哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2018 年春季学期

计算机学院大二软件构造课程

Lab 2 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

本次试验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现 ADT。具体来说：

（1）针对给定的应用问题，从问题中描述识别所需的 ADT；

（2）设计 ADT 规约，并评估质量

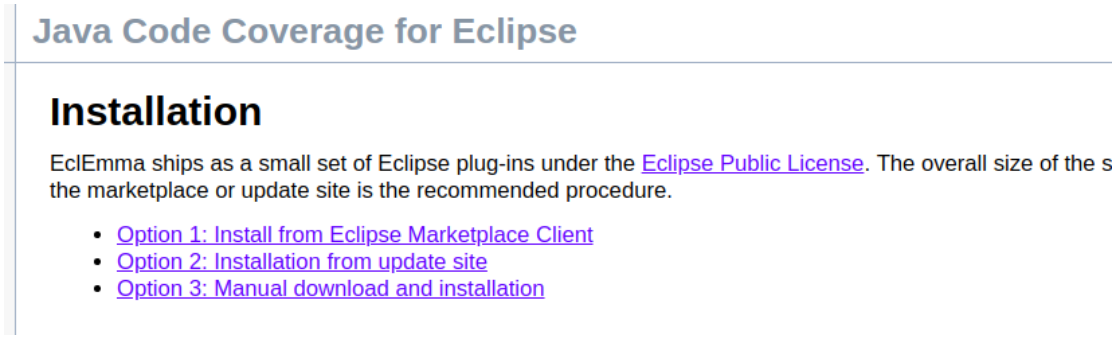
（3）根据 ADT 规约设计测试用例

（4）根据规约设计 ADT 的多种实现，针对每种实现，设计其表示、表示不变性、抽象过程

（5）使用 OOP 实现 ADT，并判定表示不变性是否违反、各实现是否存在表示泄露

2 实验环境配置

本实验要求配置插件 EclEmma，官网提供了三种安装方法，任选其一即可



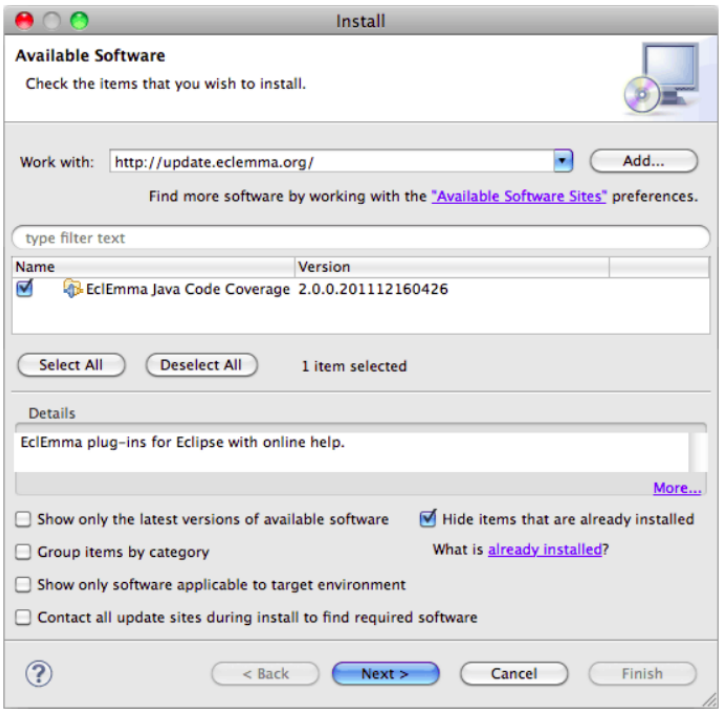
我选择了第二种安装方法，也就是从他给的网址下载压缩包，然后按照指引一路导入到 Eclipse 中即可，比较简单，并不是很难，安装流程官网也已经给出，按照指引一步一步去操作就好

Option 2: Installation from Update Site

The update site for EclEmma is **http://update.eclemma.org/**. Perform the following steps to install EclEmma from the update site:

1. From your Eclipse menu select *Help* → *Install New Software...*

2. In the *Install* dialog enter http://update.eclemma.org/ at the *Work with* field.



3. Check the latest EclEmma version and press *Next*

4. Follow the steps in the installation wizard.

下面是我仓库的网址：

https://github.com/ComputerScienceHIT/Lab2-1160301008

3 实验过程

3.1 Poetic Walks

首先要实现 concreteVerticesGraph 和 concreteEdgesGraph 两种生成图的方法，对于图中顶点或边的操作都是继承自 Graph 接口然后利用他们来生成我们想要的传统意义的图，然后，参照 Poetic Walk 中的描述来进行一个类似于扩句的操作，达到所谓的“创作出更美妙的诗篇”。

From there, given this dry bit of business-speak input:

> Seek to explore new and exciting synergies!

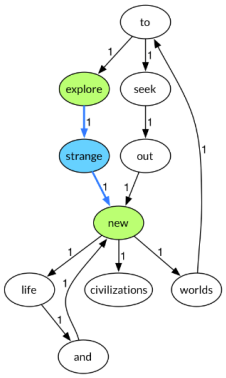
the poet will use the graph to generate a poem by inserting (where possible) a *bridge word* between each pair of input words:

w_1		w_2	path?	bridge word
seek	→	*to*	no two-edge-long path from *seek* to *to*	—
to	→	*explore*	no two-edge-long path from *to* to *explore*	—
explore	→	*new*	one two-edge-long path from *explore* to *new* with weight 2	*strange*
new	→	*and*	one two-edge-long path from *new* to *and* with weight 2	*life*
and	→	*exciting*	no vertex *exciting*	—
exciting	→	*synergies!*	no vertices *exciting* or *synergies!*	—

Hover over rows above to visualize poem generation on the right. In this example, the outcome is deterministic since no word pair has multiple bridge words tied for maximum weight. The resulting output poem is:

> Seek to explore strange new life and exciting synergies!

Exegesis is left as an exercise for the reader.



3.1.1 Get the code and prepare Git repository

使用指令:

`git clone https://github.com/rainywang/Spring2018_HITCS_SC_Lab2.git`

将源码复制到本地仓库.

3.1.2 Problem 1: Test `Graph <String>`

首先，根据要求，要完善 GraphInstanceTest 操作，来达到实验要求，每个测试方法都要先按照 Graph 接口中每个方法所给出的 spec 来进行测试。

（1）set 方法测试：

```
/**
 * Add, change, or remove a weighted directed edge in this graph.
 * If weight is nonzero, add an edge or update the weight of that edge;
 * vertices with the given labels are added to the graph if they do not
 * already exist.
 * If weight is zero, remove the edge if it exists (the graph is not
 * otherwise modified).
 *
 * @param source label of the source vertex
 * @param target label of the target vertex
 * @param weight nonnegative weight of the edge
 * @return the previous weight of the edge, or zero if there was no such
 *         edge
 */
public int set(L source, L target, int weight);
```

根据 set 函数的要求，要求如果输入的边不存在的话，则新建一条边加入到图中，并且返回 0，如果存在这条边，那么修改这条边的值，并返回接旧的权重值，如果传进去的参数为 0，那么相当于删除掉这条边，于是我建立了下面的测试文件：

```
    @Test
    public void testSet() {
        Graph<String> graph = emptyInstance();
        graph.add("source");
        graph.add("target");
        assertEquals(0, graph.set("source", "target", 5));
        assertEquals(5, graph.set("source", "target", 6));
        assertEquals(6, graph.set("source", "target", 0));
        assertEquals(0, graph.set("source", "target", 5));

    }
```

和如下的 test strategy

```
//   - testSet(): New an Edge object, then set the weight, first-time add must return 0,
//     then set a different weight, the method will return the old weight, at last set
//     the weight as 0, which will delete the edge, at this time, if we set this edge with
//     a weight again, it will return 0 again.
```

新建一条边，然后加入到图中，测试是不是返回 0，然后修改这条边的权重，然后修改这条边的权重，观察是不是返回旧的权重值，然后删除这条边，判断是否加入成功。至此，这个方法就测试完毕。

（2）add 方法测试：

```
/**
 * 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 add(L vertex);
```

加入一个点，若图中已存在这个点，那么就返回 false，否则就将这个点加入到图中，至此，这是 add（）方法的功能实现。

所以我写了如下的测试文件：

```
@Test
public void testAdd() {
    Graph<String> graph = emptyInstance();
    assertTrue("expected true here", graph.add("vertex"));
    assertFalse(graph.add("vertex"));
    assertTrue("expected flase here", graph.add("vertex1"));
}
```

和如下的 test strategy

```
//    - testAdd(): Add a vertex to the graph for the first time, then add the same
//      label vertex for the second time, expected false this time. Then add a different
//      label vertex, still return true;
```

只需要加入一个存在的点判断返回值为 false，再加入一个新的点判断返回 true 即可。

（3）source（）/target（）方法测试：

```
/**
 * Get the source vertices with directed edges to a target vertex and the
 * weights of those edges.
 *
 * @param target a label
 * @return a map where the key set is the set of labels of vertices such
 *         that this graph includes an edge from that vertex to target, and
 *         the value for each key is the (nonzero) weight of the edge from
 *         the key to target
 */
public Map<L, Integer> sources(L target);

/**
 * 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, and
 *         the value for each key is the (nonzero) weight of the edge from
 *         source to the key
 */
public Map<L, Integer> targets(L source);
```

这两个方法类似，可以说是互补方法，分别是返回所有以这个点为起点的边的另一个节点，和所有以这条边为终点的边的另一个节点。

所以我有如下的测试文件：

```
public void testSources() {
    Graph<String> graph = emptyInstance();
    assertTrue(graph.add("a"));
    assertTrue(graph.add("b"));
    assertTrue(graph.add("c"));
    assertTrue(graph.add("d"));
    assertEquals(0, graph.set("a", "b", 3));
    assertEquals(0, graph.set("a", "d", 5));
    assertEquals(0, graph.set("c", "d", 4));
    assertEquals(0, graph.set("b", "d", 6));
    assertTrue(graph.sources("d").containsKey("a"));
    assertTrue(graph.sources("d").containsKey("b"));
    assertTrue(graph.sources("d").containsKey("c"));

}
@Test
public void testTargets() {
    //Graph<String> graph = emptyInstance();
    Graph<String> graph = emptyInstance();
    assertTrue(graph.add("1"));
    assertTrue(graph.add("2"));
    assertTrue(graph.add("3"));
    assertTrue(graph.add("4"));
    assertEquals(0, graph.set("1", "2", 3));
    assertEquals(0, graph.set("1", "4", 5));
    assertEquals(0, graph.set("3", "4", 4));
    assertEquals(0, graph.set("2", "4", 6));
    assertTrue(graph.targets("1").containsKey("2"));
    assertTrue(graph.targets("1").containsKey("4"));
    assertTrue(graph.targets("2").containsKey("4"));
}
```

和如下的 test strategy

```
//   - testSources()(testTargets()): Add some edges to the graph, then check the set sources(targets) is okay.
```

（4）remove（）方法测试

```
/**
 * Remove a vertex from this graph; any edges to or from the vertex are
 * also removed.
 *
 * @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 remove(L vertex);
```

这个方法就是要首先从顶点集中删除掉这个节点，然后再删除掉所有和这个节点相关的边，无论是以这条边为起点的边，还是以这条边为终点的边，都要删除掉，这样就可以来进行测试。

有如下的测试文件：

```
@Test
public void testRemove() {
    Graph<String> graph = emptyInstance();
    assertTrue(graph.add("vertex2"));
    assertTrue(graph.remove("vertex2"));
    assertFalse(graph.remove("vertex3"));
}
```

并有如下的 test strategy

```
//   - testRemove(): Add some vertices to the graph, then call the method remove() to remove
//     them, then check the return bool value is okay.
```

3.1.3 Problem 2: Implement `Graph <String>`

这里要针对两种生成图的方法来进行图的实现，并编写 AF、RI、checkrep（）等

3.1.3.1 Implement `ConcreteEdgesGraph`

首先是对 Edge 类进行编写，实现如下：

```
class Edge<L> {

    // Abstraction function:
    //    the class Edge represents a edge in the graph, every edge is from source to target,
    //    also has a weight which is more than 1 or equal to 1.
    // Representation invariant:
    //    source is a non-null L, which represents the begin of a edge.
    //    target is a non-null L, which represents the end of a edge.
    //    weight is an integer more than 1 or equal to 1, which represents the length of a edge.
    //    According to the spec, L must be immutable type.
    //    For example: source = "a"; target = "b", weight = 5
    //    so the directed graph must contains edges below:
    //    a ---5--> b
    // Safety from rep exposure:
    //    All fields are private and final.
    //    source and target are both immutable
    //    may be need defensive copy
    private final L source, target;
    private final int weight;

    public Edge(L source, L target, int weight) {
        assert weight > 0;
        this.source = source;
        this.target = target;
        this.weight = weight;
    }
    private void checkRep() {
        assert source != null;
        assert target != null;
        assert weight >= 0;
    }
```

```
    public L getSource() {
        L s = source;
        return s;
    }
    public L getTarget() {
        L t = target;
        return t;
    }
    public int getWeight() {
        return weight;
    }

    public String toString() {
        return (source + "->" + target + weight + " ");
    }
}
```

Edge 类的编写很多是针对于 Graph 的操作，Edge 类中的成员变量有 :L source，L target， int weight，这样有了起点、终点、每条边的权重就可以唯一确定一条边。

接下来是对 ConcreteEdgesGraph 类的编写，根据 Graph 中的 spec 来进行编写

```java
public class ConcreteEdgesGraph<L> implements Graph<L> {

    private final Set<L> vertices = new HashSet<>();
    private final List<Edge<L>> edges = new ArrayList<>();

    // Abstraction function:
    //    The class "Edge" represents the edge in the graph, every edge has a source vertex,
    //    a target vertex and a weight, different edges can have the same weight.
    //    The ConcreteEdgesGraph can create a graph based on Edge.
    // Representation invariant:
    //    vertices is a set of the object L
    //    edges is a list of all the edges in the graph, which can not be repeated, and the
    //    weight must be an integer greater than 1 or equal to 1
    //    vertices.size() must be greater than edges.size() * 2 or equal to edges.size() * 2
    // Safety from rep exposure:
    //    All fields are private and final
    //    vertices and edges are both mutable, so the methods must have defensive copies to avoid sharing
    //    fields with client, such as the method "vertices()".
```

首先明确，vertices 是一个存着所有节点的集合，edges 是一个存着所有边的 list，这样子，一个图就可以这样被表示出来。下面是各个方法的具体实现：

（1）add（）方法就只要像 vertices 中来进行加顶点就好,只需要在加入定点之前来进行判断一下就好，若图中已经存在这个点，那么就返回 false，否则就可以加入，其实运用 Hashset 中的元素不重复的性质就可以实现。

```java
@Override
public boolean add(L vertex) {
    // throw new RuntimeException("not implemented");
    if (!vertices.contains(vertex)) {
        vertices.add(vertex);
        return true;
    }

    else
        return false;
}
```

（2）set 方法首先就要判断图中是不是已经存在这条边，如果存在的话， 那么就找到这条边并修改边的权重，并返回旧的权重值。若不存在这条边并且 weight 参数不为 0，那么就在图中新建这条边，并返回 0。如果 weight 参数为 0，那么将该边删除。

```
@Override
public int set(L source, L target, int weight) {
    // throw new RuntimeException("not implemented");
    //assert weight >= 0;

    if (weight > 0) {
        for (int i = 0; i < edges.size(); ++i) {
            if (edges.get(i).getSource().equals(source) && edges.get(i).getTarget().equals(target)) {
                int previousWeight = edges.get(i).getWeight();
                edges.remove(i);
                Edge<L> e = new Edge<L>(source, target, weight);
                edges.add(i, e);
                //edges.get(i).setWeight(weight);
                return previousWeight;
            }
        }

        Edge<L> edge = new Edge<L>(source, target, weight);
        //edge.checkRep();
        edges.add(edge);

        return 0;
    }
    else {

        for (int i = 0; i < edges.size(); ++i) {
            if (edges.get(i).getSource().equals(source) && edges.get(i).getTarget().equals(target)) {
                int previousWeight = edges.get(i).getWeight();
                edges.remove(edges.get(i));

                return previousWeight;
            }
        }
```

（3）vertices 方法在这个类中比较好写，只要将成员变量 vertices 进行返回即可，注意 set 为 mutable 型数据结构，所以需要防止表示暴露，进行 defensive copy。

```
@Override
public Set<L> vertices() {
    // throw new RuntimeException("not implemented");
    Set<L> verticesCopy = new HashSet<>();
    for(L x: vertices)
        verticesCopy.add(x);
    return verticesCopy;
}
```

（4）sources 方法/targets 方法，只需要遍历 edges list，来进行寻找所有以该点为起点的边或以该点为终点的边就好，这两个方法算是互补方法。

```java
@Override
public Map<L, Integer> sources(L target) {
    // throw new RuntimeException("not implemented");
    Map<L, Integer> sources = new HashMap<>();
    for (int i = 0; i < edges.size(); ++i) {
        if (edges.get(i).getTarget().equals(target)) {
            sources.put(edges.get(i).getSource(), edges.get(i).getWeight());
        }
    }
    return sources;
}

@Override
public Map<L, Integer> targets(L source) {
    // throw new RuntimeException("not implemented");
    Map<L, Integer> targets = new HashMap<>();
    for (int i = 0; i < edges.size(); ++i) {
        if (edges.get(i).getSource().equals(source)) {
            targets.put(edges.get(i).getTarget(), edges.get(i).getWeight());
        }
    }
    return targets;
}
```

（5）remove 方法主要就是先将该点从 vertices 中移除掉，然后将所有与该点相关联的边也都删除掉。

```java
@Override
public boolean remove(L vertex) {
    // throw new RuntimeException("not implemented");
    if (!vertices.contains(vertex)) {
        return false;
    }
    else {
        vertices.remove(vertex);
        for (int i = 0; i < edges.size(); ++i) {
            if (edges.get(i).getSource().equals(vertex) || edges.get(i).getTarget().equals(vertex))
                edges.remove(edges.get(i));
        }
        return true;
    }
}
```

（6）toString 方法就是将该图以 String 的形式输出，这样就可以更加直观的显示这张图，

```java
@Override
public String toString() {
    if (vertices.isEmpty()) {
     return "The graph is empty!";
 }
    else if(edges.isEmpty()) {
        return "The graph has no edges!";
    }
     else {
         String s = new String();
         for(int i = 0; i < edges.size(); ++i) {
             s += edges.get(i).toString();
         }
         return s;
     }
 }
```

这里我为每个节点写了一个 toString 方法，用来输出每个节点信息，在这里可以

循环调用。

3.1.3.2 Implement `ConcreteVerticesGraph`

实现过 ConcreteEdgesGraph 之后，这个类和那个类似，思路相对来说比较明确，

首先还是 Vertex 类的编写

```
class Vertex<L> {


    // Abstraction function:
    //    Vertex represents a vertex in the directed graph.
    //    name is a L type object which means the label of a vertex.
    //    sources is a map, which stored the edge whose source is this vertex.
    //    targets is a map, which stored the edge whose target is this vertex.
    //    For example: name = "a"; sources = [<"b", 5>]; targets = [<"c", 3>];
    //    so the graph must contains two edges below:
    //    a --5--> b     c --3--> a
    // Representation invariant:
    //    Vertex label must be non-null
    //    A vertex can be neither its own source nor target.
    //    weight must be an integer more than 1 or equal to 1.
    //    A directed graph cannot have repeated edges.
    // Safety from rep exposure:
    //    Fields must be private and final.
    //    According to the spec, L must be immutable.
    //    sources and targets are both mutable type, so operations must have defensive
    //    copies to avoid sharing fields with client.

    private final L name;
    private final Map<L, Integer> sources = new HashMap<>();
    private final Map<L, Integer> targets = new HashMap<>();
    public Vertex(L name) {
        ...
```

每个 Vertex 类中，都有 L name， 来代表这个节点的标签，然后有一个 sources 和 targets 的 map 来相应的存取边，分别代表以该边为起点的边和以该边为终点的边。

```
    }
    private void checkRep() {
        assert !sources.keySet().contains(name);
        assert !targets.keySet().contains(name);
    }
    public L getName() {
        return name;
    }
    public void setSourceWeight(L target, int weight) {
        if(sources.containsKey(target)) {
            sources.remove(target);
        }
        sources.put(target, weight);
    }
    public void setTargetWeight(L source, int weight) {
        if(targets.containsKey(source)) {
            targets.remove(source);
        }
        targets.put(source, weight);
    }
    public void removeSourceEdge(L target) {
        sources.remove(target);
    }
    public void removeTargetEdge(L source) {
        targets.remove(source);
    }
    public Map<L, Integer> getSources() {
```

```
public Map<L, Integer> getTargets() {
    return targets;
}
public String toString() {
    String a = new String();
    for(L x: sources.keySet()) {
        a += name + "->" + x + sources.get(x) + " ";
    }
    return a;
}
```

其次是对 ConcreteVerticesGraph 类的实现：

```
public class ConcreteVerticesGraph<L> implements Graph<L> {

    private final List<Vertex<L>> vertices = new ArrayList<>();

    // Abstraction function:
    //    Represents the directed graph with vertices with weight.
    //    Every edge can be represented by the start point and the end point.
    // Representation invariant:
    //    Only one instance of a vertex can exist in vertices.
    //    Weight must be an integer more than 1 or equal to 1. A graph can not
    //    have repeated edges.
    // Safety from rep exposure:
    //    vertices is a mutable list, so must make defensive copies to avoid
    //    sharing fields with client.
    //    Vertex is also a mutable type, so still need the defensive copies to
    //    avoid sharing with client.
    public ConcreteVerticesGraph() {

    }
```

首先是 AF 、RI 、rep 的编写，以下是各个方法的编写：

上面已经实现了对 ConcreteEdgesGraph 的编写，这样每个方法都已经熟悉了思路，只需要针对不同的数据结构，也就是 Vertex 类来进行编写，难度不是特别的高。

（1）add（）方法实现：

```
@Override public boolean add(L vertex) {
    //throw new RuntimeException("not implemented");
    for(int i = 0; i < vertices.size(); ++i) {
        if(vertices.get(i).getName().equals(vertex))
            return false;
    }
    Vertex<L> v = new Vertex<L>(vertex);
    vertices.add(v);
    return true;
}
```

和之前一样，加点之前先遍历一遍所有的点集，如果之前没加过该点，那么将其加入到点集中，并返回 true，否则返回 false。

（2）

```
@Override public int set(L source, L target, int weight) {
    //throw new RuntimeException("not implemented");
    int previousWeight = 0, flag = 0;
    if(weight > 0) {
        for(int i = 0; i < vertices.size(); ++i) {
            if(vertices.get(i).getName() == source) {    //设置source中的以source为起点的边
                flag = 0;
                Iterator<Map.Entry<L, Integer>> it = vertices.get(i).getSources().entrySet().iterator();
                while (it.hasNext()) {
                    Map.Entry<L, java.lang.Integer> entry = (Map.Entry<L, java.lang.Integer>) it
                            .next();
                    if(entry.getKey().equals(target)) {  //之前含有这条边，修改weight就好
                        previousWeight = entry.getValue();
                        vertices.get(i).setSourceWeight(target, weight);
                        flag = 1;
                    }
                }
                if(flag == 0) {
                    vertices.get(i).setSourceWeight(target, weight);
                }
                flag = 0;
            }
            else if(vertices.get(i).getName() == target) {  //设置target中以target为终点的边
                flag = 0;
                Iterator<Map.Entry<L, Integer>> it = vertices.get(i).getTargets().entrySet().iterator();
                while (it.hasNext()) {
                    Map.Entry<L, java.lang.Integer> entry = (Map.Entry<L, java.lang.Integer>) it
```

```
                while (it.hasNext()) {
                    Map.Entry<L, java.lang.Integer> entry = (Map.Entry<L, java.lang.Integer>) it
                            .next();
                    if(entry.getKey().equals(source)) {  //这条边已存在 修改weight
                        previousWeight = entry.getValue();
                        vertices.get(i).setTargetWeight(source, weight);
                        flag = 1;
                    }
                }
                if(flag == 0) {
                    vertices.get(i).setTargetWeight(source, weight);
                }
                flag = 0;
            }
            else continue;
        }
    }
    else {
        for(int i = 0; i < vertices.size(); ++i) {
            if(vertices.get(i).getName().equals(source)) {
                vertices.get(i).removeSourceEdge(target);
            }
            else if(vertices.get(i).getName().equals(target)) {
                vertices.get(i).removeTargetEdge(source);
            }
            else continue;
        }
    }
```

Set 方法和之前的一样，这里不再细说

（3）remove 方法实现

```
@Override public boolean remove(L vertex) {
    //throw new RuntimeException("not implemented");
    List<L> l = new ArrayList<>();
    for(int i = 0; i < vertices.size(); ++i)
        l.add(vertices.get(i).getName());
    if(!l.contains(vertex))
        return false;
    else {
    for(int i = 0; i < vertices.size(); ++i) {
        if(vertices.get(i).getName().equals(vertex))
            vertices.remove(vertices.get(i));
        else {
            Iterator<Map.Entry<L, Integer>> it1 = vertices.get(i).getSources().entrySet().iterator();
            while (it1.hasNext()) {
                Map.Entry<L, java.lang.Integer> entry = (Map.Entry<L, java.lang.Integer>) it1
                        .next();
                if(entry.getKey().equals(vertex)) {
                    vertices.get(i).removeSourceEdge(vertex);
                }
            }
            Iterator<Map.Entry<L, Integer>> it2 = vertices.get(i).getTargets().entrySet().iterator();
            while (it2.hasNext()) {
                Map.Entry<L, java.lang.Integer> entry = (Map.Entry<L, java.lang.Integer>) it2
                        .next();
                if(entry.getKey().equals(vertex)) {
                    vertices.get(i).removeTargetEdge(vertex);
                }
```

这里的 remove 方法相对来说比较长，是因为数据结构的改变，造成了遍历的时候不太方便，所以代码会变长。

（4）vertices（）方法实现

```
@Override public Set<L> vertices() {
    //throw new RuntimeException("not implemented");
    Set<L> allVertices = new HashSet<>();
    for(int i = 0; i < vertices.size(); ++i) {
        allVertices.add(vertices.get(i).getName());
    }
    return allVertices;
}
```

直接返回成员变量 vertices 即可，注意 Set 是 mutable 变量，所以为了防止表示暴露，要进行 defensive copy。

（5）source（）/target（）方法实现

```
    @Override public Map<L, Integer> sources(L target) {
        //throw new RuntimeException("not implemented");
        for(int i = 0; i < vertices.size(); ++i) {
            if(vertices.get(i).getName().equals(target)) {
                return vertices.get(i).getTargets();
            }
        }
        Map<L, Integer> map = new HashMap<>();
        return map;
    }

    @Override public Map<L, Integer> targets(L source) {
        //throw new RuntimeException("not implemented");
        for(int i = 0; i < vertices.size(); ++i) {
            if(vertices.get(i).getName().equals(source)) {
                return vertices.get(i).getSources();
            }
        }
        Map<L, Integer> map = new HashMap<>();
        return map;
    }
```

同样的是互补方法

（6）toString 方法

```
@Override public String toString() {
    if(vertices.size() == 0) {
        return "The graph is empty!";
    }
    else if(vertices.size() != 0 && haveNoEdges()) {
        return "The graph has no edges!";
    }
    else {
    String a = new String();
    for(Vertex<L> v: vertices) {
        a += v.toString();
    }
    return a;
    }
}
```

这里和上面的 toString 方法思路一样，都是调用 Vertex 类里面的 toString，然后

空图和没有边的时候会进行特判，从而特殊输出。

3.1.4 Problem 3: Implement generic `Graph<L>`

这里要实现泛型，也就是不能只用 String 来表示一个点的标签，也可以换成别的数据结构，比如 Integer 等。具体操作网站上已经给的很详细

3.1. Make the implementations generic

a. Change the declarations of your concrete classes to read:

```
public class ConcreteEdgesGraph<L> implements Graph<L> { ... }

class Edge<L> { ... }
```

and:

```
public class ConcreteVerticesGraph<L> implements Graph<L> { ... }

class Vertex<L> { ... }
```

3.1.4.1 Make the implementations generic

就像 MIT 网站给的样例一样，只需要将 `String` 这个标签换成 `L` 就可以实现泛型

```
public class ConcreteVerticesGraph<L> implements Graph<L> {

    private final List<Vertex<L>> vertices = new ArrayList<>();
```

```
public class ConcreteEdgesGraph<L> implements Graph<L> {

    private final Set<L> vertices = new HashSet<>();
    private final List<Edge<L>> edges = new ArrayList<>();
```

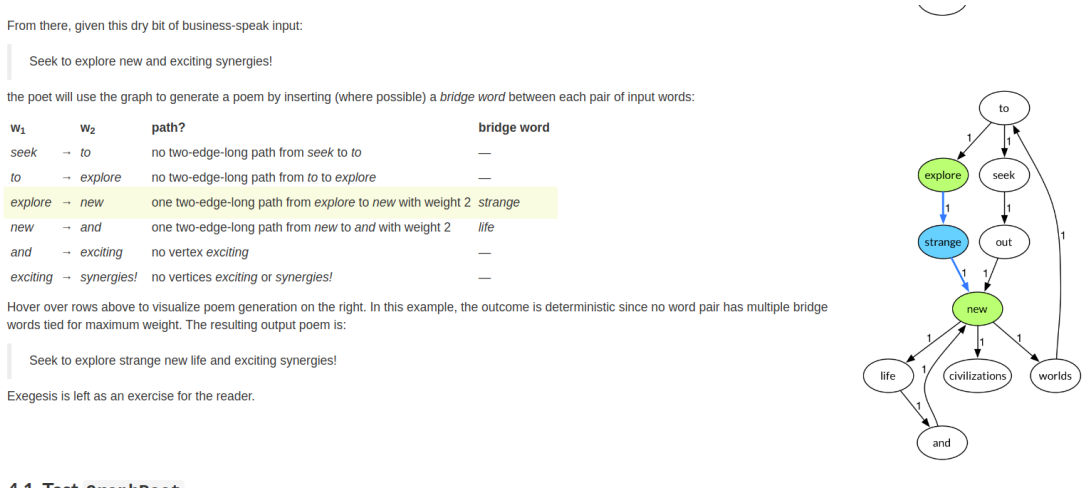
3.1.4.2 Implement `Graph.empty()`

这里可以利用我们已经写好的类来进行实现，我选择了 ConcreteEdgesGraph 类

```java
/**
 * Create an empty graph.
 *
 * @param <L> type of vertex labels in the graph, must be immutable
 * @return a new empty weighted directed graph
 */
public static <L> Graph<L> empty() {
    //throw new RuntimeException("not implemented");
    return new ConcreteEdgesGraph<>();

}
```

3.1.5 Problem 4: Poetic walks

实现创造”优美诗句“，也就是类似于扩句一样的功能，也就是已经有一个建好的图，然后输入一句话或者一段话，然后挨着的两个单词只要是在图中有一条距离为 2 的单相路，就将中间的那个元素插入到两个单词之间，具体如下图



From there, given this dry bit of business-speak input:

> Seek to explore new and exciting synergies!

the poet will use the graph to generate a poem by inserting (where possible) a *bridge word* between each pair of input words:

w_1		w_2	path?	bridge word
seek	→	*to*	no two-edge-long path from *seek* to *to*	—
to	→	*explore*	no two-edge-long path from *to* to *explore*	—
explore	→	*new*	one two-edge-long path from *explore* to *new* with weight 2	*strange*
new	→	*and*	one two-edge-long path from *new* to *and* with weight 2	*life*
and	→	*exciting*	no vertex *exciting*	—
exciting	→	*synergies!*	no vertices *exciting* or *synergies!*	—

Hover over rows above to visualize poem generation on the right. In this example, the outcome is deterministic since no word pair has multiple bridge words tied for maximum weight. The resulting output poem is:

> Seek to explore strange new life and exciting synergies!

Exegesis is left as an exercise for the reader.

在 explore 和 new 之间有一条长度为 2 的路，所以就可以在两个单词之间加入 strange，seek 和 to 之间就没有这样的路，所以就不用加

3.1.5.1 Test `GraphPoet`

有如下的 test strategy

```
public class GraphPoetTest {

    // Testing strategy
    //   - wordsOfCorpusTest(): method wordsOfCorpus() will return a list of the words
    //     which are created by split file corpus by blank space, so, check some random
    //     words in the list is okay.
    //   - createGraphFromCorpusTest():
    //     check some properties of the created graph is okay, such as vertices.size()
    //     the position of a word in the list.
```

方法实现如下：

```
    File corpus = new File("src/P1/poet/mugar-omni-theater.txt");

    @Test(expected=AssertionError.class)
    public void testAssertionsEnabled() {
        assert false; // make sure assertions are enabled with VM argument: -ea
    }


    @Test
    public void wordsOfCorpusTest() throws IOException {
        GraphPoet gp = new GraphPoet(corpus);
        assertEquals(11, gp.getCorpusList().size());
        assertTrue(gp.getCorpusList().contains("this"));
        assertTrue(gp.getCorpusList().contains("system."));
    }
    @Test
    public void createGraphFromCorpusTest() throws IOException {
        GraphPoet gp = new GraphPoet(corpus);
        gp.createGraphFromCorpus(gp.getCorpusList());
        assertEquals(11, gp.getGraph().vertices().size());
        assertEquals(2, gp.getGraph().set("test", "of", 1));
   }
    }
```

3.1.5.2 Implement `GraphPoet`

```
*/
public class GraphPoet {

    private final Graph<String> graph = Graph.empty();
    private final List<String> corpusList = new ArrayList<>();
    // Abstraction function:
    //    Represents a poem generator, combined input string and the corpus,
    //    we can get a beautiful poem.
    // Representation invariant:
    //    graph is a non-null graph
    //    corpusList is a list contains all words in the corpus.
    // Safety from rep exposure:
    //    All fields are private and final.
    //    graph is a mutable type, so references to it are provided for the
    //    client to mutate.
    //    corpusList is a mutable list, so we must have a method getCorpusList()
    //    to avoid sharing the fields with client so that client cannot modify it.
```

有如上的 AF、RI、rep，corpusList 是用来存已知文本，来进行建图

```
public void wordsOfCorpus(File corpus) throws IOException{

    try(Scanner sc = new Scanner(new BufferedReader(new FileReader(corpus)))) {
        while(sc.hasNext()) {
            corpusList.add(sc.next().toLowerCase());
        }
    }
}
public void createGraphFromCorpus(List<String> list) {

    for(int i = 0; i < list.size() - 1; ++i) {
        graph.add(list.get(i));
        graph.add(list.get(i + 1));
        int previousWeight = graph.set(list.get(i), list.get(i + 1), 1);
        graph.set(list.get(i), list.get(i + 1), previousWeight + 1);

    }

}
```

以上两个方法用于根据已有的文本来进行图的生成。建图的时候注意，如果有两个单词 连续出现多次，那么出现几次，两个单词之间的边权重就为几。例如出现三次，那么权重就为 3.

```
/**
 * Generate a poem.
 *
 * @param input string from which to create the poem
 * @return poem (as described above)
 */
public String poem(String input) {
    String[] inputWords = input.split(" ");
    List<String> inputWordsList = new ArrayList<>();
    Map<Integer, String> map = new HashMap<>();
    for(int i = 0; i < inputWords.length; ++i) {
        inputWordsList.add(inputWords[i]);
    }
    for(int i = 0; i < inputWordsList.size() - 1; ++i) {
        for(String x: graph.targets(inputWordsList.get(i).toLowerCase()).keySet()) {
            if(graph.targets(inputWordsList.get(i).toLowerCase()).get(x).intValue() == 1) {
                for(String y: graph.targets(x).keySet()) {
                    if(y.equals(inputWordsList.get(i + 1).toLowerCase()) && graph.targets(x).get(y).intValue() == 1) {
                        map.put(i + 1, x);
                    }
                }
            }
            else continue;
        }
    }
    List<String> list = new ArrayList<>();
    for(int i = 0; i < inputWordsList.size(); ++i) {
        if(map.containsKey(i)) {
            list.add(map.get(i));
            map.remove(i);
            i -= 1;
        }
        else list.add(inputWordsList.get(i));
```

这里用来进行“诗意漫步”，也就是循环比较来进行单词的扩充

3.1.5.3 Graph poetry slam

（1）

corpus：

```
This is a test of the Mugar Omni Theater sound system.
```

输入及输出

```
Test the system.
>>>
Test of the system.
```

（2）corpus：

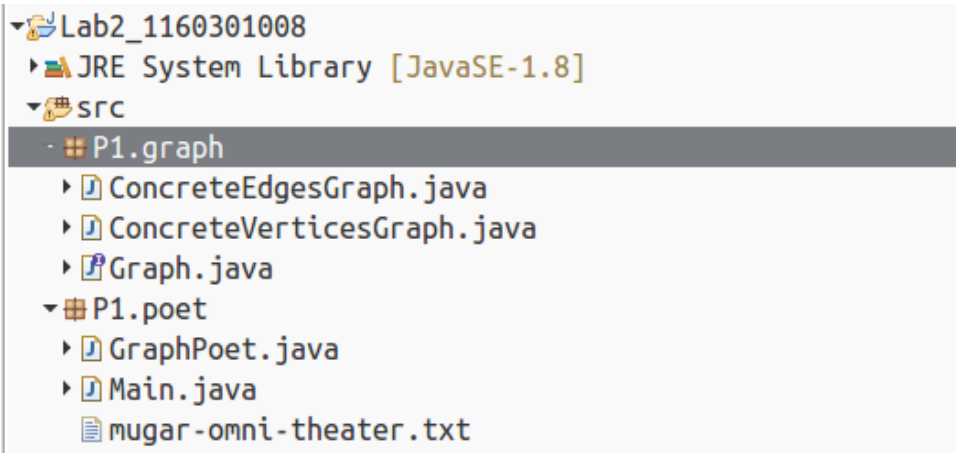
```
1 Do you think it is a very cool day?
2
```

输入及输出

```
Problems @ Javadoc Declaration Console Cover
<terminated> Main (3) [Java Application] /opt/java/bin/
I think it is a cool day
>>>
I think it is a very cool day
```

3.1.6 Before you're done

请按照 http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done 的说明，检查你的程序。



3.2 Re-implement the Social Network in Lab1

实验的这一部分主要是想要利用我们在 Poetic Walk 中实现的图来进行建图，重写实验一中 Social Network 部分

3.2.1 FriendshipGraph 类

首先，在 FriendGraph 类中有一个利用 P1 中曾经写好的 ConcreteVerticesGraph 类生成的图，如下图

```
Graph<Person> graph = new ConcreteVerticesGraph<>();
public void addVertex(Person p) {
    graph.add(p);
}
```

利用泛型，可以将 Graph 中的元素类型换成 Person 类。

（1）addVertex 方法：

```
Graph<Person> graph = new ConcreteVerticesGraph<>();
public void addVertex(Person p) {
    graph.add(p);
}
```

由于 graph 中有已经实现好的 add 方法，直接调用即可。

（2）addEdge 方法：

```
public void addEdge(Person p1, Person p2) {
    graph.set(p1, p2, 1);
}
```

同样可以利用实现好的 Graph 类中的方法，直接调用 graph.set（）即可。

（3）getDistance 方法：

和 lab1 中的思路一样，采取广度优先搜索即可搜索到最短路径，如果搜到最后还是搜不到最短路径，那么返回 -1，只不过遍历的时候更多的借助于 Graph 中实现过的方法，比如说 add 方法，可以说 P1 中的图由于内置方法设计的很

好，在这里实现一些功能十分方便，也体现了 ADT 的便捷。只不过这里和 lab1 中不一样的一点是，这里的 Person 类是 immutable 的，所以在图中我新添加了两个 map，一个来进行存储 flag 看是否被访问，另一个 map 来进行存储每个点的 dist 性质。

```
public int getDistance(Person p1, Person p2) {
    Iterator<Person> it = graph.vertices().iterator();
    Map<Person, Boolean> flag = new HashMap<>();
    Map<Person, Integer> dist = new HashMap<>();
    while (it.hasNext()) {
       Person person = (Person) it.next();
       flag.put(person, false);
       dist.put(person, 0);
   }
    Queue<Person> queue = new LinkedList<>();
    queue.add(p1);
    flag.put(p1,true);
    if(p1.equals(p2))
        return 0;
    while (!queue.isEmpty()) {
        for(Person p: graph.sources(queue.peek()).keySet()) {
            if(flag.get(p) == false) {
               queue.add(p);
               flag.put(p,true);
               dist.put(p, dist.get(queue.peek()) + 1);
               if(p.equals(p2)) {
                   return dist.get(p);
               }
           }
        }
        queue.remove();
    }
    return -1;
}
```

3.2.2 Person 类

```
package P2;

import java.util.ArrayList;

public class Person {
    private String name;    //名字

    public Person(String s) {
        this.name = s;
    }
}
```

只存储一个名字即可，实现类似于 P1 中的 String 功能即可。

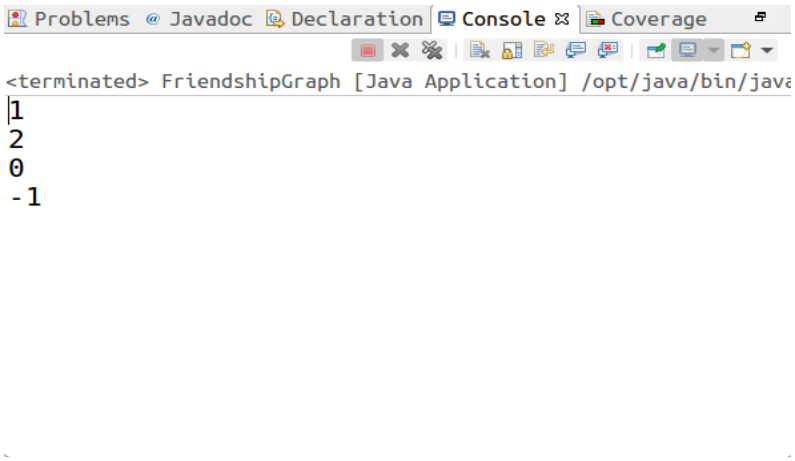
3.2.3 客户端 main()

这里采用了和 Lab1 中一样的测试用例，

代码如下：

```
public static void main(String[] args) {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person("Rachel");
    Person ross = new Person("Ross");
    Person ben = new Person("Ben");
    Person kramer = new Person("Kramer");
    graph.addVertex(rachel);
    graph.addVertex(ross);
    graph.addVertex(ben);
    graph.addVertex(kramer);
    graph.addEdge(rachel, ross);
    graph.addEdge(ross, rachel);
    graph.addEdge(ross, ben);
    graph.addEdge(ben, ross);
    System.out.println(graph.getDistance(rachel, ross));
    System.out.println(graph.getDistance(rachel, ben));
    System.out.println(graph.getDistance(rachel, rachel));
    System.out.println(graph.getDistance(rachel, kramer));

}
```

运行结果如下：

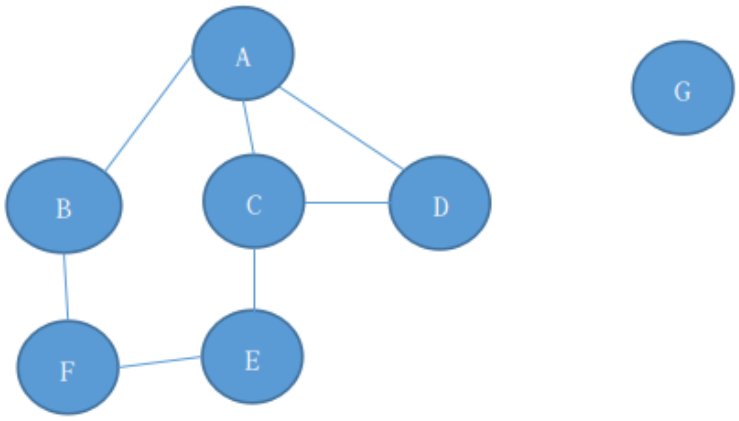


就是新建几个点，然后判断距离即可

3.2.4 测试用例

测试思路：

初始化一些人，并且做出一个简单的图，其中有孤立于其他的点，来检测不连通的情况。图如下



初始化代码如下：

```
/**
 * Tests that assertions are enabled.
 */
@Test(expected=AssertionError.class)
public void testAssertionsEnabled() {
    assert false;
}
FriendshipGraph graph = new FriendshipGraph();
Person a = new Person("A");
Person b = new Person("B");
Person c = new Person("C");
Person d = new Person("D");
Person e = new Person("E");
Person f = new Person("F");
Person g = new Person("G");
```

```
// Test strategy
//   - addVertexTest(): New some Person objects, then add them into the graph,
//     then call the method vertices() to check add successfully or not.
//   - addEdgeTest(): Similarly to the addVertexTest(), add the Edge object into
//     the graph, then check the Person.friends contains the other person of the
//     edge or not.
//   - getDistanceTest(): According to the graph we created just now, we can check
//     the distance of any two vertex in the graph.
//     When we are creating the graph, we should make some isolated points, which
//     means have no edges with other vertex, this can make sure our test strategy
//     more completed.
```

有如上的 test strategy

代码如下：

（1）测试 addVertex 方法：

```
@Test(expected=AssertionError.class)
public void testAssertionsEnabled() {
    assert false;
}
FriendshipGraph graph = new FriendshipGraph();
Person a = new Person("A");
Person b = new Person("B");
Person c = new Person("C");
Person d = new Person("D");
Person e = new Person("E");
Person f = new Person("F");
Person g = new Person("G");
/**
 * Test addVertex
 */
@Test
public void addVertexTest() {
    graph.addVertex(a);
    graph.addVertex(b);
    graph.addVertex(c);
    graph.addVertex(d);
    graph.addVertex(e);
    graph.addVertex(f);
    graph.addVertex(g);
    assertTrue(graph.graph.vertices().contains(a));
    assertTrue(graph.graph.vertices().contains(b));
    assertTrue(graph.graph.vertices().contains(c));
    assertTrue(graph.graph.vertices().contains(d));
    assertTrue(graph.graph.vertices().contains(e));
    assertTrue(graph.graph.vertices().contains(f));
    assertTrue(graph.graph.vertices().contains(g));

}
```

（2）测试 addEdge 方法：

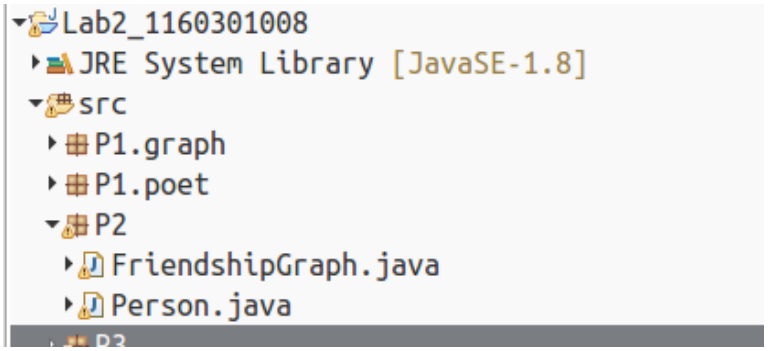
```java
/**
 * Test addEdge
 */

@Test
public void addEdgeTest() {
    graph.addVertex(a);
    graph.addVertex(b);
    graph.addVertex(c);
    graph.addVertex(d);
    graph.addVertex(e);
    graph.addVertex(f);
    graph.addVertex(g);
    graph.addEdge(a, b);
    graph.addEdge(b, a);
    graph.addEdge(a, c);
    graph.addEdge(c, a);
    graph.addEdge(a, d);
    graph.addEdge(d, a);
    graph.addEdge(b, f);
    graph.addEdge(f, b);
    graph.addEdge(c, e);
    graph.addEdge(e, c);
    graph.addEdge(f, e);
    graph.addEdge(e, f);
    graph.addEdge(c, d);
    graph.addEdge(d, c);
    assertTrue(a.friends.contains(b));
    assertTrue(a.friends.contains(c));


}
```

（3）getDistance 方法测试：

```java
/**
 * Test getDistance
 */

@Test
public void getDistanceTest() {
    graph.addVertex(a);
    graph.addVertex(b);
    graph.addVertex(c);
    graph.addVertex(d);
    graph.addVertex(e);
    graph.addVertex(f);
    graph.addVertex(g);
    graph.addEdge(a, b);
    graph.addEdge(b, a);
    graph.addEdge(a, c);
    graph.addEdge(c, a);
    graph.addEdge(a, d);
    graph.addEdge(d, a);
    graph.addEdge(b, f);
    graph.addEdge(f, b);
    graph.addEdge(c, e);
    graph.addEdge(e, c);
    graph.addEdge(f, e);
    graph.addEdge(e, f);
    graph.addEdge(c, d);
    graph.addEdge(d, c);
    assertEquals(1, graph.getDistance(a, b));
    assertEquals(2, graph.getDistance(a, f));
    assertEquals(0, graph.getDistance(a, a));
    assertEquals(-1, graph.getDistance(a, g));
    assertEquals(1, graph.getDistance(e, f));
}
```

3.2.5 提交至 Git 仓库

如何通过 Git 提交当前版本到 GitHub 上你的 Lab3 仓库。



3.3 The Transit Route Planner（选做，额外给分）

P3 要求根据输入文本建立一个交通网络图，然后输入起始站和终点站，从而规划出一条最短路。由于每条边带有权重，所以不能使用简单的 BFS，而是使用 Dikjstra 算法来进行最短路的求取。

3.3.1 Stop 设计

每个 Stop 要存储很多信息，在这里我把 Stop 类直接设置为交通图中的点，所以，和大多数人不一样的一点就是我在 Stop 中存储了时间，也就是如果一个站有多条线路经过的话，就建立不同的 Stop 用以区分，区分标志就是 RouteName 和 Time，也就是说，同一个车站可能会有很多的点在图中呈现。设计如下：

```java
import java.util.HashMap;

public class Stop {
    private double latitude = 0, longitude = 0;
    private String name = new String();
    private int time = 0;
    private String routeName = new String();
    public boolean flag = false;
    public boolean visit = false;
    public Map<Stop, Integer> friends = new HashMap<>();
    public Stop virtualStop;
    public int virtualNumber = 0;
    private int dist = Integer.MAX_VALUE;
    private String forwardStopName = new String();
    private int forwardStopTime = 0;
    private String forwardStopRouteName = new String();
    public Stop(double latitude, double longitude, String name, int time) {
        this.name = name;
        this.latitude = latitude;
        this.longitude = longitude;
        this.time = time;
    }
```

flag 表示是否存在虚点，这个后续会介绍，便于 Dijkstra 搜索最短路。visit 用于判断在 Dijkstra 中当前节点是否已经搜索出最短路。

3.3.2 Itinerary 类设计

知名见意，这个类就是要记录我们最后规划出来的行程，所以设计如下：

```java
import java.net.PasswordAuthentication;

public class Itinerary {
    private String instructions = new String();
    private int startTime;
    private int endTime;
    private Stop startLocation;
    private Stop endLocation;
    private List<Stop> path = new ArrayList<>();
    public Itinerary(List<Stop> Path) {
        path = Path;
        startLocation = Path.get(Path.size() - 2);
        endLocation = Path.get(1);
        startTime = Path.get(Path.size() - 1).getTime();
        endTime = endLocation.getTime();
    }
    public int getStartTime() {
        return startTime;
    }
    public int getEndTime() {
        return endTime;
    }
    public int getWaitTime() {
        return endTime - startTime;
    }
    public Stop getStartLocation() {
        return startLocation;
    }
    public Stop getEndLocation() {
        return endLocation;
    }
```

```java
public String getInstructions() {
    instructions += ("You start at " + startLocation.getName() + " at " + path.get(path.size() - 1).getTime() + '\n');
    instructions += ("At first, you take " + startLocation.getRouteName() + " at " + startLocation.getName() + " at " + startLoca
    for(int i = path.size() - 3; i >= 1; --i) {
        if(path.get(i).getName().equals(path.get(i + 1).getName())
                && path.get(i).getRouteName().equals(path.get(i + 1).getRouteName())) {
            continue;
        }
        else if(path.get(i).getName().equals(path.get(i + 1).getName())
                && !path.get(i).getRouteName().equals(path.get(i + 1).getRouteName())) {
            instructions += ("Change route here, wait for " + (path.get(i).getTime() - path.get(i + 1).getTime()) + '\n');
            instructions += ("take " + path.get(i).getRouteName() + '\n');
        }
        else {
            instructions += ("take " + path.get(i).getRouteName() + " arrive at " + path.get(i).getName() + " at " + path.get(i)
        }
    }
    instructions += ("You arrive at the destination at " + endLocation.getTime() +", totally spend " + (endTime - startTime));
    return instructions;
}
```

其中，我在构造函数中传进去了一个 List<Stop> Path，也就是在 RoutePlanner 中计算出来的最短路。

3.3.3 TrafficGraph 类设计

我重新实现了图，代码如下：

```java
package P3;

import java.util.ArrayList;

public class TrafficGraph {
    private List<Stop> vertices = new ArrayList<>();

    public void set(Stop source, Stop target , int weight) {
            source.friends.put(target, weight);
    }
    public void add(Stop st) {
        vertices.add(st);
    }
    public void remove (Stop source, Stop target) {
        if(source.friends.keySet().contains(target)) {
            source.friends.remove(target);
        }
    }
    public void removeVertex(Stop st) {
        vertices.remove(st);
        for(Stop stop: vertices) {
            if(stop.friends.containsKey(st))
                stop.friends.remove(stop);
        }
    }
    public List<Stop> getVertices() {
        return vertices;
    }

}
```

针对于我的 Stop 类中的数据结构来进行了优化和重写，并且优化了 set 函数，因为最后的图中涉及权重为 0 的边，为了防止换乘多次。

3.3.4 RoutePlannerBuilder 设计

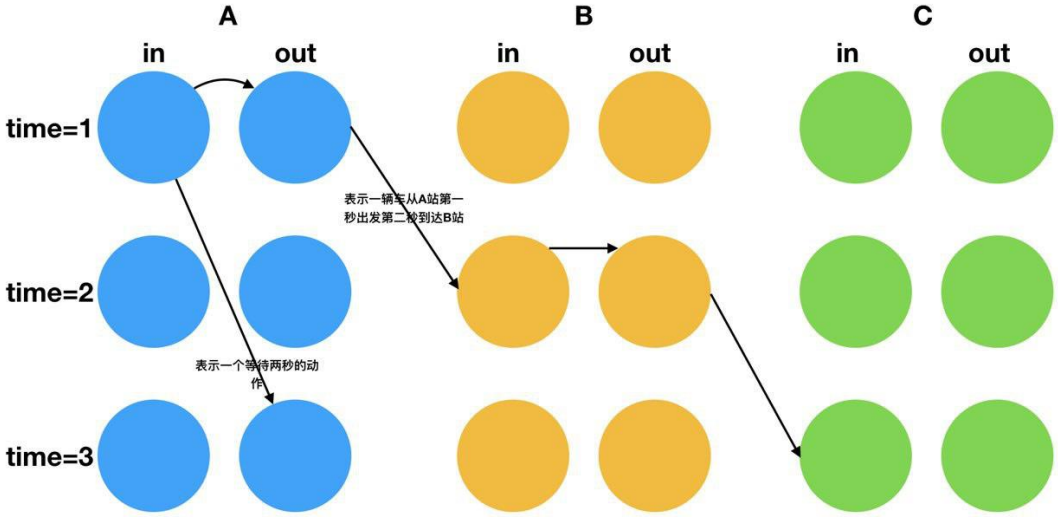
在这个类中，实现的功能就是从文件中读取数据进来，并保存在一个 map 中，我新建了一个 Route 类，作为 map 的 key，然后 value 值是一个存着这条路线上所有 stop 的 list，作为参数传进 RoutePlanner 对象中，便于建图。

```
public class RoutePlannerBuilder {
    public RoutePlanner build(String filename, int maxWaitLimit) {
        //List<Stop> stopsInTheFile = new ArrayList<>();
        Map<Route, List<Stop>> allRoutes = new HashMap<>();
        String routeName = new String();
        File Filename = new File(filename);
        Route route = new Route(" ", 0);
        try {                                   //read all routes in the csv file into the Map "allRoutes"
            InputStreamReader reader = new InputStreamReader(new FileInputStream(filename));
            BufferedReader br = new BufferedReader(reader);
            String myline = "";
            while((myline = br.readLine()) != null) {

                String[] oneLineInTheFile = myline.split(",");
                if(oneLineInTheFile.length == 2) {
                    //stopsInTheFile.clear();
                    List<Stop> stopsInTheFile = new ArrayList<>();
                    routeName = oneLineInTheFile[0];
                    route = new Route(routeName, Integer.valueOf(oneLineInTheFile[1]));
                    allRoutes.put(route, stopsInTheFile);
                }
                else if(oneLineInTheFile.length == 4) {
                    Stop oneStop = new Stop(Double.valueOf(oneLineInTheFile[1]), Double.valueOf(oneLineInTheFile[2]),
                            oneLineInTheFile[0], Integer.valueOf(oneLineInTheFile[3]));
                    allRoutes.get(route).add(oneStop);
                }
        }

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

3.3.5 RoutePlanner 设计

在这里首先实现了建图，也就是 createGraph 方法，这里建图的时候，如果想用 Dijkstra 来进行搜索最短路，则一定要涉及拆点，拆点的思路与如下：

如果要涉及换乘，那么将两条 route 上的该站都拆成两个点，其中一个是虚点，所有指向该点的路还指向该点，该点和他的虚点之间有一条权重为 0 的路，所有以该点为起点的路都从虚点来进行指出，具体思路如下图：



由于 Dijkstra 是单源最短路径，而起点和终点可能会涉及多个点在图中存在，所以这样比较麻烦，所以再新建两个虚点表示起点和终点，这样就实现了单源最短

路径。

建图的时候最难的就是拆点，尤其是换乘站的拆点，代码设计如下：

```
for(i = 0; i < sameNameStop.size(); ++i) {   //两两枚举 建虚点加边
    for(j = 0; j < sameNameStop.size(); ++j) {

        if(i == j || sameNameStop.get(i).getRouteName().equals(sameNameStop.get(j).getRouteName()) ||
                (sameNameStop.get(i).getTime() - sameNameStop.get(j).getTime()) < 0) {
            continue;
        }
        else if(sameNameStop.get(i).getTime() - sameNameStop.get(j).getTime() >= 0 &&
                sameNameStop.get(i).getTime() - sameNameStop.get(j).getTime() <= maxWaitTime)
        {
            if(sameNameStop.get(i).flag == false && sameNameStop.get(j).flag == false) {            //开始建虚点
                Stop virtual_0 = new Stop(sameNameStop.get(i).getLatitude(), sameNameStop.get(i).getLongitude(),
                        sameNameStop.get(i).getName(), sameNameStop.get(i).getTime());
                sameNameStop.get(i).flag = true;
                virtual_0.flag = true;
                virtual_0.virtualNumber = 1;
                virtual_0.setRouteName(sameNameStop.get(i).getRouteName());
                sameNameStop.get(i).virtualStop = virtual_0;
                virtual_0.virtualStop = sameNameStop.get(i);
                trafficGraph.add(virtual_0);
                for(Stop s: sameNameStop.get(i).friends.keySet()) {
                    trafficGraph.set(virtual_0, s, sameNameStop.get(i).friends.get(s).intValue());
                }
                sameNameStop.get(i).friends.clear();
                trafficGraph.set(sameNameStop.get(i), virtual_0, 0);
```

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
3.20-3.23	空闲时间大概五小时	完成 P1	没有完成
3.24	空闲时间大概五小时	完成 P1	完成
3.25	两小时	完成 P2	完成
3.28-4.2	大概八小时	完成 P3	完成

5 实验过程中遇到的困难与解决途径

其实 P1 和 P2 相对好写，只是 P3 比较难，当然，这可能也是他作为附加实验的原因，但其实想通了也没觉得有很难，其实难就难在拆点建图的思想，如果没有奥赛经验的话，很难想到这一点，当然，我也是通过室友讲解，才学会了这个拆点建图的方法，因为一个 Stop 可能有多条路线经过，而且还有可能有多个班次，这样如果不拆点，就造成了多重边，而 Dijkstra 不能处理多重边情况，所以一

定要拆点, 其次就是代码实现的时候, 拆点的时候真的很容易就写出 bug, 我用了大概三天去写代码加上 debug, 还是有难度的.

6 实验过程中收获的经验、教训、感想

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) 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？

ADT 可以复用, 直接面向场景一次性, 太特殊, 只针对于当前情况, ADT 可以针对不同相似场景来进行一些小的改动.

(2) 使用泛型和不使用泛型的编程，对你来说有何差异？

更熟悉不使用泛型的编程, 因为之前都没有使用泛型编程, 所以这次有一点不适应

(3) 在给出 ADT 的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？

不适应

(4) 本实验设计的 ADT 在三个应用场景下使用，这种复用带来什么好处？

复用时难度降低, 不用从零实现, 可以减小很多工作量

(5) 为 ADT 撰写 specification, invariants, RI, AF，时刻注意 ADT 是否有 rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？

在工程上可以有效的保护我们的代码, 避免黑客等恶意破坏, 这在将来的应用中十分重要. 愿意

(6) 关于本实验的工作量、难度、deadline。

适中.