CHAPTER 2

ARRAYS AND STRUCTURES

All the programs in this file are selected from
Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed
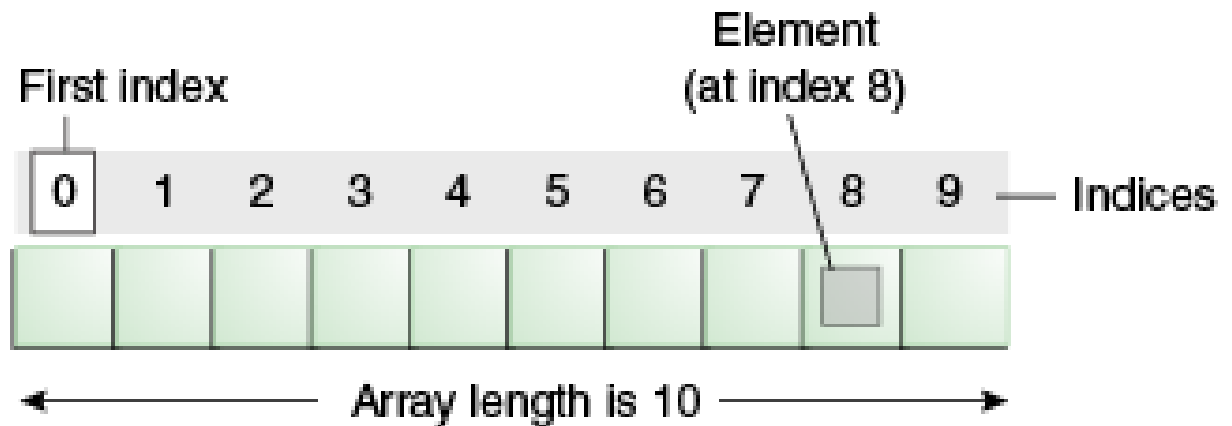"Fundamentals of Data Structures in C",

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.

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 $\in$ Array, $i \in$ *index*, x $\in$ *item, j, size* $\in$ integer

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

  Item Retrieve(A, i) ::= **if** ($i \in$ *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*

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*sizeof(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]   (value)

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
int main(void) {
    int one[] = {0, 1, 2, 3, 4};              宣告一個陣列並給予值
    print1(&one[0], 5);
}
void print1(int *ptr, int rows){
    int i;
    printf("Address    Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
int main(void) {
    int one[] = {0, 1, 2, 3, 4};
    print1(&one[0], 5);
}
void print1(int *ptr, int rows){
    int i;
    printf("Address     Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

陣列one的長度

陣列one的索引值

取address

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
int main(void) {
    int one[] = {0, 1, 2, 3, 4};
    print1(&one[0], 5);
}
void print1(int *ptr, int rows){
    int i;
    printf("Address     Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}               八位數的無符號十進位整數，不足以0填滿
```

1-dimension array addressing

目的: 顯示一個陣列的address和value

```c
int main(void) {
    int one[] = {0, 1, 2, 3, 4};
    print1(&one[0], 5);
}
void print1(int *ptr, int rows){
    int i;
    printf("Address     Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

五位數的有符號十進位整數，不足以空白填滿

1-dimension array addressing

目的: 顯示一個陣列的address和value

int main(void) {

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

print1(&one[0], 5);

}

索引值	0	1	2	3	4
address	06422284	06422288	06422292	06422296	06422300
one	0	1	2	3	4

* 每個int是4 bytes，所以address之間的間隔差4

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
void print1(int *ptr, int rows){
    int i;
    printf("Address      Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

索引值 0   1   2   3   4

address 06422284 06422288 06422292 06422296 06422300

ptr+0

i = 0

| one | 0 | 1 | 2 | 3 | 4 |

*(ptr+0)

Output:
Address   Contents
06422284   0

1-dimension array addressing

目的: 顯示一個陣列的address和value

```c
void print1(int *ptr, int rows){
    int i;
    printf("Address      Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

索引值	0	1	2	3	4
address	06422284	06422288	06422292	06422296	06422300

ptr+1

i = 1

one	0	1	2	3	4

*(ptr+1)

Output:
Address   Contents
06422284   0
06422288   1

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
void print1(int *ptr, int rows){
    int i;
    printf("Address     Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

索引值	0	1	2	3	4
address	06422284	06422288	06422292	06422296	06422300

ptr+2

i = 2

one	0	1	2	3	4

*(ptr+2)

Output:

Address	Contents
06422284	0
06422288	1
06422292	2

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
void print1(int *ptr, int rows){
    int i;
    printf("Address     Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

索引值 | 0 | 1 | 2 | 3 | 4
address | 06422284 | 06422288 | 06422292 | 06422296 | 06422300

ptr+3

i = 3

one | 0 | 1 | 2 | 3 | 4

*(ptr+3)

Output:

Address	Contents
06422284	0
06422288	1
06422292	2
06422296	3

1-dimension array addressing

目的: 顯示一個陣列的address和value

```
void print1(int *ptr, int rows){
    int i;
    printf("Address    Contents\n");
    for(i = 0;i < rows;i++){
        printf("%08u %5d\n", ptr+i, *(ptr+i));
    }
}
```

索引值   0   1   2   3   4

address   06422284   06422288   06422292   06422296   06422300

ptr+4

i = 4

one	0	1	2	3	4

*(ptr+4)

Output:

Address	Contents
06422284	0
06422288	1
06422292	2
06422296	3
06422300	4

Multiple Dimension Array

- Two dimension
  - int arr[2][3];

- Three dimension
  - int arr[2][3][4];

- N dimension
  - int arr[2][3][4][…];

Multidimensional Arrays

C also allows an array to have more than one dimension.

For example, a two-dimensional array consists of a certain <u>number of rows</u> <u>and columns</u>:

```
const int NUMROWS = 3;
const int NUMCOLS = 7;
int Array[NUMROWS][NUMCOLS];
```

	0	1	2	3	4	5	6
0	4	18	9	3	-4	6	0
1	12	45	74	15	0	98	0
2	84	87	75	67	81	85	79

```
Array[2][5]
```
3rd value in 6th column
```
Array[0][4]
```
1st value in 5th column

The declaration must specify the number of rows and the number of columns, and both must be constants.

Processing a 2-D Array

A one-dimensional array is usually processed via a for loop.

Similarly, a two-dimensional array may be processed with a nested for loop:

```
for (int Row = 0; Row < NUMROWS; Row++) {

    for (int Col = 0; Col < NUMCOLS; Col++) {

        Array[Row][Col] = 0;

    }

}
```

Each pass through the inner for loop will initialize all the elements of the current row to 0.

The outer for loop drives the inner loop to process each of the array's rows.

Higher-Dimensional Arrays

An array can be declared with multiple dimensions.

2 Dimensional
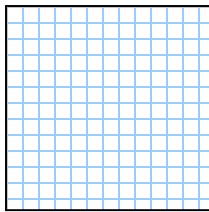
3 Dimensional
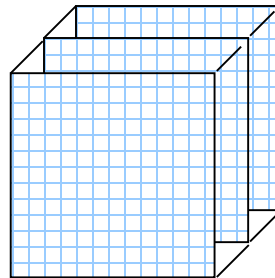
`double Coord[100][100][100];`

Multiple dimensions get difficult to visualize graphically.

•
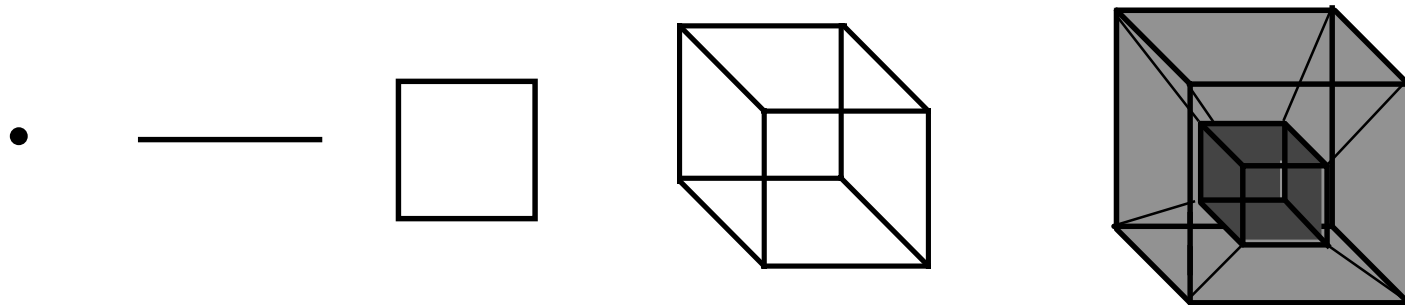
Structures (records)

```
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

Example: Add fields for male and female.

```
typedef struct sex_type {
        enum tag_field {female, male} sex;
        union {
                int children;
                int beard;
                } u;        Similar to struct, but only one field is
        };                  active.


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=0 (False);

Implementation structure and unions

目的: 實作structure 和 unions

```
typedef struct {
    enum {female, male} sex;
    union {
        int children;
        int beard;
    } u;
} sex_type;

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

列舉(Enumeration)是以關鍵字enum開始加上一個列舉名稱，並以大括號括住要群組管理的常數。

Implementation structure and unions

說明階層關係

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

sex

Implementation structure and unions

說明階層關係

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

sex
u   -> children
      -> beard

Implementation structure and unions

說明階層關係

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

sex_info -> sex
            -> u   -> children
                    -> beard

Implementation structure and unions

說明階層關係

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

name

Implementation structure and unions

說明階層關係

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

name
age

Implementation structure and unions

說明階層關係

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

name
age
salary

Implementation structure and unions

說明階層關係

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

name
age
salary
型別為結構sex_type的sex_info -> sex
                                    -> u   -> children
                                          -> beard

Implementation structure and unions

說明階層關係

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

human_being -> name
　　　　　　 -> age
　　　　　　 -> salary
　　　　　　 ->型別為結構sex_type的sex_info -> sex
　　　　　　　　　　　　　　　　　　　　　　　　 -> u　 -> children
　　　　　　　　　　　　　　　　　　　　　　　　　　　　　 -> beard

Implementation structure and unions

階層關係轉換成程式碼

human_being person1;

human_being -> name

       -> age
       -> salary
       ->型別為結構sex_type的sex_info -> sex

                                 -> u   -> children
                                       -> beard

person1 -> person1.name
      -> person1.age
      -> person1.salary
      -> person1.sex_info -> person1.sex_info.sex
                       -> person1.sex_info.u   -> person1.sex_info.u.children
                                        -> person1.sex_info.u.beard

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;

| a | | → | b | | → | c | / |
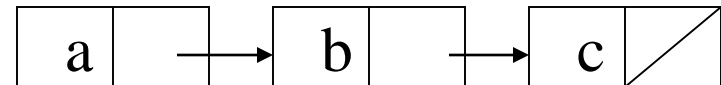
Ordered List Examples

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

1. Find the length, n, of the list.

2. Read the items from left to right (or right to left).

3. Retrieve the i'th element.

4. Store a new value into the i'th position.

5. Insert a new element at the position i, causing elements numbered i, $i+1$, …, n to become numbered $i+1$, $i+2$, …, $n+1$

6. 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

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 <u>***a_i in Coefficients*** **and** ***e_i in Exponents***</u>, 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** <u>the largest exponent in</u>

                             poly

Polynomial Attach(*poly,coef, expon*) ::= **if** (*expon* $\in$ *poly*) **return** error

                                 else return <u>the polynomial poly</u>

                               <u>with the term *<coef, expon>*</u>

                               <u>inserted</u>

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*

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 =              /* a < b */
        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

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

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

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

	starta	finisha	startb			finishb	avail
coef	2	1	1	10	3	1	
exp	1000	0	4	3	2	0	
	0	1	2	3	4	5	6
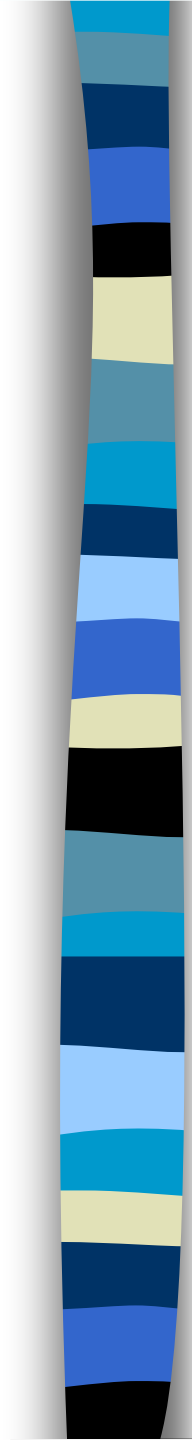
specification	representation
poly	<start, finish>
A	<0,1>
B	<2,5>

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

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;
```

Polynomial Addition - data structure 2

目的: 將方程式A(X)和方程式B(X)相加

padd()程式碼說明:
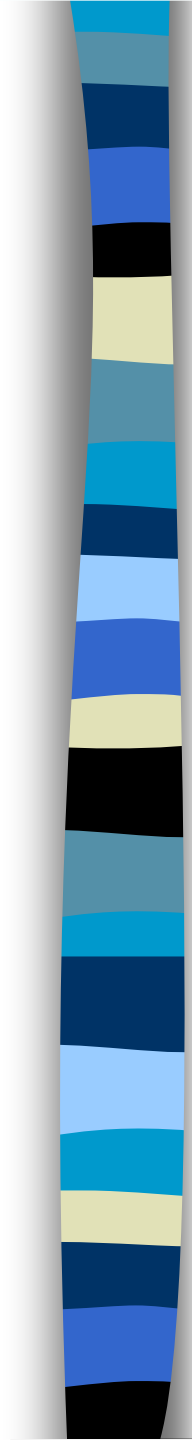
```c
void padd (int starta, int finisha, int startb, int finishb,int startd, int finishd){
    float coefficient;
    startd = avail;
    while (starta <= finisha && startb <= finishb){
        switch (COMPARE(terms[starta].expon,terms[startb].expon)) {
            case -1:
                attach(terms[startb].coef, terms[startb].expon);
                startb++;
                break;
```

avail是D(X)的開始位置

```c
#define COMPARE(x, y) ((x < y) ? -1: (x == y)? 0: 1)
```

a 的 expon < b 的 expon : case -1
如果方程式A的次方<方程式B的次方成立，則放入D(X)

```
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++;
}
```

Polynomial Addition - data structure 2

目的: 將方程式A(X)和方程式B(X)相加

```
case 0:
    coefficient = terms[starta].coef + terms[startb].coef;
    if (coefficient){
        attach (coefficient, terms[starta].expon);
        starta++;
        startb++;
    }
    break;
case 1:
    attach(terms[starta].coef, terms[starta].expon);
    starta++;
  }
}
```

```
#define COMPARE(x, y) ((x < y) ? -1: (x == y)? 0: 1)
```
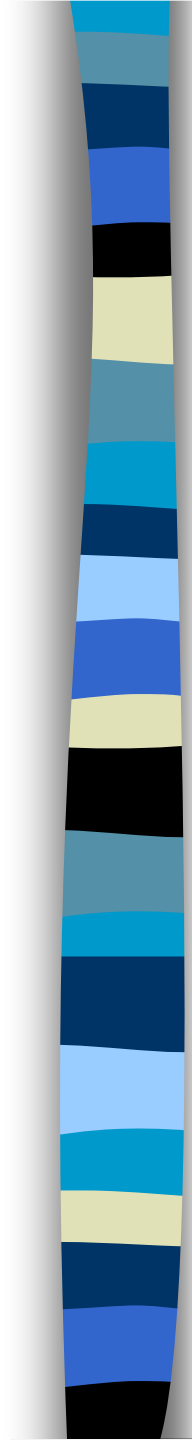
a 的 expon = b 的 expon : case 0
　　把兩方程式項目的係數相加且相加後係數不為0則放入D(X)
a 的 expon > b 的 expon : case 1
　　如果方程式A的次方>方程式B的次方成立，則放入D(X)

/* 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

Polynomial Addition - data structure 2

目的: 將方程式A(X)和方程式B(X)相加
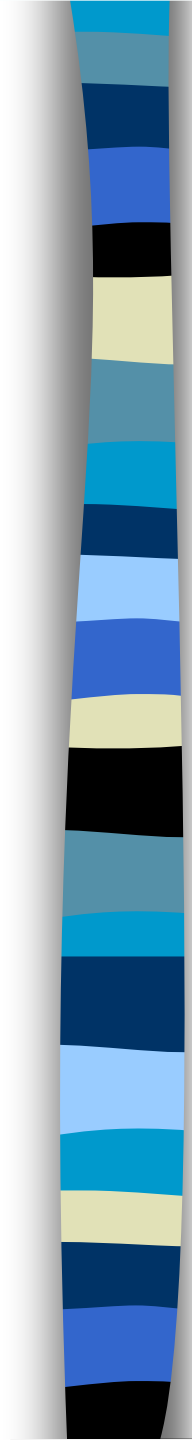
```
for( ; starta <= finisha; starta++){
    attach(terms[starta].coef, terms[starta].expon);
}
for( ; startb <= finishb; startb++){
    attach(terms[startb].coef, terms[startb].expon);
    finishd =avail -1;
}
}
```

第一個for loop
    如果多項式B已經結束時，將多項式A剩下的項目放進D(X)
第二個for loop
    如果多項式A已經結束時，將多項式B剩下的項目放進D(X)

```c
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;
}
```

Problem: Compaction is required
                when polynomials that are no longer needed.
                (data movement takes time.)

Polynomial Addition - data structure 2

目的: 將方程式A(X)和方程式B(X)相加

attach()程式碼說明:

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

如果沒有超過陣列最達長度，則將項目的指數及係數放到D(X)中

Sparse Matrix

$$
\begin{array}{c}
& \text{col 1} \quad \text{col 2} \quad \text{col 3} \\
\begin{array}{l}
\text{row 1} \\
\text{row 2} \\
\text{row 3} \\
\text{row 4} \\
\text{row 5}
\end{array}
\left[
\begin{array}{ccc}
-27 & 3 & 4 \\
6 & 82 & -2 \\
109 & -64 & 11 \\
12 & 8 & 9 \\
48 & 27 & 47
\end{array}
\right]
\end{array}
$$

5*3

$$
\begin{array}{c}
\text{col0} \quad \text{col1} \quad \text{col2} \quad \text{col3} \quad \text{col4} \quad \text{col5} \\
\begin{array}{l}
\text{row0} \\
\text{row1} \\
\text{row2} \\
\text{row3} \\
\text{row4} \\
\text{row5}
\end{array}
\left[
\begin{array}{cccccc}
15 & 0 & 0 & 22 & 0 & -15 \\
0 & 11 & 3 & 0 & 0 & 0 \\
0 & 0 & 0 & -6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
91 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 28 & 0 & 0 & 0
\end{array}
\right]
\end{array}
$$

6*6

(a)   15/15   (b)   8/36
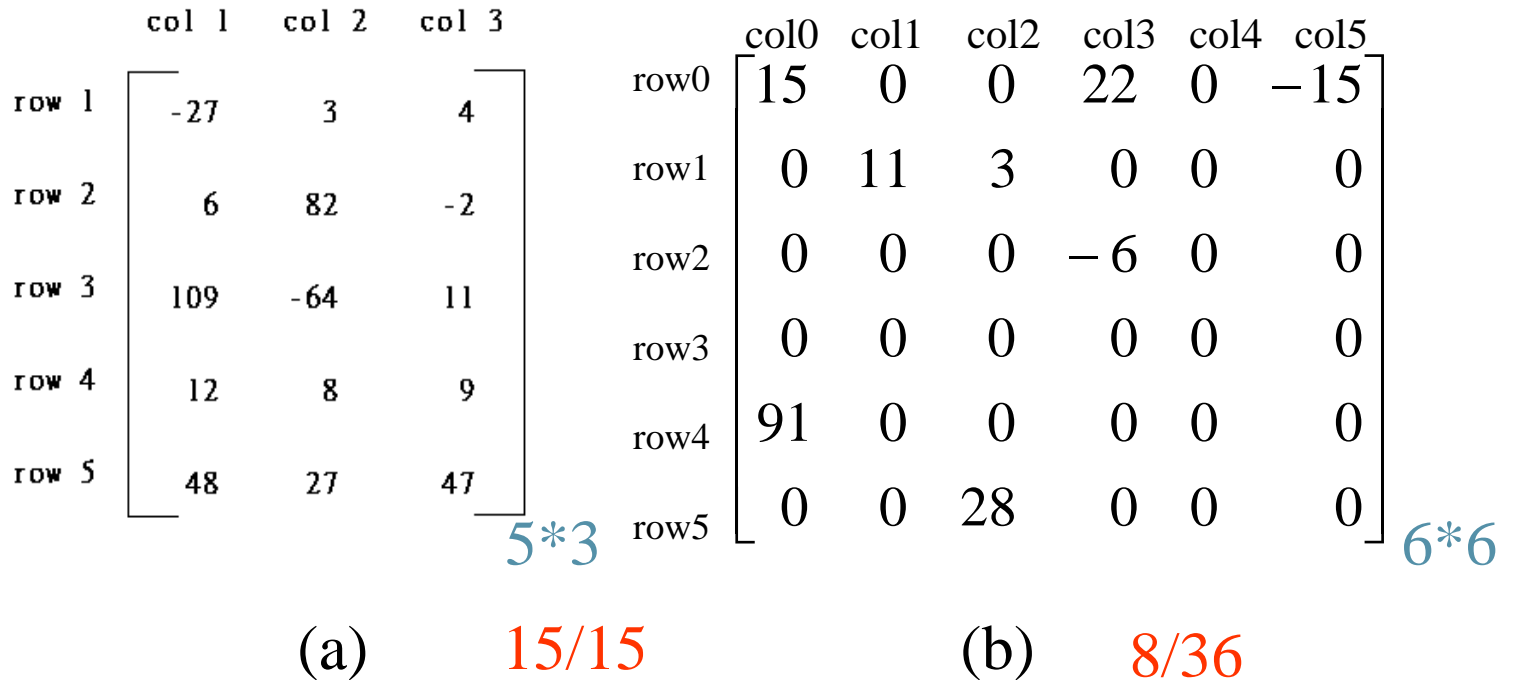
Figure 2.4: Two matrices

sparse matrix
data structure?

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* $\in$ *Sparse_Matrix*, *x* $\in$ *item, i, j, max_col,*
    max_row $\in$ *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 <u>interchanging the row and column value of every triple.</u>

Sparse_Matrix Add(*a, b*) ::=
>>> **if** <u>the dimensions of a and b are the same</u>
>>> **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** <u>number of columns in **a** equals number of rows in **b**</u>
>>> **return** the matrix *d* produced by multiplying a by b according to the formula: $d\,[i]\,[j] = \Sigma(a[i][k] \bullet b[k][j])$ where *d (i, j)* is the *(i,j)*th element
>>> **else return** error.

Structure 2.3: Abstract data type Sparse-Matrix

(1) Represented by a two-dimensional array.
     Sparse matrix wastes space.

(2) Each element is characterized by <row, col, value>.

	row	col	value			row	col	value
			# of rows (columns)					
			# of nonzero terms					
a[0]	6	6	8		b[0]	6	6	8
[1]	0	0	15		[1]	0	0	15
[2]	0	3	22		[2]	0	4	91
[3]	0	5	-15		[3]	1	1	11
[4]	1	1	11	transpose →	[4]	2	1	3
[5]	1	2	3		[5]	2	5	28
[6]	2	3	-6		[6]	3	0	22
[7]	4	0	91		[7]	3	2	-6
[8]	5	2	28		[8]	5	0	-15

(a)   (b)

row, column in ascending order

Figure 2.5: Sparse matrix and **its transpose** stored as triples

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]

# of rows
# of columns
# of nonzero terms

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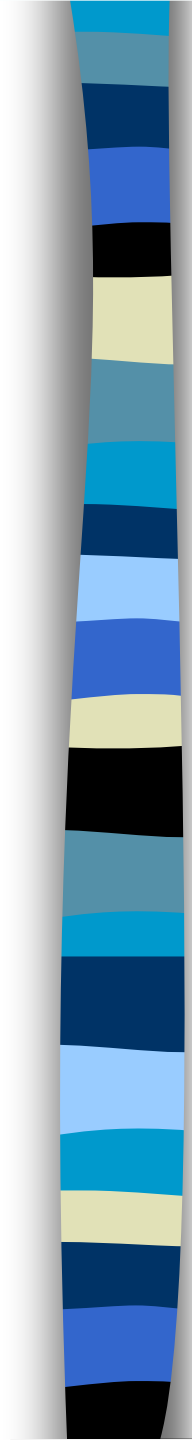
        (0, 5, -15) ====> (5, 0, -15)

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

Move elements down very often.

(2) For all elements in column j,

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

```c
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;
        for (i = 0; i < a[0].col; i++)
        /* transpose by columns in a */
            for( j = 1; j <=  n; j++)
            /*  find elements from the current column */
            if (a[j].col == i) {
            /* 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

Scan the array "**columns**" times.
The array has "**elements**" elements.
$\Longrightarrow O(columns*elements)$

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
void transpose (term a[], term b[]){
    int n, i, j, currentb;
    n = a[0].value;
    b[0].row = a[0].col;
    b[0].col = a[0].row;
    b[0].value = n;
    if (n > 0) {
        currentb = 1;
```

矩陣中非0項目

紀錄轉置後的矩陣，從位置1開始，位置0紀錄矩陣的基本資料

透過a[0]的基本資料，知道轉置後的資本資料。
row, col: a的col轉置後是b的row
　　　　　a的row轉置後是b的col
value: 轉置後的非0項目數量不變

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

$i = 0$

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1] $j = 1$	0	0	15	[1]	0	0	15
[2]	0	3	22	[2]			
[3]	0	5	-15	[3]			
[4]	1	1	11	[4]			
[5]	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7]	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1]	0	0	15	[1]	0	0	15
[2] j = 2	0	3	22	[2]			
[3]	0	5	-15	[3]			
[4]	1	1	11	[4]			
[5]	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7]	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1]	0	0	15	[1]	0	0	15
[2]	0	3	22	[2]			
[3] j = 3	0	5	-15	[3]			
[4]	1	1	11	[4]			
[5]	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7]	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1]	0	0	15	[1]	0	0	15
[2]	0	3	22	[2]			
[3]	0	5	-15	[3]			
[4] j = 4	1	1	11	[4]			
[5]	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7]	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1]	0	0	15	[1]	0	0	15
[2]	0	3	22	[2]			
[3]	0	5	-15	[3]			
[4]	1	1	11	[4]			
[5] j = 5	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7]	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value			row	col	value
a[0]	6	6	8	b[0]		6	6	8
[1]	0	0	15	[1]		0	0	15
[2]	0	3	22	[2]				
[3]	0	5	-15	[3]				
[4]	1	1	11	[4]				
[5]	1	2	3	[5]				
[6] j = 6	2	3	-6	[6]				
[7]	4	0	91	[7]				
[8]	5	2	28	[8]				

Transpose matrix

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7] j = 7	4	0	91
[8]	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]			
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```c
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 0

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8] j = 8	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]			
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 1

		row	col	value
a[0]		6	6	8
[1]	j = 1	0	0	15
[2]		0	3	22
[3]		0	5	-15
[4]		1	1	11
[5]		1	2	3
[6]		2	3	-6
[7]		4	0	91
[8]		5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]			
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```c
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

$i = 1$

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1]	0	0	15	[1]	0	0	15
[2] $j=2$	0	3	22	[2]	0	4	91
[3]	0	5	-15	[3]			
[4]	1	1	11	[4]			
[5]	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7]	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 1

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3] j = 3	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]			
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 1

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4] j = 4	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

$i = 1$

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5] $j = 5$	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

$i = 1$

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6] $j = 6$	2	3	-6
[7]	4	0	91
[8]	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]			
[5]			
[6]			
[7]			
[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

$i = 1$

	row	col	value		row	col	value
a[0]	6	6	8	b[0]	6	6	8
[1]	0	0	15	[1]	0	0	15
[2]	0	3	22	[2]	0	4	91
[3]	0	5	-15	[3]	1	1	11
[4]	1	1	11	[4]			
[5]	1	2	3	[5]			
[6]	2	3	-6	[6]			
[7] $j = 7$	4	0	91	[7]			
[8]	5	2	28	[8]			

Transpose matrix

目的: 實作轉置矩陣

transpose()程式碼說明:

```
for (i = 0; i < a[0].col; i++){
    for( j = 1; j <= n; j++){
        if (a[j].col == i) {
            b[currentb].row = a[j].col;
            b[currentb].col = a[j].row;
            b[currentb].value = a[j].value;
            currentb++;
        }
    }
}
```

i = 1

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8] j = 8	5	2	28

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]			
[5]			
[6]			
[7]			
[8]			

以此類推

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^2*rows" 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.

```
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++) /*計算 row_terms的值*
         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];
    /*計算 starting_pos的值*/
```

columns

elements

columns

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

*Program 2.9: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.

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```c
void fast_transpose(term a[], term b[]){
    int row_terms[MAX_COL], starting_pos[MAX_COL];
    int i, j;
    int num_cols = a[0].col;
    int num_terms = a[0].value;
    b[0].row = num_cols;
    b[0].col = a[0].row;
    b[0].value = num_terms;
```

透過a[0]的基本資料，知道轉置後的資本資料。
row, col: a的col轉置後是b的row
          a的row轉置後是b的col
value: 轉置後的非0項目數量不變

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

row_terms

0	1	2	3	4	5
0	0	0	0	0	0

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

i = 1

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

row_terms

0	1	2	3	4	5
1	0	0	0	0	0

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

$i = 2$

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

row_terms

0	1	2	3	4	5
1	0	0	1	0	0

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

i = 3

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

row_terms

0	1	2	3	4	5
1	0	0	1	0	1

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

i = 4

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

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

row_terms

0	1	2	3	4	5
1	1	0	1	0	1

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

i = 5

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

row_terms

0	1	2	3	4	5
1	1	1	1	0	1

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

i = 6

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

row_terms

0	1	2	3	4	5
1	1	1	2	0	1

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```c
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

i = 7

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

row_terms

0	1	2	3	4	5
2	1	1	2	0	1

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

i = 8

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

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

row_terms

0	1	2	3	4	5
2	1	2	2	0	1

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

$i = 1$

row_terms

0	1	2	3	4	5
2	1	2	2	0	1

starting_pos

0	1	2	3	4	5
1	3				

```
starting_pos[1]=
starting_pos[0] +
row_terms [0]
```

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

$i = 2$

row_terms

0	1	2	3	4	5
2	1	2	2	0	1

starting_pos

0	1	2	3	4	5
1	3	4			

```
starting_pos[2]=
starting_pos[1] +
row_terms [1]
```

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

i = 3

row_terms

0	1	2	3	4	5
2	1	2	2	0	1

starting_pos

0	1	2	3	4	5
1	3	4	6		

```
starting_pos[3]=
starting_pos[2] +
row_terms [2]
```

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

$i = 4$

row_terms

0	1	2	3	4	5
2	1	2	2	0	1

starting_pos

0	1	2	3	4	5
1	3	4	6	8	

```
starting_pos[4]=
starting_pos[3] +
row_terms [3]
```

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
```

$i = 5$

row_terms

0	1	2	3	4	5
2	1	2	2	0	1

starting_pos

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

```
starting_pos[5]=
starting_pos[4] +
row_terms [4]
```

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
```

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

i = 1

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]			
[3]			
[4]			
[5]			
[6]			
[7]			
[8]			

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

	row	col	value
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

a[2].col

i = 2

starting_pos

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

	row	col	value
b[0]	6	6	8
b[1]	0	0	15
b[2]			
b[3]			
b[4]			
b[5]			
b[6]	3	0	22
b[7]			
b[8]			

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```c
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
```

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

i = 3

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]			
[3]			
[4]			
[5]			
[6]	3	0	22
[7]			
[8]	5	0	-15

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

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

i = 4

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]			
[3]	1	1	11
[4]			
[5]			
[6]	3	0	22
[7]			
[8]	5	0	-15

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
```

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

i = 5

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]			
[3]	1	1	11
[4]	2	1	3
[5]			
[6]	3	0	22
[7]			
[8]	5	0	-15

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

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

i = 6

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]			
[3]	1	1	11
[4]	2	1	3
[5]			
[6]	3	0	22
[7]	3	2	-6
[8]	5	0	-15

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
}
```

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

i = 7

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]			
[6]	3	0	22
[7]	3	2	-6
[8]	5	0	-15

Fast transpose matrix

目的: 實作轉置矩陣

fast_transpose()程式碼說明:

```
if (num_terms > 0){
    for (i = 0; i < num_cols; i++){
        row_terms[i] = 0;
    }
    for (i = 1; i <= num_terms; i++){
        row_terms[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];
    }
    for (i = 1; i <= num_terms; i++){
        j = starting_pos[a[i].col]++;
        b[j].row = a[i].col;
        b[j].col = a[i].row;
        b[j].value = a[i].value;
    }
}
```

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

i = 8

starting_pos

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

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	-6
[8]	5	0	-15

Compare

	space	time
2D array	O(rows * cols)	O(rows * cols)
Transpose	O(elements)	O(cols * elements)
Fast Transpose	O(elements+MAX_COL)	O(cols + elements)

Sparse Matrix Multiplication

Definition: $[D]_{m*p}=[A]_{m*n} * [B]_{n*p}$

Procedure: Fix a row of A and find all elements in column j of B for j=0, 1, ..., p-1.

Alternative 1. Scan all of B to find all elements in j.

Alternative 2. Compute the transpose of B.

(Put all column elements consecutively)

$d_{ij}=a_{i0}*b_{0j}+a_{i1}*b_{1j}+...+a_{i(n-1)*}b_{(n-1)j}$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}=\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

106

Figure 2.5: Multiplication of two sparse matrices

【例如】

$$C_{11} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ \vdots \\ B_{n1} \end{bmatrix}$$

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

【實例】

×

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

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}_{3 \times 3}$$

1*1+2*1+3*0=3

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

Sparse Matrix Multiplication

目的: 實作稀疏矩陣a*矩陣b

做法說明:

$$\begin{pmatrix} 5 & 0 & 0 \\ 0 & 11 & 3 \\ 0 & 0 & 2 \end{pmatrix} \times \begin{pmatrix} 5 & 0 & 2 \\ 0 & 11 & 0 \\ 0 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 25 & 0 & 10 \\ 0 & 130 & 0 \\ 0 & 6 & 0 \end{pmatrix}$$

a

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 11 & 3 \\ 0 & 0 & 2 \end{bmatrix}$$

	row	col	value
a[0]	3	3	4
[1]	0	0	5
[2]	1	1	11
[3]	1	2	3
[4]	2	2	2
[5]	3	0	0

set boundary condition

b

$$\begin{bmatrix} 5 & 0 & 2 \\ 0 & 11 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

	row	col	value
b[0]	3	3	4
[1]	0	0	5
[2]	0	2	2
[3]	1	1	11
[4]	2	1	3

transpose

	row	col	value
newB[0]	3	3	4
[1]	0	0	5
[2]	1	1	11
[3]	1	2	3
[4]	2	0	2
[5]	3	0	0

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 1, rowBegin = 1, column = 0

a[i].row=0, row=0

=> False

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

i = 1, j = 1, rowBegin = 1, column = 0

newB[j].row=0, column=0
=> False

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 1, rowBegin = 1, column = 0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[1].col=0 = newB[1].col=0

=> case 0

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 1, rowBegin = 1, column = 0

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[i++].value=5

newB[j++].value=5

sum=25 i=2   j=2

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB	[0]	3	3	4
[1]	0	0	5		[1]	0	0	5
[2]	1	1	11		[2]	1	1	11
[3]	1	2	3		[3]	1	2	3
[4]	2	2	2		[4]	2	0	2
[5]	3	0	0		[5]	3	0	0

i = 2, j = 2, rowBegin = 1, column = 0

a[i].row=1, row=0

=> True

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB	[0]	3	3	4
[1]	0	0	5		[1]	0	0	5
[2]	1	1	11		[2]	1	1	11
[3]	1	2	3		[3]	1	2	3
[4]	2	2	2		[4]	2	0	2
[5]	3	0	0		[5]	3	0	0

i = 2, j = 2, rowBegin = 1, column = 0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

	row	col	value
d[0]	3	3	4
[1]	0	0	25
[2]			
[3]			
[4]			

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB	[0]	3	3	4
[1]	0	0	5		[1]	0	0	5
[2]	1	1	11		[2]	1	1	11
[3]	1	2	3		[3]	1	2	3
[4]	2	2	2		[4]	2	0	2
[5]	3	0	0		[5]	3	0	0

i = 2, j = 2, rowBegin = 1, column = 0

i=1   j=2   column=1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 2, rowBegin = 1, column = 1

a[i].row=0, row=0

=> False

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

i = 1, j = 2, rowBegin = 1, column = 1

newB[j].row=1, column=1
=> False

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 2, rowBegin = 1, column = 1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[1].col=0 < newB[2].col=1

=> case -1

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

$i = 1, j = 2, rowBegin = 1, column = 1$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

i=2

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 1, column = 1

a[i].row=1, row=0

=> True

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial
rowsA=a[0].row=3   colsB=b[0].col=3
colsA=a[0].col=3
totalA=a[0].value=4   totalB=b[0].value=4
row=a[1].row=0

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 1, column = 1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

	row	col	value
d[0]	3	3	4
[1]	0	0	25
[2]			
[3]			
[4]			

storeSum()判斷
沒有新的sum值需要儲存
故陣列沒有增加資料

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 1, column = 1

i=1   j=4   column=2

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 4, rowBegin = 1, column = 2

a[i].row=0, row=0

=> False

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 4, rowBegin = 1, column = 2

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

newB[j].row=2, column=2
=> False

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 4, rowBegin = 1, column = 2

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[1].col=0 = newB[4].col=0

=> case 0

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 4, rowBegin = 1, column = 2

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[i++].value=5

newB[j++].value=2

sum=10 i=2   j=5

initial

rowsA=a[0].row=3 colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4 totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 5, rowBegin = 1, column = 2

a[i].row=1, row=0

=> True

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB	[0]	3	3	4
[1]	0	0	5		[1]	0	0	5
[2]	1	1	11		[2]	1	1	11
[3]	1	2	3		[3]	1	2	3
[4]	2	2	2		[4]	2	0	2
[5]	3	0	0		[5]	3	0	0

$i = 2, j = 5, rowBegin = 1, column = 2$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;   break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

	row	col	value
d[0]	3	3	4
[1]	0	0	25
[2]	0	2	10
[3]			
[4]			

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 5, rowBegin = 1, column = 2

i=1   j=5   column=3

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 5, rowBegin = 1, column = 3

a[i].row=0, row=0
=> False

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

initial

rowsA=a[0].row=3 colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4 totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

$i = 1, j = 5, rowBegin = 1,$ column = 3

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

newB[j].row=3, column=3

=> False

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 1, j = 5, rowBegin = 1, column = 3

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[1].col=0 = newB[5].col=0

=> case 0

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value			row	col	value
a[0]	3	3	4	newB	[0]	3	3	4
[1]	0	0	5		[1]	0	0	5
[2]	1	1	11		[2]	1	1	11
[3]	1	2	3		[3]	1	2	3
[4]	2	2	2		[4]	2	0	2
[5]	3	0	0		[5]	3	0	0

$i = 1, j = 5, rowBegin = 1, column = 3$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[i++].value=5

newB[j++].value=0

sum=0   i=2   **j=6**

initial

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[1].row=0

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 6, rowBegin = 1, column = 3

column = 0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

i=2   rowBegin=2   row=1   134

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 1, rowBegin = 2, column = 0

a[i].row=1, row=1

=> False

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 1, rowBegin = 2, column = 0

```c
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

newB[j].row=0, column=0
=> False

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 1, rowBegin = 2, column = 0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column = newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[2].col=1 > newB[1].col=0
=> case 1

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 1, rowBegin = 2, column = 0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

j=2

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 2, column = 0

a[i].row=1, row=1

=> False

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;   break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

Round 2

rowsA=a[0].row=3 colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4 totalB=b[0].value=4

row=a[2].row=1

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

i = 2, j = 2, rowBegin = 2, column = 0

newB[j].row=1, column=0
=> True

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 2, column = 0

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

	row	col	value
d[0]	3	3	4
[1]	0	0	25
[2]	0	2	10
[3]			
[4]			

storeSum()判斷
沒有新的sum值需要儲存
故陣列沒有增加資料

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

$i = 2, j = 2, rowBegin = 2, column = 0$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

i=2   newB[2].row=1 column=1

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

$i = 2$, $j = 2$, rowBegin = 2, column = 1

a[i].row=1, row=1

=> False

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 2, column = 1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

newB[j].row=1, column=1
=> False

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 2, rowBegin = 2, column = 1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[2].col=1 = newB[2].col=1
=> case 0

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

$i = 2, j = 2, rowBegin = 2, column = 1$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[i++].value=11

newB[j++].value=11

sum=121   i=3   j=3

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 3, j = 3, rowBegin = 2, column = 1

a[i].row=1, row=1

=> False

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

$i = 3, j = 3, \text{rowBegin} = 2, \text{column} = 1$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

newB[j].row=1, column=1

=> False

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value		row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 3, j = 3, rowBegin = 2, column = 1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;   break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[3].col=2 = newB[3].col=2

=> case 0

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 3, j = 3, rowBegin = 2, column = 1

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

a[i++].value=3

newB[j++].value=3

sum=11*11+3*3=130   i=4   j=4

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 4, j = 4, rowBegin = 2, column = 1

a[i].row=2, row=1

=> True

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB	[0]	3	3	4
[1]	0	0	5		[1]	0	0	5
[2]	1	1	11		[2]	1	1	11
[3]	1	2	3		[3]	1	2	3
[4]	2	2	2		[4]	2	0	2
[5]	3	0	0		[5]	3	0	0

$i = 4, j = 4, rowBegin = 2, column = 1$

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;   break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

	row	col	value
d[0]	3	3	4
[1]	0	0	25
[2]	0	2	10
[3]	1	1	130
[4]			

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 4, j = 4, rowBegin = 2, column = 1

i=2   j=4   column=2

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

Round 2

rowsA=a[0].row=3   colsB=b[0].col=3

colsA=a[0].col=3

totalA=a[0].value=4   totalB=b[0].value=4

row=a[2].row=1

	row	col	value			row	col	value
a[0]	3	3	4	newB[0]	3	3	4
[1]	0	0	5	[1]	0	0	5
[2]	1	1	11	[2]	1	1	11
[3]	1	2	3	[3]	1	2	3
[4]	2	2	2	[4]	2	0	2
[5]	3	0	0	[5]	3	0	0

i = 2, j = 4, rowBegin = 2, column = 2

```
for (i = 1; i <= totalA; ) {
    column = newB[1].row;
    for (j = 1; j <= totalB+1;) {
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            for (; newB[j].row == column; j++) ;
            column =newB[j].row;
        }else if(newB[j].row != column){
            storeSum(d, &totalD, row, column, &sum);
            sum = 0;
            i = rowBegin;
            column = newB[j].row;
        }else switch (COMPARE (a[i].col, newB[j].col)) {
            case -1:
                i++;    break;
            case 0:
                sum += (a[i++].value * newB[j++].value);break;
            case 1:
                j++;
        }
    }
    for (; a[i].row == row; i++) ;
    rowBegin = i;
    row = a[i].row;
}
```

以此類推

	row	col	value
d[0]	3	3	4
[1]	0	0	25
[2]	0	2	10
[3]	1	1	130
[4]	2	1	6

$$\begin{pmatrix} 5 & 0 & 0 \\ 0 & 11 & 3 \\ 0 & 0 & 2 \end{pmatrix} \times \begin{pmatrix} 5 & 0 & 2 \\ 0 & 11 & 0 \\ 0 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 25 & 0 & 10 \\ 0 & 130 & 0 \\ 0 & 6 & 0 \end{pmatrix}$$

```c
void mmult (term a[ ], term b[ ], term d[ ] )
/* multiply two sparse matrices */
{
    int i, j, column, totalb = b[].value, totald = 0;
    int rows_a = a[0].row, cols_a = a[0].col,
    totala = a[0].value; int cols_b = b[0].col,
    int row_begin = 1, row = a[1].row, sum =0;
    int new_b[MAX_TERMS][3];
    if (cols_a != b[0].row){
    /*compare the row of a and the col of b*/
        fprintf (stderr, "Incompatible matrices\n");
        exit (1);
}
```

```
fast_transpose(b, new_b); /* the transpose of b is placed in new_b */
/* set boundary condition */   cols_b + totalb
a[totala+1].row = rows_a; /* a[0].row*/
new_b[totalb+1].row = cols_b;
new_b[totalb+1].col = 0;
for (i = 1; i <= totala; ) {/* a[0].val* /    at most rows_a times
    column = new_b[1].row;  /* b[1].col*/
    for (j = 1; j <= totalb+1;) {  /*b[0].val*/  at most cols_b times
    /* mutiply row of a by column of b */
    if (a[i].row != row)  {   /* a[1].row */
        storesum(d, &totald, row, column, &sum);
        i = row_begin;
        for (; new_b[j].row == column; j++)
          ;
        column =new_b[j].row;  /* next col */
    }
```
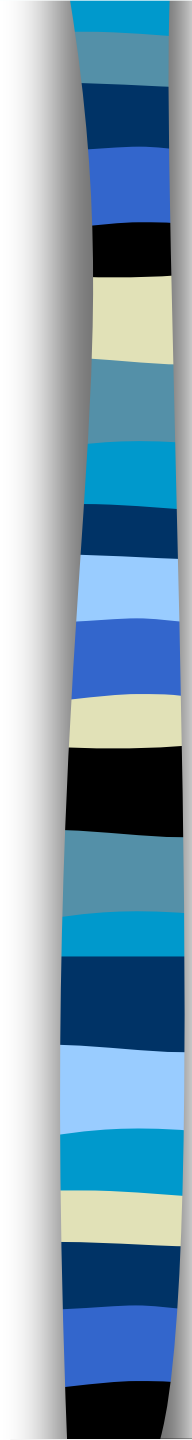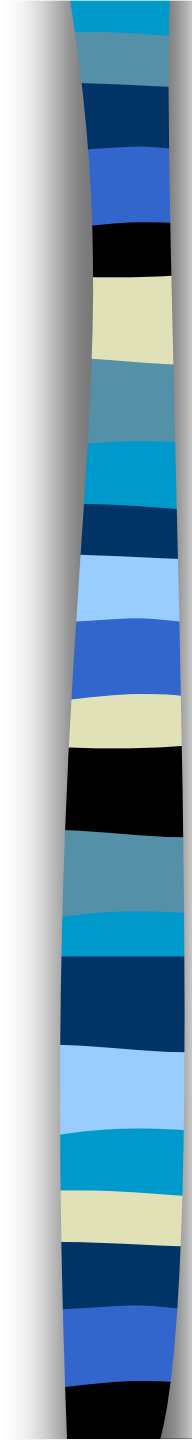
```c
        else switch (COMPARE (a[i].col, new_b[j].col)) {
            case -1: /* go to next term in a */
                    i++; break;
            case 0: /* add terms, go to next term in a and b */
                    sum += (a[i++].value * new_b[j++].value);
                    break;
            case 1: /* advance to next term in b*/
                    j++
        }
    } /* end of for j <= totalb+1 */          at most cols_b times
    for (; a[i].row == row; i++)
        ;
    row_begin = i; row = a[i].row;
} /* end of for i <=totala */              at most rows_a times
d[0].row = rows_a; /* a[0].row*/
d[0].col = cols_b; /* b[0].cols*/
d[0].value = totald;
}
```

```c
void storesum(term d[ ], int *totald, int row, int column,
                                              int *sum)
{
/* if *sum != 0, then it along with its row and column
     position is stored as the *totald+1 entry in d */
     if (*sum)
        if (*totald < MAX_TERMS) {
           d[++*totald].row = row;
           d[*totald].col = column;
           d[*totald].value = *sum;
         }
       else {
         fprintf(stderr, "Numbers of terms in product
                                exceed %d\n", MAX_TERMS);
   exit(1);
   }
}
```

Program 2.11: storsum function