《数据结构与算法》课程组
重庆大学计算机学院

Data Structures & Algorithms

7

ARRAY

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 ∈ 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:

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

- where the matrix is named A and has m rows and n columns. And a_{ij} is the element in i-th row and j-th 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:**

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

- **Row-major order:**

  The elements of each row are stored in order.

1	2	3	4	5	6	7	8	9

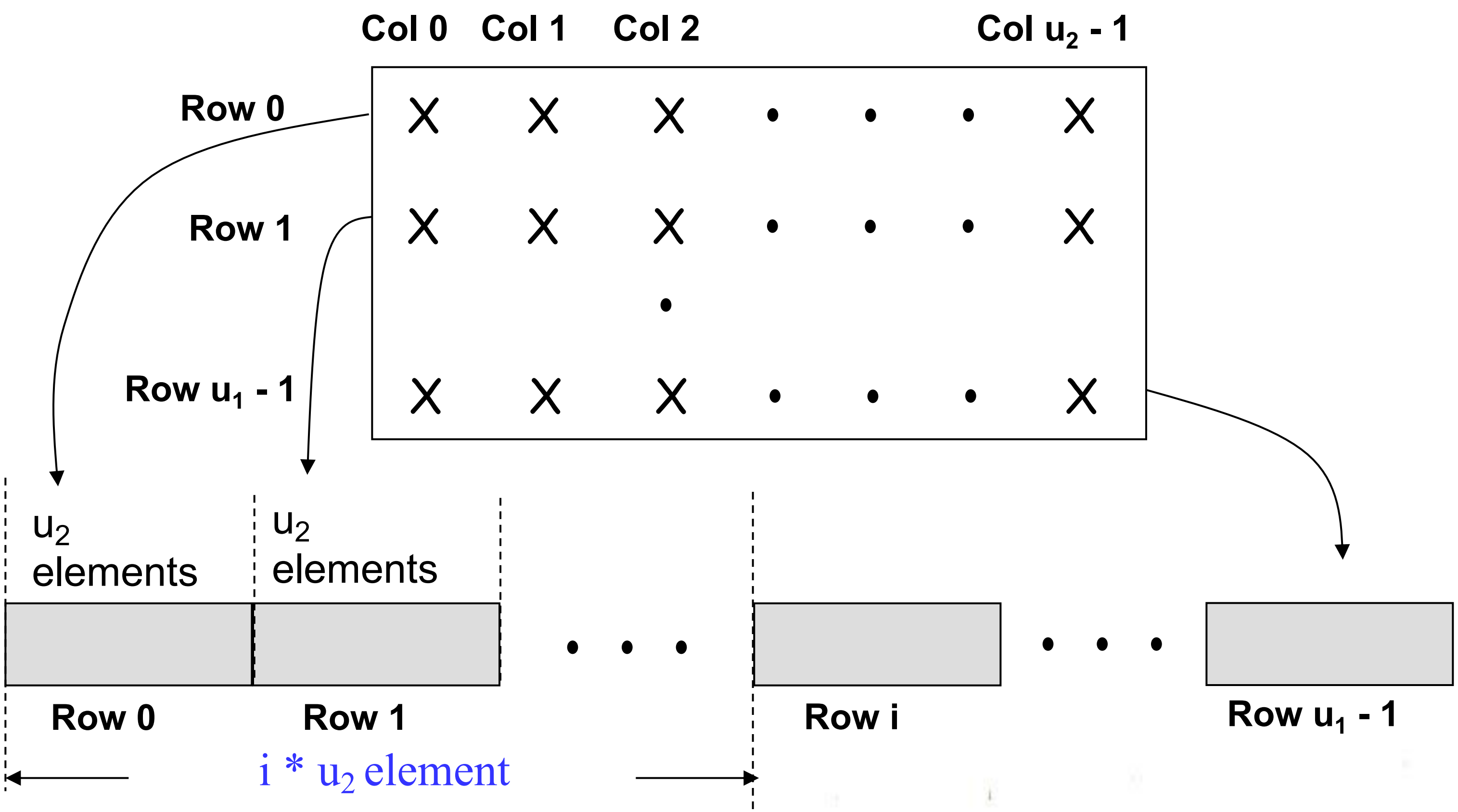
- **Column-major order:**

  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**

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

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

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

Symmetric Matrix

- The matrix A is symmetric if it has the property A equal to 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:**

$$
\boldsymbol{A} = \begin{bmatrix}
a_{11} & a_{21} & a_{31} & \cdot & \cdot & \cdot & a_{n1} \\
a_{21} & a_{22} & a_{32} & & & & \cdot \\
a_{31} & a_{32} & a_{33} & & & & \cdot \\
\cdot & & & \cdot & & & \cdot \\
\cdot & & & & \cdot & & \cdot \\
\cdot & & & & & \cdot & \cdot \\
a_{n1} & \cdot & \cdot & \cdot & \cdot & \cdot & a_{nn}
\end{bmatrix}
$$

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:

  $k=i*(i-1)/2+j-1$ $\quad i>=j$, lower triangular part

  $k=j*(j-1)/2+i-1$ $\quad i<j$, upper triangular part

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

- 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 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:**

$$
\begin{array}{c}
\qquad\qquad \left| \leftarrow \; k \; \rightarrow \right| \qquad\qquad \\
A = \begin{bmatrix}
a_{11} & a_{21} & a_{31} & \cdot & \cdot & a_{q1} & 0 & \cdot & \cdot & 0 \\
a_{21} & a_{22} & a_{32} & & & & & 0 & & \cdot \\
a_{31} & a_{32} & a_{33} & & & & & & 0 & \cdot \\
\cdot & & & \cdot & & & & & & 0 \\
\cdot & & & & \cdot & & & & & \cdot \\
a_{q1} & & & & & \cdot & & & & \cdot \\
0 & & & & & & \cdot & & & \cdot \\
\cdot & 0 & & & & & & \cdot & & \cdot \\
\cdot & & 0 & & & & & & \cdot & \\
0 & \cdot & \cdot & 0 & \cdot & \cdot & \cdot & & & a_{nn}
\end{bmatrix}
\end{array}
$$

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

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

– Represented by a two-dimensional array.

– Each element is characterized by <row, col, value>.

$$
\begin{array}{c c}
 & \begin{array}{cccccc} \text{col1} & \text{col2} & \text{col3} & \text{col4} & \text{col5} & \text{col6} \end{array} \\
\begin{array}{c} \text{row0} \\ \text{row1} \\ \text{row2} \\ \text{row3} \\ \text{row4} \\ \text{row5} \end{array} &
\begin{bmatrix}
15 & 0 & 0 & 22 & 0 & -15 \\
0 & 11 & 3 & 0 & 0 & 0 \\
0 & 0 & 0 & -6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
91 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 28 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

row, column in ascending order

rows columns

nonzero terms

transpose

	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)

	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

```
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++;
         }
   }
}
```

(Selection sort)

==> O(columns*elements)

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

《数据结构与算法》课程组

重庆大学计算机学院

End of Chapter