6. Ricorsione

Una funzione è detta ricorsiva se all'interno della sua definizione richiama sé stessa. Più in generale, un algoritmo è detto di tipo ricorsivo se è espresso in termini di sé stesso; ovvero se l'esecuzione dell'algoritmo su un insieme di dati comporta la semplificazione o suddivisione dell'insieme di dati e l'applicazione dello stesso algoritmo agli insiemi di dati semplificati.
Tale tecnica risulta particolarmente utile per eseguire dei compiti ripetitivi su di un set di variabili in input. L'algoritmo richiama sé stesso generando una sequenza di chiamate che ha termine al verificarsi di una condizione particolare che viene chiamata condizione di terminazione, che in genere si ha con particolari valori di input.
La tecnica ricorsiva permette di scrivere algoritmi eleganti e sintetici per molti tipi di problemi comuni, anche se non sempre le soluzioni ricorsive sono le più efficienti. Questo è dovuto al fatto che comunemente la ricorsione viene implementata utilizzando le funzioni, e che l'invocazione di una funzione ha un costo rilevante, e questo rende più efficienti gli algoritmi iterativi. Notiamo, solo a fini nozionistici che, in alcuni casi la ricorsione è altrettanto efficiente di un ciclo iterativo: alcuni moderni linguaggi di tipo "funzionale" non hanno neppure il concetto di ciclo ed usano esclusivamente la ricorsione che viene automaticamente ottimizzata.

6.1. Fattoriale ricorsivo

Per capire meglio cosa s'intenda con ricorsione consideriamo il caso del fattoriale, già discusso ed implementato in maniera ricorsiva al paragrafo 5.1. Come detto il fattoriale di un numero naturale *n* è il prodotto dei primi enne numeri naturali, ossia:

$n! = 1 \times 2 \times 3 \times ..... \times (n - 1) \times n$

Rielaborando la definizione, e ricordando che 0! = 1 e che 1! = 1, ci si accorge di come sia possibile darne una versione ricorsiva.
Sia a tal proposito:

$n! = (1 \times 2 \times 3 \times ..... \times n\text{-}1) \times n;$

si ottiene:

$n! = (n\text{-}1)! \times n$

da cui, iterando,

$n! = (n\text{-}2)! * (n\text{-}1) * n,$

continuando ad iterare la definizione, arriveremo alle condizioni di terminazione, per cui il risultato cercato è noto:

$0! = 1! = 1.$

Di seguito una possibile implementazione ricorsiva del fattoriale fatta in VBA

```
Public Function Factorial(N As Integer) As Single
   Factorial = 1
   If N <= 1 Then Exit Function 'La condizione di escape, che deve sempre essere presente in funzioni ricorsive
   Factorial = N * Factorial(N - 1)
End Function
```

Come si vede tale implementazione è molto più compatta e leggibile rispetto all'analoga definizione iterativa introdotta al paragrafo 5.1. Riassumiamone il funzionamento, supponendo che N sia pari a 4.

- Dato che 4 è maggiore di uno, la funzione non esce (la condizione Exit Function viene ignorata) e restituisce il seguente valore: $Fattoriale_1 = 4 \times Fattoriale\ (4 - 1)$, dove il pedice indica l'ordine di chiamata; serve ora una nuova chiamata per calcolare Fattoriale (4 - 1).
- Dato che 3 è maggiore di uno la funzione non esce e restituisce il seguente valore: $Fattoriale_2 = 3 \times Fattoriale\ (3 - 1)$; serve ora una nuova chiamata per calcolare Fattoriale (3 - 1).
- Dato che 2 è maggiore di uno la funzione non esce e restituisce il seguente valore: $Fattoriale_3 = 2 \times Fattoriale\ (2 - 1)$; serve ora una nuova chiamata per calcolare Fattoriale (2 - 1).
- Finalmente si raggiunge la condizione di escape, dato che N = 1, e l'ultima chiamata di fattoriale restituisce il valore di 1.
- Tale valore viene passato a $Fattoriale_3$ (tuttora in attesa) che restituisce il valore di 2 x 1 = 2.
- Tale valore viene passato a $Fattoriale_2$ (tuttora in attesa) che restituisce il valore di 3 x 2 = 6.
- Tale valore viene passato a $Fattoriale_1$ (tuttora in attesa) che restituisce il valore di 4 x 6 = 24 e, finalmente l'esecuzione termina.

Come si vede si crea una sorta di coda (o meglio di pila) di funzioni che restano in attesa di ricevere un valore e che andranno via via ad esaurirsi (a chiudersi) dall'ultima sino alla prima. Affinché ciò sia possibile deve sempre essere fornita una condizione di *escape* che permetta di terminare il flusso di chiamate sequenziali e di attivare il processo, inverso, di chiusura. In questo caso la condizione di escape corrisponde alla condizione If N <= 1 Then Fattoriale = 1.

6.1. Serie di Fibonacci e Memoization

Consideriamo ora un secondo classico esempio, quello della serie di Fibonacci, definita nel modo seguente:

F(1) = 1
F(2) = 2
F(n) = F(n - 1) + F(n - 2) per ogni n > 2

In pratica si genera la seguente successione:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

La definizione stessa è di tipo ricorsivo per cui non ci sarebbe ragione di non usare una funzione ricorsiva per creare la serie di Fibonacci. Una possibile implementazione VBA è mostrata di seguito.

```
Public Function Fib(N As Integer) As Single
  If N <= 2 Then
    Fib = 1
    Exit Function
  End If
  Fib = Fib(N - 1) + Fib(N - 2)
End Function
```

Notiamo subito due cose:

- Innanzitutto, tale implementazione restituisce esclusivamente l'ennesimo valore della serie. Ad esempio, scrivendo Fib(8) si ottiene, come risultato, il valore 21. Non viene invece restituita la serie dei numeri precedenti a 21.

- Inoltre, tale formulazione è molto inefficiente, basta mettere un numero N alto (già un valore prossimo a 50, potrebbe dare problemi) e il tempo d'esecuzione diventa molto lungo. Perché? Ciò è dovuto al fatto che lo stesso numero di Fibonacci viene ricalcolato più volte, come mostrato nello schema seguente.
    - Fibonacci (8) = Fibonacci (7) + Fibonacci (6)
    - Fibonacci (7) = Fibonacci (6) + Fibonacci (5)
    - Fibonacci (6) = Fibonacci (5) + Fibonacci (4)
    - ….

A fronte di ciò, per migliorare le cose, ogni volta che si calcola uno specifico numero di Fibonacci conviene memorizzarlo, di modo che tale valore possa essere usato alla prossima esecuzione del codice. Se torniamo al caso di prima, dato che abbiamo calcolato Fibonacci (8), implicitamente abbiamo calcolato tutta la sequenza di Fibonacci sino a 21, ossia: {1, 1, 2, 3, 5, 8, 13, 21}. Se avessimo memorizzato tali valori a questo punto potremmo sveltire l'esecuzione di nuove chiamate, ad esempio Fib(10) richiederebbe solo di calcolare Fib(9) e Fib(8). Fib(8) è noto, mentre Fib(9) = Fib(8) + Fib(7) è costituito da due numeri memorizzati. Non serve quindi effettuare nessuna ricorsione, basta "leggere" i valori memorizzati per ottenere: Fib(10) = Fib(9) + 21 = (21 +13) + 21 = 55. Inoltre, salvare tutti i numeri della sequenza permetterebbe anche di restituire l'intera sequenza e non solo il valore in posizione enne. Tale tecnica prende il nome di *"memoization"*.
Detto ciò, come fare? Dato che le variabili definite all'interno di una funzione "restano in vita" solo fintantoché la funzione resta attiva, una possibile soluzione (ne vedremo un'altra successivamente) consiste nella creazione di una variabile globale in cui andare a registrare, di volta in volta i valori della sequenza di Fibonacci. Essendo una sequenza è normale pensare ad un vettore di Single; inoltre indicizzando tale vettore a partire da 1, possiamo direttamente associare all'elemento in posizione i-esima del nostro vettore all'elemento in posizione i-esima della sequenza di Fibonacci.
Il codice complessivo, comprensivo di una funzione usata per stampare a video la sequenza è mostrato di seguito.

```
Public FibSeq(1 To 1000) As Single 'Il vettore contenente la sequenza
Public Function FibMem(N As Integer) As Single
  If N <= 2 Then
    FibMem = 1
    If FibSeq(1) = 0 Then 'Se il nostro vettore è vuoto lo riempiamo con i primi valori
      FibSeq(1) = 1
      FibSeq(2) = 1
    End If
    Exit Function
  End If
  If FibSeq(N) <> 0 Then
    FibMem = FibSeq(N) 'Se il valore cercato è già presente nel vettore ci limitiamo a leggere tale valore
    Exit Function
  End If  'Altrimenti
  FibMem = FibMem(N - 1) + FibMem(N - 2) 'Usiamo la ricorsione per calcolare il valore
```

```
    FibSeq(N) = FibMem 'Una volta calcolato il valore lo scriviamo nel nostro vettore
End Function


'Procedura che calcola l'ennesimo valore della sequenza di Fibonacci e plotta a video l'intera sequenza
Public Sub Show_Fib(N As Integer)
Dim F As Single
Dim i As Integer
    F = FibMem(N)
    For i = 1 To 1000
        If FibSeq(i) <> 0 Then
            Debug.Print FibSeq(i)
        Else
            Exit For
        End If
    Next i
End Sub
```

6.2. Incolonnare i numeri – Funzioni operanti su stringa

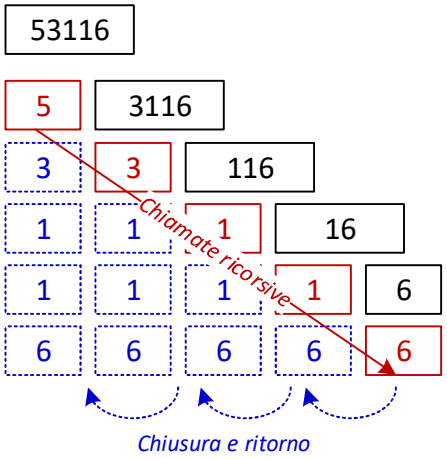
Supponiamo di avere un numero interno di enne cifre e di volerlo stampare in verticale incolonnando ciascuna cifra di cui è composto. Considerando, ad esempio, il numero 123, vorremmo stampare a video la seguente sequenza:

1
2
3

Per farlo è possibile pensare ad un approccio iterativo, ma in questo caso anche un approccio ricorsivo può funzionare bene. A differenza di prima, però, non disponiamo di una formulazione ricorsiva di una formula matematica. Per scrivere un algoritmo ricorsivo dobbiamo allora: *(i)* scomporre il problema in sotto problemi e *(ii)* definire una condizione di chiusura. In questo caso possiamo fare così:

- La funzione. chiamiamola "Impila", riceve in input un numero e lo converto in stringa:
- Valuta il numero di caratteri della stringa.
    - Se la stringa ha un solo carattere – questa sarà la condizione di chiusura – il numero è già "incolonnato", pertanto è sufficiente aggiungere a tale carattere il ritorno carrello "VbNewLine" (per andare a capo al momento dello stampaggio). Il carattere concatenato con VbNewLine sarà l'output restituito dalla funzione impila;
    - Se la stringa ha più di un carattere si prende il primo, lo si concatena con VbNewLine e si concatena il tutto con il risultato restituito da una nuova chiamata alla funzione Impila a cui viene passata la parte di stringa restante.

Questo è sintetizzato dallo schema seguente, relativa al numero 53116.



Esempio di funzionamento della funzione ricorsiva 'Impila'

Di seguito è infine mostrato il codice implementativo in VBA.

```
Public Function Impila(N As Integer) As String
Dim S As String
Dim i As Integer
N = Abs(N)
S = CStr(N)
  If Len(S) = 1 Then 'Len(Stringa) restituisce la lunghezza
    Impila = S & vbNewLine 'La e commerciale (&) è l'operatore di concatenamento delle stringhe
  Else
    i = CInt(Mid(S, 2, Len(S) - 1)) 'Mid(stringa, start,n) restituisce gli enne caratteri, a partire da start, di stringa
    Impila = Left(S, 1) & vbNewLine & Impila(i) 'Left(stringa,n) restituisce gli enne caratteri, a partire da sinistra
  End If
End Function

Public Sub Print_impila(N As Integer)
Dim S As String
  S = Impila(N)
  Debug.Print S
End Sub
```

6.2.1 Funzioni operanti su stringhe

Come visto, la funzione ricorsiva che effettua l'impilamento delle cifre di un numero, sfrutta alcune funzioni di base operanti su stringa (disponibili nella libreria standard delle funzioni VBA). Cogliamo allora l'occasione per discutere brevemente le principali funzioni operanti su stringa elencate nella tabella successiva:

Funzione	**Effetto**
Space(N)	Crea una stringa composta da N spazi
String(N, Chr)	Crea una stringa composta da N ripetizione del carattere Chr
CStr(Val)	Converte in formato stringa la variabile Val espressa in un qualsiasi tipo
&	Simbolo di concatenazione con Type Casting
+	Simbolo di concatenazione senza Type Casting
Len(S)	Restituisce la lunghezza della stringa S
InStr([Start], S1, S2,[Compare])	Verifica che la stringa S2 sia compresa in S1 a partire dal carattere start, procedendo da sinistra verso destra
InStrRev(S1, S2,[Start],[Compare])	Verifica che la stringa S2 sia compresa in S1 a partire dal carattere start, procedendo da destra verso sinistra
LCase(S)	Scrive S a caratteri minuscoli
UCase(S)	Scrive S a caratteri maiuscoli
StrConv(S,form_type)	Formatta S secondo lo stile definito dal parametro form_type
LTrim(S)	Rimuove gli spazi a sinistra della stringa
RTrim(S)	Rimuove gli spazi a destra della stringa
Trim(S)	Rimuove gli spazi sia a destra che a sinistra
Replace(S, Sf, Sr, [start], [N],[comp])	Rimpiazza le sottostringhe Sf (contenute in S) con la sottostringa Sr, a partire da start e per un numero di volte pari ad N
StrReverse(S)	Restituisce la stringa di input S letta al contrario
Split(S, [delimiter])	Restituisce un Array di variant i cui elementi sono tutte le sottostringhe di S che nella stringa originale (ossia S) erano separate dal delimitatore
Join(A, [delimiter])	Concatena in un'unica stringa i componenti dell'array A separandoli tramite il delimitatore passato in input
Asc(Char)	Restituisce il numero che codifica il carattere Char nel codice Ascii
Chr(N)	Restituisce il carattere codificato da N nel codice Ascii
StrCompare(S1, S2, [Comp])	Confronta la stringa S1 con la S2. Restituisce 0 se sono uguali, -1 se S2 > S1 e 1 se S1 > S2

1 – Stringhe di lunghezza fissa

Per creare una stringa di lunghezza fissa si usa la seguente notazione:

```
Dim a As String*N
```

dove N rappresenta il numero (fisso) di caratteri di cui sarà composta la stringa.

In maniera alternativa (e più elegante) possiamo ottenere lo stesso risultato usando la funzione **Space(Numero_spazi)**:

```
Dim b As String
b = Space(N) 'Stringa con N spazi bianchi
```

che restituisce una stringa con tanti spazi quanti quelli definiti dal parametro di input.

In effetti una piccola differenza c'è. Infatti, la stringa 'a' ha lunghezza fissa, mentre 'b' non lo è. Ad esempio, se scrivessimo:

```
Dim a As String*10, b As String
b = Space(N) 'Ora se plottati a video a e b apparirebbero uguali
a = "AAAAAAAAAAAAAA" 'assegniamo ad 'a' una stringa di 13 A
b = "AAAAAAAAAAAAAA" 'assegniamo a 'b' una stringa di 13 A
'mentre 'b' contiene tutti e 13 i caratteri A, 'a' essendo di lunghezza fissa ne contiene solo 10
```

Più in generale possiamo usare la funzione **String(N_caratteri, Carattere)**:

```
Dim b  As String
b = String(N_caratteri, Carattere)
```

in cui N_caratteri rappresenta il numero fisso di caratteri di cui sarà composta la stringa e Carattere rappresenta il carattere di riempimento. Ad esempio, scrivendo:

```
Dim a As String, b As String
a = Space(10)
b = String(10, "A")
```

la stringa 'a' risulterà composta di 10 caratteri vuoti, mentre 'b' sarà composto di 10 caratteri "A".

2 – Convertire e Concatenare le stringhe

Per convertire una variabile da un qualsiasi formato ad una stringa (si parla di Type casting) si usa la funzione **CStr(variabile)**.[2]

Ad esempio, il seguente codice fa in modo che la variabile 'b' abbia un valore "10000", in cui l'uno e gli zeri vanno intesi come caratteri e non come cifre numeriche.

```
Dim a As Integer, b As String
a = 10000
b = CStr(a)
```

Per concatenare due o più stringhe possiamo usare, alternativamente il simbolo **'+'** o il simbolo **'&'**. In generale è più indicato l'uso di '&', dato che permette di concatenare, in un'unica variabile stringa, variabili non necessariamente di tipo stringa. In pratica, tale operatore esegue automaticamente tutti i Type Casting necessari (conversione implicita) alla corretta concatenazione.

[2] *Funzioni simili possono essere usate per convertire una variabile in un altro tipo. Ad esempio CInt, CDbl, CDate convertono una variabile, rispettivametne, in formato intero, double e data.*

Ad esempio, le due seguenti scritture sono equivalenti:

```
Dim a As Integer, b As Integer
Dim S As String
a = 10
b = 20
S = Cstr(a) + “+” Cstr(b) “=” Cstr(a + b) 'Restituisce “10 + 20 = 30”
S = a & “+” &  b & “=” & (a + b) 'Restituisce “10 + 20 = 30”
```

Viceversa, l’espressione successiva è scorretta:

```
S = a + “+” +  b + “=” + (a + b) 'Espressione errata
```

Osserviamo inoltre che mentre CStr(var) potrebbe dare errore qualora l’input var fosse Nullo, l’operatore & non ha questo problema e converte eventuali variabili nulle in stringhe vuote (“”).

3 – Lunghezza di una stringa

Per valutare la lunghezza di una stringa è possibile utilizzare la funzione **Len(S)** che prende in input un solo parametro, la stringa S:

```
Dim a As String
a = "abcdefghilmnopqrstuvz"
 MsgBox Len(a) ' La message box mostra il valore di 21
```

4 – Ricerca di una sottostringa

Per capire se una stringa contiene al suo interno una sottostringa ci sono due funzioni speculari InStr e InstrRev. In particolare, la funzione **InStr** prende in input la posizione di partenza da cui deve iniziare il confronto, parametro opzionale (di default la ricerca parte dall’inizio), la stringa principale e la sottostringa che si vuole ricercare. Come ultimo parametro opzionale è possibile specificare il tipo di confronto: 0 per binario (e quindi case sensitive), 1 per testuale (case insensive).

Di seguito un esempio:

```
Dim a As String, b As String
a = "Francesco_Francesco"
b = "Cesco"
MsgBox (InStr(1, a, b, 0)) ' risultato 0 perché il confronto è binario e quindi case sensitive
MsgBox (InStr(1, a, b, 1)) ' risultato 5 perché confronto letterario e trova la prima occorrenza al quinto carattere
MsgBox (InStr(10, a, b, 1)) ' risultato 15 perché la ricerca parte dal decimo elemento
```

La funzione **InstringRev** è analoga alla precedente, ma cerca la sottostringa da destra verso sinistra. La sintassi cambia un po’ dato che i primi due parametri (obbligatori) sono la stringa in cui effettuare la ricerca e la stringa da ricercare; gli ultimi due sono, rispettivamente il punto da cui iniziare la ricerca (da destra verso sinistra) e il tipo di confronto da effettuare. Per iniziare dall’ultimo carattere si può usare il valore -1. Ad esempio:

```
Dim a As String, b As String
a = "Francesco_Francesco"
b = "Cesco"
MsgBox (InStr(10, a, b, 1)) 'Dalla decima verso sinistra
MsgBox (InStrRev(a, b, -1, 1)) 'Dall’ultima verso destra
 MsgBox (InStrRev(a, b, 19, 1)) 'Dall’ultima verso destra
```

5 – Estrarre caratteri

Per ottenere una sottostringa a partire da una stringa iniziale si possono usare le seguenti tre funzioni: Left, Right e Mid.

- **Left(S, N)** restituisce la sottostringa costituita dei primi N caratteri della stringa S a partire dal primo (quello più a sinistra).
- **Rigth(S, N)** restituisce la sottostringa costituita degli ultimi N caratteri della stringa S a partire da quello più a destra.
- **Mid(S, Start, [N])** restituisce la sottostringa costituita da N caratteri a partire dal carattere in posizione Start della stringa originale S. Se N è omesso (parametro opzionale), la sotto stringa restituita va dal carattere in posizione Start al carattere finale di S.

Un semplice esempio è mostrato di seguito:

```
Dim a As String, b As String
a = "Francesco"
b = Left(a,3) 'Restituisce Fra
b = Mid(a, 1,3) 'Restituisce Fra come nel caso precedente
b = Right(a,5) 'Restituisce Cesco
b = Mid(a, 5) 'Restituisce Cesco come nel caso precedente
```

6 – Modificare una stringa

Esistono delle funzioni che permettono di restituire una stringa modificata secondo alcune regole:

- **LCase** e **UCase** rispettivamente restituiscono una stringa in minuscole e maiuscole;
- **StrConv** amplia le due funzioni precedenti dato che accetta un secondo parametro di input che specifica il tipo di modifica (formattazione) da apportare alla stringa:

```
StrConv ("pippo PLUTO paPErinO", vbUpperCase) ' PIPPO PLUTO PAPERINO
StrConv ("pippo PLUTO paPErinO", vbLowerCase) ' pippo pluto paperino
StrConv ("pippo PLUTO paPErinO", vbProperCase) ' Pippo Pluto Paperino
```

Come possiamo notare, con vbProberCase la prima lettera di ogni parola della stringa venga convertita in maiuscolo. Altri parametri per la funzione StrConv servono più che altro per la conversione in lingue cinese e giapponese.

Per "pulire" una stringa possono usarsi le seguenti quattro espressioni:

- **LTrim** e **RTrim** restituiscono una stringa prima di spazi vuoti iniziali o finali;
- **Trim** invece rimuove gli spazi sia iniziali che finali;
- **Replace**, invece, richiede come parametri: una stringa iniziale, la sottostringa da trovare ed il valore con il quale deve essere sostituita. Come parametri aggiuntivi, ma opzionali, richiede la posizione della stringa principale da cui iniziale la ricerca (di default la ricerca inizia dal primo carattere), il numero di sostituzioni massime che devono essere eseguite (di default tutte le sostituzioni possibili vengono effettuate) ed il tipo di confronto (binario o letterale).

Esiste anche una funzione, **StrReverse**, che restituisce la stringa di input letta al contrario.

6 – Da Stringa ad Array e Viceversa

A volte può essere utile suddividere una stringa in più sottostringhe separate da un elemento di delimitazione. Ad esempio, detto "-" l'elemento di delimitazione vorremmo poter suddividere la stringa "Pippo-Pluto-Topolino" in tre stringhe contenenti i tre distinti personaggi Disney. Ovviamente potremmo creare una funzione personalizzata atta allo scopo; a tal fine[3] sarebbe sufficiente impiegare, in maniera integrata, le funzioni Instring (per trovare la posizione di ciascun carattere di

[3] *La scrittura di tale funzione è lasciata come esercizio al lettore*

delimitazione) e la funzione Mid per generare le sottostringhe volute. Più rapidamente possiamo utilizzare la funzione **Split(S, [Delimiter])** che fa esattamente ciò che volevamo. In pratica Split crea, a partere da una stringa S, un vettore di tipo variant i cui elementi sono valorizzati con le sottostringhe di S delimitate dal carattere delimitatore (se non passato di default viene utilizzato lo spazio). Per questo motivo, per utilizzare tale funzione dobbiamo assegnarla ad una variabile variant precedentemente definita, come mostrato di seguito:

```
Dim S As String, D As String
Dim V As Variant
S = “Pippo-Pluto-Topolino”
D = “-”
V = Split(S, D)
Debug.Print V(0) 'L’indicizzazione parte da 0, mostra il primo elemento ossia Pippo
Debug.Print V(1) 'mostra il secondo elemento ossia Pluto
Debug.Print V(2) 'mostra il terzo elemento ossia Topolino
```

Esiste anche l’operatore **Join(Array, [Delimiter])** che svolge l’operazione inversa, ossia combina gli elementi di un vettore in un’unica stringa, concatenando tali elementi mediante il simbolo delimiter. Anche in questo caso, se l’ultimo parametro (opzionale) non viene passato alla funzione, di default si usa lo spazio come elemento di delimitazione. Ad esempio, il codice seguente ricombina in un’unica stringa il nome dei tre personaggi Disney precedentemente analizzati.

```
S = Join(V)
Debug.Print S 'Il delimiter non è presente, per cui si usa lo spazio e si ottiene “Pippo Pluto Topolino”
```

7 – Caratteri e codici ASCII

Come si legge su Wikipedia, con US-ASCII si intende un sistema di codifica dei caratteri a 7 bit, comunemente utilizzato nei calcolatori, proposto dall'ingegnere dell'IBM Bob Bemer nel 1961, e successivamente accettato come standard dall'ISO, con il nome di ISO/IEC 646. Attualmente, lo standard che sta prendendo piede e che dovrebbe essere il successore di ASCII è UTF-8, specie da quando è diventato la codifica principale di Unicode per internet.
La tabella seguente è relativa al codice US ASCII, ANSI X3.4-1986 (ISO 646 International Reference Version). In particolare:

- I codici decimali da 0 a 31 e il 127 sono caratteri non stampabili (codici di controllo).
- Il 32 corrisponde al carattere di "spazio".
- I codici dal 32 al 126 sono caratteri stampabili.

Tabella codici ASCII non stampabili

Binario	Ottale	Decimale	Esadecimale	Abbr	PR	CS	CEC	Descrizione
000 0000	000	0	00	NUL	$^{N}_{U}{}_{L}$	^@	\0	Null character
000 0001	001	1	01	SOH	$^{S}_{O}{}_{H}$	^A		☺
000 0010	002	2	02	STX	$^{S}_{T}{}_{X}$	^B		☻
000 0011	003	3	03	ETX	$^{E}_{T}{}_{X}$	^C		♥
000 0100	004	4	04	EOT	$^{E}_{O}{}_{T}$	^D		♦
000 0101	005	5	05	ENQ	$^{E}_{N}{}_{Q}$	^E		♣
000 0110	006	6	06	ACK	$^{A}_{C}{}_{K}$	^F		♠
000 0111	007	7	07	BEL	$^{B}_{E}{}_{L}$	^G	\a	•
000 1000	010	8	08	BS	$^{B}_{S}$	^H	\b	◘
000 1001	011	9	09		$^{H}_{T}$	^I	\t	○
000 1010	012	10	0A	LF	$^{L}_{F}$	^J	\n	◙
000 1011	013	11	0B	VT	$^{V}_{T}$	^K	\v	♂
000 1100	014	12	0C	FF	$^{F}_{F}$	^L	\f	♀

000 1101	015	13	0D	CR	$_{CR}$	^M	\r	♪
000 1110	016	14	0E	SO	$_{SO}$	^N		♫
000 1111	017	15	0F	SI	$_{SI}$	^O		☼
001 0000	020	16	10	DLE	$_{DLE}$	^P		►
001 0001	021	17	11	DC1	$_{DC_1}$	^Q		◄
001 0010	022	18	12	DC2	$_{DC_2}$	^R		↕
001 0011	023	19	13	DC3	$_{DC_3}$	^S		‼
001 0100	024	20	14	DC4	$_{DC_4}$	^T		¶
001 0101	025	21	15	NAK	$_{NAK}$	^U		§
001 0110	026	22	16	SYN	$_{SYN}$	^V		▬
001 0111	027	23	17	ETB	$_{ETB}$	^W		↨
001 1000	030	24	18	CAN	$_{CAN}$	^X		↑
001 1001	031	25	19	EM	$_{EM}$	^Y		↓
001 1010	032	26	1A	SUB	$_{SUB}$	^Z		→
001 1011	033	27	1B	ESC	$_{ESC}$	^[		←
001 1100	034	28	1C	FS	$_{FS}$	^\		∟
001 1101	035	29	1D	GS	$_{GS}$	^]		↔
001 1110	036	30	1E	RS	$_{RS}$	^^		▲
001 1111	037	31	1F	US	$_{US}$	^_		▼
111 1111	177	127	7F	DEL	$_{DEL}$	^?		Delete (equivalente a Backspace)

Tabella codici ASCII stampabili

in	Oc	Dc	Hx	Car	Bin	Oc	Dc	Hx	Car	Bin	Oc	Dc	Hx	Car
010 0000	040	**32**	20	**Spazio**	100 0000	100	**64**	40	**@**	110 0000	140	**96**	60	**`**
010 0001	041	**33**	21	**!**	100 0001	101	**65**	41	**A**	110 0001	141	**97**	61	**a**
010 0010	042	**34**	22	**"**	100 0010	102	**66**	42	**B**	110 0010	142	**98**	62	**b**
010 0011	043	**35**	23	**#**	100 0011	103	**67**	43	**C**	110 0011	143	**99**	63	**c**
010 0100	044	**36**	24	**$**	100 0100	104	**68**	44	**D**	110 0100	144	**100**	64	**d**
010 0101	045	**37**	25	**%**	100 0101	105	**69**	45	**E**	110 0101	145	**101**	65	**e**
010 0110	046	**38**	26	**&**	100 0110	106	**70**	46	**F**	110 0110	146	**102**	66	**f**
010 0111	047	**39**	27	**'**	100 0111	107	**71**	47	**G**	110 0111	147	**103**	67	**g**
010 1000	050	**40**	28	**(**	100 1000	110	**72**	48	**H**	110 1000	150	**104**	68	**h**
010 1001	051	**41**	29	**)**	100 1001	111	**73**	49	**I**	110 1001	151	**105**	69	**i**
010 1010	052	**42**	2A	*****	100 1010	112	**74**	4A	**J**	110 1010	152	**106**	6A	**j**
010 1011	053	**43**	2B	**+**	100 1011	113	**75**	4B	**K**	110 1011	153	**107**	6B	**k**
010 1100	054	**44**	2C	**,**	100 1100	114	**76**	4C	**L**	110 1100	154	**108**	6C	**l**
010 1101	055	**45**	2D	**-**	100 1101	115	**77**	4D	**M**	110 1101	155	**109**	6D	**m**
010 1110	056	**46**	2E	**.**	100 1110	116	**78**	4E	**N**	110 1110	156	**110**	6E	**n**
010 1111	057	**47**	2F	**/**	100 1111	117	**79**	4F	**O**	110 1111	157	**111**	6F	**o**
011 0000	060	**48**	30	**0**	101 0000	120	**80**	50	**P**	111 0000	160	**112**	70	**p**
011 0001	061	**49**	31	**1**	101 0001	121	**81**	51	**Q**	111 0001	161	**113**	71	**q**
011 0010	062	**50**	32	**2**	101 0010	122	**82**	52	**R**	111 0010	162	**114**	72	**r**
011 0011	063	**51**	33	**3**	101 0011	123	**83**	53	**S**	111 0011	163	**115**	73	**s**
011 0100	064	**52**	34	**4**	101 0100	124	**84**	54	**T**	111 0100	164	**116**	74	**t**
011 0101	065	**53**	35	**5**	101 0101	125	**85**	55	**U**	111 0101	165	**117**	75	**u**
011 0110	066	**54**	36	**6**	101 0110	126	**86**	56	**V**	111 0110	166	**118**	76	**v**
011 0111	067	**55**	37	**7**	101 0111	127	**87**	57	**W**	111 0111	167	**119**	77	**w**
011 1000	070	**56**	38	**8**	101 1000	130	**88**	58	**X**	111 1000	170	**120**	78	**x**
011 1001	071	**57**	39	**9**	101 1001	131	**89**	59	**Y**	111 1001	171	**121**	79	**y**
011 1010	072	**58**	3A	**:**	101 1010	132	**90**	5A	**Z**	111 1010	172	**122**	7A	**z**
011 1011	073	**59**	3B	**;**	101 1011	133	**91**	5B	**[**	111 1011	173	**123**	7B	**{**
011 1100	074	**60**	3C	**<**	101 1100	134	**92**	5C	****	111 1100	174	**124**	7C	**|**
011 1101	075	**61**	3D	**=**	101 1101	135	**93**	5D	**]**	111 1101	175	**125**	7D	**}**
011 1110	076	**62**	3E	**>**	101 1110	136	**94**	5E	**^**	111 1110	176	**126**	7E	**~**
011 1111	077	**63**	3F	**?**	101 1111	137	**95**	5F	**_**					

Per operare con il codice ASCII, VBA mette a disposizione due funzioni:

- **Asc(Ch)** - Prende in input un carattere e restituisce il suo codice ASCII
- **Chr(Code)** - Prende in input un numero e restituisce carattere corrispondente.

8 – Confronto tra stringhe

Un'altra utile funzione è StrComp(S1, S2, [Compare]), funzione che permette di ordinare/confrontare due stringhe le stringhe. Tale funzione prende in input due stringhe, ed eventualmente la modalità di confronto, e restituisce un intero che indica quale delle due stringhe sia alfabeticamente maggiore (1 se è maggiore la prima stringa, -1 se è maggiore la seconda, 0 in caso di parità). Si noti che, quando si effettua un confronto letterale, il confronto viene fatto carattere per carattere, considerando l'ordinamento alfabetico; inoltre, i numeri precedono le lettere e le lettere maiuscole precedono le minuscole (così come appare evidente dalle tabelle ASCII sovra riportate).

```
' Se non si specifica il tipo di confronto, viene assunto di tipo binario, per cui Case Sensitive
StrCompare("alfa", "alfa") ' restituisce 0, stessa stringa,
StrCompare("alfa", "beta") ' restituisce -1 alfa viene prima di beta, è più piccolo
StrCompare("alfa", "alFa") ' restituisce 1 alfa viene prima di alFa (prima maiuscole e poi minuscole)
StrCompare("alfa", "ALFA", vbTextCompare) ' restituisce 0 perché case insensitive
StrCompare("alfa", "1alfa") ' restituisce 1 perché i numeri vengono prima delle lettere
StrCompare("100", "99") ' restituisce -1, la stringa 100 viene prima della stringa "99", perché 1 precede 9
```

8 – Distanza di Levenshtein

Qualora si avesse la necessità di verificare la similitudine tra due stringhe è possibile utilizzare la distanza di Levenshtein (o Edit Distance) funzione che restituisce un intero rappresentante il grado di somiglianza tra due stringhe. In particolare, come indicato da Wikipedia, nella teoria dell'informazione e nella teoria dei linguaggi, la distanza di Levenshtein, o distanza di edit, è una misura della differenza fra due stringhe. Introdotta dallo scienziato russo Vladimir Levenshtein nel 1965, tale misura serve a determinare quanto due stringhe siano simili.

In particolare, la distanza di Levenshtein tra due stringhe S_1 e S_2 è il numero minimo di modifiche elementari che consentono di trasformare S_1 in S_2, dove per modifica elementare si intende: *(i)* la cancellazione di un carattere, *(ii)* la sostituzione di un carattere con un altro o, *(iii)* l'inserimento di un carattere. Per fare un esempio consideriamo le stringhe "bar" e "biro". In questo caso la distanza di Levenshtein vale 2, dato che, per trasformare "bar" in "biro" occorrono almeno due modifiche:

- bar → bir (sostituzione di 'a' con 'i')
- bir → biro (inserimento di 'o')

Sfortunatamente, tra le funzioni presenti nella libreria standard di VBA non figura la distanza di Levenshtein; comunque, tale funzione è facilmente implementabile, ad esempio mediante il seguente algoritmo di calcolo.

Per prima cosa, senza perdere di generalità, supponiamo che il numero dei caratteri di S_1 sia minore del numero dei caratteri di S_2 e definiamo con *"allineamento"* una qualsiasi stringa A ottenuta aggiungendo dei "gap" (o spazi vuoti, simbolicamente indicati con il quadratino ■) a S_1 per uniformarne la lunghezza con quella di S_2. Questa definizione ci consente di ridefinire la *Edit Distance come il costo dell'allineamento ottimo, indicato con* A^*, dove:

- il costo *c(a, b)* necessario per uniformare due caratteri, nella medesima posizione, di A^* e S_2 è definito nel modo seguente:

$$c(a,b) = \begin{cases} 0 \quad \text{se a} = \text{b} \\ 1 \text{ se (a} \neq \text{b) o (a} = \blacksquare\text{) o (b} = \blacksquare\text{)} \end{cases}$$

- il costo complessivo di un intero allineamento è pari alla somma dei costi *c(a,b)* di ciascuna posizione.

Facciamo subito un esempio. Consideriamo la stringa S_1 = 'abcd' e di S_2 = 'a12c3f'; in questo caso la Edit Distance vale 4 dato che bisogna effettuare quattro operazioni: introdurre il carattere '1', sostituire 'b' con '2', sostituire il carattere 'd' con '3' e introdurre il carattere 'f'. Questo è immediatamente osservabile mediante il costo del seguente allineamento ottimo, dove in rosso sono stati evidenziati i caratteri di costo unitario

$$A^* = a \blacksquare b\ c \blacksquare d$$
$$S_2 = a\ 1\ 2\ c\ 3\ f$$

Viceversa, un allineamento non ottimo potrebbe essere il seguente:

$$A = a \blacksquare b\ c \blacksquare d$$
$$S_2 = a\ 1\ 2\ c\ 3\ f$$

In questo caso, infatti, per uniformare le lunghezze sono i due gap sono stati introdotti subito dopo il carattere 'a', ma così facendo il costo totale risulta pari a 5, valore maggiore rispetto al precedente.

È allora necessario trovare un modo per creare allineamenti ottimali, ossia per stabilire il punto ottimale in cui introdurre i gap.

L'algoritmo per fare ciò si basa sull'osservazione seguente. Siano $S_1 = a_1 \dots a_m$ e $S_2 = b_1 \dots b_n$ (con *m* < *n*) le due stringhe di partenza. Denotiamo con *e(i, j)* l'edit distance tra le due sottostringhe (prefissi) $s_{1,i} = a_1 \dots a_i$ e $s_{2,j} = b_1 \dots b_j$ ottenute considerando i primi caratteri i e j, rispettivamente delle stringhe S_1e S_2. Se *e(i-1, j-1)*, *e(i, j-1)* ed *e(i-1, j)* fossero noti, sarebbe immediato calcolare *e(i , j)*. Ciascuno dei tre valori corrisponde infatti a un allineamento ottimo:

- *e(i-1, j-1)* è il costo dell'allineamento ottimo di $s_{1,i-1}$ e $s_{2,j-1}$
- *e(i-1, j)* è il costo dell'allineamento ottimo $s_{1,i-1}$ e $s_{2,j}$
- *e(i, j-1)* è il costo dell'allineamento ottimo $s_{1,i}$ e $s_{2,j-1}$

Pertanto, l'allineamento tra $s_{1,i}$ e $s_{2,j}$ può essere esteso nel modo seguente:

- da $s_{1,i-1}$ $s_{2,j-1}$ facendo combaciare a_i e b_j
- da $s_{1,i-1}$ $s_{2,j}$ allineando un gap con b_j
- da $s_{1,i}$ $s_{2,j-1}$ allineando un gap con a_i

Ovviamente per mantenere l'ottimalità, bisognerà scegliere l'operazione che consente di ottenere l'allineamento di costo minimo che, per quanto detto varrà:

$$e(i,j) = \begin{cases} e(i-1, j-1) + c(a_i, b_j) \\ e(i-1, j) + 1 \\ e(i,, j-1) + 1 \end{cases}$$

Possiamo allora calcolare tutte le distanze *e(i, j)* in maniera ciclica partendo da *e(0, 0)*, distanza tra due stringhe vuote, sino ad arrivare a *e(m, n)*, distanza tra le due stringhe considerate. Tali valori possono essere organizzati in una matrice ***M*** di dimensioni *(m + 1)×(n + 1)*, convenzionalmente indicizzata a partire da zero, come quella riportata di seguito, in cui ε indica la stringa vuota.

	ε	b_1	b_2	⋯	b_n
ε	*e(0, 0)*	*e(0, 1)*	*e(0, 2)*	⋯	*e(0, n)*
a_1	*e(1, 0)*	*e(1, 1)*	*e(1, 2)*	⋯	*e(1, n)*
a_2	*e(2, 0)*	*e(2, 1)*	*e(2, 2)*	⋯	*e(2, n)*
⋮	⋮	⋮	⋮	⋮	⋮
a_m	*e(m, 0)*	*e(m, 1)*	*e(m, 2)*	⋯	*e(m, n)*

È evidente che per allineare ε con una stringa S composta di 'i' caratteri, si paga un costo pari ad 'i', dato che è necessario inserire esattamente 'i' caratteri per passare da ε a S.
A fronte di ciò, inizialmente la matrice sarà così composta:

	ε	b_1	b_2	⋯	b_n
ε	*0*	*1*	*2*	⋯	*n*
a_1	*1*	*Null*	*Null*	⋯	*Null*
a_2	*2*	*Null*	*Null*	⋯	*Null*
⋮	⋮	⋮	⋮	⋮	⋮
a_m	*m*	*Null*	*Null*	⋯	*Null*

Creata la matrice ***M***, si tratta di calcolare tutti i valori nulli partendo da *e(1, 0)* fino ad arrivare al valore finale di *e(m, n)*. Ad esempio, ciclando prima per riga e poi per colonna si può procedere così:

- per ogni riga i, con i che varia da 1 a m:
  - per ogni colonna j, con j che varia da 1 a n:
    - se a_i è uguale a b_j, porre c = 0, altrimenti porre c = 1
    - porre M[i,j] = min{M[i-1,j-1] + c , M[i,j-1] + 1 , M[i-1,j] + 1 }

Una possibile implementazione VBA è mostrata di seguito:

```
Function Edit_Dst(ByVal S1 As String, ByVal S2 As String) As Long

Dim i As Long, j As Long
Dim L1 As Long, L2 As Long
Dim M() As Long 'La matrice delle distanze e(i,j)
    L1 = Len(S1)
    L2 = Len(S2)
    ReDim M(0 To L1, 0 To L2)

    'Preparazione della matrice (0,1,…,m sulla prima colonna)
    For i = 0 To L1
        M(i, 0) = i
    Next i

    'Preparazione della matrice (0,1,…,n sulla prima riga)
    For j = 0 To L2
        M(0, j) = j
    Next j

    'Compilazione della matrice M
    For i = 1 To L1
        For j = 1 To L2
            If Asc(Mid(S1, i, 1)) = Asc(Mid(S2, j, 1)) Then 'Confr. del carattere
                M(i, j) = M(i - 1, j - 1)
            Else
                M(i, j) = Application.WorksheetFunction.Min _
```

```
                                            (M(i - 1, j) + 1, _
                                             M(i, j - 1) + 1, _
                                             M(i - 1, j - 1) + 1)
            End If
        Next j
    Next i
    Edit_Dst = M(L1, L2)
End Function
```

Se si volesse stampare anche la matrice ***M***, si potrebbero includere le seguenti linee nel codice precedente:

```
Dim Sep3 As String * 3
Dim Sep2 As String * 2

'Intestazione riga
  S = Space(3) & "e" & Space(2)
  For i = 1 To L2
    Sep3 = CStr(i)
    S = S & Sep3
  Next i
  S = S & vbNewLine
  S = S & String((L2 + 2) * 3, "-") & vbNewLine

'Valori e intestazione di colonna
  For i = 0 To L1
    For j = -1 To L2
      If j = -1 Then
        If i = 0 Then Sep2 = "e"
        If i <> 0 Then Sep2 = CStr(i)
        Sep3 = Sep2 & "|"
      End If
      If j >= 0 Then Sep3 = CStr(M(i, j))
      S = S & Sep3
    Next j
    S = S & vbNewLine
  Next i
  Debug.Print S
```

Operando in questo modo, se eseguissimo la seguente funzione ?Edit_Dst("abcd","a12c3f") otterremmo la seguente rappresentazione a video

```
  e  1  2  3  4  5  6
-------------------------
e |0  1  2  3  4  5  6
1 |1  0  1  2  3  4  5
2 |2  1  1  2  3  4  5
3 |3  2  2  2  2  3  4
4 |4  3  3  3  3  3  4
```

6.3. Esempio di ricorsione con jagged array

Un jagged array è un array monodimensionale i cui elementi sono essi stessi array monodimensionali. In pratica un jagged array è una struttura dati composta da un array padre che contiene, al suo interno, una serie di vettori figli. A prima vista sembra quasi che un jagged array coincida esattamente con una matrice bidimensionale. Tale visione, seppur concettualmente corretta, non è del tutto esatta. Ciò che differenzia una matrice bidimensionale da un jagged array è il fatto che: *(i)* mentre le righe di una matrice devono necessariamente contenere lo stesso numero di elementi, *(ii)* i vettori figli di un jagged array possono avere dimensioni differenti. In altri termini, è possibile ottenere matrici bidimensionali con righe di dimensioni variabili (inoltre, come vedremo, oltre alle dimensioni può variare anche il tipo di dati contenuto in ciascuna riga). Una visualizzazione grafica di quanto detto è mostrata nello schema seguente che riporta una matrice tradizionale ***M****(i,j)*, con 3 righe e3 colonne e un jagged array ***J****(i)(k)* con 3 righe e numero di colonne variabile.

	A(1,1)	A(1,2)	A(1,3)
A	A(2,1)	A(2,2)	A(2,3)
	A(3,1)	A(3,2)	A(3,3)

	J (1) (1)	J (1) (2)	
J	J (2) (1)		
	J (3) (1)	J (3) (2)	J (3) (3)

In VBA la creazione di un jagged array richiede di utilizzare come "vettore padre" un vettore di tipo Variant; viceversa i "vettori figli" possono essere vettori di qualsiasi natura.

Supponiamo ad esempio di voler creare un jagged arry come quello della figura precedente e supponiamo inoltre che i vettori figli siano, rispettivamente di tipo Integer, String e Double. Una possibile implementazione è mostrata di seguito[4].

```
Dim J(1 To 3)
Dim j_1(1 To 2) As Integer
Dim j_2(1 To 1) As String
Dim j_3(1 to 3) As Double
j_1(1)= 1
j_1(2) = 2
J_2(1) = "a"
J_3(1) = "0.1"
J_3(2) = "0.2"
J_3(3) = "0.3"
J(1) = j_1
J(2) = j_2
J(3) = j_3
```

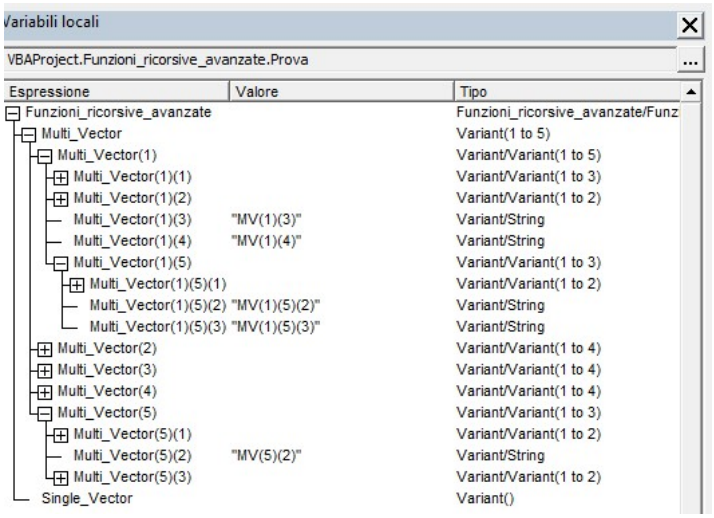
[4] *Per le altre modalità di creazione di un jagged array si rimanda agli esempi successivi e/o agli esempi presenti sul file Excel allegato alla presente dispensa*

Infine, per accedere ai valori di J è necessario usare la notazione (i) (j), come mostrato di seguito:

```
Dim I As integer, S As String, D As Double
I = J(1)(2)
S = J(2)(1)
D = J(3)(3)
```

Vediamo ora alcuni esempi di funzioni ricorsive operanti su jagged array.

La prima crea in maniera casuale un jagged array di jagged array, ossia una struttura dati in cui i vettori figli possono a loro volta essere jagged array, ossia possono essi stessi contenere altri array. In particolare, in questo caso, la dimensione di ogni sotto-jagged array è inferiore (almeno di 1) della dimensione del corrispondente vettore padre. Infine, ogni elemento finale contiene il suo "indirizzo", per esempio MV(1)(1)(1) sarà proprio uguale a "MV(1)(1)(1)", come mostrato nello screen-hot seguente che riporta un esempio di esecuzione della suddetta procedura.



Esempio di un Jagged Array

```
Public Function Crea_MV(D1 As Integer, Max_D2 As Integer, Optional Code As String = "MV(") As Variant
Dim MV() As Variant
Dim C As String
Dim i As Integer, j As Integer
  ReDim MV(1 To D1) 'Dimensione del vettore padre
  For j = 1 To D1
    C = Code & CStr(j) & ")"
    i = Application.WorksheetFunction.RandBetween(1, Max_D2) 'Dimensione random del vettore figlio
    If i = 1 Then
      MV(j) = C 'Siamo ad una foglia della struttura. Scriviamo il valore che è pari all'indirizzo
    Else
      MV(j) = Crea_MV(i, i - 1, C & "(") ' Richiamiamo la procedura passando il nuovo codice
```

```
    End If
  Next j
Crea_MV = MV
End Function
```

La seconda procedura “linearizza un jagged array”; in pratica esplora il jagged array registrandone tutti i valori (le foglie) in un unico array monodimensionale. Una possibile implementazione in VBA è mostrata di seguito; si noti l’uso della funzione privata Is_Vector, basata sulla gestione degli errori, ed utilizzata per verificare se una variabile variant sia anche un vettore.

```
Global Single_Vector()
Public Sub Disfa_MV(MV)
Dim i As Integer
Dim B As Boolean
  For i = 1 To UBound(MV)
    If Not Is_Vector(MV(i)) Then
      ReDim Preserve Single_Vector(0 To UBound(Single_Vector) + 1)
      Single_Vector(UBound(Single_Vector)) = MV(i)
    Else
      Call Disfa_MV(MV(i))
    End If
  Next i
End Sub

Private Function Is_Vector(V As Variant) As Boolean
Is_Vector = False
On Error GoTo Err:
  If UBound(V) > -1 Then
    Is_Vector = True
    Exit Function
  End If
Err:
  'Do nothing
End Function
```

La terza ed ultima funzione, crea un jagged array partendo da una codifica alfanumerica del tipo: [1,2,3 [4,5,6 [7,8,9]]], in cui le parentesi quadre servono a indicare dove s'innesta un nuovo vettore. Ad esempio, la codifica precedente corrisponde alla seguente struttura:

```
MV(0) =
        MV(0)(0) = 1
        MV(0)(1) = 2
        MV(0)(2) = 3
        MV(0)(3) =
                   MV(0)(3)(0) = 4
                   MV(0)(3)(1) = 5
                   MV(0)(3)(2) = 6
                   MV(0)(3)(3) =
                                 MV(0)(3)(3)(0) = 7
                                 MV(0)(3)(3)(1) = 8
                                 MV(0)(3)(3)(2) = 9
```

```
Public Function Crea_MV_FS(Code As String, Optional Delimiter1 As String = "[", _
                                                    Optional Delimiter2 As String _ = "]") As Variant
Dim Chr As String, Str As String
Dim MV() As Variant


  ReDim MV(0 To 0)


  Do While Len(Code) > 0
    Chr = Left(Code, 1) 'Prima cifra a partire da sinistra
    Code = Mid(Code, 2, Len(Code) - 1) 'Tutta la parte restante
    If (Asc(Chr) >= 48 And Asc(Chr) <= 57) Then 'Codice ascii di 0, 1, ..., 9
      Str = Str + Chr 'Concateniamo la stringa
    Else
      If Not IsEmpty(MV(UBound(MV))) And (Chr <> Delimiter2 Or Str <> "") Then ReDim Preserve MV(0 To
UBound(MV) + 1)  'Se l'ultimo valore è pieno ne aggiungiamo uno
      If Chr = Delimiter1 Then MV(UBound(MV)) = Crea_MV_FS(Code, Delimiter1, Delimiter2)
      If Str <> "" Then
        MV(UBound(MV)) = Str
        Str = ""
      End If
    End If
  Loop
Crea_MV_FS = MV
End Function
```