how U doin'?

$$🐳^{🐳^{🐳}} = \int_{🎃}^{🎅} 🐏 \, d🍀$$

Sparse Matrix*

10170437 Mark Taylor

June 24, 2020

Contents

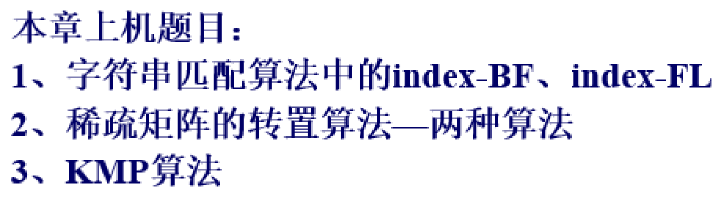
1 Problems **2**

2 Code **2**
2.1 SMatrix.h 2
2.2 SMatrix_test.cpp 7
2.3 mySort.h 8

3 Output **9**
3.1 Win10 9
3.2 Linux 10

4 Appendix **11**

1 Problems



本章上机题目：
1、字符串匹配算法中的index-BF、index-FL
2、稀疏矩阵的转置算法—两种算法
3、KMP算法

2 Code

2.1 SMatrix.h

*This article was typeset by Mark Taylor using the LaTeX document processing system.

```cpp
// Sparse Matrix header
#pragma once
#ifndef SMATRIX_H
#define SMATRIX_H
//#include <initializer_list>
#include <iostream>
#include <cassert>

#define use_mySort
#if defined use_mySort
#define Fast3way_partition
#define QUICK_INSERTION_SORT
#include "mySort.h"
#else
#include <algorithm>
#endif // defined use_mySort

constexpr size_t defaultSize = 20;

// forward declaration, though may be unnecessary in VS
// see more at https://stackoverflow.com/questions/61983237/how-to-enable-a-\
                          friend-classs-friend-function-access-its-private-members-direct
// see also <http://eel.is/c++draft/class.friend#11>
template<typename T> class SMatrix;

// tri-tuple term for sparse matrix by the form <row, col, value>
template<typename T>
class TriTuple {
        template<typename U> friend class SMatrix;
        template<typename U> // enable friend of SMatrix to access private members of TriTuple
        friend std::ostream& operator<<(std::ostream& os, const SMatrix<U>& M);
private:
        size_t _row, _col;
        T _val;
public:
        TriTuple(size_t row = 0, size_t col = 0, T value = {})
                : _row(row), _col(col), _val(value) {}

        TriTuple<T>& operator=(const TriTuple<T>& term) {
                _row = term._row;
                _col = term._col;
                _val = term._val;
                return *this;
        }
};

template<typename T>
class SMatrix {
public:
        SMatrix(size_t maxSize = defaultSize);
        SMatrix(size_t rows, size_t cols, std::initializer_list<TriTuple<T>> elemList, size_t maxSize =
        ↪  defaultSize);
        SMatrix(const SMatrix<T>& M);
        ~SMatrix() { delete[] _arr; };

        void printHeader()const;
        SMatrix<T>& operator=(const SMatrix<T>& M);
        SMatrix<T> transpose()const;
        SMatrix<T> fast_transpose()const;

        SMatrix<T> add(const SMatrix<T>& M)const;
        SMatrix<T> multiply(const SMatrix<T>& M)const;
        template <typename U>
        friend SMatrix<U> operator+(const SMatrix<U>& A, const SMatrix<U>& B);
        template <typename U>
        friend SMatrix<U> operator*(const SMatrix<U>& A, const SMatrix<U>& B);

private:
        size_t _rows, _cols;// # of rows & columns
        size_t _terms;                // # of terms
        TriTuple<T>* _arr;        // stored by 1-dimensional array
        size_t _maxSize;

        template<typename U>
        friend std::ostream& operator<<(std::ostream& os, const SMatrix<U>& M);
};

template<typename T>
inline SMatrix<T>::SMatrix(size_t maxSize)
        :_rows(0), _cols(0), _terms(0), _maxSize(maxSize)
{
        _arr = new TriTuple<T>[maxSize] {};
        assert(_arr != nullptr);
}

template<typename T>
SMatrix<T>::SMatrix(size_t rows, size_t cols, std::initializer_list<TriTuple<T>> elemList, size_t maxSize)
        : _rows(rows), _cols(cols), _maxSize(maxSize)
{
        // assignment examples: {20, 20, {{1,3,20.6},{9,7,18.9},{15,12,21.3}}, 30 }
        _arr = new TriTuple<T>[maxSize];
        assert(_arr != nullptr);

        size_t i = 0;
        for (auto it = elemList.begin(); it != elemList.end() && i < maxSize; ++it) {
                assert((it->_row) <= _rows);
                assert((it->_col) <= _cols);
                _arr[i++] = *it;
        }

        _terms = i;

        // we may need to sort those trituples in case they're NOT input by rows in ascending order
        // e.g. {{17,3,20}, {9,14,90}, {15,12,50}}
#if defined use_mySort
        mySortingAlgo::
#else
        std::
#endif // defined use_mySort
                sort(_arr, _arr + _terms, [](const TriTuple <T>& a, const TriTuple<T>& b) {
                return (a._row < b._row);
                });
}

template<typename T>
SMatrix<T>::SMatrix(const SMatrix<T>& M)
        :_rows(M._rows), _cols(M._cols), _terms(M._terms), _maxSize(M._maxSize)
{
        _arr = new TriTuple<T>[_maxSize];
        assert(_arr != nullptr);
        for (size_t i = 0; i < _terms; ++i)
                _arr[i] = M._arr[i];
```

```
}

template<typename T>
inline void SMatrix<T>::printHeader()const {
        std::cout << "-----------------------------------\n";
        std::cout << "row            col            value\n";
        std::cout << "-----------------------------------\n";
}

template<typename T>
SMatrix<T>& SMatrix<T>::operator=(const SMatrix<T>& M) {
        _rows = M._rows;
        _cols = M._cols;
        _terms = M._terms;
        _arr = new TriTuple<T>[M._maxSize];
        assert(_arr != nullptr);
        for (size_t i = 0; i < _terms; ++i)
                _arr[i] = M._arr[i];

        return *this;
}

template<typename T>
SMatrix<T> SMatrix<T>::transpose()const
{
        SMatrix<T> B(_maxSize);
        B._rows = _rows;
        B._cols = _cols;
        B._terms = _terms;
        if (_terms > 0) {
                size_t posB = 0;
                // Since sparse matrix is stored by rows, we need to traverse by
                // columns to find those nonzeros & exchange their <row, col>.
                for (size_t j = 0; j < _cols; ++j) {
                        for (size_t k = 0; k < _terms; ++k) {
                                if (_arr[k]._col == j) {
                                        B._arr[posB]._row = j;
                                        B._arr[posB]._col = _arr[k]._row;
                                        B._arr[posB]._val = _arr[k]._val;
                                        ++posB;
                                }
                        }
                }
        }
        return B;
}

template<typename T>
SMatrix<T> SMatrix<T>::fast_transpose()const
{
        // The main idea is to record the initial indices of each
        // cols (which have non-zero terms) in the storage array.
        // kinda like count sort
        SMatrix<T> B(_maxSize);
        B._rows = _rows;
        B._cols = _cols;
        B._terms = _terms;

        int* rowStart = new int[_cols + 1]{ 0 }; // # of non-zero terms of each col of A

        // count frequences
        for (size_t i = 0; i < _terms; ++i)
                ++rowStart[_arr[i]._col + 1];

        // transform counts into indices
        for (size_t i = 0; i < _cols; ++i)
                rowStart[i + 1] += rowStart[i];

        int j;
        for (size_t i = 0; i < _terms; ++i) {
                j = rowStart[_arr[i]._col]++;        // i-th item of A transposed to the j-th position of B
                B._arr[j]._row = _arr[i]._col;
                B._arr[j]._col = _arr[i]._row;
                B._arr[j]._val = _arr[i]._val;
        }
        delete[] rowStart;

        return B;
}

template<typename T>
SMatrix<T> SMatrix<T>::add(const SMatrix<T>& B)const
{
        assert(_rows == B._rows && _cols == B._cols);
        size_t i = 0, j = 0; // position of A, B
        size_t index_A, index_B; // full position of A, B

        // result array. In general, (_maxSize + B._maxSize) is smaller
        SMatrix<T> C((_maxSize + B._maxSize) < (_rows * _cols) ? _maxSize + B._maxSize : _rows * _cols);
        C._rows = _rows;
        C._cols = _cols;

        size_t k = 0; // position of C
        while (i < _terms && j < B._terms) {
                index_A = _arr[i]._row * _cols + _arr[i]._col;
                index_B = B._arr[j]._row * _cols + B._arr[j]._col;
                if (index_A < index_B) {// push the item that has smaller index
                        C._arr[k++] = _arr[i++];
                }
                else if (index_A > index_B) {
                        C._arr[k++] = B._arr[j++];
                }
                else {// same position, add these two items together
                        C._arr[k]._row = _arr[i]._row;
                        C._arr[k]._col = _arr[i]._col;
                        C._arr[k++]._val = _arr[i++]._val + B._arr[j++]._val;
                }
        }

        // copy residual part
        while (i < _terms) {
                C._arr[k++] = _arr[i++];
        }
        while (j < B._terms) {
                C._arr[k++] = B._arr[j++];
        }
        C._terms = k;

        return C;
}

template<typename T>
```

```cpp
inline SMatrix<T> SMatrix<T>::multiply(const SMatrix<T>& B)const
{
        // to be implemented
        return SMatrix<T>();
}

template<typename U>
inline SMatrix<U> operator+(const SMatrix<U>& A, const SMatrix<U>& B)
{
        return A.add(B);
}

template<typename U>
inline SMatrix<U> operator*(const SMatrix<U>& A, const SMatrix<U>& B)
{
        return A.multiply(B);
}

template<typename U>
std::ostream& operator<<(std::ostream& os, const SMatrix<U>& M)
{
        M.printHeader();
        for (size_t i = 0; i < M._terms; ++i) {
                os << M._arr[i]._row << "\t\t" << M._arr[i]._col << "\t\t" << M._arr[i]._val << '\n';
        }
        return os;
}

#endif // !SMATRIX_H
```

Listing 1: Sparse Matrix header

2.2 SMatrix_test.cpp

```cpp
#include "SMatrix.h"
#include <iostream>

using namespace std;

int main()
{
        int m = 20, n = 20;
        SMatrix<int> A(m, n, { {17,3,20}, { 9,14,90 }, { 15,12,50 } ,
                {3,8,10}, {11,4, 80}, {7,12,30}, {9,11,60}, {15,4,70} });
        cout << "Original sparse matrix A (" << m << 'x' << n << "):\n"
                 << A << "\n";

        auto B = A.transpose();
        cout << "B = A.transpose():\n"
                 << B << "\n";

        auto C = A.fast_transpose();
        cout << "\nC = A.fast_transpose():\n"
                 << C << "\n";


        cout << "\n**Addition test**\n";
        SMatrix<int> D(m, n, { {19,3,20}, { 6,3,90 }, { 15,12,50 } ,
                {3,8,10}, {11,4, 80}, {5,2,30}, {4,10,60}, {16,4,70} });
        cout << "Another sparse matrix D (" << m << 'x' << n << "):\n"
                 << D << "\n";

        auto E = A + D;
        cout << "\nE = A + D:\n"
                 << E << "\n";

        cout << "Press any key to leave...\n";
        char wait;
        cin >> noskipws >> wait;
        return 0;
}
```

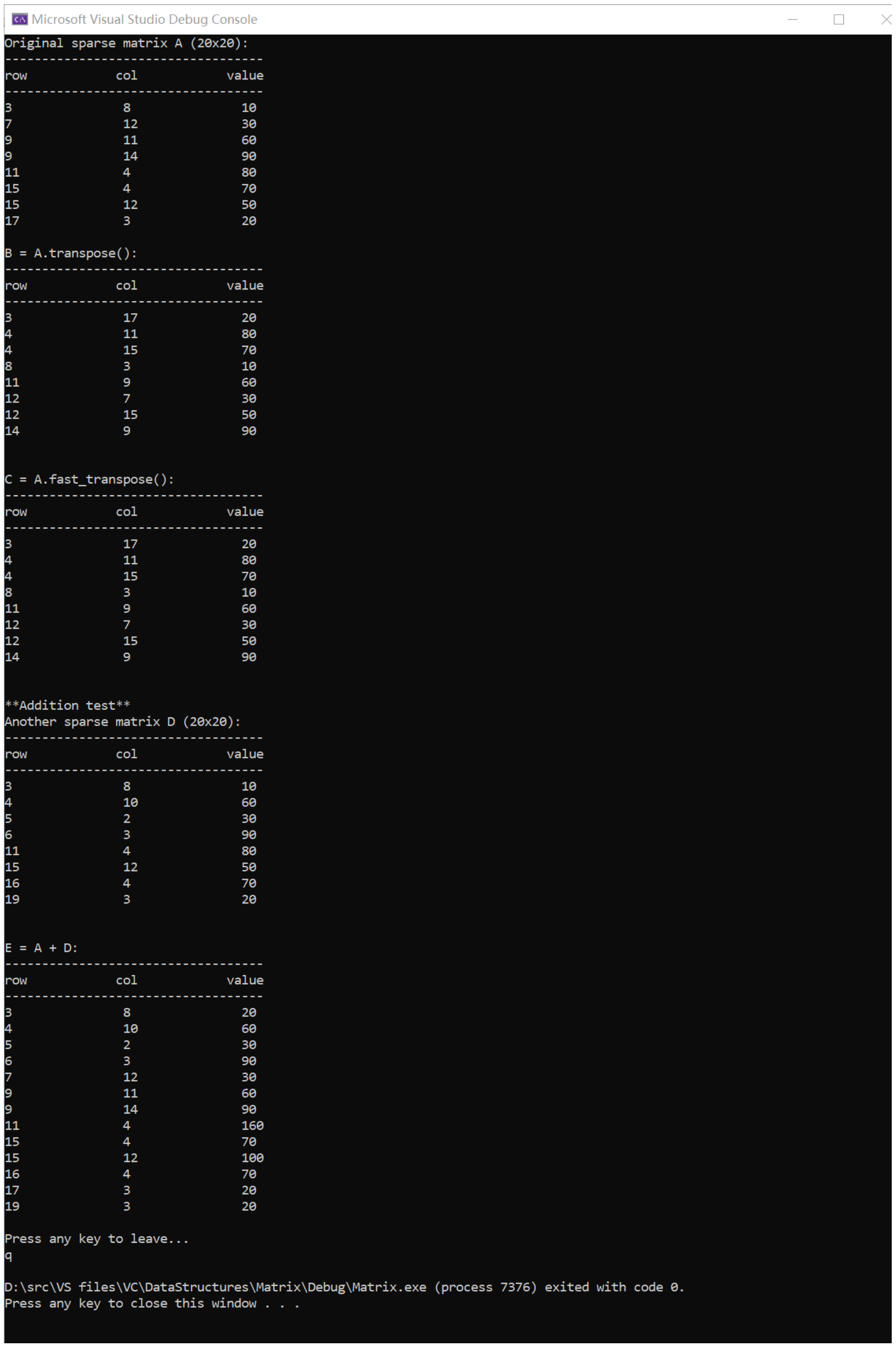
Listing 2: Sparse Matrix test

2.3 mySort.h

This is a large file, see it in the *src* folder, or view it online here.

3 Output

3.1 Win10

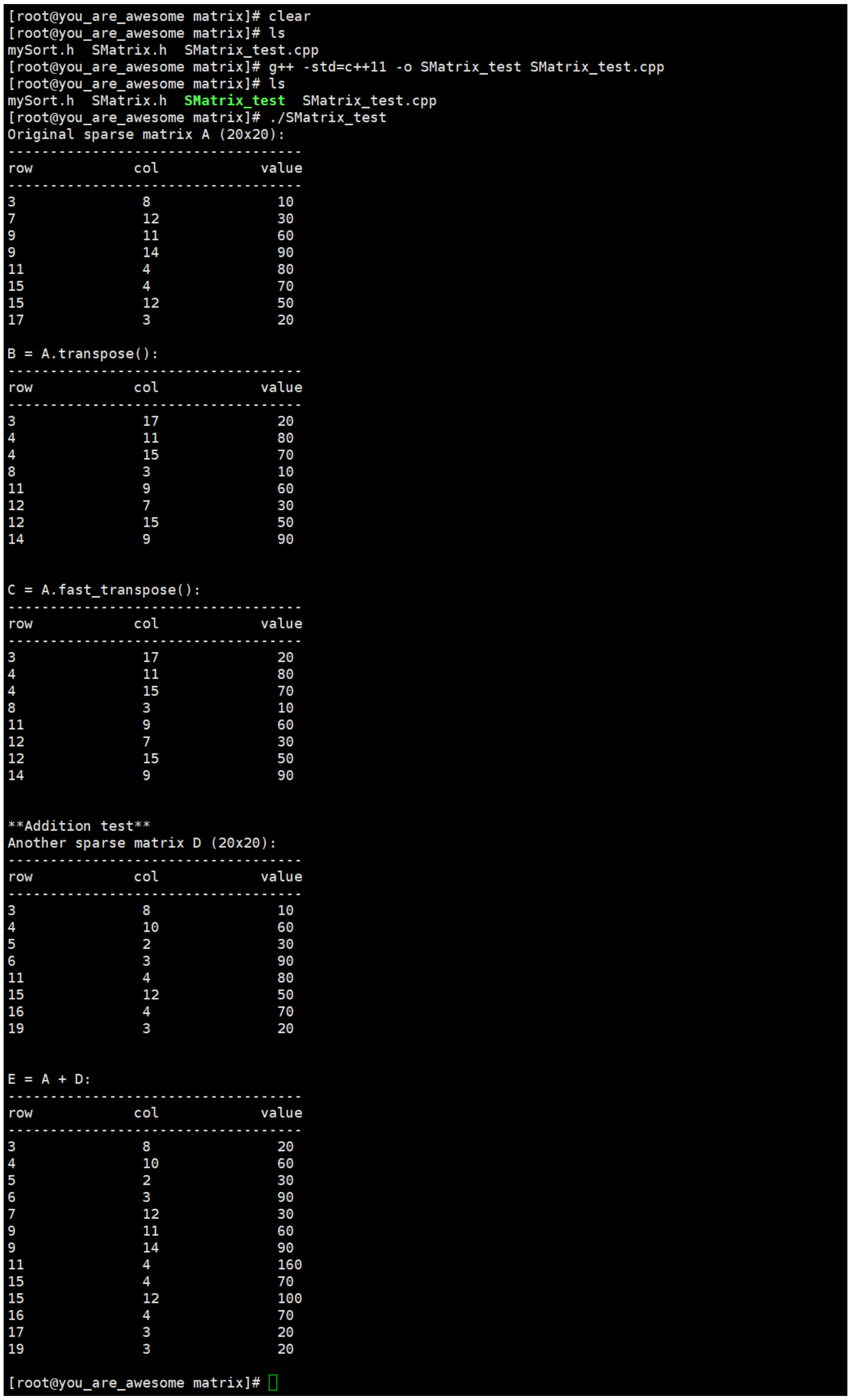


Microsoft Visual Studio Debug Console

```
Original sparse matrix A (20x20):
-----------------------------------
row             col             value
-----------------------------------
3               8               10
7               12              30
9               11              60
9               14              90
11              4               80
15              4               70
15              12              50
17              3               20

B = A.transpose():
-----------------------------------
row             col             value
-----------------------------------
3               17              20
4               11              80
4               15              70
8               3               10
11              9               60
12              7               30
12              15              50
14              9               90


C = A.fast_transpose():
-----------------------------------
row             col             value
-----------------------------------
3               17              20
4               11              80
4               15              70
8               3               10
11              9               60
12              7               30
12              15              50
14              9               90


**Addition test**
Another sparse matrix D (20x20):
-----------------------------------
row             col             value
-----------------------------------
3               8               10
4               10              60
5               2               30
6               3               90
11              4               80
15              12              50
16              4               70
19              3               20


E = A + D:
-----------------------------------
row             col             value
-----------------------------------
3               8               20
4               10              60
5               2               30
6               3               90
7               12              30
9               11              60
9               14              90
11              4               160
15              4               70
15              12              100
16              4               70
17              3               20
19              3               20

Press any key to leave...
q

D:\src\VS files\VC\DataStructures\Matrix\Debug\Matrix.exe (process 7376) exited with code 0.
Press any key to close this window . . .
```

Figure 1: Sparse Matrix test results in Win10

3.2 Linux



```
[root@you_are_awesome matrix]# clear
[root@you_are_awesome matrix]# ls
mySort.h  SMatrix.h  SMatrix_test.cpp
[root@you_are_awesome matrix]# g++ -std=c++11 -o SMatrix_test SMatrix_test.cpp
[root@you_are_awesome matrix]# ls
mySort.h  SMatrix.h  SMatrix_test  SMatrix_test.cpp
[root@you_are_awesome matrix]# ./SMatrix_test
Original sparse matrix A (20x20):
-----------------------------------
row           col           value
-----------------------------------
3              8              10
7              12             30
9              11             60
9              14             90
11             4              80
15             4              70
15             12             50
17             3              20

B = A.transpose():
-----------------------------------
row           col           value
-----------------------------------
3              17             20
4              11             80
4              15             70
8              3              10
11             9              60
12             7              30
12             15             50
14             9              90


C = A.fast_transpose():
-----------------------------------
row           col           value
-----------------------------------
3              17             20
4              11             80
4              15             70
8              3              10
11             9              60
12             7              30
12             15             50
14             9              90


**Addition test**
Another sparse matrix D (20x20):
-----------------------------------
row           col           value
-----------------------------------
3              8              10
4              10             60
5              2              30
6              3              90
11             4              80
15             12             50
16             4              70
19             3              20


E = A + D:
-----------------------------------
row           col           value
-----------------------------------
3              8              20
4              10             60
5              2              30
6              3              90
7              12             30
9              11             60
9              14             90
11             4              160
15             4              70
15             12             100
16             4              70
17             3              20
19             3              20

[root@you_are_awesome matrix]# 
```

Figure 2: Sparse Matrix test results in Linux (CentOS)

4 Appendix

Hello from the Beatles.😃