Università degli Studi di Milano

Laurea Specialistica in Genomica Funzionale e Bioinformatica

Array e matrici

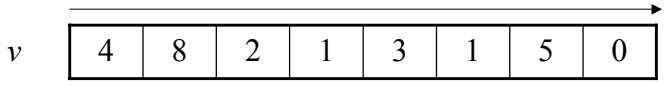
Giorgio Valentini

e –mail: *valentini@dsi.unimi.it*

DSI – Dipartimento di Scienze dell' Informazione
Università degli Studi di Milano

Array e matrici come generalizzazioni multidimensionali di vettori (1)

- I vettori sono sequenze ordinate di elementi omogenei. Un vettore *v* è rappresentabile tramite una *struttura unidimensionale*:



- Essendo strutture unidimensionali è possibile accedere o modificare un elemento di un vettore utilizzando un *unico indice*:

```
> v[2]
8
> v[1] <- 7
> v[1]
7
```

Array e matrici come generalizzazioni multidimensionali di vettori (2)

- In R è possibile rappresentare *estensioni bidimensionali* di vettori (matrici)

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

- Essendo strutture bidimensionali, è necessaria una *coppia di indici* per accedere o modificare un elemento di una matrice :

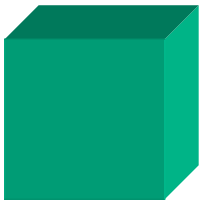
```
> m[1,3] # seleziona el. I
         # riga e III colonna
> 6
> m[2,4] <- 0
```

3

Array e matrici come generalizzazioni multidimensionali di vettori (3)

- In generale in R è possibile rappresentare *estensioni multidimensionali* di vettori (array):

Es: array tridimensionale



Per accedere ad un elemento sono necessari 3 indici.

Es:

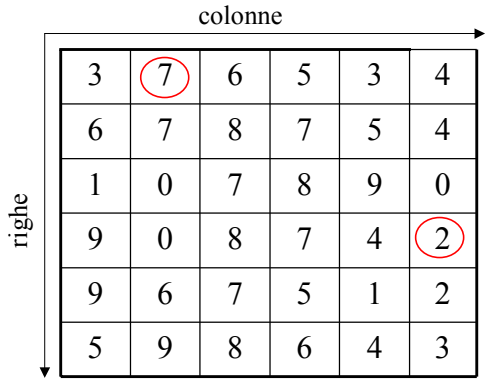
```
> a[1,3,4]
```

- In R si possono costruire array di dimensione arbitraria (limitatamente alle disponibilità di memoria)
- Le matrici sono a tutti gli effetti array bidimensionali
- Sugli array sono applicabili le medesime operazioni di accesso e modifica (estese a più dimensioni) utilizzabili per i vettori

4

Matrici

- In R le matrici sono array bidimensionali:



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

- Gli elementi di una matrice sono selezionati tramite una coppia di indici racchiusi fra parentesi quadre:

- `m[1,2]` (m variabile cui è associata la matrice) identifica l' elemento della riga 1 e colonna 2

- `m[4,6]` identifica l' elemento della riga 4 e colonna 6

5

Costruzione di matrici (1)

Le matrici possono essere costruite tramite la funzione matrix a partire da un vettore esistente

```
> x <- 1:24
> m <- matrix (x, nrow=4) # genera una matrice con 4
# righe utilizzando gli elementi del vettore x
> m  # si noti l' assegnamento dei valori "per colonne"
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> length(m)
[1] 24
> mode(m)
[1] "numeric"
> dim(m)
[1] 4 6
```

L' attributo dim mostra che la matrice *m* è un array bidimensionale 4 X 6

6

Costruzione di matrici (2)

La funzione matrix può avere anche altri argomenti:

```
> x <- 1:12
> m <- matrix (x, ncol=4) # si può specificare il n. delle colonne
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m <- matrix (x, ncol=4, byrow=TRUE) # inserimento el. per righe
> m
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> m <- matrix (x, ncol=5)  # anche per le matrici si applica
                           # la "regola del riciclo"
Warning message:
data length [12] is not a sub-multiple or multiple of the number of
columns [5] in matrix
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10    1
[2,]    2    5    8   11    2
[3,]    3    6    9   12    3
```

7

Le funzioni cbind e rbind

Le matrici possono essere costruite anche tramite le funzioni cbind ed rbind.

cbind forma matrici legando insieme vettori o matrici "orizzontalmente":

```
> x <- 1:3
> y <- 4:6
> m <- cbind(x,y)
> m
     x y
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

rbind forma matrici legando insieme vettori o matrici "verticalmente":

```
> x <- 1:3
> y <- 4:6
> m <- rbind(x,y)
> m
  [,1] [,2] [,3]
x    1    2    3
y    4    5    6
```

Se i vettori costituenti non sono della stessa lunghezza si applica la regola del riciclo

8

Costruzione di matrici con cbind per "giustapposizione" di matrici

```
> x<-1:12
> y<-13:24
> m1<-matrix(x,nrow=3)
> m2<-matrix(y,nrow=3)
> m <- cbind(m1,m2)
> m
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    4    7   10   13   16   19   22
[2,]    2    5    8   11   14   17   20   23
[3,]    3    6    9   12   15   18   21   24
> m2<-matrix(y,nrow=2)
> m <- cbind(m1,m2)
Error in cbind(...) : number of rows of matrices must
   match (see arg 2)
```

9

Operazioni con le matrici (1)

Somma e prodotto con costanti

```
> x <- 1:12
> A <- matrix(x,nrow=3)
> A + 2
     [,1] [,2] [,3] [,4]
[1,]    3    6    9   12
[2,]    4    7   10   13
[3,]    5    8   11   14
> B <- A * 2
> B
     [,1] [,2] [,3] [,4]
[1,]    2    8   14   20
[2,]    4   10   16   22
[3,]    6   12   18   24
```

Somma e prodotto "elemento per elemento"

```
> A+B
     [,1] [,2] [,3] [,4]
[1,]    3   12   21   30
[2,]    6   15   24   33
[3,]    9   18   27   36
> A*B
     [,1] [,2] [,3] [,4]
[1,]    2   32   98  200
[2,]    8   50  128  242
[3,]   18   72  162  288
```

10

Operazioni con le matrici (2)

Prodotto di matrici "righe X colonne": Operatore `%*%`

```
Si ricordi che A e B sono
matrici 3X4:
> A%*%B # il numero delle
# colonne di A deve essere
# uguale al numero di righe di
# B!
Error in A %*% B : non-
conformable arguments
> A%*%t(B) # t(B) indica la
           # trasposta di B
     [,1] [,2] [,3]
[1,]  332  376  420
[2,]  376  428  480
[3,]  420  480  540
```

Prodotto di matrici per vettori:

```
> z <- 1:4
> A%*%z
     [,1]
[1,]   70
[2,]   80
[3,]   90
> z%*%A
Error in z %*% A : non-
conformable arguments
> z%*%t(A)
     [,1] [,2] [,3]
[1,]   70   80   90
```

11

Altri operatori e funzioni per le matrici

In R esiste un repertorio amplissimo di operatori e funzioni specifiche per le matrici (si veda la documentazione):

- Ad es: la funzione *diag*, che ha un significato diverso a seconda che il suo argomento sia un intero, un vettore ed una matrice (si provi).
- La funzione *solve* consente di invertire una matrice e può essere usata per risolvere sistemi lineari. Ad es, se `A` è una matrice quadrata non singolare e `b` un vettore compatibile con `A`, allora `x<-solve(A,b)` risolve il sistema lineare `Ax=b`. (si provi ...)
- Esistono inoltre funzioni per il calcolo degli autovalori ed autovettori di matrici simmetriche, per il calcolo della svd di una matrice e molte altre.

12

Array

- Gli array sono generalizzazioni multidimensionali di vettori
- Consentono di costruire strutture complesse in più dimensioni
- Informalmente si possono pensare come sequenze ordinate di sottostrutture di dimensione inferiore. Ad es: una matrice può essere vista come una sequenza di elementi, di cui ognuno è un vettore; un array tridimensionale come una sequenza di matrici; un array 4-D come una sequenza di arrray 3-D.
- Si noti comunque che non esiste una gerarchia fra le dimensioni: si può accedere ad una sottostruttura scegliendo come "principale" una qualsiasi delle dimensioni.

13

Costruzione di array

La costruzione di array avviene tramite la funzione array:

Es. 1 (array bidimensionale):

```
> z <- array(1:6, c(2,3))
> z
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

L' array bidimensionale `z` è una matrice che si sarebbe potuto costruire equivalentemente nel modo seguente:

```
> z <- matrix(1:6,nrow=2)
> z
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Es. 2 (array tridimensionale)

```
> z <- array(1:24, c(2,3,4))
> z # array tridimensionale
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
, , 3
     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18
, , 4
     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

14

Sintassi della funzione array

array (*vettore di dati, vettore dimensioni, vettore del nome delle dimensioni*)

Vettore dati : un qualsiasi vettore di dati che viene utilizzato per "riempire" l' array

Vettore dimensioni: attributo *dim* dell' array, cioè un vettore di lunghezza pari al numero delle dimensioni dell' array che fornisce l' indice massimo per ogni dimensione

Vettore del nome delle dimensioni: attributo *dimnames* dell' array, cioè lista che fornisce un nome (stringa di caratteri) alle diverse dimensioni.

15

Esempi di array (1)

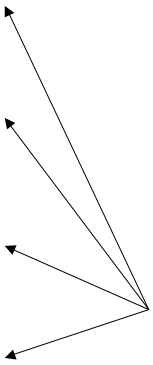
```
> x<-1:12  # vettore dei dati per l' array
> d<-c(2,6) # vettore delle dimensioni dell' array (attr.dim)
> z<-array(x,d) # costruzione dell' array: il III argomento
                #  dimnames è assente e per default è NULL
> z   # array bidimensionale 2X6
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> x # x è un vettore
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> dim(x)  # come vettore non possiede l' attributo dim
[1] NULL
> dim(x)<-d  # si può assegnare una dimensione ad un vettore ...
> x  # trasformandolo in un array bidimensionale
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> is.array(x) # la funzione is.array conferma che ora x è un array
[1] TRUE
> dim(x)<-NULL # cancellazione dell' attributo dim
> x  # x torna ad essere un vettore
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> is.array(x)
[1] FALSE
> is.vector(x)
[1] TRUE
```

16

Esempi di array (2)

Si noti come vengano inseriti gli elementi nell' array utilizzando l' indice più "a sinistra":

```
> z <- array(1:24, c(2,3,4))
> z # array tridimensionale
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
, , 3
     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18
, , 4
     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```



Essendo un array tridimensionale sono necessari 3 indici per accedere agli elementi: z[x,y,w]

Il primo elemento viene inserito in posizione z[1,1,1]; il secondo in z[2,1,1] (si muove per primo l' indice più a sinistra). Quindi si passa al II indice: z[1,2,1] e z[2,2,1].

Riempita la I matrice 2X3 con i primi 6 elementi,si muove finalmente il III indice, accedendo al I elemento della II matrice 2X6: z[1,1,2]

Quindi l' indice più a sinistra è quello che si "muove più velocemente", mentre quello più a destra corrisponde a quello più lento.

Un array tridimensionale 2X3X4 si può quindi pensare come l' "impilamento" di 4 matrici 2X3.

17

Accesso agli elementi di array e matrici

Le regole di accesso per array e matrici seguono quelle già viste per i vettori, considerando però l' esistenza di più indici e quindi la possibilità di utilizzare un vettore per ogni dimensione:

- Vettori di indici interi positivi
- Vettore di indici interi negativi
- Vettore di indici logici
- Vettori di indici a caratteri

Si utilizza quindi un vettore di indici *per ogni dimensione* dell' array

18

Esempi di accesso agli elementi di una matrice

```
> m <- matrix (1:12,nrow=2)
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> m[1,2]
[1] 3
> m [1,3:4] # el. I riga,col. 3 e 4
[1] 5 7
> m [1:2,4:6] # I e II riga, col.
              # dalla IV alla VI
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
> m[1,] #tutti gli el. della I riga
[1]  1  3  5  7  9 11
> m[,3] #tutti gli el. III col.
[1] 5 6
> m[,c(1,2,6)]
     [,1] [,2] [,3]
[1,]    1    3   11
[2,]    2    4   12
```

```
> m[,-4] # esclusione el. IV
         # colonna
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    9   11
[2,]    2    4    6   10   12
> m[m>4] # el.>4 (si ottiene un
         # vettore)
[1]  5  6  7  8  9 10 11 12

> dimnames(m)<list(c("r1","r2"),
paste("c",1:6,sep="")) # viene
# dato un nome alle componenti
> m
   c1 c2 c3 c4 c5 c6
r1  1  3  5  7  9 11
r2  2  4  6  8 10 12
> m["r1","c2"] # vettori indice
               # a caratteri
[1] 3
> m["r1",]
c1 c2 c3 c4 c5 c6
 1  3  5  7  9 11
```

19

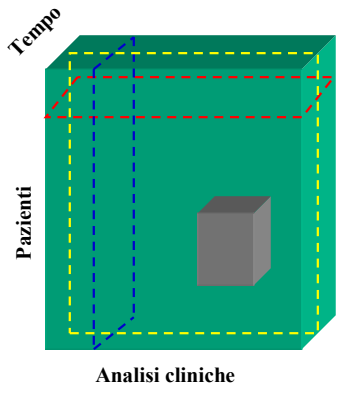
Esempi di accesso agli elementi di un array

Si consideri l' array z della slide 17:

```
> z <- array(1:24, c(2,3,4))
```

Si può accedere a "fettine" della struttura tridimensionale da ognuna delle 3 dimensioni, ottenendo 3 diversi array bidimensionali (matrici):

```
> z[1,,]
     [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    3    9   15   21
[3,]    5   11   17   23
> z[,1,]
     [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    2    8   14   20
> z[,,1]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```



E' possibile accedere ad altre sottoparti della struttura 3D, analogamente a quanto visto nel caso bidimensionale (matrici)

20

Operazioni con gli array

Le operazioni sono effettuate elemento per elemento ed il risultato è un array il cui attributo *dim* è lo stesso degli operandi.

Es:

```
> x<-1:12
> y<-x*2
> z1<-array(x,c(2,3,2))
> z2<-array(y,c(2,3,2))
> z3 <- z1*z2 # molt. el. per el.
> z1+z2 # somma el. per el.
, , 1
     [,1] [,2] [,3]
[1,]    3    9   15
[2,]    6   12   18
, , 2
     [,1] [,2] [,3]
[1,]   21   27   33
[2,]   24   30   36
```

```
> z1/z2 # div. el. per el.
, , 1
     [,1] [,2] [,3]
[1,]  0.5  0.5  0.5
[2,]  0.5  0.5  0.5
, , 2
     [,1] [,2] [,3]
[1,]  0.5  0.5  0.5
[2,]  0.5  0.5  0.5
```

Gli operandi devono avere il medesimo attributo dim:

```
> z3<-array(y,c(2,3,4))
> z1-z3
Error in z1 - z3 : non-conformable
arrays
```

21

Esercizi

1. Costruire una matrice 10X10 composta da numeri casuali in almeno 2 modi diversi utilizzando le funzioni matrix ed array.
2. Costruire 2 matrici di caratteri a piacere *x* e *y*, la prima di dimensione 3X4, la seconda di dimensione 5X3. Modificare con un unico assegnamento la matrice *x* in modo da sostituire le sue 2 prime colonne con le ultime 2 righe di *y*.
3. Costuire un array *a* 3X4X5 costituito dai primi 60 numeri naturali positivi. Moltiplicarlo per un array della medesima dimensione ma composto da numeri casuali. E' possibile moltiplicare l' array *a* per un vettore numerico di lunghezza 60? Se si, cosa si ottiene? Modificare infine l' array a in modo da ottenere un array 4-D 3X5X2X2. Verificare infine gli attributi di a utilizzando funzioni opportune.
4. Costruire due matrici *M* ed *N* che abbiano entrambe 5 colonne. Costruire, se possibile, tramite *rbind* una matrice di 5 colonne che abbia come righe le righe di entrambe le matrici. Utilizzando *M* ed *N*, è possibile costruire una matrice tramite *cbind*?
5. Date du matrici quadrate *A* e *B* di dimensione 3X3 costituite da numeri casuali, calcolarne il prodotto elemento per elemento e "righe per colonne". Scelto un vettore *b* di 3 elementi, risolvere il sistema lineare $Ax=b$ che ne deriva.
6. Quale struttura dati si potrebbe utilizzare per modellare un insieme di dati numerici relativi a diverse tipologie di analisi per un insieme di diversi pazienti? Se tali dati dovessero essere rilevati ad intervalli di tempo differenti, quali altre struttura dati si potrebbero utilizzare?

22