ABC delle Matrici in C#

Versione 1.1

Massimo Antonellini, 2017

1. Introduzione

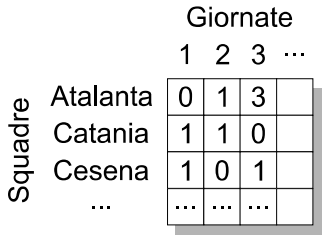
Cosa sono?

Le matrici sono vettori a due o più dimensioni (di solito non più di 3).

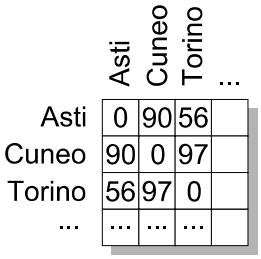
A cosa servono?

A memorizzare tabelle di dati dello stesso tipo (interi, reali, stringhe, ecc.).

Esempio 1. Punteggi ottenuti dalle squadre in ciascun giorno di campionato. Nella matrice **le righe indicano le squadre** di calcio e **le colonne alle giornate** di campionato. Quindi all'incrocio tra la riga di Catania e la colonna della terza giornata troveremo i 0 punti ottenuti dal Catania in quella giornata.



Esempio 2. Distanze stradali tra coppie di città di un elenco. Nella matrice **le città corrispondono sia alle righe che alle colonne** e all'incrocio tra la riga di Asti e la colonna di Torino troveremo la distanza in chilometri (56) tra le due città. Dove il numero della riga è uguale a quello della colonna, (stessa città) la distanza è zero.



Come si dichiara una matrice? (versione "ingenua")

La dichiarazione è simile a quella di un vettore, ma:

1. le parentesi quadre del tipo di dato contengono una o più virgole, una in meno delle dimensioni della matrice
2. le parentesi quadre della `new` contengono due o più costanti, tante quante sono le dimensioni.

Esempio. Per memorizzare i **punteggi delle 38 giornate di campionato di 20 squadre** di calcio dichiariamo la matrice:

```
byte[,] punteggio = new byte[20, 38];              // byte = un valore tra 0 e 255
```

Indici

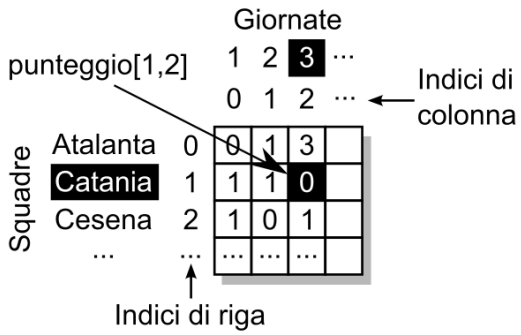
Come per i vettori, **in C# gli indici di una matrice sono numeri interi che partono da zero.**

Esempio. Per identificare un elemento della matrice dell'esempio precedente è necessario utilizzare 2 indici, uno per la riga e uno per la colonna, dove gli indici di riga saranno compresi tra 0 e 19 e gli indici di colonna saranno compresi tra 0 e 37. Pertanto i punti del Catania (la seconda squadra dell'elenco) nella terza giornata di campionato si troveranno nell'elemento `punteggio[1,2]`:

Nota 1: Le matrici sono utilizzate in numerosi problemi matematici, ad esempio per memorizzare i coefficienti di un sistema di equazioni lineari. Nella formulazione di questi problemi, solitamente gli indici partono da 1 e non da 0.

Nota 2: L'idea che il primo indice di un elemento indichi la riga e il secondo indichi la colonna **è una pura convenzione** e dipende da come la matrice è disegnata!

Nota 3: L'informazione che l'Atalanta corrisponde alla prima riga della tabella, quella del Catania alla seconda e così via, deve essere memorizzata in un'altra struttura di dati, ad esempio un vettore di stringhe.



Come si dichiara una matrice? (versione "professionale") Dipende dai dati!

Se non conosciamo quanti dati dovremo memorizzare, possiamo dichiarare una matrice nel modo più generale:

```
const int MAXR = ...;                // Righe e colonne disponibili
const int MAXC = ...;

int[,] m = new int[MAXR, MAXC];      // La matrice contiene MAXR x MAXC elementi

int rm = 0;                          // Righe e colonne effettivamente usate
int cm = 0;
```

La dichiarazione di una matrice può essere semplificata e ottimizzata risparmiando spazio se abbiamo qualche informazione sul numero e sul significato dei dati da memorizzare. Ad esempio, per memorizzare i punteggi ottenuti nelle varie giornate del campionato di calcio, se le squadre sono sempre 20, potremmo scrivere:

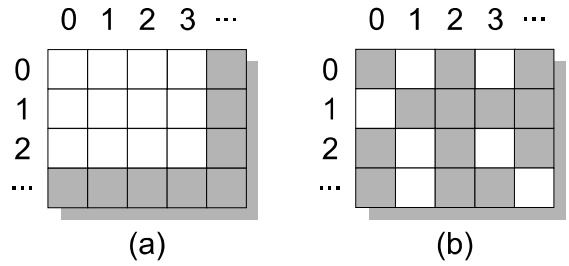
```
const int MAXSQUADRE = 20;
const int MAXGIORNATE = (MAXSQUADRE-1) * 2;

int[,] mpunteggio = new int[MAXSQUADRE, MAXGIORNATE];

int ngiornate = 0;                   // Giornate di campionato già svolte
```

In questo modo:

- la matrice non conterrà spazio inutilizzato;
- la variabile `ngiornate` permetterà tenere traccia di quante giornate sono già state svolte e quindi di inserire i punteggi della prossima giornata nella posizione corretta.

Matrici rettangolari e matrici sparse

Nei casi più comuni, i dati di una matrice vengono memorizzati sotto forma di **matrice rettangolare**, formata da un certo numero di righe, ciascuna delle quali contiene lo stesso numero di elementi. In alcuni casi, i dati sono distribuiti in modo disordinato all'interno della matrice, con molti valori "vuoti". Queste matrici sono chiamate **matrici sparse** e per esse esistono **speciali tecniche di memorizzazione** (lasciate per uno studio futuro) che permettono di risparmiare spazio.



(a) Matrice rettangolare.
(b) Matrice sparsa.

Alcune tecniche per elaborare sia le matrici rettangolari che quelle sparse sono descritte in seguito.

Matrici simmetriche e triangolari

Una matrice viene detta **simmetrica** se **per ogni valore degli indici i e j vale la proprietà m[i, j] = m[j, i]**. In altre parole, gli elementi a destra e sinistra della diagonale sono uguali. Nota: la diagonale è costituita dagli elementi m[i, j] dove i = j.

Esempio: La matrice delle distanze tra le città è una matrice simmetrica, perché la distanza tra due città A e B è (più o meno) la stessa che tra B e A. In altre parole, prese due città, la loro distanza viene sempre memorizzata due volte! Notate anche che la distanza tra una città e sé stessa è sempre zero (la diagonale).

	Asti	Cuneo	Torino	...
Asti	0	90	56	
Cuneo	90	0	97	
Torino	56	97	0	
...	...	...	...	

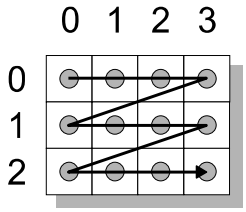
In questi casi è possibile (anche se non sempre sensato) memorizzare i dati solo una volta, ottenendo una **matrice triangolare**.

Scansione di una matrice

La scansione di una matrice (a 2 dimensioni) può essere effettuata per righe o per colonne. In entrambi i casi, la scansione richiede l'uso di **due cicli annidati, uno per le righe e l'altro per le colonne**. Le due figure illustrano l'ordine in cui gli elementi dell'array `m` vengono elaborati, rispettivamente nella scansione per righe e per colonne (con `rm = 3` e `cm = 4`).

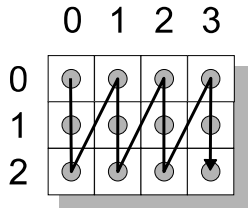
Scansione della matrice `m` per righe:

```
for (i = 0; i < rm; i = i + 1)          // Per tutte le righe
{
   for (j = 0; j < cm; j = j + 1)       // Per tutte le colonne
   {
      «elabora l'elemento m[i,j]»;
   }
}
```



Scansione della matrice `m` per colonne, **scambiando l' ordine dei due cicli**:

```
for (j = 0; j < cm; j = j + 1)          // Per tutte le colonne
{
   for (i = 0; i < rm; i = i + 1)       // Per tutte le righe
   {
      «elabora l'elemento m[i,j]»;
   }
}
```



Esempio: Somma

Per calcolare la somma dei dati di una matrice è sufficiente aggiungere ad una delle scansioni le istruzioni per sommare i dati.

```
somma = 0;
for (i = 0; i < rm; i = i + 1)          // Scansione per righe
{
   for (j = 0; j < cm; j = j + 1)
   {
      somma = somma + m[i,j];
   }
}
```

Critica

In molti casi, eseguire la somma di un'intera matrice non ha senso.

Esempi:

- **coefficienti delle equazioni di un sistema lineare**: non ha senso sommare né l'intera matrice, né singole righe o colonne;
- **distanze chilometriche tra coppie di città**: ha esclusivamente senso sommare le distanze di un percorso che attraversa più città;
- **punteggi delle squadre di calcio nelle varie giornate**; non ha senso sommare l'intera matrice, ma ha senso sommare tutti i punteggi di una riga, perché forniscono il punteggio della squadra nella classifica.

Esempio: Ricerca sequenziale (e altri cicli con uscita anticipata)

La ricerca sequenziale è simile a quella su un vettore, ma occorre un metodo per **uscire contemporaneamente da due cicli**.

```
n = ...;                                // n è il dato da cercare
for (i = 0; i < rm; i = i + 1)
{
   for (j = 0; j < cm; j = j + 1)
   {
      if (m[i,j] == n)
      {
         posR = i;                      // Salva la posizione in posR e posC
         posC = j;
         goto uscita;                   // Esci da entrambi i cicli
      }
   }
}
uscita: «istruzioni da eseguire dopo la ricerca»;
```

Critica

L'istruzione `goto` permette di eseguire un salto incondizionato ad un'istruzione di un programma ed **è considerata una specie di "bestemmia" dagli studiosi dei moderni linguaggi di programmazione**, perché se usata in modo "selvaggio" produce programmi illeggibili. Come esercizio, provate a riscrivere il ciclo senza `goto`. Vi serviranno due istruzioni `while` e una variabile booleana `esci` da inserire nelle condizioni di entrambi i cicli, che – naturalmente – servirà per uscire dai due cicli.

Caricamento dei dati in una matrice rettangolare

In una matrice rettangolare i dati possono essere caricati per righe o per colonne. Ma tutto questo ha senso? Alcuni esempi aiuteranno a ragionare su questo aspetto. Cominciamo con la matrice "generica" che dovrebbe andare bene un po' per tutto.

```
const int MAXRIGHE = ...;                   // Righe e colonne disponibili
const int MAXCOLONNE = ...;

int[,] m = new int[MAXRIGHE, MAXCOLONNE];   // MAXRIGHE x MAXCOLONNE elementi

int rm = 0;                                 // Righe e colonne effettivamente usate
int cm = 0;
```

Caricando i dati per righe, verranno memorizzati prima i dati della prima riga (`m[0,0], m[0,1], m[0,2]`, ecc.), poi quelli della seconda riga (`m[1,0], m[1,1], m[1,2]`, ecc.), e così via. Invece, se si caricano i dati per colonne, verranno memorizzati prima i dati della prima colonna (`m[0,0], m[1,0], m[2,0]`, ecc.), poi quelli della seconda colonna (`m[0,1], m[1,1], m[2,1]`, ecc.), ecc.

Proviamo a caricare i dati per righe, senza preoccuparci troppo per i controlli:

```
rm = Convert.ToInt32(Interaction.InputBox("Introdurre il numero delle righe"));
cm = Convert.ToInt32(Interaction.InputBox("Introdurre il numero delle colonne"));

for (i = 0; i < rm; i = i + 1)
{
   for (j = 0; j < cm; j = j + 1)
   {
      m[i,j] = Convert.ToInt32(Interaction.InputBox("Introdurre l'elemento m[" + i + "," + j + "]"));
   }
}
```

Critica

In molti casi, il caricamento di un'intera matrice non ha senso.

Esempi:

- **coefficienti delle equazioni** di un sistema lineare: ha senso caricarli tutti insieme;
- **punteggi delle squadre di calcio nelle varie giornate**: ha senso caricare una colonna alla volta, cioè tutti i punteggi di una giornata di campionato;
- **distanze chilometriche tra coppie di città:** ha senso caricarle tutte insieme, ma occorrerebbe caricare i duplicati una sola volta. In altre parole, la distanza tra `Asti` e `Cuneo` andrebbe letta una sola volta e inserita due volte nella matrice, nelle posizioni simmetriche rispetto alla diagonale. In pratica, se `i` è la posizione di `Asti` e `j` quella di `Cuneo`, la stessa distanza dovrà essere inserita sia in `m[i,j]` che in `m[j,i]`.

Matrici sparse e lookup

Come detto, una matrice sparsa è una matrice in cui molti elementi possiedono il valore zero o, comunque, un valore speciale che indica che il dato non è presente. Esistono tecniche che permettono di memorizzare matrici sparse risparmiando spazio e consumando un po' di tempo di CPU per accedere ai dati, lasciate allo studente per uno studio successivo. Per semplicità, assumeremo che le matrici sparse vengano rappresentate come una normale matrice di C#, con relativo spreco di spazio.

Applicazioni

Alcune applicazioni delle matrici sparse:

- **rappresentazione di sistemi complessi, incluse le carte geografiche**:
  - collegamenti stradali diretti tra città (chilometri)
  - collegamenti punto-punto tra computer di una rete (1 se due computer sono direttamente collegati, 0 altrimenti);
- **problemi demografici (statistiche di una popolazione)**:
  - numero di incidenti stradali nei comuni italiani in base a certe caratteristiche del territorio e della popolazione (righe=abitanti per metro quadro, colonne = reddito medio, valori = numero di incidenti stradali in un anno);
  - incidenza dell'infarto (righe = pressione sanguigna, colonne = valore del colesterolo totale, valori = numero casi di infarto in un comune/regione/nazione;
- **videogame**: posizione (x, y) delle varie sprite sullo schermo per determinare eventuali collisioni.

Lookup (accesso diretto)

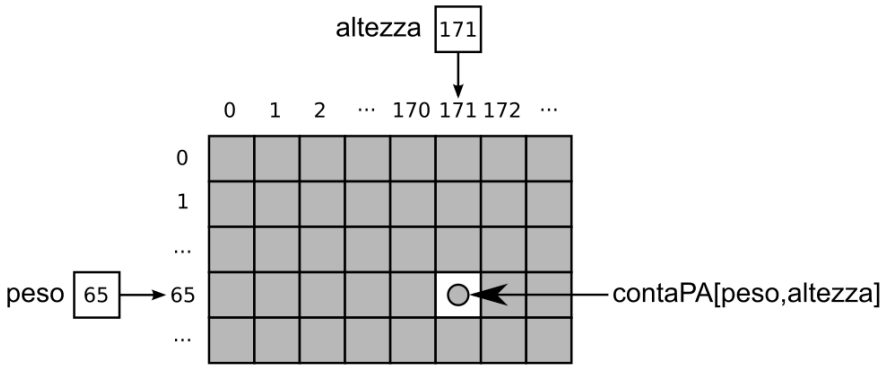
Il **lookup** è una **tecnica per accedere direttamente a un singolo elemento di una matrice (o vettore) in base alla sua posizione**. In altre parole, per "trovare" l'elemento non effettueremo una ricerca sequenziale o di altro tipo, ma accederemo all'elemento tramite i suoi **indici di riga e colonna** (vedi seguito).

Esempio: Rapporto peso-altezza degli studenti di una scuola

Una scuola vuole produrre statistiche sul **rapporto tra peso e altezza** dei suoi studenti. Per questo scopo utilizza una matrice `contaPA` (cioè conta Peso-Altezza) che ha **tante righe quanti sono i possibili pesi** degli studenti (da 0 a 199, per stare comodi) e **tante colonne quante sono le possibili altezze** (da 0 a 219). Ogni elemento della matrice è un **contatore** intero di tipo `short` che conta **quanti studenti corrispondono a quella combinazione di peso e altezza**.

```
const int MAXPESO = 200;                          // Ogni posizione rappresenta un chilogrammo
const int MAXALTEZZA = 220;                       // Ogni posizione rappresenta un centimetro

short[,] contaPA = new short[MAXPESO, MAXALTEZZA];    // 200 x 220 = 44000 elementi da 2 byte
```

La figura mostra come si accede al contatore che corrisponde agli studenti di 65 kg di peso e 171 cm di altezza (vedi seguito per un esempio di caricamento dei dati).



Note:

- Poiché la matrice contiene 44000 elementi e la maggioranza delle scuole non supera i 1000 studenti, dopo il caricamento dei dati, più di 43000 contatori conterranno ancora il valore iniziale zero.
- Nella dichiarazione della matrice mancano i contatori delle righe e colonne effettivamente usate; potrebbe tuttavia essere utile dichiarare quattro contatori per memorizzare peso e altezza minimi e massimi degli studenti in modo da eseguire la scansione della matrice ignorando le zone della matrice completamente vuote.
- La dimensione della matrice può essere ridotta considerando la bassa probabilità che uno studente pesi meno di 30 o più di 120 kg, o che sia più basso di 130 cm. La matrice avrebbe quindi 120–30 = 90 righe e 220–130= 90 colonne e sarebbe quindi di 8100 elementi: un risparmio dell'80%! Eventuali studenti con peso minore di 30 kg verrebbero quindi "contati" nella prima riga, che in realtà avrebbe il significato di "peso minore o uguale di 30 kg" e così via.

Caricamento

Il **caricamento di una matrice sparsa** generica di nome `m` richiede di effettuare il lookup degli elementi inseriti, come segue:

```
while («ci sono ancora dati da inserire»)
{
   i = «numero della riga»;
   j = «numero della colonna»;
   m[i,j] = «valore da inserire»;
}
```

Nell'esempio della matrice `contaPA`, caricare i dati significa incrementare uno degli elementi (contatori) ogni volta che vengono letti il peso e l'altezza di uno studente. Il ciclo (in cui mancano i necessari controlli) dovrebbe essere simile a questo:

```
str_peso = Interaction.Inputbox("Peso dello studente (Annulla per terminare)");
str_altezza = Interaction.Inputbox("Altezza dello studente (Annulla per terminare)");

while (str_peso != "" && str_altezza != "")
{
   peso = Convert.ToInt32(str_peso);
   altezza = Convert.ToInt32(str_altezza);

   contaPA[peso, altezza] = contaPA[peso, altezza] + 1;

   str_peso = Interaction.Inputbox("Peso dello studente (Annulla per terminare)");
   str_altezza = Interaction.Inputbox("Altezza dello studente (Annulla per terminare)");
}
```

Rimozione: Ha senso?

La rimozione (cancellazione) di dati da una matrice è un'altra operazione che, dopo un'attenta analisi, spesso sembra non avere senso.

Esempi:

- **distanze chilometriche tra coppie di città**: cancellarle significa che non esiste più collegamento diretto tra le due città (interruzione della strada?), occorre quindi inserire due zeri (o altri valori appropriati) nei due punti corrispondenti della matrice;
- **punteggi delle squadre di calcio nelle varie giornate**: potrebbe aver senso cancellare un'intera riga se una squadra viene squalificata per l'intero campionato (poco probabile) o un'intera colonna se i risultati di un'intera giornata vengono annullati. (Colpo di stato? Guerra atomica?)

Visualizzazione: DataGridView

Introduzione

La `DataGridView` è uno dei controlli più sofisticati di Visual Studio. Questo controllo permette di:

- visualizzare tabelle di dati e matrici;
- inserire, modificare e cancellare dati;
- associare `function` all'evento di inserimento e ad altri eventi;
- caricare automaticamente i dati nella `DataGridView` tramite una query SQL o un'altra sorgente di dati.

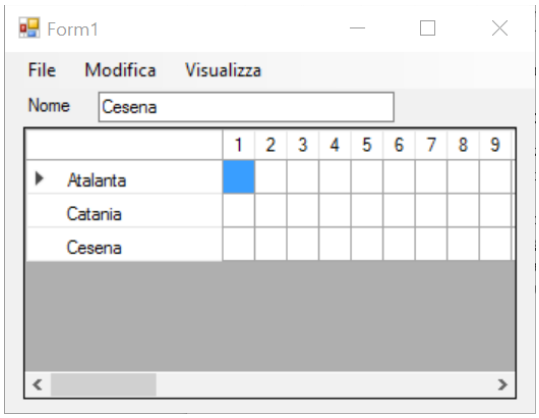
Gli esempi mostreranno solo una piccola parte delle sue funzionalità, per **visualizzare i risultati del campionato di calcio**.

Un semplice programma: campionato di calcio

E' possibile scrivere un semplice programma per memorizzare in una matrice e visualizzare in una `DataGridView` i punteggi ottenuti dalle squadre di calcio nel campionato di serie A. Il programma utilizza:

- un vettore `vs` per memorizzare i **nomi delle squadre** di calcio;
- una matrice `mpunteggi` per memorizzare i **punti delle squadre** nelle varie **giornate** di campionato, il cui numero è memorizzato nella variabile `ngiornate`;
- la `DataGridView` `dgPunteggi`, usata per visualizzare sia i nomi delle squadre che i punteggi.

Il form dell'applicazione contiene:



- il menu dei comandi, dove la scelta `Modifica` include i comandi `Inserisci squadra` e `Registra punteggi`;
- la `TextBox` per inserire i nomi delle squadre;
- la `DataGridView` con i **nomi delle squadre** inseriti nei **titoli delle righe**; i **numeri delle giornate** inseriti nei **titoli delle colonne** e gli **spazi per inserire i punteggi** delle varie giornate di campionato.

Impostazioni iniziali

Nella function dell'evento di caricamento del form `Form1_Load` inseriremo il seguente codice per semplificare (impoverire?) il funzionamento della `DataGridView`:

```
// Impedisci all'utente di aggiungere e cancellare righe
dgPunteggi.AllowUserToAddRows = false;
dgPunteggi.AllowUserToDeleteRows = false;

// Impedisci all'utente di modificare le celle
dgPunteggi.EditMode = DataGridViewEditMode.EditProgrammatically;

// Ridimensiona automaticamente le celle della griglia in base al loro contenuto
dgPunteggi.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader;

// Imposta la larghezza dei titoli delle righe a 130 pixel (basteranno per i nomi di tutte le squadre?)
dgPunteggi.RowHeadersWidth = 130;
```

Aggiungere le colonne e i numeri delle giornate di campionato

Il codice seguente può essere inserito in `Form1_Load` oppure nella function che inserisce i risultati della prima giornata.

```
// Aggiungi 38 colonne in un colpo solo!
dgPunteggi.ColumnCount = 38;

// Inserisci come titoli delle colonne i numeri delle giornate di campionato: 1, 2, 3, ... 38
for (i = 1; i <= 38; i = i + 1)
{
   dgPunteggi.Columns[i-1].HeaderText = i.ToString();       // Gli indici sono spostati di un'unità [i-1]
}
```

Aggiungere righe con i nomi delle squadre di calcio

La function che carica nel vettore tramite un ciclo i nomi delle squadre li visualizzerà nella `DataGridView` con le seguenti istruzioni:

```
// Aggiungi una riga - si può fare anche con dgPunteggi.Rows.Add()
dgPunteggi.RowCount = dgPunteggi.RowCount + 1;

// Inserisci il nome di una squadra come titolo della riga
dgPunteggi.Rows[dgPunteggi.RowCount-1].HeaderCell.Value = «nome squadra»;
```

Inserire i punteggi

Per inserire nella `DataGridView` il punteggio della **i-esima squadra** ottenuto nella **n-esima giornata** di campionato, scriveremo:

```
dgPunteggi.Rows[i].Cells[ngiornate].Value = «punteggio»;
```

Proteggere righe o colonne dalla scrittura

Se non avessimo protetto tutti i dati fin dall'inizio contro le modifiche dell'utente, potremmo proteggere le colonne delle giornate di campionato in cui sono già stati inseriti tutti i dati, come segue:

```
// Sostituendo Columns con Rows (e ngiornate con un altro valore) si può proteggere una riga
dgPunteggi.Columns[ngiornate].ReadOnly = true;
```