作业 5 参考答案（by 况鸿翔）

P163-32

```
//下三角矩阵的映射公式
int lowerTriangularMatrix :: map(int x, int y) {
    if (x < 1 || x > rows || y < 1 || y > cols)
        throw matrixIndexOutOfBounds();
    if (x < y)
        return 0;
    return x * (x - 1) / 2 + y - 1;
}

//上三角矩阵的映射公式
int upperTriangularMatrix :: map(int x, int y) {
    if (x < 1 || x > rows || y < 1 || y > cols)
        throw matrixIndexOutOfBounds();
    if (x > y)
        return 0;
    return (x - 1) * (2 * cols - x + 2) / 2 + y - x;
}

//下三角矩阵乘上三角矩阵的计算
template<class T>
T** matrixMultiply(lowerTriangularMatrix A, upperTriangularMatrix B) {
    if (A.cols != B.rows)
        throw matrixSizeMismatch();
    T** ans = new T*[A.rows];
    for (int i = 0; i < A.rows; i++)
        ans[i] = new int[B.cols];
    for (int i = 0; i < A.rows; i++)
        for (int j = 0; j < B.cols; j++)
        {
            int sum = 0;
            for (int k = 0; k < A.cols; k++)
                sum += A.element[A.map(i+1, k+1)] * B.element[B.map(k+1, j+1)];
            ans[i][j] = sum;
        }
    return ans;
}

//时间复杂度为 O(A.rows * A.cols * B.cols)
```

P173-41

```
//获取目标元素
template<class T>
T& sparseMatrix :: get(int theRow, int theCol) {
    if (theRow < 1 || theRow > rows || theCol < 1 || theCol > cols)
        throw matrixIndexOutOfBounds();
    int num = terms.size();
    for (int i = 0; i < num; )
    {
        if (terms[i].row > theRow)                    //目标位置不在 terms 中
            break;
        if (terms[i].row < theRow)                    //还未查询至第 theRow 行
            i++;
        else                                          //查询到第 theRow 行
        {
            while(1)
            {
                if (terms[i].col > theCol)            //目标位置不在 terms 中
                    break;
                if (terms[i].col < theCol)            //还未查询至第 theCol 列
                    i++;
                else                                  //查找到目标位置
                    return terms[i].value;
            }
            break;
        }
    }
    return noElement;                                 //noElement 为 T 类型的 0 元素
}
//时间复杂度为 O(terms.listSize)
//考虑二分查找可将时间复杂度降至 O(log2(terms.listSize))
//实现二分查找时可对三元组的小于号进行运算符重载以方便比较其前后位置

//设置目标元素
template<class T>
void sparseMatrix :: set(int theRow, int theCol, T theValue) {
    if (theRow < 1 || theRow > rows || theCol < 1 || theCol > cols)
        throw matrixIndexOutOfBounds();
    int num = terms.size();
    int index;                                    //插入的新三元组在 terms 中的下标
    for (int i = 0; i < num; )
    {
        if (terms[i].row > theRow)                //目标位置在当前位置和上一个位置之间
```

```
            {
                index = i;
                break;
            }
            if (terms[i].row < theRow)              //还未查询至第 theRow 行
                i++;
            else                                    //查询到第 theRow 行
            {
                while(1)
                {
                    if (terms[i].col > theCol)      //目标位置在当前位置和上一个位置之间
                    {
                        index = i;
                        break;
                    }
                    if (terms[i].col < theCol)      //还未查询至第 theCol 列
                        i++;
                    else                            //目标位置已有非零元素，无需移动后续元素
                    {
                        terms[i].value = theValue;
                        return;
                    }
                }
                break;
            }
        }
        terms.reSet(num + 1);                       //更改 terms 的大小
        for (int i = num; i > index; i--)           //将下标为 index～num-1 的元素向后移动
            set(i, terms[i - 1]);
        set(index, matrixTerm<T>(theRow, theCol, theValue));
    }
    //时间复杂度为 O(terms.listSize)
    //考虑二分查找可将时间复杂度降至 O(log2(terms.listSize))
    //实现二分查找时可对三元组的小于号进行运算符重载以方便比较其前后位置
```