Arrays

College of Computer Science, CQU

Outline

- Array ADT
- Matrix
- Symmetric Matrix
- Triangular Matrix
- Symmetric Band Matrix
- Sparse Matrix

    Representation， Transposing

Arrays

- Array:

  a set of pairs (index and value)

- data structure:

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

- representation (possible):

  implemented by using consecutive memory.

The Array ADT

- **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.

  Methods:

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

  Array Create(j, list)

  // **return** an array of j dimensions where list is a j-tuple whose kth element is the

  //size of the kth dimension. Items are undefined.

  Item Retrieve(A, i)

  // **if** (i   index) **return** the item associated with index value i in array A

  // **else return** error

  Array Store(A, i, x)

  // **if (**i in index) **return** an array that is identical to array A except the new pair

  //<i, x> has been inserted **else return** error

Matrices

- Two-dimensional arrays are a particularly common representation for matrices.
- A matrix, also referred to as a general matrix, is an m by n ordered collection of numbers. It is represented symbolically as:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

- where the matrix is named **A** and has m rows and n columns. And a_{ij} is the element in ith row and jth column of matrix A.

Matrices

- A matrix appears as two-dimensional,but physically it is stored in a linear fashion. How to represent this two-dimensional array?

$$A=\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrices

- Common ways to index into multi-dimensional arrays include:
- Row-major order:

  The elements of each row are stored in order.

1	2	3	4	5	6	7	8	9

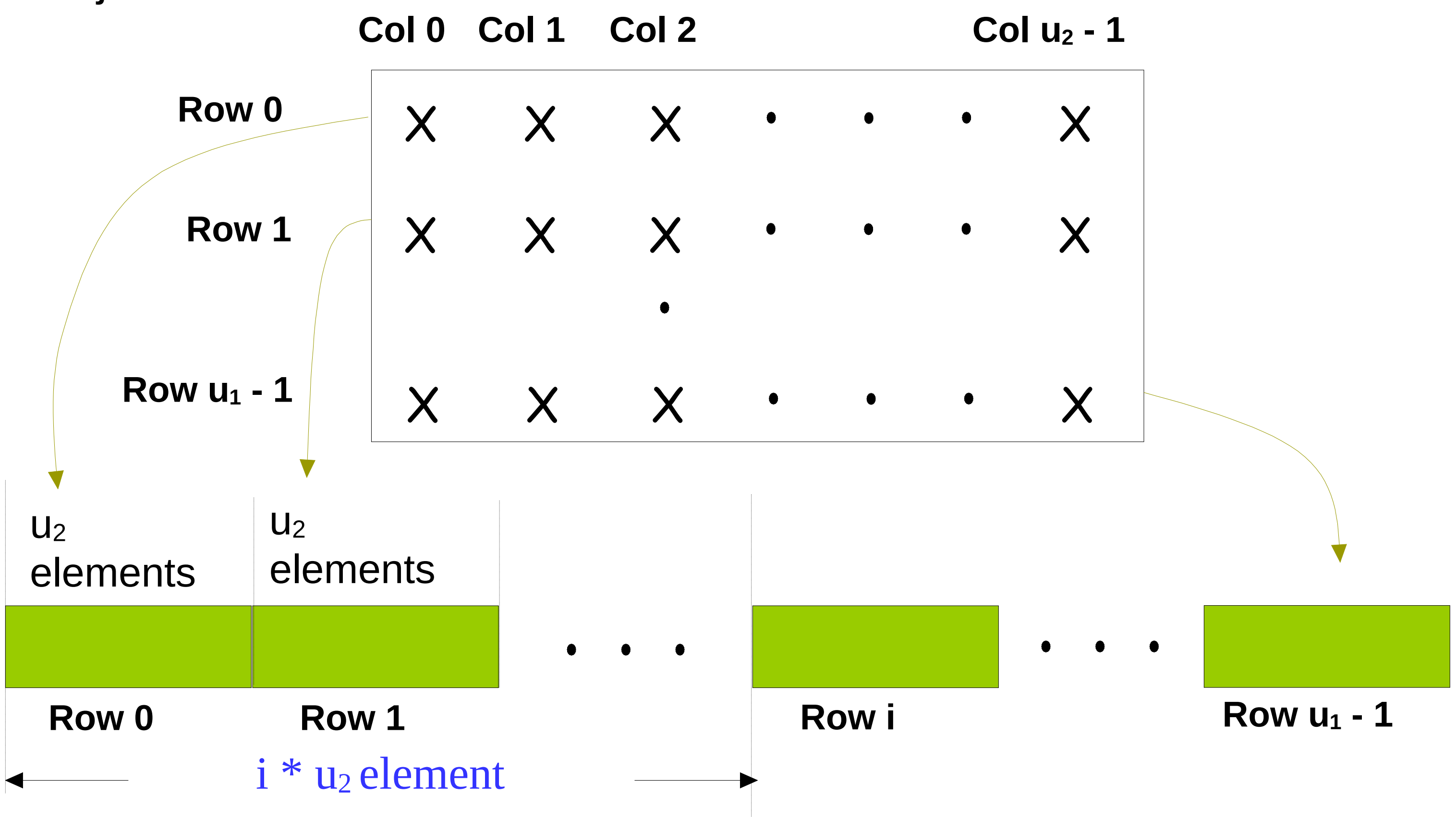
- Column-major order:

  The elements of each column are stored in order.

1	4	7	2	5	8	3	6	9

Matrices

Row-major order:



Matrices

- So,in order to map logical view to physical structure,we need indexing formula.

  - Row-major order:Assume that the base address is at M,the address of a_{ij} will be obtained as

    $Address(a_{ij})=M+(i-1)*n+j-1$

  - Column-major order:Considering the base address at M,the formula will stand as

    $Address(a_{ij})=M+(j-1)*n+i-1$

Symmetric Matrix

- The matrix **A** is symmetric if it has the property **A** equal to $\mathbf{A}^T$, which means:
  - It has the same number of rows as it has columns; that is, it has n rows and n columns.
  - The value of every element a_{ij} on one side of the main diagonal equals its mirror image a_{ji} on the other side: a_{ij} equal to a_{ji}.

Symmetric Matrix

- The following matrix illustrates a symmetric matrix of order n; that is, it has n rows and n columns. The subscripts on each side of the diagonal appear the same to show which elements are equal:

$$A = \begin{bmatrix} a_{11} & a_{21} & a_{31} & \cdots & a_{n1} \\ a_{21} & a_{22} & a_{32} & & \vdots \\ a_{31} & a_{32} & a_{33} & & \vdots \\ \vdots & & & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{bmatrix}$$

Symmetric Matrix

- When a symmetric matrix is stored in lower-packed storage mode, the lower triangular part of the symmetric matrix is stored, including the diagonal, in a one-dimensional array.
- The lower triangle can be packed by row or columns. The matrix is packed sequentially row by row (column by column) in n(n+1)/2 elements of a one-dimensional array.
- When the matrix is packed sequentially row by row ,to calculate the location k of element a_{ij} of matrix **A** in an array, use the following formula:

$$k=i*(i-1)/2+j-1 \qquad i>=j, \text{ lower triangular part}$$

$$k=j*(j-1)/2+i-1 \qquad i<j, \text{ upper triangular part}$$

Triangular Matrix

A matrix of the form

$$
\begin{bmatrix}
a_{0,0} & & & & & \\
a_{1,0} & a_{1,1} & & & & \\
a_{2,0} & a_{2,1} & a_{2,2} & & & \\
\vdots & & & & & \\
a_{n-1,0} & a_{n-1,1} & & \cdots & a_{n-1,n-1} & \\
a_{n,0} & a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n}
\end{bmatrix}
$$

is called a **triangular matrix**.

Triangular Matrix

- There are two types of triangular matrices: upper triangular matrix and lower triangular matrix. Triangular matrices have the same number of rows as they have columns; that is, they have n rows and n columns.
- A matrix **U** is an upper triangular matrix if its nonzero elements are found only in the upper triangle of the matrix, including the main diagonal; that is: u_{ij} equal to 0（or constant C） if i greater than j
- A matrix **L** is an lower triangular matrix if its nonzero elements are found only in the lower triangle of the matrix, including the main diagonal; that is: l_{ij} equal to 0 （or constant C） if i less than j

Triangular Matrix

- The following matrices, **U** and **L**, illustrate upper and lower triangular matrices of order n, respectively:

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & . & . & . & u_{1n} \\ 0 & u_{22} & u_{23} & & & & . \\ 0 & 0 & u_{33} & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & . \\ 0 & . & . & . & . & 0 & u_{nn} \end{bmatrix} \qquad L = \begin{bmatrix} l_{11} & 0 & 0 & . & . & . & 0 \\ l_{21} & l_{22} & 0 & & & & . \\ l_{31} & l_{32} & l_{33} & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & 0 \\ l_{n1} & . & . & . & . & . & l_{nn} \end{bmatrix}$$

Triangular Matrix

- When a lower-triangular matrix is stored in lower-triangular-packed storage mode, the lower triangle of the matrix is stored, including the diagonal, in a one-dimensional array. The lower triangle is packed by row or by columns. The elements are packed sequentially, row by row (column by column), in n(n+1)/2 elements of a one-dimensional array. To calculate the location of each element of the triangular matrix in the array, use the technique described in Symmetric Matrix.
- When an upper-triangular matrix is stored in upper-triangular-packed storage mode, the upper triangle of the matrix is stored, including the diagonal, in a one-dimensional array.

Symmetric Band Matrix

- A symmetric band matrix is a symmetric matrix whose nonzero elements are arranged uniformly near the diagonal, such that: a_{ij} equal to 0 if $|i-j|$ greater than k, where k is the half band width.

Symmetric Band Matrix

- The following matrix illustrates a symmetric band matrix of order n, where the half band width k equal to q-1:

$$\left|\leftarrow \; k \; \rightarrow\right|$$

$$A = \begin{bmatrix}
a_{11} & a_{21} & a_{31} & . & . & a_{q1} & 0 & . & . & 0 \\
a_{21} & a_{22} & a_{32} & & & & 0 & & & . \\
a_{31} & a_{32} & a_{33} & & & & & 0 & & . \\
. & & & . & & & & & & 0 \\
. & & & & . & & & & & . \\
a_{q1} & & & & & . & & & & . \\
0 & & & & & & . & & & . \\
. & 0 & & & & & & . & & . \\
. & & 0 & & & & & & . & \\
0 & . & . & 0 & . & . & . & & & a_{nn}
\end{bmatrix}$$

- Only the band elements of the symmetric band matrix are stored.

Sparse Matrix

A sparse matrix is a matrix having a relatively small number of nonzero elements.

$$
\begin{array}{c|ccc|}
 & \text{col 1} & \text{col 2} & \text{col 3} \\
\hline
\text{row 1} & -27 & 3 & 4 \\
\text{row 2} & 6 & 82 & -2 \\
\text{row 3} & 109 & -64 & 11 \\
\text{row 4} & 12 & 8 & 9 \\
\text{row 5} & 48 & 27 & 47 \\
\hline
\end{array}
$$

15/15

	col1	col2	col3	col4	col5	col6
row0	15	0	0	22	0	15
row1	0	11	3	0	0	0
row2	0	0	0	6	0	0
row3	0	0	0	0	0	0
row4	91	0	0	0	0	0
row5	0	0	28	0	0	0

8/36

sparse matrix
data structure?

Sparse Matrix Representation

- The standard representation of a matrix is a two dimensional array defined as

  a[***MAX_ROWS***][***MAX_COLS***]

  - We can locate quickly any element by writing ***a***[***i***][***j***]
- Sparse matrix wastes space
  - We must consider alternate forms of representation.
  - Our representation of sparse matrices should store only nonzero elements.
  - Each element is characterized by <row, col, value>.

Sparse Matrix Representation

- Figure shows how the sparse matrix is represented in the array ***a***.
  - Represented by a two-dimensional array.
  - Each element is characterized by <row, col, value>.

	col1	col2	col3	col4	col5	col6
row0	15	0	0	22	0	15
row1	0	11	3	0	0	0
row2	0	0	0	6	0	0
row3	0	0	0	0	0	0
row4	91	0	0	0	0	0
row5	0	0	28	0	0	0

row, column in ascending order

rows columns nonzero terms

	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]	5	2	28

(a)

transpose

	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

(b)

Transposing A Matrix

- Transpose a Matrix
  - 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.
  - For all elements in column j,
    - place element <i, j, value> in element <j, i, value>

Transposing A Matrix

Assign
A[i][j] to B[j][i]

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

For all columns i

For all elements in column j

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

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

==> O(columns*elements)

EX: A[6][6] transpose to B[6][6]

i=1 j=8
a[i].col = 2 != i

Matrix A

	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]	5	2	28

	Row	Col	Value
0	6	6	8
1	0	0	15
2	0	4	91
3	1	1	11

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

Set Up row & column in B[6][6]

And So on…

Reference

- 《数据结构（ C 语言版）》，严蔚敏，吴伟民编著，清华大学出版社，1997年第1版,P91-99

-END-