|  |  |  |
| --- | --- | --- |
| **软件学院实验报告** | | |
| 姓名： 胡逢彬 学号： 2225060803 专业： 网络工程 年级：2022级 | | |
| 课程名称 | | 数据结构 |
| 实验名称 | | 实验6、压缩矩阵的2种转置运算 |
| 实验的准备阶段 | 实验内容 | **实验6、压缩矩阵的2种转置运算**  （1）实验目的  通过该实验，让学生理解矩阵压缩存储的概念、方法等相关知识，掌握用三元组表的方式如何进行矩阵的压缩存储，并在此基础上进行转置操作，理解转置和快速转置两种矩阵转置算法的思想。  （2）实验内容  用三元组表压缩存储矩阵，实现创建矩阵、显示以及教材中介绍的两种转置算法。  （3）参考界面  1.创建矩阵  2.销毁矩阵  3.输出矩阵  4.转置矩阵  5.快速转置矩阵  *具体要求：*请认真查看测试用例  （4）验收/测试用例   * 创建矩阵：   *注意：*检查非零元素个数是否小于等于行数乘列数；检查是否能拦截元素下标重复输入；检查是否能控制输入的非零元素的下标是递增的（即按照行序输入，先输入小的下标，再输入较大的下标）。  *注意：*输入的过程中如果有一个输入错了，不要让用户从头再把所有的输入一次，只需把刚才输入错误的，重新输入正确即可。   1. 输入：4（行数） 4（列数） 25（非零元个数），会提示：输入错误，非零元素个数要小于等于行数乘列数，请从新输入。 2. 输入：4（行数） 4（列数） 5（非零元个数） 3. 先输入：（1,1,1） （2,3,2） 4. 再输入（2,3,6），会提示：输入错误，输入的下标重复，请重新输入！ 5. 再输入（1,1,6），会提示：输入错误，输入的下标重复，请重新输入！ 6. 继续输入（3,1,3） （3,4,5） 7. 再输入（3,2,9），会提示：输入错误，下标输入时要递增输入，请重新输入！ 8. 再输入（2,3,8），会提示：输入错误，下标输入时要递增输入，请重新输入！ 9. 最后输入（4,2,4）  * 显示   屏幕上输出  1 0 0 0  0 0 2 0  3 0 0 5  0 4 0 0   * 转置   屏幕上输出  1 0 3 0  0 0 0 4  0 2 0 0  0 0 5 0 |
| 实验类型 | 验证性 |
| 实验的重点、难点 | 重点： 矩阵的转置算法  难点： 快速转置、输入时按行序非递减输入的控制 |
| 实验环境 | VC++6.0 |
| 实验的实施阶段 | 实验步骤及完成任务情况 | 一、设计思想  当设计程序框架时，可以先创建一个名为 CompressedMatrix 的类来管理压缩矩阵的操作。  1. 创建矩阵函数（\\_\_init\\_\_）:  - 初始化压缩矩阵，接收矩阵的行数、列数和元素列表，将其存储为压缩形式。  2. 销毁矩阵函数（destroy）:  - 释放矩阵所占用的内存空间。  3. 输出矩阵函数（display）:  - 将压缩矩阵按照人类可读的形式展示出来。  4. 转置矩阵函数（transpose）:  - 实现一般的压缩矩阵转置操作。  5. 快速转置矩阵函数（fast\_transpose）:  - 在进行压缩矩阵转置时使用快速算法，提高转置效率。  在程序主函数中，设计一个循环菜单结构：  1. 提供选项让用户选择操作：  - 创建矩阵  - 销毁矩阵  - 输出矩阵  - 转置矩阵  - 快速转置矩阵  用户通过输入选项进行相应操作，直到选择退出程序为止。  二、主要源代码  #include <iostream>  #include <vector>  using namespace std;  // 定义三元组表结构体  struct Triplet {  int row;  int col;  int value;  };  // 创建矩阵  //void createM(vector<Triplet>& M) {  // int rows, cols, count;  // cout << "请输入矩阵的行数、列数和非零元素个数：";  // cin >> rows >> cols >> count;  //  // Triplet tmp;  // tmp.row=rows;tmp.col=cols;tmp.value=count;  // M.push\_back(tmp);  // // 输入非零元素的行号、列号和值  // for (int i = 0; i < count; i++) {  // Triplet element;  // cout << "请输入第" << i+1 << "个非零元素的行号、列号和值：";  // cin >> element.row >> element.col >> element.value;  // M.push\_back(element);  // }  //}  // 创建矩阵  const int N=110;  int vis[N][N];  void createM(vector<Triplet>& M) {  int rows, cols, count;  cout << "请输入矩阵的行数、列数和非零元素个数："<<endl;  cin >> rows >> cols >> count;    if(count > rows\*cols)  {  cout<<"输入错误，非零元素个数要小于等于行数乘列数，请从新输入"<<endl;  return ;  }  Triplet tmp;  tmp.row=rows;tmp.col=cols;tmp.value=count;  M.push\_back(tmp);  for(int i=0;i<=rows;i++)  {  for(int j=0;j<=cols;j++)  {  vis[i][j]=0;  }  }  // 输入非零元素的行号、列号和值  for (int i = 0; i < count; i++) {  Triplet element;  cout << "请输入第" << i+1 << "个非零元素的行号、列号和值："<<endl;  cin >> element.row >> element.col >> element.value;  // element.row--; element.col--;  if(vis[element.row][element.col]==0)  {  M.push\_back(element);  vis[element.row][element.col]=1;  }  else  {  cout<<"输入错误，输入的下标重复，请重新输入！"<<endl;  i--;  }  }  }  // 显示矩阵  void displayM(const vector<Triplet>& M) {  int rows = M[0].row;  int cols = M[0].col;    // 创建二维数组，初始化为零  int \*\*arr = new int\*[rows+1];  for (int i = 0; i <=rows; i++) {  arr[i] = new int[cols+1]{0};  }    // 将非零元素填充到对应位置上  for (int i = 1; i <= M[0].value; i++) {  int row = M[i].row;  int col = M[i].col;  int value = M[i].value;  arr[row][col] = value;  }    // 输出二维数组  for (int i = 1; i<=rows; i++) {  for (int j = 1;j<=cols; j++) {  cout << arr[i][j] << " ";  }  cout << endl;  }    // 释放二维数组内存  for (int i = 0; i <=rows; i++) {  delete[] arr[i];  }  delete[] arr;  }  // 转置矩阵 - 方法一：直接转置  vector<Triplet> transposeM(const vector<Triplet>& M) {  vector<Triplet> T(M.size());  // T.push\_back(M[0]);  // 将行数、列数和非零元素个数调换  T[0].row = M[0].col;  T[0].col = M[0].row;  T[0].value = M[0].value;  int q=1;  for(int col=1;col<=M[0].col;col++)  {  for(int p=1;p<=M[0].value;p++)  {  if(M[p].col==col)  {  T[q].col=M[p].row;  T[q].row=M[p].col;  T[q].value=M[p].value;  q++;  }  }  }  return T;  }  // 转置矩阵 - 方法二：快速转置  vector<Triplet> fastTransposeM(const vector<Triplet>& M) {  vector<Triplet> T(M.size());    // 设置转置矩阵的行数、列数和非零元素个数  T[0].row = M[0].col;//T.mu=M.nu  T[0].col = M[0].row;//T.nu=T.mu;  T[0].value = M[0].value;//T.tu=M.tu;    // 建立辅助数组用于记录原矩阵每列非零元素个数  int numCols = M[0].col;  int \*numTerms = new int[numCols]{0};    // 统计每列非零元素个数  for (int i = 1; i <= M[0].value; i++) {  int col = M[i].col;  numTerms[col]++;  }    // 建立辅助数组，记录转置后每列在转置矩阵中的起始位置  int \*startPos = new int[numCols]{0};    // 计算每列起始位置  for (int i = 2; i <= numCols; i++) {  startPos[i] = startPos[i - 1] + numTerms[i - 1];  }    // 将非零元素转置  for (int i = 1; i <= M[0].value; i++) {  int col = M[i].col;  int j = startPos[col];  T[j].row = M[i].col;  T[j].col = M[i].row;  T[j].value = M[i].value;  startPos[col]++;  }    delete[] numTerms;  delete[] startPos;    return T;  }  int main() {  vector<Triplet> M;    while (true) {  int option;  cout << "请选择操作：" << endl;  cout << "1. 创建矩阵" << endl;  cout << "2. 销毁矩阵" << endl;  cout << "3. 输出矩阵" << endl;  cout << "4. 转置矩阵" << endl;  cout << "5. 快速转置矩阵" << endl;  cout << "0. 退出" << endl;  cout << "请输入选项：";  cin >> option;    switch (option) {  case 1:  createM(M);  break;  case 2:  M.clear();  break;  case 3:  displayM(M);  break;  case 4: {  vector<Triplet> T = transposeM(M);  cout << "直接转置结果：" << endl;  displayM(T);  break;  }  case 5: {  vector<Triplet> T = fastTransposeM(M);  cout << "快速转置结果：" << endl;  displayM(T);  break;  }  case 0:  return 0;  default:  cout << "无效的选项，请重新选择！" << endl;  break;  }  }    return 0;  } |
| 实验结果的处理阶段 | 实验结果 | IMG_256 |
| 实验结果总结 | 需要数组越界问题 |