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：201700140056 | 姓名：李港 | | 班级：跟18.2（17.4） |
| 实验题目：实验五 数组和矩阵 | | | |
| 实验学时：2h | | 实验日期：2019.10.18 | |
| 实验目的：  掌握稀疏矩阵结构的描述及操作的实现。 | | | |
| 软件开发工具：  Virtual Studio 2019 | | | |
| 1. **实验内容** 2. 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，提供操作：  两个稀疏矩阵相加、两个稀疏矩阵相乘、输出矩阵（以通常的阵列形式输出）。 3. 键盘输入矩阵的行数、列数；按行输入矩阵的各元素值，建立矩阵； 4. 对建立的矩阵执行相加、相乘的操作，输出操作的结果矩阵。 5. **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）**   **总体思路：**   * 1. 实现默认构造、复制构造、析构功能。   2. “稀疏”的实现：采用不同数组线性表，不为0的元素加入到表中。   3. 加法：首先进行同型判断，然后进行逐元素的相加与赋值。   4. 乘法：首先进行尺寸判断，然后进行逐元素的相乘相加与赋值   5. 枚举常见错误，如下标越界，加乘时两矩阵不匹配等。   **数据结构：**   * 1. 采用之前编写的vector类作为底层数据结构   2. 使用node类存储矩阵元素所在位置的的行、列与数值   struct node {  int col;  int row;  int num;  };  **算法：**   * 1. “稀疏”的具体实现      1. void set(int r, int c, T value);         1. 下标判断         2. 若元素在元素数组中存在，则修改元素数值         3. 若元素不在元素数组中，则建立元素并赋值      2. int get(int r, int c)const;         1. 下标判断         2. 若元素在元素数组中存在，则返回其数值         3. 若元素不在元素数组中，则返回0   2. 加法      1. 判断是否同型      2. 对元素分别进行相加与赋值   3. 乘法      1. 判断能否进行相乘      2. 分别进行元素乘、加、赋值  1. **测试结果（测试输入，测试输出）**   **1.课堂检查：**    **2.平台提交**  输入：  **9**  输出：  **-1**  结果正确   1. **分析与探讨（结果分析，若存在问题，探讨解决问题的途径）**   本实验最终结果正确，在实验过程中有以下问题或心得：   * 1. 本实验对数组线性表与C++的类机制进行了更深层次的使用。   2. 在实验过程中遇到了一些问题，其中一部分是对C/C++语言本身理解的的问题，一部分是编译器相关的问题；对编程语言的使用越深入，就越容易遇到一些古怪问题，如微软实现的函数与自己编写的函数发生冲突等。   3. 在可能导致错误的地方提前进行检查与报错可以极大地方便后续调试。   4. 随着矩阵功能的不断添加，底层类vector也要增加新的接口，如reset，“=”重载等。   5. 加、乘等运算符的重载返回矩阵类常量，而重载等号运算符返回矩阵的常量引用：      1. 在加，乘中，结果的载体既不是左值也不是右值，而是临时建立的矩阵，其存活时长较短，应以值，而不是引用的形式来返回运算结果。      2. 在重载“=”运算符中，被修改的是左值，该左值从外部传入，生命周期较长，可以以引用的形式返回，以期实现连续等号赋值。   6. 被const调用的函数也需要有const属性，等号运算符两边都是const，复制构造函数接受const引用不返回值   7. 由于要兼容C，C++的各种特性都打了折扣，比如编译器明明可以多次分析源码而不必关注所谓“前向声明”问题，但是为了兼容C，C++必须“假装”自己只便利了一次源码。  1. **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）**   文件1 矩阵头文件  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* SparseMatrix.h \*  \* Copyright (C) 2019.10.11 TriAlley lg139@139.com \*  \* @brief 矩阵加乘等的实现 \*  \* @license GNU General Public License (GPL) \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include<iostream>  #include"vector.h"  using namespace std;  enum SparseMatrix\_err { not\_match, SparseMatrix\_index\_out\_of\_range };  class SparseMatrix {  private:  struct node {  int col;  int row;  int num;  };  int rows;  int cols;  vector<node> arr;  void \_checkrc(int r, int c) {  if (r > rows || c > cols ||r<1||c<1) {  throw SparseMatrix\_index\_out\_of\_range;  }  }  public:  SparseMatrix(int irows=5, int icols=5) {  rows = irows;  cols = icols;  }  /\*复制构造函数\*/  SparseMatrix(SparseMatrix& raw) {  rows = raw.getRows();  cols = raw.getCols();  arr = \*(new vector<node>);  for (int r = 0; r < rows; r++) {  for (int c = 0; c < cols; c++) {  setValue(r + 1, c + 1, raw.getValue(r + 1, c + 1));  }  }  //默认所有元素都是0  }  /\*= 运算符重载，该函数与复制构造函数同功能\*/  const SparseMatrix& operator = ( const SparseMatrix& raw ) {  rows = raw.getRows ();  cols = raw.getCols ();  arr = \*( new vector<node> );  for ( int r = 0; r < rows; r++ ) {  for ( int c = 0; c < cols; c++ ) {  setValue ( r + 1, c + 1, raw.getValue ( r + 1, c + 1 ) );  }  }  return \*this;  }  /\*设置某位置元素的值\*/  void setValue(int r, int c, int value) {  /\*检查下标是否越界\*/  \_checkrc(r, c);  if (arr.getlength() == 0) {//如果矩阵还没有非0元素  if (value == 0) {//如果要赋值0，则不进行操作，直接返回  return;  }else {//如果赋的数值不为0，则建立新结点并加入到arr中  node temp;  temp.col = c;  temp.row = r;  temp.num = value;  arr.push(temp);  return;  }  /\*如果矩阵已经有非0元素，则在arr中找到相应行、列的元素\*/  }else {  for (int i = 0; i < arr.getlength(); i++) {  if (arr[i].col == c && arr[i].row == r) {  /\*如果元素要赋的值为0，则从vector中删除该元素\*/  if (value == 0) {  arr.del(i);    /\*如果元素要赋的值不为0，则赋值\*/  }else {  arr[i].num = value;  }  return;  }  }  /\*如果数组中找不到该元素\*/  if (value == 0) {//如果元素要赋的值为0，则直接返回  return;  }else {//如果元素要赋的值不为0，则直接加入该新元素  node temp;  temp.col = c;  temp.row = r;  temp.num = value;  arr.push(temp);  return;  }  }  }  /\*输出矩阵，传入输出流的引用\*/  void show(ostream& out)const {  for (int r = 0; r < rows; r++) {  for (int c = 0; c < cols; c++) {  out << getValue(r+1, c+1) << " ";  }  out << endl;  }  out << endl;  }  /\*获取值\*/  int getValue(int r, int c)const {  /\*如果矩阵没有非0元素，则直接返回0\*/  if (arr.getlength() == 0) {  return 0;  }  /\*寻找元素并返回值\*/  for (int i = 0; i < arr.getlength(); i++) {  if (arr[i].col == c && arr[i].row == r) {  return arr[i].num;  }  }  /\*如果找不到元素，则直接返回0\*/  return 0;  }  /\*返回行数\*/  int getRows() const{return rows;}  /\*返回列数\*/  int getCols()const {return cols;}  };  /\*矩阵加法\*/  const SparseMatrix operator + (const SparseMatrix& l, const SparseMatrix& r) {  /\*首先进行同型判断\*/  if (l.getCols() != r.getCols() && l.getRows()!= r.getRows()) {  throw not\_match;  }  /\*建立临时矩阵\*/  SparseMatrix temp(l.getRows(), l.getCols());  /\*分别进行相加赋值\*/  for (int tr = 0; tr < l.getRows(); tr++) {  for (int tc = 0; tc < l.getCols(); tc++) {  temp.setValue(tr + 1, tc + 1, (l.getValue(tr + 1, tc + 1) + r.getValue(tr + 1, tc + 1)));  }  }  return temp;  }  /\*矩阵减法\*/  const SparseMatrix operator \* ( const SparseMatrix& l, const SparseMatrix& r ) {  /\*首先判断能否相乘\*/  if ( l.getCols ()!=r.getRows ()) {  throw not\_match;  }  /\*建立临时数组，拥有左值的行数与右值的列数\*/  SparseMatrix temp ( l.getRows (), r.getCols () );  /\*乘积第m行第n列的元素等于左矩阵的第m行元素与右矩阵的第n列对应元素乘积之和\*/  for ( int m = 1 ; m <= temp.getRows (); m++ ) {  for ( int n = 1; n <= temp.getCols (); n++ ) {  int value = 0;  for ( int i = 1; i <= l.getCols ();i++) {  value += l.getValue ( m, i )\* r.getValue ( i, n );  }  temp.setValue ( m, n, value );  }  }  return temp;  }  /\*输出运算符的重载，直接调用show函数\*/  ostream& operator << ( ostream& out, SparseMatrix& in ) {  in.show ( out );  return out;  }  文件2 vector头文件（对矩阵进行了适应性修改）  /\*  Author: trialley  Date: 2019-7-26  Licence: MIT  \*/  #pragma once  #include<cstring> //memset£¬memcpy  #include<cstdlib> //malloc  enum err{index\_out\_of\_range }err;  template<typename T>  class vector {  private:  T\* head; //the array head pointer  int maxindex; //the length-1 of array  int frontindex; //the last item+1  void exlength() {  T\* newhead = new T[(maxindex + 1) \* 2];  memcpy(newhead, head, sizeof(T)\*(maxindex + 1));  maxindex = maxindex \* 2 + 1;  delete[] head;  head = newhead;  }  public:  vector(vector& raw) {  maxindex = 49;  frontindex = -1;  head=new T[50];  memset(head, 0, sizeof(T) \* 50);  for (int i = 0; i < raw.getlength(); i++) {  push(raw[i]);  }  }  vector(int n = 50) {  maxindex = n - 1;  frontindex = -1;  head = new T[n];  memset(head, 0, sizeof(T) \* n);  }  ~vector() {  delete[] head;  }  /\*新增的reset函数\*/  void reset (int n=50) {  delete[] head;  maxindex = n - 1;  frontindex = -1;  head = new T[n];  memset ( head, 0, sizeof ( T ) \* n );  }  T& operator [] (int i) {  if (i > frontindex) {  throw index\_out\_of\_range;  }  return head[i];  }  T& operator [] (int i)const {  if (i > frontindex) {  throw index\_out\_of\_range;  }  return head[i];  }  void push(T in) {  if (frontindex == maxindex) {  exlength();  }  head[++frontindex] = in;  }  void del(int i) {  if (i > frontindex) {  throw index\_out\_of\_range;  }  for (int j = i; j <= frontindex; j++) {  head[j] = head[j + 1];  }  frontindex--;  }  void insert(int i, T in) {  if (i > frontindex) {  throw index\_out\_of\_range;  }  //判断后移是否越界  if (frontindex + 1 > maxindex) {  exlength();  }  //向后复制时需要总尾开始，否则第一个元素将会覆盖全部  for (int j = frontindex; j >= i; j--) {  head[j+1] = head[j];  }  head[i] = in;  frontindex++;  }  int find(T target) {  for (int i = 0; i <= frontindex; i++) {  if (head[i] == target) {  return i;  }  }  return -1;  }  int getlength() const { return frontindex + 1; }  int getmaxlength() const { return maxindex + 1; }  };  文件3 矩阵测试  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* main.cpp \*  \* Copyright (C) 2019.10.11 TriAlley lg139@139.com \*  \* @brief 矩阵加乘的测试 \*  \* @license GNU General Public License (GPL) \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include<iostream>  using namespace std;  #include"./SparseMatrix.h"  int main() {  SparseMatrix a(2,3);  SparseMatrix b(3,2);  int row = a.getRows ();  int col = a.getCols ();  for ( int r = 1; r <=row; r++ ) {  for ( int c = 1; c <= col; c++ ) {  a.setValue ( r, c, 1 );  b.setValue ( c, r, 1 );  }  }  SparseMatrix c;  c = a \* b;  cout << "矩阵乘法测试：" << endl;  cout << a << endl;  cout << b << endl;  cout << "相乘结果" << endl;  cout << c << endl;  SparseMatrix d ( 3, 4 );  SparseMatrix e ( 3, 4 );  row = d.getRows ();  col = e.getCols ();  for ( int r = 1; r <= row; r++ ) {  for ( int c = 1; c <= col; c++ ) {  d.setValue ( r, c, r \* c );  e.setValue ( r, c, r\*c );  }  }  SparseMatrix f;  f = d+e;  cout << "矩阵加法测试：" << endl;  cout << d << endl;  cout << e << endl;  cout << "相加结果" << endl;  cout << f << endl;  return 0;  } | | | |