

数据结构课程设计报告

二级学院 计算机科学与工程学院

专 业 软件工程

学生姓名 周翔辉

学 号 11603080122

指导教师 王森

时 间 2017.1

**目录**

[数据结构课程设计报告 3](#_Toc501984028)

[1. 题目1 3](#_Toc501984029)

[1.2 题目设计 4](#_Toc501984030)

[1.3 系统测试 6](#_Toc501984031)

[2. 题目1 9](#_Toc501984032)

[2.1 题目说明 9](#_Toc501984033)

[2.2 题目设计 9](#_Toc501984034)

[2.3 系统测试 12](#_Toc501984035)

[3 题目三 17](#_Toc501984036)

[3.1题目说明 17](#_Toc501984037)

[3.2 题目设计 17](#_Toc501984038)

[3.3 系统测试 21](#_Toc501984039)

[4 题目四 23](#_Toc501984040)

[4.1题目说明 23](#_Toc501984041)

[4.2 题目设计 25](#_Toc501984042)

[4.3 系统测试 31](#_Toc501984043)

[5 题目五 35](#_Toc501984044)

[5.1题目说明 35](#_Toc501984045)

[5.2 题目设计 36](#_Toc501984046)

[5.3 系统测试 37](#_Toc501984047)

[6 题目六 38](#_Toc501984048)

[6.1题目说明 38](#_Toc501984049)

[6.2 题目设计 40](#_Toc501984050)

[6.3 系统测试 42](#_Toc501984051)

[7.课程设计总结 43](#_Toc501984052)

### 数据结构课程设计报告

### 1. 题目1

###### 1.1 题目说明

利用哈夫曼编码进行信息通讯可以大大提高信道利用率，缩短信息传输时间，降低传输成本。但是，这要求在发送端通过一个编码系统对待传数据预先编码；在接收端将传来的数据进行译码(复原)。对于双工信道 (即可以双向传输信息的信道)，每端都需要一个完整的编/译码系统。试对于任意的一段文本（可能是直接输入的，也可能是保存在本地文件中或者网络上的)，写一个哈夫曼码的编译码系统。

[基本要求]  
一 个完整的系统应具有以下功能：

**(l)I:初始化 (Initialization)**。从终端读入字符集大小 n，及 n 个字符和 m 个权值，建立哈夫曼树，并将它存于文件 hfmtree 中。

**(2)C:编码 (Coding)**。利用已建好的哈夫曼树(如不在内存，则从文件 hfmtree 中读入)，对文件 tobetrans 中的正文进行编码，然后将结果存入文件 codefile 中。

**(3)D:解码(Decoding)**。利用已建好的哈夫曼树将文件 codefile 中的代码进行译码，结果存入文件 textfile 中。

**(4)P:打印代码文件 (Print)**。将文件 codefile 以紧凑格式显示在终端上，每行 50 个代码。同时将此字符形式的编码文件写入文件 codeprint 中。

**(5)T:打印哈夫曼树 (Tree printing)**。将已在内存中的哈夫曼树以直观的方式 (树或凹入  
表形式)显示在终端上，同时将此字符形式的哈夫曼树写入文件 treeprint 中。

[实现提示]  
 可以根据题目要求把程序划成 5 个模块，设计成菜单方式，每次执行一个模块后返回菜单。除了初始化(I)过程外，在每次执行时都经过一次读取磁盘文件数据。这是因为如果在程序执行后一直没有进行初始化(I)过程，为了能使后面的操作顺利进行，可以通过读取旧的数据来进行工作。比如：如果程序的工作需要的字符集和权值数据是固定的，只要在安装程序时进行一次初始(I)化操作就可以了。再次运行程序时，不管进行哪项操作都可以把需要的数据读入到内存。

### 1.2 题目设计

###### 1.2.1 系统静态模型

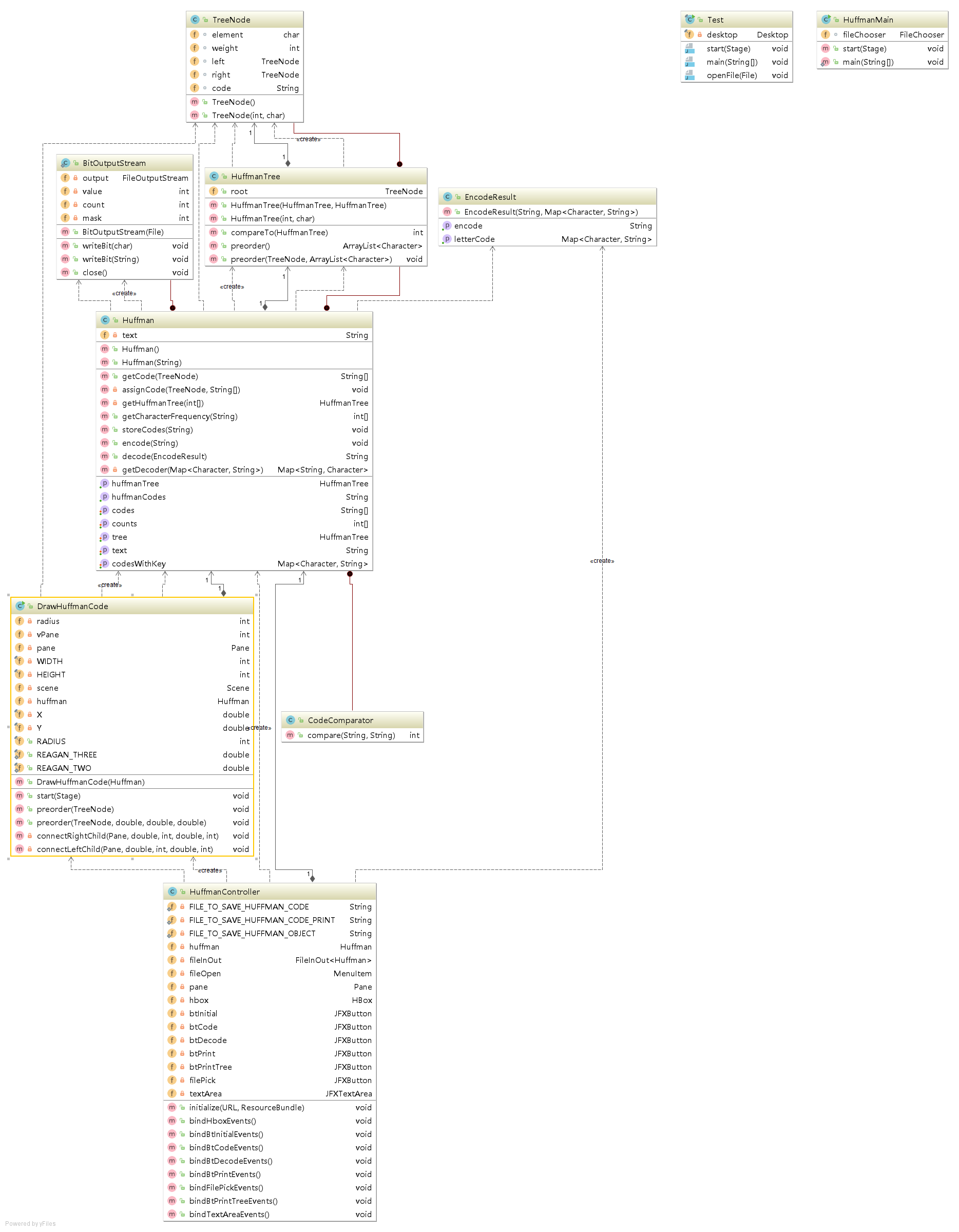


图1-1 题目1系统静态模型

###### 1.2.2 存储结构设计

使用数组存储编码的的哈夫曼编码

###### 1.2.3 关键算法设计

1 编码

**public void** encode(String filename) **throws** IOException {  
 BufferedInputStream fileInput = **new** BufferedInputStream(  
 **new** FileInputStream(**new** File(filename)));  
 BitOutputStream output = **new** BitOutputStream(**new** File(filename + **".new"**));  
  
 **int** r;  
 **while** ((r = fileInput.read()) != -1 ) {  
 output.writeBit(**codes**[r]);  
 }  
}

2 解码

**public** String decode(EncodeResult decodeResult) {  
 *// 解码得到的字符串* StringBuffer decodeStr = **new** StringBuffer();  
 *// 获得解码器* Map<String, Character> decodeMap = getDecoder(decodeResult  
 .getLetterCode());  
 *// 解码器键集合* Set<String> keys = decodeMap.keySet();  
 *// 待解码的（被编码的）字符串* String encode = decodeResult.getEncode();  
 *// 从最短的开始匹配之所以能够成功，是因为哈夫曼编码的唯一前缀性质  
 // 临时的可能的键值* String temp = **""**;  
 *// 改变temp值大小的游标* **int** i = 1;  
 **while** (encode.length() > 0) {  
 temp = encode.substring(0, i);  
 **if** (keys.contains(temp)) {  
 Character character = decodeMap.get(temp);  
 decodeStr.append(character);  
 encode = encode.substring(i);  
 i = 1;  
 } **else** {  
 i++;  
 }  
 }  
 **return** decodeStr.toString();  
}

### 1.3 系统测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 1 | | 版本号 | | 1 |
| 测试环境 | Windows10 JDK1.8 IDEA2017.2 | | | | |
| 用例名称 | 编码 | | | | |
| 前提条件 | 输入ascii码以内的字符 | | | | |
| 测试步骤 | 读取输入字符，建立哈夫曼树，展示编码 | | | | |
| 输入数据 | The process | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 | 无 | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.20 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.20 |
| 再测试人 | 周翔辉 | 再测试日期 | | | 2017.12.20 |
| 问题修改摘要 | 无 | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 2 | | 版本号 | | 1 |
| 测试环境 | Windows10 JDK1.8 IDEA2017.2 | | | | |
| 用例名称 | 解码 | | | | |
| 前提条件 | 编码之后 | | | | |
| 测试步骤 | 读取输入字符，建立哈夫曼树，展示编码 | | | | |
| 输入数据 | 01111110001010101110100101100110110 | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 | 无 | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.20 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.20 |
| 再测试人 | 周翔辉 | 再测试日期 | | | 2017.12.20 |
| 问题修改摘要 | 无 | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 3 | | 版本号 | | 1 |
| 测试环境 | Windows10 JDK1.8 IDEA2017.2 | | | | |
| 用例名称 | 打印哈夫曼树 | | | | |
| 前提条件 | 编码之后 | | | | |
| 测试步骤 | 读取输入字符，建立哈夫曼树，展示编码 | | | | |
| 输入数据 | Hello world | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 | 无 | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.20 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.20 |
| 再测试人 | 周翔辉 | 再测试日期 | | | 2017.12.20 |
| 问题修改摘要 | 无 | | | | |
| 修改人 |  | | 修改日期 | |  |

### 2. 题目1

### 2.1 题目说明

一副牌中抽去大小王剩下 52 张（如果初练也可只用 1～10 这 40 张牌)，任意抽取 4 张牌（称牌组)，用加、减、乘、除（可加括号)把牌面上的数算成 24。每张牌必须用一次且只能用一次，如抽出的牌是 3、 8、 8、 9，那么算式为（ 9-8)×8×3 或 3×8+（ 9-8)或（ 9- 8÷8)×3等。本题主要考查栈、集合、数组、递归、穷举等知识。

### 2.2 题目设计

###### 2.2.1 系统静态模型

1) 版本1

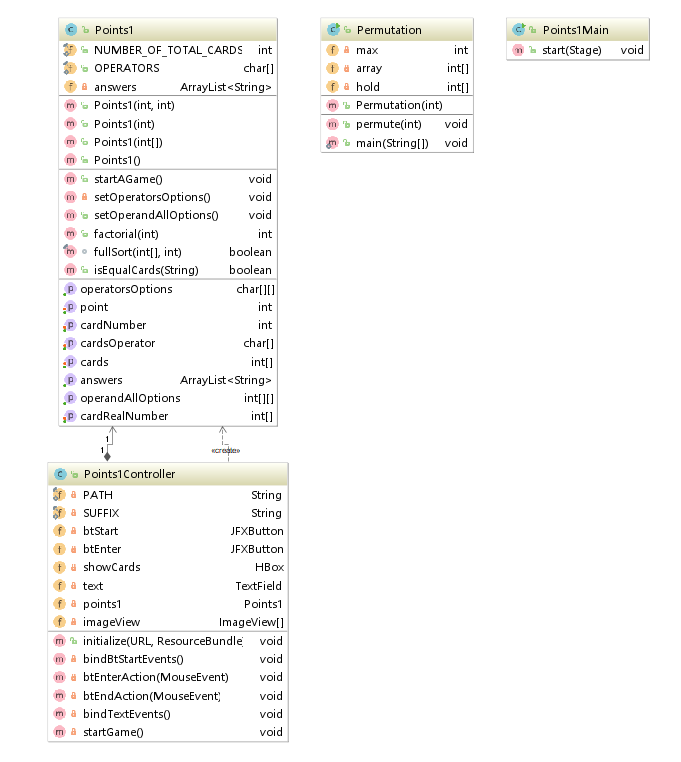


图2-1 题目2版本1系统静态模型

2) 版本2

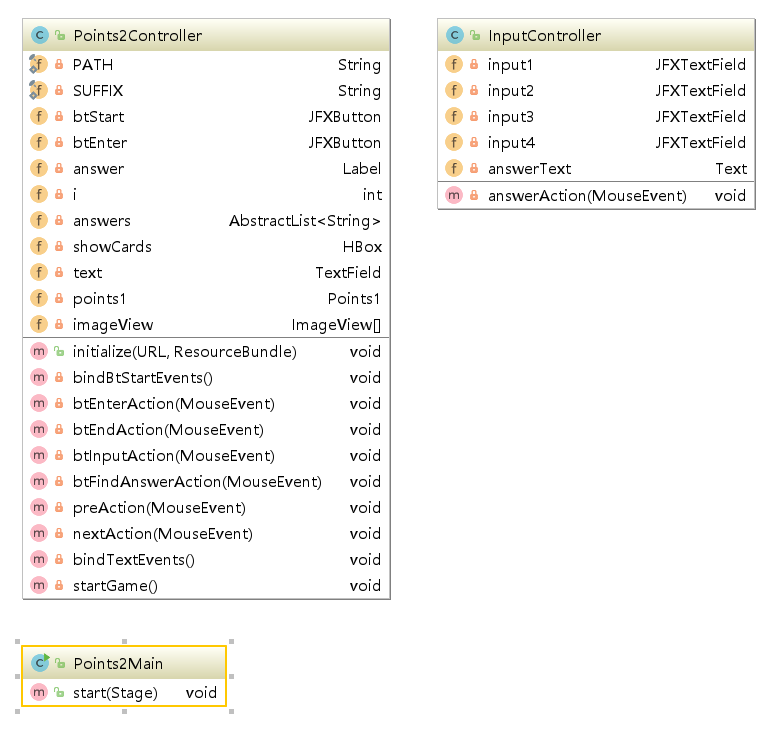


图2-2 题目2版本2系统静态模型

3) 版本3

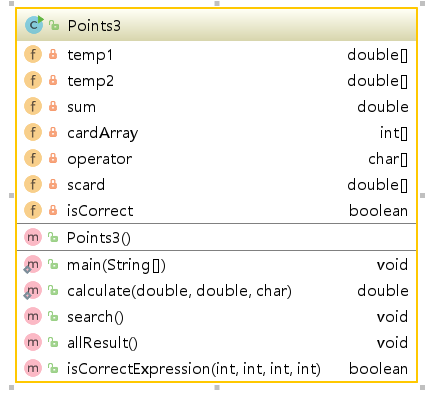


图2-3 题目2版本3系统静态模型

###### 2.2.2 存储结构设计

使用栈存储操作符和操作数；

使用数组存储答案。

###### 2.2.3 关键算法设计

1）计算中缀表达式，使用栈；

**public static double** calculateInfix(String expression){  
 *//创建一个operandStack栈来存储操作数* Stack<Double> operandStack  
 = **new** Stack<>();  
  
 *//创建一个operatorStack栈来存储操作符* Stack<Character> operatorStack  
 = **new** Stack<Character>();  
  
 *//*······中间计算过程  
 **return** operandStack.pop();  
}

2）穷举操作符和操作数

*//穷举操作符的种数***private void** setOperatorsOptions(){  
  
 **int** length = (**int**)Math.*pow*(**cardNumber**, **cardNumber** -1);  
 **operatorsOptions** = **new char**[length][**cardNumber**-1];  
 **for** (**int** i = 0; i < length; i++) {  
 **for** (**int** l = 0; l < **cardNumber**-1; l++) {  
 *//关键操作* **operatorsOptions**[i][l] = ***OPERATORS***[(i) % ((**int**)Math.*pow*(**cardNumber**, **cardNumber**-1-l)) / ((**int**)Math.*pow*(**cardNumber**, **cardNumber**-1-l-1))];  
 }  
 }  
}

*//穷举操作数***public void** setOperandAllOptions() {  
 **int** length = factorial(**cardNumber**);  
 **operandAllOptions** = **new int**[length][**cardNumber**];  
 Arrays.*sort*(**cards**);  
 **for** (**int** i = 0; i < length ; i++) {  
 **operandAllOptions**[i] = **cards**.clone();  
 fullSort(**cards**, **cardNumber**);  
 }  
}

3) 表达式转化

**public static** String  
changeCharsNumbersToExpression(**char**[] operators, **int**[] operands){  
 String expression = **""**;*//最终的表达式* LinkedList<String> queue = **new** LinkedList<>();  
 queue.offer(operands[0] + **" "**);  
 **for** (**int** i = 0; i < operators.**length**; i++) {  
 queue.offer(operators[i] + **" "**);  
 queue.offer(operands[i+1] + **" "**);  
 **if**(i != operators.**length**-1){  
 queue.addFirst(**"("**);  
 queue.addLast(**")"**);  
 }  
  
 }  
 **for** (**int** i = 0; i < queue.size(); i++) {  
 expression += queue.get(i);  
 }  
 **return** expression;  
}

### 2.3 系统测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 1 | | 版本号 | | 1.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 24点扑克牌游戏 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 | 根据展示的纸牌输入合适的表达式 | | | | |
| 输入数据 | 表达式（9-6）+ 11 + 7 | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 2 | | 版本号 | | 1.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 24点扑克牌游戏 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 | 根据展示的纸牌输入合适的表达式 | | | | |
| 输入数据 | 表达式(7+1)\* (13-10) | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 3 | | 版本号 | | 2.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 24点扑克牌游戏 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 | 输入1-13内的四个数，寻找符合24点的 | | | | |
| 输入数据 | 2 3 4 4 | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 4 | | 版本号 | | 2.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 24点扑克牌游戏 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 |  | | | | |
| 输入数据 |  | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

### 3 题目三

### 3.1题目说明

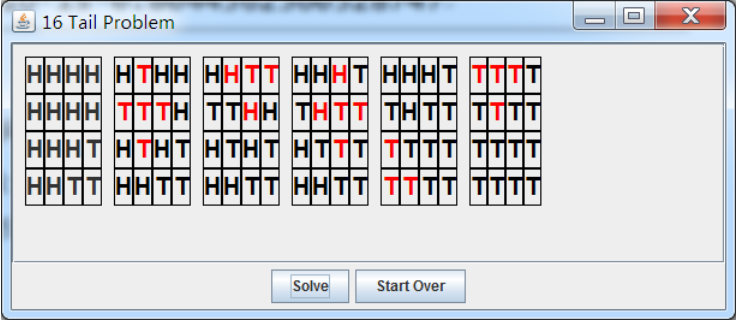
教材中的 9 枚硬币问题使用的是 3\*3 的矩阵，假设在一个 4\*4 的矩阵中放置了 16 枚硬币。该问题可进一步变化为如 2\*3， 2\*4， 3\*4 等等任意结构的情形。本题主要考查对图的结构和图的广度优先遍历操作的掌握。

**版本 1**（ 30%)

参考 9 枚硬币反面问题的模型，建立 16 枚硬币反面问题的模型，以及其他结构的模型。

**版本 2**（ 40%)

参考 9 枚硬币反面问题的解决方法，解决 16 枚硬币的反面问题。进一步解决其它结构的反面问题。



**版本 3**（ 30%)

修改硬币翻转规则，如规则改为对角线上的邻居被翻转，或者任意自定义的翻转规则。重新解决上述问题。

### 3.2 题目设计

###### 3.2.1 系统静态模型

1) 版本1

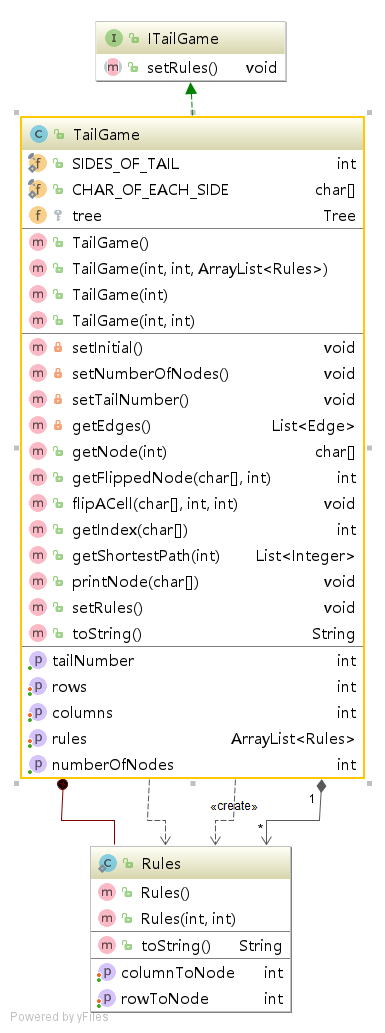


图3-1 题目3 版本1 uml图

2) 版本2

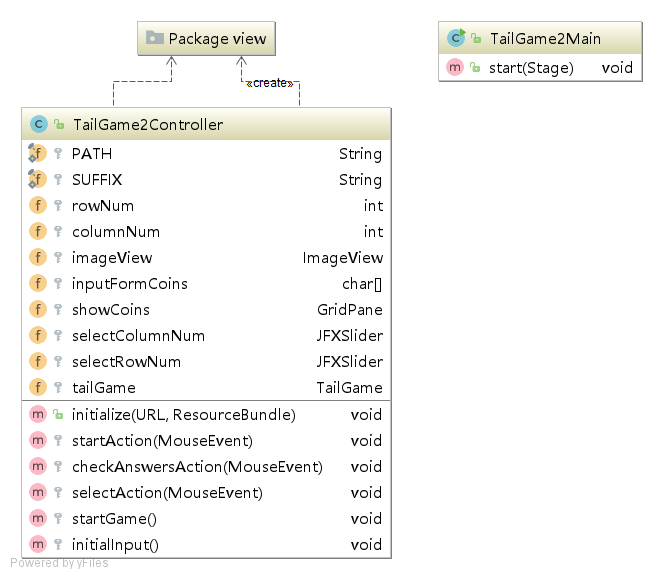


图3-2 题目三 版本2 uml图

3) 版本3

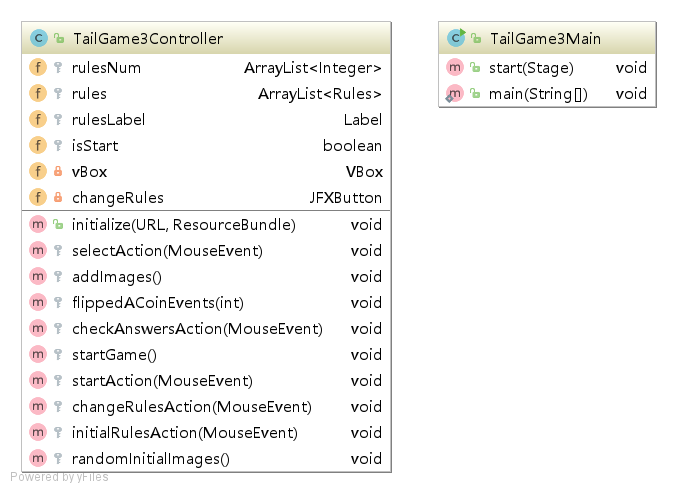


图3-3 题目三 版本3 uml图

###### 3.2.2 存储结构设计

1）用图来保存硬币的所有情况；

2）用数组来保存图中每个节点硬币的翻转情况。

###### 3.2.3 关键算法设计

1）设置规则

**public void** setRules(ArrayList<Rules> rules) {  
  
 **this**.**rules** = rules;  
}

*//规则类***public static class** Rules{  
 **int rowToNode**;*//相对于position的行数* **int columnToNode**;*//相对与position的列数* **public** Rules() {  
 }  
  
 **public** Rules(**int** rowToNode, **int** columnToNode) {  
 **this**.**rowToNode** = rowToNode;  
 **this**.**columnToNode** = columnToNode;  
 }  
  
 **public int** getRowToNode() {  
 **return rowToNode**;  
 }  
  
 **public void** setRowToNode(**int** rowToNode) {  
 **this**.**rowToNode** = rowToNode;  
 }  
  
 **public int** getColumnToNode() {  
 **return columnToNode**;  
 }  
  
 **public void** setColumnToNode(**int** columnToNode) {  
 **this**.**columnToNode** = columnToNode;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Rules{"** +  
 **"rowToNode="** + **rowToNode** +  
 **", columnToNode="** + **columnToNode** +  
 **'}'**;  
 }  
}

2）翻转节点

**public int** getFlippedNode(**char**[] node, **int** position){  
 **int** row = position / **rows**;*//指定下标所在的行数* **int** column = position % **rows**;*//指定下标的列数  
  
 //设置规则* **for** (**int** i = 0; i < **rules**.size(); i++) {  
 flipACell(node, row + **this**.**rules**.get(i).getRowToNode(),   
 column + **this**.**rules**.get(i).getColumnToNode());  
 }  
  
 **return** getIndex(node);  
}

### 3.3 系统测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 1 | | 版本号 | | 2.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 16枚硬币的反面问题 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 | 点击可以改变初始状态，查看答案按钮开始求解过程 | | | | |
| 输入数据 |  | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 2 | | 版本号 | | 3.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 16枚硬币的反面问题 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 | 点击可以改变初始状态，查看答案按钮开始求解过程 | | | | |
| 输入数据 | 规则 | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

### 4 题目四

### 4.1题目说明

线性表、树、图的操作和演示

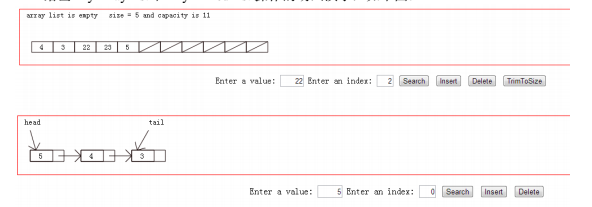
**版本 1** （40%)

实现接口 MyList、抽象类 MyAbstractList、基于顺序存储线性表 MyArrayList、基于链

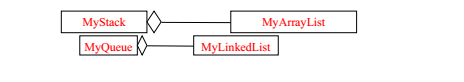
式存储的线性表 MyLinkedList。实现方式如下：



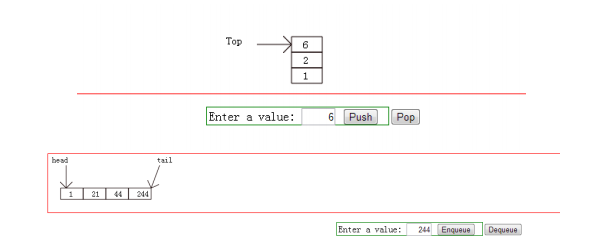
给出 MyArrayList 和 MyLinkedList 操作的动画演示，如下图：



实现栈 MyStack 和队列 MyQueue。实现方式如下：



给出 MyStack 和 MyQueue 的动画操作，如下图：

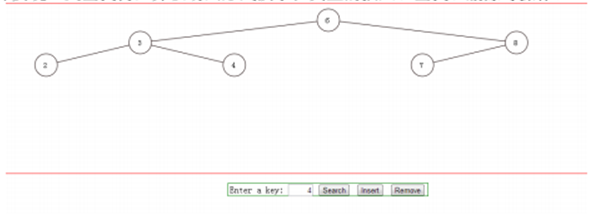


MyArrayList MyStack

MyLinkedList MyQueue

**版本 2** （30%)

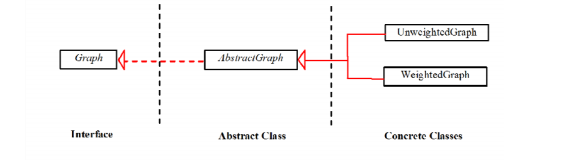
设计实现二叉查找树，并以动画形式演示在其上的插入、查找、删除等操作。



**版本 3** （30%)

设计实现接口 Graph、抽象类 AbstractGraph、具体类 UnweightedGraph 和 WeightedGraph

类。实现方式如下：



并用动画方式显示无权图的深度优先遍历、广度优先遍历；

使用动画方式显示带权图的最小生成树、最短路径：

### 4.2 题目设计

###### 4.2.1 系统静态模型

1）版本一

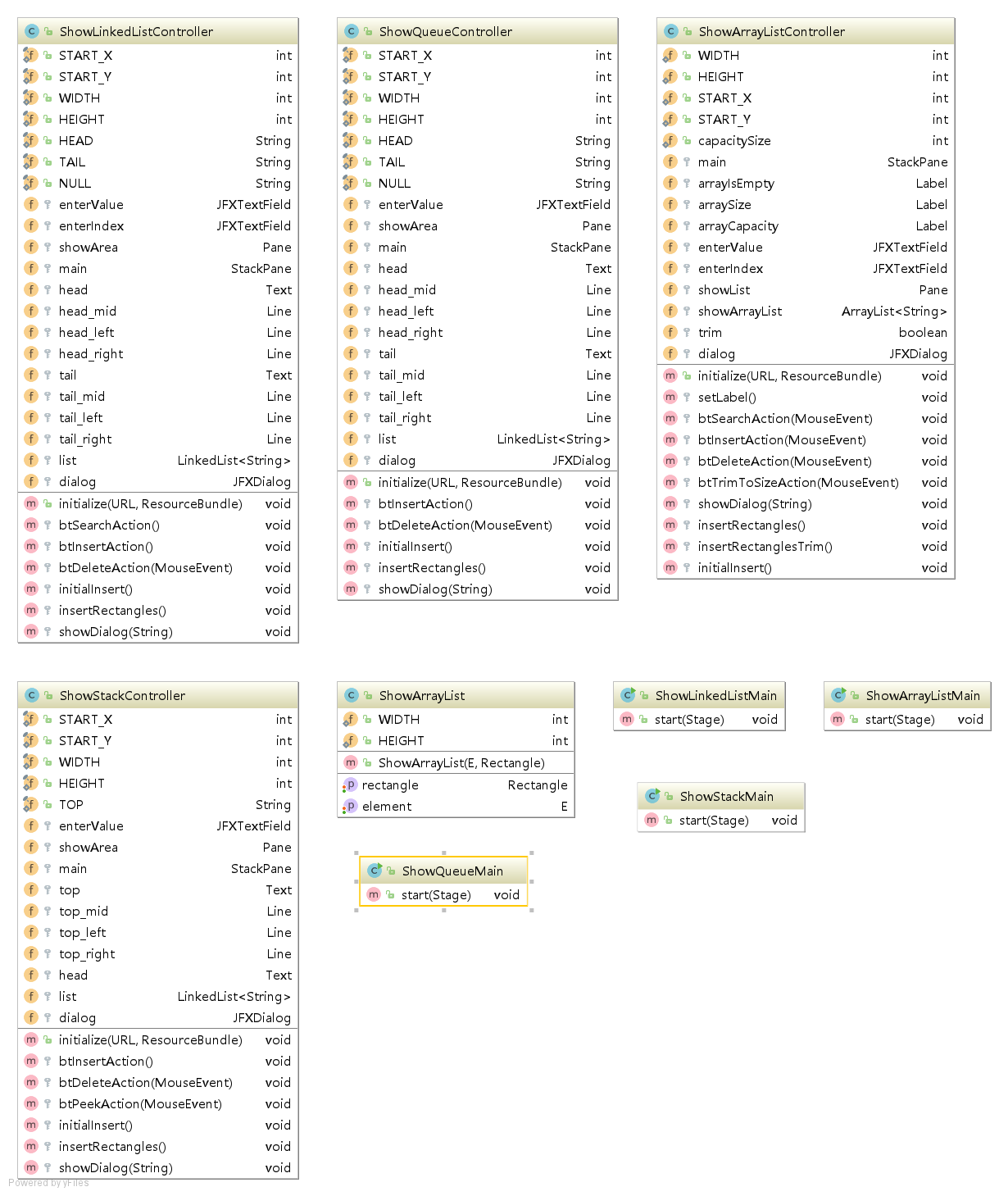


图4-1 版本1 集合框架的uml图

2）版本二

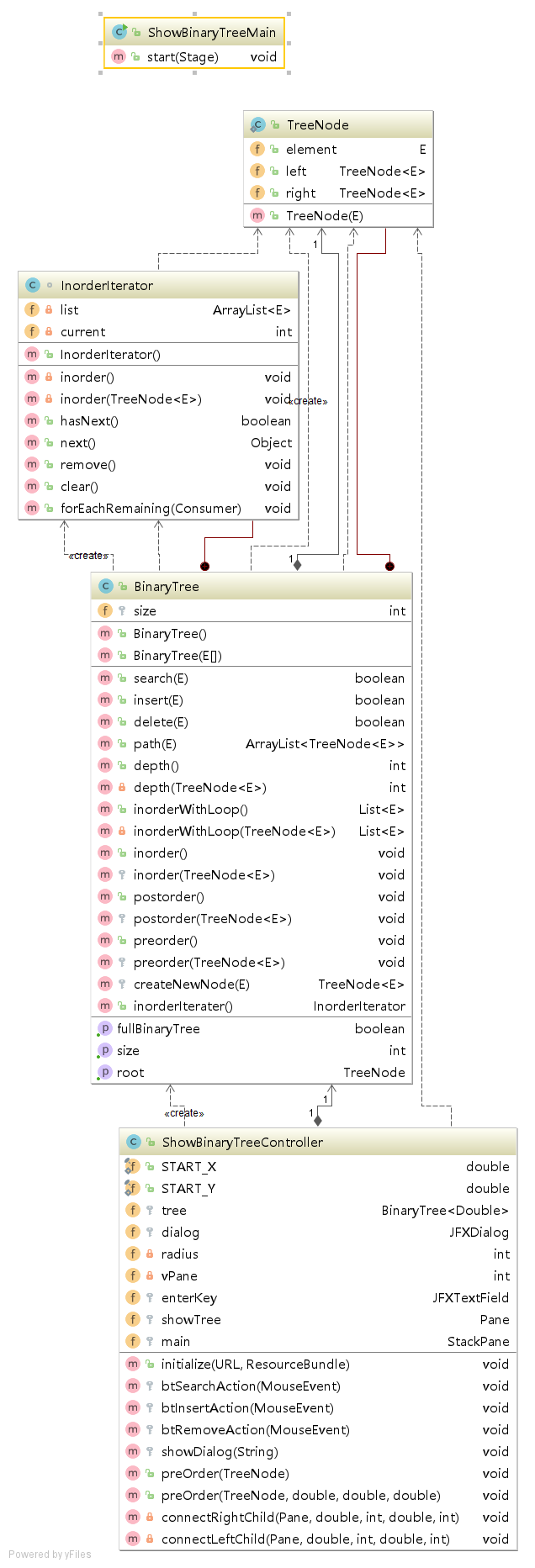


图4-2 版本2 二叉树的uml图

3）版本三

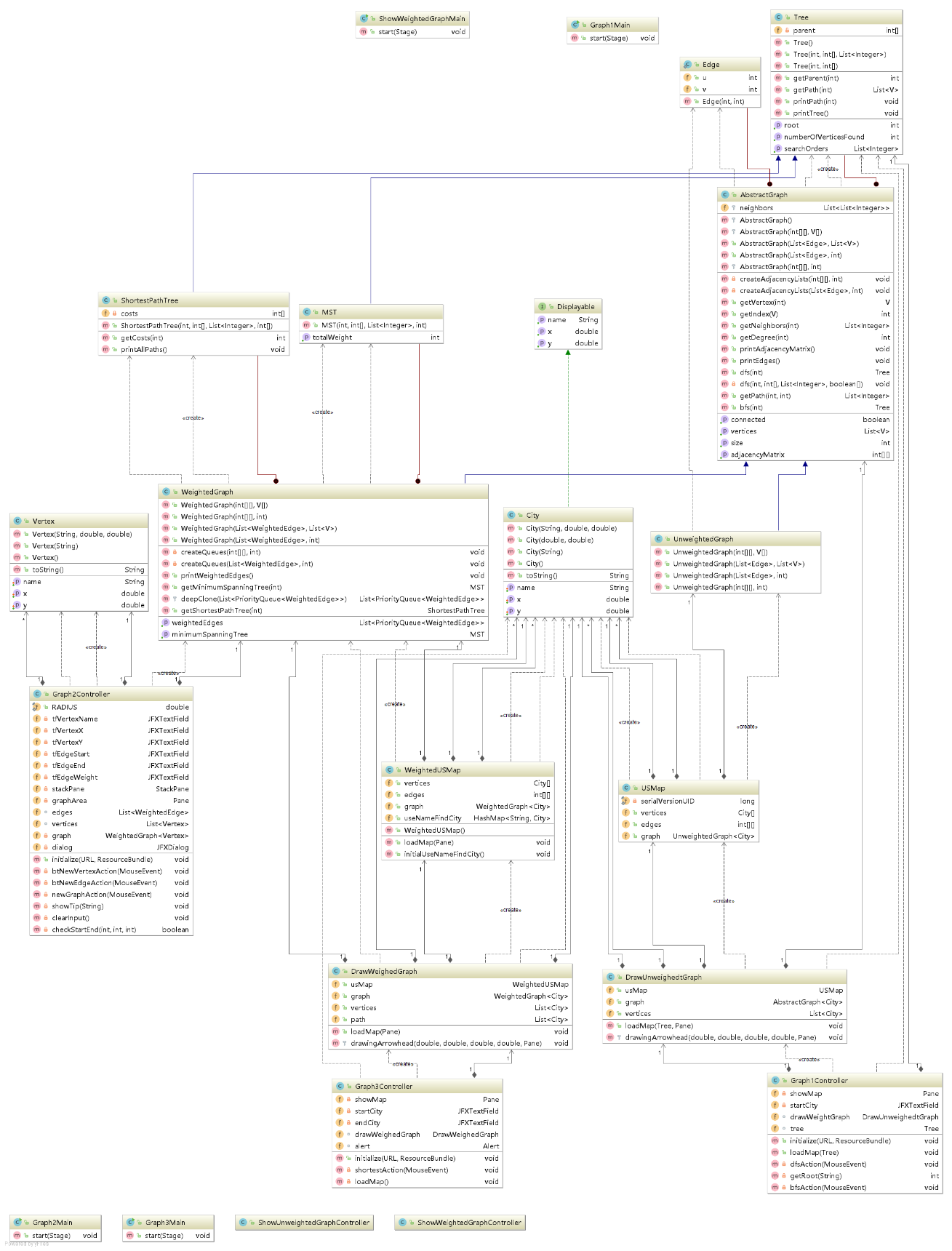


图4-3 版本3展示图的uml图

###### 4.2.2 存储结构设计

线性表存储输入的数据

###### 4.2.3 关键算法设计

1. 版本一

* 线性表的增加删除
* 链表的增加删除
* 栈的增加删除
* 队列的增加删除

2) 版本二

删除：

**public boolean** delete(E e) {  
 *//定位父节点和当前节点* TreeNode<E> parent = **null**;  
 TreeNode<E> current = **root**;  
 *//循环遍历树找到父节点和当前节点* **while** (current != **null**){  
 *//当e比current的元素大时，说明e应该加在其右边* **if**(e.compareTo(current.**element**) > 0){  
 parent = current;  
 current = current.**right**;  
 }  
 *//当e比current的元素小时，说明e应该加在其左边* **else if**(e.compareTo(current.**element**) < 0){  
 parent = current;  
 current = current.**left**;  
 }  
 *//该元素已经找到* **else** {  
 **break**;  
 }  
 }  
 *//当树中没有此元素时返回false* **if**(current == **null**){  
 **return false**;  
 }  
 *//当该节点没有左子节点时* **if**(current.**left** == **null**){  
 *//连接父节点, 当父节点为空时说明删除该元素后树没有元素* **if**(parent == **null**){  
 **root** = current.**right**;  
 }  
 *//当删除节点后父节点不为空时* **else** {  
 *//如果被删除元素e小于父节点的元素，则将父节点的左节点赋给被删除元素的右节点* **if**(e.compareTo(parent.**element**) < 0){  
 parent.**left** = current.**right**;  
 }  
 *//将父节点的右节点赋给被删除元素的右节点* **else** {  
 parent.**right** = current.**right**;  
 }  
 }  
  
 }  
 **else**{  
 TreeNode<E> parentOfRightMost = current;  
 TreeNode<E> rightMost = current.**left**;  
 **while** (rightMost.**right** != **null**){  
 parentOfRightMost = rightMost;  
 rightMost = rightMost.**right**;  
 }  
 current.**element** = rightMost.**element**;  
 **if**(parentOfRightMost.**right** == rightMost){  
 parentOfRightMost.**right** = rightMost.**left**;  
 }  
 **else** {  
 parentOfRightMost.**left** = rightMost.**left**;  
 }  
 }  
 **size**--;  
 **return true**;  
}

1. 版本3

得到最短路径：

**public** ShortestPathTree getShortestPathTree(**int** sourceIndex){  
 List<Integer> T = **new** ArrayList<>();  
 T.add(sourceIndex);  
 **int** numberOfVertices = **vertices**.size();  
 **int**[] parent = **new int**[numberOfVertices];  
 parent[sourceIndex] = -1;  
 **int**[] costs = **new int**[numberOfVertices];  
 **for** (**int** i = 0; i < costs.**length**; i++) {  
 costs[i] = Integer.***MAX\_VALUE***;  
 }  
 costs[sourceIndex] = 0;  
 List<PriorityQueue<WeightedEdge>> queues = deepClone(**this**.**queues**);  
  
 **while**(T.size() < numberOfVertices){  
 **int** v = -1;  
 **int** smallestCost = Integer.***MAX\_VALUE***;  
 **for** (**int** u : T  
 ) {  
 **while** (!queues.get(u).isEmpty() &&  
 T.contains(queues.get(u).peek().**v**)){  
 queues.get(u).remove();  
 }  
 **if**(queues.get(u).isEmpty()){  
 **continue**;  
 }  
 WeightedEdge edge = queues.get(u).peek();  
 **if**(costs[u] + edge.**weight** < smallestCost){  
 v = edge.**v**;  
 smallestCost = costs[u] + edge.**weight**;  
 parent[v] = u;  
 }  
 }  
 T.add(v);  
 costs[v] = smallestCost;  
 }  
 **return new** ShortestPathTree(sourceIndex, parent, T, costs);  
}

得到一棵最小生成树

**public** MST getMinimumSpanningTree(**int** startingIndex) {  
 List<Integer> T = **new** ArrayList<>();  
 T.add(startingIndex);  
 **int** numberOfVertices = **vertices**.size();*//顶点个数* **int**[] parent = **new int**[numberOfVertices];*//顶点的父顶点  
 //初始化所有顶点的父顶点为-1* **for** (**int** i = 0; i < parent.**length**; i++) {  
 parent[i] = -1;  
 }  
 **int** totalWeight = 0;*//生成树的权重和  
 //克隆这个队列，以便保持原有的队列* List<PriorityQueue<WeightedEdge>> queues = deepClone(**this**.**queues**);  
 *//是否找到所有的顶点？* **while**(T.size() < numberOfVertices){  
 *//在T中搜索最小的边* **int** v = -1;  
 **double** smallestWeight = Double.***MAX\_VALUE***;  
 **for** (**int** u : T  
 ) {  
 **while** (!queues.get(u).isEmpty() && T.contains(queues.get(u).peek().**v**)){  
 queues.get(u).remove();  
 }  
 **if**(queues.get(u).isEmpty()){  
 **continue**;  
 }  
 *//当前u的邻接边最小权值* WeightedEdge edge = queues.get(u).peek();  
 **if**(edge.**weight** < smallestWeight){  
 v = edge.**v**;  
 smallestWeight = edge.**weight**;  
 parent[v] = u;  
 }  
 }  
 T.add(v);  
 totalWeight += smallestWeight;  
 }  
 **return new** MST(startingIndex, parent, T, totalWeight);  
}

### 4.3 系统测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 1 | | 版本号 | | 1.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 线性表的演示 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 |  | | | | |
| 输入数据 | 1 2 3 | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 2 | | 版本号 | | 2.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 树的演示 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 |  | | | | |
| 输入数据 | 5 6 4 2 1 3 | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 3 | | 版本号 | | 3.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 图的演示 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 |  | | | | |
| 输入数据 | 起点城市 Dallas  4个点  起点城市：Los Angeles 终点城市：Boston | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

### 5 题目五

### 5.1题目说明

一个农夫带着—只狼、一只羊和—棵白菜，身处河的南岸。他要把这些东西全部运到北岸。他面前只有一条小船，船只能容下他和—件物品，另外只有农夫才能撑船。如果农夫在场，则狼不能吃羊，羊不能吃白菜，否则狼会吃羊，羊会吃白菜，所以农夫不能留下羊和白菜自己离开，也不能留下狼和羊自己离开，而狼不吃白菜。请求出农夫将所有的东西运过河的方案。

### 5.2 题目设计

###### 5.2.1 系统静态模型

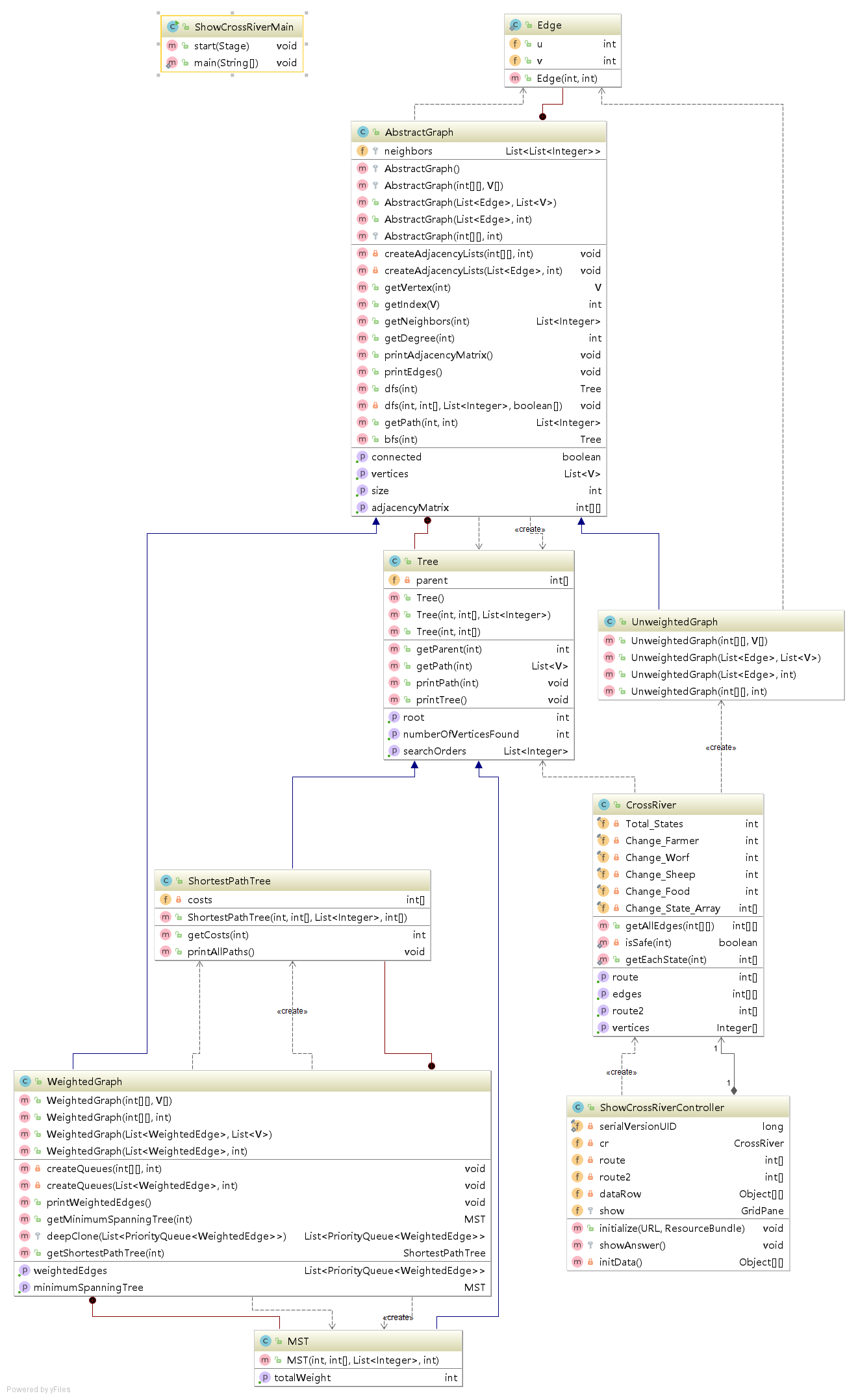


图 5-1 题目5农夫过河问题uml图

###### 5.2.2 存储结构设计

数组存储在不同情况下农夫，狼，羊，白菜的位置

###### 5.2.3 关键算法设计

1. 使用十进制数和二进制数表示不同的状态
2. 二进制的的操作表示状态的改变和判断

*//得到安全的状态״̬***public int**[][] getEdges() {  
 **int**[][] edges = **new int**[**Total\_States**][4];*//16个4位二进制状态* **for**(**int** i = 0 ; i < edges.**length** ; i++) *//共16个状态   
 //初始状态赋值为-1* **for**(**int** j = 0 ;j< edges[i].**length** ;j++){ *//4位二进制数* edges[i][j] = -1 ;*//初始化所有为-1，即未访问状态* }   
 **for**(**int** locate = 0 ; locate < 16 ;locate++) {   
 **int** nextState = -1;   
 **for** (**int** i = 0; i < 4; i++) {   
 **if** (i == 0) {  
 *//农夫自己走* nextState = locate ^ **Change\_State\_Array**[i];*//当两对应的二进位相异时，结果为1* }  
 **else**{   
 nextState = locate ^ **Change\_State\_Array**[0] *//农夫走并和另外的* ^ **Change\_State\_Array**[i];  
 }   
 **if** (*isSafe*(nextState)){  
 edges[locate][i] = nextState ; *//如果下一个状态安全，则存入数组* }  
 }   
 }   
 **return** edges ;  
}

### 5.3 系统测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 1 | | 版本号 | | 1.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 农夫过河问题 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 |  | | | | |
| 输入数据 |  | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

### 6 题目六

### 6.1题目说明

迷宫问题

[**问题描述**]

以一个 m\*n 的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。设计一个程序，

对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

[**基本要求**]

**（1）** 实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得

的通路以三元组（i,j,d)的形式输出，其中：（i,j)指示迷宫中的一个坐标，d 表示走到

下一坐标的方向。

**（2）** 编写递归形式的算法，求得迷宫中所有可能的通路；

**（3）** 以方阵形式输出迷宫及其通路。（选做)

[**测试数据**]

迷宫的测试数据如下：左上角（1，1)为入口，右下角（9，8)为出口。

1 2 3 4 5 6 7 8

0 0 1 0 0 0 1 0

0 0 1 0 0 0 1 0

0 0 0 0 1 1 0 1

0 1 1 1 0 0 1 0

0 0 0 1 0 0 0 0

0 1 0 0 0 1 0 1

0 1 1 1 1 0 0 1

1 1 0 0 0 1 0 1

1 1 0 0 0 0 0 0

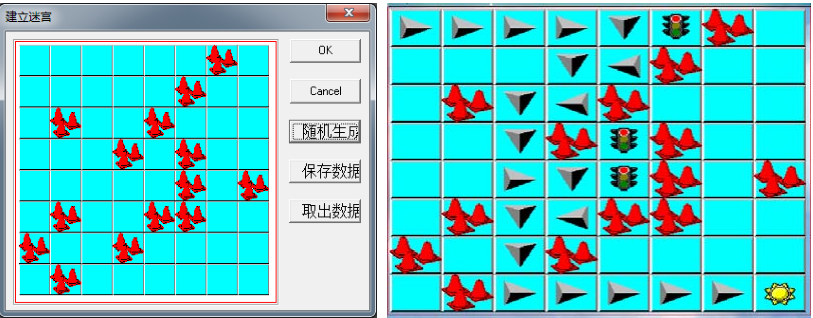
[**实现提示**]

计算机解迷宫通常用的是“穷举求解”方法，即从入口出发，顺着某一个方向进行探索，若能走通，则继续往前进；否则沿着原路退回，换一个方向继续探索，直至出口位置，求得一条通路。假如所有可能的通路都探索到而未能到达出口，则所设定的迷宫没有通路。

可以二维数组存储迷宫数据，通常设定入口点的下标为（1，1)，出口点的下标为（m,n)。

为处理方便起见，可在迷宫的四周加一圈障碍。对于迷宫中任一位置，均可约定有东、南、西、北四个方向可通。

图形界面实现可参见下图。



### 6.2 题目设计

###### 6.2.1 系统静态模型

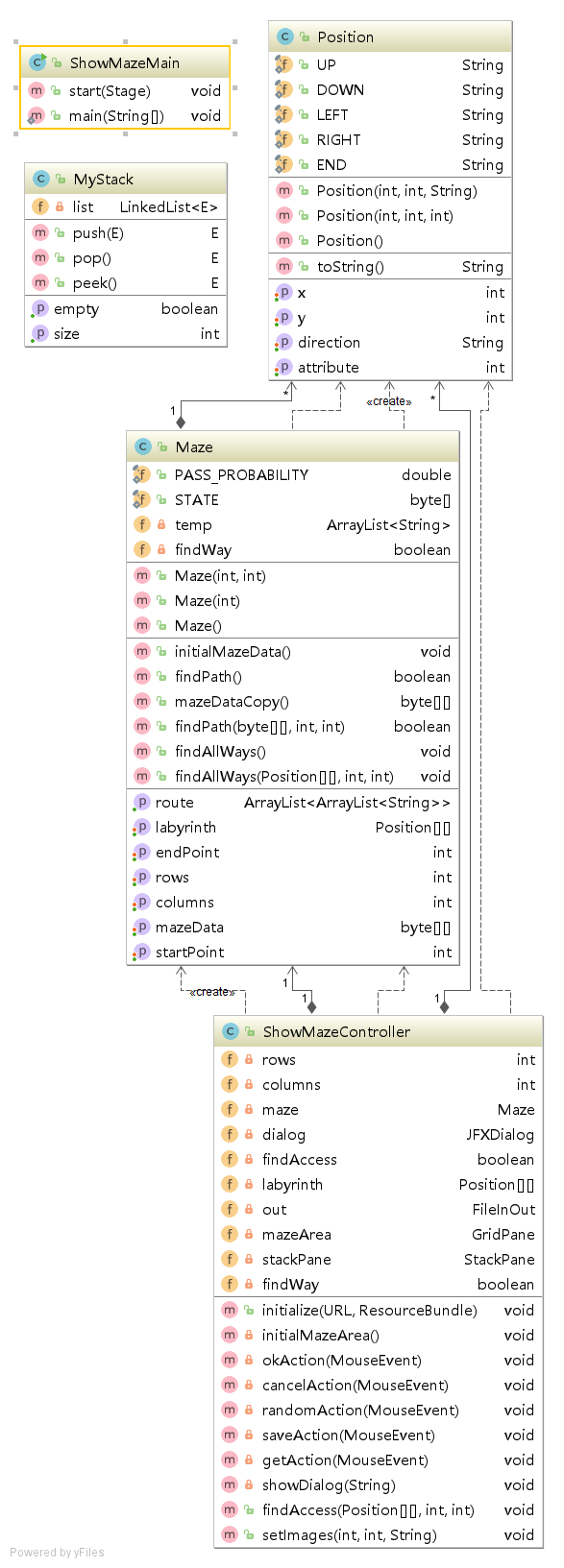


图6-1 迷宫问题uml图

###### 6.2.2 存储结构设计

二位数组存储迷宫数据

###### 6.2.3 关键算法设计

1. 寻找通路

逐一判断当前位置的东西南北方向是否有通路，记录走过的路径

1. 递归寻找通路

寻找路径的过程中，将可以走的位置的信息存在对象中存入到栈中

**public boolean** findPath(**byte** mazeData[][], **int** x, **int** y) {  
 *//到达出口时* **if** (x == **endPoint**/**this**.**columns** && y == **endPoint**%**this**.**columns**) {  
 **findWay** = **true**;  
 **return true**;  
 }  
 **if** (**findWay**) {  
 **return false**;  
 }  
 *//向右搜寻* **if** (x + 1 < mazeData.**length** && y < mazeData[x + 1].**length** && (mazeData[x + 1][y] == ***STATE***[0])) {  
 mazeData[x + 1][y] = ***STATE***[1];  
 *//route.add(new Position(x, y, Position.DOWN));* findPath(mazeData, x + 1, y);  
 }  
 *//向下搜寻* **if** (y + 1 < mazeData[x].**length** && x < mazeData.**length** && (mazeData[x][y + 1] == ***STATE***[0])) {  
 mazeData[x][y + 1] = ***STATE***[1];  
 *//route.add(new Position(x, y, Position.RIGHT));* findPath(mazeData, x, y + 1);  
 }  
 *//向左搜寻* **if** (x - 1 < mazeData.**length** && x - 1 > 0 && y < mazeData[x - 1].**length** && (mazeData[x - 1][y] == ***STATE***[0])) {  
 mazeData[x - 1][y] = ***STATE***[1];  
 *//route.add(new Position(x, y, Position.UP));* findPath(mazeData, x - 1, y);  
 }  
 *//向上搜寻* **if** (y - 1 > 0 && y - 1 < mazeData[x].**length** && x < mazeData.**length** && (mazeData[x][y - 1] == ***STATE***[0])) {  
 mazeData[x][y - 1] = ***STATE***[1];  
 *// route.add(new Position(x, y, Position.LEFT));* findPath(mazeData, x, y - 1);  
 }  
 *//没有找到出口* **if** (x == 0 && y == 0 && !**findWay**) { **return false**;  
 }  
 **return true**;  
}

### 6.3 系统测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 1 | | 版本号 | | 1.0 |
| 测试环境 | Windows10, JDK1.8, IDEA2017.02 | | | | |
| 用例名称 | 迷宫问题 | | | | |
| 前提条件 |  | | | | |
| 测试步骤 |  | | | | |
| 输入数据 |  | | | | |
| 预期输出 |  | | | | |
| 实际输出 |  | | | | |
| 问题描述 |  | | | | |
| 设计人 | 周翔辉 | 设计日期 | | 2017.12.25 | |
| 测试人 | 周翔辉 | 测试日期 | | | 2017.12.25 |
| 再测试人 |  | 再测试日期 | | |  |
| 问题修改摘要 |  | | | | |
| 修改人 |  | | 修改日期 | |  |

### 7.课程设计总结

工作意识、工作方法、问题及解决过程、

通过本次课程设计我学会了很多，不仅加强了我对Java知识的了解，也增加了我对编程的兴趣，更加懂的的编程不只是只靠自己脑子去想的，需要很多的学习与参考，学以致用才是我们现阶段的目标，而不是好高骛远。我想从这些方面来总结一下本次课程设计。

1 工作意识：