Report of Assignment 9

——SparseMatrix

Meng Wei

Results:

In our experiment, we implement the storage and operators (+ & *) of Sparse matrix with whose element is characterized by <row, col, value>. When we test our code, we use two sparse matrices m1 and m2 to test the operator + and the operator *. Sparse matrices m1 and m2 are as follows:

$$m1=\begin{bmatrix}0&0&0&0&0\\0&1&1&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}, m2=\begin{bmatrix}0&0&0&0&0\\0&1&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}.$$

When we use the operator +, we can get

$$m3=m1+m2=\begin{bmatrix}0&0&0&0&0\\0&2&1&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$, so our code for operator + is right.

For the operator *, we can get

$$m4=m1*m2=\begin{bmatrix}0&0&0&0&0\\0&2&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$, so our code for operator * is also right.

PS: Codes as follows:

Sparse Matrix:

```
//SparseMatrix.h
//
#include <iostream>
#include <vector>
using namespace std;

//triple struct
struct term {
    int col;//the column of nonzero elements
    int row;//the row of nonzero elements
    double value;//the value of nonzero elements
    term(int i, int j, double val)
    {
```

```
        row = i;
        col = j;
        value = val;
    }
};

//the declaration of Class SparseMatrix
class SparseMatrix
{
    friend SparseMatrix operator +(const SparseMatrix&m1, const
SparseMatrix&m2);//add two sparse matrix together
    friend SparseMatrix operator *(const SparseMatrix&m1, const
SparseMatrix&m2);//sparse matrix times sparse matrix
public:
    SparseMatrix(int s1, int s2); //default constructor
    SparseMatrix(const SparseMatrix &);//copy constructor
    SparseMatrix& operator=(const SparseMatrix &);//assignment operator
    vector<term> data;
    int nrow, ncol;    //the number of row and column of Sparse Matrix
    double get(int i, int j);//get the i-th row, j-th column element
    void set(int i, int j, double val);//change the i-th row, j-th column element
by value val
};

// SparseMatrix.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include <vector>
#include"SparseMatrix.h"
using namespace std;

//define default constructor
SparseMatrix::SparseMatrix(int s1, int s2)
{
    nrow = s1;
    ncol = s2;
}

//define copy constructor
SparseMatrix::SparseMatrix(const SparseMatrix & S)
{
    nrow = S.nrow;
```

```
    ncol = S.ncol;
    data = S.data;
}

//define assignment operator =
SparseMatrix& SparseMatrix::operator=(const SparseMatrix & S)
{
    data = S.data;
    nrow = S.nrow;
    ncol = S.ncol;
    return *this;
}

//define function get
double SparseMatrix::get(int i, int j)
{
    for (auto k : data)
    {
        if (k.row == i&&k.col == j)
            return k.value;
    }
    return 0;
}

//define function set
void SparseMatrix::set(int i, int j, double val)
{
    term temp(i, j, val);
    data.push_back(temp);
}


//define the friend function operator +
SparseMatrix operator +(const SparseMatrix& m1, const SparseMatrix& m2)
{
    SparseMatrix matadd(m1.nrow, m1.ncol);

    for (auto k1 : m1.data)
    {
        term add(k1.row, k1.col, k1.value);
        for (auto k2 : m2.data)
        {
            if (k2.row == k1.row&&k2.col == k1.col)
            {
```

```
                add.value += k2.value;
            }
        }
        matadd.data.push_back(add);
    }


    for (auto k2 : m2.data)
    {
        term add(k2.row, k2.col, k2.value);
        for (auto k1 : m1.data)
            if (k2.row == k1.row&&k2.col == k1.col) continue;
        matadd.data.push_back(add);
    }
    return matadd;
}

//define the friend function operator *
SparseMatrix operator *(const SparseMatrix& m1, const SparseMatrix& m2)
{
    SparseMatrix matproduct(m1.nrow, m2.ncol);

    int row = 0;
    int col = 0;
    double value = 0;
    term product(row, col, value);
    for (auto k1 : m1.data)
    {
        product.row = k1.row;
        for (auto k2 : m2.data)
        {

            if (k2.row == k1.col)  //the column of the first matrix m1 must be equal
to the row of the second matrix m2
            {
                product.col = k2.col;
                product.value += k1.value*k2.value;
            }
        }
    }
    matproduct.data.push_back(product);
    return matproduct;
}
```

```
int main()
{
    SparseMatrix m1(5, 5), m2(5, 5), m3(5, 5), m4(5, 5);
    m1.set(1, 1, 1);
    m1.set(1, 2, 1);
    m2.set(1, 1, 1);
    m2.set(2, 1, 1);

    //test operator +
    m3 = m1 + m2;
    for (int i = 0; i < m3.nrow; i++)
        for (int j = 0; j < m3.ncol; j++)
        {
            cout << m3.get(i, j) << endl;
        }
    cout << "----------------------------------------" << endl;

    //test operator *
    m4 = m1 * m2;
    for (int i = 0; i < m4.nrow; i++)
        for (int j = 0; j < m4.ncol; j++)
        {
            cout << m4.get(i, j) << endl;
        }
    return 0;
}
```