

**2024年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 洪梓淇 |
| 学号 | 2022111601 |
| 班号 | 2237102 |
| 电子邮件 | cristaux60.08@gmail.com |
| 手机号码 | 18800411800 |

**目录**

[1 实验目标概述 1](#_Toc97648181)

[2 实验环境配置 1](#_Toc97648182)

[3 实验过程 1](#_Toc97648183)

[3.1 Poetic Walks 1](#_Toc97648184)

[3.1.1 Get the code and prepare Git repository 1](#_Toc97648185)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc97648186)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc97648187)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc97648188)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc97648189)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc97648190)

[3.1.4.1 Make the implementations generic 2](#_Toc97648191)

[3.1.4.2 Implement Graph.empty() 2](#_Toc97648192)

[3.1.5 Problem 4: Poetic walks 2](#_Toc97648193)

[3.1.5.1 Test GraphPoet 2](#_Toc97648194)

[3.1.5.2 Implement GraphPoet 2](#_Toc97648195)

[3.1.5.3 Graph poetry slam 2](#_Toc97648196)

[3.1.6 使用Eclemma检查测试的代码覆盖度 2](#_Toc97648197)

[3.1.7 Before you’re done 2](#_Toc97648198)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc97648199)

[3.2.1 FriendshipGraph类 2](#_Toc97648200)

[3.2.2 Person类 3](#_Toc97648201)

[3.2.3 客户端main() 3](#_Toc97648202)

[3.2.4 测试用例 3](#_Toc97648203)

[3.2.5 提交至Git仓库 3](#_Toc97648204)

[4 实验进度记录 3](#_Toc97648205)

[5 实验过程中遇到的困难与解决途径 3](#_Toc97648206)

[6 实验过程中收获的经验、教训、感想 4](#_Toc97648207)

[6.1 实验过程中收获的经验和教训（必答） 4](#_Toc97648208)

[6.2 针对以下方面的感受（必答） 4](#_Toc97648209)

# 实验目标概述

本次实验要求设计并实现一个以String为标签的图ADT，实现图节点和边的CRUD操作。然后将这个图ADT改成一个依靠泛型标签的类，并使用泛型实现Poetic Walk功能。然后利用这个图的功能实现实验一的任务三中已经实现的功能。

# 实验环境配置

本次实验的环境配置与上一次基本相同，且测试覆盖度的分析是JetBrains IntelliJ IDEA自带的功能。不需要额外的环境配置。

在这里给出你的GitHub Lab2仓库的URL地址（HIT-Lab2-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab2-2022111601

# 实验过程

## Poetic Walks

本题目的大概内容如下：实现一个Graph<String>类型，用于ADT的开发。除了设计一个Graph的ADT外，还需要设计和实现边和顶点的ADT。然后将上述实现改为泛型实现，然后使用语料库“训练”一个诗人，“他”能够补全遗漏一个词的诗句。

### Get the code and prepare Git repository

直接使用clone指令即可从Git服务器上获取源代码（本次实验中为GitHub中获取）。在本地管理Git Repository的过程大致如下：

首先使用git init指令创建一个Git存储仓库，然后每次实现完成一个或若干个函数的时候，进行一次commit（具体情况取决于涉及函数的复杂程度）

为了练习使用Git分支功能的能力，我将GitHub Classroom中的代码克隆后（其中有一个README文件），然后使用git merge origin/main –allow-unrelated-history命令合并到了当前的分支，然后使用git switch -c dev创建一个名为dev的分支。即使在实际的Git Flow中，dev分支和main是两个相对平行的分支，但是我本次实验只是为了体验分支的用法，就使用了main和dev两分支，dev向main合并的Workflow。此后在每次实现一个主要功能模块的时候，将dev合并到main中即可。（实际实践中应该是由feature分支向dev分支合并，develop分支分出release分支，然后release分支改动后合并入main和dev，实际开发中，甚至可能没有main分支）

### Problem 1: Test Graph <String>

虽然对于ADT的测试不能严格地按照函数来划分，但是按函数分测试用例也是一种可行的方案。所以本次实验的测试使用了首先按函数划分，然后再按等价类划分。为了查找方便，我将测试用例的方法命名为“test［方法号］［用例号］”的格式命名。例如2号方法set的第一个测试用例就命名为test201。测试用例的设计大概可以按非法输入和有效输入分类，其中对于非法输入，只需断言其抛出异常即可。对于有效输入，需要根据其类型来分析测试用例的设计。例如对于add操作，一个可能的有效输入就是添加了重复的Vertex，类似的情况同样适用于set和remove。对于一些既有返回值又有操作的方法，需要同时对返回值和操作结果进行断言。

### Problem 2: Implement Graph <String>

Graph<String>的实现：Graph<String>仅仅是一个接口，仅静态成员函数empty()需要实现。本方法使用的实现是生成一个ConcreteEdgeGraph<String>

#### Implement ConcreteEdgesGraph

这是基于边的实现类，设计上，这个实现类有顶点的集合和边的列表，其中顶点是按标签存储的，也就是未来的泛型参数类型的对象。边是使用List存储的。这个边的类型Edge需要自行实现。实现边时，我为边设计了弧头标签、弧尾标签和权值三个字段。由于此类提供的字段较多，实现起来相对方便。在这个类的实现中有一些细节，例如即使目前实现的是以String作为泛型参数的类，但是还是需要在一些情况下对String进行toString()，这些是为了改成泛型实现的便利，此外，由于泛型实现需要对泛型类型进行比较相等的操作，所以需要使用equal来分析两个对象是否相等。下同。

#### Implement ConcreteVerticesGraph

这是基于顶点的实现类。设计上，这个实现类只有顶点的集合，并需要自行实现Vertex类。Vertex类实际上未来也需要成为一个泛型类，在这里可以直接按泛型类实现。但是当时由于我没有仔细看后续的要求，就直接按一般类进行了实现。Vertex需要一个标签字段。由于Graph中没有一个字段用来表示边，且目前实现的图是一个有向图，所以可以使用Vertex的字段来存储边的信息。可以在Vertex中添加一个Map<String, Integer>字段，存储每一条边连接的顶点和权值。由于实际的便利程度相似，所以为了便于理解，存储的边表示从this发出的边。

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

由于Graph<L>是一个接口，只需要实现其静态函数成员。在之前已经实现的String作为泛型参数的情况下，只需直接将String改为L即可

#### Make the implementations generic

由于前面所述，我已经在实现String作为泛型参数的图时为泛型预留了一定的操作空间，这里只需要先修改泛型参数，再按照报错把所有需要改的String改成L即可。

#### Implement Graph.empty()

这里只需将原来生成String为泛型参数改为L为泛型参数即可。

### Problem 4: Poetic walks

#### Test GraphPoet

这部分需要测试一个GraphPoet能否真正补全遗漏的单词，以及如果面临遗漏连续多个单词时是否会进行补全。所以采用了非法输入，单词完整，遗漏一个词，遗漏多个词四种情况。

#### Implement GraphPoet

经过考虑，这个类的功能和Graph的差异较大，使用委托可以获得更好的效率，更低的耦合读，而不是使用继承。此函数中只需要调用我已经实现的Graph类的其他方法即可实现其功能。其构造函数需要利用其语料库构建一个图，推测部分需要找两个节点的所有中间节点并求出权值之和，取最大的一项（但我个人认为应该取权值积最大的一个作为中间单词，例如none of them is中，如果需要补全them，使用权值的积更合适，因为如果使用权值和，在较大语料库的前提下，of the是常出现的。所以of和the之间的权值可能非常大。但是the和is出现的权值相对of和the出现的权值比较要小很多，所以这里of后面可能会被错误地匹配为the，而不是them，如果使用乘法，the is的低出现率可以极大地拉低乘积，相比按和判断更为合适）。

#### Graph poetry slam

这里需要实现的是一个推断功能。首先需要判断两个词之间是否直接相连，如果直接相连，可以直接走过去。但是如果没有直接相连，可以看中间是否有一个词语可以作为bridge，如果找到，则用这个词作为桥连接两个词。在遍历到桥的时候，需要遍历到结束，如果遇到新的桥权值更大的话，需要替换掉原来的桥。

### 使用Eclemma/Code Coverage for Java检查测试的代码覆盖度

经过测试，源代码的行覆盖率为87%，除去main函数外，测试了每一个public函数。测试的覆盖率良好

### Before you’re done

使用git remote add lab [git@github.com](mailto:git@github.com):ComputerScienceHIT/HIT-SC-Lab-2022111601指令即可将本地仓库和远程仓库建立联系。以下是仓库的文件结构（tree指令的输出结果）

user@workstation:~/Repositories/sclab2$ tree .

.   
├── doc   
│   └── Lab-2 Report Template.docx   
├── LICENCE   
├── pom.xml   
├── README.md   
├── sclab2.iml   
├── src   
│   ├── P1   
│   │   ├── graph   
│   │   │   ├── ConcreteEdgesGraph.java   
│   │   │   ├── ConcreteVerticesGraph.java   
│   │   │   └── Graph.java   
│   │   └── poet   
│   │       ├── GraphPoet.java   
│   │       ├── Main.java   
│   │       └── mugar-omni-theater.txt   
│   └── P2   
│       ├── P2.java   
│       ├── Person.java   
│       └── SocialNetwork.java   
├── target   
│   ├── classes   
│   │   ├── P1   
│   │   │   ├── graph   
│   │   │   │   ├── ConcreteEdgesGraph.class   
│   │   │   │   ├── ConcreteVerticesGraph.class   
│   │   │   │   ├── Edge.class   
│   │   │   │   ├── Graph.class   
│   │   │   │   └── Vertex.class   
│   │   │   └── poet   
│   │   │       ├── GraphPoet.class   
│   │   │       ├── Main.class   
│   │   │       └── mugar-omni-theater.txt   
│   │   └── P2   
│   │       ├── P2.class   
│   │       ├── Person.class   
│   │       └── SocialNetwork.class   
│   ├── generated-sources   
│   │   └── annotations   
│   ├── generated-test-sources   
│   │   └── test-annotations   
│   └── test-classes   
│       ├── P1   
│       │   ├── graph   
│       │   │   ├── ConcreteEdgesGraphTest.class   
│       │   │   ├── ConcreteVerticesGraphTest.class   
│       │   │   ├── GraphInstanceTest.class   
│       │   │   └── GraphStaticTest.class   
│       │   └── poet   
│       │       └── GraphPoetTest.class   
│       └── P2   
│           └── SocialNetworkTest.class   
└── test   
   ├── P1   
   │   ├── graph   
   │   │   ├── ConcreteEdgesGraphTest.java   
   │   │   ├── ConcreteVerticesGraphTest.java   
   │   │   ├── GraphInstanceTest.java   
   │   │   └── GraphStaticTest.java   
   │   └── poet   
   │       └── GraphPoetTest.java   
   └── P2   
       └── SocialNetworkTest.java   
  
27 directories, 37 files

## Re-implement the Social Network in Lab1

在这里简要概述你对该任务的理解。由于本次实验实现了Graph类，所以可以调用Graph类来辅助社交网络的实现。可以认为，本任务是一种调用和调试开源ADT的能力的训练。

### FriendshipGraph类

备注：由于疏忽，我这里直接将类命名为SocialNetwork

由于Graph类过于复杂，且和FriendshipGraph类关系较小，没有必要让SocialNetwork作为其子类。所以本例中使用委托关系来建立两者的联系。对于获取相邻顶点等基本功能，可以直接调用Graph的一个简单函数解决，但是对于查询距离，可以使用Graph已经提供的方法和BFS算法解决。

### Person类

直接使用上次实验中的Person类

### 客户端main()

直接使用上次实验中的main方法

### 测试用例

直接使用上次实验中的测试用例

### 提交至Git仓库

使用git remote add lab [git@github.com](mailto:git@github.com):ComputerScienceHIT/HIT-SC-Lab-2022111601指令即可将本地仓库和远程仓库建立联系。以下是仓库的文件结构（tree指令的输出结果）

user@workstation:~/Repositories/sclab2$ tree .

.   
├── doc   
│   └── Lab-2 Report Template.docx   
├── LICENCE   
├── pom.xml   
├── README.md   
├── sclab2.iml   
├── src   
│   ├── P1   
│   │   ├── graph   
│   │   │   ├── ConcreteEdgesGraph.java   
│   │   │   ├── ConcreteVerticesGraph.java   
│   │   │   └── Graph.java   
│   │   └── poet   
│   │       ├── GraphPoet.java   
│   │       ├── Main.java   
│   │       └── mugar-omni-theater.txt   
│   └── P2   
│       ├── P2.java   
│       ├── Person.java   
│       └── SocialNetwork.java   
├── target   
│   ├── classes   
│   │   ├── P1   
│   │   │   ├── graph   
│   │   │   │   ├── ConcreteEdgesGraph.class   
│   │   │   │   ├── ConcreteVerticesGraph.class   
│   │   │   │   ├── Edge.class   
│   │   │   │   ├── Graph.class   
│   │   │   │   └── Vertex.class   
│   │   │   └── poet   
│   │   │       ├── GraphPoet.class   
│   │   │       ├── Main.class   
│   │   │       └── mugar-omni-theater.txt   
│   │   └── P2   
│   │       ├── P2.class   
│   │       ├── Person.class   
│   │       └── SocialNetwork.class   
│   ├── generated-sources   
│   │   └── annotations   
│   ├── generated-test-sources   
│   │   └── test-annotations   
│   └── test-classes   
│       ├── P1   
│       │   ├── graph   
│       │   │   ├── ConcreteEdgesGraphTest.class   
│       │   │   ├── ConcreteVerticesGraphTest.class   
│       │   │   ├── GraphInstanceTest.class   
│       │   │   └── GraphStaticTest.class   
│       │   └── poet   
│       │       └── GraphPoetTest.class   
│       └── P2   
│           └── SocialNetworkTest.class   
└── test   
   ├── P1   
   │   ├── graph   
   │   │   ├── ConcreteEdgesGraphTest.java   
   │   │   ├── ConcreteVerticesGraphTest.java   
   │   │   ├── GraphInstanceTest.java   
   │   │   └── GraphStaticTest.java   
   │   └── poet   
   │       └── GraphPoetTest.java   
   └── P2   
       └── SocialNetworkTest.java   
  
27 directories, 37 files

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2024-04-13 | 21:00~00:51 | 完成图测试部分的编写 | 未完成，仅完成了测试策略的设计 |
| 2024-04-14 | 12:00~13:50 | 完成图测试部分的编写 | 按期完成 |
| 2024-04-14 | 16:00~18:00 | 完成toString函数的测试 | 超时11分钟完成 |
| 2024-04-15 | 05:00~10:00 | 完成了所有的测试部分 | 超时26分钟完成 |
| 2024-04-15 | 10:26~11:26 | 完成按边实现的类初步设计 | 按期完成 |
| 2024-04-15 | 12:00~13:00 | 根据测试结果修改部分实现 | 按期完成 |
| 2024-04-18 | 18:30~20:00 | 完成按边实现的图 | 剩余部分未完成 |
| 2024-04-20 | 00:00~02:00 | 实现按边实现的图的剩余部分 | 按期完成 |
| 2024-04-20 | 12:30~13:00 | 根据测试修改实现 | 未完成，仍有bug |
| 2024-04-21 | 13:00~14:00 | 修改了所有的bug并通过全部测试 | 按期完成 |
| 2024-04-21 | 14:00~16:00 | 完成按边实现的图基本功能 | 超时14分钟完成 |
| 2024-04-23 | 10:00~10:30 | 完成了按边实现的查询功能 | 按期完成 |
| 2024-04-23 | 13:00~14:00 | 完成按边实现的所有实现并通过测试 | 超时26分钟完成 |
| 2024-04-23 | 16:00~17:00 | 完成泛型实现 | 超时6分钟完成 |
| 2024-04-25 | 01:00~03:30 | 完成Poetic Walk并通过所有测试 | 按期完成 |
| 2024-04-25 | 14:00~14:30 | 完成实验一社交网络的重实现 | 超时16分钟完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 如何在对实现一无所知的情况下去写测试用例 | 可以先理清每一个函数的逻辑，预期的输入和输出，以及函数处理的过程中可能造成的改变。然后对于函数的预期功能进行测试。实际上可以说函数的测试是基于需求的，而不是基于实现的。所以不应该将写测试的依托寄托于实现上 |
| 对于Poetic Walk的英文描述比较迷惑 | 根据其互动性展示和AI翻译解决了问题 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训（必答）

首先，本次实验中我体会到了JetBrains IntelliJ IDEA的强大，该IDE可以实现对于代码的优化，例如如果我使用assertEquals函数来判断一个boolean值，但是这种情况下IDE就会提醒我使用assertTrue（false同理）来判断。

其次，在ADT设计的部分，本次实验让我体会到了一个设计得比较好的ADT可以让调用者减轻压力，而测试是保障ADT质量的一个必要因素。如果ADT的设计未经过测试（我第一次运行测试时，只通过了约30%的测试用例），那么ADT的可靠性无法保证，这也就直接导致了在后续利用ADT时无法正常使用。此外也让我体会到了分离的思想，这次实验中分离了ADT的设计和调用，以便于我测试好ADT后再去设计调用。这样的过程便于分析bug的归属。

最后，通过本次对于Git分支的使用练习，我熟练了git分支的使用，合并冲突的解决等问题。

## 针对以下方面的感受（必答）

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的设计给我带来一种比较强烈的“分离感”，也就是说可以分离ADT设计和ADT调用，避免调试和测试时发生“层次污染”，更有利于快速找出bug所在。但是对于实际效率来说，建于现在委托式实现的流行，使用ADT可能会在一定程度上亏损性能，这也让我初步体会到了工程思维中取舍的意义，由于现在的性能是普遍充足的，所以可以牺牲一定的性能来换取较高的开发效率。此外，这种性能的损失也让我意识到了SRP的价值，如果一个类含有过多的责任，那么在委托这种类对象的时候，性能损失更大。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

泛型编程，让我感受到的是它似乎类似于函数的重载，是对于开发的一种便利。泛型可以让使用者调用更加舒适，也可以规范开发者的行为。

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？

对于ADT的开发者来说，先写测试往往给开发者一个比较好的情绪价值，发现通过代码的修改和优化逐渐通过更多的测试用例对于开发阶段的开发者来说，是一个正面的情绪体验，也让开发者对于可以投入使用有了一个更明确的定位。此外，测试也可以确保ADT的可靠性。如果一个ADT未经测试就投入具体场景的使用，会发生层级之间的污染（这个是我自创的一个词汇，意思是ADT这一层的bug在实现和调用中才体现出来，感觉这种说法很形象），导致bug的排查变得困难。本次实验中我并不能很好地适应这样的模式，因为如果没写出任何实现就开始写测试，会让人比较难以想象。但是本次为了适应这样的开发模式，我还是先写了测试。

1. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？

减少开发的时间，提高开发效率，例如在实现Poetic Walk和Social Network时，开发的周期要短很多。且由于ADT是单独开发的，且是经过测试的，不会发生bug的层级污染。

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？

这些工作的意义是明确一个开发规范，在开发之前就明确好一个规范，并遵守之，可以避免一些意外，避免给调用者输出不合理的内容等等。这样的做法对于日后的开发意义重大。

1. 关于本实验的工作量、难度、deadline。

本次实验的工作量较大，难度适中，推迟后的deadline合适。

1. 《软件构造》课程进展到目前，你对该课程有何收获和建议？

软件构造这门课的目的在我心中更加清晰。这门课实践和理论相结合，且相较其他课程更加重视实践的设计有利于学到具体而有用的内容，可以启发我自己的思考，这一门课的理论部分实际上并不是非常晦涩，结合实践的程度高，可以切实体会实践和理论的相互推动，和利用实践和理论两条腿走路的方法论。让我意识到如何写出优雅的代码，补上了大一时C语言课未讲授相应内容的缺口。启发式的授课模式也让我有更大的兴趣去了解更多知识，例如设计模式，float类型对于相等的判断方式等等，也为我养成了不理解就看看源代码的一种思维。

此外这门课的实验非常好，虽然难度和工作量远高于国内的其他课程，但是也让我学会了大量的知识，体会到了实践中什么是好的，什么是不好的，也让我有更大的兴趣去探索国际上其他优秀的公开课的实验。

有一些关于课程的建议，例如可以考虑使用Markdown进行报告排版，甚至在GitHub仓库中用README文件作为报告。问过学长学姐发现，这门课之前内容更加庞大，建议将并行、优化等等诸多内容作为选修课供大家学习。