

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 任庆吉 |
| 学号 | 1160300422 |
| 班号 | 1603004 |
| 电子邮件 | 1091298335@qq.com |
| 手机号码 | 18846088521 |

**目录**

[1 实验目标概述 1](#_Toc508910344)

[2 实验环境配置 1](#_Toc508910345)

[3 实验过程 1](#_Toc508910346)

[3.1 待开发的四个应用场景 1](#_Toc508910347)

[3.2 面向复用的设计：Graph<L, E> 1](#_Toc508910348)

[3.3 面向复用的设计：Vertex 1](#_Toc508910349)

[3.4 面向复用的设计：Edge 1](#_Toc508910350)

[3.5 可复用API设计 2](#_Toc508910351)

[3.6 图的可视化：第三方API的复用（选做） 2](#_Toc508910352)

[3.7 设计模式应用 2](#_Toc508910353)

[3.7.1 使用State/Memento模式进行Vertex的状态管理（选做） 2](#_Toc508910354)

[3.7.2 使用factory method模式构造Vertex对象 2](#_Toc508910355)

[3.7.3 使用factory method模式构造Edge对象 2](#_Toc508910356)

[3.7.4 使用abstract factory或builder模式构造Graph对象 2](#_Toc508910357)

[3.7.5 使用Strategy模式调用centrality度量算法 2](#_Toc508910358)

[3.7.6 使用Composite模式设计超边对象（选做） 2](#_Toc508910359)

[3.7.7 使用decorator模式构造不同特征的Edge对象（选做） 2](#_Toc508910360)

[3.7.8 使用其他设计模式（选做） 2](#_Toc508910361)

[3.8 读取基于语法的文件并进行语法解析 2](#_Toc508910362)

[3.9 图操作指令的输入和处理 2](#_Toc508910363)

[3.10 应用设计与开发 3](#_Toc508910364)

[3.10.1 单词网络GraphPoet 3](#_Toc508910365)

[3.10.2 微博社交网络SocialNetwork 3](#_Toc508910366)

[3.10.3 网络拓扑图NetworkTopology 3](#_Toc508910367)

[3.10.4 电影网络MovieGraph 3](#_Toc508910368)

[4 实验进度记录 3](#_Toc508910369)

[5 实验过程中遇到的困难与解决途径 3](#_Toc508910370)

[6 实验过程中收获的经验、教训、感想 3](#_Toc508910371)

# 实验目标概述

本次实验覆盖课程第 3、 5、 6 章的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：

 子类型、泛型、多态、重写、重载

 继承、代理、组合

 常见的 OO 设计模式

 语法驱动的编程、 正则表达式

 基于状态的编程

 API 设计

本次实验给定了四个具体应用（ Lab 2 中的 GraphPoet、 Lab 1 中的SocialNetwork、网络拓扑结构 NetworkTopology、电影网络 MovieGraph），学生不是直接针对四个应用分别编程实现，而是通过 ADT 和泛型等抽象技术，开发一套可复用的 ADT 及其实现，在 Lab 2 所完成的抽象数据类型 Graph<L>的基础上，进一步扩展至 Graph<L,E>，充分考虑这些应用之间的相似性和差异性，使ADT 有更大程度的复用和更容易面向各种变化（可维护性）。

# 实验环境配置

git@github.com:ComputerScienceHIT/Lab3-1160300422.git

# 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## 待开发的四个应用场景

1. 单词网络 GraphPoet：具体参见 Lab 2 中 3.1 节的说明，节点为“单词”（ label 为该单词的文本字符串，无其他属性），边为两个单词在文本中的相邻关系，边的权重是相邻出现的次数（ 值域为正整数）。 图中可以出现loop，即一条边的起点和终点为同一个节点。 它是单重有向图、 带权图、单模图。

(2) 微博社交网络 SocialNetwork：节点为“用户”（ label 为用户姓名，其他属性包括：性别、年龄）， 边为两个用户的社交关系。两个用户之间最多可存在 3 种类型的社交关系边，分别表征“好友关系”（ A 关注了 B，边的方向为 AB）、评论关系（ A 曾评论过 B 的微博，边的方向为 AB）、转发关系（ A 曾转发过 B 的微博，边的方向为 AB）。 边的权重表示二者通过特定社交关系类型进行交互的频度，取值范围为(0,1]，图中所有边的权值之和=1。 当有新边加入图、 从图中删除已有边、调整已有边的权值时，需重新调整其他各边的权值，以满足所有边权值之和=1 的约束。图中不能出现 loop，即一个人与自己不能是好友/评论/转发关系。 该图是一个多重有向图、带权图、单模图，由此可知任意两个节点之间的边的总数最多为 6（ 3 种类型的社交关系边， 2 个方向）。

(3) 网络拓扑图 NetworkTopology：节点为“计算机”、“服务器”、“路由器”（ label 为主机名， 属性包括： IP 地址），边为它们之间的网络连接关系（但计算机之间不能直接相连、服务器之间不能直接相连），权重为网络连接的带宽（例如 1、 20、 100）。 图中不能出现 loop，即一条边的起点和终点为不能为同一个节点。 它是简单图、带权图、多模图。

(4) 电影网络 MovieGraph： 节点为“电影” （ label 为电影名，还有三个属性：上映年份、拍摄国家、 IMDb 评分）、“演员”（ label 为姓名，属性：年龄、性别）、“导演”（ label 为姓名， 属性： 年龄、性别）， 有两种无向边：演员 A 参演了电影 M、导演 D 执导了电影 M。这是单重图、多模图，演员和电影之间的边有权值（表示 A 在 M 中的角色次序，用正整数表示），导演和电影之间的边无权值。 图中不能出现 loop，即一条边的起点和终点不能为同一个节点。 参演过同一部电影的所有演员形成一条超边，超边无权值。

## 基于语法的图数据输入

以下语法（ grammar） 用于撰写特定格式的输入文件，你的程序读入该文件并使用正则表达式 parser 对其进行解析，从中抽取信息，构造图结构。

GraphType =“GraphPoet”|“SocialNetwork”|“NetworkTopology”|“MovieGraph”

图的类型，上述四种取值之一

GraphName = Label 该图的标题，字符串

VertexType = type1,…,typen

包含的节点类型列表，每个 type 的取值来自于表 3。如果是单模图，这里只

会出现一个类型。

Vertex = <Label1,type1,<attr1,...,attrk>>

...

Vertex = <Labelm,typem,<attr1,...,attrk>>

每行代表一个节点， type 为节点类型，来自于 VertexType 定义的列表。不

同节点的标签不能重复。 attr1,...,attrk 为该节点的属性值列表， 具体信息

参见表 3，属性值列表的次序与表 3 中的属性次序一致。 如果该节点无属性值，则简化为 Vertex = <Labelm,typem>。

EdgeType = type1,…,typen

图中包含的边的类型列表，每个 type 取值来自于表 5。

Edge = <Label, type, Weight, StartVertex, EndVertex, Yes|No>

...

Edge = <Label, type, Weight, StartVertex, EndVertex, Yes|No>

每行代表一条简单边， type 为边的类型，来自于 EdgeType 的列表； Yes|No表示该条边是否为有向： 若为有向，则该边为 StartVertex -> EndVertex。StartVertex 和 EndVertex 是在该行出现之前已经定义的 Vertex 的标签（Label）。 不同边（包含简单边和超边）的标签不能重复。 Weight 为边的权值（ 正数），当其值为“ -1”的时候表示它为不带权的边。

HyperEdge = <Label, Type, {Vertex1, …, Vertexn}>

...

HyperEdge = <Label, Type, {Vertex1, …, Vertexn}>

每 行 代 表 一 条 超 边 ， Type 为 出 现 在 EdgeType 列 表 中 的 类 型 ，Vertex1,…,Vertexn 表示超边包含的节点列表（ n>1），且各个 Vertex 均为之前已经定义的 Vertex 的标签。

Label：由 word character(\w，即[a-zA-Z\_0-9])构成。

利用这个要求练习了一遍正则表达式。对于每个图，大致的思路就是，读文件的每一行，先整句匹配，如果匹配的上，再分割。

如果正则表达式很简单，每一项都确定，可以用group分组获得语句的信息，对于有些情况，比如点的信息，超边点的内容，语句长度不确定，无法分组，就采取二次匹配，把不确定的信息提取出来，然后用“”匹配，取出信息。

## 面向复用的设计：Graph<L, E>

实验 2 中所设计的泛型 Graph<L>仅对节点的类型进行了参数化，这不够满

足 3.2 节中给出的多种图应用场景需要。为此你需要设计新的接口：Graph<L,E>其中 L 和 E 分别代表节点和边的类型。

该接口提供的操作如下：

 public boolean addVertex(L v) 向图中增加一个节点

对于本次实验，我详细设计了addVertex方法，这个方法的返回值，说实话，没有意义，重要的是对于client的调用都能产生正确的反馈。所以设计规约如下：

如果添加一个点v，v的label已经出现在图中，因为label是唯一的标识，所以会替换掉原来的边，即删除旧的（调用removeVertex方法，保证逻辑正确），添加新的。

否则直接添加到图中即可。

 public boolean removeVertex(L v) 从图中删除一个节点 v。如果v 是某条边的两端之一，则该边被删除；如果某节点属于某条超边，若该节点删除后该条超边仍可合法存在，则该超边继续保留，否则就删除之。

同上，只强调这个方法造成的影响，不讨论返回值。从图中找到这个点v，如果找不到就什么都不做。如果找到就删掉。然后遍历图的所有边，对于有向边和无向边，如果该边包含点v就一并删去；对于超边，如果包含点v，先确认该超边有几个顶点，如果大于2，就删掉超边中的这个点，否则直接将该超边删去。

 public Set<L> vertices() 返回图的节点集合

做一个防御式拷贝

 public Map<L, List<Double>> sources(L target) 与Lab2中Graph接口的同名操作含义相同， 返回的Map中Key为source节点，List<Double>为当前节点与该source节点之间的所有边的权值； 如果与target相连的边包括无向边，则无向边的另一端节点也需包含在返回值Map中；不需考虑超边。

没什么特别的，与实验2无异，不过要考虑多重边。

 public Map<L, List<Double>> targets(L source) 与Lab2中

Graph接口的同名操作含义相同；如果与source相连的边包括无向边，则无向边的另一端节点也需包含在返回值Map中；不需考虑超边。

 public boolean addEdge(E edge)增加一条边（包括超边）

如果edge的label在图中出现，就将旧的删除，新的添加进去。如果图中存在某一条边，类型与edge相同，起点终点也都相同，则认为这是一个修改边的操作，保持原边的label，起终点不变，修改其他不同的，比如weight；

 public boolean removeEdge(E edge) 删除一条边（包括超边）

这个本没有什么问题，但当我做movie graph时发现，如果删除掉一条MovieActorRelation，我无法找到对应的超边，并修改它。我认为这是这个实验的bug，正确的实现应该是，超边随着其他边的变化动态变化。

当然，我没有改，因为没法改，改了那个语法输入就做不了了。

在编写可运行的GUI页面时，我按照我的思路修改了，助教可以试试，动态删除一个点或边，超边会相应做出改动。

 public Set<E> edges() 返回边的集合（包括超边）

接下来需要设计一个类来实现接口 Graph<L,E>。

class ConcreteGraph implements Graph<Vertex,Edge>

在四个应用中，我分别从 ConcreteGraph 派生出更具体的、面向应用的类，分别为：

GraphPoet

SocialNetwork

NetworkTopology

MovieGraph

## 面向复用的设计：Vertex

定义 Graph<L, E>中的 L，即节点抽象类 Vertex

abstract class Vertex

Vertex 中定义各应用的共性数据表示以及作用在其上的操作，至少应覆盖以下方面：

属性：

 String label 外部可观察的标签信息

操作：

 Vertex(String label) 构造函数

 abstract void fillVertexInfo (String[] args) 为特定应用中的具体节点添加详细属性信息，所需的信息参见表 3 的最后一列，参数的次序与表 3 保持一致。 因为不同的 Vertex 子类型的属性不同，故该操作可设计为 abstract，在具体子类里实现之。

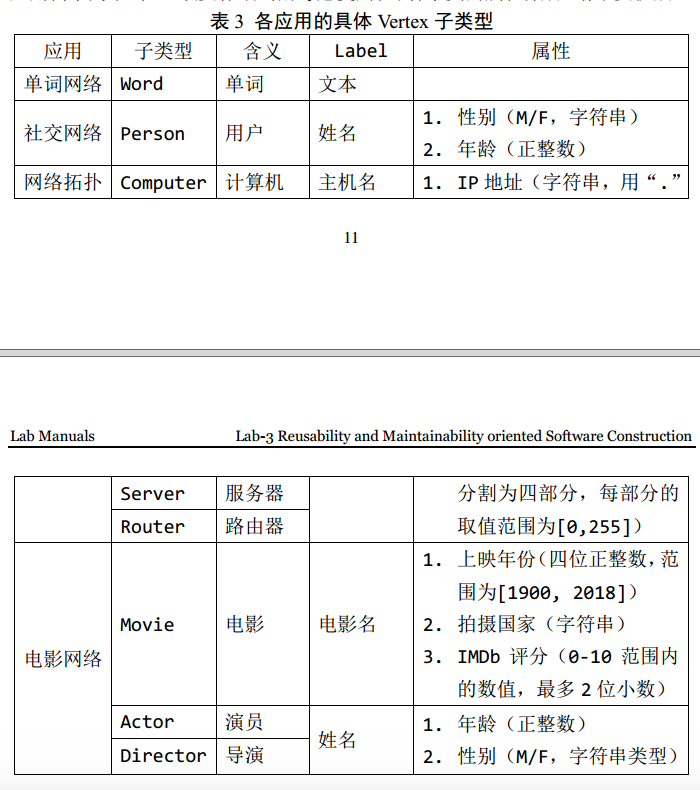
 public String getLabel() 返回节点 label 的取值

 toString() 如果针对不同子类型有不同的 string 表示，则需要在各个子类型中 override 该函数。

 equals() 类似于 toString()

 hashCode() 类似于 toString()

 其他你认为可抽象至 Vertex 的共性操作这是一个 mutable 的 ADT。



各应用具体的Vertex有其各自的特点，各自的属性，但是都具有label这一属性，所以都是从Vertex类继承来的。他们的constructor除了要继承自父类，还要调用fillVertexInfo方法对他们特有的属性进行修饰。

每个类前都写了AF，RI助教可以自行查看。

## 面向复用的设计：Edge

定义 Graph<L, E>中的 E，即代表边的类 Edge

abstract class Edge

Edge 中定义各应用的共性数据表示以及作用在其上的操作，至少应覆盖以

下方面：

属性：

 Collection vertices 该边中包含的所有节点， 其内部元素的类型应该为 Vertex。 用HashSet实现Collection，因为不重复，而且调用contains时效率较高，缺点是不能定位具体第几个点，但并没有必要。

 String label 外部可观察的标签信息，字符串

 double weight 权值， 若为带权边，则为非负数；若为无权边则该属性为-1

操作：

 Edge(String label, double weight) 构造函数

 abstract public boolean addVertices(List<Vertex> vertices)

如果是超边， vertices.size()>=2， 该函数添加 vertices 中的所有节点到该超边； 如果是有向边， vertices.size()=2，该操作将vertices 中的第一个元素作为 source，将第二个元素作为 target；如果是无向边， vertices.size()=2，无需考虑次序； 如果是 loop，vertices.size()=1。

 boolean containVertex(Vertex v) 该边中是否包含指定的点 v

写好Vertex类的equals后就很好实现。

 Set<Vertex> vertices() 边包含的点集

进行防御式拷贝

 abstract Set<Vertex> sourceVertices() 返回该边的源节点集合

 abstract Set<Vertex> targetVertices() 返回该边的目标节点集合

对于有向边我额外加了sources和targets属性，直接返回一个复制。

对于无向边返回他的点集。

 toString() 不同类型的边，其字符串表示应有不同，需表征出端点、方向、权重、 label，具体形式自行设计，在子类型中 override；

 equals() 类似于 toString()

 hashCode() 类似于 toString()

考虑到不同图应用中包含不同类型的边（有向边、无向边、超边）， 需从 Edge

派生子类型，通过 override 实现 Edge 中的各个抽象方法，也可增加子类的特有方法：

 DirectedEdge 有向边

要加两个属性

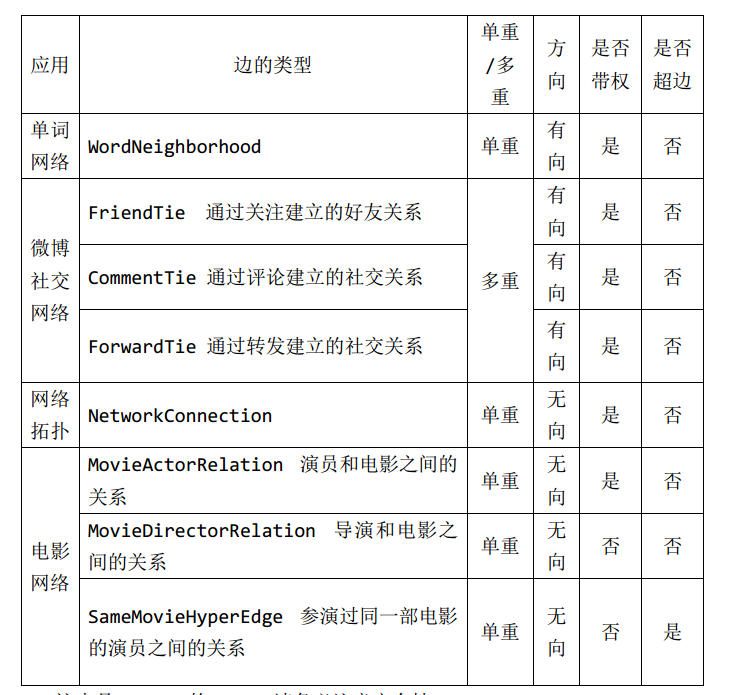
**private** Set<Vertex> sources = **new** HashSet<Vertex>();

**private** Set<Vertex> targets = **new** HashSet<Vertex>();

 UndirectedEdge 无向边

 HyperEdge 超边

在各个应用中， 在以上三个子类基础上继续派生子类， 具体信息如下表所示。



每个类的具体特点，都写在每个类的AF和RI中。不再赘述。

## 可复用API设计

基于四个应用构造的具体图结构，可在其上开展一系列关于图的计算。针对 ConcreteGraph<L,E>设计 API 并实现具体代码。 请遵循 façade 设计模式，将所有 API 放置在一个 helper 类 GraphMetrics 当中。

(1) 完成 degree centrality、 closeness centrality、 betweenness

centrality 三种中心度的度量函数 API（不考虑超边）：

 static double degreeCentrality(Graph<L,E> g, L v) 计算图 g 中节点 v 的 degree centrality

 static double degreeCentrality(Graph<L,E> g) 计算图 g 的总体 degree centrality

 static double closenessCentrality(Graph<L,E> g, L v) 计算图 g 中节点 v 的 closeness centrality

 static double betweennessCentrality(Graph<L,E> g, L v)

计算图 g 中节点 v 的 betweenness centrality

针对有向图，再增加两个度量函数 API：

 static double inDegreeCentrality(Graph<L,E> g, L v) 计

算图 g 中节点 v 的 indegree centrality

 static double outDegreeCentrality(Graph<L,E> g, L v) 计

算图 g 中节点 v 的 outdegree centrality

以上这几个数学概念从维基百科上弄懂后，就没什么了，就按照定义计算即可。有难度的是closenessCentrality和betweennessCentrality，前者需要用dijkstra计算最短路径，后者需要用修正的dijkstra计算最短路径的条数。具体算法不写在这里了。

(2) 完成以下 API （ 不考虑超边） ：

 static double distance(Graph<L, E> g, L start, L end)

节点 start 和 end 之间的最短距离（需要区分有向图和无向图）

这个功能计算centrality时要用到，所以写了一个dijkstra算法。

 static double eccentricity(Graph<L,E> g, L v)

 static double radius(Graph<L,E> g)

 static double diameter(Graph<L,E> g)

这三个概念还没学习，没写，在GUI上留出了空。有时间会写。

## 图的可视化：第三方API的复用（选做）

使用 JUNG 包中所提供的 API，为四个应用添加可视化功能。同时设计了四种可视化的操作，便于对图进行操作。具体内容在3.10应用设计与开发部分说明。

## 设计模式应用

### 使用State/Memento模式进行Vertex的状态管理（选做）

### 使用factory method模式构造Vertex对象

这部分代码在src 的package factory.VertexFactory里

所有点的工厂都继承自抽象类VertexFactory，里面有一个createVertex抽象方法

**public** **abstract** Vertex createVertex(String label, String[] args);

子类都要实现该方法，该方法包括调用相应点的constructor以及他的fillVertexInfo方法，为他独有的属性进行修饰。然后return该点。

### 使用factory method模式构造Edge对象

这部分代码在src 的package factor.EdgeFactory里

所有边的工厂都继承自抽象类EdgeFactory，抽象类中存在抽象方法

**public** **abstract** Edge createEdge(String label, List<Vertex> vertices, **double** weight);

label是边的label，vertices是组成该边的点，如果是有向边，则按照vertices中的顺序构建。weight是权值。

各子类实现抽象方法，先调用相应的constructor，然后调用addVertices方法把点添加到边中。然后return该边。

### 使用abstract factory或builder模式构造Graph对象

这部分代码在src 下的 package factory.GraphFactory里

首先创建抽象类

**public** **abstract** **class** GraphFactory {

**public** **abstract** Graph createGraph(String filePath);

}

图的工厂功能是，读一个按照规格撰写的文件，然后用之前提到的正则表达式读取信息，构建graph。

除了GraphPoet我都撰写了测试文档。对于GraphPoet，实验手册上说这个图的要求和lab2一样，所以我仍然按照lab2的要求，撰写了一个文档，让factory自己去从文档中生成GraphPoet，没用正则表达式。

### 使用Strategy模式调用centrality度量算法

根据我在课上学的strategy模式，如果对于client有几种解决问题的方法，需要调用不同的方法，进行组合时，我应该用strategy模式进行封装，便于用户调用。

比如，我现在有三个方法A，B，C，client有三个需求，X，Y，Z，实现X需求需要调用A和B，实现Y需求需要调用B和C，实现Z需求需要调用A和B，那么我可以为client封装三个不同的策略以应对三个需求。

但是，对于这个实验的这个例子，完全没有必要，只是画蛇添足，我觉得，如果想要我们锻炼上课学的各种模式，完全可以设计出更好的方案，而不是让我们强行往模式上凑，我觉得这容易让我们养成一味地用模板套用问题，而不是自己去思考，去权衡的坏习惯。

### 使用Composite模式设计超边对象（选做）

### 使用decorator模式构造不同特征的Edge对象（选做）

### 使用其他设计模式（选做）

## 图操作指令的输入和处理（选做）

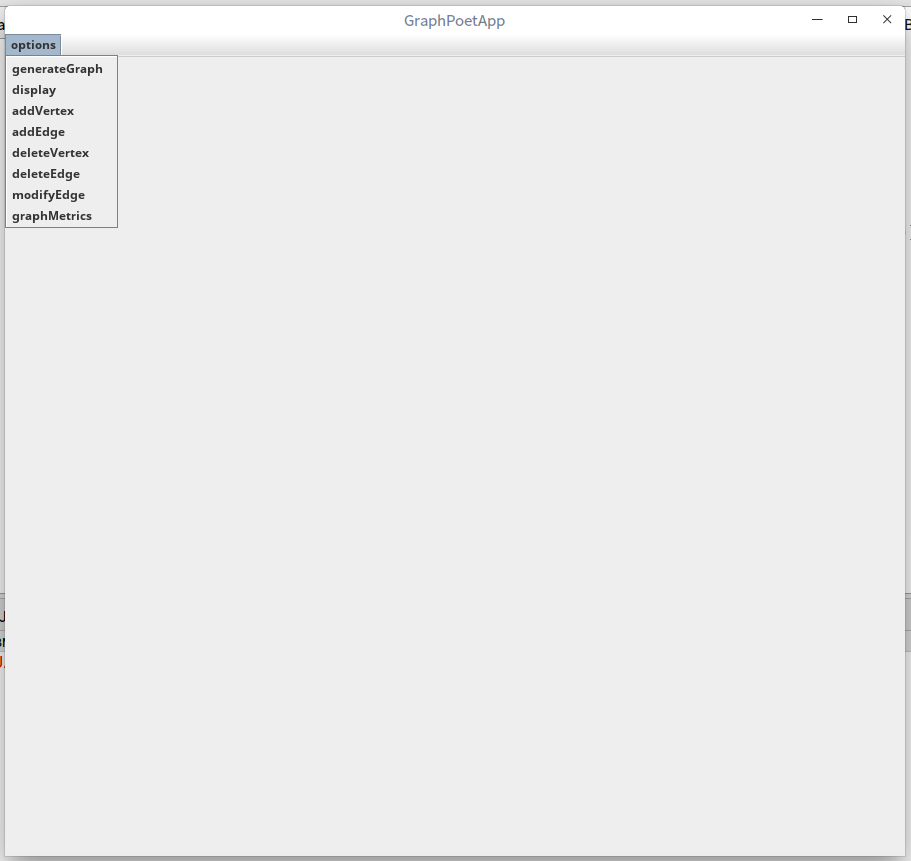
使用façade设计模式，完善ParseCommandHelper类

## 应用设计与开发

利用上述设计和实现的ADT，实现手册里要求的各项功能。对于四个图，我都设计了可视化操作，我会在下面给出使用效果。我也会把测试样例留给助教，便于检查。这部分代码在application 下的package app中。

### 单词网络GraphPoet

首先运行程序，可以看到菜单栏有一个options，我把所有操作都放在这里。



generateGraph ：从一个文件读入数据生成一个图，需要注意的是，每种图的输入文件都不尽相同，所以在读入文件时已经进行了判断，如果读入的类型与当前app不符，会终止程序。

display ：利用jung把图显示出来

addVertex ：添加一个点，实质是调用addVertex方法，所以允许添加一个存在的点

addEdge ： 添加一条边，实质是调用addEdge方法，允许添加存在的边

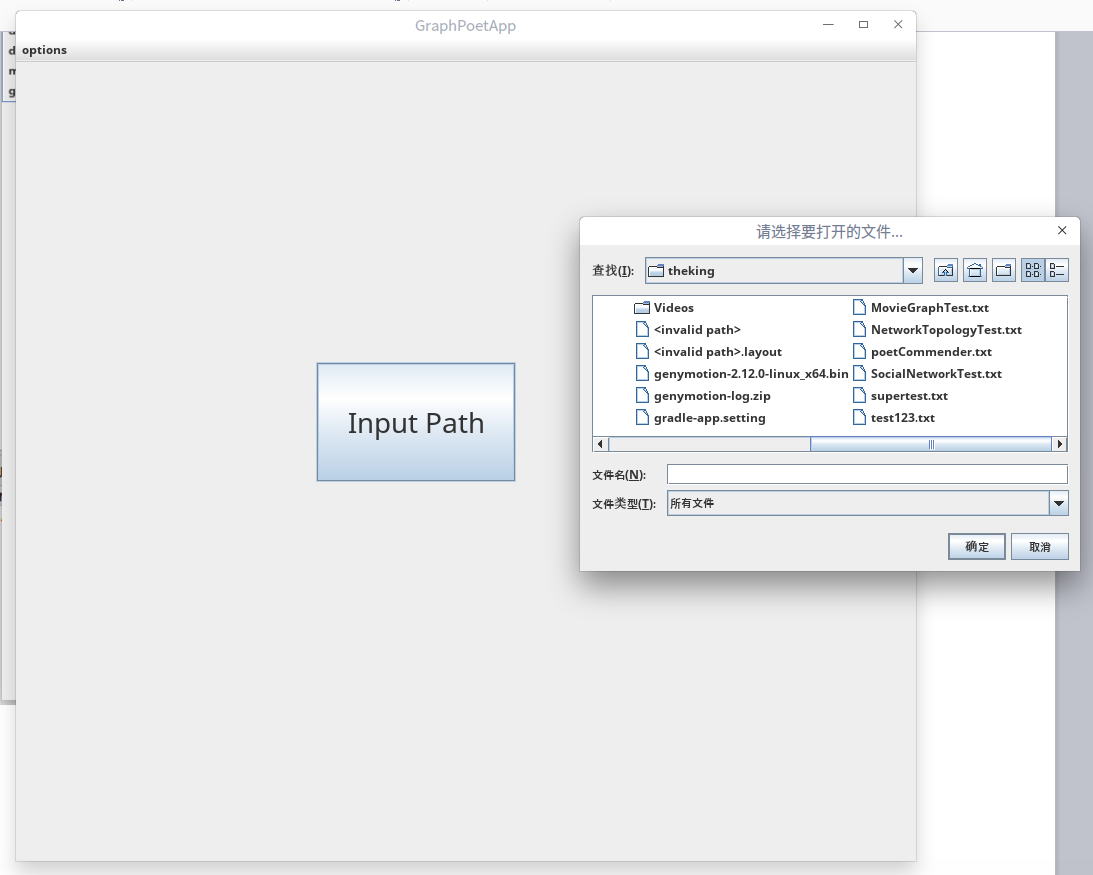
deleteVertex ：删除一个点

deleteEdge ：删除一条边

modifyEdge ：修改一条已存在的边

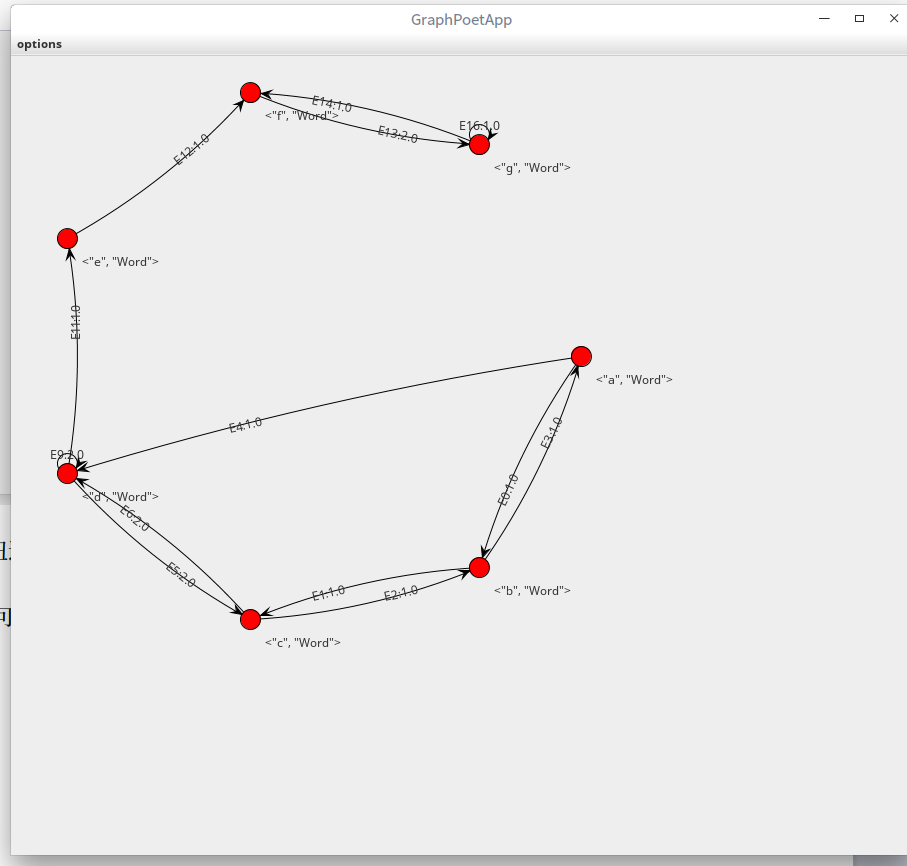
graphMetrics ：对当前图进行一系列数据的计算

generateGraph界面



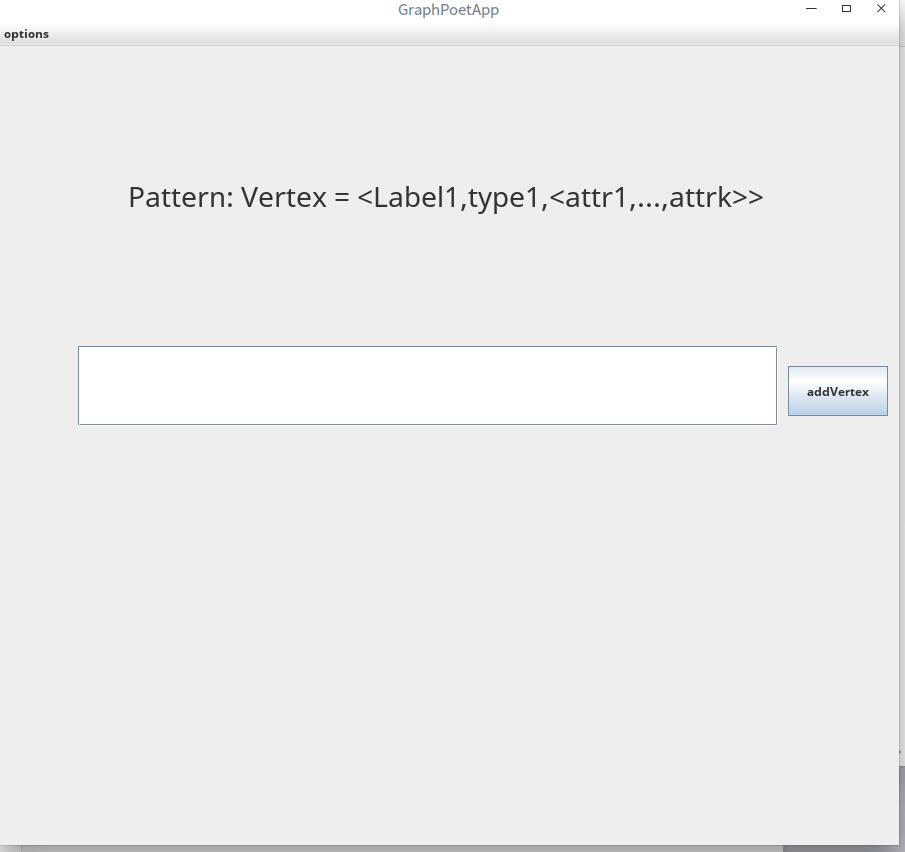
点击Input Path按钮选择路径，打开一个文件，我会把所有的测试文件放在根目录下，便于助教调试，助教也可以用自己的测试样例，但必须符合语法规范，而且图的类型吻合。

然后点击display。



可以看到图显示出来了。

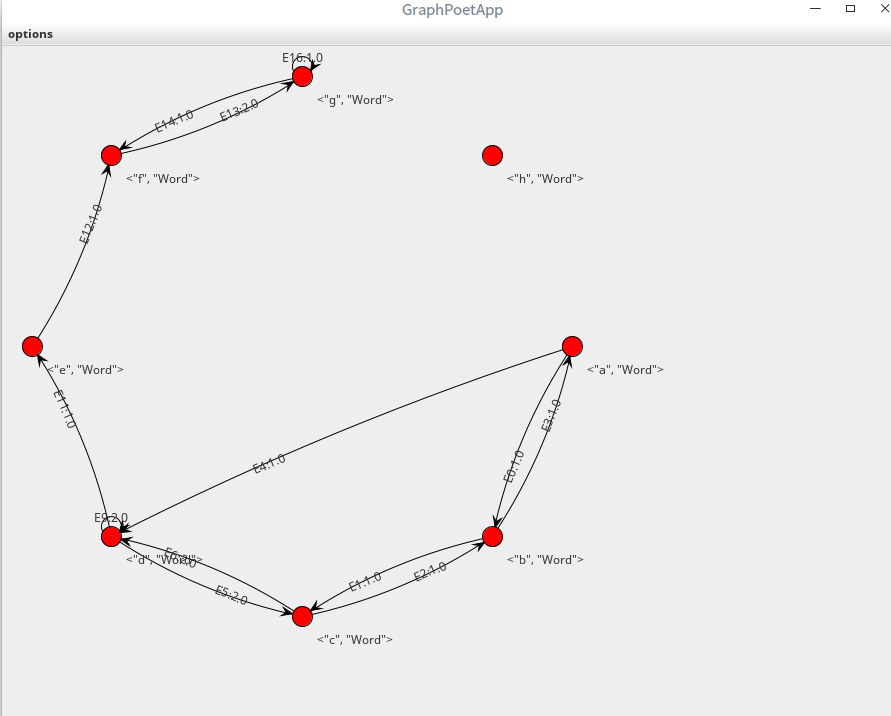
然后点击addVertex



输入 Vertex = <"h", "Word"> ，点击按钮

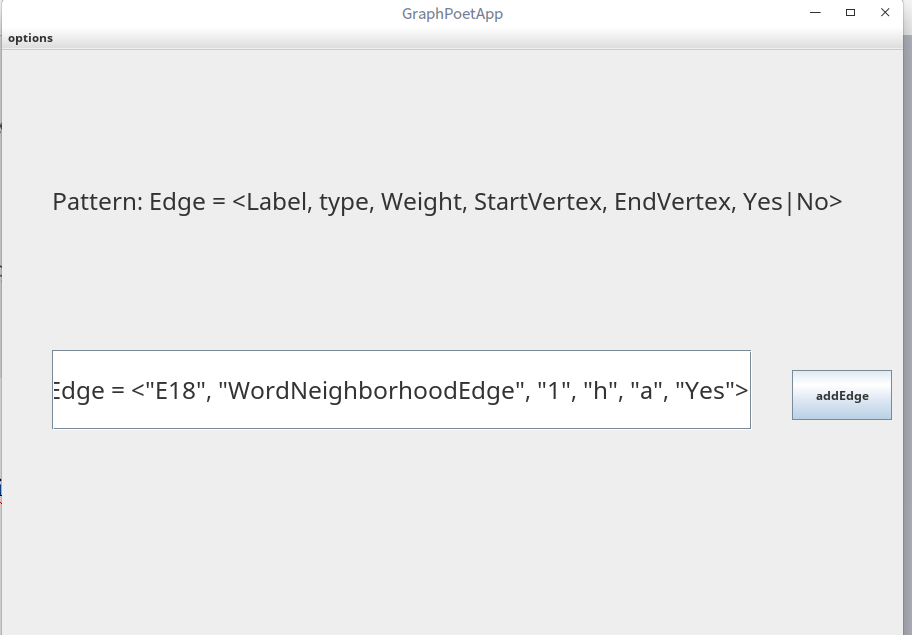
注意，这里我采用了与实验要求相同的规约，便于统一输入规范。

然后点击display可以看到点 h 确实生成了，

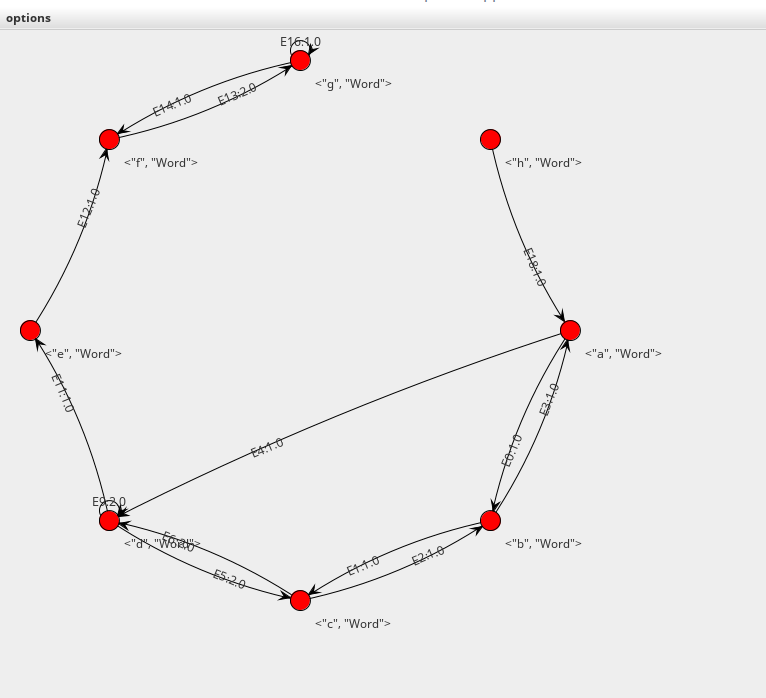


然后点击addEdge，输入

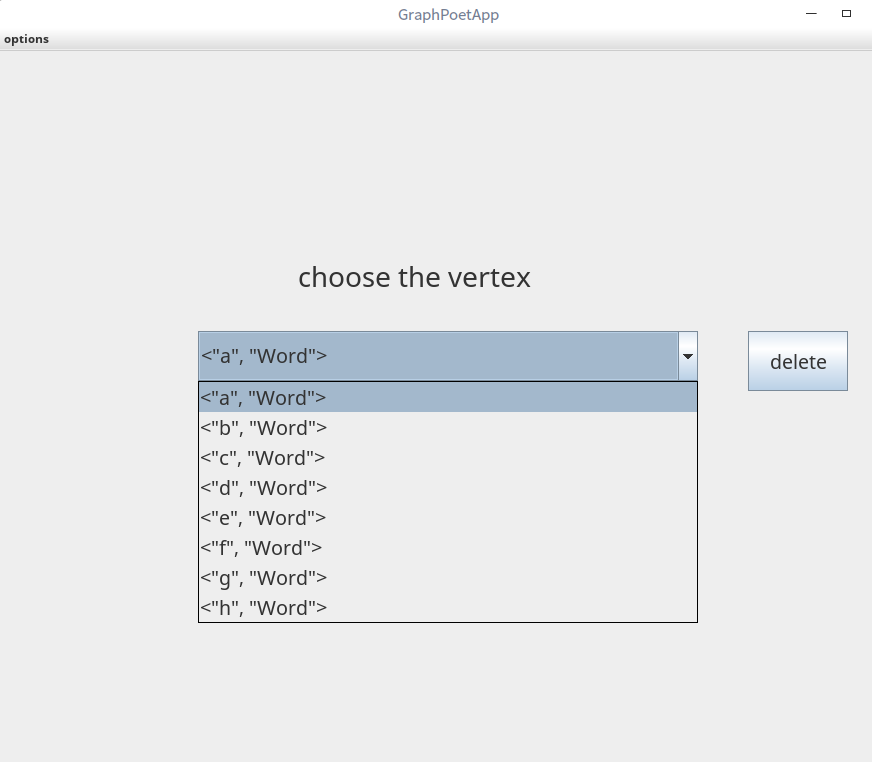
Edge = <"E18", "WordNeighborhoodEdge", "1", "h", "a", "Yes">



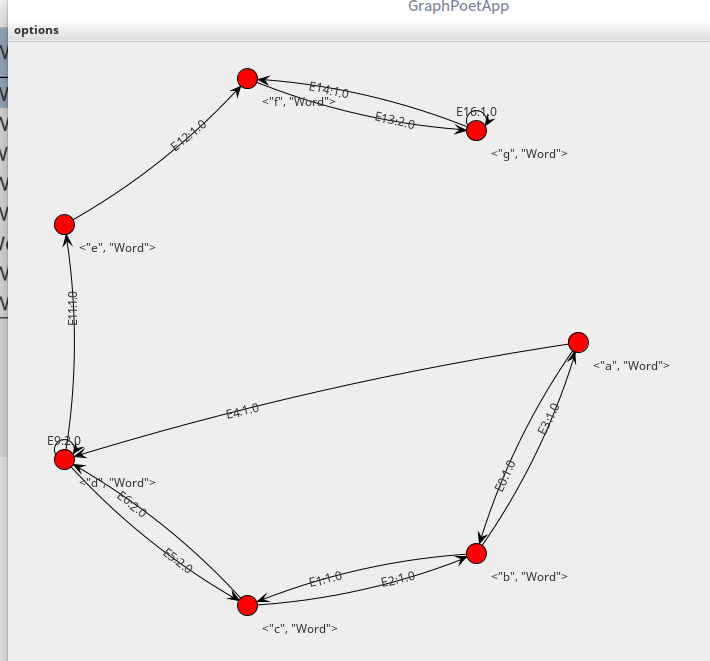
边 h→a也生成了



然后点击deleteVertex，可以看到一个下拉框，可以在这里选择点，然后删除

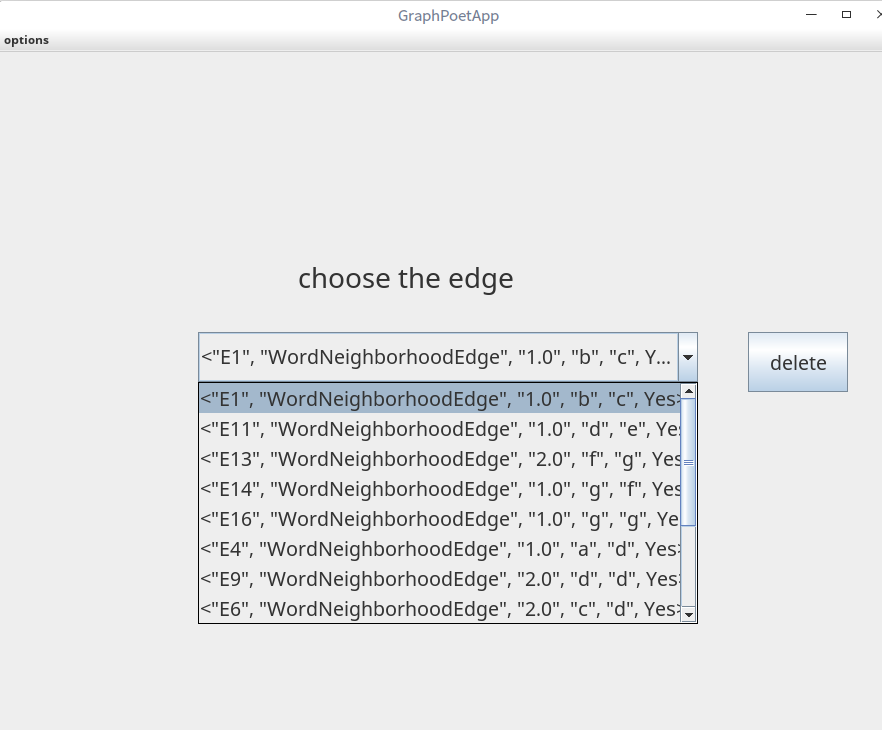


我们删除刚添加的点h

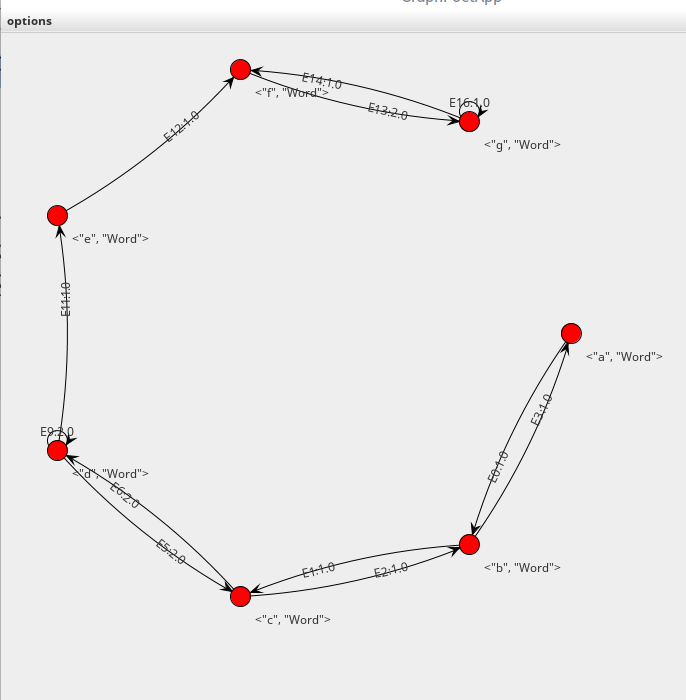


可以看到不光点h，和h相关的边h→a也不见了，说明实现了正确的功能。

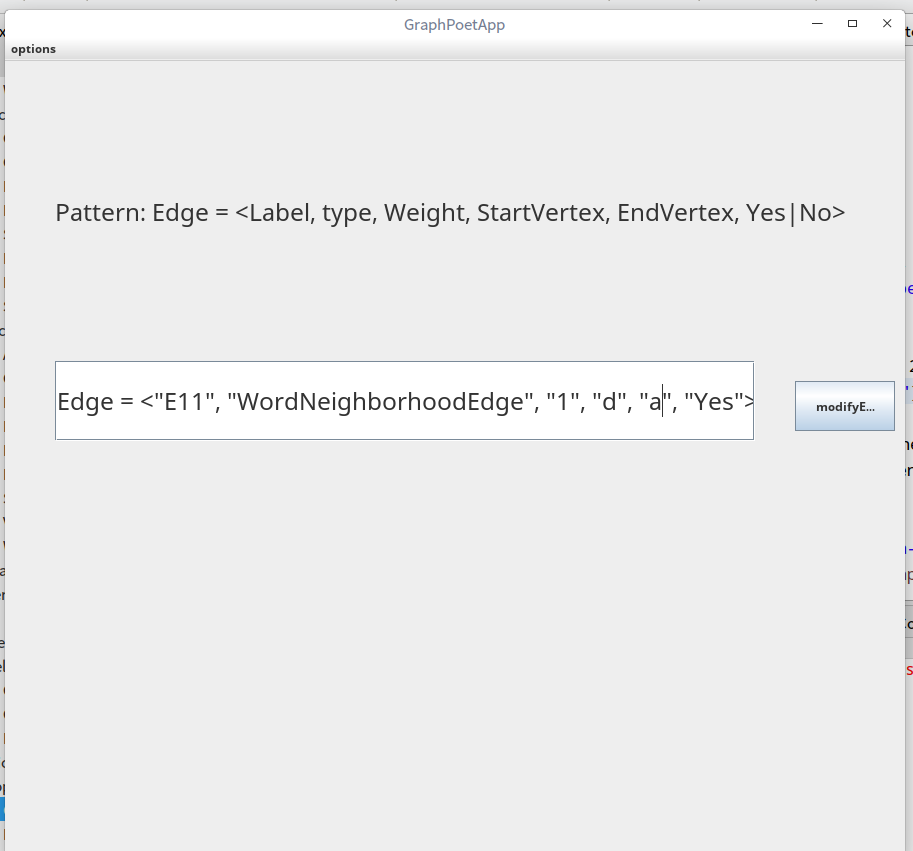
然后试验deleteEdge，与deleteVertex差不多，也是让用户用鼠标选择



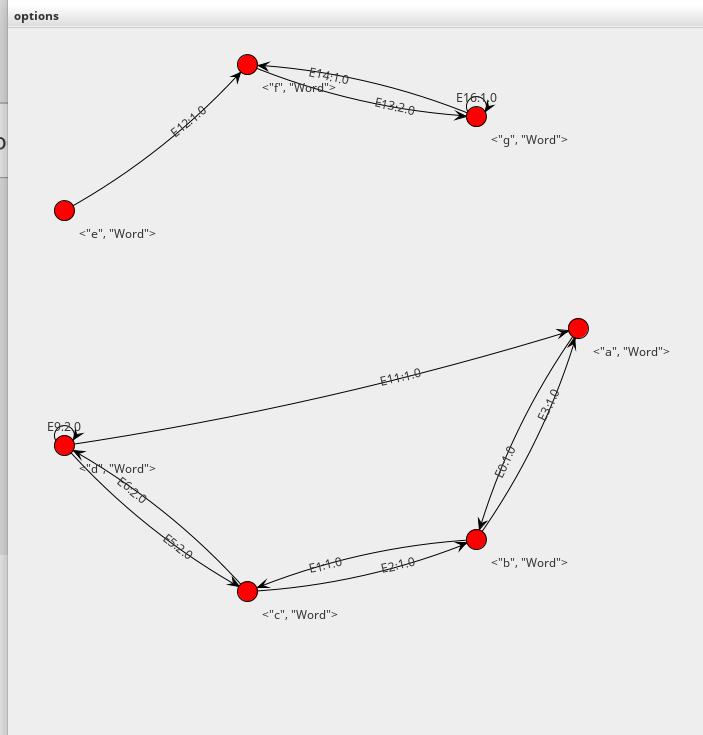
删除E4



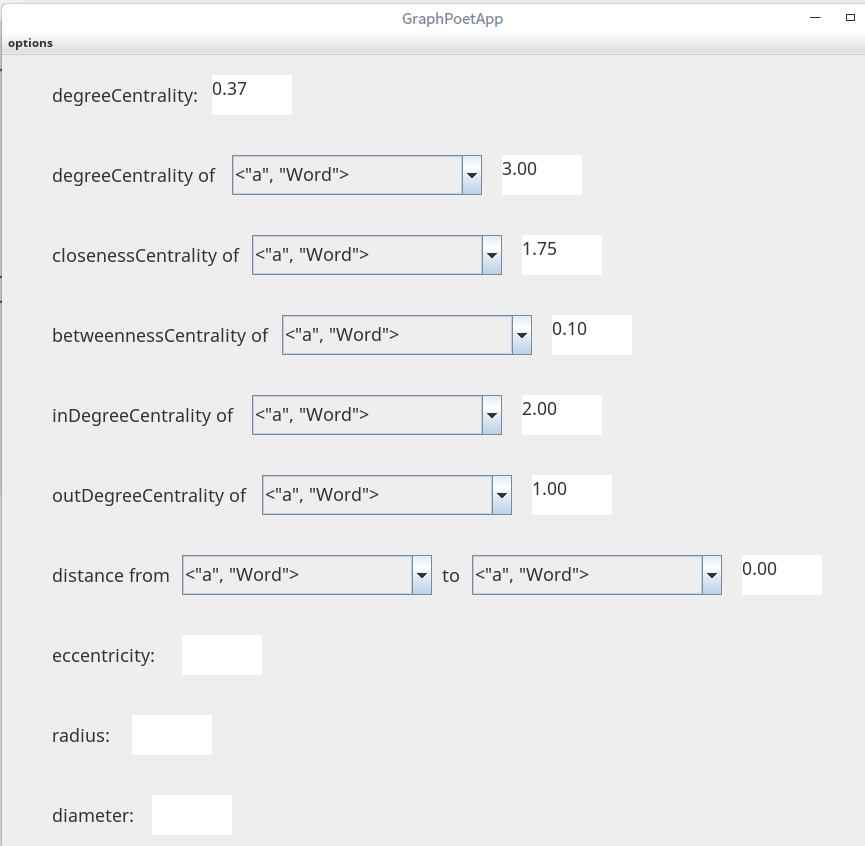
modifyEdge界面与addEdge相同



把E11从d→e变成d→a



最后是graphMetrics，

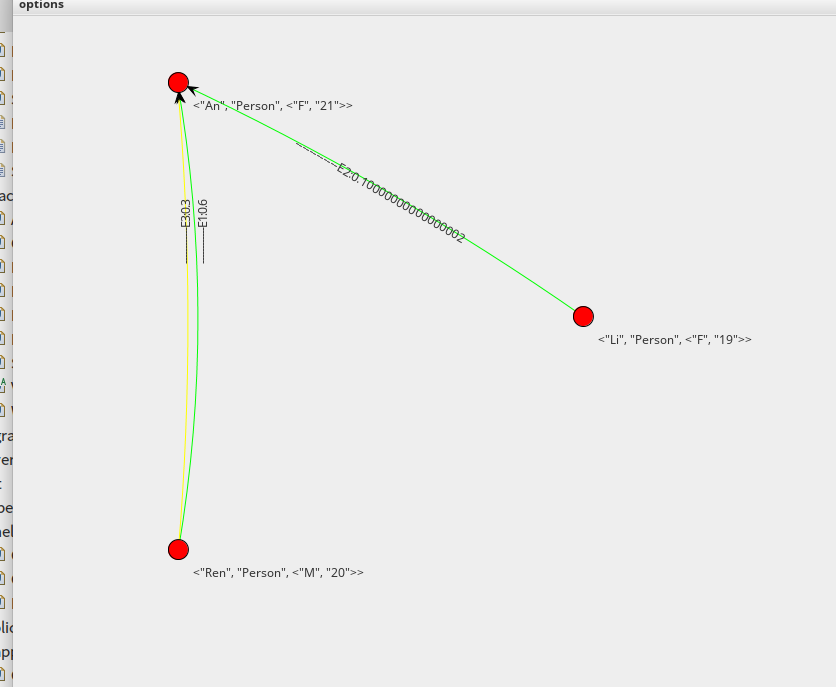


对于点的操作仍然通过鼠标完成，结果保留了两位小数。

其他图应用几乎相同，但是显示出来的风格有些不同。

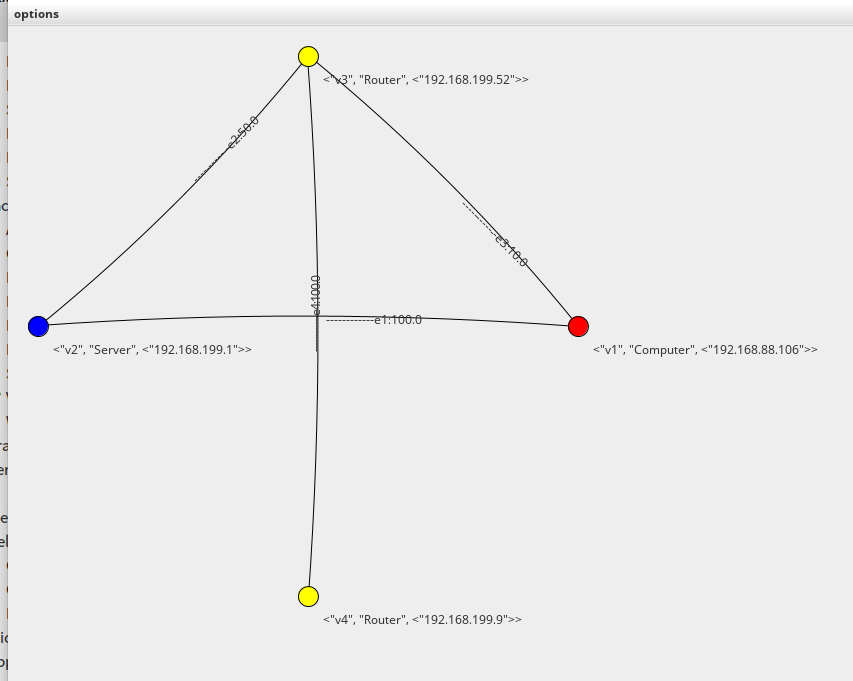
### 微博社交网络SocialNetwork

SocialNetwork用不同颜色的线表示不同关系

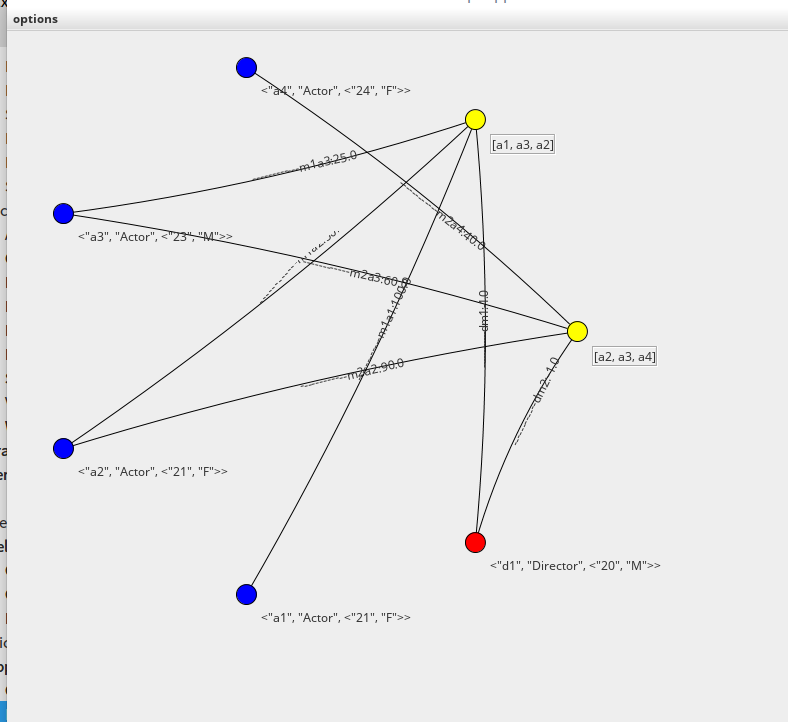


### 网络拓扑图NetworkTopology

NetworkTopology用不同颜色区分Router，Server，Computer



### 电影网络MovieGraph



MovieGraph用不同点指明Actor，Movie，Director，特别地，我按照我之前提到的解决超边问题的方法，把超边改成了根据图中MovieActorRelation生成的，我用一个文本框标注在Movie旁，表示他的超边。助教可以亲手试试，删掉一条MovieActorRelation或者删掉一个Actor，对应Movie的超边会发生变化，我觉得这样比较符合实际。

而且在这个图中存在负权边，无法计算最短路径，而且考虑到这个图的实际意义，计算中心度没什么价值（一般说来，movie的中心度最高），所以没在这个app中实现GraphMetrics的计算。

## 应对四个应用面临的新变化（任选两个）

### 单词网络GraphPoet

### 微博社交网络SocialNetwork

### 网络拓扑图NetworkTopology

### 电影网络MovieGraph

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 4.18-4.20 | 不定时 | 完成GraphPoet | 完成 |
| 4.21-4.24 | 不定时 | 完成SocialNetwork | 完成 |
| 4.28-4.29 | 不定时 | 完成NetworkTopology | 完成 |
| 4.30-5.1 | 不定时 | 完成MovieGraph | 完成 |
| 5.2-5.4 | 不定时 | 学习jung，设计GUI | 完成 |
| 5.5 | 全天 | 修改GUI，完成GraphMetrics | 完成 |

# 实验过程中遇到的困难与解决途径

在本次实验中，我遇到了很多问题，但说实话，几乎都是对实验要求产生的疑惑，经常会写着写着，发现不对劲，然后还要修改之前的代码，其中一方面是实验手册有些地方确实表述的不清楚，有歧义。但是还有很大一部分是因为自己对整个项目的进度不清楚造成的，在很长的一段时间内我都不知道要干什么，最终要做出一个什么样的东西，所以在设计一些结构时感到迷茫。其实只要知道要写什么，代码的实现都不难。就最后写GraphMetrics时用到了dijkstra算法，以及他的一个修正。其他地方没用到什么算法。

# 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

1. 重新思考Lab2中的问题：面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？本实验设计的ADT在四个图应用场景下使用，你是否体会到复用的好处？面向应用考虑的东西要更多，要考虑实际情况对ADT做相应的改变。但是如果ADT的结构设计的好，便于复用，难度就会降低很多。
2. 重新思考Lab2中的问题：为ADT撰写复杂的specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后的编程中坚持这么做？这些东西不弄清楚，就根本没法写，很重要。checkRep也能帮我找到很多bug。我愿意。
3. 之前你将别人提供的API用于自己的程序开发中，本次实验你尝试着开发给别人使用的API，是否能够体会到其中的难处和乐趣？难处是要考虑client使用时是否方便，要考虑client使用API是否方便，还是比较困难的。
4. 在编程中使用设计模式，增加了很多类，但在复用和可维护性方面带来了收益。你如何看待设计模式？设计模式的使用可以使client调用时更加清楚，也使类与类之间的结构更加清楚，如果合适，使用设计模式是一种很好的方法，但是有些画蛇添足的操作就没意义了。
5. 你之前在使用其他软件时，应该体会过输入各种命令向系统发出指令。本次实验你开发了一系列命令行指令，使用语法和正则表达式去解析它们并映射到对后台程序的调用。你对语法驱动编程有何感受？正则表达式很好用。
6. 关于本实验的工作量、难度、deadline。量是真的大，难度不大说实话，deadline长短没啥意义，你懂的。
7. 到目前为止你对《软件构造》课程的评价。

强烈希望在每次实验之后能提供一份标程让同学们欣赏。

提出一点不足，感觉不到这个实验和我们课上讲的有什么关联，感觉锻炼的针对性不强。与之对比的是上学期的计算机系统实验，每次做完实验，都对课本上的内容有不同的理解。