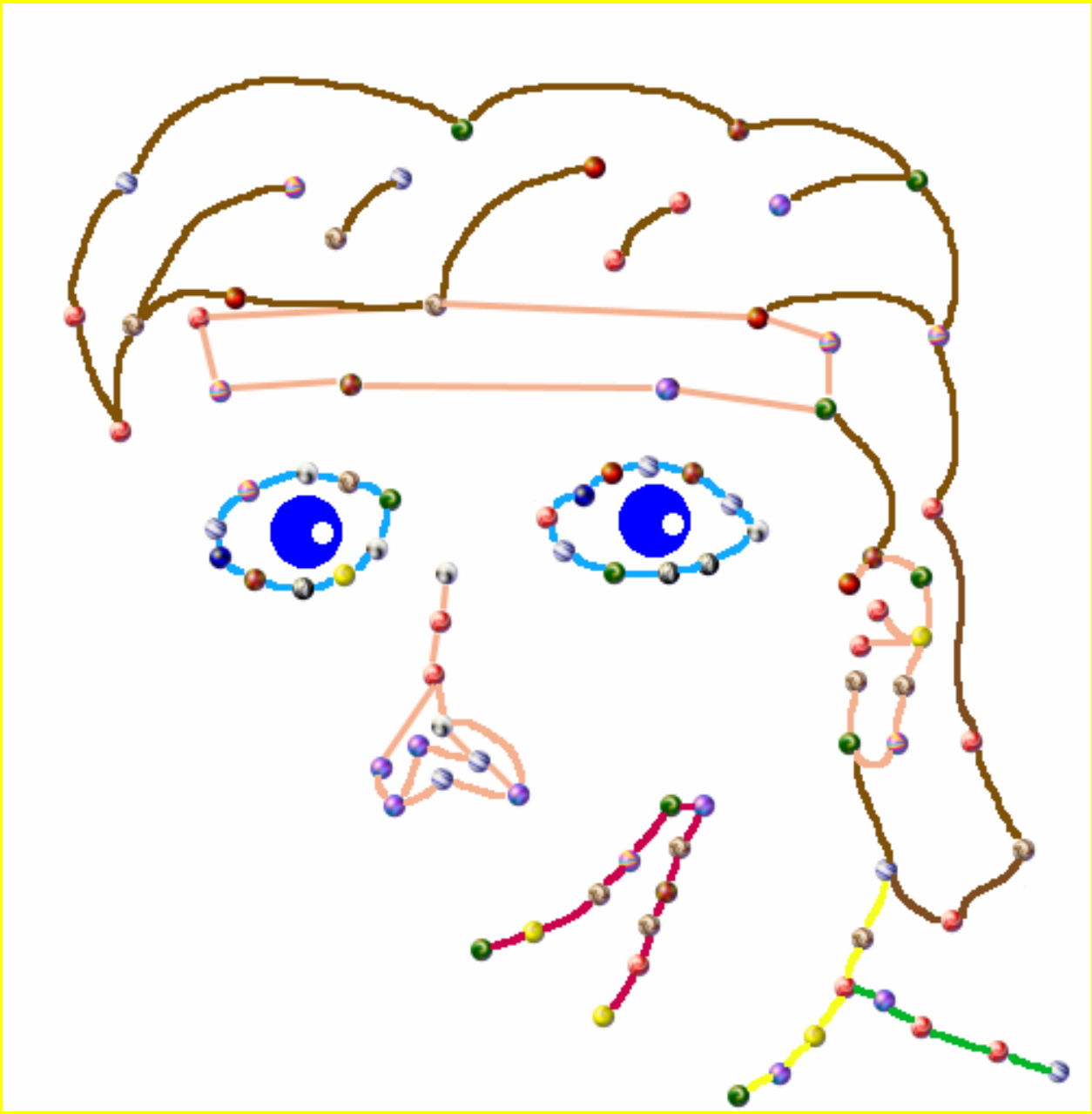
DESMATRON

TEORIA DEI GRAFI



Teoria dei Grafi
Release 1.0.0

Author: Desmatron
Date of Release: October 28, 2004

Author website: desmatron.altervista.org
Book website: desmatron.altervista.org/teoriadeigrafi/index.html
teoriadeigrafi.altervista.org
E-mail: desmatron@email.it
teoriadeigrafi@email.it

PREMESSA

Questa è la prima edizione di un libro che non vuole avere la pretesa di essere un formale testo di Teoria dei Grafi bensì, una gradevole lettura per tutti quegli esperti in materia ed un valido aiuto per coloro che approcciano questi 'recenti' studi per la prima volta.

Per quanto riguarda il numero di release, ho pensato di utilizzare l'ultima cifra come indicativa per eventuali correzioni di una stessa release. La seconda cifra dovrebbe star ad indicare l'aggiunta di qualche esercizio, considerazione o dimostrazione di qualcosa già affrontata mentre la prima cifra avrebbe lo scopo d'indicare una release dove sia stato aggiunto qualcosa di non affrontato; nuovi teoremi, argomenti o altro.

Ho riletto il testo una sola volta per motivi di tempo e mi scuso sin d'ora per eventuali mancanze o inesattezze. Cercherò di fare del mio meglio per trovare eventuali errori che, se volete, potrete segnalare alle e-mails indicate a pagina 1.

Infine, se oltre a segnalare errori vorrete suggerire anche qualche argomento, fatelo pure.

Buon divertimento ☺

Teoria dei Grafi
Capitolo 1

Introduzione

Innanzitutto poniamoci la domanda chiave:

"Di che cosa si occupa la Teoria dei Grafi"?

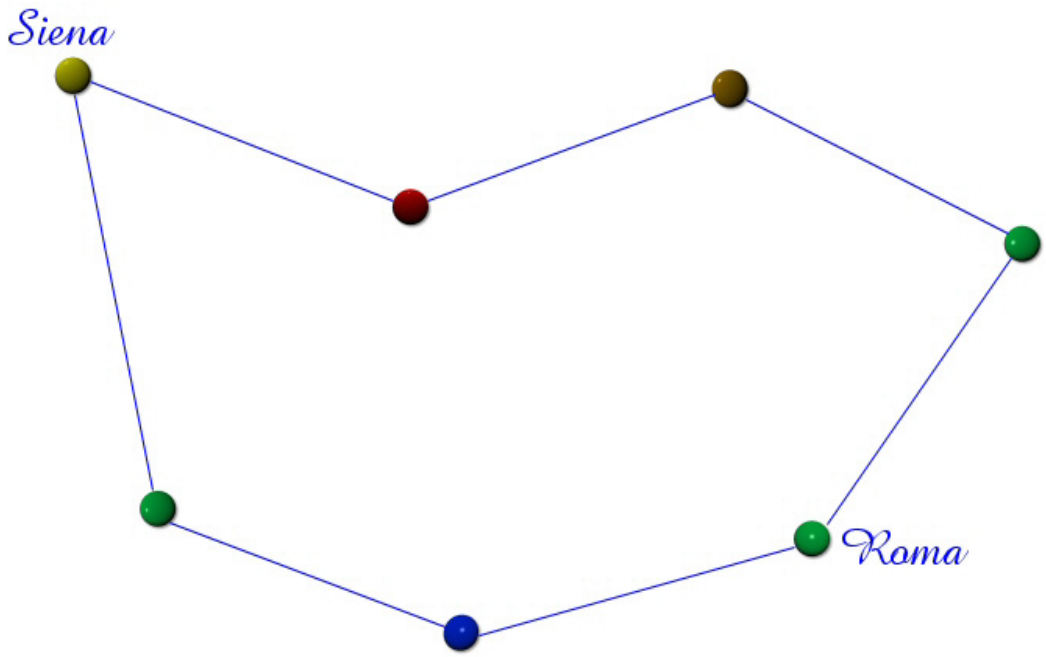
Questa materia relativamente molto recente fa parte di tutto un filone matematico-ingegneristico che sta sotto il nome di **Ricerca Operativa** (*Operations Research* è il termine anglossasone).
In quest'ambito più vasto si vanno a risolvere problemi di minimo (e viceversa di massimo) sotto opportune restrizioni poste dal problema preso in esame e con particolari metodi che sono tuttora oggetto di studio. Si parla comunque quasi sempre di ottimizzazione di un problema piuttosto complesso. Gli algoritmi creati ad hoc per la risoluzione di problemi all'apparenza irrisolubili costituiscono l'ossatura di tutta la Ricerca Operativa e semplici ragionamenti possono essere coadiuvati da potenti computers per la risoluzione di problemi con centinaia o migliaia di variabili.

Tornando però ad analizzare la Teoria dei Grafi, c'é da dire che a differenza di molte altri rami della Ricerca Operativa, questa opera certamente sotto la visualizzazione grafica di archi, nodi e flussi. Vedremo meglio in dettaglio questi termini, ma sottolineo che qualsiasi problema di Grafi e Reti apparentemente descrivibile solo in forma grafica, ha invece una sua possibile descrizione matematica e in particolare, una formulazione di programmazione lineare, lineare intera o non lineare.

La Teoria dei Grafi è in possesso di algoritmi che permettono di capire quale sia il percorso minimo su di un grafo qualsiasi e per grafo qualsiasi intendo quell'insieme di nodi, archi e flussi che descrive una situazione reale e non. L'esempio più eclatante di un grafo è quello della rete stradale, dove gli incroci sono nodi e le strade sono archi. Possiamo anche descrivere la possibilità di un tratto stradale di sopportare un certo numero di veicoli tramite la variabile capacità massima del flusso. Parliamo invece semplicemente di flusso quando vogliamo descrivere il numero di unità (in questo caso autoveicoli) che passano sul tratto al tempo t.

Ma vediamo un primo esempio di grafo semplice nella figura 1.1.

Figura 1.1



Come si può facilmente capire abbiamo semplificato due percorsi che passano dalla città di Siena e di Roma. Da notare quattro cose:

- non abbiamo né punti di partenza né punti d'arrivo
- non abbiamo restrizioni sul verso di percorrenza di ciascun arco
- non abbiamo restrizioni sulla capacità massima di ciascun arco
- non abbiamo informazioni sul flusso degli archi

Vedremo comunque più avanti tutte le tipologie di grafo.

Nella figura 1.2 ho riportato invece un esempio classico di Ricerca Operativa che potrete risolvere con un software adeguato come il Lindo 6.1.
Potete scaricare i 3.3 Mb di programma al seguente sito:

- **http://www.lindo.com/cgi/frameset.cgi?leftdwnld.html;downloadf.html**

Si può vedere dalla figura che i problemi di Ricerca Operativa consistono in generale nell'ottimizzazione di una funzione obiettivo. Quest'ottimizzazione verterà sulla minimizzazione se parliamo ad esempio di costi di progetto o sulla massimizzazione se parliamo di ricavi monetari.

Figura 1.2

$$\max 2X_1 - 3X_2 + 7X_3$$

$$\begin{aligned} \text{st} \quad & X_1 - 2X_2 \leq 1 \\ & X_2 + 3X_3 \geq 2 \\ & 2X_1 + 2X_3 \leq 3 \end{aligned}$$

In questo caso abbiamo voluto massimizzare una funzione che prevede tre variabili (X_1, X_2 e X_3) sotto i vincoli descritti dal *'subject to'* (st). Le restrizioni che necessita ciascun problema sono quasi sempre rappresentabili, anche il fatto che si voglia dal problema una soluzione delle tre variabili di tipo intero. Quest'ultimo caso ricade nell'Integer Linear Programming ovvero, nella Programmazione Lineare Intera dove sono esplicitamente richieste soluzione appartenenti ai numeri naturali.

In generale si può dire che le peculiarità della Teoria dei Grafi e della Ricerca Operativa siano quelle di abbreviare i tempi computazionali di qualsiasi problema. Una volta trovata una buona strada per l'ottimo, è compito degli studiosi trovare un algoritmo ancora più efficiente per rendere minimi i tempi di calcolo dei computers.

Fatta questa breve e semplice premessa, addentriamoci nel mondo della Teoria dei Grafi.

Teoria dei Grafi
Capitolo 2

Definizioni di base

Andiamo a vedere una per una le simbologie e le definizioni che useremo lungo tutto il libro.

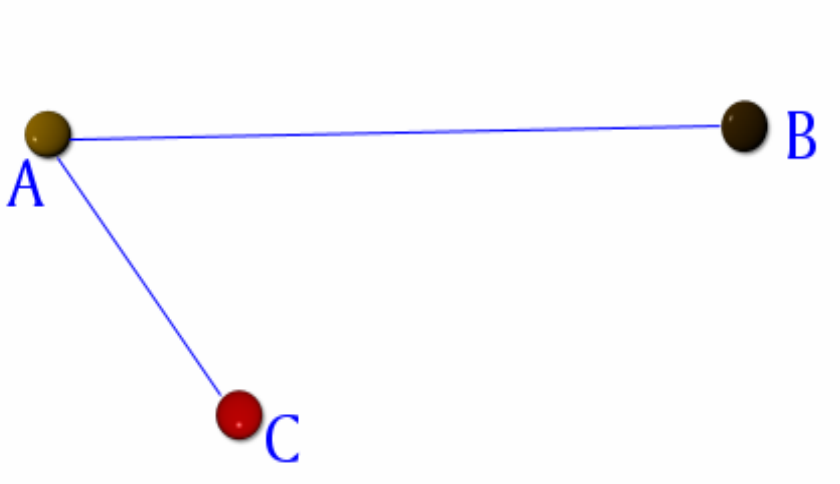
Grafo

Con la dicitura G = (V,E) indichiamo il generico grafo con V nodi ed E archi.
La denominazione delle lettere deriva dal fatto che in inglese si indicano i nodi con la parola *"vertex"* e gli archi con la parola *"edge"*.

Nodi adiacenti

Due nodi si dicono adiacenti se sono connessi da un arco.

Figura 2.1

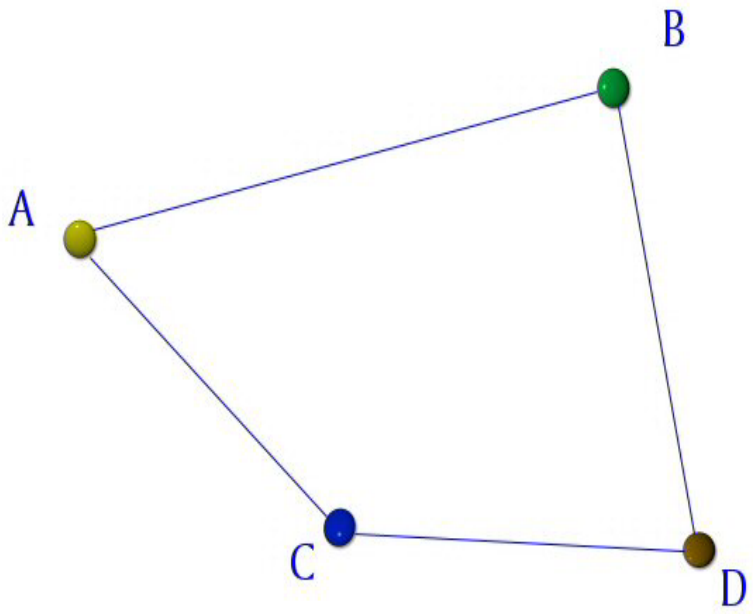


I nodi A e B sono adiacenti, così come lo sono A e C, ma non sono adiacenti tra loro i nodi B e C.

Archi adiacenti

Si dicono archi adiacenti quegli archi che hanno un nodo in comune.

Figura 2.2



Sono adiacenti gli archi:

- AC-AB
- AC-CD
- CD-DB
- DB-BA

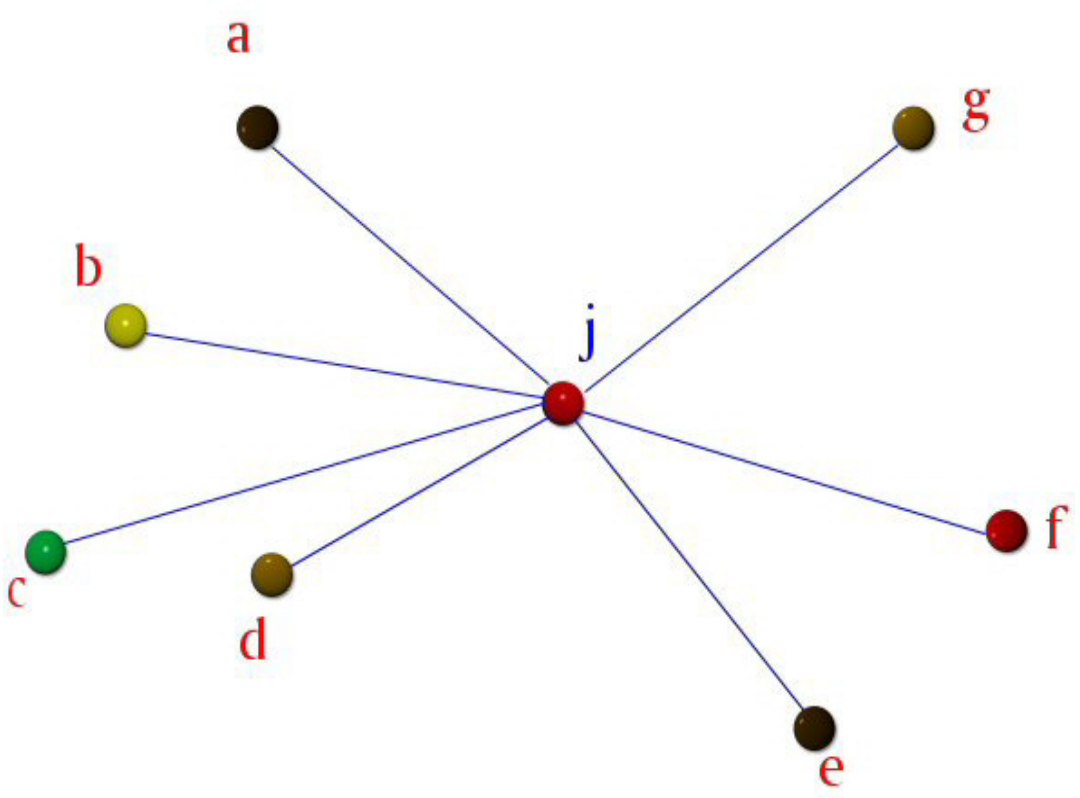
Non sono adiacenti gli archi:

- AC-BD
- AB-CD

Vicinato

Per vicinato s'intende l'insieme di tutti i nodi adiacenti al generico nodo j.

Figura 2.3



In questo caso il vicinato di j è il seguente insieme: {a,b,c,d,e,f,g}.

Archi multipli e loops

Gli archi multipli e i loops non verranno trattati molto in questo libro, ma per completezza d'informazione diciamo che i primi sono archi differenti che collegano due stessi nodi, mentre i secondi sono archi che hanno come punto di partenza ed arrivo sempre lo stesso generico nodo j.

Si vedano le relative figure 2.4 e 2.5.

Figura 2.4

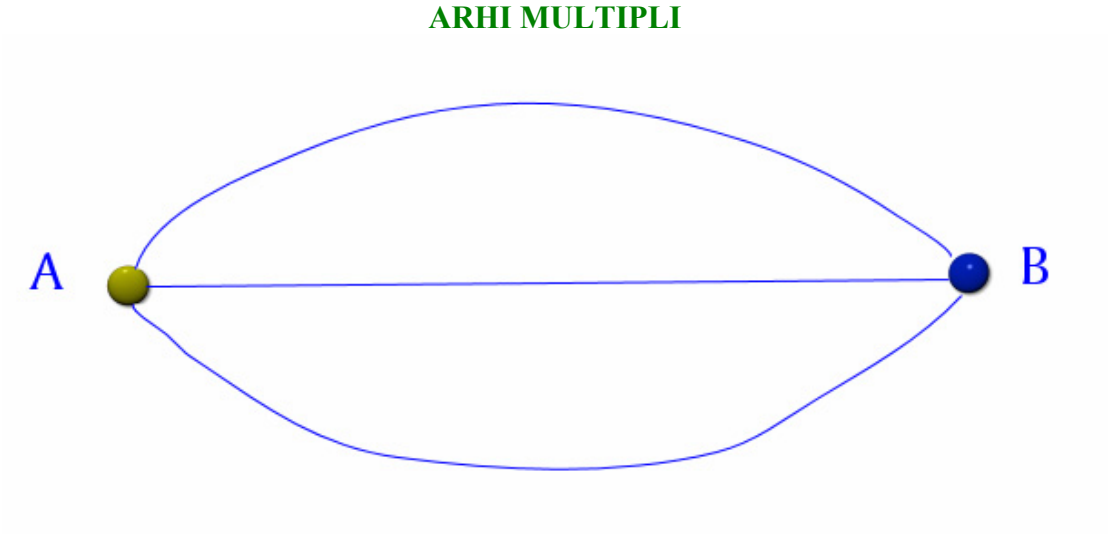
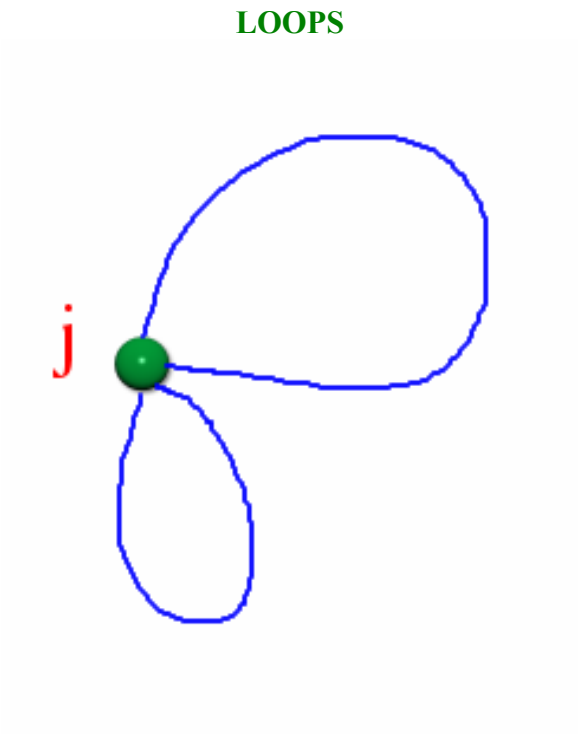


Figura 2.5



Grafo semplice

Si dice grafo semplice quel grafo che non contiene né archi multipli né loops.

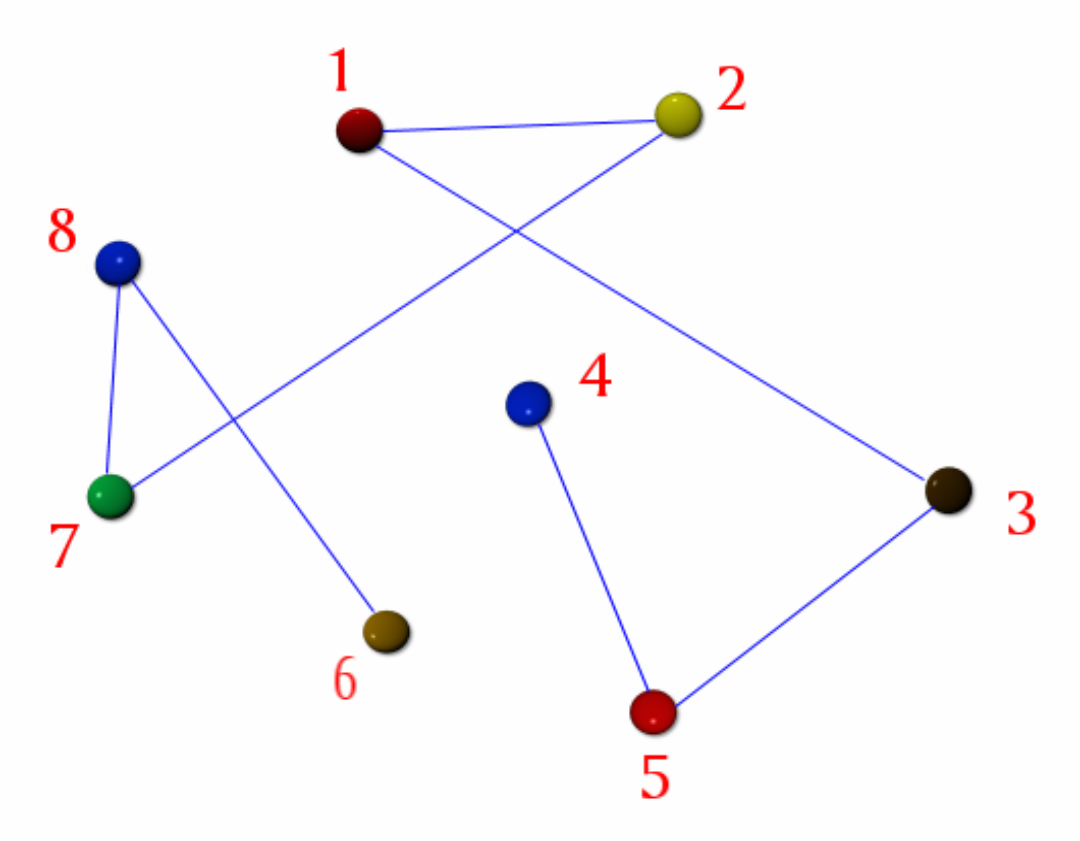
Alcuni esempi di descrizione di un grafo

Come abbiamo visto a pagina 4, un grafo generico lo possiamo indicare con la dicitura (V,E), ma esistono anche altri modi per dire che G è un grafo contenente V nodi ed E archi.
Vediamone alcune:

- G = G (V,E)
- G = (V(G), E(G))
- G = { 12,27,78,86,45,53,31 }

Nell'ultimo esempio abbiamo indicato l'insieme degli archi del grafo G. Questo tipo di descrizione contiene una preziosa informazione: i nodi che collegano ed il numero di nodi presenti in G.

Figura 2.6



Da notare la totale indifferenza nel descrivere un arco non orientato nel modo AB o BA; entrambe le diciture indicano lo stesso arco. Dunque, potremo descrivere indifferentemente il grafo di figura

2.6 nel seguente modo: G = { 21,72,78,86,54,53,31 } dove abbiamo cambiato la posizione dei nodi di tre archi; il primo, il secondo e il terzultimo.
Vedremo più avanti il significato di archi orientati e le loro conseguenti implicazioni.

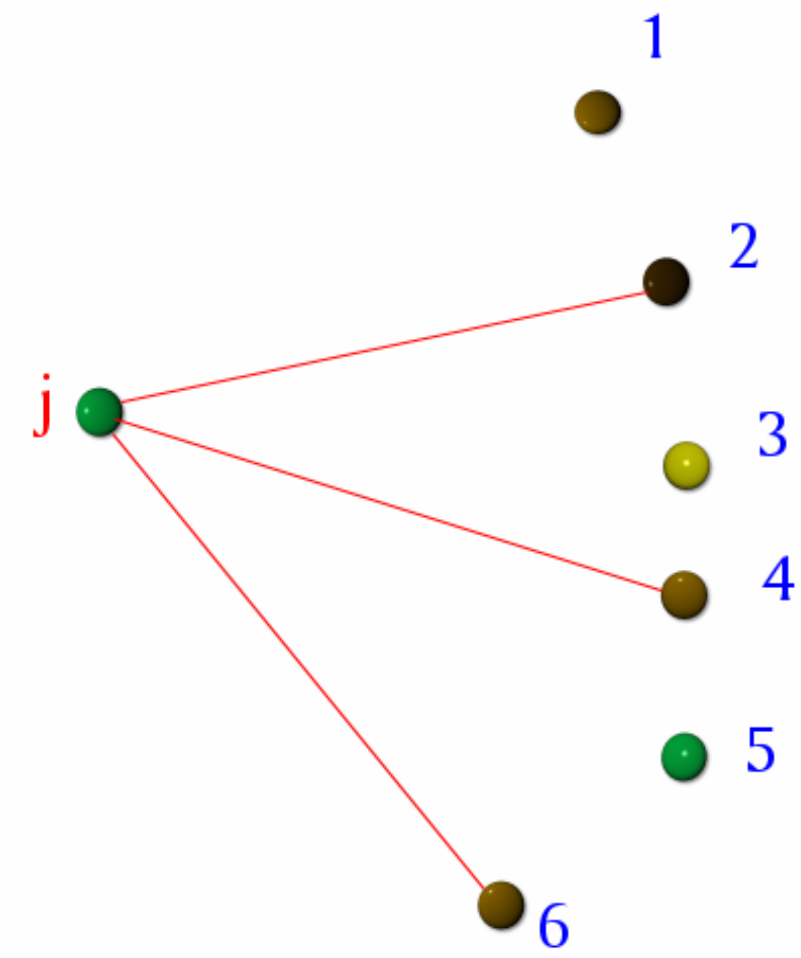
Un'ulteriore ed interessante notazione per descrivere un grafo è questa:

- G = (n, f(n))

Come si vede, gli archi sono descritti in funzione dei nodi e questo ci dice che i nodi sono collegati tra loro con un criterio che obbedisce ad una legge ben definita, la funzione f(n) appunto.

Esempio 2.1

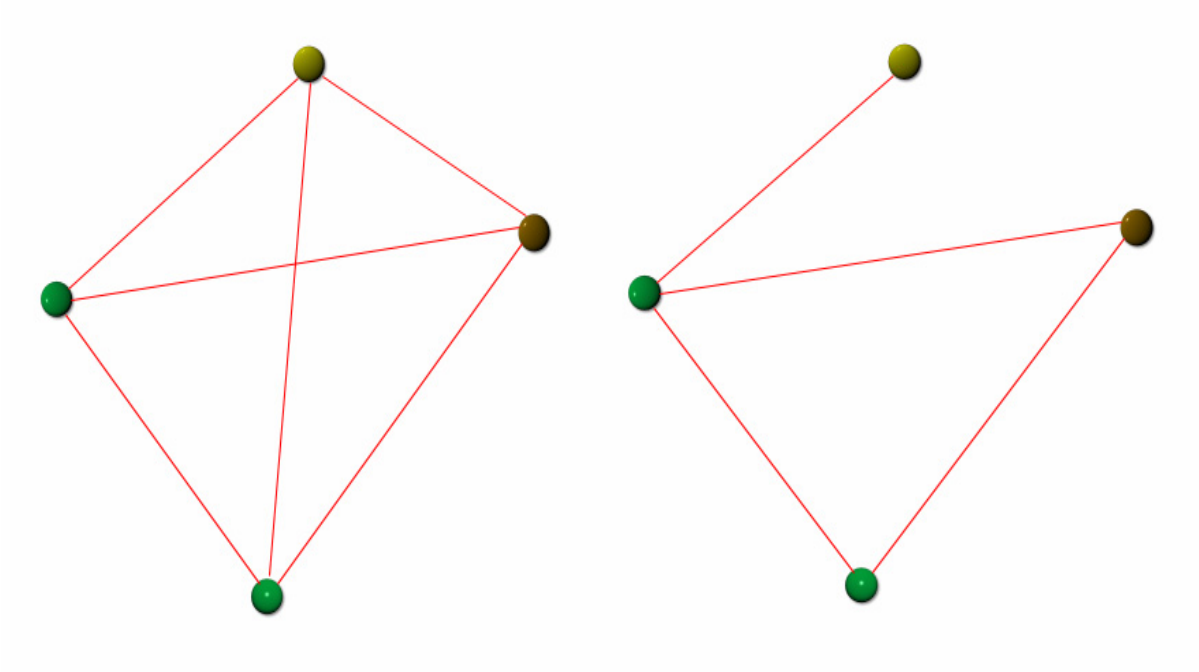
G = (j, 2n) per n = 1,2,….,6



Sottografi

I sottografi rappresentano una qualsivoglia partizione G' del grafo originario G.

Esempio 2.2



GRAFO G ⟶ **SOTTOGRAFO G'**

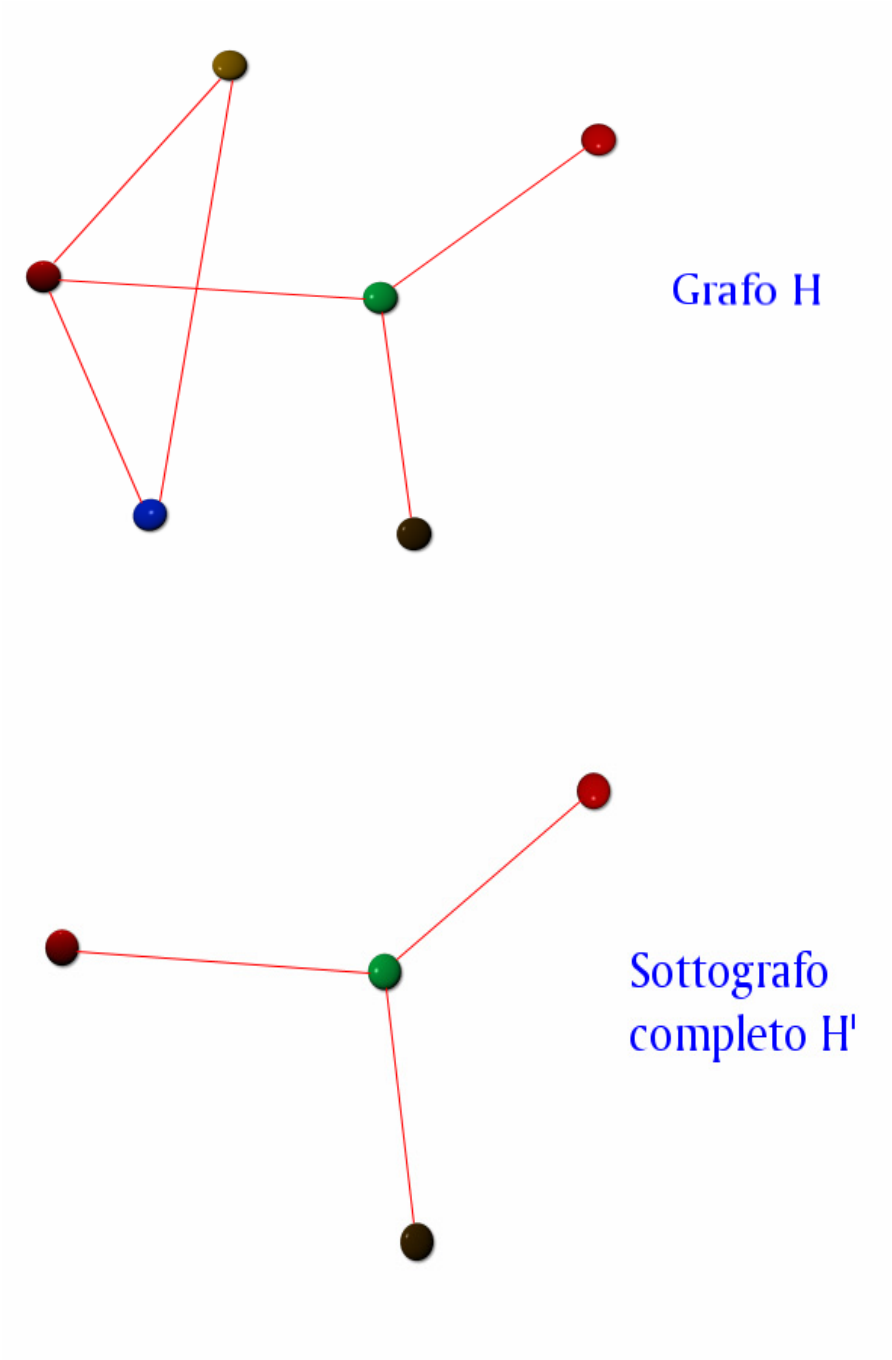
Sottografo completo

Considerando i nodi facenti parte della partizione G', se questi preservano gli archi originari, allora essi formano un sottografo completo.

L'esempio 2.2 non rappresenta un sottografo completo in quanto vi sono tutti i nodi originari, ma manca più di un arco. Quando avviene questo è immediato capire che la sola assenza di un arco determina un sottografo incompleto.

Nell'esempio 2.3 abbiamo invece un sottografo completo del grafo H.

Esempio 2.3



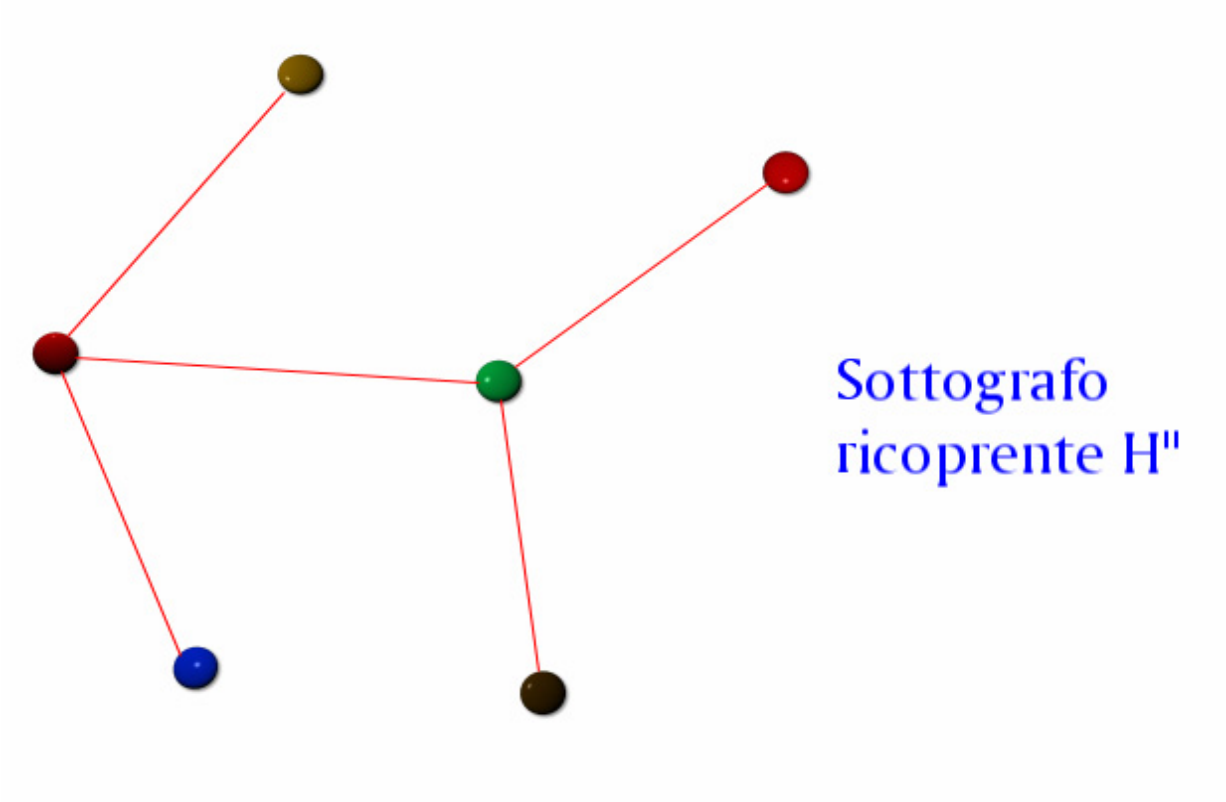
Faccio notare come l'esistenza di un arco sia legata alla previa esistenza di due nodi. Un nodo non connesso si dice **nodo isolato** o **nodo singolo**, mentre non esistono archi isolati.

Quando non siamo in presenza di un sottografo completo, questo si dice allora *indotto*.

Sottografo ricoprente

E' quel sottografo non completo che connette ciascun nodo originario. Riprendendo l'esempio precedente ci troveremmo di fronte ad un grafo simile:

Esempio 2.4



Non esistono dunque, in un sottografo ricoprente, **nodi esposti** o isolati.

In inglese, il termine *spanning* sta proprio ad indicare 'ricoprente'. Incontreremo più avanti questo vocabolo.

Ordine di un grafo

Il numero di nodi di un grafo G costituisce il suo ordine:

$$n = |V(G)|$$

Dimensione di un grafo

La dimensione di un grafo è data dal numero di archi m che possiede il grafo.

Considerando grafi semplici, possiamo stabilire un range di appartenenza per qualsiasi grafo:

$$0 \leq m \leq \binom{n}{2}$$

Ricordo che con n indichiamo il numero di nodi presenti nel grafo e che con il simbolo:

$$\binom{n}{2}$$

indichiamo tutte le possibili combinazioni di n oggetti presi due a due.
Dalle proprietà delle combinazioni, abbiamo che:

$$C(n,r) = \frac{n!}{(n-r)!\,r!}$$

dove

$$C(n,r) = \binom{n}{r}$$

ovvero, 'n su r' sta ad indicare tutte le combinazioni possibili di n oggetti presi a gruppi di r.
Risolvendo il nostro 'n su 2', avremo che:

$$\binom{n}{2} = \frac{n!}{(n-2)!\,2!} = \frac{[1 \cdot 2 \cdot 3 \cdot \ldots\ldots \cdot (n-2)] \cdot (n-1) \cdot n}{[1 \cdot 2 \cdot 3 \cdot \ldots\ldots \cdot (n-2)] \cdot 2!}$$

Che semplificando numeratore e denominatore diviene:

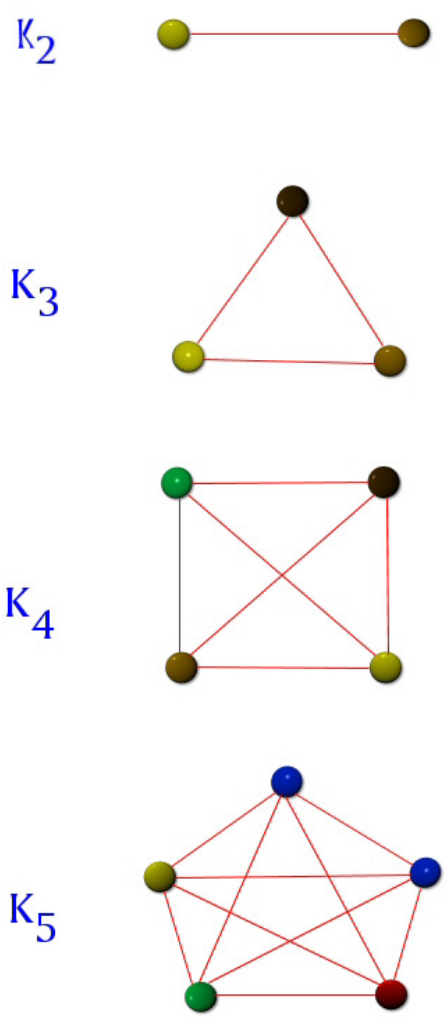
$$\frac{n \cdot (n-1)}{2}$$

Dato dunque un grafo semplice con n nodi ed m archi, possiamo dire a priori che il numero di archi può variare da un minimo di zero ad un massimo di n su 2. Nel caso fossimo in presenza del massimo numero di archi disponibile, avremmo un grafo completo.

Grafo completo o clique

Si dice completo quel grafo semplice per il quale non è più possibile aggiungere un arco senza ricrearne uno già presente.

Esempi di grafi completi



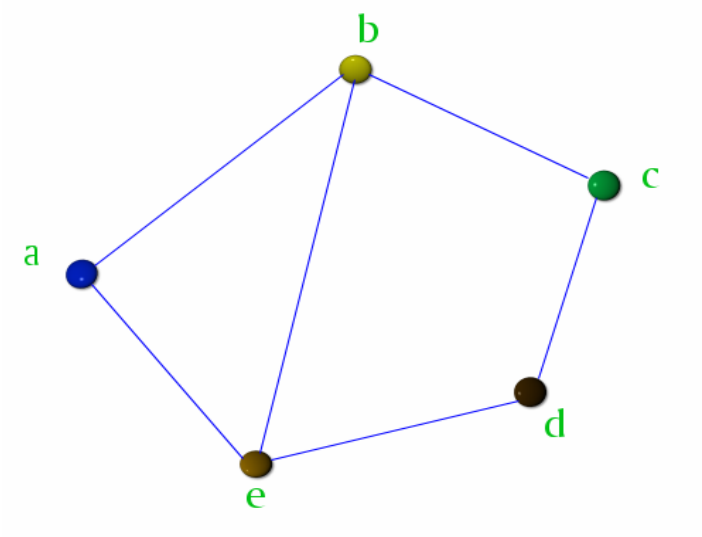
Con la lettera K ed il pedice n siamo soliti indicare una clique di ordine n, ovvero un grafo completo con n nodi.

Grafo complemento

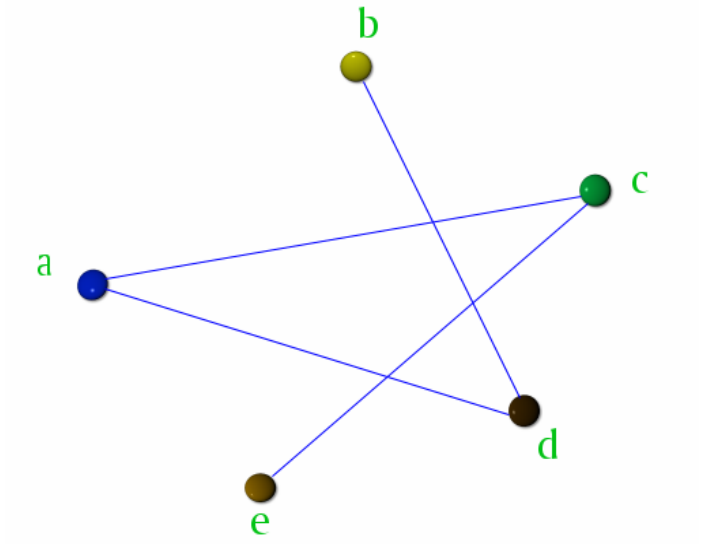
Un grafo complemento di G è un grafo formato dagli stessi nodi, ma da tutti gli altri archi non presenti in G.

Esempio 2.5

GRAFO G



e il suo GRAFO COMPLEMENTO



Teoria dei Grafi
Capitolo 3

Coloring

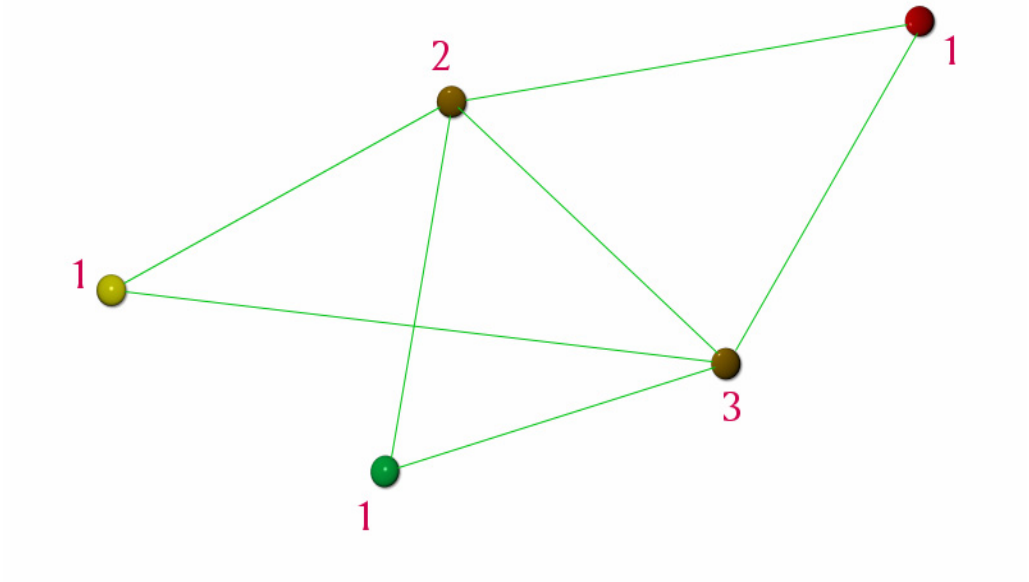
Numero cromatico di un grafo

E' il minimo numero di colori necessari tale che due nodi adiacenti non abbiano mai lo stesso colore.

Con etichetta e colore indicheremo la stessa cosa ovvero, lo 'status' di un nodo.

Esempio 3.1

Nel seguente grafo possiamo notare che con 3 colori o come in questo caso 3 numeri, siamo in grado d'etichettare tutti i nodi:



Come si può vedere dall'esempio 3.1, non vi sono nodi adiacenti che abbiano la stessa etichetta (o numero da 1 a 3). Dunque abbiamo colorato correttamente il grafo.

Notiamo adesso una cosa importante; non è possibile colorare questo grafo con due colori poiché avremo sempre due nodi 'in conflitto' tra loro e se facciamo caso alle cliques presenti nel grafo, osserviamo che queste sono tutte di ordine tre e che in particolare sono le clique di ordine massimo.

Quest'osservazione ci permette di dire che data una clique di ordine n in un grafo G, non sarà possibile colorare quest'ultimo con un numero di colori minori di n.

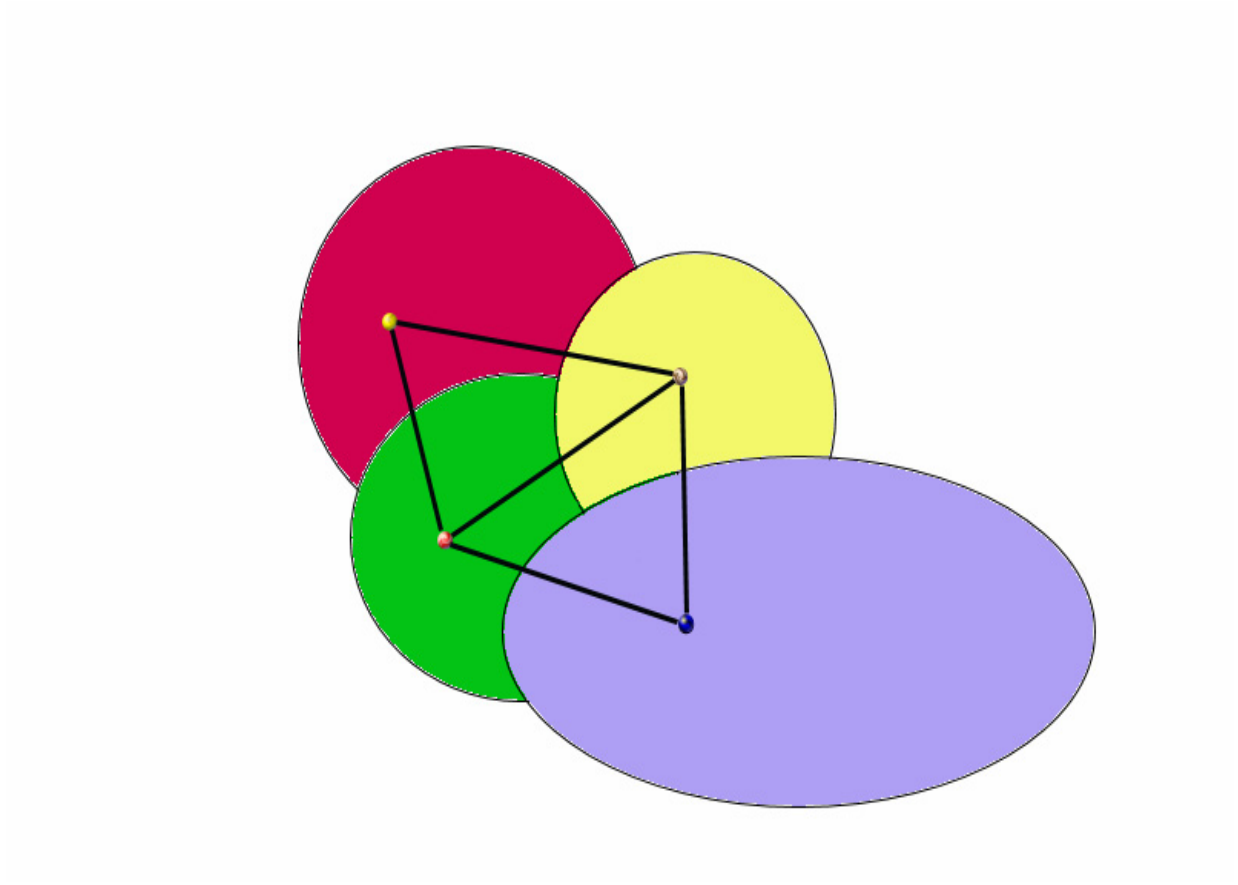
Si usa indicare il numero cromatico di un grafo con la lettera greca 'chi': χ (G).

Il problema del coloring ha avuto enorme successo con il famoso **"Teorema dei quattro colori",** il quale ha stabilito che per colorare una qualsiasi mappa bidimensionale sono sufficienti per l'appunto solamente quattro colori.

Dalla figura 3.1 si capisce come si possa passare dal problema reale ad un problema su grafi.

Figura 3.1

Supponiamo che ogni regione debba essere colorata con un unico colore e supponiamo d'indicare ogni regione con un nodo e l'adiacenza tra una regione ed un'altra con un arco che connette i due nodi.

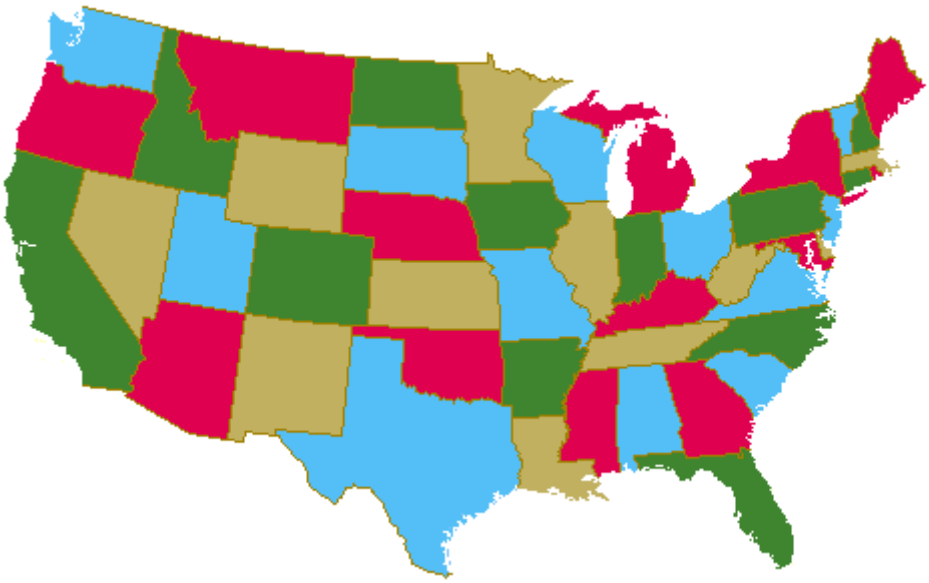


Si vede bene come si possa risolvere il problema di colorazione minima di una situazione reale con l'ausilio della teoria dei grafi.

Quest'altra figura che potrete trovare al seguente sito:

- http://www.math.gatech.edu/~thomas/FC/fourcolor.html

vi chiarirà sicuramente le idee:



Se volete approfondire l'argomento del coloring, qui ci sono una serie di links che potrebbero esservi utili per cominciare:

- dal **The Geometry Junkyard** http://www.ics.uci.edu/~eppstein/junkyard/color.html

un altro buon punto di partenza contenente programmi di coloring e generazione di grafi è:

- http://www.cs.ualberta.ca/~joe/Coloring/

per quanto riguarda invece il problema dei quattro colori ed il suo conseguente teorema:

- http://www.math.gatech.edu/~thomas/FC/fourcolor.html

un ultimo sito degno di nota è quello della *University of Southern Denmark* nel quale potrete trovare altro materiale interessante;

- http://www.imada.sdu.dk/Research/Graphcol/

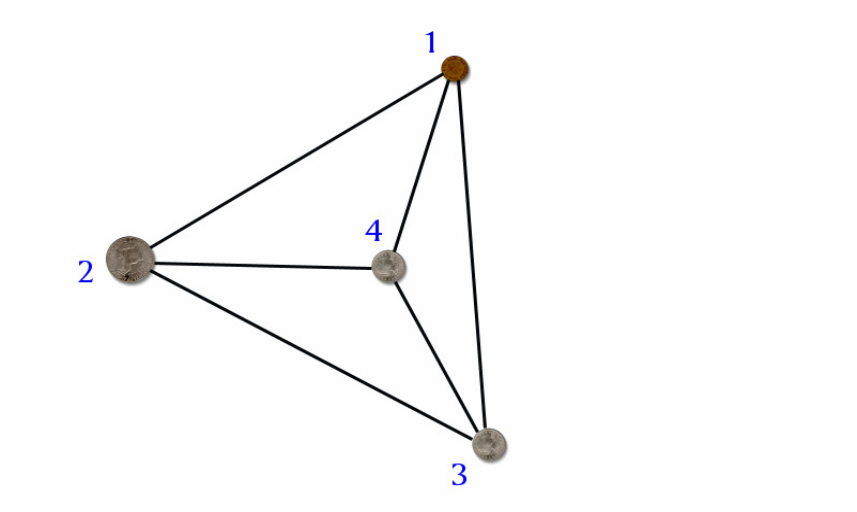
Grafo R-partito

Indichiamo con la terminologia di grafo R-partito, quel grafo che ha una clique di ordine massimo pari ad R.

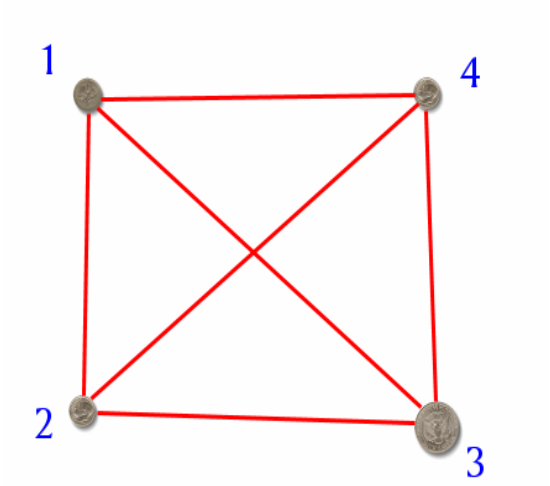
Un triangolo sarà ad esempio un grafo 3-partito, mentre un quadrato sarà un grafo 2-partito o bipartito.

Questo grafo è invece 4-partito.

Figura 3.2



Se lo si vede da quest'altra angolazione, si capisce perché…



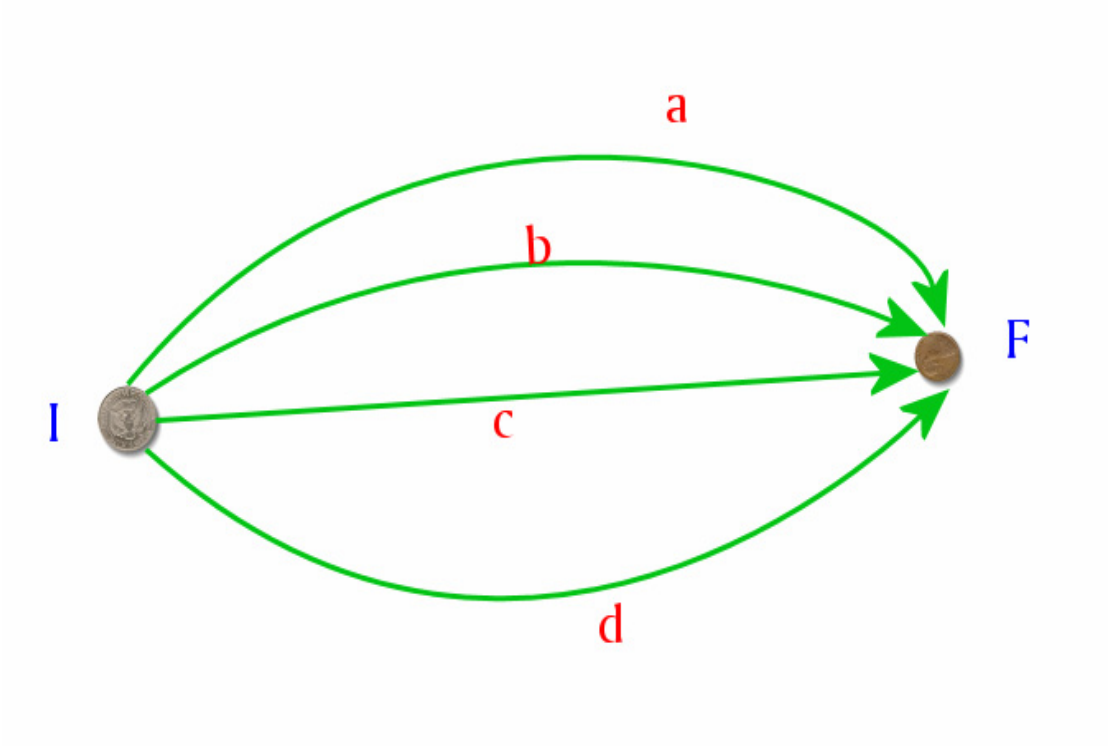
Cammino

E' il percorso che va da un nodo ad un altro.

Cammini indipendenti

Due cammini sono indipendenti se la loro intersezione è nulla.

Figura 3.3



Trail

Cammino nel quale non si passa due volte sullo stesso arco.

Path

E' un cammino che non passa due volte sullo stesso arco o sullo stesso nodo.

Ciclo o path chiuso

Path nel quale il nodo iniziale è uguale a quello finale.

Grafo aciclico

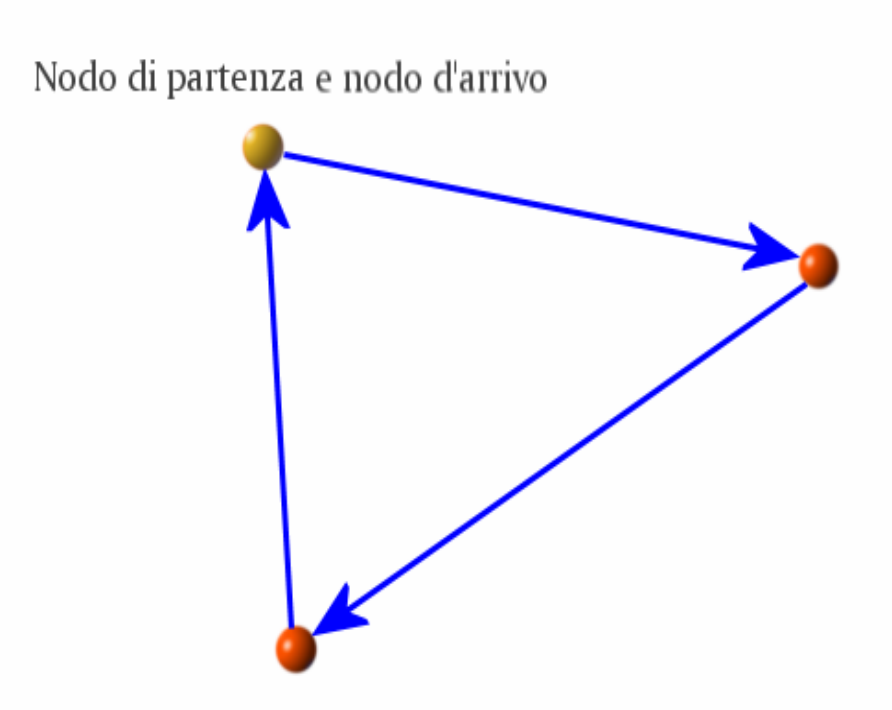
Grafo che non contiene cicli.

Grafo connesso

Un grafo si dice connesso se esiste un cammino tra qualunque coppia di nodi.

Cammino chiuso

Cammino che ha come vertice di arrivo lo stesso vertice di partenza.



Vediamo ora un'importante caratteristica dei grafi, quella dell'isomorfismo. Questa proprietà ci permette di capire se due grafi disegnati in modo diverso, rappresentino in realtà lo stesso grafo.

Isomorfismo tra grafi

Vediamo quando due grafi si dicono isomorfi tra loro.

Condizione necessaria N° 1

Due grafi sono isomorfi se hanno lo stesso ordine e la stessa dimensione. Questo significa che devono avere lo stesso numero di vertici e di archi.

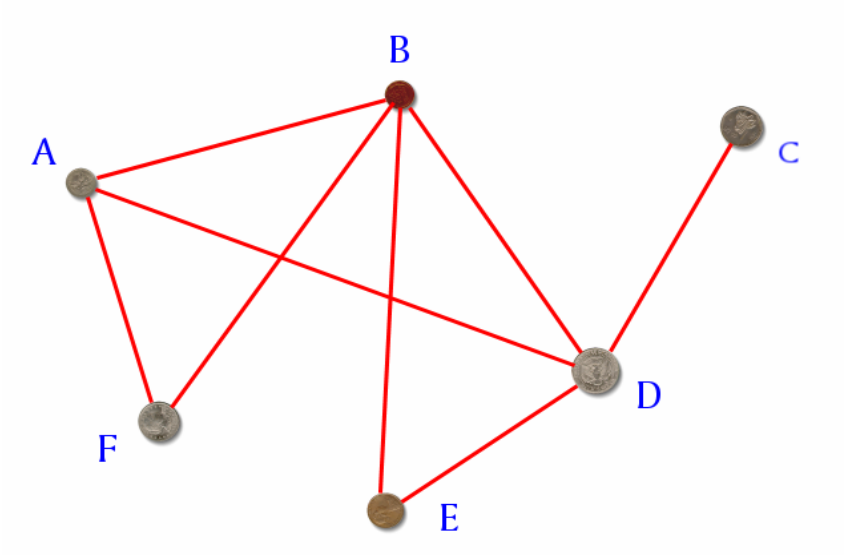
Condizione necessaria N° 2

Due grafi si dicono isomorfi se hanno la stessa sequenza grafica.

Sequenza grafica

E' il vettore ordinato dei gradi dei nodi componenti il grafo. L'ordine va dal più alto al più basso.

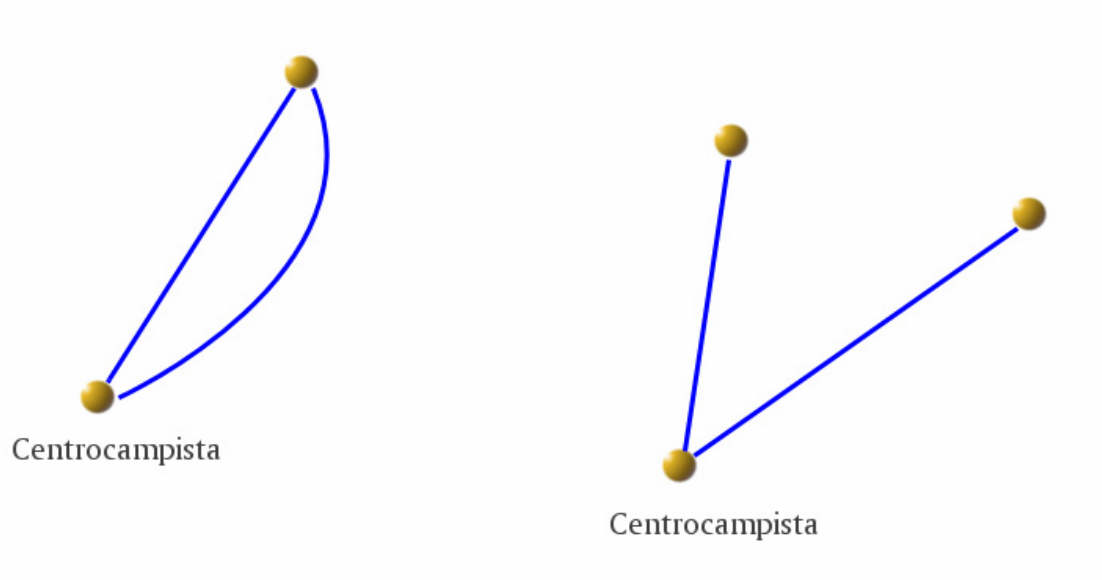
Esempio 3.2



Grado di un nodo

E' il numero dei nodi al quale è collegato il generico nodo j.

Faccio notare come questa definizione non sia corretta solo nel caso di grafi multipli.



Il grado del centrocampista è in entrambi i casi pari a due, ma noi considereremo solo il secondo. Infatti, in quest'ultimo caso il centrocampista è connesso a due nodi, mentre nel primo ad uno solo.

Ora, nel caso dell'esempio 3.2 abbiamo i seguenti gradi dei nodi:

- A = 3
- B = 4
- C = 1
- D = 4
- E = 2
- F = 2

La corrispettiva sequenza grafica è: { 4,4,3,2,2,1 }.

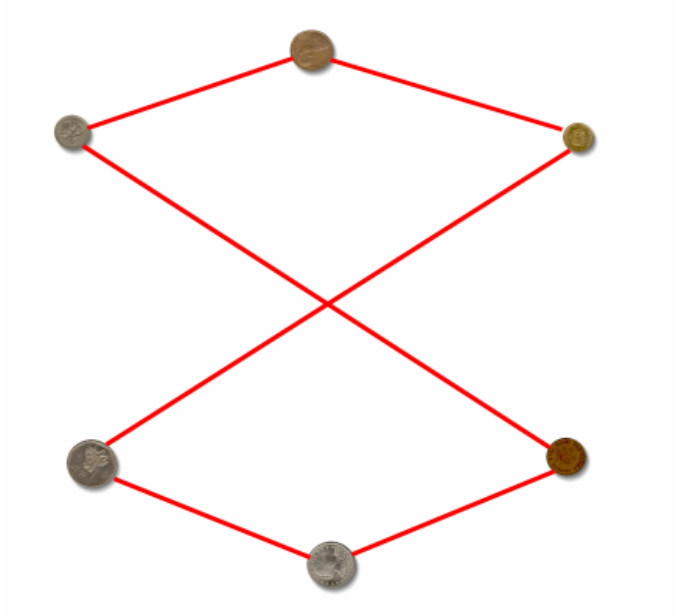
IMPORTANTE

Bisogna fare attenzione a notare come due grafi isomorfi abbiano la stessa sequenza grafica, ma due grafi con la stessa sequenza grafica non è detto che siano isomorfi.
Le due condizioni enunciate prima sono infatti solo necessarie, ma non sufficienti.

Grafo regolare

Se la sequenza grafica è composta da tutti numeri uguali il grafo si dice regolare.

Esempio 3.3

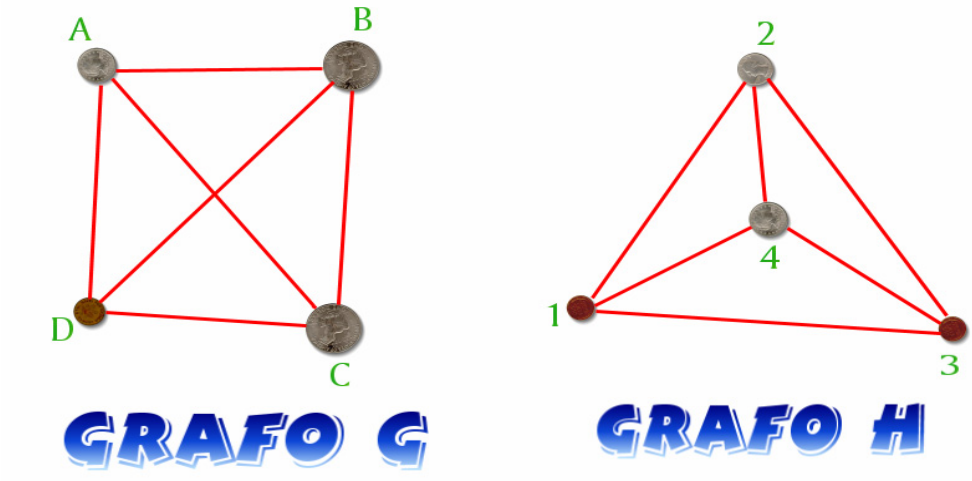


Nell'esempio 3.3 abbiamo un grafo regolare. Non esistono infatti nodi il cui grado sia diverso da 2.

In sintesi, due grafi isomorfi soddisfano una funzione f: V → V' che permette di associare biunivocamente i vertici di un grafo a quelli dell'altro.

Nel prossimo esempio vediamo come sia possibile stabilire una corrispondenza biunivoca tra due grafi G ed H e come la loro sequenza grafica sia la stessa.

Esempio 3.4



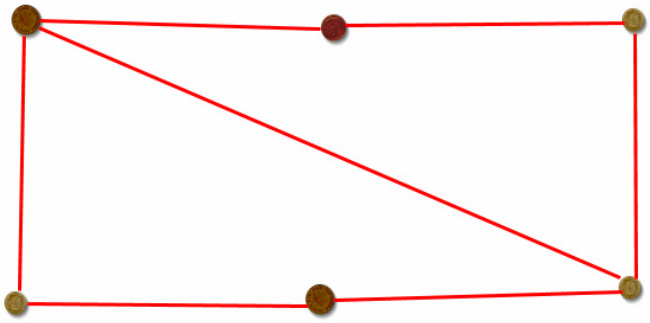
Una possibile funzione di biunivocità che leghi i nodi del grafo G a quelli del grafo H è la seguente:

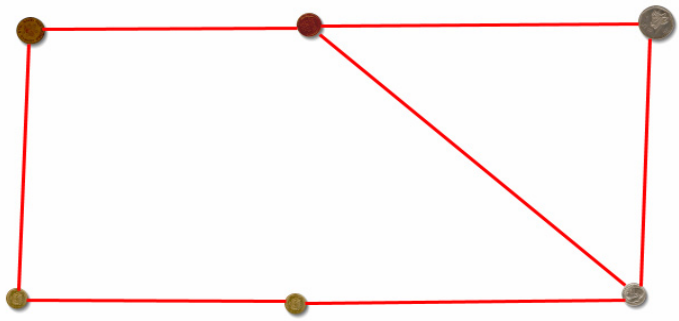
- A → 2
- B → 4
- C → 3
- D → 1

La rispettiva sequenza grafica del grafo G è { 3,3,3,3 }, essendo i gradi del nodo A, B, C e D uguali a 3. Per quanto riguarda il grafo H, la sequenza grafica è la medesima.
Osserviamo infine che i due grafi sono regolari.

Esempio 3.5

In questo esempio invece verifichiamo come la condizione N° 2 non rappresenti una condizione sufficiente, ma solo necessaria.
Vediamo come due grafi con la stessa sequenza grafica possano risultare non isomorfi.





Entrambi i grafi hanno la stessa sequenza grafica: { 3,3,2,2,2,2 }, ma come si vede facilmente dall'assenza nel primo grafo di una clique di ordine 3, non è possibile stabilire una corrispondenza biunivoca tra loro.

Teoria dei Grafi
Capitolo 4

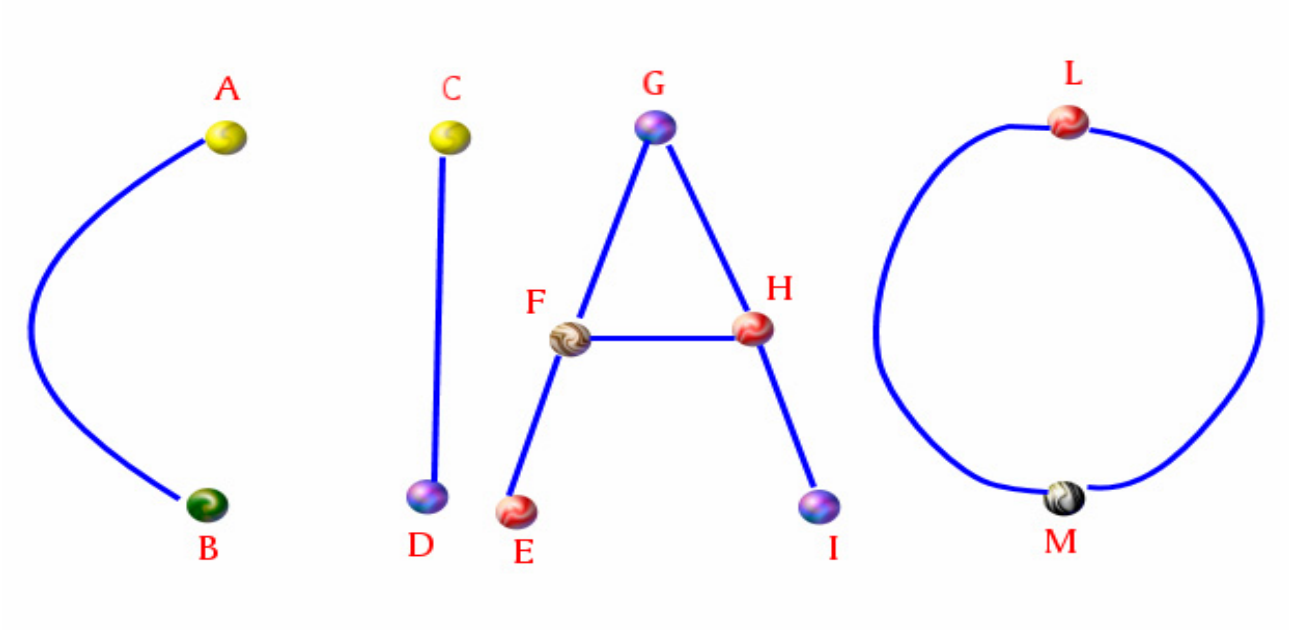
Grafo orientato

In un grafo orientato gli archi possono essere percorsi in un solo verso.

Matrice d'adiacenza

Questa matrice quadrata ha dimensioni n×n, dove con n indichiamo il numero di nodi del grafo. Nella generica casella ij che connette il nodo i al nodo j metteremo zero se i due nodi non sono connessi o un numero positivo che ci dica con quanti archi i due nodi sono connessi.

Esempio 4.1



La rispettiva matrice d'adiacenza del grafo dell'esempio 4.1 è la seguente:

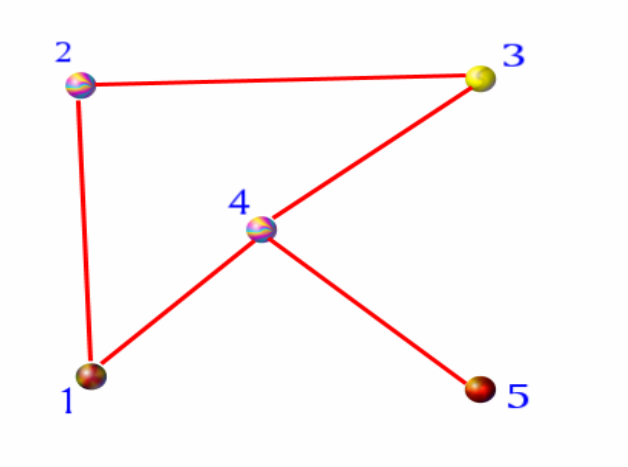
	A	B	C	D	E	F	G	H	I	L	M
A	0	1	0	0	0	0	0	0	0	0	0
B	1	0	0	0	0	0	0	0	0	0	0
C	0	0	0	1	0	0	0	0	0	0	0
D	0	0	1	0	0	0	0	0	0	0	0
E	0	0	0	0	0	1	0	0	0	0	0
F	0	0	0	0	1	0	1	1	0	0	0
G	0	0	0	0	0	0	1	0	1	0	0
H	0	0	0	0	0	1	1	0	1	0	0
I	0	0	0	0	0	0	0	1	0	0	0
L	0	0	0	0	0	0	0	0	0	0	2
M	0	0	0	0	0	0	0	0	0	2	0

Osserviamo che ci troviamo di fronte ad un grafo multiplo, la lettera 'O' viene infatti creata connettendo due nodi con più di un arco.
Se esaminiamo invece solo grafi semplici, possiamo estrapolare per la matrice d'adiacenza le seguenti proprietà:

Proprietà 1

La somma degli elementi della generica riga r ci dice con quanti nodi è connesso il nodo della riga.

Esempio 4.2



La rispettiva matrice d'adiacenza ci dice che il nodo 1 è collegato a 2 nodi, il nodo 5 ad uno solo e il nodo 4 è quello che ha più connessioni. Lo stesso ragionamento vale per gli altri due nodi.

	1	2	3	4	5	
1	0	1	0	1	0	2
2	1	0	1	0	0	2
3	0	1	0	1	0	2
4	1	0	1	0	1	3
5	0	0	0	1	0	1

Proprietà 2

La matrice d'adiacenza nel caso di grafi semplici è sempre **unimodulare** ovvero, il determinante di qualsiasi suo minore sarà pari ad 1,0 o -1.

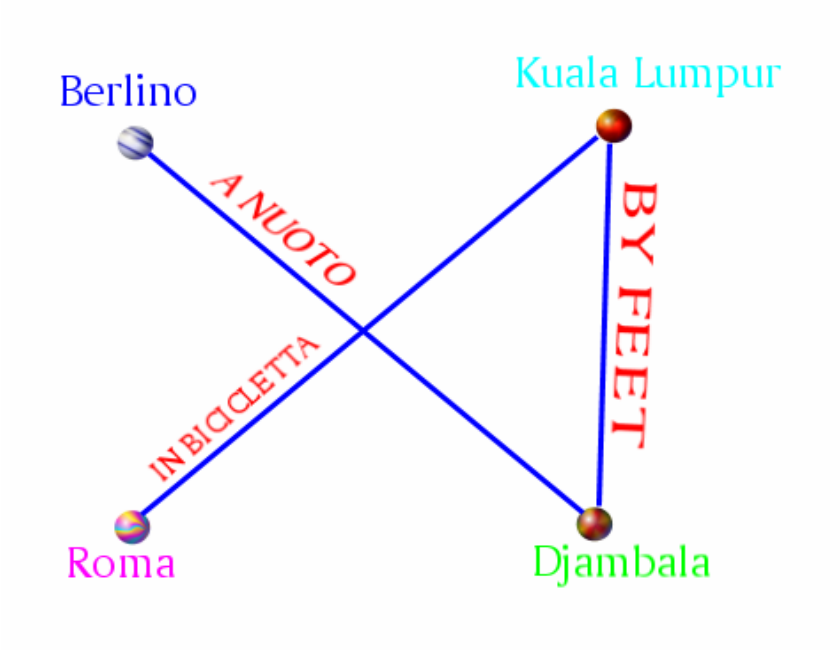
Proprietà 3

La matrice d'adiacenza è una matrice simmetrica.

Matrice d'incidenza

Questa matrice è invece n×m dove sulle righe abbiamo i nodi e sulle colonne gli archi del grafo.

Esempio 4.3



La matrice viene riempita in questo modo:

- sulla riga di ogni nodo si mette un 1 in corrispondenza dell'arco con il quale viene connesso ad altri nodi

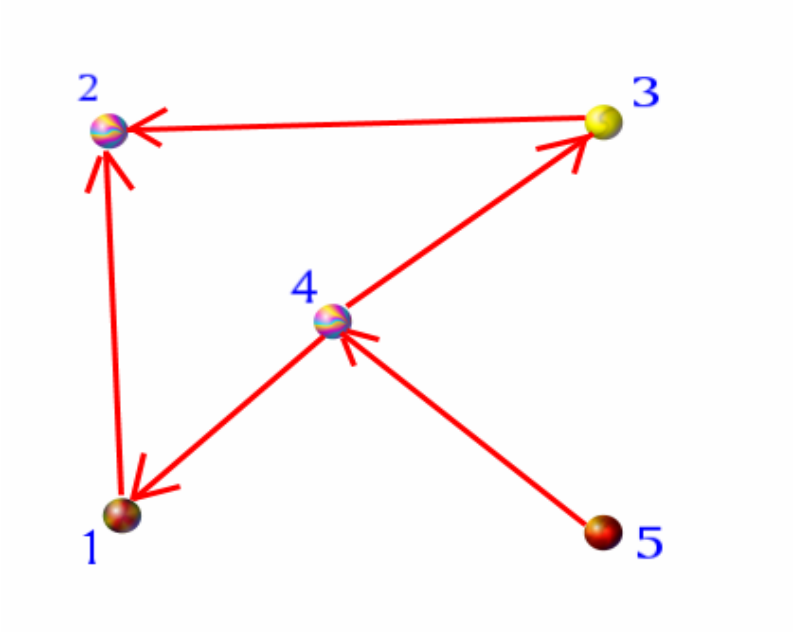
	A NUOTO	IN BICICLETTA	BY FEET
Berlino	1	0	0
Roma	0	1	0
Kuala Lumpur	0	1	1
Djambala	1	0	1

Proprietà

La matrice d'incidenza ha come somma della generica riga r il grado del nodo alla quale appartiene. Roma ha grado 1 come Berlino, mentre Djambala e Kuala Lumpur hanno grado 2.

Ora, nel caso di grafi orientati, la matrice d'adiacenza non sarà più simmetrica in quanto l'1 verrà messo solo dove l'arco è percorribile.

Esempio 4.4

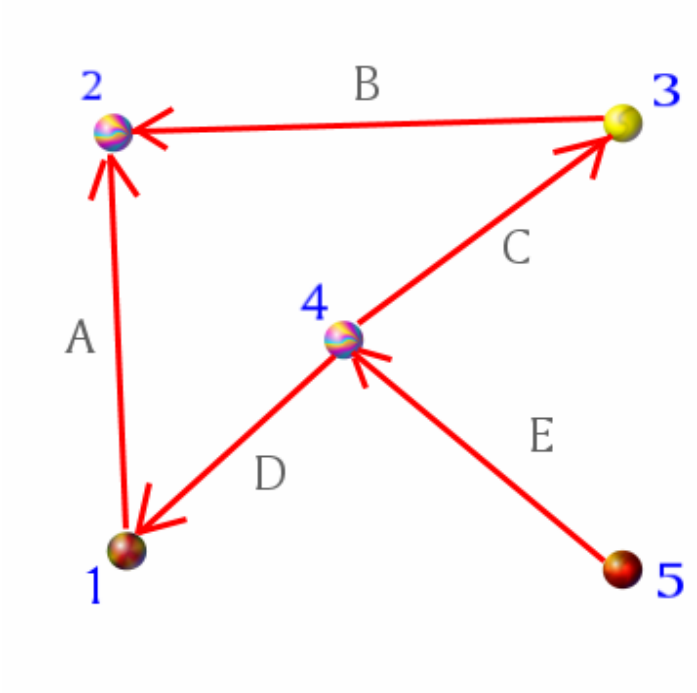


	1	2	3	4	5
1	0	1	0	0	0
2	0	0	0	0	0
3	0	1	0	0	0
4	1	0	1	0	0
5	0	0	0	1	0

Come si vede può vedere, la matrice d'adiacenza è asimmetrica.

Se volessimo descrivere il grafo con la matrice d'incidenza dovremmo prima etichettare gli archi e poi mettere un '1' per un arco uscente dal nodo ed un '-1' per un arco entrante.

Esempio 4.5



	A	B	C	D	E
1	1	0	0	-1	0
2	-1	-1	0	0	0
3	0	1	-1	0	0
4	0	0	1	1	-1
5	0	0	0	1	0

HANDSHAKING LEMMA

$$\sum_{i=0}^{n} d(v_i) = 2m$$

Questo significa sostanzialmente una cosa: un arco connette due nodi, né 3, né 1, né altro…solo 2 nodi.

Questo lemma è verificabile per grafi non orientati, sulle matrici d'adiacenza e d'incidenza; la somma di tutti gli '1' è pari a 2m, cioè al doppio del numero degli archi presenti nel grafo.
Ciò significa che, rappresentando gli '1' una connessione e dunque la presenza di un arco su due nodo, la somma dei gradi di tutti i nodi del grafo è pari al doppio degli archi.

Una conseguenza importante di questo lemma è che, essendo 2m un numero pari, non potrà esistere nella sommatoria dei gradi dei nodi una quantità dispari di gradi dispari, altrimenti la somma non sarebbe pari.

Se ad esempio avessimo 2+4+5+6+7+9 saremmo certi dell'inesistenza di questo grafo (sempre considerando che sia semplice) poiché la sommatoria dei gradi è pari a 33. Abbiamo appunto 3 nodi di grado dispari.

Nel caso dei grafi dell'esempio 4.2 e 4.3 abbiamo rispettivamente le seguenti sommatorie per il grado dei nodi:

- 2+2+2+3+1 =10
- 1+1+2+2 = 6

Teoria dei Grafi
Capitolo 5

Crescita delle funzioni

Supponiamo che f(x) e g(x) siano due funzioni da R → R.

Notazione 'big-O'

$$f(x) = O(g(x)) \quad se \exists\ C,K \quad t.c. \quad |f(x)| \leq C\,|g(x)| \quad \forall\ x > k$$

Esempio 5.1

$f(x) = 2x^2 + 3x + 9$

possiamo maggiorare la f(x) in questo modo: $0 \leq 2x^2 + 3x + 9 \leq 2x^2 + 3x^2 + 9x^2 = 14\,x^2$

questo significa che la nostra C = 14, che $14\,x^2 \geq f(x) = O(x^2)$.

Dunque la funzione f(x) è un 'big-O' della funzione x^2 e in particolare, la funzione $14\,x^2$ andrà ad infinito più velocemente di f(x) a partire da un determinato valore di x; quello dove le due funzioni f(x) e g(x) s'intersecano.

Quando abbiamo visto a pagina 13 che il massimo numero di archi per un grafo semplice è pari a:

$$\frac{n \cdot (n-1)}{2}$$

allora anche in questo caso possiamo definire una notazione 'big-O' per questa funzione. Il risultato sarà che il massimo numero di archi per un grafo è uguale ad $O(n^2)$.

Quando definiamo una notazione big-O per una funzione f(x), stiamo stabilendo una complessità computazionale per la funzione f(x).

Ad esempio, la funzione fattoriale **$n! = O(n^n)$** infatti, possiamo maggiorare il fattoriale in questo modo:

$n! = 1 \cdot 2 \cdot 3 \cdot \ldots\ldots \cdot (n-1) \cdot n \leq n \cdot n \cdot n \cdot n \cdot n \cdot \ldots\ldots\ldots\ldots\ldots \cdot n \cdot n \cdot n \cdot n = n^n$

dove n^n è la moltiplicazione di n per se stesso n volte. Abbiamo perciò maggiorato, 1 con n, 2 con n, 3 con n, ……, (n-1) con n e infine n con se stesso.

Utilizzando questi risultati possiamo dire che **$\log n! = O(n \log n)$**:

$\log n! \leq \log n^n$ va da sé per quello detto sopra

e per le proprietà dei logaritmi, $\log n^n = n \log n$.

Complessità della somma di due funzioni

Quando abbiamo una somma di funzioni, la complessità totale è data dalla complessità maggiore tra tutte le funzioni.

$(f_1(x) + f_2(x))(x) = O(\max (g_1(x)+g_2(x))$

Ad esempio:

- $2n^3 + 7n^2 + 4n! = O(n^n)$
- $2n^3 + 7n^2 = O(n^3)$
- $4\log(n^2) + 7n^3 = O(n^3)$

Complessità del prodotto di due funzioni

La complessità del prodotto di più funzioni è pari al prodotto delle complessità di ciascuna funzione.

$f_1(x)\, f_2(x) = O(g_1(x) g_2(x))$

Vediamo qualche esempio:

- $n^3 \log n = O(n^3 \log n)$
- $4n^2 \log n! = O(n^2 \cdot n \log n) = O(n^3 \log n)$

Vediamo adesso qualche esempio che preveda sia somme che prodotti:

- $7n + 8n^2\log n + (4n!)^3 = O(n + n^2\log n + (n^n)^3) = O(n^{3n})$
- $8n!n^2\log(n!) + n^n = O(n^n \cdot n^2 \cdot n \log n + n^n) = O (n^n \cdot n^2 \cdot n \log n) = O(n^n \cdot n^{3n} \cdot \log n)$

Qui sotto ho messo una tabella riassuntiva per le complessità di alcune funzioni:

PER n CHE VA AD ∞		
LA PIU' VELOCE	n^n	LA PIU' COMPLESSA
	$n!$	
	2^n	
	n^2	
	$n\log n$	
	n	
LA PIU' LENTA	$\log n$	LA MENO COMPLESSA

Vediamo adesso le combinazioni, le disposizioni e le permutazioni.

Combinazioni o coefficiente binomiale

$$C(n,r) = \frac{n!}{(n-r)!\ r!}$$

Questa notazione sta ad indicare il numero possibile di combinazioni di n oggetti presi a gruppi di r.

Come avevamo visto a pagina 13, il massimo numero di archi per un grafo semplice era pari a

$$\frac{n \cdot (n-1)}{2}$$

Se avessimo ad esempio 18 macchine a disposizione nel nostro garage, ma nostra moglie ne prende 16 al giorno, a noi ne rimarrebbero solo 2. Supponendo che la scelta di nostra moglie sia assolutamente casuale e che prima o poi riusciremmo a guidarle tutte, vogliamo sapere quanti sono i giorni che ci vedranno uscire ogni volta con un accoppiamento diverso.

Sono circa 5 mesi, più esattamente 153 giorni.

Sottolineo il fatto che con questa formula si considerano uguali le seguenti combinazioni e dunque, verranno contate una sola volta:

- { a,b,c }
- { a,c,b }
- { b,a,c }
- { b,c,a }
- { c,a,b }
- { c,b,a }

Infatti:

$$\begin{bmatrix} 3 \\ 3 \end{bmatrix} = C(3,3) = \frac{3!}{(3-3)!\ 3!} = \frac{6}{0!6} = 1$$

Ricordo che **0! = 1** per definizione.

Se volessimo invece considerare differenti le 6 combinazioni viste sopra, allora dovremmo parlare di r-permutazioni.

r-permutazioni o disposizioni

$$P(n,r) = \frac{n!}{(n-r)!}$$

In questo caso vediamo come si possa tener conto in ogni combinazione della posizione di ciascun oggetto.

Nel caso precedente di 3 oggetti presi 3 a 3:

- P(3,3) = 6 / 1 = 6

che sono tutte e sei le combinazioni descritte sopra.

Ora, nel caso in cui ci trovassimo n = r parleremo di permutazioni.

Permutazioni

P(n,n) = n!

E vediamo adesso due formule.

Formula N°1

C(n,r) = C(n,n-r)

Questo perché:

$$C(n,n-r) = \frac{n!}{(n-r)!\ [n-(n-r)]!} = \frac{n!}{r!\ (n-r)!}$$

Che è proprio pari a C(n,r).

Formula N°2

P(n,r) = C(n,r) P(r,r)

Ovvero, il numero di r-permutazioni è ottenibile dal prodotto delle r-combinazioni per le permutazioni.

Lemma di Pascal

$$\begin{bmatrix} n \\ r \end{bmatrix} = \begin{bmatrix} n-1 \\ r \end{bmatrix} + \begin{bmatrix} n-1 \\ r-1 \end{bmatrix}$$

Ed infine vediamo il ben noto binomio di Newton.

Teorema binomiale o formula di Newton

$$(a+b)^n = \sum_{i=0}^{n} \begin{bmatrix} n \\ i \end{bmatrix} b^i \; a^{n-i}$$

Teoria dei Grafi
Capitolo 6

Algoritmo

Un algoritmo consiste in una procedura prestabilita per la risoluzione di un problema in un numero finito di passi.

Vediamo dunque le complessità computazionali di alcuni algoritmi.

ALGORITMO DI RICERCA DEL MASSIMO

Descrivo qui sotto l'algoritmo nella forma del cosiddetto 'pseudocodice'. Lo pseudocodice accomuna quasi tutti i linguaggi di programmazione ovvero, descrive le operazioni base di:

- FAI QUESTO fin tanto che ACCADE QUEST'ALTRO
- PER questa condizione ESEGUI QUESTE OPERAZIONI
- SE accade questo ALLORA fai quest'altro

Che corrispondono in genere alle più note diciture di:

- **do while**
- **for**
- **if**

Bene, a questo punto vediamo l'algoritmo:

```
max = A1;

for i = 2 to n
    if max < Ai   then max = Ai;
```

Esaminiamo cosa fa l'algoritmo:

1. max = A_1 ha complessità 1...viene eseguito solo una volta;
2. for i = 2 to n
   if max < A_i then max = A_i

   per valori invece che vanno da 2 ad n, l'algoritmo esegue ogni volta il controllo sulla disuguaglianza max < A_i.....da 2 ad n sono (n-1) passi, il controllo ha complessità singola, perciò la complessità è (n-1) · 1 = (n-1)

La complessità dell'algoritmo è dunque [1+(n-1)] = O(n).

ALGORITMO DI RICERCA BINARIA

```
i = 1;
j = n;

while (i < j)
begin
   m = [ (i+j) / 2]
   if x >Am then i = m+1
   else j = m
end
   if x = Ai then location = i
else
   location = 0
```

Supponendo una lista ordinata di elementi { A_1, A_2, A_3, ……., A_n }, l'algoritmo di ricerca binaria troverà l'elemento x in un tempo computazionale pari ad $O(\log_2 n)$.

L'algoritmo opera praticamente una 'divisione' della lista ordinata { A_1, A_2, A_3, ……., A_n }. Se n è dispari, sarà considerata la parte intera inferiore di m.
Se la disuguaglianza viene verificata si provvederà a dividere allo stesso modo la seconda metà altrimenti la prima.

Quest'esempio aiuterà a capire meglio l'algoritmo.

Esempio 6.1

Nella lista ordinata LIST vogliamo trovare il numero 19.

LIST = { 1,2,4,5,6,9,10,12,15,18,20,24 }

x = 19

m = parte inferiore di 13/2….sei elementi per parte.

Dividiamo a metà LIST:

{ 1,2,4,5,6,9 } { 10,12,15,18,20,24 }

9 < 19

dividiamo a metà la seconda parte:

{ 10,12,15 } { 18,20,24 }

15 < 19

dividiamo ancora a metà la seconda parte:

{ 18 } { 20,24 }

18 < 19

ancora una divisione della seconda lista:

{ 20 } { 24 }

20 > 19 → 20 < 19 < 24

Nel peggiore dei casi dovremmo dividire le liste 2^m volte e la certezza d'arresto dell'algoritmo sarà data dalla seguente disuguaglianza:

- $[n/2^m] \leq 1$

quando avviene questo la lista sarà composta da un solo elemento e dunque il termine dell'algoritmo sarebbe certo.

Da quest'ultima disuguaglianza possiamo estrapolare la n facendo il logaritmo:

$$(\log_2 n) \leq [\log_2 (2^m)] = m$$

Questo significa che non appena m sarà maggiore o uguale al logaritmo in base 2 di n, l'algoritmo avrà termine.

ALGORITMO DI RICERCA LINEARE

Questo è il più semplice algoritmo di ricerca.
Nel caso peggiore la complessità è O(n) ovvero, esamina tutti gli elementi della lista fino a quando non trova l'elemento cercato.

ALGORITMO BUBBLE SORT

Quest'algoritmo non è di ricerca, ma di ordinamento e per far questo esegue un confronto esaustivo su determinate coppie della lista da ordinare.

```
int i,j

for ( i=0; i < n-1; i++ )
  for ( j=n-i; j>i; j-- )
    if (A[j] < A[j-1] )
    {
       temp = A [j];
       A[j] = A[j-1];
       A[j-1] = temp;
    }
end
```

Supponiamo di avere ad esempio la seguente lista da ordinare:

- LIST = { 8,2,45,1,23,40,6,5,18,21 }

partendo dal findo di LIST confronteremo via via tutte le coppie (j-1,j) e se queste non sono in ordine crescente le invertiremo:

- LIST = { 8,2,45,1,23,40,6,5,18,21 } OK
- LIST = { 8,2,45,1,23,40,6,5,18,21 } OK

- LIST = { 8,2,45,1,23,40,5,6,18,21 }
- LIST = { 8,2,45,1,23,5,40,6,18,21 }
- LIST = { 8,2,45,1,5,23,40,6,18,21 }
- LIST = { 8,2,45,1,5,23,40,6,18,21 } OK
- LIST = { 8,2,1,45,5,23,40,6,18,21 }
- LIST = { 8,1,2,45,5,23,40,6,18,21 }
- LIST = { 1,8,2,45,5,23,40,6,18,21 }
- LIST = { 1,8,2,45,5,23,40,6,18,21 } OK
- LIST = { 1,8,2,45,5,23,40,6,18,21 } OK
- LIST = { 1,8,2,45,5,23,6,40,18,21 }
- LIST = { 1,8,2,45,5,6,23,40,18,21 }
- LIST = { 1,8,2,45,5,6,23,40,18,21 } OK
- LIST = { 1,8,2,5,45,6,23,40,18,21 }
- ……
- …..
- ……..
- LIST = { 1,2,5,6,8,18,21,23,40,45 }

E' facile capire come la complessità di quest'algoritmo possa essere un $O(n^2)$ in quanto, nel caso peggiore esamineremo (n-1) confronti per n elementi.

ALGORITMO D'INSERZIONE

Anche in questo caso ci troviamo di fronte ad un algoritmo d'ordinamento.
La sua complessità è $O(n^2)$ e ciò che fa è scorrere la lista da ordinare e trovare un elemento (j+1) minore del precedente elemento j. Riposiziona (o inserisce) l'eventuale elemento 'anomalo' nella lista e continua fino alla fine di quest'ultima.

Se ad esempio ci troviamo di fronte a LIST = { 24,14,3,7 } si avrà:

- LIST = { 24,14,3,7 } OK
- LIST = { 14,24,3,7 }
- LIST = { 3,14,24,7 }
- LIST = { 3,7,14,24 }

Il riposizionamento avviene ovviamente sugli elementi già esaminati.

Teoria dei Grafi
Capitolo 7

K-fattorizzazione

La K-fattorizzazione di un grafo è la ricerca in quest'ultimo di sottografi indotti che siano K-regolari ovvero, che abbiano tutti i nodi di ugual grado.

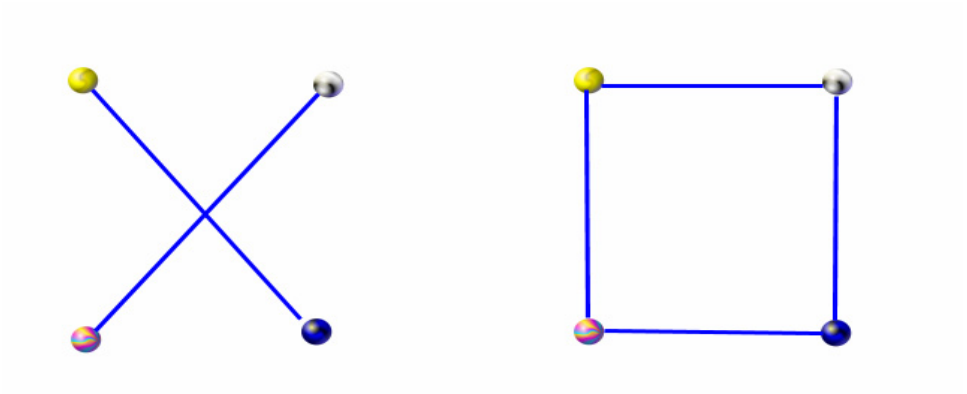
Esempio 7.1

Le K-fattorizzazioni di una clique d'ordine 4 sono semplicemente 4 triangoli e in generale possiamo dire che il triangolo rappresenta una 3-fattorizzazione di K_4.

TEOREMA 7.1

Se un grafo G non è connesso, allora il suo grafo complemento G' sarà connesso.

Esempio 7.2



Il primo grafo è composto da due coppie di nodi connessi a due a due da un arco; il grafo è non connesso, ma il suo grafo complemento, che è un quadrato, è chiaramente connesso.

Ricordo che un grafo risulta non connesso quando anche un solo nodo è privo di archi entranti, uscenti o non orientati.

LEMMA 7.1

In proposito esiste un lemma che ci dice:

- un grafo connesso con n nodi ha almeno (n-1) archi

Ed è abbastanza intuitivo capire come questa proprietà derivi dal solito fatto che la connessione tra due nodi avvenga con un arco. Avevamo discusso di questo nel cap.4 con l'Handshaking Lemma.

Vediamo ora una condizione che ci permette di capire dal numero di nodi ed archi quando un grafo sia aciclico.

LEMMA 7.2

Un grafo connesso con n nodi è aciclico quando contiene al massimo (n-1) archi.

Pensiamo al quadrato senza un lato, 4 nodi e 3 archi.

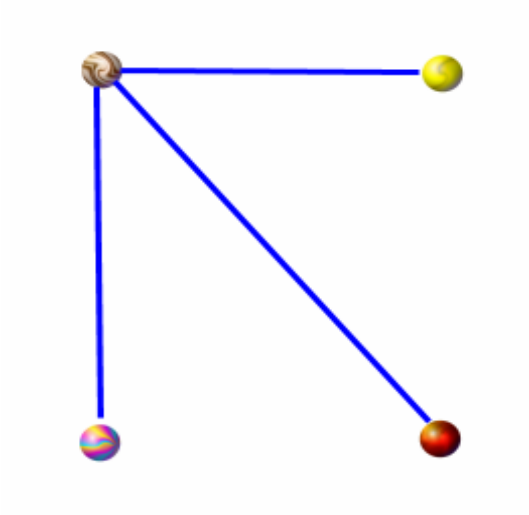
TEOREMA 7.2

Sia dato G(v,e) dove v è il numero di nodi ed e il numero di archi.

1. G è connesso
2. G è aciclico
3. G ha (v-1) archi

Se sono vere almeno due di queste tre proprietà, allora sarà vera anche la terza.

Esempio 7.3



Il grafo dell'esempio 7.3 verifica le prime due condizioni; non esistono nodi singoli, non esistono cicli. Dunque possiamo dire che ha tre archi.

Allo stesso modo possiamo dire che ha (v-1) archi, non ha cicli, dunque è connesso. O ancora non ha nodi esposti, ha tre archi e perciò è aciclico.

Albero

Si dice albero quel grafo G con le seguenti caratteristiche:

- ha n nodi
- ha (n-1) archi
- è connesso
- è aciclico

Foglia

Nodo di un albero con grado 1.

- ogni albero ha almeno 2 foglie
- due nodi di un albero sono connettibili da un solo cammino

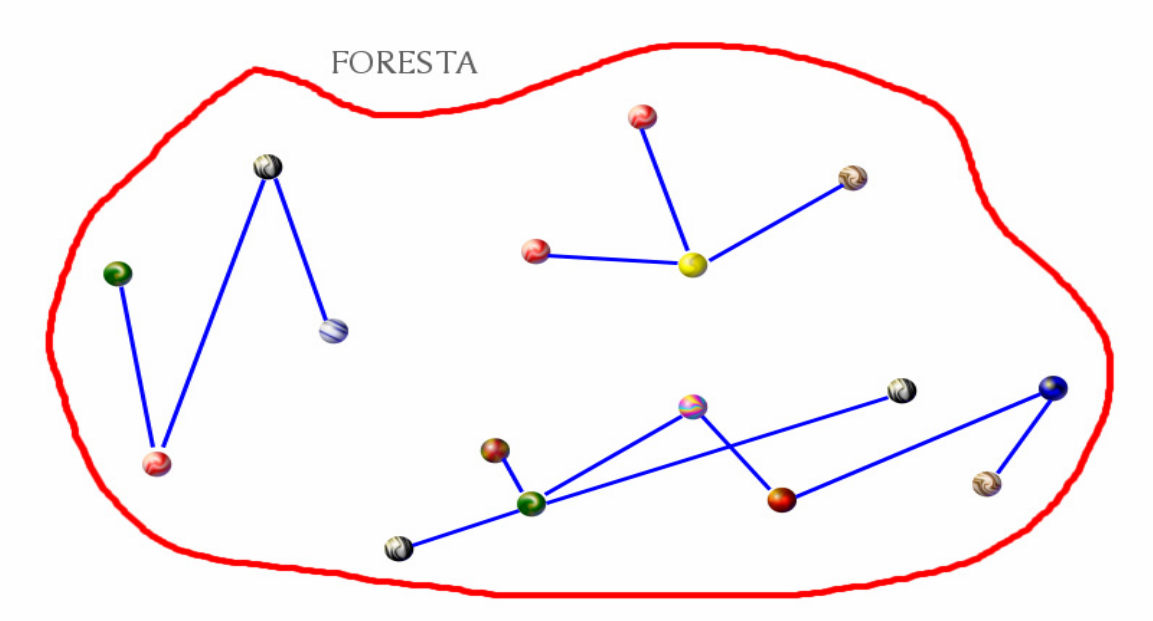
Faccio notare come il minimo numero di foglie di un albero sia pari a 2; praticamente una retta.
Il massimo numero di foglie invece, è dato dalla seguente formula:

- se n è pari → n/2
- se n è dispari → (n+1)/2

dove come al solito, s'intende con n il numero di nodi del grafo.

Foresta

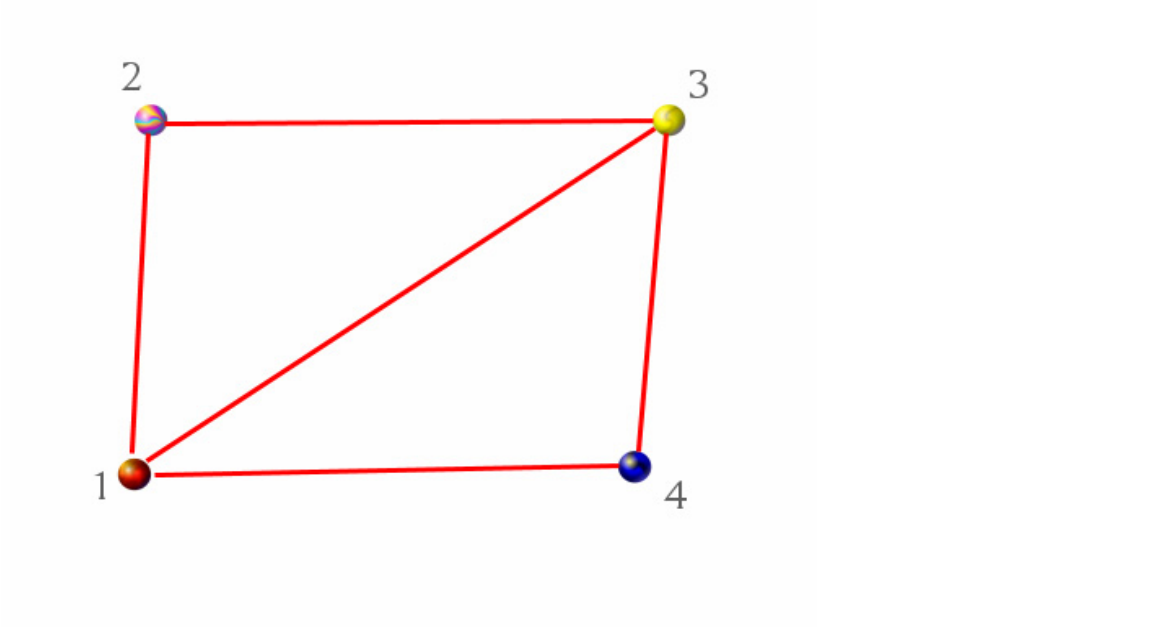
Grafo le cui componenti connesse e non connesse sono alberi.



Cammino Euleriano

E' un cammino sul generico grafo G che passa per ogni arco una ed una sola volta.

Esempio 7.4



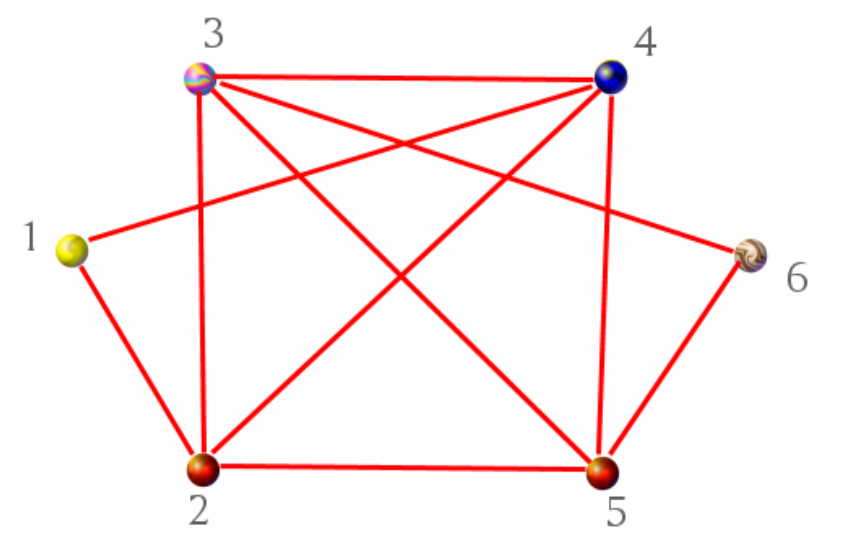
Un cammino Euleriano nell'esempio 7.4 potrebbe essere: { 1,2,3,4,1,3 }.

Ciclo Euleriano

E' un cammino Euleriano che torna al punto di partenza.

Nel caso dell'esempio 7.4 questo non sarebbe possibile, vedremo tra breve perché.

Esempio 7.5



In questo caso un possibile ciclo Euleriano sarà: { 1,2,3,4,2,5,3,6,5,4,1 }.

Grafo Euleriano

E' quel grafo G che ammette almeno un ciclo Euleriano.

- un cammino Euleriano è possibile solo in grafi connessi dove il grado dei nodi è pari o al più è dispari solo per due nodi
- un ciclo Euleriano è possibile solo in grafi connessi che abbiano tutti i nodi di grado pari

Vediamo che nel grafo dell'esempio 7.4, il grado dei nodi 1 e 3 è dispari. Il grafo ammette infatti un cammino Euleriano, ma non un ciclo. Nell'esempio 7.5 invece è possibile tracciare un ciclo Euleriano poiché tutti i nodi hanno grado pari.

Teoria dei Grafi
Capitolo 8

Nodo marcato

Un nodo si dice marcato quando quest'ultimo risulta essere raggiungibile.

Ordine j di un nodo

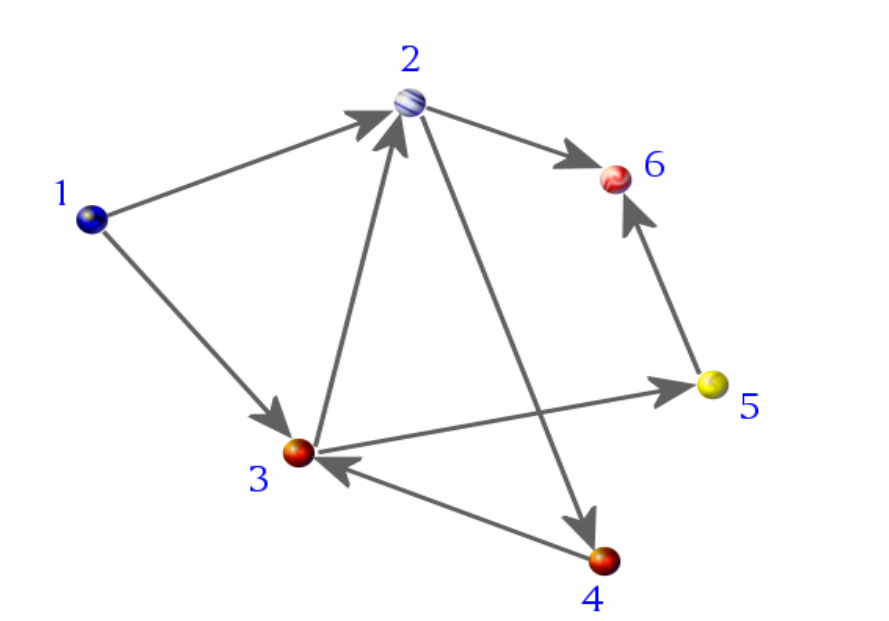
Sta ad indicare il numero di archi che bisogna percorrere prima di raggiungere il generico nodo i.

Lista d'adiacenza

In una colonna metteremo la lista ordinata dei nodi e per ogni generico nodo j descriveremo in un'altra colonna la lista ordinata dei nodi marcabili da j.

Considerando il seguente grafo:

Esempio 8.1



avremo una lista d'adiacenza del genere:

1	**2,3**
2	**4,6**
3	**2,5**
4	**3**
5	**6**
6	**NULL**

METODO FIFO – Ricerca in ampiezza

Il termine FIFO sta per 'First In First Out' e l'algoritmo si propone di cercare un albero sul generico grafo connesso ed orientato G.

Gli 'steps' dell'algoritmo sono i seguenti:

---

1. PRENDO IL 1° NODO DELLA 1ª COLONNA DELLA LISTA D'ADIACENZA E LO INSERISCO IN UN INSIEME INIZIALMENTE VUOTO CHE CHIAMERO' LIST

2. PRENDO IL PRIMO ELEMENTO J DI LIST E SE ESISTE DA QUESTO NODO UN ARCO AMMISSIBILE CHE PUNTA AL NODO W SCRIVERO':

   PRED(W) = J
   NEXT = 1 + i per i = 1,2,3,.....
   ORDER(W) = NEXT
   LIST = { J,W }

3. QUANDO UN NODO DELLA LISTA NON HA PIU' ARCHI AMMISSIBILI LO ELIMINO E SE W E' STATO GIA' ESAMINATO NON LO INSERISCO IN LIST

4. TERMINO L'ALGORITMO QUANDO TUTTI GLI ELEMENTI CHE HO NELLA LISTA NON HANNO PIU' ARCHI AMMISSIBILI

Vediamo come funziona sul grafo dell'esempio 8.1.

STEP 0

Prendo il primo nodo nella colonna di sinistra della lista d'adiacenza: 1.

LIST = { 1 }

pred(1) = null
next = 1
order(1) = 1
LIST = { 1 }

STEP 1

LIST = { 1 }

esistono archi ammissibili dal nodo 1? Si, prendo il primo della lista cioè 2.

pred(2) = 1
next = 2
order(2) = 2
LIST= { 1,2 }

STEP 2

LIST= { 1,2 }

esistono ancora archi ammissibili dal nodo 1? Si, prendo il successivo della lista cioè 3.

pred(3) = 1
next = 3
order(3) = 3
LIST= { 1,2,3 }

STEP 3

LIST= { 1,2,3 }

esistono ancora archi ammissibili dal nodo 1? No, cancello il nodo 1 dalla lista.

next = 4
LIST= { 2,3 }

STEP 4

LIST= { 2,3 }

Scelgo il nodo successivo in LIST che è stato il primo ad entrare tra quelli attualmente presenti (First In First Out): 2.
Esistono archi ammissibili dal nodo 2? Si, prendo il primo della lista cioè 4.

pred(4) = 2
next = 5
order(4) = 5
LIST= { 2,3,4 }

STEP 5

LIST= { 2,3,4 }

esistono ancora archi ammissibili dal nodo 2? Si, prendo il successivo della lista cioè 6.

pred(6) = 2
next = 6
order(6) = 6
LIST= { 2,3,4,6 }

STEP 6

LIST= { 2,3,4,6 }

esistono ancora archi ammissibili dal nodo 2? No, cancello il nodo 2 dalla lista.

next = 7
LIST= { 3,4,6 }

STEP 7

LIST= { 3,4,6 }

Scelgo il nodo successivo in LIST seguendo il criterio FIFO: 3.
Esistono archi ammissibili dal nodo 3? Il nodo 2 l'ho già esaminato, prendo il nodo 5.

pred(5) = 3
next = 8
order(5) = 8
LIST= { 3,4,6,5 }

STEP 8

LIST= { 3,4,6,5 }

esistono ancora archi ammissibili dal nodo 3? No, cancello il nodo 3 dalla lista.

next = 9
LIST= { 4,6,5 }

STEP 9

LIST= { 4,6,5 }

Scelgo il prossimo nodo da LIST : 4.
Esistono archi ammissibili dal nodo 4? Il nodo 3 l'ho già esaminato; 4 non ha più nodi marcabili e dunque lo cancello dalla lista.

next = 10
LIST= { 6,5 }

STEP 10

LIST= { 6,5 }

Scelgo il prossimo nodo da LIST : 6.
Esistono archi ammissibili dal nodo 6? No, cancello il nodo 6 dalla lista.

next = 11
LIST= { 5 }

STEP 11

LIST= { 5 }

Scelgo il nodo successivo da LIST: 5.
Esistono archi ammissibili dal nodo 5? No, perché il nodo 6 l'ho già esaminato, dunque tolgo 5 dalla lista e termino l'algoritmo.

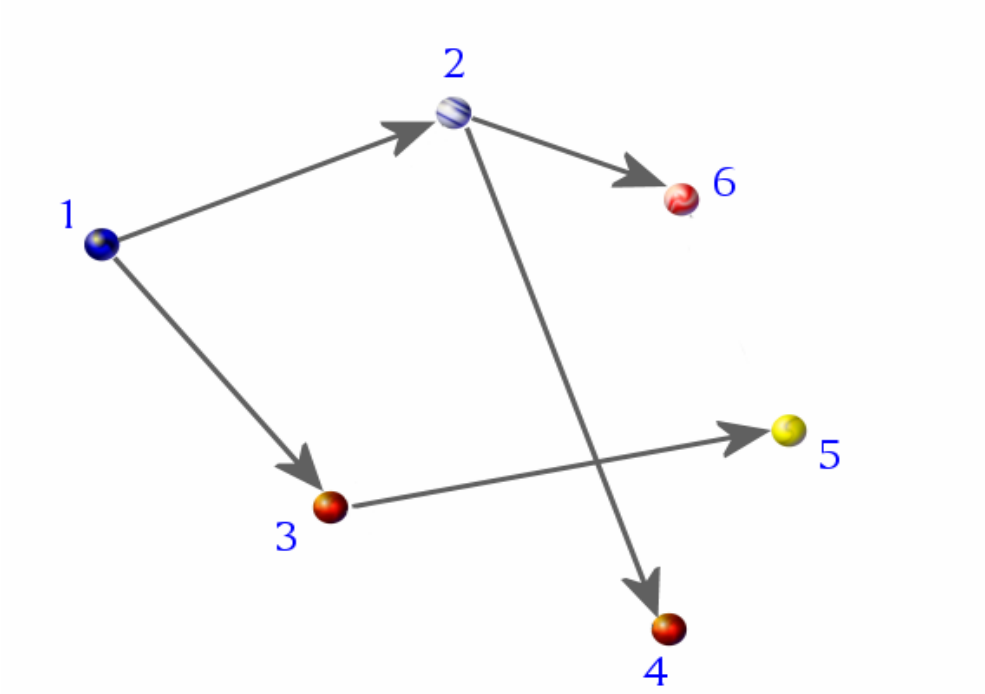
next = 12
LIST= { }

Possiamo adesso costruire il relativo albero per il grafo andando a prendere tutti gli ultimi predecessori che incontriamo andando a ritroso sugli steps:

- pred(5) = 3 STEP 7
- pred(6) = 2 STEP 5
- pred(4) = 2 STEP 4
- pred(3) = 1 STEP 2
- pred(2) = 1 STEP 1

- pred(1) = NULL STEP 0

L'albero ottenuto è il seguente:



Passiamo adesso ad esaminare l'algoritmo LIFO.

METODO LIFO – Ricerca in profondità

Quest'algoritmo si differenzia dal precedente nell'elemento che preleva di volta in volta; nel FIFO infatti il primo ad entrare era il primo ad essere esaminato, mentre in questo caso l'ultimo ad entrare sarà il primo ad essere esaminato.
Proveremo quest'algoritmo sullo stesso grafo dell'esempio 8.1 utilizzando la seguente struttura:

---

1. PRENDO IL 1° NODO DELLA 1ª COLONNA DELLA LISTA D'ADIACENZA E LO INSERISCO IN UN INSIEME INIZIALMENTE VUOTO CHE CHIAMERO' LIST

2. PRENDO L'ULTIMO ELEMENTO J DI LIST E SE ESISTE DA QUESTO NODO UN ARCO AMMISSIBILE CHE PUNTA AL NODO W SCRIVERO':

   PRED(W) = J
   NEXT = 1 + i per i = 1,2,3,.....
   ORDER(W) = NEXT
   LIST = { J,W }

   QUANDO UN NODO DELLA LISTA NON HA PIU' ARCHI AMMISSIBILI LO ELIMINO E SE W E' STATO GIA' ESAMINATO NON LO INSERISCO IN LIST

5. TERMINO L'ALGORITMO QUANDO TUTTI GLI ELEMENTI CHE HO NELLA LISTA NON HANNO PIU' ARCHI AMMISSIBILI

STEP 0

Prendo il primo nodo nella colonna di sinistra della lista d'adiacenza: 1.

LIST = { 1 }

pred(1) = null
next = 1
order(1) = 1
LIST = { 1 }

STEP 1

LIST = { 1 }

esistono archi ammissibili dal nodo 1? Si, prendo il primo della lista cioè 2.

pred(2) = 1
next = 2
order(2) = 2
LIST= { 1,2 }

STEP 2

LIST= { 1,2 }

esistono ancora archi ammissibili dal nodo 1? Si, prendo il successivo della lista cioè 3.

pred(3) = 1
next = 3
order(3) = 3
LIST= { 1,2,3 }

STEP 3

LIST= { 1,2,3 }

esistono ancora archi ammissibili dal nodo 1? No, cancello il nodo 1 dalla lista.

next = 4
LIST= { 2,3 }

STEP 4

LIST= { 2,3 }

Scelgo l'ultimo nodo in LIST (Last In First Out): 3.
Esistono archi ammissibili dal nodo 3? Si, prendo il primo della lista cioè 2, ma è stato già esaminato, dunque scelgo il successivo della lista d'adiacenza: 5.

pred(5) = 3
next = 5
order(5) = 5
LIST= { 2,3,5 }

STEP 5

LIST= { 2,3,5 }

esistono ancora archi ammissibili dal nodo 3? No, cancello il nodo 3 dalla lista.

next = 6
LIST= { 2,5 }

STEP 6

LIST= { 2,5 }

Scelgo l'ultimo nodo in LIST: 5.
Esistono archi ammissibili dal nodo 5? Si, prendo l'unico della lista cioè 6.

pred(6) = 5
next = 7
order(6) = 7
LIST= { 2,5,6 }

STEP 7

LIST= { 2,6 }

esistono ancora archi ammissibili dal nodo 5? No, cancello il nodo 5 dalla lista.

next = 8
LIST= { 2,6 }

STEP 8

LIST= { 2,6 }

Scelgo l'ultimo nodo in LIST: 6.
Esistono archi ammissibili dal nodo 6? No, cancello il nodo 5 dalla lista.

next = 9
LIST= { 2 }

STEP 9

LIST= { 2 }

Scelgo l'unico nodo in LIST: 2.
Esistono archi ammissibili dal nodo 2? Si, prendo il primo della lista cioè 4.

pred(4) = 2
next = 10
order(4) = 10
LIST= { 2,4 }

STEP 10

LIST= { 2,4 }

Esistono ancora archi ammissibili dal nodo 2? No, cancello il nodo 2 dalla lista.
marcabili e dunque lo cancello dalla lista.

next = 11
LIST= { 4 }

STEP 11

LIST= { 4 }

Scelgo l'unico nodo in LIST: 4.
Esistono archi ammissibili dal nodo 4? Si, ma li ho già esaminati dunque l'algoritmo ha termine ed elimino il nodo 4 dalla lista.

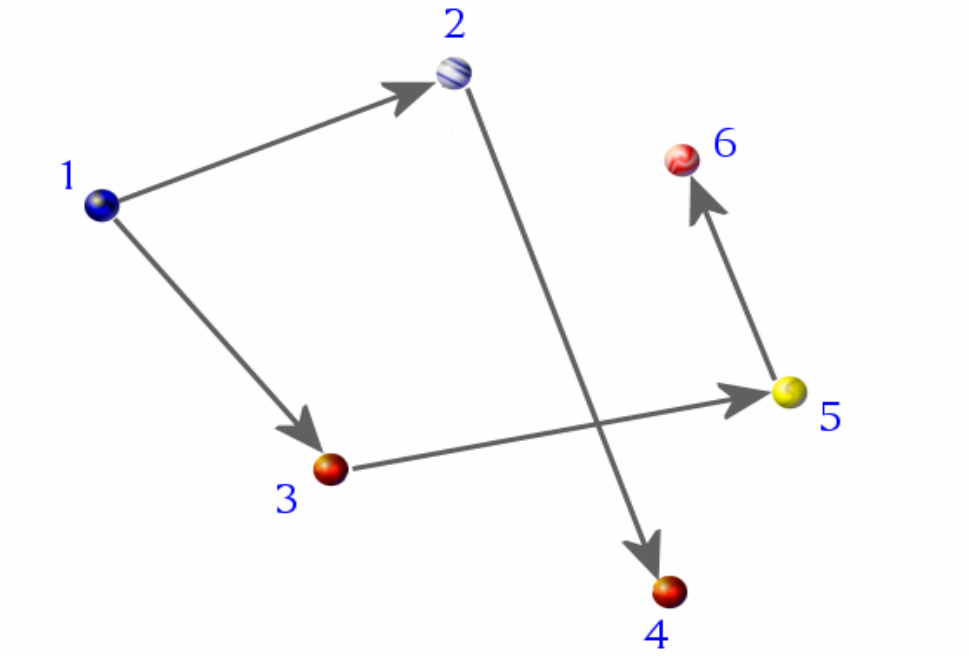
next = 12
LIST= { }

La lista dei predecessori è la seguente:

- pred(4) = 2 STEP 9
- pred(6) = 5 STEP 6
- pred(5) = 3 STEP 4
- pred(3) = 1 STEP 2
- pred(2) = 1 STEP 1

- pred(1) = NULL STEP 0

E questo è l'albero ottenuto:



Ordinamento topologico

In un grafo aciclico connesso ed orientato è possibile stabilire un ordine di precedenza tra due qualsiasi nodi i e j ovvero, è possibile maggiorare l'ordine di uno dei due nodi rispetto all'altro.

Inoltre, per ogni grafo aciclico connesso potrà esistere più d'un ordinamento topologico.

L'algoritmo dell'ordinamento topologico ci permette di posizionare lungo una retta i nodi del grafo e dare una sola direzione univoca a tutti gli archi orientati (in genere destra).

Faccio notare due cose:

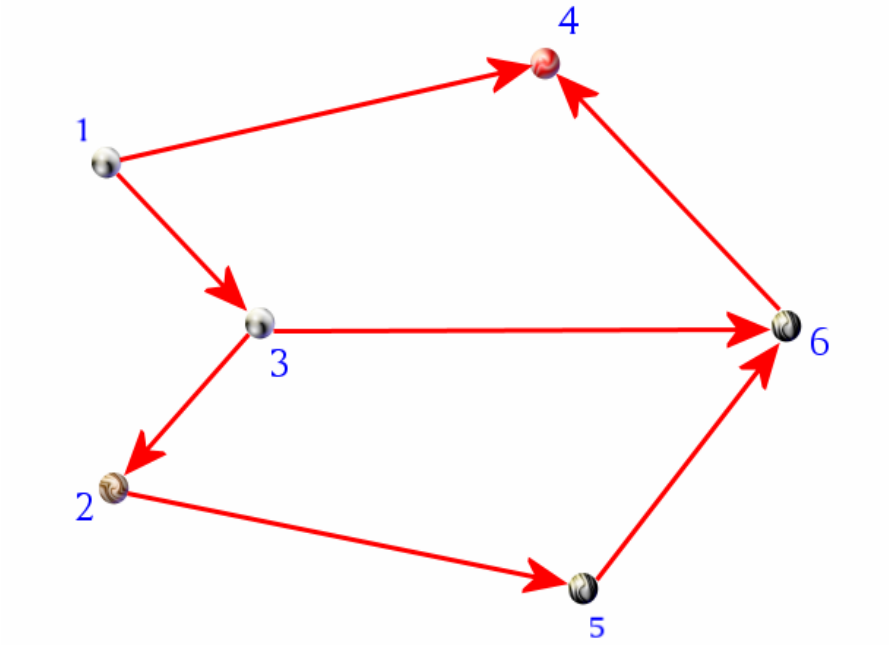
- se un grafo è aciclico, esiste almeno un ordinamento topologico
- se un grafo è aciclico, esiste almeno un nodo che ha grado d'ingresso pari a zero

ALGORITMO

- **seleziono i nodi nel grafo che non hanno archi entranti**
- **cancello i loro archi uscenti e costruisco l'ordine dei nodi dell'ordinamento topologico**

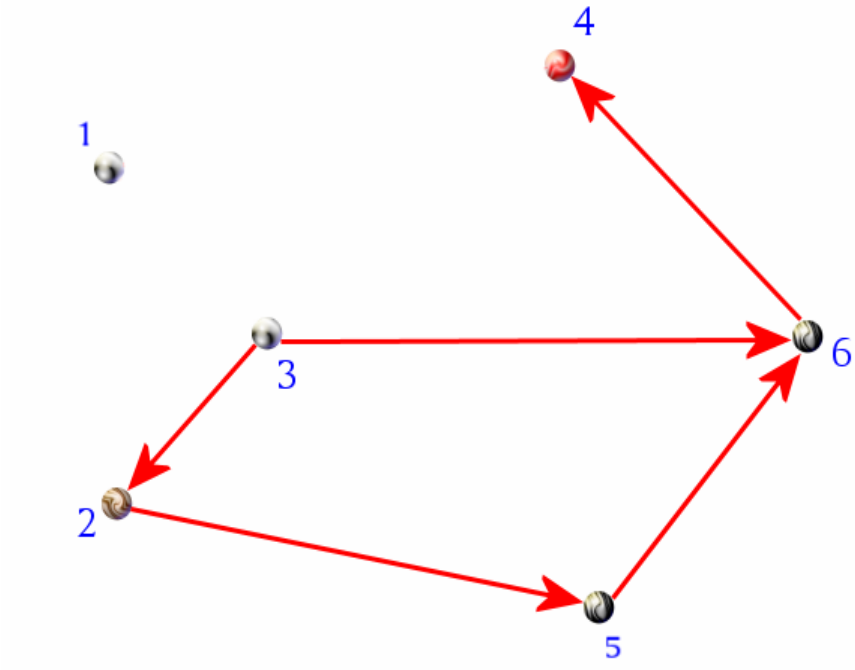
L'algoritmo va ovviamente iterato fino a quando non verifico più la prima condizione, ma vediamo come funziona l'ordinamento topologico su questo grafo d'esempio:

Esempio 8.2



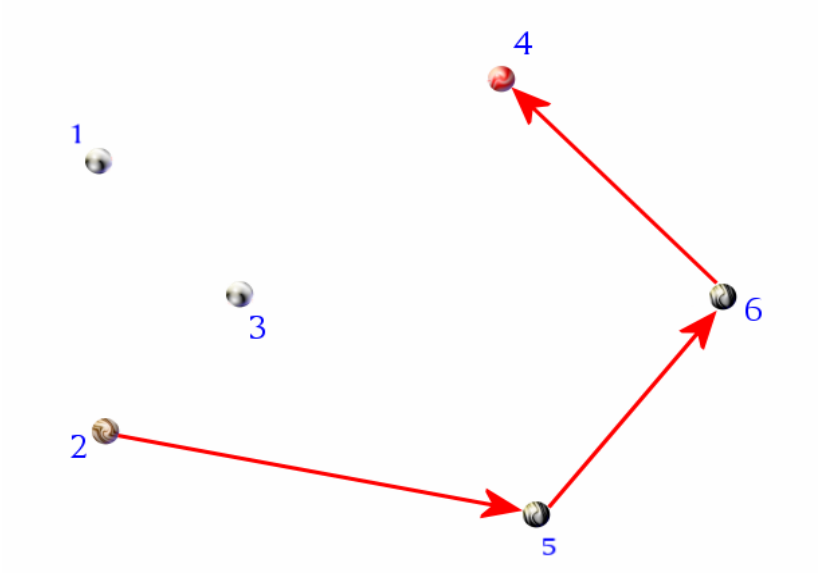
STEP 1

- seleziono i nodi che non hanno archi entranti: solo il nodo 1.
- cancello i suoi archi uscenti e lo metto nell'ordine topologico: { 1 }



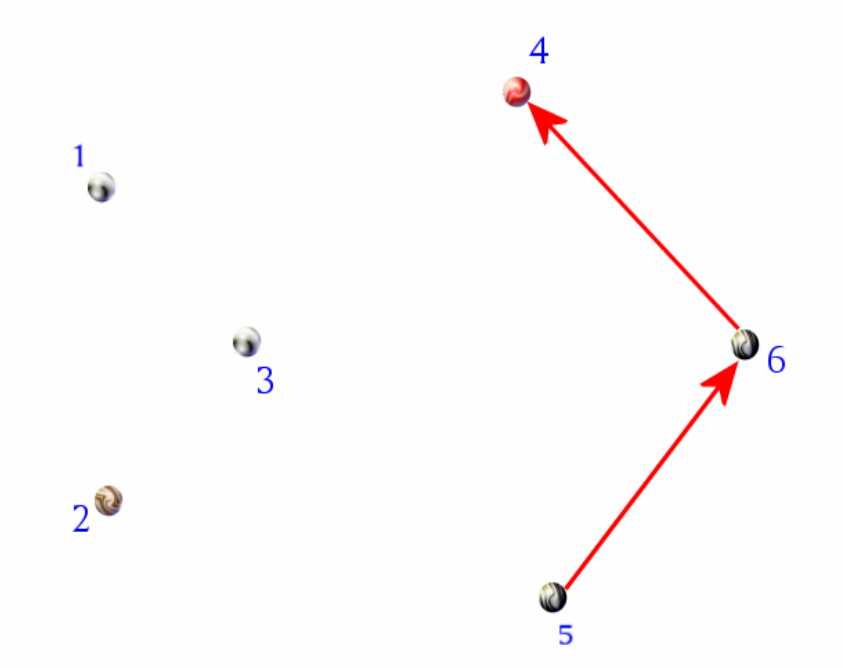
STEP 2

- seleziono i nodi che non hanno archi entranti: solo il nodo 3.
- cancello i suoi archi uscenti e lo metto nell'ordine topologico: { 1,3 }



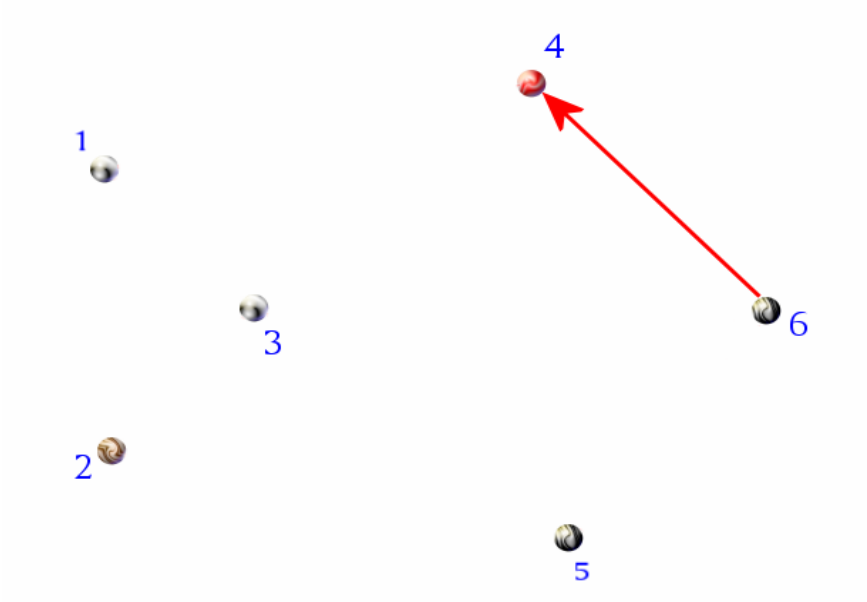
STEP 3

- seleziono i nodi che non hanno archi entranti: solo il nodo 2.
- cancello i suoi archi uscenti e lo metto nell'ordine topologico: { 1,3,2 }



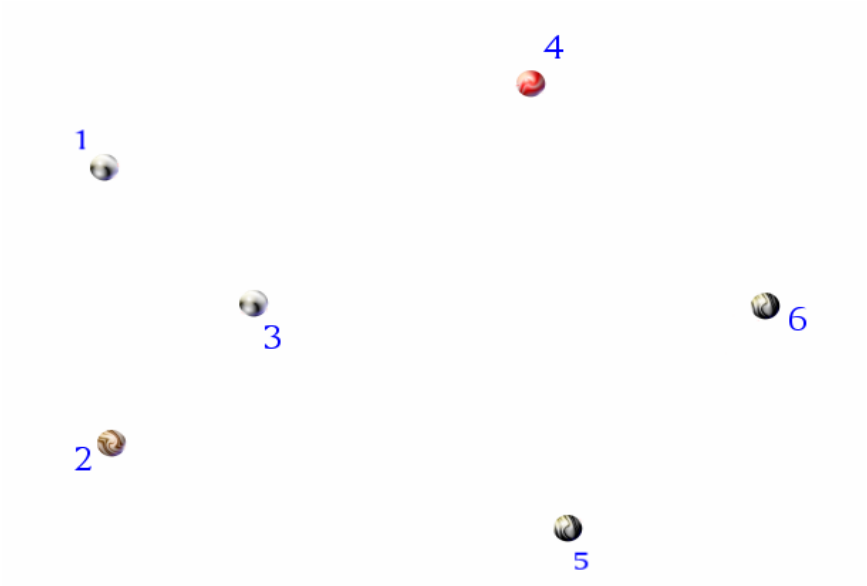
STEP 4

- seleziono i nodi che non hanno archi entranti: solo il nodo 5.
- cancello i suoi archi uscenti e lo metto nell'ordine topologico: { 1,3,2,5 }

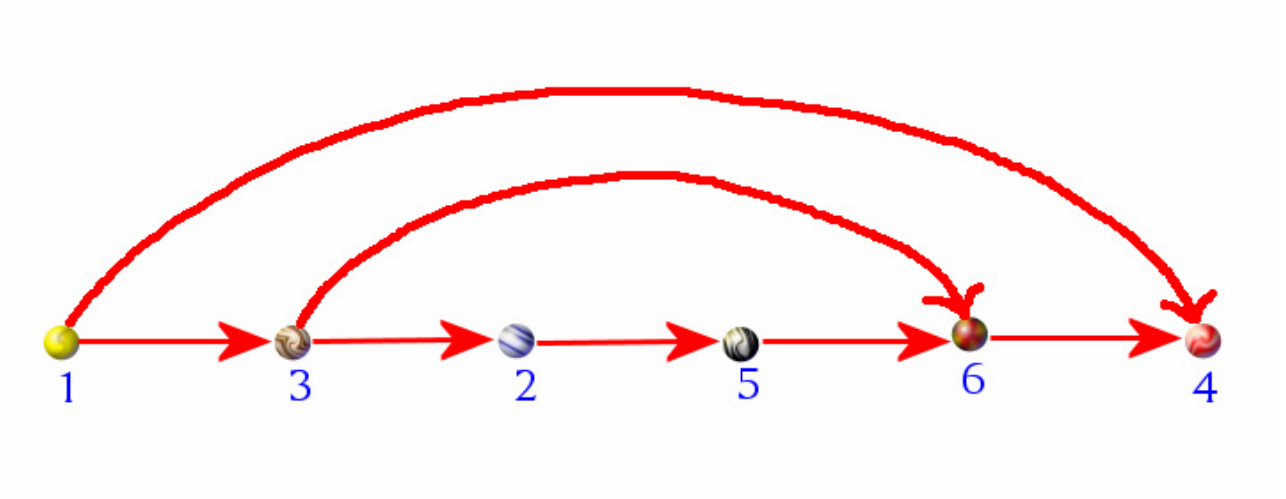


STEP 5

- seleziono i nodi che non hanno archi entranti: è rimasto solo 4 fuori dalla lista.
- lo metto nell'ordine topologico: { 1,3,2,5,6,4 }



Visualizziamo a questo punto l'ordinamento topologico appena costruito:



Come si vede dal grafo, sono verificate tutte le condizioni suddette:

- abbiamo dato un ordine di precedenza tra tutte le possibili coppie di nodi (i,j)
- abbiamo posizionato lungo una retta tutti i nodi del grafo dando loro una direzione univoca
- l'ordinamento topologico esiste
- il nodo 1 ha grado d'ingresso pari a zero

MINIMUM SPANNING TREE
(minimo albero ricoprente)

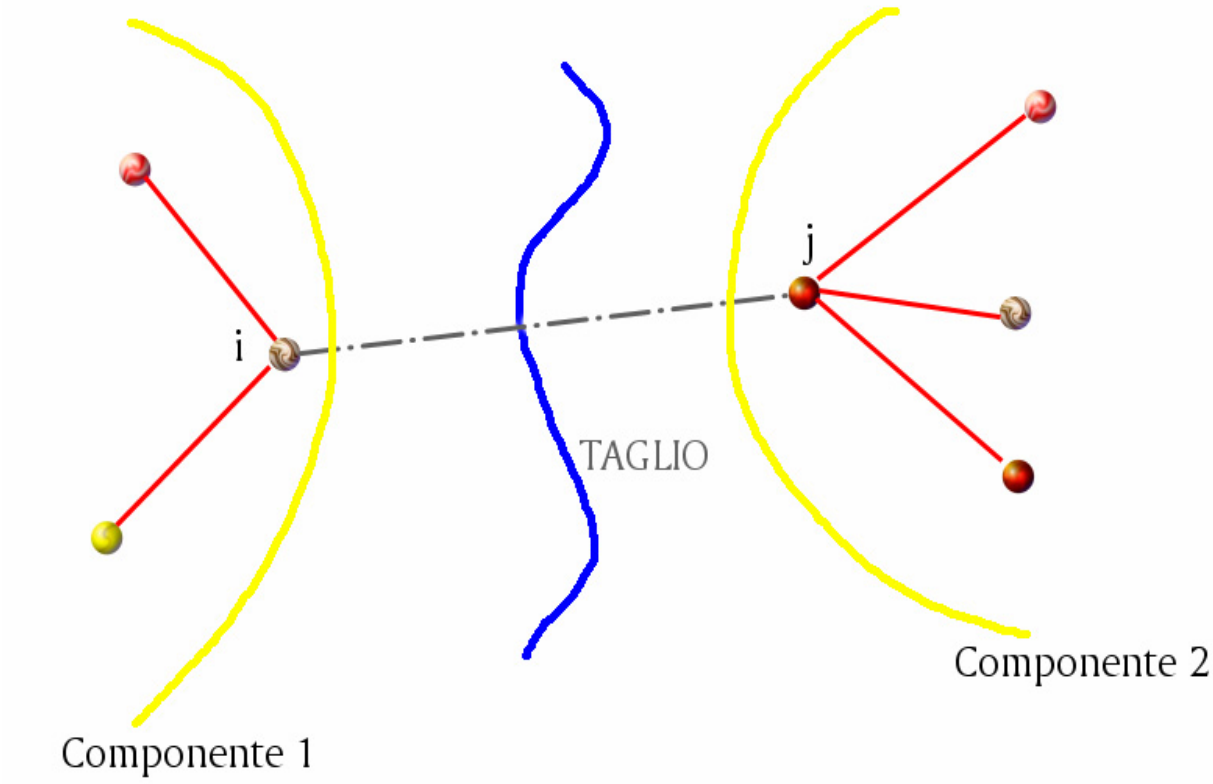
Come abbiamo visto con gli algoritmi FIFO e LIFO, è possibile creare su di un grafo connesso orientato e non, un albero che copra tutti i nodi del grafo stesso.
Quando diamo ad ogni arco un peso o un valore di costo, esisterà tra tutti gli "spanning trees" un "minimum spanning tree" tale che la somma di tutti i pesi degli archi che lo compongono sia la minima possibile.

Vedremo tra breve algoritmi atti alla ricerca del minimum spanning tree.

Taglio

Per taglio s'intende quell'arco che crea due componenti connesse.

Esempio 8.3



Formalmente, indichiamo questo taglio nel seguente modo:

- (i,j) ϵ [S,S']

dove S è la prima componente ed S' la seconda.

TEOREMA 8.1

Un albero G si dice minimo ricoprente se e solo se soddisfa la condizione di ottimalità del taglio.

Questa condizione è la seguente:

- per ogni (i,j) ϵ G si ha che $C_{ij} \leq C_{kl}$
- ogni C_{kl} sia contenuto nel taglio [S,S']

ovvero, scelto un generico arco C_{ij} del grafo G possiamo sempre trovare un arco C_{kl} del grafo originario (del quale abbiamo trovato l'albero minimo ricoprente) che pesi più di C_{ij} e che ogni arco non appartenente all'albero faccia parte dell'opportuno taglio [S,S'].

ALGORITMO DI KRUSKAL (1950)

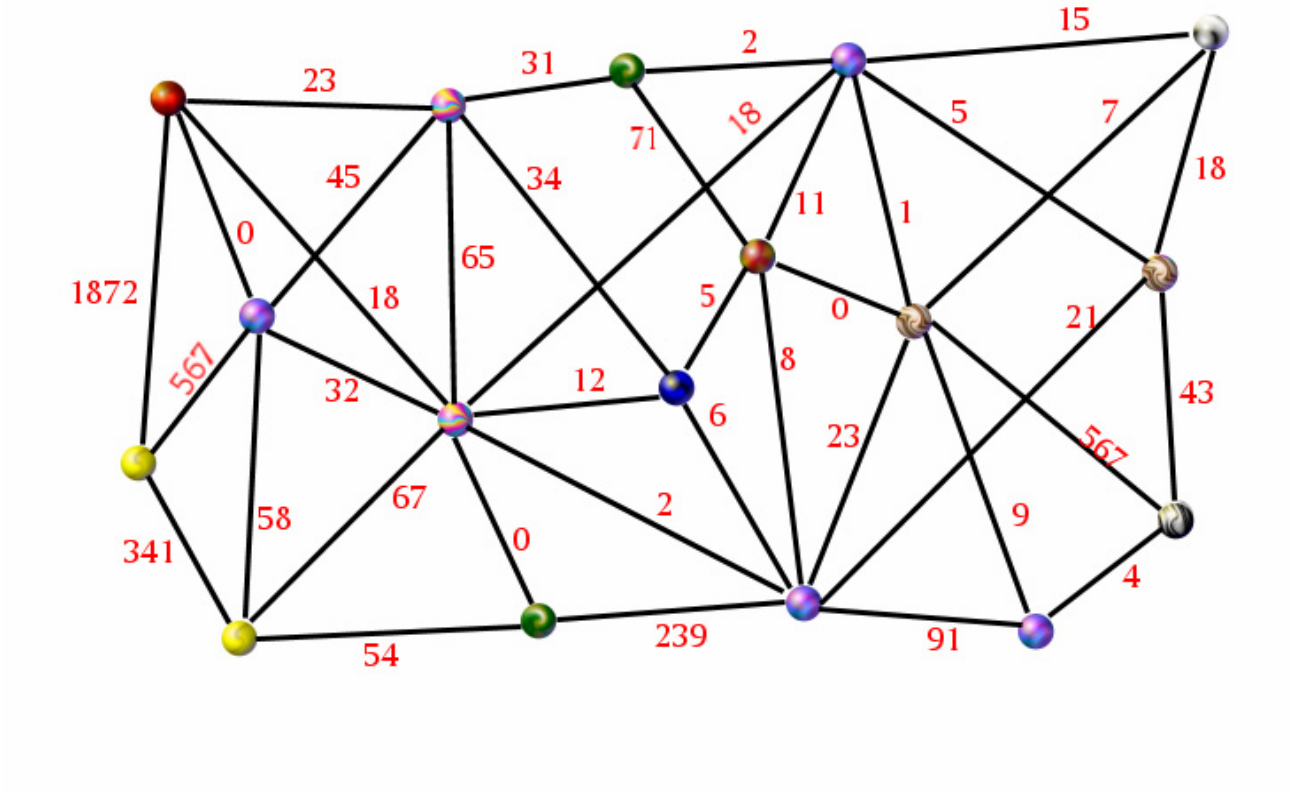
Quest'algoritmo è ancora più semplice di quello topologico e ci permette di trovare il Minimum Spanning Tree con un semplice criterio d'ordinamento.

Ma vediamo come funziona.

1. ordino in modo crescente i pesi degli archi e li cancello dal grafo lasciando solo i nodi
2. scelgo via via l'arco più 'leggero' e lo posiziono nella sua locazione originaria
3. se nel fare il passo 2 ottengo un ciclo, non metto quest'ultimo arco e passo al successivo

Testiamolo con un esempio.

Esempio 8.4



PASSO 1

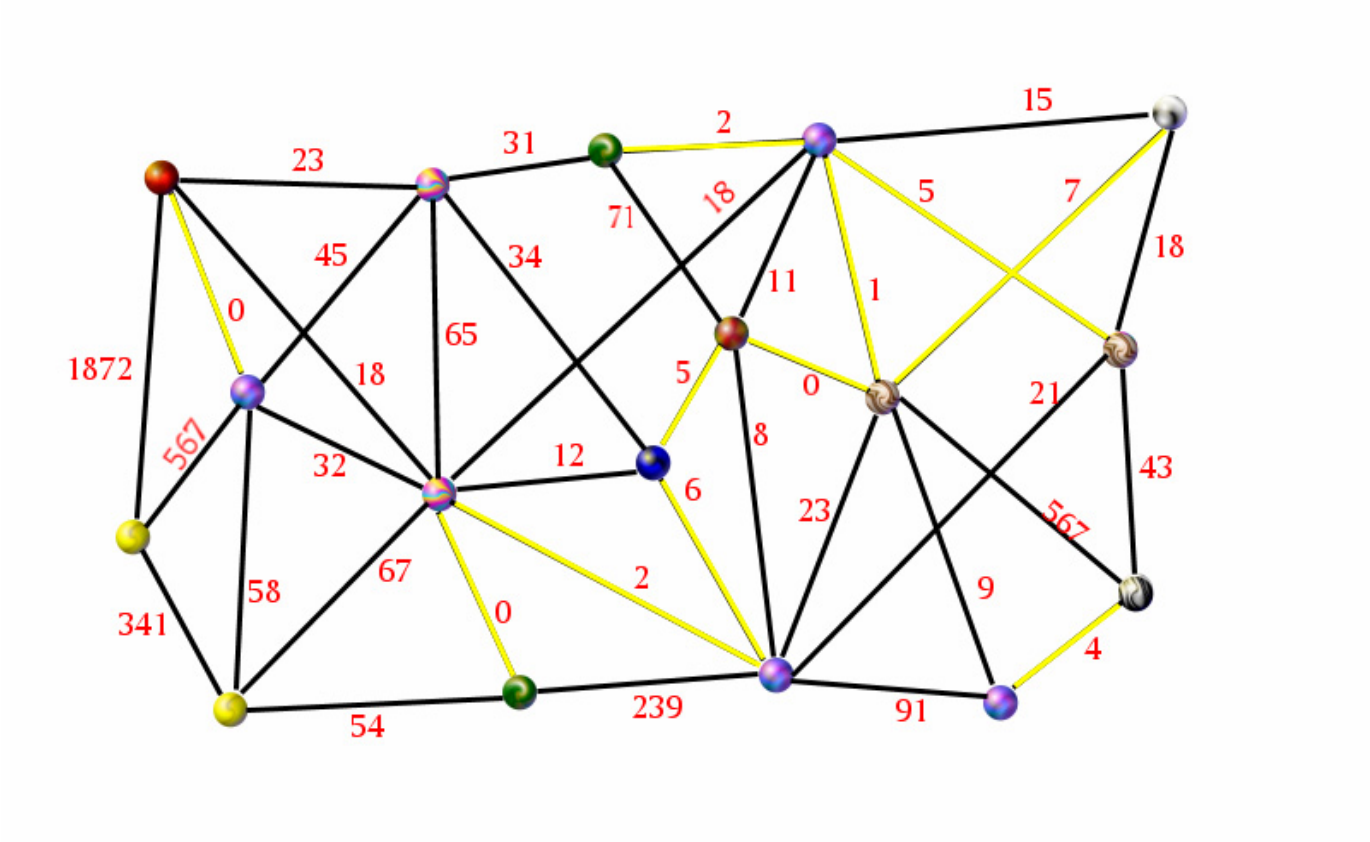
1	0
2	0
3	0
4	1
5	2
6	2
7	4
8	5
9	5
10	6
11	7
12	8
13	9
14	11
15	12
16	15
17	18
18	18
19	18
20	21
21	23
22	23
23	31
24	32
25	34
26	43
27	45
28	54
29	58
30	65
31	67
32	71
33	91
34	239
35	341
36	567
37	567
38	1872

PASSO 2

Evidenzierò in rosso gli archi riposizionati, supponendo che chi non sia evidenziato non si trovi al momento sul grafo.

Eseguiamo l'algoritmo sui primi 11 archi:

1	0
2	0
3	0
4	1
5	2
6	2
7	4
8	5
9	5
10	6
11	7

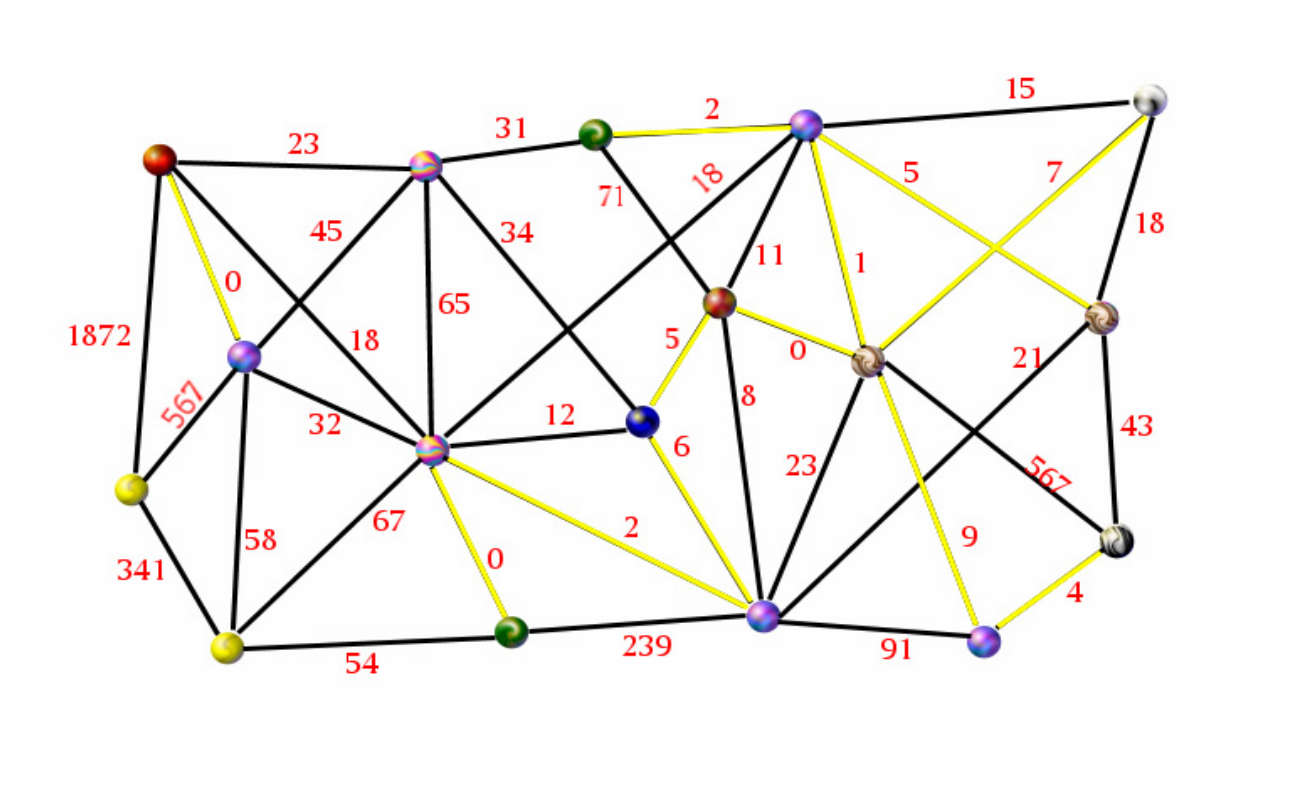


Come si può vedere, fino a questo punto non si è venuto a creare nessun ciclo, ma al passo successivo e cioè, con la scelta del dodicesimo arco che ha peso pari a 8 siamo, saremo costretti a non rimetterlo nel grafo ed ad andare oltre. L'arco di peso 8 mi crea infatti il ciclo { 8,6,5 }.

12	8

Continuiamo dunque con i successivi. Possiamo aggiungere l'arco di peso 9, ma non quello di peso 11 poiché mi creerà un ciclo { 11,1,0 }:

13 9
14 11



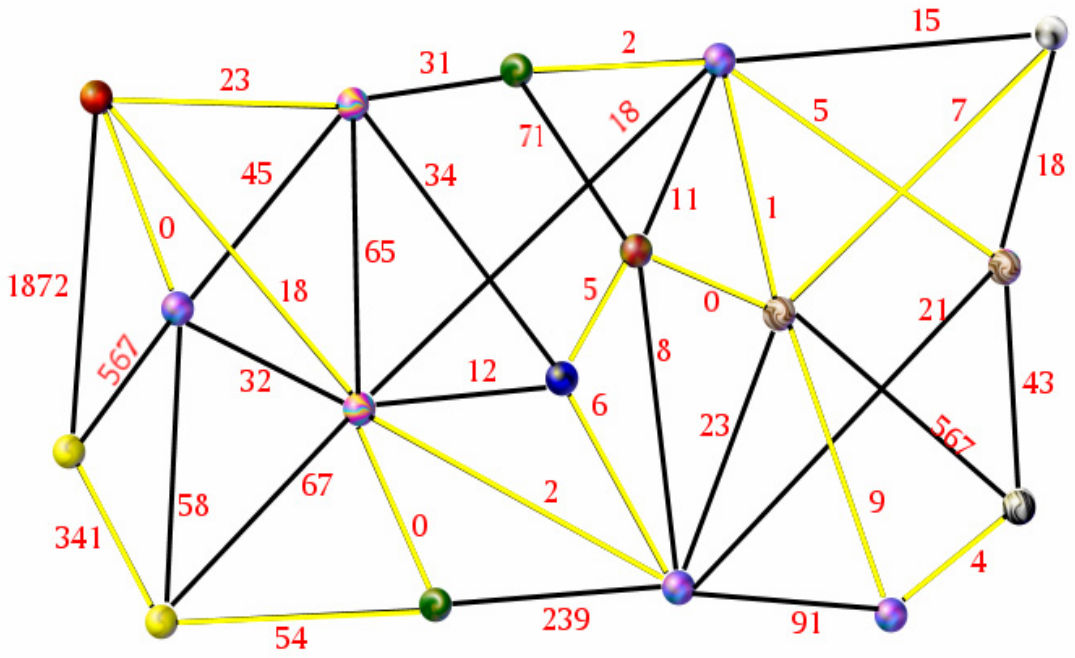
15	12
16	15
17	18
18	18
19	18
20	21
21	23
22	23
23	31
24	32
25	34
26	43
27	45
28	54
29	58
30	65

31	67
32	71
33	91
34	239
35	341
36	567
37	567
38	1872

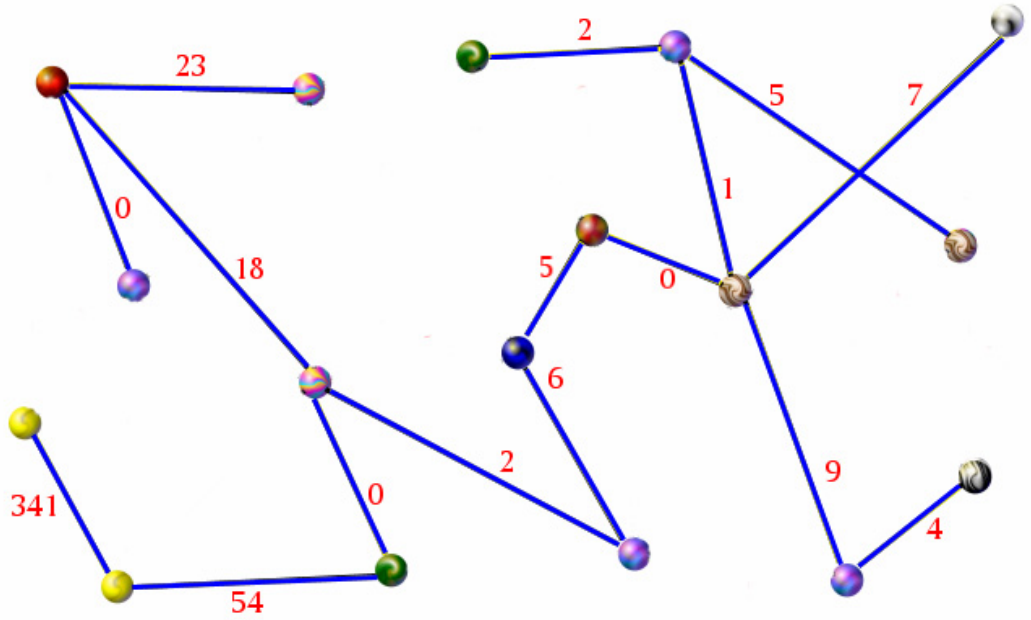
I rimanenti archi mi creano i seguenti cicli:

12 → { 12,2,6 }
15 → { 15,1,7 }
18 → { 18,7,1,5 }
18
18 → { 18,2,6,5,0,1 }
21 → { 21,5,1,0,5,6 }
23 → { 23,6,5,0 }
23
31 → { 31,2,1,0,5,6,2,18,23 }
34 → { 34,6,2,18,23 }
43 → { 43,4,9,1,5 }
45 → { 45,0,23 }
54
58 → { 58,0,18,0,54 }
65 → { 65,23,18 }
67 → { 67,0,54 }
71 → { 71,2,1,0 }
91 → { 91,6,5,0,9 }
239 → { 239,0,2 }
341
567 → { 567,0,18,0,54,341 }
567 → { 567,9,4 }
1872 → { 1872,18,0,54,341 }

E questo è l'albero finale ottenuto:



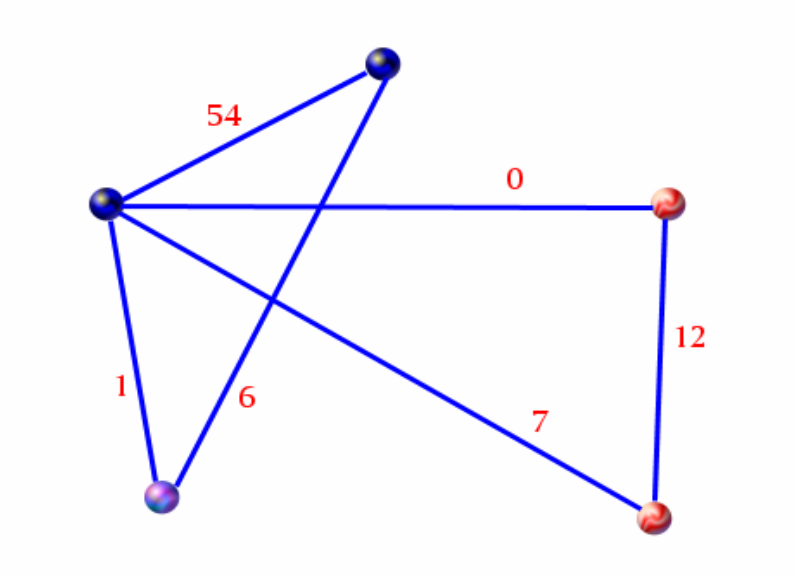
Come si vede, siamo riusciti ad aggiungere solamente altri quattro archi, quelli di peso 18, 23, 54 e 341. Quando capita di avere più archi dello stesso peso è indifferente prendere prima l'uno e poi l'altro. Vediamo adesso l'albero appena creato con l'algoritmo di Kruskal su un disegno migliore.



La somma dei pesi di quest'albero è pari a 477 ovvero, la minima possibile per poter formare un albero. Ricordo come ogni nodo sia connesso e dunque raggiungibile.

Vediamo ora lo stesso algoritmo su di un grafo più semplice come quello dell'esempio 8.5.

Esempio 8.5



PASSO 1

Metto in ordine crescente i pesi degli archi:

0
1
6
7
12
54

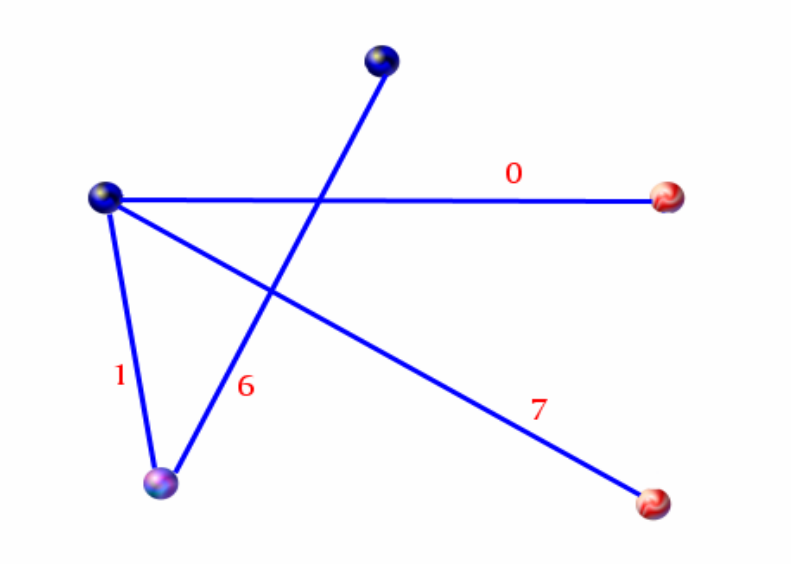
PASSO 2

Possiamo svolgere l'algoritmo ad occhio.

Scelgo l'arco di peso 0, non ho cicli.
Scelgo l'arco di peso 1, non ho cicli.
Scelgo l'arco di peso 6, non ho cicli.
Scelgo l'arco di peso 7, non ho cicli.

Aggiungendo l'arco di peso 12 o quello di peso 54, ottengo dei cicli…dunque li escludo.

L'albero ottenuto è il seguente:



Passiamo adesso ad un altro algoritmo, quello di Prim.

ALGORITMO DI PRIM

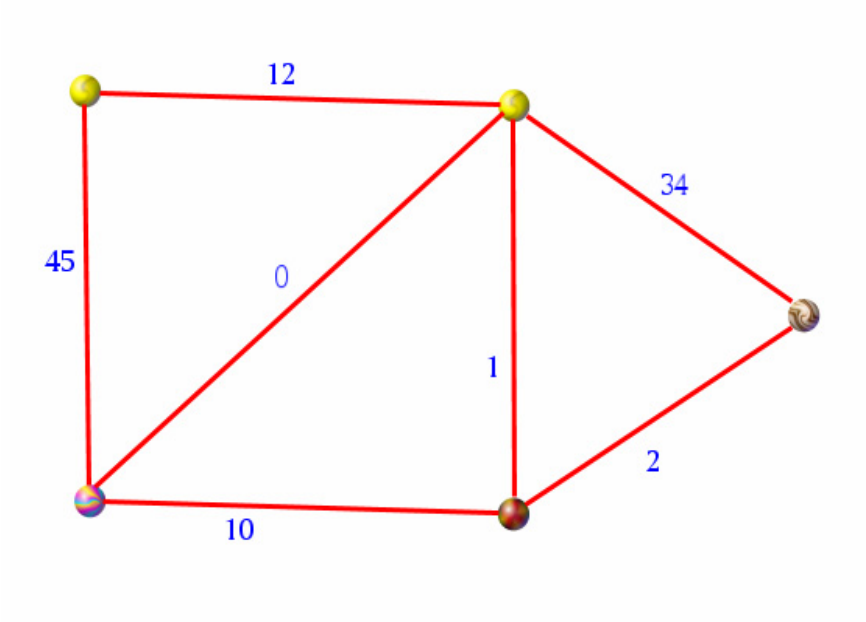
Quest'algoritmo ci permette di trovare l'albero minimo ricoprente mediante un altro processo.
Non si tratta questa volta di aggiungere via via gli archi fino ad ottenere dei cicli, ma effettueremo dei tagli per ottenere il Minimum Spanning Tree.

L'algoritmo ha il seguente schema:

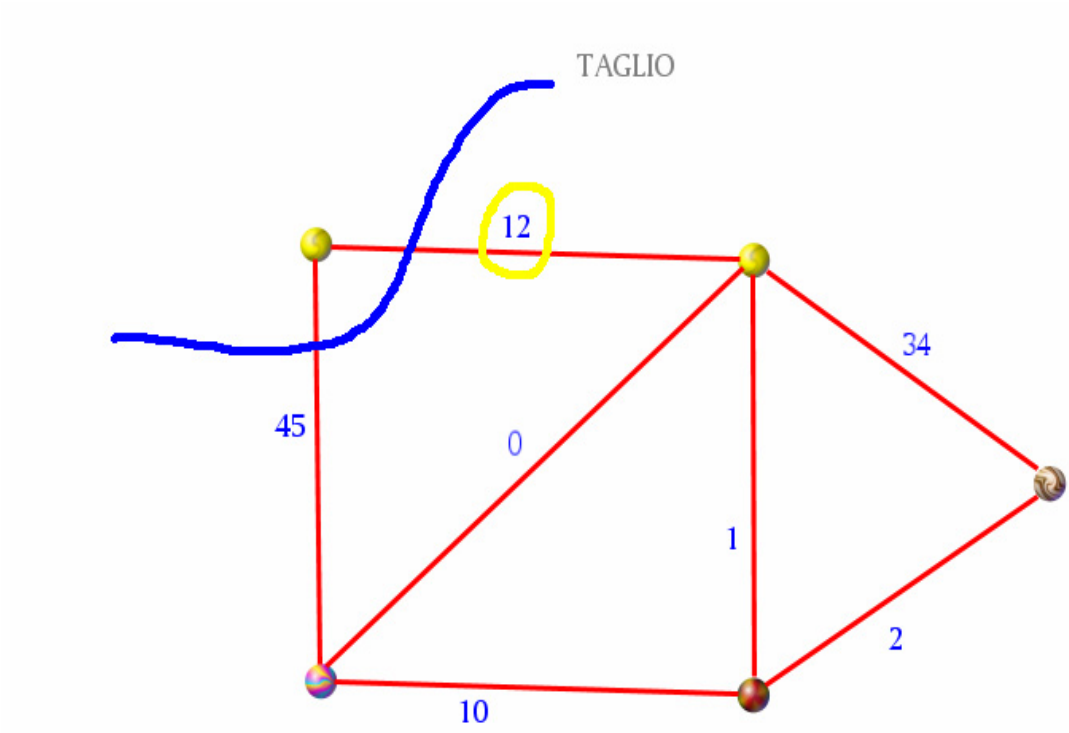
- LIST = { null }
- scelgo un qualsiasi nodo j dal grafo G originale el'aggiungo a LIST
- eseguo un taglio su LIST
- aggiungo al grafo T dell'albero in 'costruzione' l'arco di peso minimo appartenente al taglio appena effettuato ed aggiungo a LIST il nodo raggiunto da quest'arco
- eseguo il taglio su LIST
- quando il taglio contiene tutti i nodi di G l'algoritmo ha termine

Testiamolo sull'esempio 8.6.

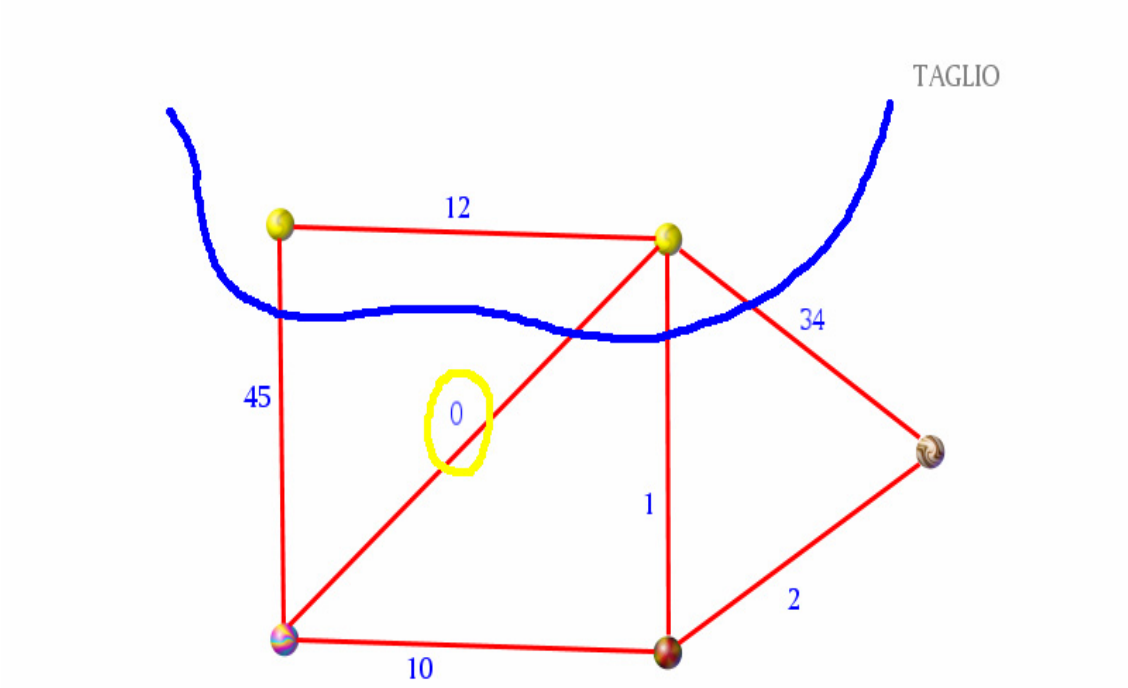
Esempio 8.6



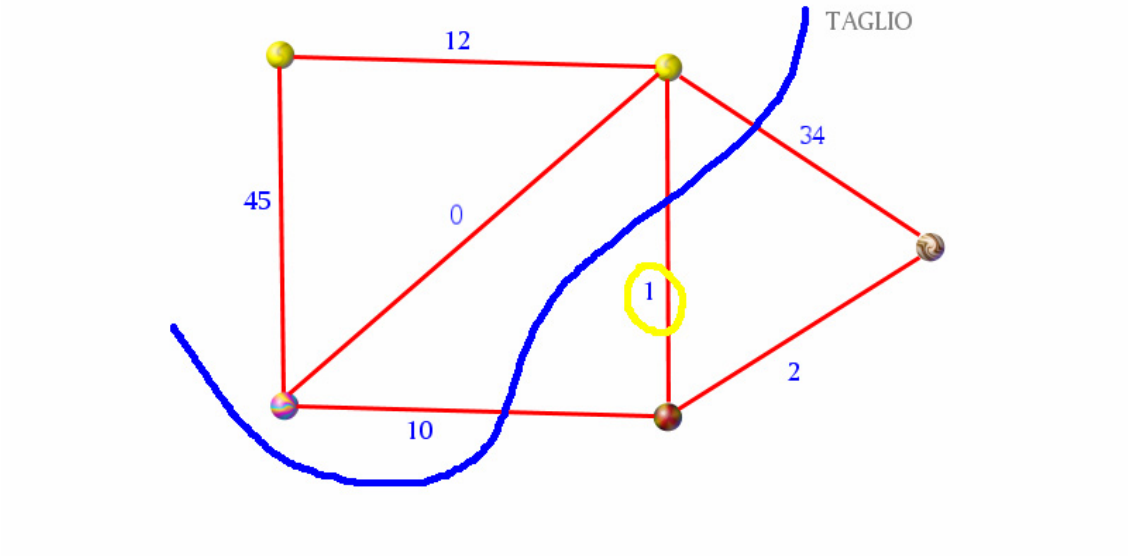
Scegliamo un nodo qualsiasi ed effettuiamo un primo taglio.



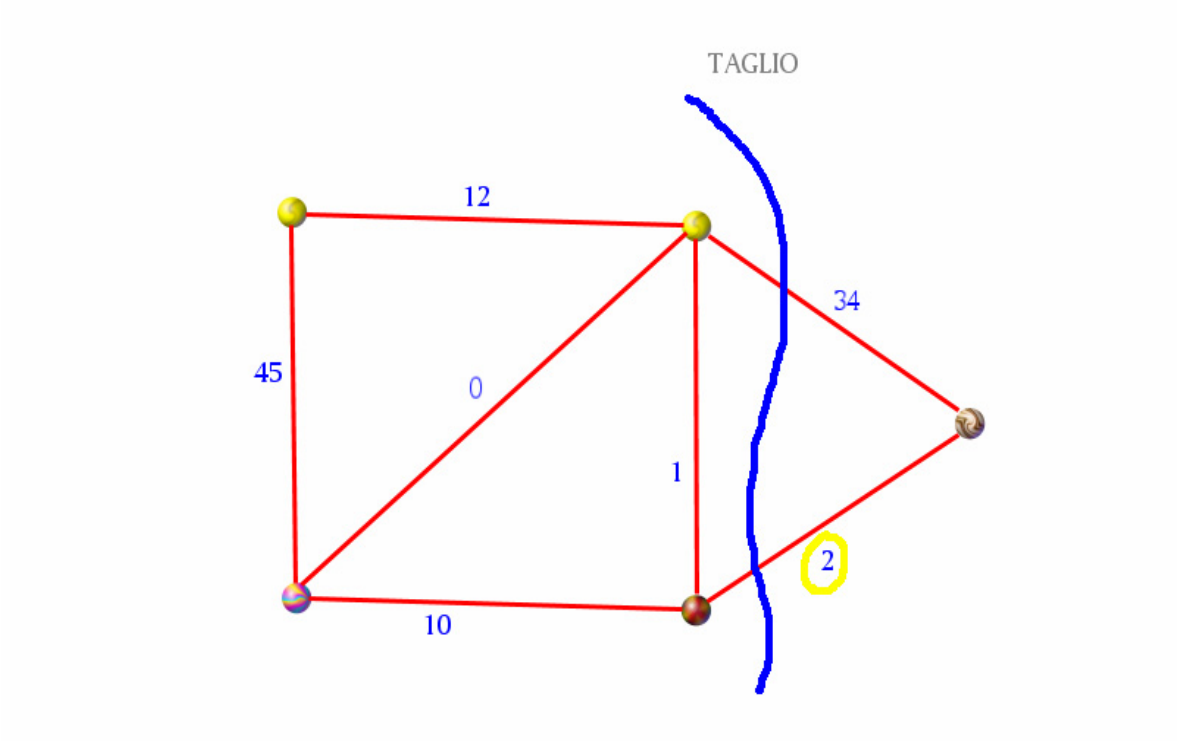
Gli archi appartenenti al taglio sono quelli di peso 12 e 45.
Scegliamo per l'albero T che stiamo costruendo l'arco di peso 12 ed effettuiamo il secondo taglio a LIST.



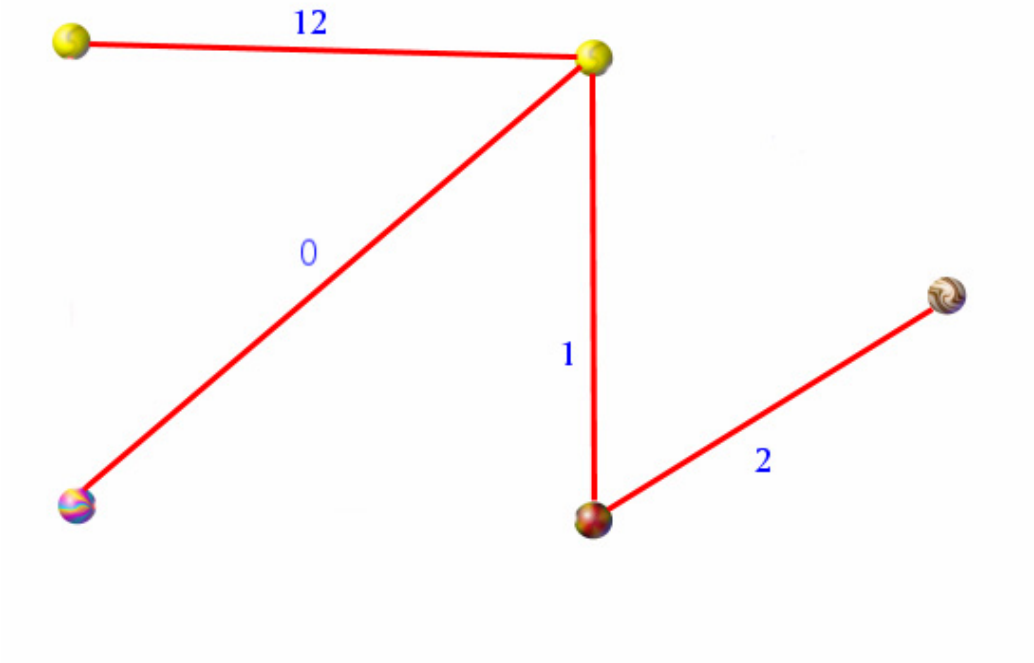
L'albero T = { 12,0 }, mentre a LIST si è aggiunto il nodo in basso a sinistra. Quest'ultimo nodo verrà compreso nel prossimo taglio:



Ovviamente tra 10, 1 e 34 sceglieremo l'arco di peso 1 che si aggiungerà a T.
Proseguiamo col taglio successivo.



Gli archi appartenenti all'ultimo taglio sono quelli di peso 2 e 34.
Scegliamo l'arco di peso 2, l'aggiungiamo a T e vediamo che nel taglio successivo tutti i nodi del grafo G sono compresi nel taglio; l'algoritmo è finito e l'albero minimo ricoprente è il seguente:



Se avessimo scelto un altro nodo per eseguire il primo taglio, il risultato sarebbe stato il medesimo ed avremmo ottenuto lo stesso albero se avessimo applicato l'algoritmo di Kruskal.

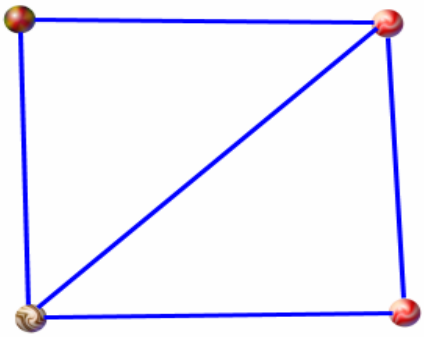
Teoria dei Grafi
Capitolo 9

Grafo planare

Si dice grafo planare è quel grafo che non ha archi intersecantesi.

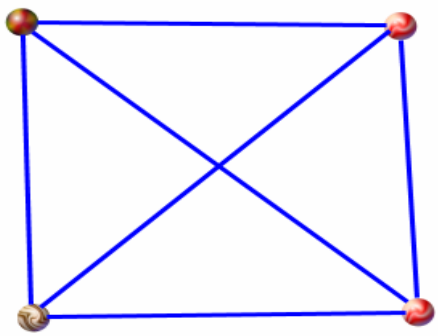
Vediamo qualche esempio.

Esempio 9.1

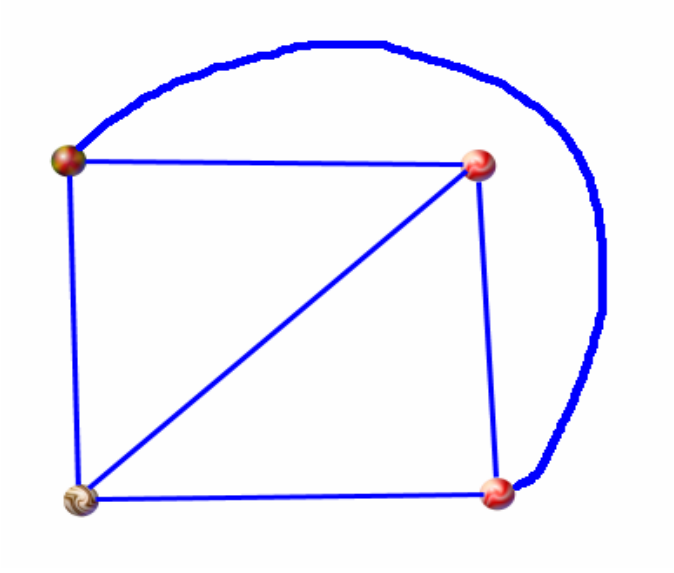


Questo è chiaramente un grafo planare.
Se aggiungessimo la seconda diagonale, potrebbe non sembrare più un grafo planare…

Esempio 9.2



E invece lo è ancora, perché se 'tiriamo' una delle due diagonali otteniamo nuovamente la condizione per un grafo planare e cioè, che nessun arco s'intersechi.



E' chiaro come siano equivalenti gli ultimi due grafi descritti.

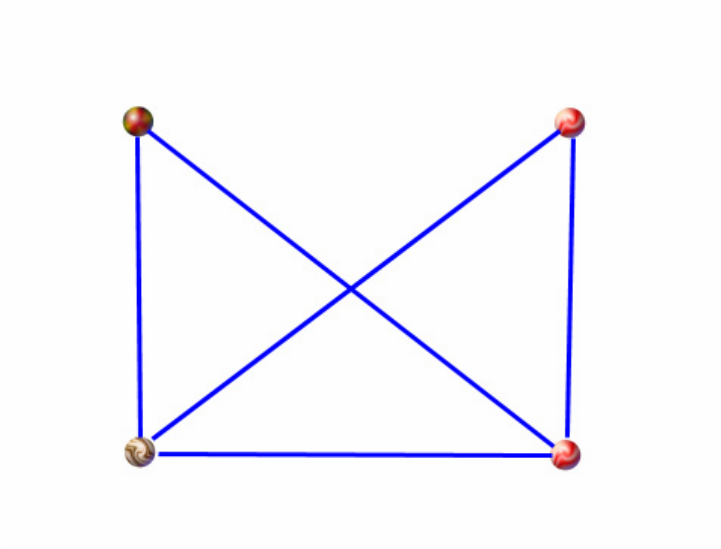
Grafo duale

Il grafo duale D di un grafo duale G è quel grafo che ha per nodi le facce di G e per archi le frontiere di quest'ultimo.

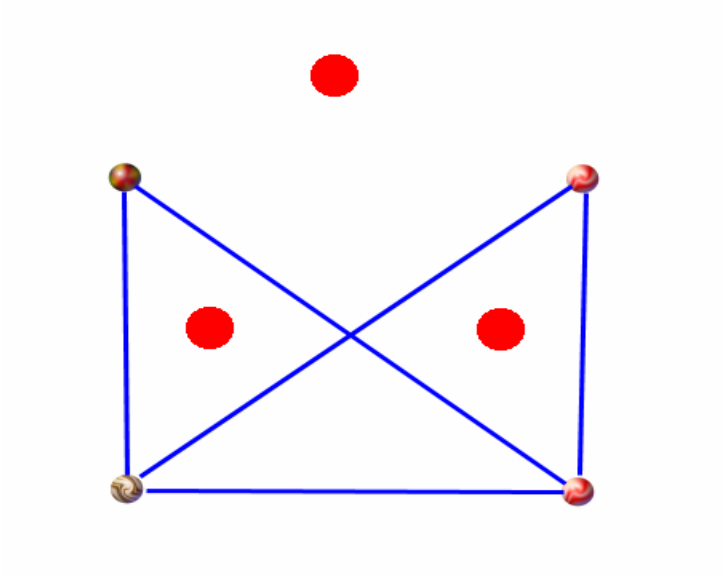
Frontiera

Per frontiera intendiamo l'arco che separa due facce distinte.

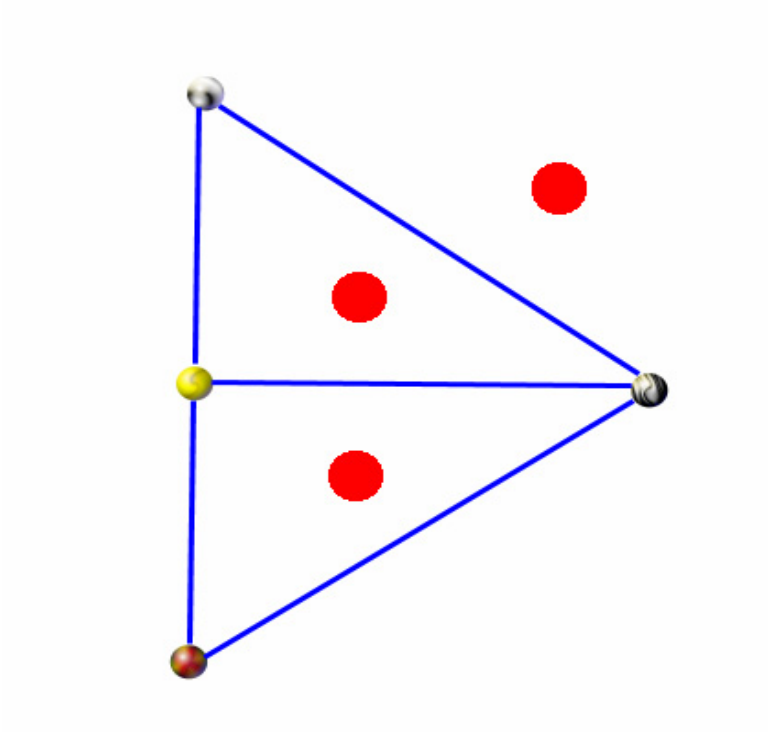
Esempio 9.3



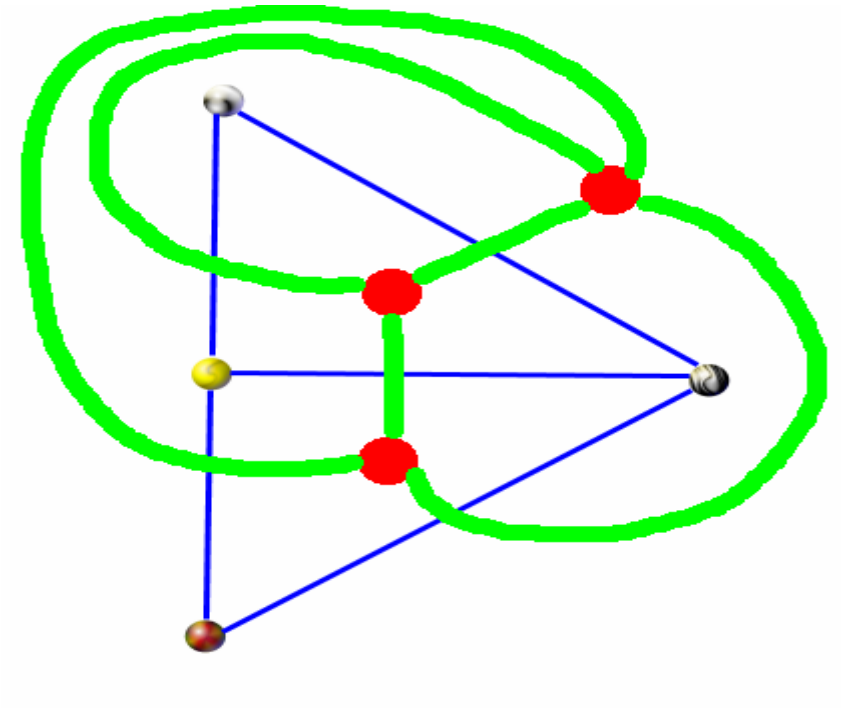
Mettiamo in ogni faccia univoca, un nodo per la costruzione del duale D.



Notiamo intanto una cosa molto importante: la faccia esterna al grafo G viene considerata e pertanto dovrà essere considerato anche il nodo di quest'ultima (quello in alto per intenderci).
Inoltre, abbiamo detto *ogni faccia univoca*, questo significa che il triangolino in basso non è da considerarsi.
Il motivo è molto semplice ed altrettanto lo è la ricerca delle facce: abbiamo detto nella definizione di duale di considerare esclusivamente grafi planari dove l'identificazione delle facce è univoca.
Se infatti vedessimo da quest'altra angolazione il grafo appena disegnato, ci parrebbe tutto più semplice.

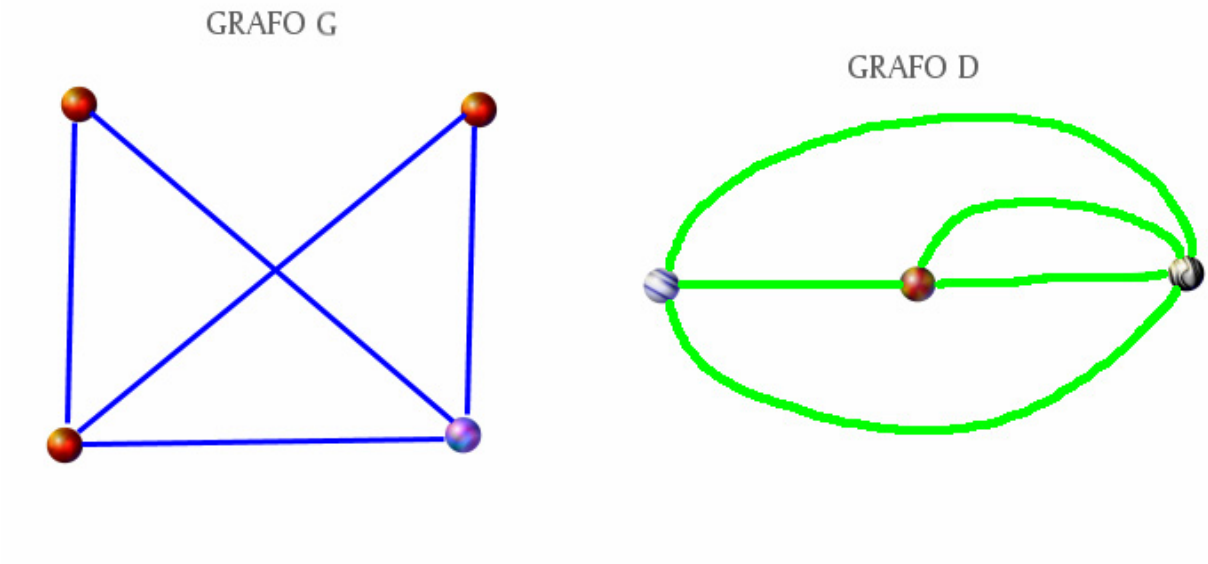


E a questo punto completiamo la definizione di grafo duale.



Ogni volta che esaminiamo un 'accoppiamento di nodi', dobbiamo in sostanza capire se questi due nodi rappresentino due regioni contigue o meno. Se sono contigue, identifichiamo tutte le loro frontiere e tracciamo gli eventuali archi del duale.

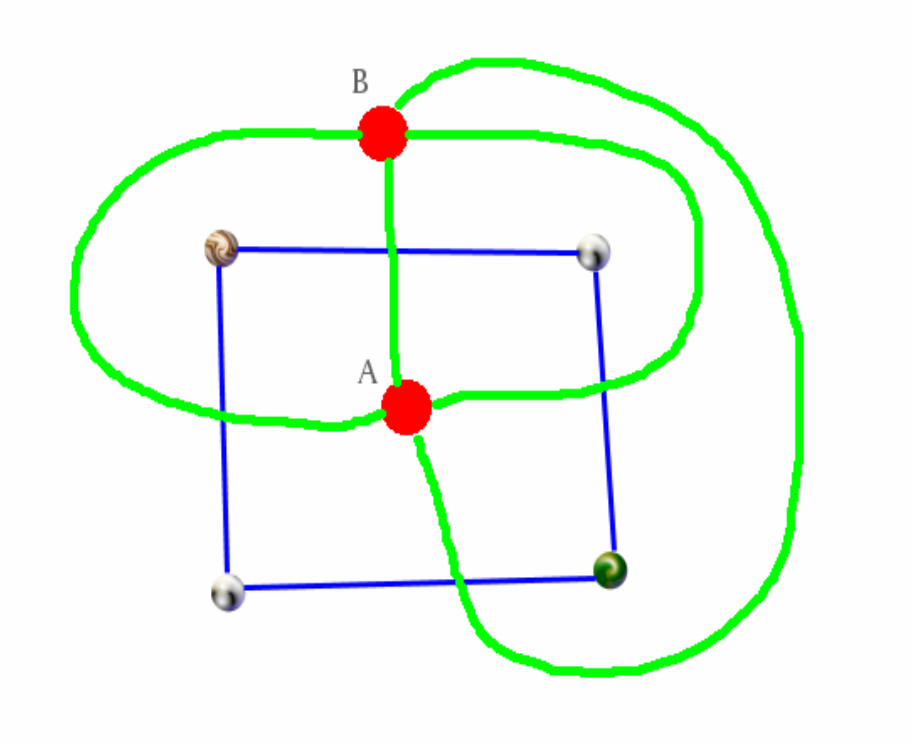
Vediamo dunque in sintesi il grafo originario G ed il suo duale D.



Osserviamo come il duale di un grafo planare possa essere un grafo multiplo e come il grado dei nodi del duale D sia pari al numero degli archi della faccia alla quale appartenevano in G.

Vediamo un altro esempio per la costruzione del duale.

Esempio 9.4



Osserviamo come il nodo A appartenga ad una faccia con 4 archi e allo stesso modo il nodo B faccia parte di una regione delimitata da quattro 'lati'; i nodi A e B del duale hanno grado pari a quattro.

PROPRIETA' DEL DUALE

- il grafo duale D' di un grafo duale D corrisponde al grafo originario G di cui D ne è il duale

Praticamente il duale del duale di G è nuovamente G. Si noti bene come questa proprietà sia verificata solo se il grafo G è connesso.

- dati due grafi isomorfi, i loro duali non sono necessariamente isomorfi

TEOREMA 9.1

Se abbiamo un grafo G planare dove;

- G è bipartito
- ogni sua faccia ha un numero pari di archi che la delimitano
- il duale di G è euleriano

il verificarsi di due di queste tre condizioni, implica il verificarsi della terza.

FORMULA DI EULERO (1758)

Se G è un grafo con n nodi, m archi, connesso, planare e con f facce, allora si ha che:

- $n-m+f = 2$

Si ricordi bene come nel numero f sia compresa anche la faccia esterna.

TEOREMA DI KURATOWSKI (1930)

Un grafo è planare se non contiene cliques di ordine 5 o grafi K_{33}.

Definizione

Ipotizzando un generico arco orientato dal nodo i al nodo j, indicheremo:

- C_{ij} costo per unità di flusso
- u_j quantità massima trasportabile sull'arco

Condizione di Bellman

$$d(j) \geq d(i) + C_{ij}$$

Ovvero, per un arco orientato da i a j e di costo C_{ij}, la distanza del nodo j può essere maggiore o uguale della distanza del nodo i. Le distanze sono supposte da una terza fonte (o terzo nodo) e i costi sono a loro volta supposti positivi.
La condizione di Bellman opportunamente modificata ci darà le condizioni necessarie e sufficienti per assicurare che un cammino sia minimo.

La condizione è la seguente:

$$d(j) = d(i) + C_{ij}$$

ALGORITMO DI DIJKSTRA

La complessità di quest'algoritmo è $O(n^2)$.

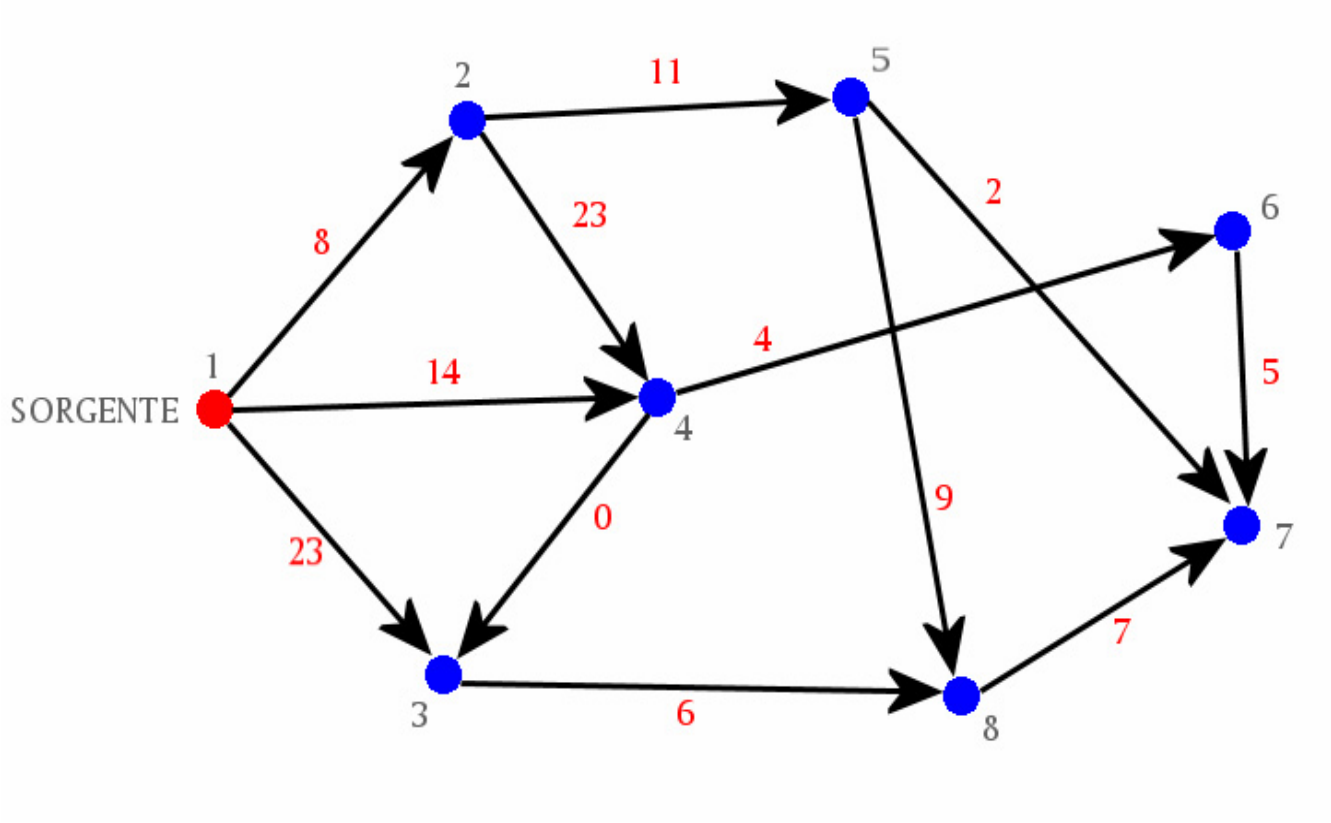
Le condizioni per implementare sul grafo G l'algoritmo di Dijkstra sono queste:

- il grafo G è aciclico
- il grafo G è connesso
- il grafo G ha un nodo sorgente

Lo scopo di quest'algoritmo è quello di scoprire la distanza minima di tutti i nodi dal nodo sorgente. L'algoritmo non fa altro che verificare su ogni nodo la condizione di Bellman non modificata.
Si riempiranno due liste e se ne svuoterà una, questa sarà quella dei nodi facenti parte della lista d'adiacenza (S'). Le altre due liste rappresenteranno via via le distanze di ogni nodo esaminato dal nodo sorgente e l'insieme dei nodi esaminati. In partenza supporremo che tutti i nodi distino una distanza infinita dal nodo sorgente; se la condizione di Bellman è verificata aggiorneremo man mano queste distanze.

Testiamolo sull'esempio 9.5 della pagina seguente.

Esempio 9.5



STEP 1

S = { null }
S' = { 1,2,3,4,5,6,7,8 }

d(S') = { 0, ∞, ∞, ∞, ∞, ∞, ∞ }

STEP 2

S = { 1 }

d(2) > d(1) + 8 = 8 SI → pred(2) = 1
d(3) > d(1) + 23 = 23 SI → pred(3) = 1
d(4) > d(1) + 14 = 14 SI → pred(4) = 1

d(S') = { 0,8,23,14, ∞, ∞, ∞, ∞}

STEP 3

S = { 1,2 }

d(4) > d(2) + 23 = 31 NO → 14 < 31
d(5) > d(2) + 11 = 19 SI → pred(5) = 2

d(S') = { 0,8,23,14, 19, ∞, ∞, ∞}

STEP 4

S = { 1,2,4 }

d(3) > d(4) + 0 = 14 SI → pred(3) = 4
d(6) > d(4) + 4 = 18 SI → pred(6) = 4

d(S') = { 0,8,14,14, 19, 18, ∞, ∞}

STEP 5

S = { 1,2,4,3 }

d(8) > d(3) + 6 = 20 SI → pred(8) = 3

d(S') = { 0,8,14,14, 19, 18, ∞, 20}

STEP 6

S = { 1,2,4,3,6 }

d(7) > d(6) + 5 = 23 SI → pred(7) = 6

d(S') = { 0,8,14,14, 19, 18, 23, 20}

STEP 7

S = { 1,2,4,3,6,5 }

d(7) > d(5) + 2 = 21 SI → pred(7) = 5
d(8) > d(5) + 9 = 28 NO → 20 < 28

d(S') = { 0,8,14,14, 19, 18, 21, 20}

STEP 8

S = { 1,2,4,3,6,5,8 }

d(7) > d(8) + 7 = 27 NO → 21 < 27

d(S') = { 0,8,14,14, 19, 18, 21, 20}

STEP 9

S = { 1,2,4,3,6,5,8,7 }

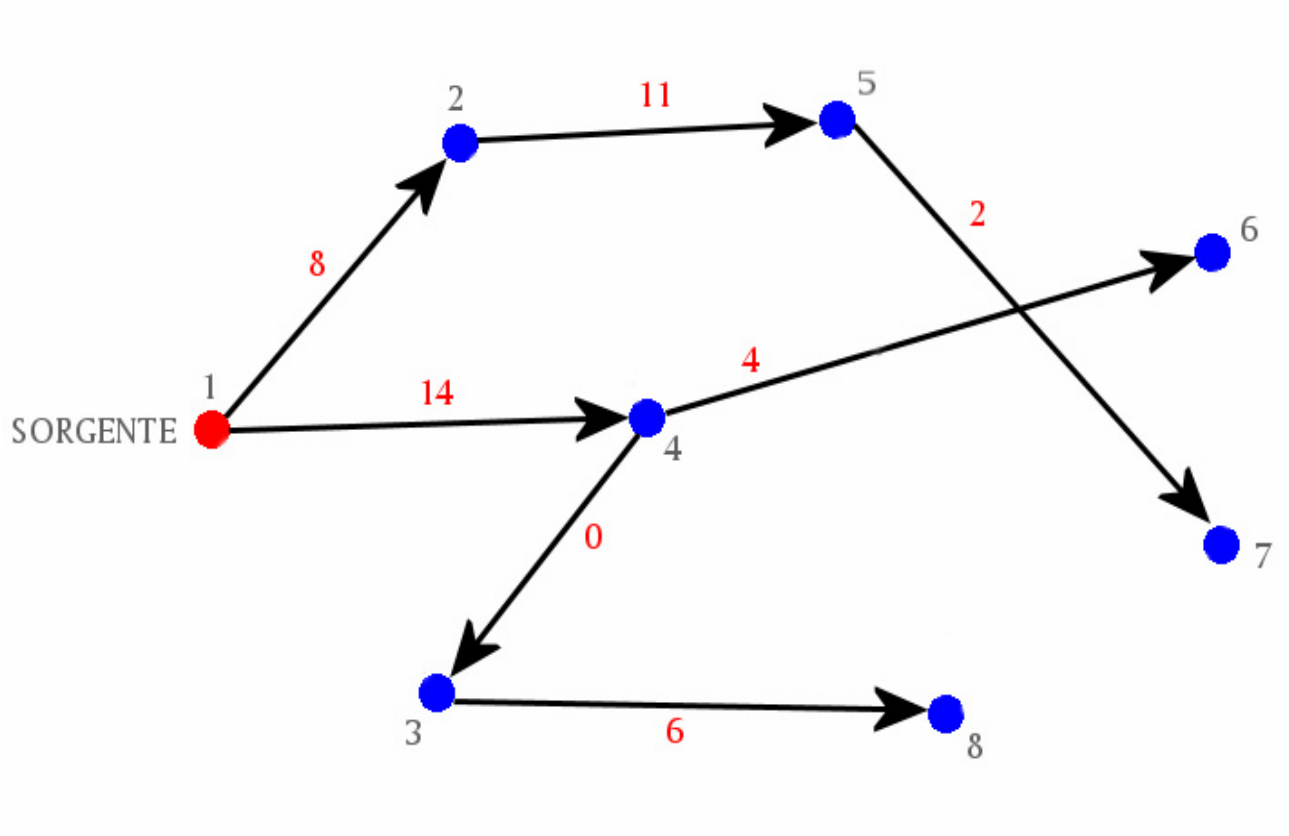
Il nodo 7 non ha archi uscenti.

d(S') = { 0,8,14,14, 19, 18, 21, 20}

L'algoritmo è terminato e la tabella riassuntiva è la seguente:

NODO	PREDECESSORE	DISTANZA DALL'ORIGINE
1	null	0
2	1	8
3	4	14
4	1	14
5	2	19
6	4	18
7	5	21
8	3	20

E vediamo il risultato appena ottenuto nel grafo alla pagina successiva.



Per ogni nodo abbiamo ottenuto la più breve strada possibile dal nodo sorgente 1.

Grafo fortemente connesso

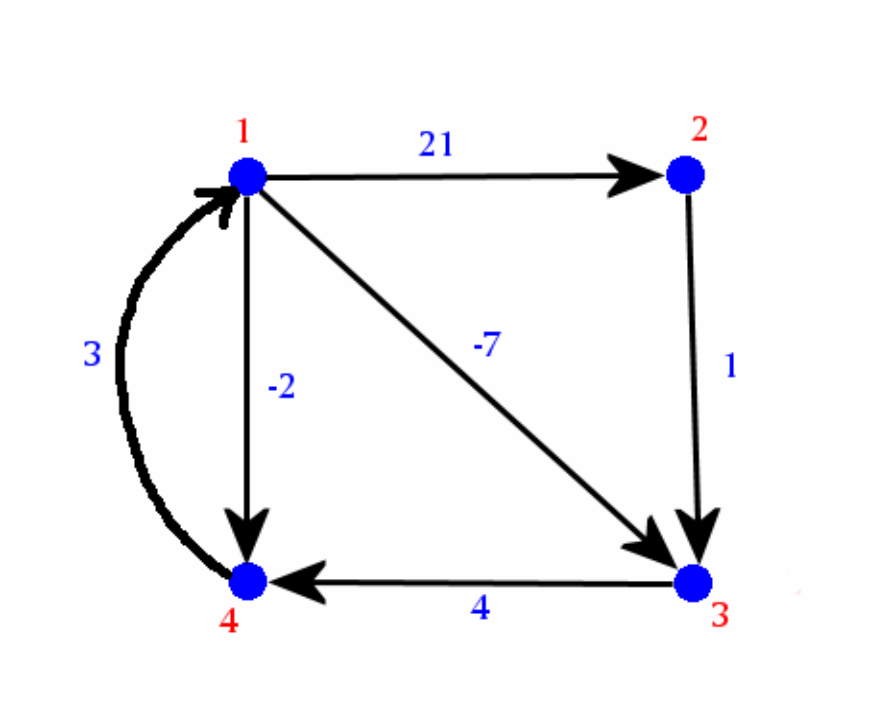
E' quel grafo orientato e connesso nel quale da ogni nodo si può raggiungere qualsiasi altro nodo del grafo.

ALGORITMO DI FLOYD-WARSHALL

Quest'algoritmo, a differenza di quello di Dijkstra, trova il cammino tra ogni coppia possibile di nodi. La stesura di tutte le condizioni di Bellman è molto più tediosa poiché esamina di volta in volta tutti i cammini possibili verso tutti gli altri nodi partendo da un nodo sorgente K.

L'esecuzione non è complessa anzi, ma l'esempio 9.6 sarà più chiaro di qualsiasi spiegazione teorica.

Esempio 9.6



La situazione iniziale è la seguente:

d(i,j)	1	2	3	4
1	0	21	-7	-2
2	infinito	0	1	infinito
3	infinito	infinito	0	4
4	3	infinito	infinito	0

Pare subito ovvio come la distanza da un generico nodo i a se stesso sia nulla, per tutte le altre, come in Dijkstra, metteremo infinito se ancora non la conosciamo o w є Z se abbiamo un arco diretto che c'indichi la distanza attuale.

In quest'altra tabella indicheremo invece i predecessori. E' facile riempirla prima di cominciare l'algoritmo poiché metteremo su ogni elemento della colonna che ha distanza diversa da zero o infinito, il numero della riga. Là dove invece avevamo infinito nella tabella delle distanze, metteremo zero. Per quanta riguarda i predecessori dell'arco che ricade su se stessi, il discorso è lo stesso visto nella tabella precedente.

pred(i,j)	1	2	3	4
1	0	1	1	1
2	0	0	2	0
3	0	0	0	3
4	4	0	0	0

Analizziamo K =1.

d(2,3) > d(2,1) + d(1,3) FALSO
d(2,4) > d(2,1) + d(1,4) FALSO

d(3,2) > d(3,1) + d(1,2) FALSO
d(3,4) > d(3,1) + d(1,4) FALSO

d(4,2) > d(4,1) + d(1,2) VERO → d(4,2) = 24; pred(4,2) = 1
d(4,3) > d(4,1) + d(1,3) VERO → d(4,3) = -4 ; pred(4,3) = 1

Le tabelle diventano:

d(i,j)	1	2	3	4
1	0	21	-7	-2
2	infinito	0	1	infinito
3	infinito	infinito	0	4
4	3	24	-4	0

pred(i,j)	1	2	3	4
1	0	1	1	1
2	0	0	2	0
3	0	0	0	3
4	4	1	1	0

Analizziamo adesso K = 2.

d(1,3) > d(1,2) + d(2,3) FALSO
d(1,4) > d(1,2) + d(2,4) FALSO

d(3,1) > d(3,2) + d(2,1) FALSO
d(3,4) > d(3,2) + d(2,4) FALSO

d(4,1) > d(4,2) + d(2,1) FALSO
d(4,3) > d(4,2) + d(2,3) FALSO

Non aggiorniamo dunque nessun termine.

d(i,j)	1	2	3	4
1	0	21	-7	-2
2	infinito	0	1	infinito
3	infinito	infinito	0	4
4	3	24	-4	0

pred(i,j)	1	2	3	4
1	0	1	1	1
2	0	0	2	0
3	0	0	0	3
4	4	1	1	0

Vediamo a K = 3.

d(1,2) > d(1,3) + d(3,2) FALSO
d(1,4) > d(1,3) + d(3,4) VERO → d(1,4) = -3; pred(1,4) = 3

d(2,1) > d(2,3) + d(3,1) FALSO
d(2,4) > d(2,3) + d(3,4) VERO → d(2,4) = 5 ; pred(2,4) = 3

d(4,1) > d(4,3) + d(3,1) FALSO
d(4,2) > d(4,3) + d(3,2) FALSO

d(i,j)	1	2	3	4
1	0	21	-7	-3
2	infinito	0	1	5
3	infinito	infinito	0	4
4	3	24	-4	0

pred(i,j)	1	2	3	4
1	0	1	1	3
2	0	0	2	3
3	0	0	0	3
4	4	1	1	0

Vediamo adesso K = 4.

d(1,2) > d(1,4) + d(4,2) FALSO
d(1,3) > d(1,4) + d(4,3) FALSO

d(2,1) > d(2,4) + d(4,1) VERO → d(2,1) = 8; pred(2,1) = 4
d(2,3) > d(2,4) + d(4,3) FALSO

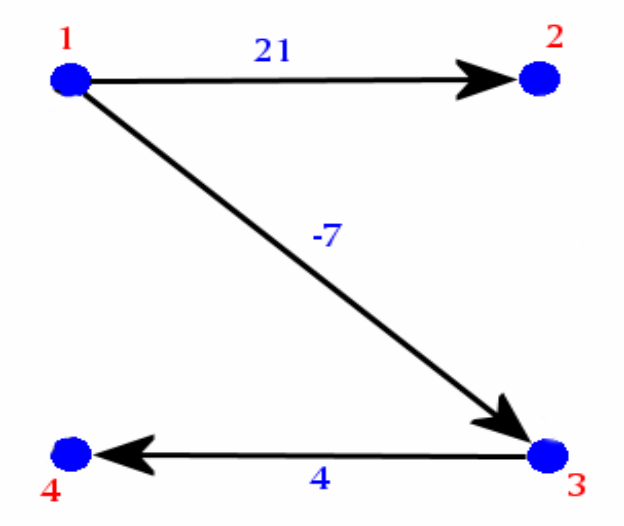
d(3,1) > d(3,4) + d(4,1) VERO → d(3,1) = 7; pred(3,1) = 4
d(3,2) > d(3,4) + d(4,2) VERO → d(3,2) =28; pred(3,2) = 4

d(i,j)	1	2	3	4
1	0	21	-7	-3
2	8	0	1	5
3	7	28	0	4
4	3	24	-4	0

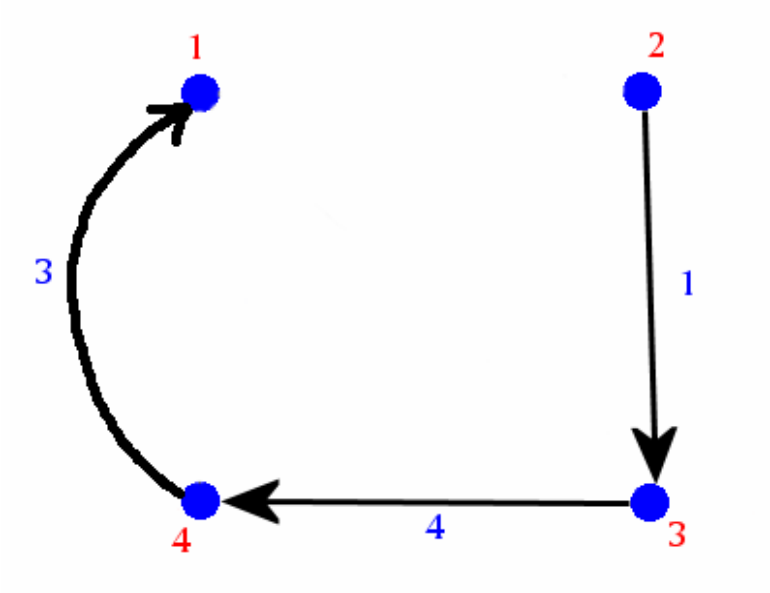
pred(i,j)	1	2	3	4
1	0	1	1	3
2	4	0	2	3
3	4	4	0	3
4	4	1	2	0

Ed i grafici corrispondenti sono questi:

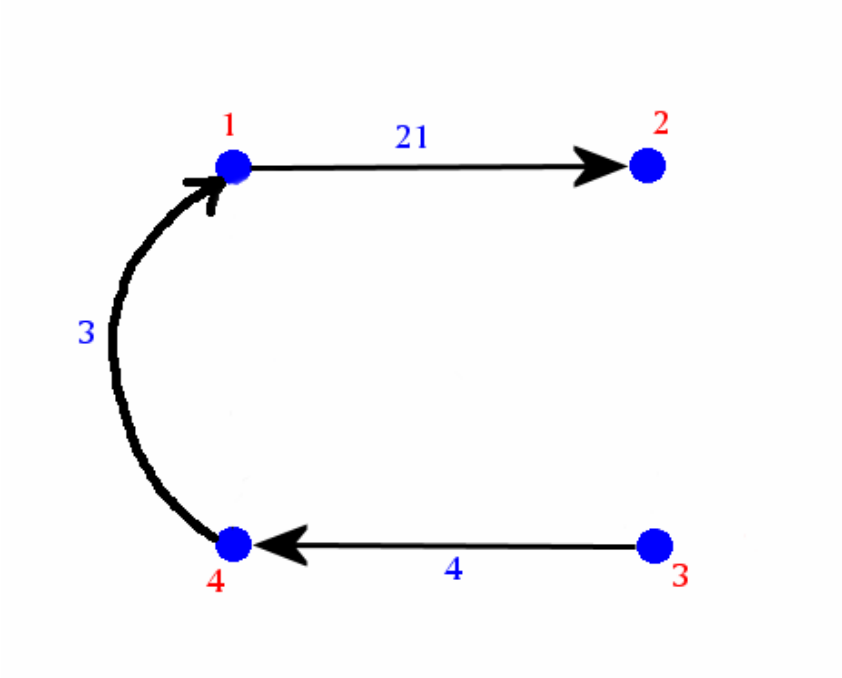
PARTENDO DAL NODO 1



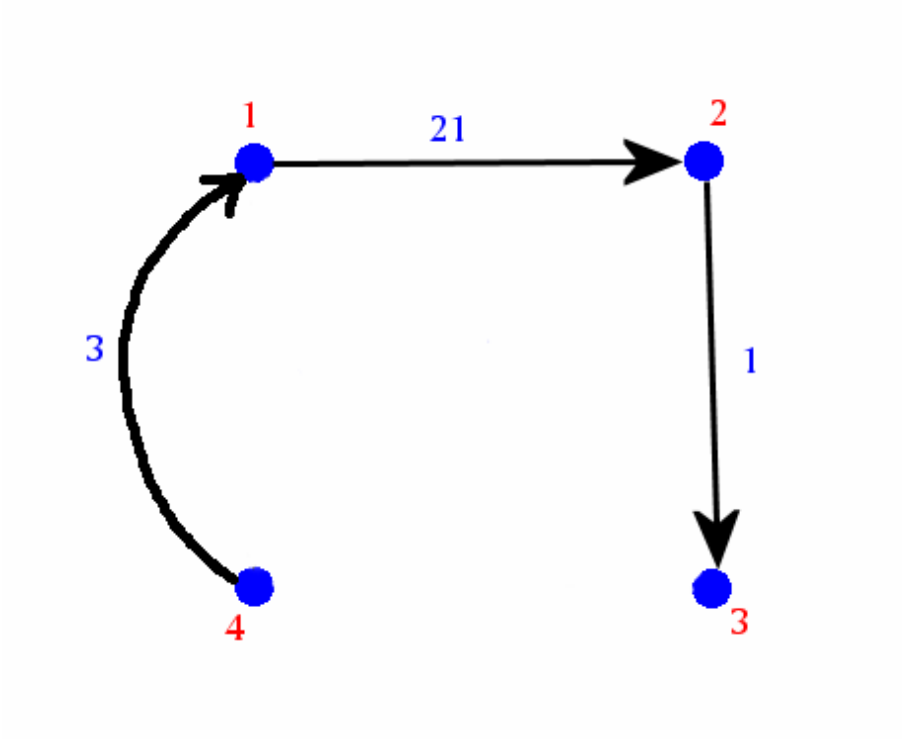
PARTENDO DAL NODO 2



PARTENDO DAL NODO 3



PARTENDO DAL NODO 4



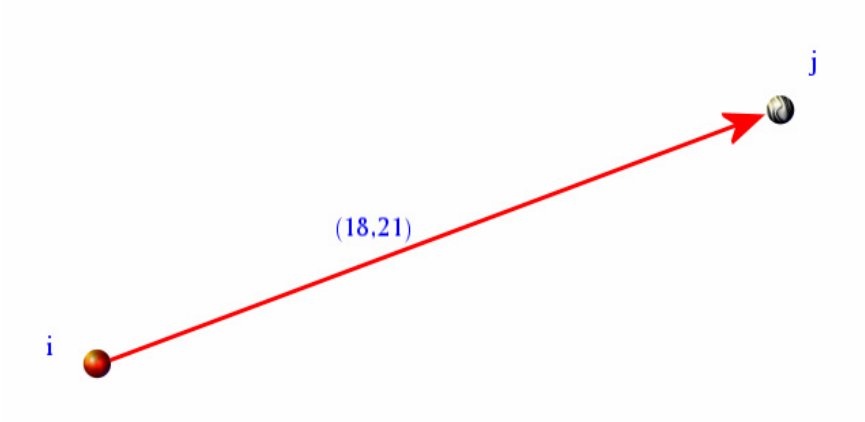
Vediamo adesso la tecnica dei cammini aumentanti.

Grafo residuo

Dato un grafo orientato e connesso, il suo grafo residuo è quello che vede per ogni generico arco dal nodo i al nodo j l'aggiunta di un nuovo arco in senso opposto. Se il primo arco originario indicava il flusso e la sua massima capacità (x_{ij}, u_{ij}), allora l'arco originario sarà quell'arco dove è possibile ancora far passare una quantità ($u_{ij} - x_{ij}$) mentre l'arco nuovo aggiunto sarà quell'arco pari ad x_{ij}.

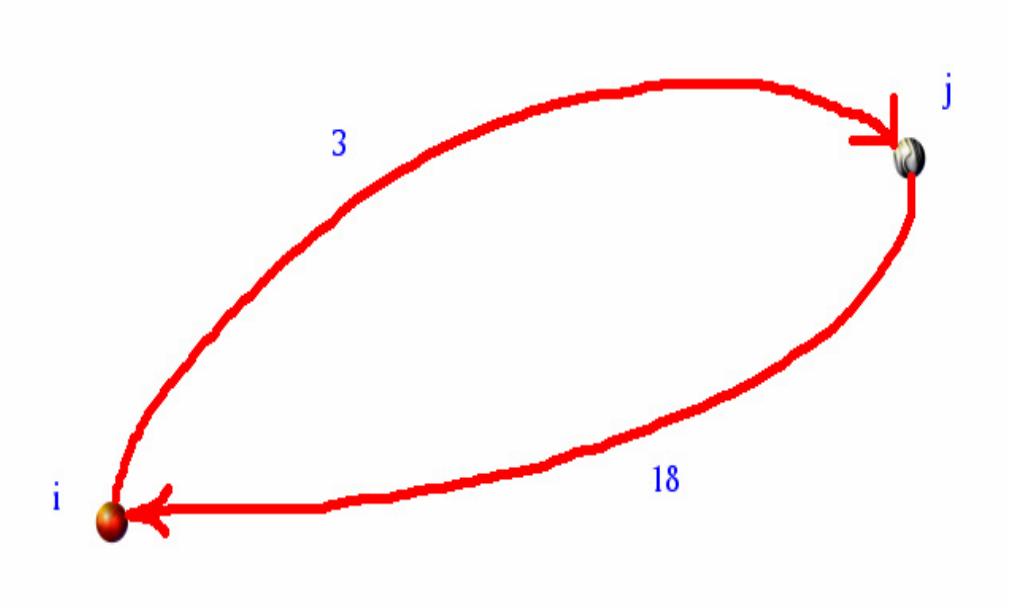
Vediamo il seguente esempio.

Esempio 9.7



Su quest'arco è possibile trasportare una quantità massima pari a 21 unità, ma attualmente il flusso è di 18 unità.

Il suo grafo residuo è:



Si può vedere come sull'arco originario sia ancora possibile far passare una quantità pari a 3 unità e cioè (21-18), mentre l'arco che torna indietro sta ad indicare il trasferimento ormai avvenuto delle 18 unità.

Cammino aumentante

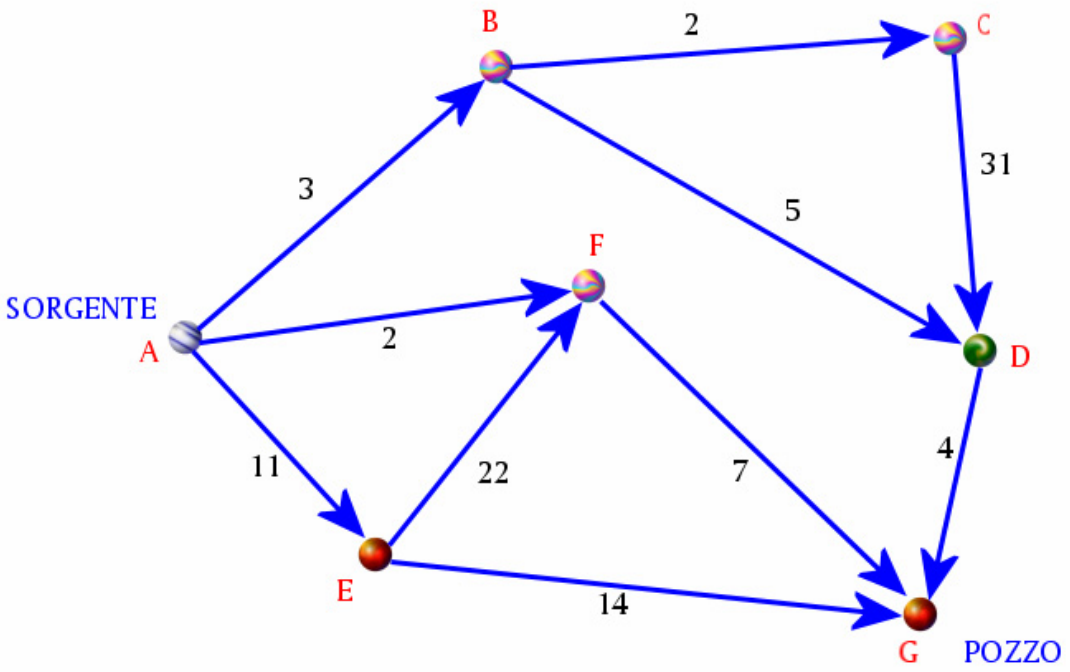
E' quel cammino dal generico nodo i al generico nodo j lungo il quale è ancora possibile trasportare una quantità k di elementi.

ALGORITMO DI FORD-FULKERSON

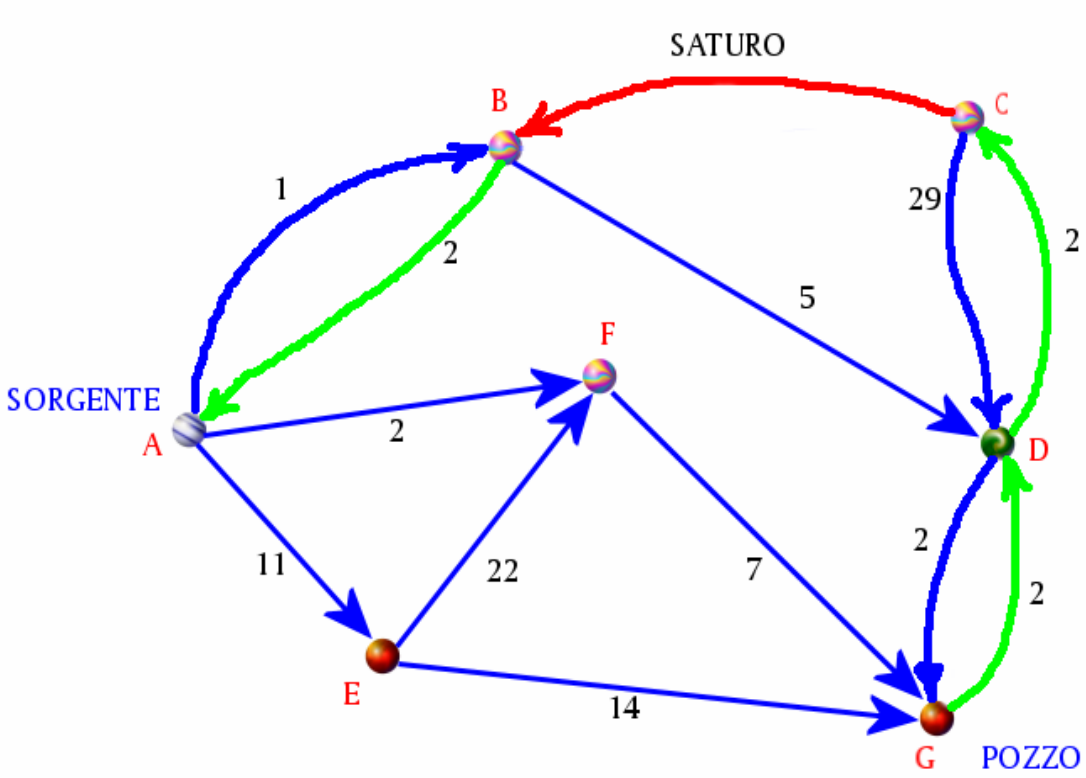
L'intento è quello di trasferire da un generico nodo sorgente i ad un generico nodo 'pozzo' j il massimo flusso possibile.

Quest'algoritmo provvederà alla soluzione di questo problema.

Sugli archi verrà indicato solo il massimo flusso; la massima capacità dell'arco.



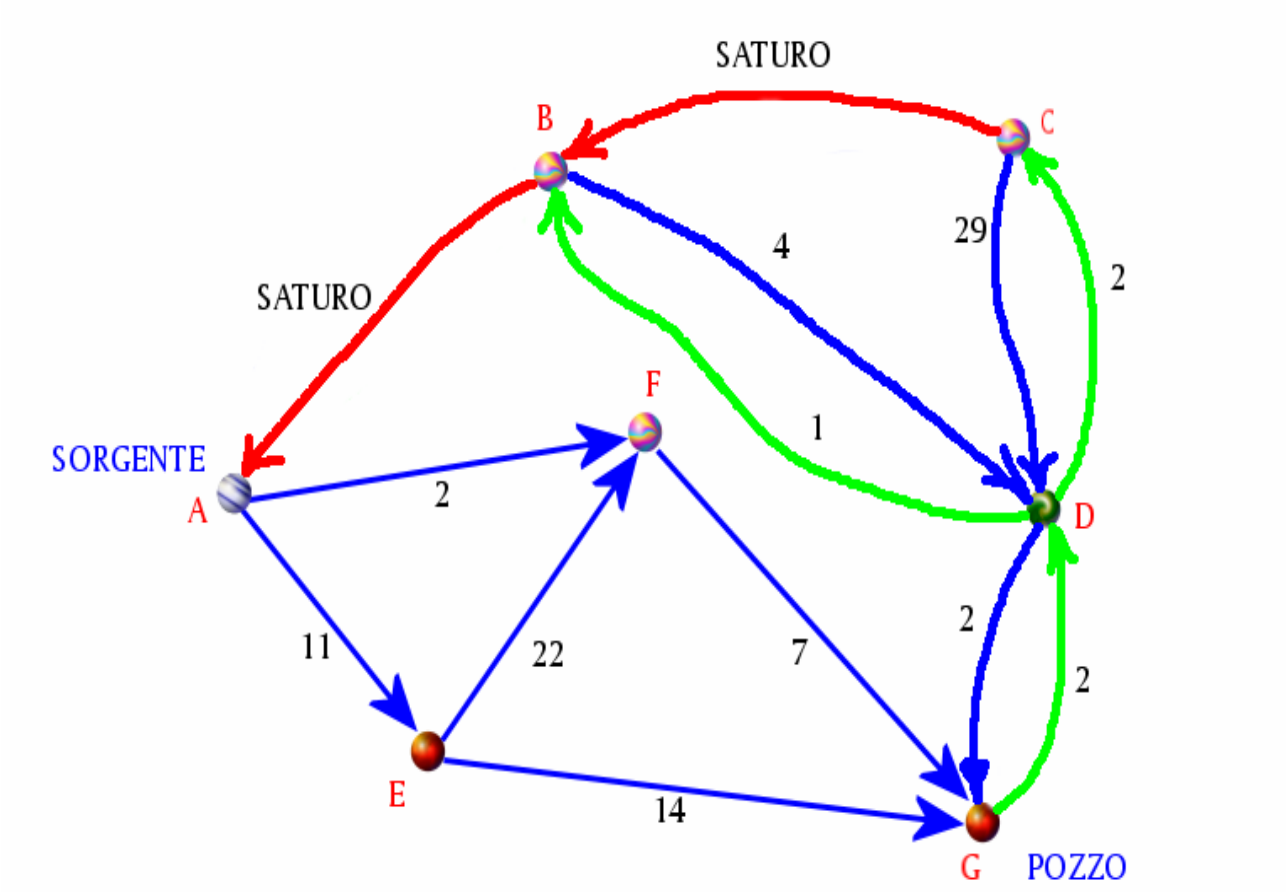
CAMMINO AUMENTANTE: **ABCDG**



La quantità totale trasportata finora è pari a V = 2.

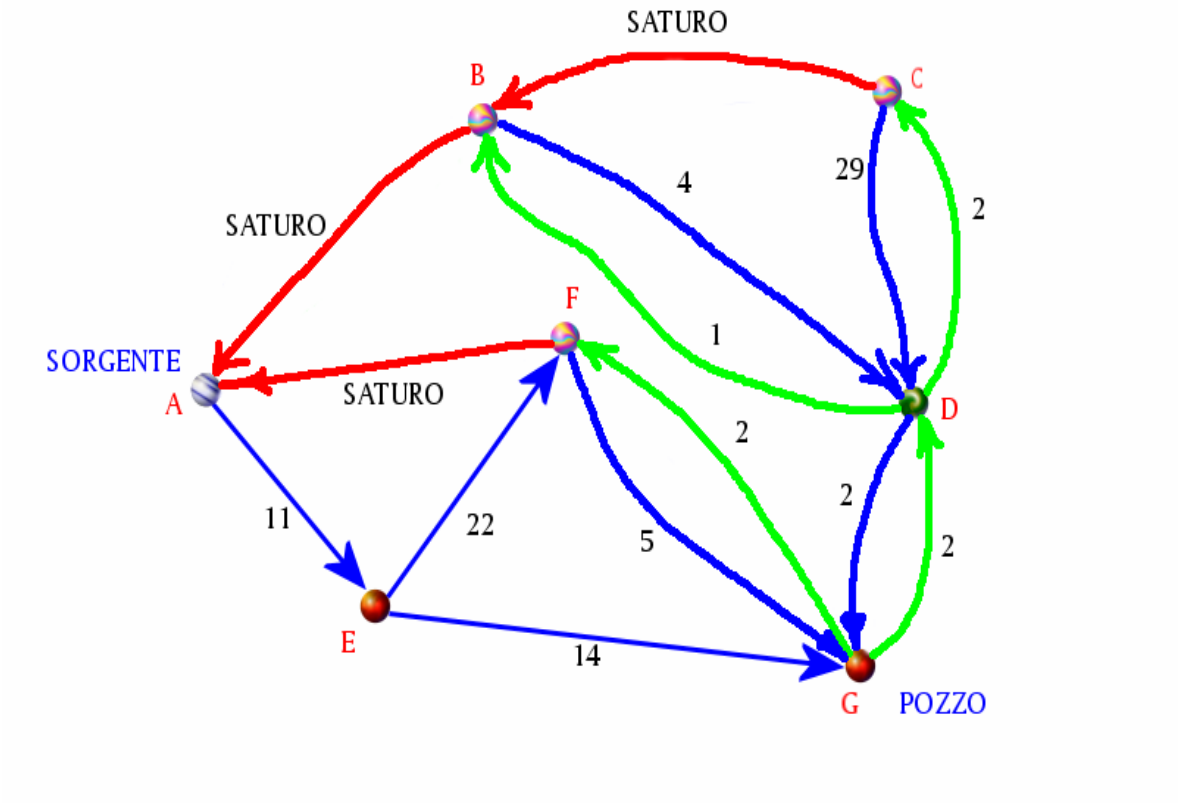
Troviamo adesso un altro cammino possibile.

CAMMINO AUMENTANTE: **ABDG**

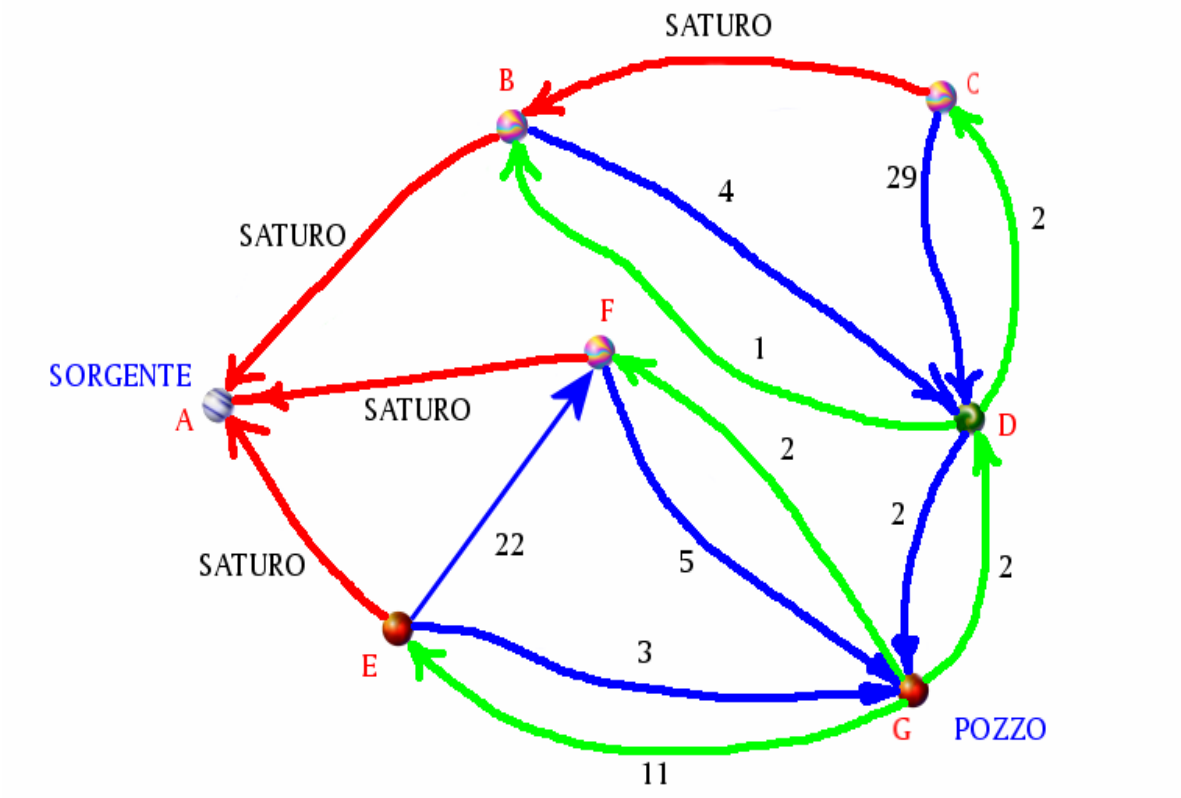


Abbiamo fatto giungere in G solo un'unità; V è ora pari a 3.

Scegliamo adesso un altro cammino aumentante: **AFG**.



La quantità giunta nel pozzo è V = 5.
Un possibile cammino aumentante può essere: **AEG**.



E V diventa pari a 16.

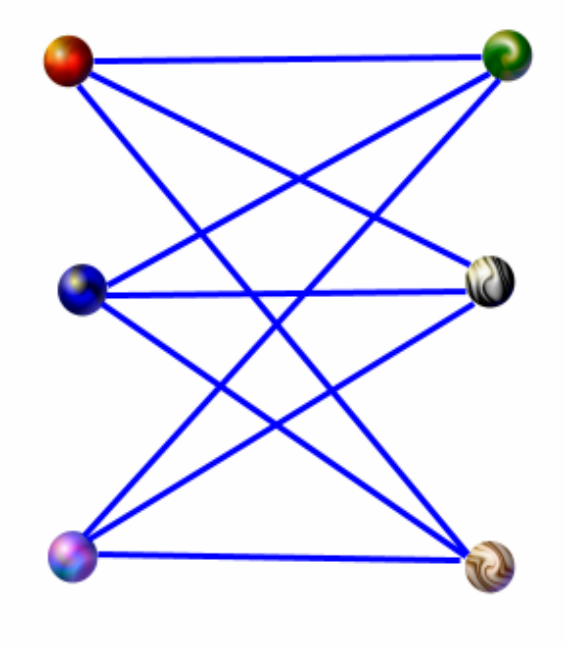
Non esistendo più cammini aumentanti (frecce blu che partano da A), l'algoritmo ha termine ed il massimo flusso trasportabile dal nodo sorgente al nodo pozzo è proprio 16.

Scegliendo altri cammini aumentanti il risultato sarà il medesimo.

APPENDICE

Abbiamo visto a pagina 85 il teorema di Kuratowski.

Questo teorema afferma che per avere la certezza che un grafo sia planare, dobbiamo esser sicuri che al suo interno non si 'celino' sottografi isomorfi a K_5 o a K_{33}, non deve avere cioè grafi completi di ordine 5 (vedi pagina 16) o grafi completi bipartiti con 3 elementi da una parte e 3 dall'altra:



Questa proprietà dei grafi planari ha generato il famoso problema dove bisogna collegare tre case e tre punti diversi di luce, acqua e gas….particolarità: le condutture non devono intersecarsi.

POSTMESSA

Spero vi siate divertiti.

Spero di aver contribuito almeno un pò nel far conoscere questa materia ancora ignota ai più, ma così potente da farvi trovare in un tempo di complessità logaritmica il miglior tragitto per raggiungere la fermata degli autobus, attendere in un lasso di tempo decisamente 'little-O' di tutte le più lunghe attese da mezzo pubblico il vostro autobus orientato per poter connettere il luogo in cui eravate con la vostra misera dimora isolata da quel grafo aciclico che state intraprendendo.

Spero anche che una volta giunti a casa con l'aiuto del buon Dijkstra possiate sentire il calore di un 'vicinato' ricoprente le vostre aspettative di socializzazione quotidiana, nell'attesa che vi si possa connettere al più presto la vostra dolce metà con un cammino euleriano per farvi sentire meno isomorfi possibile.

Infine, spero che non siate costretti ad r-permutare la vostra intensa serata con un traffico automobilistico non orientato, ciclico, non ricoprente, irregolare e poco topologico. Il mio forte augurio è che voi siate sulla lista d'adiacenza del locale nel quale state andando senza che dobbiate operare un taglio al portafoglio per corrompere la condizione di Bellman di ogni buttafuori:

$$C(b) \geq S(p) + m$$

Dove C(b) è il costo del buttafuori, S(p) è pari all'ammontare liquido del vostro portafoglio ed m sta per macchina…
…spero non la vostra.

Mi raccomando, non siate planari e se giocate a calcio, non fatevi marcare come un povero albero da giardinetto per cani.

Alla prossima ☺

Desmatron

POLLICE