

**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 | 冯运 |
| 学号 | 1160300524 |
| 班号 | 1603005 |
| 电子邮件 | f18846188605@gmail.com |
| 手机号码 | 18846188605 |

**目录**

[1 实验目标概述 1](#_Toc507927440)

[2 实验环境配置 1](#_Toc507927441)

[3 实验过程 1](#_Toc507927442)

[3.1 Poetic Walks 1](#_Toc507927443)

[3.1.1 Get the code and prepare Git repository 1](#_Toc507927444)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc507927445)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc507927446)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc507927447)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc507927448)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc507927449)

[3.1.4.1 Make the implementations generic 2](#_Toc507927450)

[3.1.4.2 Implement Graph.empty() 2](#_Toc507927451)

[3.1.5 Problem 4: Poetic walks 2](#_Toc507927452)

[3.1.5.1 Test GraphPoet 2](#_Toc507927453)

[3.1.5.2 Implement GraphPoet 2](#_Toc507927454)

[3.1.5.3 Graph poetry slam 2](#_Toc507927455)

[3.1.6 Before you’re done 2](#_Toc507927456)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc507927457)

[3.2.1 FriendshipGraph类 2](#_Toc507927458)

[3.2.2 Person类 3](#_Toc507927459)

[3.2.3 客户端main() 3](#_Toc507927460)

[3.2.4 测试用例 3](#_Toc507927461)

[3.2.5 提交至Git仓库 3](#_Toc507927462)

[3.3 The Transit Route Planner（选做，额外给分） 3](#_Toc507927463)

[4 实验进度记录 3](#_Toc507927464)

[5 实验过程中遇到的困难与解决途径 3](#_Toc507927465)

[6 实验过程中收获的经验、教训、感想 4](#_Toc507927466)

# 实验目标概述

本次实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：针对给定的应用问题，从问题描述中识别所需的ADT；设计ADT规约（pre-condition、post-condition）并评估规约的质量；根据ADT的规约设计测试用例；ADT的泛型化；根据规约设计ADT的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）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外泄（repexposure）；测试ADT的实现并评估测试的覆盖度；使用ADT及其实现，为应用问题开发程序；在测试代码中，能够写出testing strategy并据此设计测试用例。

# 实验环境配置

实验需要配置elclipse,但是我之前已经习惯于使用jetbrain系列的开发ide,所以我这次使用了IDEA，同时下载安装了jdk，版本号是jdk1.8.0\_101，将jdk的安装路径加入了系统的环境变量。在IDEA里面下载安装了junit插件,配置好了java的测试环境。

Lab2的仓库url:

https://github.com/ComputerScienceHIT/Lab2-1160300524.git

# 实验过程

## Poetic Walks

这个问题要实现图的抽象类，这个抽象数据类型需要分别实现基于点的实现和基于边的实现，这两种实现方式虽然不同，但在ADT的使用过程，没有区别。

### Get the code and prepare Git repository

如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

在合适的目录下， 在命令行工具下依次输入以下指令就可以完成。

git clone <https://github.com/rainywang/Spring2018_HITCS_SC_Lab2.git>

mkdir Lab2 #创建本地项目文件夹

cd Lab2

git init #创建本地git仓库

提交代码命令：git add .

Git commit –m ‘[commit message]’

将提交同步到远程仓库：

如果还未规定过远程仓库的话，需要先： git remote add origin <https://github.com/ComputerScienceHIT/Lab2-1160300524.git>

然后git push origin master

这样就把本地的提交记录同步到了github 的远程仓库

### Problem 1: Test Graph <String>

分别给每个函数写相应的测试函数，这些测试函数放在了GraphInstanceTest.java文件中

测试的策略：

等价类划分：给图中添加的点，未存在，已存在

给图中添加的边的权值：>0, = 0,<0

然后在给个测试函数中将这些等价类进行笛卡尔乘积，得到相应的测试用例。

例如：在测add（）函数时，分别添加已存在的点和未存在的点，观察返回的值

在测set()函数时，分别添加两个已存在的点，边的权重>0,两个未存在的点，边的权重>0,一个已存在的边，但是边的权重<=0,分别观察函数返回值的情况。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

boolean add(L vertex)

首先判断点集合中是否存在这个点，如果存在，就返回false,如果不存在，就向点集合中添加这个点，同时返回true.

int set(L source, L target, int weight)

首先遍历边的集合，看是否有与传入边相同的边，如果有，就先删除边，如果传入的weight>0,就将新的边加入集合，如果weight==0,就不做额外的处理。

如果边的集合中没有相同的边，就将新的边加入边的集合。

boolean remove(L vertex)

首先判断点的集合中是否有这个点，如果没有就返回false,.如果有就将这个点从点的集合中删除，然后遍历边的集合，将边的顶点中包含这个顶点的边删除。

Set<L> vertices()

新实例化一个点的集合，将原来点的集合clone到新的点集合，将新的点集合返回，这样可以避免数据的溢出

Map<L, Integer> sources(L target)

遍历边的集合，然后将边中包含target的边加入map中，返回

Map<L, Integer> targets(L source)

遍历边的集合，然后将边中包含source顶点的边加入map中，返回

#### Implement ConcreteVerticesGraph

boolean add(L vertex)

首先在点的集合中寻找是否这个点已经存在，如果存在，就返回false,如果不存在，就将这个点加入点的集合。

int set(L source, L target, int weight)

首先判断如果weight>0，就先将这两个点加入点的集合，然后分别寻找到source,target这两个点，将source点的出边添加或删除指定边，将target的入边添加或删除指定边。

boolean remove(L vertex)

首先判断点的集合中是否有这个点，如果没有，就返回false,如果有，就将这个点删除，然后将这个点的入边和出边都删除。

Set<L> vertices()

将图的点集合克隆后，将克隆后的对象返回。

Map<L, Integer> sources(L target)

将这个点的所有入边集合克隆后，将克隆后的对象返回

Map<L, Integer> targets(L source)

将这个点的所有出边集合克隆后，将克隆后的对象返回

### Problem 3: Implement generic Graph<L>

#### Make the implementations generic

将原来的String 都修改成 泛型<L>

#### Implement Graph.empty()

返回一个新的实例化对象ConcreteVerticesGraph,因为这个对象的效率更高。

### Problem 4: Poetic walks

#### Test GraphPoet

**通过文件读入了几句话，然后输入几句话，比对返回的句子与人工计算出的句子是否相同。**

**等价类划分：**

**两个单词之间不存在 连接词，两个单词之间只有一个连接词，两个单词之间有多个连接词（需要筛选）**

#### Implement GraphPoet

GraphPoet(File corpus)

这个函数用来读入文件，然后根据文件内容读入的句子建立图，图的点是文件中的单词，图的边权重是两个单词前后相接情况出现的次数。

String getBridge(String source, String target)

这个函数输入两个相邻的单词，然后遍历图，寻找权重最大的bridge-word，并返回，如果没有符合条件的单词，就返回””. 这个函数中我使用了广度优先搜索，只需搜索两层的点即可。

String poem(String input)

这个函数拆解输入的input，然后分别调用getBridge函数，将返回的单词加在相邻两个单词之间即可。

#### Graph poetry slam

在这一步中，我直接运行了main函数，然后可以输出正确的结果，运行正常。

### Before you’re done

请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "before_youre_done)的说明，检查你的程序。

如何通过Git提交当前版本到GitHub上你的Lab2仓库。

在这里给出你的项目的目录结构树状示意图。

Lab2

Src

P1

Graph

ConcreteEdgesGraph.java

ConcrteteVerticesGraph.java

Graph.java

Poet

GraphPoet.java

Main.java

mugar-omni-theater.txt

Test

P1

Graph

ConcreteEdgesGraph.java

ConcteteVerticesGraph.java

GraphInstanceTest.java

GraphStaticTest.java

Poet

Data.txt

GraphPoetTest.java

## Re-implement the Social Network in Lab1

这个任务让我们体会抽象数据类型的重要作用，只需要构造一次，就可以在其他程序中多次复用，大大提高编程的效率。

### FriendshipGraph类

这个类中只需要一个P1中的graph对象，然后addVertex()，addEdge()函数都是调用graph中的add(),set()函数即可轻松完成。

getDistance()函数依然使用广度优先搜索，大部分程序都不需要修改，只需要将搜索某个点出边的代码替换成 Graph类中的targets()函数，即可轻松实现。

### Person类

这个类只有一个字符串用来存储人的名字，然后有一个obseravor的函数，用来得到用户的名字。

### 客户端main()

将Lab1上的main函数中要求的操作复制过来即可。

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));

### 测试用例

我的测试分为两种测试思路，一种是靠大循环次数来验证稳定性和效率，同时也能测出正确性，在这个测试用例中，我设了1000次循环，即往关系表中添加1000个person对象，添加1000个边，最后比对getdistance的运行结果是否正确。

for (int i = 0; i < 1000; i++) {

Assert.assertEquals(i, graphTest.getDistance(persons[0], persons[i]));

}

第二种测试用例就是手动添加一些人，添加一些复杂的边，形成一个复杂的关系图，然后手动计算某些人之间的距离，与getDistance函数计算出的值进行比较，如果有不同的值，就无法通过jtest的测试。

### 提交至Git仓库

Lab2

Src

P2

FriendShipGraph.java

Person.java

Test

P2

FriendShipGraphTest.java

Edge.txt

## The Transit Route Planner（选做，额外给分）

这个任务依然是复用P1构造的抽象数据类型，但是问题更加复杂，难度也更大。这个任务的结构包括：根据数据文件建立图的部分，图的节点对象构建部分，对旅程的数据抽象部分，根据图计算最佳路径的部分。

### Stop接口

这个接口包含获得公交站的名字，获得公交站的经纬度位置。

在具体的实现类（myStop.java）中，除了实现这几个接口函数，还需要重写hashCode(),equals(),toString()这三个函数。

同时，还自己设计了一个新的类 StopEvent，这个类抽象自一个车站在不同时间的状态，所以这个类包括车站的stop对象，时间，时间对应的公交路线，这个类的实例将被当作图的顶点

### RoutePlannerBuild接口

在具体的实现类中，需要读入一个公交系统的信息文件，将数据的每一行解析为一个StopEvent实例，将同一条路线上的车站之间设置边，边的权重为两个状态之间转化需要的时间。这样就完成了图的建立工作。

### RoutePlanner接口

在这个类中，需要针对RoutePlannerBuild建好的图进行计算，得到两个车站之间用时最短的路径。

在计算用时最短路径时，需要首先确定起始点，这个起始点是在所有车站状态中的起始站台上，距离用户当前时间最近的一个状态作为起始状态，然后对图使用dijitstra算法，求出这个站点状态到其他所有的站点状态的最短路径，在这些最短路径中，挑选符合条件的目标站点状态，即可。同时根据生成的最短路径，返回一个描述路径的对象Itinerary.

### TripSegment接口

这个接口有两个实现类，分别为WaitSegment和BusSegment

WaitSegment 是用来表述乘客在一个站点等待公交车到来的过程，BusSegment是用来描述乘客在一个站点乘坐某趟公交车到达别的车站的过程。

两个实现类的最主要的区别在于 toString()的描述不同.

### Itinitary类

在这个类中，rep是一个列表，列表中包含WaitSegment,BusSegment，有这些组成一个乘客的旅程描述。

然后在这个类中，有一个getInstructions()函数，由这个函数产生一段描述性的语句，来告诉用户如何等车，如何乘坐车，乘坐哪趟车。

### Main类

这个类是我自己实现的一个与用户的交互界面，用户可以自己选择需要的操作，然后根据提示输入需要的信息，然后得到相应的相应结果。

交互信息如下：

input the choice number to execute the specific instruction

>>>1.input the name you want to search, I will show the stops contain the string

>>>2.input stop name now, the stop name at destination and time now ,I will return the instruction about how to get there

>>>if you want to exit the system,please input #

# 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 |
| 3.19 | 9：00-12：00 | 完成任务一 | 顺利完成 |
| 3.24 | 9：00-11：00 | 完成任务二 | 顺利完成 |
| 3.25 | 9：00-22：00 | 完成任务三 | 顺利完成 |

# 实验过程中遇到的困难与解决途径

遇到的问题：

1、作业的spec是全英文的，在读懂要求时，花了不少的时间。

2、在P1中的rep, rep invariant等概念刚开始一直没有弄明白，后来还是在MIT的官网上找到了相应的讲义，将这些概念搞懂了。

3、P3的数据结构在设计时遇到了不少麻烦，刚开始不知道应该如何设计，后来仔细分析了问题的实际场景，才终于抽象出了一个较为合理的数据结构。

# 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. 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？

面向ADT的编程，写过的抽象数据类型依然能够在很多的应用场景中得到复用，在开发其他的应用场景时，可以节约开发的时间，提高开发的效率。而直接面向应用场景编程设计的数据结构不需要考虑复用，可以针对当前的场景设计，这样的设计一遍性能要高于面向ADT的设计，但是使用的范围很窄，复用率低。

面向ADT的编程更像是构建工具的工程。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型需要考虑更多的特殊情况，做更多的判断，同时由于数据类型的不同，在安全方面，使用泛型更容易发生rep泄露的问题。

1. 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？

优势是可以将开发和测试的工作分离，测试的工作不再依赖于开发的进度，使得测试可以和开发同时进行，并且可以是不同的人员来完成。

而且在根据ADT写测试可以帮助自己更深刻的理解规约，使得实现的代码bug更少。

1. 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？

在很大程度上提高了开发的效率，同时，使得项目的结构更加明了，清晰。

1. 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？

这些工作的意义是为帮助ADT的使用这更好的理解这个ADT的功能，特性，使用情景，便于这个抽象数据类型的开发和复用。

检测rep exposure 是在编写ADT中提醒自己进行相应的自我检测，尽可能的从内部减少安全隐患，同时也是让ADT的使用者更放心的使用这个ADT,避免为安全性做过多无用的防御措施。

1. 关于本实验的工作量、难度、deadline。

这个实验的工作量相当大，我最终的代码量达到了3000行以上，花费了大约一周的时间，每天花费的时间也在10个小时以上。难度也比较大，尤其是P3，但是做完实验后，感觉自己的收获特别大。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

收获很大，对以后进行大型软件的开发有了理论基础，懂得了开发规约的重要性，这门课值得去投入大量时间，精力去学好这门课。