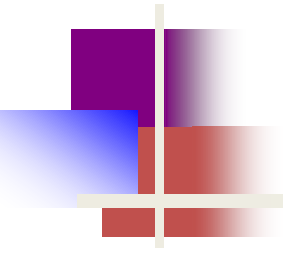


# 第二章
# 陣列(Array)

# 本章學習目標

1. 讓學生了解**一維、二維及多維陣列的結構**及**表示方法**。
2. 讓學生了解**矩陣**中常見的**各種運算( 轉置、相加、相乘及稀疏矩陣)** 。

# 本章內容

1. 陣列的觀念
2. 一維陣列的觀念
3. 二維陣列的觀念
4. 多維陣列的觀念
5. 陣列在記憶體中的表示法
6. 多項式(polynomials)
7. 矩陣(Matrices)
8. 特殊矩陣

# 2-1 陣列的觀念

【定義】陣列是指一群具有相同名稱及資料型態的變數之集合。

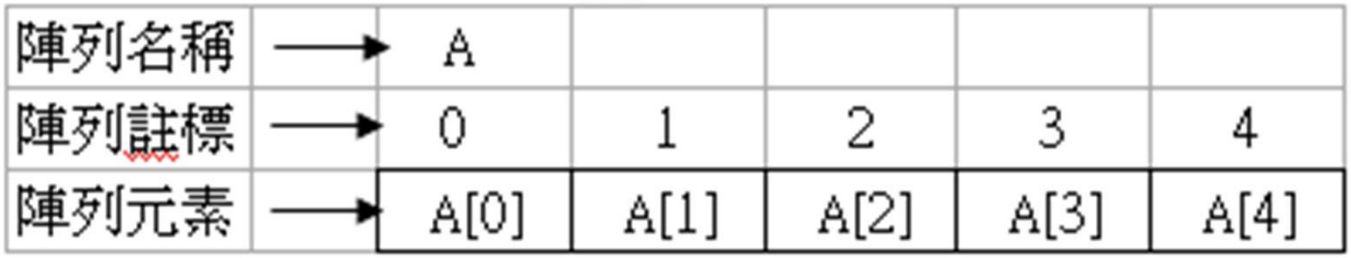
【特性】

1. 佔用連續記憶體空間。
2. 用來表示有序串列之一種方式。
3. 各元素的資料型態皆相同。
4. 支援隨機存取(Random Access)與循序存取(Sequential Access)。
5. 插入或刪除元素時較為麻煩。因為須挪移其他元素。

【例如】

假設我們需要5個整數變數來存放資料時，那就必須要宣告一個A陣列為整數型態，其註標是按照順序排列從0~4共有5項，其含義如下：

int A[5];




【說明】

(1)A陣列表示內共有5個陣列元素，也就是有5個變數，分別為A[0]、A[1]、A[2]……A[4]。

(2) 每一個陣列元素可以存放一筆資料。

【優點】

(1) 利用註標（Index）可以快速的輸入資料。

```
輸入：for(i=0;i<5;i++)   //利用「迴圈結構」
          A[i]=i*2+1;      //快速「輸入資料」到「陣列」中
```

(2) 利用註標（Index）一次可以輸出大批的資料。

```
輸出：for(i=0;i<5;i++)   //利用「迴圈結構」
          Print(A[i]);     //從「陣列」一次「輸出大批」的資料
```

# 2-2 一維陣列

【定義】宣告陣列時，其括弧內的「註標」個數，只有一個時稱為「一維陣列」。

在一維陣列中，常使用的運算指令有五種。

1.讀取(Read)

2.寫入(Write)

3.插入(Insert)

4.刪除(Delete)

5.複製(Copy)

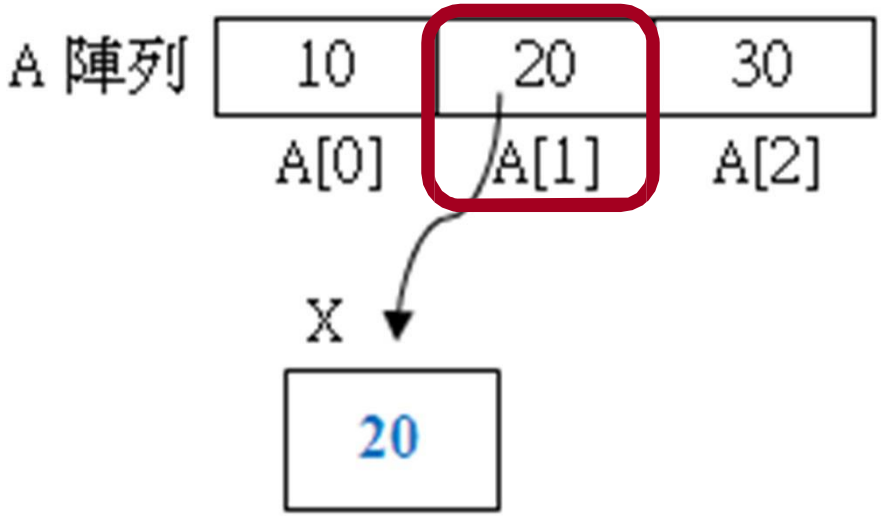
# 1.讀取(Read)

【定義】利用註標（Index）來「讀取」資料。

【例如】將A陣列的第二個元素放到X目的變數中。

【寫法】X= A[1];　　//陣列的註標是從0開始

【圖解】



# 2.寫入(Write)

【定義】利用註標（Index）來「寫入」資料。

【例如】將數值50寫入到陣列的第二個索引位置中。

【寫法】A[1]=50;　//陣列的註標是從0開始

【圖解】

# 3.插入(Insert)

【定義】在指定的註標 i 的位置插入一項新元素，原來註標i和之後的元素都必須要再往後挪移一個位置。

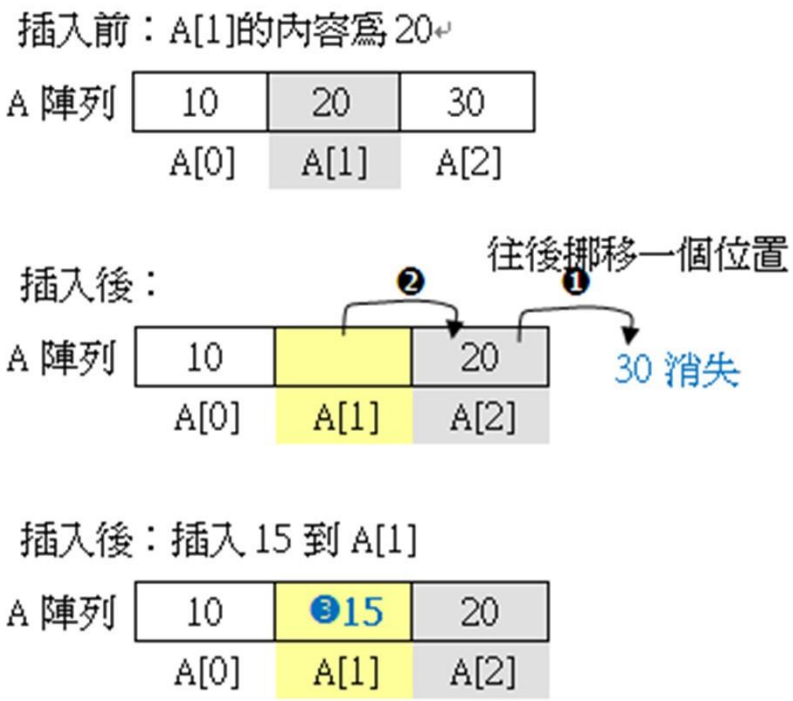
【例如】將在註標1的位置插入一項新元素(15)。

【演算法】

```
01  Procedure ArrayInsert(int A[],int Max,int i ,int value)
02   Begin
03     If(i>0 && i<=Max)                    //判斷欲插入位置i是否存在，如果有，則
04      {
05        for(count=Max-1;count>i;count--)   //i位置及後面的元素逐一往後挪
06           A[count]=A[count-1];
07        A[i]=value;                        //最後再將新元素插入到第i位置
08      }
09    Else                                  //如果欲插入位置i不存在，則
10      Return 0;                           //傳回0
11   End
12  End Procedure
```

# 3.插入(Insert)(續)

【圖解】




【說明】

首先將30往後挪移一個位置，再將A[1]的元素20，往後挪移放到A[2]位置中，最後再插入15到A[1]中。

# 4.刪除(Delete)

【定義】指刪除指定的註標 i 位置的元素，原來註標 i 的元素被刪除，為了避免浪費記憶體空間，因此，之後的元素都必須要再往前挪一個位置。

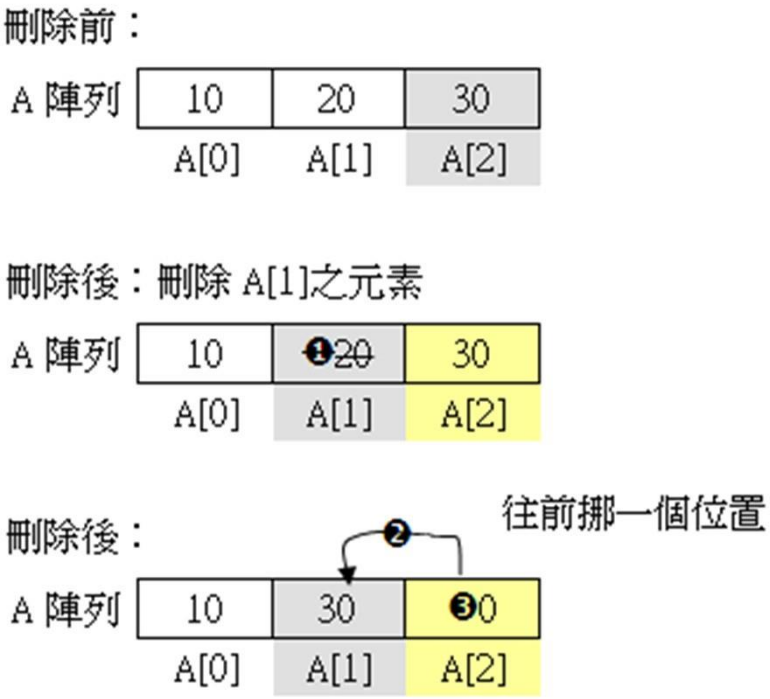
【例如】將在註標1的位置刪除一項舊元素。

【演算法】

```
01  Procedure ArrayDelete(int A[],int Max,int i)
02   Begin
03    If(i>0 && i<=Max)                    //判斷欲刪除元素位置i是否存在，如果有，則
04      {
05        for(count=i;count<Max-1;count++)  //i位置後面的元素逐一往前挪
06           A[count]=A[count+1];
07        A[Max-1]=0;                       //最後再將0放到最後一個位置
08      }
09    Else                                 //如果欲插入位置i不存在，則
10      Return 0;                          //傳回0
11   End
12  End Procedure
```

# 4.刪除(Delete) (續)

【圖解】




【說明】

首先將A[1]的元素20刪除，再將A[2]的元素30往前挪移一個位置，最後再寫入0到A[2]中。

# 5.複製(Copy)

【定義】指將「來源陣列」的元素內含值全部逐一copy到「目的陣列」。

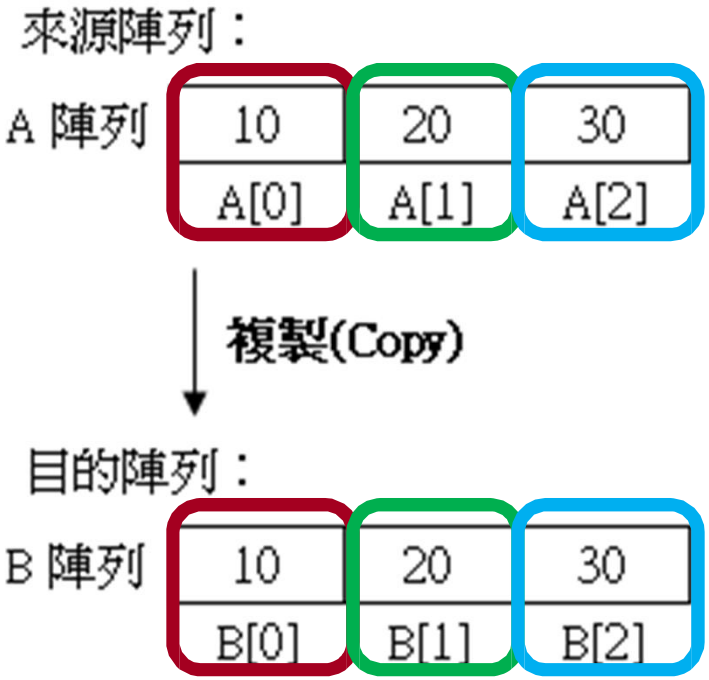
【例如】將A陣列的元素內含值全部逐一copy到B陣列中。

【演算法】

```
01  Procedure ArrayCopy(int A[],int B[], int Max)
02   Begin
03     If(Max>0)                          //判斷陣列是否有元素內含值，如果有，則
04       {
05         for(count=0;count<Max-1;count++)   //A陣列全部逐一copy到B陣列
06             B[count]=A[count];
07       }
08     Else                               //如果陣列沒有元素，則
09       Return 0;                        //傳回0
10   End
11  End Procedure
```

# 5.複製(Copy) (續)

【圖解】




【說明】

A陣列的元素內含值全部逐一copy到B陣列中，例如A[0]元素會被copy到B[0]中，A[1]元素放到B[1]中，以此類推。

# 2-2.1 陣列的宣告

(1) 變數宣告

int A, B, C; //宣告三個變數(A,B,C)為整數型態

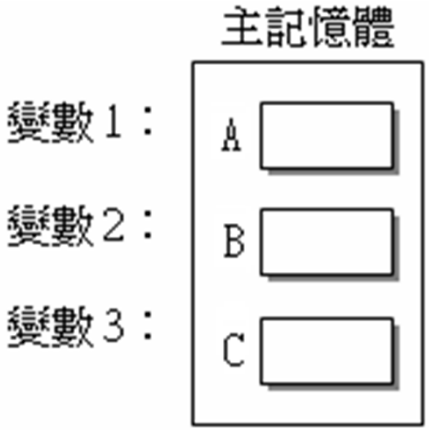



圖2-1 不連續的記憶體空間的配置

說明：以上三個變數與變數之間都是個別獨立的記憶體空間。

# 2-2.1 陣列的宣告 (續)

(2) 陣列宣告

int A[3]; //宣告一維陣列A，共有A[0]、A[1]、A[2]三個元素

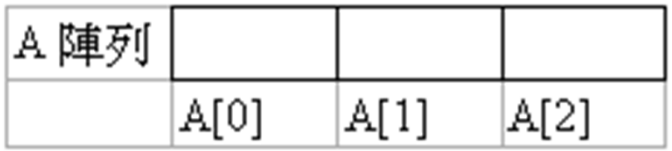



圖2-2 連續的記憶體空間的配置

說明：

以上三個記憶體空間，可以讓我們連續儲存多項資料，

並且資料與資料之間都是按照順序排列的記憶體空間。

# 2-2.2 陣列的儲存方式

【定義】陣列名稱之後加上“註標”即可存取陣列元素。

【舉例】宣告一個A[3]的陣列，並分別儲存10,20,30

```
int A[3];
for(i=0;i<3;i++)
    A[i]=(i+1)*10;      //指把10指定給A陣列中的第0項的資料中
```

| A 陣列 | 10 | 20 | 30 |
|---|---|---|---|
| | A[0] | A[1] | A[2] |

【實例】請依序輸入六位同學的成績到陣列中，並計算及輸出「總和」

第一種寫法：使用陣列，但未使用for迴圈演算法

使用陣列，但未使用 for 迴圈演算法


```
01  Procedure Method1( )
02    Begin
03      int A[6]={60,70,80,85,90,100};                  //輸入(初值設定)
04      sum = A[0] + A[1] + A[2] + A[3] + A[4] + A[5];   //處理
05      printf("總和爲：%3d",sum);                        //輸出
06    End
07  End Procedure
```

## 第二種寫法：使用陣列，並且使用for迴圈演算法(最佳)

使用陣列，並使用 for 迴圈演算法

使用for迴圈演算法

```
01  Procedure Method2()
02    Begin
03      int A[6]={60,70,80,85,90,100};          //輸入(初值設定)
04      for (i = 0; i<=5;i++ )                  //處理
05            sum+=A[i];
06      printf("總和爲：%3d",sum);               //輸出
07    End
08  End Procedure
```

# 2-3 二維陣列的觀念

在前面所介紹一維陣列，可以視為直線方式來存取資料，這對於一般的問題都可以順利的處理，但是對於比較複雜的問題時，那就必須要使用二維陣列來處理。否則會增加程式的複雜度。

例如：計算4位同學的5科成績之總分與平均的問題。

# 2-3 二維陣列的觀念(續)

【定義】宣告陣列時，其括弧內的「註標」個數，有兩個時稱為「二維陣列」。

【語法】資料型態陣列名稱[M][N];

【說明】M代表列數，N代表行數

【存取方法】利用二維陣列中的兩個註標來表示。

# 2-3 二維陣列的觀念(續)

【宣告】int Score[4][5];

//列註標表示範圍： 0~3 共有4列

//行註標表示範圍： 0~4 共有5行

在宣告之後，主記憶的邏輯配置如圖 2-3 所示：

| 列＼行 | 第 0 行 | 第 1 行 | 第 2 行 | 第 3 行 | 第 4 行 |
|---|---|---|---|---|---|
| 第 0 列 | Score [0] [0] | Score [0] [1] | Score [0] [2] | Score [0] [3] | Score [0] [4] |
| 第 1 列 | Score [1] [0] | Score [1] [1] | Score [1] [2] | Score [1] [3] | Score [1] [4] |
| 第 2 列 | Score [2] [0] | Score [2] [1] | Score [2] [2] | Score [2] [3] | Score [2] [4] |
| 第 3 列 | Score [3] [0] | Score [3] [1] | Score [3] [2] | Score [3] [3] | Score [3] [4] |

5行

4列

圖 2-3 二維陣列的邏輯配置

說明：利用二維陣列中的兩個註標來表示。

# 2-4 多維陣列的觀念

【定義】宣告陣列時，其括弧內的「註標」個數，是二個以上時，就稱為「多維陣列」。其中最常見是三維陣列，其圖形為三度空間的立體圖形，並且我們可以將三維陣列視為多個二維陣列的組合。

【語法】資料型態陣列名稱[L][M][N];

【說明】L代表二維陣列個數

M代表列數

N代表行數

【舉例】設計有某一個大學，3次月考，全班4位同學的5科目成績時。

利用三維陣列來存取每人學生的成績。

```
int Score[3][4][5];
```


**【說明】**

此例子中Score陣列共有三個註標，故Score陣列是一個三維陣列。

//其中，第一個註標為：二維陣列的個數： 0~2 共有3個二維陣列

第二個註標為：列註標表示範圍： 0~3 共有4列

第三個註標為：行註標表示範圍： 0~4 共有5行

【圖解】

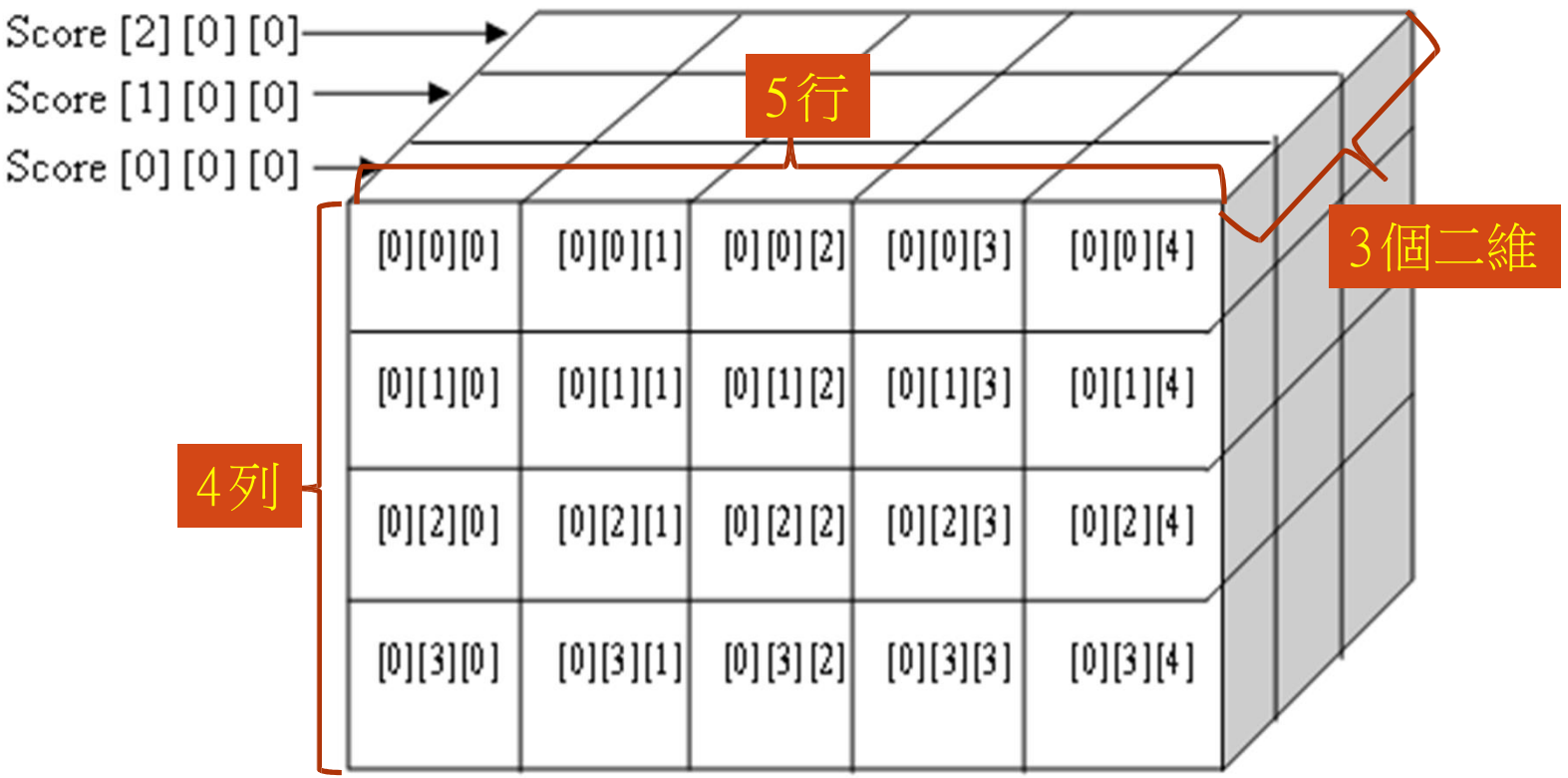



圖 2-4 三維陣列的邏輯配置

【說明】宣告Score是由3個(0~2)二維陣列，每個二維陣列包含4列(0~3)，5行(0~4)組合而成的整數三維陣列。

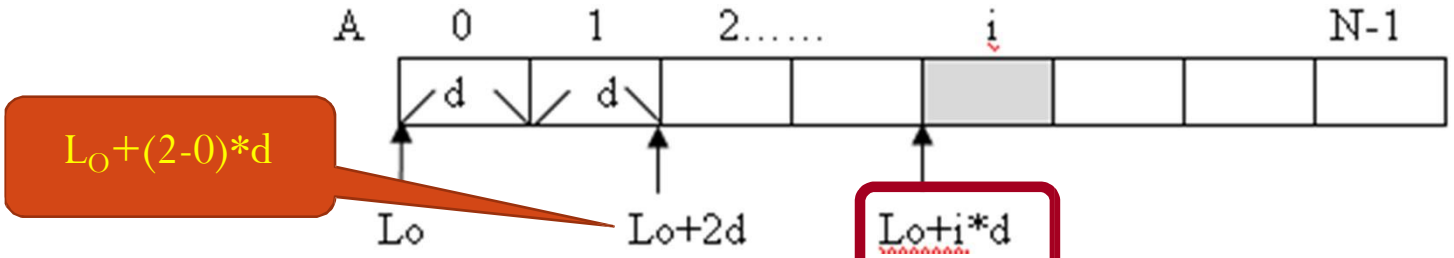
並且共計有 3×4×5 ＝ 60元素。

# 2-5 陣列在記憶體中的表示法

陣列是由一連串的記憶體組合而成，其陣列元素之儲存位址計算，大致上，可分為一維陣列與二維陣列來說明：

Ⅰ. 一維陣列

[題目1]若陣列A有N個元素，其陣列的起始位址為Lo，並且索引值從0開始，d 為元素大小，則A[i]的起始位置為多少？




令： Lo為起始位址

d為元素大小

則A[i]之位置計算=Lo+i*d

【舉例】假設每一個整數佔用2個byte，若A陣列的起始位址是100開始，則A[5]的起始位址為多少？

令：起始位址Lo=100

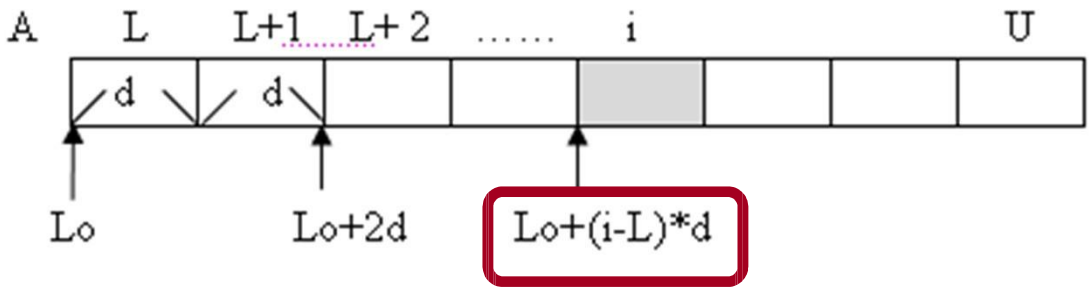
元素大小d=2

則A[5]之位置計算=Lo+i*d
=100+5*2
=100+10=110

[題目2]若陣列A的索引從L到U，其陣列的起始位址為Lo，

d為元素大小，則A[i]的起始位置為多少？

宣告方式：A[L…U]




令：□Lo為起始位址

d為元素大小

則A[i]之位置計算=Lo+(i-L)*d

【舉例】假設每一個整數佔用2個byte，若A[10]起始位址是200開始，

則A[20]的位址為多少？

令：Lo起始位址=200

d元素大小=2

則A[20]之位置計算=Lo+(i-L)*d
=200+(20-10)*2
=200+10*2=220

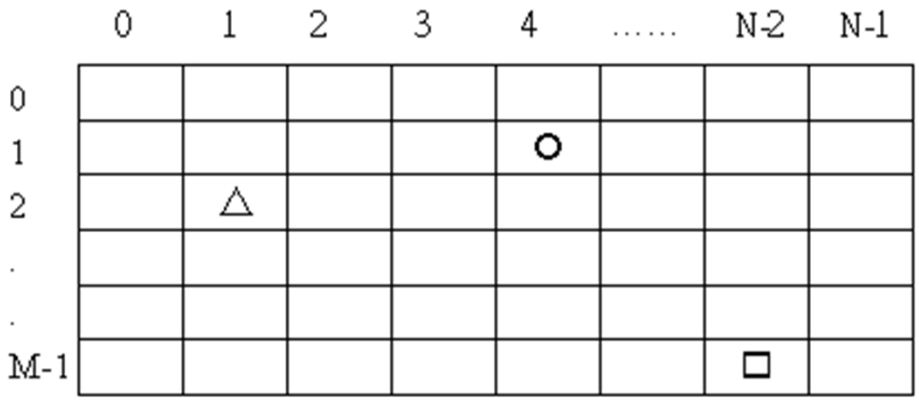
## Ⅱ. 二維陣列

宣告方式：A[0…M-1, 0…N-1]

其中：M代表列數(Row)，橫向。

N代表行數(Column)，縱向。

所以，共有M*N格。




說明：O圖的儲存位置：A[1,4]

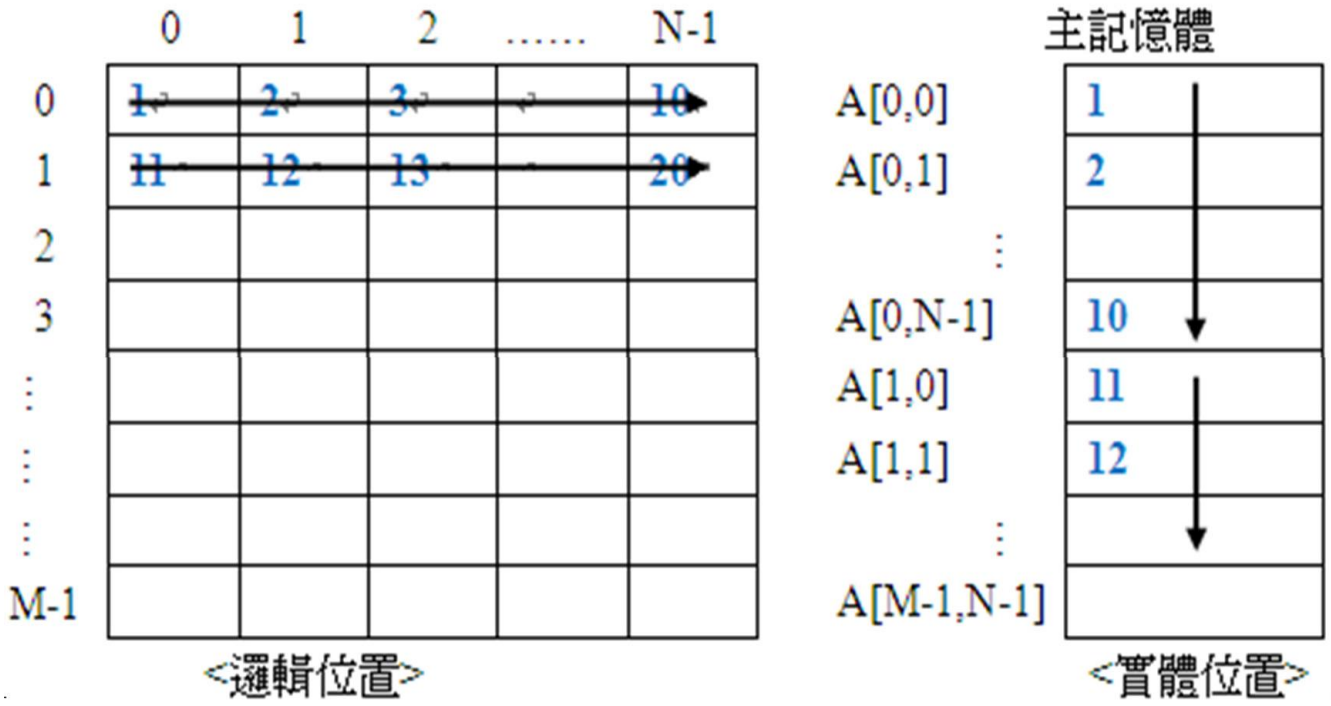
△圖的儲存位置：A[2,1]

□圖的儲存位置：A[M-1,N-2]

# 2-5.1 Row-major(以列為主)

【定義】以列為主的二維陣列要轉為一維陣列時，是將二維陣列「由上往下」一列一列讀入一維陣列。亦即將二維陣列儲存的邏輯位置轉換成實際電腦中主記憶體的存儲方式。

【圖解】



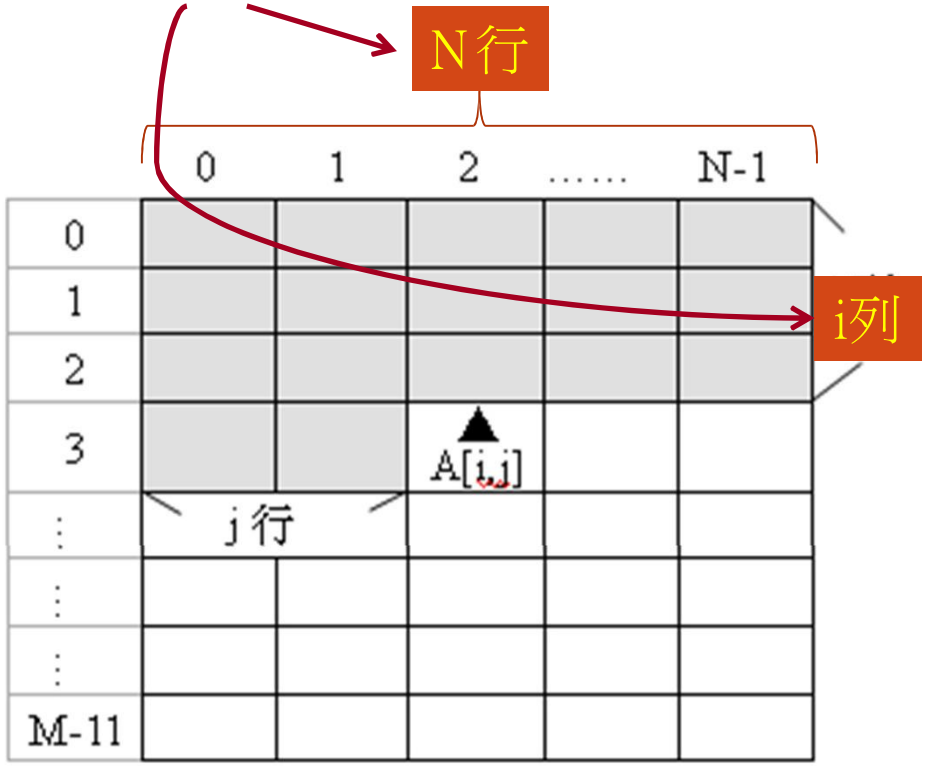
【以列為主的儲存公式】

令Lo為起始位址

d為元素大小

則二維陣列A[i,j]位置會儲存到一維陣列的那一個位置呢？

公式=Lo+[i*N+j]*d



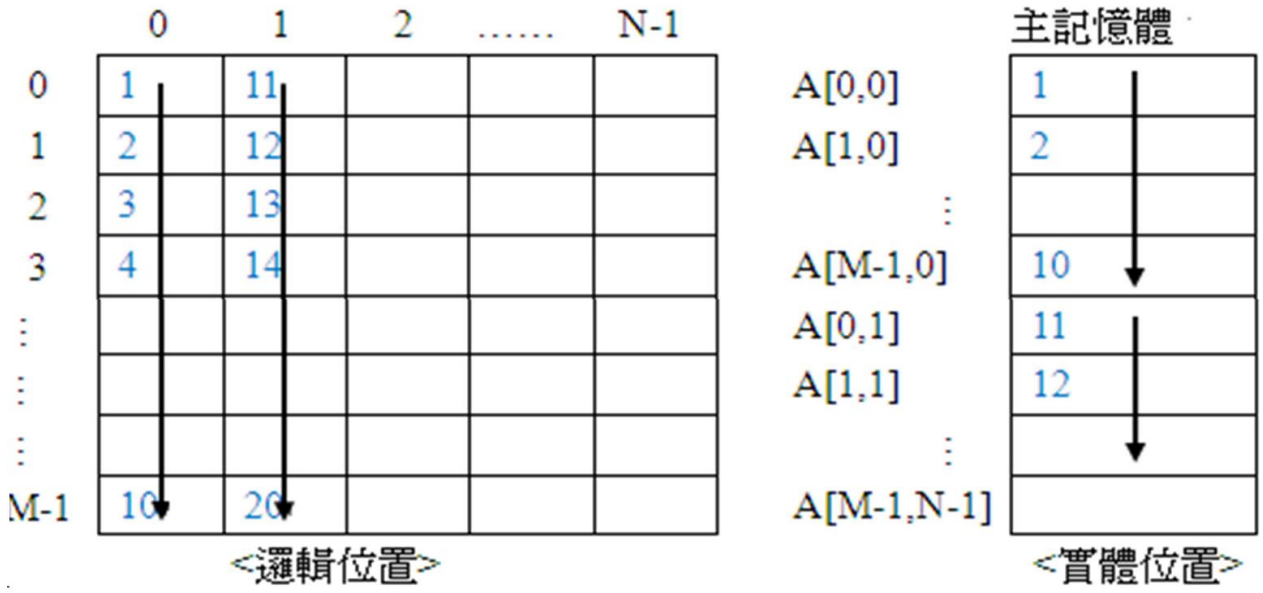

▲的位置爲：A[3,2]=Lo+[3*N+2]*d=Lo+(3N+2)*d

因此，當 Lo=1, N=10，d=1 時，則 A[3,2]=1+(3*10+2)*1=33

# 2-5.2 Column-major(以行為主)

【定義】以行為主的二維陣列要轉為一維陣列時，必須將二維陣列「由左往右」一行一行讀入一維陣列。亦即將二維陣列儲存的邏輯位置轉換成實際電腦中主記憶體的存儲方式。

【圖解】



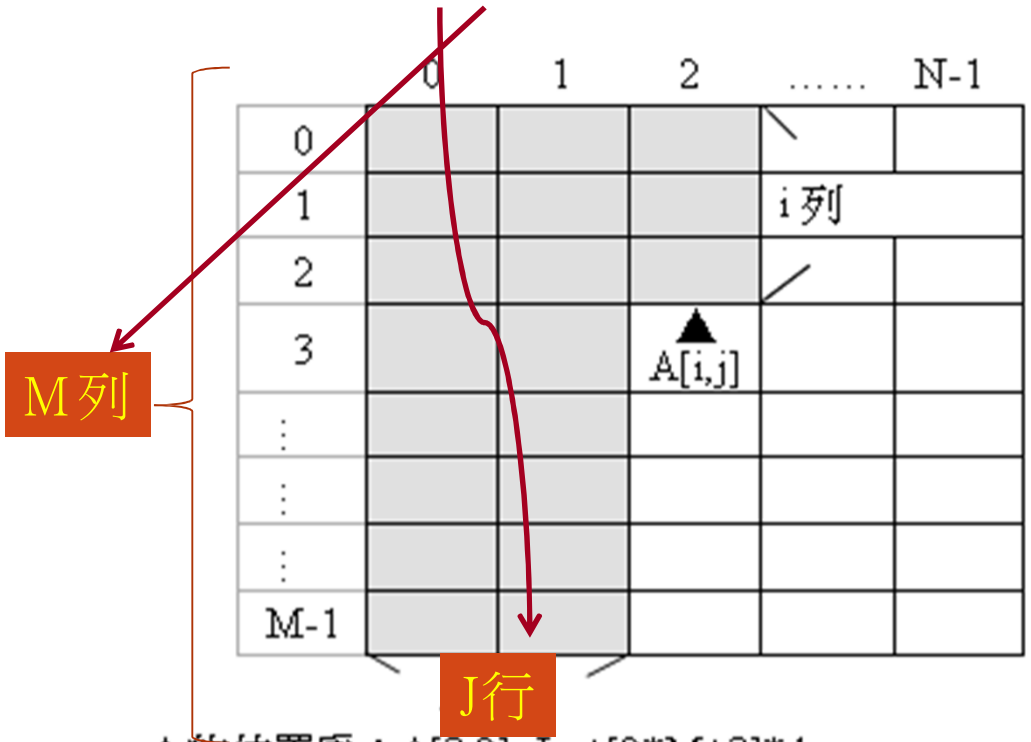
【以行為主的儲存公式】

令Lo為起始位址

d為元素大小

則二維陣列A[i,j]位置會儲存到一維陣列的那一個位置呢？

公式=Lo+[ j*M+i]*d




▲的位置爲：A[3,2]=Lo+[2*M+3]*d

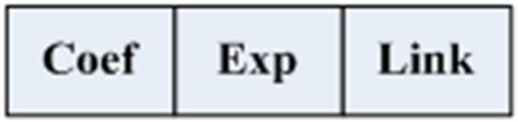
因此，當 Lo=1, M=10，d=1 時，則 A[3,2]=1+(2*10+3)*1=24

# 2-6 多項式(polynomials)

多項式(polynomial)的表示式為$P(x)=A_mX^m+A_{m-1}X^{m-1}+.....+A_1X^1+A_0X^0$，

其中Ai為非零項的係數，且多項式的每一項均使用三個欄位來表示

(分別為coef, exp, link)。其節點的資料結構如下所示：




其中：Coef：表示該變數的係數

Exp：表示該變數的指數

Link：表示指向下一個節點的指標

## 【表示方法】

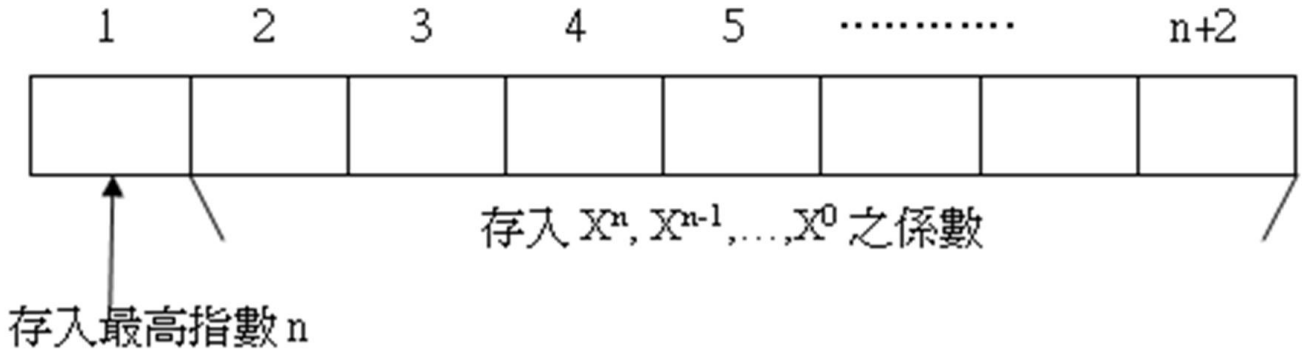
【方法一】依照指數高低依序儲存係數

【方法二】只儲存非零項次的係數與指數

# 【方法一】依照指數高低依序儲存係數

【作法】假設最高指數為n，則準備一個一維陣列A[1..n+2]，

其內容如下：

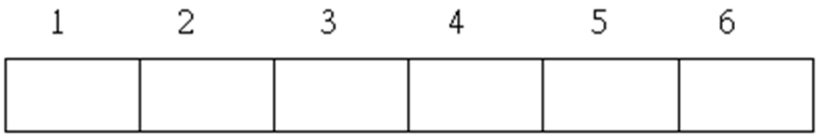


【練習】假設$f(x)=7X^4+5X^2+3X$

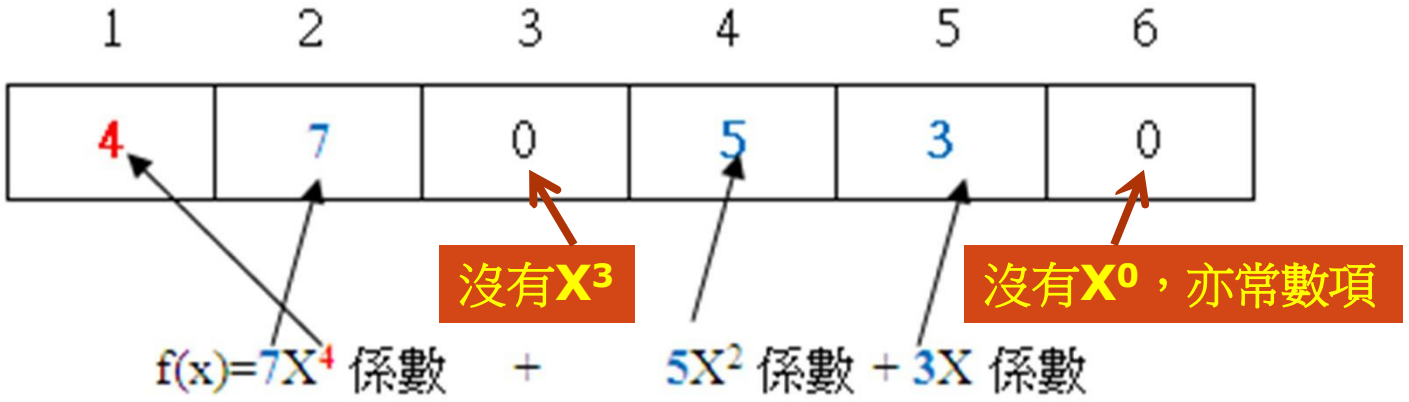
因為最高指數為4，則準備一個一維陣列A[1..6]，

【解答】

步驟一：準備一個一維陣列A[1..6]




步驟二：存入最高指數及分別存入$X^n, X^{n-1},...,X^0$之係數



【優點】

(1)只要儲存係數，比較節省儲存指數空間。

(2)適用於零項次較少的多項式。

【缺點】

不適用於零項次極多的多項式，儲存時非常浪費空間。

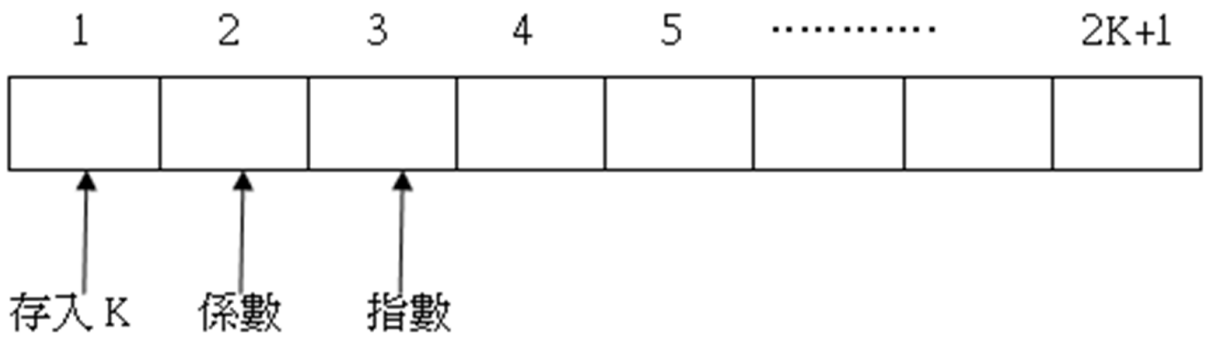
Ex: $f(X)=5X^{100}+1$

則必須要準備一個一維陣列A[1..102]，在實際使用上只用 了 3格，因此，會浪費99格。

# 【方法二】只儲存非零項次的係數與指數

【作法】假設多項式有K個非零項次，則準備一個一維陣列

A[1..2K+1]，其內容如下：

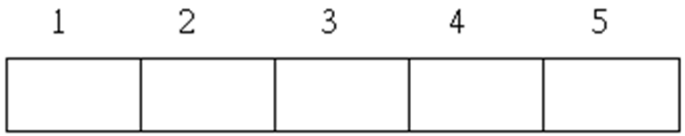


【練習】假設$f(X)=5X^{100}+1$

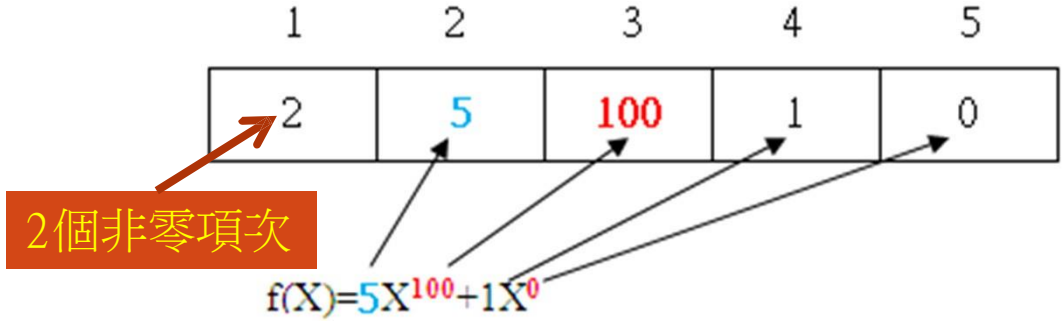
因為有2個非零項次，則準備一個一維陣列A[1..5]

【解答】

步驟一：準備一個一維陣列A




步驟二：存入K、係數及指數




【優點】適用於零項次極多的多項式。

【缺點】當非零項次極多時，不適用。

# 2-7 矩陣(Matrices)

【定義】

類似二維陣列，它是利用一個m × n矩陣來表示這個矩陣擁有m列（Rows）和n行（Columns）。

一般而言，資料結構上常用到的矩陣有四種：

1. 矩陣轉置(Matrix Transposition)
2. 矩陣相加(Matrix Addition)
3. 矩陣相乘(Matrix Multiplication)
4. 稀疏矩陣(Sparse Matrix)

# 2-7.1 矩陣轉置(Matrix Transposition)

【定義】

假設有一個(m × n)矩陣A，則我們可以將A矩陣轉置為(n × m)的B矩陣，並且B矩陣的第j列第i行的元素等於A矩陣的第i列第j行的元素，

以數學式來表示為：$B_{ji}=A_{ij}$

【假設】A矩陣與B矩陣的m 與n都是從1 開始計算，因此，A,B矩陣的表示如下：

$$A=\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}_{m\times n} \qquad B=A^t=\begin{bmatrix} A_{11} & A_{21} & \cdots & A_{m1} \\ A_{12} & A_{22} & \cdots & A_{m2} \\ \cdots & \cdots & \cdots & \cdots \\ A_{1n} & A_{2n} & \cdots & A_{mn} \end{bmatrix}_{n\times m}$$

例如：

$$A=\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times 3} \qquad B=A^t=\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}_{3\times 2}$$

說明：A矩陣的第i列第j行的元素等於B矩陣的第j列第i行的元素

# 【演算法】

「矩陣轉置」之演算法

```
01  Procedure Matrix_Transpose(int m, int n, int A[m][n], int B[n][m])
02   Begin
03    for(i = 1; i <= m; i++)        //外迴圈，先掃瞄第1列到第m列
04      for(j = 1; j <= n; j++)      //內迴圈，再掃瞄第1行到第n行
05          //將A陣列的第i列第j行的元素放到B陣列的第j列第i行的元素中
06          B[j][i] = A[i][j];
07   End
08  End Procedure
```

# 2-7.2 矩陣相加(Matrix Addition)

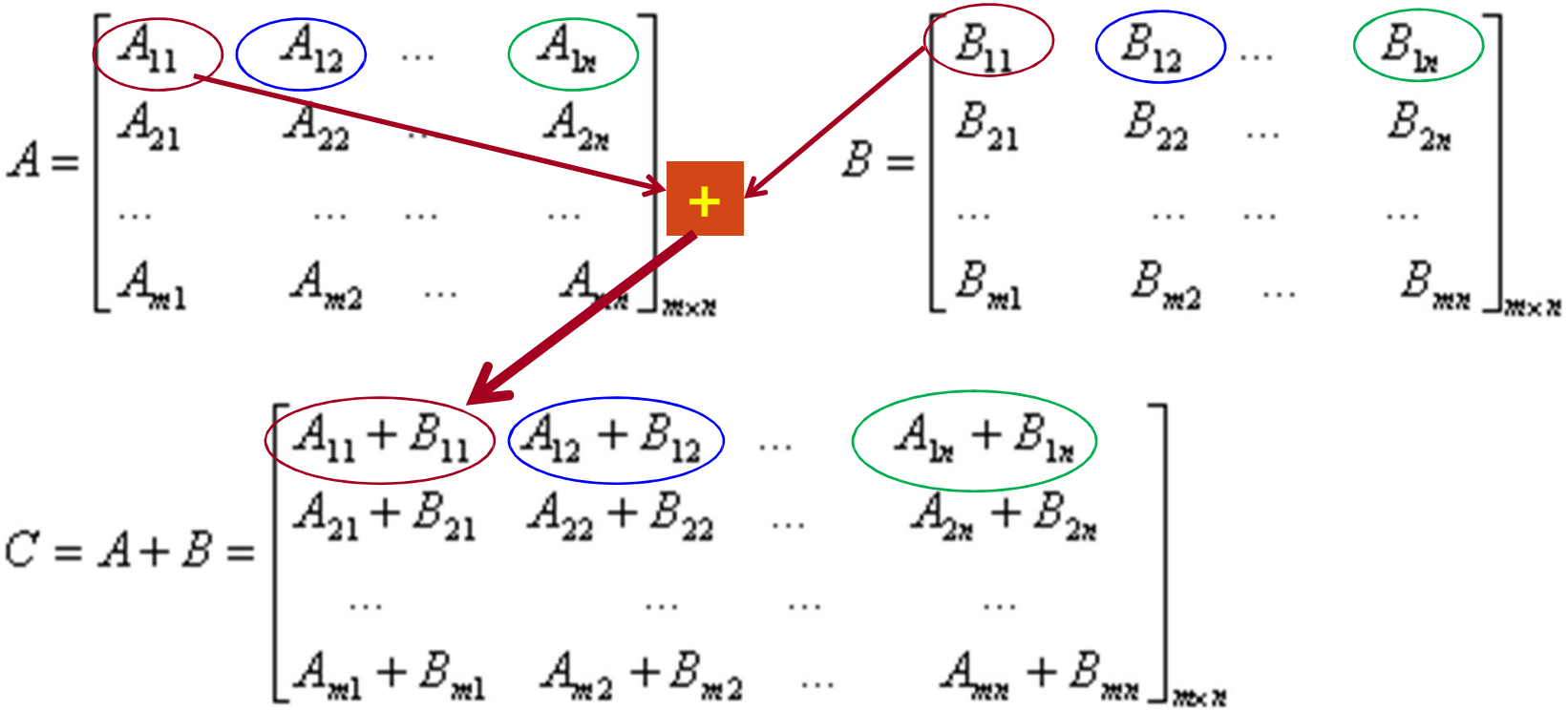
【定義】

假設A,B都是(m × n)矩陣，則我們可以將A矩陣加上B矩陣以得到一個C矩陣，並且此C矩陣亦為(m × n)矩陣。因此：

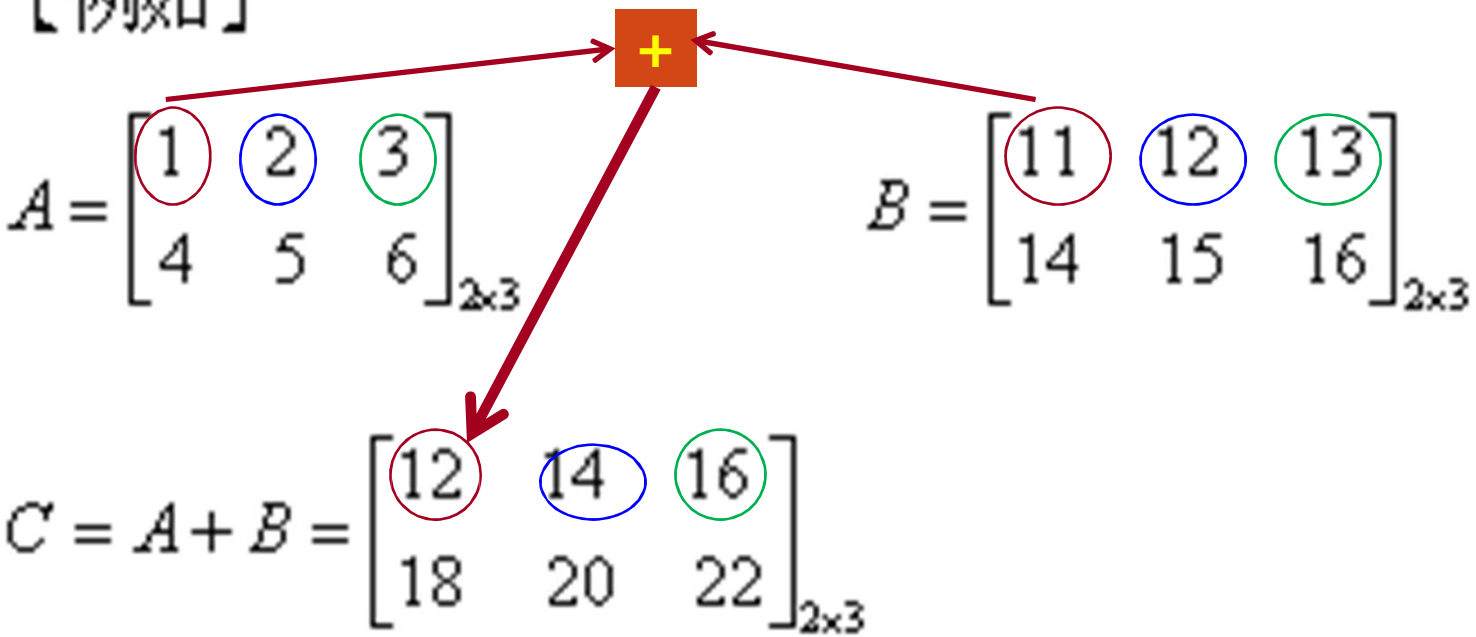
在C矩陣上的第i列第j行的元素必定等於A矩陣的第i列第j行的元素加上B矩陣的第i列第j行的元素。

以數學式來表示為：$C_{ij} = A_{ij} + B_{ij}$

【假設】A、B、C矩陣的m與n都是從1開始計算，因此，A,B兩個矩陣相加等於C矩陣，其表示如下：



【例如】


+
$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}_{2\times3}$
$B=\begin{bmatrix}11 & 12 & 13\\ 14 & 15 & 16\end{bmatrix}_{2\times3}$
$C=A+B=\begin{bmatrix}12 & 14 & 16\\ 18 & 20 & 22\end{bmatrix}_{2\times3}$

# 【演算法】

「矩陣相加」之演算法

```
01  Procedure Matrix_Add(int m, int n, int A[m][n], int B[m][n], int C[m][n])
02   Begin
03     for(i = 1; i <= m; i++)          //外迴圈，先掃瞄第1列到第m列
04      for(j = 1; j <= n; j++)         //內迴圈，再掃瞄第1行到第n行
05        /*將A陣列的第i列第j行的元素加上B陣列的第i列第j行的元素之後，
06          放到C陣列的第i列第j行的元素中 */
07        C[i][j] = A[i][j] + B[i][j];
08   End
09  End Procedure
```

# 2-7.3 矩陣相乘(Matrix Multiplication)

【定義】

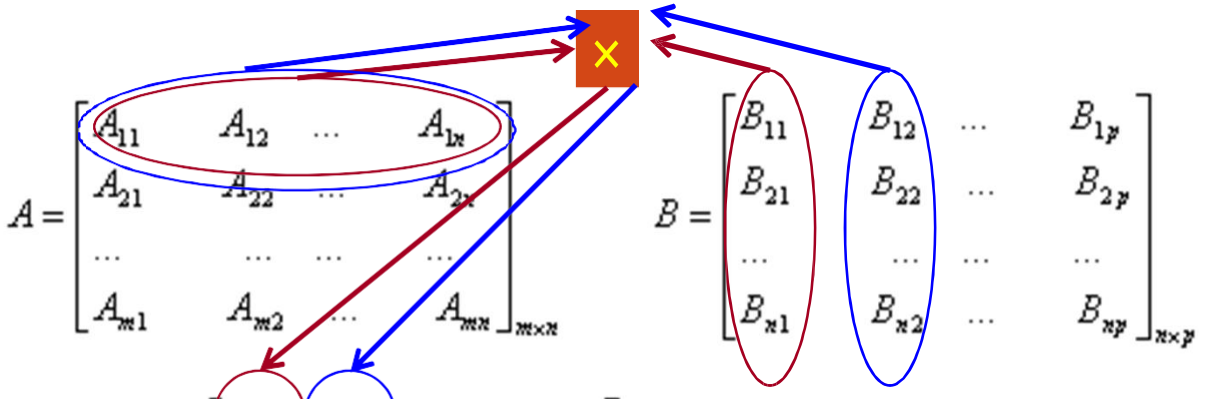
假設A為(m × n)矩陣，而B為(n × p)矩陣，

則我們可以將A矩陣乘上B矩陣以得到一個(m × p)的C矩陣。

因此，在C矩陣上的第i列第j行的元素必定等於A矩陣的第i列乘上B矩陣的第j行(兩個向量的內積)，以數學式來表示為：

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

【假設】A、B、C矩陣的m與n都是從1開始計算，因此，A,B兩個矩陣相乘等於C矩陣，其表示如下：



$$A=\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}_{m\times n} \qquad B=\begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ B_{n1} & B_{n2} & \cdots & B_{np} \end{bmatrix}_{n\times p}$$

$$C=A\times B=\begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1P} \\ C_{21} & C_{22} & \cdots & C_{2P} \\ \cdots & \cdots & \cdots & \cdots \\ C_{m1} & C_{m2} & \cdots & C_{mp} \end{bmatrix}_{m\times p}$$

$$\text{其中}C_{ij}=\begin{bmatrix} A_{i1} & A_{i2} & \cdots & A_{in} \end{bmatrix}\times\begin{bmatrix} B_{1j} \\ B_{2j} \\ \vdots \\ B_{nj} \end{bmatrix}$$

$$=A_{i1}\times B_{1j}+A_{i2}\times B_{2j}+\cdots\cdots+A_{in}\times B_{nj}$$

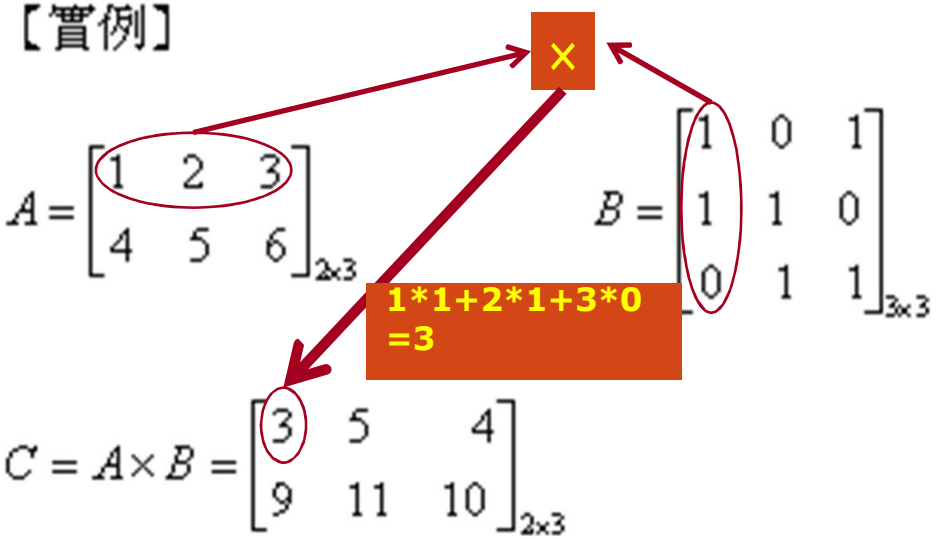
$$=\sum_{k=1}^{n} A_{ik}\times B_{kj}$$

【例如】

$$C_{11} = [A_{11} \quad A_{12} \quad \cdots \quad A_{1n}] \times \begin{bmatrix} B_{11} \\ B_{21} \\ \vdots \\ B_{n1} \end{bmatrix}$$

$$= A_{11} \times B_{11} + A_{12} \times B_{21} + \cdots\cdots + A_{1n} \times B_{n1}$$

【實例】




$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times3} \qquad B = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}_{3\times3}$$

$$C = A \times B = \begin{bmatrix} 3 & 5 & 4 \\ 9 & 11 & 10 \end{bmatrix}_{2\times3}$$

# 【演算法】

「矩陣相乘」之演算法

```
01  Procedure Matrix_Mul(int m, int n, int p, int A[m][n], int B[n][p], int C[m][p])
02   Begin
03   for(i = 0; i < m; i++)      //外迴圈，先掃瞄第1列到第m列
04    for(j = 0; j < n; j++)     //內迴圈，再掃瞄第1行到第n行
05     {
06       C[i][j]=0;
07          /*將A矩陣的第i列第j行的元素加上B矩陣的第i列第j行的
08            元素的結果放到C矩陣上的第i列第j行的元素中*/
09       for(k=0;k<p;k++)
10         C[i][j] = C[i][j] + A[i][k] * B[k][j];
11     }
12   End
13  End Procedure
```

# 2-7.4 稀疏矩陣(Sparse Matrix)

【定義】是指矩陣中大部份元素都沒有使用，元素稀稀落落，

所以稱為稀疏矩陣。

【概念】在M × N 的矩陣中，多數的資料值為0 。

【處理方法】

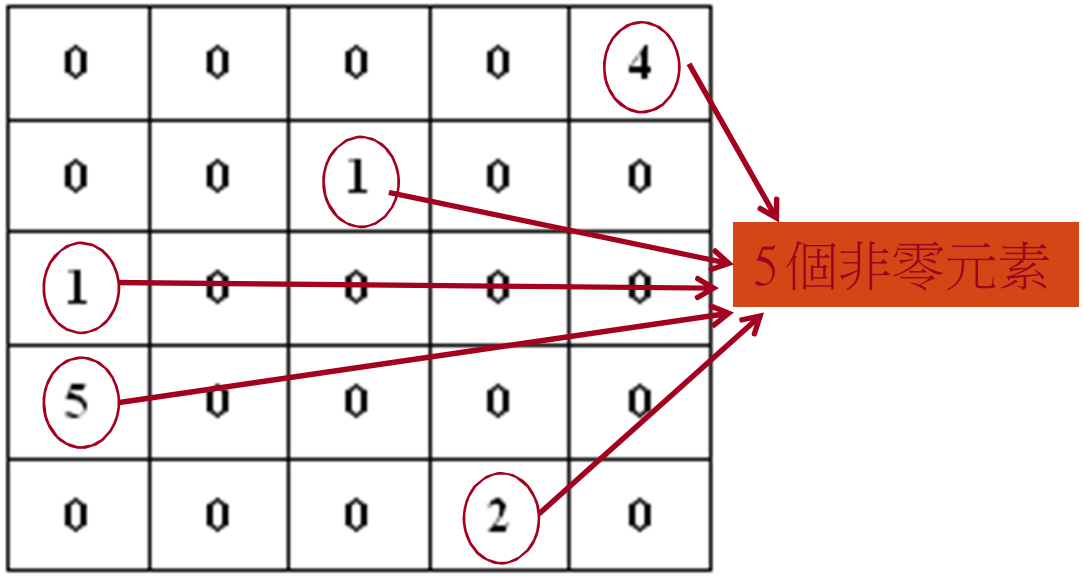
【方法一】直接利用M × N的二維陣列來一一對應儲存。

【方法二】利用3-tuple結構來儲存非零元素

【方法一】直接利用M × N的二維陣列來一一對應儲存。

【缺點】

1. 浪費空間：因為多數為0。
2. 浪費時間：因為要處理一些不必要的計算。



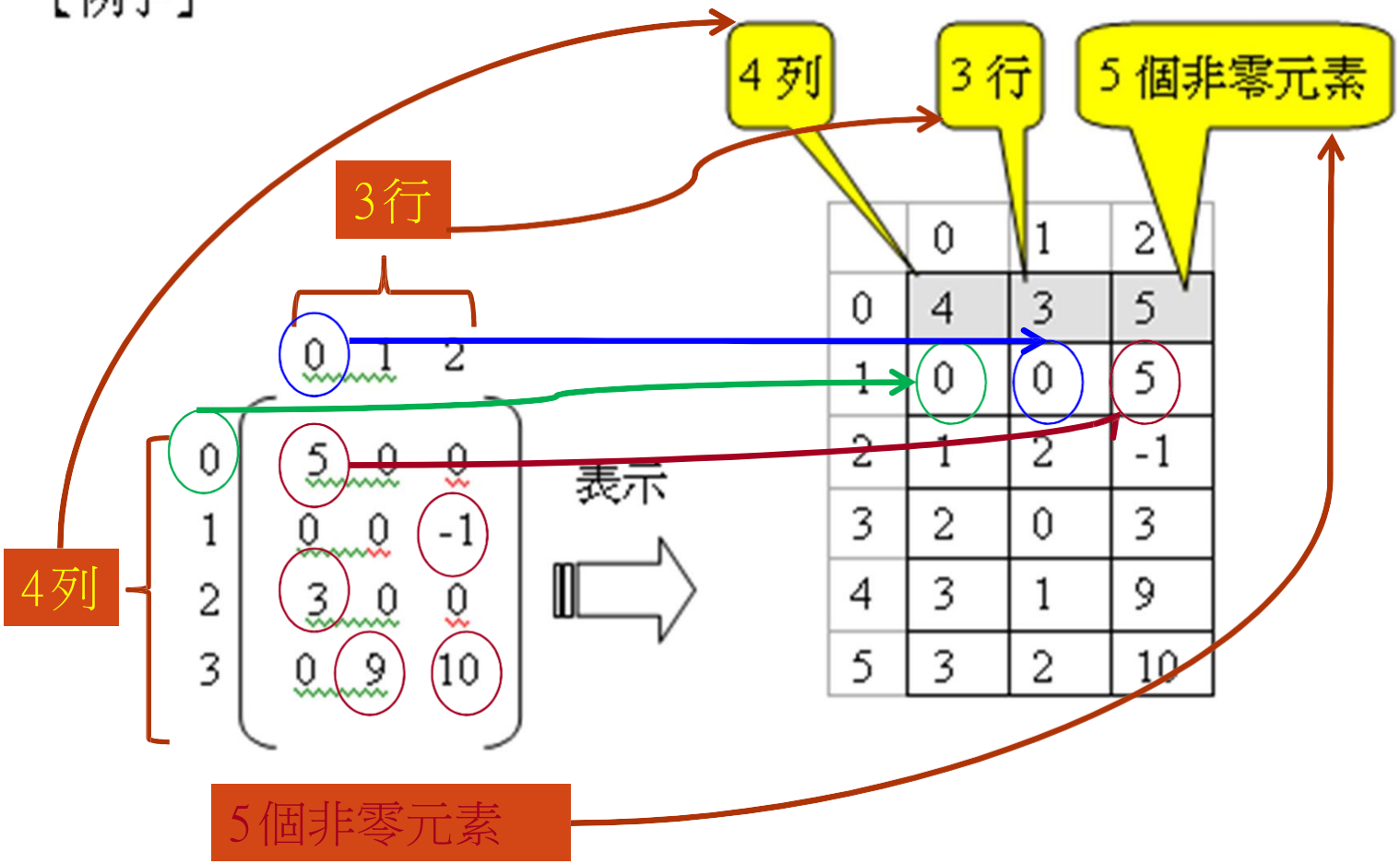

多數為0，所以浪費「空間」與「時間」

【方法二】利用3-tuple結構來儲存非零元素

【作法】假設有一個M*N的稀疏矩陣中共有K個非零元素，則必須要準備一個二維陣列A[0..K,0..2]


【優點】只儲存非0之資料，因此，可以減少記憶體空間的浪費。

【例子】


4 列
3 行
5 個非零元素
3行
0 1 2
0
1
2
3
5 0 0
0 0 -1
3 0 0
0 9 10
4列
表示
5個非零元素
0 1 2
0 4 3 5
1 0 0 5
2 1 2 -1
3 2 0 3
4 3 1 9
5 3 2 10

# 2-8 特殊矩陣

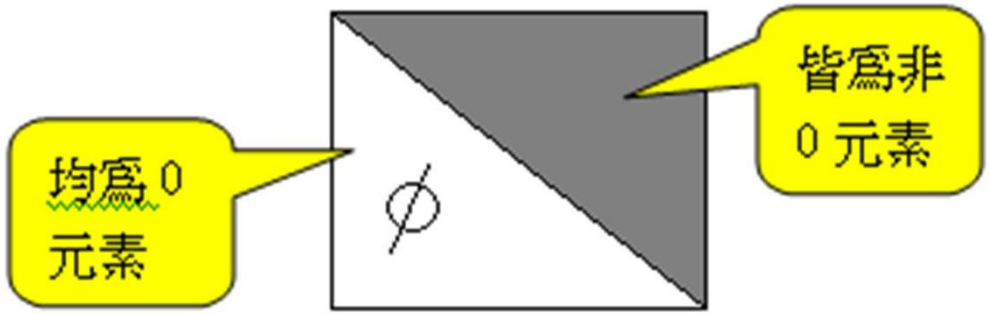
在前面已經介紹資料結構中，常用的四種矩陣之外，我們在本單元中，將介紹比較特殊的矩陣，例如：上、下三角矩陣。

## 2-8.1 上三角矩陣(Upper-Triangular Matrix)

(一)定義：

上三角矩陣是矩陣在對角線以下的元素均為0，即$A_{ij} = 0$，$i > j$。

【概念圖】



(二)元素個數(非 0 元素)=1+2+3+…+n= $\frac{n(n+1)}{2}$

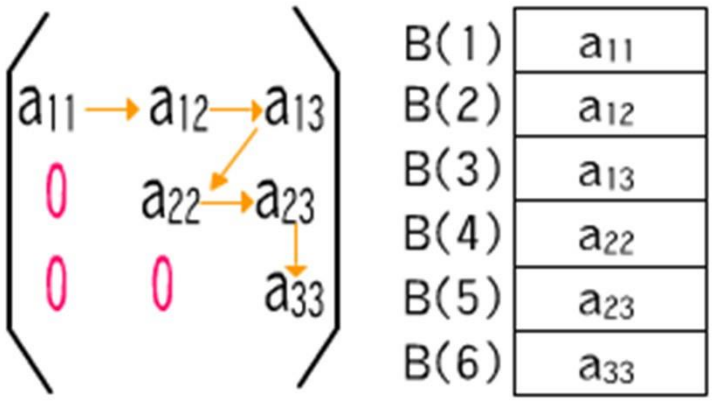
0 元素個數=$n^2$- $\frac{n(n+1)}{2}$

【例如】

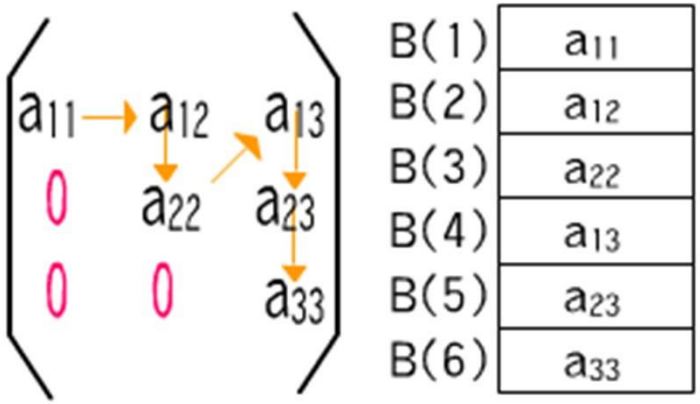
| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 |
| 2 | **0** (2,1) | 6 | 7 | 8 | 9 |
| 3 | **0** (3,1) | **0** (3,2) | 10 | 11 | 12 |
| 4 | **0** (4,1) | **0** (4,2) | **0** (4,3) | 13 | 14 |
| 5 | **0** (5,1) | **0** (5,3) | **0** (5,3) | **0** (5,4) | 15 |

(三)上三角矩陣的儲存

1.以列爲主




2.以行爲主

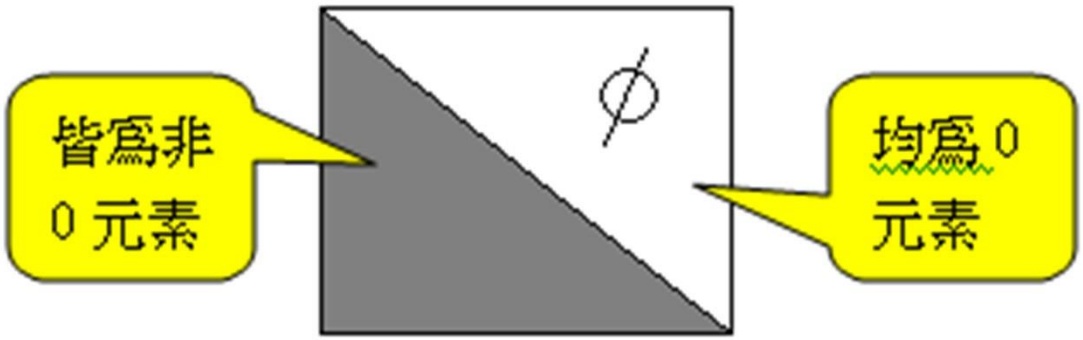


## 2-8.2 下三角矩陣(Lower-Triangular Matrix)

(一)定義：下三角矩陣是矩陣在對角線以上的元素均為0，

即**Aij = 0 ，i < j。**

【概念圖】



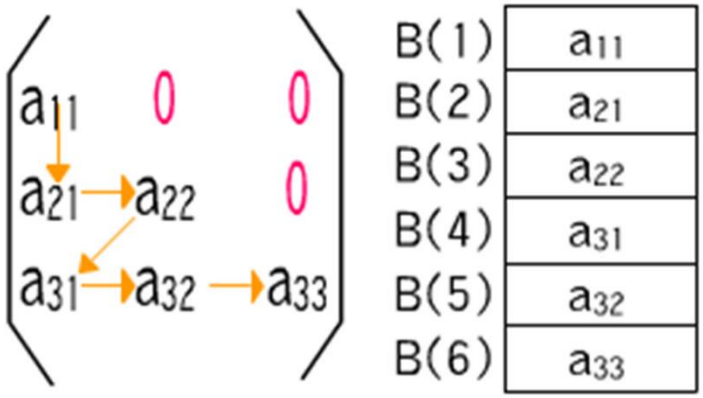
(二)元素個數(非 0 元素)=1+2+3+…+n= $\frac{n(n+1)}{2}$

【例如】

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | **0** (1,2) | **0** (1,3) | **0** (1,4) | **0** (1,5) |
| 2 | 2 | 6 | **0** (2,3) | **0** (2,4) | **0** (2,5) |
| 3 | 3 | 7 | 10 | **0** (3,4) | **0** (3,5) |
| 4 | 4 | 8 | 11 | 13 | **0** (4,5) |
| 5 | 5 | 9 | 12 | 14 | 15 |

(三)下三角矩陣的儲存

1.以列爲主




2.以行爲主