32.下上三角矩阵乘法

按照题目要求，结果用二维数组存储，使用 matrix 类存储的代码版本见文末附录。
上下三角矩阵的实现亦见附录。

```
const int** operator * ( LowerTriangularMatrix <int> & l, LowerTriangularMatrix<int> & r ) {
    /*首先判断能否相乘*/
    if ( l.getN () != r.getN () ) {
        throw not_match;
    }
    int size = l.getN ();

    /*建立结果数组*/
    int** result = new int* [size];
    for ( int i = 0; i < size; i++ ) {
        result[i] = new int[size];
    }

    /*乘积第 m 行第 n 列的元素等于左矩阵的第 m 行元素与右矩阵的第 n 列对应元素乘积之和*/
    for ( int m = 1; m <= size; m++ ) {
        for ( int n = 1; n <= size; n++ ) {
            int value = 0;
            for ( int i = 1; i <= size; i++ ) {
                value += l.get ( m, i ) * r.get ( i, n );
            }
            result[m][n] = value;
        }
    }
    return result;
}
```

复杂度：
O(n^3)
结果矩阵与三角矩阵的赋值取值操作时间复杂度均为 O(1)
矩阵乘法本身需要进行三层嵌套循环，即进行 n^3 次元素加法操作
故总体复杂度为 O(n^3)

34.C 形矩阵

1)给出一个 4*4C 形矩阵及其压缩表示方式

```
1 1 1 1
1 0 0 0
1 0 0 0
1 1 1 1
```

压缩表示：
使用 3n-2 长度的一维数组：1 1 1 1 1 1 1 1 1 1
当 i==1 L(i,j)=element[j-1]
当 i==n L(i,j)=element[2*n-2+j]
当 j==1 L(i,j)=element[n+i-1]
其他情况，L(i,j)=0

2）证明最多有 3n-2 个非 0 元素
第一行，最后一行，第一列都无 0 的情况下非零元素最多，此时两行一列有 3n 个元素，减去行列重合处的两个元素，即非零元素最多有 3n-2 个。

35.反对角矩阵

1)给出一个 4*4 反对角矩阵的例子

```
0 0 0 1
0 0 1 0
0 1 0 0
1 0 0 0
```

2）证明最多有 n 个非 0 元素
由 “下标符合 i+j != n+1 的元素值为 0” 可知，所有不位于反对角线上的元素都为 0
又因为反对角线有 n 个元素，当这些元素都不为 0 时，非零元素最多，有 n 个。

3）用 n 个元素的一维矩阵表示反对角矩阵
一维数组表示反对角线，由于一行只有一个元素，因此只用行数作为数组的索引-1 就可以。
当 i+j!=n+1　　L(i,j)=0
当 i+j==n+1　　L(i,j)=element[i-1]

36.等对角矩阵

1)证明 n*n 等对角矩阵最多有 2n-1 个不同元素
由题意得，除第一行与第一列外，每个元素都与其左上方元素相等
也即当第一行与第一列元素各不相同时，其数目就是该题所指的最大不同元素数，为 2n-1

2）设计映射，将等对角矩阵映射到 2n-1 长度数组。
当 i<=j　　L(i,j)=element[j-i]
当 i>j　　L(i,j)=element[n+i-j-1]

4)等对角矩阵乘法
按照题目要求，结果用二维数组存储，使用 matrix 类存储的代码版本见文末附录。

```
/*等对角矩阵乘法*/
const int** operator * ( EquidiagonalMatrix <int>& l, EquidiagonalMatrix<int>& r ) {
    /*首先判断能否相乘*/
    if ( l.getN () != r.getN () ) {
        throw not_match;
    }
    int size = l.getN ();

    /*建立结果数组*/
    int** result= new int * [size];
    for ( int i = 0; i < size; i++ ) {
        result[i] = new int[size];
    }

    /*乘积第 m 行第 n 列的元素等于左矩阵的第 m 行元素与右矩阵的第 n 列对应元素乘积之和*/
    for ( int m = 1; m <= size; m++ ) {
        for ( int n = 1; n <= size; n++ ) {
            int value = 0;
            for ( int i = 1; i <= size; i++ ) {
                value += l.get ( m, i ) * r.get ( i, n );
            }
            result[m][n]=value;
        }
    }
    return result;
}
```

两个等对角矩阵相乘，其结果不一定是等对角矩阵，所以结果矩阵应用普通矩阵存储
复杂度同 32 题，为 O(n^3)

41.稀疏矩阵的赋值与取值

使用数组线性表的 find 函数需要为 node 编写==运算符重载，此题选择手动判断相等而不采用 find 函数

get 方法：(由于零值判断是类型相关的，编写适合所有数据类型的零值判断函数可能需要用到 type_traits，较为复杂，所以此处默认所有类型 T 均为数值类型，其零值为数字零)

```
/*获取值*/
T get(int r, int c)const {
    /*如果矩阵没有非 0 元素，则直接返回 0*/
    if (terms.getlength() == 0) {
        return 0;
    }

    /*寻找元素并返回值*/
    for (int i = 0; i < terms.getlength(); i++) {
```

```
        if (terms[i].col == c && terms[i].row == r) {
            return terms[i].num;
        }
    }

    /*如果找不到元素，则直接返回 0*/
    return 0;
}
```

set 方法：（对于零值元素，有直接返回和精细处理两种方法；直接返回降低赋值复杂度，增加取值复杂度，精细处理反之，此处展示直接返回，精细处理见附录）

```
/*设置某位置元素的值*/
void set(int r, int c, int value) {
    /*检查下标是否越界*/
    _checkrc(r, c);
    if ( value == 0 ) {//如果要赋值 0，则不进行操作，直接返回
        return;

    //如果矩阵还没有非 0 元素
    if (terms.getlength() == 0) {
            node temp;
            temp.col = c;
            temp.row = r;
            temp.num = value;
            terms.push(temp);
            return;

    /*如果矩阵已经有非 0 元素，则在 terms 中找到相应行、列的元素并修改*/
    }else {
        for (int i = 0; i < terms.getlength(); i++) {
            if (terms[i].col == c && terms[i].row == r) {
                terms[i].num = value;
                return;
            }
        }

        /*如果数组中找不到该元素*/
        node temp;

        temp.col = c;
        temp.row = r;
        temp.num = value;
        terms.push(temp);
        return;
        }
    }
}
```

每次赋值或取值最坏都要遍历整个数组进行检索，检索复杂度为 O(n)；
除 insert 外，其他操作复杂度均为 O(1)；
insert 函数与寻找元素的代码的复杂度呈现互补状态，寻找元素执行耗时，则 insert 不耗时；寻找元素不耗时，则 insert 耗时，所以此处复杂度仍然为 O(1)。
故两个函数复杂度均为 O(n)，n 代表 terms 当前长度。

第 8 章 9.1

```
typedef enum { pointer_is_null, newLength_less_than_zero } arrayStack_err;

/*改变容量的函数*/
void changeLength ( T*& p, int oldLength, int newLength ) {
    /*错误检查*/
    //新长度小于 0 则报错
    if ( newLength < 0 ) {
        throw newLength_less_than_zero;
    }
    //指针为空则报错
    if ( p == NULL || p == nullptr ) {
        throw pointer_is_null;
    }

    /*进行复制*/
    T* temp = new T[newLength];
    memcpy ( temp, p, min ( oldLength, newLength ) * sizeof ( T ) );
    delete[] p;
    p = temp;
```

```
}

void pop () {
    if ( stackTop == -1 )
        throw stackEmpty ();

    stack[stackTop--].~T ();

    /*增加的用于缩小 stack 空间的代码*/
    if ( size () < arrayLength / 4 ) {
        changeLength ( stack, arrayLength, arrayLength/2 );
    }
}
```

附录：

上下三角矩阵乘法 matrix 版：

```
/*矩阵乘法*/
const Matrix<int> operator * (  LowerTriangularMatrix<int>& l,  UpperTriangularMatrix<int>& r ) {
    /*首先判断能否相乘*/
    if ( l.getN () != r.getN () ) {
        throw not_match;
    }
    int size = l.getN ();
    /*建立临时数组，拥有左值的行数与右值的列数*/
    Matrix<int> temp ( size );
    /*乘积第 m 行第 n 列的元素等于左矩阵的第 m 行元素与右矩阵的第 n 列对应元素乘积之和*/
    for ( int m = 1; m <= size; m++ ) {
        for ( int n = 1; n <= size; n++ ) {
            int value = 0;
            for ( int i = 1; i <= size; i++ ) {
                value += l.get ( m, i ) * r.get ( i, n );
            }
            temp.set ( m, n, value );
        }
    }
    return temp;
}
```

等对角矩阵乘法 matrix 版：

```
/*对角矩阵乘法*/
const Matrix<int> operator * ( EquidiagonalMatrix <int>& l, EquidiagonalMatrix<int>& r ) {
    /*首先判断能否相乘*/
    if ( l.getN () != r.getN () ) {
        throw not_match;
    }
    int size = l.getN ();

    /*建立临时数组*/
    Matrix<int> temp ( size );

    /*乘积第 m 行第 n 列的元素等于左矩阵的第 m 行元素与右矩阵的第 n 列对应元素乘积之和*/
    for ( int m = 1; m <= size; m++ ) {
        for ( int n = 1; n <= size; n++ ) {
            int value = 0;
            for ( int i = 1; i <= size; i++ ) {
                value += l.get ( m, i ) * r.get ( i, n );
            }
            temp.set ( m, n, value );
        }
    }
    return temp;
}
```

上下三角矩阵的实现：

```
template<class T>
class LowerTriangularMatrix {
private:
    T* arr;
    int n;
    bool _checkIndex ( int i, int j ) {
        cout << i << " " << j << endl;
        if ( i < 1 || j < 1 || i > n||j>n) {
            throw index_err;
        }
```

```
    }

public:
    typedef enum err{ index_err }err;
    explicit LowerTriangularMatrix (int in) {
        n = in;
        arr = new T[n* ( n + 1 ) / 2];
    }
    ~LowerTriangularMatrix () {
        delete[] arr;
    }
    T get ( int i, int j ) {
        _checkIndex ( i, j );
        if ( i < j ) {
            return  0;
        }
        return arr[i *( i - 1 ) / 2 + j - 1];
    }
    void set ( int i, int j,T item ) {
        _checkIndex (i,j);
        if ( i < j ) {
            return;
        }
        cout << i * ( i - 1 ) / 2 + j - 1 << endl;
        arr[i* ( i - 1 ) / 2 + j - 1]=item;
    }
    int getN ()const { return n; }
};
template<class T>
class UpperTriangularMatrix {
private:
    T* arr;
    int n;
    bool _checkIndex ( int i, int j ) {
        cout << i << " " << j << endl;

        if ( i < 1 || j < 1 || i > n || j>n ) {
            throw index_err;
        }
    }

public:
    typedef enum err { index_err }err;
    explicit UpperTriangularMatrix ( int in ) {
        n = in;
        arr = new T[n* ( n + 1 ) / 2];
    }
    ~UpperTriangularMatrix () {
        delete[] arr;
    }
    T get ( int i, int j ) {
        _checkIndex ( i, j );
        if ( i > j ) {
            return 0;
        }
        cout << n * ( n + 1 ) / 2 - ( n - i + 2 ) * ( n - i + 1 ) / 2 + ( j - i );
        return arr[n * ( n + 1 ) / 2 - ( n - i + 2 )*( n - i + 1 ) / 2 + ( j - i )];
    }
    void set ( int i, int j, T item ) {
        _checkIndex ( i, j );
        if ( i > j ) {
            return;
        }
        cout << n * ( n + 1 ) / 2 - ( n - i + 2 ) * ( n - i + 1 ) / 2 + ( j - i ) << endl;

        arr[n * ( n + 1 ) / 2 - ( n - i + 2 )*( n - i + 1 ) / 2 + ( j - i )] = item;
    }

    int getN ()const { return n; }
};
```

等对角矩阵的实现：

```
template<class T>
class EquidiagonalMatrix {
private:
    T* arr;
    int n;
    bool _checkIndex ( int i, int j ) {
        cout << i << " " << j << endl;
        if ( i < 1 || j < 1 || i > n || j>n ) {
            throw index_err;
        }
    }
```

```
public:
    typedef enum err { index_err }err;
    explicit EquidiagonalMatrix ( int in ) {
        n = in;
        arr = new T[2 * n - 1];
    }
    ~EquidiagonalMatrix () {
        delete[] arr;
    }
    T get ( int i, int j ) {
        _checkIndex ( i, j );
        if ( i <= j ) {
            return arr[j - i];
        }
        if ( i > j ) {
            return arr[n + i - j - 1];
        }
    }
    void set ( int i, int j, T item ) {
        _checkIndex ( i, j );
        if ( i <= j ) {
            arr[j - i] = item;
        }
        if ( i > j ) {
            arr[n + i - j - 1] = item;
        }
    }

    int getN ()const { return n; }
};
```

matrix 实现：

```
template<class T>
class Matrix {
private:
    T** arr;
    int n;
    bool _checkIndex ( int i, int j )const {
        cout << i << " " << j << endl;
        if ( i < 1 || j < 1 || i > n || j>n ) {
            throw index_err;
        }
    }

public:
    typedef enum err { index_err }err;
    explicit Matrix ( int in ) {
        n = in;
        arr = new T * [in];
        for ( int i = 0; i < in; i++ ) {
            arr[i] = new T[in];
        }
    }
    Matrix ( Matrix& raw ) {
        n = raw.getN ();
        arr = new T * [n];
        for ( int i = 0; i < n; i++ ) {
            arr[i] = new T[n];
        }
        for ( int r = 0; r < n; r++ ) {
            for ( int c = 0; c < n; c++ ) {
                set ( r + 1, c + 1, raw.get ( r + 1, c + 1 ) );
            }
        }
    }
    ~Matrix () {
        for ( int i = 0; i < n; i++ ) {
            delete[] arr[i];
        }
        delete[] arr;
    }
    const T get ( int i, int j ) const {
        _checkIndex ( i, j );
        return arr[i - 1][j - 1];
    }
    void set ( int i, int j, T item ) {
        _checkIndex ( i, j );
        arr[i - 1][j - 1] = item;
    }
    int getN ()const { return n; }
    /*输出矩阵，传入输出流的引用*/
    void show ( ostream& out )const {
        for ( int r = 0; r < n; r++ ) {
            for ( int c = 0; c < n; c++ ) {
```

```
                out << arr[r][c] << " ";
            }
            out << endl;
        }
        out << endl;
    }
    /*= 运算符重载，该函数与复制构造函数同功能*/
    const Matrix<int>& operator = ( const Matrix<int>& raw ) {

        for ( int i = 0; i < n; i++ ) {
            delete[] arr[i];
        }
        delete[] arr;
        n = raw.getN ();
        arr = new T * [n];
        for ( int i = 0; i < n; i++ ) {
            arr[i] = new T[n];
        }
        for ( int r = 0; r < n; r++ ) {
            for ( int c = 0; c < n; c++ ) {
                set ( r + 1, c + 1, raw.get ( r + 1, c + 1 ) );
            }
        }
        return *this;
    }
};
```

矩阵赋值精细处理：

```
/*设置某位置元素的值*/
void set(int r, int c, int value) {
    /*检查下标是否越界*/
    _checkrc(r, c);

    if (terms.getlength() == 0) {//如果矩阵还没有非 0 元素
        if (value == 0) {//如果要赋值 0，则不进行操作，直接返回
            return;
        }else {//如果赋的数值不为 0，则建立新结点并加入到 terms 中
            node temp;

            temp.col = c;
            temp.row = r;
            temp.num = value;
            terms.push(temp);
            return;
        }

    /*如果矩阵已经有非 0 元素，则在 terms 中找到相应行、列的元素并修改*/
    }else {
        for (int i = 0; i < terms.getlength(); i++) {
            if (terms[i].col == c && terms[i].row == r) {
                /*如果元素要赋的值为 0，则从 vector 中删除该元素*/
                if (value == 0) {
                    terms.del(i);

                /*如果元素要赋的值不为 0，则赋值*/
                }else {
                    terms[i].num = value;
                }
                return;
            }
        }

        /*如果数组中找不到该元素*/
        if (value == 0) {//如果元素要赋的值为 0，则直接返回
            return;
        }else {//如果元素要赋的值不为 0，则直接加入该新元素
            node temp;

            temp.col = c;
            temp.row = r;
            temp.num = value;
            terms.push(temp);
            return;
        }
    }
}
```