

第 7 章

数组和矩阵

本章内容

- 7.1 数组
- 7.2 矩阵
- 7.3 特殊矩阵
- 7.4 稀疏矩阵

7.1 数组

- 7.1.1 抽象数据类型
- 7.1.2 C++数组的索引
- 7.1.3 行主映射和列主映射
- 7.1.4 用数组的数组来描述
- 7.1.5 行主描述和列主描述
- 7.1.6 不规则二维数组

7.1.1 抽象数据类型

抽象数据类型 Array {

实例

形如(index,value)的数对集合，其中任意两个数对的index值都各不相同.

操作

get(index): 返回索引为index的数对中的值

set(index, value): 加入一个新数对(index, value)，如果索引index值相同的数对已存在，则用新数对覆盖

7.1.2 C++数组的索引

- K维数组的索引(或下标):
 - $[i_1][i_2][i_3]\dots[i_k]$
- k维数组创建:
 - `int score[u1][u2][u3]\dots[uk].` (u_i ---正的常量或有常量表示的表达式)
 - $0 \leq i_j < u_j \quad 0 \leq j \leq k$
- 数组元素的个数:
 - $n = u_1 u_2 u_3 \dots u_k$
- 内存空间:
 - $n \times \text{sizeof}(\text{int})$ 字节.
- c++ 编译器为数组预留空间:
 - ***start* -----start + sizeof(score) -1**

7.1.3 行主映射和列主映射

- 为了实现与数组相关的函数set和get，必须确定索引值在 $[start, start+n*\text{sizeof}(\text{score})-1]$ 中的相应位置：
 - $[i_1][i_2]\dots[i_k] \rightarrow start + \text{map}(i_1, i_2, \dots, i_k) * \text{sizeof}(\text{int})$
- $\text{map}(i_1, i_2, \dots, i_k) : 0 \text{---} n-1$
 - 索引为 $[i_1][i_2]\dots[i_k]$ 的数组元素映射为在 $[start, start+\text{sizeof}(\text{score})-1]$ 中的第 $\text{map}(i_1, i_2, \dots, i_k)$ 个元素。
- 对一维数组：
 - $\text{map}(i_1) = i_1$
- 对二维数组
在 `int score[3][6]` 中，各索引的排列：

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

行主映射

- 行主映射
- `int s[u1][u2]`
 - `s[0][0] s[0][1]s[0][u2-1]`
 - `s[1][0] s[1][1]s[1][u2-1]`
 - `.....`
 - `s[u1-1][0] s[u1-1][1]s[u1-1][u2-1]`
- 映射函数: $\text{map}(i_1, i_2) = u_2 * i_1 + i_2$

列主映射

- 列主映射
 - `int s[u1][u2]`
 - `s[0][0] s[0][1]s[0][u2-1]`
 - `s[1][0] s[1][1]s[1][u2-1]`
 - `.....`
 - `s[u1-1][0] s[u1-1][1]s[u1-1][u2-1]`
-
- 映射函数: $\text{map}(i_1, i_2) = u_1 * i_2 + i_1$
 - 在 C++ 使用的是哪一种?
 - 行主映射!

行主映射和列主映射

- 对三维数组 ($\text{int score}[u_1][u_2][u_3]$)

- 索引按行主次序排列

- $\text{int score}[3][2][4]$:

[0][0][0],[0][0][1],[0][0][2],[0][0][3],[0][1][0],[0][1][1],[0][1][2],[0][1][3],
[1][0][0],[1][0][1],[1][0][2],[1][0][3],[1][1][0],[1][1][1],[1][1][2],[1][1][3],
[2][0][0],[2][0][1],[2][0][2],[2][0][3],[2][1][0],[2][1][1],[2][1][2],[2][1][3]

- ◆ 首先列出所有第一个坐标为0的索引，然后是第一个坐标为1的索引，
...
 - ◆ 第一个坐标相同的所有索引按其第二个坐标的递增次序排列，前两个
坐标相同的所有索引按其第三个坐标的递增次序排列。

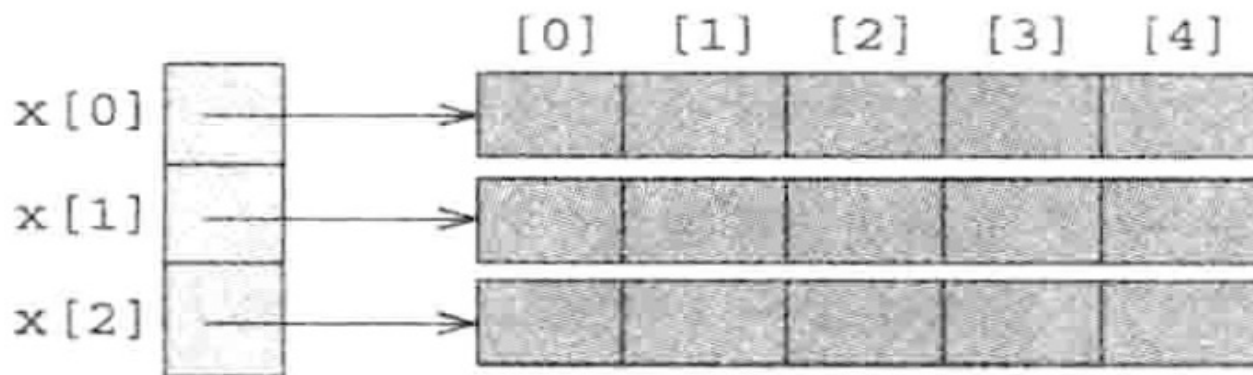
映射函数:

- $\text{map}(i_1, i_2, i_3) = i_1 u_2 u_3 + i_2 u_3 + i_3$

- 列主映射，映射函数？

7.1.4 多维数组—用数组的数组来描述

- 多维数组可以用数组的数组描述
- 如二维数组 `int x[3][5]` 的表示



- 占用空间： $4 \times 3 + 4 \times 5 \times 3 = 72$
- C++定位`x[i][j]`的过程：
 - 利用一维数组的映射函数找到指针`x[i]`
 - 利用一维数组的映射函数找到指针`x[i]`所指的行中索引为`j`的元素

7.1.5 多维数组— 行主描述和列主描述

- 创建一个一维数组(`int y[15]`), 按行主映射或列主映射, 将多维数组的元素映射到这个一维数组中。
- 如二维数组 `int x[3][5]`的行主描述
- 占用空间: $3 \times 5 \times 4 = 60$
- 定位`x[i][j]`的过程:
 - 利用二维数组的映射函数($\text{map}(i_1, i_2) = u_2 * i_1 + i_2$)计算`x[i][j]`在一维数组`y`中的位置`u`,
 - 利用一维数组的映射函数($\text{map}(i_1) = i_1$)访问元素`y[u]`

7.1.6 不规则二维数组

- 不规则二维数组：每一行的元素个数不相等

.....

```
int numberOfRows=5;
```

```
int length[5]={6,3,4,2,7};//每一行的长度
```

```
int** irregularArray=new int* [numberOfRows];
```

```
//分配每一行的空间
```

```
for (int i = 0; i ≤ numberOfRows; i++)
```

```
    irregularArray[i]=new int [length[i)];
```

```
//使用不规则二维数组
```

```
irregularArray[2][3]=5;
```

```
irregularArray[4][6]=irregularArray[2][3]+2;
```

.....

7.2 矩阵

- 7.2.1 定义和操作
- 7.2.2 类 `matrix`

7.2.1 定义和操作

- $m \times n$ 矩阵: m 行和 n 列的表.
- $M(i,j)$: 矩阵 M 中第 i 行、第 j 列 $1 \leq i \leq m$, $1 \leq j \leq n$ 的元素.
- 常用矩阵操作
 - 转置
 - 矩阵加
 - 矩阵乘

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

矩阵操作

■ 转置

$$M^T(i,j)=M(j,i), 1\leq i\leq n, 1\leq j\leq m$$

■ 矩阵相加

- $A, B : m \times n$ 矩阵

- $C=A+B$

$$C(i,j) = A(i,j) + B(i,j), 1\leq i\leq m, 1\leq j\leq n$$

■ 矩阵相乘

- $A : m \times n$ 矩阵; $B : n \times q$ 矩阵.

- $C=A*B : m \times q$ 矩阵

$$C(i, j) = \sum_{k=1}^n A(i, k) * B(k, j) \quad 1 \leq i \leq m, 1 \leq j \leq q$$

使用二维数组描述矩阵

- `int M[rows][cols];`
 - `M(i,j): M[i-1][j-1]`
 - P56 程序2-19 矩阵转置
 - P60 程序2-21 矩阵加法
 - P60 程序2-22 两个 $n*n$ 矩阵相乘
 - P60 程序2-23 一个 $m*n$ 矩阵与一个 $n*p$ 矩阵相乘
- `int M[rows+1][cols+1];`
 - 数组的0行和0列不用
 - `M(i,j): M[i][j]`

7.2.2 类matrix

- 类matrix用一个一维数组element, 按行主次序存储
- $\mathbf{M} : m \times n$ matrix
 - $\text{map}(i,j)=(i-1)*n + j - 1$

matrix类声明

```
template<class T>
class matrix {
    friend ostream& operator<<(ostream& ,const matrix<T>& );
public:
    matrix(int theRows = 0, int theColumns = 0);
    matrix(const matrix<T>& ); //复制构造函数
    ~matrix() {delete [] element;}
    int rows() const {return theRows;}
    int columns() const {return theColumns;}
    T& operator ( ) (int i, int j) const;
    matrix<T>& operator = (const matrix<T>& );
    matrix<T> operator + ( ) const; // 一元加法
    matrix<T> operator + (const matrix<T>& ) const;
    matrix<T> operator - ( ) const; // 一元减法
    matrix<T> operator - (const matrix<T>& ) const;
    matrix<T> operator * (const matrix<T>& ) const;
    matrix<T>& operator += (const T& x);
private:
    int theRows, theColumns; // 矩阵行数和列数
    T *element; // 元素数组
};
```

matrix 类的构造函数

```
template<class T>
matrix<T>::matrix(int theRows, int theColumns)
{ // 矩阵构造函数
  // 检验行数和列数的有效性
  if (theRows < 0 || theColumns < 0) throw
                                     illegalParameterValue("...");
  if ((theRows == 0 || theColumns == 0)
      && (theRows != 0 || theColumns != 0)) throw
                                     illegalParameterValue(".....");

  //创建矩阵
  this-> theRows = theRows;
  this-> theColumns = theColumns;
  element = new T [theRows* theColumns];
}
```

matrix 类的复制构造函数

```
template<class T>
matrix<T>::matrix(const matrix<T>& m)
{ // 矩阵的复制构造函数
  // 创建矩阵
  theRows = m.rows;
  theColumns = m.theColumns;
  element = new T [theRows* theColumns];

  // 复制m的每一个元素
  copy (m.element,
        m.element + theRows* theColumns,
        element);
}
```

matrix 类对 ‘=’ 的重载

```
template<class T>
matrix<T>& matrix<T> operator = (const matrix<T>& m)
{ // 重载赋值操作符=, *this=m
    if (this != &m)
    { // 不是自我赋值
        delete [] element; // 释放原空间
        theRows = m.rows;
        theColumns = m.theColumns;
        element = new T [theRows* theColumns];
        //复制m的每一个元素
        copy (m.element,
              m.element+ theRows* theColumns,
              element);
    }
    return *this;
}
```

matrix 类对 ‘ () ’ 的重载

```
template<class T>
```

```
    T& matrix<T>::operator ( ) (int i, int j) const
```

```
    { // 返回一个指向元素(i,j)的引用
```

```
    if (i < 1 || i > theRows || j < 1 || j > theColumns)
```

```
        throw matrixIndexOutOfBounds();
```

```
    return element[(i-1)*theColumns +j-1];
```

```
    }
```

matrix 类对 ‘+’ 的重载

```
template<class T>
matrix<T> matrix<T>::operator + (const matrix<T>& m) const
{
    //返回矩阵w = (*this) + m.
    if (theRows != m.theRows || theColumns != m.theColumns)
        throw matrixSizeMismatch( );

    //生成结果矩阵w
    matrix<T> w(theRows, theColumns);
    for (int i = 0; i < theRows* theColumns; i++)
        w.element[i] = element[i] + m.element[i];
    return w;
}
```

matrix 类对 ‘*’ 的重载—1/2

```
template<class T>
matrix<T> matrix<T>::operator * (const matrix<T>& m) const
{ // 矩阵乘法，返回结果矩阵 w = (*this)*m.
  if (theColumns != m.theRows)
    throw matrixSizeMismatch();

  matrix<T> w(theRows, m.theColumns); // 结果矩阵

  // 为 *this, m 和 w 定义游标，并设定初始位置为 (1,1)
  int ct=0, cm=0, cw=0;
```



```
// 对所有的i和j计算w(i,j)
for (int i=1; i<=theRows; i++)
{
    // 计算出结果矩阵的第i行
    for (int j=1; j<=m.theColumns; j++)
    {
        // 计算w(i,j)的第一项
        T sum=element[ct]*m.element[cm];
        for (int k=2; k<=theColumns; k++) // 累加其余项
        {
            ct++; // 指向* this第i行的下一项
            cm+=m.theColumns; // 指向m的第j列的下一项
            sum+=element[ct]*m.element[cm];
        }
        w.element[cw++]=sum; // 保存w(i,j)
        ct-=theColumns-1; // 第i行行首
        cm=j; // 第j+1列起始
    }
    ct+=theColumns; // 重新调整至下一行的行首
    cm=0; // 重新调整至第一列起始
}
return w;
}
```

7.3 特殊矩阵

- 7.3.1 定义和应用
- 7.3.2 对角矩阵
- 7.3.3 三对角矩阵
- 7.3.4 三角矩阵
- 7.3.5 对称矩阵

7.3.1 定义和应用

- 方阵(square matrix)是指具有相同行数和列数的矩阵.
- 常用方阵:
 - 对角矩阵 (**diagonal**) :
 - 当且仅当 $i \neq j$ 时, 有 $M(i,j) = 0$
 - 三对角矩阵(**tridiagonal**):
 - 当且仅当 $|i-j| > 1$ 时, 有 $M(i,j) = 0$
 - 下三角矩阵(**lower triangular**):
 - 当且仅当 $i < j$ 时, 有 $M(i,j) = 0$
 - 上三角矩阵(**upper triangular**):
 - 当且仅当 $i > j$ 时, 有 $M(i,j) = 0$
 - 对称矩阵(**symmetric**):
 - 当且仅当对于所有的 i 和 j , 有 $M(i,j) = M(j,i)$

特殊矩阵

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

a) 对角矩阵

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{bmatrix}$$

b) 三对角矩阵

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{bmatrix}$$

c) 下三角矩阵

$$\begin{bmatrix} 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 8 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

d) 上三角矩阵

$$\begin{bmatrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{bmatrix}$$

e) 对称矩阵

特殊矩阵

- $n \times n$ 特殊矩阵描述
 - 使用二维数组
 - 使用矩阵
 - 需要 $n^2 \times \text{sizeof}(T)$ 字节空间
- 压缩存储

7.3.2 对角矩阵

2	0	0	0
0	1	0	0
0	0	4	0
0	0	0	6

- 矩阵描述
 - 使用1维数组element,
 - 矩阵 $D \rightarrow T \text{ element}[n]$
 - $\text{element}=[2, 1, 4, 6]$

类diagonalmatrix声明

```
template<class T>
class diagonalMatrix {
public:
    diagonalMatrix(int theN = 10); // 构造函数
    ~diagonalMatrix() {delete [] element;} // 析构函数
    T get(int , int ) const;
    void set(int, int , const T&);
private:
    int n; //矩阵维数
    T *element; // 存储对角元素的一维数组
};
```

diagonalMatrix::get

```
template <class T>
T diagonalMatrix<T>::get(int i, int j) const
{
    //返回矩阵中(i,j)位置上的元素 .
    //检验i和j是否有效
    if (i < 1 || j < 1 || i > n || j > n) throw matrixIndexOutOfBounds();

    if (i == j)
        return element[i-1]; //对角线上的元素

    else return 0;    //非对角线上的元素
}
```


diagonalMatrix::set

```
template<class T>
void diagonalMatrix<T>::set(int i, int j, const T& newValue)
{ // 存储矩阵中位置(i,j)上元素的新值.
    // 检验i和j是否有效
    if (i < 1 || j < 1 || i > n || j > n) throw matrixIndexOutOfBounds();

    if (i==j)
        element[i-1]=newValue; // 存储对角元素的值
    else // 非对角线上的元素必须是0
        if (newValue!=0) throw illegalParameterValue(".....");
}
```

7.3.3 三对角矩阵

2	1	0	0
3	1	3	0
0	5	2	7
0	0	9	0

- 三条对角线：
 - 主对角线： $i=j$
 - 低对角线（主对角线之下的对角线）： $i=j+1$
 - 高对角线（主对角线之上的对角线）： $i=j-1$
- 三条对角线上的 $3n-2$ 个元素存储到一维数组：
 $element[3n-2]$
- 映射方式
 - 按行映射： $element=[2,1,3,1,3,5,2,7,9,0]$
 - 按列映射： $element=[2,3,1,1,5,3,2,9,7,0]$
 - 按对角线映射（从最下面的对角线开始）：
 $element=[3,5,9,2,1,2,0,1,3,7]$

三对角矩阵

■ 按对角线映射

$$\text{map}(i,j)=\begin{cases} i-2 & i=j+1 \\ n+i-2 & i=j \\ 2*n+i-2 & i=j-1 \end{cases}$$

tridiagonalMatrix类中的get

```
template <class T>
T tridiagonalMatrix<T>::get(int i, int j) const
{
    //返回矩阵中(i,j)位置上的元素 .
    //检验i和j是否有效
    if (i < 1 || j < 1 || i > n || j > n) throw matrixIndexOutOfBounds();
    //确定要返回的元素
    switch (i-j)
    {
        case 1:           //低对角线
            return element[i-2];
        case 0:           // 主对角线
            return element[n+i-2];
        case -1:          // 高对角线
            return element[2*n+i-2];
        default: return 0;
    }
}
```

7.3.4 三角矩阵



下三角矩阵



上三角矩阵

- 非0区域的元素总数:

- $1+2+\dots+n = \sum_{i=1}^n i = n(n+1)/2$

三角矩阵

- 矩阵描述

- 使用1维数组element, 矩阵 $T \rightarrow \text{element}[n(n+1)/2]$

2	0	0	0
5	1	0	0
0	3	1	0
4	2	7	0

- 映射

- 按行映射

$\text{elemrnt} = [2, 5, 1, 0, 3, 1, 4, 2, 7, 0]$

- 按列映射

$\text{elemrnt} = [2, 5, 0, 4, 1, 3, 2, 1, 7, 0]$

下三角矩阵

2	0	0	0
5	1	0	0
0	3	1	0
4	2	7	0

- 按行映射，下三角矩阵 L 映射到数组element

$$L(i,j) = 0 \quad i < j$$

$$L(i,j) = \text{element}[1+2+\dots+(i-1)+(j-1)] \quad i \geq j$$

$$= \text{element}[i(i-1)/2 + j-1]$$

$$\text{map}(i,j) = i(i-1)/2 + j-1 \quad i \geq j$$

下三角矩阵

2	0	0	0
5	1	0	0
0	3	1	0
4	2	7	0

- 按列映射，下三角矩阵 L 映射到数组element
- 前 $j-1$ 列元素个数： $n, n-1, \dots, n-j+2$
- 第 j 列之前元素： $i-j$

$$\text{map}(i,j) = (j-1)(2n-j+2)/2 + i-j \quad i \geq j$$

比如： $\text{map}(4,2)=6$

上三角矩阵

2	1	3	0
0	1	3	8
0	0	1	6
0	0	0	0

- 按列映射

$$U(i,j) = ? \quad i > j$$

$$U(i,j) = ? \quad i \leq j$$

- 按行映射 ?

7.3.5 对称矩阵

- 矩阵描述：
 - 存储矩阵的上三角或下三角

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

7.4 稀疏矩阵

- 7.4.1 基本概念
- 7.4.2 用单个线性表描述(数组存储)
- 7.4.3 用多个线性表描述(链式存储)

7.4.1 基本概念

- 如果一个 $m \times n$ 矩阵中有“许多”元素为0，则称该矩阵为稀疏矩阵（sparse）。
- 不是稀疏的矩阵被称为稠密矩阵（dense）。
- 在稀疏矩阵和稠密矩阵之间并没有一个精确的界限。
 - 我们规定若一个矩阵是稀疏矩阵，则其非0元素的数目应小于 $n^2/3$ ，在有些情况下应小于 $n^2/5$ ，
 - 对角矩阵,三对角矩阵（ $n \times n$ ）：稀疏矩阵
 - 三角矩阵：稠密矩阵

7.4.2 用单个线性表描述

- 可以按行主次序把无规则稀疏矩阵映射到一个线性表中

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

◆4×8矩阵

terms	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

线性表描述

- terms中的元素类型**matrixTerm<T>**，包括成员：
 - int row:矩阵元素的行
 - int col:矩阵元素的列
 - T value:矩阵元素的值

稀疏矩阵映射到arrayList

- terms采取数组描述 \Rightarrow terms是arraylist的实例
- arrayList添加方法:
 - reSet(newSize): 把表的元素个数改为newSize, 必要时增大数组容量
 - set(theIndex, theElement): 使元素theElement成为表中索引为theIndex的元素
 - clear(): 使表的元素个数为0

类sparseMatrix声明

```
template<class T>
class sparseMatrix {
public:
    void transpose(SparseMatrix<T> &b) ;
    void add(sparseMatrix<T> &b, sparseMatrix<T> &c) ;
private:
    int rows, ;    // 矩阵行数
        cols;    // 矩阵列数
    arrayList<matrixTerm<T>> terms; // 矩阵非0元素表
};
```

类sparseMatrix中的矩阵转置

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

terms	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

■ 转置后:

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

terms	0	1	2	3	4	5	6	7	8
row	2	2	3	4	4	5	6	7	8
col	2	4	4	1	3	2	3	1	2
value	6	4	5	2	9	7	8	1	3

sparseMatrix::transpose— 1/3

```
template<class T>
void sparseMatrix<T>::transpose(sparseMatrix<T> &b)
{ // 把*this的转置结果送入b

    // 设置转置矩阵特征
    b.cols=rows;
    b.rows=cols;
    b.terms.reSet(terms.size());

    // 初始化以实现转置
    int *colSize, *rowNext;
    colSize=new int[cols+1]; //输入矩阵在第i列的非0元素个数
    rowNext=new int[cols+1]; // 转置矩阵第i行首个非0元素在b中的索引
```

sparseMatrix::transpose— 2/3

```
// 计算*this每一列的非0元素数
for (int i=1; i<=cols; i++) //初始化
    colSize[i]=0;
for (arrayList<matrixTerm<T> > ::iterator i = terms.begin( );
     i != terms.end( ); i++)
    colSize[(*i).col]++;

// 求出b中每一行的起始点
rowNext[1]=0;
for (int i=2; i<=cols; i++)
    rowNext[i]= rowNext[i-1]+colSize[i-1];
```

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

terms	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

sparseMatrix::transpose — 3/3

// 实施从*this到b的转置复制

```
matrixTerm<T> mTerm;
```

```
for (arrayList<matrixTerm<T> > ::iterator i = terms.begin( );  
      i != terms.end( ); i++)
```

```
{
```

```
    int j=rowNext[(*i).col]++; // 在b中的位置
```

```
    mTerm.row=(*i).col;
```

```
    mTerm.col=(*i).row;
```

```
    mTerm.value=(*i).value;
```

```
    b.terms.set(j, mTerm);
```

```
}
```

```
}
```

0 0 0 2 0 0 1 0

0 6 0 0 7 0 0 3

0 0 0 9 0 8 0 0

0 4 5 0 0 0 0 0

terms	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

两个稀疏矩阵相加示例

0 0 0 0
0 6 0 4
0 0 0 5
2 0 9 0

0 7 0 0
0 0 8 0
1 0 0 2
0 3 0 0

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

terms	0	1	2	3	4
row	2	2	3	4	4
col	2	4	4	1	3
value	6	4	5	2	9

terms	0	1	2	3	4
row	1	2	3	3	4
col	2	3	1	4	2
value	7	8	1	2	3

terms	0	1	2	3	4	5	6	7	8
row	1	2	2	2	3	3	4	4	4
col	2	2	3	4	1	4	1	2	3
value	7	6	8	4	1	7	2	3	9

sparseMatrix::Add

```
template<class T>
void sparseMatrix<T>::Add(sparseMatrix<T> &b, sparseMatrix<T> &c)
{
    //计算c = (*this)+b.
    //检验相容性
    if (rows != b.rows || cols != b.cols)
        throw matrixSizeMismatch(); //不能相加
    //设置结果矩阵c的特征
    c.rows = rows;
    c.cols = cols;
    c.terms.clear( ) = 0; //初值
    int csize=0;
    //定义*this 和b的迭代器
    arrayList<matrixTerm<T> > ::iterator it=terms.begin( );
    arrayList<matrixTerm<T> > ::iterator ib=b.terms.begin( );
    arrayList<matrixTerm<T> > ::iterator itEnd=terms.end( );
    arrayList<matrixTerm<T> > ::iterator ibEnd=b.terms.end( );
}
```

//遍历*this 和b ,把相关的元素值相加

while (it != itEnd && ib != ibEnd)

{//每一个元素的行主索引+列数

int tIndex = (*it).row*cols+(*it).col;

int bIndex = (*ib).row*cols+(*ib).col;

if (tIndex < bIndex) //b 的元素在后

{c.terms.insert(cSize++,*it);

it++;} // *this的下一个元素

else {if (tIndex == bIndex) //位置相同

{//仅当和不为0时才添加到c中

if ((*it).value+(*ib).value != 0)

{matrixTerm<T> mTerm;

mTerm.row = (*it).row;

mTerm.col = (*it).col;

mTerm.value = (*it).value+(*ib).value;

c.terms.insert(cSize++, mTerm); }

it++; ib++;} // *this 和b的下一个元素

else {c.terms.insert(cSize++,*ib); ib++;} //b的下一个元素

}

terms	0	1	2	3	4
row	2	2	3	4	4
col	2	4	4	1	3
value	6	4	5	2	9

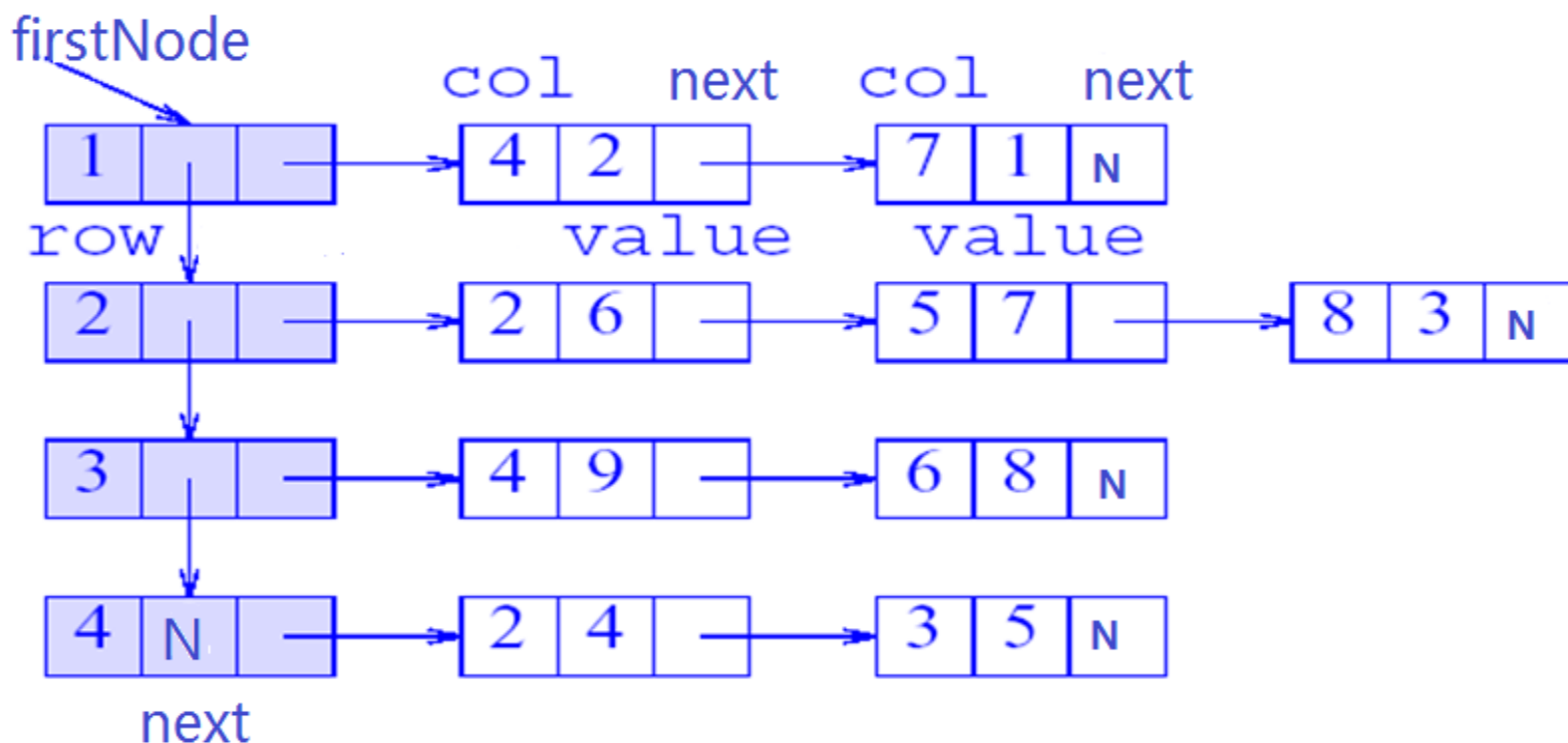
terms	0	1	2	3	4
row	1	2	3	3	4
col	2	3	1	4	2
value	7	8	1	2	3

terms	0	1	2	3	4	5	6	7	8
row	1	2	2	2	3	3	4	4	4
col	2	2	3	4	1	4	1	2	3
value	7	6	8	4	1	7	2	3	9

```
//复制剩余元素
for (; it != itEnd; it++)
    c.terms.insert(cSize++,*it) ;
for (; ib != ibEnd; ib++)
    c.terms.insert(cSize++,*ib) ;
}
```

7.4.3 用多个线性表描述

■ 稀疏矩阵的链式存储



作业

- P163 32
- P173 41