**面向对象程序设计**

**课 题：实现数据结构中TREE结构的API**

**姓 名： 李家玺**

**学 号： 2020404164**

**专业年级： 2020级计算机科学与技术**

**同组姓名学号：**

**同组姓名学号：**

2021年12月25日

目录

[1、 选题介绍与设计目的 3](#_Toc91100873)

[1.1课程设计目的及要求 3](#_Toc91100874)

[1.2 问题描述 3](#_Toc91100875)

[二、分析与设计 4](#_Toc91100876)

[2.1、面向对象分析 4](#_Toc91100877)

[2.2、面向对象设计 5](#_Toc91100878)

[三、系统实现与测试 5](#_Toc91100879)

[3.1、系统的实现 5](#_Toc91100880)

[3.2、系统测试 5](#_Toc91100881)

[四、总结和分析 5](#_Toc91100882)

# 一、选题介绍与设计目的

## 1.1课程设计目的及要求

1. 课程设计的目的：

通过对课程设计的题目，学会分析存在的问题，利用所学知识java解决问题。锻炼分析问题和解决问题的能力，复习、巩固Java语言的基础知识，加强对java的理解与应用。培养学生在开发项目中的团队合作精神，提高创新意识，增强学生的设计和实现应用想法的能力。

1. 课程设计的要求：

支持各种类型树的基本操作，如创建，遍历，插入，删除，更新等。

重点突出二叉树的操作，如普通二叉树、平衡二叉树、完全二叉树、前中后序遍历等、满足上述集中树的基本特性。

提供完整的API帮助文档。

## 1.2 问题描述

1. 问题描述

N.沃思曾经出过一本书《算法+数据结构=程序》。一个好的程序离不开好的算法和数据结构。一个好的程序离不开好的算法和数据结构。拿经典的二分查找来说，在有序的集合里面，每一次比较集合的中间元素、然后进行折半。相对暴力的从集合头查到集合尾，在集合元素个数超过4个之后，二分查找得时间少很多。二分算法每一次对集合元素的个数取半，假设集合元素个数为N，则二分查找的时间是 ,暴力查找的时间可达到N次，两者相差是巨大的。数据结构和算法有着千丝万缕的关系。不妨联想二分查找算法和二叉树的关系，如果按左孩子比父亲小，右孩子比父亲大的逻辑来建立一棵二叉树，当要在树上找某个元素的时候，其本质就是二分查找。有时候可以从好的数据结构得到好的算法，也可以从好的算法得到好的数据结构。而两个一起能实现一个好的程序。

树是一种重要的数据结构类型，在JAVA中很多容器都是基于树实现的。例如TreeSet就是基于红黑树实现的一种数据结构。虽然JAVA中有很多用树实现的容器，但是这些容器都已经由树扩展到了具体的功能。JAVA中没有树的类是单纯记录节点的，当我们想自己用树实现一种功能的时候，需要提前自己写一个树类。考虑到代码的重复，及树的建立。。。。。等问题。该课程设计在完成树的基础上，另外搭建了二叉树、平衡二叉树、完全二叉树。并完成说明这些类的API文档。以方便今后的对树类的使用。

1. 背景及意义。

树作为一个数据结构，在编程中起到十分重要的作用。

数据结构是指相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素之间的关系组成。记为：Data\_Structure=(D,R)。其中D是数据元素的集合，R是该集合中所有元素之间的关系的有限集合。

一般认为，一个数据结构是由数据元素依据某种逻辑联系组织起来的。对数据元素间逻辑关系的描述称为数据的逻辑结构；数据必须在计算机内存储，数据的存储结构是数据结构的实现形式，是其在计算机内的表示；此外讨论一个数据结构必须同时讨论在该类数据上执行的运算才有意义。一个逻辑数据结构可以有多种存储结构，且各种存储结构影响数据处理的效率。

在许多类型的程序的设计中，数据结构的选择是一个基本的设计考虑因素。许多大型系统的构造经验表明，系统实现的困难程度和系统构造的质量都严重的依赖于是否选择了最优的数据结构。许多时候，确定了数据结构后，算法就容易得到了。有些时候事情也会反过来，我们根据特定算法来选择数据结构与之适应。不论哪种情况，选择合适的数据结构都是非常重要的。

选择了数据结构，算法也随之确定，是数据而不是算法是系统构造的关键因素。这种洞见导致了许多种软件设计方法和程序设计语言的出现，面向对象的程序设计语言就是其中之一。

# 二、分析与设计

## 2.1、面向对象分析

**“树”API文档实现一下功能：**

* 能查阅所有树的类
* 能查阅相应树的方法
* 能查阅相应方法的解释

**“树”的API实现以下功能：**

* 提供一个接口。之后的树类都需实现该接口。
* 提供一个树类。能实现任意阶数的树。
* 提供二叉树、平衡二叉树、完全二叉树类。
* 树能存储所有类型的数据。
* 所有的树提供增加节点、删除节点、修改节点、查找节点、遍历、先序建立的方法。

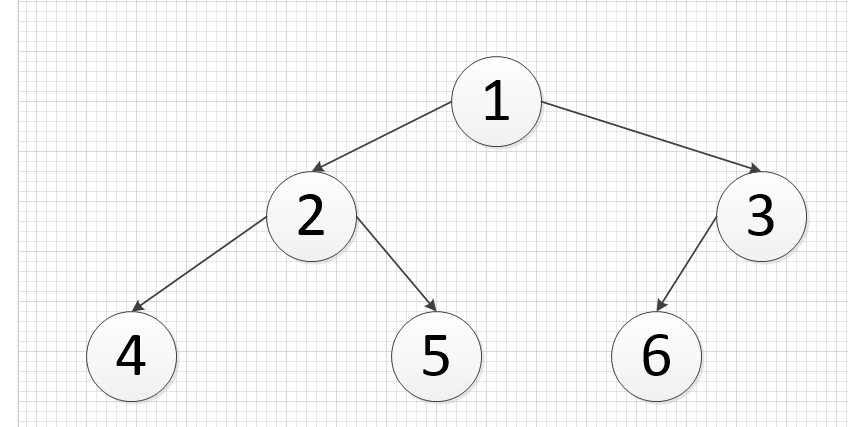
**需求说明：**

1. 先序建立：

通过添加一组数据，构建一颗任意阶数的树。添加的时候需要事先假设空节点，达到先序建立的效果。

1. 遍历：

分别完成树的“根->左子->右子”的先序遍历，“左子->右子->根”的后续遍历。对于二叉树完成“左子->根->右子”的中序遍历。

若构建这样的二叉树（假设数据为Integer）：****

先序建立：输入数据组为： 1 2 4 null 5 null null 3 6 null null null

 样例解释：

如图代表每个空指针对于连接的元素。

前序遍历的效果：1 2 4 5 3 6

中序遍历的效果：4 2 5 1 6 3

后续遍历的效果：4 5 2 6 3 1

1. 增加节点：

设置两种增加方法，缺省情况下在默认添加到左子树下，或者添加到给定节点下。

1. 删除节点：

删除一个与目标相同的节点，若存在多个，则删除先序遍历中的第一个节点。

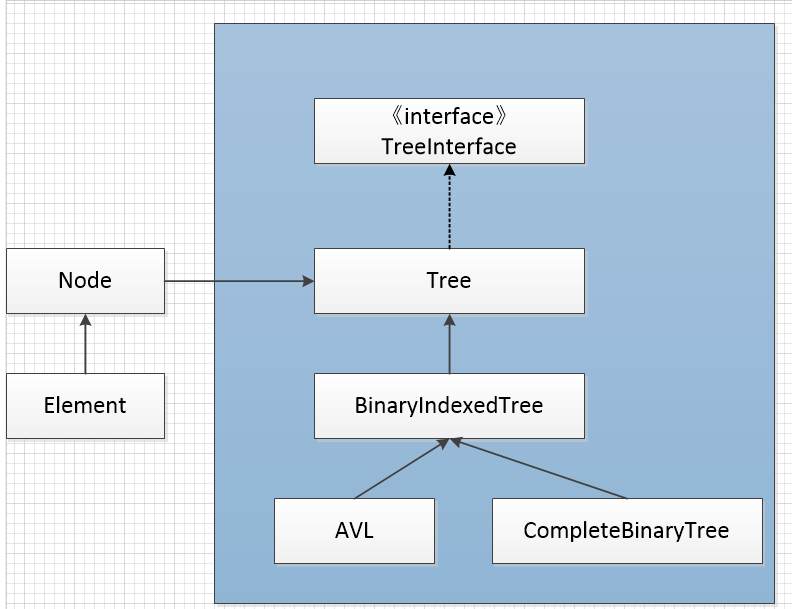
1. 修改节点：

修改与目标匹配的节点，若存在多个，则修改先序遍历中的第一个节点。

1. 查找节点：

判断目标节点是否存在，若存在多个，则返回先序遍历中的第一个匹配的节点的信息。

## 2.2、面向对象设计

该API组成如下：

所有类结构初步分析：

从“大”往“小”进行分析，二叉树是特殊的树，平衡二叉树、完全二叉树是特殊的二叉树。从继承的关系上来说，二叉树继承树，平衡、完全二叉树继承二叉树。节点是实现树的最小单元，节点由所存数据和指针方向构成。

Tree类实现TreeInterface接口，“接口+实现”最常见的优势就是实现类和接口分离，在更换实现类的时候，不用更换接口功能。接口对程序起到规范的作用，同时有益于以后的维护和拓展。Tree类中实现树的大部分基础方法。

二叉树类和Tree有很多相同相似的方法，因此二叉树继承Tree类，提高了代码的复用性，降低了代码量。在构造一个二叉树上面，为了提高二叉树类的健壮性，每次插入子节点的时候，需要对父节点进行二外的判断，防止出现二叉树类里面有三阶的树。在遍历二叉树上面，Tree类的先、后续遍历完美匹配二叉树的先、后续遍历，因此可以直接使用。在修改、查询、删除二叉树节点上面，Tree类相对应的方法也是完美匹配二叉树的。

平衡二叉树继承二叉树。平衡二叉树增加了查找节点，计算树的高度以及通过左旋转、右旋转、左右旋转、右左旋转维持二叉树的平衡。为了满足平衡二叉树的要求，该树建立的时候需要满足左右平衡因子相等，同理在增加、删除节点后也需要对树进行维护。因此需要重写二叉树创建、增加节点、删除节点的方法。但是其他方法同样适用。

完全二叉树继承二叉树。该树为了满足完全二叉树的要求，也需重写创建、增加节点、删除节点的方法。其余方法同样设用。

所有树的最基本的单元是节点。节点和树的关系，就像细胞和动植物的关系。一个节点类，至少包含三种属性：1.记录父节点是哪一个（即一个父指针）。2.记录有哪些子节点（即多个子指针）。3.记录自己所存储的数据。通过不断的变换父子指针的关系，最终能建成一棵树。

**第一步：TreeInterface接口的设计。**

**第一步：节点类的设计。**

节点的UML图：

|  |
| --- |
| Node |
| -father: Node  -son: ArrayList< Node >  -elem: Object |
| +Node(elem:Object)  +addSon(son:Node)  +addFather(father:Node)  +toString() |

并提供相应属性的get和set方法。

1. Element数据类型的选择。

从最简单的节点设计开始说起，一个节点最低需要满足能存储数据的要求。所有的数据结构、不管是多复杂，多繁琐，最终所要达到的目的是存储数据，本API中，将树细化成节点、每个节点搭载数据。最开始设计节点类的时候，element属性细化到Integer,但这样API的可用性大打折扣，数据类型如此之多，不能只存储固定的数据类型。于是将数据类型调整至Object。

仅将Element扩展成Object是不够的。为提高API的健壮性、防止使用者无意添加进去不同类型数据，还需使用泛型。泛型的使用让类型更安全。通过知道使用泛型定义的变量的类型限制，编译器可以在一个高得多的程度上验证类型假设。没有泛型，这些假设就只存在于程序员的头脑中。泛型的使用消除了强制类型转换。泛型的一个附带好处是，消除源代码中的许多强制类型转换。这使得代码更加可读，并且减少了出错机会。

综上所述，在Element类型为Object的基础上，还添加了泛型参数的使用。

1. 父、子节点类型的选择。

从C语言中能获得很好的构建树的方法——指针。下面是C语言中构建一颗二叉树最常用的节点类型：

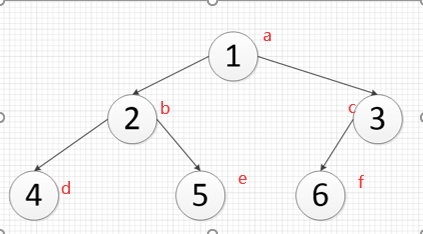
**typedef struct node{**

**char a;**

**struct node \* l, \* r;**

**}node,\*tree;**

从以上的代码能获得很大的启示：节点还需要存子节点的内存地址。换句话说该节点能指向下一个节点。在C语言里面能直接用指针实现。但在JAVA里面没有指针。

使用数组模拟指针。如图的一棵二叉树：

用红色字母表示所记录的元素。那么每一个节点都可以有属于自己的唯一编号，这样我们就可以让节点额外记录自己父亲节点的编号、子节点的编号，从而实现一颗树。但这样实现一个树过于繁琐、迂回。很容易把节点编号、节点元素等东西弄混。简而言之，不够简洁明了。

为找到一种更直接的方式，要先说JVM三大内存区: 栈内存区、堆内存区、方法内存区。实力化一个对象后。实例化的对象存在堆内存区，他的名字存在栈内存区、指向他在堆的位置。这种方式酷似C语言的指针。因此每个节点的父指针可以直接是他的父节点，这样也不会照成堆内存的重复使用，也达到了简洁明了的特点。同时，也能直接获得他的父、子节点，易于之后的操作。

根据树的性质：

1.每个节点有零个或多个子节点；

2.没有父节点的节点称为根节点；

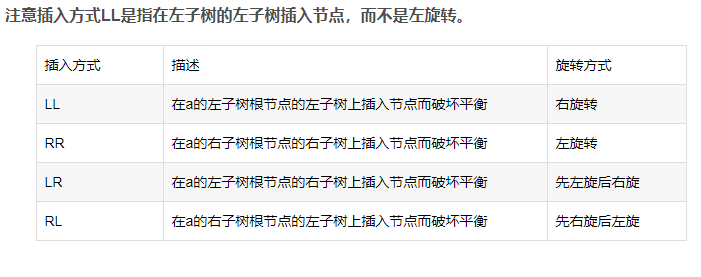
3.每一个非根节点有且只有一个父节点；

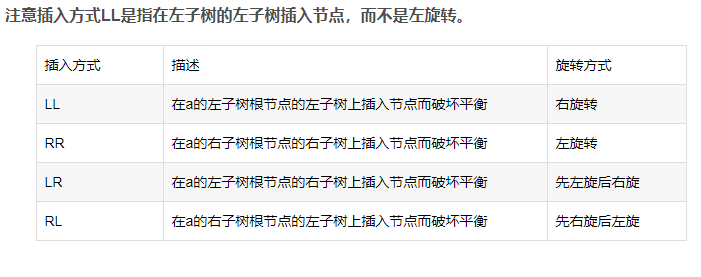
4.除了根节点外，每个子节点可以分为多个不相交的子树。

因此父节点只需要一个就够，但是子节点可能有多个。因此，子节点用AarrayList将所有的都存起来。首先在子节点数量随机的情况下，AarrayList不会造成内存浪费。其次，AarrayList也能完美扩展成二叉树（及用脚标0表示左子树、1角标表示右子树）。

## 平衡二叉树的实现：

首先需要先了解平衡二叉树的性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树，是一棵有顺序的树，具有其他二叉树的性质，但是又加了限定条件。判断是否保持平衡的一个重要条件便是新引入的一个概念：平衡因子，是指左子树与右子树的高度差，当这个高度差的绝对值超过1就证明为保持平衡，就要通过旋转使树达到平衡。平衡二叉树是对二叉排序树的优化。旋转分为左旋转、右旋转、左旋右旋、右旋左旋。插入、删除等操作可以通过继承树大类中定义的方法来实现。





# 三、系统实现与测试

## 3.1、系统的实现

按前面分析所得出的程序功能，使用面向对象技术，采用Java语言编码实现程序各部分功能。

## 3.2、系统测试

Jdk15.02

第一组数据：

开始：

{1 ,2, 3, null,4,null , 5 ,null,null}

操作：

t.preOrderTree();  
t.size();  
t.isEmpty();  
t.toArray() ;//万物皆对象  
t.addNode(1);  
t.addNode(1,1);  
t.remove(2);

理论值：

Node is ：1

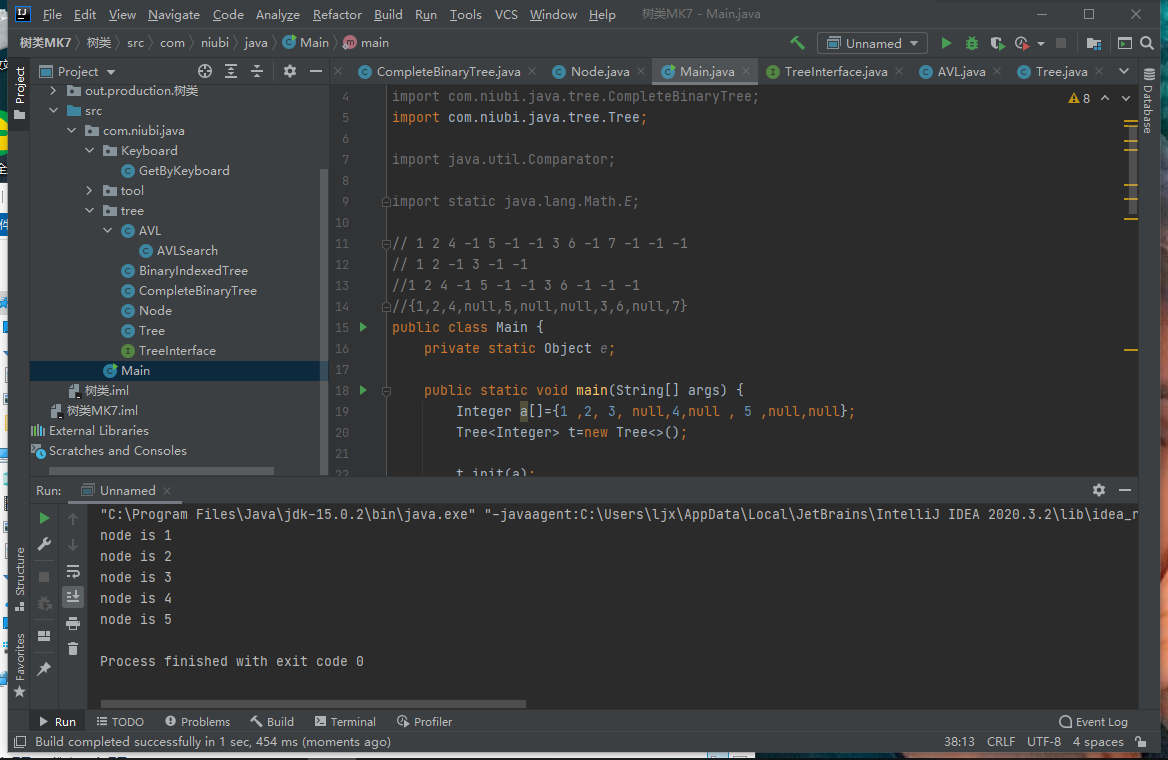
Node is ：2

Node is ：3

Node is ：4

Node is ：5

实际值：



第二组数据：

{1,2,4,null,5,null,null,3,6,null,7}

操作：

t.preOrderTree();  
t.size();  
t.isEmpty();  
t.toArray() ;//万物皆对象  
t.addNode(1);  
t.addNode(1,1);  
t.remove(2);

理论值：

node is 1

node is 2

node is 3

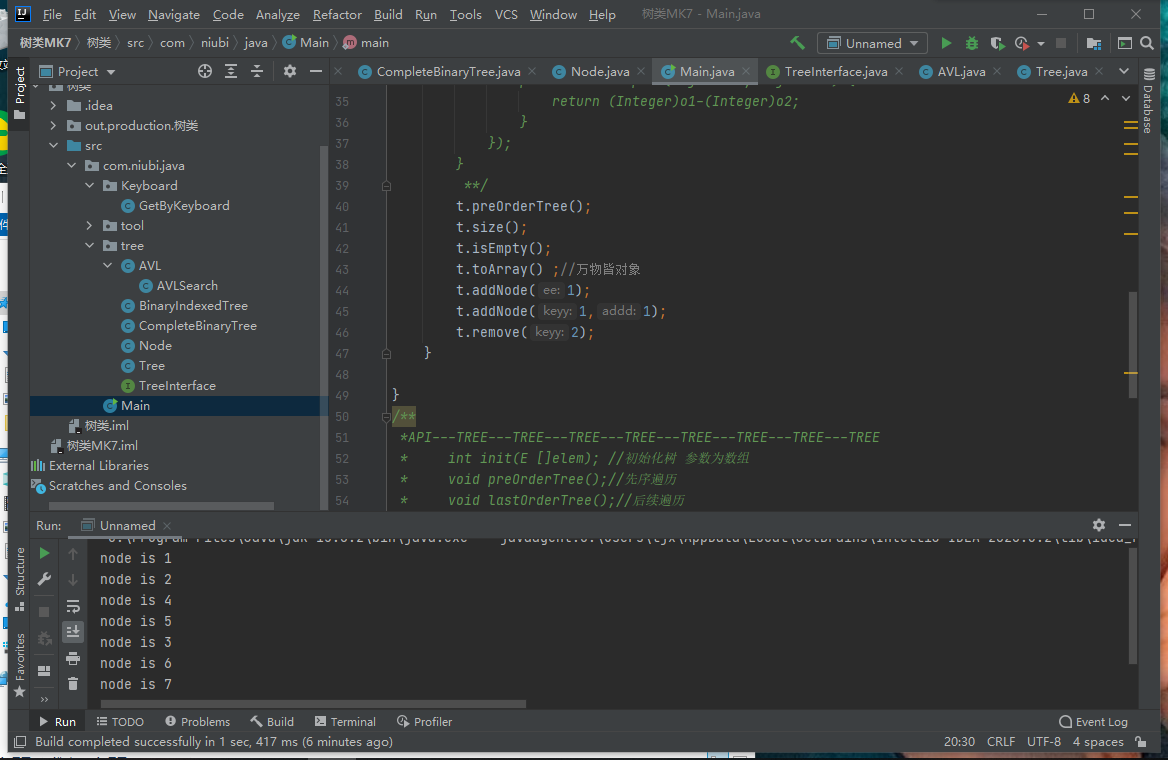
node is 4

node is 5

node is 6

node is 7

实际值：



在测试过后发现完全二叉树中出现错误，将完全二叉树部分先剔除在外，只研究树类，修改代码后：

第一组数据：

1 ,2, 3, null,4,null , 5 ,null,null

操作：

t.preOrderTree();  
t.size();  
t.isEmpty();  
t.toArray() ;//万物皆对象  
t.addNode(1);  
t.addNode(1,1);  
t.remove(2);  
t.reviseNode(2,2);  
t.clear();

理论值：

1

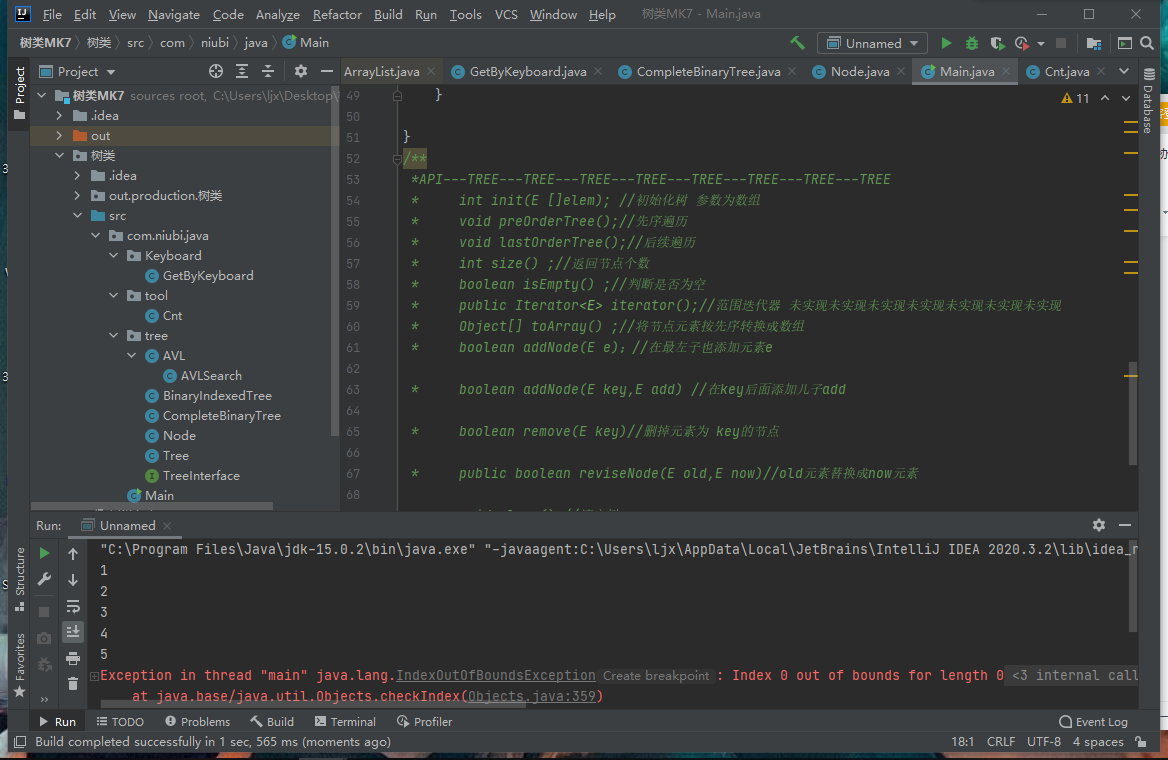
2

3

4

5

实际值：



第二组数据：

1,2,4,null,5,null,null,3,6,null,7

操作：

t.preOrderTree();  
t.size();  
t.isEmpty();  
t.toArray() ;//万物皆对象  
t.addNode(1);  
t.addNode(1,1);  
t.remove(2);  
t.reviseNode(2,2);  
t.clear();

理论值：

1

2

3

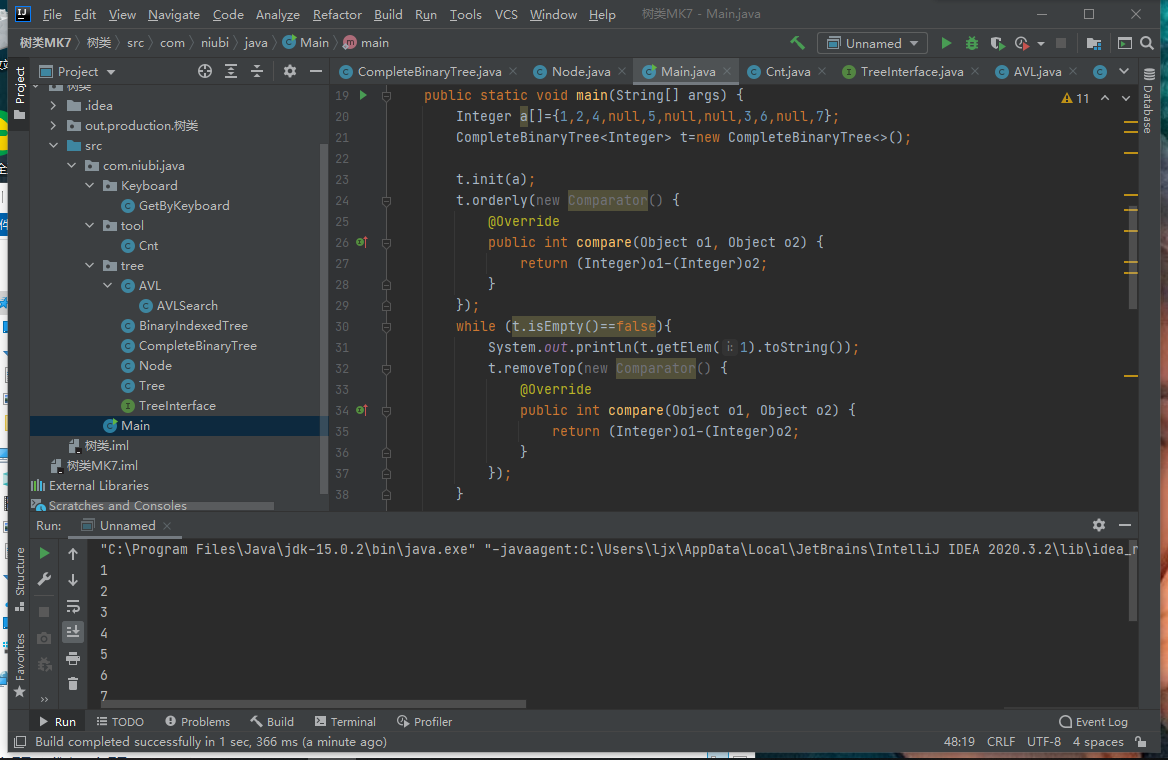
4

5

6

7

实际值：



# 

# 四、总结和分析

课程设计从分析课题入手，围绕树分散到不同的树的类型。根据不同的类型进行分工：

在完成任务的过程中，碰到的第一个问题便是分析课题。二叉树属于这个学期新接触的课题，无法完全分析清楚如何构建和树的特性相关的接口。其次便是，此学期学习的完全二叉树与平衡二叉树有所不同，意味着需要重新学习平衡二叉树的构建和旋转。于是，我们便通过搜查各种关于平衡二叉树的概念以及原理编写代码，使其能与树类接通。平衡的条件在我们的理解看来就是左子树和右子树最低节点高度差绝对值不超过1，通过用旋转达到这个条件，后来又发现左旋转右旋转是有顺序的，破坏平衡的节点是这个树的右子树的左子树时，要进行先右旋转再左旋转来矫正，无法正确求出平衡因子的值。左子树与右子树算法不同，导致最开始代码无法得出正确的平衡效果。