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：201805130155 | 姓名： 赵雨晗 | | 班级： 18计科3班 |
| 实验题目：数组和矩阵 | | | |
| 实验学时：2 | | 实验日期： 2019.10.15 | |
| 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：  Windows : Vs Code + MingW | | | |
| 1. 实验内容   1、 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）。  2、 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；  3、 对建立的矩阵执行相加、相乘的操作，输出操作的结果矩阵   1. 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）   （1） 稀疏矩阵内部存储为线性表，线性表的元素为结构体matrixTerm，其中存储了行主映射后的下标和值。线性表中按下标递增存储保持有序。  （2） 使用成员函数get(i, j)获取第i行第j列的值，在线性表中查询时使用二分查找，如果找到则返回否则返回0。  （3） 矩阵加法为两个线性表的合并，相同下标的值对应相加。  （4） 矩阵乘法为 对a中当前非零元素，找出b中与其列数相同的元素，在答案对应位置上加上两数的乘积。  （5） 矩阵转置的方法为：根据原矩阵的下标推算出对应元素所在的行列数，然后以行为列、以列为行插入结果矩阵中。  3. 测试结果（测试输入，测试输出）     1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   一开始TLE了，必须要将算法优化才能通过此题（不能带log）   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）   #include<iostream>  #include<cstdio>  #include<cstring>  #include<algorithm>  #include<stdexcept>  using namespace std;  template <class T>  class ArrayList{  protected:  T\* \_Ele;  int Arr\_len;  int list\_size;  public:  class iterator;  iterator begin(){  return iterator(\_Ele);  }  iterator end() {  return iterator(\_Ele + Arr\_len);  }  class iterator{  protected:  T\* position;  public:  typedef bidirectional\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  iterator(T\* thePosition = NULL){position = thePosition;}  T& operator\*() const{return \*position;}  T\* operator->() const{return &\*position;}  iterator &operator++(){  ++position;return \*this;  }  iterator operator++(int){  iterator old = \*this;  ++position;  return old;  }  iterator& operator--(){  --position; return \*this;  }  iterator operator--(int){  iterator old = \*this;  --position;  return old;  }  };  T& operator[] (int a){  return \_Ele[a];  }  void operator = (const ArrayList<T> &A){  if (&A == this) return;  delete[] \_Ele;  \_Ele = new T[A.list\_size];  Arr\_len = A.Arr\_len;  list\_size = A.list\_size;  for (int i = 0; i < Arr\_len; ++i) \_Ele[i] = A.\_Ele[i];  }  ArrayList(int init\_L = 10);  ArrayList(const ArrayList<T>& );  void push\_back(const T &x);  void output();  int size(){  return Arr\_len;  }  void clear() {Arr\_len = 0;}  void ch\_sort();  void qsort(){sort(\_Ele, \_Ele + Arr\_len);}  };  template <class T>  ArrayList<T> :: ArrayList(int init\_L){  list\_size = init\_L;  \_Ele = new T[init\_L];  Arr\_len = 0;  }  template <class T>  ArrayList<T> ::ArrayList(const ArrayList<T> &t) {  list\_size = t.list\_size;  \_Ele = new T[list\_size];  Arr\_len = t.Arr\_len;  copy(t.\_Ele, t.\_Ele + t.Arr\_len, \_Ele);  }  template <class T>  void ArrayList<T> ::push\_back(const T &x) {  if (list\_size == Arr\_len) {  ArrayList<T>A(\*this);  list\_size = list\_size \* 2;  delete[] \_Ele;  \_Ele = new T[list\_size];  copy(A.\_Ele, A.\_Ele + Arr\_len, \_Ele);  }  \_Ele[Arr\_len++] = x;  }  template <class T>  void ArrayList<T> ::output() {  for (int i = 0; i < Arr\_len; ++i)cout<<\_Ele[i]<<' ';  cout<<endl;  }  template <class T>  void ArrayList<T> ::ch\_sort() {  bool sorted = false;  for (int size = Arr\_len; !sorted && (size > 1); size--){  int ind = 0;  sorted = true;  for (int i = 1; i < size; ++i)  if (\_Ele[ind] < \_Ele[i]) ind = i;  else sorted = false;  swap(\_Ele[ind], \_Ele[size - 1]);  }  }  template<typename T>  struct matrixTerm{  int index;  T value;  bool operator != (const matrixTerm<T> &x) const {return !(index == x.index && value == x.value); }  bool operator < (const matrixTerm<T> &x) const {  return index < x.index;  }  void operator = (const matrixTerm<T> &a) {  index = a.index;  value = a.value;  }  friend ostream & operator << (ostream & out, const matrixTerm &a){  out<<a.value;  return out;  }  };  template <typename T>  class SparseMatrix{  public:  SparseMatrix(int \_row, int \_col, int size) : term(size), row(\_row), col(\_col) {}  void set(){  scanf("%d%d", &row, &col);  term.clear();  int index = 1;  for (int i = 1; i <= row; i++){  for (int j = 1, v; j <= col; ++j){  scanf("%d", &v);  if (v) term.push\_back((matrixTerm<T>){index, v});  index++;  }  }  // printf("11111\n");  // term.output();  }  void change(int \_row, int \_col){  row = \_row;  col = \_col;  term.clear();  }  void operator= (const SparseMatrix<T>& a){  row = a.row, col = a.col;  term = a.term;  }  T get(int, int) const;  int get\_row() {return row;}  int get\_col() {return col;}  void insert(int, int, const T&);  void print(ostream&);  void add(int, int, T);  SparseMatrix<T> transpose() const;  SparseMatrix<T> operator+(SparseMatrix<T>&);  SparseMatrix<T> operator\*(SparseMatrix<T>&);  protected:  int row, col;  ArrayList<matrixTerm<T> > term;  };  template<typename T>  T SparseMatrix<T>::get(int \_row, int \_col) const{  if (\_row < 1 || \_row > row || \_col < 1 || \_col > col) throw out\_of\_range("the index is out of range");  int index = (\_row - 1) \* col + \_col;  int l = 0, r = term.size(), mid;  while(l < r){  mid = (l + r) / 2;  if (index > term[mid].index) l = mid + 1;  else r = mid;  }  if (l == term.size()) return 0;  else if (term[l].index == index) return term[l].value;  else return 0;  }  template<typename T>  void SparseMatrix<T>::print(ostream &out) {  int n = term.size();  int k = 0, index = 1;  out << row << ' ' << col << endl;  T value;  for (int i = 1; i <= row; ++i){  for (int j = 1; j <= col; ++j){  if (k >= n || index < term[k].index) value = 0;  else value = term[k++].value;  out << value << ' ';  index ++;  }  out << endl;  }  }  template<typename T>  SparseMatrix<T> SparseMatrix<T>::transpose() const{  SparseMatrix<T> res(col, row);  for (int i = 0; i < this -> term.size(); ++i){  int index = term[i].index;  T value = term[i].value;  int c = index % col;  int r = index / col + 1;  if (c == 0){  c = col, r--;  }  res.insert(c, r, value);  }  return res;  }  template <typename T>  void SparseMatrix<T>::insert(int \_row, int \_col, const T& value){  if (\_row < 1 || \_row > row || \_col < 1 || \_col > col) throw out\_of\_range("the index is out of range");  int index = (\_row - 1) \* col + \_col;  term.push\_back((matrixTerm<T>){index, value});  }  template<typename T>  SparseMatrix<T> SparseMatrix<T>::operator+(SparseMatrix<T> &a){  if (row != a.row || col != a.col) throw logic\_error("No matching of row and column");  SparseMatrix<T> ans(row, col, 10010);  matrixTerm<T> tmp;  int i = 0, j = 0;  while(i < term.size() && j < a.term.size()){  if (term[i].index < a.term[j].index)  tmp = term[i++];  else if (term[i].index > a.term[j].index)  tmp = a.term[j++];  else {  tmp = (matrixTerm<T>){term[i].index, term[i].value + a.term[j].value};  i++, j++;  }  ans.term.push\_back(tmp);  }  while(i < term.size()) ans.term.push\_back(term[i++]);  while(j < a.term.size()) ans.term.push\_back(a.term[j++]);  return ans;  }  template<typename T>  void SparseMatrix<T>::add(int \_row, int \_col, T value){  int n = term.size(), res = -1;  int index = (\_row - 1) \* col + \_col;  for (int i = n - 1; i >= 0; --i){  if (index == term[i].index) res = i;  // if ((term[i].index - 1) % col + 1 < \_row) break;  }  if (res == -1)  term.push\_back((matrixTerm<T>){index, value});  else term[res].value = term[res].value + value;  }  inline int calc(int r, int \_c, int c){return (r - 1) \* c + \_c;}  template<typename T>  SparseMatrix<T> SparseMatrix<T>::operator\*(SparseMatrix<T> &a) {  if (col != a.row) throw logic\_error("No matching of row and column");  SparseMatrix<T> ans(row, a.col, 10010);  T cnt[10010];  memset(cnt, 0, sizeof(cnt));  ArrayList<T> item(a.col + 1);  int pos[101];  memset(pos, -1, sizeof(pos));  for (int i = 0; i < a.term.size(); ++i){  int ar = a.term[i].index / col + 1;  int ac = a.term[i].index % a.col;  if (ac == 0) ar--;  if (pos[ar] == -1) pos[ar] = i;  }  for (int i = 0; i < term.size(); ++i){  int c = term[i].index % col;  int r = term[i].index / col + 1;  if (c == 0){  c = col;  r--;  }  if (pos[c] == -1) continue;  for (int j = pos[c]; j < a.term.size(); ++j){  int ac = a.term[j].index % a.col;  int ar = a.term[j].index / col + 1;  if (ac == 0){  ac = a.col;  ar--;  }  if (c == ar) cnt[calc(r, ac, col)] = cnt[calc(r, ac, col)] + term[i].value \* a.term[j].value;  else break;  }  }  int tot = row \* a.col;  for (int i = 1; i <= tot; ++i)  if (cnt[i] != 0)  ans.term.push\_back((matrixTerm<T>){i, cnt[i]});  return ans;  }  int main(){  int op, n;  // ArrayList<int>k(2);  // for (int i = 1; i <= 10; ++i) k.push\_back(i), k.output();  SparseMatrix<int> M(1, 1, 10010), M2(1, 1, 10010);  cin>>n;  for (int i = 1; i <= n; ++i){  scanf("%d", &op);  if (op == 1) M.set();  // M.print(cout);  else if (op == 2){  M2.set();  // M2.print(cout);  if (M.get\_col() == M2.get\_row()) M = M \* M2;  else M = M2, printf("-1\n");  } else if (op == 3){  M2.set();  if (M.get\_row() == M2.get\_row() && M.get\_col() == M2.get\_col()) M = M + M2;  else M = M2, printf("-1\n");  } else M.print(cout);  }  } | | | |