



Array

- The array as an abstract data type
- Sparse matrix
- The String as an data type



Array

- The array as an abstract data type
- Sparse matrix
- The string as an data type



Array

- A set of pairs: <index, value>
- Two operations: retrieve, store
- 在高级语言中的一维数组只能按元素的下标直接存取数组元素的值。

0	1	2	3	4	5	6	7	8	9
35	27	49	18	60	54	77	83	41	02



Array

多维数组

- 是一维数组的推广
- 特点：每一个数据元素可以有多个直接前驱和多个直接后继
- 数组元素的下标一般具有固定的下界和上界
- 比其他复杂的非线性结构简单

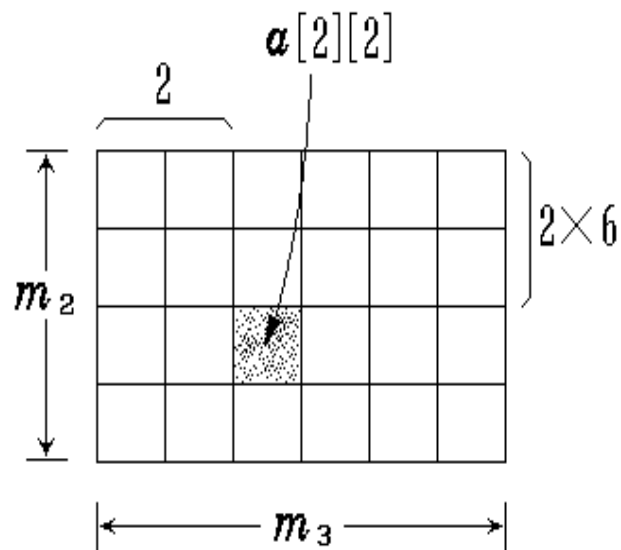
例如

- 二维数组的数组元素有两个直接前驱，两个直接后继，必须有两个下标（行、列）以标识该元素的位置。

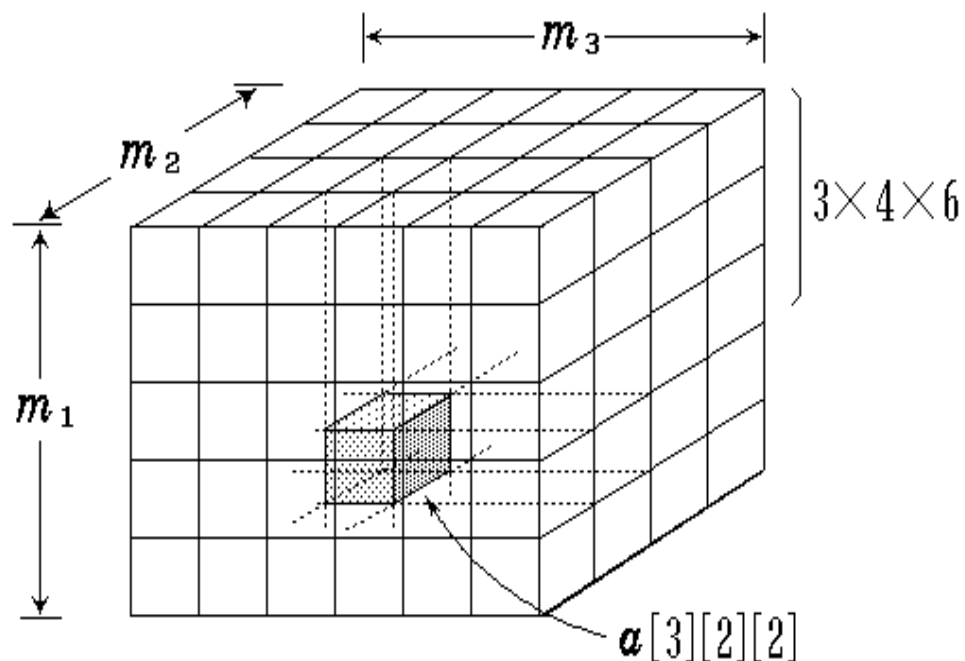
二维数组

三维数组

$$m_1=5 \quad m_2=4 \quad m_3=6$$



- 行向量 下标 i
- 列向量 下标 j
-



- 页向量 下标 i
- 行向量 下标 j
- 列向量 下标 k



GeneralArray

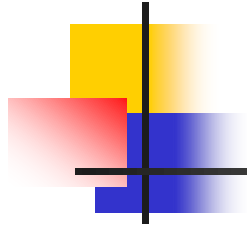
```
class GeneralArray {
```

```
// a set of pairs <index, value> where for each value of  
// index in IndexSet there is a value of type float. IndexSet is  
// a finite ordered set of one or more dimensions.
```

```
public:
```

```
    GeneralArray(int j, RangeList list, float initValue =  
                                                         defaultValue);
```

```
// The constructor GeneralArray creates a j  
// dimensional array of floats; the range of the kth  
// dimension is given by the kth element of list.  
// For all  $i \in \text{IndexSet}$ , insert  $\langle i, \text{initValue} \rangle$  into the array.
```



```
float Retrieve(index i);
```

```
// if ( $i \in \text{IndexSet}$ ) return the float associated with i in the  
// array; else throw an exception.
```

```
void Store(index i, float x);
```

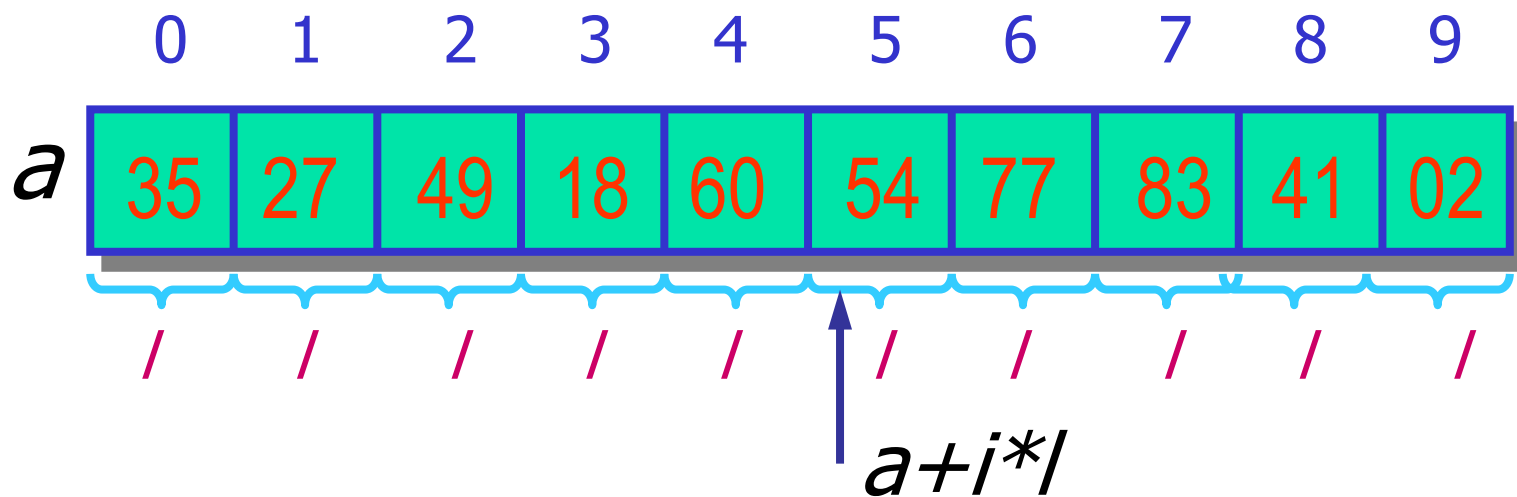
```
// if ( $i \in \text{IndexSet}$ ) replace the old value associated with i  
// by x; else throw an exception.
```

```
}; //end of GeneralArray
```

数组的连续存储方式

一维数组

$$\text{LOC}(i) = \begin{cases} a, & i = 0 \\ \text{LOC}(i-1) + l = a + i * l, & i > 0 \end{cases}$$



二维数组中数组元素的顺序存放

1) 行优先存放

设数组开始存放位置 $\text{LOC}(0, 0) = a$, 每个元素占用 l 个存储单元

$$\text{LOC}(j, k) = a + (j * m + k) * l$$

$$a = \begin{pmatrix} a[0][0] & a[0][1] & \Lambda & a[0][m-1] \\ a[1][0] & a[1][1] & \Lambda & a[1][m-1] \\ a[2][0] & a[2][1] & \Lambda & a[2][m-1] \\ \text{M} & \text{M} & \text{O} & \text{M} \\ a[n-1][0] & a[n-1][1] & \Lambda & a[n-1][m-1] \end{pmatrix}$$



2) 列优先存放

设数组开始存放位置 $\text{LOC}(0, 0) = a$, 每个元素占用 l 个存储单元

$$\text{LOC}(j, k) = a + (k * n + j) * l$$

$$a = \begin{pmatrix} a[0][0] & a[0][1] & \Lambda & a[0][m-1] \\ a[1][0] & a[1][1] & \Lambda & a[1][m-1] \\ a[2][0] & a[2][1] & \Lambda & a[2][m-1] \\ \text{M} & \text{M} & \text{O} & \text{M} \\ a[n-1][0] & a[n-1][1] & \Lambda & a[n-1][m-1] \end{pmatrix}$$

三维数组

- 各维元素个数为 m_1, m_2, m_3
- 下标为 i_1, i_2, i_3 的数组元素的存储地址：
(按页 / 行 / 列存放)

$$\text{LOC}(i_1, i_2, i_3) = a + (i_1 * m_2 * m_3 + i_2 * m_3 + i_3) * l$$

前 i_1 页
总元素
个数

第 i_1 页
前 i_2 行
总元素
个数

第 i_2 行
前 i_3 列
元素个
数



n 维数组

- 各维元素个数为 $m_1, m_2, m_3, \dots, m_n$
- 下标为 $i_1, i_2, i_3, \dots, i_n$ 的数组元素的存储地址:

$$\begin{aligned} \text{LOC} (i_1, i_2, \dots, i_n) &= a + \\ & (i_1 * m_2 * m_3 * \dots * m_n + i_2 * m_3 * m_4 * \dots * m_n + \dots \dots \\ & \qquad \qquad \qquad + i_{n-1} * m_n + i_n) * l \\ &= a + \left(\sum_{j=1}^{n-1} i_j * \prod_{k=j+1}^n m_k + i_n \right) * l \end{aligned}$$



Array

- The array as an abstract data type
- **Sparse matrix**
- The string as an data type



The Sparse Matrix

**A general matrix consists of m rows and n columns
($m \times n$) of numbers, as:**

	0	1	2
0	-27	3	4
1	6	82	-2
2	109	-64	11
3	12	8	9
4	48	27	47



The Sparse Matrix

	0	1	2	3	4	5
0	15	0	0	22	0	-15
1	0	11	3	0	0	0
2	0	0	0	-6	0	0
3	0	0	0	0	0	0
4	91	0	0	0	0	0
5	0	0	28	0	0	0



The Sparse Matrix

A matrix of $m \times m$ is called a **square.**

A matrix with many zero entries is called **sparse.**



The Sparse Matrix

Representation:

- A natural way ---
 - $a[m][n]$
 - access element by $a[i][j]$, easy operations. **But** for sparse matrix, wasteful of both memory and time.
- Alternative way ---
 - store nonzero elements explicitly. 0 as default.



SparseMatrix

```
class SparseMatrix
```

```
{ // a set of <row, column, value>, where row, column are  
  // non-negative integers and form a unique combination;  
  // value is also an integer.
```

```
public:
```

```
    SparseMatrix ( int r, int c, int t);
```

```
    // creates a  $r \times c$  SparseMatrix with a capacity of t nonzero  
    // terms
```

```
    SparseMatrix Transpose ( );
```

```
    // return the SparseMatrix obtained by transposing *this
```

```
    SparseMatrix Add ( SparseMatrix b);
```

```
    SparseMatrix Multiply ( SparseMatrix b);
```

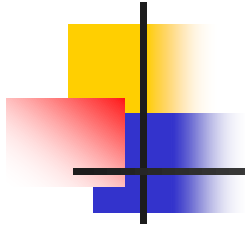
```
};
```



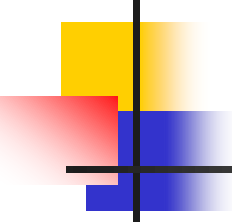
Sparse Matrix Representation

Use triple $\langle \text{row}, \text{col}, \text{value} \rangle$, sorted in ascending order by $\langle \text{row}, \text{col} \rangle$.

```
class SparseMatrix;  
class MatrixTerm {  
friend class SparseMatrix;  
Private:  
    int row, col, value;  
};
```



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

- 
-
- 设矩阵 A 中有 s 个非零元素。令 $e = s/(m*n)$, 称 e 为矩阵的稀疏因子, 例如 $e \leq 0.05$ 时



Transposing a Matrix

Transpose:

If an element is at position [i][j] in the original matrix, then it is at position [j][i] in the transposed matrix.

```
for(col=0;col<n;col++)  
    for(row=0;row<m;row++)  
        n[col][row]=m[row][col];  
T(n)=O(m×n)
```

用三元组表表示的稀疏矩阵及其转置

原矩阵三元组表

	行 (row)	列 (col)	值 (value)
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	-6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28

转置矩阵三元组表

	行 (row)	列 (col)	值 (value)
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	-6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16



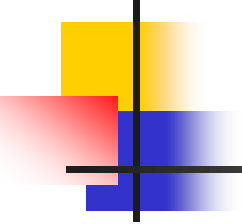
Main Idea

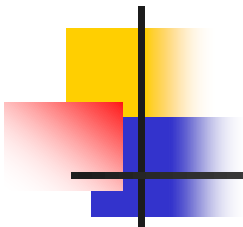
- 设矩阵列数为 **Cols**，对矩阵三元组表扫描**Cols** 次。
- 第 **k** 次检测列号为 **k** 的项。
- 第 **k** 次扫描找寻所有列号为 **k** 的项，将其行号变列号、列号变行号，顺次存于转置矩阵三元组表。



Fast Transposing

- 对原矩阵A 扫描一遍
- 按 A 中每一元素的列号
- 立即确定在转置矩阵 B 三元组表中的位置，并装入它

- 
- 为加速转置速度，建立辅助数组 **rowSize** 和 **rowStart**
 - ◆ **rowSize**记录矩阵转置前各列，即转置矩阵各行非零元素个数；
 - ◆ **rowStart**记录各行非零元素在转置三元组表中开始存放位置。
 - 扫描矩阵三元组表，根据某项列号，确定它转置后的行号，查 **rowStart** 表，按查到的位置直接将该项存入转置三元组表中。

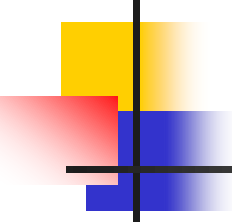


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	语 义
rowSize	1	1	1	2	0	2	1	矩阵 A 各列非零元素个数
rowStart	0	1	2	3	5	5	7	矩阵 B 各行开始存放位置
A三元组	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
行row	0	0	1	1	2	3	4	5
列col	3	6	1	5	3	5	0	2
值value	22	15	11	17	-6	39	91	28



Fast Transposing

```
template <class E>
void SparseMatrix<E>::
    FastTranspos (SparseMatrix<E>& B)
{
    int *rowSize = new int[Cols];    //列元素数数组
    int *rowStart = new int[Cols];  //转置位置数组
    B.Rows = Cols;  B.Cols = Rows;
    B.Terms = Terms;
    if (Terms > 0) {
        int i, j;
        for (i = 0; i < Cols; i++) rowSize[i] = 0;
```



```
for (i = 0; i < Terms; i++)
    rowSize[smArray[i].col]++;
rowStart[0] = 0;
for (i = 1; i < Cols; i++)
    rowStart[i] = rowStart[i-1]+rowSize[i-1];
for (i = 0; i < Terms; i++) {
    j = rowStart [smArray[i].col];
    B.smArray[j].row  = smArray[i].col;
    B.smArray[j].col  = smArray[i].row;
    B.smArray[j].value = smArray[i].value;
    rowStart [smArray[i].col]++;
} //
} //
delete [ ] rowSize;  delete [ ] rowStart;
} //FastTrans
```