CS205 C++ project

group member: Chen Tianle, Li Qifei and Chen Pengzhen

title about the project: Matrix

Basic structure

```
template<class T>
class matrix {};
```

We build the matrix in a class to make it an object (since Object-Oriented Design and Analysis is a more widely used, advanced design concept).

To build the basic information, we should have some member variables, for a matrix, we use a normal way to build: two-dimensional array; also, we should first define the number of row and column of the matrix. So m and n are used(only when we know the row and column of matrix, we can put value in it).

It is generally believed that member variables should not be able to be directly modified externally, so the rows, columns, and corresponding values of the matrix should be private. This is more in line with the code design specifications.

the block below show the way we declare the member variables:

```
private:
    int n, m; // number of row and column
    std::vector<std::vector<std::pair<int, T>>> R, C; // record the matrix by R
and C
```

After building the members of the matrix, we should make some constructors.

Firstly, we make a constructor with no variable. Since we give the user more free to build it on their own.

Then, a more useful constructor is build with the parameters _n and _m. Using this constructor, you can give a space n * m for your matrix and the size of the matrix is declared(Don't worry for changing the size of the matrix, we give you a function named resize).

The following api is the function body:

```
matrix() {}

matrix(int _n, int _m) {
    assert(_n > 0 && _m > 0);
    n = _n, m = _m;
    R.resize(n);
    C.resize(m);
}
```

Also, we should noticed that the matrix may be used in many ways such as int, double, and even complex.

To make our code more adaptable, and to make it work with different types of data, we use template class.

Then when we put different types of the data, it can change it for our need automatically.

```
template<class T>
```

And for a good design, there must be many useful api that can reduce the work of coder(such as the resize function we mentioned before). We design this function:

```
void resize(int _n, int _m);//change the size of matrix
void O();//make a zero matrix with size m * n(you declare before)
void I();//make a identity matrix with size m * n
void sort();//sort the elements in the matrix
std::vector<std::vector<T>> getAll() const;//get the all information of the
matrix
void print();//print all elements of the matrix
```

some design requirements

Of course, we should implement some functionality to complement our class to make it easier to use (otherwise it will behave like a normal 2D array). And now we can refer to the requirement in the document that given by SA.

1

```
It supports all matrix sizes, from small fixed-size matrices to arbitrarily large
dense matrices, and even sparse matrices.
```

of course we do this successfully! Not only can we declare the matrix with the size we want, but also we can resize it.

```
matrix(int _n, int _m)//declare in constructor
void resize(int _n, int _m)//resize the matrix
```

The sparse matrix is maintained by `vector<vector<pair<int, int>>>`, using something similar to an adjacency list for each **row** to store data, so the upper limit of the amount of data stored is $O(N+K)$, where N is the number of rows and K is the number of elements of the sparse matrix, which ensures that the sparse matrix can be maintained efficiently. Wherein, in order to facilitate the operation of matrix multiplication, the **columns** are similarly maintained as the above process.

insert data

as described above

```
void insert(int x, int y, T w) {
    R[x].push_back({y, w});
    C[y].push_back({x, w});
}
```

multiply

calculate

$$\begin{pmatrix} a_{0,0} & \dots & a_{0,m-1} \\ a_{1,0} & \dots & a_{1,m-1} \\ \vdots & & \\ a_{n-1,0} & \dots & a_{n-1,m-1} \end{pmatrix} \begin{pmatrix} b_{0,0} & \dots & b_{0,k-1} \\ b_{1,0} & \dots & b_{1,k-1} \\ \vdots & & \\ b_{m-1,0} & \dots & b_{n-1,k-1} \end{pmatrix}$$

```
matrix<T> operator*(const matrix<T> &o) const
```

add

Similar to matrix multiplication.

```
matrix<T> operator+(const matrix<T> &o) const
```

2

```
It supports all standard numeric types, including std::complex, integers, and is
easily extensible to custom numeric types.
```

As we are saying before, we use template to achieve it. Our matrix can support the int, double and complex as the data type and so on.

Related **data type** application.

> It's shown in the **test code.**

`int`

```
if(0){
        bool ok=true;

        int T=10; // number of test cases
        while(T--){
            int N=rand()%50+1;
            Matrix<int> A(N, N), B(N, N);
            Mat A_(N, N), B_(N, N);

            for(int i=0; i<N; i++) for(int j=0; j<N; j++){
                int val=rand()%100+1;
                A.insert(i, j, val);
                A_.w[i+1][j+1]=val;
            }
            for(int i=0; i<N; i++) for(int j=0; j<N; j++){
                int val=rand()%100+1;
                B.insert(i, j, val);
                B_.w[i+1][j+1]=val;
            }
            A=A*B;
            A_=A_*B_;
            auto A_get=A.getAll();
            for(int i=0; i<N; i++) for(int j=0; j<N; j++) ok&=(A_get[i]
[j]==A_.w[i+1][j+1]);
        }
        puts(ok? "OK": "NG");
    }
```

`complex<double>`

```
// test: std::complex
    if(0){
        const int N=3;
        Matrix<complex<double>> A(N, N);
        A.insert(0, 0, {1.0, 1.0});
        A.insert(1, 2, {2.0, 3.0});
        puts("A:");
```

```
        A.print();

        Matrix<complex<double>> B(N, N);
        B.insert(0, 0, {1.0, 1.0});
        B.insert(1, 1, {2.0, 2.0});
        B.insert(2, 2, {3.0, 3.0});
        puts("B:");
        B.print();

        A=A*B;
        puts("result of A * B");
        A.print();
    }
```

3

```
It supports matrix and vector arithmetic, including addition, subtraction, scalar multiplication, scalar division, transposition, conjugation, element-wise multiplication, matrix-matrix multiplication, matrix-vector multiplication, dot product and cross product
```

This is very important for the class we design.

To make it computable, we choose to overloaded operator instead of implement many functions(more elegant). In this way matrix can calculate naturally.

```
matrix<T> operator*(const matrix<T> &o) const;
vector<T> operator*(const vector<T> &o) const;
matrix<T> operator+(const matrix<T> &o) const;
matrix<T> operator-(const matrix &o) const;
matrix<T> operator*(const T &k) const;
matrix<T> operator/(const T &k) const;
matrix<T> dot(matrix &o);
```

4

```
It supports basic arithmetic reduction operations, including finding the maximum value, finding the minimum value, summing all items, calculating the average value (all supporting axis-specific and all items).
```

This question is normal and easy to design, we show the code directly:

```
T get_max();
T get_min();
T get_sum();
T get_average();
```

> Notice: do not forget to use the template to satisfy all data types.

5

```
It supports computing eigenvalues and eigenvectors, calculating traces, computing
inverse and computing determinant.
```

This is really a big job!!!

First, focus on the eigenvalues and eigenvector, since we can't do the calculate by code in the same way as we usually do in out brains, it makes this question difficult!

At first, by scanning from the web, we try to use QR disintegration to finish the job:

The practical QR algorithm [edit]

Formally, let A be a real matrix of which we want to compute the eigenvalues, and let A_0:=A. At the k-th step (starting with $k = 0$), we compute the QR decomposition A_k=Q_kR_k where Q_k is an orthogonal matrix (i.e., $Q^T = Q^{-1}$) and R_k is an upper triangular matrix. We then form $A_{k+1} = R_kQ_k$. Note that

$$A_{k+1} = R_kQ_k = Q_k^{-1}Q_kR_kQ_k = Q_k^{-1}A_kQ_k = Q_k^TA_kQ_k,$$

so all the A_k are similar and hence they have the same eigenvalues. The algorithm is numerically stable because it proceeds by *orthogonal* similarity transforms.

1.Householder变换进行QR分解

反射矩阵：任取单位向量w,反射矩阵H=E-2WW^T,显然HH^T=E，H是正交阵

定理：任取两个模长相等的的向量x,y，一定存在一个反射矩阵H，使得Hx=y，
此时w=(x-y)/(|x-y|)（向量的差除以向量差的模）

应用：现在我们取矩阵的一列为x,m=|x|,y=m*[1,0,0,...0]^T 根据上面的定理求出H，使得Hx=y,是不是通过正交变化就把那一列化成了[m,0,0,0]^T,这样就达到了将下三角元素全化为0的效果。看下图，举个例子来说明QR分解过程：

```
void QR(Matrix A, Matrix &Q, Matrix &R);
Matrix Quasi_upper_tri(Matrix A);
void QR(Matrix A, Matrix &Q, Matrix &R);
void Calc(double a, double b, double c, double d);
void Gauss(Matrix A, double r);
void LoopQR(Matrix &A, double *r);
```

But we failed. After thinking many times, we think that for the matrix not symmetry, we can only estimate the value, so there may be problems such as insufficient precision. Also, we want to use inverse to find the eigenvector, but there isn't an inverse of the matrix with rank < n.

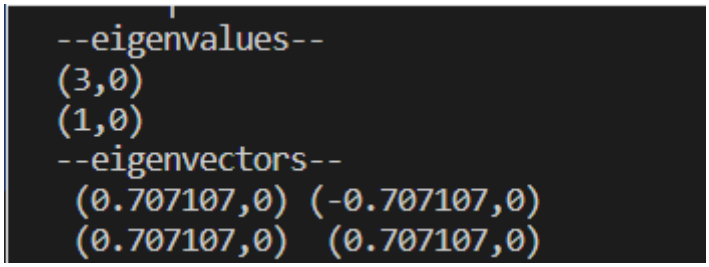
Finally we found an extension package for cpp: eigen. It only need to download, and we can put the information in it to calculate the eigenvalues and eigenvectors.

```
//head include
#include "Eigen/Dense"
#include "Eigen/Eigen"
add_executable(cpp_matrix main.cpp Matrix.h Eigen/Dense)

//code to use
Eigen::MatrixXf mat(n,n);
for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            mat(i,j) = a[i][j];
```

```
        }
}
Eigen::EigenSolver<Eigen::MatrixXf> es(mat);
std::cout << es.eigenvalues() << std::endl;
```

Then it's output is show as below:



And, we do same things and spend many times to finish calculating traces, computing inverse and computing determinant.

```
T get_trace();
matrix<T> get_inv();
T get_det();
```

6

```
It supports the operations of reshape and slicing.
```

reshape

Iterate all elements in the matrix and reinsert them into the result matrix.

```
    matrix<T> reshape(int row, int col) {
        assert(row * col == n * m);
        matrix<T> res(row, col);
        for (int i = 0; i < n; i++) {
            for (auto &[x, y]: R[i]) {
                int id = i * m + x;
                int fir = id / col, sec = id % col;
                res.insert(fir, sec, y);
            }
        }

        return res;
    }
```

slicing

Just sweep the matrix and insert the data whose index of row and column are both in the required range into the result matrix.

```
    matrix<T> slicing(pair<int, int> row, pair<int, int> col) {
        assert(row.x < n && row.y < n && col.x < m && col.y < m);
        if (row.x > row.y) std::swap(row.x, row.y);
        if (col.x > col.y) std::swap(col.x, col.y);
        matrix<T> res(row.y - row.x + 1, col.y - col.x + 1);

        for (int i = row.x; i <= row.y; i++) {
            for (auto &[x, y]: R[i])
                if (x >= col.x && x <= col.y) {
                    int fir = i - row.x, sec = x - col.x;
                    res.insert(fir, sec, y);
                }
        }
        return res;
    }
```

7

convolutional operations

By inputting the matrix to be processed `in1` and convolution kernel `in2`, and selecting one of the three modes of `full`, `same` and `vaild`, the matrix after convolution operation is returned (the `full` mode is returned by default when the mode is not selected).

```
    //卷积
    Matrix<T> conv(const Matrix<T> &in1, const Matrix<T> &in2, std::string mode
= "full"){

        Matrix<T> res(in1.n + in2.n-1, in1.m + in2.m-1);
            auto mat1 = in1.getAll();
            auto mat2 = in2.getAll();
            for(int i=0; i < in1.n+in2.n-1; i++)
                for(int j=0; j < in1.m+in2.m-1; j++)
                {
                    T temp = 0;
                    for(int m=0; m<in1.n; m++)
                        for(int n=0; n<in1.m; n++)
                            if((i-m) >= 0 && (i-m) < in2.n && (j-n) >= 0 && (j-
n) < in2.m )
                                temp += mat1[m][n]*mat2[i-m][j-n];
                    res.insert(i,j,temp);

                }

        if(mode == "full")
            return res;


        Matrix<T> res2(in1.n, in1.m);
            auto mat3 = res.getAll();
```

```
            for(int i=0; i < in1.n; i++)
                for(int j=0; j < in1.m; j++)
                {
                    res2.insert(i,j,mat3[i+floor((in2.n)*1.0/2)]
[j+floor((in2.m)*1.0/2)]);
                }


        if(mode == "same")
            return res2;

        Matrix<T> res3(in1.n-in2.n+1, in1.m-in2.m+1);
            auto mat4 = res2.getAll();
            for(int i=0; i < in1.n-in2.n+1; i++)
                for(int j=0; j < in1.m-in2.m+1; j++)
                {
                    res3.insert(i,j,mat4[i+(in2.n-1)/2][j + (in2.m-1)/2]);
                }


        if (mode == "vaild")
            return res3;

        return res;

    }
```

8

Transfer the matrix from OpenCV to the matrix of this library and vice versa.

In this part, what we need to do just copy the value from one to another one. The only difference is that the matrix in the project is assigned with `insert`, while opencv can directly assign values to `mat.at<T>(i,j)` perform assignment operations, such as `mat.at<T>(i,j) == 1;`.

```
    Matrix<T> cvToMat(cv::Mat& m1)
    {
        Matrix<T> res(m1.size().height,m1.size().width);
        T *c =  m1.ptr<T>(0);
        for (int i = 0; i < m1.size().height; i++)
        {
            for (int j = 0; j < m1.size().width; j++)
            {
                res.insert(i,j,*(c++));
            }

        }
        return res;
    }

    cv::Mat matToCv(Matrix<T> mat)
    {

        cv::Mat mat1 = cv::Mat_ <T>(mat.n, mat.m);
        auto mat2 = mat.getAll();
```

```
        for (int i = 0; i < mat.n; i++)
        {
            for (int j = 0; j < mat.m; j++)
        {
            mat1.at<T>(i,j) = mat2[i][j];
        }

        }
        return mat1;
    }
```

9

`It should process likely exceptions as much as possible.`

Including judge whether the size of matrix is valid and handle some other problem.

E.g `assert(n==m);` to guarantee the matrix is square, else the program will terminate.

The advance of the project

handling the sparse matrix

Applying the `two pointer` algorithm when multiplying.

It can effectively reduce the amount of operations for multiplication in **sparse matrices**.

```
matrix<T> operator*(const matrix<T> &o) const {
        assert(m == o.n);
        matrix<T> res(n, o.m);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < o.m; j++) {
                T val = 0;
                for (int k = 0, ko = 0; k < R[i].size(); k++) {
                    while (ko < o.C[j].size() && o.C[j][ko].x < R[i][k].x) ko++;
                    if (ko < o.C[j].size() && o.C[j][ko].x == R[i][k].x) val +=
o.C[j][ko].y * R[i][k].y;
                }
                res.insert(i, j, val);
            }
        }
        res.sort();
        return res;
    }
```

compare test:

```
// test: compare the cost of time in A * B and if it's correct
    if(0){
        const int N=1000;
        Matrix<int> A(N, N), B(N, N);
        Mat A_(N, N), B_(N, N);
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) if(rand()%100==0){
            int val=rand()%100+1;
            A.insert(i, j, val);
```

```
            A_.w[i+1][j+1]=val;
        }
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) if(rand()%100==0){
            int val=rand()%100+1;
            B.insert(i, j, val);
            B_.w[i+1][j+1]=val;
        }
        startclk=clock();
        A=A*B;
        endclk=clock();

        cout<<"cost "<<(endclk-startclk)<<" ms"<<endl;

        startclk=clock();
        A_=A_*B_;
        endclk=clock();

        cout<<"cost "<<(endclk-startclk)<<" ms"<<endl;

        bool match=true;
        auto A_get=A.getAll();
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) match&=(A_get[i]
[j]==A_.w[i+1][j+1]);
        puts(match? "YES": "NO");
    }
```

runtime compare:

```
718750 ms
7062500 ms
```

The testcase and output

Total test file:

```
#include<bits/stdc++.h>
#include "Matrix.h"
using namespace std;

#define rep(i,a,b) for(int i=(a);i<=(b);i++)

struct Mat{
    int n, m;
    vector<vector<int>> w;

    Mat(){}
    Mat(int _n, int _m){
        n=_n, m=_m;
        w.resize(n+1, vector<int>(m+1, 0));
    }
```

```
    void I(){
        assert(n==m);
        rep(i,1,n) rep(j,1,n) w[i][j]=(i==j);
    }

    Mat operator * (const Mat &o)const{
        assert(m==o.n);
        Mat res(n, o.m);

        rep(i,1,n) rep(j,1,o.m){
            int &t=res.w[i][j];
            rep(k,1,o.n) (t+=w[i][k]*o.w[k][j]);
        }

        return res;
    }

    Mat operator + (const Mat &o)const{
        assert(m==o.m && n==o.n);
        Mat res(n, m);

        rep(i,1,n) rep(j,1,m){
            res.w[i][j]=w[i][j]+o.w[i][j];
        }

        return res;
    }

    Mat fpow(int p){
        Mat res(n, m);
        Mat x=*this;
        res.I();
        for(; p; p>>=1, x=x*x) if(p&1) res=res*x;
        return res;
    }
};

int main(){
    srand(time(0));
    // freopen("output.out", "w", stdout);

    clock_t startclk, endclk;

    // test: compare the cost of time in A * B and if it's correct
    if(0){
        const int N=1000;
        Matrix<int> A(N, N), B(N, N);
        Mat A_(N, N), B_(N, N);
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) if(rand()%100==0){
            int val=rand()%100+1;
            A.insert(i, j, val);
            A_.w[i+1][j+1]=val;
        }
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) if(rand()%100==0){
            int val=rand()%100+1;
            B.insert(i, j, val);
            B_.w[i+1][j+1]=val;
```

```
        }
        startclk=clock();
        A=A*B;
        endclk=clock();

        cout<<"cost "<<(endclk-startclk)<<" ms"<<endl;

        startclk=clock();
        A_=A_*B_;
        endclk=clock();

        cout<<"cost "<<(endclk-startclk)<<" ms"<<endl;

        bool match=true;
        auto A_get=A.getAll();
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) match&=(A_get[i]
[j]==A_.w[i+1][j+1]);
        puts(match? "YES": "NO");
    }

    // test: if it's correct
    if(0){
        bool ok=true;

        int T=10; // number of test cases
        while(T--){
            int N=rand()%50+1;
            Matrix<int> A(N, N), B(N, N);
            Mat A_(N, N), B_(N, N);

            for(int i=0; i<N; i++) for(int j=0; j<N; j++){
                int val=rand()%100+1;
                A.insert(i, j, val);
                A_.w[i+1][j+1]=val;
            }
            for(int i=0; i<N; i++) for(int j=0; j<N; j++){
                int val=rand()%100+1;
                B.insert(i, j, val);
                B_.w[i+1][j+1]=val;
            }
            A=A*B;
            A_=A_*B_;
            auto A_get=A.getAll();
            for(int i=0; i<N; i++) for(int j=0; j<N; j++) ok&=(A_get[i]
[j]==A_.w[i+1][j+1]);
        }
        puts(ok? "OK": "NG");
    }

    // test: read the matrix by stdin (read tuple (row, column, value))
    if(0){
        Matrix<int> A;
        // input size of Matrix A
        try{
            puts("Enter rows and columns:");
            int n, m; cin>>n>>m;
            A.resize(n, m);
```

```
            // input k data
            puts("Enter number of datas:");
            int k; cin>>k;
            for(int i=1; i<=k; i++){
                cout<<"Entering the "<<i<<" th data: ";
                int x, y, z; cin>>x>>y>>z;
                if(x>=n || y>=m) throw "input for the matrix out-of-bounds!";
                A.insert(x, y, z);
            }

            A.print();
        }
        catch(const char *e){
            cout<<e<<endl;
        }
    }

    // test: std::complex
    if(0){
        const int N=3;
        Matrix<complex<double>> A(N, N);
        A.insert(0, 0, {1.0, 1.0});
        A.insert(1, 2, {2.0, 3.0});
        puts("A:");
        A.print();

        Matrix<complex<double>> B(N, N);
        B.insert(0, 0, {1.0, 1.0});
        B.insert(1, 1, {2.0, 2.0});
        B.insert(2, 2, {3.0, 3.0});
        puts("B:");
        B.print();

        A=A*B;
        puts("result of A * B");
        A.print();
    }

    // test: get max, min, sum
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        // 1 0 0
        // 0 4 2
        // 0 1 3

        cout<<"max: "<<A.get_max()<<endl;
        cout<<"min: "<<A.get_min()<<endl;
        cout<<"sum: "<<A.get_sum()<<endl;
    }

    // test: A + B
    if(0){
```

```
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0
        // 0 4 2
        // 0 1 3

        Matrix<int> B(N, N);
        B.insert(0, 1, 12);
        B.insert(1, 0, 1);
        B.insert(1, 1, 5);
        B.insert(2, 1, 6);
        B.sort();
        // 0 12 0
        // 1 5 0
        // 0 6 0

        A=A+B;
        A.print();
    }

    // test: compare the cost of time in A + B and if it's correct
    if(0){
        const int N=5000;
        Matrix<int> A(N, N), B(N, N);
        Mat A_(N, N), B_(N, N);
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) if(rand()%500==0){
            int val=rand()%100+1;
            A.insert(i, j, val);
            A_.w[i+1][j+1]=val;
        }
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) if(rand()%500==0){
            int val=rand()%100+1;
            B.insert(i, j, val);
            B_.w[i+1][j+1]=val;
        }
        startclk=clock();
        A=A+B;
        endclk=clock();

        cout<<"cost "<<(endclk-startclk)<<" ms"<<endl;

        startclk=clock();
        A_=A_+B_;
        endclk=clock();

        cout<<"cost "<<(endclk-startclk)<<" ms"<<endl;

        bool match=true;
        auto A_get=A.getAll();
        for(int i=0; i<N; i++) for(int j=0; j<N; j++) match&=(A_get[i]
[j]==A_.w[i+1][j+1]);
        puts(match? "YES": "NO");
```

```
    }

    // test: A - B
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0
        // 0 4 2
        // 0 1 3

        Matrix<int> B(N, N);
        B.insert(0, 1, 12);
        B.insert(1, 0, 1);
        B.insert(1, 1, 5);
        B.insert(2, 1, 6);
        B.sort();
        // 0 12 0
        // 1 5 0
        // 0 6 0

        A=A-B;
        A.print();
    }

    // test: A * k
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0
        // 0 4 2
        // 0 1 3

        A=A*10;
        A.print();
    }

    // test: A / k
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
```

```
        // 1 0 0
        // 0 4 2
        // 0 1 3

        A=A/2;
        A.print();
    }

    // test: dot(A, B)
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0
        // 0 4 2
        // 0 1 3

        Matrix<int> B(N, N);
        B.insert(0, 1, 12);
        B.insert(1, 0, 1);
        B.insert(1, 1, 5);
        B.insert(2, 1, 6);
        B.sort();
        // 0 12 0
        // 1 5 0
        // 0 6 0

        A=A.dot(B);
        A.print();
    }

    // test: transposition
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 0, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0
        // 2 4 0
        // 0 1 3

        Matrix<int> res=A.transposition();
        res.print();
    }

    // test: Matrix A * vector B
    if(0){
        const int N=3;
        Matrix<int> A(N, N);
```

```
        vector<int> B, res;
        A.insert(0, 0, 1);
        A.insert(1, 1, 2);
        A.insert(2, 2, 3);
        A.sort();
        // 1 0 0
        // 0 2 0
        // 0 0 3

        B={1, 2, 3};
        res=A*B;
        for(auto i: res) cout<<i<<' ';
        cout<<endl;

        A.O();
        A.insert(0, 0, 1);
        A.insert(1, 2, 1);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0
        // 0 0 1
        // 0 1 0

        B={1, 114, 514};
        res=A*B;
        for(auto i: res) cout<<i<<' ';
        cout<<endl;
    }

    // test: conjugation
    if(0){
        const int N=2;
        Matrix<complex<int>> A(N, N);
        A.insert(0, 0, {1, 1});
        A.insert(1, 0, {2, 3});
        A.insert(1, 1, {4, 7});
        A.sort();
        // (1,1) (0,0)
        // (2,3) (4,7)

        auto res=A.conjugation<int>();
        res.print();
    }

    // test reshape
    if(0){
        const int N=3, M=4;
        Matrix<int> A(N, M);
        A.insert(0, 0, 1);
        A.insert(0, 3, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0 1
        // 0 4 2 0
        // 0 1 3 0
```

```
        A=A.reshape(2, 6);
        A.print();
    }

    // test slicing
    if(0){
        const int N=3, M=4;
        Matrix<int> A(N, M);
        A.insert(0, 0, 1);
        A.insert(0, 3, 1);
        A.insert(1, 2, 2);
        A.insert(1, 1, 4);
        A.insert(2, 2, 3);
        A.insert(2, 1, 1);
        A.sort();
        // 1 0 0 1
        // 0 4 2 0
        // 0 1 3 0

        A=A.slicing({1, 1}, {2, 3});
        A.print();
    }

    // test inv
    if(0){
        const int N=3, M=3;
        Matrix<double> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 1, 2);
        A.insert(2, 2, 3);
        A.sort();
        // 1 0 0
        // 0 2 0
        // 0 0 3

        auto B=A.get_inv();
        B.print();

        puts("============================");

        A.O();
        A.insert(0, 0, 8), A.insert(0, 1, 1), A.insert(0, 2, 4);
        A.insert(1, 0, 4), A.insert(1, 1, 6), A.insert(1, 2, 5);
        A.insert(2, 0, 2), A.insert(2, 1, 1), A.insert(2, 2, 3);
        A.sort();
        // [8, 1, 4],
        // [4, 6, 5],
        // [2, 1, 3]

        // output of numpy:
        // [[ 0.18571429  0.01428571 -0.27142857]
        // [-0.02857143  0.22857143 -0.34285714]
        // [-0.11428571 -0.08571429  0.62857143]]

        B=A.get_inv();
        B.print();
```

```
        puts("=================");

        A.O();
        A.insert(0, 0, 8), A.insert(0, 1, 1), A.insert(0, 2, 4);
        A.insert(1, 0, 4), A.insert(1, 1, 6), A.insert(1, 2, 5);
        A.insert(2, 0, 2), A.insert(2, 1, 1), A.insert(2, 2, 3);
        A.sort();
        // -1 1
        // 1 -1

        B=A.get_inv();
        B.print();
    }

    // test det
    if(0){
        const int N=3, M=3;
        Matrix<double> A(N, N);
        A.insert(0, 0, 1);
        A.insert(1, 1, 2);
        A.insert(2, 2, 3);
        A.sort();
        // 1 0 0
        // 0 2 0
        // 0 0 3

        auto res=A.get_det();
        printf("%lf\n", res);

        A.O();
        A.insert(0, 0, 8), A.insert(0, 1, 1), A.insert(0, 2, 4);
        A.insert(1, 0, 4), A.insert(1, 1, 6), A.insert(1, 2, 5);
        A.insert(2, 0, 2), A.insert(2, 1, 1), A.insert(2, 2, 3);
        A.sort();
        // [8, 1, 4],
        // [4, 6, 5],
        // [2, 1, 3]

        // output of numpy:
        // 69.99999999999996

        res=A.get_det();
        printf("%lf\n", res);

        A.O();
        A.insert(0, 0, 32), A.insert(0, 1, 54), A.insert(0, 2, 4);
        A.insert(1, 0, 4), A.insert(1, 1, 6), A.insert(1, 2, 55);
        A.insert(2, 0, 2), A.insert(2, 1, 45), A.insert(2, 2, 3);
        A.sort();
        // [32, 54, 4],
        // [4, 6, 55],
        // [2, 45, 3]

        // output of numpy:
        // -72660.00000000001

        res=A.get_det();
        printf("%lf\n", res);
```

```
    }

    //test conv
    if(1){

            Matrix<int> A(5, 5);
            A.insert(0, 0, 17);A.insert(0, 1, 24);A.insert(0, 2, 1);A.insert(0,
3, 8);A.insert(0, 4, 15);
            A.insert(1, 0, 23);A.insert(1, 1, 5);A.insert(1, 2, 7);A.insert(1,
3, 14);A.insert(1, 4, 16);
            A.insert(2, 0, 4);A.insert(2, 1, 6);A.insert(2, 2, 13);A.insert(2,
3, 20);A.insert(2, 4, 22);
            A.insert(3, 0, 10);A.insert(3, 1, 12);A.insert(3, 2, 19);A.insert(3,
3, 21);A.insert(3, 4, 3);
            A.insert(4, 0, 11);A.insert(4, 1, 18);A.insert(4, 2, 25);A.insert(4,
3, 2);A.insert(4, 4, 9);
            A.sort();


            Matrix<int> B(3, 3);
            B.insert(0, 0, 1);B.insert(0, 1, 2);B.insert(0, 2, 1);
            B.insert(1, 0, 0);B.insert(1, 1, 2);B.insert(1, 2, 0);
            B.insert(2, 0, 3);B.insert(2, 1, 1);B.insert(2, 2, 3);
            B.sort();

            Matrix<int> D(2, 2);
            D.insert(0, 0, 1);D.insert(0, 1, 2);
            D.insert(1, 0, 0);D.insert(1, 1, 2);

            // D.print();

            Matrix<int> C(6, 6);
            C = A.conv(A,D,"full");
            C.print();
            cout<< "----------"<<endl;
            C = A.conv(A,D,"same");
            C.print();
            cout<< "----------"<<endl;
            C = A.conv(A,D,"vaild");
            C.print();


        }

    // test cvToMat
    if(0){

             cv::Mat m4 = (cv::Mat_ <int>(3, 3) << 1, 2, 3, 4, 4, 6, 7, 8, 9);
             Matrix<int> C(3,3);
             C = C.cvToMat(m4);
            //  C = A.conv(A,D,"full");
            //  C.print();
            //  cout<< "----------"<<endl;

            C.print();


        }
```

```
    // test matToCv
    if(0){

             Matrix<int> B(3, 3);
            B.insert(0, 0, 1);B.insert(0, 1, 2);B.insert(0, 2, 1);
            B.insert(1, 0, 0);B.insert(1, 1, 2);B.insert(1, 2, 0);
            B.insert(2, 0, 3);B.insert(2, 1, 1);B.insert(2, 2, 3);
            B.sort();
            cv::Mat mat1;
            mat1 = B.matToCv(B);
            std::cout<< mat1 <<std::endl;
        }



    // test
    if(0){
        set_n(3);
        // Matrix<double> A(2, 2);
        Matrix<double> A(3, 3);
        // A.insert(0, 0, 2), A.insert(0, 1, 1);
        // A.insert(1, 0, 1), A.insert(1, 1, 2);
        A.insert(0, 0, 8), A.insert(0, 1, 1), A.insert(0, 2, 4);
        A.insert(1, 0, 4), A.insert(1, 1, 6), A.insert(1, 2, 5);
        A.insert(2, 0, 2), A.insert(2, 1, 1), A.insert(2, 2, 3);

        // output of numpy:
        // [10.88362733  4.76722829  1.34914438]

        A.set_eigenValues();
        vector<double> v = A.get_eigenValues();
        for(auto i: v) cout<<i<<' ';
        cout<<endl;

        auto vec = A.get_eigenVector(v[1]);

        for(auto i: vec) cout<<i<<' ';
        cout<<endl;
    }

    return 0;
}
```

```
test: compare the cost of time in A * B and if it's correct
cost 718 ms
cost 7062 ms
YES

test: if it's correct
OK

test: std::complex
A:
(1,1) (0,0) (0,0)
```

```
(0,0) (0,0) (2,3)
(0,0) (0,0) (0,0)
B:
(1,1) (0,0) (0,0)
(0,0) (2,2) (0,0)
(0,0) (0,0) (3,3)
result of A * B
(0,2) (0,0) (0,0)
(0,0) (0,0) (-3,15)
(0,0) (0,0) (0,0)

test: get max, min, sum
max: 4
min: 0
sum: 11

test: A + B
1 12 0
1 9 2
0 7 3

test: compare the cost of time in A + B and if it's correct
cost 46 ms
cost 312 ms
YES

test: A - B
1 -12 0
-1 -1 2
0 -5 3

test: A * k
10 0 0
0 40 20
0 10 30

test: A / k
0 0 0
0 2 1
0 0 1

test: dot(A, B)
0 0 0
0 20 0
0 6 0

test: transposition
1 2 0
0 4 1
0 0 3

test: Matrix A * vector B
1 4 9
1 514 114

test: conjugation
(1,-1) (2,-3)
(0,0) (4,-7)
```

```
1 0 0 1 0 4
2 0 0 1 3 0

test: slicing
2 0

test: inv
1 0 0
0 0.5 0
0 0 0.333333
============================
0.185714 0.0142857 -0.271429
-0.0285714 0.228571 -0.342857
-0.114286 -0.0857143 0.628571
==================
0.185714 0.0142857 -0.271429
-0.0285714 0.228571 -0.342857
-0.114286 -0.0857143 0.628571

test: det
6.000000
70.000000
72660.000000

test: conv
CONV2(A,B)
-----full-----
17 58 66 34 32 38 15
23 85 88 35 67 76 16
55 149 117 163 159 135 67
79 78 160 161 187 129 51
23 82 153 199 205 108 75
30 68 135 168 91 84 9
33 65 126 85 104 15 27
-----same-----
85 88 35 67 76
149 117 163 159 135
78 160 161 187 129
82 153 199 205 108
68 135 168 91 84
-----vaild-----
117 163 159
160 161 187
153 199 205
-----full-----
CONV2(A,D)
17 58 49 10 31 30
23 85 65 30 60 62
4 60 35 60 90 76
10 40 55 85 85 50
11 60 85 90 55 24
0 22 36 50 4 18
-----same-----
85 65 30 60 62
60 35 60 90 76
40 55 85 85 50
60 85 90 55 24
```

```
22 36 50 4 18
-----vaild-----
85 65 30 60
60 35 60 90
40 55 85 85
60 85 90 55
----------

test test cvToMat
1 2 3
4 4 6
7 8 9

test matToCv
[1, 2, 1;
 0, 2, 0;
 3, 1, 3]

test eigenValue and eigenVector
--eigenvalues--
(3,0)
(1,0)
--eigenvectors--
 (0.707107,0) (-0.707107,0)
 (0.707107,0)  (0.707107,0)
```