

# **Programmazione Procedurale e Logica**

## Relazione sul progetto della sessione invernale 2014-2015

Matteo Incicco [MAT. 261716]

**Docente: Prof. Marco Bernardo**

## Specifica del problema

Scrivere una libreria ANSI C che gestisce le relazioni binarie esportando le seguenti funzioni. La prima funzione C restituisce una relazione binaria acquisita da tastiera. La seconda funzione C ha come parametro di ingresso una relazione binaria e la stampa a video. La terza funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una relazione d'ordine parziale, stampando a video quale proprietà non vale nel caso la relazione non sia tale. La quarta funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una relazione d'ordine totale, stampando a video quale proprietà non vale nel caso la relazione non sia tale. La quinta funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una relazione d'equivalenza, stampando a video quale proprietà non vale nel caso la relazione non sia tale. La sesta funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una funzione matematica; se non lo è, allora si stamperà a video quale elemento violi la proprietà, altrimenti si stamperà a video un messaggio che indica se la funzione è iniettiva, suriettiva o biiettiva.

## Analisi del problema

L'input del problema è costituito da una relazione binaria inserita dall'utente.

L'output consiste nella comunicazione della relazione binaria acquisita e stabilire, su richiesta, se essa è: una relazione d'ordine parziale, una relazione d'ordine totale, una relazione d'equivalenza o una funzione matematica. Comunicare inoltre quali proprietà non valgono se la relazione non è del tipo richiesto.

La relazione che intercorre fra input ed output è data da dei corollari della logica matematica:

*Data una relazione  $R \subseteq A \times A$ , diciamo che essa è:*

- *Riflessiva sse  $(a, a) \in R$  per ogni  $a \in \text{campo}(R)$ .*
- *Simmetrica sse  $(a1, a2) \in R$  implica  $(a2, a1) \in R$  per ogni  $a1, a2 \in \text{campo}(R)$ .*
- *Antisimmetrica sse  $(a1, a2) \in R$  implica  $(a2, a1) \notin R$  per ogni  $a1, a2 \notin \text{campo}(R)$ ,  $a1 \neq a2$ .*
- *Transitiva sse  $(a1, a2) \in R$  e  $(a2, a3) \in R$  implicano  $(a1, a3) \in R$  per ogni  $a1, a2, a3 \in \text{campo}(R)$ .*

*Una relazione è una relazione d'ordine parziale sse è riflessiva, antisimmetrica e transitiva.*

*Una relazione è una relazione d'ordine totale sse soddisfa anche la seguente proprietà chiamata dicotomia:  $(a1, a2) \in R$  o  $(a2, a1) \in R$  per ogni  $a1, a2 \in \text{campo}(R)$ .*

*Una relazione è una relazione d'equivalenza sse è riflessiva, simmetrica e transitiva.*

*Una relazione è una relazione funzionale tra  $A$  e  $B$  sse per ogni  $a \in \text{dom}(R)$  esiste esattamente un  $b \in B$  tale che  $(a, b) \in R$ .*

*Data una funzione  $f: A \rightarrow B$ , diciamo che essa è:*

- *Iniettiva sse per ogni  $b \in B$  esiste al più un  $a \in \text{dom}(f)$  tale che  $f(a) = b$ .*
- *Suriettiva sse per ogni  $b \in B$  esiste almeno un  $a \in \text{dom}(f)$  tale che  $f(a) = b$ .*
- *Biiettiva sse per ogni  $b \in B$  esiste esattamente un  $a \in \text{dom}(f)$  tale che  $f(a) = b$ .*

## Progettazione dell'algoritmo

Occorre fare in modo che l'utente inserisca una relazione binaria corretta completa di punteggiatura. Gli elementi delle coppie devono essere una combinazione di numeri interi positivi e/o lettere minuscole dell'alfabeto.

La relazione binaria inserita deve essere sottoposta a controlli per verificare la sua correttezza dal punto di vista sintattico.

La relazione verrà poi inserita in una struttura dati che conterrà gli elementi delle coppie privi di punteggiatura. La struttura sarà necessaria per eseguire più liberamente le operazioni e le verifiche sulla relazione.

La relazione verrà poi ricomunicata all'utente con la punteggiatura.

Il programma poi dovrà verificare se la relazione soddisfa le proprietà della logica matematica: riflessività, simmetria, antisimmetria, transitività. Di conseguenza sarà anche in grado di dimostrare se essa è una relazione d'ordine parziale, totale o una funzione matematica.

Se la relazione è una funzione matematica si dovrà comunicare la tipologia della funzione: iniettiva, suriettiva o biiettiva.

## Funzioni del programma

Come da richiesta, le operazioni/verifiche saranno richiamate dall'utente tramite l'utilizzo di funzioni.

I passi dell'algoritmo e, dunque, le funzioni, sono i seguenti:

- Acquisizione e validazione della relazione binaria.
- Stampa della relazione binaria con punteggiatura.
- Verifica se la relazione è d'ordine parziale. Comunicare eventuali proprietà non soddisfatte.
- Verifica se la relazione è d'ordine totale. Comunicare eventuali proprietà non soddisfatte.
- Verifica se la relazione è d'equivalenza. Comunicare eventuali proprietà non soddisfatte.
- Verifica se la relazione è una funzione matematica. Comunicare l'elemento che nega la verifica. Se è una funzione matematica comunicare, invece, se essa è iniettiva, suriettiva o biiettiva.

Di seguito verranno illustrati dettagliatamente i passaggi che comportano la risoluzione del problema.

### Acquisizione della relazione binaria

Si è deciso di acquisire la relazione binaria per intero, ovvero completa di punteggiatura, parentesi graffe e tonde e senza spazi.

Esempio

$\{(ab,a),(a,a),(a,ab)\}$

Così facendo l'utente ha la possibilità di inserire la relazione per intero, proprio come si trova scritta sui libri di logica matematica. Verrà, quindi, lasciato all'algoritmo il compito di dividere e organizzare la relazione in una apposita struttura per semplificare le operazioni/verifiche su di essa. I caratteri concessi per gli elementi sono: l'alfabeto (comprensivo delle lettere inglesi) e i numeri interi positivi.

Esempio

Accettati	Non accettati
ab3	ab-3
a	A
1	1,3

Le illustrazioni sottostanti la colonna "Non accettati" non possono essere elementi della nostra relazione in quanto, contengono lettere maiuscole, numeri negativi e numeri con la virgola.

Concludendo, la relazione verrà inizialmente acquisita come stringa di caratteri.

Per una corretta acquisizione e allocazione della memoria verrà prima richiesto all'utente la lunghezza della relazione considerando anche la punteggiatura.

Esempio

La relazione  $\{(ad2,2b),(ad2,ad2)\}$  è lunga 20 caratteri in quanto devono essere contate anche le parentesi graffe, le parentesi tonde e le virgole.

{	(	a	d	2	,	2	b	)	,	(	a	d	2	,	a	d	2	)	}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

## Validazione relazione binaria

L'utente è imprevedibile; di conseguenza è necessaria un'accurata validazione dell'input. Se l'input inserito non viene validato c'è il rischio che le varie funzioni non riescano ad operare sulla relazione e, di conseguenza, non svolgano il loro "compito".

Prima di arrivare alla validazione della relazione occorre controllare che il numero inserito dall'utente (ovvero la lunghezza dell'input) sia effettivamente un numero.

Per validare l'intero è necessario acquisire come stringa il numero inserito dall'utente e, successivamente, controllare la stringa carattere per carattere.

Ogni carattere dovrà essere quindi un numero da 0 a 9.

Esempio

2	6
---	---

Verrà controllato da sinistra verso destra. Il primo carattere può essere un numero che va da 1 a 9. Quindi si controllerà in primo luogo che sia diverso da 0.

Se supera la precedente verifica verrà confrontato con i seguenti numeri:

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

e via dicendo verso destra.

Se tutti i caratteri passano il controllo la stringa verrà convertita in intero e si proseguirà con l'acquisizione e la validazione della relazione.

Per verificare la correttezza della relazione occorre:

1 Validare la punteggiatura:

1.1 Controllo delle parentesi graffe.

1.2 Controllo delle parentesi tonde, delle virgole e conteggio delle coppie.

1.3 Conteggio delle virgole e delle tonde chiuse. Come "prova del nove", il numero delle virgole e delle tonde verrà confrontato con il numero delle coppie.

2 Validare gli elementi.

Le graffe debbono trovarsi alle estremità della stringa.

Di conseguenza basterà controllare il primo e l'ultimo carattere:

Esempio:

{	(	a	,	b	)	}
↑						↑

Se quest'ultima verifica è andata a buon fine, si andrà a verificare l'esistenza delle parentesi tonde e delle virgole.

Si controllerà subito che nella seconda posizione sia presente una parentesi tonda aperta:

Esempio:

{	(	a	,	b	)	}
	↑					

Poi si andrà alla ricerca della virgola partendo dalla seconda posizione dopo la tonda:

Esempio:

{	(	a	,	b	)	}
			↑			

L'ultimo passaggio sarà quello di andare a trovare la tonda chiusa (partendo sempre dalla posizione precedente più 2 posizioni) e di conseguenza aumentare il contatore delle coppie:

Esempio:

{	(	a	,	b	)	}
					↑	

Se la stringa continua si andrà a verificare la presenza di una virgola dopo la tonda chiusa e, di conseguenza, rinizierà il controllo ripartendo dalla parentesi tonda aperta.

Ora verrà scandita di nuovo la relazione contando le virgole e le parentesi tonde chiuse presenti. Il numero delle virgole e delle tonde verrà confrontato con la quantità delle coppie trovate. Il numero delle virgole dovrà essere il doppio della quantità delle coppie meno 1; mentre il numero delle tonde chiuse trovate dovrà essere uguale al numero di coppie presenti nella relazione.

Esempio:

Se le coppie presenti in una relazione sono 3 le virgole presenti dovranno essere 5 e, di conseguenza, le tonde chiuse 3.

Grazie a questo controllo è possibile fare affidamento sulla punteggiatura per "indicizzare" la posizione degli elementi.

Prima del controllo degli elementi, verranno salvati gli indici che delimitano la posizione degli elementi all'interno della relazione inserita dall'utente.

Per fare ciò si utilizzano due array che lavorano in parallelo. Uno che segna gli elementi dell'insieme A e l'altro gli elementi dell'insieme B.

Esempio:

{	(	a	d	2	,	2	b	)	,	(	a	d	2	,	a	d	2	)	}
					↑									↑					

L'array A conterrà la posizione delle virgole che delimitano due elementi (occorrenze dispari).

{	(	a	d	2	,	2	b	)	,	(	a	d	2	,	a	d	2	)	}
								↑										↑	

L'array B conterrà la posizione delle parentesi tonde chiuse.

A questo punto occorre verificare che gli elementi siano sintatticamente corretti.

Per fare ciò si scandirà di nuovo la relazione puntando stavolta solo agli elementi, grazie all'utilizzo dei due array precedentemente creati.

{	(	a	d	2	,	2	b	)	,	(	a	d	2	,	a	d	2	)	}
		↑	↑	↑		↑	↑				↑	↑	↑		↑	↑	↑		

Gli elementi verranno, dunque, controllati carattere per carattere (come avvenuto per la validazione dell'intero).

Ora la relazione è completamente validata a livello sintattico; si potrà perciò proseguire con l'inserimento di quest'ultima all'interno della struttura.

### Creazione e inserimento della relazione nella struttura

Appena prima dell'inserimento sarà necessario allocare la struttura con il numero delle coppie. Poi, utilizzando sempre i due array "a" e "b" precedentemente creati, si copieranno tutti gli elementi (carattere per carattere) nella struttura.

Esempio:

La relazione:

{	(	a	d	2	,	2	b	)	,	(	a	d	2	,	a	d	2	)	}
		↑	↑	↑		↑	↑				↑	↑	↑		↑	↑	↑		

Verrà sistemata nella struttura nel seguente modo:

STRUTTURA	elem. A	elem. B
Prima coppia	ad2	2b
Seconda coppia	ad2	ad2

Sarà così più semplice far riferimento ad un elemento e/o coppia della relazione e di conseguenza si avrà uno snellimento del codice nelle operazioni e nelle verifiche delle varie proprietà.

Una volta caricata la struttura sarà necessario effettuare l'ultimo controllo: verificare se sono state immesse delle coppie uguali.

Una scelta di progetto è quella di non far immettere all'utente coppie uguali in quanto possono far sfalsare le varie funzioni ricorsive.

Se sono presenti coppie uguali l'utente dovrà reinserire la struttura.

### Stampa della relazione binaria con punteggiatura

Una volta acquisita e validata la relazione sarà necessario, come richiesto, implementare una funzione che sia in grado di stamparla.

Ora la relazione è "spoglia", ovvero priva di punteggiatura, di conseguenza sarà necessario "ricostruirla" per mostrarla a video.

Le virgole e le parentesi tonde presenti variano in base al numero delle coppie, mentre le parentesi graffe sono fisse all'inizio e alla fine della relazione. Di conseguenza il flusso delle operazione appare come segue:

- Stampa della parentesi graffa aperta "{".



- Stampa delle coppie delimitandole con le parentesi tonde e separando i due elementi con la virgola.
- Se sono presenti due coppie o più sarà necessario inserire una virgola tra quest'ultime.
- Terminate le coppie sarà necessario chiudere la parentesi aperta all'inizio della relazione: "}".

N.B dopo ogni virgola verrà inserito uno spazio.

Esempio:

La struttura:

STRUTTURA	elem. A	elem. B
Prima coppia	ad2	2b
Seconda coppia	ad2	ad2

dovrà essere stampata a video come segue:

{	(	a	d	2	,	2	b	)	,	(	a	d	2	,	a	d	2	)	}
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

### Verifica se la relazione è d'ordine parziale\totale

Per la verifica della relazione d'ordine parziale è necessario soffermarci sulle proprietà che essa richiede.

Una relazione è di tipo d'ordine parziale se sono soddisfatte le seguenti proprietà:

- Riflessività
- Antisimmetria
- Transitività

Di conseguenza sarà necessario implementare quest'ultime.

Una relazione è riflessiva "sse  $(a, a) \in R$  per ogni  $a \in \text{campo}(R)$ "; sarà quindi necessario verificare se ogni elemento presente nel campo(R) "riflette" su se stesso.

Esempio:

La relazione  $R = \{(a, b), (b, a)\}$  non è riflessiva.

Per esserere riflessiva deve contenere altre due coppie del tipo :  $(a, a)$  e  $(b, b)$ .

Quindi la relazione sopracitata è riflessiva se si presenta come segue:  $\{(a, b), (b, a), (a, a), (b, b)\}$  (senza considerare l'ordine delle coppie).

Soffermandoci sull'esempio possiamo da subito evincere le fasi dell'algoritmo che scansionerà la relazione:

- 1 Se nella coppia sono presenti due elementi uguali (es.  $(a, a)$ ) è già verificata la riflessività (per quest'ultima); e si potrà passare al prossimo controllo.
- 2 Se nella coppia sono presenti due elementi diversi (es.  $(a, b)$ ) allora:
  - 2.1 Verrà chiamata due volte una funzione che controllerà che sia presente (in tutta la relazione) una coppia con due elementi uguali all'elemento  $a$  (quindi che esista  $(a, a)$ ) e una coppia con due elementi uguali all'elemento  $b$  (quindi che esista  $(b, b)$ ).
- 3 Se le verifiche sovrastanti hanno dato un risultato positivo per tutte le coppie della relazione allora essa è riflessiva.

Una relazione è antisimmetrica "sse  $(a1, a2) \in R$  implica  $(a2, a1) \notin R$  per ogni  $a1, a2 \notin campo(R), a1 \neq a2$ ".

Esempio:

La relazione  $R = \{(a, b), (b, a), (a, a), (b, b)\}$  non è antisimmetrica.

Per far sì che la relazione sia antisimmetrica è necessario che (nel caso sovrastante) non esista la coppia  $(b, a)$ .

Quindi la relazione sopracitata è antisimmetrica se viene presentata come segue:  $\{(a, b), (a, a), (b, b)\}$ .

Soffermandoci sull'esempio possiamo da subito evincere le fasi dell'algoritmo che scansionerà la relazione:

- 1 Si cerca nella relazione una coppia composta da due elementi diversi (es.  $(a, b)$ ).
- 2 Se c'è allora si chiama una funzione (la stessa usata per la verifica della riflessività) che cerca se nella relazione è presente, quindi, una coppia inversa (es.  $(b, a)$ ).
  - 2.1 Se la coppia è presente allora vuol dire che la relazione non è antisimmetrica.
  - 2.2 Se non è presente si può procedere ai prossimi controlli.
- 3 Se non c'è nessuna coppia con elementi diversi allora la proprietà non è verificata.
- 4 Se invece sono state trovate coppie con elementi diversi ma nessuna ha verificato la simmetria allora la relazione si può definire antisimmetrica.

Una relazione è transitiva *sse*  $(a1, a2) \in R$  e  $(a2, a3) \in R$  implicano  $(a1, a3) \in R$  per ogni  $a1, a2, a3 \in \text{campo}(R)$ .

Esempio:

La relazione  $R = \{(a, b), (b, c)\}$  non è transitiva.

Per far sì che sia transitiva dovrà essere necessariamente presente una terza coppia:  $(a, c)$ . Quindi la relazione sovraccitata è transitiva se viene presentata come segue:  $\{(a, b), (b, c), (a, c)\}$ .

Soffermandoci sull'esempio possiamo da subito evincere le fasi dell'algoritmo che scansionerà la relazione:

- 1 Si cerca nella relazione una coppia composta da due elementi diversi (es.  $(a, b)$ ).
- 2 Se viene trovata, viene chiamata una funzione ausiliare (per snellire il codice) che si occuperà di trovare anche una coppia che, come primo elemento, ha l'elemento  $b$  della coppia prima esaminata:  $(b, c)$ .

2.1 Ora per verificare la transitività sarà necessario chiamare la funzione che cerca una coppia (quella usata per la riflessività e la simmetria). Questa funzione dovrà cercare una coppia con al primo posto: l'elemento  $a$  della prima coppia e, al secondo posto: l'elemento  $b$  della seconda coppia:  $(a, c)$ .

2.1.1 Se la trova, è verificata la transitività e l'algoritmo procede con i prossimi controlli alle prossime coppie della relazione.

2.1.2 Se non lo trova la relazione non è transitiva.

- 3 Ultimo controllo: se non sono presenti coppie uguali e non è mai stata verificata una coppia che implichi il controllo dell'esistenza di una terza coppia, allora la relazione non è transitiva.

Ora sarà necessario verificare se la relazione soddisfa anche la dicotomia:  $(a1, a2) \in R$  o  $(a2, a1) \in R$  per ogni  $a1, a2 \in \text{campo}(R)$ .

Esempio:

La relazione  $\{(a, b), (a, a), (b, b), (c, c), (b, c), (a, c)\}$  non è una relazione d'ordine parziale in quanto soddisfa anche la dicotomia.

Per essere una relazione d'ordine parziale è necessario inserire una coppia di elementi che non siano in "relazione" con tutti gli altri elementi del  $\text{campo}(R)$ .

Aggiungendo quindi la coppia  $(d, d)$  la relazione sarà d'ordine parziale.

Per verificare ciò, i passaggi da seguire sono i seguenti:

- 1 Si prende in considerazione ogni coppia della relazione: dalla prima all'ultima.
- 2 Si chiama una funzione ausiliare che scandirà la relazione in cerca di una coppia "(a, b)" o "(b, a)" per ogni elemento della relazione.
  - 2.1 Se è presente almeno una delle due coppie, allora, la dicotomia è verificata per la coppia presa in considerazione e si continuerà con i confronti.
  - 2.2 Se non è presente nessuna delle due coppie, la dicotomia non è verificata per la relazione.

Si userà un array di controllo per verificare le varie proprietà soddisfatte e non.

Così facendo sarà semplice sapere quali proprietà sono da stampare a video qualora la relazione non è del tipo richiesto.

Sappiamo che ogni funzione restituirà un intero positivo se la proprietà è soddisfatta, e "0" se non lo è.

Di conseguenza è facile costruire l'array.

Esempio:

1
1
1
0

Sta ad indicare che la relazione è (1) riflessiva, (1) antisimmetrica (nel caso della relazione d'ordine parziale e totale), (1) transitiva e (0) non dicotomica.

Quindi se è stata chiamata la funzione che verifica se la relazione è di tipo d'ordine parziale il risultato sarà positivo.

Se è stata invece chiamata la funzione che verifica se la relazione è di tipo totale allora si dovrà stampare che non è soddisfatta la proprietà chiamata dicotomia (0).

### Verifica se la relazione è d'equivalenza

Una relazione è d'equivalenza se è: riflessiva, simmetrica, transitiva.

Visto che la prima e l'ultima proprietà sono state precedentemente descritte, passeremo alla verifica della simmetria.

Una relazione è simmetrica *sse*  $(a1, a2) \in R$  *implica*  $(a2, a1) \in R$  *per ogni*  $a1, a2 \in \text{campo}(R)$ .

Esempio:

La relazione  $\{(a, b), (a, a), (b, b)\}$  non è simmetrica; per far sì che lo sia occorre inserire una coppia "(b, a)".

Ricapitolando, la relazione è simmetrica se viene presentata come segue:  $\{(a, b), (b, a), (a, a), (b, b)\}$ .

I passi dell'algoritmo per verificare la simmetria sono i seguenti:

- 1 Si cerca nella relazione una coppia composta da due elementi diversi (es. "(a, b)").
- 2 Se c'è si chiama una funzione (la stessa usata per la verifica della riflessività) che cerca se nella relazione è presente una coppia inversa (es. "(b, a)").
  - 2.1 Se la coppia è presente vuol dire che è simmetrica e si proseguono i controlli sul resto della relazione.
  - 2.2 Se non è presente la relazione non è simmetrica.

Per la relazione d'equivalenza l'array di controllo dovrà essere come segue:

1
1
1

Dove, in ordine: (1) riflessiva, (1) simmetrica, (1) transitiva.

### **Verifica se la relazione è una funzione matematica**

La relazione è una funzione da A a B, *sse per ogni  $a \in \text{dom}(R)$  esiste esattamente un  $b \in B$  tale che  $(a, b) \in R$ .*

Quindi una relazione del tipo:  $\{(a, b), (a, c)\}$  non è una funzione in quanto l'elemento "a" fa riferimento a due elementi dell'insieme B.

La relazione è una funzione se (nel caso dell'esempio sovrastante) viene presentata come segue:  $\{(a, b), (d, c)\}$ .

Di conseguenza, è intuitivo capire come verificare che la relazione sia una funzione.

È stata implementata una procedura che conta gli "accoppiamenti" di ogni coppia presente nella relazione:

- 1 Per ogni a che trova nelle coppie di tipo (a, b), verifica se ci sono più coppie (a, x).
  - 1.1 Se non sono presenti si continuerà con i controlli.
  - 1.2 Se sono presenti più coppie (a, x), la relazione non è sicuramente una funzione matematica. Verrà, come richiesto, stampato a video l'elemento che viola la verifica.

Una volta verificato che la relazione è una funzione matematica. Andremo a controllare a che tipologia essa appartiene.

Per esaminarla in modo corretto occorre conoscere l'intero insieme B.

Di conseguenza, sarà opportuno chiedere all'utente se l'insieme B è finito con gli elementi della relazione o ne sono presenti di altri.

In quest'ultimo caso verrà chiesto all'utente di inserire il numero di elementi contenuti nell'insieme che non sono presenti nella relazione; quindi, si procede con l'acquisizione e la validazione degli elementi dell'insieme.

Le regole grammaticali sono le stesse usate per gli elementi della relazione.

Una volta validato l'elemento (carattere per carattere, con la funzione utilizzata per la relazione), verrà inserito in una struttura apposita per contenerlo. Quest'ultima è molto simile a quella della relazione ma conterrà un solo elemento per dimensione.

Ora si hanno tutti i dati e si proseguirà con la verifica dell'iniettività.

Una funzione è iniettiva *sse per ogni  $b \in B$  esiste al più un  $a \in \text{dom}(f)$  tale che  $f(a) = b$ .*

Esempio:

La funzione  $\{(a, b), (d, c)\}$  è solo iniettiva se verranno inseriti altri elementi nell'insieme B.

Si può controllare da subito l'iniettività della funzione.

Verrà usata una funzione che conterà tutti gli accoppiamenti con ogni elemento b della relazione:

- 1 Se c'è più di un accoppiamento con un elemento b della relazione la funzione non è iniettiva.
- 2 Altrimenti la funzione verrà data per iniettiva.

La verifica della sola iniettività avverrà in seguito.

Ora passiamo al controllo della suriettività. Una funzione è suriettiva *sse per ogni  $b \in B$  esiste almeno un  $a \in \text{dom}(f)$  tale che  $f(a) = b$ .*

La funzione  $\{(a, b), (c, b)\}$  è suriettiva se non verranno inseriti altri elementi nell'insieme B.

La suriettività verrà quindi posta a priori. Sarà non soddisfatta solo se esistono altri elementi nell'insieme B.

Di conseguenza per ogni elemento b dell'insieme B:

- 1 Cerca se esistono coppie (x, b).
  - 1.1 Se non esistono, la funzione non è suriettiva perché il  $\text{cod}(f)$  non comprende tutto l'insieme B (di conseguenza verrà escluso anche il possibile errore dell'utente di reinserire un elemento già presente nella relazione).
  - 1.2 Se esistono e non ci sono altri elementi "scoperti" allora sarà suriettiva.

Una funzione è biiettiva *sse per ogni  $b \in B$  esiste esattamente un  $a \in \text{dom}(f)$  tale che  $f(a) = b$ .*

La funzione  $\{(a, b), (d, c)\}$  è biiettiva se non verranno inseriti altri elementi nell'insieme B, in quanto, è sia iniettiva che suriettiva.

Di conseguenza verrà comunicato all'utente il tipo di funzione.

## **Implementazione dell'algoritmo**

Nelle pagine successive è riportato il codice sorgente dell'implementazione dell'algoritmo in ANSI C comprensivo di commenti.

```
/* strutture definite per memorizza la relazione
   e l'insieme B */

typedef struct
{
    char *elem_a;           /* i.: elem. ins. A della rel. */
    char *elem_b;           /* i.: elem. ins. A della rel. */
} rel_t;

typedef struct
{
    char *elem;              /* i.: elemento dell'insieme */
} insieme_t;

/*****
/* dichiarazione delle funzioni esportabili */
*****/

int acquisizione_relazione(rel_t **pp);
void stampa_relazione(rel_t *pp,
                      int dim);
int relazione_parziale(rel_t *pp,
                      int dim);
int relazione_totale(rel_t *pp,
                    int dim);
int relazione_equivalenza(rel_t *pp,
                          int dim);
int funzione_matematica(rel_t *pp,
                       int dim);
```



```

/*****
/* inclusione delle librerie standard del c */
*****/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*****
/* inclusione della libreria creata */
*****/

#include "libreria.h"

/*****
/* dichiarazione delle funzioni */
*****/

int valida_intero(const char *intero_stringa,
                 int *intero_validato);
int controlla_carattere(const char carattere);
int controlla_punteggiatura(const char *relazione,
                           int lunghezza,
                           int *conta_coppie);
int cerca_coppia(rel_t *pp,
                 int dim,
                 char* a,
                 char* b);
int riflessiva(rel_t *pp,
              int dim);
int antisimmetrica(rel_t *pp,
                  int dim);
int simmetrica(rel_t *pp,
              int dim);
int transitiva(rel_t *pp,
              int dim);
int verifica_transitiva(rel_t *pp,
                      int dim,
                      char* a,
                      char* b,
                      int* coppie_b);
int totale(rel_t *pp,
          int dim);
int confrontabile(rel_t *pp,
                 int dim,
                 char* x,
                 char* y);
int carica_insieme(insieme_t **insieme);
int conta_accoppiamenti_a(rel_t *pp,
                        int dim,
                        char* a);
int conta_accoppiamenti_b(rel_t *pp,
                        int dim,
                        char* b);

/*****
/* definizione delle funzioni */
*****/

/* definizione della funzione che valida
   un intero, restituirà 1 se è errato o
   0 se è corretto */
int valida_intero(const char *intero_stringa,    /* i.: stringa intero da validare */
                 int *intero_validato)          /* o.: intero validato */
{
    /* dichiarazione delle variabili locali alla funzione */

```

```
int i = 0,                                /* 1.: indice */
    lun;                                  /* 1.: lunghezza stringa intero */

int errore = 0;                           /* 0.: errore che rest. la fun. */

lun = strlen(intero_stringa);

/* controlla che lun non sia più grande di 5 */
if (lun >= 6)
    errore = 1;
else
{
    /* controlla che nella prima posizione non ci
       sia uno 0 */
    if (intero_stringa[i] != '0')
    {
        /* controlla che in ogni posizione dell'array ci
           sia una cifra da 0 a 9 */
        while (i < lun)
        {
            if (errore == 1)
                i = lun;
            else
            {
                /* se l'elemento nella posizione i è diverso da un
                   numero da 0 a 9 assegna 1 a "errore" */
                if (intero_stringa[i] != '0' &&
                    intero_stringa[i] != '1' &&
                    intero_stringa[i] != '2' &&
                    intero_stringa[i] != '3' &&
                    intero_stringa[i] != '4' &&
                    intero_stringa[i] != '5' &&
                    intero_stringa[i] != '6' &&
                    intero_stringa[i] != '7' &&
                    intero_stringa[i] != '8' &&
                    intero_stringa[i] != '9')
                    errore = 1;
            }

            i++;
        }
    }
    else
        errore = 1;
}

/* se non ha trovato nessun errore usa la funzione atoi
   per convertire la stringa in intero e assegnarlo a
   "intero_validato" */
if (errore != 1)
    *intero_validato = atoi(intero_stringa);

return errore;
}

/* definizione della funzione che controlla se il carattere
   è corretto, restituisce '1' se c'è un errore altrimenti '0' */
int controlla_carattere(const char carattere) /* 1.: carattere da controllare */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i;                                  /* 1.: indice array */

    char dizionario[] = {'a', 'b', 'c',           /* 1.: dizion. car. consentiti */
                          'd', 'e', 'f',
                          'g', 'h', 'i',
                          'j', 'k', 'l',
                          'm', 'n', 'o',
                          'p', 'q', 'r',
                          's', 't', 'u',
```

```

        'v', 'w', 'x',
        'y', 'z', '0',
        '1', '2', '3',
        '4', '5', '6',
        '7', '8', '9'};

int lun = 36; /* l.: car. dizionario */
int errore = 1; /* l.: esito funzione */

/* ciclo che confronta il carattere con l'array dizionario
   e se trova una ricorrenza assegna 0 a errore */
for (i = 0;
     i < lun && errore != 0;
     i++)
{
    if (carattere == dizionario[i])
        errore = 0;
}

/* ritorna l'esito della funzione */
return errore;
}

/* definizione della funzione che valida la
   punteggiatura della relazione, ritorna 1
   se c'è un errore, altrimenti 0; tornerà ,
   di conseguenza anche il numero delle coppie
   di elementi prensenti nella relazione */
int controlla_punteggiatura(const char *relazione, /* i.: relazione da validare */
                           int lunghezza, /* l.: lunghezza relazione */
                           int *dim) /* l./o: dim. per array */
{
    /* dichiarazione delle variabili locali alla funzione */
    int errore = 0, /* l.: errore */
        i = 0, /* l.: indice */
        j = 0, /* l.: indice */
        k = 0, /* l.: indice */
        numero_virgole = 0, /* l.: num. virg. nella rel. */
        numero_chiuse = 0, /* l.: num. par. chiuse nella rel. */
        conta_coppie = 0; /* l./o: num. coppie rel. */

    /* controlla che nella prima posizione
       e nell'ultima ci siano le parentesi
       '{' '}' */
    if (relazione[lunghezza - 1] != '}' ||
        relazione[0] != '{')
        errore = 1;

    /* controlla il contenuto della relazione usando come
       punti di riferimento la punteggiatura: '(', ',', ' ' */
    if (errore != 1)
    {
        for (i = 1;
             i < lunghezza - 3 && errore == 0;
             i = k + 1)
        {
            /* controlla la '(' */
            if (relazione[i] != '(')
                errore = 1;
            else
            {
                /* trova la prossima ',' */
                for (j = i + 2, errore = 1;
                     j < lunghezza - 3 && errore == 1;
                     j++)
                {
                    if (relazione[j] == ',')

```

```

        errore = 0;
    }

    /* trova la prossima ')' e controlla la ','
       successiva o la terminazione della relazione */
    if (errore != 1)
    {
        for (k = j + 1, errore = 1;
             k < lunghezza - 1 && errore == 1;
             k++)
        {
            if (relazione[k] == ')')
            {
                if ((lunghezza - 2) == k ||
                    relazione[k + 1] == ',')
                {
                    conta_coppie++;
                    errore = 0;
                }
            }
        }
    }
}

/* conta le ',', se non ci sono il giusto numero di ',' c'è un errore */
for (i = 2;
     i < lunghezza - 1;
     i++)
{
    if (relazione[i] == ',')
        numero_virgole++;
}
if (numero_virgole != (conta_coppie * 2 - 1))
    errore = 1;

/* esegue la stessa operazione con le parentesi tonde chiuse */
if (errore != 1)
{
    for (i = 2;
         i < lunghezza - 1;
         i++)
    {
        if (relazione[i] == ')')
            numero_chiuse++;
    }

    if (numero_chiuse != conta_coppie)
        errore = 1;
}

/* assegna a 'dim' il numero di coppie presenti
   nella relazione */
*dim = conta_coppie;

return errore;
}

/* definizione delle funzione che acquisisce una relazione,
   la valida, e la memorizza in una struttura apposita */
int acquisizione_relazione(rel_t **puntatore) /* i.: relazione da validare */
{
    /* dichiarazioni delle variabili locali alla funzione */
    int i, /* 1.: indice */
        j, /* 1.: indice */
        k; /* 1.: indice */

    char intero_validare[6]; /* 1.: intero da validare */

```

```

int l_relazione;          /* i.: lunghezza relazione */
char *relazione;          /* i.: relazione */

int errore = 0,           /* l.: controllo errore */
    virgole,              /* l.: contatore virgole */
    inizio,               /* l.: indice inizio controllo */
    fine;                 /* l.: indice fine controllo */

int *a,                   /* l.: indici elem. insieme A */
    *b;                   /* l.: indici elem. insieme B */
int dim,                  /* l.: dim. array 'a' e 'b' */
    l_mas_a,              /* l.: lungh. massima elem. a */
    l_mas_b = 0;          /* l.: lungh. massima elem. b */

rel_t *pp; /* puntatore a rel_t di appoggio (poi verrà copiato
           dentro *puntatore) */

char *temp,               /* l.: stringa temporanea */
    *temp_confr;          /* l.: stringa temporanea */

int ris = 0;              /* l.: esito confronto */

/* acquisizione */
do
{
    do
    {
        printf("\n\nQuanto è lunga la relazione?\n"
            "Si consideri anche la punteggiatura:\n"
            "Es. la relazione R = {(ab,b), (2b,a)} è lunga 15 caratteri.\n\n"
            "N.B. il numero da inserire deve essere maggiore di 7,\n"
            "    in quanto, la relazione più corta è lunga 7 caratteri.\n"
            "N.B. la relazione, per semplicità, va inserita senza spazi.\n");
        scanf("%s",
            intero_validare);
        /* controlla che sia stato inserito un intero e,
           se necessario, stampa il relativo errore */
        errore = valida_intero(intero_validare,
            &l_relazione);
        /* controlla che il numero inserito sia maggiore
           di 7, in quanto, la relazione più piccola è
           lunga 7 caratteri */
        if (l_relazione < 7)
            errore = 1;

        if (errore != 0)
        {
            printf("\nErrore: è stato inserito un numero non valido.\n\n");
        }
    } while (errore != 0);

    /* allocazione relazione */
    relazione = (char *) calloc(l_relazione + 1,
        sizeof(char));

    printf("\n\nInserire la relazione lunga %d caratteri.\n"
        "Una relazione tra l'insieme A e l'insieme B\n"
        "è un sottoinsieme di A x B.\n"
        "Es. {(a1,b1), (a2,b2), (a3,b3)}\n\n"
        "N.B. Gli elementi delle coppie possono essere\n"
        "    numeri interi positivi e/o lettere minuscole.\n"
        "N.B. La relazione, per semplicità, va inserita senza spazi.\n"
        "N.B. Non sono ammesse coppie uguali.\n",
        l_relazione);

    /* acquisizione relazione */

```

```
scanf("%s",
      relazione);

/* scrive '\0' alla fine dell'array */
relazione[l_relazione] = '\0';

/* validazione punteggiatura relazione */
errore = controlla_punteggiatura (relazione,
                                  l_relazione,
                                  &dim);

/* "creazione" array 'a' e 'b' */
if (errore != 1)
{
    /* allocazione a in base al numero delle coppie */
    a = (int *) calloc(dim,
                      sizeof(int));
    /* allocazione b in base al numero delle coppie - 1 */
    b = (int *) calloc(dim,
                      sizeof(int));

    /* scansiona l'array segnando le posizioni delle ','
       e delle ')' */
    for (i = 2, j = 0, virgole = 0;
         i < l_relazione - 1;
         i++)
    {
        if (relazione[i] == ',')
        {
            virgole++;
            /* segna solo le virgole tra
               due parentesi (occorrenze dispari) */
            if (virgole % 2 == 1)
            {
                a[j] = i;
                j++;
            }
        }
        /* segna le parentesi chiuse nell'array B,
           nella stessa posizione in cui ha salvato
           la virgola interna nell'array A */
        else if (relazione[i] == ')')
            b[j - 1] = i;
    }
}

if (errore != 1)
{
    /* controlla che gli elementi della relazione siano giusti */
    for (i = 0;
         i < dim && errore != 1;
         i++)
    {
        /* controlla gli elementi A */
        if (i == 0)
        {
            inizio = 2;
            fine = a[i];
            do
            {
                /* controlla ogni singolo carattere */
                errore = controlla_carattere(relazione[inizio]);
                inizio++;
            } while (inizio < fine && errore != 1);
        }
        else
        {
            inizio = b[i - 1] + 3;
            fine = a[i];
        }
    }
}
```

```
        do
        {
            /* controlla ogni singolo carattere */
            errore = controlla_carattere(relazione[inizio]);
            inizio++;
        } while (inizio < fine && errore != 1);
    }

    /* controlla gli elementi B */
    if (errore != 1)
    {
        inizio = a[i] + 1;
        fine = b[i];
        do
        {
            /* controlla ogni singolo carattere */
            errore = controlla_carattere(relazione[inizio]);
            inizio++;
        } while (inizio < fine && errore != 1);
    }
}

/* stampa messaggio di errore */
if (errore == 1)
{
    printf("\nErrore: la relazione inserita non è corretta.\n\n");
}

/* creazione struttura */
if (errore != 1)
{
    /* conta la lunghezza massima degli elem. ins. A */
    for (l_mas_a = a[0] - 2, i = 1;
        i < dim;
        i++)
    {
        if (l_mas_a < (a[i] - b[i - 1] - 3))
            l_mas_a = a[i] - b[i - 1] - 3;
    }

    /* conta la lunghezza massima degli elem. ins. B */
    for (i = 0;
        i < dim;
        i++)
    {
        if (l_mas_b < b[i] - a[i] - 1)
            l_mas_b = b[i] - a[i] - 1;
    }

    /* dichiarazione della struttura che
       conterrà la relazione */
    /* alloca dim strutture (dim contiene
       il numero di coppie da gestire) */
    pp = (rel_t *) malloc(dim * sizeof(rel_t));

    /* deve allocare due puntatori a char interni
       per ogni struttura che ha allocato
       nell'istruzione precedente (ovvero "dim" strutture) */
    for (i = 0;
        i < dim;
        i++)
    {
        /* alloca un array di caratteri sufficiente
           per il più grande elemento di A */
        pp[i].elem_a = (char *) malloc(l_mas_a * sizeof(char));

        /* alloca un array di caratteri sufficiente
           per il più grande elemento di B */
    }
}
```

```
    pp[i].elem_b = (char *) malloc(l_mas_b * sizeof(char));
}

/* caricamento struttura */

/* il primo for "posiziona" l'indice per gli
   array 'a' e 'b' e per la struttura 'rel' */
for (j = 0;
     j < dim;
     j++)
{
    /* copia elem. A */
    if (j == 0)
    {
        /* copia carattere per carattere gli elem.
           della relazione nella struttura */
        for (inizio = 2, fine = a[j], k = 0;
             inizio < fine;
             inizio++, k++)
        {
            pp[j].elem_a[k] = relazione[inizio];
        }
        /* inserimento carattere di terminazione stringa */
        pp[j].elem_a[k] = '\\0';
    }
    else
    {
        /* copia carattere per carattere gli elem.
           della relazione nella struttura */
        for (inizio = b[j - 1] + 3, fine = a[j], k = 0;
             inizio < fine;
             inizio++, k++)
        {
            pp[j].elem_a[k] = relazione[inizio];
        }
        /* inserimento carattere di terminazione stringa */
        pp[j].elem_a[k] = '\\0';
    }

    /* copia elem. B */
    for (inizio = a[j] + 1, fine = b[j], k = 0;
         inizio < fine;
         inizio++, k++)
    {
        pp[j].elem_b[k] = relazione[inizio];
    }
    /* inserimento carattere di terminazione stringa */
    pp[j].elem_b[k] = '\\0';
}

/* allocazione stringhe temporanee */
temp = (char *) malloc((l_mas_a + l_mas_b + 1) * sizeof(char));
temp_confr = (char *) malloc((l_mas_a + l_mas_b + 1) * sizeof(char));

/* controlla che nessuna coppia di
   valori sia uguale ad un'altra */
for (i = 0;
     i < dim && errore == 0;
     i++)
{
    /* concatena la stringa */
    strcpy(temp,
           pp[i].elem_a);
    strcat(temp,
           pp[i].elem_b);

    for (j = i + 1;
         j < dim && errore == 0;
         j++)
```



```

    {
        /* concatena la stringa da
           confrontare */
        strcpy(temp_confr,
               pp[j].elem_a);
        strcat(temp_confr,
               pp[j].elem_b);

        /* confronta le due stringhe per
           vedere se sono coppie uguali */
        ris = strcmp(temp,
                     temp_confr);
        if (ris == 0)
        {
            errore = 1;
            printf("\nErrore: non sono ammesse coppie uguali\n"
                  "      nella relazione.\n");
        }
    }
}

} while (errore != 0);

/* assegnamento 'pp' a '*puntatore'
   per ritornare la struttura */
*puntatore = pp;

return dim;
}

/* definizione della funzione che stampa
   una relazione binaria passata come parametro */
void stampa_relazione(rel_t *pp,      /* i.: relazione da stampare */
                     int dim)        /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i;                          /* l.: indice */

    /* messaggio di stampa della
       relazione binaria */
    printf("\nStampa relazione binaria: {");

    /* ciclo che stampa tutte le coppie
       della relazione */
    for (i = 0;
         i < dim;
         i++)
    {
        /* se non sta per stampare la prima
           coppia aggiunge la ',' per separare
           dalla coppia precedente */
        if (i > 0)
        {
            printf(", ");
        }

        /* stampa elemento A ed elemento B */
        printf("(%s, %s)", pp[i].elem_a, pp[i].elem_b);
    }

    /* una volta terminato stampa la parentesi
       graffa di chiusura */
    printf("}\n");
}

/* funzione che restituisce 1 se la coppia (a, b)
   data in input è presente nella relazione, 0 altrimenti */

```

```

int cerca_coppia(rel_t *pp,          /* 1.: relazione */
                int dim,             /* 1.: dimensione struttura */
                char* a,              /* i.: elem. a da cercare */
                char* b)              /* i.: elem. b da cercare */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i,                            /* 1.: indice */
        trovata = 0;                  /* o.: esito ricerca */

    /* ciclo che scandisce la struttura
       per cercare la coppia data in input */
    for (i = 0;
         i < dim && !trovata;
         i++)
    {
        /* se l'elemento 'a' della coppia da cercare
           è uguale all'elemento 'a' della struttura e
           l'elemento 'b' della coppia da cercare è uguale
           all'elemento 'b' della struttura allora assegna
           1 a trovata */
        if ((strcmp(a,
                    pp[i].elem_a) == 0) &&
            (strcmp(b,
                    pp[i].elem_b) == 0))
        {
            trovata = 1;
        }
    }

    return trovata;
}

/* definizione della funzione che restituisce 1
   se la relazione è riflessiva, 0 se non lo è */
int riflessiva(rel_t *pp,          /* i.: relazione */
               int dim)            /* 1.: dimensione struttura */
{
    /* dichiarazioni delle variabili locali
       alla funzione */
    int i,                            /* 1.: indice */
        riflessiva = 1;              /* o.: esito funzione */

    for (i = 0;
         i < dim && riflessiva;
         i++)
    {
        /* verifica che le due
           coppie non siano uguali */
        if (strcmp(pp[i].elem_a,
                    pp[i].elem_b) != 0)
        {
            /* per ogni coppia (a, b), lancia
               cerca_coppia su (a, a) e (b, b),
               così confronta ogni elemento
               dell'insieme con se stesso */
            if ((!cerca_coppia(pp,
                                dim,
                                pp[i].elem_a,
                                pp[i].elem_a)) ||
                (!cerca_coppia(pp,
                                dim,
                                pp[i].elem_b,
                                pp[i].elem_b)))
            {
                /* una delle due ricerche è fallita,
                   quindi la relazione non è sicuramente
                   riflessiva (per ogni elemento a devo
                   trovare una coppia (a, a)) */
            }
        }
    }
}

```

```
        riflessiva = 0;
    }
}

return riflessiva;
}

/* definizione della funzione che restituisce 1
   se la relazione è antisimmetrica, altrimenti 0 */
int antisimmetrica(rel_t *pp,          /* i.: relazione */
                  int dim)             /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i,                             /* l.: indice */
        antisimmetrica = 1,            /* o.: esito funzione */
        c = 0;                         /* l.: controllo */

    for (i = 0;
         i < dim && antisimmetrica;
         i++)
    {
        /* verifica che le due
           coppie non siano uguali */
        if (strcmp(pp[i].elem_a,
                   pp[i].elem_b) != 0)
        {
            /* se trova almeno una coppia non
               uguale vuol dire che sono state
               effettuate le verifiche successive */
            c++;

            /* per ogni coppia (a, b), cerca
               una coppia (b, a), se ne trova una,
               non c'è antisimmetria */
            if (cerca_coppia(pp,
                             dim,
                             pp[i].elem_b,
                             pp[i].elem_a))
            {
                antisimmetrica = 0;
            }
        }
    }

    /* se ci sono solo coppie uguali
       non c'è antisimmetria */
    if (c == 0)
        antisimmetrica = 0;

    return antisimmetrica;
}

/* definizione della funzione che restituisce 1
   se la relazione è Simmetrica, altrimenti 0 */
int simmetrica(rel_t *pp,              /* i.: relazione */
               int dim)                /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i,                             /* l.: indice */
        simmetrica = 1;                /* o.: esito funzione */

    for (i = 0;
         i < dim && simmetrica;
         i++)
    {
        /* verifica che le due
```

```

    coppie non siano uguali */
    if (strcmp(pp[i].elem_a,
               pp[i].elem_b) != 0)
    {
        /* per ogni coppia (a, b), cerca
           una coppia (b, a); se non ne
           trova una, non c'è proprietà
           simmetrica */
        if (!cerca_coppia(pp,
                          dim,
                          pp[i].elem_b,
                          pp[i].elem_a))
        {
            /* non c'è (b, a),
               assegna 0 a simmetrica */
            simmetrica = 0;
        }
    }
}

return simmetrica;
}

/* definizione della funzione che verifica
   se una relazione è transitiva */
int transitiva(rel_t *pp,                /* i.: relazione */
               int dim)                  /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i,                               /* l.: indice */
        transitiva = 1,                  /* o.: esito funzione */
        c_uguali = 0,                   /* l.: cont. coppie uguali */
        c_b = 0;                        /* l.: coppie che iniziano con b */

    for (i = 0;
         i < dim && transitiva;
         i++)
    {
        /* verifica che le due
           coppie non siano uguali */
        if (strcmp(pp[i].elem_a,
                   pp[i].elem_b) != 0)
        {
            /* per ogni coppia (a, b)
               chiama la verifica transitiva
               su quei valori */
            if (!verifica_transitiva(pp,
                                     dim,
                                     pp[i].elem_a,
                                     pp[i].elem_b,
                                     &c_b))
            {
                /* la verifica transitiva su (a,b)
                   è fallita, possiamo già dire
                   che la relazione non è transitiva */
                transitiva = 0;
            }
        }
        else
            c_uguali++;
    }

    /* se non ci sono coppie uguali e non è
       entrato nel controllo interno della funzione
       "verifica_transitiva" allora non è verificata la
       transitività */
    if (c_uguali == 0 &&

```

```

    c_b == 0)
    transitiva = 0;

    return transitiva;
}

/* definizione della funzione ausiliaria
   che verifica che la relazione tra l'elemento
   a e b valga anche per ogni c per cui c'è una
   coppia (b, c) */
int verifica_transitiva(rel_t *pp,      /* i.: relazione */
                        int dim,        /* l.: dimensione struttura */
                        char* a,        /* i.: elem. a da cercare */
                        char* b,        /* i.: elem. b da cercare */
                        int* coppie_b) /* l.: coppie che iniz. b */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i,                          /* l.: indice */
        transitiva = 1;             /* o.: esito funzione */

    for (i = 0;
         i < dim && transitiva;
         i++)
    {
        /* cerchiamo tutte le coppie che
           iniziano con l'elemento "b" */
        if ((strcmp(b,
                    pp[i].elem_a) == 0))
        {
            /* è stata trovata una coppia che inizia con b */
            *coppie_b = *coppie_b + 1;

            /* ha trovato una coppia (b, c), vediamo
               se trova nella relazione anche una
               coppia (a, c) */
            if (!cerca_coppia(pp,
                              dim,
                              a,
                              pp[i].elem_b))
            {
                /* non ha trovato nessuna coppia, la
                   relazione non è sicuramente transitiva */
                transitiva = 0;
            }

            /* se arriva qui ha trovato una
               coppia (a, c), va avanti con
               le prossime fino alla fine
               della verifica */
        }
    }

    return transitiva;
}

/* definizione della funzione che verifica
   se una relazione ha proprietà di totalità */
int totale(rel_t *pp,      /* i.: relazione */
           int dim)        /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i = 0,              /* l.: indice */
        j = 0,              /* l.: indice */
        totale = 1;         /* l.o.: esito funzione */

    /* per ogni elemento a e b di ogni

```

```

    coppia (a, b), verifica che siano
    confrontabili tra loro e con tutti
    gli elementi x e y delle successive
    coppie (x, y); alla prima verifica
    fallita possiamo dire che la relazione
    è parziale */
for (i = 0;
     i < dim && totale;
     i++)
{
    /* controlla a con a */
    if (!confrontabile(pp, dim, pp[i].elem_a, pp[i].elem_a))
    {
        totale = 0;
    }

    /* controlla b con b */
    if (!confrontabile(pp, dim, pp[i].elem_b, pp[i].elem_b))
    {
        totale = 0;
    }

    /* controlla se a, b sono confrontabili tra loro */
    if (!confrontabile(pp, dim, pp[i].elem_a, pp[i].elem_b))
    {
        totale = 0;
    }

    /* confronto a e b con tutti gli x e y delle coppie successive */
    for (j = i;
         j < dim;
         j++)
    {
        if (!confrontabile(pp,
                           dim,
                           pp[i].elem_a,
                           pp[j].elem_a) ||
            !confrontabile(pp,
                           dim,
                           pp[i].elem_a,
                           pp[j].elem_b) ||
            !confrontabile(pp,
                           dim,
                           pp[i].elem_b,
                           pp[j].elem_a) ||
            !confrontabile(pp,
                           dim,
                           pp[i].elem_b,
                           pp[j].elem_b))
        {
            totale = 0;
        }
    }
}

return totale;
}

/* definizione della funzione ausiliaria che
per una coppia di valori x e y cerca se
sono confrontabili nella relazione (ovvero
se esiste una coppia (x, y) o (y, x)) */
int confrontabile(rel_t *pp, /* i.: relazione */
                  int dim, /* l.: dimensione struttura */
                  char* x, /* i.: elem. x da confrontare */
                  char* y) /* i.: elem. y da confrontare */
{
    /* dichiarazione delle variabili locali alla funzione */

```

```

int confrontabile = 0;          /* o.: esito funzione */

if ((!cerca_coppia(pp,
                  dim,
                  x,
                  y)) && (!cerca_coppia(pp,
                                      dim,
                                      y,
                                      x)))
{
    /* non ha trovato nessuna coppia (x, y)
       o (y, x), i due valori non sono
       quindi confrontabili nella relazione */
    confrontabile = 0;
}
else
{
    /* se arriva qui, almeno una delle due
       coppie è stata trovata, quindi i
       valori sono confrontabili nella relazione */
    confrontabile = 1;
}

return confrontabile;
}

/* definizione della funzione che verifica
   se una relazione binaria è una relazione
   d'ordine parziale e stampa a video le proprietà
   che falliscono nel caso in cui non sia tale */
int relazione_parziale(rel_t *pp,          /* i.: relazione */
                      int dim)            /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int esito = 0;                        /* o.: esito funzione */
    int proprieta[4];                     /* l.o.: proprietà non soddisf. */

    /* controlla riflessività */
    proprieta[0] = riflessiva(pp,
                              dim);

    /* controlla antisimmetria */
    proprieta[1] = antisimmetrica(pp,
                                   dim);

    /* controlla transitività */
    proprieta[2] = transitiva(pp,
                              dim);

    /* controlla totalità */
    proprieta[3] = totale(pp,
                          dim);

    /* verifica se la relazione è di tipo
       d'ordine parziale */
    if (proprieta[0] == 1 &&
        proprieta[1] == 1 &&
        proprieta[2] == 1 &&
        proprieta[3] == 0)
    {
        printf("\nSi tratta di una relazione d'ordine parziale\n"
               "in quanto soddisfa le seguenti proprietà :\n"
               "- Riflessività \n- Antisimmetria\n- Transitività \n");
        esito = 1;
    }
}

```

```

/* se non è una relazione di tipo
d'ordine parziale stampa le relative
proprietà non soddisfatte */
if (esito == 0)
{
    if (proprieta[0] == 1 &&
        proprieta[1] == 1 &&
        proprieta[2] == 1 &&
        proprieta[3] == 1)
    {
        printf("\nNon si tratta di una relazione d'ordine parziale\n"
            "perchè è verificata anche la dicotomia.\n");
    }
    else
    {
        printf("\nNon si tratta di una relazione d'ordine parziale\n"
            "in quanto non soddisfa le seguenti proprietà :\n");

        if (proprieta[0] != 1)
            printf("- Riflessività \n");

        if (proprieta[1] != 1)
            printf("- Antisimmetria\n");

        if (proprieta[2] != 1)
            printf("- Transitività \n");
    }
}

return esito;
}

/* definizione della funzione che verifica se
una relazione binaria è una relazione d'ordine
totale e stampa a video la proprietà che
falliscono nel caso in cui non sia tale */
int relazione_totale(rel_t *pp,          /* i.: relazione */
                    int dim)             /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int esito = 0,                       /* o.: esito funzione */
        proprieta[4];                   /* l.o.: proprietà non sodd. */

    /* controlla riflessività */
    proprieta[0] = riflessiva(pp,
                             dim);

    /* controlla antisimmetria */
    proprieta[1] = antisimmetrica(pp,
                                   dim);

    /* controlla transitività */
    proprieta[2] = transitiva(pp,
                              dim);

    /* controlla totalità */
    proprieta[3] = totale(pp,
                          dim);

    /* verifica se la relazione è di tipo
d'ordine totale */
    if (proprieta[0] == 1 &&
        proprieta[1] == 1 &&
        proprieta[2] == 1 &&
        proprieta[3] == 1)
    {

```



```
printf("\nSi tratta di una relazione d'ordine totale\n"
      "in quanto soddisfa le seguenti proprietà :\n"
      "- Riflessività \n- Antisimmetria\n- Transitività \n"
      "- Dicotomia\n");
esito = 1;
}

/* se non è una relazione di tipo
   d'ordine totale stampa le relative
   proprietà non soddisfatte */
if (esito == 0)
{
    printf("\nNon si tratta di una relazione d'ordine totale\n"
          "in quanto non soddisfa le seguenti proprietà :\n");

    if (proprieta[0] != 1)
        printf("- Riflessività \n");

    if (proprieta[1] != 1)
        printf("- Antisimmetria\n");

    if (proprieta[2] != 1)
        printf("- Transitività \n");

    if (proprieta[3] != 1)
        printf("- Dicotomia\n");
}

return esito;
}

/* definizione della funzione che verifica
   se una relazione binaria è una relazione
   di equivalenza e stampa a video la proprietà
   che falliscono nel caso in cui non sia tale */
int relazione_equivalenza(rel_t *pp,      /* i.: relazione */
                          int dim)        /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int esito = 0,                      /* o.: esito funzione */
        proprieta[3];                  /* l.o.: proprietà non sodd. */

    /* controlla riflessività */
    proprieta[0] = riflessiva(pp,
                             dim);

    /* controlla simmetria */
    proprieta[1] = simmetrica(pp,
                              dim);

    /* controlla transitività */
    proprieta[2] = transitiva(pp,
                              dim);

    /* verifica se la relazione è di tipo
       d'equivalenza */
    if (proprieta[0] == 1 &&
        proprieta[1] == 1 &&
        proprieta[2] == 1)
    {
        printf("\nSi tratta di una relazione d'equivalenza\n"
              "in quanto soddisfa le seguenti proprietà :\n"
              "- Riflessività \n- Simmetria\n- Transitività \n");
        esito = 1;
    }
}
```

```

/* se non è una relazione di tipo
   d'equivalenza stampa le relative
   proprietà non soddisfatte */
if (esito == 0)
{
    printf("\nNon si tratta di una relazione d'equivalenza\n"
           "in quanto non soddisfa le seguenti proprietà :\n");

    if (proprieta[0] != 1)
        printf("- Riflessività \n");

    if (proprieta[1] != 1)
        printf("- Simmetria\n");

    if (proprieta[2] != 1)
        printf("- Transitività \n");
}

return esito;
}

/* definizione della funzione che restituisce
   il numero di coppie che iniziano per "a"
   (es. (a,b) (a,c)...) */
int conta_accoppiamenti_a(rel_t *pp, /* i.: relazione */
                          int dim, /* l.: dimensione struttura */
                          char* a) /* i.: elem. da confrontare */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i, /* l.: indice */
        conta = 0; /* o.: contatore */

    for (i = 0;
         i < dim;
         i++)
    {
        if ((strcmp(a,
                    pp[i].elem_a) == 0))
        {
            /* coppia (a, y) trovata,
               si incrementa il contatore */
            conta++;
        }
    }

    return conta;
}

/* definizione della funzione che restituisce
   il numero di coppie che hanno come secondo elemento
   'b' (es. (a,b) (c,b)...) */
int conta_accoppiamenti_b(rel_t *pp, /* i.: relazione */
                          int dim, /* l.: dimensione struttura */
                          char* b) /* i.: elem. da confrontare */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i, /* l.: indice */
        conta = 0; /* o.: contatore */

    for (i = 0;
         i < dim;
         i++)
    {
        if ((strcmp(b,
                    pp[i].elem_b) == 0))
        {
            /* coppia (x, b) trovata,

```

```

        si incrementa il contatore */
        conta++;
    }
}

return conta;
}

/* definizione della funzione che chiede
all'utente di inserire una serie di valori
di un insieme */
int carica_insieme(insieme_t **insieme) /* i.: insieme B */
{
    /* dichiarazione delle variabili locali alla funzione */
    int elem_agg = 0, /* i.: numero elementi */
        i = 0, /* l.: indice */
        j = 0, /* l.: indice */
        l_temp; /* l.: lung. temp. */

    char risposta, /* i.: risp. utente */
        intero_validare[3]; /* l.: intero da validare */

    int errore = 0; /* l.: esito controllo */

    /* puntatore a insieme_t di appoggio
    (poi lo copieremo dentro *insieme) */
    insieme_t *nuovo_insieme;

    /* variabile di appoggio per i dati
    inseriti: accetta elementi di,
    massimo, 100 caratteri */
    char temp_elem[100];

    /* stampa messaggio direttiva */
    printf("\nPer verificare se la funzione è iniettiva,\n"
        "suriettiva o biiettiva è necessario conoscere\n"
        "l'insieme B.\n\nN.B. Gli elementi dell'insieme B avranno\n"
        "le stesse\nregole grammaticali degli elementi della\n"
        "relazione.\n");

    /* domanda per insieme B */
    do
    {
        printf("\nL'insieme B è composto da altri elementi oltre\n"
            "quelli elencati nella relazione?\n\n(s/n)\n");

        /* acquisizione risposta saltando newline */
        scanf("%c%c",
            &risposta,
            &risposta);

        /* messaggio di errore */
        if (risposta != 's' &&
            risposta != 'n')
        {
            printf("\nErrore: carattere non accettato.\n"
                "\nInserire 's' se la risposta è affermativa,\n"
                "'n' se la risposta è negativa.\n");
        }
    } while (risposta != 's' &&
        risposta != 'n');

    /* se la risposta è affermativa si procede con
    l'acquisizione dei valori aggiuntivi */
    if (risposta == 's')
    {
        /* acquisizione numero elem. aggiuntivi */
        do

```

```
{
    printf("\nDa quanti altri elementi è composto l'insieme B?\n");

    /* acquisizione intero sotto forma di stringa */
    scanf("%s",
        intero_validare);

    /* controlla che sia stato inserito un intero e,
       se necessario, stampa il relativo errore */
    errore = valida_intero(intero_validare,
        &elem_agg);

    /* stampa messaggio di errore */
    if (errore != 0)
    {
        printf("\nErrore: è stato inserito un numero non valido.\n\n");
    }

} while (errore != 0);

/* allocazione insieme */
nuovo_insieme = (insieme_t *) malloc(elem_agg *
    sizeof(insieme_t));

printf("Caricare %d elementi dell'insieme B"
    " che non\nsono già inclusi nelle coppie"
    " della relazione\n",
    elem_agg);

/* caricamento elementi nell'insieme B */
for(i = 0;
    i < elem_agg;
    i++)
{
    do
    {
        printf("Inserire elemento numero %d:\n",
            i + 1);
        /* acquisizione elemento */
        scanf("%s",
            temp_elem);

        /* acquisizione lunghezza elem. inserito */
        l_temp = strlen(temp_elem);

        for(j = 0, errore = 0;
            j < l_temp && errore == 0;
            j++)
        {
            /* controlla carattere per carattere */
            errore = controlla_carattere(temp_elem[j]);

            /* stampa messaggio di errore */
            if (errore == 1)
            {
                printf("\nErrore: l'elemento inserito non è corretto.\n\n");
            }
        }
    } while (errore != 0);

    /* allocazione un nuovo elemento */
    nuovo_insieme[i].elem = (char *) malloc((int)l_temp
        * sizeof(char));

    /* copio l'elemento inserito nel nuovo
       elemento allocato */
    strcpy(nuovo_insieme[i].elem,
        temp_elem);
}
```

```

}

/* assegnamento 'nuovo_insieme' a '*insieme'
   per ritornare la struttura */
*insieme = nuovo_insieme;

return elem_agg;
}

/* definizione della funzione che verifica se
   la relazione è una funzione matematica.
   Se non lo è, stampa la proprietà che fallisce.
   Se lo è, indica se è iniettiva, suriettiva o biettiva */
int funzione_matematica(rel_t *pp,      /* i.: relazione */
                       int dim)        /* l.: dimensione struttura */
{
    /* dichiarazione delle variabili locali alla funzione */
    int i = 0;                          /* l.: indice */

    int no_funzione = 0,                 /* l.: esito ver. */
        iniettiva = 1,                   /* l.: esito inie. */
        suriettiva = 1;                  /* l.: esito suri. */

    int num_elementi_a;                  /* l.: numero elem. a*/

    /* controlla se è una funzione */
    for (i = 0;
         i < dim && !no_funzione;
         i++)
    {
        /* per ogni a che trova nelle coppie
           (a, b), verifica se ci sono più
           coppie (a, x). Se ci sono, la relazione
           non è sicuramente una funzione matematica */
        if (conta_accoppiamenti_a(pp,
                                   dim,
                                   pp[i].elem_a) > 1)
        {
            /* una funzione non può avere più risultati
               per ogni X del dominio */
            no_funzione = 1;
            printf("\nLa relazione non è una funzione matematica:\n"
                  "ci sono più valori del codominio in relazione \n"
                  "con l'elemento %s del dominio.\n",
                  pp[i].elem_a);
        }

        /* con questa funzione si conta quante coppie (x, b)
           ci sono con l'elemento "b" a destra */
        num_elementi_a = conta_accoppiamenti_b(pp,
                                                dim,
                                                pp[i].elem_b);

        /* controllo iniettività */
        if (num_elementi_a > 1)
        {
            /* se ci sono più coppie con l'elemento "b"
               di cui sopra, la funzione non è iniettiva
               (perché associa più elementi del dominio
               a un particolare elemento "b" del codominio) */
            iniettiva = 0;
        }
    }

    if (!no_funzione)
    {
        /* caricamento insieme B */
        int dim_b = 0; /* dimensione dell'insieme B */
    }
}

```

```
insieme_t *insieme_b = NULL; /* puntatore all'insieme B */

dim_b = carica_insieme(&insieme_b);

/* controllo suriettività */
for (i = 0;
     i < dim_b && suriettiva;
     i++)
{
    /* per ogni elemento "b" dell'insieme B,
       cerca se esistono coppie (x, b). Se
       non esistono, la funzione non è suriettiva
       perché il cod(f) non comprende tutto
       l'insieme B */
    if (conta_accoppiamenti_b(pp,
                              dim,
                              insieme_b[i].elem) <= 0)
    {
        suriettiva = 0;
    }
}

/* procede con la verifica delle proprietà */
if (!iniettiva && !suriettiva)
{
    printf("\nLa relazione è una funzione matematica.\n");
}
else
{
    printf("\nLa relazione è una funzione matematica\n"
           "ed è");

    if (iniettiva && suriettiva)
    {
        printf(" biiettiva.\n");
    }
    else if (iniettiva)
    {
        printf(" iniettiva.\n");
    }
    else if (suriettiva)
    {
        printf(" suriettiva.\n");
    }
}

return 1;
}
```

```
/* **** */
/* inclusione delle librerie standard del c */
/* **** */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* **** */
/* inclusione della libreria creata */
/* **** */

#include "libreria.h"

/* **** */
/* definizione della funzione main */
/* **** */

int main(void)
{
    /* dichiarazione delle variabili locali alla funzione */
    int dim = 0; /* i.: dimensione struttura */

    /* inizializzazione struttura */
    rel_t *pp = NULL;

    /* chiamata alla funzione che si occupa
       di acquisire e validare la relazione */
    dim = acquisizione_relazione(&pp);

    /* chiamata alla funzione che stampa
       a video la relazione */
    stampa_relazione(pp,
                     dim);

    /* chiamata alla funzione che controlla
       se la relazione è di tipo parziale */
    relazione_parziale(pp,
                       dim);

    /* chiamata alla funzione che controlla
       se la relazione è di tipo totale */
    relazione_totale(pp,
                     dim);

    /* chiamata alla funzione che controlla
       se la relazione è di tipo d'equivalenza */
    relazione_equivalenza(pp,
                           dim);

    /* chiamata alla funzione che controlla
       se la relazione è una funzione matematica */
    funzione_matematica(pp,
                         dim);

    return (0);
}
```

```
main: main.o libreria.o Makefile
    gcc -ansi -Wall -O main.o libreria.o -o progetto

main.o: main.c libreria.h Makefile
    gcc -ansi -Wall -O -c main.c

libr.o: libreria.c libreria.h Makefile
    gcc -ansi -Wall -O -c libreria.c
pulisci:
    rm -f main.o libreria.o
pulisci_tutto:
    rm -f main main.o libreria.o
```



## Testing del programma

Di seguito vengono illustrati degli esempi di input e output.

Esempi di input:

Errato:

```
Quanto è lunga la relazione?  
Si consideri anche la punteggiatura:  
Es. la relazione  $R = \{(ab,b),(2b,a)\}$  è lunga 15 caratteri.  
  
N.B. il numero da inserire deve essere maggiore di 7,  
in quanto, la relazione più corta è lunga 7 caratteri.  
N.B. la relazione, per semplicità, va inserita senza spazi.  
a  
  
Errore: è stato inserito un numero non valido.
```

Corretto:

```
Quanto è lunga la relazione?  
Si consideri anche la punteggiatura:  
Es. la relazione  $R = \{(ab,b),(2b,a)\}$  è lunga 15 caratteri.  
  
N.B. il numero da inserire deve essere maggiore di 7,  
in quanto, la relazione più corta è lunga 7 caratteri.  
N.B. la relazione, per semplicità, va inserita senza spazi.  
7  
  
Inserire la relazione lunga 7 caratteri.  
Una relazione tra l'insieme A e l'insieme B  
è un sottoinsieme di  $A \times B$ .  
Es.  $\{(a1,b1),(a2,b2),(a3,b3)\}$   
  
N.B. Gli elementi delle coppie possono essere  
numeri interi positivi e/o lettere minuscole.  
N.B. La relazione, per semplicità, va inserita senza spazi.  
N.B. Non sono ammesse coppie uguali.
```

Errato:

```
Inserire la relazione lunga 7 caratteri.  
Una relazione tra l'insieme A e l'insieme B  
è un sottoinsieme di  $A \times B$ .  
Es.  $\{(a1,b1),(a2,b2),(a3,b3)\}$   
  
N.B. Gli elementi delle coppie possono essere  
numeri interi positivi e/o lettere minuscole.  
N.B. La relazione, per semplicità, va inserita senza spazi.  
N.B. Non sono ammesse coppie uguali.  
 $\{(A,b)\}$   
  
Errore: la relazione inserita non è corretta.
```

Corretto:

```
Inserire la relazione lunga 7 caratteri.  
Una relazione tra l'insieme A e l'insieme B  
è un sottoinsieme di  $A \times B$ .  
Es.  $\{(a1,b1),(a2,b2),(a3,b3)\}$   
  
N.B. Gli elementi delle coppie possono essere  
numeri interi positivi e/o lettere minuscole.  
N.B. La relazione, per semplicità, va inserita senza spazi.  
N.B. Non sono ammesse coppie uguali.  
 $\{(a,a)\}$   
  
Stampa relazione binaria:  $\{(a, a)\}$ 
```

Relazione d'ordine parziale errata:

```
Stampa relazione binaria:  $\{(a, b), (b, c), (a, c)\}$   
  
Non si tratta di una relazione d'ordine parziale  
in quanto non soddisfa le seguenti proprietà:  
- Riflessività
```

Relazione d'ordine parziale corretta:

```
Stampa relazione binaria:  $\{(a, b), (b, c), (a, c), (a, a), (b, b), (c, c), (d, d)\}$   
  
Si tratta di una relazione d'ordine parziale  
in quanto soddisfa le seguenti proprietà:  
- Riflessività  
- Antisimmetria  
- Transitività
```

Relazione totale errata:

```
Stampa relazione binaria: {(a, b), (b, c), (a, c), (a, a), (b, b), (c, c), (d, d)}  
  
Non si tratta di una relazione d'ordine totale  
in quanto non soddisfa le seguenti proprietà:  
- Dicotomia
```

Relazione totale corretta:

```
Stampa relazione binaria: {(a, b), (b, c), (a, c), (a, a), (b, b), (c, c)}  
  
Si tratta di una relazione d'ordine totale  
in quanto soddisfa le seguenti proprietà:  
- Riflessività  
- Antisimmetria  
- Transitività  
- Dicotomia
```

Relazione d'equivalenza errata:

```
Stampa relazione binaria: {(a, b), (b, c), (a, c), (a, a), (b, b), (c, c)}  
  
Non si tratta di una relazione d'equivalenza  
in quanto non soddisfa le seguenti proprietà:  
- Simmetria
```

Relazione d'equivalenza corretta:

```
Stampa relazione binaria: {(a, b), (b, c), (a, c), (a, a), (b, b), (c, c), (b, a), (c, b), (c, a)}  
  
Si tratta di una relazione d'equivalenza  
in quanto soddisfa le seguenti proprietà:  
- Riflessività  
- Simmetria  
- Transitività
```

Relazione funzionale errata:

```
Stampa relazione binaria: {(a, b), (b, c), (a, c)}  
  
La relazione non è una funzione matematica:  
ci sono più valori del codominio in relazione  
con l'elemento a del dominio.
```

Relazione funzionale suriettiva:

```

Stampa relazione binaria: {(a, b), (b, c), (d, c)}

Per verificare se la funzione è iniettiva,
suriettiva o biiettiva è necessario conoscere
l'insieme B.

N.B. Gli elementi dell'insieme B avranno le stesse
regole grammaticali degli elementi della relazione.

L'insieme B è composto da altri elementi oltre
quelli elencati nella relazione?

(s/n)
n

La relazione è una funzione matematica
ed è suriettiva.

```

Relazione funzionale iniettiva:

```

Stampa relazione binaria: {(a, b), (b, c), (d, e)}

Per verificare se la funzione è iniettiva,
suriettiva o biiettiva è necessario conoscere
l'insieme B.

N.B. Gli elementi dell'insieme B avranno le stesse
regole grammaticali degli elementi della relazione.

L'insieme B è composto da altri elementi oltre
quelli elencati nella relazione?

(s/n)
s

Da quanti altri elementi è composto l'insieme B?
1
Caricare 1 elementi dell'insieme B che non
sono già inclusi nelle coppie della relazione
Inserire elemento numero 1:
h

La relazione è una funzione matematica
ed è iniettiva.

```

Relazione funzionale biiettiva:

```

Stampa relazione binaria: {(a, b), (b, c), (d, e)}

Per verificare se la funzione è iniettiva,
suriettiva o biiettiva è necessario conoscere
l'insieme B.

N.B. Gli elementi dell'insieme B avranno le stesse
regole grammaticali degli elementi della relazione.

L'insieme B è composto da altri elementi oltre
quelli elencati nella relazione?

(s/n)
n

La relazione è una funzione matematica
ed è biiettiva.

```

## Verifica del programma

Si dice tripla di Hoare una tripla della seguente forma:  $Q \ S \ R$ , dove  $Q$  è un predicato detto preconditione,  $S$  è un'istruzione ed  $R$  è un predicato detto postcondizione.

La tripla  $Q \ S \ R$  è vera sse l'esecuzione dell'istruzione  $S$  inizia in uno stato della computazione in cui  $Q$  è soddisfatta e termina raggiungendo uno stato della computazione in cui  $R$  è soddisfatta.

Di seguito è riportata una parte del programma validata tramite tripla di Hoare.

```
if ((!cerca_coppia(pp,
                    dim,
                    x,
                    y)) && (!cerca_coppia(pp,
                                          dim,
                                          y,
                                          x)))
{
    confrontabile = 0;
}
else
{
    confrontabile = 1;
}
```

Sia  $(!cerca\_coppia(pp, dim, x, y)) \ \&\& \ (!cerca\_coppia(pp, dim, y, x)) = z \rightarrow z = 1 \vee z = 0$

$wp(S, R) = ((Beta \rightarrow wp(S, R)) \vee (!Beta \rightarrow wp(S, R)))$

$Q \Rightarrow (z = 1) \vee (z = 0)$

$R \Rightarrow ((confrontabile = 0) \vee (confrontabile = 1))$

$S \Rightarrow ((z = 1) \rightarrow (confrontabile = 0)), ((z = 0) \rightarrow (confrontabile = 1))$

ovvero:

$((z = 1) \rightarrow ((confrontabile = 0) \vee (confrontabile = 1))) = \text{Vero}$

$((z = 0) \rightarrow ((confrontabile = 0) \vee (confrontabile = 1))) = \text{Vero}$

$(\text{Vero} \vee \text{Vero}) = \text{Vero}$