山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202000130143	姓名： 郑凯饶	班级： 计科 20.1
实验题目：数组与矩阵		
实验学时：2	实验日期： 1029	

实验目的：

1.掌握稀疏矩阵结构的描述及操作的实现。

软件开发环境：

Windows 10 家庭中文版 64 位（10.0，版本 18363）

Dev-C++ IDE

1. 实验内容

参照课本的相关定义，创建稀疏矩阵类，采用行主顺序把稀疏矩阵非 0 元素映射到一维数组中，并实现操作：两个稀疏矩阵相加、两个稀疏矩阵相乘、稀疏矩阵的转置、输出矩阵。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

```
template<class T>
class sparseMatrix {
    // 友元函数
    friend ostream& operator << (ostream&, sparseMatrix<int>&);
    friend istream& operator >> (istream&, sparseMatrix<int>&);

    public:
        void transpose(sparseMatrix<T> &b);                 // O(1e4)
        int add(sparseMatrix<T> &b, sparseMatrix<T> &c);    // O(2e2)
        int Mul(sparseMatrix<T> &, sparseMatrix<T> &);
        int Mul_(sparseMatrix<T> &, sparseMatrix<T> &);     // O(1e6)
        void in();                  // 重置
        void SortbyCol();           // 列主次 prepared for Q

    private:
        int Rows, Cols;
        vector<matrixTerm<T> > terms;
};
```

友元函数重载输入输出运算符：根据题目要求实现稀疏矩阵矩阵类输入输出的相关操作。

函数方法部分：

Transpose()：使用桶结构处理出矩阵每一列的元素个数，可以计算出转置之后元素在映射的一维数组中的位置。

Add()：稀疏矩阵相加，用归并的思想将矩阵中相同位置的元素相加，若结果不为 0 存入 Res 矩阵。

Mul()：稀疏矩阵的乘法，该方法和正常矩阵乘法相同，将 P 矩阵的第 i 行元素和 Q 矩阵第 j 行的元素相乘累加得到 Res 矩阵的元素(i, j)。主要解决问题是元素是存储在映射后的一维数组空间，不太容易直接按照人的思维进行乘法（尝试过，但是因为边界条件出了不少 BUG 放弃了）。最后的解决方案是预处理出 P 矩阵的行首位置以及 Q 矩阵的列首位置，去模拟矩阵乘法操作。矩阵 P 为 m*n，Q 为 n*q 时，对应时间复杂度是 O(mnq)。

Mul_()：同样也是稀疏矩阵的乘法，只不过方法是更为“暴力”地遍历两个矩阵的一维数组，寻找行列匹配的元素进行相乘累加，间接用二维数组暂存结果，可以做映射操作映射为一维数组，不过本质上仍是二维数组。因不符合题目要求，再设计了上面的 Mul()方法。时间复杂度和稀疏矩阵的元素个数 n 有关，为 O(n^2)。

In()：这个方法是为了将 OJ 输入的二维矩阵的形式转化为稀疏矩阵的方式存储。

sortbyCol()：由于稀疏矩阵类中的矩阵默认是按照行主序的方式存储，而在矩阵乘法是要预处理 Q 矩阵的列首，这里用该方法将矩阵转化为列主序的方式存储可以 O(n)确定列首。

数据成员部分：

Rows：稀疏矩阵行数。

Cols：稀疏矩阵列数。

Terms：vector 对象，为二维矩阵映射之后的一维数组。

3. 测试结果（测试输入，测试输出）

在 OJ 平台上成功提交。

一开始，因为 Mul_()的时间复杂度不符合要求一直 TLE，我添加了一句优化的判断，以 1600ms 左右的时间通过大测试点 test20。

3597120610069e6	202000130143	A	✓Accepted	100	1717 ms	6260 KiB	C++11	2 days ago
3596534d44069c3	202000130143	A	✓Accepted	100	1684 ms	6240 KiB	C++11	2 days ago
35962d1dbc069b9	202000130143	A	×Time Limit Exceeded	95	3000 ms	6252 KiB	C++11	2 days ago
3595f33170069aa	202000130143	A	×Time Limit Exceeded	95	2999 ms	6236 KiB	C++11	2 days ago

重新设计后的 Mul()方法显然综合的时间上更优，可以在 180ms 左右通过 test20。

35b98c347406d74	202000130143	A	✓Accepted	100	188 ms	6316 KiB	C++11	5 hours ago
35b96f2cd006d70	202000130143	A	×Wrong Answer	75	157 ms	6292 KiB	C++11	5 hours ago

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

整个 sparseMatrix 的设计比较曲折。一开始我是使用了教科书上的例程进行改写，完成了包括重载输入输出、矩阵加法、矩阵转置的方法设计。主要更改是将例程中作者自主设计的 arrayList 改为 vector，对使用的一些方法进行了调整，对 resize()的滥用导致我调试了很久。

之后就是矩阵乘法超时，中途我一度以为是重载的输入输出函数时间上不够优秀，改为标准输入输出之后，对比发现两者几乎相同。锁定问题后，我增加了一句优化通过了 OJ 的 test。但是方法上仍不符合题目要求。

于是我还是想设计第三版的稀疏矩阵乘法，之前我尝试过直接模拟，但是细节问题太多了，不是很好直接实现。

经过和同学讨论，我考虑预处理出 P 的行首和 Q 的列首，这样能比较方便地模拟。最后成功了，时间上也更优。

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

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


// 1、arratList -> vector
// 2、reload operator << and >> : T -> int
// 3、
   reSet -> resize(num): If vector's size < num, it will extend though fill 0 eleme
   nt
// 4、rows -> Rows cols -> Cols

```

```
template <class T>
struct matrixTerm
{
 int row, col;
    T val;

    operator T() const {return val;}
    // type conversion from matrixTerm to T
    bool operator < (const matrixTerm &A) {
     if (col == A.col) return row < A.row;
     return col < A.col;
 }
};

template<class T>
class sparseMatrix {
 // 友元函数
    friend ostream& operator << (ostream&, sparseMatrix<int>&);
    friend istream& operator >> (istream&, sparseMatrix<int>&);

 public:
  void transpose(sparseMatrix<T> &b);     // O(1e4)
  int add(sparseMatrix<T> &b, sparseMatrix<T> &c); // O(2e2)
  int Mul(sparseMatrix<T> &, sparseMatrix<T> &);
  int Mul_(sparseMatrix<T> &, sparseMatrix<T> &);  // O(1e6)
  void in();     // 重置
     void SortbyCol();   // 列主次 prepared for Q

 private:
  int Rows, Cols;
  vector<matrixTerm<T> > terms;
};

// sparseMatrix's method achieve
// output
// explict code tooverload with T = int for test as compiler
// unable to generate
ostream& operator << (ostream& out, sparseMatrix<int>& x)
{// Put x in output stream.

   // put matrix characteristics
 out << x.Rows << " " << x.Cols  << '\n';

 vector<matrixTerm<int> >::iterator it = x.terms.begin();
 for (int i = 1; i <= x.Rows; i++) {
  for (int j = 1; j <= x.Cols; j++) {
```

```
63.    if (i == it -> row && j == it -> col && it != x.terms.end()) {
64.     out << it -> val;
65.     it ++;
66.    }
67.    else out << 0;
68.    if (j == x.Cols) out << " \n";
69.    else out << " ";
70.   }
71. }
72. return out;
73.}
74.
75.istream& operator >> (istream& in, sparseMatrix<int>& x)
76.{// Input a sparse matrix.
77.
78.    // input matrix characteristics
79. int numberOfTerms;
80. in >> x.Rows >> x.Cols >> numberOfTerms;
81.    // set size of x.terms and ensure enough capacity
82.     x.terms.resize(0); // numberOfTerms
83. x.terms.clear();
84.
85.    // input terms
86.     matrixTerm<int> mTerm;
87.     for (int i = 0; i < numberOfTerms; i++)
88.     {
89.         in >> mTerm.row >> mTerm.col >> mTerm.val;
90.     x.terms.push_back(mTerm);
91.     }
92.     return in;
93.}
94.
95.template<class T>
96.void sparseMatrix<T>::transpose(sparseMatrix<T> &b)
97.{// Return transpose of *this in b.
98.
99. // set transpose characteristics
100.     b.Cols = Rows;
101.     b.Rows = Cols;
102.     b.terms.resize(terms.size());
103.
104.     // initialize to compute transpose
105.     int* colSize = new int[Cols + 1];
106.     int* rowNext = new int[Cols + 1];
107.
108.     // find number of entries in each column of *this
```

```
109.     for (int i = 1; i <= Cols; i++) // initialize
110.      colSize[i] = 0;
111.
112.     // 类似桶排序过程
113.     for (vector<matrixTerm<int> >::iterator i = terms.begin(); i != terms.end()
    ; i++)
114.      colSize[i -> col]++;
115.
116.     // find the starting point of each row of b
117.     rowNext[1] = 0;
118.     for (int i = 2; i <= Cols; i++){
119.      rowNext[i] = rowNext[i - 1] + colSize[i - 1];
120.     }
121.
122.     // perform the transpose copying from *this to b
123.     // 确定转置之后的映射关系, 直接set
124.     matrixTerm<T> mTerm;
125.     for (vector<matrixTerm<int> >::iterator i = terms.begin(); i != terms.end()
    ; i++)
126.     {
127.      int j = rowNext[i -> col]++; // position in b
128.      mTerm.row = (*i).col;
129.      mTerm.col = (*i).row;
130.      mTerm.val = (*i).val;
131.      //   b.terms.set(j, mTerm);
132.      b.terms[j] = mTerm;
133.     }
134.    /* 3 1 4 2 5 9 */
135.    }
136.
137.    template<class T>
138.    int sparseMatrix<T>::add(sparseMatrix<T> &b, sparseMatrix<T> &c)
139.    {// Compute c = (*this) + b.
140.
141.     // verify compatibility
142.     if (Rows != b.Rows || Cols != b.Cols) {
143.      // P = Q
144.      return -1;
145.     }
146.     //     throw matrixSizeMismatch(); // incompatible matrices
147.
148.     // set characteristics of result c
149.     c.Rows = Rows;
150.     c.Cols = Cols;
151.     c.terms.clear();
152.
```

```
153.    // define iterators for *this and b
154.    vector<matrixTerm<int> >::iterator it = terms.begin();
155.    vector<matrixTerm<int> >::iterator ib = b.terms.begin();
156.    vector<matrixTerm<int> >::iterator itEnd = terms.end();
157.    vector<matrixTerm<int> >::iterator ibEnd = b.terms.end();
158.
159.    // move through *this and b adding like terms
160.    while (it != itEnd && ib != ibEnd)
161.    {
162.     // row-major index plus cols of each term
163.     int tIndex = (*it).row * Cols + (*it).col;
164.     int bIndex = (*ib).row * Cols + (*ib).col;
165.
166.     if (tIndex < bIndex)
167.     {// b term comes later
168.      c.terms.push_back(*it);
169.      it++;
170.     }
171.     else {
172.      if (tIndex == bIndex)
173.         {// both in same position
174.
175.              // append to c only if sum not zero
176.             if ((*it).val + (*ib).val != 0)
177.             {
178.                 matrixTerm<T> mTerm;
179.                 mTerm.row = (*it).row;
180.                 mTerm.col = (*it).col;
181.                 mTerm.val = (*it).val + (*ib).val;
182.        c.terms.push_back(mTerm);
183.             }
184.
185.             it++;
186.             ib++;
187.          }
188.          else
189.          {// a term comes later
190.       c.terms.push_back(*ib);
191.             ib++;
192.          }
193.        }
194.    }
195.
196.    // copy over remaining terms
197.    for (; it != itEnd; it++) c.terms.push_back(*it);
198.    for (; ib != ibEnd; ib++) c.terms.push_back(*ib);
```

```
199.
200.     return 0;
201.   }
202.
203.   template<class T>
204.   void sparseMatrix<T>::in() {
205.     // 矩阵 -> 稀疏矩阵
206.     this -> terms.clear();
207.
208.     int _in; cin >> Rows >> Cols;
209.     for (int i = 1; i <= Rows; i++) {
210.       for (int j = 1; j <= Cols; j++) {
211.         cin >> _in;
212.         if (_in) {
213.           this -> terms.push_back({i, j, _in});
214.         }
215.       }
216.     }
217.   }
218.
219.   template<class T>
220.   void sparseMatrix<T>::SortbyCol() {
221.     sort (terms.begin(), terms.end());
222.   }
223.
224.   //template<class T>
225.   //int sparseMatrix<T>::Mul(sparseMatrix<T> &A, sparseMatrix<T> &Res) {
226.   //
227.   // if (A.Rows != Cols) return -1;
228.   //
229.   // A.SortbyCol ();
230.   //
231.   // Res.Rows = Rows;
232.   // Res.Cols = A.Cols;
233.   // Res.terms.clear();
234.   //
235.   // vector<matrixTerm<int> >::iterator it = terms.begin();
236.   // vector<matrixTerm<int> >::iterator it_;
237.   // vector<matrixTerm<int> >::iterator End = terms.end();
238.   // vector<matrixTerm<int> >::iterator End_ = A.terms.end();
239.   //
240.   //
241.   // for (;1;) {           // 执行一次，  运算 P 矩阵的一行
242.   //   auto pRow = it;
243.   //   int pCol, acc = 0;        // 前驱，  累加器
244.   //   for (it_ = A.terms.begin(); ;) {
```

```
245.    //   pCol = it_ -> col;
246.    //   if (it -> col == it_ -> row) {
247.    //    acc += it -> val * it -> val;
248.    //    it++;
249.    //    it_++;
250.    //   }
251.    //   else if (it -> col < it_ -> row) {
252.    //    it++;
253.    //   }
254.    //   else if (it -> col > it_ -> row) {
255.    //    it_++;
256.    //   }
257.    //   if (it_ == End) break;
258.    //   if (pRow -> row != it -> row) {      // P 换行, 可能此时 Q 也换列
259.    //    while (it_ != End && pCol == it_ -> col) it_++; // Q 换列
260.    //    Res.terms.push_back({pRow -> row, pCol, acc});
261.    //    acc = 0;
262.    //    it = pRow;
263.    //   }
264.    //   if (it_ == End) break;
265.    //   if (it_ -> col != pCol) {       // Q 换列
266.    //    Res.terms.push_back({it -> row, pCol, acc});
267.    //    acc = 0;
268.    //    it = pRow;
269.    //   }
270.    //  }
271.    //  while (it != End && pRow -> row == it -> row) it++;  // P 换行
272.    //  if (it == End) break;
273.    // }
274.    //
275.    // return 0;
276.    //}
277.
278.
279.    template<class T>
280.    int sparseMatrix<T>::Mul(sparseMatrix<T> &A, sparseMatrix<T> &Res) {
281.     if (A.Rows != Cols) return -1;
282.
283.     A.SortbyCol ();
284.
285.     Res.Rows = Rows;
286.     Res.Cols = A.Cols;
287.    // Res.terms.clear();
288.     Res.terms.resize(0);
289.
290.     vector<matrixTerm<int> >::iterator it = terms.begin();
```

```
291.     vector<matrixTerm<int> >::iterator it_ = A.terms.begin();
292.     vector<matrixTerm<int> >::iterator End = terms.end();
293.     vector<matrixTerm<int> >::iterator End_ = A.terms.end();
294.
295.     // pretreatment for P 行首,  Q 列首
296.     int P[505], Q[505], p = 0, q = 0;
297.     int preofP[505], preofQ[505];
298.
299.     memset(P, 0, sizeof P);
300.     memset(Q, 0, sizeof Q);
301.     memset(preofP, 0, sizeof preofP);
302.     memset(preofQ, 0, sizeof preofQ);
303.
304.     for (int pre = 0, cnt = 0; ; it++) {
305.      if (it == End) {
306.       P[p] = cnt;
307.       break;
308.      }
309.      if(pre) {
310.       if (pre != it -> row) {
311.        P[p] = cnt;
312.        cnt = 0, p++;
313.       }
314.      }
315.      cnt++;
316.      pre = it -> row;
317.     }
318.
319.     for (int pre = 0, cnt = 0; ; it_++) {
320.    //  cout << it_ -> row << " " << it_ -> col << '\n';
321.      if (it_ == End_) {
322.       Q[q] = cnt;
323.       break;
324.      }
325.      if(pre) {
326.       if (pre != it_ -> col) {
327.        Q[q] = cnt;
328.        cnt = 0, q++;
329.       }
330.      }
331.      cnt++;
332.      pre = it_ -> col;
333.     }
334.
335.     for (int i = 1; i <= p + 1; i++) {
336.      preofP[i] = preofP[i - 1] + P[i - 1];
```

```
337.     }
338.     for (int i = 1; i <= q + 1; i++) {
339.      preofQ[i] = preofQ[i - 1] + Q[i - 1];
340.     }
341.
342.    // for (int i = 0; i <= q + 1; i++) {
343.    //  cout << preofQ[i] << " ";
344.    // }
345.    // cout << "**\n";
346.     it = terms.begin();
347.     it_ = A.terms.begin();
348.
349.     // Mul
350.
351.     for (int i = 0; i <= p; i++) {
352.      for (int j = 0; j <= q; j++) {
353.       auto iter = it + preofP[i];
354.       auto iter_ = it_ + preofQ[j];
355.       // End
356.       auto ed = it + preofP[i + 1];
357.       auto ed_ = it_ + preofQ[j + 1];
358.
359.       int acc = 0, curRow = iter -> row, curCol = iter_ -> col;
360.       while (iter != ed && iter_ != ed_) {
361.        if (iter -> col == iter_ -> row) {
362.         acc += iter -> val * iter_ -> val;
363.         iter++;
364.         iter_++;
365.        }
366.        else if (iter -> col < iter_ -> row) {
367.         iter++;
368.        }
369.        else if (iter -> col > iter_ -> row) {
370.         iter_++;
371.        }
372.       }
373.    //   cout << acc << "**\n";
374.    //   cout << curRow << " " << curCol << " " << acc << "**\n";
375.       Res.terms.push_back({curRow, curCol, acc});
376.      }
377.     }
378.     return 0;
379.    }
380.
381.
382.    template<class T>
```

```
383.  int sparseMatrix<T>::Mul_(sparseMatrix<T> &A, sparseMatrix<T> &Res) {
384.   if (A.Rows != Cols) return -1;
385.
386.
387.   Res.Rows = Rows;
388.   Res.Cols = A.Cols;
389.   Res.terms.clear();
390.
391.   vector<matrixTerm<int> >::iterator it = terms.begin();
392.   vector<matrixTerm<int> >::iterator it_;
393.   vector<matrixTerm<int> >::iterator End = terms.end();
394.   vector<matrixTerm<int> >::iterator End_ = A.terms.end();
395.
396.  // int R[505][505];
397.   int R[250005];
398.   // (500 - 1) * 500 + 500
399.   memset(R, 0, sizeof R);
400.
401.   for (;it != End; it++) {
402.    for (it_ = A.terms.begin(); it_ != End_; it_++) { // it -> row "++"
403.     if (it -> col == it_ -> row) {
404.      R[(it -> row - 1) * 500 + it_ -> col] += (it -> val) * (it_ -> val);
405.     }
406.     if (it -> col < it_ -> row) break;
407.    }
408.   }
409.   int H = 100;
410.  // if (terms.size() * A.terms.size() > (int) 5e2) H = 100;
411.
412.   for (int i = 1; i <= H; i++) {
413.    for (int j = 1; j <= H; j++) {
414.     int ind = (i - 1) * 500 + j;
415.     if (R[ind]) {
416.      Res.terms.push_back({i, j, R[ind]});
417.     }
418.    }
419.   }
420.
421.   return 0;
422.  }
423.
424.  class op {
425.   public:
426.    op(int n) { this -> n = n; }
427.    void run();
428.   private:
```

```
    int n;
};

void op::run() {
 sparseMatrix<int> P, Q, Res;

 for (int i = 1, ch; i <= n; i++) {
  cin >> ch;
  if (ch == 1) {
   P.in();
  }
  else if (ch == 2) {
   // Mul
   cin >> Q;
   if (P.Mul(Q, Res) == -1) {
    cout << -1 << '\n';
    P = Q;
   }
   else P = Res;
  }
  else if (ch == 3) {
   cin >> Q;
   if(P.add(Q, Res) == -1) {
    cout << -1 << '\n';
    P = Q;
   }
   else P = Res;
  }
  else if (ch == 4){
   cout << P;
  }
  else if (ch == 5) {
   // transpose
   P.transpose(Res);
   P = Res;
  }
 }
}

int main(){
// freopen("out.txt", "w", stdout);
 ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

 int num; cin >> num;
 op op_(num);
 op_.run();
```

```
475.     return 0;
476.    }
```