**上海电力学院**

**数据结构（JAVA）课程设计**



**题　　目:**  社会网络分析系统的设计和实现

**姓 名：**

**学 号：** 20131809

**院　　系：**　　 计算机科学与技术学院

**专业年级：**计算机科学与技术（电企方向）/2013级

1. **年 1 月 27日**

**目录**

[一、 设计题目 1](#_Toc410117375)

[二、 需求分析 1](#_Toc410117376)

[1）运行环境（软、硬件环境） 1](#_Toc410117377)

[2）输入的形式和输入值的范围 1](#_Toc410117378)

[3）输出的形式描述 2](#_Toc410117379)

[4）功能描述 2](#_Toc410117380)

[5）测试数据 2](#_Toc410117381)

[三、 概要设计 3](#_Toc410117382)

[1）抽象数据类型定义描述 3](#_Toc410117383)

[2）功能模块设计（如主程序模块设计） 4](#_Toc410117384)

[3）模块层次调用关系图 4](#_Toc410117385)

[四、 详细设计 5](#_Toc410117386)

[五、 调试分析 11](#_Toc410117387)

[1）问题和改进 11](#_Toc410117388)

[2）时间&空间复杂度 12](#_Toc410117389)

[六、 测试结果 13](#_Toc410117390)

[七、 结论 15](#_Toc410117391)

[八、 附录：程序设计源代码 16](#_Toc410117392)

## 设计题目

**【实验题目】**

社会网络分析系统的设计和实现

**【问题描述】**

实现一个社会网络分析系统。

**【基本要求】**

系统功能主要包括：

1. 对email数据进行预处理,利用数据结构课程中图中的理论，建立社会网络的邻接矩阵。
2. 利用度的概念,找出社会网络中核心人物、活跃人物和边缘人物。
3. 利用子图概念分析社会网络的结构，找出小团体和联系小团体的桥接人物。
4. 能查找任何人的交往圈子。

## 需求分析

### 1）运行环境（软、硬件环境）

软件：MyEclipse Professional 2014

硬件：能运行MyEclipse Professional 2014的硬件平台

如CPU：Intel酷睿i5；内存4G；操作系统Windows8

### 2）输入的形式和输入值的范围

数据类型：文本（txt）

范围：在txt文本的最大存储范围内

### 3）输出的形式描述

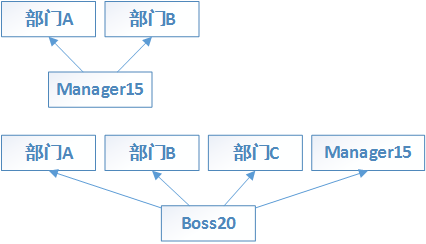
采用界面形式输出

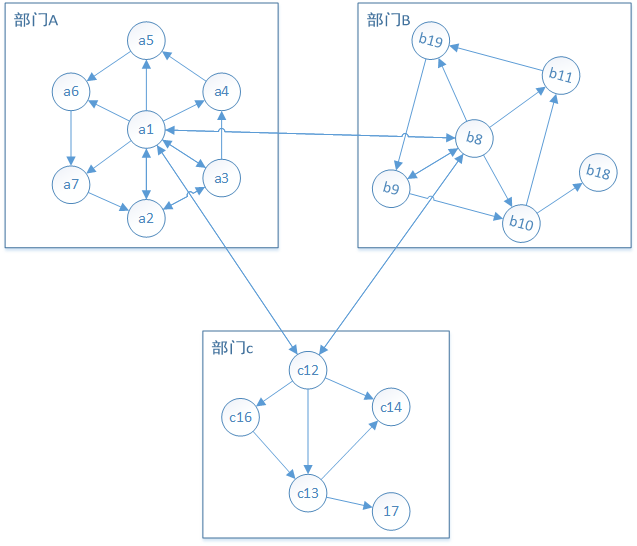
1. 该社会网络中个人的收发邮件数量
2. 该社会网络中的核心人物、活跃人物、边缘人物
3. 该社会网络中的小团体、桥接人物
4. 该社会网络中单个人的交往圈子

### 4）功能描述

1. 对email数据进行预处理,利用数据结构课程中图中的理论，建立社会网络的邻接矩阵
2. 利用度的概念,找出社会网络中核心人物、活跃人物和边缘人物
3. 利用子图概念分析社会网络的结构，找出小团体和联系小团体的桥接人物
4. 能查找任何人的交往圈子

### 5）测试数据

****

****

## 概要设计

### 1）抽象数据类型定义描述

**public** SeqList(**int** size){//数组构造函数

**public** Edge(**int** start,**int** dest,**int** weight){//边构造函数

**public** AdjMatrixGraph(**int** size){//建立邻接矩阵

**public** AdjMatrixGraph(T[] vertices,Edge[] edges){//建立邻接矩阵

**public** **int** vertexCount(){//邻接矩阵中顶点的个数

**public** T get(**int** i){//取出第i个顶点的内容

**public** String toString(){//输出邻接矩阵

**public** **int** insertVertex(T x){//插入邻接矩阵的顶点

**public** **void** insertEdge(Edge edge){//插入邻接矩阵的度

**public** **void** insertEdge(**int** i,**int** j,**int** weight){//插入邻接矩阵的度

**public** **int**[] outdegres(){//邻接矩阵中每个对象的出度总和

**public** **int**[] indegres(){//邻接矩阵中每个对象的入度总和

### 2）功能模块设计（如主程序模块设计）

**public** String activeperson(){//活跃人物

**public** String mainperson(){//核心人物

**public** String marginalperson(){//边缘人物

**public** **boolean** isExist(String[] temp,String str,**int** t){//是否存在该人

**public** String[] group\_one(String[] vertices,String str){//个人的圈子

**public** String[] Connecter(){//桥接人

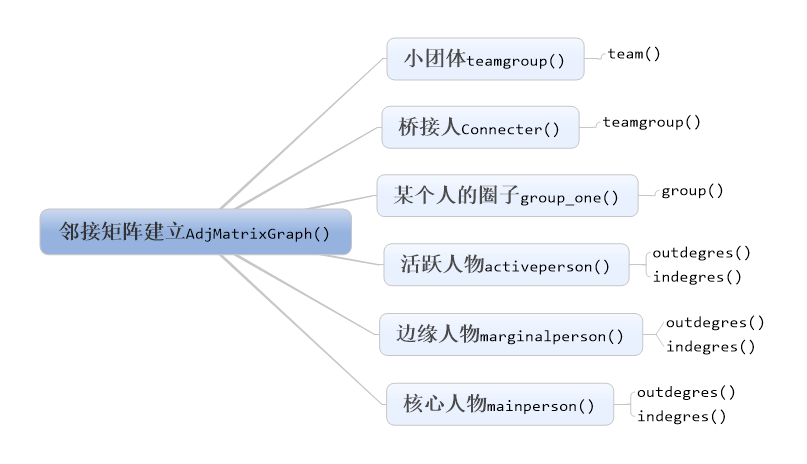
**public** String[] group(String[] vertices,String str){//圈子

**public** String[][] teamgroup(){//小团体

**public** **int**[][] team(){//团体

**public** **boolean** isConnection(**int**[][] temp,**int** i,**int** j,**int** t){//是否与其他人有联系

### 3）模块层次调用关系图

****

## 详细设计

在程序内定义一个AdjMatrixGraph的类来实现程序的基本操作。定义import\_email这个类来实现文本的读取。定义MainFrm来实现数据的输出。

**AdjMatrixGraph类:**

**protected** SeqList<T> vertexlist;

**protected** **int**[][] adjmatrix;

**protected** **int**[] visit;

**private** **final** **int** MAX\_WEIGHT=99999;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String activeperson(){//活跃人物

**int** max=-1;

**int** line=-1;

**for**(**int** i=0;i<**this**.outdegres().length;i++){

**if**(**this**.outdegres()[i]>max){

max=**this**.outdegres()[i];

line=i;

}

}

**return** (String)**this**.get(line);

}

利用AdjMatrixGraph中的某个人的出度总和函数，来计算每个人的出度总和也就是某个人发出邮件的数量，找出其中发送邮件数量最大的人作为活跃人物。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String mainperson(){//核心人物

**int**[] temp = **new** **int** [**this**.outdegres().length];

**int** max=-1;

**int** line=-1;

**for**(**int** i=0;i<**this**.outdegres().length;i++)

temp[i]=**this**.outdegres()[i]+**this**.indegres()[i];

**for**(**int** i=0;i<temp.length;i++){

**if**(temp[i]>max){

max=**this**.outdegres()[i];

line=i;

}

}

**return** (String)**this**.get(line);

}

利用AdjMatrixGraph中的某个人的出度和入度总和函数，来计算每个人的出度和入度总和也就是某个人发出和接收邮件的数量，找出其中发送和接收邮件数量最大的人作为活跃人物。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String marginalperson(){//边缘人物

**int**[] temp = **new** **int** [**this**.outdegres().length];

**int** min=99999;

**int** line=-1;

**for**(**int** i=0;i<**this**.outdegres().length;i++)

temp[i]=**this**.outdegres()[i]+**this**.indegres()[i];

**for**(**int** i=0;i<temp.length;i++){

**if**(temp[i]<min){

min=temp[i];

line=i;

}

}

**return** (String)**this**.get(line);

}

利用AdjMatrixGraph中的某个人的出度和入度总和函数，来计算每个人的出度和入度总和也就是某个人发出和接收邮件的数量，找出其中发送和接收邮件数量最小的人作为活跃人物。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **boolean** isExist(String[] temp,String str,**int** t){//是否存在该人

**for**(**int** i=0;i<t;i++){

**if**(temp[i].equals(str))

**return** **true**;

}

**return** **false**;

}

在查找某人的圈子时候，判断此人是否已经有过此人。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String[] group\_one(String[] vertices,String str){//个人的圈子

String[] init=**this**.group(vertices, str);

**int** count=0;

**while**(init[count]!=**null**)

count++;

String[] temp=**new** String[count];

**for**(**int** i=0;i<count;i++)

temp[i]=init[i];

**return** temp;

}

调用group()函数利用返回值的String[]类进行整理，截取前面的内容丢弃null。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String[] Connecter(){//桥接人

**int** count=0;

**for**(**int** i=0;i<**this**.visit.length;i++){

**if**(visit[i]>=2){

count++;

}

}

String[] connector = **new** String[count];

count=0;

**for**(**int** i=0;i<**this**.visit.length;i++){

**if**(visit[i]>=2){

connector[count]=(String)**this**.get(i);

count++;

}

}

**return** connector;

}

在访问小团体teamgroup()的时候，声明的visit[]数组来计算每个人访问的次数，在visit[]中的值大于等于2的时候此人就处于两个团体之中，也就是作为桥接人。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String[] group(String[] vertices,String str){//圈子

String[] temp = **new** String[vertices.length];

**int** location=-1;

**int** count=0;

**for**(**int** i=0;i<vertices.length;i++){

**if**(vertices[i].equals(str)){

location=i;**break**;

}

}

**for**(**int** i=0;i<vertices.length;i++)

**if**(**this**.adjmatrix[location][i]!=0 && **this**.adjmatrix[location][i]!=MAX\_WEIGHT){

temp[count]=(String)**this**.get(i);

count++;

}

**for**(**int** i=0;i<vertices.length;i++)

**if**(**this**.adjmatrix[i][location]!=0 && **this**.adjmatrix[i][location]!=MAX\_WEIGHT){

**if**(!**this**.isExist(temp, (String)**this**.get(i),count)){

temp[count]=(String)**this**.get(i);

count++;

}

}

**return** temp;

}

查找制定人在邻接矩阵中的i行j列的位置，记录位置。执行循环，第一次查找制定人所在行和列数值不为0和99999的数，记录个数，建立特定的数组。第二次，再次循环把行和列数值不为0和99999的数的特定人赋值到数组中，返回数组。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** String[][] teamgroup(){//小团体

**int**[][] temp = **this**.team();

**int** count=0;

**for**(**int** i=0;i<temp.length;i++)

**if**(!(temp[i][0]==0 || temp[i][1]==0))

count++;

String [][] group=**new** String [count][temp.length/2];

**int** t=count;count=0;

**for**(**int** i=0;i<temp.length && count<t;i++){

**if**(!(temp[i][0]==0 || temp[i][1]==0)){

**for**(**int** j=0;temp[i][j]!=0;j++){

group[count][j]=(String)**this**.get(temp[i][j]-1);

}

count++;

}

}

**return** group;

}

接收team()传回来的数组，进行处理，截断输出。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **int**[][] team(){//团体

visit = **new** **int**[**this**.vertexCount()];

**int**[][] temp = **new** **int**[**this**.vertexCount()][**this**.vertexCount()];

**for**(**int** i=0;i<**this**.vertexCount();i++){

**if**(visit[i]==0){

temp[i][0]=i+1;

visit[i]++;

**int** t=1;

**for**(**int** j=0;j<**this**.vertexCount();j++){

**if**(**this**.isConnection(temp, i, j, t)){

temp[i][t]=j+1;

visit[j]++;

t++;

}

}

}

}

**return** temp;

}

先指定一个对象，从这个对象开始，先从此对象发送过的邮件人中查找，当某一个对象拜访过后，其对应visit[]中的值加1，查找与这一个对象有联系的人，有联系的人为有过邮件上的往来的人，利用isConnection()的方法进行，如果此人定义为有联系的人，把这个人的坐标存入temp[][]的数组中，再进行下一步的查找，查找到另一个时，要确定这个人是否与temp[][]数组中的人都有过邮件上的往来，如果都有就存入temp[][]数组，没有就丢弃。这一对象查找完毕后，查找下一对象，在查找下一对象之前，先访问这一对象的visit[]数组中的值是否为0，只有当他没有访问过时，才能从这一对象开始查找。最后访问所有的对象，并返回数组。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **boolean** isConnection(**int**[][] temp,**int** i,**int** j,**int** t){//是否与其他人有联系

**int** flag=0;

**for**(**int** m=0;m<t;m++){

**if**(**this**.adjmatrix[temp[i][m]-1][j]!=0 && **this**.adjmatrix[temp[i][m]-1][j]!=MAX\_WEIGHT && **this**.adjmatrix[j][temp[i][m]-1]!=0 && **this**.adjmatrix[j][temp[i][m]-1]!=MAX\_WEIGHT){

flag++;

}

}

**if**(flag==t)

**return** **true**;

**else** **return** **false**;

}

利用传回来的temp[][]数组判断联系人与数组中的所有人是否两两存在邮件上的往来，返回boolean值。

**import\_email类：**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** Edge[] import\_email\_edges(){//从文件中读取出信息存入边的数组

**public** String[] import\_email\_vertices(){//从文件中读取出信息存入顶点的数组

**public** **boolean** isExists(String[] str,String word,**int** t){//判断这个顶点是否存在

**public** **int** vertices\_where(String[] str,String word){//查找这个值的顶点值在哪

**JFrame：**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** MainFrm(){//主界面

**public** GroupFrm(String str) {//查找小团体界面

**public** SpecialPersonFrm() {//查找特殊人界面

**public** TeamGroupFrm(){//查找桥接人界面

## 调试分析

### 1）问题和改进

**【问题&改进1】小团体与桥接人的定义**

小团体:在小团体中，每个人都要与剩余的其他人进行两两的邮件收发。

桥接：在两个或者多个小团体中都出现的人物。

**【问题&改进2】桥接人的寻找**

如果是在查找完小团体之后，在从小团体之间找出共用的桥接人，就会对所有的小团体进行排序查找算法，这样会影响算法的效率。所以我运用了visit[]数组，如果此对象经过了拜访，其对应的值就加1。最后，在查找桥接人的时候只要查找visit[]数组中拜访次数大于等于2的情况就可以了，大大节省时间。

**【问题&改进3】Boolean值函数的使用**

在使用Boolean值函数进行判断该成员是否在数组内的时候一直出现空指针的问题，最后发现是数组中的值还没有填满的情况下使用数组.length方法会发生空指针的问题，所以我在接下来的使用中，使用count数值来统计已经存放了多少数据，并在函数调用的时候传入count值作为循环结束条件。

### 2）时间&空间复杂度

**1.时间复杂度**

**public** **int**[] outdegres()--------**O(n^2)**

**public** **int**[] indegres()---------**O(n^2)**

**public** String activeperson()----**O(n)**

**public** String mainperson()------**O(n)**

**public** String marginalperson()--**O(n)**

**public** **boolean** isExist()--------**O(n)**

**public** String[] group\_one()-----**O(n)**

**public** String[] Connecter()-----**O(n)**

**public** String[] group()---------**O(n^2)**

**public** String[][] teamgroup()---**O(n)**

**public** **int**[][] team()-----------**O(n^3)**

**public** **boolean** isConnection()---**O(n)**

**2.空间复杂度**

**public** **int**[] outdegres()--------**O(n)**

**public** **int**[] indegres()---------**O(n)**

**public** String activeperson()----**O(1)**

**public** String mainperson()------**O(1)**

**public** String marginalperson()--**O(1)**

**public** **boolean** isExist()--------**O(1)**

**public** String[] group\_one()-----**O(n)**

**public** String[] Connecter()-----**O(1)**

**public** String[] group()---------**O(n)**

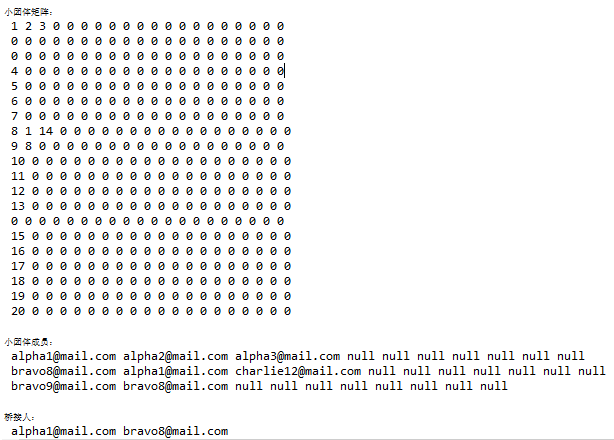
**public** String[][] teamgroup()---**O(n)**

**public** **int**[][] team()-----------**O(n)**

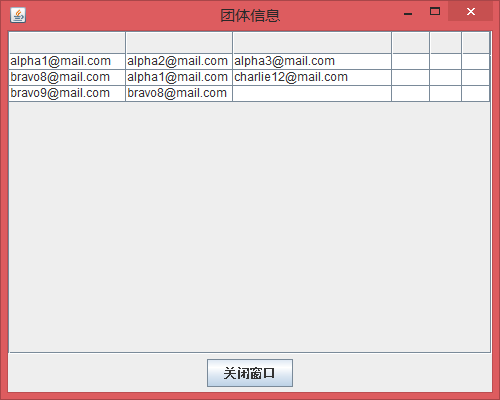
**public** **boolean** isConnection()---**O(1)**

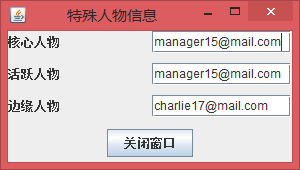
## 测试结果

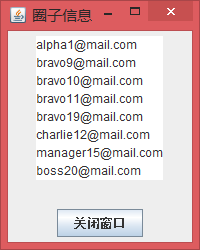


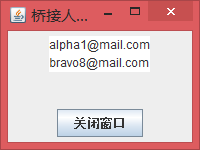












## 结论

在本次程序的编写过程中，我先对题目进行分析，也对程序的要求进行分析，确定小团体、桥接人、活跃人物、核心人物、边缘人物的定义。接着开始程序大体框架的搭建和所要运用的类进行定义。

我编写程序中，有不清楚的地方就询问老师来帮助我完成要求，对E-mail的处理操作和文件读取操作是以前没有学过的，所以我就上网找资料自主学习这个操作，并在程序中使用。在程序编写的时候我要先处理实验测试数据，在实验测速数据的处理方面先画出图，然后再进行测试数据的编写。我要接近真实的电子邮件环境，所以对测试数据的要求就大大增加了。

在以后的编写中多多使用“断点”进行编译和执行，可以大大提高编写效率。这一次的课程设计中，我收获了很多，希望在今后的编程中有所使用。

## 附录：程序设计源代码

**package** project;

**public** **class** AdjMatrixGraph<T> {

**protected** SeqList<T> vertexlist;

**protected** **int**[][] adjmatrix;

**protected** **int**[] visit;

**private** **final** **int** MAX\_WEIGHT=99999;

**public** AdjMatrixGraph(**int** size){

size=size<10?10:size;

**this**.vertexlist=**new** SeqList(size);

**this**.adjmatrix=**new** **int**[size][size];

**for**(**int** i=0;i<size;i++)

**for**(**int** j=0;j<size;j++)

**this**.adjmatrix[i][j]=(i==j)?0:MAX\_WEIGHT;

}

**public** AdjMatrixGraph(T[] vertices,Edge[] edges){

**this**(vertices.length);

**if**(vertices==**null**)

**return**;

**for**(**int** i=0;i<vertices.length;i++)

insertVertex(vertices[i]);

**if**(edges!=**null**)

**for**(**int** j=0;j<edges.length;j++)

insertEdge(edges[j]);

}

**public** **int** vertexCount(){

**return** **this**.vertexlist.length();

}

**public** **int** getWeight(**int** i,**int** j){

**return** **this**.adjmatrix[i][j];

}

**public** T get(**int** i){

**return** **this**.vertexlist.get(i);

}

**public** String toString(){

String str="顶点集合："+**this**.vertexlist.toString()+"\n 邻接矩阵 \n";

**int** n=**this**.vertexCount();

**for**(**int** i=0;i<n;i++){

**for**(**int** j=0;j<n;j++)

str+=**this**.adjmatrix[i][j]==MAX\_WEIGHT?" ∞":" "+**this**.adjmatrix[i][j];

str+="\n";

}

**return** str;

}

**public** **int** insertVertex(T x){

**this**.vertexlist.append(x);

**if**(**this**.vertexCount()>**this**.adjmatrix.length){

**int** temp[][]=adjmatrix,i,j;

**this**.adjmatrix=**new** **int**[temp.length\*2][temp.length^2];

**for**(i=0;i<temp.length;i++){

**for**(j=0;j<temp.length;j++)

**this**.adjmatrix[i][j]=temp[i][j];

**for**(j=temp.length;j<temp.length\*2;i++)

**this**.adjmatrix[i][j]=MAX\_WEIGHT;

}

**for**(i=temp.length;i<temp.length\*2;i++)

**for**(j=0;j<temp.length\*2;j++)

**this**.adjmatrix[i][j]=(i==j)?0:MAX\_WEIGHT;

}

**return** **this**.vertexlist.length()-1;

}

**public** **void** insertEdge(Edge edge){

**this**.insertEdge(edge.start,edge.dest,edge.weight);

}

**public** **void** insertEdge(**int** i,**int** j,**int** weight){

**int** n=**this**.vertexCount();

**if**(i>=0&&i<n&&j>=0&&i!=j){

**if**(**this**.adjmatrix[i][j]==MAX\_WEIGHT)

**this**.adjmatrix[i][j]=0;

**this**.adjmatrix[i][j]+=weight;

}

}

**public** **int**[] outdegres(){

**int**[] temp = **new** **int**[**this**.adjmatrix.length];

**for**(**int** i=0;i<**this**.adjmatrix.length;i++){

**for**(**int** j=0;j<**this**.adjmatrix[0].length;j++){

**if**(**this**.adjmatrix[i][j] !=MAX\_WEIGHT)

temp[i]=temp[i]+**this**.adjmatrix[i][j];

}

}

**return** temp;

}

**public** **int**[] indegres(){

**int**[] temp = **new** **int**[**this**.adjmatrix.length];

**for**(**int** i=0;i<**this**.adjmatrix[0].length;i++){

**for**(**int** j=0;j<**this**.adjmatrix.length;j++){

**if**(**this**.adjmatrix[j][i] !=MAX\_WEIGHT)

temp[i]=temp[i]+**this**.adjmatrix[j][i];

}

}

**return** temp;

}

**public** String activeperson(){

**int** max=-1;

**int** line=-1;

**for**(**int** i=0;i<**this**.outdegres().length;i++){

**if**(**this**.outdegres()[i]>max){

max=**this**.outdegres()[i];

line=i;

}

}

**return** (String)**this**.get(line);

}

**public** String mainperson(){

**int**[] temp = **new** **int** [**this**.outdegres().length];

**int** max=-1;

**int** line=-1;

**for**(**int** i=0;i<**this**.outdegres().length;i++)

temp[i]=**this**.outdegres()[i]+**this**.indegres()[i];

**for**(**int** i=0;i<temp.length;i++){

**if**(temp[i]>max){

max=**this**.outdegres()[i];

line=i;

}

}

**return** (String)**this**.get(line);

}

**public** String marginalperson(){

**int**[] temp = **new** **int** [**this**.outdegres().length];

**int** min=99999;

**int** line=-1;

**for**(**int** i=0;i<**this**.outdegres().length;i++)

temp[i]=**this**.outdegres()[i]+**this**.indegres()[i];

**for**(**int** i=0;i<temp.length;i++){

**if**(temp[i]<min){

min=temp[i];

line=i;

}

}

**return** (String)**this**.get(line);

}

**public** **boolean** isExist(String[] temp,String str,**int** t){

**for**(**int** i=0;i<t;i++){

**if**(temp[i].equals(str))

**return** **true**;

}

**return** **false**;

}

**public** String[] group\_one(String[] vertices,String str){

String[] init=**this**.group(vertices, str);

**int** count=0;

**while**(init[count]!=**null**)

count++;

String[] temp=**new** String[count];

**for**(**int** i=0;i<count;i++)

temp[i]=init[i];

**return** temp;

}

**public** String[] Connecter(){

**int** count=0;

**for**(**int** i=0;i<**this**.visit.length;i++){

**if**(visit[i]>=2){

count++;

}

}

String[] connector = **new** String[count];

count=0;

**for**(**int** i=0;i<**this**.visit.length;i++){

**if**(visit[i]>=2){

connector[count]=(String)**this**.get(i);

count++;

}

}

**return** connector;

}

**public** String[] group(String[] vertices,String str){

String[] temp = **new** String[vertices.length];

**int** location=-1;

**int** count=0;

**for**(**int** i=0;i<vertices.length;i++){

**if**(vertices[i].equals(str)){

location=i;**break**;

}

}

**for**(**int** i=0;i<vertices.length;i++)

**if**(**this**.adjmatrix[location][i]!=0 && **this**.adjmatrix[location][i]!=MAX\_WEIGHT){

temp[count]=(String)**this**.get(i);

count++;

}

**for**(**int** i=0;i<vertices.length;i++)

**if**(**this**.adjmatrix[i][location]!=0 && **this**.adjmatrix[i][location]!=MAX\_WEIGHT){

**if**(!**this**.isExist(temp, (String)**this**.get(i),count)){

temp[count]=(String)**this**.get(i);

count++;

}

}

**return** temp;

}

**public** String[][] teamgroup(){

**int**[][] temp = **this**.team();

**int** count=0;

**for**(**int** i=0;i<temp.length;i++)

**if**(!(temp[i][0]==0 || temp[i][1]==0))

count++;

String [][] group=**new** String [count][temp.length/2];

**int** t=count;count=0;

**for**(**int** i=0;i<temp.length && count<t;i++){

**if**(!(temp[i][0]==0 || temp[i][1]==0)){

**for**(**int** j=0;temp[i][j]!=0;j++){

group[count][j]=(String)**this**.get(temp[i][j]-1);

}

count++;

}

}

**return** group;

}

**public** **int**[][] team(){

visit = **new** **int**[**this**.vertexCount()];

**int**[][] temp = **new** **int**[**this**.vertexCount()][**this**.vertexCount()];

**for**(**int** i=0;i<**this**.vertexCount();i++){

**if**(visit[i]==0){

temp[i][0]=i+1;

visit[i]++;

**int** t=1;

**for**(**int** j=0;j<**this**.vertexCount();j++){

**if**(**this**.isConnection(temp, i, j, t)){

temp[i][t]=j+1;

visit[j]++;

t++;

}

}

}

}

**return** temp;

}

**public** **boolean** isConnection(**int**[][] temp,**int** i,**int** j,**int** t){

**int** flag=0;

**for**(**int** m=0;m<t;m++){

**if**(**this**.adjmatrix[temp[i][m]-1][j]!=0 && **this**.adjmatrix[temp[i][m]-1][j]!=MAX\_WEIGHT && **this**.adjmatrix[j][temp[i][m]-1]!=0 && **this**.adjmatrix[j][temp[i][m]-1]!=MAX\_WEIGHT){

flag++;

}

}

**if**(flag==t)

**return** **true**;

**else** **return** **false**;

}

}

**package** project;

**public** **class** Edge **implements** Comparable<Edge>{

**public** **int** start,dest,weight;

**public** Edge(**int** start,**int** dest,**int** weight){

**this**.start=start;

**this**.dest=dest;

**this**.weight=weight;

}

**public** String toString(){

**return** "("+start+","+dest+","+weight+")";

}

**public** **int** compareTo(Edge e) {

**if**(**this**.start!=e.start)

**return** **this**.start-e.start;

**return** **this**.dest-e.dest;

}

}

**package** project;

**public** **class** SeqList<T>{

**private** Object[] element;

**private** **int** len;

**public** SeqList(**int** size){

**this**.element=**new** Object[size];

**this**.len = 0;

}

**public** SeqList(SeqList<T> list){

**this**(list.len);

**this**.len=list.len;

}

**public** SeqList(){

**this**(64);

}

**public** **boolean** isEmpty(){

**return** **this**.len==0;

}

**public** **int** length(){

**return** **this**.len;

}

**public** T get(**int** i){

**if**(i>=0&&i<**this**.len)

**return** (T)**this**.element[i];

**return** **null**;

}

**public** **void** set(**int** i, T x){

**if**(x==**null**)

**return**;

**if**(i>=0&&i<**this**.len)

**this**.element[i] = x;

**else**

**throw** **new** IndexOutOfBoundsException(i+"");

}

**public** String toString(){

String str = "(";

**if**(**this**.len>0)

str += **this**.element[0].toString();

**for**(**int** i=1;i<**this**.len;i++)

str +=","+**this**.element[i].toString();

**return** str+")";

}

**public** **void** insert(**int** i, T x){

**if**(x==**null**)

**return**;

**if**(**this**.len==element.length){

Object[] temp = **this**.element;

**this**.element=**new** Object[temp.length\*2];

**for**(**int** j=0;j < temp.length;i++)

**this**.element[j]=temp[j];

}

**if**(i<0)

i=0;

**if**(i>**this**.len)

i=**this**.len;

**for**(**int** j=**this**.len-1;j>=i;j--)

**this**.element[j+1] = **this**.element[j];

**this**.element[i]=x;

**this**.len++;

}

**public** **void** append(T x){

insert(**this**.len,x);

}

**public** T remove(**int** i){

**if**(**this**.len==0||i<0||i>=len)

**return** **null**;

T old = (T)**this**.element[i];

**for**(**int** j=0;j<**this**.len-1;j++)

**this**.element[j] = **this**.element[j+1];

**this**.element[**this**.len-1]=**null**;

**this**.len--;

**return** old;

}

**public** **void** removeAll(){

**this**.len=0;

}

}

**package** project;

**import** java.io.\*;

**import** java.util.\*;

**public** **class** import\_email {

**public** Edge[] import\_email\_edges(){

String[] vertices = **this**.import\_email\_vertices();

String[][] temp =**new** String[999][2];

Edge[] edges = **null**;

**int** count=0;

**try**{

Scanner scanner = **new** Scanner(**new** FileInputStream("D:\\E-mail.txt"));

**while**(scanner.hasNext()){

temp[count][0] = scanner.next();

temp[count][1] = scanner.next();

count++;

}

edges=**new** Edge[count];

**for**(**int** i=0;i<count;i++)

edges[i] =**new** Edge(vertices\_where(vertices,temp[i][0]),vertices\_where(vertices,temp[i][1]),1);

}

**catch**(IOException e){

e.printStackTrace();

}

**return** edges;

}

**public** String[] import\_email\_vertices(){

String[] temp =**new** String[999];

String[] vertices=**null**;

**int** count=0;

**try**{

Scanner scanner = **new** Scanner(**new** FileInputStream("D:\\E-mail.txt"));

**while**(scanner.hasNext()){

String word = scanner.next();

**if**(!**this**.isExists(temp,word,count)){

temp[count]=word;

count++;

}

}

vertices=**new** String[count];

**for**(**int** i=0;i<count;i++){

vertices[i]=temp[i];

}

}

**catch**(IOException e){

e.printStackTrace();

}

**return** vertices;

}

**public** **boolean** isExists(String[] str,String word,**int** t){

**for**(**int** i=0;i<t;i++){

**if**(str[i].equals(word))

**return** **true**;

}

**return** **false**;

}

**public** **int** vertices\_where(String[] str,String word){

**for**(**int** i=0;i<str.length;i++){

**if**(str[i].equals(word))

**return** i;

}

**return** -1;

}

**public** **static** **void** main(String[] args){

import\_email email = **new** import\_email();

String[] vertices=email.import\_email\_vertices();

**for**(**int** i=0;i<vertices.length;i++){

System.*out*.println(vertices[i]);

}

Edge[] edges=email.import\_email\_edges();

**for**(**int** i=0;i<edges.length;i++){

System.*out*.println(edges[i].start+" "+edges[i].dest+" "+edges[i].weight);

}

}

}

**package** project;

**import** javax.swing.\*;

**import** javax.swing.table.DefaultTableModel;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** MainFrm **extends** JFrame **implements** ActionListener {

**private** JTable table = **null**;

**private** String[] cols = { "编号","发件人","发件数量","收件数量"};

**private** JButton Btn1 = **new** JButton("查询特殊人员");

**private** JButton Btn2 = **new** JButton("查询单人圈子");

**private** JButton Btn3 = **new** JButton("查询团体");

**private** JButton Btn4 = **new** JButton("桥接人");

import\_email email = **new** import\_email();

String[] vertices = email.import\_email\_vertices();

Edge[] edges=email.import\_email\_edges();

AdjMatrixGraph<String> adjMatrixGraph = **new** AdjMatrixGraph<String>(vertices,edges);

**private** **void** initTable() {

String[][] rows = **new** String[adjMatrixGraph.vertexCount()][4];

**for**(**int** i=0;i<adjMatrixGraph.vertexCount();i++){

rows[i][0]=String.*valueOf*(i+1);

rows[i][1]=vertices[i];

rows[i][2]=String.*valueOf*(adjMatrixGraph.outdegres()[i]);

rows[i][3]=String.*valueOf*(adjMatrixGraph.indegres()[i]);

}

table = **new** JTable(rows, cols);

}

**public** MainFrm(){

JPanel jp = (JPanel) **this**.getContentPane();

initTable();

JScrollPane jsp\_table = **new** JScrollPane(table);

table.getColumnModel().getColumn(0).setPreferredWidth(1);

jp.add(jsp\_table);

JPanel jp\_top = **new** JPanel();

jp\_top.add(Btn1);

jp\_top.add(Btn2);

jp\_top.add(Btn3);

jp\_top.add(Btn4);

jp.add(jp\_top, BorderLayout.*NORTH*);

Btn1.addActionListener(**this**);

Btn2.addActionListener(**this**);

Btn3.addActionListener(**this**);

Btn4.addActionListener(**this**);

**this**.setTitle("社交网络分析系统");

**this**.setSize(500, 400);

**this**.setVisible(**true**);

**this**.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

}

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource()==Btn1){

**new** SpecialPersonFrm();

}

**if**(e.getSource()==Btn2){

**int** row=table.getSelectedRow();

**if**(row>-1){

String str = String.*valueOf*(table.getValueAt(row, 1));

**new** GroupFrm(str);

}**else** JOptionPane.*showMessageDialog*(**this**, "请选择一个成员！");

}

**if**(e.getSource()==Btn3){

**new** TeamGroupFrm();

}

**if**(e.getSource()==Btn4){

**new** ConnectorFrm();

}

}

**public** **static** **void** main(String[] args){

**new** MainFrm();

}

}

**package** project;

**import** javax.swing.\*;

**import** javax.swing.table.DefaultTableModel;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** ConnectorFrm **extends** JFrame **implements** ActionListener{

import\_email email = **new** import\_email();

String[] vertices = email.import\_email\_vertices();

Edge[] edges=email.import\_email\_edges();

AdjMatrixGraph<String> adjMatrixGraph = **new** AdjMatrixGraph<String>(vertices,edges);

JButton btn = **new** JButton("关闭窗口");

JTextArea txt= **new** JTextArea();

**public** ConnectorFrm() {

String temp=**null**;

adjMatrixGraph.teamgroup();

**for**(**int** i=0;i<adjMatrixGraph.Connecter().length;i++){

**if**(temp==**null**)

temp= adjMatrixGraph.Connecter()[i];

**else** temp=temp+"\n"+adjMatrixGraph.Connecter()[i];

}

txt.setText(temp);

JPanel jp = (JPanel) **this**.getContentPane();

JPanel jp1 = **new** JPanel();

JPanel jp2 = **new** JPanel();

jp1.add(txt);

jp2.add(btn);

jp.add(jp1, BorderLayout.*NORTH*);

jp.add(jp2, BorderLayout.*SOUTH*);

btn.addActionListener(**this**);

**this**.setSize(200, 150);

**this**.setVisible(**true**);

**this**.setTitle("桥接人信息");

}

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == btn) {

dispose();

}

}

**public** **static** **void** main(String[] args){

**new** ConnectorFrm();

}

}

**package** project;

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** GroupFrm **extends** JFrame **implements** ActionListener{

import\_email email = **new** import\_email();

String[] vertices = email.import\_email\_vertices();

Edge[] edges=email.import\_email\_edges();

AdjMatrixGraph<String> adjMatrixGraph = **new** AdjMatrixGraph<String>(vertices,edges);

JButton btn\_add = **new** JButton("关闭窗口");

JTextArea txt= **new** JTextArea();

**public** GroupFrm(String str) {

String temp=**null**;

**for**(**int** i=0;i<adjMatrixGraph.group\_one(vertices, str).length;i++){

**if**(temp==**null**)

temp= adjMatrixGraph.group\_one(vertices, str)[i];

**else** temp=temp+"\n"+adjMatrixGraph.group\_one(vertices, str)[i];

}

txt.setText(temp);

JPanel jp = (JPanel) **this**.getContentPane();

JPanel jp1 = **new** JPanel();

JPanel jp2 = **new** JPanel();

jp1.add(txt);

jp2.add(btn\_add);

jp.add(jp1, BorderLayout.*NORTH*);

jp.add(jp2, BorderLayout.*SOUTH*);

btn\_add.addActionListener(**this**);

**this**.setSize(200, 250);

**this**.setVisible(**true**);

**this**.setTitle("圈子信息");

}

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == btn\_add) {

dispose();

}

}

}

**package** project;

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** SpecialPersonFrm **extends** JFrame **implements** ActionListener{

import\_email email = **new** import\_email();

String[] vertices = email.import\_email\_vertices();

Edge[] edges=email.import\_email\_edges();

AdjMatrixGraph<String> adjMatrixGraph = **new** AdjMatrixGraph<String>(vertices,edges);

JButton btn\_add = **new** JButton("关闭窗口");

JLabel lb\_mainperson = **new** JLabel("核心人物");

JLabel lb\_activeperson = **new** JLabel("活跃人物");

JLabel lb\_marginalperson = **new** JLabel("边缘人物");

JTextField txt\_mainperson = **new** JTextField();

JTextField txt\_activeperson = **new** JTextField();

JTextField txt\_marginalperson = **new** JTextField();

**public** SpecialPersonFrm() {

txt\_mainperson.setText(adjMatrixGraph.mainperson());

txt\_activeperson.setText(adjMatrixGraph.activeperson());

txt\_marginalperson.setText(adjMatrixGraph.marginalperson());

JPanel jp = (JPanel) **this**.getContentPane();

JPanel jp1 = **new** JPanel();

JPanel jp2 = **new** JPanel();

jp1.setLayout(**new** GridLayout(3, 2, 5, 10));

jp1.add(lb\_mainperson);

jp1.add(txt\_mainperson);

jp1.add(lb\_activeperson);

jp1.add(txt\_activeperson);

jp1.add(lb\_marginalperson);

jp1.add(txt\_marginalperson);

jp2.add(btn\_add);

jp.add(jp1, BorderLayout.*NORTH*);

jp.add(jp2, BorderLayout.*SOUTH*);

btn\_add.addActionListener(**this**);

**this**.setSize(300, 170);

**this**.setVisible(**true**);

**this**.setTitle("特殊人物信息");

}

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == btn\_add) {

dispose();

}

}

}

**package** project;

**import** javax.swing.\*;

**import** javax.swing.table.DefaultTableModel;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** TeamGroupFrm **extends** JFrame **implements** ActionListener {

**private** JTable table = **null**;

**private** String[] cols = {" "," "," "," "," "," "};

**private** JButton Btn = **new** JButton("关闭窗口");

import\_email email = **new** import\_email();

String[] vertices = email.import\_email\_vertices();

Edge[] edges=email.import\_email\_edges();

AdjMatrixGraph<String> adjMatrixGraph = **new** AdjMatrixGraph<String>(vertices,edges);

**private** **void** initTable() {

String[][] rows = **new** String[adjMatrixGraph.teamgroup().length][6];

**for**(**int** i=0;i<adjMatrixGraph.teamgroup().length;i++){

**for**(**int** j=0;adjMatrixGraph.teamgroup()[i][j]!=**null**;j++){

rows[i][j]=adjMatrixGraph.teamgroup()[i][j];

}

}

table = **new** JTable(rows, cols);

}

**public** TeamGroupFrm(){

JPanel jp = (JPanel) **this**.getContentPane();

initTable();

JScrollPane jsp\_table = **new** JScrollPane(table);

table.setAutoResizeMode(1);

jp.add(jsp\_table);

JPanel jp\_top = **new** JPanel();

jp\_top.add(Btn);

jp.add(jp\_top, BorderLayout.*SOUTH*);

Btn.addActionListener(**this**);

**this**.setTitle("团体信息");

**this**.setSize(500, 400);

**this**.setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource()==Btn){

dispose();

}

}

}