

CH02 Array

陣列(Array)

- An array is a set of pairs, <index, value>, usually implemented as a consecutive memory locations.
- Two standard operations:
 - Retrieve a value
 - Store aa value

```
public abstract class GeneralArray {  
    GeneralArray(int RangeList list, int initValue);  
    public abstract int retrieve(index i);  
    public abstract void store(index l, int value);  
}
```

Other operations (<https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>)

陣列的表示法-

一維陣列 (one dimension array)

- 若陣列是 $A(0 : u-1)$ ，並假設每一個元素佔 d 個空間，則 $A(i) = a_0 + i * d$ ，其中 a_0 是陣列的起始位置。
- 若 $d=1$ 則陣列個元素位址如下

陣列元素：	$A(0)$	$A(1)$	$A(2)$...	$A(i)$...	$A(u-1)$
位 址：	a_0	a_0+1	a_0+2	...	$a_0+(i)$...	$a_0+(u-1)$

陣列的表示法

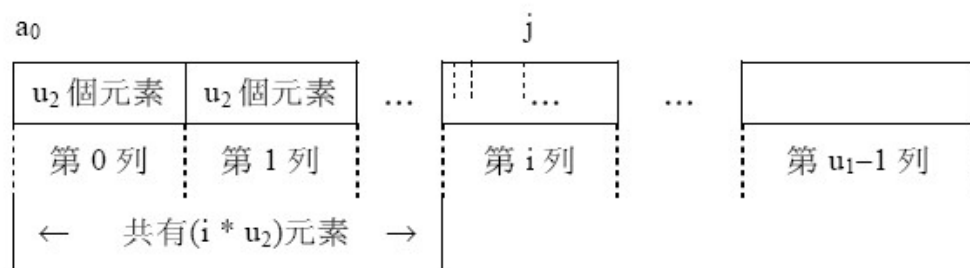
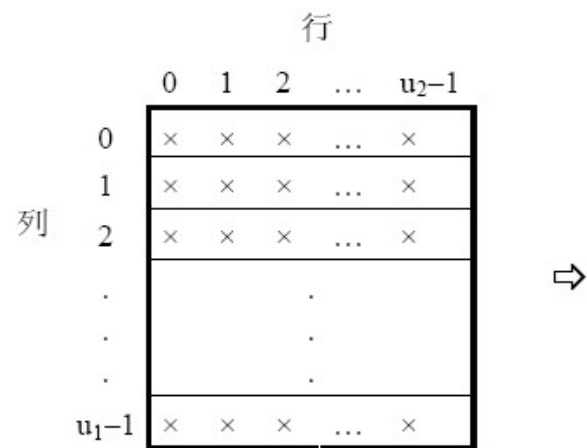
二維陣列（two dimension array）

- 假若有一陣列是 $A[0 : u_1 - 1, 0 : u_2 - 1]$ ，表示此陣列有 u_1 列及 u_2 行；每一列是由 u_2 個元素組成。
- 二維陣列化成一維陣列時，對映方式有二種：一種以列為主（row-major），二為以行為主（column-major）。

陣列的表示法

二維陣列 - 以列為主

- 視此陣列有 u_1 個元素 $0, 1, 2, \dots, u_1-1$ ，每一元素有 u_2 個單位，每個單位佔 d 個空間。
其情形如下圖2-1所示：
- 由圖知 $A(i, j) = a_0 + i * u_2 * d + j * d$ ，其中 a 為此陣列第一個元素的位址。

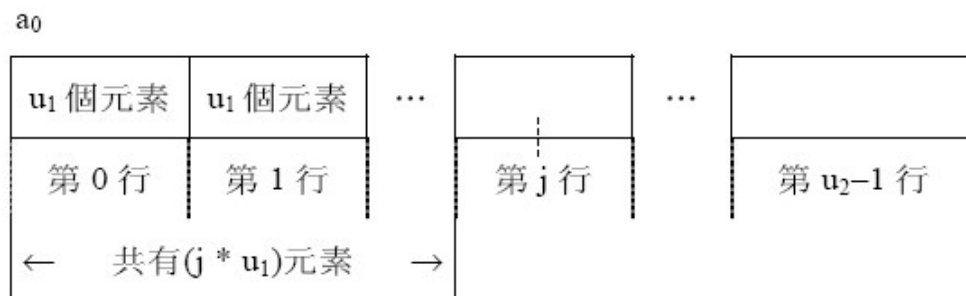


陣列的表示法

二維陣列 - 以行為主

- 視此陣列有 u_2 個元素 $0, 1, 2, \dots, u_2$ ，其中每一元素含有 u_1 個單位，每單位佔 d 個空間，其情形如下圖所示：
- 由圖知 $A(i, j) = a_0 + j * u_1 d + i * d$ 。

		行				
		0	1	2	...	u_2-1
列	0	×	×	×		×
	1	×	×	×		×
	2	×	×	×		×
	⋮	⋮	⋮	⋮	...	⋮
	⋮	⋮	⋮	⋮		⋮
	u_1-1	×	×	×		×



作業02-01

1. 有一陣列 $A(0:100)$ ， $A(0)$ 位址是100， d 為2，試問 $A(16)$ 的位址？
2. 有一陣列 $A(-3:10)$ ， $A(-3)$ 位址是100， d 為1，試問 $A(5)$ 的位址？
3. 有一二維陣列 $A(-3:5, -4:2)$ ， $A(-3,-4)$ 位址是100， d 為1及以列為主，試問 $A(1,1)$ 的位址？

陣列的表示法

三維陣列

- 一般三維陣列皆先化為二維陣列後再對映到一維陣列，對映方式也有二種：
 - 以列為主；
 - 以行為主。

假若有一三維陣列 $A[0:u_1-1, 0:u_2-1, 0:u_3-1]$ ，如圖 2-3 所示：

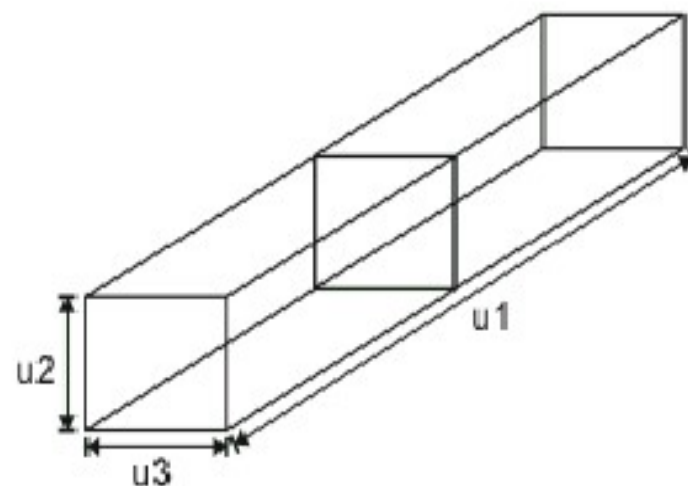
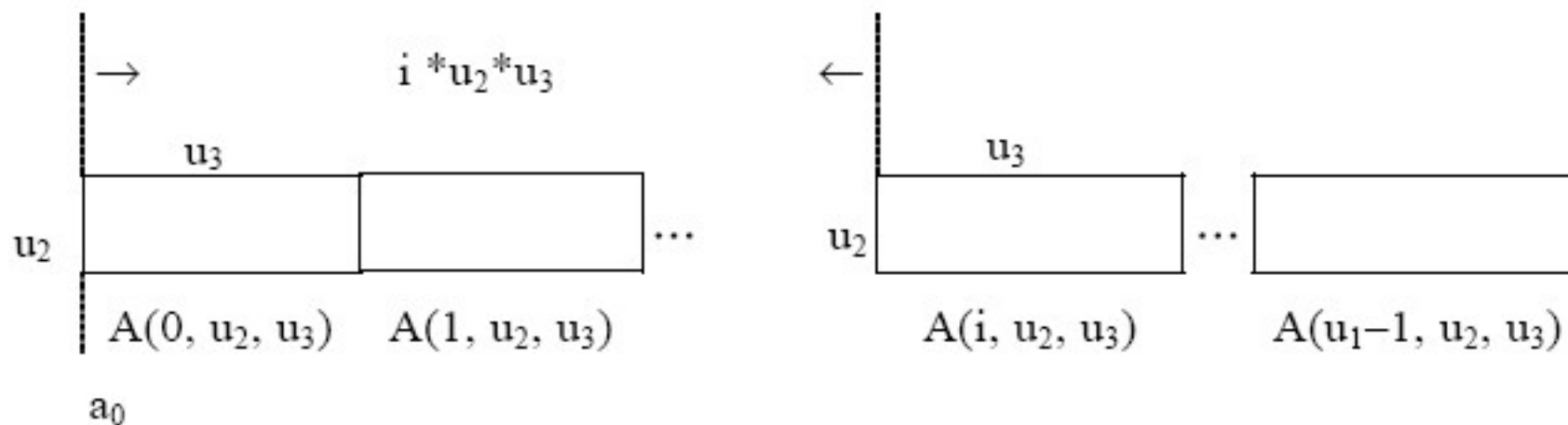


圖 2-3 三維陣列以 u_1 個二維陣列來表示

陣列的表示法

三維陣列 - 以列為主

視此陣列有 u_1 個 $u_2 \times u_3$ 的二維陣列，每一個二維陣列有 u_2 個元素，每個 u_2 皆有 u_3d 個空間。

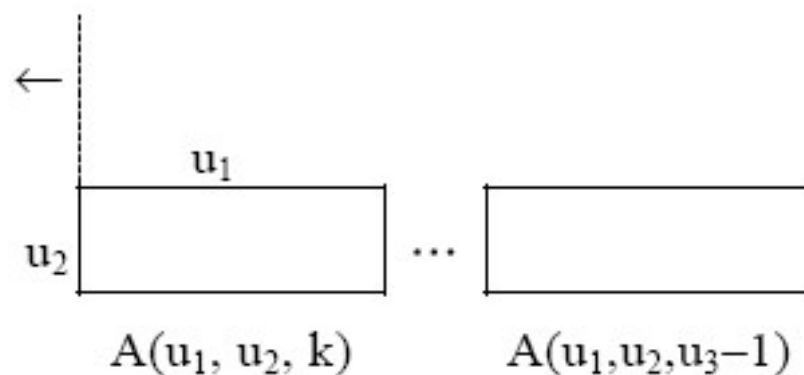
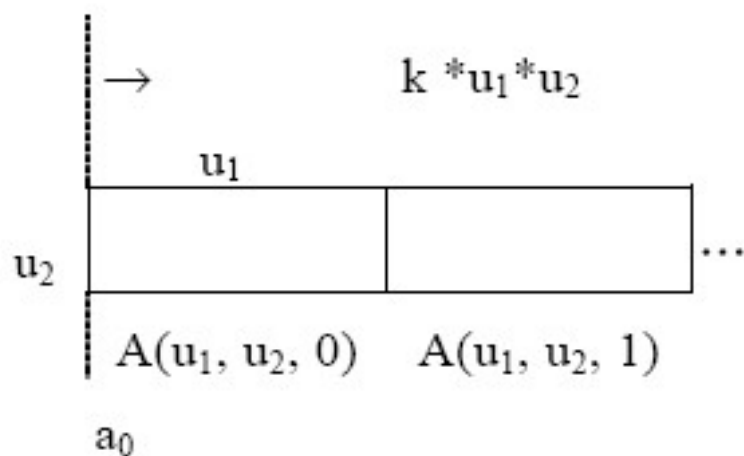


$$A(i, j, k) = a_0 + i * u_2 * u_3 * d + j * u_3 * d + k * d$$

陣列的表示法

三維陣列 - 以行為主

視此陣列有 u_3 個 $u_2 \times u_1$ 的二維陣列，每一個二維陣列有 u_2 個元素，每個 u_2 皆有 $u_1 d$ 個空間。



$$A(i, j, k) = a_0 + k * u_1 * u_2 * d + j * u_1 * d + i * d$$

陣列的表示法- n維陣列

假若有一n 維陣列(n dimension array)為 $A(0 : u_1-1, 0 : u_1-2, 0 : u_3-1, \dots, 0 : u_n-1)$ ，表示A陣列為n 維陣列，同樣n 維陣列亦有二種表示方式：(1)以列為主，(2)以行為主。

- 【以列為主 (Row Major)】

$$\begin{aligned} A(i_1, i_2, \dots, i_n) = & a_0 + (i_1)u_2u_3\cdots u_n d \\ & + (i_2)u_3u_4\cdots u_n d \\ & + (i_3)u_4u_5\cdots u_n d \\ & + \dots \\ & + (i_{n-1})u_n d \\ & + (i_n) d \end{aligned}$$

- 【以行為主 (Column Major)】

$$\begin{aligned} A(i_1, i_2, \dots, i_n) = & a_0 + (i_n)u_{n-1}u_{n-2}\cdots u_1 d \\ & + (i_{n-1})u_{n-2}u_{n-3}\cdots u_1 d \\ & + (i_{n-2})u_{n-3}u_{n-4}\cdots u_1 d \\ & + \dots \\ & + (i_2)u_1 d \\ & + (i_1) d \end{aligned}$$

Java語言的陣列表示方法

Java 語言的一維陣列表示如下：

```
int A []= new int[20];
```

表示A陣列有20個整數元素，從A[0]到A[19]

```
int A [][] = new int[20][10];
```

表示 A 陣列有 20 列、10 行，如下圖所示：

	←	共 10 行			→
↑ 20 列 ↓			...		
			...		
			...		
	:	:			
	.	.			

The Matrix Abstract Data Type

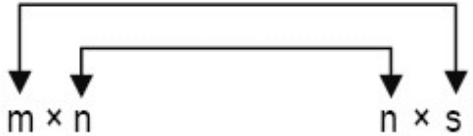
- 一個 $m \times n$ 矩陣可以一個 $m \times n$ 的二維陣列表示 `int A[][] = new int[m][n]`
 - `Matrix(rows, ncols)`: Creates a new matrix containing `nrows` and `ncols` with each element initialized to 0.
 - `numRows()`: Returns the number of rows in the matrix.
 - `numCols()`: Returns the number of columns in the matrix.
 - `getitem (row, col)`: Returns the value stored in the given matrix element. Both row and col must be within the valid range.
 - `setitem (row, col, scalar)`: Sets the matrix element at the given row and col to scalar. The element indices must be within the valid range.
 - `scaleBy(scalar)`: Multiplies each element of the matrix by the given scalar value. The matrix is modified by this operation.
 - `transpose()`: Returns a new matrix that is the transpose of this matrix.
 - `add (secondMatrix)`: Creates and returns a new matrix that is the result of adding this matrix to the given `rhsMatrix`. The size of the two matrices must
 - be the same.
 - `subtract (secondMatrix)`: The same as the `add()` operation but subtracts the two matrices.
 - `multiply (secondMatrix)`: Creates and returns a new matrix that is the result of multiplying this matrix to the given `rhsMatrix`. The two matrices must be of appropriate sizes as defined for matrix multiplication.

multiply (secondMatrix)

假設 $A = (a_{ij})$ 是一 $m \times n$ 的矩陣，而 $B = (b_{ij})$ 為 $n \times s$ 的矩陣，則 AB 的乘積為 $m \times s$ 的矩陣

$$(AB)_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

如下圖所示：



$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ \hline a_{i1} & a_{i2} & \cdots & a_{in} \\ \hline \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}
 \begin{bmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1s} \\ b_{21} & \cdots & b_{2j} & \cdots & b_{2s} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & \cdots & b_{nj} & \cdots & b_{ns} \end{bmatrix}
 \begin{bmatrix} c_{11} & \cdots & c_{1j} & \cdots & c_{1s} \\ \vdots & & \vdots & & \vdots \\ c_{i1} & \cdots & \boxed{c_{ij}} & \cdots & c_{is} \\ \vdots & & \vdots & & \vdots \\ c_{m1} & \cdots & c_{mj} & \cdots & c_{ms} \end{bmatrix}$$

▶▶▶ 【範例】

$$\begin{bmatrix} 2 & 1 & -3 \\ -2 & 2 & 4 \end{bmatrix}
 \begin{bmatrix} -1 & 2 \\ 0 & -3 \\ 2 & 1 \end{bmatrix}
 =
 \begin{bmatrix} 2(-1) + 1(0) + (-3)2 & 2(2) + 1(-3) + (-3)1 \\ (-2)(-1) + 2(0) + 4(2) & (-2)2 + 2(-3) + 4(1) \end{bmatrix}$$

$$=
 \begin{bmatrix} -8 & -2 \\ 10 & -6 \end{bmatrix}$$

multiply (rhsMatrix)

```
public void multiply(Ch02IntMatrix secondMatrix ){
    if(this.numCols()!=secondMatrix.numRows()) throw new NumberFormatException("兩個矩陣大小不一致，不可相乘!");
    Ch02IntMatrix firstMatrix = this;
    int[][] tempMatrix = new int[firstMatrix.numRows()][secondMatrix.numCols()];
    for(int i = 0;i<firstMatrix.numRows() ;i++)
    {
        for(int j = 0;j<secondMatrix.numCols();j++)
        {
            int temp = 0;
            for(int x = 0;x<secondMatrix.numRows();x++)
            {
                temp=temp+firstMatrix.matrix[i][x]*secondMatrix.matrix[x][j];
            }
            tempMatrix[i][j] = temp;
        }
    }
    this.matrix = tempMatrix;
}
```

transpose()

- $B[j][i]=A[i][j]$
- 矩陣A及其轉置矩陣B

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 3 & 8 & 13 & 18 & 23 \\ 4 & 9 & 14 & 19 & 24 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$$

作業02-02

- 請實作transpose()

上三角形和下三角形表示法

- 若一矩陣的對角線以下的元素均為零時，亦即 $a_{ij}=0$ ， $i>j$ ，則稱此矩陣為上三角形矩陣（**upper triangular matrix**）。
- 反之若一矩陣的對角線以上的元素均為零，亦即 $a_{ij}=0$ ， $i<j$ ，此矩陣稱為下三角形矩陣（**lower triangular matrix**），如下圖所示：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

(a)上三角形矩陣

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

(b)下三角形矩陣

- 由上述得知一個 $n \times n$ 的上、下三角形矩陣共有 $[n(n+1)]/2$ 個元素，依序對映至 $D(0:[n(n+1)]/2 - 1)$ 。

上三角形表示法-以列為主

- 以列為主：
一個 $n \times n$ 的上三角形矩陣其元素分別對映至D陣列，如下所示：

$$\begin{array}{cccccccccccc}
 a_{11} & a_{12} & a_{13} & a_{14} & a_{22} & a_{23} & a_{24} & \dots & a_{ij} & \dots & a_{nn} \\
 \hline
 D(1) & D(2) & D(3) & D(4) & D(5) & D(6) & D(7) & \dots & D(k) & \dots & D([n(n+1)]/2)
 \end{array}$$

$\therefore a_{ij} = D(k)$ 其中 $k = n(i-1) + j - [i(i-1)]/2$

- 例如圖2-4之(a)的 a_{34} 元素對映D(k)：
 $k = 4(3-1) + 4 - [3(3-1)]/2 = 8 + 4 - 3 = 9$

$$\begin{bmatrix}
 a_{11} & a_{12} & a_{13} & a_{14} \\
 0 & a_{22} & a_{23} & a_{24} \\
 0 & 0 & a_{33} & a_{34} \\
 0 & 0 & 0 & a_{44}
 \end{bmatrix}$$

Diagram illustrating the mapping of the element a_{34} to the D array. The matrix is shown with rows 1 to 3 highlighted by a red box (labeled $n(i-1)$) and row 3 highlighted by a blue box (labeled j). The element a_{34} is shown in the third row, fourth column. The lower triangular part of the matrix is zero.

下三角形表示法 - 以列為主

- 假使是一個 $n \times n$ 的下三角形矩陣，其元素分別對映至D陣列，如下所示：

$$\begin{array}{ccccccccccc}
 a_{11} & a_{21} & a_{22} & a_{31} & a_{32} & \dots & a_{ij} & \dots & & & a_{nn} \\
 \hline
 D(1) & D(2) & D(3) & D(4) & D(5) & \dots & D(k) & \dots & & & D([n(n+1)]/2)
 \end{array}$$

$\therefore a_{ij} = D(k)$ 其中 $k = [i(i-1)]/2 + j$

- 例如圖2-4之(b)的下三角形矩陣的 a_{32} 位於 $D(k)$ ，而 $k = [3(3-1)]/2 + 2 = 5$

$$\begin{bmatrix}
 a_{11} & 0 & 0 & 0 \\
 a_{21} & a_{22} & 0 & 0 \\
 a_{31} & a_{32} & a_{33} & 0 \\
 a_{41} & a_{42} & a_{43} & a_{44}
 \end{bmatrix}$$

上三角形表示法 - 以行為主

- 以行為主：
上三角形矩陣的對應情形如下：

$$\begin{array}{cccccccccc}
 a_{11} & a_{12} & a_{22} & a_{13} & a_{23} & a_{33} & \dots & a_{ij} & \dots & a_{nn} \\
 \hline
 D(1) & D(2) & D(3) & D(4) & D(5) & D(6) & \dots & D(k) & \dots & D([n(n+1)]/2)
 \end{array}$$

$$\therefore a_{ij} = D(k) \quad \text{其中 } k = [j(j-1)]/2 + i$$

- 例如圖2-4之(a)的 a_{34} 位於 $D(k)$ ，其中
 $k = [4(4-1)]/2 + 3 = 6 + 3 = 9$

下三角形表示法-以行為主

- 而下三角形矩陣對應情形如下：

a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{ij}	a_{nn}
D(1)	D(2)	D(3)	D(4)	D(5)	D(k)	D([n(n+1)]/2)

$\therefore a_{ij}=D(k)$ 其中 $k=n(j-1)-[j(j-1)]/2+i$

- 如圖2-4之(b)的 a_{32} 位於 $D(k)$ ，其中
 $k=4(2-1)-[2(2-1)/2]+3=4-1+3=6$

作業02-03

- 請寫一個function，此function傳入一個以一維陣列儲存的5x5矩陣，function可檢查此矩陣是否為上三角矩陣，
 - Boolean checkUpperTriangular(int[] matrix)
 - 陣列的第一個元素為 a_{11}

稀疏矩陣(1)

若一矩陣中有大多數元素為 0 時，則稱此矩陣為稀疏矩陣(sparse matrix)。到底要多少個 0 才算是疏稀，則沒有絕對的定義，一般而言，大於 1/2 個就可稱之，如下列矩陣為一稀疏矩陣。

$$\begin{bmatrix} 0 & 15 & 0 & 0 & -8 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 \\ 72 & 0 & 0 & 0 & 20 & 0 \end{bmatrix}$$

稀疏矩陣 (2)

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

index	row	col	value
0	0	3	3
1	0	5	-1
2	2	2	2
3	2	5	1
4	3	3	1
5	4	0	2
6	5	1	7
7	6	4	-3
8	7	6	2

- 原矩陣為 8*8，共有 9 個非 0 元素

SparseMatrixItem

```
class SparseMatrixItem{
    public int row;
    public int col;
    public int value;
    SparseMatrixItem(int nRow,int nCol,int nValue) {
        this.row=nRow;
        this.col=nCol;
        this.value=nValue;
    }
}
```

transpose() 實作

```
public Ch02SparseMatrix transpose(){
    Ch02SparseMatrix sparseMatrix = new Ch02SparseMatrix(this.cols, this.rows);
    if(this.num>0){
        for(int c=0; c<this.cols; c++){
            for(int i=0; i<this.num; i++){
                if(this.termArray[i].col==c){
                    sparseMatrix.addATerm(new SparseMatrixItem(c, termArray[i].row,
                        termArray[i].value));
                }
            }
        }
    }
    return sparseMatrix;
}
```

this.num 存非0元素的個數

fastTranspose()

```
public Ch02SparseMatrix fastTranspose() {
    Ch02SparseMatrix sparseMatrix = new Ch02SparseMatrix(this.cols, this.rows);
    if(this.num>0) {
        int[] rowSize =new int[this.cols];
        int[] rowStart = new int[this.cols];
        for(int i=0;i<this.num;i++){
            rowSize[this.termArray[i].col-1]=rowSize[this.termArray[i].col-1]+1;
        }
        rowStart[0]=0;
        for(int i=1;i<this.cols;i++){
            rowStart[i] = rowStart[i-1]+rowSize[i-1];
        }
        for(int i=0;i<this.num;i++){
            int j = rowStart[this.termArray[i].col-1];
            SparseMatrixItem item = new
                SparseMatrixItem(this.termArray[i].col,this.termArray[i].row,this.termArray[i].value);
            sparseMatrix.termArray[j] = item;
            rowStart[this.termArray[i].col-1] = rowStart[this.termArray[i].col-1] + 1;
        }
    }
    return sparseMatrix;
}
```

作業02-04

- 請用您熟悉的程式語言完成`fastTranspose()`演算法

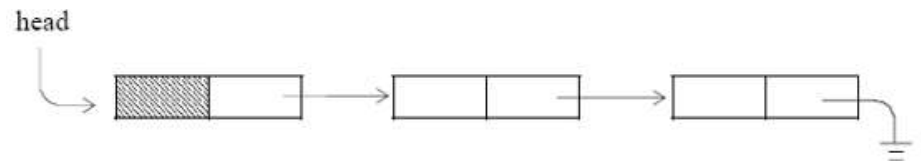
進階作業02-01 (加分)

- 請完成**SparseMatrix**的乘法

線性串列(Ordered or linear list)

- 線性串列
 - 有序串列可以是空集合，或者可寫成 $(a_0, a_1, a_2, \dots, a_{n-2}, a_{n-1})$
 - 存在唯一的第一個元素 a_0 與存在唯一的最後一個元素 a_{n-1}
 - 除了第一個元素外，每一個元素都有唯一的先行者(predecessor)，如 a_i 的先行者是 a_{i-1} 。
 - 除了最後一個元素外，每一個元素都有唯一的後續者(successor)，如 a_i 的先續者是 a_{i+1} 。
- 線性串列經常發生的操作如下：
 - 計算串列的長度。
 - 由左至右或由右至左讀此串列。
 - 取出串列中的第 i 項； $0 \leq i < n$ 。
 - 在第 i 項加入一個新值， $0 \leq i < n$ ，使其原來的第 i 、 $i+1$ 、.....、 n 項變為第 $i+1$ 、 $i+2$ 、.....、 $n+1$ 項。
 - 刪除第 i 項， $0 \leq i < n$ ，使原來的第 $i+1$ 、 $i+2$ 、.....、 n 項變為第 i 、 $i+1$ 、.....、 $n-1$ 項。
- 可使用Array或鏈結串列(linked list)實作線性串列

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]	X[9]
73	65	52	24	83	17	35	96	41	9



多項式表示法 (非物件導向方式)

- 有一多項式 $p=a_nx^n+a_{n-1}x^{n-1}+...+a_1x+a_0$ ，我們稱A為n次多項式， a_ix^j 是多項式的項（ $0 \leq i \leq n, 1 \leq j \leq n$ ）其中 a_i 為係數， x 為變數， j 為指數。
- 多項式使用線性串列來表示有兩種方法：
 - 使用一個 $n+2$ 長度的陣列，依據指數由大至小依序儲存係數，陣列的第一個元素是此多項式最大的指數，如 $p=(n, a_n, a_{n-1}, ..., a_0)$ 。
 - 另一種方法只考慮多項式中非零項的係數，若有 m 項，則使用一個 $2m+1$ 長度的陣列來儲存，分別存每一個非零項的指數與係數，而陣列中的第一個元素是此多項式非零項的個數。
- 例如有一多項式 $p=8x^5+6x^4+3x^2+12$ 分別利用第1種和第2種方式來儲存，其情形如下：
 - $p=(5, 8, 6, 0, 3, 0, 12)$
 - $p=(4, 5, 8, 4, 6, 2, 3, 0, 12)$

多項式表示法 (物件導向方式)

- 多項式使用物件陣列來儲存
 - 將多項式每一項以一個物件Term表示，物件中有兩個屬性分別為指數與係數。
 - 若多項式有m項，使用一個m長度的陣列來儲存，陣列的每一個元素是一個Term物件
- 例如有一多項式 $p=8x^5+6x^4+3x^2+12$ 利用物件導向方式來儲存，其情形如右：

```
class Term{  
    float coef;  
    int exp;  
    Term(float ncoef,int nexp){  
        this.coef=ncoef;  
        this.exp=nexp;  
    }  
}
```

```
Term[] p = new Term[4];  
p[0] = new Term(8,5);  
p[1] = new Term(6,4);  
p[2] = new Term(3,2);  
p[3] = new Term(12,0);
```

多項式ADT

```
public abstract class Polynomial {  
    private Term[] termArray;  
    Polynomial(Term[] t);  
    public abstract void addATerm(Term newTerm);    //新增一個Term到termArray  
    public abstract void removeATerm(Term term); //由termArray移除一個Term  
    public abstract Polynomial add(Polynomial poly);    //將poly與目前Polynomial相加  
    public abstract Polynomial muli(Polynomial poly); //將poly與目前Polynomial相乘  
    public abstract void printPolynomial(); //印出Polynomial的內容  
}
```

兩個多項式相加

```
public Polynomial add(Polynomial secondP){
    Polynomial finalP = new Polynomial(new Term[10]);
    int firstPos=0, secondPos=0;
    int firstNumTerms = this.numTerms();
    int secondNumTerms = secondP.numTerms();
    while((firstPos<firstNumTerms)&&secondPos<secondNumTerms){
        if(termArray[firstPos].exp==secondP.termArray[secondPos].exp){
            float t = termArray[firstPos].coef+secondP.termArray[secondPos].coef;
            finalP.addATerm(new Term(t, termArray[firstPos].exp));
            firstPos=firstPos+1;
            secondPos = secondPos+1;
        }
        else if(termArray[firstPos].exp<secondP.termArray[secondPos].exp)
        {
            finalP.addATerm(new Term(secondP.termArray[secondPos].coef, secondP.termArray[secondPos].exp));
            secondPos = secondPos+1;
        }
        else {
            finalP.addATerm(new Term(this.termArray[firstPos].coef, this.termArray[firstPos].exp));
            firstPos=firstPos+1;
        }
    }
    for(;firstPos<firstNumTerms;firstPos++)
        finalP.addATerm(new Term(this.termArray[firstPos].coef, this.termArray[firstPos].exp));
    for(;secondPos<secondNumTerms;secondPos++)
        finalP.addATerm(new Term(secondP.termArray[secondPos].coef, secondP.termArray[secondPos].exp));
    return finalP;
}
```

作業02-05

- 請用您熟悉的程式語言實作多項式相加的演算法

進階作業02-02 (加分)

- 請實作多項式ADT

```
public abstract class Polynomial {  
    private Term[] termArray;  
    Polynomial(Term[] t);  
    public abstract void addATerm(Term newTerm); //新增一個Term到termArray  
    public abstract void removeATerm(Term term); //由termArray移除一個Term  
    public abstract Polynomial add(Polynomial poly); //將poly與目前Polynomial相加  
    public abstract Polynomial multi(Polynomial poly); //將poly與目前Polynomial相乘  
    public abstract void printPolynomial(); 印出Polynomial的內容  
}
```

進階作業 02-03- String Pattern Matching

- 假設我們有兩個字串s和pat，pat是一個要在s中搜尋的樣式，我們將要用函式find來決定pat是否在s中。假如pat在s中，他會傳回和pat相同子字串的起始位置索引i，假如pat不在s中，或pat式空字串，他會傳回-1。

- 如

String s="abcabacdea"

String pat="bac"

則s.find(bac) = 4

String pat="abe"

則s.find(bac) = -1