1. Introduzione ai grafi e alle reti

1.1 Grafi, cammini, alberi: formalizzazione e memorizzazione

1.1.1 Grafi orientati e non orientati

$G = (N, A)$, in cui N è un insieme finito di elementi e A è un insieme (senza ripetizioni) di coppie di elementi (distinti) di N, è detto un *grafo*. N è detto insieme di *nodi* o *vertici*; usualmente viene indicato mediante i primi $n = |N|$ numeri naturali: $N = \{1, 2,\ldots, n\}$. L'insieme degli archi A, con $|A| = m$, è in generale indicato con $A = \{a_1, a_2,\ldots, a_m\}$, in cui il k-esimo arco viene indifferentemente rappresentato anche come coppia di nodi: $a_k = (i,j)$.

Se la coppia non è ordinata, cioè (i,j) e (j,i) sono lo stesso arco, allora l'arco è detto *non orientato*, altrimenti è detto *orientato* e in tal caso (i,j) e (j,i) rappresentano due archi diversi. I nodi i e j sono detti *estremi* dell'arco $a_k = (i,j)$, che è *incidente* in essi; i e j sono detti *adiacenti*. Se l'arco è orientato, allora i è la *coda* e j è la *testa* di $a_k = (i,j)$; a_k è *uscente* da i e *entrante* in j; inoltre, j è un *successore immediato* di i e questo è un *predecessore immediato* di j.

Un grafo i cui archi sono tutti non orientati è detto *grafo non orientato*, ed è detto *grafo orientato* se tutti i suoi archi sono orientati. Un grafo può essere rappresentato graficamente utilizzando un cerchio numerato per ogni nodo e un segmento tra due nodi per ogni arco; se l'arco è orientato si usa una "freccia" per rappresentare l'ordinamento dell'arco; in figura 1.1 sono rappresentati un grafo non orientato (a) e un grafo orientato (b).

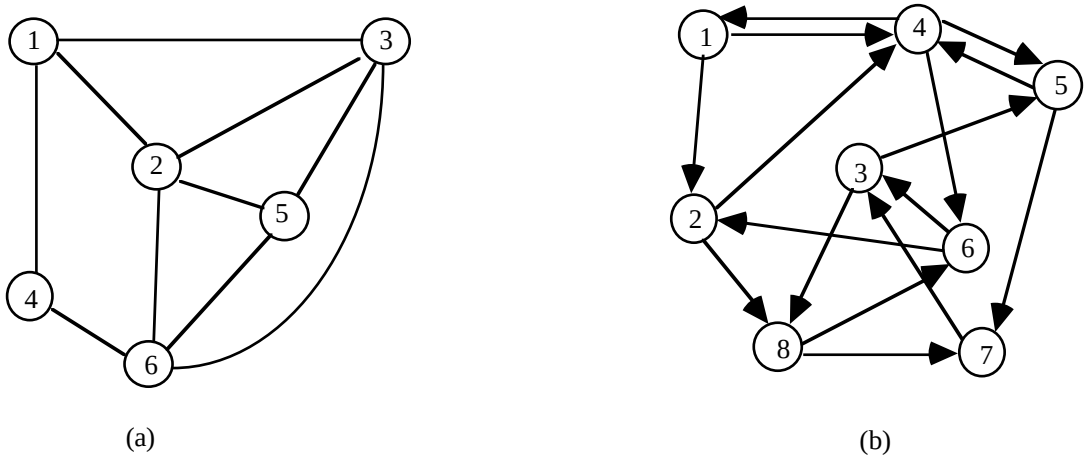


Figura 1.1 - Un grafo non orientato (a) e uno orientato (b)

Dato un grafo (orientato o non orientato), per ogni nodo $i \in N$ si indica con $N(i)$ *l'insieme dei nodi adiacenti* ad i e con $S(i)$ *l'insieme degli archi incidenti* in i, detto *stella* di i; $|N(i)| = |S(i)|$ è detto il *grado* del nodo i ed è indicato con g_i. Vale la relazione $m = \Sigma_i\, g_i$; essendo $g_i \le n$, se G è non orientato si ha che $m \le n^2/2$ (si ricorda che non vi sono ripetizioni di archi e non esistono coppie di archi uguali), mentre se G è orientato si ha che $m \le n^2$.

Dato un grafo orientato, per ogni nodo $i \in N$, si indica con $FN(i)$ e $BN(i)$ rispettivamente gli *insiemi dei nodi successori* e*predecessori immediati* , con $FS(i)$ e $BS(i)$ si indicano rispettivamente *gli insiemi degli archi uscenti* da i, o *stella uscente* di i, e *degli archi entranti* in i, o *stella entrante* di i:

$$FS(i) = \{(x,y) \in A: x=i\}, \qquad BS(i) = \{(x,y) \in A: y=i\}.$$

Valgono le relazioni $FN(i) \cup BN(i) = N(i)$ e $FS(i) \cup BS(i) = S(i)$.

$|FN(i)| = |FS(i)| = g_i^+$ è detto *grado uscente* mentre $|FN(i)| = |FS(i)| = g_i^-$ è detto *grado entrante* di *i*.

Dato un grafo $G = (N, A)$, il grafo $G' = (N, A')$, con $A' \subseteq A$, è detto *grafo parziale* di G; il grafo $G'' = (N'', A'')$, con $N'' \subseteq N$ e $A'' = \{(i,j) \in A: i, j \in N''\}$, è detto *grafo indotto* da N''. Un grafo $G^* = (N'', A^*)$, con $N'' \subseteq N$ e $A^* \subseteq A''$, è detto *sottografo* di G.

1.1.2 Cammini, cicli

Una sequenza di q coppie $C = \{(i_0,i_1), (i_1,i_2), \ldots, (i_{q-1},i_q)\}$, in cui o (i_{h-1},i_h) oppure (i_h,i_{h-1}) è un arco di G, $h = 1,\ldots, q$, è detto *cammino* del grafo tra i nodi i_0 e i_q. Se C è formato da archi orientati (i_{h-1},i_h), $h = 1,\ldots, q$, allora è detto *cammino orientato* da i_0 a i_q; i_0 è detto *origine* e i_q è detto *destinazione* di C. Una porzione di C è detta *sottocammino*. Se $i_0 = i_q$, C è detto un *ciclo* (eventualmente *orientato*).

Un cammino (ciclo) non contenente cicli come sottocammini (propri) è detto un *cammino (ciclo) semplice* (eventualmente *orientato*), in caso contrario è detto *non semplice*. Nei cammini (cicli) non semplici vi è sempre ripetizione di nodi; essi possono avere anche ripetizioni di archi, si veda la figura 1.2.

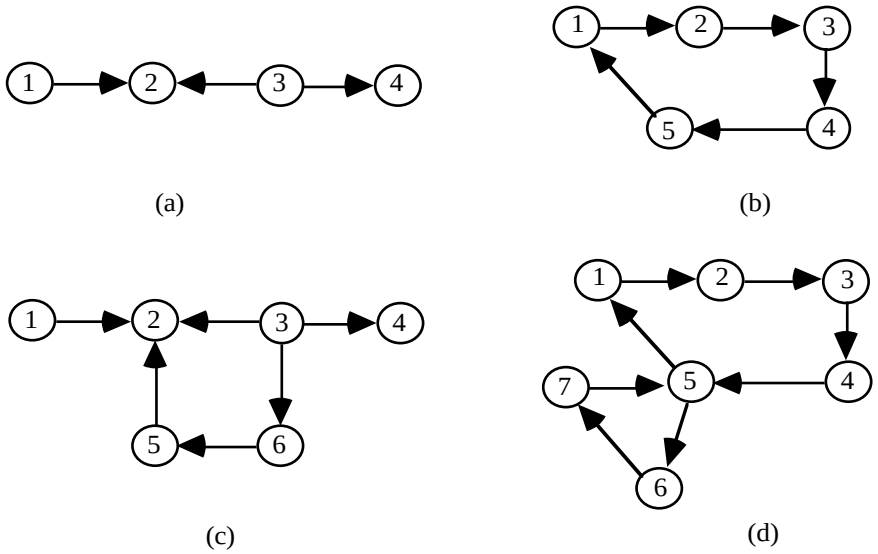


Figura 1.2 - (a) un cammino semplice; (b) un ciclo orientato semplice; (c) un cammino non semplice con ripetizione di archi; (d) un ciclo orientato non semplice senza ripetizione di archi

Un ciclo (orientato) semplice formato da *n* archi è detto *ciclo Hamiltoniano*; esso passa per ogni nodo del grafo una e una sola volta. Un ciclo (orientato) non semplice senza ripetizione di archi formato da *m* archi è detto *ciclo Euleriano*; esso passa attraverso ciascun arco del grafo una e una sola volta. Si può dimostrare che un grafo possiede un ciclo Euleriano se e solo se il grado di ogni nodo è pari, e che un grafo orientato possiede un ciclo Euleriano orientato se e solo se il grado uscente e il grado entrante di ogni nodo sono uguali.

Esercizio 1.1
Dimostrare l'affermazione precedente.

In un grafo $G = (N, A)$ due nodi sono *connessi* se esiste un cammino tra di essi. In un grafo orientato $G = (N, A)$ un nodo j è detto *connesso* ad un nodo i se esiste un cammino orientato da i a j.

1.1.3 Tagli e connettività

Dato un grafo $G = (N, A)$, una partizione di N in due sottoinsiemi non vuoti N' e N'' è detta un *taglio* del grafo e viene indicata con (N', N''); l'insieme degli archi aventi un estremo in N' e l'altro in N'' è detto *insieme degli archi del taglio* e verrà denotato con $A(N',N'')$:

$$A(N',N'') = \{(i,j) \in A: i \in N' \text{ e } j \in N'', \text{ oppure } j \in N' \text{ e } i \in N''\}.$$

Se il grafo è orientato, per ogni taglio si possono distinguere due insiemi di archi del taglio, l'insieme $A^+(N',N'')$ detto degli archi *diretti* del taglio e l'insieme $A^-(N',N'')$ detto degli archi *inversi* del taglio:

$$A^+(N',N'') = \{(i,j) \in A: i \in N' \text{ e } j \in N''\},$$
$$A^-(N',N'') = \{(i,j) \in A: j \in N' \text{ e } i \in N''\};$$

ovviamente $A(N',N'') = A^+(N',N'') \cup A^-(N',N'')$.

Mediante i tagli è possibile ridefinire quando due nodi i e j sono connessi; infatti essi sono *connessi* se non esiste un taglio (N', N'') tale che $i \in N'$, $j \in N''$ e $A(N',N'') = \emptyset$. Analogamente per i grafi orientati, un nodo j è *connesso* ad un nodo i se non esiste un taglio (N', N'') tale che $i \in N'$, $j \in N''$ e $A^+(N',N'') = \emptyset$.

Esercizio 1.2
Dimostrare l'equivalenza delle due definizioni di connessione tra due nodi e di un nodo ad un altro, basate rispettivamente sui cammini e sui tagli.

Un grafo $G = (N, A)$ è detto *connesso* se, comunque vengono presi due suoi nodi, essi sono connessi, altrimenti è detto *non connesso*. Un grafo orientato $G = (N, A)$ è detto *fortemente connesso* se, comunque vengono presi due suoi nodi, ciascuno di essi è connesso all'altro.

Dato un grafo non connesso $G = (N, A)$, ciascun grafo connesso indotto da un sottoinsieme massimale di N (rispetto all'inclusione) è detto una componente connessa di G (si veda la figura 1.3).

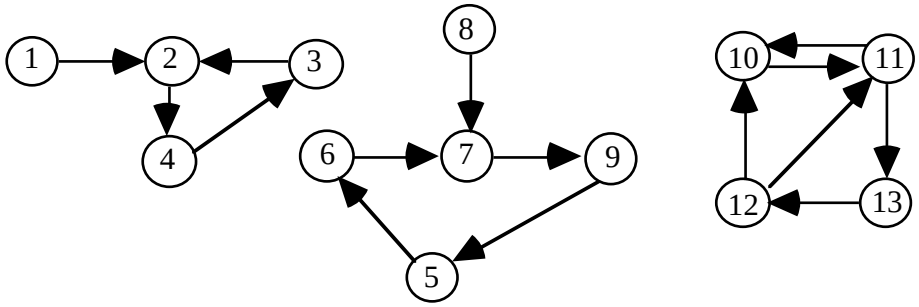


Figura 1.3 - Un grafo formato da tre componenti connesse; la componente connessa indotta da {10, 11, 12, 13} è un grafo fortemente connesso

Un grafo è detto *completo* se esiste un arco tra ogni coppia di nodi. Pertanto un grafo non orientato completo possiede $m = n(n\text{-}1)/2$ archi, mentre un grafo orientato possiede $m = n(n\text{-}1)$ archi.

Un grafo $G = (N, A)$ è detto *bipartito* se esiste un taglio (N', N'') il cui insieme degli archi $A(N',N'')$ coincide con A. Un grafo (orientato) bipartito è detto *completo* se per ogni coppia di nodi non appartenenti al medesimo sottoinsieme del taglio, esiste un arco

(orientato); indicando con $n' = |N'|$ e $n'' = |N''|$ le cardinalità dei due sottoinsiemi, il numero di archi è $m = n'n''$ se il grafo è non orientato, mentre è $m = 2n'n''$ nel cao sia orientato.

Esercizio 1.3
Dimostrare le affermazioni fatte sul numero di archi dei grafi completi e dei grafi bipartiti completi (orientati e non).

1.1.4 Alberi

Un grafo connesso e privo di cicli è detto un *albero*. Un albero $T = (N, A)$, con $|N| = n$, è tale che $|A| = m = n-1$. Sono equivalenti alla precedente le seguenti definizioni: un albero è un grafo connesso con n-1 archi; un albero è un grafo privo di cicli con n-1 archi.

Esercizio 1.4
Dimostrare che per un albero vale la relazione $m = n$-1.

Esercizio 1.5
Dimostrare l'equivalenza delle definizioni date.

Un albero *radicato* è un albero in cui sia stato selezionato un nodo, detto *radice* dell'albero; i nodi possono essere ordinati per "livelli" in modo ricorsivo: la radice è posta al livello 0 e i nodi adiacenti ad essa sono posti al livello 1; al livello k+1 appartengono i nodi che non appartengono al livello k-1 e che sono adiacenti ai nodi del livello k, si veda la figura 1.4.

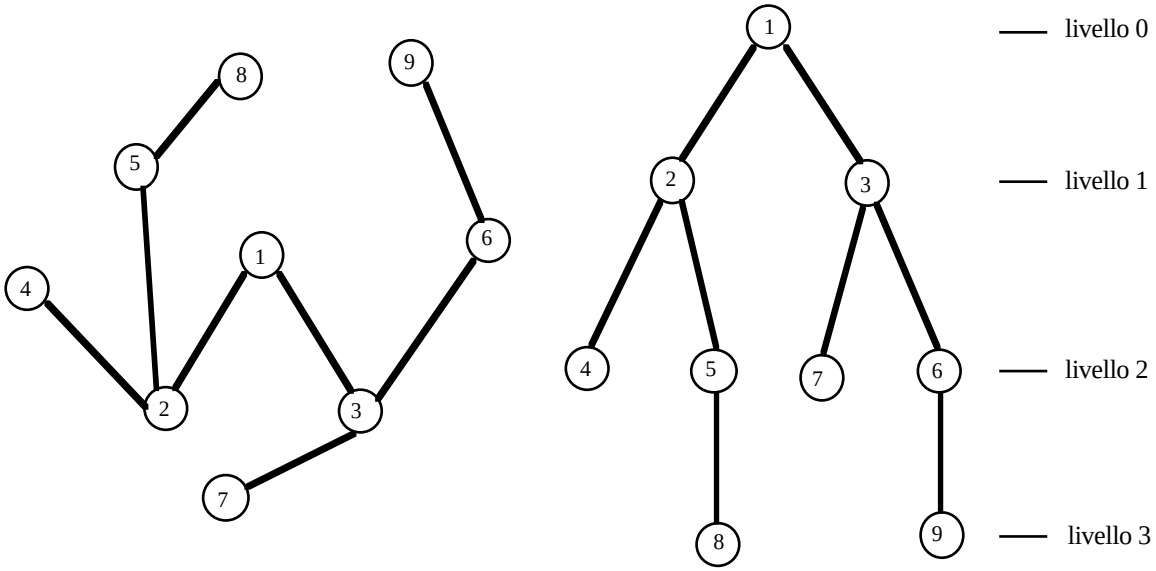


Figura 1.4 - Un albero (a) e lo stesso albero radicato nella radice 1 (b)

Ogni arco dell'albero radicato connette due nodi appartenenti a livelli adiacenti. Per ciascun arco (i,j) con i a livello k e j a livello k+1, i è detto *padre* di j e questi *figlio* di i.; la radice (ovviamente) non ha padre. Nodi aventi lo stesso padre sono detti *fratelli*; nell'ordinamento dell'albero radicato per livelli non si tiene conto dell'ordinamento tra fratelli, per alcuni problemi particolari alle volte si impone anche un ordinamento tra fratelli e in tal caso si parlerà di *primogenito, secondogenito, …, ultimogenito*.

Un nodo senza figli è detto una *foglia* dell'albero radicato.

Si noti che esiste un solo cammino tra la radice e qualsiasi nodo dell'albero; la lunghezza (in numero di archi) di tale cammino è uguale al livello cui appartiene il nodo destinazione del cammino.

Un nodo *i* che appartiene al cammino dalla radice ad un nodo *j* è detto un *antenato* di *j* e questi un *discendente* di *i*; si noti che ogni nodo è antenato e discendente di sé stesso. Un *sottoalbero T*(*j*) di un albero radicato è il grafo indotto dall'insieme dei nodi discendenti di *j*; in altri termini *T*(*j*) è formato da *j*, da tutti gli altri suoi discendenti e da tutti gli archi dell'albero tra questi nodi.

Esempio 1.1
In figura 1.4(b), il livello 2 è formato dai nodi 4, 5, 6 e 7; i figli del nodo 3 sono 7 e 6, mentre il padre di 3 è 1; i nodi 4 e 5 sono fratelli; l'insieme delle foglie è {4, 7, 8, 9}; l'insieme degli antenati di 3 è {1, 3} e quello dei discendenti è {3, 6, 7, 9}. Il sottoalbero *T*(2) è disegnato in figura 1.5

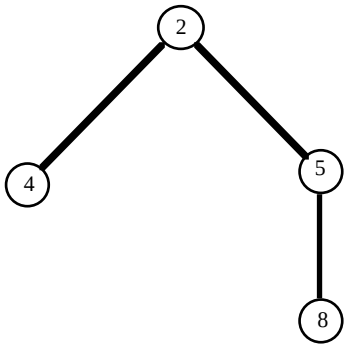


Figura 1.5 - Il sottoalbero *T*(2) dell'albero radicato in figura 1.4(b)

Esercizio 1.6
Determinare gli antenati e i discendenti del nodo 2 e i sottoalbero *T*(3) dell'albero di figura 1.4(b).

Un albero radicato, i cui archi sono orientati, è detto *orientato* se tutti i suoi archi sono orientati dal padre verso il figlio o dal figlio verso il padre.

Dato un grafo $G = (N, A)$, un suo grafo parziale $T = (N, A_T)$ che sia un albero è detto *albero di copertura* di *G* (o *spanning tree*); in figura 1.6 sono mostrati un grafo e un suo albero di copertura.

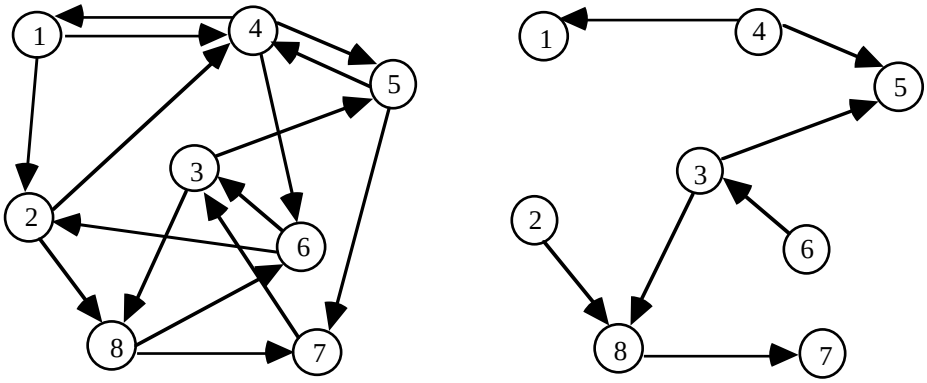


Figura 1.6 - Un grafo orientato e un suo albero di copertura

Un grafo le cui componenti connesse sono alberi è detto una *foresta*.

1.2 Rappresentazione di grafi e alberi

1.2.1 Matrici di adiacenza e di incidenza, liste di adiacenza

Dato un grafo $G = (N, A)$ la sua *matrice di adiacenza* $A = [a_{ij}]$ è una matrice quadrata di ordine n i cui valori sono:

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

Se il grafo è non orientato la matrice è simmetrica e contiene $2m$ elementi di valore 1; se è orientato il numero di elementi non nulli è m.

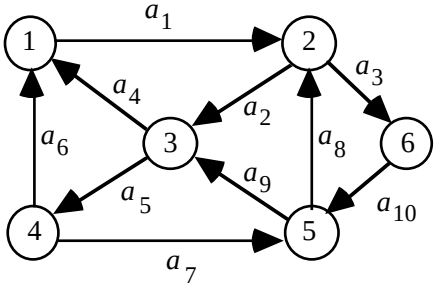
La *matrice di incidenza* $E = [e_{ik}]$ di un grafo è una matrice $n \times m$ (le righe corrispondono ai nodi e le colonne agli archi) il cui generico elemento e_{ik}, se G è non orientato, assume i seguenti valori:

$$e_{ik} = \begin{cases} 1, & \text{se } i \text{ è un nodo estremo dell'arco } a_k, \\ 0, & \text{altrimenti;} \end{cases}$$

se invece G è orientato, esso assume i seguenti valori:

$$e_{ik} = \begin{cases} -1, & \text{se } i \text{ è la coda dell'arco } a_k, \\ 1, & \text{se } i \text{ è la testa dell'arco } a_k, \\ 0, & \text{altrimenti.} \end{cases}$$

Esempio 1.2: In figura 1.7, un grafo orientato e la sua matrice di incidenza



$E =$

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
1	-1	0	0	1	0	1	0	0	0	0
2	1	-1	-1	0	0	0	0	1	0	0
3	0	1	0	-1	-1	0	0	0	1	0
4	0	0	0	0	1	-1	-1	0	0	0
5	0	0	0	0	0	0	1	-1	-1	1
6	0	0	1	0	0	0	0	0	0	-1

Figura 1.7 - Un grafo orientato e la sua matrice di incidenza

La *lista di adiacenza* per stelle uscenti di un grafo orientato è la sequenza $\{FN(1), FN(2),\ldots, FN(n)\}$ degli insiemi dei successori immediati dei nodi del grafo. Nell'esempio in figura 1.7, la lista di adiacenza è $\{\{a_1\}, \{a_2,a_3\}, \{a_4,a_5\}, \{a_6,a_7\}, \{a_8,a_9\}, \{a_{10}\}\}$. In modo analogo si definisce la lista di adiacenza per stelle entranti $\{BN(1), BN(2),\ldots, BN(n)\}$.

Un albero radicato di radice r è rappresentato mediante la *funzione predecessore* $p(.)$:

$$p(j) = \begin{cases} i, & \text{se } i \text{ è padre di } j, \\ nil, & \text{se } j \text{ è la radice } (j = r). \end{cases}$$

Se gli archi dell'albero radicato sono archi orientati, è possibile utilizzare la funzione predecessore per memorizzare l'orientamento di ciascun arco che connette un nodo con il padre:

$$p(j) = \begin{cases} i\ , & \text{se } i \text{ è padre di } j \text{ e l'arco tra essi è } (i,j), \\ -i\ , & \text{se } i \text{ è padre di } j \text{ e l'arco tra essi è } (j,i), \\ nil, & \text{se } j \text{ è la radice } (j = r). \end{cases}$$

1.2.2 Rappresentazione in memoria di grafi e alberi

Un grafo *G* può essere memorizzato mediante le *liste di adiacenza* per stelle uscenti e/o per stelle entranti. Le liste di adiacenza sono realizzate mediante due liste di record *Node_Set* e *Arc_Set*. In figura 1.8 sono illustrati un grafo e la sua memorizzazione.

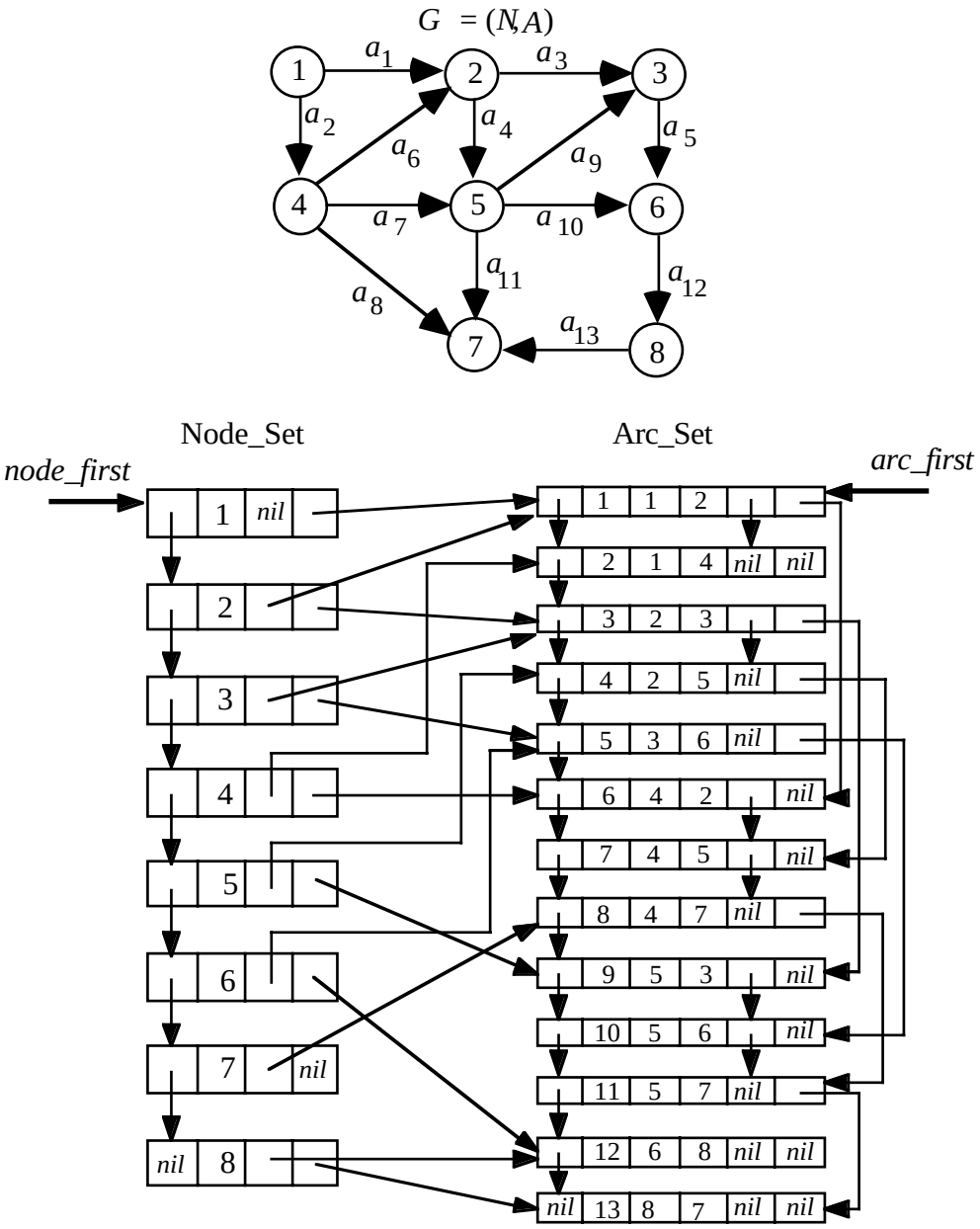


Figura 1.8 - la struttura per liste di adiacenza

Node_Set è formata da *n* records, ciascuno dei quali è composto da quattro campi informativi. Il record relativo al nodo *i* è strutturato nel modo seguente:

$\{succ_i, nome_i, primo\_BS_i, primo\_FS_i\}$,

dove:

$succ_i$ punta al record di Node_Set relativo al nodo seguente (*nil* se è l'ultimo nodo);

$nome_i$ contiene il nome del nodo (in generale l'indice i);

$primo\_BS_i$ e $primo\_FS_i$ puntano ai record di Arc_Set relativi al primo arco rispettivamente della stella entrante e della stella uscente (*nil* se rispettivamente $BS(i) = \emptyset$ e $FS(i) = \emptyset$).

Arc_Set è formata da m records, ciascuno dei quali è composto da sei campi informativi; il record relativo all'arco $a_k = (i,j)$ è strutturato nel modo seguente:

$$\{succ_k, nome_k, coda_k, testa_k, succ\_FS_k, succ\_BS_k\},$$

dove:

$succ_k$ punta al record di Arc_Set relativo all'arco seguente (*nil* se è l'ultimo arco);

$nome_k$ contiene il nome dell'arco (in generale l'indice k),

$coda_k$ e $testa_k$ puntano ai record di Node_Set relativi rispettivamente al nodo coda i ed al nodo testa j o alternativamente contengono gli indici di tali nodi,

$succ\_FS_k$ e $succ\_BS_k$ puntano ai record di Arc_Set relativi all'arco seguente a_k rispettivamente nella stella uscente $FS(i)$ e nella stella entrante $BS(j)$ (*nil* se a_k è l'ultimo arco rispettivamente di $FS(i)$ e $BS(j)$).

Inoltre due puntatori, *node_first* e *arc_first*, individuano rispettivamente il primo record di Node_Set e di Arc_Set.

Tale struttura dati permette agevolmente sia la scansione dei nodi e degli archi, sia la scansione degli archi di una stessa stella uscente od entrante quando si conosca il nodo relativo. Essa permette anche inserimenti e rimozioni di nodi e/o di archi. La struttura dati può essere ulteriormente ampliata; ad esempio, se il grafo è pesato, a ciascun record di nodo e/o di arco viene aggiunto un puntatore ai pesi relativi al nodo e/o all'arco (o, se esiste un solo peso per nodo e/o per arco, il campo informativo contiene direttamente il peso). Un altro esempio di ampliamento della struttura consiste nell'introduzione di puntatori inversi dei puntatori sopra definiti; la presenza di tali puntatori consente la rimozione di un nodo o di un arco senza dover scorrere la lista, alla ricerca del record precedente, per effettuare l'aggiornamento dei puntatori.

Esercizio 1.7
Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, calcoli il grado entrante ed il grado uscente di ciscun nodo e valutarne la complessità.

Esercizio 1.8
Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, inserisca un nuovo nodo $n+1$ e valutarne la complessità.

Esercizio 1.9
Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, inserisca un nuovo arco $a_{m+1} = (i,j)$ e valutarne la complessità.

Esercizio 1.10
Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, rimuova il nodo i e tutti gli archi incidenti in esso e valutarne la complessità.

Esercizio 1.11
Risolvere il problema dell'esercizio precedente utilizzando una struttura ampliata con i puntatori inversi.

Spesso, nella risoluzione di problemi su grafi, le rimozioni di nodi ed archi sono *temporanee* in quanto un particolare sottoproblema da risolvere è relativo ad un dato sottografo del grafo originario e sottoproblemi successivi sono relativi ad altri sottografi; non si vuole pertanto *distruggere* la struttura di dati che descrive il grafo otiginario. In tal caso, la rimozione fisica dei record relativi a nodi ed archi risulta altamente inefficiente, così come inefficiente è la replica della struttura per liste di adiacenza, una per ciascuna fase di modifica del grafo originario. Risulta più conveniente affiancare, in ciascun record, ai puntatori "statici" che definiscono il grafo originario, nuovi puntatori *dinamici* che definiscono il sottografo corrente, ed operare gli opportuni aggiornamenti su di essi.

La struttura per liste di adiacenza può essere anche semplificata sotto alcune ipotesi, in generale facilmente verificate, quali: i nodi hanno gli indici 1, 2, …, *n*, gli archi possono essere arbitrariamente ordinati, non si prevedono aggiunte di nuovi nodi ed archi, le eventuali rimozioni temporanee di nodi ed archi possono essere facilmente implementate mediante l'inserimento di puntatori dinamici. In tal caso le liste a puntatori Node_Set ed Arc_Set possono essere agevolmente realizzate mediante vettori di records (*array_pointers*) facendo corrispondere l'indice del nodo o dell'arco con l'indice della componente del vettore contenente le informazioni relative al nodo o all'arco.

Per realizzare la lista di adiacenza per stelle uscenti, con il minimo spazio-memoria, è sufficiente sostituire a Node_Set il vettore *Pntr_FS[.]*, ad *n*+1 componenti, ed a Arc_Set il vettore *Head_Arc[.]*, ad *m* componenti. Gli archi vengono ordinati per stelle uscenti (cioè gli archi della stella uscente di *i*+1 seguono gli archi della stella uscente di *i*). Pntr_FS[*i*] è l'indice della componente di Head_Arc[.] contenente il nodo testa del primo arco di *FS*(*i*); se *FS*(*i*) = Ø, allora si pone Pntr_FS[*i*] = Pntr_FS[*i*+1]. Ovviamente Pntr_FS[1] = 1; il puntatore relativo al nodo fittizio *n*+1 punta all'*m*+1-esimo arco (arco fittizio). In figura 1.9 è riportata la struttura relativa a Pntr_FS[.] e Head_Arc[.] relativa al grafo di figura 1.8. L'occupazione di memoria di questa lista di adiacenza è *m*+*n*+1.

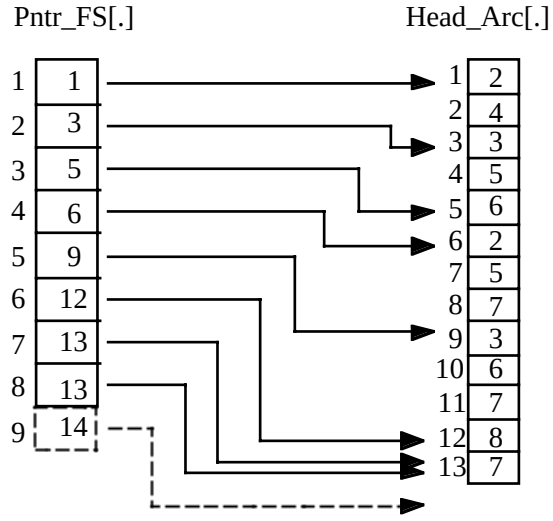


Figura 1.9 - Una struttura semplificata per stelle uscenti

Analogamente si può realizzare la lista di adiacenza per stelle entranti mediante due vettori Pntr_BS[.], ad *n*+1 componenti, e Tail_Arc[.], ad *m* componenti.

Esercizio 1.12
Costruire la lista di adiacenza (per stelle entranti) del grafo in figura 1.8.

1.2.3 Algoritmi elementari su grafi

In questo paragrafo verranno introdotti alcuni algoritmi che consentono di risolvere problemi che ricorrono sovente nella teoria dei flussi su grafi e che, di conseguenza, sono considerati alla stregua di operatori elementari della teoria.

Visita di un grafo

Sia $G = (N, A)$, con $|N| = n$ e $|A| = m$, un grafo orientato; ricordiamo che $FS(i) = \{(i,j) \in A\}$ e $BS(i) = \{(j,i) \in A\}$ indicano rispettivamente la stella uscente e la stella entrante del nodo i. Ad ogni arco (i,j) del grafo siano associate due etichette $f^+(i,j), f^-(i,j) \in \{true, false\}$ che indicano una condizione di utilizzabilità dell'arco stesso da i verso j e da j verso i rispettivamente. Il primo problema che prenderemo in esame consiste nell'esplorare il grafo G, partendo da un nodo r, nel seguito indicato come *radice*, potendo percorrere il generico arco (i,j) da i verso j solo se $f^+(i,j) = true$ e (all'indietro) da j verso i solo se $f^-(i,j) = true$ (si noti che possono esistere due archi contrapposti (i,j) e (j,i); associate ad essi vi sono le due coppie di etichette $f^+(i,j)$, $f^-(i,j)$ e $f^+(j,i)$, $f^-(j,i)$).

L'algoritmo riceve in input G, r, f^+, f^- e restituisce in output l'albero non orientato $T_r = (N_r, A_r)$ contenente gli archi utilizzati per raggiungere la prima volta i nodi connessi a T. Tale albero è chiamato *albero della visita*. Esso è rappresentato mediante la funzione predecessore $p(.)$:

$$p(j) = \begin{cases} 0, & \text{se } j \text{ non è raggiunto nella visita } (j \notin N_r), \\ i, & \text{se } j \text{ è raggiunto la prima volta attraverso } (i,j), \\ -i, & \text{se } j \text{ è raggiunto la prima volta attraverso } (j,i), \\ nil, & \text{se } j \text{ è la radice } (j = r). \end{cases}$$

Al termine della visita la funzione $p(.)$ definisce l'albero della visita $T_r = (N_r, A_r)$:

$$N_r = \{i \in N: p(i) \neq 0\}, \quad A_r = \{(p(j),j): p(j) > 0\} \cup \{(j,-p(j)): p(j) < 0\}.$$

La procedura di visita (vedi figura 1.10) si avvale dell'*insieme dei nodi candidati Q*, cioè dell'insieme dei nodi raggiunti nell'esplorazione ma ancora non utilizzati per proseguirla. La funzione predecessore è realizzata mediante il vettore $P[.]$, in cui la componente i-esima contiene il valore $p(i)$, per ogni i.

```
Procedure Visita (G,r,f+,f-,P):
    begin
      for i := 1 to n do P[i] :=0;  {inizializzazione}
      P[r] := nil; Q := {r};
      repeat
        select i from Q; Q := Q \ {i};                {selezione e rimozione di un nodo da Q}
        for each (i,j) ∈ FS(i) such that f+(i,j) do
          if P[j] = 0 then
            begin P[j] := i; Q := Q ∪ {j} end;
        for each (j,i) ∈ BS(i) such that f-(j,i) do
          if P[j] = 0 then
            begin P[j] := -i; Q := Q ∪ {j} end
      until Q = ø
    end.
```

Figura 1.10 - La procedura di visita di un grafo orientato

Analisi della complessità

Ogni nodo *i* del grafo viene inserito in *Q* solamente la prima volta che viene raggiunto. Pertanto non si avranno più di *n* inserzioni e rimozioni di nodi da *Q* e ogni arco (*i*,*j*) verrà esaminato al più due volte, una come arco uscente da *i* e l'altra come arco entrante in *j*. Quindi il ciclo **repeat…until** verrà ripetuto al più *n* volte e ciascuna operazione interna ad esso verrà ripetuta al più 2*m* volte. Supponendo che le operazioni di gestione di *Q* abbiano complessità *O*(1), la complessità di *Visita* è *O*(*m*).

Osserviamo che la funzione predecessore può essere convenientemente inserita nella struttura dati del paragrafo 1.2.2, inserendo un ulteriore campo in ciascun record di Node_Set. Infatti, per il record corrispondente al nodo *i*, è sufficiente aggiungere un campo contenente il puntatore al nodo *p*(*i*), un campo (booleano) per l'orientamento dell'arco (*i* ,*p*(*i*)) o (*p*(*i*),*i*) ed eventualmente un campo contenente il puntatore a tale arco in Arc_Set. Come vedremo il riferimento diretto all'arco (e quindi ai suoi relativi attributi quali ad es. capacità, costo, etc.) sarà utile nei successivi paragrafi.

Esempio 1.3

Applicare la procedura **Visita**(G,r,f,P) al grafo in figura partendo dal nodo *r*=1, con la seguente condizione di utilizzabilità: $f^+(i,j) = true$, $f^-(i,j) = false$ per ogni $(i,j) \in A$. Le stelle uscenti sono ordinate per indice dei nodi adiacenti, *Q* è implementato mediante una *fila* (*queue*).

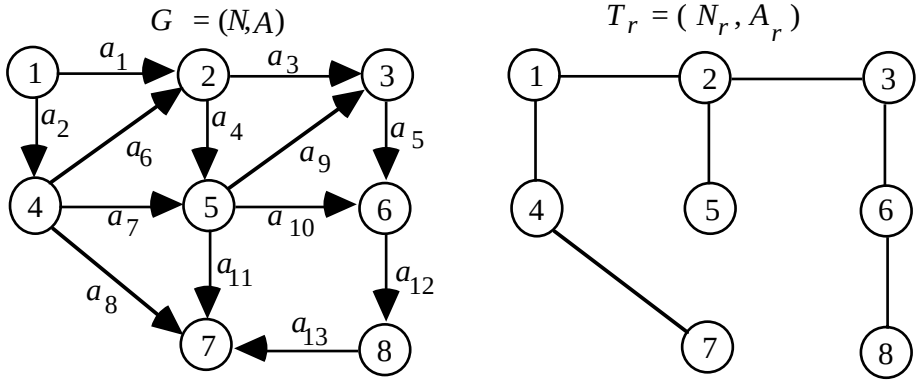


Figura 1.11 - Un esempio di applicazione della procedura *Visita*

Ordine di inserzione in (e rimozione da) *Q*: 1, 2, 4, 3, 5, 7, 6, 8.

La rimozione di 1 causa l'inserimento di 2 e 4 mediante gli archi (1,2) e (1,4); la rimozione di 2 causa l'inserimento di 3 e 5 mediante gli archi (2,3) e (2,5); la rimozione di 4 causa l'inserimento di 7 mediante l'arco (4,7), infatti (4,2) e (4,5) connettono nodi marcati, etc.

La funzione predecessore è definita dal vettore *P* = [*nil*, 1, 2, 1, 2, 3, 4, 6].

Esercizio 1.13

Realizzare la procedura *Visita* per grafi memorizzati mediante liste di adiacenza.

Esercizio 1.14

Fornire un algoritmo di visita, di complessità *O*(*m*), per verificare se un dato grafo non orientato, eventualmente non connesso, sia aciclico. Adattare l'algoritmo al caso di grafi orientati.

Esercizio 1.15

Fornire un algoritmo di visita, di complessità *O*(*m*), per verificare se un dato grafo non orientato, eventualmente non connesso, sia bipartito.

Esercizio 1.16

Fornire un algoritmo di visita, di complessità *O*(*m*), per verificare se un dato grafo, orientato e connesso, sia fortemente connesso (suggerimento: è sufficiente verificare che un arbitrario nodo *r* del grafo è connesso a tutti gli altri nodi e questi sono connessi ad *r* madiante cammini orientati).

Visite di un albero

Sia $T_B = (N, A_B)$ un albero di copertura di G di radice r e sia $p(.)$ la relativa funzione predecessore; essa rende naturale la visita da un nodo verso i propri antenati sino alla radice. Più complesso risulta visitare, mediante $p(.)$, i figli di un nodo o il suo sottoalbero. Al fine di effettuare visite anche verso i discendenti vengono introdotte altre funzioni che descrivono l'albero stesso.

Si dice *visita anticipata* di T_B una visita dei nodi secondo la seguente regola: "un nodo i viene visitato solo se tutti i nodi appartenenti all'unico cammino in T_B tra r e i sono stati visitati". Pertanto la visita inizia dalla radice dell'albero e termina in una sua foglia. Osserviamo che la visita anticipata visita anche gli archi di T. Infatti, quando viene visitato un nodo $i \neq r$, viene anche implicitamente visitato l'arco $(i,-p(i))$ (o $(p(i),i)$); quindi la visita anticipata induce un ordinamento sui nodi e sugli archi di T_B.

La funzione $va(.)$ associa ad ogni nodo i il nodo che verrà visitato dopo i attraverso una visita anticipata di T_B; inoltre, $va(.)$ associa all'ultimo nodo visitato il primo della visita. In figura 1.12 viene fornito un esempio di visita anticipata; essa permette di visitare consecutivamente i nodi di ciascun sottoalbero.

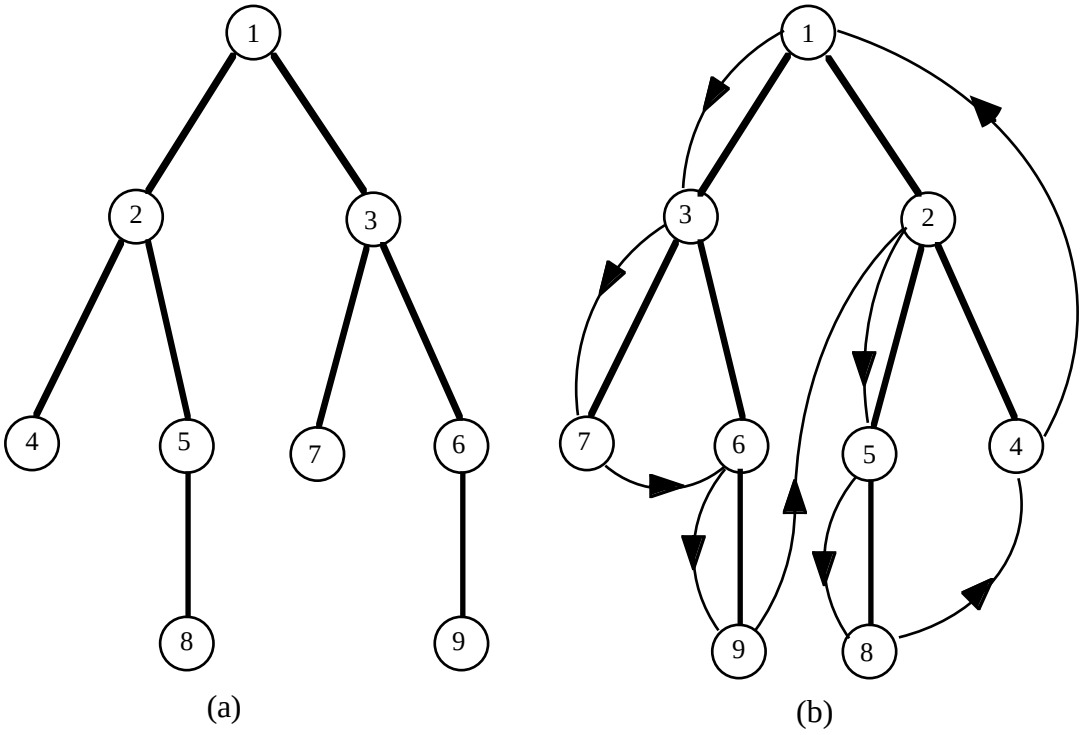


Figura 1.12 - La funzione $va(.)$ della visita anticipata

Definiamo *visita posticipata* di T_B una visita dei nodi secondo la seguente regola: "un nodo i viene visitato solo se tutti i suoi nodi figli sono stati visitati". Una funzione di visita posticipata è $vp:N \rightarrow N$, dove $vp(i)$ è il nodo visitato immediatamente dopo i (se i è il primo nodo visitato si pone $vp(r) = i$). Una particolare visita posticipata vp è data dalla funzione inversa di va: $va^{-1}(j) = i \Leftrightarrow va(i) = j$. Con riferimento all'esempio in figura 1.12 si ha:

N	1	2	3	4	5	6	7	8	9
va	3	5	7	1	8	9	6	4	2
va^{-1}	4	9	1	8	2	7	3	5	6

Livello dei nodi di un albero

La funzione livello *lev*(.) dei nodi dell'albero T_B associa ad ogni nodo *i* il suo livello, cioè il numero di archi dell'unico cammino in T_B tra la radice *r* e *i*; *lev*(.) può essere facilmente calcolata, date le funzioni predecessore *p*(.) e visita anticipata *va*(.), mediante la seguente procedura *Livello*. Indichiamo con *L*[.] il vettore che realizza *lev*(.).

```
Procedure Livello(r, P,Va, L):
  begin
    L[r]:=0; u:=Va[r];                 {il livello della radice è zero}
      while u ≠ r do
        begin
          L[u] := L[|P[u]|] + 1;       {il livello di u è uguale al livello del predecessore + 1}
          u := Va[u]                   {si prende il nodo successivo nella visita anticipata}
        end
  end.
```

Figura 1.13 - La procedura **Livello**

Esercizio 1.17
Determinare *lev*(.) per l'albero in figura 1.12(a).

1.3 Reti di flusso: costi, domande, capacità, flussi

1.3.1 Rete, bilancio, costi

G è detto un *grafo pesato* se ai nodi e/o agli archi vengono associati uno o più valori reali, detti *pesi*. Un grafo pesato è detto una *rete* se i pesi associati ai nodi rappresentano il loro *bilancio*, cioè la quantità offerta o domandata di un determinato bene, e se i pesi associati agli archi rappresentano i *costi* e le *capacità* degli archi. Più precisamente:

- ad ogni nodo $i \in N$ viene associato il valore reale b_i, detto *bilancio* del nodo; esso rappresenta la quantità di un determinato bene che deve essere fatto arrivare sino ad *i* (in tal caso b_i è un valore positivo, detto *domanda*), o che viene fatto partire da *i* ($b_i < 0$, detto *offerta*). Se:

$$b_i \begin{cases} < 0, & i \text{ è detto } \textit{nodo origine, sorgente} \text{ o } \textit{di input}; \\ > 0, & i \text{ è detto } \textit{nodo destinazione, pozzo} \text{ o } \textit{di output}; \\ = 0, & i \text{ è detto } \textit{nodo di trasferimento}; \end{cases}$$

- ad ogni arco $a_k = (i,j)$ vengono associati il *costo* c_k (c_{ij}) che ogni unità del bene causa nell'attraversare l'arco, e le *capacità inferiore* l_k (l_{ij}) e *superiore* u_k (u_{ij}), cioè rispettivamente il minimo ed il massimo numero di unità del bene che possono attraversare l'arco.

1.3.2 Flussi, vincoli di conservazione e di capacità

Un vettore $x \in \boldsymbol{R}^m$ è un *flusso* sul grafo *G* se verifica i seguenti *vincoli di conservazione del flusso*:

$$\sum_{(j,i)\in BS(i)} x_{ji} \quad - \quad \sum_{(i,j)\in FS(i)} x_{ij} = b_i, \qquad \text{per ogni } i \in N.$$

Il valore x_k, o x_{ij}, è detto *flusso* dell'arco $a_k = (i,j)$. I vincoli di conservazione del flusso possono essere rappresentati in modo compatto utilizzando la matrice di incidenza del grafo *E*:

$$Ex = b,$$

dove $b = [b_i]$ è il vettore dei bilanci. Un flusso è detto *ammissibile* se sono verificati i seguenti *vincoli di capacità* sugli archi:

$$l_{ij} \le x_{ij} \le u_{ij}, \qquad \text{per ogni arco } (i,j) \in A;$$

che possono essere scritti sotto forma vettoriale, in cui $l = [l_{ij}]$ e $u = [u_{ij}]$:

$$l \le x \le u.$$

Un vettore $x \in \boldsymbol{R}^m$ è detto uno *pseudoflusso* se verifica i *vincoli di capacità*; ovviamente uno pseudoflusso che sia un flusso è un flusso ammissibile. Senza perdere di generalità, come sarà mostrato nel seguito, si può sempre supporre che le capacità inferiori degli archi siano nulle:

$$l_{ij} = 0, \qquad \text{per ogni arco } (i,j) \in A.$$