



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	穆添愉
学号	1160301008
班号	1603010
电子邮件	1417553133@qq.com
手机号码	15636094072

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	1
3.1 待开发的四个应用场景	1
3.2 基于语法的图数据输入	5
3.3 面向复用的设计： <code>Graph<L, E></code>	11
3.4 面向复用的设计： <code>Vertex</code>	13
3.5 面向复用的设计： <code>Edge</code>	16
3.6 可复用 API 设计	21
3.7 的可视化：第三方 API 的复用（选做）	25
3.8 设计模式应用	30
3.8.1 使用 State/Memento 模式进行 <code>Vertex</code> 的状态管理（选做）	30
3.8.2 使用 factory method 模式构造 <code>Vertex</code> 对象	30
3.8.3 使用 factory method 模式构造 <code>Edge</code> 对象	31
3.8.4 使用 abstract factory 或 builder 模式构造 <code>Graph</code> 对象	32
3.8.5 使用 Strategy 模式调用 centrality 度量算法	33
3.8.6 使用 Composite 模式设计超边对象（选做）	34
3.8.7 使用 decorator 模式构造不同特征的 <code>Edge</code> 对象（选做）	34
3.8.8 使用其他设计模式（选做）	35
3.9 图操作指令的输入和处理（选做）	35
3.10 应用设计与开发	36
3.10.1 单词网络 GraphPoet	40
3.10.2 微博社交网络 SocialNetwork	41
3.10.3 网络拓扑图 NetworkTopology	42
3.10.4 电影网络 MovieGraph	42
3.11 应对四个应用面临的新变化（任选两个）	43
3.11.1 单词网络 GraphPoet	43
3.11.2 微博社交网络 SocialNetwork	43
3.11.3 网络拓扑图 NetworkTopology	44

3.11.4 电影网络 MovieGraph	44
4 实验进度记录	44
5 实验过程中遇到的困难与解决途径	45
6 实验过程中收获的经验、教训、感想	45

1 实验目标概述

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：

1. 子类型、泛型、多态、重写、重载
2. 继承、代理、组合
3. 常见的 OO 设计模式
4. 语法驱动的编程、正则表达式
5. 基于状态的编程
6. API 设计

2 实验环境配置

<https://github.com/ComputerScienceHIT/Lab3-1160301008>

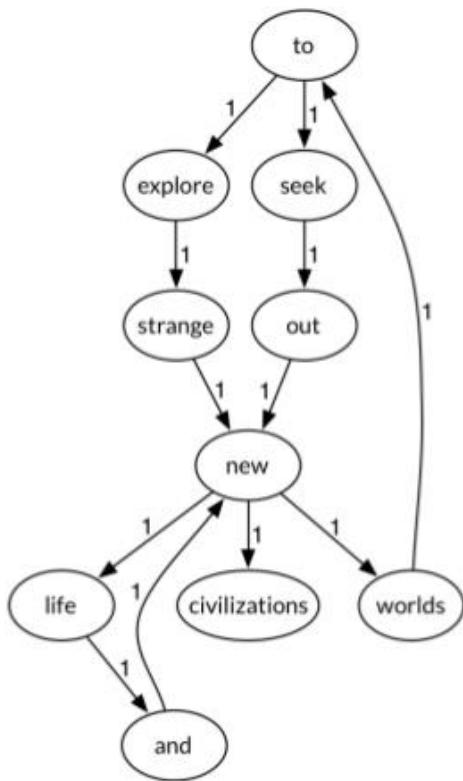
3 实验过程

3.1 待开发的四个应用场景

1. 单词网络(GraphPoet)

节点为”单词”，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)。属性为 label。

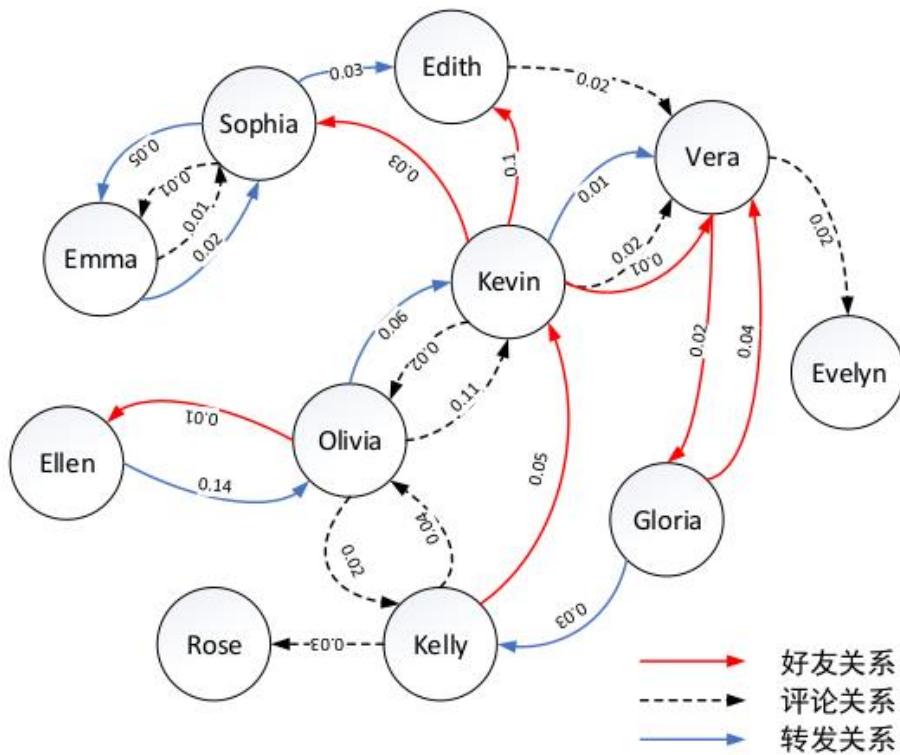
具体如下图所示：



2. 微博社交网络 (SocialNetwork)

节点为“用户”，label 为姓名，属性包含年龄、性别，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)。图的总权重为 1, 所以每条边的权重范围是(0,1]，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。

具体如下图：



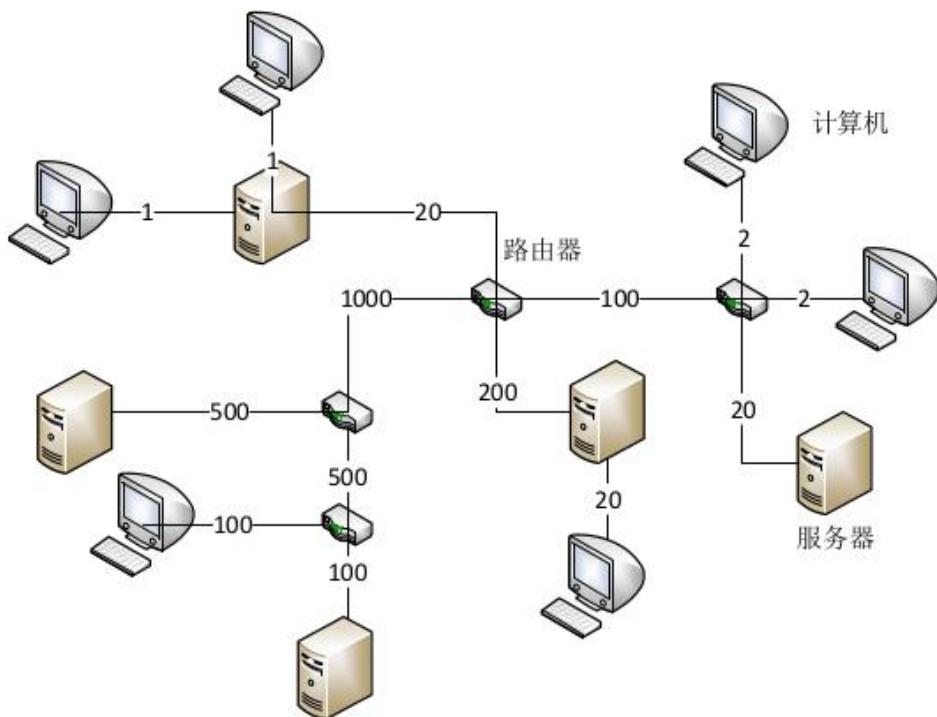
3. 网络拓扑图 (Network Topology)

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不能为同一个节点。

具体如下图:



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、

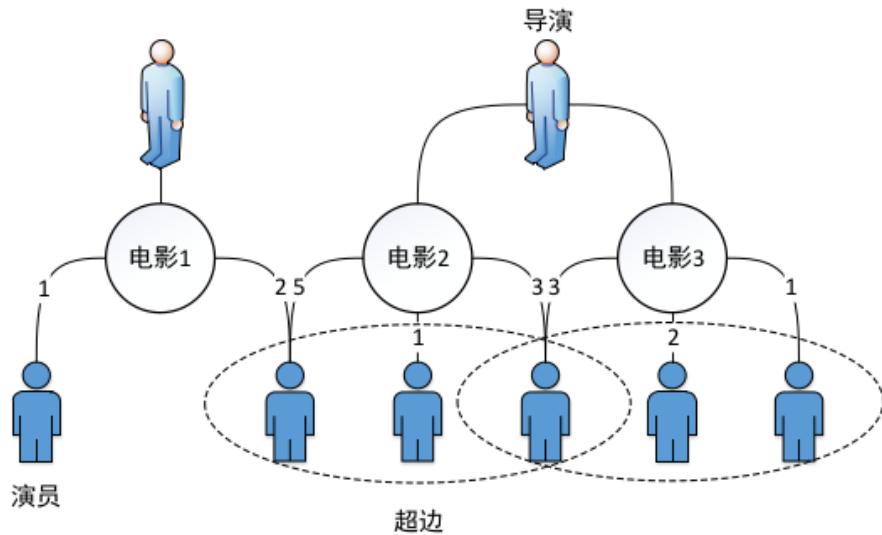
IMDb 评分)、“演员”(label 为姓名,属性:年龄、性别)。

“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。

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。

具体如下：



3.2 基于语法的图数据输入

正则表达式部分见后面的 factory 部分，来进行正则表达式的分析。

1. 对于 GraphPoet

这个图的输入比较简单，根据样例，就可以写出一个类似的输入文件如下：

```

GraphType=GraphPoet
VertexType=Word
Vertex=<a,Word>
Vertex=<b,Word>
Vertex=<c,Word>
Vertex=<d,Word>
EdgeType=WordNeighborhood
Edge=<ab,WordNeighborhood,1,a,b,YES>
Edge=<ac,WordNeighborhood,1,a,c,YES>
Edge=<bd,WordNeighborhood,1,b,d,YES>
Edge=<cd,WordNeighborhood,1,c,d,YES>
  
```

为了检测我的 factory 的正确性，我在 factory 中写了 main 方法来进行检测，main 方法代码如下：

```

public static void main(String[] args) {
    GraphPoetFactory mgf = new GraphPoetFactory();
    mgf.createGraph("src/GraphPoet.txt");
    for(Vertex x: mgf.graph.vertices()) {
        System.out.println(x.toString());
    }
    for(Edge x: mgf.graph.edges()) {
        System.out.println(x.toString());
    }
}

```

输出结果如下：

```

<terminated> GraphPoet
<"a", "Word">
<"b", "Word">
<"c", "Word">
<"d", "Word">
Label:
ab
Vertices:
a b
Weight:
1.0

Label:
cd
Vertices:
c d
Weight:
1.0

Label:
ac
Vertices:
a c
Weight:
1.0

Label:
bd
Vertices:
b d

```

2. 微博社交网络

同样是根据样例写文件，文件如下：

```

GraphType=SocialNetwork
GraphName=MySocialNetwork
VertexType=Person
Vertex=<Wang,Person,<M,56>>
Vertex=<Li,Person,<F,23>>
Vertex=<Liu,Person,<M,18>>
Vertex=<Zhao,Person,<F,22>>
EdgeType=ForwardConnection,CommentConnection,FriendConnection
Edge=<LiForwardLiu,ForwardConnection,1,Li,Liu,Yes>
Edge=<ZhaoForwardLiu,ForwardConnection,0.5,Zhao,Liu,Yes>
Edge=<WangCommentZhao,CommentConnection,0.3,Wang,Zhao,Yes>
Edge=<LiFriendLiu,FriendConnection,0.8,Li,Liu,Yes>

```

同样是调用 main 方法， 然后打印出图中的所有点和边 输出结果如下：

```
<terminated> SocialNetworkFactory [Java Application] /opt/java/bin/java (201
<"Li", "Person", <"F", "23">>
<"Wang", "Person", <"M", "56">>
<"Liu", "Person", <"M", "18">>
<"Zhao", "Person", <"F", "22">>
Label:
WangCommentZhao
Vertices:
Wang Zhao
Weight:
0.059999999999999984

Label:
LiFriendLiu
Vertices:
Li Liu
Weight:
0.8

Label:
ZhaoForwardLiu
Vertices:
Zhao Liu
Weight:
0.06999999999999998

Label:
LiForwardLiu
Vertices:
...
```

需要注意的是，对于 SocialNetwork，加入一条边之后，就要进行权重的修改，由于是 double 类型的除法，会出现很多位小数，这样在测试的时候判断是否相等会为我们带来麻烦，所以需要有一个误差范围比如说 0.001 之类

3. 网络拓扑图

同样是根据样例写文件，文件如下：

```
GraphType=NetworkTopology
GraphName=LabNetwork
VertexType=Computer,Router,Server
Vertex=<Server1,Server,<192.168.1.2>>
Vertex=<Router1,Router,<192.168.1.1>>
Vertex=<Computer1,Computer,<192.168.1.3>>
EdgeType=NetworkConnection
Edge=<R1S1,NetworkConnection,100,Router1,Server1,No>
Edge=<C1S1,NetworkConnection,10,Computer1,Server1,No>
```

输出结果如下：

```
<terminated> NetworkTopologyFactory [Java Application] /opt/java/bin/
<"Computer1", "Computer", <"192.168.1.3">>
<"Server1", "Server", <"192.168.1.2">>
<"Router1", "Router", <"192.168.1.1">>
Label:
C1S1
Vertices:
Computer1 Server1
Weight:
10.0

Label:
R1S1
Vertices:
Server1 Router1
Weight:
100.0
```

4. 电影网络图

输入文件如下：

```
1GraphType=MovieGraph
2GraphName=MyFavoriteMovies
3VertexType=Movie,Actor,Director
4Vertex=<TheShawshankRedemption,Movie,<1994,USA,9.3>>
5Vertex=<TheShawshankRedemption2,Movie,<1994,USA,9.3>>
6Vertex=<FrankDarabont,Director,<M,59>>
7Vertex=<MorganFreeman,Actor,<M,81>>
8Vertex=<TimRobbins,Actor,<M,60>>
9Vertex=<TheGreenMile,Movie,<1999,USA,8.5>>
10Vertex=<TomHanks,Actor,<M,62>>
11EdgeType=MovieActorRelation,MovieDirectorRelation,SameMovieHyperEdge
12Edge=<SRFD,MovieDirectorRelation,-1,TheShawshankRedemption,FrankDarabont,No>
13Edge=<GMFD,MovieDirectorRelation,-1,TheGreenMile,FrankDarabont,No>
14Edge=<SRMF,MovieActorRelation,1,TheShawshankRedemption,TimRobbins,No>
15Edge=<SRTR,MovieActorRelation,2,TheShawshankRedemption,MorganFreeman,No>
16Edge=<GMTH,MovieActorRelation,1,TheGreenMile,TomHanks,No>
17HyperEdge=<ActorsInSR,SameMovieHyperEdge,{TimRobbins,MorganFreeman}>
```

输出结果如下：

```

<terminated> MovieGraphFactory [Java Application] /opt/java/bin/java (2018年5月4日 下午9:15:32)
<"FrankDarabont", "Director", <"59", "M">>
<"MorganFreeman", "Actor", <"81", "M">>
<"TheGreenMile", "Movie", <"1999", "USA", "8.5">>
<"TimRobbins", "Actor", <"60", "M">>
<"TomHanks", "Actor", <"62", "M">>
<"TheShawshankRedemption", "Movie", <"1994", "USA", "9.3">>
<"TheShawshankRedemption2", "Movie", <"1994", "USA", "9.3">>
Label:
SRMF
Vertices:
TimRobbins TheShawshankRedemption
Weight:
1.0

Label:
GMFD
Vertices:
FrankDarabont TheGreenMile
Weight:
-1.0

Label:
GMTH
Vertices:
TheGreenMile TomHanks
Weight:
1.0

Label:
ActorsInSR
Vertices:
MorganFreeman TimRobbins

```

至此，四个图的正则解析结果基本是正确的。下面进行对其中的正则表达式的解析

首先，有以下成员变量

```

private Graph<Vertex, Edge> graph;
private String graphName = new String();
private List<String> vertexTypes = new ArrayList<>();
private List<String> edgeTypes = new ArrayList<>();

```

(1) 对于 GraphName 的解析

```

Matcher matcher = Pattern.compile("GraphName=(\\w+)").matcher(string); // match the graph name.
if (matcher.matches()) {
    graphName = matcher.group(1);
    //System.out.println(graphName);
    return true;
}

```

`(\\w+)`用来匹配一串数字或字母，在这里可以直接将匹配到的字符串赋给成员变量 `GraphName`。

(2) 对于 VertexType 的解析

```

Matcher matcher2 = Pattern.compile("VertexType=(\\w+),(\\w+),(\\w+)").matcher(string); // match the vertex type.
if(matcher2.matches()) {
    vertexTypes.add(matcher2.group(1));
    vertexTypes.add(matcher2.group(2));
    vertexTypes.add(matcher2.group(3));
    return true;
}

```

对于每个图，都会将匹配到的 VertexType 加入到存有 vertexTypes 的 List 中，这样便于今后可能会用到。

(3) 对于 Vertex 的匹配

```

Matcher matcher3 = Pattern.compile("Vertex=<(\\w+),(\\w+),<([MF]),(\\d+)>>").matcher(string); // match the vertex.
if(matcher3.matches()) {
    if(matcher3.group(2).equals("Actor")) {
        Actor actor = new Actor(matcher3.group(1));
        actor.fillVertexInfo(new String[] {matcher3.group(3), matcher3.group(4)});
        graph.addVertex(actor);
        return true;
    }
    if(matcher3.group(2).equals("Director")) {
        Director director = new Director(matcher3.group(1));
        director.fillVertexInfo(new String[] {matcher3.group(3), matcher3.group(4)});
        graph.addVertex(director);
        return true;
    }
}

```

这里使用 MovieGraph 来进行举例，所以图中匹配的是 Actor 和 Director，(\d+)用来匹配一串数字，[MF]用来匹配性别，只能是 M 或者 F，然后将匹配到的字符串进行处理（有的可能要转成数字）来进行 Vertex 节点信息的填写，然后将其加入到图中，这样，就完成了对于节点的匹配。

(4) 对于特殊数字的限制

```

Matcher matcher4 = Pattern.compile("Vertex=<(\\w+),Movie,<(\\d{4}),(\\w+),([\\d\\.]+)>>").matcher(string);
if(matcher4.matches()) {
    Movie movie = new Movie(matcher4.group(1));
    movie.fillVertexInfo(new String[] {matcher4.group(2), matcher4.group(3), matcher4.group(4)});
    graph.addVertex(movie);
    return true;
}

```

图中，展示的是对于年代的四位数字的限制和对于小数的匹配，如图，

(\d{4})表示只能匹配四位数字，(\d\.)表示对于小数的匹配。

(5) 对于边的匹配

```

Matcher matcher6 = Pattern.compile("Edge=<(\w+), (\w+), ([\w-\d]+), (\w+), (\w+), (\w+)>").matcher(string);
if(matcher6.matches()) {
    //System.out.println(Integer.valueOf(matcher6.group(3)));
    Edge edge = null;
    if(matcher6.group(2).equals("MovieActorRelation")) {
        edge = new MovieActorRelation(matcher6.group(1), Integer.valueOf(matcher6.group(3)));
    }
    else if(matcher6.group(2).equals("MovieDirectorRelation")) {
        edge = new MovieDirectorRelation(matcher6.group(1), Integer.valueOf(matcher6.group(3)));
    }
    if(edge == null) return false;
    Vertex v1 = null;
    Vertex v2 = null;
    for(Vertex x: graph.vertices()) {
        if(x.getLabel().equals(matcher6.group(4)))
            v1 = x;
        else if(x.getLabel().equals(matcher6.group(5)))
            v2 = x;
    }
    if(v1 != null && v2 != null) {
        edge.addVertices(Arrays.asList(v1, v2));
        graph.addEdge(edge);
        return true;
    }
    else return false;
}

```

这里对于边的匹配有一点复杂，不只是匹配到符合要求的字符串就进行信息的填写，首先要进行判断的就是边里的节点是不是已经存在于图中了，如果不存在的话，按照前面的 spec，应该直接报错，然后是进行判断 edge 是否为空，如果不为空，那么就可以成功加入到图中。

(6) 对于 ip 地址匹配

```

Matcher matcher3 = Pattern.compile("Vertex=<(\w+), (\w+), <(\d+\.\d+\.\d+\.\d+)>>").matcher(string);
if(matcher3.matches()) {
    if(matcher3.group(2).equals("Server")) {
        Server server = new Server(matcher3.group(1));
        server.fillVertexInfo(new String[] {matcher3.group(3)});
        graph.addVertex(server);
        return true;
    }
    else if(matcher3.group(2).equals("Computer")) {
        vertex.Computer computer = new vertex.Computer(matcher3.group(1));
        computer.fillVertexInfo(new String[] {matcher3.group(3)});
        graph.addVertex(computer);
        return true;
    }
    else if(matcher3.group(2).equals("Router")) {
        Router router = new Router(matcher3.group(1));
        .....
    }
}

```

采用上述方式对于形如“192.168.2.1”的 ip 地址进行匹配
主要的正则表达式就是上述几种。

3.3 面向复用的设计：**Graph<L, E>**

Graph<L, E>中主要是一些抽象出来的每个图都会有的公共方法，
有如下的抽象方法：

(1) **empty ()**，用于产生一个空图

```

/**
 * Create an empty graph.
 * @param <L, E> type of vertex labels and edge labels in the graph.
 * @return a new empty weighted graph.
 */
public static <L extends Vertex , E extends Edge> Graph<L, E> empty() {
    return new ConcreteGraph<L, E>();
}

```

(2) addVertex () 方法

```

/**
 * Add a vertex to this graph.
 *
 * @param vertex label for the new vertex
 * @return true if this graph did not already include a vertex with the
 *         given label; otherwise false (and this graph is not modified).
 */
public boolean addVertex(L vertex);

```

(3)removeVertex()方法

```

/**
 * Remove a vertex from this graph; if the edge is an undirected edge or
 * directed edge. But if it's a hyper edge, the edge can still exist if
 * the edge without this vertex is still legal,
 * @param vertex label of the vertex to remove.
 * @return true if this graph included a vertex with the given label;
 *         otherwise false (and this graph is not modified).
 */
public boolean removeVertex(L vertex);

```

(4) vertices () 方法

```

/**
 * Get all the vertices in this graph.
 *
 * @return the set of labels of vertices in this graph
 */
public Set<L> vertices();

```

(5) targets()方法

```

/**
 * Get the target vertices with directed edges from a source vertex and the
 * weights of those edges.
 * @param source a label.
 * @return a map where the key set is the set of labels of vertices such
 *         that this graph includes an edge from source to that vertex,
 *         not only the directed edges, but also the undirected edges. and
 *         the value for each key is the (nonzero) weight of the edge from
 *         source to the key
 */
public Map<L, Double> targets(L source);

```

(6) addEdge () 方法

```
/**
 * Add an edge to the graph, including the hyper edge.
 * @param edge a label of the edge
 * @return true if the graph didn't have the edge before; otherwise is false
 *         (and this graph is not modified)
 */
public boolean addEdge(E edge);
```

(7) removeEdge () 方法

```
/**
 * Delete an edge to the graph, including the hyper edge.
 * @param edge a label of the edge
 * @return true if the graph have the edge; otherwise is false
 *         (and this graph is not modified)
 */
public boolean removeEdge(E edge);
```

(8) edges () 方法

```
/**
 * Get all the edges in this graph.
 * @return the set of labels of edges in this graph
 */
public Set<E> edges();
```

3.4 面向复用的设计：Vertex

将共有的属性和方法在抽象类定义出来，差异化的属性或方法写在子类中，使用继承和重写来实现。

将 String label 定义在 Vertex 抽象类中，每一个继承自该类或继承自该类的子类都具有该属性。将 fillVertexInfo、getLabel（公有的）方法定义在 Vertex 中通过构造函数传入 label。

1 . Word 类直接继承自 Vertex，重写 toString、equals、hashCode 方法，并实现父类中定义的抽象方法，由于 Word 类没有属性，

fillVertexInfo 方法的实现体为空。加入一个变量 type = "Word"

- 2 . Person 类直接继承自 Vertex, 重写 toString、equals、hashCode 方法, 添加性别、年龄属性, 通过 fillVertexInfo 方法给特有的两个属性赋值。定义 type="Person"
- 3 . NetworkVertex 类继承自 Vertex 类, 作为 Computer、Server、Router 的父类, 添加三个类的共有属性— IP 地址, 并在 Network Vertex 中实现 fillVertexInfo 方法, 定义 type="NetworkVertex"
- 4 . Computer 类继承自 NetworkVertex, 由于 Vertex 中的抽象方法在 Network Vertex 类中已经实现, 在 Computer 类中只需定义 type="Computer"
- 5 . Server 类继承自 NetworkVertex, 只需定义 type="Server"
- 6 . Router 类继承自 NetworkVertex, 只需定义 type="Router"
- 7 . MoviePerson 继承自 Vertex, 作为 Actor 和 Director 的父类, 定义年龄、性别两个属性, 通过 fillVertexInfo 方法为两个属性赋值。
- 8 . Actor 和 Directeor 都继承自 MoviePerson, 只需分别定义 type="Actor" 和 type="Director"

有如下的 AF、RI 等

```

public abstract class Vertex {
    private final String label;      //the label of the vertex.

    // Abstraction function:
    //   -label is the label of each vertex, so different vertex cannot have the same label;
    //   -label is the only symbol of the vertex.
    //   -Vertex class can create a Vertex object.
    // Representation invariant:
    //   -label cannot be null.
    // Safety from rep exposure:
    //   -There's no method return the mutable.
}

```

(1)

下图是构造器、填写信息部分、getLabel 部分

```

/**
 * Constructor of Vertex.
 * @param label, String object, which can not be null.
 */
public Vertex(String label) {
    this.label = label;
}

/**
 * Filling the info of the vertex.
 * @param args, an array, which stores the info of the vertex.
 */
public abstract void fillVertexInfo(String[] args);

/**
 * Get the label.
 * @return this.label
 */
public String getLabel() {
    return label;
}
...

```

(2)

下图是重写的 `toString ()` 方法

```

/**
 * Override the toString() method.
 * To show the String of vertex, different kinds of Vertex has
 * different shows.
 * @return the label
 */
@Override
public String toString() {
    return label;
}

```

(3) 重写的 equals () 方法

```
/**
 * Override the equals() method.
 * Determine the two Vertex objects are the same or not.
 * @return true if equal, false if not equal.
 */
@Override
public boolean equals(Object that) {
    if(that == null) {
        return false;
    }
    if(this == that) {
        return true;
    }
    if(that instanceof Vertex) {
        Vertex vertex = (Vertex) that;
        return vertex.getLabel().equals(label);
    }
    return false;
}
```

(4) 重写的 hashCode () 方法

```
/**
 * Override the hashCode() method.
 * Compute the hash code of a Vertex object.
 * @return the new hash code of a Vertex object.
 */
@Override
public int hashCode() {
    int result = 17;
    result = 31 * result + label.hashCode();
    return result;
}
```

利用 label 来进行 hashCode 的计算

3.5 面向复用的设计： Edge

将所有类型的边的共有属性和方法抽象出来，放在 Edge 抽象类中。

toString()、equals()、hashCode() 在每个子类中都需要重写，下面不再重复

需要在 Edge 抽象类中定义的属性有 label、weight，通过构造函数来传入 label 和 weight；定义的抽象方法有 getType()、addVertices、containVertex()、vertices()、sourceVertices()、targetVertices()。

DirectedEdge 继承自 Edge，作为所有具体有向边类型的父类，调用父类的构造函数。用两个 Vertex 类型的变量 source 和 target 表示边的起点和终点，重写 addVertices()方法时，将 vertices 的第一个元素传给 source，第二个元素传给 target。SourceVertices () 方法，返回一个只包含起点 source 的集合，targetVertices()方法类似。

UndirectedEdge 继承自 Edge，作为所有具体无向边类型的父类，调用父类的构造函数。在这里我同样是用了 Vertex 类型的变量来存储边的端点，不考虑顺序。在实现 sourceVertices()方法时，将包含两个端点的集合返回，targetVertices()方法类似。

HyperEdge 继承自 Vertex，作为具体的超边类型的父类。用一个 Vertex 类型的集合 Set<Vertex> 来存储超边上的点。重写 addVertices()方法时，需要判断传入的列表中的元素是否已在边上，如果不在，将列表中的顶点加入集合，返回 true；否则添加顶点并返回 false。

对于 sourceVertices() 和 targetVertices() 方法，对后续的实验并没有作用，在这里都是返回的该边上的 Vertices 的集合。

WordEdge 为继承自 DirectedEdge，为有向边。由于定义在 Edge 中的抽象方法已经在父类中得到实现并且不用改进。为 WordEdge 增加一个 type="WordEdge"，构造函数调用父类的。重写 toString()、hashCode()、equals() 方法时，为了简化可以先调用父类中已经写好的，

然后将子类的特征添加进去。

有如下的 AF、RI 等

```
public abstract class Edge {
    protected final String label;           //标签
    protected double weight;                //权重
    private final List<Vertex> allVertices = new ArrayList<>();

    // Abstraction function:
    //   -label represents the label of the edge.
    //   -weight represents the weight of the edge.
    //   -allVertices is a List object which contains all Vertex in the edge.
    // Representation invariant:
    //   -label cannot be null.
    //   -weight must more than 0 as long as it is not the hyper edge.
    //   -After adding vertex to the edge, allVertices.size() must more than 2
    //   or be equal to 2 as long as the edge is not a LOOP.
    // Safety from rep exposure:
    //   -Some methods such as vertices() will return a Set, which is a mutable
    //   collection, so, to prevent exposing the representation to the client,
    //   we must make the defensive copy.
```

(1) 构造器

```
/**
 * The constructor
 */
public Edge(String label, double weight) {
    this.label = label;
    this.weight = weight;
}
```

(2) getLabel () 方法

```
/**
 * The constructor
 */
public Edge(String label, double weight) {
    this.label = label;
    this.weight = weight;
}
```

(3) getWeight () 方法

```

    /**
     * Get the weight.
     * @return the double weight.
     */
    public double getWeight() {
        return weight;
    }

```

(4) setWeight () 方法

```

    /**
     * Set the weight.
     * @param weight
     */
    public void setWeight(double weight) {
        this.weight = weight;
    }

```

(5) addVertices () 方法

```

    /**
     * Add Vertex into the edge.
     * -If the edge is a HyperEdge, then assert the vertices.size() >= 2, then add all
     * vertices to the graph.
     * -If the edge is a DirectedEdge, then assert the vertices.size() == 2, then the
     * first element is the source, and the second element is the target.
     * -If the edge is a Undirected Edge, then the order is arbitrary.
     * -If it represents a loop, assert the vertices.size() == 1.
     * @param vertices, the List object which has at least two elements.
     * @return return whether the add is successful.
     */
    public abstract boolean addVertices(List<Vertex> vertices);

```

(6) containsVertex () 方法

```

    /**
     * Determine the edge contains the given vertex or not.
     * @param V, the given vertex.
     * @return contains or not.
     */
    public abstract boolean containVertex(Vertex V);

```

(7) vertices () 方法

```

/**
 * Return the vertices contained in the edges.
 * @return a Set which has all Vertex objects in the edge.
 */
public abstract Set<Vertex> vertices();

```

(8) sourceVertices () / targetVertices ()

```

/**
 * Return the sources.
 * -If it is a directed edge, return the source vertex.
 * -If it is an undirected edge, return the both two vertices.
 * -If it is a hyper edge, return all the vertices.
 * @return a Set which represents the source vertex.
 */
public abstract Set<Vertex> sourceVertices();

/**
 * Return the targets.
 * -If it is a directed edge, return the target vertex.
 * -If it is an undirected edge, return the both two vertices.
 * -If it is a hyper edge, return all the vertices.
 * @return a Set which represents the target vertex.
 */
public abstract Set<Vertex> targetVertices();

```

(9) 重写 toString () 方法

```

/**
 * Override the method toString(). Return all the info of the edge.
 */

@Override
public String toString() {
    String newString = new String();
    newString += ("Label: " + '\n');
    newString += (label + '\n');
    newString += ("Vertices: " + '\n');
    for(Vertex x: this.vertices()) {
        newString += (x.getLabel() + " ");
    }
    newString += '\n';
    newString += ("Weight: " + '\n');
    newString += (this.weight + "\n");
    //newString += '\n';
    return newString;
}

```

(10) 重写 hashCode () 方法

```
/**  
 * Override the method hashCode().  
 * Return new hash code.  
 */  
@Override  
public int hashCode() {  
    int result = 17;  
    result = result * 31 + label.hashCode();  
    return result;  
}
```

(11) 重写 equals () 方法

```
/**  
 * Override the method equals().  
 */  
@Override  
public boolean equals(Object that) {  
    if(this == that)  
        return true;  
    if(that == null)  
        return false;  
    if(that instanceof Edge) {  
        Edge other = (Edge) that;  
        return other.vertices().equals(this.vertices()) && other.getLabel().equals(this.getLabel())  
            && (other.getWeight() == this.getWeight());  
    }  
    return false;  
}
```

3.6 可复用 API 设计

(1) degreeCentrality

代码如下：

```

/**
 * calculate the degree centrality of a vertex
 * @param g, the graph
 * @param v, the vertex
 * @return the degree centrality.
 */
public static double degreeCentrality(Graph<Vertex, Edge> g, Vertex v) {
    int cnt = 0;
    for(Edge x: g.edges()) {
        if(x.vertices().contains(v))
            ++cnt;
    }
    return (double) cnt / (g.edges().size() - 1);
}

public static double degreeCentrality(Graph<Vertex, Edge> g) {
    double maxDegreeCentrality = -1;
    double sum = 0;
    int size = g.vertices().size();
    for(Vertex x: g.vertices()) {
        if(degreeCentrality(g, x) > maxDegreeCentrality) {
            maxDegreeCentrality = degreeCentrality(g, x);
        }
    }
    for(Vertex x: g.vertices()) {
        sum += Math.abs(maxDegreeCentrality - degreeCentrality(g, x));
    }
    return (double) sum / (size * size - 3 * size + 2);
}

```

这里采用维基百科上给出的公式：

计算一个点的 degree centrality：

The degree centrality of a vertex v , for a given graph $G := (V, E)$ with $|V|$ vertices and $|E|$ edges, is defined as

$$C_D(v) = \deg(v)$$

计算一个图的 degree centrality：

$$\text{So, for any graph } G := (V, E), C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v*) - C_D(v_i)]}{|V|^2 - 3|V| + 2}$$

(2) closeness centrality 的计算

这里，本来打算用维基百科上的方法，想通过 Dijkstra 算法来求解最短路，并代入公式计算，但是老师说了可以调用第三方库，于是采用了调用 JUNG 包的方法，使用其中已经封装好的 API，代码如下图

```

/**
 * Get the closeness of a vertex
 * @param g, the graph
 * @param v, the vertex
 * @return the closeness
 */
public static double closenessCentrality(Graph<Vertex, Edge> g, Vertex v) {
    if(g instanceof GraphPoet || g instanceof SocialNetwork) {
        SparseMultigraph<Vertex, Edge> graph = new SparseMultigraph<>();
        for(Vertex x: g.vertices()) {
            graph.addVertex(x);
        }
        for(Edge y: g.edges()) {
            if(y.vertices().size() > 1)
                graph.addEdge(y, y.getList().get(0), y.getList().get(1), EdgeType.DIRECTED);
        }
        ClosenessCentrality<Vertex, Edge> ranker = new ClosenessCentrality<>(graph);
        return ranker.getVertexScore(v);
    }
    else {
        SparseMultigraph<Vertex, Edge> graph = new SparseMultigraph<>();
        for(Vertex x: g.vertices()) {
            graph.addVertex(x);
        }
        for(Edge y: g.edges()) {
            if(y instanceof HyperEdge)
                continue;
            else graph.addEdge(y, y.getList().get(0), y.getList().get(1), EdgeType.UNDIRECTED);
        }
        ClosenessCentrality<Vertex, Edge> ranker = new ClosenessCentrality<>(graph);
    }
}

```

其中，调用了 ClosenessCentrality 类来进行计算即可。

官网给出的 document 如下：

ClosenessCentrality

`public ClosenessCentrality(Hypergraph<V, E> graph)`

Creates an instance which measures distance on the graph without edge weights.

Parameters:

`graph -`

由此，计算出某一个 Vertex 的 closeness centrality。

(3) betweenness centrality 的计算

```

    * @param v, the vertex
    * @return the betweenness
    */
    public static double betweennessCentrality(Graph<Vertex, Edge> g, Vertex v) {
        if(g instanceof GraphPoet || g instanceof SocialNetwork) {
            SparseMultigraph<Vertex, Edge> graph = new SparseMultigraph<>();
            for(Vertex x: g.vertices()) {
                graph.addVertex(x);
            }
            for(Edge y: g.edges()) {
                if(y.vertices().size() > 1)
                    graph.addEdge(y, y.getList().get(0), y.getList().get(1), EdgeType.DIRECTED);
            }
            BetweennessCentrality<Vertex, Edge> ranker = new BetweennessCentrality<>(graph);
            return ranker.getVertexScore(v);
        }
        else {
            SparseMultigraph<Vertex, Edge> graph = new SparseMultigraph<>();
            for(Vertex x: g.vertices()) {
                graph.addVertex(x);
            }
            for(Edge y: g.edges()) {
                if(y instanceof HyperEdge)
                    continue;
                else graph.addEdge(y, y.getList().get(0), y.getList().get(1), EdgeType.UNDIRECTED);
            }
            BetweennessCentrality<Vertex, Edge> ranker = new BetweennessCentrality<>(graph);
            //ranker.evaluate();
            return ranker.getVertexScore(v);
        }
    }
}

```

这里同样是采用调用第三方库的方法，来进行计算

官方文档如下：

BetweennessCentrality

```
public BetweennessCentrality(Graph<V,E> graph)
```

Calculates betweenness scores based on the all-pairs unweighted shortest paths in the graph.

Parameters:

graph - the graph for which the scores are to be calculated

(4) indegreeCentrality/ outdegreeCentrality 的计算

```

    /**
     * Only for the directed graph, calculate the in degree centrality.
     * @param g, the graph
     * @param v, the vertex
     * @return the in degree centrality.
     */
    public static double inDegreeCentrality(Graph<Vertex, Edge> g, Vertex v) {
        return g.sources(v).size();
    }

    /**
     * Only for the directed graph, calculate the out degree centrality.
     * @param g, the graph.
     * @param v, the vertex.
     * @return the out degree centrality.
     */
    public static double outDegreeCentrality(Graph<Vertex, Edge> g, Vertex v) {
        return g.targets(v).size();
    }
}

```

直接调用 targets () 方法就可以计算。

(5) GraphDistance 的计算

- **Distance(Graph<L, E> g, L start, L end)**

计算节点 start 和 end 之间的最短距离，使用 dijkstra 算法求得节点 start 的单源最短路径，取终点为 end 的最短距离返回。

- **Eccentricity(Graph<L, E> g, L v)**

一个顶点的离心率为该顶点到其他顶点最短距离的最大值。

实现方法：dijkstra 算法求得 v 的单源最短路径，取单点到其他顶点最短距离的最大值。此处需要注意，不能考虑无法到达的顶点。

- **Radius(Graph<L, E> g)**

图的半径：图中顶点离心率的最小值。

实现方法：调用上面实现的 eccentricity()方法，计算图中每一个顶点的离心率，取最小值。

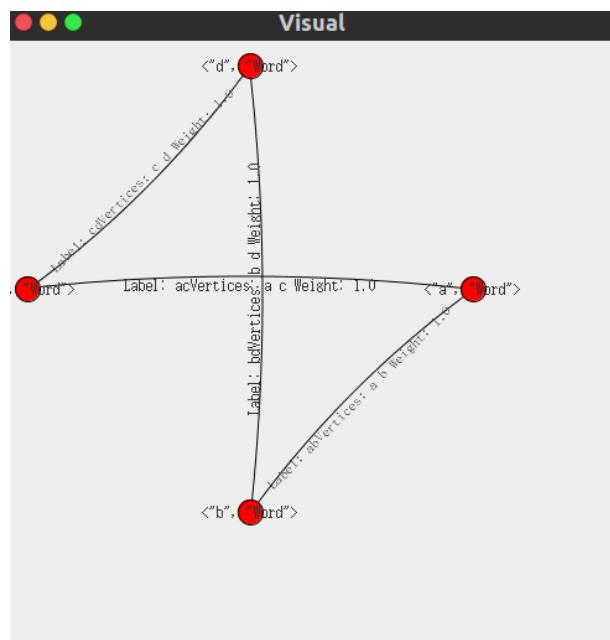
- **Diameter(Graph<L, E> g)**

图的直径：与图的半径相反，为图中顶点离心率的最大值。

实现方法：调用上面实现的 eccentricity()方法，计算每一个顶点的离心率取最大值。

3.7 的可视化：第三方 API 的复用（选做）

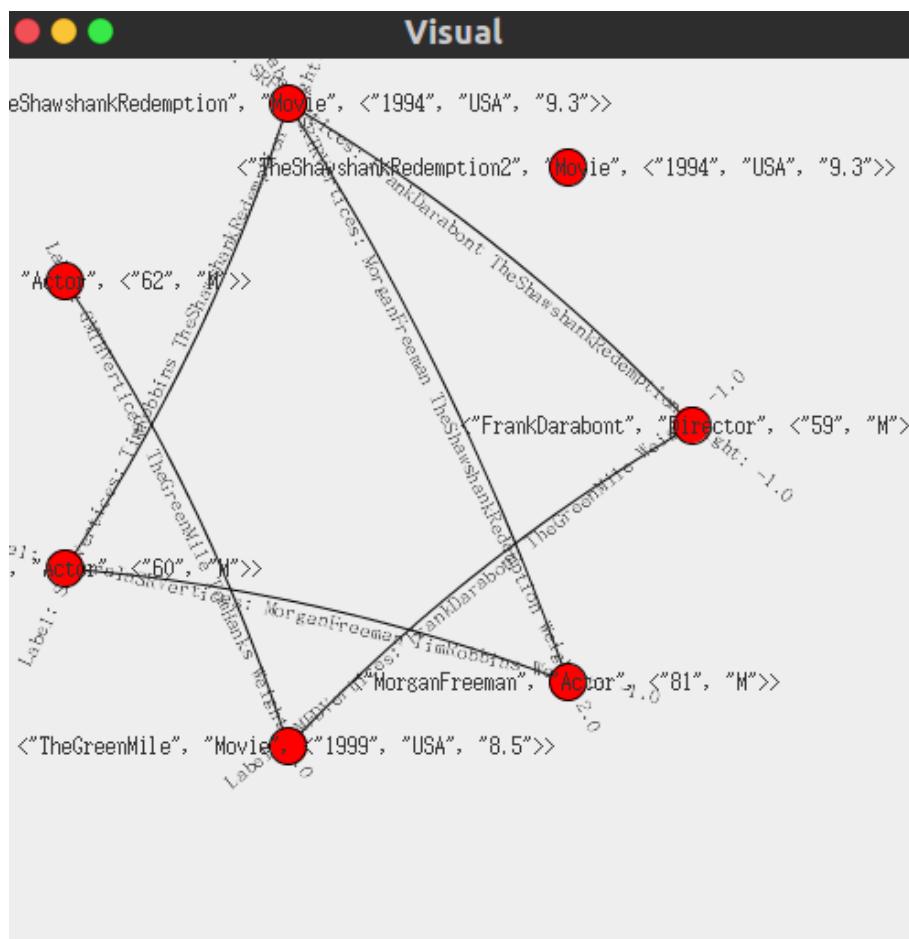
(1) GraphPoet



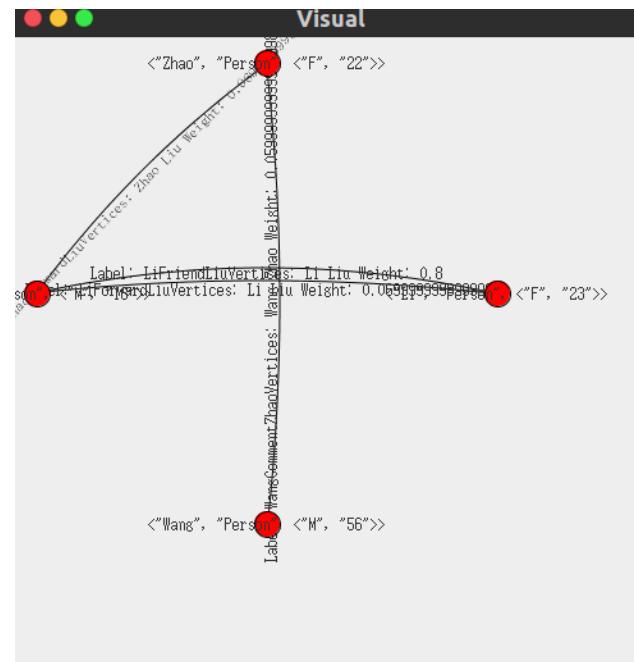
```
Problems @ Javadoc Declaration Search Coverage [Console ]  
GraphPoetApp [Java Application] /opt/java/bin/java (2018年5月6日 下午10:03:45)  
-----GraphPoetApp-----  
1. Input the filepath of graph  
2. Command for graph  
3. Vertex centrality  
4. Graph centrality, radius and diameter  
5. Shortest distance between two vertices  
6. Exit  
7. Filter the edge  
8. GUI  
-----Dedigned by muty  
  
Please choose a function(1 - 8) and input the number.  
1  
Function 1, please input the path to concrete the graph.  
src/GraphPoet.txt  
Concrete graph successfully!  
Please choose a function(1 - 8) and input the number.  
8  
Please input the right number(1 - 8)  
Please choose a function(1 - 8) and input the number.
```

(2) MovieGraph

```
--MovieGraphApp--  
1. Input the filepath of graph  
2. Command for graph  
3. Vertex centrality  
4. Graph centrality, radius and diameter  
5. Shortest distance between two vertices  
6. Exit  
7. GUI.  
-----Dedigned by muty  
  
Please choose a function(1 - 7) and input the number.  
1  
Function 1, please input the path to concrete the graph.  
src/MovieGraph.txt  
Concrete graph successfully!  
Please choose a function(1 - 7) and input the number.  
7  
Please input the right number(1 - 7)  
Please choose a function(1 - 7) and input the number.
```



(3) SocialNetwork



```
SocialNetworkApp [Java Application] /opt/java/bin/java (2018年5月6日 下午10:09:47)
-----SocialNetworkApp-----
1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit
7. Set everyone a weight.
8. GUI.
-----Dedigned by muty
```

```
Please choose a function(1 - 8) and input the number.
1
Function 1, please input the path to concrete the graph.
src/SocialNetwork.txt
Concrete graph successfully!
Please choose a function(1 - 8) and input the number.
8
Please input the right number(1 - 8)
Please choose a function(1 - 8) and input the number.
```

(4) NetworkTopology

```

NetworkTopologyApp [Java Application] /opt/java/bin/java (2018年5月6日 下午10:17:51)
-----NetworkTopologyApp-----
1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit
7. GUI
-----Designed by muty

```

Please choose a function(1 - 7) and input the number.

1

Function 1, please input the path to concrete the graph.

src/NetworkTopology.txt

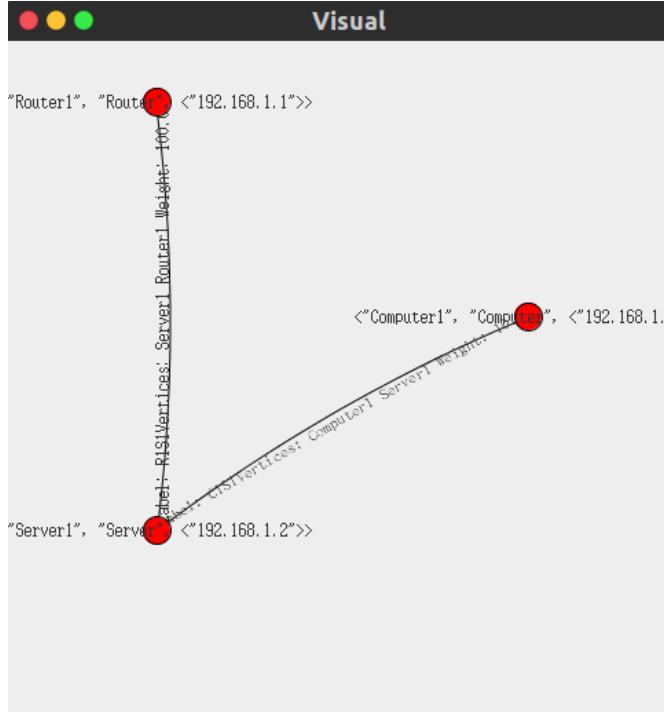
Concrete graph successfully!

Please choose a function(1 - 7) and input the number.

7

Please input the right number(1 - 7)

Please choose a function(1 - 7) and input the number.



GraphVisualizationHelper.java 代码如下：

```

//build GUI window
private void createWindow() {
    frame=new JFrame("Visual");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

```

这里是初始化一个 Window，用来存放我们的图。

```

public static void visualize(Graph<Vertex, Edge> g) {
    graph=new SparseMultigraph<Vertex, Edge>();
    for(Vertex v:g.vertices())
        graph.addVertex(v);
    for(Edge e:g.edges())
    {
        if(e.vertices().size()<=2)
        {
            List<Vertex> list = new ArrayList<>();
            for(Vertex ve:e.vertices())
                list.add(ve);
            graph.addEdge(e, list);
        }
        else
        {
            List<Vertex> list = new ArrayList<>();
            for(Vertex ve:e.vertices())
                list.add(ve);
            for(int i=0;i<list.size()-1;i++)
            {
                Edge ed = new SameMovieHyperEdge("actors in a same movie",-1);
                List<Vertex> edl = new ArrayList<>();
                edl.add(list.get(i));
                edl.add(list.get(i+1));
                ed.addVertices(edl);
                graph.addEdge(ed, list.get(i),list.get(i+1));
            }
        }
    }
}

```

这里是创建图的过程，将我们的图中的点和边复制到我们作为参数的图中。

```

//show GUI
private void showGraph() {
    Layout<Vertex, Edge> layout = new CircleLayout<Vertex, Edge>(graph);
    layout.setSize(new Dimension(400, 400));
    BasicVisualizationServer<Vertex, Edge> vv = new BasicVisualizationServer<Vertex,
    vv.setPreferredSize(new Dimension(500, 500));
    vv.getRenderingContext().setVertexLabelTransformer(new ToStringLabeller<Vertex>());
    vv.getRenderingContext().setEdgeLabelTransformer(
        new ToStringLabeller<Edge>());
    vv.getRenderer().getVertexLabelRenderer().setPosition(Position.CNTR);
    frame.getContentPane().add(vv);
    frame.pack();
    frame.setVisible(true);
}

```

这里是将 GUI 图像展现出来部分的代码

3.8 设计模式应用

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

3.8.2 使用 factory method 模式构造 Vertex 对象

Vertex 工厂代码如下：

```

1 package factory;
2
3 import vertex.Vertex;
4
5 public abstract class VertexFactory {
6     public abstract Vertex createVertex(String label, String[] args);
7 }
8

```

具体的 vertex 可以继承自 vertexfactory，代码如下：

```

1 package factory;
2
3 import vertex.Actor;
4
5 public class ActorVertexFactory extends VertexFactory {
6     @Override
7     public Vertex createVertex(String label, String[] args) {
8         Vertex vertex = new Actor(label);
9         vertex.fillVertexInfo(args);
10        return vertex;
11    }
12
13 }
14
15

public class PersonVertexFactory extends VertexFactory {
    @Override
    public Vertex createVertex(String label, String[] args) {
        Vertex vertex = new Person(label);
        vertex.fillVertexInfo(args);
        return vertex;
    }
}

```

3.8.3 使用 factory method 模式构造 Edge 对象

有如下的抽象类：

```

package factory;
import java.util.List;
public abstract class EdgeFactory {
    public abstract Edge createEdge(String label, double weight, List<Vertex> vertices);
}

```

当想生成具体的边类的时候，就可以采用继承的方法，如下图所示：

```

1 package factory;
2
3 import edge.Edge;[]
4 public class NetworkConnectionFactory extends EdgeFactory{
5     @Override
6     public Edge createEdge(String label, double weight, List<Vertex> vertices) {
7         Edge edge = new NetworkConnection(label, weight);
8         edge.addVertices(vertices);
9         return edge;
10    }
11 }
12
13 package factory;
14
15 import java.util.List;[]
16
17 public class FriendConnectionFactory extends EdgeFactory {
18     @Override
19     public Edge createEdge(String label, double weight, List<Vertex> vertices) {
20         Edge edge = new FriendConnection(label, weight);
21         edge.addVertices(vertices);
22         return edge;
23    }
24 }
25

```

3.8.4 使用 abstract factory 或 builder 模式构造 Graph 对象

首先有一个抽象工厂如下：

```

1 package factory;
2
3 import edge.Edge;[]
4
5 public abstract class GraphFactory {
6     public abstract Graph<Vertex, Edge> createGraph(String filePath);
7 }
8
9
10

```

然后有四个具体的图工厂类如下：

```

private Graph<Vertex, Edge> graph;
private String graphName = new String();
private List<String> vertexTypes = new ArrayList<>();
private List<String> edgeTypes = new ArrayList<>();
private boolean parse(String string) {
    if(string.equals("GraphType=GraphPoet")) {
        graph = new GraphPoet();
        return true;
    }
    Matcher matcher = Pattern.compile("GraphName=(\\w+]").matcher(string);
    if(matcher.matches()) {
        graphName = matcher.group(1);
        return true;
    }
    Matcher matcher2 = Pattern.compile("VertexType=(\\w+]").matcher(string);
    if(matcher2.matches()) {
        vertexTypes.add(matcher2.group(1));
        return true;
    }
    Matcher matcher3 = Pattern.compile("Vertex=<(\\w),Word>").matcher(string);
    if(matcher3.matches()) {
        Word word = new Word(matcher3.group(1));
        graph.addVertex(word);
        return true;
    }
}

```

```

        Matcher matcher4 = Pattern.compile("EdgeType=(\\w+)" ).matcher(string);
        if(matcher4.matches()) {
            edgeTypes.add(matcher4.group(1));
            return true;
        }
        Matcher matcher5 = Pattern.compile("Edge=<(\\w+),WordNeighborhood,(\\d+),(\\w),(\\w),(\\w+)" );
        if(matcher5.matches()) {
            WordEdge edge = new WordEdge(matcher5.group(1), Integer.valueOf(matcher5.group(2)));
            Vertex v1 = null;
            Vertex v2 = null;
            for(Vertex x: graph.vertices()) {
                if(x.getLabel().equals(matcher5.group(3)))
                    v1 = x;
                else if(x.getLabel().equals(matcher5.group(4)))
                    v2 = x;
            }
            if(v1 != null && v2 != null) {
                edge.addVertices(Arrays.asList(v1, v2));
                graph.addEdge(edge);
                return true;
            }
            else return false;
        }
        System.out.println("Wrong input! " + string);
        return false;
    }
}

```

每个工厂类都会有一个 parse 方法来解析输入的文件，然后根据解析结果来进行建边或者建点加入到图中。

3.8.5 使用 Strategy 模式调用 centrality 度量算法

首先有一个 CentralityContext 类，通过传入不同的参数 (String) 来进行 setCentralityStrategy。

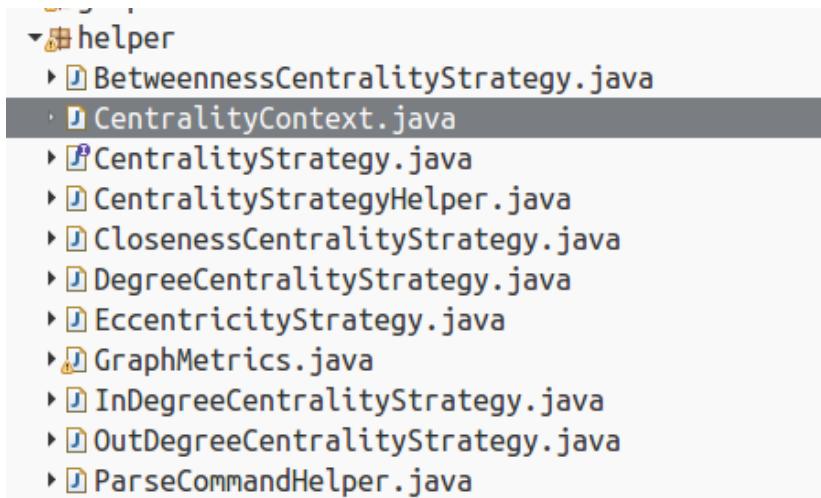
```

public class CentralityContext {
    public static double setCentralityStrategy(String string, Graph<Vertex, Edge> g, Vertex v) {
        return CentralityStrategyHelper.parseAndExecute(string, g, v);
    }
}

```

其中的 String 就代表着要求的不同的度。

然后有一个 CentralityStrategyHelper 类，去根据不同的 string 来调用不同的 strategy 方法，这些方法都实现一个接口，CentralityStrategy，由此实现了将输入与实现分开，所以，最终目录如下



3.8.6 使用 Composite 模式设计超边对象 (选做)

即在相同的类型的 HyperEdge 对象之间建立起一个树型层次结构，修改 HyperEdge 中的属性为一个 Vertex 类型，并将所有的超边加入到一个 List<HyperEdge> 对象中，如果在超边中需要添加一个顶点，需要新建一个超边对象，Vertex 类型赋值为新建的顶点对象，向新创建的 HyperEdge 对象中的 List 只添加 Vertex，并将 Vertex 添加至上层 HyperEdge 的 List 中。

3.8.7 使用 decorator 模式构造不同特征的 Edge 对象 (选做)

最后写完代码，尝试使用 Decorator 模式去进行修改，发现修改量很大，首先对于本次试验，在不考虑给边增加特性的时侯，Decorator 的作用不是特别大，其次，如果想要 Decorator 模式，需要将 Edge 定义为接口，而不是抽象类，所以，需要从根本去进行修改，时间仓促，遂放弃，在这里特此说明原因。

3.8.8 使用其他设计模式 (选做)

为了方便四个应用的客户端获得最短距离。由于 GraphDistance 中的最短距离方法需要传入的参数为 Vertex v 或者说是 L v 类型，但是从客户端只能传入 String 类型。所以在不改变原有代码的前提下，采用**适配器模式**。客户端只需要传入顶点的 label，由适配器来确定具体的 Vertex 类型，然后再调用 GraphDistance 中的方法，并返回或输出最短距离。

3.9 图操作指令的输入和处理 (选做)

(1) vertex --add "newVertex" "Word"

向图中加入新的节点

(2) vertex --delete regex

regex 是一个可用于正则匹配的字符串，比如说 vertex --delete “[a-zA-Z]\d{3}[a-zA-Z]”是删除 label 中有三个连续数字且数字前后为字母的节点。

(3) edge --add label type [weighted=Y|N] weight [directed=Y|N] v1, v2

用于增加一条边，但是我觉得这条指令有一点小瑕疵，就是一旦 type 确定，这条边是不是有向边就已经确定了，没有必要加上[weighted]这一部分。

(4) edge --delete regex

删除一条边，和删除节点是类似的。按照边的类型来进行加边

(5) 超边的添加

HyperEdge --add label type vertex1, vertex2 ... vertexn

关于超边是单独进行加边处理的，就是先要判断所有的点是不是都在 graph 中存在，如果不存在的话，则不能加边成功。

3.10 应用设计与开发

大部分的功能实现都是在 concreteGraph 这个 Graph 的实现类中实现的，只是有部分操作针对具体情况要进行 override，所以，这里先来介绍一下 concreteGraph 类

首先有如下的 AF、RI、rep exposure

```
package graph;

import java.util.ArrayList;
/*
 * Concrete a graph.
 */

public class ConcreteGraph<L extends Vertex, E extends Edge> implements Graph<L, E> {
    protected final List<L> allVertices = new ArrayList<>();
    protected final List<E> allEdges = new ArrayList<>();

    // Abstraction function:
    // -allVertices is a list where stored all the vertices in the graph.
    // -allEdges is a list where stored all the edges in the graph.
    // -The ConcreteGraph class can create a graph.
    // Representation invariant:
    // -allVertices is a list of L objects, which does not allow the same element.
    // -allEdges is a list of E objects, which also does not allow the same element.
    // -Assert allVertices.size() greater than allEdges.size() * 2 or equal to
    //     allEdges.size() * 2.
    // Safety from rep exposure:
    // -Some methods will return a Set such as the method vertices(), so to avoid the
    //     representation exposure to the client, we must have the defensive copies.
    // -In the graph, we have a private and final collection to store the vertices and
    //     the edges, so when we need return the vertices or the edges, we can make a copy
    //     to make sure only we know the data.
}
```

(1) addVertex () 方法：

首先在图中我有一个 list 去存储所有的点和所有的边，于是，就要先遍历一遍 allVertices 列表，去判断点是否已经在图中，如果已经存在的话，就返回 flase，否则，加点，返回 true。

```

/**
 * Add a vertex to this graph.
 *
 * @param vertex label for the new vertex
 * @return true if this graph did not already include a vertex with the
 *         given label; otherwise false (and this graph is not modified).
 */
public boolean addVertex(L vertex) {
    if(!allVertices.contains(vertex)) {
        allVertices.add(vertex);
        return true;
    }
    else return false;
}

```

(2) removeVertex () 方法

删点，不只是在 allVertices 中将点删除掉，还要遍历存储边的列表，将那些含有这个点的边删除掉。对于超边来说，如果删除掉其中一个点，该超边如果仍然满足超边的要求的话，依旧保留超边。

```

/**
 * Remove a vertex from this graph; if the edge is an undirected edge or
 * directed edge. But if it's a hyper edge, the edge can still exist if
 * the edge without this vertex is still legal,
 * @param vertex label of the vertex to remove.
 * @return true if this graph included a vertex with the given label;
 *         otherwise false (and this graph is not modified).
 */
public boolean removeVertex(L vertex) {
    if(allVertices.contains(vertex)) {
        allVertices.remove(vertex);
        Iterator<E> it = allEdges.iterator();
        while (it.hasNext()) {
            E edge = (E) it.next();
            if(edge.vertices().contains(vertex))
                it.remove();
        }
        return true;
    }
    else return false;
}

```

(3) vertices () 方法

将 allVertices 中的所有元素全部都加入到 set 中，然后返回这个 set 即可

```

/**
 * Get all the vertices in this graph.
 *
 * @return the set of labels of vertices in this graph
 */
@Override
public Set<L> vertices() {
    Set<L> vertices = new HashSet<>();
    for(L x: allVertices) {
        vertices.add(x);
    }
    return vertices;
}

```

(4) sources() / targets() 方法

```

/**
 * Get the source vertex of the directed edges to a target vertex and the
 * weights of those edges. If it is an undirected edge.
 * @param target a label
 * @return a map where the key is the set of labels of vertices such
 *         that this graph includes an edge from source to that vertex,
 *         not only the directed edges, but also the undirected edges.
 *         and the value for each key is the (nonzero) weight of the edge
 *         from source to the key
 */
@Override
public Map<L, Double> sources(L target) {
    Map<L, Double> sources = new HashMap<>();
    for(E x: allEdges) {
        if(x.vertices().contains(target)) {
            for(Vertex y: x.sourceVertices()) {
                if(!y.equals(target)) // avoid to put itself into the map
                    sources.put((L)y, x.getWeight());
            }
        }
    }
    return sources;
}

/**
 * Get the target vertices with directed edges from a source vertex and the
 * weights of those edges.
 * @param source a label
 * @return a map where the key set is the set of labels of vertices such
 *         that this graph includes an edge from source to that vertex,
 *         not only the directed edges, but also the undirected edges. and
 *         the value for each key is the (nonzero) weight of the edge from
 *         source to the key
 */
@Override
public Map<L, Double> targets(L source) {
    Map<L, Double> targets = new HashMap<>();
    for(E x: allEdges) {
        if(x.vertices().contains(source)) {
            for(Vertex y: x.targetVertices()) {
                if(!y.equals(source)) // avoid to put itself into the map
                    targets.put((L)y, x.getWeight());
            }
        }
    }
    return targets;
}

```

这个针对于有向图而言，返回起点或是终点，但是对于无向图，将两

个点都返回即可，遍历 list 去寻找所有的点就可以。

(5) addEdge () / removeEdge ()

加边和删边操作，在 allEdges 中进行删除操作即可。

```
/*
 * Add an edge to the graph, including the hyper edge.
 * @param edge a label of the edge
 * @return true if the graph didn't have the edge before; otherwise is false
 *         (and this graph is not modified)
 */
@Override
public boolean addEdge(E edge) {
    if(!allEdges.contains(edge)) {
        allEdges.add(edge);
        return true;
    }
    else return false;
}

/*
 * Delete an edge to the graph, including the hyper edge.
 * @param edge a label of the edge
 * @return true if the graph have the edge; otherwise is false
 *         (and this graph is not modified)
 */
@Override
public boolean removeEdge(E edge) {
    if(allEdges.contains(edge)) {
        allEdges.remove(edge);
        return true;
    }
    else return false;
}
```

(6) edges ()

只需要将 allEdges 中的元素复制到一个 list 中，再返回该 list 即可。

```
/*
 * Get all the edges in this graph.
 * @return the set of labels of edges in this graph
 */
@Override
public Set<E> edges() {
    Set<E> edges = new HashSet<>();
    for(E x: allEdges) {
        edges.add(x);
    }
    return edges;
}
```

3.10.1 单词网络 GraphPoet

GraphPoetApp 如下：

```
GraphPoetApp [Java Application] /opt/java/bin/java (2018年5月6日 下午4:13:06)
-----GraphPoetApp-----
1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit
-----Designed by muty

Please choose a function(1 - 6) and input the number.
1
Function 1, please input the path to concrete the graph.
src/GraphPoet.txt
Concrete graph successfully!
Please choose a function(1 - 6) and input the number.
```

首先一定要调用第一个功能，因为首先一定要生成图，然后才能进行后续操作。其次是一些图的指令，也就是之前提到过的加点加边等操作，利用正则表达式来进行解析，从而通过 parseAndExecute 类来进行相应的操作。3 可以用来进行对于点的 centrality 计算，具体操作如下图：

```
Please choose a function(1 - 6) and input the number.
3
Function 3, please input the label of the vertex.
b
The closeness of the b is 1.0
The betweenness of the b is 0.5
The degree centrality of the b is 0.6666666666666666
The eccentricity of the b is 1.0
The indegree centrality of the b is 2.0
The outdegree centrality of the b is 2.0
Please choose a function(1 - 6) and input the number.
```

第 4 部分也就是对于图的一些相关属性来进行计算，操作如下图：

```
Please choose a function(1 - 6) and input the number.
```

```
4
```

```
Function 4, you do not need make any input.
```

```
Graph centrality: 0.0
```

```
Graph radius: 1.0
```

```
Graph diameter: 2.0
```

第 5 部分可以进行最短距离的计算，操作如下图：

```
Please choose a function(1 - 6) and input the number.
```

```
5
```

```
Function 5: calculate the shortest distance.
```

```
Please input the source label
```

```
c
```

```
Please input the target label
```

```
d
```

```
The shortest distance is 1.0
```

最后是退出程序的部分，如下图：

```
Please choose a function(1 - 6) and input the number.
```

```
6
```

```
Thank you for using.
```

剩下的三个 App 文件也是类似的运行方式，不再提供具体的截图。

3.10.2 微博社交网络 SocialNetwork

```
SocialNetworkApp [Java Application] /opt/java/bin/java (2018年5月6日 下午4:46:03)
```

```
-----SocialNetworkApp-----
```

1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit

```
-----Dedigned by muty
```

```
Please choose a function(1 - 6) and input the number.
```

功能：与之前的相类似，在 GraphPoet 中已经详细阐述过。

关于这个图，我有一个疑问，就是实验手册中说加边删边都要修改其

他边的权重，那么在读文件依次加边的时候，是不是也要修改其他边的权重，如果修改的话，那么也就是说最后得到的图和文件中的图完全不一样，还有，这样的话一定要保证第一条边权重为 1，可是在解析文件的时候很难保证会先添加那条权重为 1 的边，一旦不能先添加权重为 1 的边，整个图的权重和就不再是 1。

3.10.3 网络拓扑图 NetworkTopology

```
NetworkTopologyApp [Java Application] /opt/java/bin/java (2018年5月6日 下午4:47:1)
----- NetworkTopologyApp -----
1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit
----- Dedigned by muty

Please choose a function(1 - 6) and input the number.
```

功能：与之前的相类似，在 GraphPoet 中已经详细阐述过。

3.10.4 电影网络 MovieGraph

```
MovieGraphApp [Java Application] /opt/java/bin/java (2018年5月6日 下午4:49:50)
----- MovieGraphApp -----
1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit
----- Dedigned by muty

Please choose a function(1 - 6) and input the number.
```

功能：与之前的相类似，在 GraphPoet 中已经详细阐述过。

3.11 应对四个应用面临的新变化（任选两个）

3.11.1 单词网络 GraphPoet

GraphPoetApp 中新加入一个功能，目的就是设置最小边权重，从而将所有边权重小于 minWeight 的边删掉

```
-----GraphPoetApp-----
1. Input the filepath of graph
2. Command for graph
3. Vertex centrality
4. Graph centrality, radius and diameter
5. Shortest distance between two vertices
6. Exit
7. Filter the edge
-----Designed by muty
```

Please choose a function(1 - 7) and input the number.

3.11.2 微博社交网络 SocialNetwork

向 Person.java 中新增方法

```
/**
 * set the weight
 */
public void setWeight(double weight) {
    personWeight = weight;
}

/**
 * Get the weight of the person
 * @return
 */
public double getWeight() {
    return personWeight;
}
```

新建 Map，存储节点的权重

```
case 7:
    Map<Vertex, Double> weightMap = new HashMap<>();
    for(Vertex x: graph.vertices()) {
        weightMap.put(x, GraphMetrics.inDegreeCentrality(graph, x));
    }
}
```

3.11.3 网络拓扑图 NetworkTopology

新建一个 WirelessRouter 类即可，只是需要在解析命令的时候多加一段代码用来新建 WirelessRouter 类的节点就好，改动不是特别大

```
if (type.equals("WirelessRouter")) {
    WirelessRouter new_vertex = new WirelessRouter(label);
    graph.addVertex(new_vertex);
}
```

新建 WirelessRouter 类，修改 VertexFactory，修改 Helper 目录下的 parseCommandHelper.java，修改 factory 目录下的 NetworkTopology.java

3.11.4 电影网络 MovieGraph

1. 创建 MovieActorRelation 类，继承自 UndirectedEdge 类。
2. EdgeFactory 的静态工厂方法中增加该类的一个分支
3. ParseAndExecute 中添加边的方法中，如果图为 MovieGraph 类型，在可以加入的边的类型中加入 MoviePublicRelation

4 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
----	-----	------	--------

5 实验过程中遇到的困难与解决途径

有关于接口和泛型的一些处理还是处理的不太熟，所以就花了不少时间，还有就是关于 centrality 那里的计算，并且这个实验的工作量比较大，用了很多时间。

6 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？
- (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？
- (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？
- (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？
- (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？
- (6) 关于本实验的工作量、难度、deadline。

关于这门实验，首先是老师对很多细节考虑不周，但是我们理解，毕竟只是几个老师来想试验，也是第一次开这门课，这我们完全

理解，但是在 QQ 群中对要求一改再改，导致昨晚和同学交流才发现有很多要求都被修改了，代码也要重新写，这让我表示很气愤，实验手册没有说清楚，按照自己的理解辛辛苦苦写好的代码又要改，改来改去，并且，所有要求都是在聊天记录中呈现的，还得翻记录，经常有老师不能自圆其说的现象发生，另外，有同学要求说将修改的一些要求写个文件放到群里，老师当时答应了，可是知道现在（截止日期前一天）还没有看到这样的文件，希望老师可以今后可以改正。

(7) 到目前为止你对《软件构造》课程的评价。