how U doin'?

$$🐳^{🐳} = \int_{🎃}^{🎅} 🐑 \, d🍀$$

Sparse Matrix*

10170437 Mark Taylor

June 24, 2020

Contents

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;
```

```
61          SMatrix<T> multiply(const SMatrix<T>& M)const;
62          template <typename U>
63          friend SMatrix<U> operator+(const SMatrix<U>& A, const SMatrix<U>& B);
64          template <typename U>
65          friend SMatrix<U> operator*(const SMatrix<U>& A, const SMatrix<U>& B);
66
67  private:
68          size_t _rows, _cols;// # of rows & columns
69          size_t _terms;                // # of terms
70          TriTuple<T>* _arr;        // stored by 1-dimensional array
71          size_t _maxSize;
72
73          template<typename U>
74          friend std::ostream& operator<<(std::ostream& os, const SMatrix<U>& M);
75  };
76
77  template<typename T>
78  inline SMatrix<T>::SMatrix(size_t maxSize)
79          :_rows(0), _cols(0), _terms(0), _maxSize(maxSize)
80  {
81          _arr = new TriTuple<T>[maxSize] {};
82          assert(_arr != nullptr);
83  }
84
85  template<typename T>
86  SMatrix<T>::SMatrix(size_t rows, size_t cols, std::initializer_list<TriTuple<T>> elemList, size_t maxSize)
87          : _rows(rows), _cols(cols), _maxSize(maxSize)
88  {
89          // assignment examples: {20, 20, {{1,3,20.6},{9,7,18.9},{15,12,21.3}}, 30 }
90          _arr = new TriTuple<T>[maxSize];
91          assert(_arr != nullptr);
92
93          size_t i = 0;
94          for (auto it = elemList.begin(); it != elemList.end() && i < maxSize; ++it) {
95                  assert((it->_row) <= _rows);
96                  assert((it->_col) <= _cols);
97                  _arr[i++] = *it;
98          }
99
100         _terms = i;
101
102         // we may need to sort those trituples in case they're NOT input by rows in ascending order
103         // e.g. {{17,3,20}, {9,14,90}, {15,12,50}}
104 #if defined use_mySort
105         mySortingAlgo::
106 #else
107         std::
108 #endif // defined use_mySort
109                 sort(_arr, _arr + _terms, [](const TriTuple <T>& a, const TriTuple<T>& b) {
110                 return (a._row < b._row);
111                 });
112 }
113
114 template<typename T>
115 SMatrix<T>::SMatrix(const SMatrix<T>& M)
116         :_rows(M._rows), _cols(M._cols), _terms(M._terms), _maxSize(M._maxSize)
117 {
118         _arr = new TriTuple<T>[_maxSize];
119         assert(_arr != nullptr);
120         for (size_t i = 0; i < _terms; ++i)
121                 _arr[i] = M._arr[i];
```

```
122 }
123
124 template<typename T>
125 inline void SMatrix<T>::printHeader()const {
126         std::cout << "---------------------------------\n";
127         std::cout << "row            col            value\n";
128         std::cout << "---------------------------------\n";
129 }
130
131 template<typename T>
132 SMatrix<T>& SMatrix<T>::operator=(const SMatrix<T>& M) {
133         _rows = M._rows;
134         _cols = M._cols;
135         _terms = M._terms;
136         _arr = new TriTuple<T>[M._maxSize];
137         assert(_arr != nullptr);
138         for (size_t i = 0; i < _terms; ++i)
139                 _arr[i] = M._arr[i];
140
141         return *this;
142 }
143
144 template<typename T>
145 SMatrix<T> SMatrix<T>::transpose()const
146 {
147         SMatrix<T> B(_maxSize);
148         B._rows = _rows;
149         B._cols = _cols;
150         B._terms = _terms;
151         if (_terms > 0) {
152                 size_t posB = 0;
153                 // Since sparse matrix is stored by rows, we need to traverse by
154                 // columns to find those nonzeros & exchange their <row, col>.
155                 for (size_t j = 0; j < _cols; ++j) {
156                         for (size_t k = 0; k < _terms; ++k) {
157                                 if (_arr[k]._col == j) {
158                                         B._arr[posB]._row = j;
159                                         B._arr[posB]._col = _arr[k]._row;
160                                         B._arr[posB]._val = _arr[k]._val;
161                                         ++posB;
162                                 }
163                         }
164                 }
165         }
166         return B;
167 }
168
169 template<typename T>
170 SMatrix<T> SMatrix<T>::fast_transpose()const
171 {
172         // The main idea is to record the initial indices of each
173         // cols (which have non-zero terms) in the storage array.
174         SMatrix<T> B(_maxSize);
175         B._rows = _rows;
176         B._cols = _cols;
177         B._terms = _terms;
178
179         int* rowSize = new int[_cols] {};        // # of non-zero terms of each col of A, initialized with
        ↪  all zeros
180         for (size_t i = 0; i < _terms; ++i)
181                 ++rowSize[_arr[i]._col];
```

```
182
183         int* rowStart = new int [_cols] {};        // initial indices of non-zero terms of each col of A
184         //rowStart[0] = 0;
185         for (size_t i = 1; i < _cols; ++i)
186                 rowStart[i] = rowStart[i - 1] + rowSize[i - 1];
187
188         int j;
189         for (size_t i = 0; i < _terms; ++i) {
190                 j = rowStart[_arr[i]._col];        // i-th item of A transposed to the j-th position of B
191                 B._arr[j]._row = _arr[i]._col;
192                 B._arr[j]._col = _arr[i]._row;
193                 B._arr[j]._val = _arr[i]._val;
194                 ++rowStart[_arr[i]._col];        // next index of same row in B
195         }
196         delete[] rowSize;
197         delete[] rowStart;
198
199         return B;
200 }
201
202 template<typename T>
203 SMatrix<T> SMatrix<T>::add(const SMatrix<T>& B)const
204 {
205         assert(_rows == B._rows && _cols == B._cols);
206         size_t i = 0, j = 0; // position of A, B
207         size_t index_A, index_B; // full position of A, B
208
209         // result array. In general, (_maxSize + B._maxSize) is smaller
210         SMatrix<T> C((_maxSize + B._maxSize) < (_rows * _cols) ? _maxSize + B._maxSize : _rows * _cols);
211         C._rows = _rows;
212         C._cols = _cols;
213
214         size_t k = 0; // position of C
215         while (i < _terms && j < B._terms) {
216                 index_A = _arr[i]._row * _cols + _arr[i]._col;
217                 index_B = B._arr[j]._row * _cols + B._arr[j]._col;
218                 if (index_A < index_B) {// push the item that has smaller index
219                         C._arr[k++] = _arr[i++];
220                 }
221                 else if (index_A > index_B) {
222                         C._arr[k++] = B._arr[j++];
223                 }
224                 else {// same position, add these two items together
225                         C._arr[k]._row = _arr[i]._row;
226                         C._arr[k]._col = _arr[i]._col;
227                         C._arr[k++]._val = _arr[i++]._val + B._arr[j++]._val;
228                 }
229         }
230
231         // copy residual part
232         while (i < _terms) {
233                 C._arr[k++] = _arr[i++];
234         }
235         while (j < B._terms) {
236                 C._arr[k++] = B._arr[j++];
237         }
238         C._terms = k;
239
240         return C;
241 }
242
```

```
template<typename T>
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

```
#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";


```

```
23          cout << "\n**Addition test**\n";
24          SMatrix<int> D(m, n, { {19,3,20}, { 6,3,90 }, { 15,12,50 } ,
25                  {3,8,10}, {11,4, 80}, {5,2,30}, {4,10,60}, {16,4,70} });
26          cout << "Another sparse matrix D (" << m << 'x' << n << "):\n"
27                   << D << "\n";
28
29          auto E = A + D;
30          cout << "\nE = A + D:\n"
31                   << E << "\n";
32
33          cout << "Press any key to leave...\n";
34          char wait;
35          cin >> noskipws >> wait;
36          return 0;
37  }
```

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

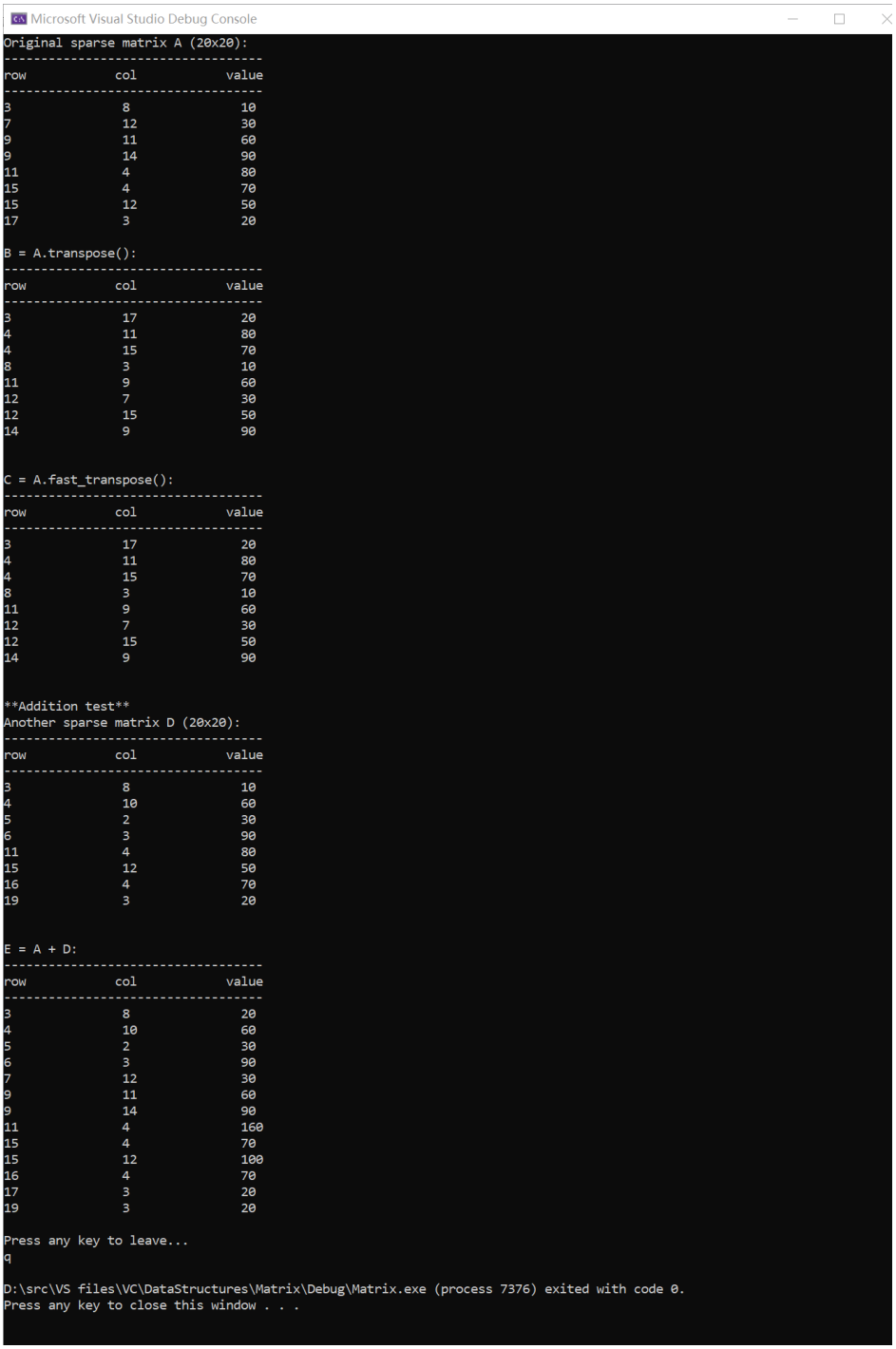

Figure 1: Sparse Matrix test results in Win10

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.😃