**武汉大学计算机学院**

**本科生实验报告**

**基本环节实验**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：高级语言程序设计实验

指 导 教 师 一：常军 讲师

指 导 教 师 二：谭成予 副教授

学 生 学 号 ：2016301500373

学 生 姓 名 ：徐一恒

二○一八年四月

**郑 重 声 明**

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 徐一恒 日期： 2018/04/08

摘 要

基本环节实验的实验目的：练习较为简单的算法，使用高级程序语言完成给定题目，熟悉需求分析、代码编写、修正调整、测试这样完整的程序设计过程。

实验设计主要遵循：面向对象程序设计思想和软件工程原理。

实验内容主要包括：1. 编写程序模拟狗随机走N\*N的地图，探究其成功率和地图大小的关系

2. 编写程序实现中缀后缀表达式转二叉表达树，和二叉表达树输出前中后缀表达式

实验结论为：1. 狗的成功率大概和地图大小成logarithmic关系

2. 表达式的中缀、后缀表达式都可以转化成二叉表达树，二叉表达树又可以通过前中后序遍历来输出前中后缀表达式。其中中缀表达式一般先转化为后缀表达式，然后再转化成表达树。

**关键词：**面向对象；随机走迷宫；二叉表达式

**目 录**

**1** **题目一**

1.1 实验题目及要求说明…………………………………………………………1

1.2 开发环境及开发日期…………………………………………………………1

1.3 代码结构………………………………………………………………………1

1.4 数据结构………………………………………………………………………2

1.5 主要函数的功能描述…………………………………………………………2

1.6 主要算法描述…………………………………………………………………3

1.7 运行效果………………………………………………………………………3

1.8 问题分析与总结………………………………………………………………4

**2**  **题目二**

2.1 实验题目及要求说明…………………………………………………………5

2.2 开发环境及开发日期…………………………………………………………5

2.3 代码结构………………………………………………………………………5

2.4 数据结构………………………………………………………………………6

2.5 主要函数的功能描述…………………………………………………………7

2.6 主要算法描述…………………………………………………………………8

2.7 运行效果………………………………………………………………………10

2.8 问题分析与总结………………………………………………………………12

**结论** …………………………………………………………………………………13

**参考文献** ……………………………………………………………………………13

**附录** …………………………………………………………………………………14

**1 题目一**

**1.1** 实验题目及要求说明

假设有一条狗放在某个城市中心点，它试图逃出城市，此城市有N条南北走向的街道和N条东西走向的街道，所有街道均匀交叉分布构成网格形式。这条狗在逃出城市的过程中，遇到每个交叉路口则按照随机概率的大小选择前进方向，它能够通过灵敏的嗅觉和记忆不走重复路。当狗走到某个交叉路口时，如果三个可选方向均指向以前走过的路口就必须回头，则陷入死胡同状态。

多次模拟此过程，以研究其成功率、总路径、平均路径等性质，并研究地图大小和这些数据间的关系。

**基本要求**

狗尝试逃出的次数设为T。

1. 假设给出某个确定的N值(N=50)，分析并输出这条狗陷入死胡同的概率是多少，行走路径的平均长度是多少？成功逃出的平均路径长度和陷入死胡同的平均路径长度各是多少？
2. 给出一组不同的N值，通过运算分析出N的规模大小与陷入死胡同的概率，这两者之间的联系。

**1.2** 开发环境及开发日期

环境：IntelliJ

日期：2018/3/24

**1.3** 代码结构

采用面向对象程序设计，使用Java作为程序语言。整个程序由两个Java文件构成，定义了两个类：Maze和MazeStatistic。

Maze类是程序的主要部分，将迷宫封装成一个类，用来模拟一次走迷宫，装有迷宫地图和狗的当前位置，定义了狗的一次移动、完整的一次走迷宫、判断可走方向，判断死亡或成功等等的函数。其中主要函数是move和process，调用获取方向、四个方向移动、判断死亡成功等子函数。

MazeStatistic是程序的统计工厂，里面包含封装好的Maze类的对象，用来进行多次模拟（比如模拟50次），用户可自行输入地图大小和模拟次数，并计算最终的统计结果，包括成功率、总路径、平均路径等。

**1.4** 数据结构

地图使用二维整形数组来表示，每个单元表示一个交叉路口，0表示未走过，1表示已经走过。

狗的当前位置用两个整形变量来表示，每次移动变换相应的坐标。

**1.5** 主要函数的功能描述

写在前面：关于函数，我在代码中严格按照Java注释的格式写了详细的注释，包括函数的作用，参数的意义，返回值的意义。因此，在此只取其中重要的比较复杂的函数做介绍。

* + 1. **int[] getDirections()**

获取四个方向的状态，用一个长度为四的一维数组表示，四个单元分别表示上下左右，依次检查这些方向是否走过，能走则值为0，不能走则值为1，最后返回这个一维数组，供move函数使用。若四个方向均不能走，则返回null

返回值：这个方向状态数组。

* + 1. **void move()**

进行一次合法的移动，对象的路程属性加1，当前位置设为已走过。通过getDirection()获得方向信息，然后通过循环生成0-3四个随机数来决定移动方向，只要随机生成的方向不能走（即getDirection()[i] != 0），就继续循环，直到能走为止，然后根据方向调用相应的移动子函数（总共四个，moveUp，moveDown，moveLeft，moveRight）

* + 1. **int process()**

模拟一次完整的走迷宫过程，循环执行move函数，直到狗死掉或者成功走出迷宫。最后返回此次狗走过的总路程长度

返回值：狗走过的总路程长度

* + 1. **void reset()**

重置整个Maze对象，将路程清零，所有路口设为未走过，将狗放回地图中心

* + 1. **void getData()**

循环调用Maze对象的process函数，记录每一次的数据，每次结束后调用Maze.reset()重置Maze对象，直到达到试验次数。最后对数据进行计算处理，得出最后统计结果

**1.6** 主要算法描述

算法非常简单直接，直接模拟狗走迷宫：狗一开始在地图中心，每次行走前先获取所有可以走的方向，然后使用随机数随机选择一个方向，让狗移动，即改变狗的当前坐标，我定义了四个移动函数表示四个方向的移动，路程加1，并将之前的坐标设为已走。若获取可以走的方向时发现无方向可走（四个路口都被走过了），则狗死亡；若已经走到迷宫最外围，则狗成功走出迷宫。

然后在MazeStatistic中模拟n次以上过程，统计数据。

稍微复杂一点的地方在获取所有可走路径和选择方向，我使用的是一个一维数组，共四个元素，分别表示上下左右四个方向，0表示能走，1表示不能走，使用一个获取方向的函数返回这个数组，若都不能走，就返回null，狗死亡。若可以走，则返回该数组，然后循环生成0-3四个随机数，直到碰到可走的那个方向，然后调用那个方向的移动函数。

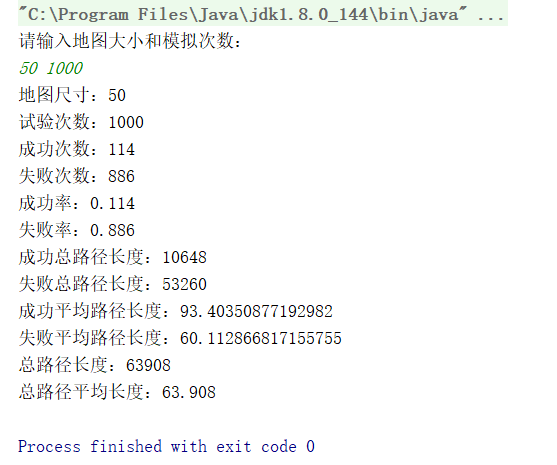
**1.7** 运行效果

运行MazeStatistic里的main函数，首先由用户自己输入迷宫大小和模拟次数，比如输入50和1000，即迷宫为50\*50，总共模拟1000次。

然后会输出此次模拟的所有统计数据，包括地图尺寸、试验次数、成功次数、失败次数、成功率、失败率、成功总路径长度、失败总路径长度、成功平均路径长度、失败平均路径长度、总路径长度、总路径平均长度。

设置合适的模拟次数，多次运行程序，改变地图大小，即可发现地图大小和成功率成反相关。

结果截图：



**1.8** 问题总结与分析

相对而言是一个简单的题目，要注意的问题第一是获得正确的移动方向，一定要设计好算法使各个方向确实可能性相等；第二个是在多次模拟过程中不要忘了重置Maze对象。最终成功地达到了要求。

**2 题目二**

**1.1** 实验题目及要求说明

完成中缀、后缀表达式的二叉表达树的建立，和中缀、后缀和前缀三种表达式之间的转换：输入一个四则运算的后缀或中缀表达式，生成其二叉表达树，然后再分别输出该表达树的前缀、中缀、后缀表达式。

**基本要求：**

1. 表达式支持的运算符自行设定，例如，四则运算。
2. 采用某种方式输入表达式，例如后缀表达式形式。将用户输入的表达式创建成如上图所示的表达式树。
3. 遍历该表达式树，分别输出该表达式的中缀表达式和后缀表达式形式。

**1.2** 开发环境及开发日期

环境：IntelliJ

日期：2018/3/24

**1.3** 代码结构

采用面向对象程序设计，使用Java作为程序语言。整个程序由一个Java文件构成，定义了两个类：ExpressionTree和ETNode。

ETNode是ExpressionTree的内部类，表示表达树中的一个节点，有item、left、right三个属性。

ExpressionTree类是程序的主要部分，包含一个root属性，用来存放根节点。构造函数，可以采用两种生成树的方式：后缀和中缀。根据用户的输入，调用后缀生成函数，或中缀生成函数，生成相应的表达树。建立表达树后，即可生成其前缀、中缀、后缀三种表达式：定义了前序遍历、中序遍历、和后序遍历三个遍历函数，调用这些函数即可获得相应表达式。

**1.4** 数据结构

表达树自然是使用二叉树表示，每个节点用一个ETNode的对象表示，包含item、left、right，item即节点中的内容，可以是运算符或者数字、变量，left为左孩子，right为右孩子。

表达式由字符串数组表示。首先读取用户输入，读为字符串，然后将此字符串分离，分离成一个一个的字符串，用字符串数组存起来。在这里我限定了用户的输入格式，运算符、操作数必须用空格隔开，括号不做要求，比如(a + b \* c) + (d \* e + f) \* g，而不是(a+b\*c)+(d\*e+f)\*g。因为后缀表达式必须使用空格间隔，否则会导致操作数的混淆，比如1234+到底是12+34，还是1+234，而且这其实是人类正常输入表达式的习惯，尤其是程序员的好习惯：程序员写代码时是要在运算符之间加空格的。而正常人在写括号时是不加空格的，因此括号不需用空格分开。同时考虑到分离字符串本身也不是此应用的重点，而且程序员也是可以要求用户遵守格式的。

在进行表达式转换和表达树建立时使用了Stack和ArrayList，Stack用于后缀表达式建立表达树时子树的存放，以及在中缀转后缀时，括号和运算符的存放。ArrayList用在中缀转后缀时输出的存放，因为用ArrayList就不用管下标了。

**1.5** 主要函数的功能描述

写在前面：关于函数，我在代码中严格按照Java注释的格式写了详细的注释，包括函数的作用，参数的意义，返回值的意义。因此，在此只取其中重要的比较复杂的函数做介绍。

**1.5.1 ETNode postGenerate(String[] items)**

生成后缀表达式的表达树。

参数items：处理过的后缀表达式字符串数组，因为我直接用了Java自带的split函数处理字符串，所以此介绍中没有字符串处理的函数。

返回值：生成的表达树的根节点

**1.5.2 String[] inToPost(String[] items)**

将一个中缀表达式转化为一个后缀表达式，用字符串数组表示

参数items：处理过的后缀表达式字符串数组

返回值：转化后的中缀表达式字符串数组

**1.5.3 void inexpression(ETNode node)**

遍历该表达树，并输出其中缀表达式

参数node：ETNode对象，要遍历的表达树的根节点

其他还有后缀和前缀两种遍历和表达式输出，函数功能和思想与此函数基本相同，不再赘述

**1.6** 主要算法描述

主要包括五个算法点：

**1.6.1 用户输入字符串的处理：**

根据我在数据结构部分的说明，我限定了用户的输入格式，因此，我可以直接使用Java内置的函数split(regex)直接分离字符串。这个函数返回分隔后的字符串数组，设定分隔符regex为空格，即可直接转换为字符串数组。

对于括号，括号会被分到和操作数在一个字符串里，我在中后缀表达式转换时做的处理，循环检测这个字符串的第一个和最后一个字符，将括号入栈，更新字符串（去掉括号），再循环。

**1.6.2 后缀表达式建立表达树：**

这个相对简单，利用Java内置的堆栈Stack类存放子树，遍历前面准备好的字符串数组，每当遇到一个操作数，则生成一个节点并入栈；每当遇到一个运算符，则将栈顶的两个子树出栈，分别作为这个运算符的左右子树，建立一个新的树，然后入栈。如此循环直到整个字符串数组的内容都遍历了，栈里剩下的那个就是该后缀表达式的表达树，返回根节点。

**1.6.3 中缀表达式转化为后缀表达式：**

这是本程序最复杂的一个算法点。建立一个ArrayList，存放输出结果，建立一个栈，用于存放运算符。遍历前面准备好的字符串数组：

每当遇到一个操作数，则直接输出。

若遇到左括号，则将其入栈；若遇到右括号，则将其和栈内最近左括号之间的所有运算符出栈并输出（根据逻辑，输出其间所有运算符是正确的，因为操作数被直接输出了，右括号表示一部分运算结束），然后再将左括号出栈。

若遇到运算符，则比较栈顶运算符和其的优先级，如果栈顶不是左括号，或栈顶优先级不小于该运算符，则出栈并输出，如此循环直到遇到左括号或优先级小于或栈为空，将该运算符入栈。

如此往复直到整个字符串数组都被遍历，依次出栈并输出栈内剩余的操作数，最终得到输出序列ArrayList，返回。

**1.6.4 中缀表达式建立表达树：**

先将表达式转化为后缀，然后直接调用后缀建立表达树的函数

**1.6.5 表达树的三种输出：**

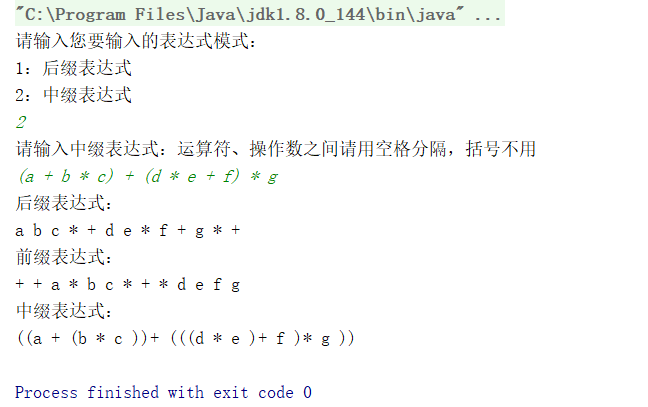
这个就很简单了，就是二叉树的前序中序和后序三种遍历，每种都是递归调用，注意输出顺序即可。

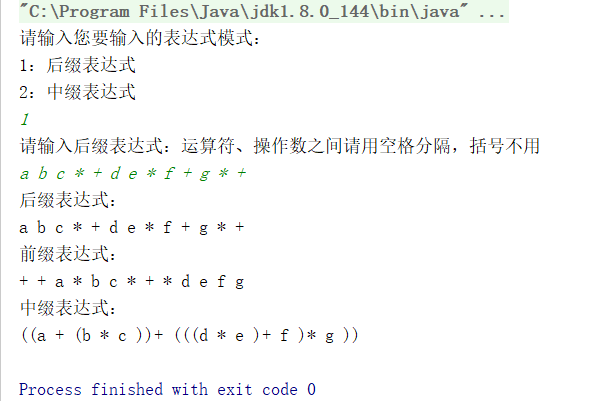
**1.7** 运行效果

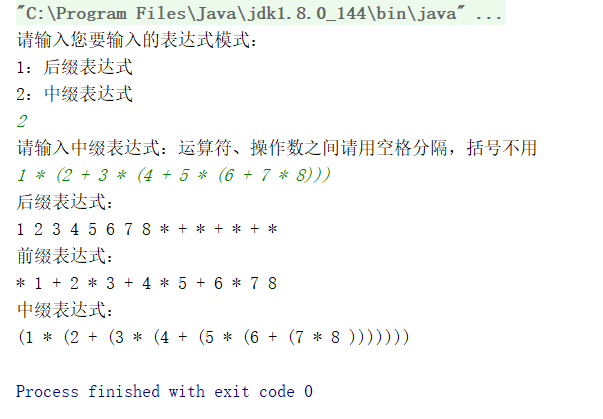
首先输出提示语让用户选择他的表达式模式，1表示后缀，2表示前缀。

然后输出提示语让用户按格式要求输入他的表达式，根据表达式建立表达树，生成ExpressionTree对象，然后调用此对象的三种表达式输出函数（也就是三种顺序的遍历），分别输出前缀、中缀、后缀表达式，完成后缀、中缀和三种表达式间的转换。

运行结果截图：







**1.8** 问题总结与分析

最大的难点在于中缀表达式转后缀表达式，算法不是很好想，需要对表达式运算的性质有较深的了解，需要用到堆栈。但Java的指针操作比C简单太多，也有相应的Stack类和库函数，所以并不是很困难。

还有一个难点是输入数据的处理，不过由于后缀表达式必须用空格隔开，我规定了输入格式，使得可以直接调用Java库函数，再次体现出Java的强大。

最终成功实现了题目要求。

结 论

达到了基本实验环节的要求和目的，证明了狗的成功率大概和地图大小成logarithmic关系。也成功完成了表达式的中缀、后缀表达式转化成二叉表达树，以及二叉表达树通过前中后序遍历来输出前中后缀表达式。其中中缀表达式一般先转化为后缀表达式，然后再转化成表达树。

参考文献

[1] Kathy Sierra. Head First Java，2007

附 录

核心代码1：

*/\*\* Get all the available directions.  
 \** ***@return*** *If no direction available, null.  
 \* Else, a array contains directions\*/***public int**[] getDirections() {  
 **int**[] directions = {1, 1, 1, 1};  
 **int** num = 0;  
 **if** (**\_map**[**\_positionX**][**\_positionY** + 1] == 0) {  
 directions[0] = 0;  
 num += 1;  
 }  
 **if** (**\_map**[**\_positionX**][**\_positionY** - 1] == 0) {  
 directions[1] = 0;  
 num += 1;  
 }  
 **if** (**\_map**[**\_positionX** - 1][**\_positionY**] == 0) {  
 directions[2] = 0;  
 num += 1;  
 }  
 **if** (**\_map**[**\_positionX** + 1][**\_positionY**] == 0) {  
 directions[3] = 0;  
 num += 1;  
 }  
 **if** (num == 0) {  
 **return null**;  
 } **else** {  
 **return** directions;  
 }  
}  
  
*/\*\** ***@return*** *Is the dog dead. \*/***public boolean** isDead() {  
 **return** getDirections() == **null**;  
}  
  
*/\*\** ***@return*** *Does the dog win. \*/***public boolean** isWin() {  
 **return \_positionX** == 0 || **\_positionX** == **\_size** - 1 || **\_positionY** == 0 || **\_positionY** == **\_size** - 1;  
}  
  
*/\*\* Make a legal move. \*/***public void** move() {  
 **\_pathLength** += 1;  
 **\_map**[**\_positionX**][**\_positionY**] = 1;  
 **int**[] directions = getDirections();  
 **int** direction;  
 **do** {  
 direction = (**int**)(Math.*random*() \* 4);  
 } **while** (directions[direction] == 1);  
 */\* For test, show the direction made by the dog. \*/  
 /\* System.out.println(direction); \*/* **switch** (direction) {  
 **case** 0: moveUp(); **break**;  
 **case** 1: moveDown(); **break**;  
 **case** 2: moveLeft(); **break**;  
 **case** 3: moveRight(); **break**;  
 **default**:  
 }  
 */\* For test, show the path of the dog. \*/  
 /\*System.out.println(\_positionX + " " + \_positionY);\*/*}

核心代码2：

*/\*\* Generate the tree by a given post-order string array.  
 \** ***@param items*** *The given string array.  
 \** ***@return*** *The root of the generated tree. \*/***public** ETNode postGenerate(String[] items) {  
 **if** (items == **null**) {  
 **return null**;  
 }  
 Stack s = **new** Stack<ETNode>();  
 **for** (**int** i = 0; i < items.**length**; i++) {  
 String item = items[i];  
 **if** (*isOperator*(item)) {  
 ETNode right = (ETNode)s.pop();  
 ETNode left = (ETNode)s.pop();  
 ETNode node = **new** ETNode(item, left, right);  
 s.push(node);  
 } **else** {  
 ETNode node = **new** ETNode(item, **null**, **null**);  
 s.push(node);  
 }  
 }  
 **return** (ETNode)s.pop();  
}

*/\*\* Convert an in-order array to a post-order array.  
 \** ***@param items*** *in-order array.  
 \** ***@return*** *post-order array after convert. \*/***public static** String[] inToPost(String[] items) {  
 ArrayList result = **new** ArrayList<String>();  
 Stack s = **new** Stack<String>();  
 **for** (**int** i = 0; i < items.**length**; i++) {  
 String item = items[i];  
 **if** (!*isOperator*(item)) {  
 **if** (item.charAt(0) == **'('**) {  
 **while** (item.charAt(0) == **'('**) {  
 s.push(**"("**);  
 item = item.substring(1);  
 }  
 result.add(item);  
 } **else if** (item.charAt(item.length() - 1) == **')'**) {  
 **int** count = 0;  
 **while** (item.charAt(item.length() - 1) == **')'**) {  
 count += 1;  
 item = item.substring(0, item.length() - 1);  
 }  
 result.add(item);  
 **while** (count > 0) {  
 **if** (!s.peek().equals(**"("**)) {  
 result.add(s.pop());  
 } **else** {  
 s.pop();  
 count -= 1;  
 }  
 }  
 } **else** {  
 result.add(item);  
 }  
 } **else** {  
 **if** (s.isEmpty()) {  
 s.push(item);  
 } **else** {  
 **while** (s.isEmpty() || *isNotLessThan*((String)s.peek(), item)) {  
 result.add(s.pop());  
 }  
 s.push(item);  
 }  
 }  
 }  
 **while** (!s.isEmpty()) {  
 result.add(s.pop());  
 }  
 String[] rs = **new** String[items.**length**];  
 **for** (**int** i = 0; i < items.**length**; i++) {  
 rs[i] = (String)result.remove(0);  
 }  
 **return** rs;  
}

教师评语评分

评语：

评分：

评阅人：

年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）