura, di trascrizione, di comprensione o di concetto, quindi assolutamente non sostituisce la presenza a lezione, la propria testa o il libro di testo del corso.

L'unico scopo di questa dispensa è facilitare il ripasso in preparazione all'esame.

Alessandro Zanin

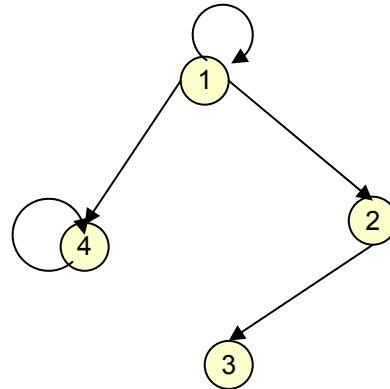


I GRAFI

Definizione di **GRAFO ORIENTATO** (DIRETTO): è una coppia $G=(V,E)$ dove V =insieme dei vertici ed $E \subseteq V \times V$ che indica una relazione binaria tra vertici

$$G=(\underbrace{\{1,2,3,4\}}_V, \underbrace{\{(1,2), (1,4), (1,1), (2,3), (4,4)\}}_E)$$

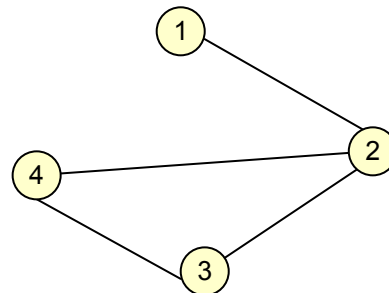
cappio →



- Definizione di **GRAFO NON ORIENTATO**: è una coppia $G=(V,E)$ dove V rappresenta i vertici ed $E = V \times V$ rappresenta una relazione simmetrica.

Dati $u,v \in V : (u,v) \in E \Leftrightarrow (v,u) \in E$ quindi $\{u,v\} \in E$

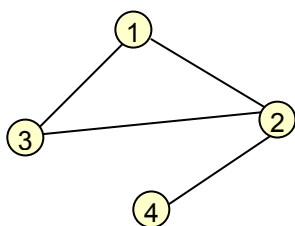
$$G=(\underbrace{\{1,2,3,4\}}_V, \underbrace{\{(1,2), (2,3), (4,3), (2,4)\}}_E)$$



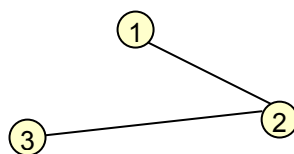
- Definizione di **SOTTOGRAFO**: dato $G=(V,E)$, un sottografo di G (G') è un grafo costruito su vertici ed archi di G

$$G'=(V',E') \quad \begin{array}{l} 1) V' \subseteq V \\ 2) E' \subseteq E \end{array}$$

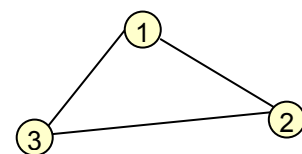
- Definizione di **SOTTOGRAFO INDOTTO $G[V']$** : dato un grafo $G=(V,E)$ e $V' \subseteq V$, il sottografo indotto da V' è definito come segue: è a sua volta un grafo $G'=(V',E')$ in cui $E'=E \cap V' \times V'$, cioè tutti gli archi che si trovano in E tali che gli estremi sono vertici di V' .
Ovvero $E'=[(u,v) \in E \mid u,v \in V']$



GRAFO



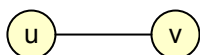
SOTTOGRAFO



SOTTOGRAFO INDOTTO
 $V'=\{1,2,3\}$



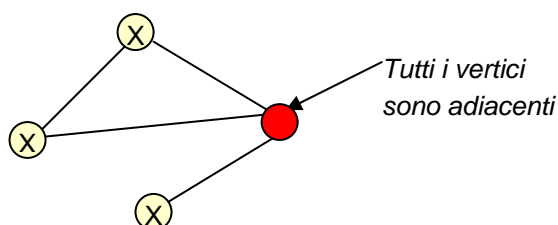
- Dati $(u,v) \in E$



1. u e v sono ADIACENTI
2. l'arco (u,v) è INCIDENTE su u e v

- Dato un $G=(V,E)$ non orientato se $u \in V$ l' "intorno" di u si definisce

$$N(u) = \{ v \in V \mid (u,v) \in E \}$$



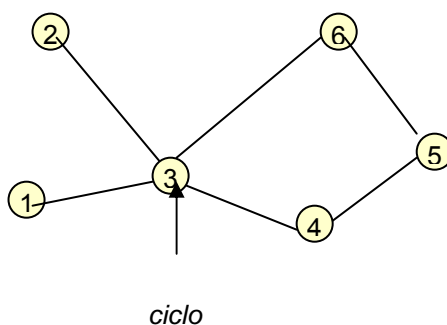
Tecnicamente rappresenta il GRADO o CARDINALITA' del vertice: $\deg(u) = |N(u)|$

- Dato un $G=(V,E)$ orientato, il grado si divide in grado entrante e grado uscente:
 - o $\text{In_degree}(u) = |\{(v,u) \in E \mid v \in V\}|$
 - o $\text{Out_degree}(u) = |\{(u,v) \in E \mid v \in V\}|$
- Un vertice si dice **ISOLATO** se ha grado 0 $\rightarrow \deg(u) = 0$
- Un vertice si dice **TERMINALE** se ha grado 1 $\rightarrow \deg(u) = 1$

CAMMINO

- Dati due vertici di un grafo $G=(V,e)$, un **CAMMINO** tra u e $v \in V$ si definisce come una sequenza di vertici $p = \langle x_0 x_1 x_2 \dots x_k \rangle$ in cui:
 1. $x_0 = u$
 2. $x_k = v$
 3. $\forall i=1 \dots k : (x_{i-1}, x_i) \in E$, ovvero deve esistere un arco tra ogni vertice toccato dal cammino
- v è **RAGGIUNGIBILE** da u se esiste un cammino che parte da u e arriva a v

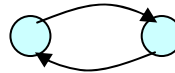
$$p = \langle 1 \underline{3} 4 5 6 \underline{3} 2 \rangle$$



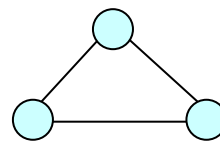
- Un cammino si dice **SEMPLICE** se non presenta vertici ripetuti (non contiene cicli)
- Dato un cammino $p = \langle x_0 \dots x_k \rangle$ un **SOTTOCAMMINO** di p (p') si ottiene come segue:
presi $0 \leq i < j \leq k$
 $p' = \langle x_i \ x_{i+1} \dots x_j \rangle$ che sarà una porzione del cammino p

- Un cammino che parte da un vertice e arriva allo stesso vertice forma un **CICLO**
 $p = \langle x_0 \ x_1 \dots x_0 \rangle$

Nei grafi orientati bastano due vertici
per formare un ciclo.



Nei grafi non orientati invece
ne servono almeno tre.



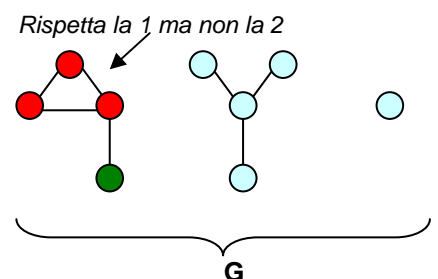
- Definizione di aciclicità: un grafo $G=(V,E)$ si dice **ACICLICO** se non contiene cicli.

• GRAFO CONNESSO

Un grafo $G=(V,E)$ si dice connesso se $\forall u,v \in V \exists p$ tra u e v , ovvero se presi due vertici qualsiasi del grafo esiste un cammino che li unisce.

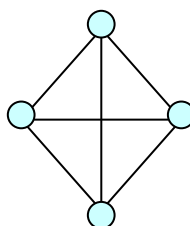
Una **COMPONENTE CONNESSA** di un grafo $G=(V,E)$ è un sottoinsieme di vertici $V' \subseteq V$ per il quale valgono queste proprietà:

1. il sottografo indotto da V' $G[V']$ è connesso
2. $\nexists V''$ tale che:
 - a. $G[V'']$ sia connesso
 - b. V' sia un sottoinsieme stretto di V'' ($V' \subset V''$)



Quindi un grafo è connesso se e solo se ha una SOLA componente connessa.

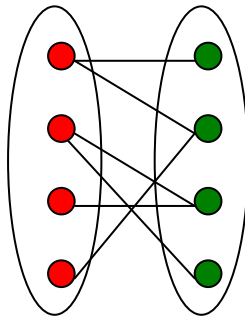
- Dato un grafo $G=(V,E)$ n.o si dice **COMPLETO** se $\forall u,v \in V (u \neq v)$ risulta che $(u,v) \in E$, ovvero esistono tutti gli archi possibili e ogni vertice è raggiungibile in un solo passo e il numero di archi sarà $|E| = [n(n-1)]/2$



- Un grafo **VUOTO** E_n non ha archi, $E=0$, a differenza del grafo completo che li ha tutti, $E=V \times V$



- **GRAFO BIPARTITO:** è un grafo in cui, se diviso in due, ogni arco passa da un sottoinsieme all'altro.



L R

$$\exists V1, V2 \subseteq V \text{ e'}$$

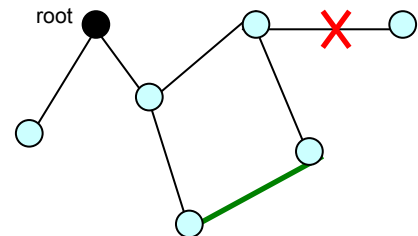
$$1. V1 \cap V2 = \emptyset$$

$$2. V1 \cup V2 = V$$

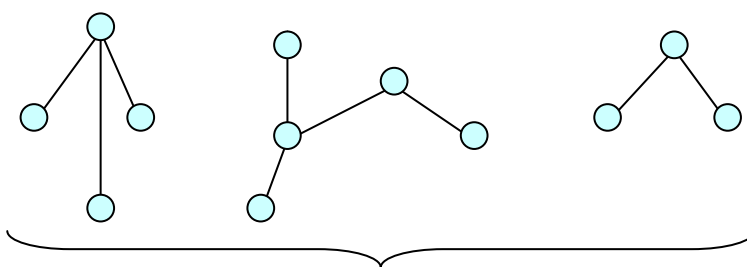
$$3. \forall (u,v) \in E: (u \in V1 \wedge v \in V2) \vee (u \in V2 \wedge v \in V1)$$

- **ALBERI LIBERI:** un grafo $G=(V,E)$ n.o si dice albero libero se G è:

- **Connesso**, ma se tolgo un arco diventa disconnesso
- **Aciclico**, ma se aggiungo un arco diventa ciclico
 - Se scelgo un vertice come origine diventa un albero radicato



- Una **FORESTA** è un grafo aciclico



Foresta

Questa foresta è formata da tre componenti connesse che sono a loro volta **alberi** ed è aciclica.

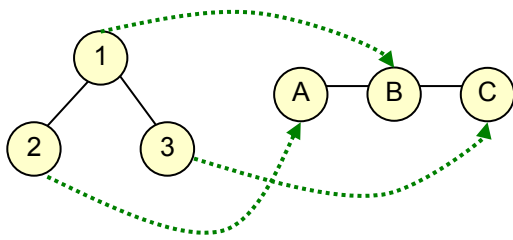
Dato $G=(V,E)$ n.o. si dice **foresta** se G è aciclico. Se si prendono le componenti connesse di una foresta ognuna di esse è un albero, in quanto aciclica e connessa.

Se G è una foresta con k componenti connesse allora il numero di archi di G sarà uguale al numero di vertici meno k :

$$|E| = |V| - k$$



ISOMORFISMO



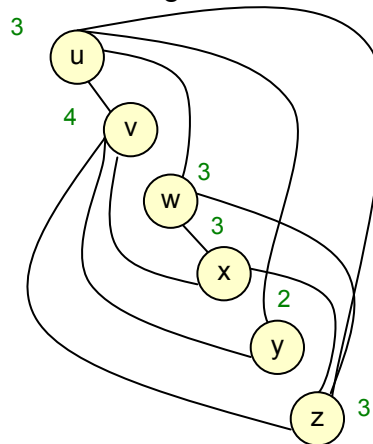
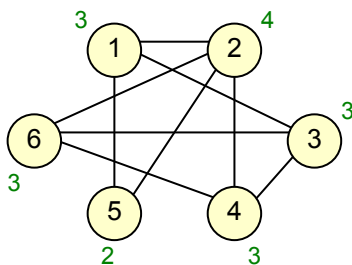
I due grafi sono solo disegnati ed etichettati in modo diverso, ma sono uguali in quanto sono preservate le relazioni tra nodi e archi.

Dati due grafi $G_1=(V_1,E_1)$ e $G_2=(V_2,E_2)$, si dicono **ISOMORFI** se esiste una funzione Φ detta isomorfismo, che manda i vertici del primo grafo nel secondo, rispettando queste proprietà:

$$G_1 \cong G_2 \text{ se } \exists \Phi : V_1 \rightarrow V_2$$

1. Φ è biunivoca
2. $(u,v) \in E_1 \Leftrightarrow (\Phi(u), \Phi(v)) \in E_2$: ovvero devono essere preservate le relazioni di adiacenza

Ci possono essere diversi isomorfismi equivalenti, infatti ad esempio nella figura precedente il vertice 2 poteva essere trasposto allo stesso modo sia in A che in C, rispettando le condizioni necessarie all'isomorfismo di due grafi.



I numeri in verde vicino ai vertici rappresentano i gradi dei vertici stessi, quindi $\deg(u)$

La degree sequence indica la sequenza dei valori dei singoli gradi di tutti i vertici del grafo

4 3 3 3 3 2

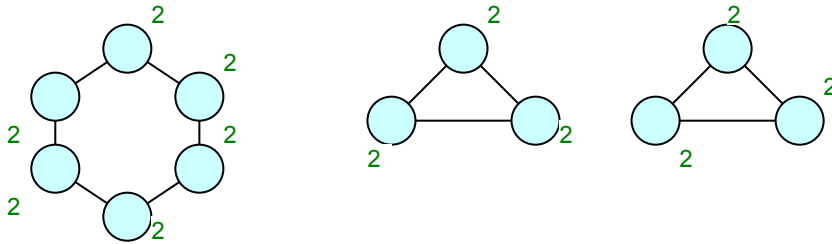
4 3 3 3 3 2

Se $u \xrightarrow{\Phi} v$ allora $\deg(u) = \deg(\Phi(u))$

Non è immediato in questo caso verificare l'isomorfismo dei due grafi, però dalla sequenza dei gradi si vede che solo due vertici hanno $\deg(u) \neq 3$ quindi $2 \rightarrow v$ e $5 \rightarrow y$



- Condizioni necessarie ma non sufficienti perché due grafi siano isomorfi. Se $G_1 \cong G_2$ allora:
 1. $|V_1| = |V_2|$
 2. $|E_1| = |E_2|$
 3. $\deg_{\text{seq}}(G_1) = \deg_{\text{seq}}(G_2)$
 4. il numero di componenti connesse di G_1 è uguale a quello di G_2
 ce ne sono molte altre, ma sono tutte necessarie, non sufficienti.



Viene violato solo l'ultima delle condizioni, infatti G_2 a più componenti connesse rispetto a G_1 .



RAPPRESENTAZIONE DI GRAFI

- Matrici di adiacenza**

Dato un grafo $G=(V,E)$ con cardinalità dei vertici $|V|=n$, la matrice di adiacenza che lo rappresenta sarà una matrice quadrata di dimensione $n \times n$.

L'elemento ij sarà 1 se tra i e j c'è un arco, 0 altrimenti.

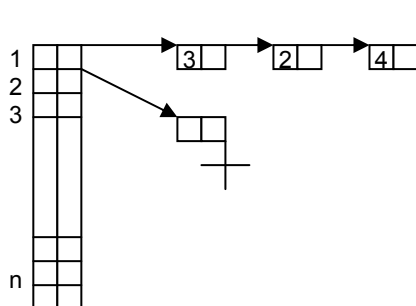
$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

Se il grafo è non orientato la matrice sarà simmetrica: $A = \bar{A}$

Se il grafo è non orientato, $i \in V$

$$\deg(i) = \sum_{j=1}^n a_{ij} = \sum_{j=1}^n a_{ji}$$

- Liste di adiacenza**



$$\delta(G) = \frac{|E|}{|V|^2} \quad \text{Grafo orientato}$$

≈ 1

$$\delta(G) = \frac{|E|}{\binom{|V|}{2}} \quad \text{Grafo non orientato}$$

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

- Un grafo è **DENSO** se $|E| \sim |V|^2$ e si usano matrici di adiacenza
- Un grafo è **SPARSO** se $|E| \sim |V|$ e si usano liste di adiacenza
- Proposizione:** se A è la matrice di adiacenza di un grafo (orientato o non orientato) e $A^2 = A \times A$, allora comunque si prenda una riga e una colonna ($\forall i, j = 1 \dots n$):
 $a^{(2)}_{ij}$ = numero cammini di lunghezza 2 tra i e j
 $a^{(2)}_{ii}$ = numero cicli di lunghezza 2 che partono da i e tornano ad i ma rappresenta anche il grado dei vertici



$i = j$ grafo n.o

La matrice è simmetrica

$$a_{ij}^{(2)} = \sum_{k=1}^n a_{ik} * a_{ki} = \sum_{k=1}^n a_{ik}^2 = \sum_{k=1}^n a_{ik} = \deg(i)$$

$A \times A \times \dots \times A = A^k = a_{ij}^{(k)}$ Per analogia dovrebbe dare il numero di cammini di lunghezza k, ma è necessaria una dimostrazione formale.

- **Proposizione:** se A è la matrice di adiacenza di G e $A^k = A \times A \times \dots \times A = a_{ij}^{(k)}$ allora $\forall i, j=1 \dots n, a_{ij}^{(k)} = \text{numero di cammini di lunghezza k tra i e j}$

Dimostrazione:

caso base)

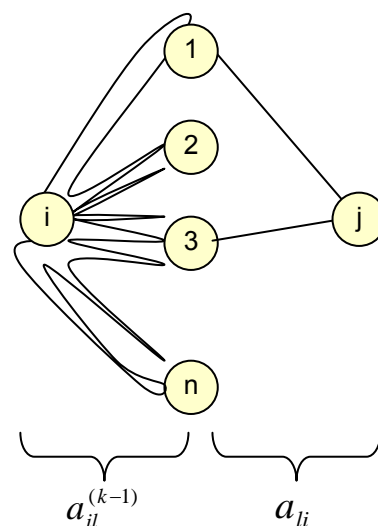
$k=2$, l'abbiamo dimostrato prima ed è vero

passo induttivo)

ip: vero per $k-1$

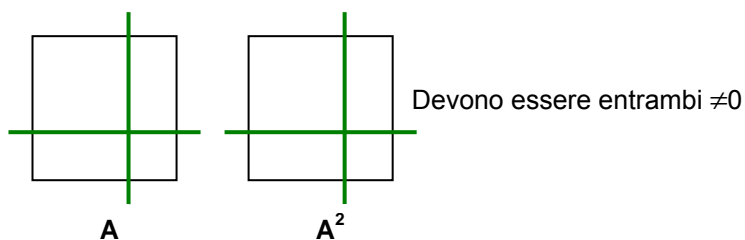
tesi: vero anche per k

$$A^{k-1} \times A = A^k \Rightarrow a_{ij}^{(k)} = \sum_{l=1}^n a_{il}^{(k-1)} * a_{lj}$$



- Dato $G=(V,E)$ n.o dimostrare che G contiene un "triangolo" (K_3), ovvero un grafo completo di cardinalità 3, se e solo se $\exists i, j \in V$ tali che A e A^2 hanno un elemento non nullo in posizione (i, j)

$$[a_{ij} \neq 0 \wedge a_{ij}^{(2)} \neq 0]$$



1. \Rightarrow ipotesi: G contiene K_3

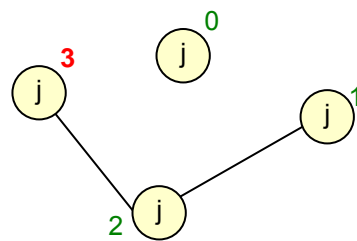
tesi: $\exists i, j \ni a_{ij} \neq 0 \wedge a_{ij}^{(2)} \neq 0$

2. \Leftarrow ipotesi: $\exists i, j \ni a_{ij} \neq 0 \wedge a_{ij}^{(2)} \neq 0$

tesi: G contiene K_3



- **Problema:** provare a costruire un grafo n. in cui tutti i gradi sono distinti



$\deg(u) = 0 \dots n-1$

NON E' POSSIBILE

- **Proposizione:** non è possibile costruire un grafo n.o. avente tutti i gradi distinti
dim. per assurdo

$G=(V,E)$ n.o. con tutti i gradi dei vertici distinti $\deg(u)=0 \dots n-1$

$\deg(u)=0$ u non è adiacente a nessun altro vertice, $(u,v) \notin E$

$\deg(v)=n-1$ v è adiacente a tutti i vertici, $(u,v) \in E$

GRAFI ORIENTATI

$G=(V,E)$ orientato $m = |E|$ numero di archi del grafo

$$m = |E| = \sum_{u \in V} \text{out_deg}(u) = \sum_{u \in V} \text{in_deg}(u)$$

Usando una matrice di adiacenza basta contare il numero di "1" che compare nella matrice

$$\begin{aligned} m = |E| &= \sum_{i=1}^n \sum_{j=1}^n a_{ij} = \sum_{i=1}^n \text{out_deg}(i) \\ &= \sum_{j=1}^n \sum_{i=1}^n a_{ij} = \sum_{j=1}^n \text{in_deg}(j) \end{aligned}$$

GRAFI NON ORIENTATI

$G=(V,E)$ n.o, si procede come prima, ma in questo caso gli archi non hanno un verso, quindi verrebbero contati 2 volte:

$$2m = \sum_{u \in V} \deg(u)$$



LEMMA DELLA STRETTA DI MANO

Se $G=(V,E)$ n.o. con m archi, $m=|E|$ e $n=|V|$, allora

$$2m = \sum_{u \in V} \deg(u)$$

Esaminando la matrice di adiacenza A , si vede che è simmetrica, essendo un grafo non orientato

$$m = |E| = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n a_{ij} = \frac{1}{2} \sum_{u \in V} \deg(u)$$

$\underbrace{\hspace{10em}}_{\deg(i)}$

• **Proposizione:** se $G=(V,E)$ n.o. allora il numero di vertici di grado dispari è pari
dimostrazione:

P = insieme dei vertici di grado pari

$$P \cap D = \emptyset \quad P \cup D = V$$

$$P = \{u \in V \mid \deg(u) \text{ è pari}\}$$

D = insieme dei vertici di grado dispari

$$D = \{u \in V \mid \deg(u) \text{ è dispari}\}$$

$$2m = \sum_{u \in V} \deg(u) = \sum_{u \in P} \deg(u) + \sum_{u \in D} \deg(u)$$

$$2m = \sum_{u \in P} 2h(u) + \sum_{u \in D} [2h(u) + 1]$$

Rappresento i gradi come numeri pari $2h$ e dispari $2h+1$

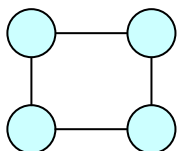
$$= 2 \sum_{u \in P} h(u) + 2 \sum_{u \in D} h(u) + \sum_{u \in D} 1$$

$\nwarrow |D|$

$$|D| = 2 \left[m - \sum_{u \in P} h(u) - \sum_{u \in D} h(u) \right]$$

Quindi il numero di vertici di grado dispari è **pari** essendo un multiplo di 2.

Vale il contrario? Cioè, il numero di vertici di grado pari è dispari? No, come si verifica facilmente con questo contro esempio:

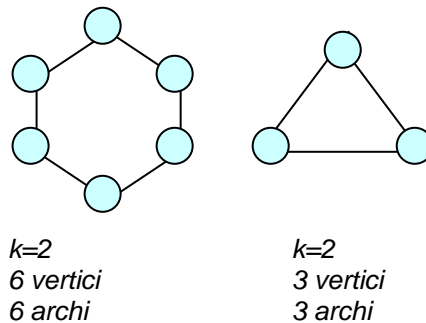


4 vertici di grado pari, quindi l'affermazione è falsa.



- Un grafo $G=(V,E)$ si dice **k-regolare** (k è un intero ≥ 0) se:

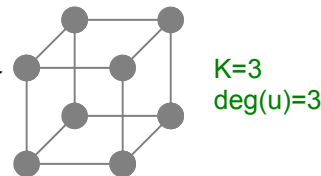
- $u \in V: \deg(u)=k$
- $2m=k_n$



Osservazioni:

- G è 2-regolare, allora $m=n$ ($2m=2n$)
- G è 3-regolare, è un **grafo cubico** \rightarrow

$$2m = 3n = 2n+n \Rightarrow n = 2m-2n = 2(m-n)$$
 Quindi n è **pari**
- G è 4-regolare: $2m = 4n \Rightarrow m = 2n$: il numero di archi è pari ed è il doppio dei vertici



Se k è dispari per il lemma della stretta di mano m è pari

- G è 6-regolare: $2m = 6n \Rightarrow m = 3n$: il numero di archi è tre volte il numero di vertici e:
 se n è pari m sarà pari
 se n è dispari m sarà dispari

- Proposizione:** dato un grafo $G=(V,E)$ n.o senza vertici isolati ($\deg(u) \geq 1$) e $m=n-1$ ($m=|E|$, $n=|V|$), allora esistono almeno 2 vertici terminali ($\deg(u)=1$)

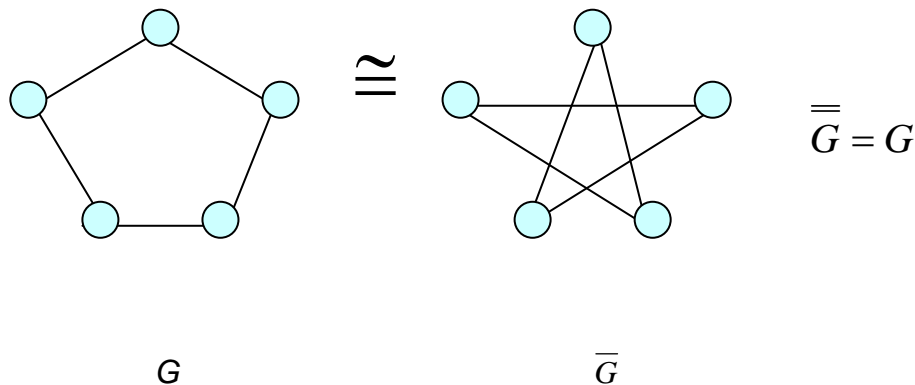
$$V_1 = \{u \in V \mid \deg(u)=1\} \rightarrow |V_1| \geq 2 ?$$

$$\begin{aligned}
 1. \quad \forall u \in V : \deg(u) \geq 1 & \rightarrow V \setminus V_1 = \{u \in V \mid \deg(u) \geq 2\} \\
 2. \quad m = n-1 & \rightarrow 2n-2 = \sum_{u \in V} \deg(u) \\
 3. \quad |V_1| \geq 2 \quad \text{tesi} & \quad \left| \begin{aligned} &= \sum_{u \in V_1} \deg(u) + \sum_{u \in V \setminus V_1} \deg(u) \\ &= |V_1| + \sum_{u \in V \setminus V_1} \deg(u) \quad \nearrow \geq 2 \\ &\geq |V_1| + \sum_{u \in V \setminus V_1} 2 \\ &\geq |V_1| + 2|V \setminus V_1| = |V_1| + 2|V| - 2|V_1| = 2n - |V_1| \\ &2n-2 \geq 2n - |V_1| \\ &|V_1| \geq 2 \end{aligned} \right.
 \end{aligned}$$



GRAFI AUTOCOMPLEMENTARI

- **Definizione 1:** dato $G=(V,E)$ n.o., il grafo complemento \overline{G} o $G^c=(V^c,E^c)$, dove:
 1. $V^c=V$
 2. $(u,v) \in E^c \Leftrightarrow (u,v) \notin E$, cioè nel grafo complemento c'è l'arco se e solo se nel grafo di origine l'arco non compare



- **Definizione 2:** un grafo $G=(V,E)$ n.o. si dice **AUTOCOMPLEMENTARE** se il grafo in questione è ISOMORFO al suo complemento ($G \cong \overline{G}$)
 - Il grafo precedente ad esempio è autocomplementare.
- **Proposizione:** se $G \cong \overline{G}$, cioè se è autocomplementare, allora se $n=|V|$, o “n” è divisibile per 4, o, se non lo è, “n-1” sarà divisibile per 4.

Dimostrazione:

$$\begin{aligned}
 G \cong \overline{G} &\Leftrightarrow |E| = |\overline{E}| \\
 m = |E| &\quad |\overline{E}| = \binom{n}{2} - m \\
 m = \binom{n}{2} - m &\Rightarrow 2m = \frac{n(n-1)}{2} \Rightarrow 4m = n(n-1)
 \end{aligned}$$

1. n è pari $\Rightarrow n-1$ è dispari $2m = (n-1)\frac{n}{2}$ quindi $\frac{n}{2}$ deve esser pari, quindi n deve essere divisibile per 4
2. n è dispari $\Rightarrow n-1$ è pari $2m = n\frac{(n-1)}{2}$ quindi $\frac{n-1}{2}$ deve essere pari, quindi $n-1$ è divisibile per 4



- **Esercizio:** si costruisca un grafo regolare, con un numero pari di vertici, autocomplementare, e si dimostri che non è possibile.

Il grafo deve quindi sottostare ai seguenti vincoli:

1. $G \cong \bar{G}$
2. G è k -regolare (k intero)
3. $|V|$ è pari

Si dimostri per assurdo: $\exists G \ni (1) \wedge (2) \wedge (3)$ (esiste un grafo G per cui valgono tutte le proprietà sopra elencate)

$$|\bar{E}| = \binom{n}{2} - m \rightarrow 2m = \frac{n(n-1)}{2}$$

Per il lemma della stretta di mano $2m = kn \rightarrow kn = \frac{n(n-1)}{2}$

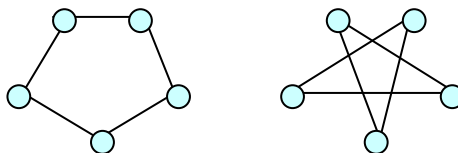
Quindi $k = \frac{n-1}{2}$, cioè n non è pari, che contraddice la terza proprietà

Alternativamente la dimostrazione poteva essere fatta in modo più diretto:

$G \cong \bar{G} \wedge k\text{-regolare} \Rightarrow \bar{G}$ è $(n-1-k)$ -regolare

Quindi $k = n-1-k$ che è assurdo.

- **Osservazione:** l'unico grafo 2-regolare autocomplementare è il ciclo di cardinalità 5 (C_5)



$$\left. \begin{array}{l} [k\text{-regolare, } k=2] \quad 2m = 2n \rightarrow m = n \\ [autocomplementare] \quad 2m = \frac{n(n-1)}{2} \end{array} \right\} \quad 2n = \frac{n(n-1)}{2} \rightarrow 4 = n-1 \rightarrow n = 5$$



GRAFI CONNESSI

- **Proposizione:** sia $G=(V,E)$ n.o. in cui $\forall u \in V: \deg(u) \geq \frac{n-1}{2}$ ($n=|V|$), se è valida questa condizione il grafo è **CONNESSO**.

Dimostrazione: in un grafo connesso qualunque coppia di vertici io prenda, esiste sempre un cammino che li unisce (ovvero il numero di componenti connesse è uguale a 1)

Per assurdo: suppongo che G non sia connesso, suppongo quindi che il numero di componenti connesse sia ≥ 2 , per la dimostrazione basta che siano $=2$.

Definisco V_1 e V_2 componenti connesse di G

$u \in V_1 : \deg(u) \geq \frac{n(n-1)}{2}$ dove i vertici

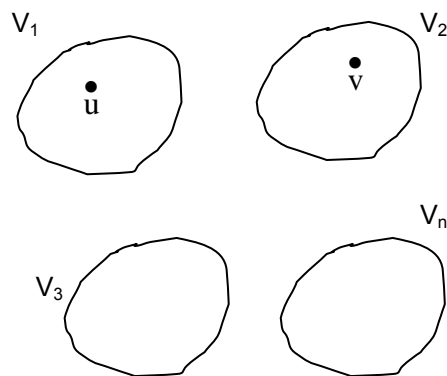
adiacenti devono essere tutti $\in V_1$:

$$|V_1| \geq \frac{n-1}{2} + 1 \wedge |V_2| \geq \frac{n-1}{2} + 1$$

$$|V_1| + |V_2| \geq \frac{n-1}{2} + 1 + \frac{n-1}{2} + 1$$

$$n = |V| \geq |V_1| + |V_2| \geq n + 1 \rightarrow n \geq n + 1$$

Che è assurdo.



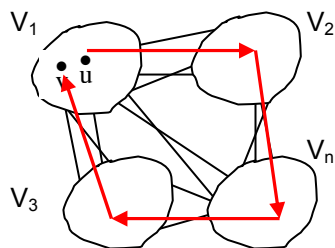
- **Esercizio:** $G=(V,E)$ n.o. e k costante intera ≥ 1

Se $\deg(u) \geq \frac{n}{k} \quad \forall u \in V$, allora il grafo è connesso.



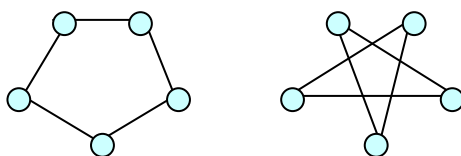
a) **G disconnesso** \Rightarrow **\bar{G} connesso**

- o **VERO**, si dimostra usando le componenti connesse: se le componenti in G sono connesse, anche se in \bar{G} prese singolarmente risultano disconnesse, prese nel complesso esisterà sempre un cammino tra le varie componenti che unisce due vertici all'interno della stessa componente



b) **G connesso** \Rightarrow **\bar{G} disconnesso**

- o **FALSO**, C_5 ad esempio lo verifica essendo $G \cong \bar{G}$ ed essendo entrambi connessi

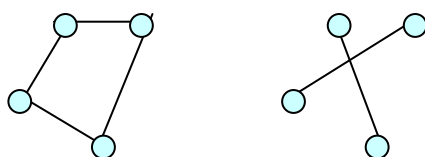


c) **G disconnesso** \Rightarrow **\bar{G} disconnesso**

- o **FALSO**, abbiamo appena dimostrato il contrario

d) **G connesso** \Rightarrow **\bar{G} connesso**

- o **FALSO**, portando un contro esempio, un grafo G con 4 vertici il cui isomorfo è disconnesso



- **Proposizione** (condizione necessaria perché un grafo G sia connesso):
sia $G=(V,E)$ n.o., allora $m \geq n-1$ (dove $m=|E|$ e $n=|V|$)

Dimostrazione:

ipotesi: G è connesso

tesi: $m \geq n-1$

Si dimostra per induzione su n :

casi base: $n=1 \quad m=0=n-1$

$n=2 \quad m=1=n-1$

Passo induttivo:

ipotesi induttiva ($n \geq 3$): la proposizione vale per tutti i grafi con al più n vertici

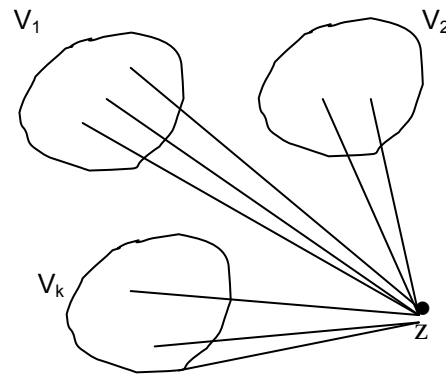
Dimostriamo quindi che l'ipotesi vale per un grafo $G=(V,E)$ n.o. con $|V|=n+1$.

Prendiamo un vertice $z \in V$, e G' sarà il sottografo indotto, $G'=G[V \setminus \{z\}]$, siano $V_1, V_2, \dots, V_k \geq 1$ le k componenti connesse di G' . Su ogni componente connessa posso applicare l'ipotesi induttiva.

$\forall i = 1 \dots k, |E_i| \geq |V_i| - 1$ per ipotesi induttiva,

$$\sum_{i=1}^k |E_i| \geq \sum_{i=1}^k (|V_i| - 1)$$

$$\sum_{i=1}^k |E_i| \geq \sum_{i=1}^k |V_i| - k$$



Sappiamo che $|E| = \sum_{i=1}^k |E_i| + \deg(z)$

cioè la somma di tutti gli archi delle componenti connesse, più tutti gli archi incidenti al vertice z che è stato rimosso, che sono k archi dovendo essere un grafo connesso, quindi $\deg(z) \geq k$, e quindi:

Tesi: $|E| \geq |V| - 1$

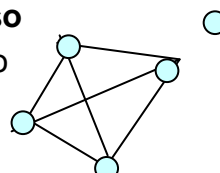
$$|E| = \sum_{i=1}^k |E_i| + \deg(z) \rightarrow |E| \geq \sum_{i=1}^k |E_i| + k \geq \underbrace{\sum_{i=1}^k |V_i|}_{=|V|-1} = |V| - 1 \leftarrow \text{il vertice } z$$

a) G è connesso $\Rightarrow m \geq n-1$

- **VERO**, l'abbiamo appena dimostrato

b) $m \geq n-1 \Rightarrow G$ è connesso

- **FALSO**, basta un contro esempio



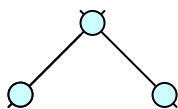
$n=5$
 $m=6$
 $m \geq n-1$ ma grafo
disconnesso



- **Proposizione:** sia $G=(V,E)$ n.o. aciclico, allora necessariamente $m \leq n-1$

Dimostrazione per induzione:

caso base (ovvio): $n=3$



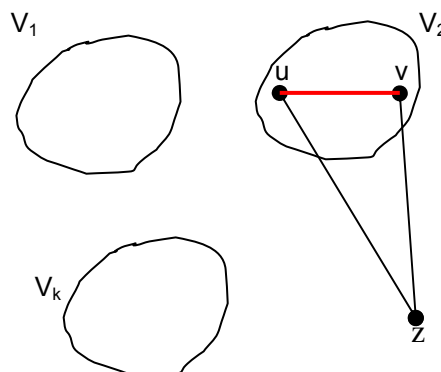
$m=0, 1, 2 \leq n-1$

Passo induttivo:

$G=(V,E)$ n.o. aciclico con $n+1$ vertici.

Procedo come nella dimostrazione

precedente, prendendo $z \in V$ e $G'=G[V \setminus \{z\}]$



Per tutte le componenti connesse posso applicare l'ipotesi induttiva (aciclicità), cioè $\forall i=1 \dots k$

$$|E_i| \leq |V_i| - 1$$

$$\sum_{i=1}^k |E_i| \leq \sum_{i=1}^k |V_i| - k$$

$$\sum_{i=1}^k |E_i| + k \leq |V_i| - 1$$

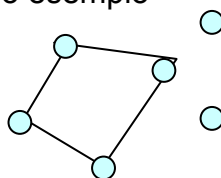
$$|E| = \sum_{i=1}^k |E_i| + \deg(z) \leq \sum_{i=1}^k |E_i| + k \leq |V_i| - 1$$

a) **G è aciclico** \Rightarrow **$m \leq n-1$**

○ **VERO**, l'abbiamo appena dimostrato

b) **$m \leq n-1$** \Rightarrow **G è aciclico**

○ **FALSO**, basta un contro esempio



$n=6$
 $m=4$
 $m \leq n-1$ ma grafo ciclico

- **Esercizio:** $\forall u \in V, \deg(u) \geq 2 \Rightarrow G$ ciclico



ALBERI LIBERI

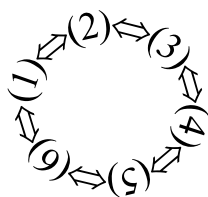
$G=(V,E)$ n.o. si dice **ALBERO LIBERO** se G è contemporaneamente connesso e aciclico.

GRAFI ACICLICI	ALBERI LIBERI	GRAFI CONNESSI
$m < n-1$	$m = n-1$	$m > n-1$
	$m > \rightarrow$	$\leftarrow m$

- **Teorema** (*proprietà degli alberi liberi*): sia $G=(V,E)$ n.o. allora le seguenti affermazioni sono equivalenti:

- (1) G è un albero libero
- (2) Due vertici qualsiasi di G sono collegati da un unico cammino semplice
- (3) G è connesso, ma se un qualunque arco di G viene rimosso, il grafo si disconnette (il numero delle componenti connesse sarà 2, “connettività fragile”)
- (4) G è connesso e $|E|=|V|-1$ (devono valere entrambe le condizioni contemporaneamente)
- (5) G è aciclico e $|E|=|V|-1$ (devono valere entrambe le condizioni contemporaneamente)
- (6) G è aciclico, ma se un qualunque arco viene aggiunto a G il grafo diventa ciclico

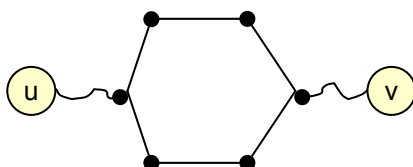
Le condizioni si implicano reciprocamente



Bisogna quindi dimostrare che
 $(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (4) \Rightarrow (5) \Rightarrow (6) \Rightarrow (1)$

Dim. (1) \Rightarrow (2)

Prendiamo due vertici $u, v \in V$ e supponiamo per assurdo che $\exists u, v \in V$ collegati tra loro da almeno due cammini diversi

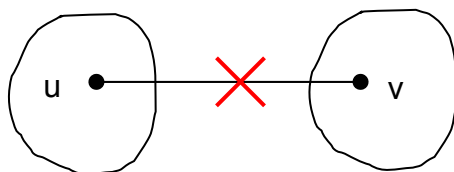


Abbiamo creato un ciclo, e questo viola la (1), se un grafo è un albero libero non ci possono essere cicli al suo interno.



Dim. (2)⇒(3)

Per ipotesi due vertici u, v sono connessi da un unico cammino, se si rimuove quest'unico cammino si creano necessariamente due componenti connesse

**Dim. (3)⇒(4)**

Bisogna dimostrare per induzione che $|E|=|V|-1$

Caso base: $n=1 \quad |E| = 0 = 1-1 = n-1$

$N=2 \quad |E| = 1 = 2-1 = n-1$

Passo induttivo:

$G=(V,E)$ con $n+1$ vertici. Provo a rimuovere un arco, disconnettendo per ipotesi il grafo in due componenti connesse V_1 e V_2 con $|V_1|+|V_2|=|V|$ e $|E|=|E_1|+|E_2|+1 \leftarrow \text{arco rimosso}$

$$\left. \begin{array}{l} |E_1|=|V_1|-1 \\ |E_2|=|V_2|-1 \end{array} \right\} \quad |E_1|+|E_2| = |V_1|+|V_2| - 2 \rightarrow |E_1|+|E_2|+1=|V_1|+|V_2|-1 \rightarrow |V_1|+|V_2|-1=|V|-1$$

Dim. (4)⇒(5)

Ipotesi: G è connesso e $|E|=|V|-1$

Tesi: G è aciclico

Dimostrazione per assurdo:

suppongo che esista un ciclo $\langle x_0 x_1 \dots x_k \rangle$ con $x_k=x_0 \quad \forall i=1 \dots k : (x_{i-1}, x_i) \in E$

$$\begin{array}{lll} - G_k=(V_k, E_k) & V_k=\{x_0, x_1, \dots, x_{k-1}\} & |V_k|=k=|E_k| \\ & E_k=\{(x_0, x_1), (x_1, x_2), \dots, (x_{k-1}, x_0)\} & E_k \subseteq E \end{array}$$

$$\begin{array}{lll} - G_{k+1}=(V_{k+1}, E_{k+1}) & V_{k+1}=V_k \cup \{x_k\} & \\ & E_{k+1}=E_k \cup \{(x_i, x_k)\} & |E_{k+1}|=k+1 \end{array}$$

Posso incrementare k in continuazione ($G_{k+2}, G_{k+3} \dots$) fino a G_n , con n archi, tutti archi appartenenti al grafo di partenza, ma il grafo di partenza ha $n-1$ archi, quindi siamo arrivati ad un assurdo.



Dim. (5)⇒(6)

Ipotesi: G è aciclico e $|E|=|V|-1$

Tesi: G è aciclico (fragile, se aggiungo un arco diventa ciclico)

Dimostrazione:

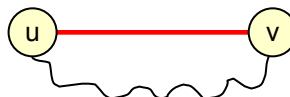
G è una foresta e supponiamo ci siano k componenti connesse ($k \geq 1$).

Chiamiamo $G_1 \dots G_k$ i sottografi indotti di G dalle k componenti connesse, che sono a loro volta alberi, ma se G_i è un albero $\forall i=1 \dots k$ $|E_i|=|V_i|-1$

$$|E| = \sum_{i=1}^k |E_i|$$

$$|E| = \sum_{i=1}^k |V_i| - k$$

$$|E| = |V| - k$$



Ma per ipotesi $|E|=|V|-1$, quindi $k=1$, ovvero ho una sola componente connessa, cioè G non è un grafo aciclico qualsiasi, ma è proprio un albero, quindi aggiungendo un arco si formerà un ciclo.

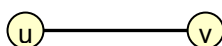
Dim. (6)⇒(1)

Ipotesi: G è aciclico (fragile)

Tesi: G è un albero libero

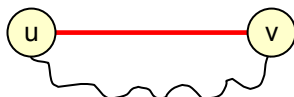
Se per ipotesi è già aciclico è sufficiente dimostrare che è connesso perché sia un albero libero, presa quindi una coppia di vertici $u, v \in V$ deve esistere un cammino che li unisce.

- 1) $u, v \in V$ sono adiacenti
 $\exists (u, v) \in E$



- 2) $u, v \in V$ non sono adiacenti
 $\exists (u, v) \in E$

Se io aggiungo questo arco per unire u e v allora nego l'ipotesi, in quanto esisterebbe sicuramente un cammino tra u e v e quindi $\exists (u, v) \in E$



CLIQUE

• **Definizione:** dato $G=(V,E)$

- a) **CLIQUE** (cricca) è un sottoinsieme di vertici tale che il grafo indotto $C \subseteq V \ni G[C]$ è completo
- b) **INSIEME INDIPENDENTE:** $C \subseteq V \ni G[C]$ è vuoto (non ha archi)
- c) **COPERTURA DI VERTICI:** $C \subseteq V \ni \forall (u, v) \in E : u \in C \vee v \in C$

Sia $G=(V,E)$ n.o e $C \subseteq V$, le seguenti affermazioni sono equivalenti:

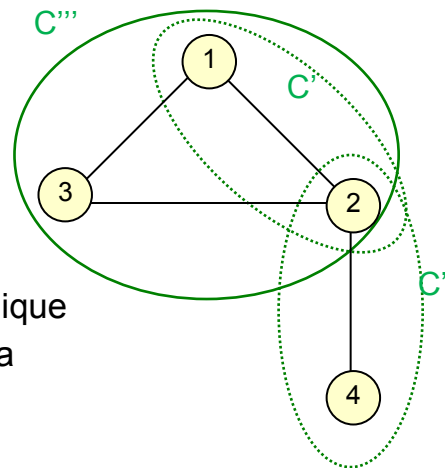
- C è clique in G
- C è un insieme indipendente in \overline{G}
- $\forall C$ è una copertura di vertici in \overline{G}

Clique **massima** e **massimale**

$C'=\{1,2\}$ clique

$C''=\{2,4\}$ clique massimale

$C'''=\{1,2,3\}$ clique massima



• **Definizione:** C è una clique, C si chiama clique

Massima se ha il maggior numero di vertici tra

Tutte le clique di G

$$[\forall C' \text{ clique di } G : |C'| \leq |C|]$$

• **Clique Number:** $\omega(G)$ =cardinalità della clique massima

C clique massima \Rightarrow clique massimale **MA**

C clique massimale \nRightarrow clique massima

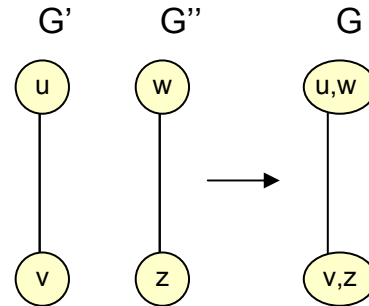


ISOMORFISMO TRA GRAFI

Grafo di associazione $G=(V,E)$

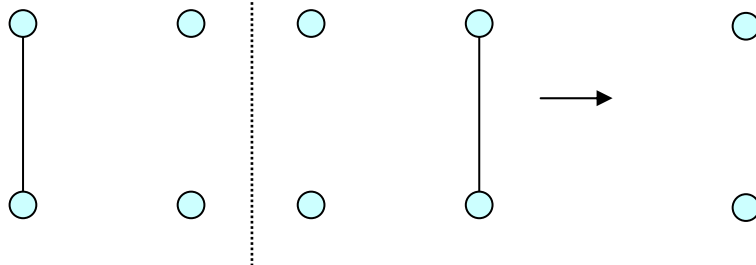
1) $V' \times V''$

$$2) ((u,v),(v,z)) \in E \Leftrightarrow \begin{cases} 1) u \neq v \\ 2) w \neq z \\ 3) (u,v) \in E' \Leftrightarrow (w,z) \in E'' \end{cases}$$

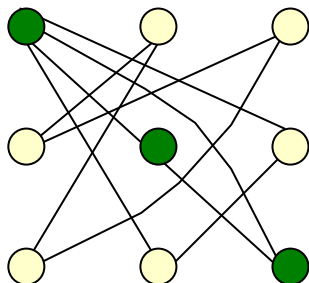
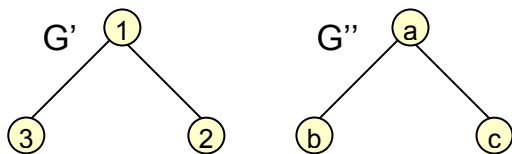


Nei casi

oppure

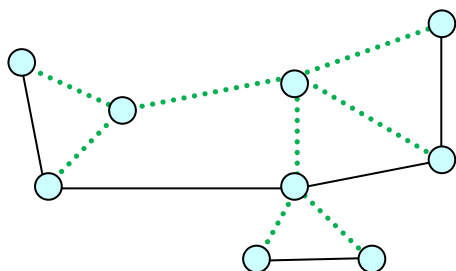


Teorema: $G' \cong G'' \Leftrightarrow \text{il clique number } \omega(G)=n \text{ (} N=|V'|=|V''| \text{)}$



MINIMUM SPANNING TREE (MST)

- Alberi di copertura (connessione o ricoprimento) minimi formati da un grafo $G=(V,E)$ non orientato
- **Definizione di albero di copertura:** è un insieme di archi $T \subseteq E$ per cui il sottografo $G'=(V,T)$ costituito da tutti i vertici del grafo di partenza e dagli archi T (non è un sottografo indotto) è un albero libero, quindi aciclico e connesso.



E' un grafo aciclico e connesso, un albero libero

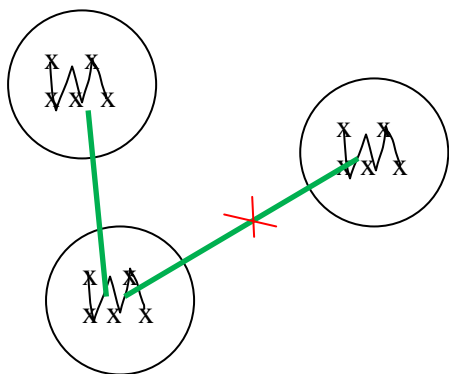
Ma posso aggiungere dei pesi sugli archi, quindi l'albero di copertura minimo sarà uno di quelli con l'equivalente minor peso totale di tutti gli archi (non col minor numero di archi, che sarà sempre $n-1$).

- $G=(V,E,\omega)$ n.o. connesso $\omega: E \rightarrow \mathbb{R}$ ω è la funzione peso degli archi
- $T \subseteq E$ $W(T) = \sum_{(u,v) \in T} \omega(u,v)$ il peso totale è la somma dei singoli pesi
- $T(G) = \{ T \in E \mid T \text{ è un albero di copertura} \}$

T è un MST se $T = \arg \min W(T')$, cioè se T è il valore che rende minima la funzione $W(T')$

CLUSTERING

Dato un grafo $G=(V,E,w)$ bisogna partizionare l'insieme dei vertici V in K sottoinsiemi C_1, C_2, \dots, C_k in modo che i vari gruppi (cluster) C_1, C_2, \dots, C_k siano omogenei



Si formano cluster omogenei di punti vicini, ad esempio, ma su questi elementi si può costruire uno spanning tree.

Se elimino gli archi più pesanti divido le varie componenti connesse, che formano appunto i cluster dell'insieme.



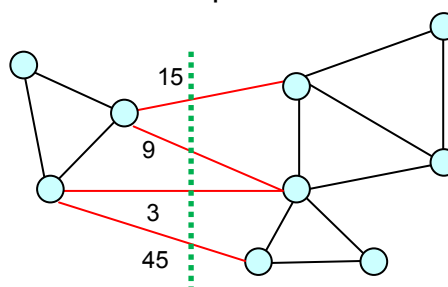
- **Definizione:** (u,v) è **sicuro** per A se $A \cup \{(u,v)\}$ è contenuto in qualche MST

GENERIC_MST (G,w)

- 1) $A = \emptyset$
- 2) while $|A| < n-1$
- 3) trova un arco sicuro per A
- 4) $A := A \cup \{(u,v)\}$

Dato un grafo connesso $G=(V,E)$ lo si vuole dividere in due parti

Per qualunque taglio di un grafo esiste un arco “tagliato” che appartiene a un MST



TAGLIO

Dato un grafo $G=(V,E)$ un taglio è un partizionamento del grafo in due parti, quindi è una coppia $(S, V \setminus S)$, $S \subseteq V$.

Dato un taglio $(S, V \setminus S)$ si dice che un arco $(u,v) \in E$ attraversa il taglio se un estremo di (u,v) è in S e l'altro in $V \setminus S$

ARCO LEGGERO

Dato un taglio $(S, V \setminus S)$, (u,v) è un arco leggero che attraversa $(S, V \setminus S)$ se:

- 1) (u,v) attraversa il taglio
- 2) $\forall (x,y) \in E$ che attraversa il taglio: $\omega(u,v) \leq \omega(x,y)$ [il peso è il minore tra tutti i pesi degli altri archi che attraversano il taglio]

RISPETTO

Dati un taglio $(S, V \setminus S)$ e un insieme di archi $A \subseteq E$, si dice che il taglio $(S, V \setminus S)$ rispetta A se \exists arco in A che attraversa il taglio.



TEOREMA FONDAMENTALE DEGLI ALBERI DI COPERTURA MINIMI

Sia $G=(V,E, \omega)$ n.o., $\omega:E\rightarrow\mathbb{R}$, connesso, e siano

- 1) Un sottoinsieme di archi $A\subseteq E$ contenuto nel MST (T)
- 2) Sia $(S, V\setminus S)$ un taglio di G che rispetti l'insieme A
- 3) Sia (u,v) arco leggero che attraversa il taglio $(S, V\setminus S)$

Allora l'arco (u,v) è sicuro per A .

Dimostrazione:

tesi: $\exists T'$ MST tale che $A\cup\{(u,v)\}\subseteq T'$

- 1) $(u,v)\subseteq T \Rightarrow A\cup\{(u,v)\}\subseteq T$
- 2) $(u,v)\notin T$

$$\begin{array}{ll} T\cup\{(u,v)\} & \exists(x,y)\in T \text{ che attraversa } (S, V\setminus S) \\ T' = T\cup\{(u,v)\} \setminus \{(x,y)\} & \omega(u,v) \leq \omega(x,y) \end{array}$$

T' è albero di copertura

$$W(T') = W(T) + \omega(u,v) - \omega(x,y) \leq W(T)$$

$W(T) \leq W(T') \leq W(T)$ ma T è un albero di copertura minimo, quindi anche T' è un albero di copertura minimo.

• **Corollario:** sia $G=(V,E)$ n.o. connesso e siano:

- a) $A\subseteq E$ contenuto in qualche MST
- b) $G_A=(V,A)$ sottografo di G con gli archi di A e sia C una componente connessa della foresta G_A
- c) Sia (u,v) un arco leggero che collega la componente connessa con un'altra componente connessa di G_A

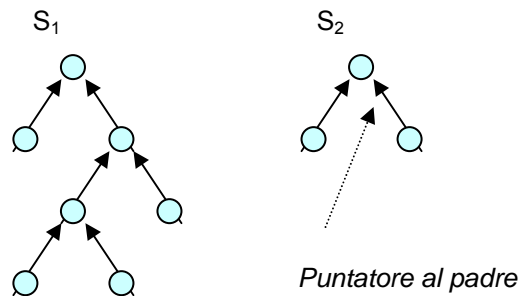
Tesi: l'arco (u,v) è sicuro per A

Se si prende il taglio $(C, V\setminus C)$, questo taglio rispetta A per costruzione, quindi si rientra nel caso del teorema fondamentale, e quindi l'arco è sicuro.



COLLEZIONE DI INSIEMI DISGIUNTI

S_1, S_2, \dots, S_n sono rappresentati come alberi e gli insiemi sono indicati con un RAPPRESENTANTE: la radice dell'albero



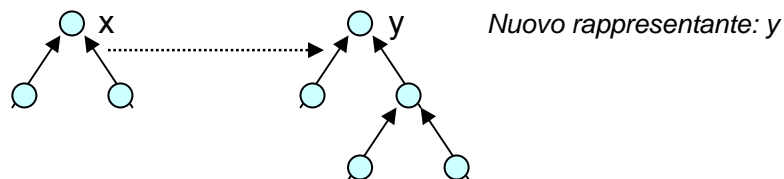
OPERAZIONI

1) MAKE_SET (x)

Accetta un elemento in ingresso e crea l'insieme costituito dall'elemento x

2) UNION (x, y)

Fa l'unione tra gli elementi rappresentati da x e da y



3) FIND_SET (x)

Restituisce il rappresentante dell'insieme che contiene l'elemento x

ALGORITOMO DI KRUSKAL

Si occupa di calcolare un possibile albero di copertura minimo su un grafo pesato.

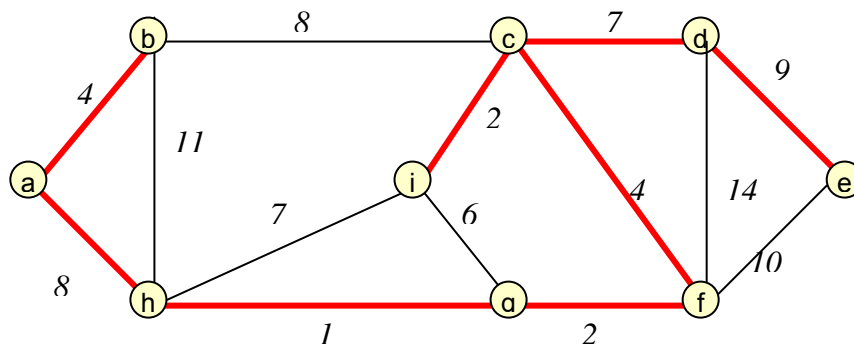
MST_KRUSKAL (G, w)

- 1) $A \leftarrow \emptyset$ *//parte dall'insieme vuoto e via via aggiunge elementi*
- 2) for each $u \in V[G]$ do *//creo tanti insiemi quanti sono i vertici del grafo*
- 3) MAKE_SET(u)
- 4) ordina gli archi di E per peso non decrescente
- 5) for each $(u,v) \in E$ in ordine di peso do
- 6) if FIND_SET(u) \neq FIND_SET(v) then *//evita che si creino cicli*
- 7) $A \leftarrow A \cup \{(u,v)\}$ *//amplia l'insieme A*
- 8) UNION(u,v) *//unisce i due insiemi*
- 9) return A



Esempio:

A —



Complessità nel caso peggiore $O(g(u))$ [grado di u]

$n=|V|$ $m=|E|$ $n \leq m-1$

- | | |
|------------------|-----------------|
| 1) $O(1)$ | |
| 2) $O(n)$ | |
| 3) $O(1)$ | |
| 4) $O(m \log m)$ | |
| 5) $O(m)$ | } $O(m \log m)$ |
| 6) $O(\log m)$ | |
| 7) $O(1)$ | |
| 8) $O(1)$ | |
| 9) $O(1)$ | |

Complessità totale: $O(m \log m)$

ALGORITMO DI PRIM

Accetta in ingresso:

Un grafo $G=(V,E)$

Una funzione peso $\omega: E \rightarrow \mathbb{R}$

Un vertice sorgente $r \in V$

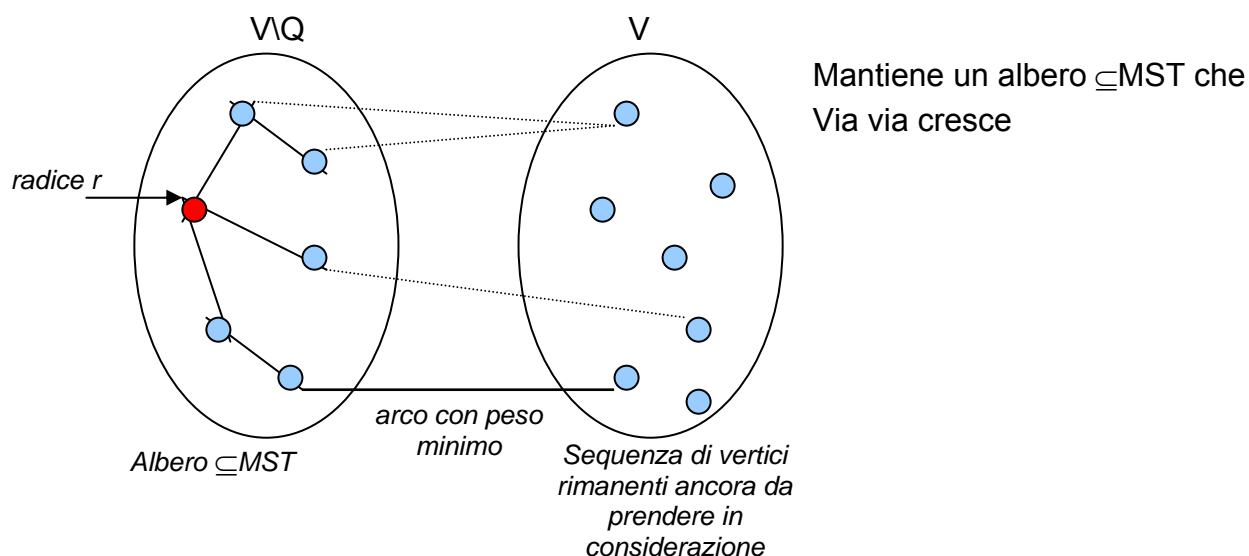
} Come per l'algoritmo di Kruskal

Mantiene istante per istante un albero contenuto in qualche MST

Q: insieme dei vertici da esaminare (inizializzato come $Q=V$), organizzato come coda con priorità (HEAP)

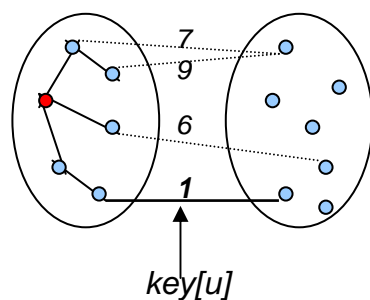


Troviamo un partizionamento del grafo in due parti



Dato $u \in V$

- a) **key[u]** è il valore del peso minimo dell'arco che collega u all'albero in costruzione



- b) **$\pi[u]$** è il predecessore di u nel grafo che si sta costruendo
In un albero radicato ogni vertice ha un unico genitore, tranne la radice (NIL)
L'albero che si sta costruendo è un albero radicato quindi $\pi[u]$ è un puntatore al genitore del vertice u .

Albero in costruzione (ad ogni iterazione)

$$A = \{(u, \pi[u]) \mid u \in V \setminus \{r\}\}$$

quindi $u \in$ all'insieme dei vertici, meno l'insieme che contiene l'albero, meno la radice che non ha genitore

Albero che risulta alla fine:

$$A = \{(u, \pi[u]) \mid u \in V \setminus \{r\}\}$$

Kruskal fa crescere una foresta
Prim fa crescere un unico albero



MST_PRIM(G,w,r)

- 1) $Q \leftarrow V[G]$ *//costruisco la coda con priorità ($V \setminus Q = \emptyset$)*
- 2) for each $u \in Q$ do *//inizializzo la chiave e il predecessore*
- 3) $key[u] \leftarrow \infty$ *//ogni chiave è posta a ∞*
- 4) $key[r] \leftarrow 0$ *//la chiave della radice è 0*
- 5) $\pi[r] \leftarrow NIL$ *//il predecessore della radice è NIL*
- 6) while $Q \neq \emptyset$ do
- 7) $u \leftarrow \text{EXTRACT_MIN}(Q)$ *//prendo il vertice in corrispondenza del quale c'è
//un arco di peso minimo collegato ad un $v \in V \setminus Q$*
- 8) for each $v \in \text{Adj}[u]$ do *//aggiorno key e π del nuovo Q adiacenti a u*
- 9) if $v \in Q$ and $\omega(u,v) < key[v]$ then
- 10) $\pi[v] \leftarrow u$
- 11) $key[v] \leftarrow \omega(u,v)$ *//aggiorno il peso (riordino l'heap)*
- 12) return $A = \{(u, \pi[u]) \mid u \in V \setminus \{r\}\}$

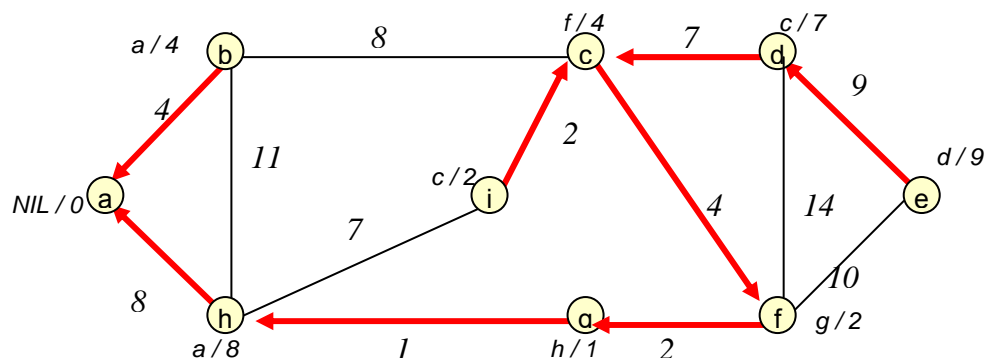
Complessità:

- 1) $O(1)$
- 2) $O(n)$
- 3) $O(1)$
- 4) $O(1)$
- 5) $O(1)$
- 6) entra nel ciclo n volte
- 7) $O(\log n)$
- 8) entra nel ciclo $O(m)$ volte per il lemma della stretta di mano
- 9) $O(1)$
- 10) $O(1)$
- 11) $O(\log n)$
- 12) $O(1)$

Complessità totale:

$O(n) + O(n \log n) + O(m \log n) =$
 $= O(n \log n) + O(m \log n)$
 ma $n \leq m-1$ quindi
 $= O(m \log n)$

Esempio: sequenza dei nodi a, b, h, g, f, c, i, d, e



ALGORITMI GREEDY (GOLOSI o MIOPI)

Sono tutti algoritmi **iterativi** ed **incrementali**, cioè procedono per un certo numero di step incrementando ad ogni passo un valore.

- Partono da un certo stato $A \leftarrow \emptyset$
- Si guardano intorno
- Si muovono verso una direzione opportuna in modo che la scelta sia la migliore possibile in quell'istante di tempo (l'arco con il minor peso)

• Selezione delle attività

C'è una risorsa che deve essere occupata da n attività. Ogni attività ha due stati: il tempo d'inizio (s) e quello di fine (f)

\underline{n} $J = \{J_1, J_2, \dots, J_n\} \quad \forall i=1 \dots n \quad (s_i, f_i)$ dove

s_i = tempo d'inizio di J_i

f_i = tempo di fine di J_i

Prese due attività J_i e J_k ($1 \leq i, k \leq n$), J_i e J_k sono compatibili se $[s_i, f_i] \cap [s_k, f_k] = \emptyset$ cioè se non c'è intersezione tra gli intervalli di tempo delle attività.

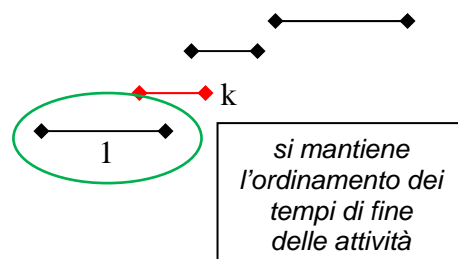
GREEDY_ACTIVITY_SELECTOR (s, f)

- 1) $N \leftarrow \text{rows}[s]$
- 2) ordina le n attività per tempo di fine crescente ($f_1 \leq f_2 \leq \dots \leq f_n$)
- 3) $A \leftarrow \{1\}$
- 4) $j \leftarrow \{1\}$
- 5) for $i \leftarrow 2$ to n do
- 6) if $s_i \geq f_j$ then
- 7) $A \leftarrow A \cup \{i\}$
- 8) $j \leftarrow i$
- 9) return A

Esiste sempre un insieme di attività mutuamente compatibili di cardinalità massima che contiene 1, cioè la prima attività che termina.

A

- 1) $1 \in A$ caso banale
- 2) $1 \notin A$ sia K l'attività che ha tempo di fine minimo tra tutte le attività
 $A' = A \setminus \{K\} \cup \{1\} \quad |A'| = |A|$



- Cerchiamo un algoritmo greedy per trovare una **clique massima**

$$C \subseteq V \quad \exists! \quad \forall u, v \in C \quad (u \neq v) \quad (u, v) \in E$$

GREEDY_CLIQUE(G)

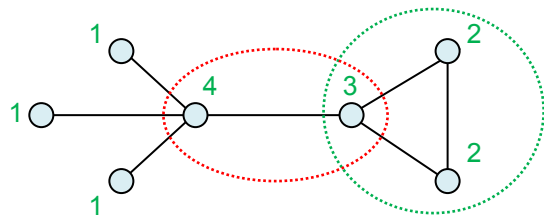
- 1) ordina i vertici di G per grado decrescente
- 2) $C \leftarrow \emptyset$
- 3) for each $u \in V[G]$ per ordine decrescente
- 4) if IS_A_CLIQUE(C, u) then
- 5) $C \leftarrow C \cup \{u\}$
- 6) return C

IS_A_CLIQUE(G)

- 1) for each $v \in V$ do
- 2) if $(u, v) \notin E$ then
- 3) return FALSE
- 4) return TRUE

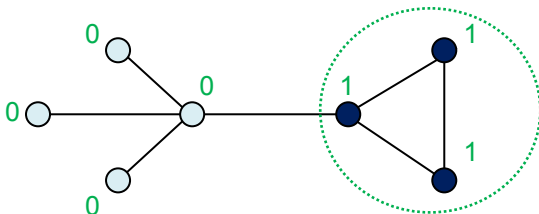
Complessità finale: $O(n^2)$

L'algoritmo inserisce per primo il vertice di grado 4, poi quello di grado 3, restituendo la clique di cardinalità 2, non quella di cardinalità che è la clique massima.

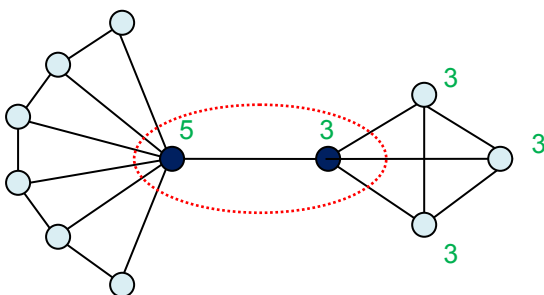


- **Osservazione:** GREEDY_CLIQUE restituisce SEMPRE una clique massimale, ma non garantisce che sia sempre massima.

Si può modificare l'ordinamento dell'algoritmo, ordinando i vertici di G per numero di triangoli incidenti decrescente



sembra funzionare, ma con un contro esempio si può dimostrare il contrario



l'algoritmo sceglie erroneamente il vertice di grado 5, perdendo già in partenza la clique massima

Gli algoritmi greedy sono corretti, ma efficienti solo in certe tipologie di problemi.

- **Struttura generale di un algoritmo greedy**

GENERIC_GREEDY

- 1) ordinamento ad hoc
- 2) $A \leftarrow \emptyset$
- 3) for each elemento x preso in ordine
- 4) if $A \cup \{x\}$ è OK then
- 5) return A



CAMMINI MINIMI

Dato un grafo $G=(V,E)$ orientato e una funzione peso $\omega: E \rightarrow \mathbb{R}$ e dati due vertici $x_0=i$ e $x_n=j$, definiamo $\mathcal{C}_{ij}=\{p \mid p \text{ è un cammino tra } i \text{ e } j\}$

Se $p \in \mathcal{C}_{ij}$ allora il peso del cammino p è $\omega(p)=\sum$

$$\delta(i,j)=\begin{cases} \min \{ \omega(p) \mid p \text{ è il cammino tra } i \text{ e } j \} \\ +\infty \text{ se } i \text{ non è raggiungibile da } j \end{cases}$$

ovvero il minimo peso tra tutti quelli degli archi in \mathcal{C}_{ij} se \mathcal{C}_{ij} ha archi, cioè se $\mathcal{C}_{ij} \neq \emptyset$

δ è anche detta **DISTANZA** tra i e j

In corrispondenza di $\delta(i,j)$ c'è $p_{ij}=\operatorname{argmin}_{p \in \mathcal{C}_{ij}} \omega(p)$, cioè l'arco che minimizza la funzione peso, cioè che ha peso $\delta(i,j)$

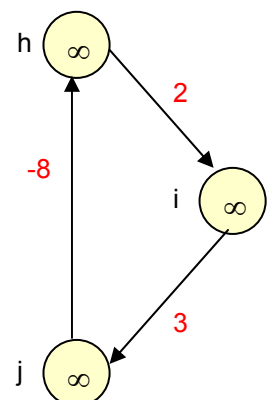
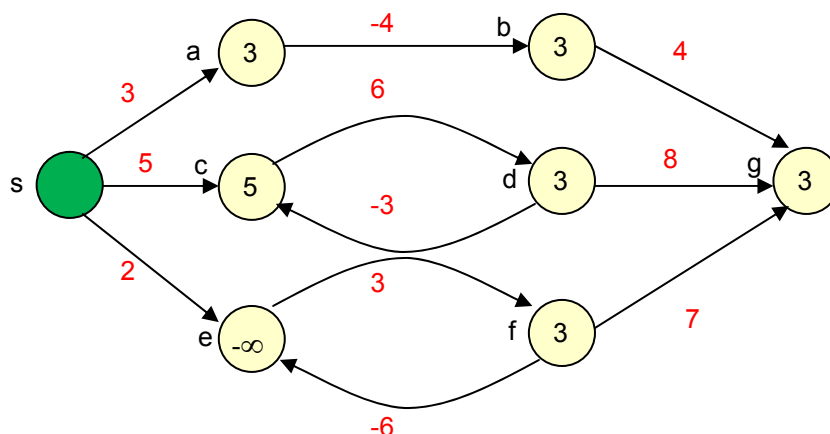
• Tipologie di problemi sui cammini minimi

SORGENTI / DESTINAZIONI	UNA	TUTTI
UNA	INPUT: $G=(V,E) \omega: E \rightarrow \mathbb{R} s,d \in V$ OUTPUT: $\delta(s,d) \mid p_{sd}=\text{opt}$	INPUT: $G=(V,E) \omega: E \rightarrow \mathbb{R} s \in V$ OUTPUT: $\forall u \in V : \delta(s,u) \mid p_{su}=\text{opt}$
TUTTI	INPUT: $G, \omega, d \in V$ OUTPUT: $\forall u \in V : \delta(u,d) \mid p_{ud}=\text{opt}$	INPUT: G, ω OUTPUT: $\forall u,v \in V : \delta(u,v) \mid p_{uv}=\text{opt}$

• Pesì negativi

$\forall u \in V$

- 1) $d[u]$ = stima della distanza tra s e u
- 2) $\pi[u]$ = predecessore (nell'albero dei cammini minimi)



Tra s e a l'unico cammino è quello di peso 3, ma tra s e c posso fare:

- | | | |
|---|----|--|
| 1) $\langle s, c \rangle :$ | 5 | } Avendo un ciclo ho infiniti cammini,
ma quello di peso minimo è il primo, 5 |
| 2) $\langle s, c, d, c \rangle :$ | 8 | |
| 3) $\langle s, c, d, c, d, c \rangle :$ | 11 | |
| 4) ...e così via | | |

Lo stesso ragionamento si può applicare al cammino tra s ed e:

- | | | |
|---|----|--|
| 1) $\langle s, e \rangle :$ | 2 | } Ci sono infiniti cammini legati al ciclo, ma il
ciclo negativo porta a non avere un cammino
minimo, quindi la distanza è $-\infty$ |
| 2) $\langle s, e, f, e \rangle :$ | -1 | |
| 3) $\langle s, e, f, e, f, e \rangle :$ | -4 | |
| 4) ...e così via | | |

• Grafo dei predecessori

$\pi[u] : G_\pi = (V_\pi, E_\pi)$ predecessore di u nell'albero dei cammini minimi

$$V_\pi = \{s\} \cup \{v \in V \mid \pi[v] \neq \text{NIL}\}$$

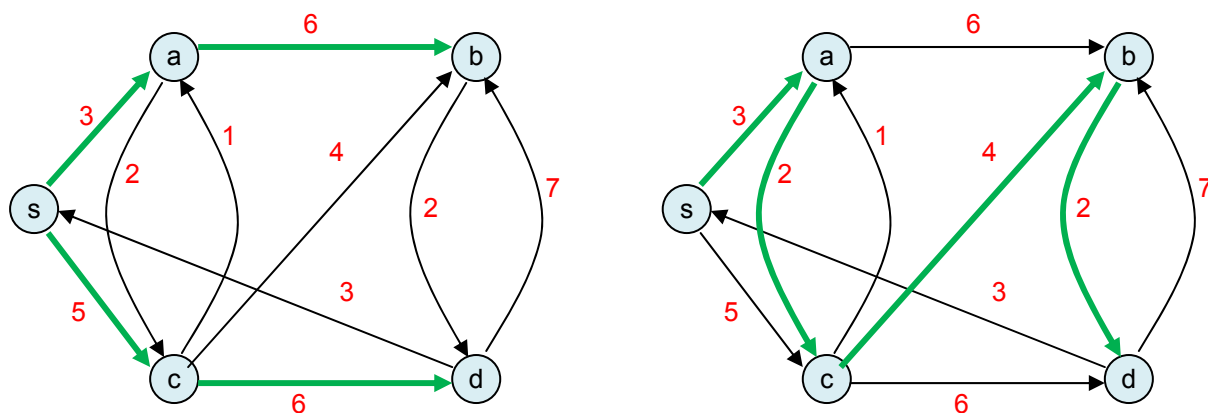
$$E_\pi = \{(\pi[u], u) \in E \mid u \in V_\pi \setminus \{s\}\}$$

$\pi[s] : \text{NIL}$

• Albero dei cammini minimi

$G' = (V', E')$, $V' \subseteq V$, $E' \subseteq E$ è un sottografo del grafo di partenza che soddisfa le seguenti proprietà:

- V' è l'insieme dei vertici di G raggiungibili da s [$V' = \{u \in V \mid \mathcal{C}_{su} \neq \emptyset\}$]
- G' è un albero radicato in s
- $\forall u' \in V'$ l'unico cammino in G' tra s e u' rappresenta un cammino minimo tra s e u' in G



Ci possono essere diversi cammini minimi in un grafo, nella figura precedente ne sono segnati in verde due equivalenti.



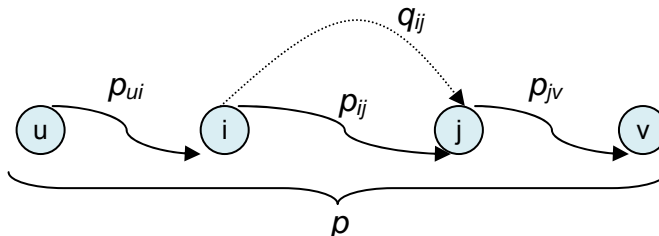
- **Osservazione: sottocammini di cammini minimi sono minimi**

Dato un grafo $G=(V,E)$ orientato, una funzione peso $\omega:E \rightarrow \mathbb{R}$ e dati due vertici $u,v \in V$ e sia p cammino minimo tra u e v , $p = \langle x_0 x_1 \dots x_{k-1} x_k \rangle$ $x_0=u$, $x_k=v$

Posto $1 \leq i \leq j \leq k$ allora $p_{ij} = \langle x_i \dots x_j \rangle$ è un cammino minimo tra x_i e x_j

Dimostrazione per assurdo:

Supponiamo per assurdo che esista un cammino q_{ij} per cui $\omega(q_{ij}) < \omega(p_{ij})$, che non è possibile essendo p un cammino minimo per ipotesi.



- **Proposizione:** sia $G=(V,E)$ orientato, $\omega:E \rightarrow \mathbb{R}$ e supponiamo che $p = \langle x_0 \dots x_k \rangle$ $x_0=u$, $x_k=v$ sia un cammino minimo tra u e v allora:

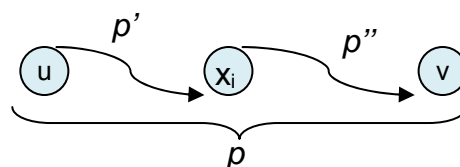
$$\delta(u,v) = \delta(u,x_i) + \delta(x_i,v) \quad \forall i=0 \dots k$$

Dimostrazione:

$$\delta(u,v) = \omega(p) = \omega(p') + \omega(p'')$$

ed essendo sottocammini di un cammino minimo allora il loro peso è uguale alla distanza tra i vertici

$$\delta(u,v) = \delta(u,x_i) + \delta(x_i,v)$$



- **Osservazione:** se p è un cammino minimo tra u e $v \in V$ e p è nella forma $p_{iu} \searrow x \rightarrow v$ allora $\delta(u,v) = \delta(u,x) + \omega(x,v)$

- **Lemma:** sia $G=(V,E)$ orientato con funzione peso $\omega:E \rightarrow \mathbb{R}$ e sia $s \in V$ il vertice sorgente, allora $\forall (u,v) \in E$ si ha che

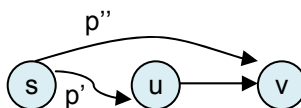
$$\delta(s,u) \leq \delta(s,u) + \omega(u,v) \quad \text{Disuguaglianza triangolare}$$

Dimostrazione:

a) $\delta(s,u) = +\infty \quad +\infty \leq +\infty$ (caso banale)



b) $\delta(s,u) < +\infty$ allora $\delta(s,v) < +\infty$ $\omega(p'') = \delta(s,v)$
 $\omega(p') = \delta(s,u)$



$\forall (u,v) \in E \quad \delta(s,v) \leq \delta(s,u) + \delta(u,v)$ con eguaglianze se e solo se l'arco (u,v) si trova su un cammino minimo tra s e v .



ALGORITMO DI DIJKSTRA

Strutture dati coinvolte:

- 1) $\pi[v]$ = predecessore di v
- 2) $d[v]$ = stima di cammino minimo

Alla fine dell'algoritmo:

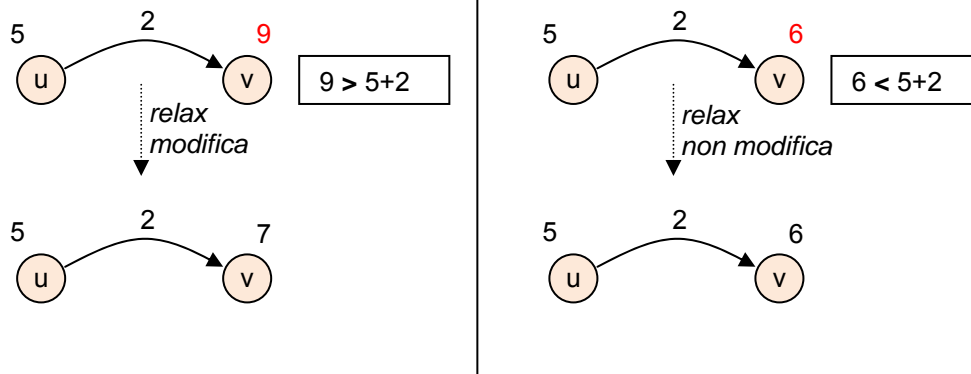
- $d[v] = \delta(s, v)$ (la stima di cammino minimo è la distanza tra il vertice sorgente e il vertice v)
- $\pi[v]$ tale che G_π è albero di cammini minimi

INIT_SINGLE_SOURCE(G, s)

- 1) for each $v \in V[G]$ do
- 2) $\pi[v] \leftarrow \text{NIL}$ //assegno ad ogni vertice il predecessore NIL
- 3) $d[v] \leftarrow \infty$ //la stima del cammino di ogni vertice sarà ∞
- 4) $d[s] \leftarrow 0$ //tranne la sorgente che è 0

RELAX(u, v, ω)

- 1) if $d[v] > d[u] + \omega(u, v)$ then
- 2) $d[v] \leftarrow d[u] + \omega(u, v)$ //aggiorno il valore di d
- 3) $\pi[v] \leftarrow u$ //aggiorno il valore del predecessore



Dopo la chiamata a RELAX(u, v, ω) si ha $d[v] \leq d[u] + \omega(u, v)$

- **Proposizione:** sia $G=(V, E)$, $\omega: E \rightarrow \mathbb{R}$ e $s \in V$ e supponiamo che sia invocata la procedura INIT_SINGLE_SOURCE(G, s), allora si ha che $\forall v \in V \delta(s, v) \leq d[v]$. Questa relazione vale per ogni sequenza di chiamate a RELAX, inoltre se $d[v] = \delta(s, v)$ (dopo un certo numero di chiamate a RELAX) $d[v]$ rimane invariato per ogni sequenza successiva di chiamate a RELAX.

Dimostrazione:

- 1) $v \neq s$ $d[v] = \infty \geq \delta(s, v)$
- 2) $v = s$
 - 2.1) v è su un ciclo negativo: $\delta(s, s) = -\infty \leq 0 = d[s]$
 - 2.2) v non è su un ciclo negativo: $\delta(s, s) = 0 = d[s]$

Supponiamo $d[v] < \delta(s, v)$ dopo una sequenza di chiamate a RELAX



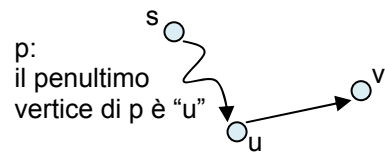
v sia il primo vertice per cui questo accade

$$\begin{aligned} d[u] + \omega(u,v) &= d[v] \\ &< \delta(s,v) \\ &\leq \delta(s,u) + \omega(u,v) \\ d[u] &< \delta(s,u) \leftarrow \text{assurdo} \end{aligned}$$

La chiamata a RELAX non altera mai i valori delle stime di u e quindi questa disuguaglianza valeva anche prima di RELAX, contraddice quindi l'ipotesi in cui si diceva che v era il primo vertice. Ora ne abbiamo trovato un altro (u) che viene prima di v . $d[v] = \delta(s,v)$ non varia dopo un numero di chiamate a RELAX. $d[v]$ parte da ∞ e viene decrementato fino al valore minimo.

$v \in V$ non è raggiungibile da s , allora subito dopo la INIT_SINGLE_SOURCE si ha che $d[v] = \delta(s,v)$, e questa relazione rimane invariata per tutte le successive chiamate a RELAX.

- **Proposizione:** sia $G=(V,E)$, $\omega:E \rightarrow \mathbb{R}$, $s,v \in V$ e sia p cammino minimo tra s e v . Supposto che si sia chiamata INIT_SINGLE_SOURCE e supposto che si sia chiamata varie volte RELAX e alla fine chiamiamo RELAX(u,v,ω), se in qualunque istante prima della chiamata all'ultima RELAX(u,v,ω) vale $d[u] = \delta(s,u)$, allora dopo aver chiamato RELAX(u,v,ω) si ha: **$d[v] = \delta(s,v)$**



Dimostrazione:

$$\begin{aligned} \delta(s,v) &\leq d[v] \\ &\leq d[u] + \omega(u,v) \\ &\leq \delta(s,u) + \omega(u,v) \\ &= \delta(s,v) \end{aligned}$$

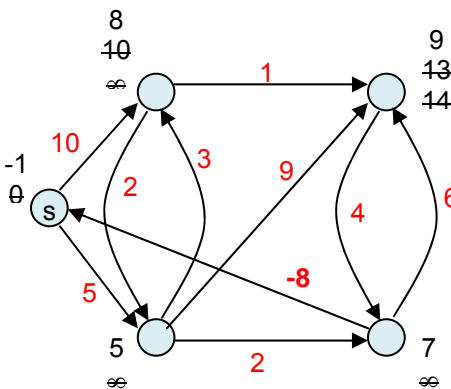
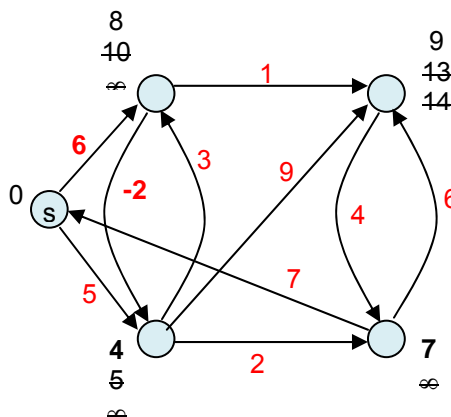
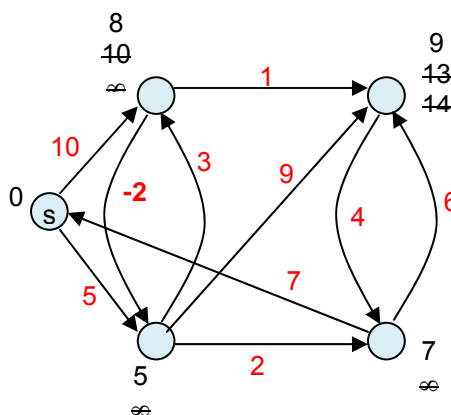
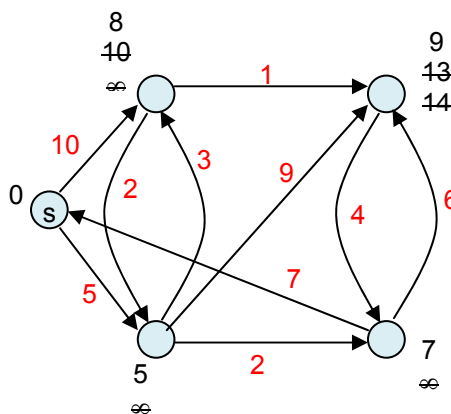
L'arco si trova quindi su un cammino minimo, vale $\delta(s,v) \leq d[v] \leq \delta(s,v) \Rightarrow d[v] = \delta(s,v)$. Assunto quindi che $\omega(u,v) \geq 0$ possiamo creare un algoritmo greedy con una coda con priorità Q che contiene i vertici non ancora estratti dall'algoritmo, ordinati secondo $d[v]$.

DIJKSTRA(G,ω,s)

- 1) INIT_SINGLE_SOURCE(G,s) //inizializzazione
- 2) $S \leftarrow \emptyset$ //insieme che contiene i vertici già usati
- 3) $Q \leftarrow V[G]$ // insieme che contiene i vertici da usare
- 4) while $Q \neq \emptyset$ do //finché non si finiscono i vertici
- 5) $u \leftarrow \text{EXTRACT_MIN}(Q)$ //riorganizzazione della coda
- 6) $S \leftarrow S \cup \{u\}$ //S è ordinate in modo crescente
- 7) for each $v \in \text{Adj}[u]$ do //guardo i vertici adiacenti ad u
- 8) RELAX(u,v,ω)



Esempio di utilizzo di Dijkstra:



Complessità dell'algoritmo di Dijkstra con $n=|V|$ e $m=|E|$

Q = array lineare

- 1) $O(n)$
 - 2) $O(1)$
 - 3) $O(1)$
 - 4) $-n$
 - 5) Q è array lineare quindi non comporta riorganizzazione dell'array $\rightarrow O(n)$
 - 6) $O(1)$
 - 7) $-m$
 - 8) $O(1)$
- $\left. \begin{array}{l} \text{5) } O(n) \\ \text{6) } O(1) \\ \text{7) } -m \\ \text{8) } O(1) \end{array} \right\} O(m)$
- $\left. \begin{array}{l} \text{4) } -n \\ \text{5) } O(n) \end{array} \right\} O(n^2)$

Q = heap binario

- 1) $O(1)$
 - 2) $O(1)$
 - 3) $O(1)$
 - 4) $-n$
 - 5) $O(\log n)$
 - 6) $O(1)$
 - 7) $-m$
 - 8) comporta un aggiustamento perché sia efficiente $\rightarrow O(\log n)$
- $\left. \begin{array}{l} \text{4) } -n \\ \text{5) } O(\log n) \end{array} \right\} O(n \log n)$
- $\left. \begin{array}{l} \text{6) } O(1) \\ \text{7) } -m \\ \text{8) } O(\log n) \end{array} \right\} O(m \log n)$

$$|E| = \sum_{u \in V} \text{out deg } ree(u) = \sum_{u \in V} \text{in deg } ree(u)$$

Dijkstra = $O(n^2 + m)$ ma dato che $m \leq n^2$, allora:

Dijkstra = $O((n+m) \log n)$, se il grafo è connesso ($m > n$) allora:

Complessità totale: $O(n^2)$

Complessità totale: $O(m \log n)$

Con un grafo **denso** ($m \approx n^2$) è meglio usare un **array lineare**, perché la complessità nel caso dell'heap binario sarebbe $O(n^2 \log n)$

Con un grafo **sparso** ($m \approx n$) è meglio usare un **heap binario** poiché la complessità è $O(n \log n)$, e in generale $n \log n < n^2$.

TEOREMA DI CORRETTEZZA DI DIJKSTRA

Sia $G=(V,E)$ un grafo orientato, con funzione peso $\omega:E \rightarrow \mathbb{R}$, e supponiamo che $\forall (u,v) \in E$ $\omega(u,v) \geq 0$ (tutti i pesi degli archi quindi sono positivi). Allora l'algoritmo è corretto, cioè:

1) $\forall u \in V : d[u] = \delta(s,u)$

2) G_π è un albero di cammini minimi alla fine di Dijkstra

Dimostrazione della prima affermazione:

ipotesi: $\omega(u,v) \geq 0$

tesi 1: alla fine $\forall u \in V : d[u] = \delta(s,u)$

tesi 2: $\forall u \in V : d[u] = \delta(s,u)$ nel momento in cui u viene estratto da Q

tesi2 \Rightarrow tesi1, quindi è sufficiente dimostrare tesi2 per assurdo.



Infatti per assurdo poniamo che tesi2 non sia vera, sia $u \in V$ il primo vertice per cui $d[u] \neq \delta(s,u)$ al momento dell'estrazione

Osservazioni:

1) $S \neq \emptyset$ [$u \neq s \rightarrow \delta(s,s) = 0 = d[s]$]

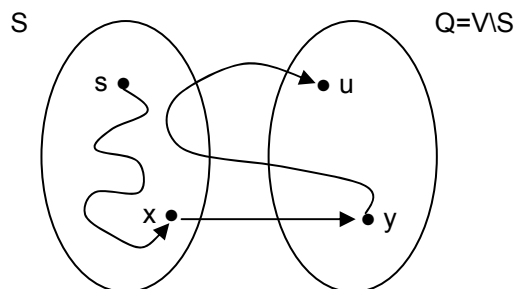
2) $\delta(s,u) < +\infty$ p cammino minimo tra s e u

$d[x] = d(s,x)$

$d[u] \leq d[y]$ poichè sia x che y devono ancora essere estratti e EXTRACT_MIN ha scelto u

$d[y] = \delta(s,y)$ questo è stato estratto x, ci sarà stata una RELAX tra tutti gli adiacenti di x, in particolare dell'arco (x,y)

$d[u] \leq \delta(s,u)$ avendo per ipotesi tutti i pesi positivi



ALGORITMO DI BELLMAN-FORD

A differenza di Dijkstra questo algoritmo funziona anche in presenza di pesi negativi sugli archi di un grafo orientato.

BELLMAN-FORD(G, ω, s)

- 1) INIT_SINGLE_SOURCE(G, s)
- 2) for $i \leftarrow 1$ to $|V[G]| - 1$ do
- 3) for each $(u,v) \in E$ do
- 4) RELAX(u, v, ω)
- 5) for each $(u,v) \in E$ do
- 6) if $d[v] > d[u] + \delta(u,v)$ then
- 7) return FALSE
- 8) return TRUE

Complessità totale: $O(n + mn + m) = O(mn)$

• Teorema di correttezza di Bellman-Ford

Sia $G(V, E)$ grafo orientato, $\omega: E \rightarrow \mathbb{R}$, $s \in V$

A) se non ci sono cicli negativi raggiungibili dalla sorgente allora alla fine di Bellman-Ford si avrà:

- 1) $\forall u \in V : d[u] = \delta(s,u)$
- 2) G_π è un albero di cammini minimi
- 3) Bellman-Ford restituisce **TRUE**

B) se ci sono cicli negativi raggiungibili dalla sorgente allora Bellman-Ford restituisce **FALSE**



Dimostrazione:

A.1) $\forall u \in V : d[u] = \delta(s, u)$?

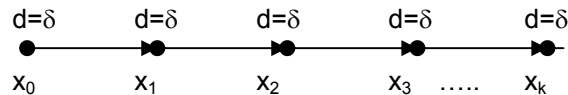
A.1.1) $\delta(s, u) = +\infty = d[u]$ dimostrato

A.1.2) $\delta(s, u) < +\infty$

sia p un cammino minimo tra s e u , $p = \langle x_0 x_1 \dots x_k \rangle$ $x_0 = s$ $x_k = u$

p è un cammino semplice, senza cicli, con $k+1$ vertici, ma $k+1 \leq n$,

quindi $k \leq n-1$ e quindi $d[x_0] = \delta(s, x_0) = \delta(s, s) = 0$



A.3) Bellman-Ford restituisce TRUE [$\forall (u, v) \in E : d[v] \leq d[u] + \omega(u, v)$] ?

sia $(u, v) \in E : d[v] = \delta(s, v)$

$\leq \delta(s, u) + \omega(u, v)$ per la proprietà triangolare

$= d[u] + \omega(u, v)$

B) [\exists un ciclo negativo \Rightarrow Bellman-Ford restituisce FALSE] ?

dim. per assurdo: supponiamo \exists un ciclo negativo raggiungibile da s

[$\mathcal{C} = \langle x_0 x_1 \dots x_k \rangle$ $\sum_{i=1}^k \omega(x_{i-1}, x_i) < 0$] e nego la tesi affermando che

Bellman-Ford restituirà TRUE, $\forall (u, v) \in E : d[v] \leq d[u] + \omega(u, v)$

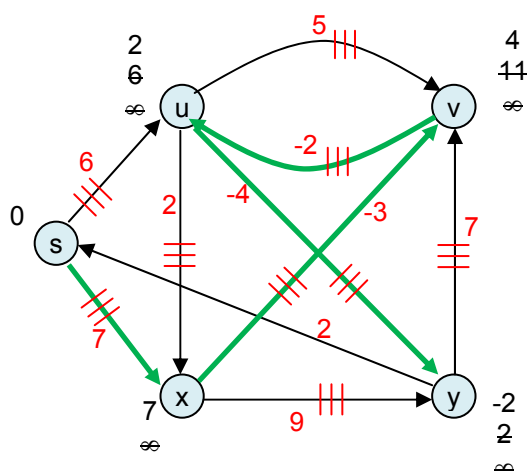
$\forall i = 1 \dots k : d[x_i] \leq d[x_{i-1}] + \omega(x_{i-1}, x_i)$

$$\sum_{i=1}^k d[x_i] \leq \sum_{i=1}^k d[x_{i-1}] + \sum_{i=1}^k \omega(x_{i-1}, x_i)$$

cioè $d[x_1] + d[x_2] + \dots + d[x_{k-1}] + d[x_k]$
 $d[x_0] + d[x_1] + d[x_2] + \dots + d[x_{k-1}]$

Quindi è un ciclo negativo poiché si possono semplificare le sommatorie, ottenendo

$$\sum_{i=1}^k \omega(x_{i-1}, x_i) \geq 0 \text{ che } \underline{\text{nega l'ipotesi.}}$$



- si segna una "taccu" per ogni RELAX
effettuata sull'arco

- simuliamo solo l'esecuzione per $d[\]$ e non
per $\pi[\]$ per semplicità



CAMMINI MINIMI TRA TUTTE LE COPPIE DI VERTICI

INPUT: $G=(V,E)$ orientato, $\omega:E \rightarrow \mathbb{R}$

OUTPUT: $\forall u,v \in V \begin{cases} d(u,v) \\ p_{u,v}^{OPT} \end{cases}$

ALL_PAIRS_DIJKSTRA(G,ω) $// \omega(u,v) \geq 0 \rightarrow \forall (u,v) \in E$

- 1) for each $s \in V$ do
- 2) **DIJKSTRA**(G,ω,s)

Complessità: $O(n \cdot n^2) = O(n^3)$ se Q è implementata come array lineare
 $O(n \cdot m \log n)$ se Q è implementata come heap binario

- se il grafo è sparso conviene usare l'heap binario: $O(n^2 \log n)$
- se il grafo è denso conviene usare l'array lineare: $O(n^3)$

ALL_PAIRS_BELLMAN_FORD(G,ω)

- 1) for each $s \in V$ do
- 2) **BELLMAN-FORD**(G,ω,s)

Complessità: $O(n \cdot m \cdot n) = O(n^2 m)$

- se il grafo è sparso: $O(n^3)$
- se il grafo è denso: $O(n^4)$

MOLTIPLICAZIONI TRA MATRICI

Dato un grafo $G=(V,E)$ con $V=\{1,2,3,\dots,n\}$ e una funzione peso $\omega:E \rightarrow \mathbb{R}$ ammettendo archi con peso negativo ma non cicli negativi allora possiamo rappresentare il grafo come una matrice $W(n \times n)$

$$W=(w_{ij}) \quad w_{ij} = \begin{cases} 0 & \text{se } i = j \text{ (la diagonale ha tutti valori } = 0) \\ \omega(i,j) & \text{se } i \neq j \text{ AND } (i,j) \in E \text{ (se esiste l'arco)} \\ \infty & \text{se } i \neq j \text{ AND } (i,j) \notin E \text{ (se non esiste l'arco)} \end{cases}$$

Dobbiamo restituire una matrice $D(n \times n)$ che rappresenti le distanze dei cammini

$$d_{ij} = \delta(i,j) \quad \forall i, j \in V$$

$$i, j \in V \quad \mathcal{C}_{ij} = \{ p = \langle x_0 x_1 \dots x_q \rangle \mid x_0 = i, x_q = j, (x_{i-1}, x_i) \in E \} = \{ p \mid p \text{ è cammino tra } i \text{ e } j \}$$

$$\delta(i, j) = \min_{p \in \mathcal{C}_{ij}} \omega(p)$$



Se vogliamo avere non più di m archi : $m \geq 1$

$C_{ij}^{(m)} = \{ p = \langle x_0 x_1 \dots x_q \rangle \mid x_0 = i, x_q = j, (x_{i-1}, x_i) \in E, q \leq m \} = \{ p \mid p \text{ è il cammino tra } i \text{ e } j \text{ con non più di } m \text{ archi} \}$

$$d_{ij}^m = \min_{p \in C_{ij}^m} \omega(p)$$

$d_{ij}^{n-1} = \delta(i, j)$ che è l'obiettivo della ricorsione degli algoritmi che vedremo

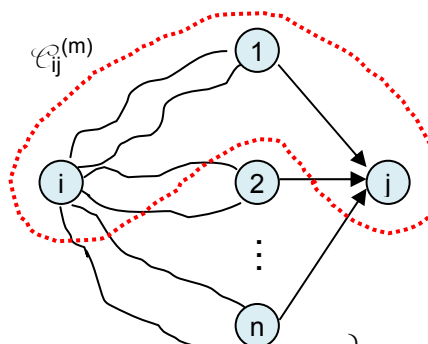
$D^{(m)} = (d_{ij}^{(m)}) \quad D^{(m)} = D \text{ (matrice delle distanze)} \quad \forall m \geq n-1$

$D^{(1)} = W$

$D^{(2)} = D^{(1)} \otimes W = W^2$

$D^{(3)} = D^{(2)} \otimes W = (W \otimes W) \otimes W = W^3$

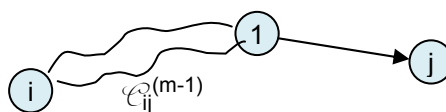
$D^{(n-1)} = D^{(n-2)} \otimes W$



se $m=1 \quad D^{(1)} = W$

se $m \geq 2 \quad d_{ij}^m = \min_{p \in C_{ij}^m} \omega(p) = \min \left\{ \min_{p \in C_{ij}^{(m)}(1)} \omega(p), \min_{p \in C_{ij}^{(m)}(2)} \omega(p), \dots, \min_{p \in C_{ij}^{(m)}(n)} \omega(p) \right\}$

il cammino minimo è il minimo dei cammini che passano per 1, 2, ..., n



$$\min_{p \in C_{ij}^{(m)}(1)} \omega(p) = \min_{p \in C_{ij}^{(m-1)}} \omega(p) + w_{ij} = d_{ij}^{(m-1)} + w_{ij} \quad \text{abbiamo trovato la ricorsione}$$

quindi: $d_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ d_{ik}^{(m-1)} + w_{kj} \}$ che porteremo in forma algoritmica $D^{(m)} = f(D^{(m-1)}, W)$

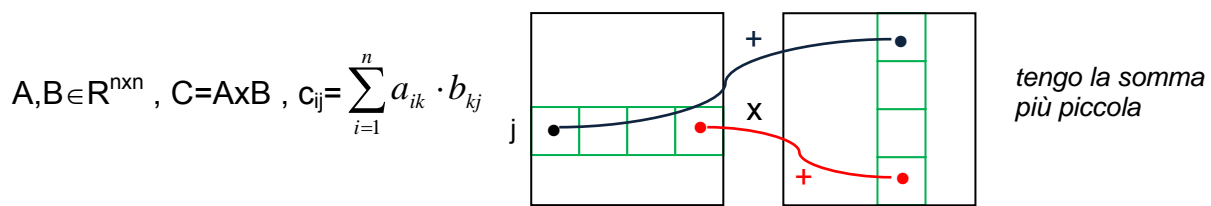
EXTENDED_SHORTEST_PATH(D,V)

- 1) $n \leftarrow \text{rows}[D]$
- 2) sia D' matrice $n \times n$
- 3) for $i \leftarrow 1$ to n do
- 4) for $j \leftarrow 1$ to n do
- 5) $d'_{ij} \leftarrow \infty$
- 6) for $k \leftarrow 1$ to n do
- 7) $d'_{ij} \leftarrow \min\{d'_{ij}, d_{ik} + w_{kj}\}$
- 8) return D'

Complessità: $O(n^3)$



Vediamo in dettaglio la formula $d_{ij}^{(m)} = \min_{1 \leq k \leq n} \{d_{ik}^{(m-1)} + w_{kj}\}$



MATRIX_MULTIPLY(D,W)

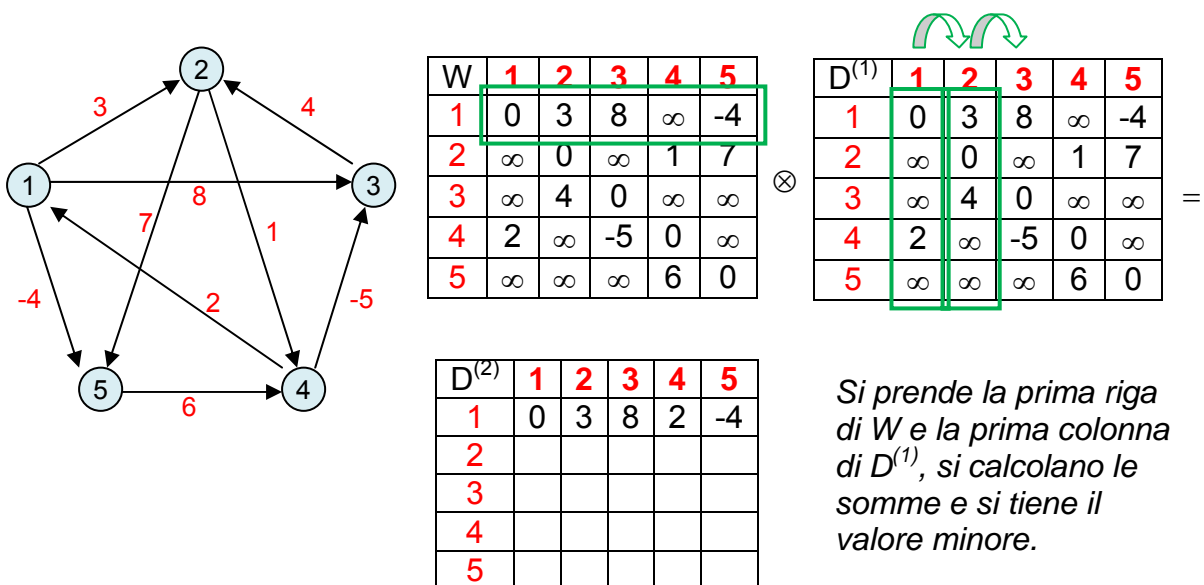
- 1) $n \leftarrow \text{rows}[D]$
- 2) sia D' matrice $n \times n$
- 3) for $i \leftarrow 1$ to n do
- 4) for $j \leftarrow 1$ to n do
- 5) $d'_{ij} \leftarrow 0$
- 6) for $k \leftarrow 1$ to n do
- 7) $d'_{ij} \leftarrow d_{ij}, d'_{ik} + w_{kj}$
- 8) return D'

Complessità: $O(n^3)$

SLOWEST_ALL_PAIRS_SHORTEST_PATH(W)

- 1) $n \leftarrow \text{rows}[D]$
- 2) $D^{(1)} \leftarrow W$
- 3) for $m \leftarrow 2$ to $n-1$ do
- 4) $D^{(m)} \leftarrow \text{EXTENDED_SHORTEST_PATH}(D^{(m-1)}, W)$
- 5) return $D^{(n-1)}$

Complessità: $O(n \cdot n^3) = O(n^4)$



METODO DELLA QUADRATURA RIPETUTA

Abbiamo visto che $\forall m \geq n-1 \ D^{(m)} = D$, quindi possiamo calcolare la matrice con $\forall m \geq n-1$

$$0 \quad D^{(1)} = W$$

$$1 \quad D^{(2)} = D^{(1)} \otimes W = W^2$$

$$3 \quad D^{(4)} = D^{(3)} \otimes W = (W \otimes W \otimes W) \otimes W = (W \otimes W) \otimes (W \otimes W) \text{ per l'associatività, quindi:}$$

$$D^{(4)} = D^{(2)} \otimes D^{(2)}$$

$$4 \quad D^{(8)} = D^{(4)} \otimes D^{(4)}$$

...

$$k \quad D^{(2^k)} = D^{(2^{k-1})} \otimes D^{(2^{k-1})}$$

Procedendo in modo esponenziale troviamo k s' $2^k \geq n-1 \Rightarrow k \geq \log_2(n-1) \Rightarrow k = \lceil \log_2(n-1) \rceil$

In questo modo nei passi successivi la matrice sarà sempre la stessa.

FASTER_ALL_PAIRS_SHORTEST_PATH(W)

1) $n \leftarrow \text{rows}[D]$

2) $Q \leftarrow W$

3) $m \leftarrow 1$

4) while $m < n-1$ do

5) $P \leftarrow Q$

6) $Q \leftarrow \text{EXTENDED_SHORTEST_PATH}(P, P)$

7) $m \leftarrow 2m$

8) return Q

Complessità: $O(n^3 \log n)$



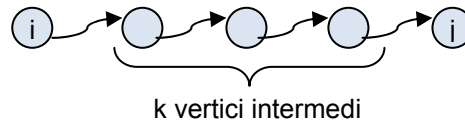
ALGORITMO DI FLOYD-WARSHALL

Dato un insieme di vertici numerati di un grafo $G=(V,E)$, $V=\{1,2,\dots,n\}$ e dati due vertici $i,j \in V$ definiamo

$\mathcal{D}_{ij}^{(k)} = \{ p = \langle x_0 \dots x_q \rangle \mid x_0 = i, x_q = j, (x_{l-1}, x_l) \in E, \forall l = 1 \dots q-1 : x_l \leq k \} = \{ p \mid p \text{ è un cammino i cui vertici intermedi non superino } k \}$

se $k=n$

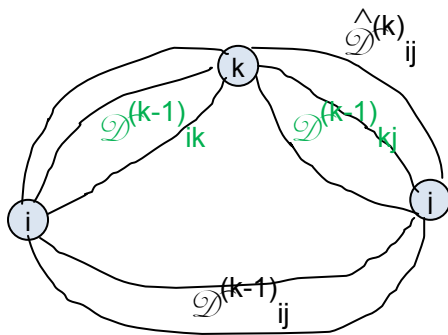
$$\mathcal{D}_{ij}^{(k)} = \mathcal{D}_{ij}^{(n)} = \mathcal{C}_{ij}$$



Quindi per un k fissato definiamo $d_{ij}^{(k)} = \min_{p \in \mathcal{D}_{ij}^{(k)}} \omega(p)$ e $\delta(i, j) = d_{ij}^{(n)}$

Si può dividere il grafo in due parti: quella dei cammini che passano per il vertice k e quella dei cammini che invece non passano per k

$$\hat{\mathcal{D}}_{ij}^{(k)} = \{ p \in \mathcal{D}_{ij}^{(k)} \mid p \text{ passa per il vertice } k \} \Rightarrow \mathcal{D}_{ij}^{(k)} = \hat{\mathcal{D}}_{ij}^{(k)} \cup \mathcal{D}_{ij}^{(k-1)}$$



Non avendo cicli, k può apparire solo una volta in ogni cammino, quindi il cammino da i a j può essere diviso in i, k e $k, j \Rightarrow \mathcal{D}_{ik}^{(k-1)}$ e $\mathcal{D}_{kj}^{(k-1)}$

$$\begin{aligned} \text{Quindi come detto prima } d_{ij}^{(k)} = \min_{p \in \mathcal{D}_{ij}^{(k)}} \omega(p) &= \min \left\{ \overbrace{\min_{p \in \mathcal{D}_{ij}^{(k-1)}} \omega(p)}^{=d_{ij}^{(k-1)}}, \min_{p \in \hat{\mathcal{D}}_{ij}^{(k)}} \omega(p) \right\} \\ &= \underbrace{\min_{p \in \mathcal{D}_{ik}^{(k-1)}} \omega(p)}_{=d_{ik}^{(k-1)}} + \underbrace{\min_{p \in \mathcal{D}_{kj}^{(k-1)}} \omega(p)}_{=d_{kj}^{(k-1)}} \end{aligned}$$

Unendo le scomposizioni otteniamo che:

$$d_{ij}^{(k)} = \min \{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \}$$

che è la formula che si userà nell'algoritmo di Floyd-Warshall

se $K=0$ $d_{ij}^{(0)} = w_{ij}$ $D^{(0)} = W$



FLOYD-WARSHALL(W)

- 1) $n \leftarrow \text{rows}[W]$
- 2) $D^{(0)} \leftarrow W$
- 3) for $k \leftarrow 1$ to n do
- 4) for $i \leftarrow 1$ to n do
- 5) for $j \leftarrow 1$ to n do
- 6) $d^{(k)}_{ij} = \min \{ d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \}$
- 7) return $D^{(n)}$

Complessità: sia temporale che spaziale $O(n^3)$

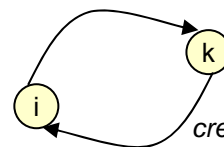
• Osservazioni

- a) $\forall k \geq 0 : d^{(k)}_{ii} = 0 \quad \forall i=1 \dots n$

Dim. per induzione:

base) $k=0 \quad D^{(0)}=W$

$$\text{ind) } d^{(k)}_{ii} = \min \{ \underbrace{d^{(k-1)}_{ii}}_{=0}, \underbrace{d^{(k-1)}_{ik} + d^{(k-1)}_{ki}}_{\geq 0} \} = 0$$



creerei un ciclo di peso negativo, che non è ammesso

- b) $\forall k, i, j \in V :$

b.1) $d^{(k)}_{ik} = d^{(k-1)}_{ik}$

b.2) $d^{(k)}_{kj} = d^{(k-1)}_{kj}$

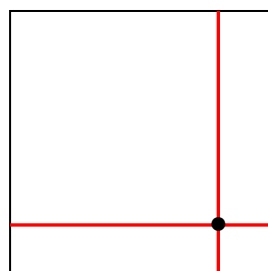
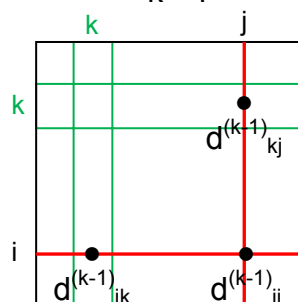
Dim. b.1 :

$$d^{(k)}_{ik} = \min \{ d^{(k-1)}_{ik}, d^{(k-1)}_{ik} + \underbrace{d^{(k-1)}_{kk}}_{=0} \} = d^{(k-1)}_{ik}$$



*durante i calcoli la riga e la colonna k restano **invariate***

Quindi supponiamo di trovarci al passo $k-1$ e voler calcolare k



= minimo tra il numero esistente e la somma dei corrispondenti sulla riga e colonna k-esima

L'esecuzione dell'algoritmo permette di sovrascrivere la matrice originale, dato che riga e colonna k non cambiano. Questo permette di modificare l'algoritmo in modo da creare un risparmio di spazio, utilizzando sempre la stessa matrice invece di due distinte.

FLOYD-WARSHALL(W)

- 1) $n \leftarrow \text{rows}[W]$
- 2) $D^{(0)} \leftarrow W$
- 3) for $k \leftarrow 1$ to n do
- 4) for $i \leftarrow 1$ to n do
- 5) for $j \leftarrow 1$ to n do
- 6) $d_{ij} = \min \{ d_{ij}, d_{ik} + d_{kj} \}$
- 7) return $D^{(n)}$

Complessità temporale: $O(n^3)$

Complessità spaziale: $O(n^2)$



Esempio di esecuzione di Floyd-Warshall usando la matrice precedente:

W	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

 \Rightarrow

W	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

 \Rightarrow

W	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

 \Rightarrow

W	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

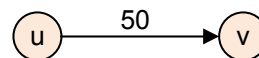
 \Rightarrow

W	1	2	3	4	5
1	0	3	8	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	2
5	8	5	1	6	0

RETI DI FLUSSO

Una **rete** $G=(V,E)$ è un grafo orientato, con associato a ciascuna coppia di vertici un numero reale che ne rappresenta la capacità, che sarà sempre ≥ 0

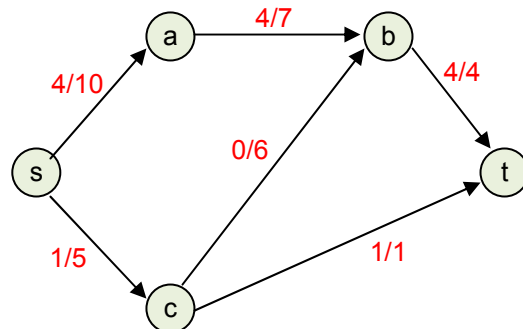
$$c: V \times V \rightarrow \mathbb{R}_T \quad c(u,v)=0 \Leftrightarrow (u,v) \notin E$$



s = sorgente

t = pozzo (destinazione)

Si suppone che $\forall u \in V \setminus \{s,t\}$ esista un cammino tra s e t che passa per u.
Nell'esempio il flusso massimo è di 5 unità.



Definizione di FLUSSO

Data una rete di flusso $G=(V,E)$, $c: V \times V \rightarrow \mathbb{R}_+$, $(s,t \in V)$

un flusso è una funzione $f: V \times V \rightarrow \mathbb{R}$ che rispetta le seguenti proprietà:

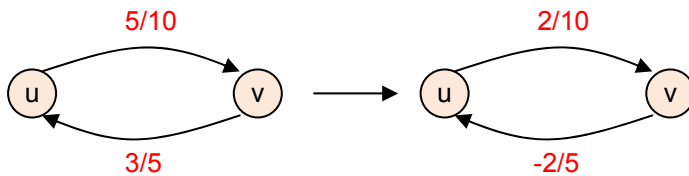
- 1) **Vincolo di capacità:** $\forall u,v \in V : f(u,v) \leq c(u,v)$ [il flusso è sempre minore della capacità su quel determinato arco]
- 2) **Vincolo di antisimmetria:** $\forall u,v \in V : f(u,v) = -f(v,u)$ [$f(u,v)$ =flusso netto tra u e v]
- 3) **Vincolo di conservazione del flusso:** $\forall v \in V \setminus \{s,t\} : \sum_{u \in V} f(u,v) = 0$



- Dato un flusso $f: V \times V \rightarrow \mathbb{R}$ il “**valore**” di f è dato dalla somma dei flussi uscenti dalla sorgente

$$|f| = \sum_{u \in V} f(s, u)$$

- $f(u, v)$ = **flusso netto** tra u e v



i valori rappresentano al netto la differenza tra ciò che entra e ciò che esce.

$u \in V \setminus \{s, t\}$ definiamo:

- il flusso netto positivo **uscente** da u come $\sum_{u \in V} f(u, v)$ con $f(u, v) > 0$ **FNPU(u)**
- il flusso netto positivo **entrante** in u come $\sum_{u \in V} f(v, u)$ con $f(v, u) > 0$ **FNPE(u)**

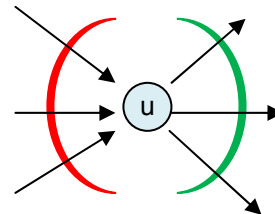
$$\text{FNPU}(u) = \text{FNPE}(u) \text{ [verifica la 3^a proprietà di prima]}$$

Dimostrazione:

prendiamo un vertice qualsiasi $u \in V \setminus \{s, t\}$, $0 = \sum_{u \in V} f(u, v)$ che può essere anche scritto così:

$$\begin{aligned} 0 &= \sum_{\substack{u \in V \\ f(u, v) < 0}} f(u, v) + \sum_{\substack{u \in V \\ f(u, v) > 0}} f(u, v) && \text{e quindi} \\ &= \sum_{\substack{u \in V \\ f(u, v) > 0}} f(u, v) = \sum_{\substack{u \in V \\ f(u, v) < 0}} -f(u, v) \\ &= \sum_{\substack{u \in V \\ f(u, v) > 0}} f(u, v) = \sum_{\substack{u \in V \\ f(u, v) > 0}} f(v, u) \end{aligned}$$

FNPU **FNPE**



• Osservazioni:

1) $\forall u \in V : f(u, u) = 0$ infatti per l'asimmetria $f(u, u) = -f(u, u) \Leftrightarrow f(u, u) = 0$

2) $\forall u, v \in V : (u, v) \notin E \Rightarrow f(u, v) = 0 = f(v, u)$

$$\left. \begin{array}{l} (u, v) \notin E \Rightarrow c(u, v) = 0 \\ (v, u) \notin E \Rightarrow c(v, u) = 0 \end{array} \right\} \begin{array}{l} f(u, v) \leq c(u, v) \\ f(v, u) \leq c(v, u) \end{array} \left. \begin{array}{l} f(u, v) \leq 0, f(v, u) \leq 0 \\ = -f(u, v) \end{array} \right\}$$

Quindi $f(u, v)$ dev'essere contemporaneamente ≤ 0 e ≥ 0 , quindi è $= 0$

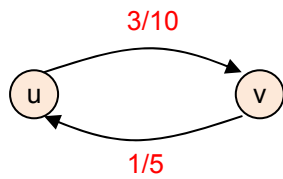
3) $\underbrace{f(u, v)} < 0 \Rightarrow (v, u) \in E$

$f(v, u) > 0$ se per assurdo $(v, u) \notin E$ [$c(v, u) = 0$] avremmo affermato che $f(v, u) > c(v, u)$



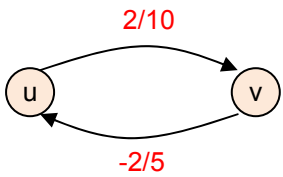
ATTENZIONE

In un caso come questo:

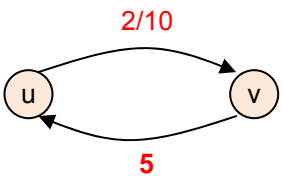


$f(u,v)$ che è il flusso netto non è 3, ma 2 ($= 3(\text{spediti}) - 1(\text{ricevuti})$)
 $f(v,u)$ non è 1 ma -2 ($= 1 - 3$)

Quindi la situazione precedente equivale a:

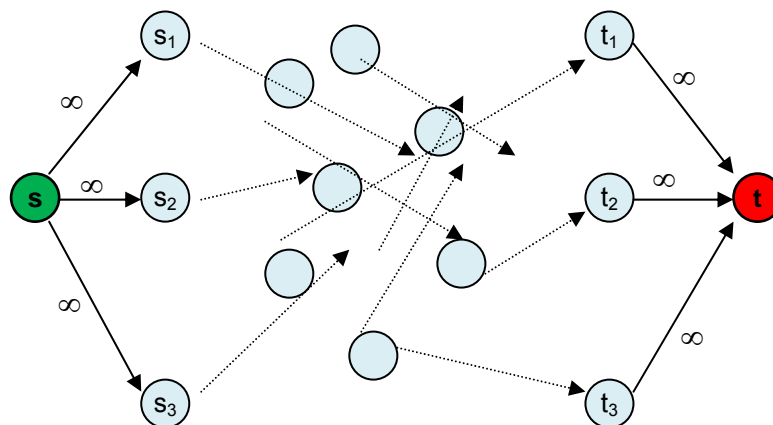


ma siccome i flussi negativi non vengono rappresentati (in quanto vengono rappresentati solo gli effettivi invii sugli archi), allora questa rete equivale a questa:



infatti in caso di flussi nulli o negativi viene solo riportata sull'arco la sua capacità

Può succedere che una rete abbia diverse sorgenti e diversi pozzi, in questo caso è sufficiente aggiungere una metasorgente e un metapozzo con capacità infinita, riconducendosi così a un problema con sorgente e pozzo singoli.



SOMMATORIA IMPLICITA

Supponiamo che X e Y siano due sottoinsiemi di vertici ($X, Y \subseteq V$), dato un flusso $f: V \times V \rightarrow \mathbb{R}$

$$f(X, Y) \triangleq \sum_{u \in X} \sum_{v \in Y} f(u, v)$$

Abbiamo quindi esteso la definizione di flusso agli insiemi

Proprietà:

- 1) prendendo un sottoinsieme X di vertici ($X \subseteq V$) $\Rightarrow f(X, X) = 0$

Dimostrazione:

$$f(X, X) = \sum_{u \in X} \sum_{v \in Y} f(u, v) = 0$$

il numero di addendi è $2|X|$

$$\dots + f(u, v) + \dots + f(v, u) + \dots$$

$- f(u, v)$ per antisimmetria

- 2) prendendo $X, Y \subseteq V \Rightarrow f(X, Y) = -f(Y, X)$ (asimmetria generalizzata)

Dimostrazione:

$$f(X, Y) = \sum_{u \in X} \sum_{v \in Y} f(u, v) = \sum_{u \in X} \sum_{v \in Y} -f(v, u) = - \sum_{u \in X} \sum_{v \in Y} f(v, u) = -f(Y, X)$$

- 3) presi $X, Y, Z \subseteq V$, $X \cap Y = \emptyset$ (devono essere disgiunti) allora

$$3.1. f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$

$$3.2. f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$$

Dimostrazione:

$$\begin{aligned} f(X \cup Y, Z) &= \sum_{x \in X \cup Y} \sum_{z \in Z} f(x, z) = \\ &= \sum_{x \in X} \sum_{z \in Z} f(x, z) + \sum_{x \in Y} \sum_{z \in Z} f(x, z) = \\ &= f(X, Z) + f(Y, Z) \end{aligned}$$

In una rete di flusso la quantità di flusso uscente dalla sorgente è uguale a quella entrante nel pozzo, e sono entrambe uguali al valore del flusso.

$$\sum_{u \in V} f(s, u) = \sum_{u \in V} f(u, t) \\ [f(s, V) = f(V, t)]$$

Dimostrazione:

$$\begin{aligned} |f| &= f(s, V) = f(V, V) - f(V \setminus \{s\}, V) & [f(V, V) = 0, -f(V \setminus \{s\}, V) = f(V, V \setminus \{s\})] \\ &= f(V, V \setminus \{s\}) = f(V, t) + f(V, V \setminus \{s, t\}) \end{aligned}$$

$$\text{quindi } f(s, V) = f(V, t)$$

$$\begin{aligned} f(V, V) &= f(\underbrace{V \setminus \{s\}}_X \cup \underbrace{\{s\}}_Y, \underbrace{V}_Z) \\ &= f(\underbrace{V \setminus \{s\}}_X, \underbrace{V}_Z) + f(\underbrace{s}_Y, \underbrace{V}_Z) \end{aligned}$$



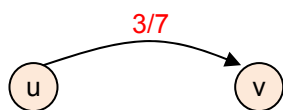
METODO DI FORD-FULKERSON

Per il seguente argomento ci si basa su tre concetti di base, ovvero:

- 1) Rete residua
- 2) Cammino aumentante
- 3) Taglio

• RETE RESIDUA

Siano dati $G=(V,E)$ rete di flusso, f flusso nella rete e sia che $\forall u,v \in V \Rightarrow f(u,v) \leq c(u,v)$.



Se su di un arco il flusso è minore della capacità, allora l'arco non è saturo, quindi posso aumentarne il flusso

$\forall u,v \in V : c_f(u,v) = c(u,v) - f(u,v)$ dove c_f è la **capacità residua** e $c_f(u,v) \geq 0$

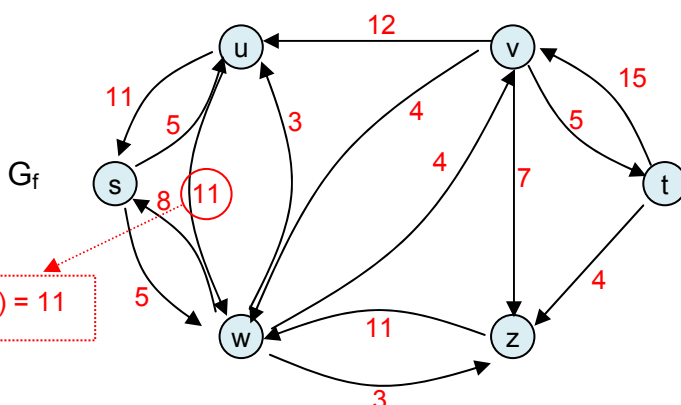
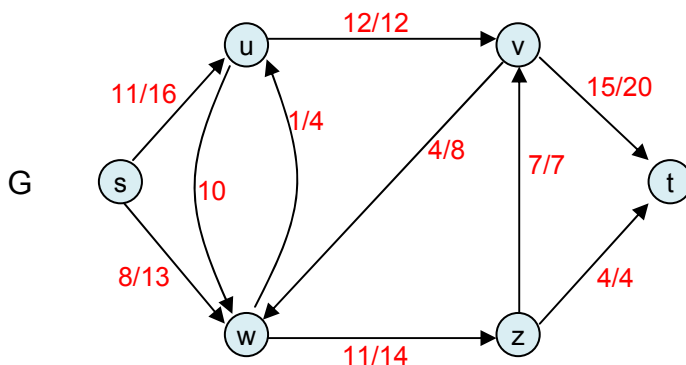
Quindi la **rete residua** di G rispetto ad f è una nuova rete con gli stessi vertici e nuovi archi rispetto a G .

$G_f = (V, E_f)$

*numero di archi
della rete residua*

$E_f = \{ (u,v) \in V \times V \mid c_f(u,v) > 0 \}$ non è detto che sia sempre $E_f \subseteq E$ ma in ogni caso $|E_f| \leq 2|E|$

Esempio:

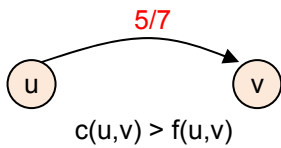


Per definizione:

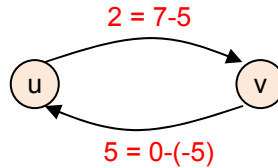
se tra due vertici non c'è nessun arco $\Rightarrow c(u,v) = f(u,v) = 0$

Nota:

se si ha



risulta



La rete residua G_f è anch'essa una rete di flusso, $G_f=(V,E_f)$ con capacità G_f

- **Proposizione:** siano $G=(V,E)$ una rete di flusso,
 f un flusso in G ,
 G_f la rete residua,
 f' un flusso sulla rete residua G_f

f
 f' : $\forall x,y \in V \rightarrow \mathbb{R}$ posso definire $f+f'$ perché le due funzioni flusso sono sullo stesso dominio
 $f + f' : \forall x,y \in V \rightarrow \mathbb{R}$

$$(f+f')(u,v) = f(u,v) + f'(u,v)$$

Allora $f+f'$ è un flusso su G , $|f+f'| = |f| + |f'|$, questo vale in quanto f' è flusso sulla rete residua G_f , infatti se f e f' sono due flussi entrambi in G non è possibile che $f + f'$ sia ancora flusso, in quanto verrebbe violato il vincolo di capacità.

Dimostrazione – devono valere i tre vincoli:

a) *Vincolo di capacità*

$$\forall u,v \in E : (f+f')(u,v) \leq c(u,v) ?$$

$$(f + f')(u,v) = f(u,v) + f'(u,v)$$

$$\leq f(u,v) + [c(u,v) - f(u,v)]$$

$$(f + f')(u,v) \leq c(u,v)$$

In quanto:

$$f'(u,v) \leq c_f(u,v)$$

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

b) *Vincolo di antisimmetria*

$$\forall u,v \in E : (f+f')(u,v) = - (f+f')(v,u) ?$$

$$(f+f')(u,v) = f(u,v) + f'(u,v)$$

$$= - [f(v,u) + f'(v,u)] = - (f+f')(v,u)$$

c) *Vincolo di conservazione del flusso*

$$\forall v \in V \setminus \{s,t\} : \sum_{u \in V} (f+f')(u,v) = 0 ?$$

$$\sum_{u \in V} (f+f')(u,v) = \sum_{u \in V} \overset{0}{f(u,v)} + \sum_{u \in V} \overset{0}{f'(u,v)} = 0$$

d) *Valore*

$$|f+f'| = \sum_{v \in V} (f+f')(s,v) ?$$



$$\sum_{v \in V} f(s,v) + \sum_{v \in V} f'(s,v) = |f| + |f'|$$

- CAMMINO AUMENTANTE**

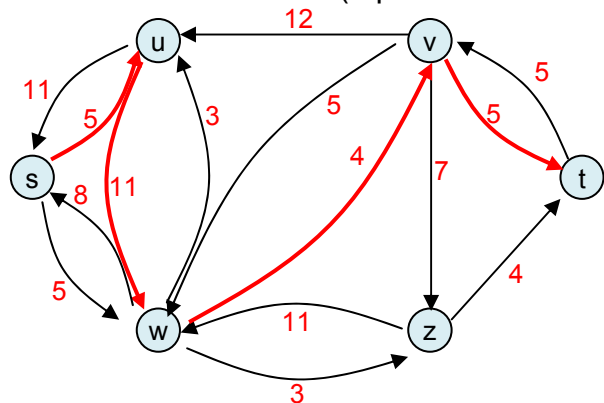
Siano $G=(V,E)$ rete, f un flusso e $G_f=(V,E_f)$ una rete residua, un cammino aumentante è un cammino semplice tra la sorgente "s" e il pozzo "t" nella rete residua (è possibile anche che questo cammino non ci sia)

Se p non esiste allora f è il flusso massimo.

- Capacità residua su p** (con p cammino aumentante):

Indicheremo con $c_f(p)$ la più piccola delle capacità che si trova sul cammino.

$C_f(p) = \min \{ c_f(u,v) \mid (u,v) \text{ è arco di } p \}$ (nella figura precedente = 4)



- Flusso sulla capacità residua**

$f_p : V \times V \rightarrow \mathbb{R}$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{se } (u,v) \text{ è arco di } p \\ -c_f(p) & \text{se } (v,u) \text{ è arco di } p \\ 0 & \text{altrimenti (assegna a tutti gli archi che } \nexists \text{ al cammino il valore 0)} \end{cases}$$

- Proposizione**

Siano $G=(V,E)$ rete di flusso, f un flusso in G , G_f la rete residua e p un cammino aumentante in G_f , allora f_p è un flusso su G_f e il suo valore coincide con $c_f(p)$, cioè

$$|f_p| = c_f(p) > 0$$

- Corollario** (per costruire un flusso minore)

Se si ha una rete G e un flusso f si può costruire una rete residua G_f e un flusso f_p (dove p è un cammino aumentante). In questo caso si avrà che:

$$|f+f_p| = |f| + |f_p| > |f|$$

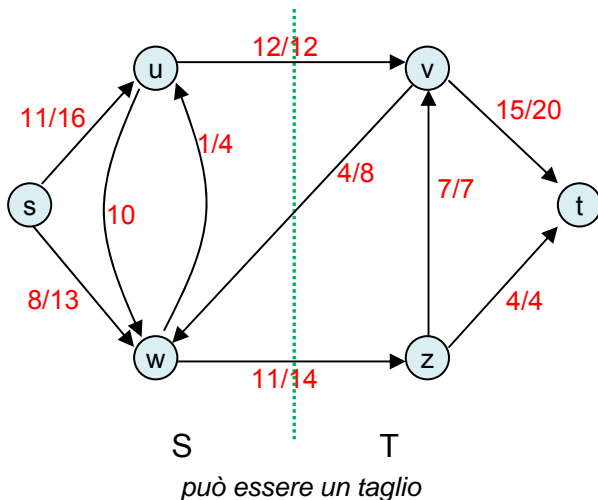


• TAGLIO

Un taglio è una coppia (S,T) dove $\{ S \cup T = V \text{ [} T = V \setminus S \text{]} \text{ e } S \cap T = \emptyset \}$, in più $s \in S$ e $t \in T$.

La capacità del taglio è : $c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$

Il flusso del taglio è : $f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v)$



$$c(S,T) = 12 + 14 = 26$$

$$f(S,T) = 12 + 11 - 4 = 19$$

devo prendere solo gli archi che attraversano il taglio

* S : T \rightarrow +

* T : S \rightarrow -

Cioè: se da S vado a T uso il segno +, mentre se da T vado a S uso - .

• Proposizione

Siano $G=(V,E)$ una rete di flusso, f un flusso in G e (S,T) un taglio su G , per qualsiasi taglio in G vale che $|f| = f(S,T)$

Dimostrazione:

$$f(S,T) = f(S,V) - F(S,S)$$

$$= f(S,V)$$

$$= f(s,V) + f(S \setminus \{s\}, V)$$

$$= |f|$$

Per il vincolo di conservazione del flusso.

$$= 0 \rightarrow f(S \setminus \{s\}, V) = \sum_{\substack{u \in S \\ u \neq s \\ u \neq t}} \left(\sum_{v \in V} f(u,v) \right)$$

Il pozzo non appartiene a S

• Proprietà:

f flusso, G rete di flusso, (S,T) taglio su $G \rightarrow |f| = f(S,T) \leq c(S,T)$

Dimostrazione:

$$|f| = f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) \leq \sum_{u \in S} \sum_{v \in T} c(u,v) = c(S,T)$$

Problema del taglio minimo

Dove il valore della capacità sia minimo il valore associate al taglio è $c(S,T)$



TEOREMA DEL TAGLIO MINIMO/FLUSSO MASSIMO

Sia f un flusso, $G=(V,E)$ una rete di flusso con sorgente "s" e pozzo "t", allora le seguenti affermazioni sono equivalenti:

- a) f è un flusso massimo
- b) non esistono cammini aumentanti in G_f (rete residua)
- c) $\exists (S,T)$ taglio in G tale che $|f| = c(S,T)$, ovvero il valore del flusso coincide con la capacità del taglio

Dimostrazione:

Dim. (a) \Rightarrow (b)

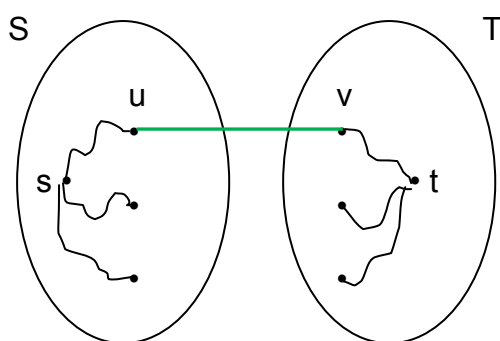
Suppongo per assurdo che esista un cammino aumentante in $G_f(p)$, posso quindi costruire un flusso $|f_p| > 0$, quindi: $|f+f_p| > |f|$ che è assurdo in quanto f è flusso massimo.

Dim. (b) \Rightarrow (c)

Costruisco $S=\{v \in V \mid v \text{ è raggiungibile da } s \text{ in } G_f\}$, con $s \in S$ e $t \notin S$

Costruisco quindi $T=V \setminus S \Rightarrow t \in T$

Siccome (S,T) è un taglio, allora $|f| = f(S,T) \leq c(S,T)$



$\forall u \in S, \forall v \in T$

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

se $f(u,v) < c(u,v)$ capacità residua > 0 , quindi esiste un arco tra u e v , quindi v è raggiungibile da s .

Questo è impossibile per costruzione perché $S \cap T = \emptyset$

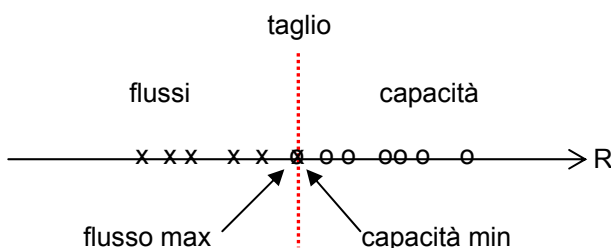
$$|f| = f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) = \sum_{u \in S} \sum_{v \in T} c(u,v) = c(S,T)$$

Dim. (c) \Rightarrow (a)

Esiste un taglio la cui capacità coincide col flusso.

Se per assurdo esistesse un flusso $> f$ andrebbe oltre la capacità minima, il che è assurdo in quanto tutti i flussi sono a sinistra del flusso massimo, e tutte le capacità sono a destra della capacità minima.

Tutti i flussi sono a sinistra della capacità minima, solo il flusso massimo coincide con la capacità minima.



ALGORITMO DI FORD-FULKERSON

Inizializza il flusso f a 0
 finchè esiste un cammino aumentante p in G_f
 $f \leftarrow f + f_p$

FORD-FULKERSON(G, s, t)

- 1) for each $(u, v) \in E$ do
- 2) $f[u, v] \leftarrow 0$ //il flusso è un array
- 3) $f[v, u] \leftarrow 0$
- 4) while \exists cammino aumentante p in G_f do
- 5) $c_f(p) \leftarrow \min\{ c_f(u, v) \mid (u, v) \text{ è arco in } p \}$
- 6) for each (u, v) in p do
- 7) $f[u, v] \leftarrow f[u, v] + c_f(p)$
- 8) $f[v, u] \leftarrow -f[u, v]$

Complessità:

$n=|V|$ $m=|E|$

$O(n)$

$O(1)$

$O(1)$

$-m$: per la visita è sempre $O(m)$

$O(1)$

$O(|f^*|)$

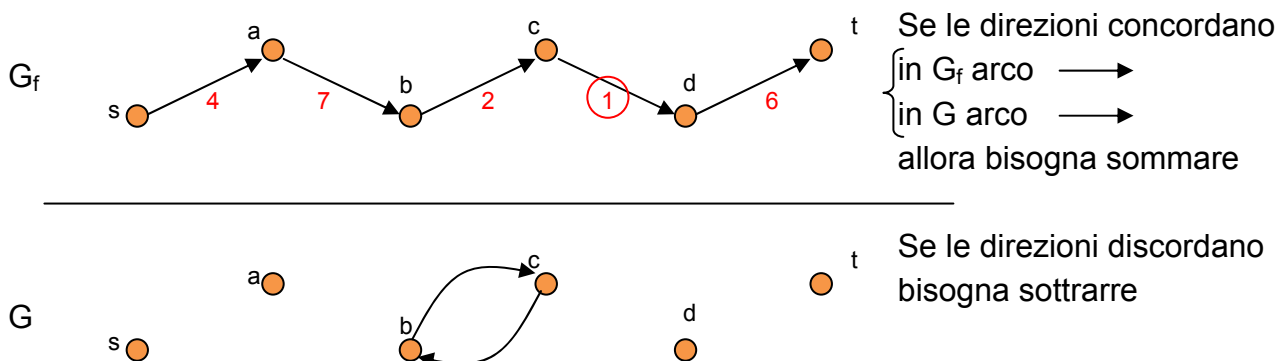
$O(1)$

$O(1)$

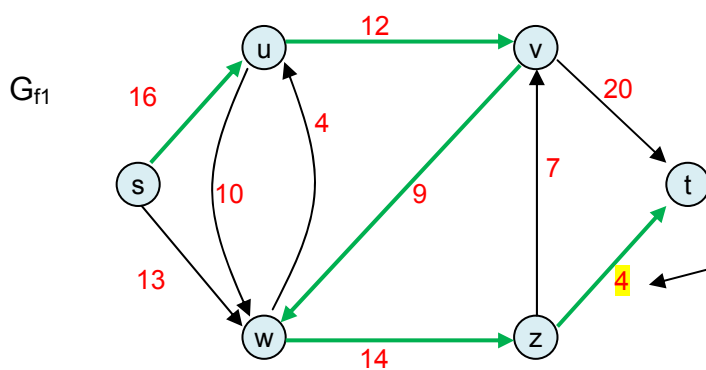
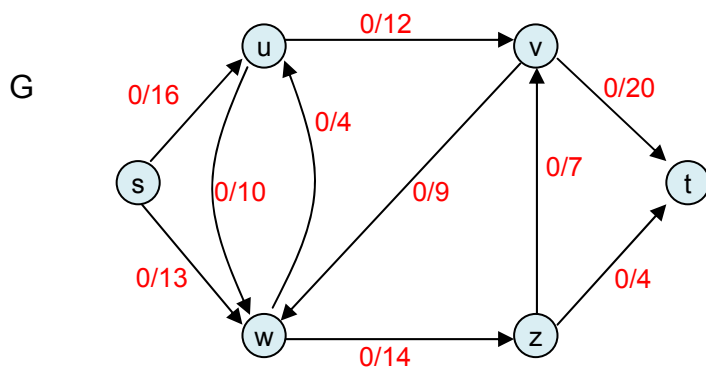
Complessità totale: $O(m \cdot |f^*|)$ dove f^* è il flusso ottimo

Appartiene alla categoria degli algoritmi PSEUDO-POLINOMIALI, in quanto dipende dal parametro f^* che se polinomiale rende l'algoritmo polinomiale, ma se fosse esponenziale (cosa possibile) renderebbe la complessità finale dell'algoritmo esponenziale.

- **Nota:** si ottiene un miglioramento continuo (sia che l'algoritmo lavori con numeri interi sia con numeri razionali), quindi l'algoritmo converge sempre e ci sarà un numero di cicli pari a $|f^*|$, dove f^* è il flusso ottimo.

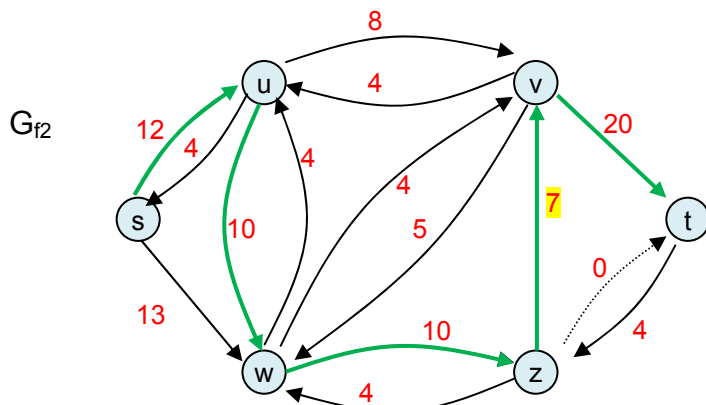
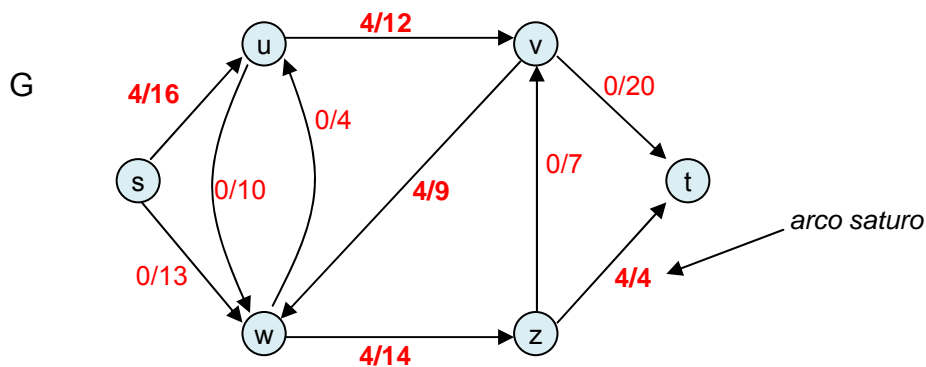


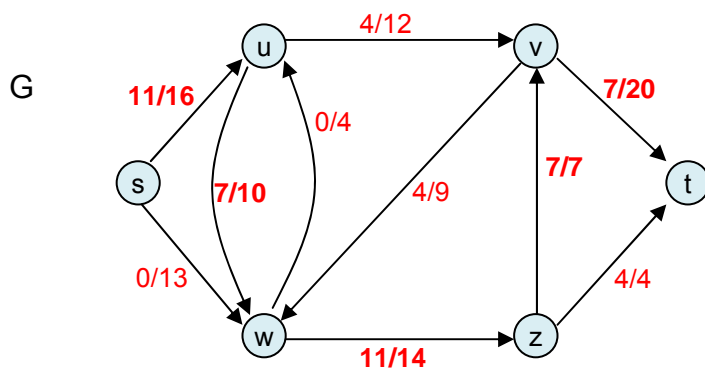
- Esempio di funzionamento:



RETE RESIDUA:

se i flussi di G sono tutti uguali a 0 allora la rete residua è uguale alla rete iniziale G
minimo

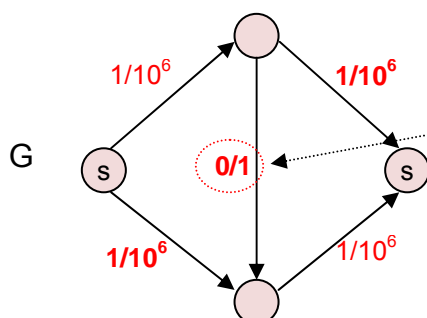
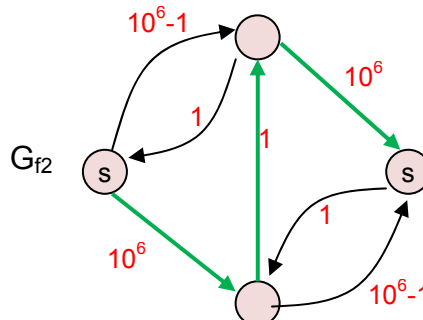
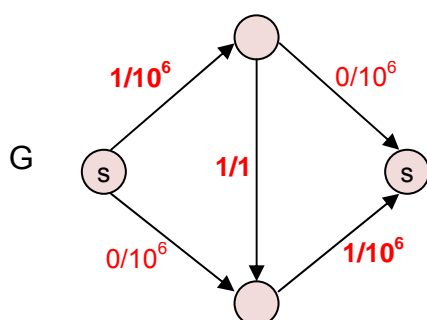
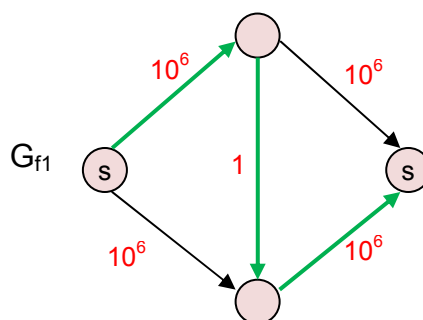
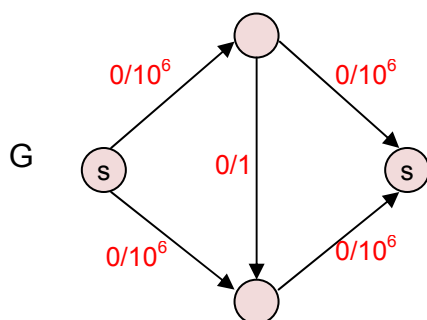




E si procede allo stesso modo in continuazione, finchè non si sarà costruita una rete residua dove non è più possibile costruire un cammino che va da “s” a “t”.

• Problema

Una cattiva scelta del cammino aumentante può tuttavia portare ad una complessità esponenziale



C'è un caso di discordanza, quindi sottraggo!
La cattiva scelta del cammino aumentante porta la complessità dell'algoritmo a $2 \cdot 10^6$, cioè esponenziale



Per migliorare l'algoritmo, al posto di scegliere un cammino aumentante qualsiasi, si sceglie il **cammino minimo** tra "s" e "t". Questo è infatti il funzionamento dell'algoritmo di Edmonds-Karp, che è uguale all'algoritmo di Ford-Fulkerson, ma sceglie il cammino aumentante minimo tra quelli esistenti.

EDMONDS-KARP(G, s, t)

- 1) for each $(u, v) \in E$ do
- 2) $f[u, v] \leftarrow 0$ //il flusso è un array
- 3) $f[v, u] \leftarrow 0$
- 4) while \exists cammino aumentante **minimo** p in G_f do
- 5) $c_f(p) \leftarrow \min\{c_f(u, v) \mid (u, v) \text{ è arco in } p\}$
- 6) for each (u, v) in p do
- 7) $f[u, v] \leftarrow f[u, v] + c_f(p)$
- 8) $f[v, u] \leftarrow -f[u, v]$

• Lemma:

Siano $G=(V, E)$ una rete di flusso, f un flusso in G , G_f la rete residua e p un cammino aumentante minimo (in relazione al numero di archi).

Siano poi $g=f+f_p$ un nuovo flusso e G_g la nuova rete residua rispetto a g .

Poniamo $\forall u$

$d_f(u)$ = distanza tra s e u in G_f

$d_g(u)$ = distanza tra s e u in G_g

Allora

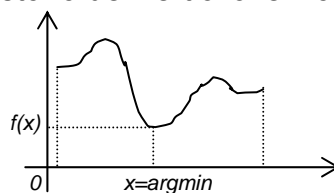
$$d_f(u) \leq d_g(u)$$

Dimostrazione:

$$F_{fg} = \{ \forall u \in V \mid d_f(u) > d_g(u) \} \quad \text{TESI: } F_{fg} = \emptyset$$

Per assurdo suppongo che $F_{fg} \neq \emptyset$, quindi esistono dei vertici che violano la proprietà $v = \operatorname{argmin}_{u \in F_{fg}} d_g(u)$

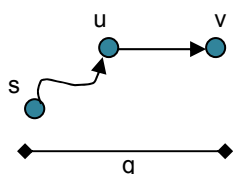
*vertice in corrispondenza
del quale si ottiene il minimo*



Per v devono valere le seguenti proprietà:

1. $d_f(v) > d_g(v)$ // v è un elemento di quell'insieme
2. $u \in F_{fg} \Rightarrow d_g(v) \leq d_g(u)$ // v è l'elemento più piccolo di quell'insieme
 - 2.1. $d_f(u) > d_g(u) \Rightarrow d_g(v) \leq d_g(u)$
 - 2.2. $d_g(v) > d_g(u) \Rightarrow d_f(u) \leq d_g(u)$ //si nega la proprietà precedente

$d_g(v) \in \mathbb{R}$ e in G_g esiste:



q : cammino minimo tra "s" e "v" in G_g

u : predecessore di v in q

quindi $d_g(v) = d_g(u) + 1 \Rightarrow d_g(u) < d_g(v)$

invocando quindi la proprietà 2.2 risulta $d_f(u) \leq d_g(u)$



Il fine della dimostrazione è quello di arrivare ad affermare che $c_f(u,v) < 0$, giungendo quindi ad un assurdo.

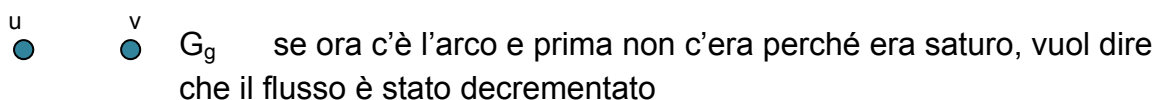
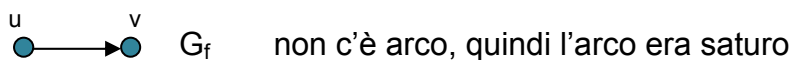
$$\begin{aligned} \text{a) } c_f(u,v) > 0 &\Leftrightarrow (u,v) \in E_f, \text{ quindi vale } d_f(v) \leq d_f(u) + 1 \\ &\leq d_g(u) + 1 \\ &\leq d_g(v) \end{aligned}$$

Assurdo affermare che $c_f(u,v) > 0$ perché contraddice la proprietà 1.

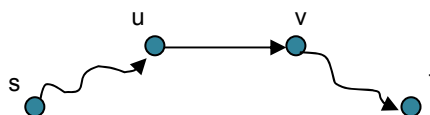
$$\text{b) } c_f(u,v) = 0$$

$$f(u,v) = c(u,v) \Leftrightarrow c_f(u,v) = 0$$

quindi non c'è arco: $(u,v) \notin E_f$, ma $(u,v) \in E_g$ per costruzione



quindi il cammino aumentante è costruito così:



Dato però che si tratta di un cammino minimo, si può affermare che:

$$\begin{aligned} d_f(v) &= d_f(u) - 1 \\ &= d_g(u) - 1 \\ &= d_g(v) - 2 < d_g(v) \end{aligned}$$

Assurdo affermare che $c_f(u,v) = 0$, perché si contraddice nuovamente la proprietà 1

Vale quindi $c_f(u,v) < 0$, ma questo è assurdo perché si contraddice l'ipotesi iniziale in cui si affermava che $F_{fg} \neq \emptyset$, quindi la capacità dell'arco non può essere < 0 .

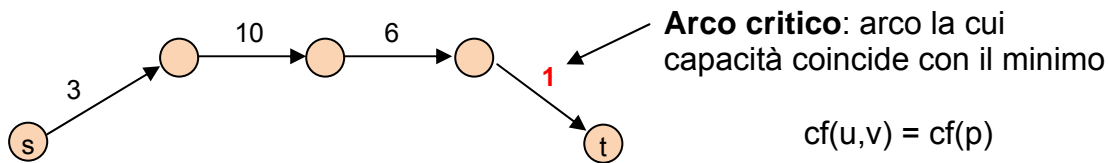
• Teorema

Se si utilizza l'algoritmo di Edmonds-Karp il numero di aumenti di flusso non può essere superiore a $O(nm)$ ($n=|V|$ e $m=|E|$) e nm è il numero di cicli massimi, ovvero $O(|f^*|)$.

Quindi l'algoritmo di **Edmonds-Karp ha una complessità pari a $O(m^2n)$**

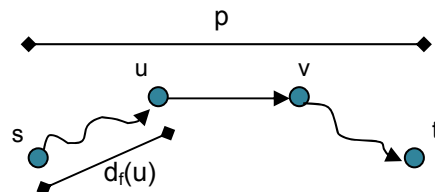


Dimostrazione:



$\forall p$ deve esistere almeno un arco critico.

Sia G_f la rete residua, (u,v) un arco critico in G_f . Dato che p è un cammino minimo, si può affermare subito che $d_f(v) = d_f(u) + 1$.
L'arco (u,v) viene immediatamente tolto dalla successiva rete residua.



Suppongo che in una successiva rete residua G_g ritorni l'arco (u,v) e che quindi ci sia stato un decremento del flusso (in corrispondenza della presenza dell'arco critico (u,v) nel cammino minimo). Si può quindi affermare che:

$$\begin{aligned} d_g(u) &= d_g(v) + 1 \\ &= d_f(u) + 1 \quad // \text{per il lemma precedente} \\ &= d_f(u) + 2 \end{aligned}$$

Se si utilizza l'algoritmo di Edmonds-Karp e si usa un arco critico questo scomparirà.

Se in una successiva esecuzione l'arco critico rientra nella rete residua allora deve valere:

$$d_g(u) = d_f(u) + 2$$

quindi il suo valore è più grande di almeno due unità.

• Osservazioni:

- Ogni volta che si genera una rete residua esiste un arco che è critico
- Un arco non può diventare critico per più di $n/2$ volte (quindi si hanno $2m$ archi)
- Il numero massimo di aumenti di flusso è $2m \cdot n/2 = mn$



ABBINAMENTO MASSIMO

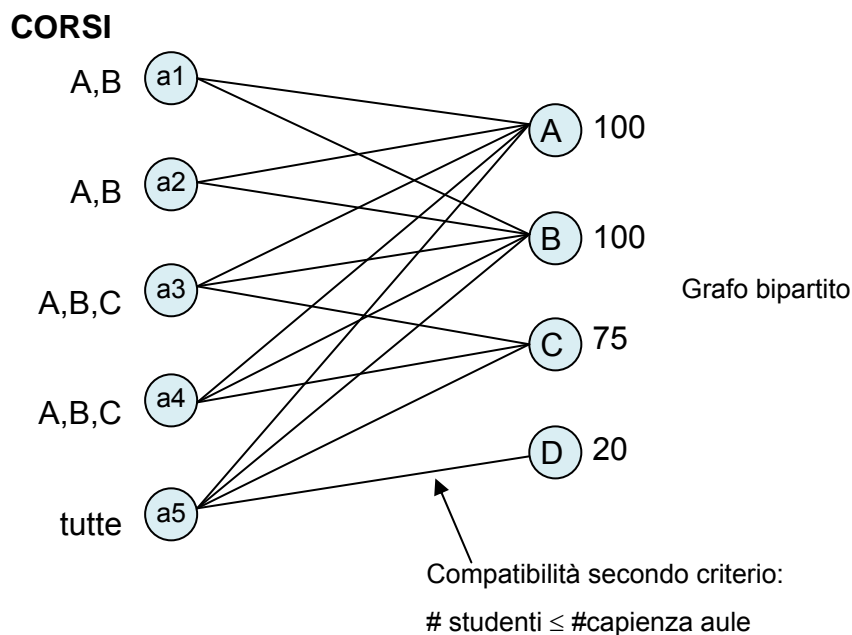
(maximum matching)

- Esempio:**

aule: A B C D
capienza: 100 100 75 20

5 corsi: a1 } ≈ 90 studenti 1° anno
 a2 }
 a3 } ≈ 70 studenti 2° anno
 a4 }
 a5 } ≤ 20 studenti 3° anno

Si determini un assegnamento di aule che massimizzi i corsi contemporanei



La proprietà del grafo bipartito è che dato $G=(V,E)$ bipartito dove $V=L \cup R$ [$L \cap R = \emptyset$] esiste $E' \subseteq E$ tale che non ci sono in E' due archi incidenti sullo stesso vertice, cioè ci sono archi solo tra L e R , non all'interno di L o di R .

Due archi si dicono **adiacenti** se sono incidenti allo stesso vertice.

- Abbinamento**

Dato un grafo $G=(V,E)$ non orientato, un abbinamento è un insieme di archi $E' \subseteq E$ tale che nessun arco in E' è adiacente ad un altro arco in E' (caratteristica fondamentale dei grafi bipartiti).



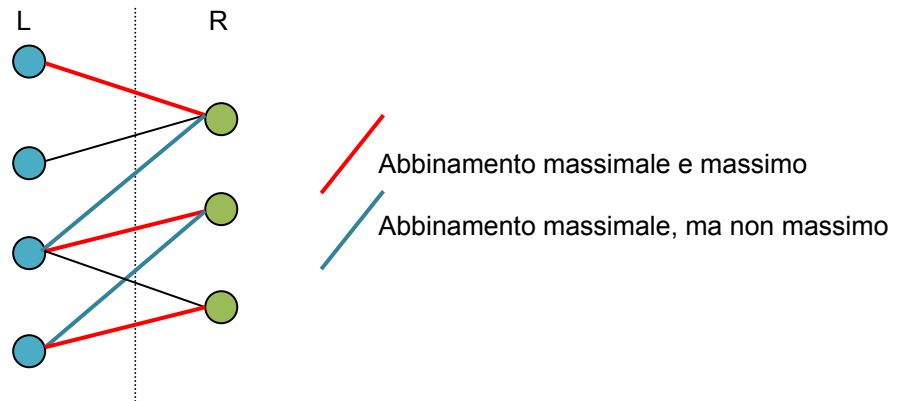
- **Abbinamento massimale**

Abbinamento E' che non è contenuto in un altro abbinamento più grande.

- **Abbinamento massimo**

Abbinamento di massima cardinalità.

Esempio:



- **Osservazione**

Se $G=(V,E)$ è un grafo bipartito allora M è un abbinamento massimo, e quindi

$$|M| \leq \min\{|L|, |R|\}$$

- **Riduzioni**

- 1) Riduzione che porta da un problema di abbinamento massimo a uno di **insieme indipendente massimo** (maximum independent set)
- 2) Riduzione che porta da un problema di abbinamento massimo a uno di **flusso massimo** (max flow)

Abbinamento massimo \rightarrow Insieme indipendente massimo

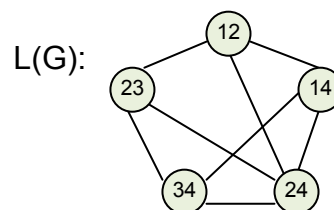
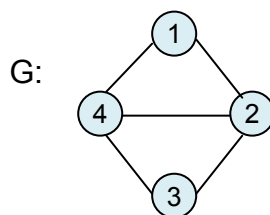
- **Definizione:** lim graph

Sia $G=(V,E)$ non orientato, si definisce lim graph (grafo degli archi) un grafo così costruito:

$G'=(V',E')$, dove

$V'=E$

$E'=((u,v), (w,z)) \in E' \Leftrightarrow (u,v) \text{ e } (w,z) \text{ appartengono ad } E \text{ e sono adiacenti in } G$



- **Teorema**

Sia $G=(V,E)$ grafo non ordinato bipartito e sia $L(G)$ il suo lim graph, $M \subseteq E$, allora vale quanto segue:

- a) M è un abbinamento in $G \Leftrightarrow M$ è un insieme indipendente in $L(G)$
- b) M è un abbin. massimale in $G \Leftrightarrow M$ è un insieme indipendente massimale in $L(G)$
- c) M è un abbin. massimo in $G \Leftrightarrow M$ è un insieme indipendente massimo in $L(G)$

Si può risolvere con MAX_CLIQUE, ma è un problema con complessità troppo elevata per essere trattato.

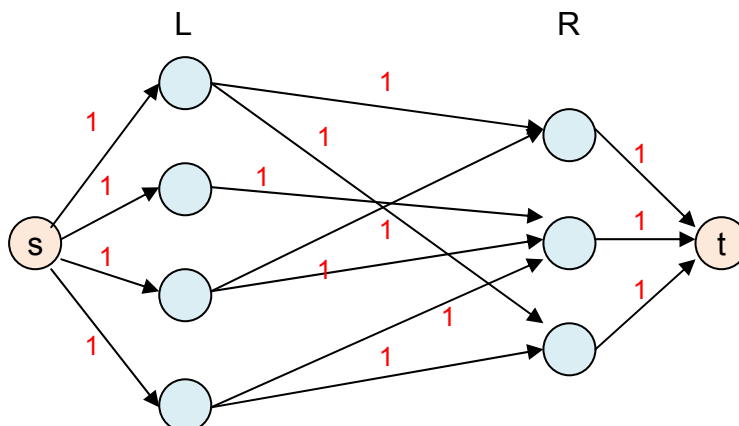
Abbinamento massimo \rightarrow Flusso massimo

Il problema di abbinamento massimo si può risolvere con il problema del flusso massimo, dato che già ne conosciamo la complessità.

A partire da $G=(V,E)$ si costruisce $G'=(V',E')$ dove:

$$V' = V \cup \{s, t\}$$

$$E' = \{ (s, u) \mid u \in L \} \cup \{ (v, t) \mid v \in R \} \cup \{ (u, v) \mid u \in L, v \in R, (u, v) \in E \}$$



Si prende in ingresso il grafo bipartito non orientato G e lo si trasforma nel grafo G' , aggiungendo “s” e “t” ($s \rightarrow L$ $t \rightarrow R$), dando un **verso** a tutti gli archi da L verso R e dando una **capacità unitaria** a tutti gli archi

- **Lemma**

Sia $G=(V,E)$ un grafo bipartito [$V=L \cup R$] e sia $G'=(V',E')$ la corrispondente rete di flusso. Se M è un abbinamento in G allora esiste un flusso f a valori interi nella rete G' tale che il valore di f è uguale alla cardinalità di M

$$|f| = |M|$$

Viceversa se f è un flusso a valori interi in G' allora esiste un abbinamento M in G' tale che

$$|M| = |f|$$

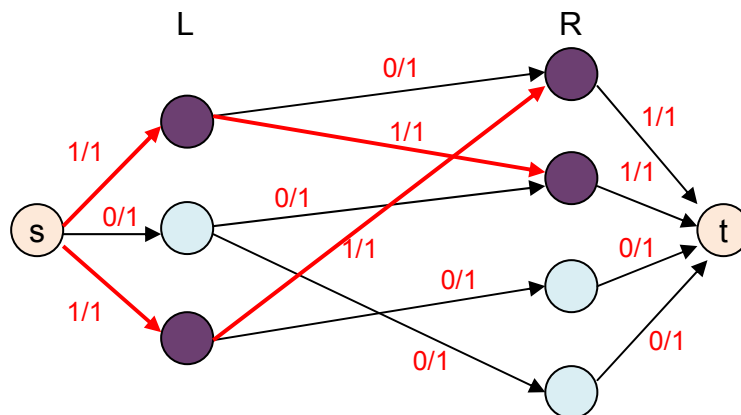
- **Nota**

$f : V \times V \rightarrow \mathbb{R}$: si dice flusso a valori interi se $f(u,v) \in \mathbb{Z}$, dove \mathbb{Z} = insieme dei numeri interi.



Dimostrazione:

Supponiamo per ipotesi di avere un abbinamento M in G , dobbiamo quindi trovare un flusso in G' .



$$\begin{aligned} \forall u \in L \quad \exists' \quad (u,v) \in M : f(s,u) = 1 [f(u,s) = -1] \\ \forall v \in R \quad \exists' \quad (u,v) \in M : f(v,t) = 1 [f(t,v) = -1] \\ \forall u,v \in M \quad \exists' \quad f(u,v) = 1 [f(v,u) = -1] \end{aligned}$$

Tutte le altre coppie di vertici avranno flusso = 0.

Dimostrazione (viceversa):

Supponiamo che $f: V \times V \rightarrow \mathbb{Z}$ e che f sia un flusso in G' , a partire dal flusso si costruisce M nel seguente modo:

$$\begin{aligned} M = \{ (u,v) \in E \mid f(u,v) > 0 \} \quad u \in L \quad v \in R \\ |f| = |M| \end{aligned}$$

Ogni vertice riceve al più flusso netto pari a 1 ed essendo tutti valori interi non possono uscire due archi con valore $\neq 0$ dallo stesso vertice, quindi non è possibile che due archi si trovino sullo stesso vertice e quindi $|f| = |M|$.

L'algoritmo che si usa deve ritornare un flusso a valori interi.

• Teorema di integralità

Se G è una rete di flusso con capacità intera allora l'algoritmo di Ford-Fulkerson restituirà un flusso a valori interi.

Gli abbinamenti massimi sono in corrispondenza biunivoca con i flussi massimi.

Complessità nell'ordine del flusso massimo $|f^*| \sim O(n)$, quindi $O(nm)$.

Non posso avere un numero di abbinamenti maggiore del numero di vertici del più piccolo tra L ed R .



NP-COMPLETEZZA

PROBLEMI P

Un qualsiasi problema \mathcal{P} è definito come una relazione binaria tra due oggetti, ed è formato da istanze \mathcal{I} e soluzioni \mathcal{S} .

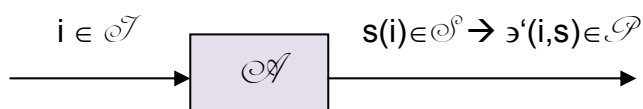
$$\mathcal{P} \subseteq \mathcal{I} \times \mathcal{S}$$

Ad esempio il problema Minimum Spanning Tree è esprimibile in questa forma:

MST: $\mathcal{I} = \{ G \mid G \text{ è un grafo non orientato pesato} \}$

$\mathcal{S} = \{ T \mid T \text{ è un albero di cammini minimi} \}$

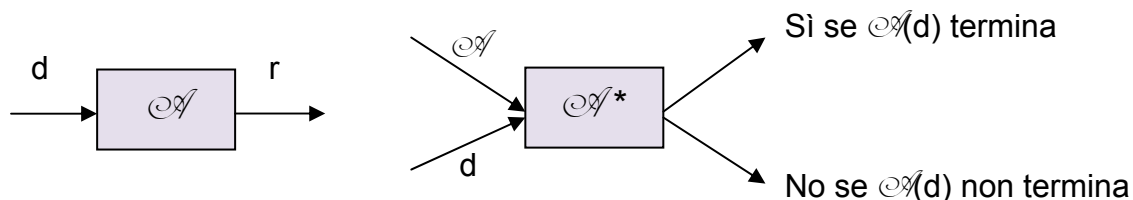
Volendo esprimere in maniera grafica questo schema generale si deve trovare un algoritmo \mathcal{A} che prenda in ingresso le singole istanze “i” dando come risultato delle soluzioni “s” che risolvano \mathcal{P} .



I problemi possono quindi essere divisi in più categorie:

- Indecidibili
- Decidibili
 - Trattabili
 - Intrattabili

I problemi **indecidibili** sono quei problemi che non trovano mai una soluzione, e che quindi non terminano mai, come ad esempio il “problema dell’alt”, in cui la terminazione di un algoritmo è strettamente legata alla terminazione di un altro algoritmo



In questo caso, ad esempio, se \mathcal{A} non termina, non potrà terminare neppure \mathcal{A}^*

I problemi decidibili invece si dividono in due categorie:

Trattabile \equiv semplice: esiste un algoritmo polinomiale per la sua risoluzione

Intrattabile \equiv difficile: non esiste un algoritmo polinomiale per la sua risoluzione



Più genericamente è possibile dividere i problemi in problemi di **ottimizzazione** come tutti quelli visti fin'ora, o in problemi di **decisione** (come saranno quelli che si vedranno d'ora in poi) caratterizzati da $\mathcal{S} = \{SI, NO\} = \{0,1\}$.

Ad esempio prendendo un sistema lineare :

$$\mathcal{S}: (A,b) \quad A \in \mathbb{R}^{m \times n} \quad b \in \mathbb{R}^m$$

Ci si può chiedere: esiste una soluzione al sistema tale che $Ax=b$ con $x \in \mathbb{R}^n$?

Uguualmente per quanto riguarda la connettività di un grafo, data un'istanza di un grafo G non orientato ci si può chiedere se G è connesso oppure no.

Se \mathcal{P} è un problema di ottimizzazione $\Rightarrow \exists \mathcal{P}'$ problema di decisione corrispondente a \mathcal{P} che ha la stessa complessità di \mathcal{P} .

Dato ad esempio $\mathcal{P} \equiv \text{MAX_CLIQUE} \Rightarrow \mathcal{P}' \equiv \text{CLIQUE}$

Istanze: G grafo, $k \in \mathbb{N}$

Domanda: esiste in G una CLIQUE c , con $|c| \geq k$?

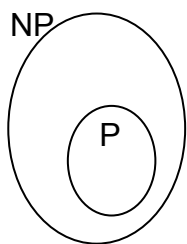
Le proprietà dei problemi \mathcal{P} quindi rendono la classe P definibile in questo modo:

$$P = \{ \mathcal{P} \mid \mathcal{P} \text{ è } \mathbf{risolvibile} \text{ mediante un algoritmo di complessità polinomiale} \}$$

PROBLEMI NP

La classe NP invece non tratta direttamente la complessità della risoluzione dei problemi, quanto piuttosto la complessità dell'algoritmo di verifica del problema, infatti la classe NP è definibile nel seguente modo:

$$NP = \{ \mathcal{P} \mid \mathcal{P} \text{ è } \mathbf{verificabile} \text{ tramite un algoritmo di complessità polinomiale} \}$$



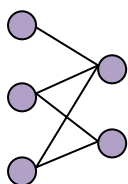
Si può affermare con certezza che $P \subset NP$, ma non si è assolutamente in grado di affermare o dimostrare che $P=NP$.

• Ciclo Hamiltoniano

Dato un grafo $G=(V,E)$ non orientato un ciclo Hamiltoniano è un ciclo semplice in G che "tocca" tutti i vertici di G , e il grafo G viene chiamato grafo Hamiltoniano.

Osservazione:

Se $G=(V,E)$ è bipartito e ha un numero dispari di vertici G non è Hamiltoniano



Comunque io costruisca gli archi in E un vertice rimarrà sempre escluso dal ciclo, che quindi non sarà Hamiltoniano.



RIDUCIBILITA' POLINOMIALE

E' un relazione lineare tra problemi: $\mathcal{P}_1 \leq_P \mathcal{P}_2 \Leftrightarrow$ comunque si prenda un'istanza del problema \mathcal{P}_1 è possibile trasformarla (ridurla) in tempo polinomiale in un'istanza equivalente del problema \mathcal{P}_2 .

Esiste quindi un algoritmo polinomiale \mathcal{A}_{12} che effettua la trasformazione.

Proprietà della riducibilità

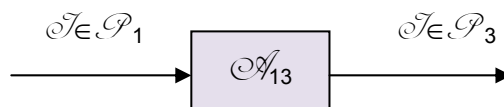
- **Proposizione: (transitività di \leq_P)**

Siano \mathcal{P}_1 , \mathcal{P}_2 e \mathcal{P}_3 problemi decisionali, se $\mathcal{P}_1 \leq_P \mathcal{P}_2$ e $\mathcal{P}_2 \leq_P \mathcal{P}_3$ allora necessariamente $\mathcal{P}_1 \leq_P \mathcal{P}_3$

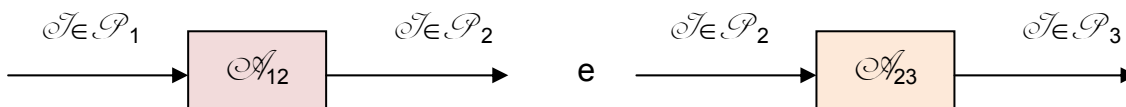
Dimostrazione:

Ipotesi: $\mathcal{P}_1 \leq_P \mathcal{P}_2$ e $\mathcal{P}_2 \leq_P \mathcal{P}_3$

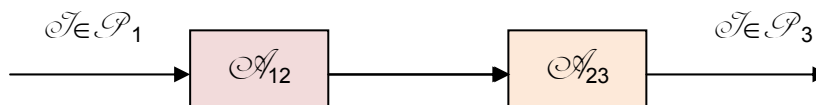
Tesi: $\mathcal{P}_1 \leq_P \mathcal{P}_3$



Bisogna trovare l'algoritmo \mathcal{A}_{13} , supponendo dati:



E' sufficiente quindi concatenare i due algoritmi dati:



La **complessità** sarà: $O(n^{k_{12}} + n^{k_{23}})$



PROBLEMI NPC

Appartengono alla classe di problemi NPC tutti quei problemi complessi, difficili o intrattabili, e la definizione della classe è la seguente:

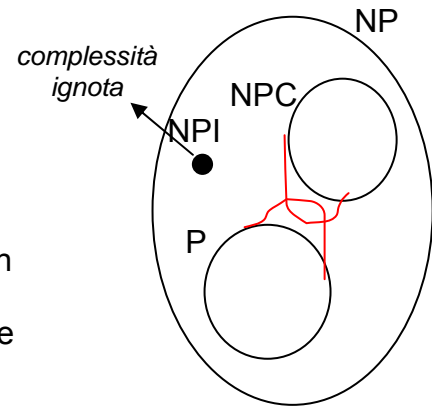
$$\text{NPC} = \left\{ \mathcal{P} \mid \begin{array}{l} 1) \mathcal{P} \in \text{NP} \text{ (}\mathcal{P}\text{ è verificabile in tempo polinomiale)} \\ 2) \mathcal{P}' \in \text{NP} : \mathcal{P}' \leq_P \mathcal{P} \end{array} \right\}$$

TEOREMA FONDAMENTALE DELLA NPCCompletezza

$$\text{Se } P \cap \text{NPC} \neq \emptyset \Rightarrow P = \text{NP}$$

Si ipotizza cioè che le classi P ed NPC abbiano almeno un elemento in comune, questo porterebbe le classi a collapsare e tutti i problemi verificabili in tempo polinomiale sarebbero anche risolvibili in tempo polinomiale.

Se invece $P \neq \text{NP} \Rightarrow P \cap \text{NPC} = \emptyset$



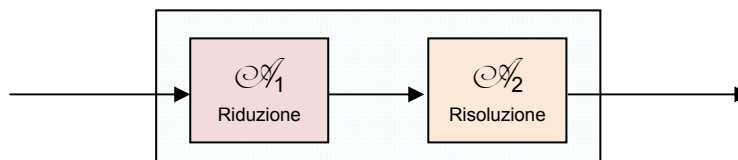
Dimostrazione:

Ipotesi: $P \cap \text{NPC} \neq \emptyset$, sia $\mathcal{P} \in P$ e $\mathcal{P} \in \text{NPC}$

Tesi: $P \subseteq \text{NP}$ (ok) e $\text{NP} \subseteq P$ (?)

Prendiamo un sottoinsieme $Q \in \text{NP}$, bisogna quindi dimostrare che $Q \in P$, perché se viene dimostrato allora $\text{NP} \subseteq P$.

Per la definizione di NPC si sa che $\forall \mathcal{P}' \in \text{NP} : \mathcal{P}' \leq_P \mathcal{P}$, quindi $Q \leq_P \mathcal{P} \in P$



• Osservazioni:

$$\left. \begin{array}{l} 1) \mathcal{P}_1 \leq_P \mathcal{P}_2 \\ 2) \mathcal{P}_2 \in P \end{array} \right\} \Rightarrow \mathcal{P}_1 \in P$$

$\mathcal{P} \in \text{NPC}$

1) $\mathcal{P} \in \text{NP}$

2) $\forall \mathcal{P}' \in \text{NP} : \mathcal{P}' \leq_P \mathcal{P}$

Per dimostrare quest'ultima è sufficiente dimostrare che $\exists Q \in \text{NPC} \ni Q \leq_P \mathcal{P} \Rightarrow \mathcal{P} \in \text{NPC}$



TEOREMA DI COOK

CIRCUIT_SAT \in NPC

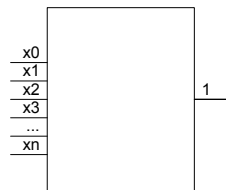
Le istanze di CIRCUIT_SAT (soddisfattibilità dei circuiti) sono l'insieme dei circuiti logici costruiti con porte AND, OR, NOT ad una sola uscita.

La domanda che ci si pone è se il circuito è soddisfattibile, ovvero se esiste una combinazione di INPUT che dia 1 in OUTPUT.

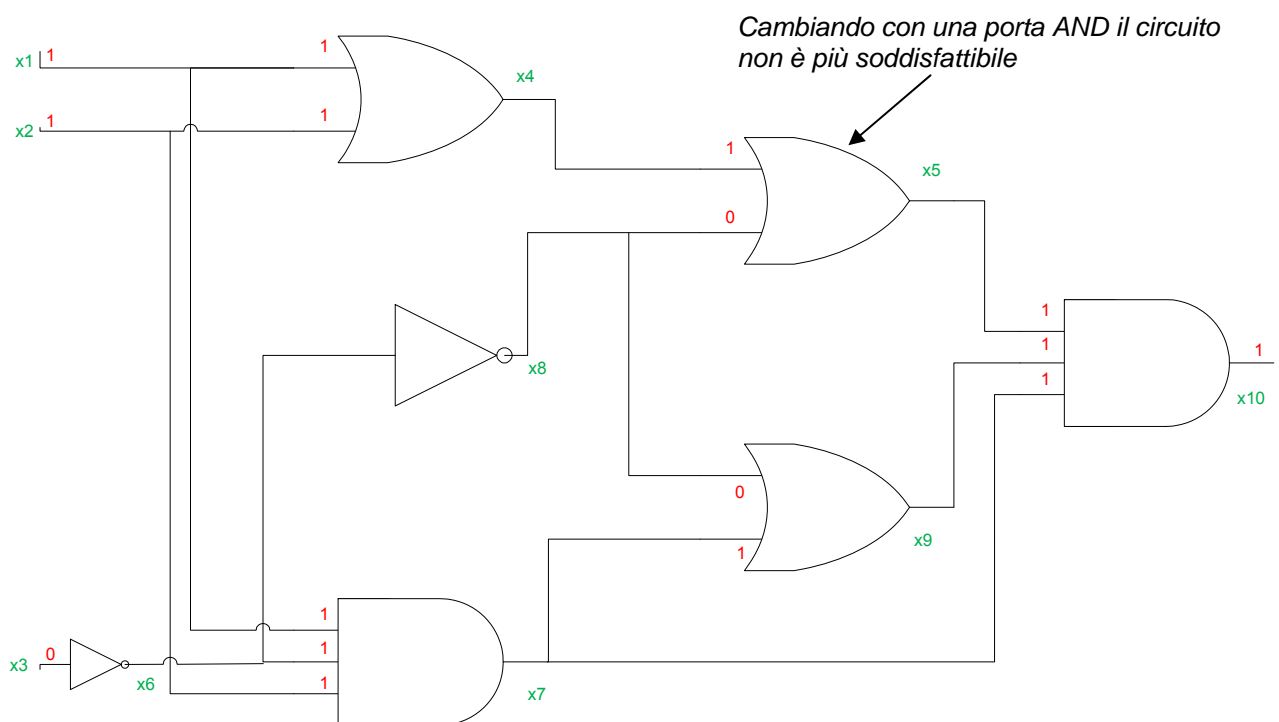
1) NOT 

2) AND 

3) OR 



Ci si chiede se è possibile con una sequenza in ingresso, avere 1 in uscita, cioè se il circuito è soddisfattibile.



1) CIRCUIT_SAT \in NP

2) $\forall \mathcal{P}' \in \text{NP} : \mathcal{P}' \leq_P \text{CIRCUIT_SAT}$ } CIRCUIT_SAT \in NPC



SAT

(soddisfattibilità di formule booleane)

La soddisfattibilità ha come istanze tutte le formule booleane costituite da:

- 1) variabili booleane (x_1, x_2, \dots, x_n)
- 2) connettivi logici ($\wedge, \vee, \sim, \Rightarrow, \Leftrightarrow$) [AND, OR, NOT, SE, SSE]
- 3) parentesi tonde

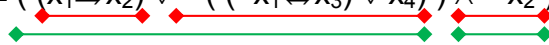
$A \Leftrightarrow B = \text{VERO}$ se $A \Rightarrow B \wedge B \Rightarrow A$
 $A \Rightarrow B = \text{FALSO}$ solo se A è VERO e B è FALSO

• Domanda

La formula in ingresso è soddisfattibile?

Ovvero: esiste una combinazione in ingresso che dia VERO in uscita?

Dobbiamo allora dimostrare che **SAT** \subseteq **NPC**, quindi, presa una formula booleana

$$\Phi \equiv ((x_1 \Rightarrow x_2) \vee \sim ((\sim x_1 \Leftrightarrow x_3) \vee x_4)) \wedge \sim x_2$$


$x_1=0$
 $x_2=0$
 $x_3=1$
 $x_4=1$

dobbiamo dimostrare che:

- 1) SAT \subseteq NP
- 2) CIRCUIT_SAT \leq_P SAT

La trasformazione del circuito logico in formula booleana inizia dando un nome (di variabile) ad ogni filo del circuito.

Oltre ai valori in INPUT si da un nome ad ogni risultato intermedio di ogni porta logica.

Prendendo il circuito precedente la sua corrispondente formula booleana è questa:

$$\Phi \equiv [x_4 \Leftrightarrow (x_1 \vee x_2)] \wedge [x_5 \Leftrightarrow (x_4 \vee x_8)] \wedge [x_6 \Leftrightarrow \sim x_3] \wedge [x_7 \Leftrightarrow (x_1 \vee x_2 \vee x_6)] \wedge [x_8 \Leftrightarrow \sim x_6]$$

$$\wedge [x_9 \Leftrightarrow (x_7 \vee x_8)] \wedge [x_{10} \Leftrightarrow (x_7 \vee x_5 \vee x_9)] \wedge x_{10} \text{ (si conclude sempre con la variabile d'uscita)}$$



3_SAT_FNC (forma normale congiuntiva)

I termini che costituiscono una formula booleana si possono raggruppare in due categorie:

- 1) **LETTERALE**: una variabile o la sua negazione ($x_7, \sim x_9, x_{10} \dots$)
- 2) **CLAUSOLA**: una disgiunzione di letterali ($l_1 \vee l_2 \vee l_3 \vee l_n$ con l_i =letterale)
 - **3_CLAUSOLA**: clausola con 3 letterali ($l_1 \vee l_2 \vee l_3$)

Esempio: $x_9 \vee \sim x_{10} \vee \sim x_2$

Una formula booleana è in forma 3_FNC se è del tipo:

$\Phi : C_1 \wedge C_2 \wedge \dots \wedge C_n$ dove $\forall i=1 \dots n$, C_i è una 3_clausola

Esempio:

$$\Phi \equiv (\overset{3 \text{ letterali}}{x_1 \vee x_2 \vee x_3}) \wedge (\overset{3 \text{ letterali}}{\sim x_1 \vee \sim x_2 \vee x_4}) \wedge (\overset{3 \text{ letterali}}{x_1 \vee \sim x_3 \vee \sim x_5}) \quad [\text{è in forma 3_FNC}]$$

- **Nota**

Nell'esempio precedente ci sono solo operazioni di AND tra le 3_clausole, quindi, affinché la formula sia vera, tutte le 3_clausole devono essere vere, ma avendo all'interno delle clausole stesse solo operazioni di OR, è sufficiente che almeno una letterale di ogni clausola sia vera per rendere vera l'intera formula

- **Teorema**

3_SAT_FNC è NPCompleto

Ha come istanze tutte le formule booleane Φ in forma 3_FNC, e su queste ci si pone la domanda: la formula Φ è soddisfattibile?

$$3_SAT_FNC \in NPC \Rightarrow \begin{array}{l} 1) 3_SAT_FNC \in NP \\ 2) 3_SAT_FNC \text{ è NP arduo} \end{array}$$

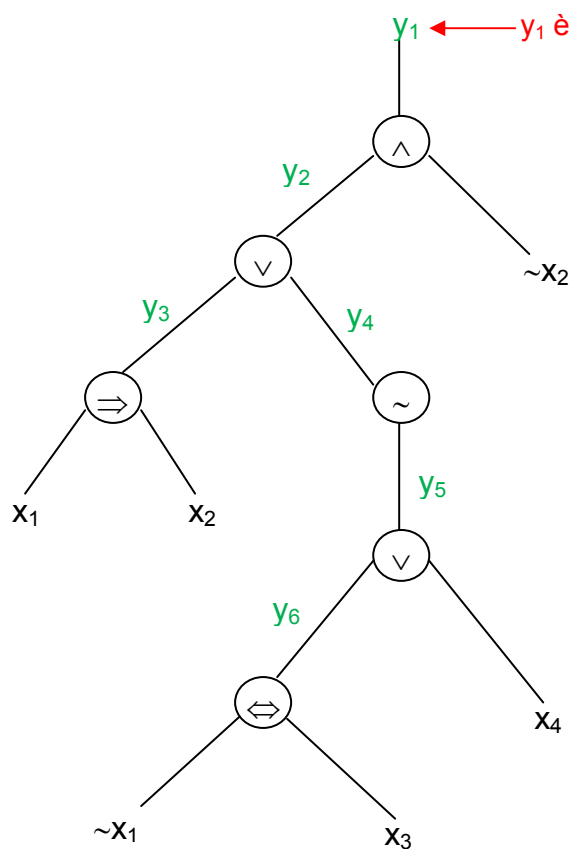
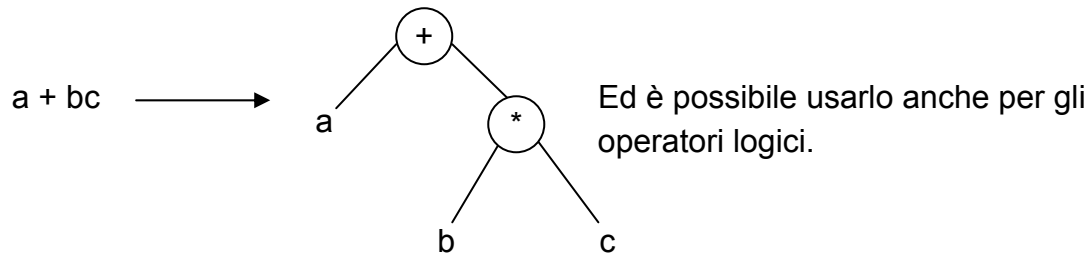
Istanza di SAT che si cercherà di portare in forma 3_FNC:

$$\Phi \equiv ((x_1 \Rightarrow x_2) \vee \sim ((\sim x_1 \Leftrightarrow x_3) \vee x_4)) \wedge (\sim x_2)$$



1) ALBERO DI PARSING DELLA FORMULA Φ

Si crea un albero, detto di parsing, della formula che si sta trasformando. Un albero di parsing funziona nel seguente modo, come mostrato dall'esempio:



- Possiamo vedere l'albero come un circuito logico più generale, con porte di implicazione ed equivalenze oltre a quelle già usate.
- Associamo ad ogni filo una nuova variabile locale y_i per poi procedere normalmente con CIRCUIT_SAT.

$$\Phi' \equiv y_1 \wedge (y_1 \Leftrightarrow (y_2 \wedge \sim x_2)) \wedge (y_2 \Leftrightarrow (y_3 \vee y_4)) \wedge (y_3 \Leftrightarrow (x_1 \Rightarrow x_2)) \wedge (y_4 \Leftrightarrow (\sim y_5)) \wedge (y_5 \Leftrightarrow (y_6 \vee x_4)) \wedge (y_6 \Leftrightarrow (\sim x_1 \Leftrightarrow x_3))$$

Quindi Φ è soddisfattibile se e solo se Φ' è soddisfattibile, e Φ' è una successione di AND, ma ciascuna delle clausole ancora non è in forma normale congiuntiva.



2) TRASFORMIAMO OGNI CLASUSOLA IN FORMA NORMALE CONGIUNTIVA.

Ogni sottoformula ha al massimo tre letterali, si costruisce quindi la tabella di verità delle clausole:

$$\Phi'' \equiv y_1 \Leftrightarrow (y_2 \wedge \sim x_2)$$

Si procede costruendo $\sim\Phi''$ e si controlla quando la formula è FALSA (cioè = 0)

y_1	y_2	x_2	$\sim x_2$	$y_2 \wedge \sim x_2$	$y_1 \Leftrightarrow (y_2 \wedge \sim x_2)$
0	0	0	1	0	1
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0

$$\sim\Phi'' \equiv (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \sim y_2 \wedge x_2) \vee (y_1 \wedge \sim y_2 \wedge \sim x_2) \vee (\sim y_1 \wedge y_2 \wedge \sim x_2)$$

3) OGNI SOTTOFORMULA CONTIENE 3 LETTERALI, MA E' IN FORMA COMPLEMENTARE A QUELLA CHE SI CERCAVA

Infatti si cercava uno schema del tipo $C_1 \wedge C_2 \wedge \dots \wedge C_n$ dove $C_i = h_1 \vee h_2 \vee \dots \vee h_n$, ma abbiamo ottenuto l'esatto opposto. Si applica quindi la proprietà di De Morgan su Φ'' ottenendo come risultato la formula finale:

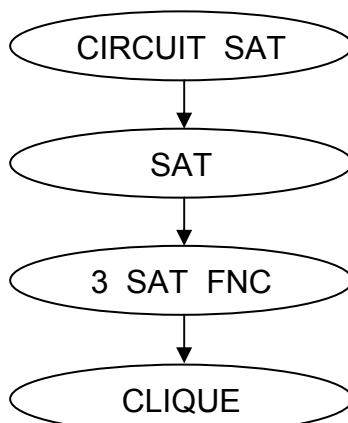
$$\Phi'' \equiv (\sim y_1 \vee \sim y_2 \vee \sim x_2) \wedge (\sim y_1 \vee y_2 \vee \sim x_2) \wedge (\sim y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \sim y_2 \vee x_2)$$

Spesso è possibile avere clausole con solo due letterali ($p' \vee p''$), in questi casi si introduce una variabile fittizia q , trasformando la formula in $(p' \vee p'' \vee q) \wedge (p' \vee p'' \vee \sim q)$.

Se si ha invece solo una letterale allora bisognerà aggiungere due variabili fittizie:

$$p \rightarrow (p' \vee q' \vee q'') \wedge (p' \vee \sim q' \vee q'') \wedge (p' \vee q' \vee \sim q'') \wedge (p' \vee \sim q' \vee \sim q'')$$

- Il tempo di computazione è polinomiale



CLIQUE

Istanze: $G=(V,E)$ non ordinati, costante k

Domanda: G contiene una clique con k vertici?

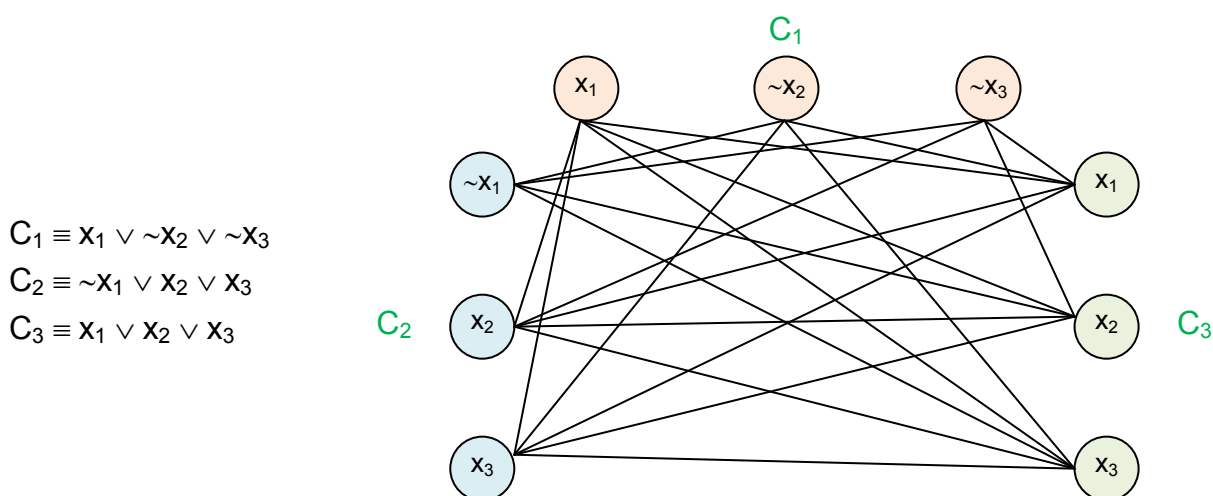
• Teorema

CLIQUE \in NPC

1) CLIQUE \in NP

2) $3_SAT_FNC \leq_p$ CLIQUE (ovvero clique è difficile almeno quanto 3_SAT_FNC)

$\Phi \equiv C_1 \wedge C_2 \wedge C_3$ ci sono quindi 3 gruppi di 3 vertici l'uno.

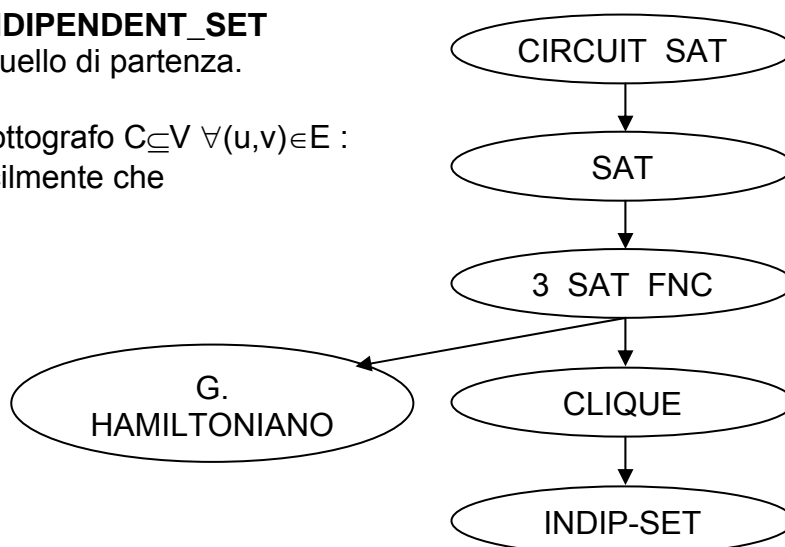


- Mai mettere archi tra vertici dello stesso gruppo
- Mai unire una variabile e la sua negazione
- Nel grafo c'è una clique di cardinalità 3 solo se Φ è soddisfattibile
- Se nel grafo c'è una clique di cardinalità 3 allora Φ è soddisfattibile

Si dimostra che **CLIQUE \leq_p INDIPENDENT_SET** utilizzando il grafo inverso di quello di partenza.

VERTEX_COVER : dato un sottografo $C \subseteq V \forall (u,v) \in E : u \in C \vee v \in C$, e si dimostra facilmente che **INDIPENDENT_SET**

↓
VERTEX_COVER



COSA FARE DAVANTI A UN PROBLEMA

NPCCompleto

1) Cercare algoritmi di **approssimazione**

VERTEX_COVER \in NPC ma si può risolvere in tempo polinomiale con una certa tolleranza

CLIQUE \in NPC non è approssimabile in nessun modo

2) Algoritmi **probabilistici**

Ripetuti un certo numero di volte danno il risultato atteso

3) Casi **speciali**

Alcuni problemi in certi casi possono avere un tempo esponenziale e in altri polinomiale, ad esempio PLANAR_CLIQUE (grafo disegnabile su un piano senza far intersecare gli archi) pur essendo CLIQUE \in NPC, PLANAR_CLIQUE si risolve in tempo polinomiale.

4) Algoritmi **esponenziali**

Algoritmo del simplesso $\min c^T x$ tale che $Ax=b, x \geq 0$

Teoricamente è esponenziale, ma in pratica funziona benissimo.

5) **Euristiche**

Algoritmi di cui non si sa dire nulla, funzionano bene in pratica, per esperienza empirica, ma non c'è modo di dimostrarli o computarli o garantirne il funzionamento.



Indice analitico

Abbinamento massimo.....	65
Alberi liberi	21
Algoritmi Greedy	33
Algoritmo di Bellman-Ford.....	42
Algoritmo di Dijkstra	38
Algoritmo di Edmonds-Karp	62
Algoritmo di Floyd-Warshall	48
Algoritmo di Ford-Fulkerson	59
Algoritmo di Kruskal	29
Algoritmo di Prim.....	30
Arco critico	64
Cammini minimi	35
Cammini minimi tra tutte le coppie di vertici	44
Cammino aumentante.....	56
Cammino e sottocammino	5
Clique.....	24
Clustering.....	26
Collezione di insiemi disgiunti	29
Correttezza di Dijkstra.....	41
Cosa fare davanti a un problema NPC.....	79
Disuguaglianza triangolare.....	37
Forma normale congiuntiva.....	75
Grafi autocomplementari.....	15
Grafi Connessi	17
Grafi e sottografi	4
Grafo Bipartito	7
Isomorfismo di grafi.....	8
Lemma della stretta di mano	13
Metodo della quadratura ripetuta	47
Metodo di Ford-Fulkerson	54
Minimum Spanning Tree (MST)	26
Moltiplicazioni tra matrici	44
NP-Completezza	69
Problemi NP	70
Problemi NPC	72
Problemi P	69
Rappresentazione di grafi	10
Rete residua.....	54
Reti di flusso	50
Riducibilità polinomiale.....	71
Soddisfattiibilità di formule booleane.....	74
Sommatoria implicita.....	53
Sottocammini di cammini minimi	37
Taglio	57
Teorema del taglio minimo/flusso massimo	58
Teorema di Cook.....	73
Teorema di integralità	68
Teorema fondamentale dei MST.....	28
Teorema fondamentale della NPCompletezza.....	72

