山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： | 姓名： | | 班级： |
| 实验题目：矩阵 | | | |
| 实验学时：4小时 | | 实验日期：2018.10.25 | |
| 实验目的：  掌握矩阵结构。 | | | |
| 硬件环境： 笔记本电脑 | | | |
| 软件环境：Win10+Vistual Studio 2017 | | | |
| 实验步骤与内容：  实验内容：  1、创建三对角矩阵类，采用按列映射方式，提供store和retrieve 方法。  2、创建下三角矩阵类，采用按列映射方式，提供store和retrieve 方法。  3、创建稀疏矩阵类，采用行主顺序把稀疏矩阵映射到一维数组中，实现稀疏矩阵的转置和两个稀疏矩阵的加法操作。  实验步骤：  term.h  #pragma once  class Term  {  friend class sparseMatrix;  private:  int row, col;  int value;  };  tridiagonalMatrix.h  #pragma once  #include "term.h"  #include <iostream>  using namespace std;  class tridiagonalMatrix  {  public:  tridiagonalMatrix(int size = 10);  void store(int row, int column, int value);  int retrieve(int row, int column);  //矩阵输入函数  void input(istream& in, ostream& out);  //矩阵输出函数  void output(ostream& out);  //重载>>  friend istream& operator>> (istream& in, tridiagonalMatrix& matrix)  {  matrix.input(in, cout);  return in;  }  //重载<<  friend ostream& operator<< (ostream& out,tridiagonalMatrix& matrix)  {  matrix.output(out);  return out;  }  private:  int \*element;//存储所有元素  int size;  };  tridiagonalMatrix.cpp  #include "tridiagonalMatrix.h"  #include <iostream>  using namespace std;  tridiagonalMatrix::tridiagonalMatrix(int size)  {  if (size < 1)  exit(EXIT\_FAILURE);  element = new int[3 \* size - 2];  this->size = size;  }  void tridiagonalMatrix::store(int row, int column, int value)  {  if (row < 1 || row > size || column < 1 || column > size|| (row - column) > 1 || (row - column) < -1)  exit(EXIT\_FAILURE);  element[2 \* column + row - 3] = value;  }  int tridiagonalMatrix::retrieve(int row, int column)  {  if (row < 1 || row > size || column < 1 || column > size|| (row - column) > 1 || (row - column) < -1)  exit(EXIT\_FAILURE);  return element[2 \* column + row - 3];  }  void tridiagonalMatrix::input(istream& in, ostream& out)  {  out << "请按列主顺序依次输入三对角矩阵元素，元素个数必须是" << (3 \* size - 2) << "个: "  << endl;  for (int j = 0; j < size; j++)  {  for (int i = j - 1; i <= j + 1; i++)  {  if (j == 0 && i == -1)  continue;  if (j == size - 1 && i == size)  continue;  int element;  in >> element;  store(i + 1, j + 1, element);  }  }  }  void tridiagonalMatrix::output(ostream& out)  {  for (int i = 0; i < size; i++)  {  for (int j = 0; j < size; j++)  if ((i - j) > 1 || (i - j) < -1)  out << "0\t";  else  {  out << element[2 \* j + i] << "\t";  }  out << endl;  }  }  lowerTriangularMatrix.h  #pragma once  #include "term.h"  #include <iostream>  using namespace std;  class lowerTriangularMatrix  {  public:  lowerTriangularMatrix(int size = 10);  void store(int row,int column,int value);  int retrieve(int row,int column);  //矩阵输入函数  void input(istream& in, ostream& out);  //矩阵输出函数  void output(ostream& out);  //重载>>  friend istream& operator>> (istream& in, lowerTriangularMatrix& matrix)  {  matrix.input(in, cout);  return in;  }  //重载<<  friend ostream& operator<< (ostream& out, lowerTriangularMatrix& matrix)  {  matrix.output(out);  return out;  }  private:  int \*element;//存储所有元素  int size;  };  lowerTriangularMatrix.cpp  #include "LowerTriangularMatrix.h"  #include <iostream>  using namespace std;  lowerTriangularMatrix::lowerTriangularMatrix(int size)  {  if (size < 1)  exit(EXIT\_FAILURE);  element = new int[size \* (size + 1) / 2];  this->size = size;  }  void lowerTriangularMatrix::store(int row, int column,int value)  {  if (row < 1 || row < column || column < 1)  exit(EXIT\_FAILURE);  element[size\*(column - 1) - 1 + row + column \* (1 - column) / 2] = value;  }  int lowerTriangularMatrix::retrieve(int row, int column)  {  if (row < 1 || row > size || column < 1 || column > size  || (row - column) > 1 || (row - column) < -1)  exit(EXIT\_FAILURE);  return element[size\*(column - 1) - 1 + row + column \* (1 - column) / 2];  }  void lowerTriangularMatrix::input(istream& in, ostream& out)  {  out << "请依次输入下三角矩阵每行的元素，元素个数必须是" << (size \* (size + 1) / 2) << "个: "  << endl;  for (int i = 0; i < size; i++)  {  for (int j = 0; j <= i; j++)  {  int element;  in >> element;  store(i + 1, j + 1, element);  }  }  }  void lowerTriangularMatrix::output(ostream& out)  {  for (int i = 0; i < size; i++)  {  for (int j = 0; j < size; j++)  if (j > i)  out << "0\t";  else  {  out << element[size\*j + i - j \* (j + 1) / 2] << "\t";  }  out << endl;  }  }  sparseMatrix.h  #pragma once  #include "term.h"  #include <iostream>  using namespace std;  class sparseMatrix  {  public:  sparseMatrix(int maxTerms = 10);  void store(int& theVal, int theRow, int theCol);  void transpose(sparseMatrix &b);  void add(sparseMatrix &b, sparseMatrix &c);  //输入函数的声明  void input(istream& in, ostream& out);  friend istream& operator>>(istream& in, sparseMatrix& matrix)  {  matrix.input(in, cout);  return in;  }    //输出函数的声明  void output(ostream& out);  friend ostream& operator<<(ostream& out, sparseMatrix& matrix)  {  matrix.output(out);  return out;  }  private:  int rows, cols; // 矩阵维数  int terms; // 非零元素个数  Term \* a; // 映射的数组  int maxTerms; //矩阵大小  };  sparseMatrix.cpp  #include "sparseMatrix.h"  using namespace std;  sparseMatrix::sparseMatrix(int maxTerm)  {  // 构造稀疏矩阵  if (maxTerm < 1)  exit(EXIT\_FAILURE);  maxTerms = maxTerm;  a = new Term[maxTerms];  terms = cols = rows = 0;  }  void sparseMatrix::store(int& theVal, int theRow, int theCol)  {  if (theRow < 1 || theCol < 1 || theRow > rows || theCol > cols)  exit(EXIT\_FAILURE);  if (terms >= maxTerms)  exit(EXIT\_FAILURE);  a[terms].row = theRow;  a[terms].col = theCol;  a[terms].value = theVal;  terms++;  return ;  }  void sparseMatrix::output(ostream& out)  {  //输出矩阵特征  out << "行数：" << rows << " " << "列数：" << cols << endl;  out << "非零元素数：" << terms << endl;  // 输出矩阵项，一行一个  for (int i = 0; i < terms; i++)  out << "a(" << a[i].row << ',' << a[i].col << ") = " << a[i].value << endl;  }  void sparseMatrix::input(istream& in, ostream& out)  {  //输入一个稀疏矩阵  //输入矩阵特征  out << "请输入行数，列数，以及非零元素个数：" << endl;  int numberOfTerms;  in >> rows >> cols >> numberOfTerms;  //检验输入的合法性  if (numberOfTerms > maxTerms)  exit(EXIT\_FAILURE);  //输入矩阵的非零元素位置以及数值  int theRow, theCol,theVal;  for (int i = 0; i < numberOfTerms; i++)  {  out << "依次输入第" << (i + 1) << "项的所在的行、列，以及数值" << endl;  in >> theRow >> theCol >> theVal;  store(theVal, theRow, theCol);  }  }  void sparseMatrix::transpose(sparseMatrix &b)  {  // 确保b有足够大的空间  if (terms > b.maxTerms)  exit(EXIT\_FAILURE);  //设置转置矩阵特征  b.cols = rows;  b.rows = cols;  b.terms = terms;  // 初始化以实现转置  int \*colSize, \*rowNext;  colSize = new int[cols + 1];  rowNext = new int[rows + 1];  // 寻找每一列的项的数目  for (int i = 1; i <= cols; i++) // 初始化  colSize[i] = 0;  for (int i = 0; i < terms; i++)  colSize[a[i].col]++;  // 寻找b中每一行的起始点  rowNext[1] = 0;  for (int i = 2; i <= cols; i++)  rowNext[i] = rowNext[i - 1] + colSize[i - 1];  // 实施到b的转置  for (int i = 0; i < terms; i++) {  int j = rowNext[a[i].col]++; // b中的位置  b.a[j].row = a[i].col;  b.a[j].col = a[i].row;  b.a[j].value = a[i].value;  }  }  void sparseMatrix::add(sparseMatrix &b, sparseMatrix &c)  {  // 计算 c = (\*this) + b.  // 检验相容性  if (rows != b.rows || cols != b.cols)  exit(EXIT\_FAILURE);  //设置结果矩阵c的特征  c.cols = cols;  c.rows = rows;  // 重新初始化稀疏矩阵c  delete[] c.a;  int newmaxTerms = b.terms + terms;  if (newmaxTerms > c.maxTerms)  c.maxTerms = newmaxTerms;  else  c.maxTerms = c.maxTerms;  c.a = new Term[c.maxTerms];  //矩阵相加  bool\* t = new bool[b.terms];  for (int i = 0; i < b.terms; i++)  t[i] = false;  int currentIndex = 0;  for (int i = 0; i < terms; i++)  {  bool matched = false;  for (int j = 0; j < b.terms; j++)  {  if (a[i].row == b.a[j].row && a[i].col == b.a[j].col)  {  //两项在同一位置  matched = true;  t[j] = true;  c.a[currentIndex].row = a[i].row;  c.a[currentIndex].col = a[i].col;  c.a[currentIndex].value = a[i].value + b.a[j].value;//元素值相加  break;  }  }  if (!matched)  {  c.a[currentIndex].row = a[i].row;  c.a[currentIndex].col = a[i].col;  c.a[currentIndex].value = a[i].value;  }  currentIndex++;  }  for (int i = 0; i < b.terms; i++)  {  if (!t[i])  {  c.a[currentIndex].row = b.a[i].row;  c.a[currentIndex].col = b.a[i].col;  c.a[currentIndex].value = b.a[i].value;  currentIndex++;  }  }  c.terms = currentIndex;  delete[] t;  }  源.cpp  #include <iostream>  #include "sparseMatrix.h"  #include "lowerTriangularMatrix.h"  #include "tridiagonalMatrix.h"  using namespace std;  int main()  {  cout << "数据结构实验四：矩阵" << endl;  cout << endl;  //三对角矩阵  cout << "三对角矩阵 " << endl;  cout << "请输入三对角矩阵的大小(行数): "<<endl;  int size;  cin >> size;  tridiagonalMatrix\* tm = new tridiagonalMatrix(size);  cin >> (\*tm);  cout << "三对角矩阵为：" << endl;  cout << (\*tm);  cout << "请输入想取出元素的行数与列数：" << endl;  int row1, col1;  cin >> row1 >> col1;  cout << "a(" << row1 << "," << col1 << ")=" << tm->retrieve(row1, col1)<<endl;  cout << endl;    //下三角矩阵  cout << "下三角矩阵 " << endl;  cout << "请输入下三角矩阵的大小(行数): " << endl;  int size2;  cin >> size2;  lowerTriangularMatrix\* ltm = new lowerTriangularMatrix(size2);  cin >> (\*ltm);  cout << "下三角矩阵为：" << endl;  cout << (\*ltm);  cout << "请输入想取出元素的行数与列数：" << endl;  int row2, col2;  cin >> row2 >> col2;  cout << "a(" << row2 << "," << col2 << ")=" << ltm->retrieve(row2, col2) << endl;  cout << endl;  //稀疏矩阵  cout << "稀疏矩阵" << endl;  cout << endl;  //稀疏矩阵转置  cout << "稀疏矩阵转置" << endl;  cout << "请输入稀疏矩阵的最大元素个数（mn/3）: " << endl;  int maxSize1;  cin >> maxSize1;  sparseMatrix\* sm = new sparseMatrix(maxSize1);//稀疏矩阵A  cout << "请输入稀疏矩阵A" << endl;  cin >> (\*sm);  cout << "输入的稀疏矩阵为：" << endl;  cout << (\*sm);  sparseMatrix\* sm2 = new sparseMatrix(maxSize1);  sm->transpose(\*sm2);  delete sm;  sm = sm2;  cout << "转置后的稀疏矩阵为：" << endl;  cout << (\*sm);  cout << endl;  //稀疏矩阵加法  cout << "稀疏矩阵加法" << endl;  cout << "请输入稀疏矩阵的最大元素个数（mn/3）: " << endl;  int maxSize2;  cin >> maxSize2;  sparseMatrix\* sm3 = new sparseMatrix(maxSize2);//稀疏矩阵B  cout << "请输入稀疏矩阵B" << endl;  cin >> (\*sm3);  cout << "输入的稀疏矩阵为：" << endl;  cout << (\*sm3);  sparseMatrix\* sm4 = new sparseMatrix(maxSize2);//稀疏矩阵C  cout << "请输入稀疏矩阵C"<<endl;  cin >> (\*sm4);  cout << "输入的稀疏矩阵为：" << endl;  cout << (\*sm3);  sparseMatrix\* sm5 = new sparseMatrix(maxSize2 + maxSize2);  sm3->add(\*sm4, \*sm5);  delete sm3;  sm3 = sm5;  cout << "相加后的稀疏矩阵为:" << endl;  cout << (\*sm3);    system("pause");  return 0;  } | | | |
| 结论分析与体会：  运行结果：  1、    2、    3、        体会： | | | |