a cura di Andrea de Prisco

Le strutture informative

di Anna Pugliese

$E = mc^2$.
Niente male come inizio di questo numero di «Appunti di Informatica»! Perché l'abbiamo tirata in ballo? È presto detto. Questa equazione ha sconvolto il mondo nella prima metà del secolo. È successo qualcosa di analogo nella seconda metà: un'altra equazione ha portato lo scompiglio nel nostro pianeta. Forse meno nobile della prima, ma sicuramente ugualmente efficace: Algoritmi + Strutture Dati = Programmi.
Se programmare è un'arte, questa consiste nell'arte di trovare algoritmi e nell'arte di utilizzare strutture dati adeguate allo scopo; su quest'ultima vogliamo posare la nostra attenzione nel corso di questa rubrica. L'obiettivo che ci prefiggiamo è quello di non riuscire più a capire che differenza c'è tra sommare due numeri e bilanciare un albero. Forse non basterà un solo appuntamento per raggiungerlo ma è inutile guardare lontano, procediamo con ordine

L'organizzazione della memoria

Tranne in un caso (il file), quando si parla di strutture dati ci si vuole riferire al modo in cui l'informazione elaborata da un calcolatore è organizzata per poter essere mantenuta, ed elaborata nel modo più efficiente, su unità di memoria principale (RAM).

La RAM di un computer, è costituita da un insieme di celle di memoria residenti ad indirizzi numerici consecutivi.

Ogni cella di memoria può contenere un byte di informazione, e questo contenuto può essere letto o sovrascritto dal processore, indicando l'indirizzo cui la cella contenitrice risiede.

Il contenuto di una cella di memoria può rappresentare un'intera informazione oppure parte di essa, o anche più informazioni messe insieme.

Ne consegue che non è possibile capire quale informazione è contenuta in un pezzo di memoria, a partire dalla semplice osservazione dei dati in esso memorizzati.

Dietro questa affermazione si cela il concetto di tipo di dato.

Con semplici parole questo concetto può essere espresso nel seguente modo: la codifica di una informazione in memoria, non basta per conoscere il valore del dato, è necessario conoscere il tipo di informazione con cui si ha a che fare, cioè il tipo del dato.

Esempi di tipi di dato sono: gli interi, i reali, le stringhe e i booleani; e questi sono solo i più diffusi, perché di tipi di dato se ne possono trovare infiniti.

Le strutture dati, che in linea di principio sono del tutto indipendenti dal tipo di dato cui appartengono i loro elementi, finiscono poi con l'essere costrutti mediante i quali è possibile generare nuovi tipi di dato, dando così vita ai vettori di interi ed ai vettori di stringhe, alle liste di booleani ed ai grafi di numeri reali, senza parlare poi degli array di record composti da liste di caratteri e chi più ne ha più ne metta.

Ricominciamo ora daccapo.

Numeri non basta

Per chi, fra i lettori, ha una certa familiarità con la logica della rappresentazione binaria, il parlare dei metodi per memorizzare informazioni di tipo numerico può costituire un argomento fin troppo banale.

Non è raro, tuttavia, che un errato approccio a questo problema costituisca il motivo per cui la memorizzazione di informazioni di tipo diverso, presenta poi fin troppi lati oscuri.

Un byte, si sa, è una sequenza di 8 bit; con 8 bit è possibile ottenere 256 diverse configurazioni, da quella con otto zeri a quella con otto uno, per cui, associando ad ogni configurazione un diverso valore numerico è possibile rappresentare 256 numeri diversi; ad esempio: 14,1; 14,2; 14,3; 14,4; 14,5; 14,6; 14,7; 14,8; 14,9; 15,0;... 39,6.

Certo, se qualcuno insiste possiamo decidere di associare alle 256 configurazioni i numeri che vanno da 0 a 255, oppure quelli che vanno da −127 a +128, ma essa costituirà solo una delle possibili scelte, e vi assicuro che nessun computer è in grado di capire che il contenuto di una delle sue celle di memoria che contiene il dato 00000000, rappresenta il numero zero, e non solo perché un computer non è in grado di capire, ma perché non è detto che sia così. E mi spiego.

Scegliamo di rappresentare i primi 256 numeri interi con le configurazioni che vanno da 00000000 a 11111111. Come rappresenteremo il numero 536?

Osserviamo la figura 1. Essa mostra proprio la soluzione al nostro quesito adottata nella maggior parte dei casi.

A prima vista le due celle di memoria 156 e 157 della figura 1, sembrano contenere rispettivamente i numeri 24 e 2, ma questa cattiva interpretazione è dovuta al fatto che non si è tenuto conto del tipo del dato memorizzato in queste due celle.

Quando si parla di numeri, in Informatica, non basta conoscerne il valore, occorre sapere qual è il tipo.

Il numero 536 memorizzato come in figura 1, è di tipo intero.

Per numero intero intendiamo qui, come spesso accade, un numero naturale compreso tra −32767 e +32768; il fatto che si sia scelto questo intervallo è una conseguenza del fatto che si vogliono utilizzare due soli byte per la sua memorizzazione, e con due byte si possono ottenere 65536 possibili diverse configurazioni binarie.

Inoltre si è scelto di memorizzare l'informazione «numero intero 536» strutturandola in modo che la prima cella di memoria contenga gli 8 bit meno significativi della sua rappresentazione binaria, e la cella successiva i rimanenti 8 bit più significativi.

Il contenuto delle due celle in figura 1, va perciò interpretato come il numero binario 0000001000011000, che corrisponde al decimale 536.

Diverso è il discorso per i numeri che vanno da 0 a 255; questi numeri possono essere considerati, oltre che di tipo intero, di tipo «Byte» (come accade nel Pascal) ed in tal caso vengono memorizzati, come dice il loro nome, mediante l'utilizzo di una sola cella di memoria.

Prima di concludere il discorso sui numeri, consideriamo l'operazione PERDUE(X)=2X. Se applichiamo questa operazione al dato memorizzato all'indirizzo 156 della Main Store in figura 1, otterremo come risultato 1072 oppure 48?

Ancora una volta l'ambiguità nasce dal fatto che non abbiamo detto qual è il tipo di dato che costituisce l'operando di PERDUE, o, per dirlo meglio, qual è il tipo di dato cui appartiene l'operazione PERDUE.

Ebbene sì, finalmente lo abbiamo detto.

Un tipo di dato è caratterizzato non solo dall'insieme dei suoi valori, ma anche dall'insieme delle operazioni su di esso eseguibili.

Ne deriva che l'operazione PERDUE di tipo intero è completamente diversa dall'operazione PERDUE di tipo byte.

Le operazioni, e non solo quelle aritmetiche, devono conoscere il modo in cui i loro operandi sono memorizzati.

A volte esse conoscono completamente i loro dati, altre volte esse sono in grado di operare su dati di classi diverse appartenenti comunque allo stesso tipo, vale a dire che si comportano, nel corso della loro esecuzione, in maniera dipendente da alcune informazioni associate ai dati cui esse sono applicate.

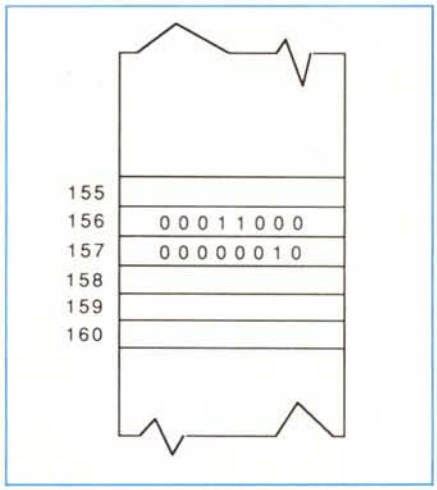

Figura 1 - Il numero intero 536, memorizzato all'indirizzo 156 della RAM di un computer.

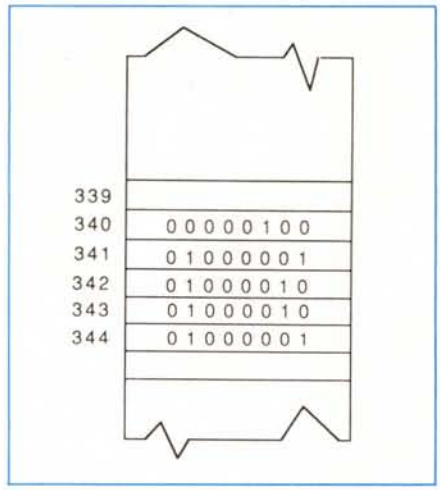

Figura 2 - La stringa "ABBA" memorizzata all'indirizzo 340 della RAM di un computer.

È il caso del tipo di dato che andremo a vedere.

Stringhe: tipi o strutture dati?

Per tipo di «stringa», si intende una sequenza di simboli appartenenti ad un certo alfabeto; questo alfabeto è solitamente l'intero alfabeto del linguaggio, per cui esempi di dato di tipo stringa, sono: «Mario Rossi», «AZ15», «+A,cl?!$36» e così via.

Le operazioni che possono essere eseguite su questo tipo di dato sono: il calcolo della lunghezza di una stringa, la concatenazione di due stringhe, l'estrazione di sottostringhe ed altre.

Una sequenza di caratteri rappresentante una stringa, ha generalmente una lunghezza compresa tra 0 e 255.

Questo limite è motivato dal modo in cui le stringhe sono rappresentate in memoria.

I caratteri di cui una stringa è composta, si sa, sono rappresentati dalla loro codifica ASCII, e poiché i codici ASCII sono 256 in tutto, ognuno di essi necessita di un solo byte di memoria.

La memorizzazione di una stringa lunga L, può dunque essere realizzata utilizzando L+1 celle di memoria consecutive, la prima delle quali esprime la lunghezza reale della stringa e le successive contengono i codici ASCII dei caratteri da cui essa è composta.

Consideriamo allora la stringa «ABBA», ed assunto che ASCII («A»)=65 e che ASCII («B»)=66, avremo che la rappresentazione in memoria di «ABBA», a partire dalla locazione di memoria avente indirizzo 340, avverrà come descritto in figura 2.

Quando una qualsiasi delle operazioni permesse su questo tipo di dato, è mandata in esecuzione su questa informazione, ciò che di tale informazione gli viene trasmesso è solo l'indirizzo di partenza.

L'operazione prima di svolgere la sua funzione essenziale, dovrà preoccuparsi di comprendere fino in fondo con che tipo di dato ha a che fare; in altri termini essa dovrà innanzitutto leggere il dato dalla cella 340 interpretandolo come un'informazione di tipo byte che

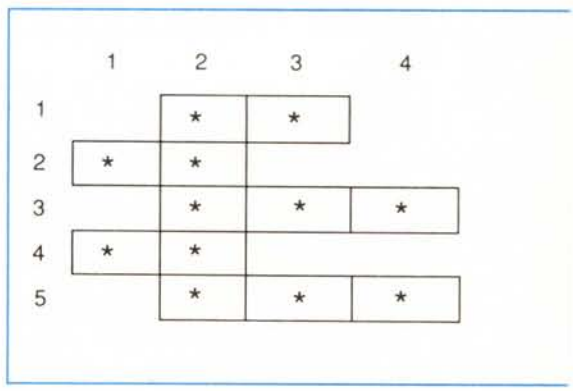

Figura 3 - Una matrice non rettangolare, con range di indici:
I1 compreso tra 1 e 5 ed
I2 compreso tra 2 e 3 per I1 = 1,
I2 compreso tra 1 e 2 per I1 = 2,
I2 compreso tra 2 e 4 per I1 = 3,
I2 compreso tra 1 e 2 per I1 = 4,
I2 compreso tra 2 e 4 per I1 = 5.

rappresenta il numero di caratteri di cui la stringa è composta. In tal modo, ad esempio, una cella di memoria contenente 00000000 può rappresentare una stringa vuota.

La STRINGA, intesa nel modo sopra descritto, può tranquillamente essere considerata, più che un tipo di dato, una struttura dati molto semplice, nel senso che i suoi elementi devono necessariamente appartenere ad un solo tipo di dato: il carattere, noto (in Pascal) con il nome «Char».

La strategia di memorizzazione di una stringa lunga L in L+1 celle di memoria consecutive ha, come ogni strategia, i suoi pregi e i suoi difetti.

L'alternativa consiste nell'utilizzare comunque un numero di celle sufficienti per contenere una stringa della massima lunghezza prevista.

Sebbene questa soluzione possa apparire inefficiente, essa è spesso adottata perché semplifica una certa quantità di problemi di diversa natura, non ultimo quello di poter utilizzare le stringhe come il tipo cui appartengono gli elementi di alcune strutture dati che per poter essere agevolmente utilizzate, presuppongono un uniforme dimensionamento dei loro elementi.

Uno dei casi più diffusi è quello che andremo a vedere.

Vettori

Il vettore, così come la stringa, è una struttura dati che permette la sequenzializzazione di elementi, questa volta di tipo qualsiasi, purché uguali tra loro.

In realtà i vettori sono molto diversi dalle stringhe, sebbene questa diversità traspaia soprattutto nell'implementazione.

Gli elementi di un vettore devono infatti essere direttamente indirizzabili, cioè elaborabili in maniera indipendente dagli altri.

Questo, in taluni linguaggi di programmazione è un requisito non soddisfatto dalle stringhe, e quantunque in altri possa invece esserlo, la cosa è in realtà simulata in quanto le procedure interne che elaborano le stringhe, operano sempre sulla stringa per intero anche se chiamate ad eseguire operazioni su parti di essa.

I vettori, di contro, sono praticamente agglomerati di variabili dello stesso tipo, che sebbene abbiano un indirizzamento leggermente più complesso, possono poi essere utilizzate alla stregua di normali variabili di quel tipo.

L'organizzazione dei vettori in memoria, prevede l'utilizzazione di un certo numero di celle consecutive destinate a contenere gli elementi consecutivi del vettore. Se gli elementi di un vettore richiedono d celle ciascuno allora i loro indirizzi in memoria avranno distanza d gli uni dagli altri.

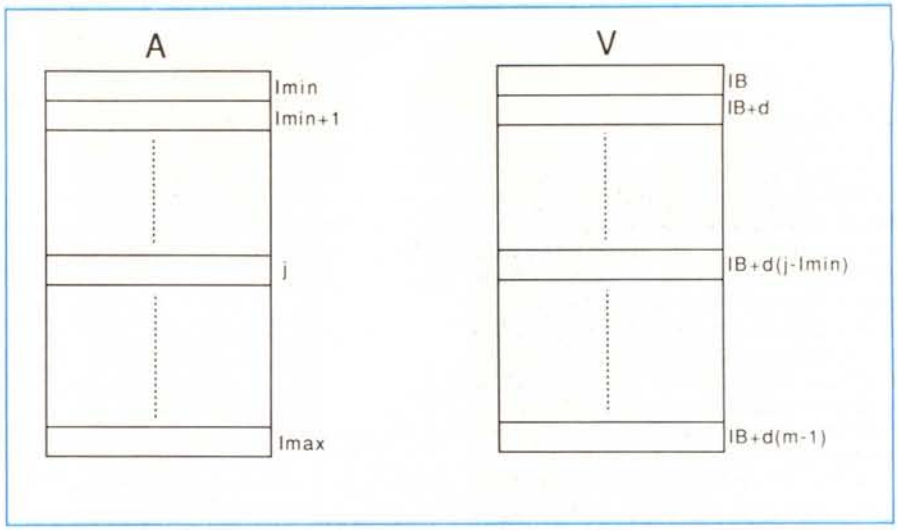

Figura 4 - Allocazione di una matrice unidimensionale in un vettore.

Tutto quello che ci interessa per caratterizzare completamente la posizione in memoria di un vettore V, sono tre parametri:

1) l'indirizzo IB del primo elemento (detto indirizzo base);
2) il numero m dei suoi elementi (detto lunghezza del vettore);
3) la dimensione (occupazione di memoria) di ogni suo elemento, d.

Per esprimere completamente la presenza in memoria di un vettore, possiamo dunque scrivere V(IB,m,d).

Il generico elemento di un vettore può essere indicato con V(j) dove j=1,2,3, ..., m

Per realizzare l'accesso diretto a tale elemento, occorre trovare l'indirizzo iniziale cui l'elemento risiede, esprimendolo in funzione dei parametri del vettore.

In tal modo ogni riferimento ad un particolare elemento del vettore potrà essere sostituito con la formula che, in funzione della sua posizione relativa all'interno del vettore, ne esprime l'indirizzo assoluto in memoria.

Venendo al dunque, la formula che esprime l'indirizzo base di V(j) è IBj=IB +d(j−1).

La prima osservazione da fare è che la lunghezza m, assieme alla dimensione d, dev'essere fissata a priori.

Una volta allocato un vettore in memoria, i suoi parametri non possono subire successive modificazioni.

Ne consegue che il vettore è una struttura dati rigida, la qual cosa è giustificata dalla vicinanza esistente tra la sua organizzazione logica e la sua effettiva implementazione fisica.

Le strutture dati per cui questo è vero, sono solitamente dette «strutture interne», contrapposte alle cosiddette «strutture astratte» in cui l'organizzazione logica prescinde quasi totalmente da quella fisica; è il caso, come vedremo, di grafi, alberi, e matrici.

Per concludere il discorso sulla memorizzazione dei vettori, dobbiamo notare una cosa non del tutto semplice, che ci serve per poter rispondere alla seguente domanda: i parametri, m e d, di un vettore V, vanno memorizzati assieme ad esso, come avveniva per la lunghezza delle stringhe, oppure no? La risposta è no, nella maggior parte dei casi.

Prescindiamo allora dai casi particolari e vediamo perché.

La migliore definizione che possiamo dare di un vettore, dal punto di vista della sua struttura astratta, è: un agglomerato di variabili.

Questo significa che l'unica operazione che può essere eseguita su un tipo di dato «vettore» è l'indirizzamento di uno dei suoi elementi; tutte le altre coinvolgono il tipo dei suoi elementi e non il vettore in se stesso.

Ora, l'indirizzamento di elementi in memoria è solo un problema di compilazione, da risolvere quindi staticamente, mediante la sostituzione dell'indirizzo fisico a quello logico nel caso delle variabili, mediante l'applicazione della formula corrispondente nel caso di un vettore.

Se ne ricava che i parametri m e d di un vettore, assieme al suo indirizzo base IB, sono i termini della formula IB-+d(j−1) da applicare per ottenere l'accesso all'elemento V(j), e che ogni riferimento a tale elemento viene risolto a tempo di compilazione ed espresso in funzione del solo j, che è l'unico parametro noto solo a tempo di esecuzione.

Questo discorso cade nel caso in cui i vettori sono utilizzati per operazioni

diverse dal singolo indirizzamento. Il discorso potrebbe essere approfondito dai lettori interessati, simulando la gestione delle strutture informative mediante l'utilizzazione di un grosso vettore, che potrebbe rappresentare la memoria principale, ed implementando autonomamente le politiche di allocazione e gestione delle strutture stesse.

Questa simulazione, che per ora potrebbe sembrare inutile e noiosa, costituisce una buona base per essere poi in grado di gestire strutture complesse, comprendendo fino in fondo i concetti che stanno alla base della gestione delle strutture informative, e, cosa assai importante, rendendo in grado di simulare qualunque struttura dati di cui il nostro linguaggio di programmazione potrebbe essere sprovvisto.

Le matrici

L'ultima tappa di questa prima parte del nostro viaggio attraverso le strutture informative, riguarda la struttura astratta detta «MATRICE».

Sebbene questa, unitamente al vettore, venga talvolta designata con il nome «array», la nostra scelta di trattare separatamente le due, è giustificata dalla profonda differenza concettuale esistente fra le due strutture.

La più generale definizione di matrice consiste nel vederla come un insieme di elementi, ciascuno dei quali è individuato dai valori (interi) assunti da un gruppo di indici $I_1, I_2, ..., I_n$.

In linea di principio, ciascun indice I_k, assume valori appartenenti ad un insieme RANGE (I_k) e tali valori sono funzione dei valori assunti dagli altri indici.

Un esempio di matrice cosiffatta è riportato in figura 3.

La gestione di matrici come questa, è abbordabile implementativamente a condizione che non ci siano più di due indici.

Particolare importanza rivestono le «matrici rettangolari», caratterizzate dal fatto che i valori assunti da ciascun indice sono gli interi compresi in un intervallo chiuso che non dipende dai valori degli altri indici.

La definizione generale di una matrice rettangolare ha la forma seguente: $A(I_{1,min}:I_{1,max};\ I_{2,min}:I_{2,max};\ ...;\ I_{n,min}:I_{n,max})$.

Il generico elemento di una matrice A avente n gruppi di indici, si indica con

$$A(i_1,i_2,...,i_n),$$

e si suol dire che la matrice è «ad n dimensioni».

Vediamo ora come si realizza l'allocazione in memoria di matrici rettangolari.

La struttura dati «matrice» viene allocata in memoria impiegando sempre la struttura interna «vettore»; questo ne giustifica la sua maggiore astrazione rispetto a quest'ultima.

Gli elementi della matrice saranno singolarmente contenuti in elementi di un vettore, per il quale dovrà essere stabilita un'opportuna dimensione d.

Non può essere che:

$$d = \prod_{i=1}^{n} (I_{i,max} - I_{i,min} + 1)$$

Se la matrice A in questione è una matrice unidimensionale, la corrispondenza tra i suoi elementi e quelli del vettore è immediata, come si evince dall'osservazione della figura 4.

alla stessa figura è possibile notare che l'indirizzo fisico dell'elemento A(j), se A è definita come

$$A(I_{min}:I_{max})$$

è: $IB+d(j-I_{min})$,

e che la dimensione di V è:

$$m = I_{max} - I_{min} + 1.$$

Per quanto riguarda le matrici ad n dimensioni, il discorso è chiaramente più complesso.

Facciamo un esempio sulla base di una matrice bidimensionale

$$A(1:4\ ;\ 1:4);$$

ne approfittiamo per dire che le matrici aventi gli indici appartenenti ad insiemi di uguale cardinalità, caso in cui ricade la nostra, sono dette matrici «quadrate», la cosa tuttavia non riveste alcuna importanza nel nostro caso.

L'idea è quella di cominciare l'allocazione della matrice, nel vettore interno, partendo da una delle dimensioni per poi continuare con le altre.

Alla base di ciò, risiede il fatto che fissati n−1 indici in una matrice ad n dimensioni, l'insieme degli elementi restanti è un vettore.

Applicando questa idea alla nostra matrice A, scegliendo di fissare il secondo indice, avremo il vettore composto da:

A(1,1) A(2,1) A(3,1) A(4,1).

Con un nuovo valore al secondo indice, avremo ancora:

A(1,2) A(2,2) A(3,2) A(4,2), poi:

A(1,3) A(2,3) A(3,3) A(4,3), ed infine:

A(1,4) A(2,4) A(3,4) A(4,4).

La *figura 5* riporta il pezzo di memoria in cui è allocata A.

Vediamo allora come si indirizzano i singoli elementi di A, a partire dall'indirizzo base di V, IB, e dalla dimensione d degli elementi di A.

Innanzitutto il vettore V va dimensionato alla lunghezza

$$m = (4-1+1)\ (4-1+1) = 16$$

che è evidentemente il numero di elementi della matrice.

Osserviamo allora che gli elementi di A aventi il secondo indice al valore 1, occupano le prime 4 posizioni di V. Quindi l'indirizzo fisico di A(i,1) è IB+d(i−1). Il discorso è analogo per il secondo gruppo di elementi di A, quelli cioè aventi il secondo indice pari a 2, purché si tenga conto che l'indirizzo di partenza per essi è: IB'=IB+4d.

L'elemento A(i,2) risiede quindi all'indirizzo IB+4d+d(i−1).

La regola generale che se ne deduce è allora quella di calcolare l'indirizzo di A(i,j) come:

$IB+4d(j-1)+d(i-1) = IB+d(4j+i-5)$.

Anche se lo spazio rimastoci ci impedisce ogni altra considerazione, è un

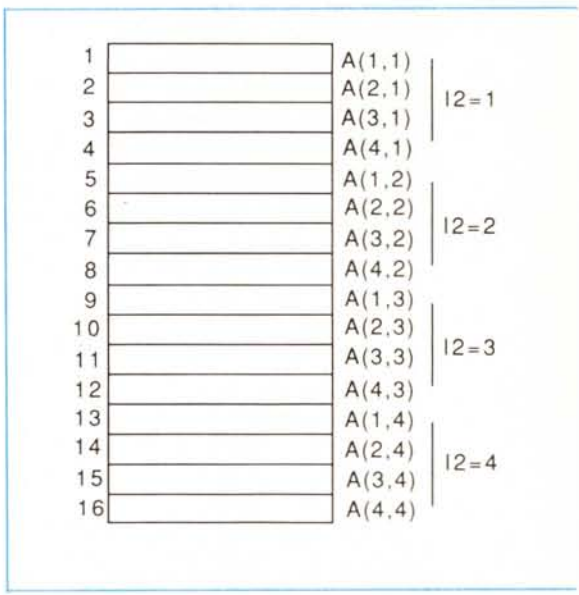

Figura 5 - L'allocazione in memoria della matrice A [1:4; 1:4].

vero peccato far rimanere nel dubbio i lettori più incalliti che stanno leggendo, ormai da un pezzo, questo articolo con carta e penna in mano.

Per cui, bando alla dimostrazione che è veramente insostenibile, il risultato è il seguente: data la matrice

$$A(I_{1,min}:\ I_{1,max};\ ...\ ;\ I_{n,min}:I_{n,max})$$

il generico elemento

$$A(i_1,i_2,...,i_n)$$

è allocato all'indirizzo fisico:

$$IB + d \cdot \sum_{k=1}^{n} (M_k \cdot I_{k,min}) + + \sum_{k=1}^{n} (d \cdot M_k \cdot i_k)$$

dove si è posto:

$M_1=1$; $M_k=m_{K-1} \cdot M_{k-1}$ per k=2,...,n

ed essendo evidentemente:

$$m_i = I_{i,max} - I_{I,min} + 1.$$

Buon lavoro a chi intende dimostrarlo.