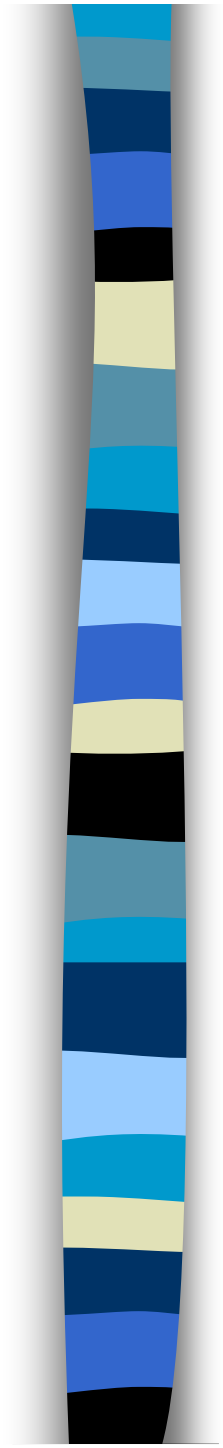


CHAPTER 2

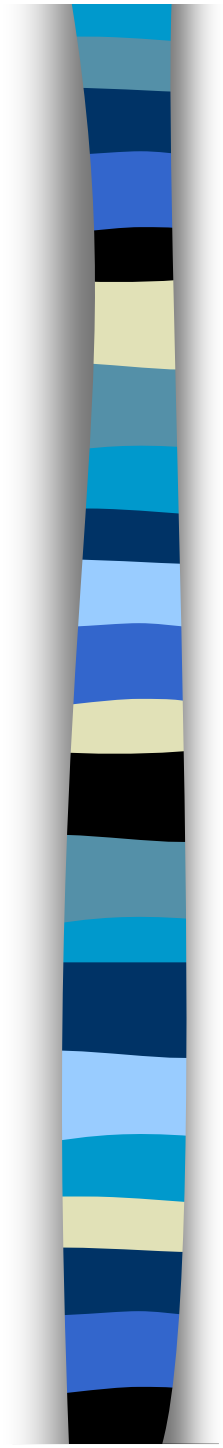
ARRAYS AND STRUCTURES

陣列 (Array)

- 陣列的抽象定義是一組具有相同資料型態 (data type) 的元素，所組成的有序集合 (ordered set)，通常儲存在一塊連續的記憶體上 。
- 陣列的應用技巧可開出善用陣列資料結構之門。
- 陣列在實體記憶空間中配置的關係可引領各位瞭解陣列資料結構如何存放在電腦中，以及程式語言、編譯器如何運作，方使陣列得以有效率地為吾人所用。

陣列的宣告與使用

- 任何資料結構的引用，皆需透過程式語言先行宣告，陣列的宣告必須包含三項要素：
  - (1)陣列的名稱；
  - (2)陣列的大小；
  - (3)陣列中元素的資料型態。



Arrays

Array: a set of index and value

data structure

For each index, there is a value associated with that index.

representation (possible)

implemented by using consecutive memory.

一維陣列-宣告

「一維陣列」（One-dimensional Array）如同前述單排信箱的資料結構，只是將相同資料型態的變數集合起來，然後使用一個變數名稱來代表，以索引值存取指定陣列元素的值。

在C語言陣列變數的宣告語法，如下所示：

資料型態 變數名稱[長度];

上述陣列宣告的資料型態可以是基本資料型態：整數、浮點數和字元，也可以是延伸資料型態的結構。

一維陣列-範例

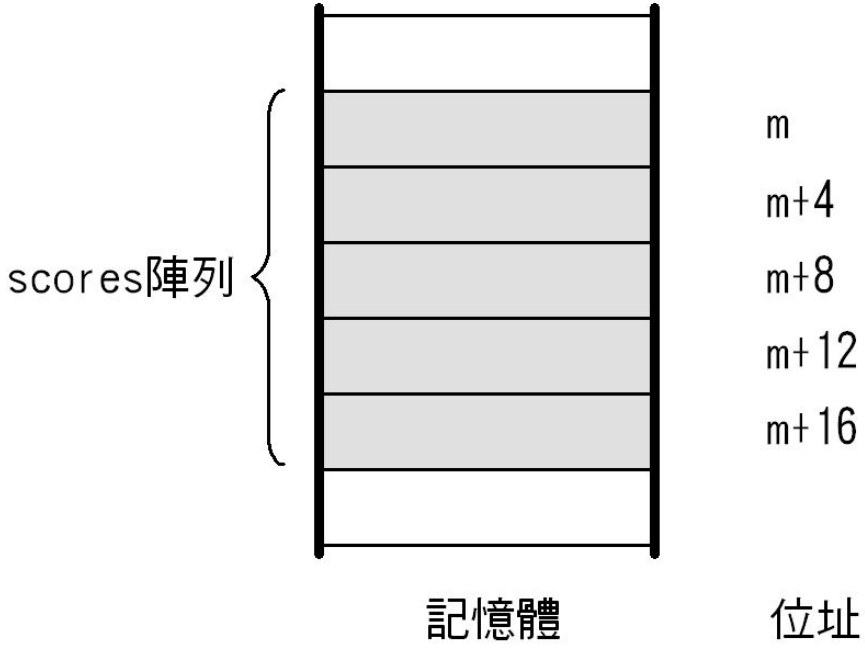
- 例如：宣告整數int的一維陣列scores[]，如下所示：

  int scores[5];

- 上述程式碼宣告大小為5的一維陣列，資料型態是int整數，陣列名稱是scores，C語言的陣列索引值是從0開始。

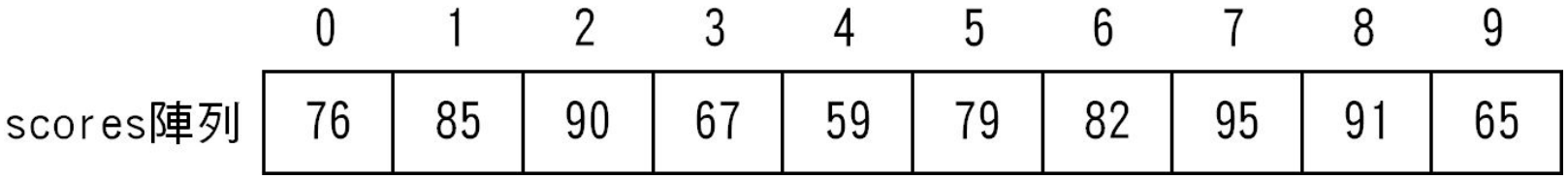
一維陣列-記憶體圖例

- 陣列scores[]的記憶體圖例，m表示陣列第1個元素的記憶體位址scores[0]，這是一塊連續的記憶體空間，如下圖所示：



一維陣列的存取與走訪-存取

- 一維陣列可以隨機存取元素值，只需花費固定時間就可以存取指定索引的元素值。例如：大小10的整數陣列scores[]儲存學生的成績資料，陣列索引值是學生學號，我們可以很容易查詢學生成績或更改學生的成績資料，如下圖所示：



一維陣列的存取與走訪-走訪

- 陣列走訪是以循序方式移動陣列索引值，然後處理每一個元素，也就是以地毯方式依序存取陣列全部的元素，在C語言通常是使用迴圈來走訪陣列，如下所示：

```
for ( i = 0; i < 10; i++ )
    printf("%d:%d ", i, scores[i]);
```

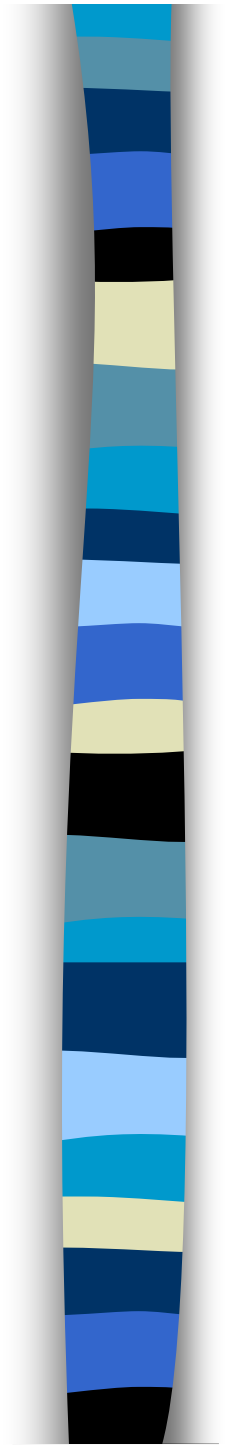
- 上述for迴圈可以走訪整個scores[]陣列的元素。

程式陣列內元素值的指定(assignment)

```
1 main()
2 { int list[5];    //list為一整數陣列，共有5個元素。
3   int i;
4     for (i=0 ; i<5 ; i++) list[i]=i+1;
5 }
```

- 陣列list 的邏輯圖示

list[0]	1
list[1]	2
list[2]	3
list[3]	4
list[4]	5



Arrays in C

int list[5], *plist[5];

list[5]: five integers
list[0], list[1], list[2], list[3], list[4]
*plist[5]: five pointers to integers
plist[0], plist[1], plist[2], plist[3], plist[4]

implementation of 1-D array

list[0]	base address = α
list[1]	α + sizeof(int)
list[2]	α + 2*sizeof(int)
list[3]	α + 3*sizeof(int)
list[4]	α + 4*size(int)

Arrays in C *(Continued)*

Compare int *list1 and int list2[5] in C.

Same: list1 and list2 are pointers.
Difference: list2 reserves five locations.

Notations:

list2 - a pointer to list2[0]
(list2 + i) - a pointer to list2[i] (&list2[i])
*(list2 + i) - list2[i]

Example: 1-dimension array addressing

int one[] = {0, 1, 2, 3, 4};

Goal: print out address and value

```
void print1(int *ptr, int rows)
{
/* print out a one-dimensional array using a pointer */
        int i;
        printf("Address Contents\n");
        for (i=0; i < rows; i++)
                printf("%8u%5d\n", ptr+i, *(ptr+i));
        printf("\n");
}
```

call print1(&one[0], 5)

Address	Contents
1228	0
1230	1
1232	2
1234	3
1236	4

***Figure 2.1:** One- dimensional array addressing (p.53)

Structure *Array* is

objects: A set of pairs <*index*, *value*> where for each value of *index* there is a value from the set *item*. *Index* is a finite ordered set of one or more dimensions, for example, {0, … , n-1} for one dimension, {(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)} for two dimensions, etc.

Functions:

for all A ∈ Array, *i* ∈ *index*, x ∈ *item*, *j*, *size* ∈ integer

Array Create(j, list) ::= **return** an array of *j* dimensions where list is a j-tuple whose *i*th element is the size of the *i*th dimension. *Items* are undefined.

Item Retrieve(A, *i*) ::= **if** (*i* ∈ *index*) **return** the item associated with index value *i* in array A **else return** error

Array Store(A, *i*, *x*) ::= **if** (*i* in *index*) **return** an array that is identical to array A except the new pair <*i*, *x*> has been inserted **else return** error

end array

*__Structure 2.1:__ Abstract Data Type *Array* (p.50)

結構的基礎

- 「結構」（Structures）是C語言的延伸資料型態，這是一種自定資料型態（User-Defined Types），可以讓程式設計者自行在程式碼定義新的資料型態。
- 結構是由一或多個不同資料型態（當然可以是相同資料型態）所組成的集合，然後使用一個新名稱來代表，新名稱是一個新的資料型態，可以用來宣告結構變數。

```
struct Point {
    int x;
    int y;
};
```

宣告結構型態-結構宣告

- 在C程式宣告結構是使用struct關鍵字來定義新型態，其語法如下所示：

```
struct 結構名稱 {
    資料型態 變數1;
    資料型態 變數2;
    ……
};
```

- 上述語法定義名為【結構名稱】的新資料型態，程式設計者可以自行替結構命名，在結構中宣告的變數稱為該結構的「成員」（Members）。

宣告結構型態-結構範例

- 例如：學生資料的結構student，如下所示：

```
struct student {
    int id;
    char name[20];
    int math;
    int english;
    int computer;
};
```

- 上述結構是由學號id、學生姓名name、數學成績math、英文成績english和電腦成績computer的成員變數所組成。

宣告結構型態-結構變數

- 當宣告student結構後，因為它是一種自訂資料型態，所以在程式碼可以使用這個新型態來宣告變數，其語法如下所示：

  struct 結構名稱 變數名稱;

- 上述宣告使用struct關鍵字開頭加上結構名稱來宣告結構變數，以本節程式範例為例，結構變數的宣告，如下所示：

  struct student std1;

  struct student std2 = {2, "江小魚", 45, 78, 66};

宣告結構型態-結構運算

- 在建立好結構變數後，就可以存取結構各成員變數的值，如下所示：

```
std1.id = 1;
strcpy(std1.name, "陳會安");
std1.math = 78;
std1.english = 65;
std1.computer = 90;
```

- ANSI-C語言支援結構變數的指定敘述，如下所示：

```
struct student std3;
std3 = std2;
```

結構陣列

- 「結構陣列」（Arrays of Structures）是結構資料型態的陣列。例如：test結構如下所示：

```
struct test {
    int math;
    int english;
    int computer;
};
```

- 上述結構擁有3個成員變數，因為test是一種新型態，所以可以使用此型態建立陣列，如下所示：

```
#define NUM_STUDENTS      3
struct test students[NUM];
```

巢狀結構

- 「巢狀結構」（Nested Structures）是在宣告的結構中擁有其它結構，例如：在student結構可以使用test結構儲存測驗成績，如下所示：
-

```
struct test {
    int math;
    int english;
    int computer;
};
struct student {
    int id;
    char name[20];
    struct test score;
};
```

建立新型態typedef

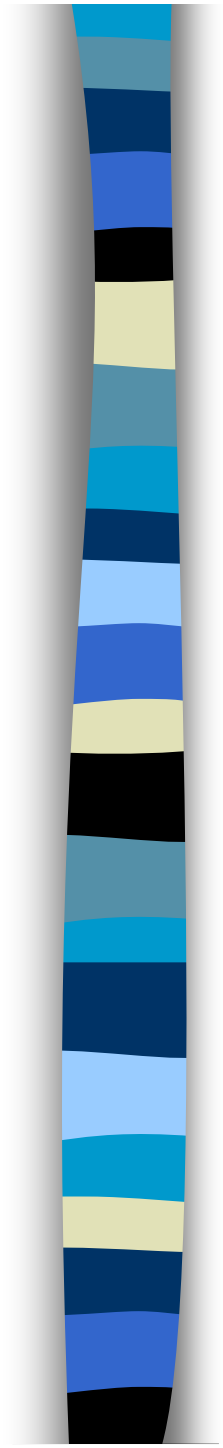
- 在宣告結構型態後，為了方便宣告，我們可以使用一個別名來取代新的資料型態，這個別名是新增的識別字，可以用來定義全新的資料型態，其語法如下所示：

  typedef 資料型態 識別字;

- 上述識別字代表資料型態，所以可以直接使用此識別字宣告變數。例如：程式範例的test結構可以使用typedef指令定義新識別字的型態和宣告變數，如下所示：

  typedef struct test score;

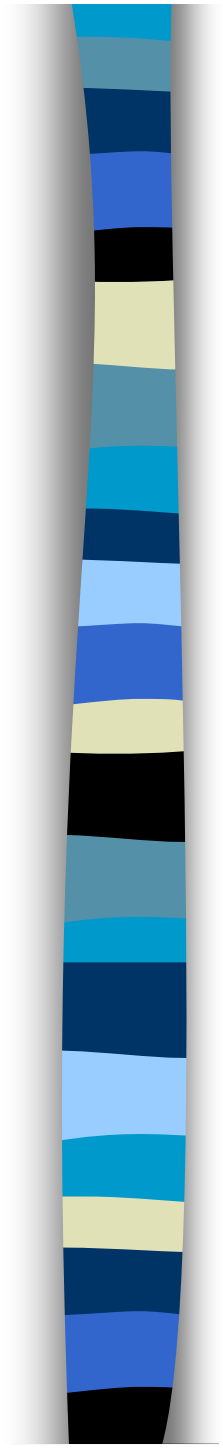
  score joe;



Structures (records)

```
struct {
        char name[10];
        int age;
        float salary;
        } person;

strcpy(person.name, “james”);
person.age=10;
person.salary=35000;
```



Create structure data type

```
typedef struct human_being {
        char name[10];
        int age;
        float salary;
        };
```

or

```
typedef struct {
        char name[10];
        int age;
        float salary
        } human_being;

human_being person1, person2;
```

Unions

Similar to struct, but only one field is active.

Example: Add fields for male and female.

```
typedef struct sex_type {
        enum tag_field {female, male} sex;
        union {
                int children;
                int beard;
                } u;
        };
typedef struct human_being {
        char name[10];
        int age;
        float salary;
        date dob;
        sex_type sex_info;
        }
```

```
human_being person1, person2;
person1.sex_info.sex=male;
person1.sex_info.u.beard=FALSE;
```

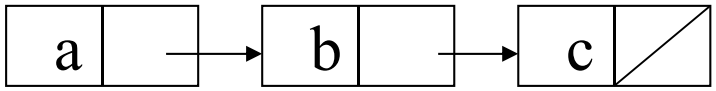
Self-Referential Structures

One or more of its components is a pointer to itself.

```
typedef struct list {
        char data;
        list *link;
        }
```

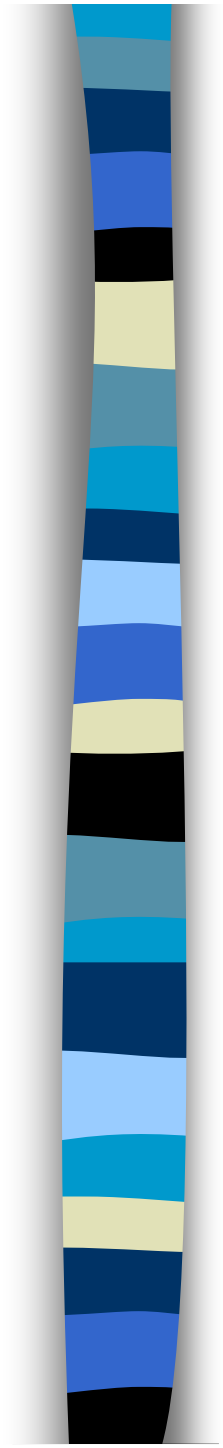
Construct a list with three nodes
item1.link=&item2;
item2.link=&item3;
malloc: obtain a node

```
list item1, item2, item3;
item1.data='a';
item2.data='b';
item3.data='c';
item1.link=item2.link=item3.link=NULL;
```



§2.2.3 Internal Implementation of Structure

- C complier stores the fields of a struct. in the same order using increasing address locations specified in the struct definition.
- C complier在設定一struct中之field的address是依fields在struct中之定義順序，依序assign地址。
- However, that holes or padding may actually occur within a structure to permit two consecutive components to be properly aligned within memory.
- A struct or union type is the amount of storage necessary to represent the largest component, including any padding that may be required.
- ※Turbo C 在 complier 之 code generation之Alignment 設定 Byte/word



作業

- P64: ex1, ex2

Ordered List Examples

ordered (linear) list: (item1, item2, item3, …, item*n*)

- (MONDAY, TUEDSAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAYY, SUNDAY)
- (2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace)
- (1941, 1942, 1943, 1944, 1945)
- (a_1, a_2, a_3, …, a_{n-1}, a_n)

Operations on Ordered List

- Find the length, n , of the list.
- Read the items from left to right (or right to left).
- Retrieve the i'th element.
- Store a new value into the i'th position.
- Insert a new element at the position i , causing elements numbered i, i+1, …, n to become numbered i+1, i+2, …, n+1
- Delete the element at position i , causing elements numbered i+1, …, n to become numbered i, i+1, …, n-1 array (sequential mapping)? (1)~(4) O (5)~(6) X

§The Polynomial Abstract Data Type

- 如何使用Array 來表示一個多項式及其operation.
- The list contain items are written in the form (item0, item1, ……,item n-1)
- The empty list , we denote as ( ).
- A polynomial is：sum of ax^e

$$5x^{10} + 7x^6 + 5x^4 + 3x + 1$$
$$A(x) = 3x^{20} + 2x^5 + 4$$

- The largest (leading) exponent of a polynomial is called its degree.
- The sum and product of polynomials：

$$A(x)+B(x)=\sum(ai+bi)x^{i}$$

$$A(x)\times B(x)=\sum(a_{i}x^{i}\cdot\sum b_{j}x^{i}))$$

Polynomials $A(X)=3X^{20}+2X^{5}+4$, $B(X)=X^{4}+10X^{3}+3X^{2}+1$

Structure *Polynomial* is

objects: $p(x) = a_1 x^{e_1} + \ldots + a_n x^{e_n}$; a set of ordered pairs of $<e_i,a_i>$ where a_i in *Coefficients* and e_i in *Exponents*, e_i are integers >= 0

functions:

for all *poly*, *poly1*, *poly2* $\in$ *Polynomial*, *coef* $\in$*Coefficients*, *expon* $\in$*Exponents*

Polynomial Zero( )	::= **return** the polynomial, $p(x) = 0$
Boolean IsZero(*poly*)	::= **if** (*poly*) **return** *FALSE* **else return** *TRUE*
Coefficient Coef(*poly*, *expon*)	::= **if** (*expon* $\in$ *poly*) **return** its coefficient **else return** Zero
Exponent Lead_Exp(*poly*)	::= **return** the largest exponent in *poly*
Polynomial Attach(*poly*,*coef*, *expon*)	::= **if** (*expon* $\in$ *poly*) **return** error **else return** the polynomial poly with the term <*coef*, *expon*> inserted

Polynomial Remove(*poly*, *expon*) ::= **if** (*expon* $\in$ *poly*) **return** the polynomial *poly* with the term whose exponent is *expon deleted*
else return error

Polynomial SingleMult(*poly*, *coef*, *expon*) ::= **return** the polynomial *poly* • *coef* • x^{expon}

Polynomial Add(*poly1*, *poly2*) ::= **return** the polynomial *poly*1 +*poly*2

Polynomial Mult(*poly1*, *poly2*) ::= **return** the polynomial *poly*1 • *poly*2

End *Polynomial*

***Structure 2.2:** Abstract data type *Polynomial* (p.67)

- For the list, unique exponents arranqed in decreasing order
- 在多項式中，每一個元素之指數皆唯一，且遞減排列。
- The polynomial is < MAX_DEGREE

```
# define MAX_DEGREE   101
 typedef struct{
        int degree;
        float coef[MAX_DEGREE];
  }polynomial;
```

- if a is of type polynomial;

$a.\deg ree = n$

$a.coef[i] = a_{n-i}, 0 \le i \le n$

(即a：coef[i]為x^{n-i}之係數)

Polynomial Addition

data structure 1:

```
#define MAX_DEGREE 101
typedef struct {
        int degree;
        float coef[MAX_DEGREE];
        } polynomial;
```

```
/* d =a + b, where a, b, and d are polynomials */
d = Zero( )
while (! IsZero(a) && ! IsZero(b)) do {
   switch COMPARE (Lead_Exp(a), Lead_Exp(b))  {
      case -1: d =
          Attach(d, Coef (b, Lead_Exp(b)), Lead_Exp(b));
          b = Remove(b, Lead_Exp(b));
          break;
     case  0: sum = Coef (a, Lead_Exp (a)) + Coef ( b, Lead_Exp(b));
        if (sum) {
           Attach (d, sum, Lead_Exp(a));
           a = Remove(a , Lead_Exp(a));
           b = Remove(b , Lead_Exp(b));
           }
         break;
```

```
case 1: d =
            Attach(d, Coef (a, Lead_Exp(a)), Lead_Exp(a));
            a = Remove(a, Lead_Exp(a));
        }
    }
insert any remaining terms of a or b into d
```

advantage: easy implementation
disadvantage: waste space when sparse

***Program 2.5 :**Initial version of *padd* function(p.68)

- This structure wastes a lot of space,
  1. f or example, a.degree << MAX_DEGREE
     遠遠小於
2. The polynomial is sparse ; the number of items with nonzero coefficient is small relative to degree of the polynomial.
   留太多為zero 之 items 於 array 中。

*(p.68)

改進:

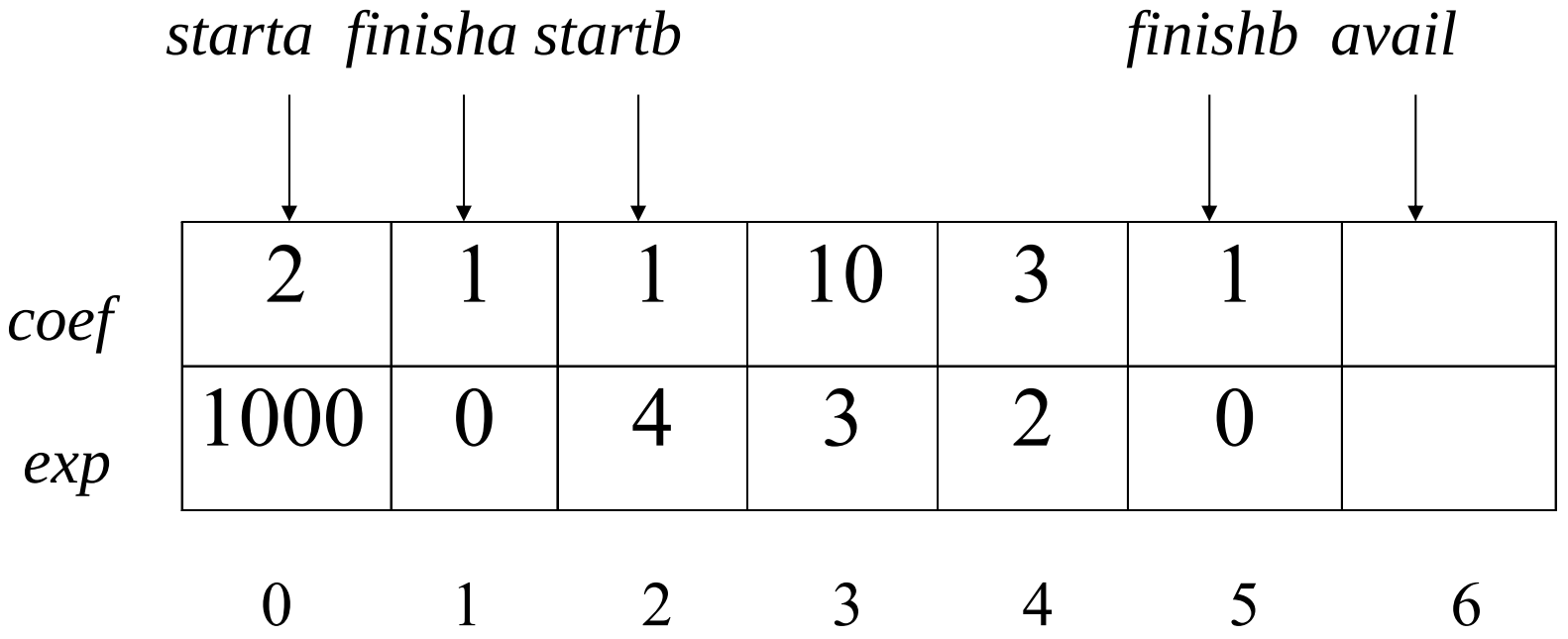
```
MAX_TERMS 100 /* size of terms array */
typedef struct {
        float coef;
        int expon;
        } polynomial;
polynomial terms[MAX_TERMS];
int avail = 0;
```

***Figure 2.2:** Array representation of two polynomials (p.63)

Data structure 2: use one global array to store all polynomials

$A(X)=2X^{1000}+1$

$B(X)=X^4+10X^3+3X^2+1$



specification	representation
poly	<start, finish>
A	<0,1>
B	<2,5>

- Translated poly into a <strart, finish>
- poly是放在list中之start index 列 finish index
- Any poly A with n nonzero items has starta and finisha = starta + n-1

  若poly有n個items, 則finish=start+n-1(指index值)
- $A = 2x^{1000} + 1$ – *need* six units storages.
  - 指數：2個，start and finish
  - 係數：2個
- However, when all items are nonzero, this representation requires about twice as much space as the first one.
  - 事先得知poly之特性，再決定用那一種。
- storage requirements: start, finish, 2*(finish-start+1)

  nonparse: twice as much as (1)

  when all the items are nonzero

Add two polynomials: D = A + B

```
void padd (int starta, int finisha, int startb, int finishb,
                                        int * startd, int *finishd)
{
/* add A(x) and B(x) to obtain D(x) */
    float coefficient;
   *startd = avail;
   while (starta <= finisha && startb <= finishb)
     switch (COMPARE(terms[starta].expon,
                                terms[startb].expon)) {
       case -1: /* a expon < b expon */
              attach(terms[startb].coef, terms[startb].expon);
              startb++
              break;
```

```
case  0: /* equal exponents */
            coefficient = terms[starta].coef +
                                terms[startb].coef;
            if (coefficient)
               attach (coefficient, terms[starta].expon);
            starta++;
            startb++;
            break;
case 1: /* a expon > b expon */
         attach(terms[starta].coef, terms[starta].expon);
         starta++;
}
```

```
/* add in remaining terms of  A(x) */
for( ; starta <= finisha; starta++)
     attach(terms[starta].coef, terms[starta].expon);
/* add in remaining terms of B(x) */
for( ; startb <= finishb; startb++)
    attach(terms[startb].coef, terms[startb].expon);
*finishd =avail -1;
}
```

Analysis: O(n+m)

where n (m) is the number of nonzeros in A(B).

***Program 2.6:** Function to add two polynomial (p.70)

```
void attach(float coefficient, int exponent)
{
/* add a new term to the polynomial */
    if (avail >= MAX_TERMS) {
       fprintf(stderr, "Too many terms in the polynomial\n");
       exit(1);
      }
      terms[avail].coef  = coefficient;
      terms[avail++].expon = exponent;
}
```

***Program 2.7:** Function to add anew term (p.71)

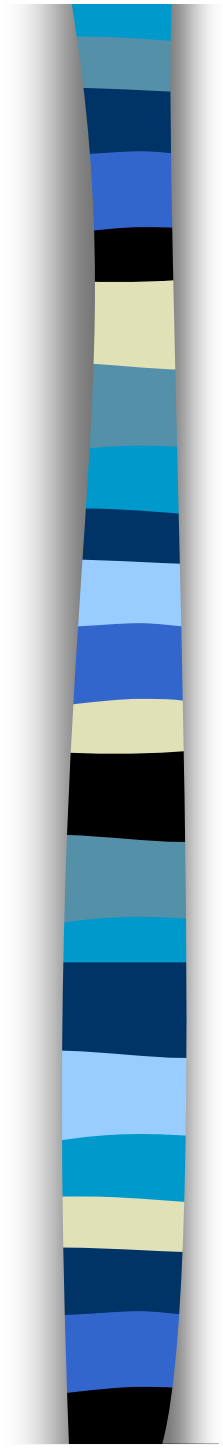
- Analysis of padd.

  Let m and n be the number of nonzero items in A and B.
- For each iteration (while loop) requires O(1) time.
- The number of iterations (while) is bounded by m+n-1, the worst case is O(m+n).

$$when \quad h(X) = \sum_{i=0}^{m} x^{2i} — 偶數項$$

$$B(X) = \sum_{i=0}^{n} x^{2i+1} — 奇數項$$

- The remaining two for loops is bounded O(m) and O(n)
  - The asymptotic computing time of this algorithm is O(m+n)
  - (O(m+n)+O(m)+O(n) O(m+n))



作業

- P72: ex2

陣列的應用 - 稀疏矩陣表示法(矩陣)

- 「矩陣」（Matrices）類似二維陣列，一個m X n矩陣表示這個矩陣擁有m列（Rows）和n欄（Columns），或稱為列和行，4 X 3矩陣的m和n是矩陣的「維度」（Dimensions）。

	第1欄	第2欄	第3欄
第1列	6	2	0
第2列	1	0	3
第3列	6	4	2
第4列	1	4	7

陣列的應用 - 稀疏矩陣表示法(稀疏矩陣說明)

- 「稀疏矩陣」（Sparse Matrices）屬於矩陣一種非常特殊的情況，因為矩陣元素大部份元素都沒有使用，元素稀稀落落，所以稱為稀疏矩陣。例如：50個元素的稀疏矩陣，真正使用的元素只有5個，如下圖所示：

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

§2.5 The Sparse Matrix ADT

- 如何用array來有效地表示-sparse matrix.
- Fig2.3 illustrated a matrix containing m rows and n columns of elements.
- In computer science , a matrix is a two-dimensional array defined as a [MAX_RROWS][MAX_COLS].

  a[i][j],where i is the row index and

  j is the column index.
- Fig2.4(b) call this a sparse matrix containing many zero entries.
- Sparse matrix wastes spaces, therefore we should store only nonzero elements.
- Structure 2.3 (p74) is the matrix ADT.

Sparse Matrix

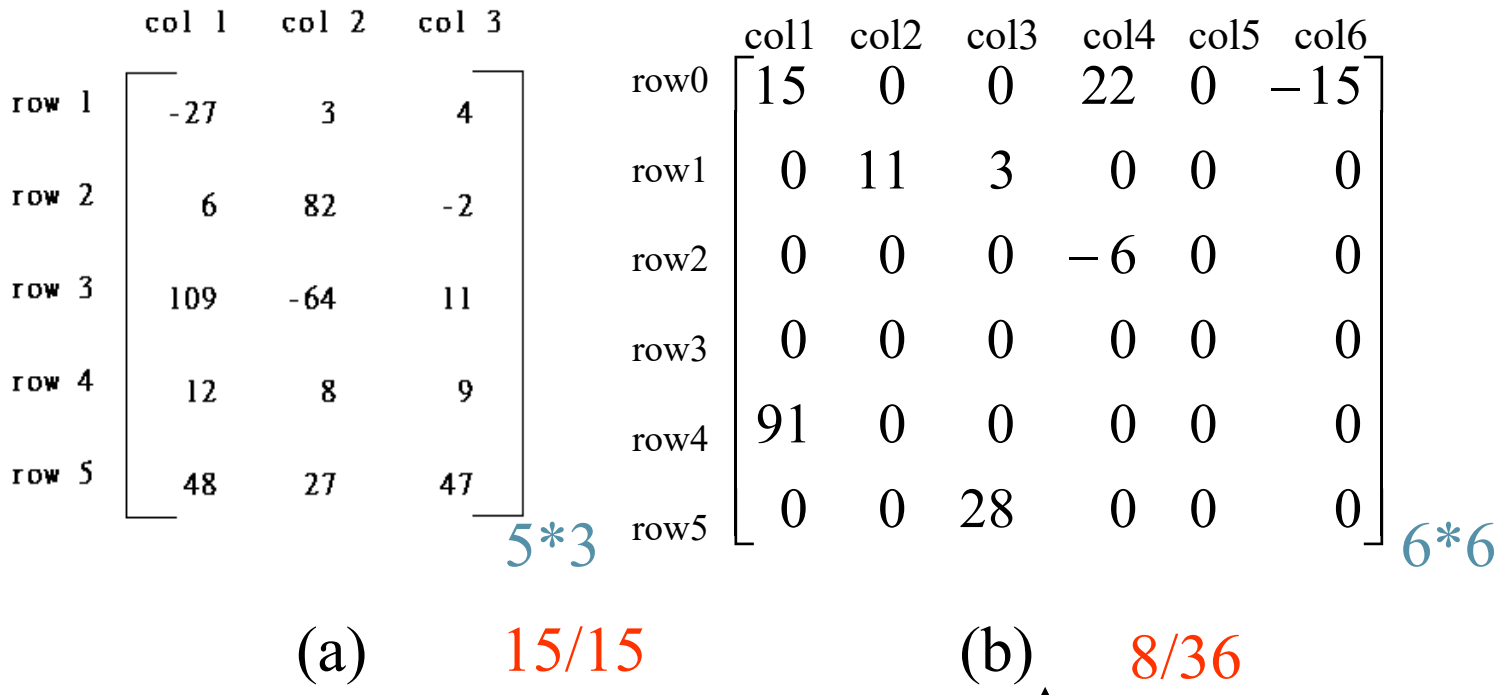


***Figure 2.4:Two matrices**

SPARSE MATRIX ABSTRACT DATA TYPE

Structure *Sparse_Matrix* is

objects: a set of triples, <*row*, *column*, *value*>, where *row* and *column* are integers and form a unique combination, and *value* comes from the set *item*.

functions:

for all *a*, *b* ∈ *Sparse_Matrix*, *x* ∈ *item*, *i*, *j*, *max_col*, *max_row* ∈ *index*

Sparse_Marix Create(*max_row*, *max_col*) ::=

return a *Sparse_matrix* that can hold up to *max_items* = *max _row* × *max_col* and whose maximum row size is *max_row* and whose maximum column size is *max_col*.

Sparse_Matrix Transpose(*a*) ::=

return the matrix produced by interchanging the row and column value of every triple.

Sparse_Matrix Add(*a*, *b*) ::=

if the dimensions of a and b are the same
return the matrix produced by adding corresponding items, namely those with identical *row* and *column* values.
else return error

Sparse_Matrix Multiply(*a*, *b*) ::=

if number of columns in a equals number of rows in **b**
return the matrix *d* produced by multiplying a by *b* according to the formula: *d* [*i*] [*j*] = Σ(a[i][k]•b[k][j]) where *d* (*i*, *j*) is the (*i*,*j*)th element
else return error.

* **Structure 2.3:** Abstract data type Sparse-Matrix (p.68)

陣列的應用 - 稀疏矩陣表示法 (稀疏矩陣表示法)

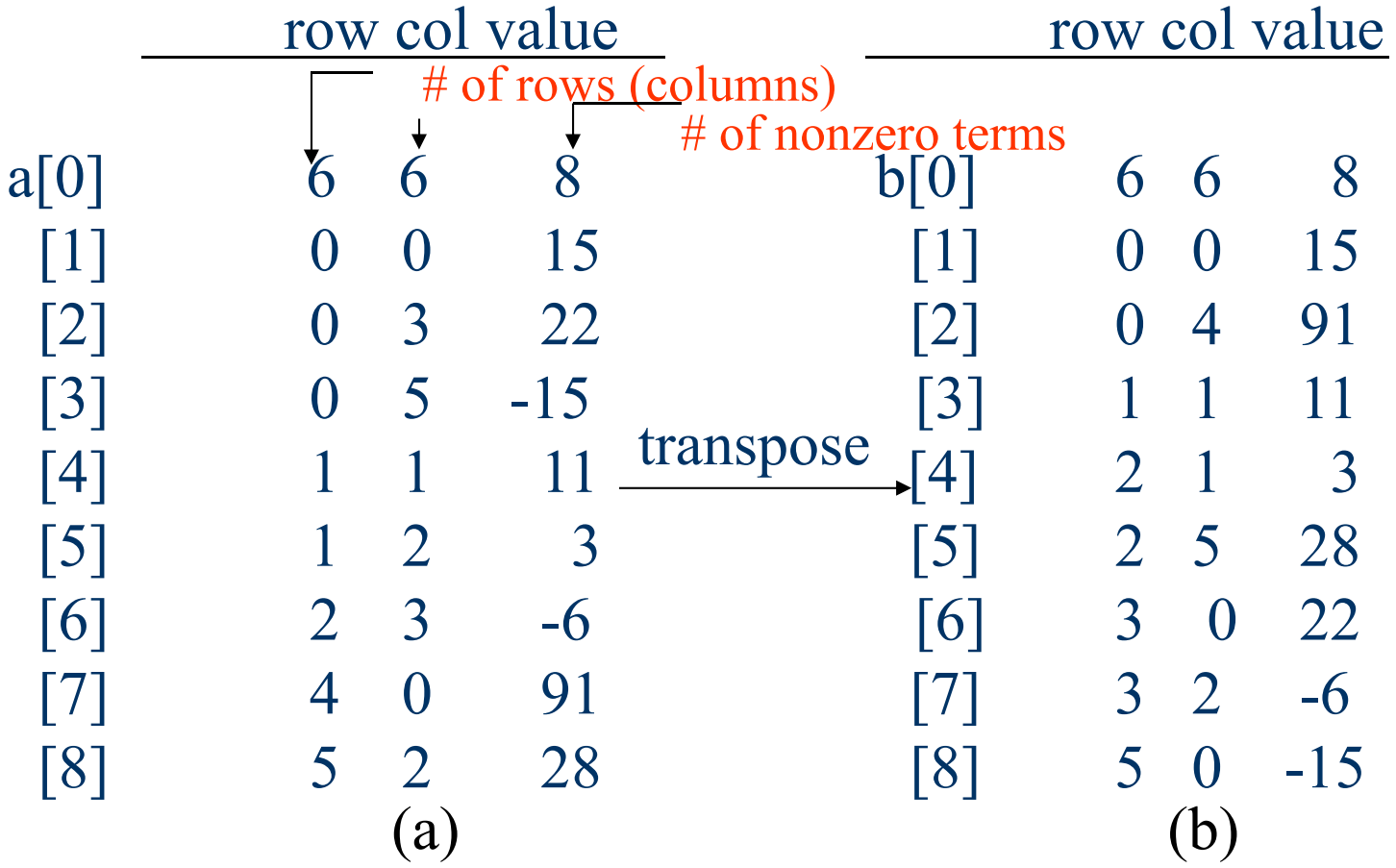
- 稀疏矩陣實際儲存資料的項目很少，如果使用二維陣列來儲存稀疏矩陣，表示大部分記憶體空間都是閒置的，為了增加記憶體的使用效率，可以採用壓縮方式儲存稀疏矩陣中只擁有值的項目，如下圖所示：

	列row	欄col	值value
smArr[0]	0	2	1
smArr[1]	1	3	9
smArr[2]	2	5	2
smArr[3]	3	4	3
smArr[4]	4	7	6

- We use an array of triples to represent a spare matrix.
- 如何用－triple來表示－matrix 2 element
  - triple：＜row, col, value＞
  - row and col are in ascending
  - We also know the number of rows, columns and nonzero elements.
- Sparse_Matrix Create(max_row, max_col)

```
#define  MAX_TERMS 101
typedef  Struct{
        int col;
       int row;
       int value;
} term;
term a[MAX_TERMS];
```

Fig2.5(a) a[0] row is the number of rows.
a[0].col is the number of cols.
a[0].value is total number of nonzero ele.

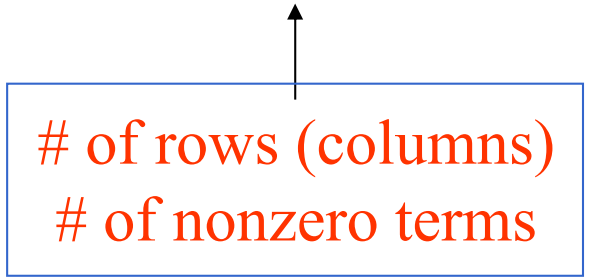


row, column in ascending order

***Figure 2.5:** Sparse matrix and its transpose stored as triples (p. 75)

Sparse_matrix Create(max_row, max_col) ::=

```
#define MAX_TERMS 101 /* maximum number of terms +1*/
   typedef struct {
              int col;
              int row;
              int value;
              } term;
   term a[MAX_TERMS]
```



* (P.69)

§Transposing a Matrix.

- Transpose a matrix is to interchange the rows and columns.

a[i][j] transpose to b[j][i]

- Algorithm：

  For each row i

    take element <i, j, value> and store

    it as element <j, i, value>

→ 要transpose <j, i, value> 之前，必須要先知道<j, i>之前之所有elements

(row and col 皆為ascending)

(0, 0,15) (0, 0, 15)

(0, 3, 22) (3, 0, 22)

(0, 5, -15) (5, 0, -15)

Transpose a Matrix

(1) for each row i

take element <i, j, value> and store it

in element <j, i, value> of the transpose.

difficulty: where to put <j, i, value>

(0, 0, 15) ====> (0, 0, 15)

(0, 3, 22) ====> (3, 0, 22)

(0, 5, -15) ====> (5, 0, -15)

(1, 1, 11) ====> (1, 1, 11)

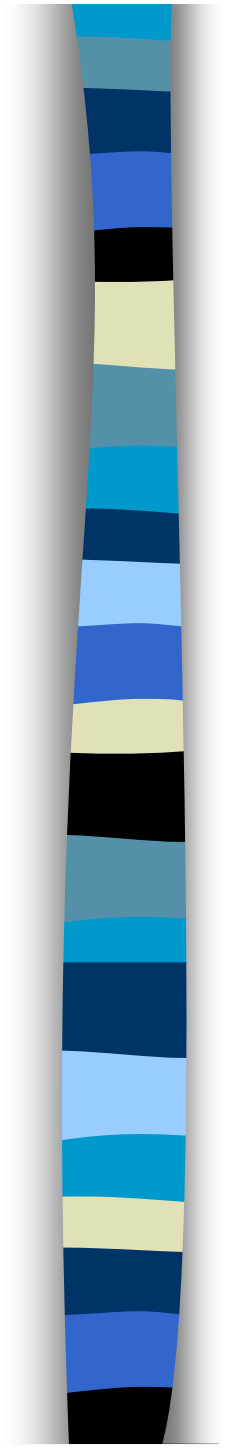
Move elements down very often.

(2) Suggesting Algorithm：

for all elements in column j

place element <i, j, value> in element < j, i, value>

The algorithm is implement in Program2.8



```
void transpose (term a[], term b[])
/* b is set to the transpose of a */
{
   int n, i, j, currentb;
   n = a[0].value;  /* total number of elements */
   b[0].row = a[0].col;  /* rows in b = columns in a */
   b[0].col = a[0].row;  /*columns in b = rows in a */
   b[0].value = n;
   if (n > 0) {                /*non zero matrix */
      currentb = 1;     /*  目前欲transpose 之 arry b 之index */
for (i = 0; i < a[0].col; i++)
      /* /*針對所有column, 且自column 0開始*/*/
         for( j = 1; j <=  n; j++)
         /*  find elements from the current column */
         if (a[j].col == i) {

      /*原在array a 中之column，符合目前所欲transpose之column */
         /* element is in current column, add it to b */
```

columns

elements

```
            b[currentb].row = a[j].col;
            b[currentb].col  = a[j].row;
            b[currentb].value = a[j].value;
            currentb++
        }
    }
}
```

*** Program 2.8:** Transpose of a sparse matrix (p.77)

Scan the array "columns" times.
The array has "elements" elements.

==> O(columns*elements)

- Suggesting Algorithm：

for all elements in column j

place element <i, j, value> in element < j, i, value>

- The algorithm is implement in Program2.8
- Analysis of transpose：

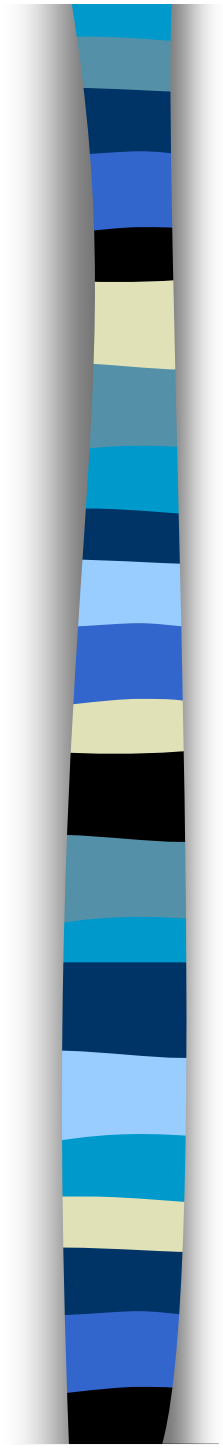
The outer for loop is a[0].col times

The inner for loop is a[0].value times

the asymptotic time is O(columns element)

– Element = col * row

– O($columns^2 \cdot rows$)



Discussion: compared with 2-D array representation

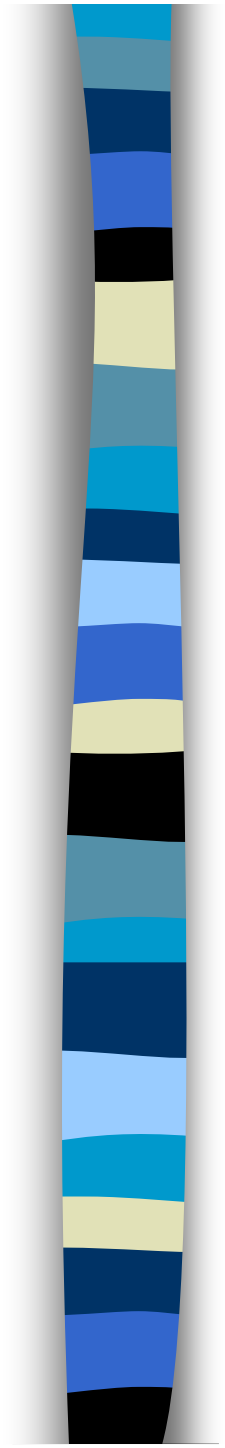
O(columns*elements) vs. O(columns*rows)

elements --> columns * rows when nonsparse
O(columns*columns*rows)

Problem: Scan the array “columns” times.

Solution:

Determine the number of elements in each column of the original matrix.
==>
Determine the starting positions of each row in the transpose matrix.

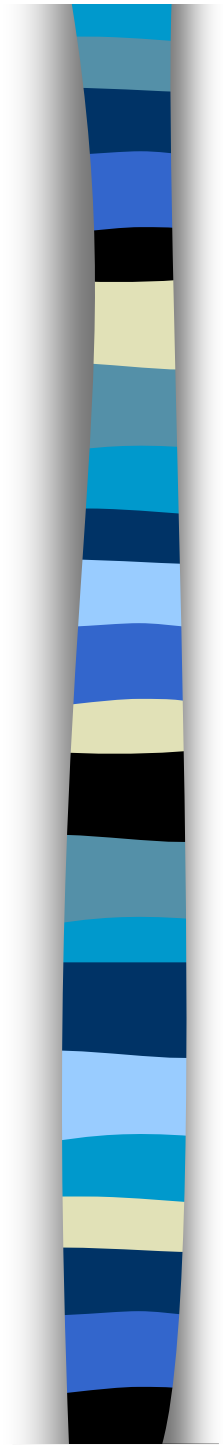


- fast-transpose (program 2.8 P72)

proceeds by first determining the number of elements in each column of the original matrix, that is , gives us the number of elements in each row of the transpose matrix.

```
void fast_transpose (term a[], term b[])
{
    /* the transpose of a is placed in b*/
  int row_terms [MAX_COL], strating_pos[MAX_COL]
  int i,j num_cols=a[0].col, num_terms=a[0].value;
  b[0].row=a[0].col; b[0].col=a[0].row;
  b[0].value=num_terms;
  if (num_terms>0) {     /*nonzero matrix */
    for (i=0;i<num_cols;i++)
            row_terms [i]= 0;
    for (i=1;i<=num_terms;i++)
            row_terms[a[i].col]++;      對某col而言，有多少項
    starting_pos[0]=1;
    for (i=1;i<num_cols;i++)
            starting_pos[i]=
    starting_pos[i-1]+row_terms[i-1];
    for (i=1;i<=num_terms;i++) {
            j=starting_pos[a[i].col]++; 後置運算
            b[j].row=a[i].col;  b[j].col=a[i].row;
            b[j].value=a[i].value;
} } }
```

Program 2.9：Fast transpose of a sparse matrix

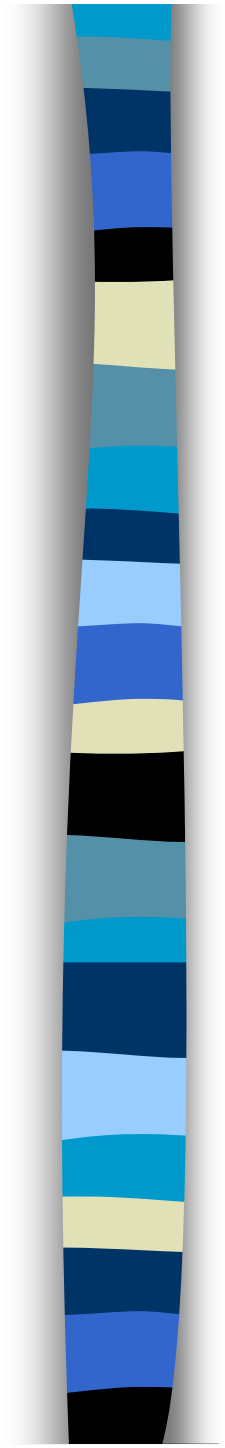


	[0]	[1]	[2]	[3]	[4]	[5]
After the third for loop row-terms=	2	1	2	2	0	1
starting-pos=	1	3	4	6	8	8

After j=start pos…++

2	4	7	9	
(i=1)	(i=4)	(i=7)	(i=3)	

- 前一個程式(program 2.8) , 是先鎖定某個col，再對全之term掃一遍，看是否屬於此col，再transpose, 因此共了 O(columns.element)
- strating-pos[i]：記住array b 之index.
  =>即已 transposed 之matrix其第i個row, 在array 處中，自index多少開始
- j = starting-pos[a[i].col]
  在a中之col, 即為b中之row，因此，每當一個u之col, 已transpose 給 b 後，原starting-pos 必須位置加一(tor b之row)
- Analysis of fast_transpose：
  The starting point of row i is row_term[i-1] + starting-pos[i-1]
  The first for loop is num_cols times
  The second for loop is num_terms times.
  The third for loop is num_cols-1 times.
  The forth for loop is num_terms times.
  =>time complexity is O(columns+elements)
  =>become O(columns.rows)
  when elements is of order cols.rows



a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

	[0]	[1]	[2]	[3]	[4]	[5]
row_terms =	2	1	2	2	0	1
starting_pos =	1	3	4	6	8	8

```
void fast_transpose(term a[ ], term b[ ])
{
/* the transpose of a is placed in b */
  int row_terms[MAX_COL], starting_pos[MAX_COL];
  int i, j, num_cols = a[0].col, num_terms = a[0].value;
  b[0].row = num_cols; b[0].col = a[0].row;
  b[0].value = num_terms;
  if (num_terms > 0){ /*nonzero matrix*/
    for (i = 0; i < num_cols; i++)
        row_terms[i] = 0;
    for (i = 1; i  <= num_terms; i++)
        row_term [a[i].col]++
    starting_pos[0] = 1;
    for (i =1; i < num_cols; i++)
        starting_pos[i]=starting_pos[i-1] +row_terms [i-1];
```

columns

elements

columns

```
                for (i=1; i <= num_terms, i++) {
elements                j = starting_pos[a[i].col]++;
                        b[j].row = a[i].col;
                        b[j].col = a[i].row;
                        b[j].value = a[i].value;
                    }
                }
            }
```

***Program 2.8:** Fast transpose of a sparse matrix

Compared with 2-D array representation

O(columns+elements) vs. O(columns*rows)

elements --> columns * rows

O(columns+elements) --> O(columns*rows)

Cost: Additional row_terms and starting_pos arrays are required.

Let the two arrays row_terms and starting_pos be shared.

作業

- P84: ex1
- P99: ex3