Report of Homework 1

PB21010362 汪兆辰

March 11, 2023

1 The Instruction of the Program File

The Program includes a header file and a main program. In the header file "matrix.h", two objects, Matrix and SparseMatrix are defined. The main program contains some test datas.

1.1 Matrix

The object Matrix is stored as a linear array. Using the function "Map" to find the location of a given element, which receives a coordinate and returns an integer. For users to visit the value of certain coordinate, the operator brackets and square brackets are defined, but actually what they do is invoking the function "Map":

```
inline int Map(int r,int c) const {
    if(r>rows||r<=0||c<=0||c>cols) {
        throw out_of_range("Out of Range!");
    }
    return (r-1)*cols+c-1;
}
```

The class contains the overload of operators. Here we just use the addition of matrix as an example, as the algorithms of other operators are actully similar.

```
Matrix operator+ (const Matrix& rhs) {
    if(rhs.cols!=cols||rhs.rows!=rows){
        throw logic_error("Size mismatch");
    }
    Matrix Mt(rows,cols);
    for(int i=0;i<rows;i++){
        for(int j=0;j<cols;j++){
```

```
                Mt.data[Map(i+1,j+1)]=rhs.data[Map(i+1,j+1)]
                +data[Map(i+1,j+1)];
            }
        }
        return Mt;
    }
```

The matrix Mt is used for returning the result of addition. The value of each element in Mt is the sum of rhs and *this at the same location. After walking through the two martrixs, we get the sum of the two and return it in Mt.

1.2 SparseMatrix

The SparseMatrix is stored as a linked-list, which contains all the nonzero elements. The type of nodes is defined as Node, which includes the row, column and the value of the element. The head of the list is defined by the parameter (-1,-1,0). In the constructor, an array of Node* is create, and for every row of the matrix, the head node is created.

```
SparseMatrix(int row,int col):rows(row),cols(col){
    matrix=new Node*[row+1];
    for(int i=1;i<=rows;i++){
        matrix[i]=new Node(-1,-1,0);
    }
}
```

To change the value of element, the function "add-element" is defined. It finds the given place and insert the new node into the list.

```
void add_element(int r,int c,R v){
    if(v==0)return;

    Node* p=matrix[r];
    while(p->next&&p->next->Col<c){
        p=p->next;
    }
    if(p->next&&p->next->Col==c){
        p->next->Data=v;
    }
    else {
```

```
            auto new_node= new Node(r,c,v);
            new_node->next=p->next;
            p->next=new_node;
        }
    }
```

The algorithm of the addition is to merge the same location of the two matrixs. When one row of the matrix has been walked through, the rest of the other matrix are just put into the result matrix.

```
SparseMatrix operator+(const SparseMatrix& rhs){
    if(rows!=rhs.rows||cols!=rhs.cols){
        throw logic_error("Size mismatch");
    }
    SparseMatrix result(rows,cols);
    for(int r=1;r<=rows;r++){
        Node* p1=matrix[r]->next;
        Node* p2=rhs.matrix[r]->next;

        while(p1&&p2){//merge the element at the same location
            if(p1->Col < p2->Col){
                result.add_element(p1->Row,p1->Col,p1->Data);
                p1=p1->next;
            }
            else if(p1->Col > p2->Col){
                result.add_element(p2->Row,p2->Col,p2->Data);
                p2=p2->next;
            }
            else {
                result.add_element(p1->Row,p1->Col,p1->Data+p2->Data);
                p1=p1->next;
                p2=p2->next;
            }
        }
        //add the rest of the elements
        while(p1){
            result.add_element(p1->Row,p1->Col,p1->Data);
            p1=p1->next;
        }
```

```
            while(p2){
                result.add_element(p2->Row,p2->Col,p2->Data);
                p2=p2->next;
            }

        }
        return result;
    }
```

2 Test and Result

2.1 Fill Matrix with random numbers

Fill two matrixs with random numbers and add them up. The running results are as follows:

```
this matrix has size (3 x 5)
the entries are:
-7.3692         -0.82700        -5.6208         3.5773          8.6939

0.38833         -9.3086         0.59400         -9.8460         -8.6632

3.7355          8.6087          0.53858         3.0784          4.0238

this matrix has size (3 x 5)
the entries are:
5.2440          -9.0507         -3.4353         5.1282          -2.6932

9.6510          5.0671          -8.5463         7.6941          -1.2718

-0.44536        -4.5019         -6.6699         7.9531          -8.7887

this matrix has size (3 x 5)
the entries are:
-38.644         7.4849          19.309          18.345          -23.415

3.7478          -47.168         -5.0765         -75.757         11.018

-1.6636         -38.755         -3.5922         24.483          -35.364
```

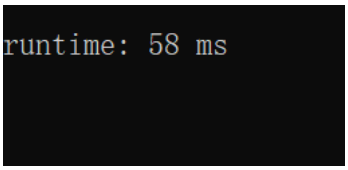
2.2 Benchmark with runtime

Using the lib chrono. Get the start time and end time of the program, by comparing the two time we get the duration of certain command.The codes are as follows:

```
auto start_time=chrono::system_clock::now();
Matrix<double> C = A*B;
auto end_time=chrono::system_clock::now();
```

```
auto duration = chrono::duration_cast<chrono::milliseconds>
(end_time - start_time);
double time_in_milliseconds = static_cast<double>(duration.count());
```

To make the duration long enough, we enlarge the scale of matrixs into (2000*2000), and use chrono to get the duration of the multiplication of two matrixs. The running results are as follows:



2.3 Compare with Eigen

Using the lib Eigen and self-defined class Matrix to assign and operate matrixs.The test codes are as follows:

```
// todo 8: use Eigen and compare

MatrixXd MtA(2,2),MtB(2,2);
MtA<<1,2,3,4;
MtB<<1,0,-1,1;

Matrix<double> selfA(2,2),selfB(2,2);
selfA(1,1)=1,selfA(1,2)=2,selfA(2,1)=3,selfA(2,2)=4;
selfB(1,1)=1,selfB(1,2)=0,selfB(2,1)=-1,selfB(2,2)=1;

cout<<"Using Eigen:"<<endl;
cout<<"the value of A:\n"<<MtA<<endl;
cout<<"the value of B:\n"<<MtB<<endl;
cout<<'\n'<<endl;
cout<<"Using Self-defined Matrix:"<<endl;
cout<<"the value of A:"<<endl;
selfA.print();
cout<<"the value of B:"<<endl;
selfB.print();
```

the running results are as follows:

```
Using Eigen:
the value of A:
1 2
3 4
the value of B:
 1  0
-1  1


Using Self-defined Matrix:
the value of A:
this matrix has size (2 x 2)
the entries are:
1.0000          2.0000

3.0000          4.0000

the value of B:
this matrix has size (2 x 2)
the entries are:
1.0000          0.0000

-1.0000         1.0000
```

2.4 Test of SparseMatrix

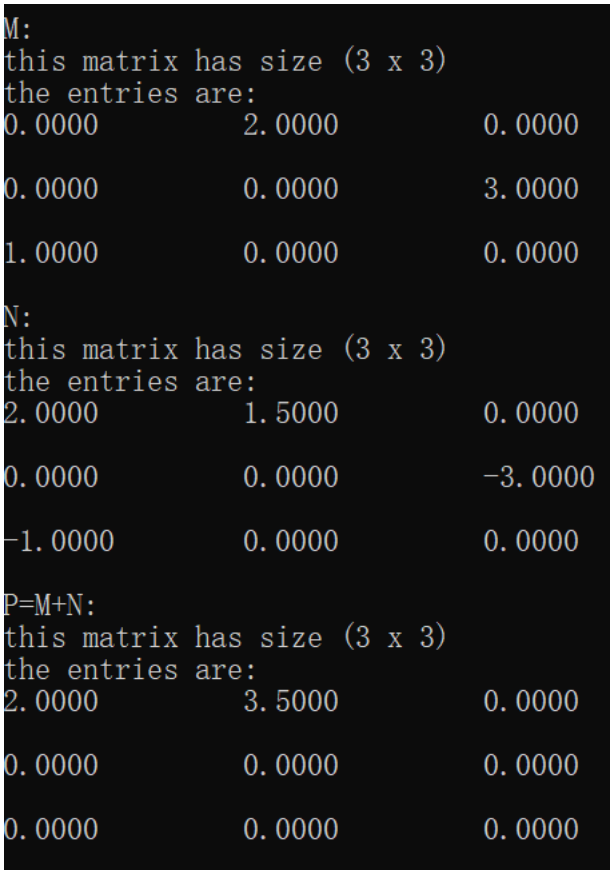
Test the assignment, addition and printing of sparse matrixs.Test code are as follows:

```cpp
//test of SparseMatrix

SparseMatrix<double> M(3,3);
M.add_element(1,2,2); M.add_element(2,3,3);
M.add_element(3,1,1);
SparseMatrix<double> N(3,3);
N.add_element(1,1,2); N.add_element(1,2,1.5);
N.add_element(2,3,-3); N.add_element(3,1,-1);

SparseMatrix<double> P=M+N;
cout<<"M:"<<endl; M.print();
cout<<"N:"<<endl; N.print();
cout<<"P=M+N:"<<endl; P.print();
```

the running results are as follows:



Reference

[1] Stephen Prata. C++ Primer Plus. Beijing: People Telecom Press, 2012.

[2] https://blog.csdn.net/qq_36336522/article/details/79410813

[3] https://blog.csdn.net/qq_45797026/article/details/108536921

[4] https://zhuanlan.zhihu.com/p/462494086