

数据结构课程设计报告

班 级：19G231

学 号：20231003496

学生姓名：张淋迩

日 期：2024年9月9日

目录

[【题目一】迷宫问题 1](#_Toc22041)

[一、课程设计题目与要求 1](#_Toc21984)

[二、需求分析 1](#_Toc16183)

[三、设计 2](#_Toc18976)

[3.1 设计思想 2](#_Toc4556)

[3.1.1数据结构设计 2](#_Toc2560)

[3.1.2算法设计 2](#_Toc23144)

[3.2 设计表示 3](#_Toc2586)

[3.2.1非递归程序 3](#_Toc29815)

[3.2.2递归程序 4](#_Toc29811)

[3.3 详细设计 5](#_Toc29942)

[3.3.1非递归程序 5](#_Toc30973)

[3.3.2递归程序 9](#_Toc1892)

[3.2.3拓展程序 11](#_Toc19241)

[四、调试分析 15](#_Toc14237)

[五、用户手册 16](#_Toc31974)

[六、测试数据及测试结果 16](#_Toc14157)

[七、源程序清单 20](#_Toc10948)

[【题目二】二叉树的应用——表达式求值 32](#_Toc25612)

[一、课程设计题目与要求 32](#_Toc18970)

[二、需求分析 32](#_Toc15216)

[三、设计 33](#_Toc12632)

[3.1 设计思想 33](#_Toc5270)

[3.1.1数据结构设计 33](#_Toc29677)

[3.1.2算法设计 33](#_Toc29413)

[3.2 设计表示 34](#_Toc13935)

[3.3 详细设计 35](#_Toc9258)

[3.3.1 基本功能程序 35](#_Toc1251)

[3.3.2 拓展程序 43](#_Toc11312)

[四、 调试分析 44](#_Toc17489)

[五、用户手册 45](#_Toc21125)

[六、测试数据及测试结果 45](#_Toc25108)

[七、源程序清单 48](#_Toc2835)

# 【题目一】迷宫问题

## 一、课程设计题目与要求

1.基本需求

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。

2.基本要求

（1）编写一个基于堆栈求解迷宫问题的非递归程序。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）编写一个求解迷宫的递归程序。

（3）使用几组数据测试，且数据的规模由小变大，即网格越来越小，障碍越来越复杂。

3.扩展要求

实现迷宫问题的可视化界面。用鼠标点击即可一步步走出迷宫，或者最终弹出没有通路的消息框。

## 二、需求分析

1.基本功能需求

（1）输入迷宫的长和宽、迷宫的入口和出口、具体的迷宫的墙和通路

（2）判断迷宫是否有解、输出迷宫具体路径

2.用户界面需求

（1）迷宫有解时画出迷宫，并通过鼠标点击一步步展示迷宫通路路径

（2）可通过按键直接清空数据，重新设计迷宫并求解

## 三、设计

### 3.1 设计思想

#### 3.1.1数据结构设计

（1）栈（Stack）：根据题目要求，设计使用栈作为正确路径的存储结构。因为栈具有先进后出特点，所以此处选择先将出口坐标放入栈中，从出口坐标向入口坐标依次寻找通路并放入栈中，最终得以通过依次拿取顶端得到从入口到出口正确坐标次序。

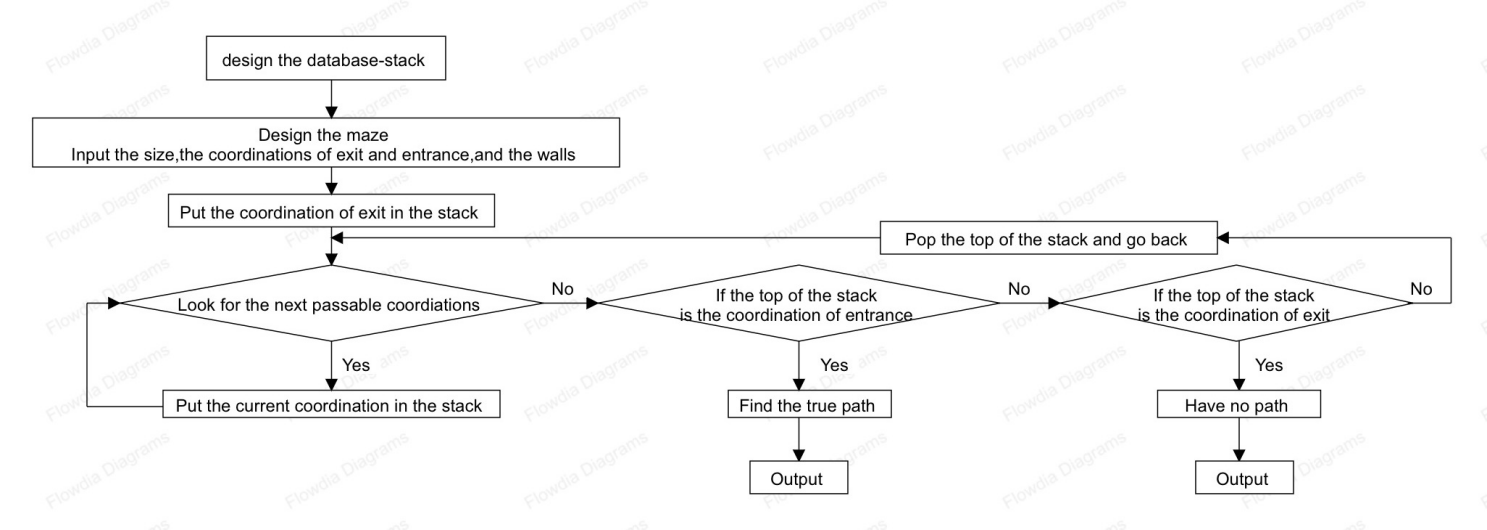
（2）二维数组：根据迷宫摆放具有行和列的特点，设计使用二维数组作为迷宫形态的存储结构。其中第一维中的每个元素代表迷宫的每一行，第二维中的每个元素代表迷宫的每一个单元格。

（3）列表：根据需要判断单元格坐标是否已经行经路过的需求特点，设计使用列表作为行经路过的所有坐标的存储结构。

#### 3.1.2算法设计

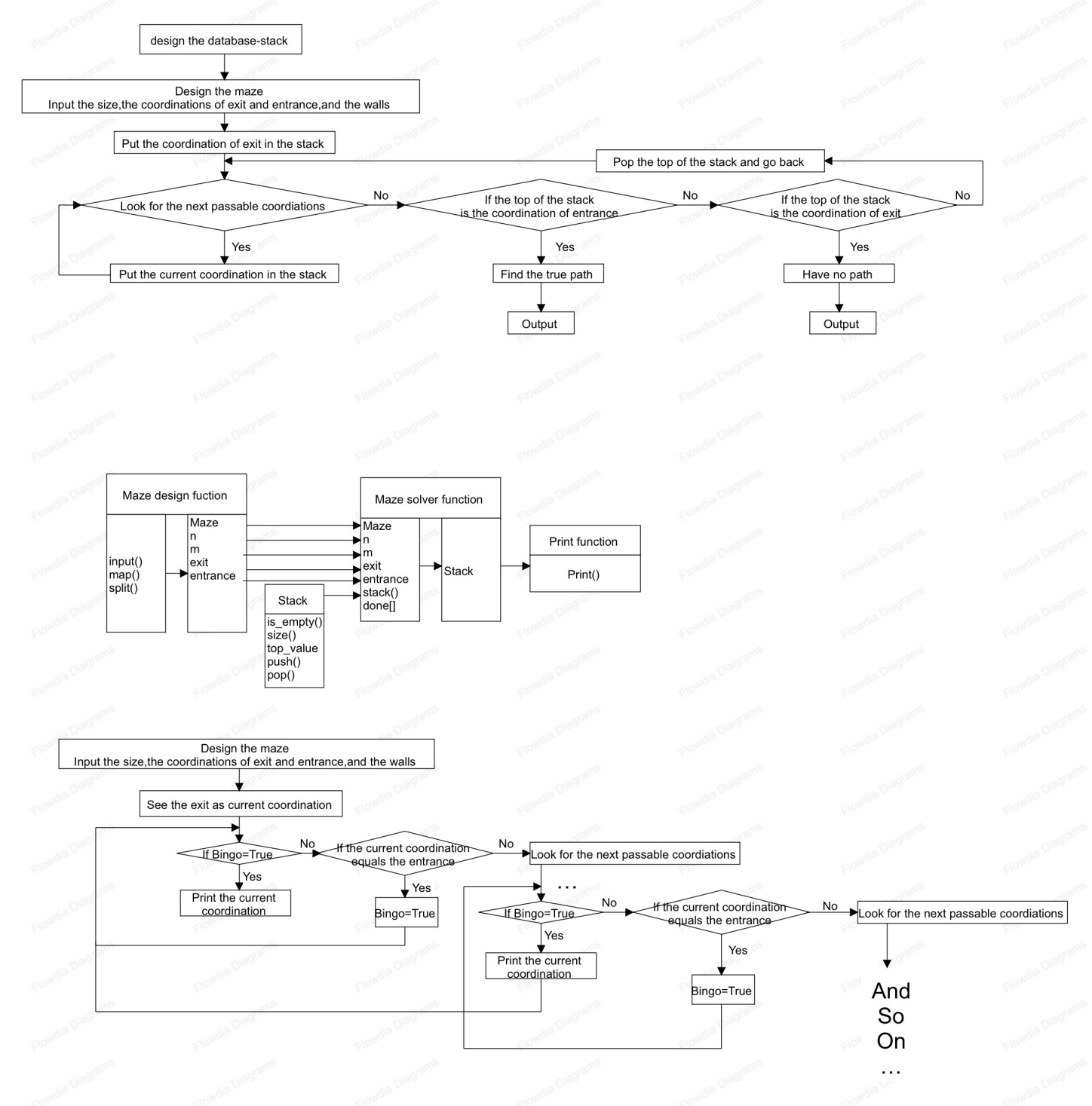
（1）非递归程序：先将出口坐标放入栈中，栈的顶端作为当前坐标。整体用while循环结构，通过对栈的顶端元素的不断调整对当前坐标进行不断更新。循环结构内部，先将当前坐标放入列表中表示已经过，后通过if选择结构找到当前坐标的可通行的所有坐标，如果可通行坐标存在，则将其可通行坐标放入栈中作为新的顶端即当前坐标进入循环再次判断，直到当前坐标为入口坐标时，循环停止，表示已找到正确路径；如果可通行坐标不存在，则表示当前坐标非正确路径，将其从栈中去除，返回上一个顶端即上一步坐标，重新判断，当不停返回直到当前坐标回到出口坐标时，循环停止，表示不存在正确路径。

非递归程序设计思想流程图如图1.1所示。

图1.1 迷宫问题非递归程序思想流程图

（2）递归程序：因为不断重复对下一坐标的筛选过程，所以将该筛选过程作为递归函数，参数为当前坐标。因为递归程序中输出的先后顺序是从内层到外层，所以将出口坐标作为最外层，找到入口坐标为最内层。当当前坐标为入口坐标时，表示找到正确路径，返回True，作为base case；当当前坐标不为入口坐标时，则通过if选择结构找到当前坐标的可通行的所有坐标，对每一种可能情况坐标再代入该函数进行判断，通过一个参数，对每层函数的返回值进行传递，作为general case。直到最内层到达入口坐标，返回值为True，层层传递影响，从内层向外层进行输出。

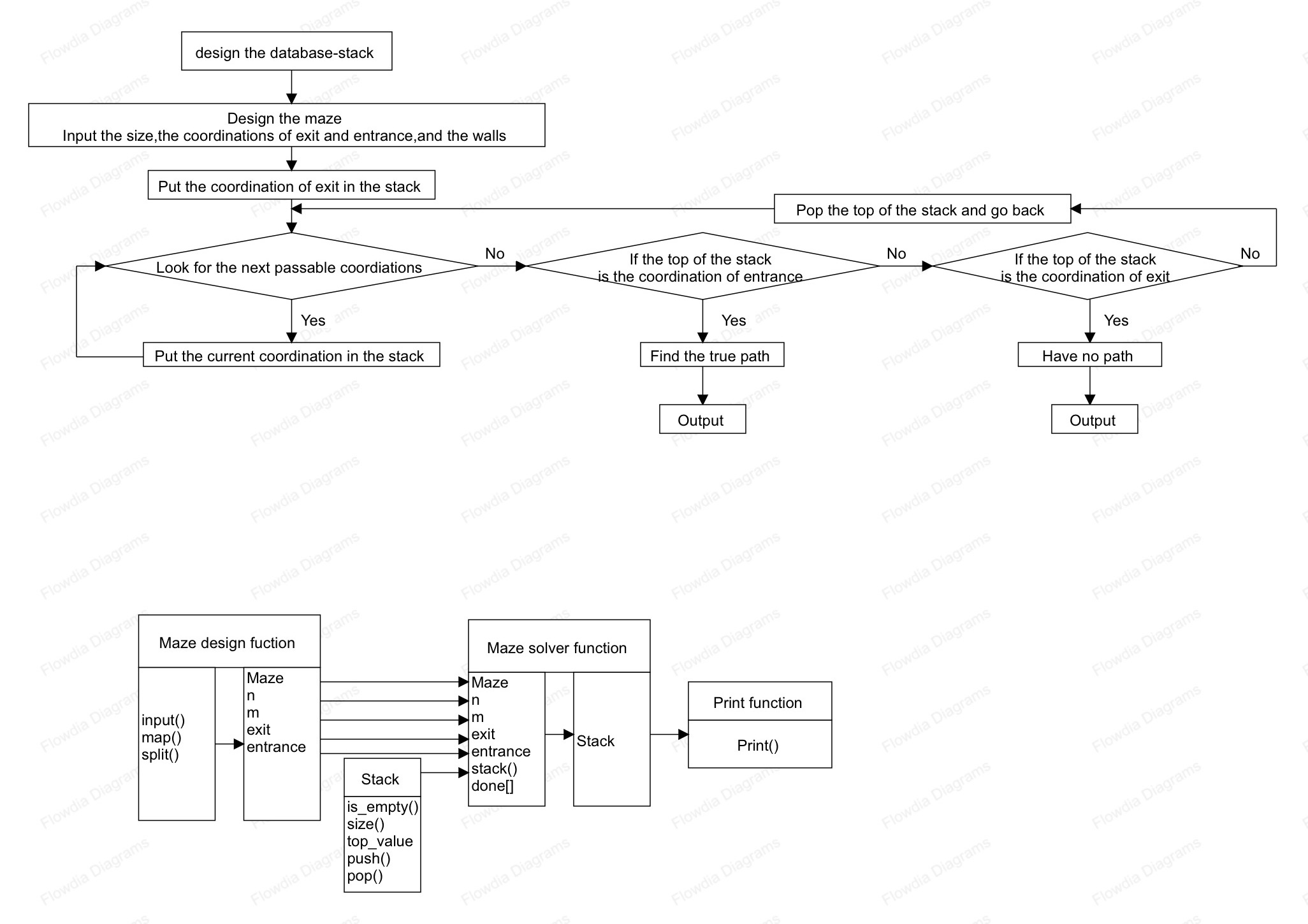
递归程序设计思想流程图如图1.2所示。

图1.2 迷宫问题递归程序思想流程图

### 3.2 设计表示

#### 3.2.1非递归程序

主要函数的调用关系如图1.3所示。

图1.3 非递归程序主要函数调用关系图

（1）Stack 模块：is\_empty()，返回布尔值，表示栈是否为空；size()，返回栈中的元素数量；top\_value()，返回栈顶元素，如果栈为空则抛出异常；push(value)， 将一个值放入栈顶；pop()，删除栈顶元素并返回，如果栈为空则抛出异常。

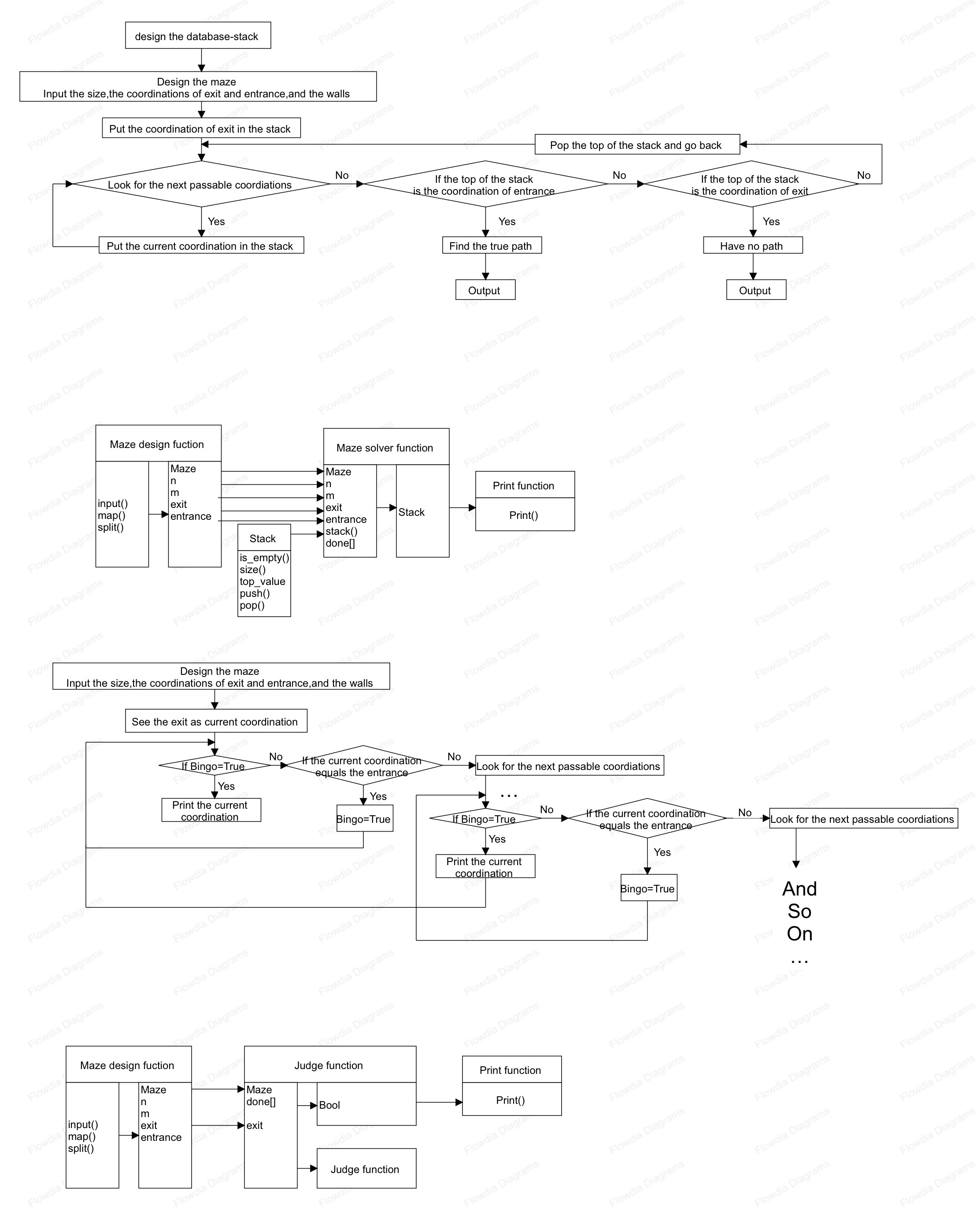
（2）迷宫设计模块：input()，接收用户的输入；map()，将用户输入内容进行分配；split()，将用户输入内容进行分割；n和m，整数存储用户输入的迷宫的长和宽；exit和entrance，列表存储用户输入的迷宫出口和入口坐标，元素为整数；Maze，二维数组存储用户输入的迷宫具体内容。

（3）迷宫解决模块：st，栈类存储正确路径；done，列表存储已访问的坐标点；

next\_coordinations，临时栈存储下一步可能的坐标。

#### 3.2.2递归程序

主要函数的调用关系如图1.4所示。

图1.4 递归程序主要函数调用关系图

（1）迷宫设计模块：input()，接收用户的输入；map()，将用户输入内容进行分配；split()，将用户输入内容进行分割；n和m，整数存储用户输入的迷宫的长和宽；exit和entrance，列表存储用户输入的迷宫出口和入口坐标，元素为整数；Maze，二维数组存储用户输入的迷宫具体内容。

（2）坐标判断模块：done，列表存储已访问的坐标点；if-else，选择语句找出所有可能通路坐标；print()，对正确路径坐标进行输出；返回布尔值。

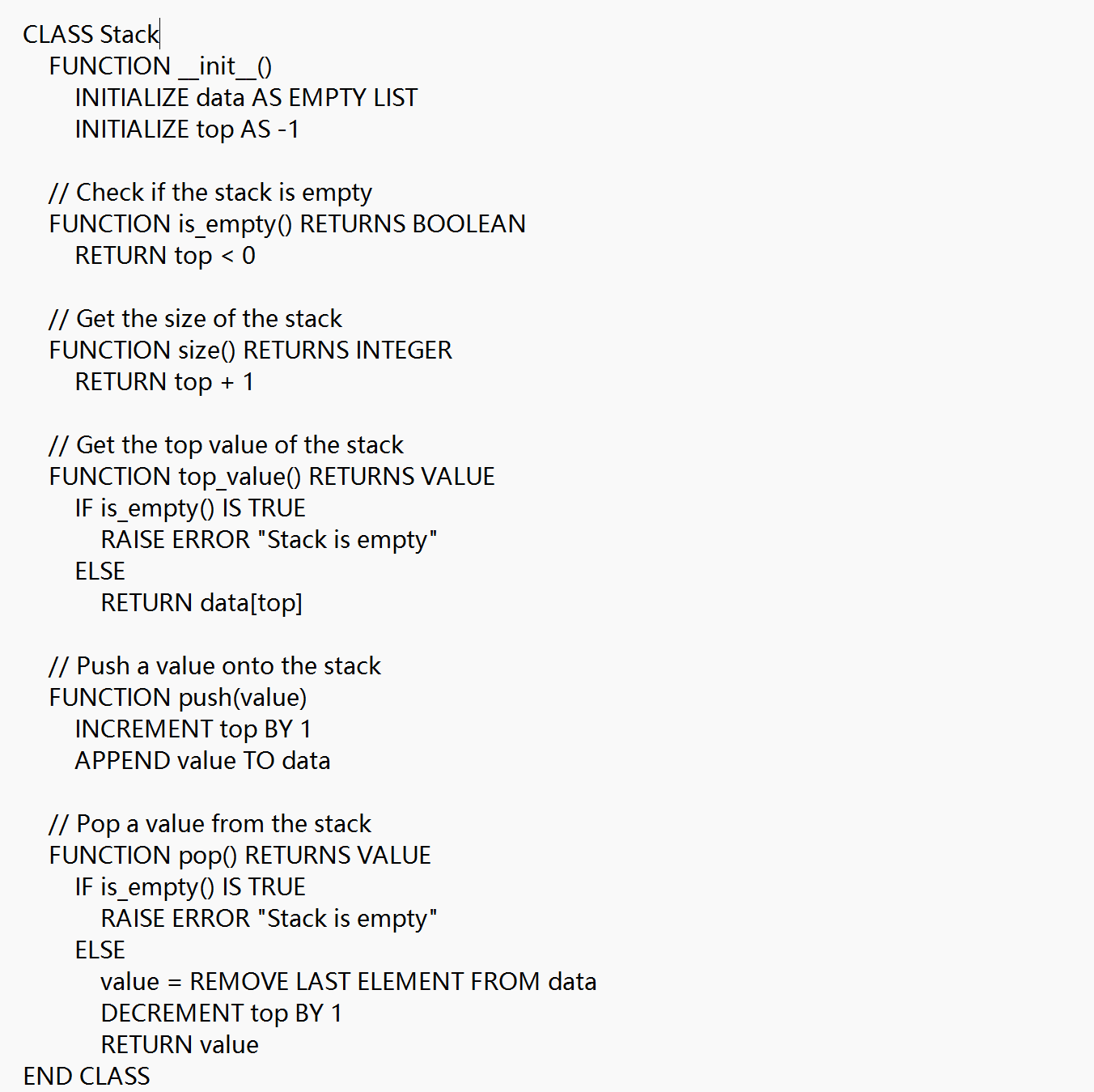
### 3.3 详细设计

#### 3.3.1非递归程序

1. 栈类设计

一个列表data，用于存储栈中的元素，一个整数top，表示栈顶元素的索引，初始化为-1。Is\_empty()函数用来判断栈是否为空，即判断top是否小于0，若小于0，则表示栈顶端序列号不存在，即为空，否则不为空，返回布尔值。Size()函数用来返回栈内元素数量，top表示最顶端元素序列号，即列表data最后一个元素的序列号，其加上1就是列表长度，也可以直接返回列表data的长度。Top\_value()函数用来返回当前顶端元素，即返回列表data的最后一个元素。Push(value)函数用来将value添加到栈中成为新的顶端，即将value添加到列表data的末端，并将表示顶端元素序列号的top值加一。Pop()函数用来将栈顶端元素去除并返回，即将列表data的末尾元素删除，并将表示顶端元素序列号的top值减一。

伪代码如图1.5所示。

图1.5 栈类设计伪代码

（2）迷宫输入设计

在程序之初，打印提示让用户选择手动输入迷宫或者自动生成迷宫。

若用户输入1为手动输入，打印提示，提醒用户输入迷宫的大小，读取用户输入内容，存储迷宫的长和宽为n和m。打印提示，提醒用户输入迷宫的通路和墙，初始化二维数组maze作为迷宫存储空间，遍历前面读取的行数，对每行内容进行输入和读取，并临时存储在列表row中，添加进二维数组maze。打印提示，提醒用户输入迷宫的出口坐标，读取用户输入内容，存储出口坐标的横纵坐标为a和b，合并成为列表exit。打印提示，提醒用户输入迷宫的入口坐标，读取用户输入内容，存储出口坐标的横纵坐标为a和b，合并成为列表entr。

若用户输入2为自动输入，初始化二维数组maze作为迷宫存储空间，使用random库，随机生成2至10的整数为迷宫的长和宽，存为n和m，通过嵌套循环，先遍历行数，每行内先初始化列表row，再在每行内遍历列数，每次在0和1中按7：3的比例随机生成一个数，添加到列表row中，列遍历结束后将row加入maze。出口默认为坐标[n-1,m-1]，入口默认为[0,0]。在赋值结束后对迷宫和出入口进行输出。

若用户输入其他则报错。

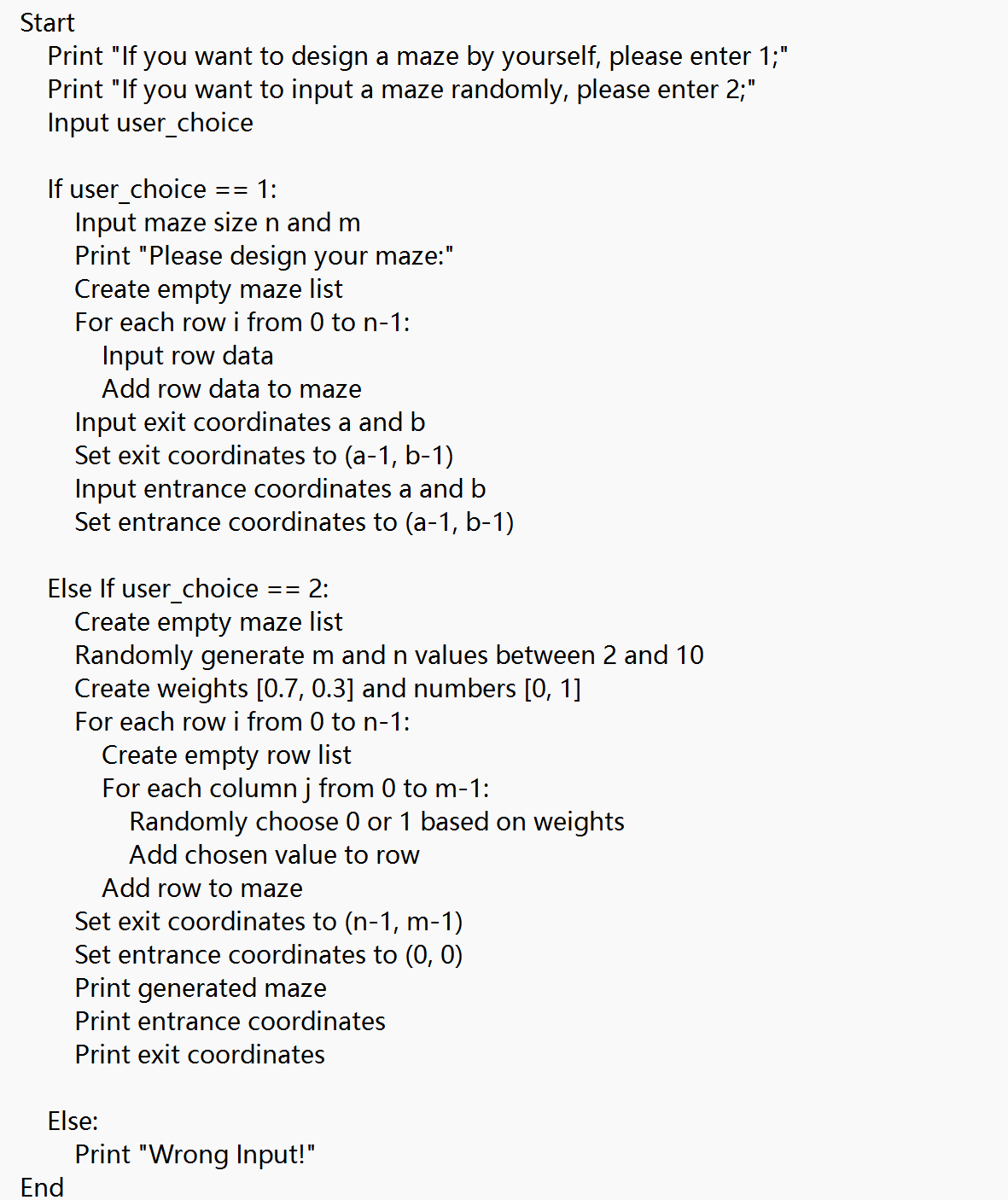
 伪代码如图1.6所示。

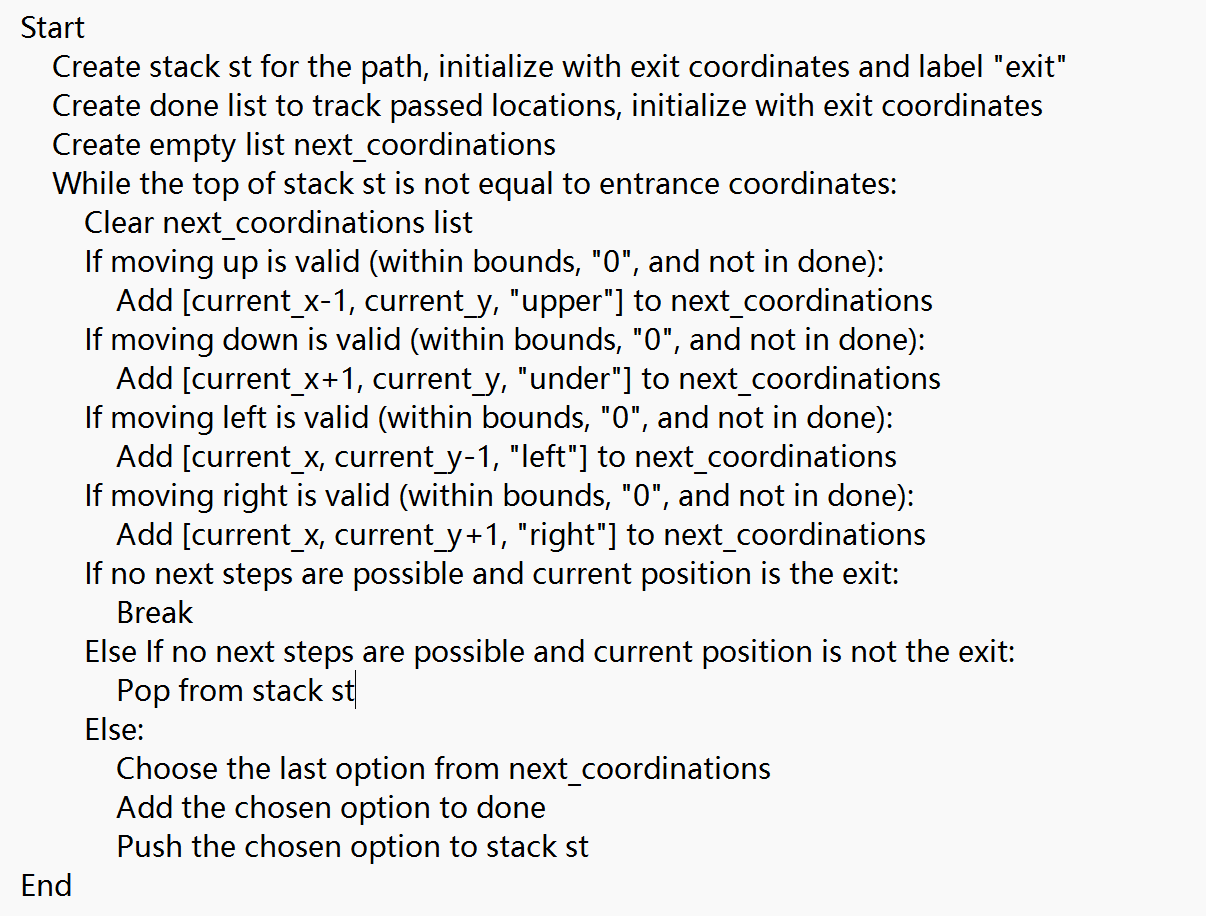
图1.6 迷宫输入设计伪代码

1. 迷宫寻解设计

一个栈st，用于存储正确路径坐标，一个列表done，表示已经经过的坐标，防止寻找重复。栈st的顶端元素则表示当前走到的坐标。

从出口往入口找，则先将出口坐标exit放入栈st和列表done。当前坐标下一个可以到达的坐标分别是其上下左右四个坐标，要防止往回走，确保列表done中没有，还要防止超出迷宫大小，与迷宫长和宽比较是否较小，最终符合要求的就是当前坐标的下一步可以到达的坐标，为方便后续输出时输出行径方向，将此时所查找的方向加入表示坐标的列表中，因为存在岔路口，符合要求的点可能不止一个，则将这些列表又存在列表next\_coordinations中。如果该列表不为空，则取出列表其一坐标作为下一步进行尝试，将该坐标加入栈st和列表st，作为新的栈顶端进行分析；如果列表为空，则表示该坐标已无路可走，该坐标不是正确路径坐标，将该坐标从栈顶端去除，返回上一个栈顶端重新分析。当不断的返回，最终返回至出口坐标时，即在前面的寻找，对栈加减元素后，顶端元素依旧是出口坐标，则表示没有正确路径。因为对每个坐标下一个坐标寻找的分析过程相同，则用while循环串联，当寻找到入口，即栈的顶端元素为入口坐标时结果循环，或当发现没有正确路径时直接退出循环。

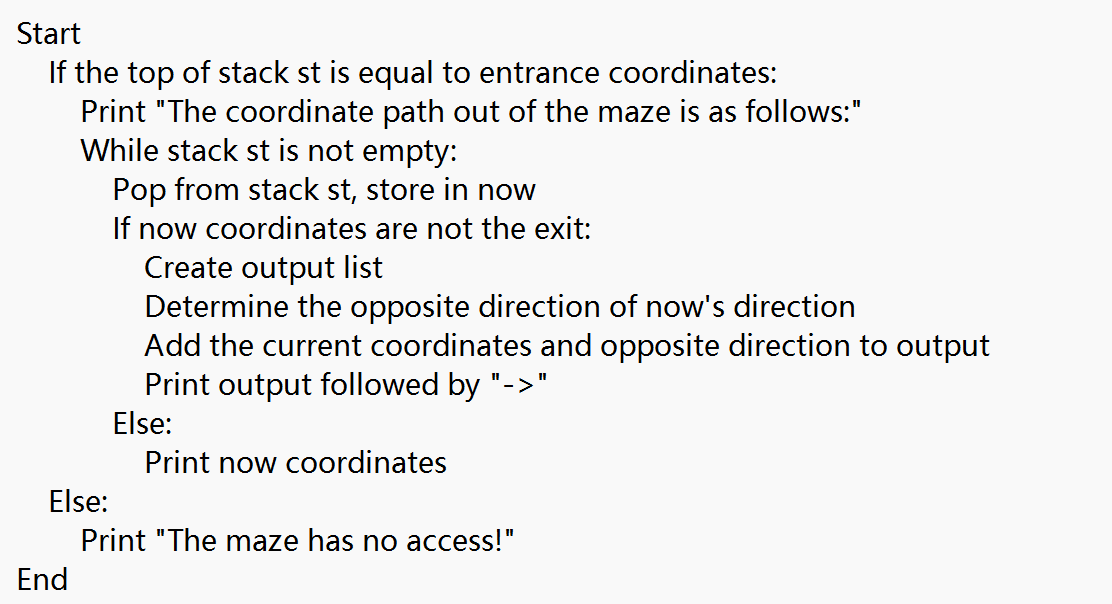
伪代码如图1.7所示。

图1.7 迷宫寻解设计伪代码

1. 输出设计

最终根据st顶端元素是否是入口坐标来判断输出。如果顶端元素是入口坐标，则表示迷宫有解，将栈st从顶端到底部进行输出就是正确路径的坐标输出，但根据题目要求，要满足[i,j,d]的输出格式，则要用pop删除并返回当前顶端元素列表的前两个元素作为当前坐标，即[i,j]，再获取第三个元素进行判断后取相反方向，添加到当前坐标列表中，作为下一坐标的方向，即d。

伪代码如图1.8所示。

图1.8 输出设计伪代码

#### 3.3.2递归程序

1. 迷宫输入设计

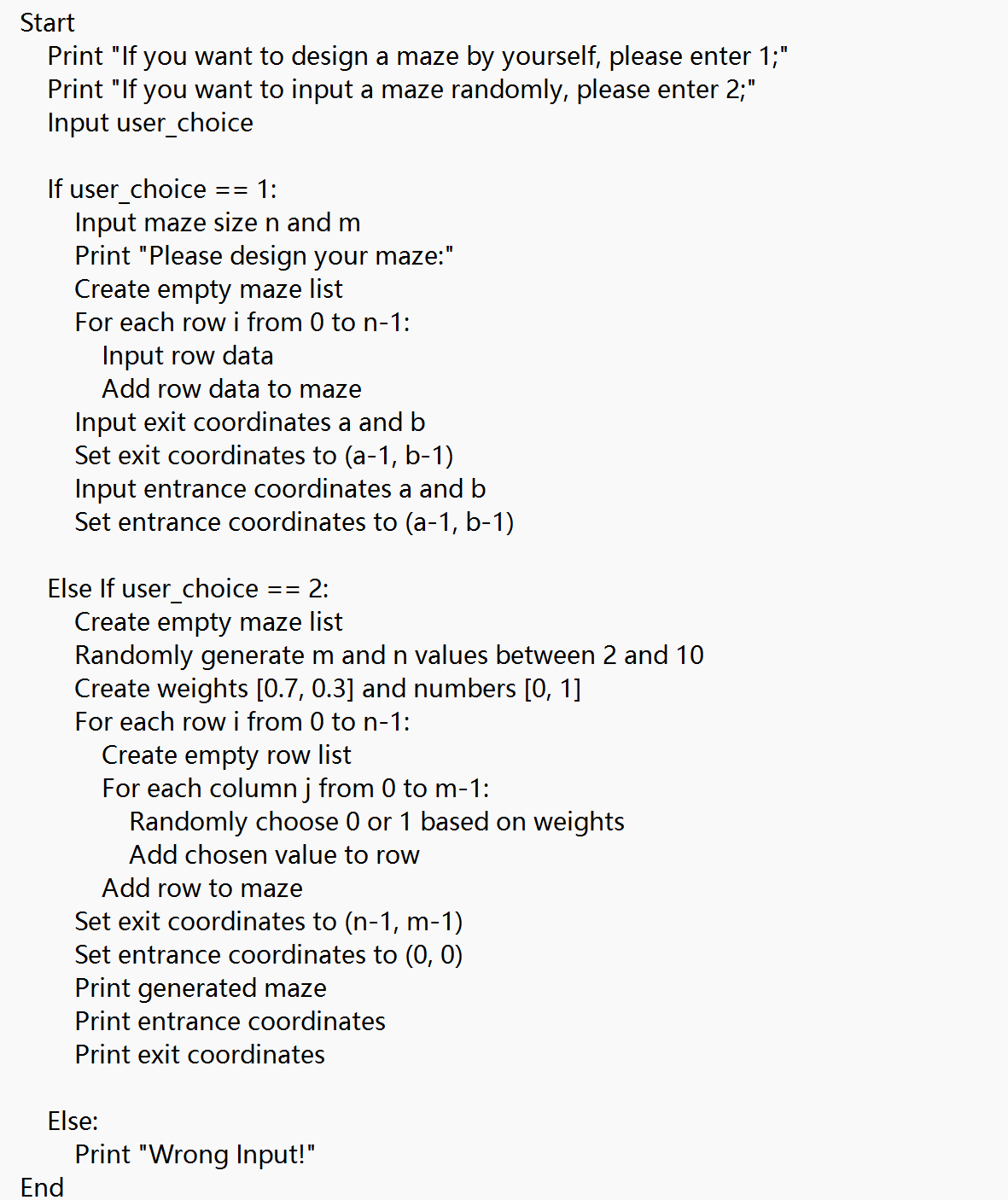
在程序之初，打印提示让用户选择手动输入迷宫或者自动生成迷宫。

若用户输入1为手动输入，打印提示，提醒用户输入迷宫的大小，读取用户输入内容，存储迷宫的长和宽为n和m。打印提示，提醒用户输入迷宫的通路和墙，初始化二维数组maze作为迷宫存储空间，遍历前面读取的行数，对每行内容进行输入和读取，并临时存储在列表row中，添加进二维数组maze。打印提示，提醒用户输入迷宫的出口坐标，读取用户输入内容，存储出口坐标的横纵坐标为a和b，合并成为列表exit。打印提示，提醒用户输入迷宫的入口坐标，读取用户输入内容，存储出口坐标的横纵坐标为a和b，合并成为列表entr。

若用户输入2为自动输入，初始化二维数组maze作为迷宫存储空间，使用random库，随机生成2至10的整数为迷宫的长和宽，存为n和m，通过嵌套循环，先遍历行数，每行内先初始化列表row，再在每行内遍历列数，每次在0和1中按7：3的比例随机生成一个数，添加到列表row中，列遍历结束后将row加入maze。出口默认为坐标[n-1,m-1]，入口默认为[0,0]。在赋值结束后对迷宫和出入口进行输出。

若用户输入其他则报错。

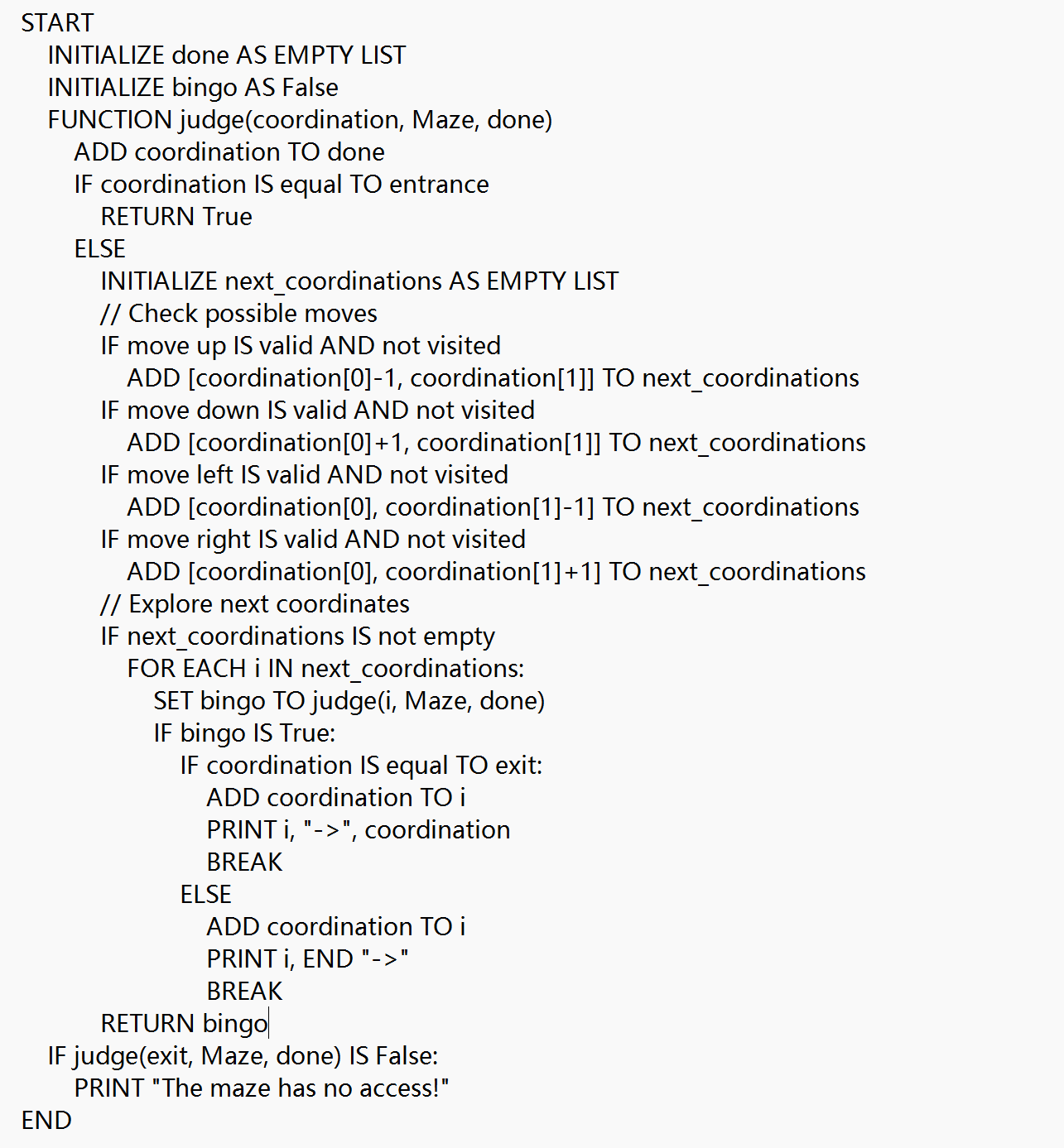
伪代码如图1.9所示。

图1.9 迷宫输入设计伪代码

（2）迷宫求解并输出设计

一个列表done，表示已经经过的坐标，防止寻找重复，一个布尔值Bingo，表示是否确定为通路一部分，用来判断是否输出。设计一个函数，要用到的参数是当前坐标coordination，具体迷宫Maze和列表done，函数的递归结束信号是当前坐标为入口坐标，返回值是True，否则，寻找当前坐标的下一坐标全部可能情况，保存在一个临时列表next\_coordinations中，通过for循环遍历该列表元素，将每个元素都回代入该函数，将下一级的返回值赋值给Bingo，最终返回Bingo。相当于每层的列表next\_coordinations的每个元素代表一条路径，只有能走到入口的路径返回值是True，又通过层层包含关系传递个Bingo，最后将呈现正确通路的Bingo值全部为True。当Bingo为True时输出coordination就是正确路径的坐标顺序，但要满足题目条件，在每层函数中，coordination是靠近出口的坐标，i是靠近入口的坐标，所以当Bingo为True时，是将coordination添加到i后再进行输出。因为只要有一条正确通路，最终函数返回都会是True，如果在函数外部，函数最终返回值为False，则代表这个迷宫没有正确通路。

伪代码如图1.10所示。

图1.10 迷宫求解并输出设计伪代码

#### 3.2.3拓展程序

（1）库调用

本程序通过PyQt5库对迷宫寻解的输入输出进行可视化。PyQt5.QtGui库中的QFont用于设置和管理文本的字体；QColor表示颜色，用于绘制图形和设置控件的颜色；QBrush用于填充图形的颜色；QPen定义绘图时颜色、线宽、线型等；QPainter，用于在控件或图形项上绘制图形和文本。PyQt5.QtWidgets库中的QApplication程序的主类，管理应用的控制流和主要设置；QMainWindow，应用的主窗口类，提供了一个主窗口的框架；QPushButton，按钮控件，可以触发点击事件；QLineEdit，单行文本编辑框，用于接受用户的输入；QLabel，标签控件，用于显示文本或图像；QTextEdit，多行文本编辑框，用于显示和编辑多行文本；QGraphicsTextItem，在 QGraphicsScene 中显示文本的图形项。

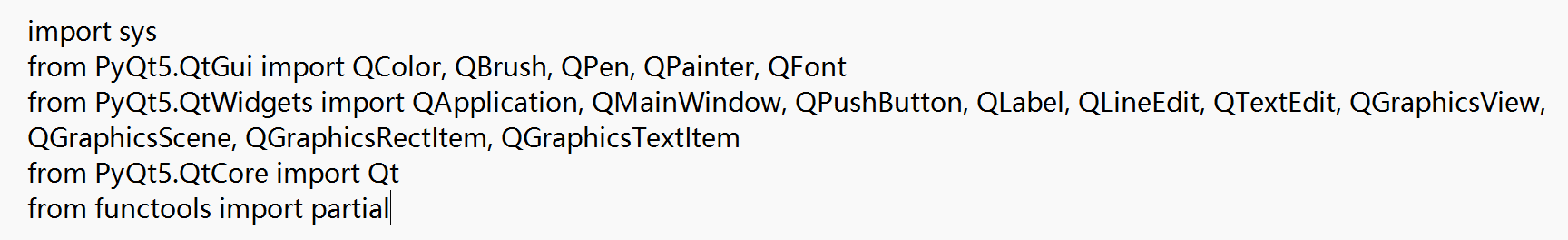
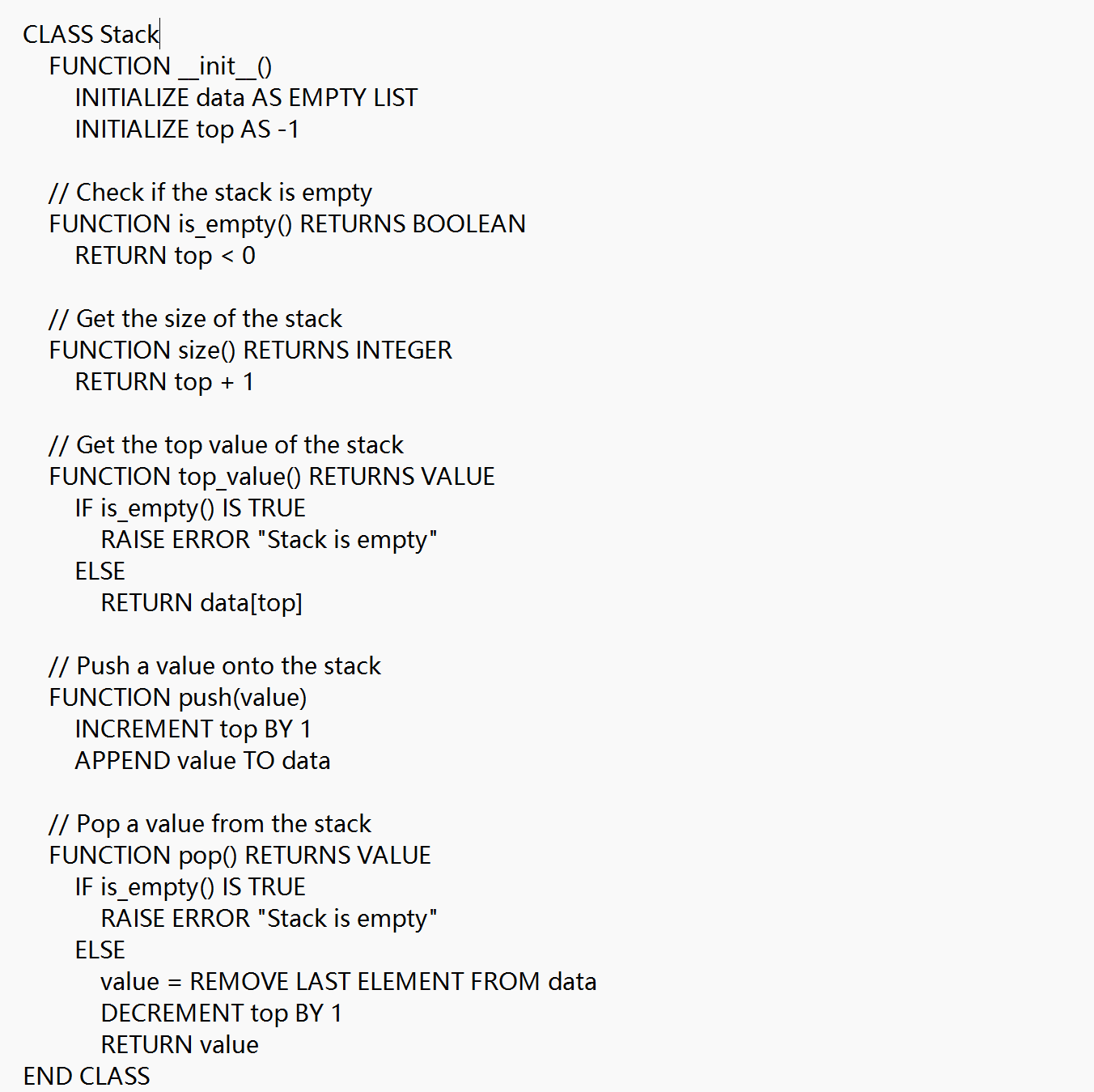
代码如图1.11所示。

图1.11 拓展程序库调用代码

（2）栈类设计

一个列表data，用于存储栈中的元素，一个整数top，表示栈顶元素的索引，初始化为-1。Is\_empty()函数用来判断栈是否为空，即判断top是否小于0，若小于0，则表示栈顶端序列号不存在，即为空，否则不为空，返回布尔值。Size()函数用来返回栈内元素数量，top表示最顶端元素序列号，即列表data最后一个元素的序列号，其加上1就是列表长度，也可以直接返回列表data的长度。Top\_value()函数用来返回当前顶端元素，即返回列表data的最后一个元素。Push(value)函数用来将value添加到栈中成为新的顶端，即将value添加到列表data的末端，并将表示顶端元素序列号的top值加一。Pop()函数用来将栈顶端元素去除并返回，即将列表data的末尾元素删除，并将表示顶端元素序列号的top值减一。

伪代码如图1.12所示。

图1.12 栈类设计伪代码

（3）界面类设计

初始化类来展示初始界面，将窗口标题设置为“Maze Solver”，窗口大小设置成1000mm\*1500mm，设置创建并布置界面上的各种控件，标签 (QLabel) 提供说明文字，行编辑框 (QLineEdit) 和文本编辑框 (QTextEdit) 供用户输入迷宫大小、入口、出口以及迷宫矩阵，按钮 (QPushButton) 用于触发solve函数，over函数和random函数，图形视图 (QGraphicsView) 和场景 (QGraphicsScene) 用于绘制迷宫和路径。

初始界面伪代码如图1.13所示。

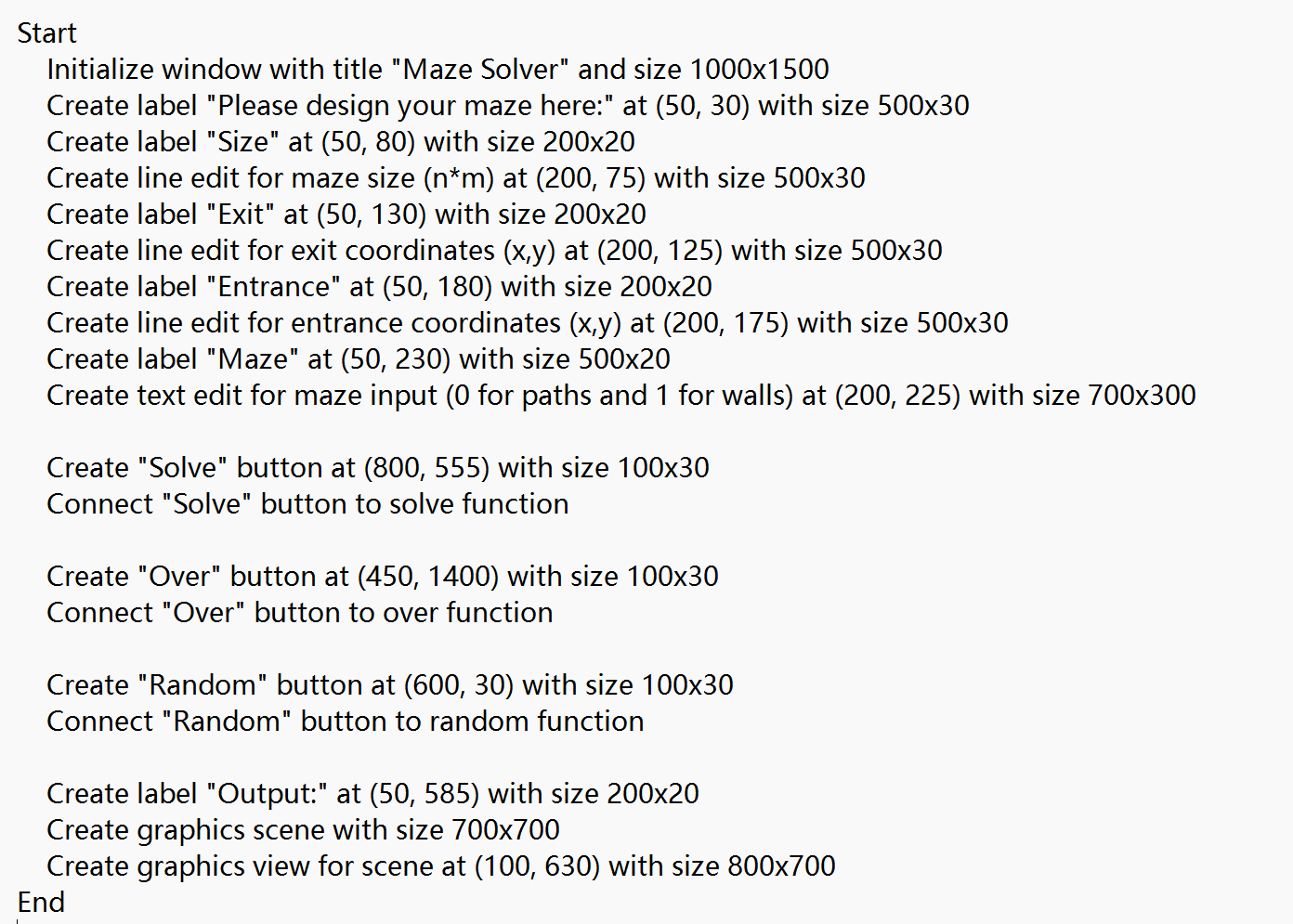
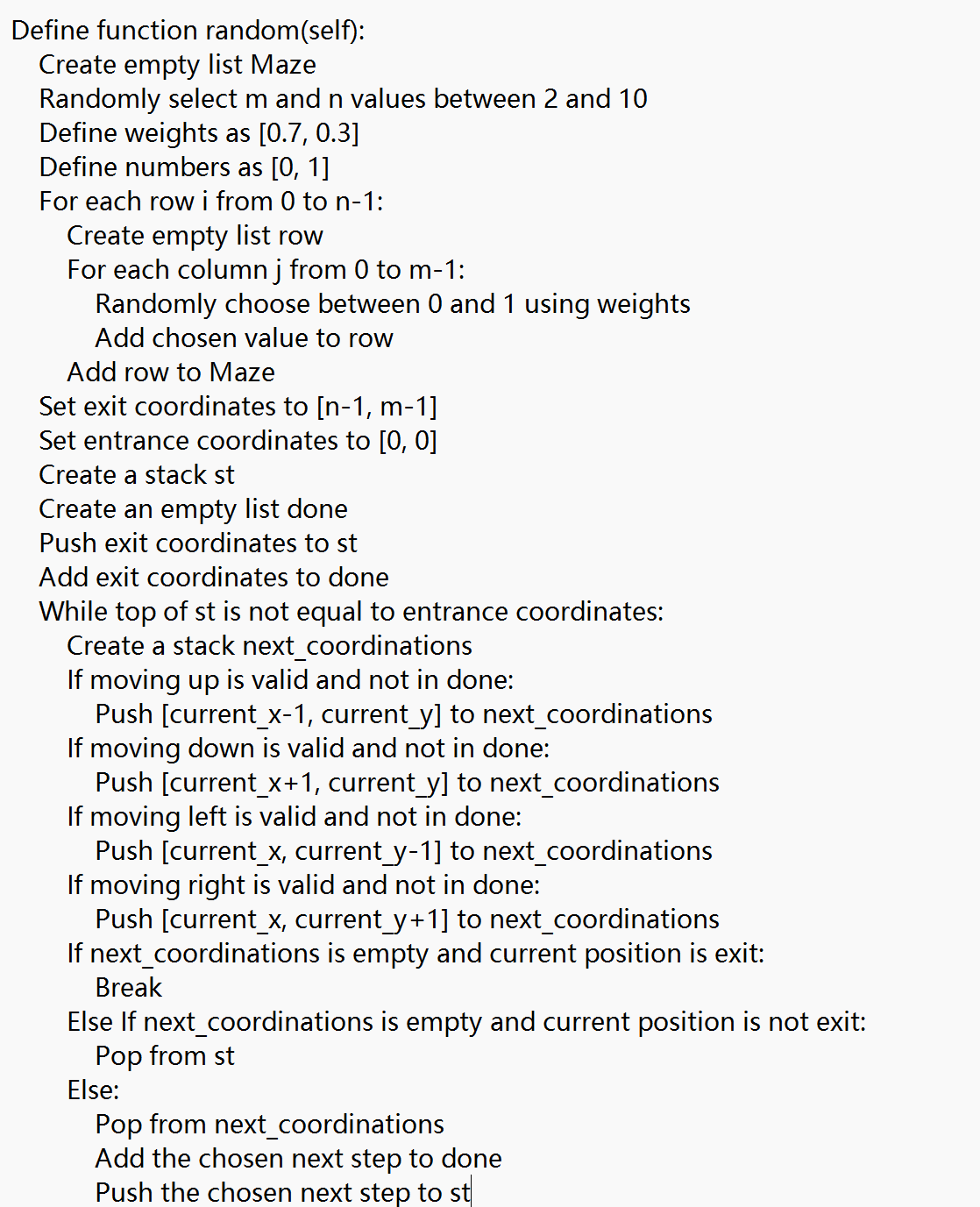


图1.13 界面类初始化设计伪代码

Random函数中，先随机生成迷宫大小和迷宫内容，使用非递归算法得出该迷宫是否有解，如果找到路径，则调用 access() 方法绘制路径，否则调用 noaccess() 方法提示无解，用参数Hello传递是否有解信号。

界面类随机生成迷宫求解伪代码如图1.14所示。

图1.14 界面类随机生成迷宫求解设计伪代码

当使用者在文本框中进行输入后点击Solve函数对迷宫求解。Solve函数中，先用text()函数得到用户所输入的内容，再将用户输入内容转化为适当的格式，使用非递归算法得出该迷宫是否有解，如果找到路径，则调用 access() 方法绘制路径，否则调用 noaccess() 方法提示无解，用参数Hello传递是否有解信号。

界面类迷宫求解设计伪代码如图1.15所示。

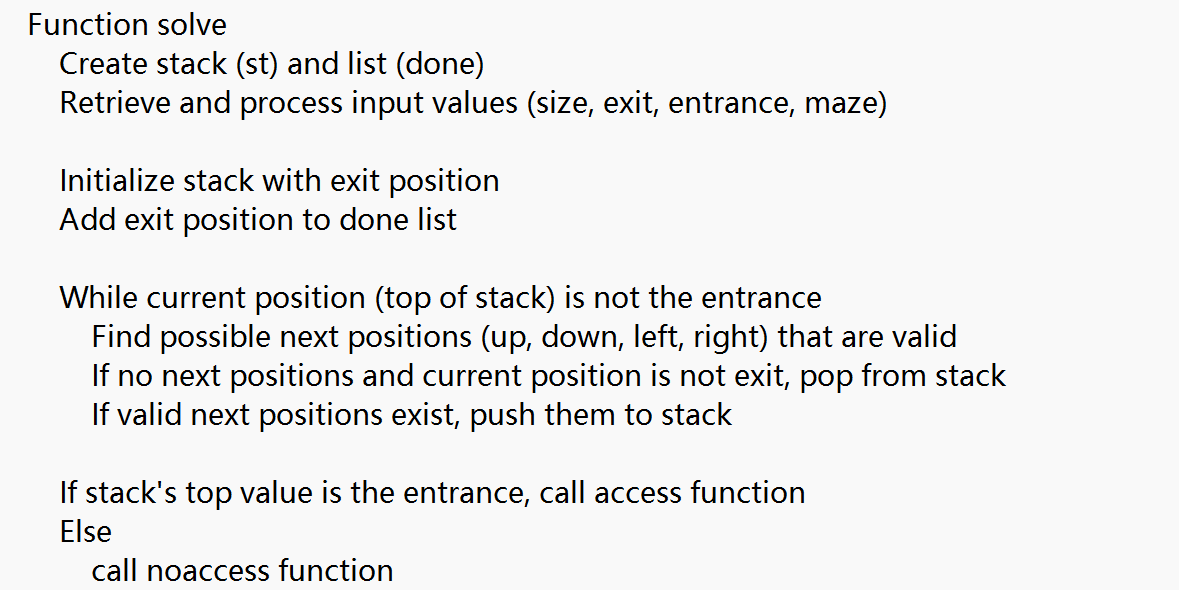


图1.15 界面类迷宫求解设计伪代码

Access函数中，先通过drawMaze()函数画出迷宫，根据画布大小和迷宫长宽计算合适的方块大小，将表示墙的方块涂成黑色，其余为白色，设置按钮 (QPushButton) 用于触发next()函数，每点击一次，当表示正确路径的栈st不为空时，将顶端元素删除并引用drawPath()函数将对应坐标的方块涂成红色，若为为空，则在画布末尾输出“Out of Maze!”表示已成功走出迷宫。

界面类通路可视化输出设计伪代码如图1.16所示。

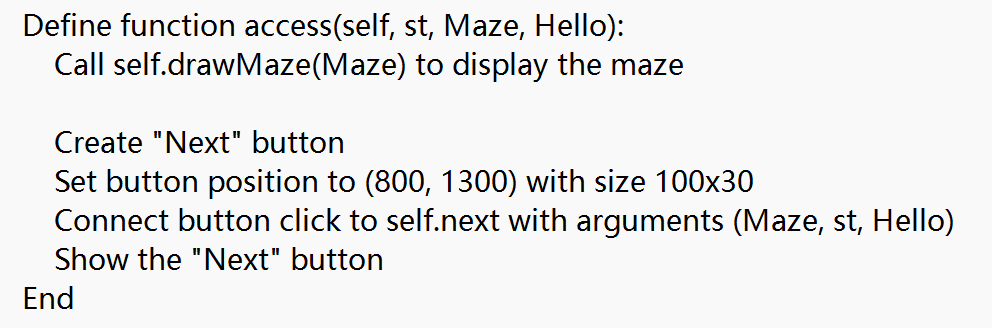


图1.16 界面类通路可视化输出设计伪代码

Noaccess函数中，先通过drawMaze()函数画出迷宫，根据画布大小和迷宫长宽计算合适的方块大小，将表示墙的方块涂成黑色，其余为白色，设置按钮 (QPushButton) 用于触发next()函数，在画布末尾输出“The maze has no access！”。

界面类无通路输出设计伪代码如图1.17所示。

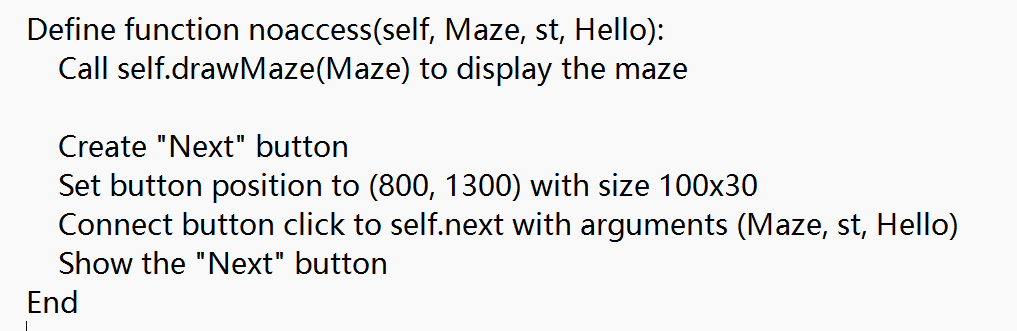
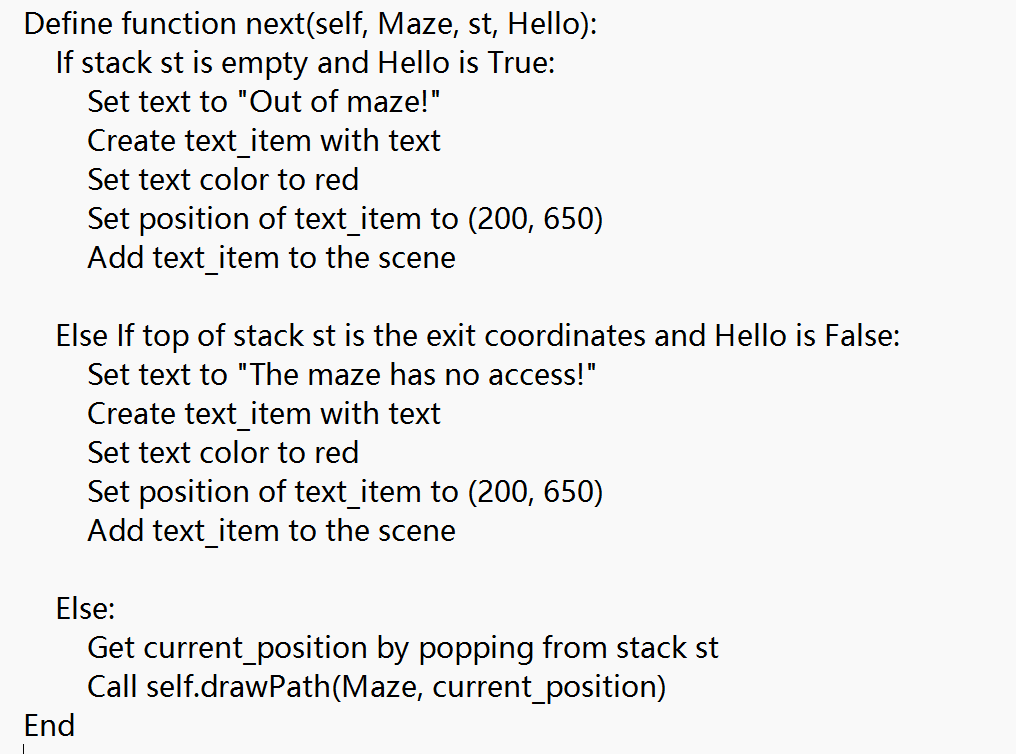


图1.17 界面类无通路输出设计伪代码

Next函数中，根据传入的Hello参数判断是否有解，当Hello为False时表示无解，直接在画布末尾打印“The maze has no access！”。当Hello为True时表示有解，在有解情况下只要当前坐标没到出口坐标，就将下一路径的方块涂成红色，为出口坐标时就在画布末尾打印“Out of Maze!”表示已成功走出迷宫。

界面类触发移动设计伪代码如图1.18所示。

图1.18 界面类出发移动设计伪代码

Over函数中，清除当前画布内容和输入内容，重新启用求解按钮，使用户可以再次设计迷宫并求解。

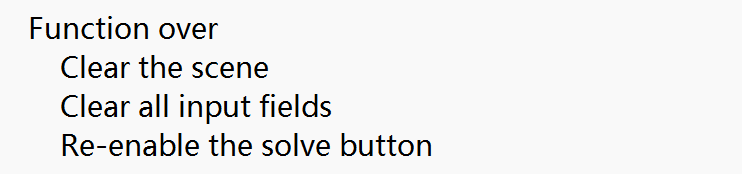
界面类清屏设计伪代码如图1.19所示。

图1.19 界面类清屏设计伪代码

## 四、调试分析

（1）过程中的困难与解决：非递归程序在最开始测试岔路口干扰情况的迷宫时，程序的输出结果总是包含尝试的错误路径的坐标。经过调试，发现是在每次循环时，next\_coordinations列表都还保留之前未走过的可通行坐标，所以在走到岔路口时，若先走向非正确通路的坐标并将次坐标代入循环，这时明明已经没有通路，next\_coordinations应该为空并将该坐标从st栈中删除，但由于next\_coordinations不为空，并未将该坐标从st栈中删除，继续寻找next\_coordinations坐标。后通过将next\_coordinations栈在每次循环内部重新初始化，让该栈只包含当前坐标的下一步坐标，保证无效坐标被正确删除。

（2）算法的时间空间复杂度分析：非递归程序空间复杂度为O(n\*m)，主要来自于迷宫存储、栈存储和已访问单元格列表；时间复杂度为O（n\*m），由于每次循环内部寻找下一坐标的操作的事件中复杂度都是常数，所以在最坏情况下，遍历所有坐标的时间复杂度就是该算法时间复杂度。递归程序空间复杂度为O（n\*m)，需要存储所有访问过的单元格以及递归栈空间；时间复杂度为O（n\*m），但实际复杂度可能更高，因为递归调用可能会多次访问同一个单元格。

（3）算法讨论：考虑树的数据结构，将迷宫所有可通行坐标作为结点存储，找从根部到指定叶的唯一路径。

## 五、用户手册

（1）非递归程序和递归程序：按照终端提示语句在冒号后进行输入，迷宫长和宽用“\*”连接，迷宫矩阵内容用空格隔开，出口入口坐标的横纵坐标用“，”连接，输入完成后按下回车键确认，等待结果输出。

（2）拓展程序：点击random按钮随机生成迷宫或者按照窗口提示语句在指定文本框内进行输入，迷宫长和宽用“\*”连接，迷宫矩阵内容用空格隔开，出口入口坐标的横纵坐标用“，”连接，信息全部输入完成后，点击“Solve”按钮，当有解时，会在下方画布中显示迷宫，点击右下角“Next”按钮一步步呈现解，最后按下最下方“Over”按钮，界面清空，可重新输入进行下一次迷宫求解。

## 六、测试数据及测试结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试用例 | 测试目的 | 正确输出 | 非递归程序输出 | 递归程序输出 | 拓展程序输出 | 当前状态 |
| 0 0 1  1 0 1  1 0 0 | 程序是否可以正常运行 | [[0, 0, 'right']->[0, 1, 'under']->[1, 1, 'under']->[2, 1, 'right']->[2, 2] | 图1.18.1 | 图1.18.2 | 图1.18.3 | 通过 |
| 0 1 1 0 1  0 0 1 0 1  1 0 0 0 0 | 程序能否处理岔路口干扰 | [0, 0, 'under']->[1, 0, 'right']->[1, 1, 'under']->[2, 1, 'right']->[2, 2, 'right']->[2, 3, 'right']->[2, 4] | 图1.18.4 | 图1.18.5 | 图1.18.6 | 通过 |
| 0 0 1 1 1  1 0 1 1 1  1 1 1 1 1  1 1 1 0 0 | 程序能否处理不存在正确通路的情况 | The maze has no access! | 图1.18.7 | 图1.18.8 | 图1.18.9 | 通过 |
| 0 0 1 1 1 0 1  1 0 0 0 0 0 0  0 1 0 1 0 1 1  1 1 1 0 0 0 0 | 程序能否处理存在较复杂干扰的情况 | [0, 0, 'right']->[0, 1, 'under']->[1, 1, 'right']->[1, 2, 'right']->[1, 3, 'right']->[1, 4, 'under']->[2, 4, 'under']->[3, 4, 'right']->[3, 5, 'right']->[3, 6] | 图1.18.10 | 图1.18.11 | 图1.18.12 | 通过 |
| 1 1 0 1 1 1 0 1 0  1 1 0 0 0 0 0 0 0  1 0 0 1 0 1 1 0 1  0 1 1 0 0 0 0 1 1  0 0 0 0 1 1 1 1 0  1 0 1 1 1 0 1 1 0 | 程序能否处理长宽更大数据量更大的迷宫情况 | [0, 8, 'under']->[1, 8, 'left']->[1, 7, 'left']->[1, 6, 'left']->[1, 5, 'left']->[1, 4, 'under']->[2, 4, 'under']->[3, 4, 'left']->[3, 3, 'under']->[4, 3, 'left']->[4, 2, 'left']->[4, 1, 'left']->[4, 0] | 图1.18.13 | 图1.18.14 | 图1.18.15 | 通过 |

图1.18 测试用例表



图1.18.1用例1非递归程序输出

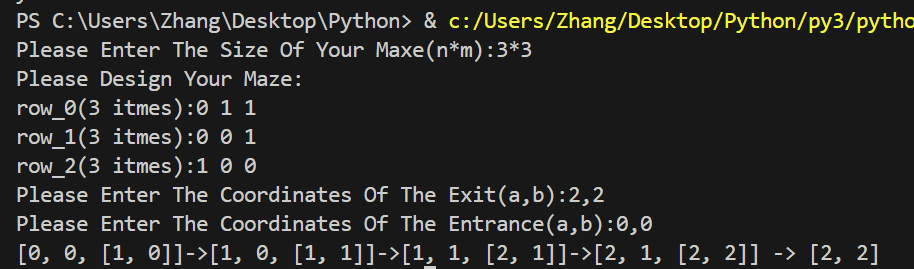


图1.18.2用例1递归程序输出

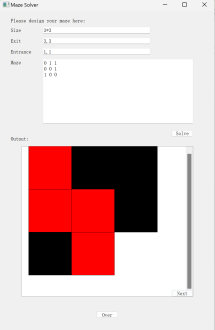
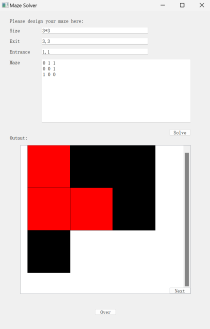
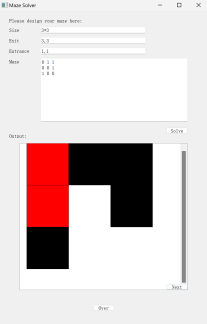
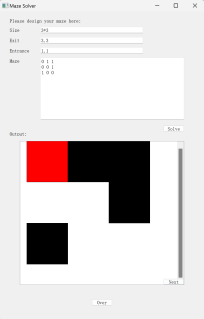
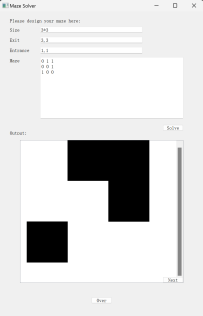


图1.18.3用例1拓展程序输出

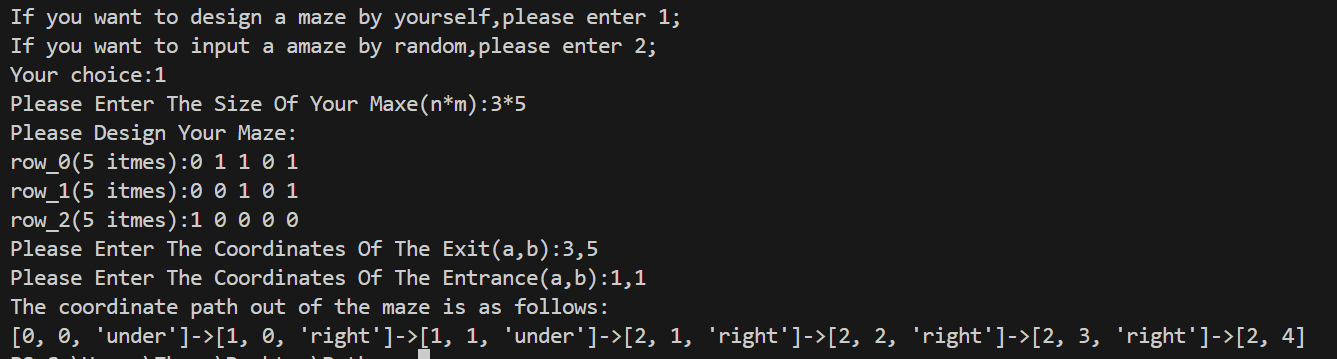


图1.18.4用例2非递归程序输出

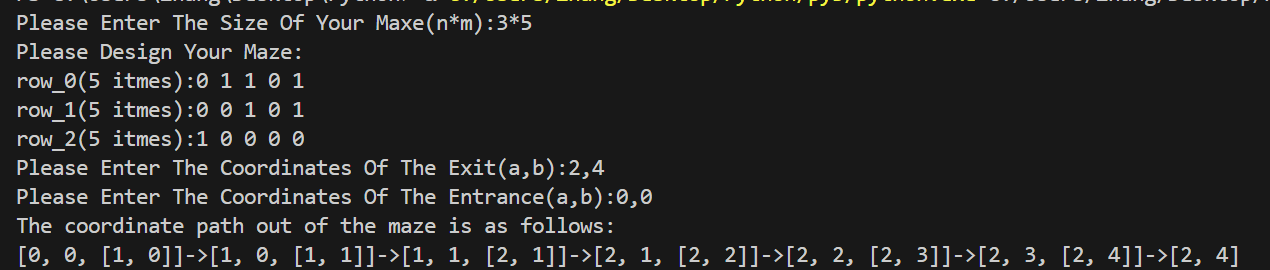


图1.18.5用例2递归程序输出

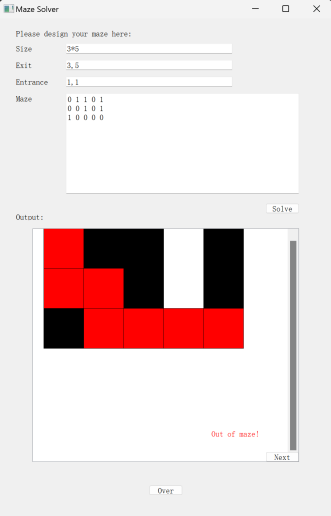


图1.18.6用例2拓展程序输出

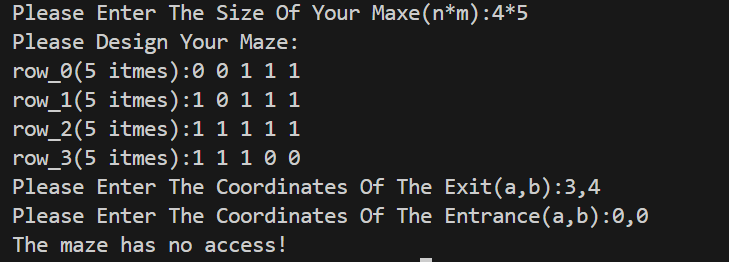


图1.18.7用例3非递归程序输出

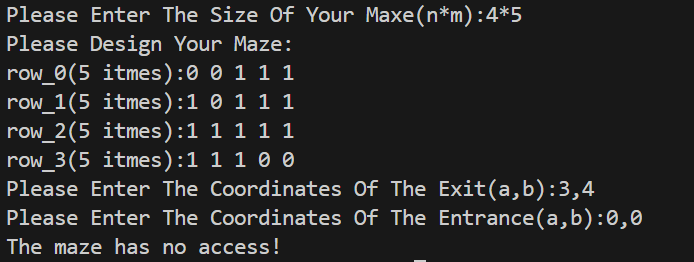


图1.18.8用例3递归程序输出



图1.18.9用例3拓展程序输出

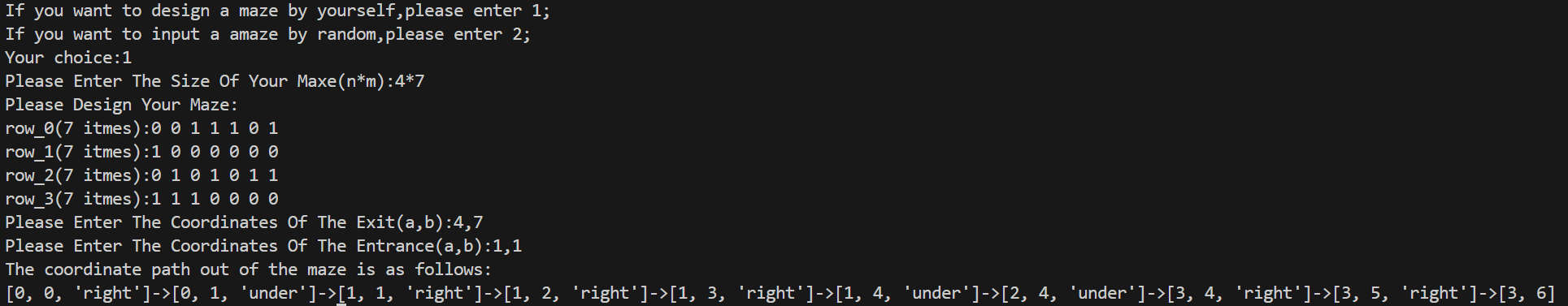


图1.18.10用例4非递归程序输出

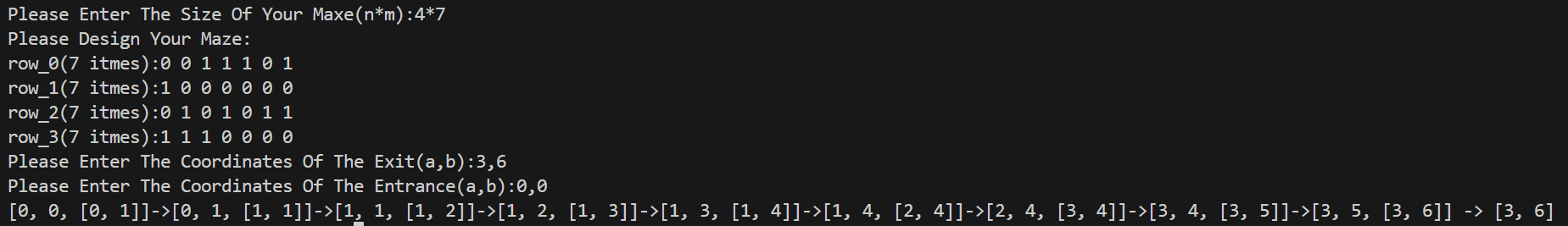


图1.18.11用例4递归程序输出

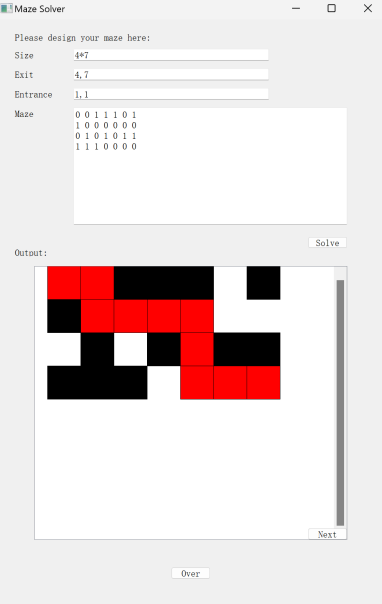


图1.18.12用例4拓展程序输出



图1.18.13用例5非递归程序输出

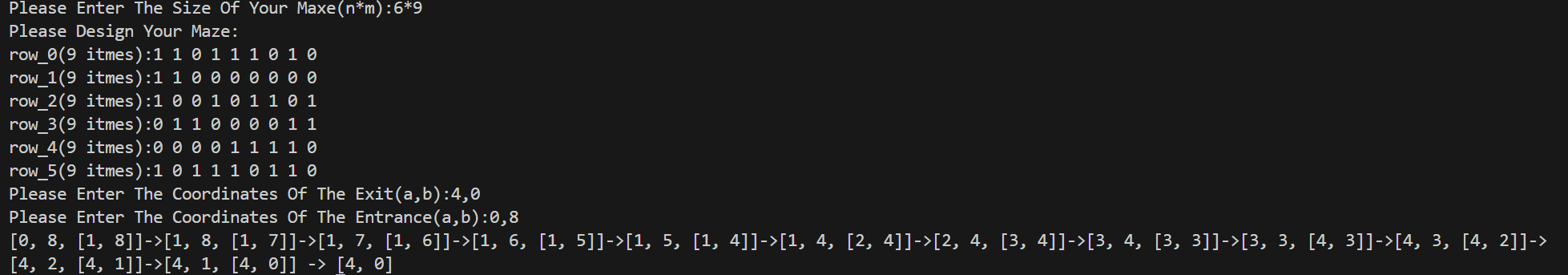


图1.18.14用例5递归程序输出

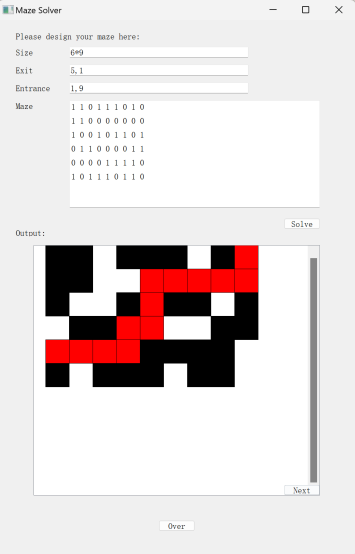


图1.18.15用例5拓展程序输出

## 七、源程序清单

（1）非递归程序

import random

#design the database-stack

class Stack():

    def \_\_init\_\_(self):

        self.data=[]

        self.top=-1

    def is\_empty(self):

        return self.top<0

    def size(self):

        return self.top+1

    def top\_value(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            return self.data[self.top]

    def push(self,value):

        self.top+=1

        self.data.append(value)

    def pop(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            value=self.data.pop()

            self.top-=1

            return value

#design a maze

print("If you want to design a maze by yourself,please enter 1;")

print("If you want to input a amaze by random,please enter 2;")

choice=int(input("Your choice:"))

if choice==1:

    n,m=map(int,input("Please Enter The Size Of Your Maxe(n\*m):").split("\*"))#n rows with each row m items

    print("Please Design Your Maze:")

    Maze=[]

    for i in range(n):

        row=input(f"row\_{i}({m} itmes):").split()

        Maze.append(row)

    a,b=map(int,input("Please Enter The Coordinates Of The Exit(a,b):").split(","))

    exit=[a-1,b-1]

    a,b=map(int,input("Please Enter The Coordinates Of The Entrance(a,b):").split(","))

    entr=[a-1,b-1]

elif choice==2:

    Maze=[]

    m=random.randint(2,10)

    n=random.randint(2,10)

    weights=[0.7,0.3]

    numbers=[0,1]

    for i in range(n):

        row=[]

        for j in range(m):

            row.append(str(random.choices(numbers,weights,k=1)[0]))

        Maze.append(row)

    exit=[n-1,m-1]

    entr=[0,0]

    print("The maze is as follows:")

    for i in range(n):

        for j in range(m):

            print(Maze[i][j],end="")

        print("\n",end="")

    print("The coordinates of the entrance is:",entr)

    print("The coordinates of the exit is:",exit)

else:

    print("Wrong Input!")

#find the way

st=Stack()#the right way

exit.append("exit")

st.push(exit)

done=[]#the place already passed

done.append([exit[0],exit[1]])

next\_coordinations=Stack()

while [st.top\_value()[0],st.top\_value()[1]]!=entr:

    next\_coordinations=[]

    #find the possibiltis of the next coordination first:1.still valid 2.is "0" 3.have not passed

    if st.top\_value()[0]-1>=0 and Maze[st.top\_value()[0]-1][st.top\_value()[1]]=="0" and [st.top\_value()[0]-1,st.top\_value()[1]] not in done:

        next\_coordinations.append([st.top\_value()[0]-1,st.top\_value()[1],"upper"])#upper

    if st.top\_value()[0]+1<n and Maze[st.top\_value()[0]+1][st.top\_value()[1]]=="0" and [st.top\_value()[0]+1,st.top\_value()[1]] not in done:

        next\_coordinations.append([st.top\_value()[0]+1,st.top\_value()[1],"under"])#under

    if st.top\_value()[1]-1>=0 and Maze[st.top\_value()[0]][st.top\_value()[1]-1]=="0" and [st.top\_value()[0],st.top\_value()[1]-1] not in done:

        next\_coordinations.append([st.top\_value()[0],st.top\_value()[1]-1,"left"])#left

    if st.top\_value()[1]+1<m and Maze[st.top\_value()[0]][st.top\_value()[1]+1]=="0" and [st.top\_value()[0],st.top\_value()[1]+1] not in done:

        next\_coordinations.append([st.top\_value()[0],st.top\_value()[1]+1,"right"])#right

    #consider a fork:if one comes to an end,we need to back and find another,until we back to the start,taht means there is really no way

    if len(next\_coordinations)==0 and [st.top\_value()[0],st.top\_value()[1]]==[exit[0],exit[1]]:

        break

    elif len(next\_coordinations)==0 and [st.top\_value()[0],st.top\_value()[1]]!=[exit[0],exit[1]]:

        k=st.pop()

    else:

        next=next\_coordinations.pop(-1)

        done.append([next[0],next[1]])

        st.push(next)

#output

if [st.top\_value()[0],st.top\_value()[1]]==entr:

    print("The coordinate path out of the maze is as follows:")

    for i in range(st.size()):

        now=st.pop()

        if [now[0],now[1]]!=[exit[0],exit[1]]:

            output=[]

            way=now[2]

            if way=="upper":

                output=[now[0],now[1],"under"]

            if way=="under":

                output=[now[0],now[1],"upper"]

            if way=="left":

                output=[now[0],now[1],"right"]

            if way=="right":

                output=[now[0],now[1],"left"]

            print(output,end="->")

        else:

            print([now[0],now[1]])

else:

    print("The maze has no access!")

（2）递归程序

import random

#design a maze

print("If you want to design a maze by yourself,please enter 1;")

print("If you want to input a amaze by random,please enter 2;")

choice=int(input("Your choice:"))

if choice==1:

    n,m=map(int,input("Please Enter The Size Of Your Maxe(n\*m):").split("\*"))#n rows with each row m items

    print("Please Design Your Maze:")

    Maze=[]

    for i in range(n):

        row=input(f"row\_{i}({m} itmes):").split()

        Maze.append(row)

    a,b=map(int,input("Please Enter The Coordinates Of The Exit(a,b):").split(","))

    exit=[a-1,b-1]

    a,b=map(int,input("Please Enter The Coordinates Of The Entrance(a,b):").split(","))

    entr=[a-1,b-1]

elif choice==2:

    Maze=[]

    m=random.randint(2,10)

    n=random.randint(2,10)

    weights=[0.7,0.3]

    numbers=[0,1]

    for i in range(n):

        row=[]

        for j in range(m):

            row.append(str(random.choices(numbers,weights,k=1)[0]))

        Maze.append(row)

    exit=[n-1,m-1]

    entr=[0,0]

    print("The maze is as follows:")

    for i in range(n):

        for j in range(m):

            print(Maze[i][j],end="")

        print("\n",end="")

    print("The coordinates of the entrance is:",entr)

    print("The coordinates of the exit is:",exit)

else:

    print("Wrong Input!")

#find the way

done=[]

bingo=False

def judge(coordination,Maze,done) :

    global bingo

    done.append(coordination)

    if coordination==entr:

        return True

    else:

        next\_coordinations=[]

        if coordination[0]-1>=0 and Maze[coordination[0]-1][coordination[1]]=="0" and [coordination[0]-1,coordination[1]] not in done:

            next\_coordinations.append([coordination[0]-1,coordination[1]])#upper

        if coordination[0]+1<n and Maze[coordination[0]+1][coordination[1]]=="0" and [coordination[0]+1,coordination[1]] not in done:

            next\_coordinations.append([coordination[0]+1,coordination[1]])#under

        if coordination[1]-1>=0 and Maze[coordination[0]][coordination[1]-1]=="0" and [coordination[0],coordination[1]-1] not in done:

            next\_coordinations.append([coordination[0],coordination[1]-1])#left

        if coordination[1]+1<m and Maze[coordination[0]][coordination[1]+1]=="0" and [coordination[0],coordination[1]+1] not in done:

            next\_coordinations.append([coordination[0],coordination[1]+1])#right

        if len(next\_coordinations)!=0:

            for i in next\_coordinations:

                bingo=judge(i,Maze,done)

                if bingo:

                    if coordination==exit:

                        i.append(coordination)

                        print(i,"->",coordination)

                        break

                    else:

                        i.append(coordination)

                        print(i,end="->")

                        break

        return bingo

if judge(exit,Maze,done)==False:

    print("The maze has no access!")

（3）拓展程序

import sys

from PyQt5.QtGui import QColor, QBrush, QPen, QPainter, QFont

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel, QLineEdit, QTextEdit, QGraphicsView, QGraphicsScene, QGraphicsRectItem, QGraphicsTextItem

from PyQt5.QtCore import Qt

from functools import partial

import random

class Stack():

    def \_\_init\_\_(self):

        self.data = []

        self.top = -1

    def is\_empty(self):

        return self.top < 0

    def size(self):

        return self.top + 1

    def top\_value(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            return self.data[self.top]

    def push(self, value):

        self.top += 1

        self.data.append(value)

    def pop(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            value = self.data.pop()

            self.top -= 1

            return value

class MazeSolver(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Maze Solver")

        self.resize(1000, 1500)

        # Create labels and line edits

        self.label\_0 = QLabel("Please design your maze here:", self)

        self.label\_0.setGeometry(50, 30, 500, 30)

        self.label\_1 = QLabel("Size", self)

        self.label\_1.setGeometry(50, 80, 200, 20)

        self.edit\_1 = QLineEdit(self)

        self.edit\_1.setPlaceholderText("n\*m")

        self.edit\_1.setGeometry(200, 75, 500, 30)

        self.label\_2 = QLabel("Exit", self)

        self.label\_2.setGeometry(50, 130, 200, 20)

        self.edit\_2 = QLineEdit(self)

        self.edit\_2.setPlaceholderText("x,y")

        self.edit\_2.setGeometry(200, 125, 500, 30)

        self.label\_3 = QLabel("Entrance", self)

        self.label\_3.setGeometry(50, 180, 200, 20)

        self.edit\_3 = QLineEdit(self)

        self.edit\_3.setPlaceholderText("x,y")

        self.edit\_3.setGeometry(200, 175, 500, 30)

        self.label\_4 = QLabel("Maze", self)

        self.label\_4.setGeometry(50, 230, 500, 20)

        self.edit\_4 = QTextEdit(self)

        self.edit\_4.setPlaceholderText("0 for paths and 1 for walls")

        self.edit\_4.setGeometry(200, 225, 700, 300)

        self.edit\_4.setFixedWidth(700)

        self.edit\_4.setFixedHeight(300)

        # Create a button

        self.solve\_button = QPushButton("Solve", self)

        self.solve\_button.setGeometry(800, 555, 100, 30)

        self.solve\_button.clicked.connect(self.solve)

        # Create "Over" button

        self.over\_button = QPushButton("Over", self)

        self.over\_button.setGeometry(450, 1400, 100, 30)

        self.over\_button.clicked.connect(self.over)

        # Create "Random" button

        self.random\_button = QPushButton("Random", self)

        self.random\_button.setGeometry(600, 30, 100, 30)

        self.random\_button.clicked.connect(self.random)

        # Create the drawing area

        self.label\_5 = QLabel("Output:", self)

        self.label\_5.setGeometry(50, 585, 200, 20)

        self.scene = QGraphicsScene(self)

        self.scene.setSceneRect(0, 0, 700, 700)

        self.view = QGraphicsView(self.scene, self)

        self.view.setGeometry(100, 630, 800, 700)

    def random(self):

        Maze=[]

        m=random.randint(2,10)

        n=random.randint(2,10)

        weights=[0.7,0.3]

        numbers=[0,1]

        for i in range(n):

            row=[]

            for j in range(m):

                row.append(str(random.choices(numbers,weights,k=1)[0]))

            Maze.append(row)

        exit=[n-1,m-1]

        entr=[0,0]

        st = Stack()

        done = []

        st.push(exit)

        done.append(exit)

        while st.top\_value() != entr:

            next\_coordinations = Stack()

            if st.top\_value()[0] - 1 >= 0 and Maze[st.top\_value()[0] - 1][st.top\_value()[1]] == '0' and [st.top\_value()[0] - 1, st.top\_value()[1]] not in done:

                next\_coordinations.push([st.top\_value()[0] - 1, st.top\_value()[1]])  # upper

            if st.top\_value()[0] + 1 < n and Maze[st.top\_value()[0] + 1][st.top\_value()[1]] == '0' and [st.top\_value()[0] + 1, st.top\_value()[1]] not in done:

                next\_coordinations.push([st.top\_value()[0] + 1, st.top\_value()[1]])  # under

            if st.top\_value()[1] - 1 >= 0 and Maze[st.top\_value()[0]][st.top\_value()[1] - 1] == '0' and [st.top\_value()[0], st.top\_value()[1] - 1] not in done:

                next\_coordinations.push([st.top\_value()[0], st.top\_value()[1] - 1])  # left

            if st.top\_value()[1] + 1 < m and Maze[st.top\_value()[0]][st.top\_value()[1] + 1] == '0' and [st.top\_value()[0], st.top\_value()[1] + 1] not in done:

                next\_coordinations.push([st.top\_value()[0], st.top\_value()[1] + 1])  # right

            if next\_coordinations.is\_empty() and st.top\_value() == exit:

                break

            elif next\_coordinations.is\_empty() and st.top\_value() != exit:

                st.pop()

            else:

                next = next\_coordinations.pop()

                done.append(next)

                st.push(next)

        if st.top\_value() == entr:

            hello=True

            self.access(Maze,st,hello)

        else:

            hello=False

            self.noaccess(Maze,st,hello)

    def solve(self):

        st = Stack()

        done = []

        input\_1 = self.edit\_1.text()

        input\_2 = self.edit\_2.text()

        input\_3 = self.edit\_3.text()

        input\_4 = self.edit\_4.toPlainText().strip()

        n, m = map(int, input\_1.split("\*"))

        x, y = map(int, input\_2.split(","))

        exit = [x - 1, y - 1]

        x, y = map(int, input\_3.split(","))

        entr = [x - 1, y - 1]

        Maze = [i.split() for i in input\_4.split('\n')]

        st.push(exit)

        done.append(exit)

        while st.top\_value() != entr:

            next\_coordinations = Stack()

            if st.top\_value()[0] - 1 >= 0 and Maze[st.top\_value()[0] - 1][st.top\_value()[1]] == '0' and [st.top\_value()[0] - 1, st.top\_value()[1]] not in done:

                next\_coordinations.push([st.top\_value()[0] - 1, st.top\_value()[1]])  # upper

            if st.top\_value()[0] + 1 < n and Maze[st.top\_value()[0] + 1][st.top\_value()[1]] == '0' and [st.top\_value()[0] + 1, st.top\_value()[1]] not in done:

                next\_coordinations.push([st.top\_value()[0] + 1, st.top\_value()[1]])  # under

            if st.top\_value()[1] - 1 >= 0 and Maze[st.top\_value()[0]][st.top\_value()[1] - 1] == '0' and [st.top\_value()[0], st.top\_value()[1] - 1] not in done:

                next\_coordinations.push([st.top\_value()[0], st.top\_value()[1] - 1])  # left

            if st.top\_value()[1] + 1 < m and Maze[st.top\_value()[0]][st.top\_value()[1] + 1] == '0' and [st.top\_value()[0], st.top\_value()[1] + 1] not in done:

                next\_coordinations.push([st.top\_value()[0], st.top\_value()[1] + 1])  # right

            if next\_coordinations.is\_empty() and st.top\_value() == exit:

                break

            elif next\_coordinations.is\_empty() and st.top\_value() != exit:

                st.pop()

            else:

                next = next\_coordinations.pop()

                done.append(next)

                st.push(next)

        if st.top\_value() == entr:

            hello=True

            self.access(Maze,st,hello)

        else:

            hello=False

            self.noaccess(Maze,st,hello)

    def noaccess(self,Maze,st,Hello):

        self.drawMaze(Maze)

        self.next\_button = QPushButton("Next", self)

        self.next\_button.setGeometry(800, 1300, 100, 30)

        self.next\_button.clicked.connect(partial(self.next,Maze, st,Hello))

        self.next\_button.show()

    def access(self, st, Maze,Hello):

        self.drawMaze(Maze)

        self.next\_button = QPushButton("Next", self)

        self.next\_button.setGeometry(800, 1300, 100, 30)

        self.next\_button.clicked.connect(partial(self.next,Maze, st,Hello))

        self.next\_button.show()

    def next(self,Maze, st,Hello):

        if st.is\_empty() and Hello ==True:

            text = "Out of maze!"

            text\_item = QGraphicsTextItem(text)

            text\_item.setDefaultTextColor(QColor(255, 0, 0))

            text\_item.setPos(200,650.0)

            self.scene.addItem(text\_item)

        elif st.top\_value()==[len(Maze)-1,len(Maze[0])-1] and Hello==False:

            text = "The maze has no access!"

            text\_item = QGraphicsTextItem(text)

            text\_item.setDefaultTextColor(QColor(255, 0, 0))

            text\_item.setPos(200,650.0)

            self.scene.addItem(text\_item)

        else:

            current\_position = st.pop()

            self.drawPath(Maze,current\_position)

    def drawMaze(self, Maze):

        cell\_size = 600//max(len(Maze),len(Maze[0]))

        wall\_color = QColor(0, 0, 0)

        for i in range(len(Maze)):

            for j in range(len(Maze[i])):

                if Maze[i][j] == '1':

                    rect\_item = QGraphicsRectItem(j \* cell\_size, i \* cell\_size, cell\_size, cell\_size)

                    rect\_item.setBrush(wall\_color)

                    self.scene.addItem(rect\_item)

        self.scene.update()

    def drawPath(self, Maze,position):

        cell\_size = 600//max(len(Maze),len(Maze[0]))

        path\_color = QColor(255, 0, 0)

        rect\_item = QGraphicsRectItem(position[1] \* cell\_size, position[0] \* cell\_size, cell\_size, cell\_size)

        rect\_item.setBrush(path\_color)

        self.scene.addItem(rect\_item)

    def over(self):

        self.scene.clear()

        self.edit\_1.clear()

        self.edit\_2.clear()

        self.edit\_3.clear()

        self.edit\_4.clear()

        self.solve\_button.setEnabled(True)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    maze\_solver = MazeSolver()

    maze\_solver.show()

    sys.exit(app.exec\_())

# 

# 【题目二】二叉树的应用——表达式求值

## 一、课程设计题目与要求

1.问题描述

一个表达式和一棵二叉树之间，存在着自然的对应关系。假设一个算术表达式可包含变量(a~z)、常量(0~9)、圆括号和二元运算符(+，-，\*，/)。

2.基本要求

（1）输入中缀形式的算术表达式，并进行合法性检查；

（2）利用堆栈将中缀形式的算术表达式变为后缀形式并输出；然后根据后缀形式，构建相应的二叉树，并打印构建二叉树的过程；

（3）对构建的二叉树进行前序遍历、中序遍历和后序遍历，打印遍历序列。

（4）对各变量赋值，基于构建的二叉树对表达式求值，并打印求值的过程。

（5）测试几组不同的数据，完成上述要求。

3.扩展要求

构造一个新的复合表达式(E1) P (E2)，其中P是运算符，E1和E2分别表示两个合法的中缀算术表达式；对复合表达式构造对应的二叉树，打印二叉树并求值。

## 二、需求分析

1.基本功能需求

（1）输入一个中缀形式的算术表达式。

（2）输出其算术表达式的后缀形式。输出二叉树构建过程。输出二叉树的前序遍历，中序遍历和后序遍历的序列。

（3）对变量赋值，表达式求值。

2.用户界面需求

（1）需要给用户提示信息，提醒用户输入中缀算术表达式的格式。

（2）在进行输出时需要给用户提示信息，提示所输出的内容介绍。

## 三、设计

### 3.1 设计思想

#### 3.1.1数据结构设计

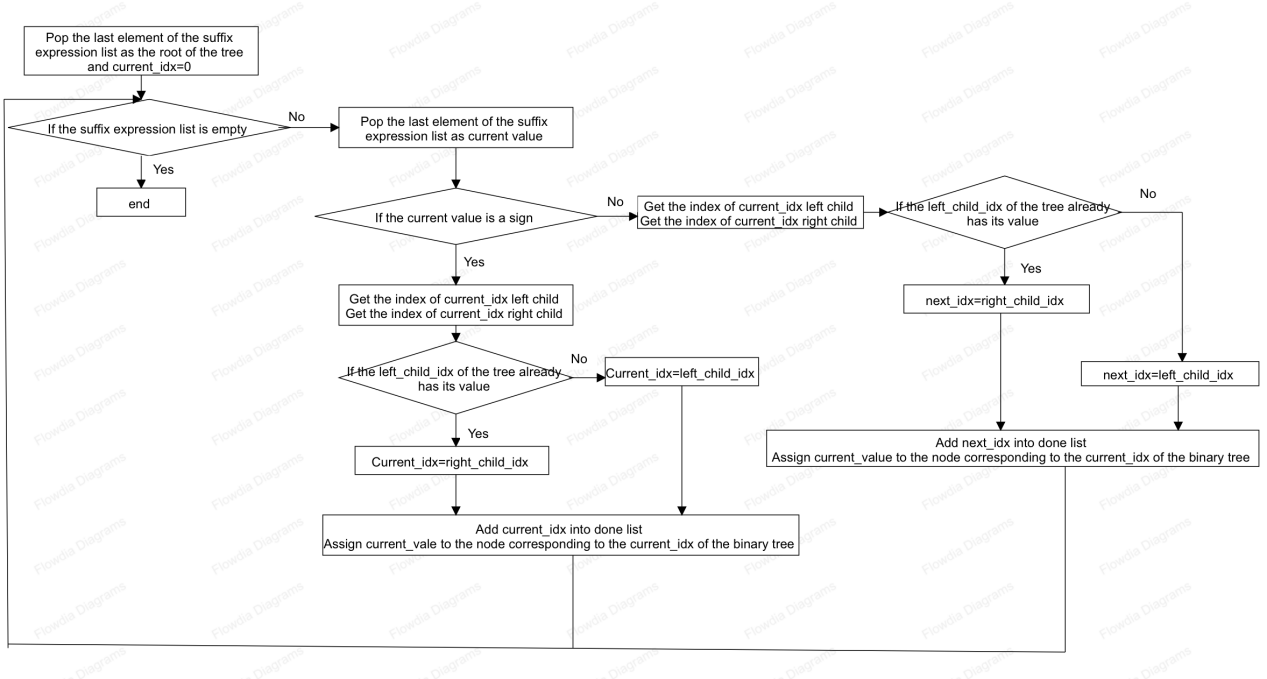
1. 栈（Stack）：根据题目要求，要用利用堆栈将中缀形式的算术表达式变为后缀形式并输出。栈确保了运算符按照正确的优先级顺序应用，并且正确处理括号。栈临时保存运算符和括号，帮助算法正确处理表达式，最终生成后缀表达式。
2. 二叉树（BinaryTree）：根据题目要求，要将后缀表达式生成二叉树，并通过二叉树对算式进行计算。表示表达式的二叉树结构，其中每个节点可以是运算符或操作数。这种结构帮助实现表达式的计算和可视化。
3. 列表：根据题目要求，列表用于存储输入的中缀算数式和转化成的后缀算数式。根据题目给定的所输入算术表达式各元素的范围，需要判断元素是否在范围内和属于哪个类别，多个列表以字符串的形式存储各类别范围内元素，用于查找。根据算法需求，在转化二叉树时，列表存储已赋值的二叉树节点的索引，用于判断该坐标是否还可以赋值。
4. 字典：根据算法需求，字典存储各符号及其对应权重，更好的判断各符号的计算优先顺序。根据算法需求，在最后赋值计算时，字典存储未知数及其对应值，可以直接通过未知数查找其值，有更好的对应关系。

#### 3.1.2算法设计

1. 构建二叉树

后缀表达式具有符号在后、数字在前且数字截尾的特点，根据此特点，将后缀表达式从末尾向开头依次获取元素按从上往下顺序放入二叉树。整体通过while循环结构，在while循环外部先删除并获取后缀表达式列表的最后一个元素，并将该元素放入二叉树当前索引为0处作为二叉树的根，每次循环内部，删除并获取后缀表达式列表的最后一个元素作为当前值，根据当前值的类型判断其对应索引为当前索引，将当前值插入二叉树的当前索引处，并将当前索引加入done列表，表示该索引已赋值，直到后缀表达式列表被删除完全，长度为0时，结束循环，获得二叉树。

构建二叉树设计思想流程图如图2.1所示。

图2.1 构建二叉树设计思想流程图

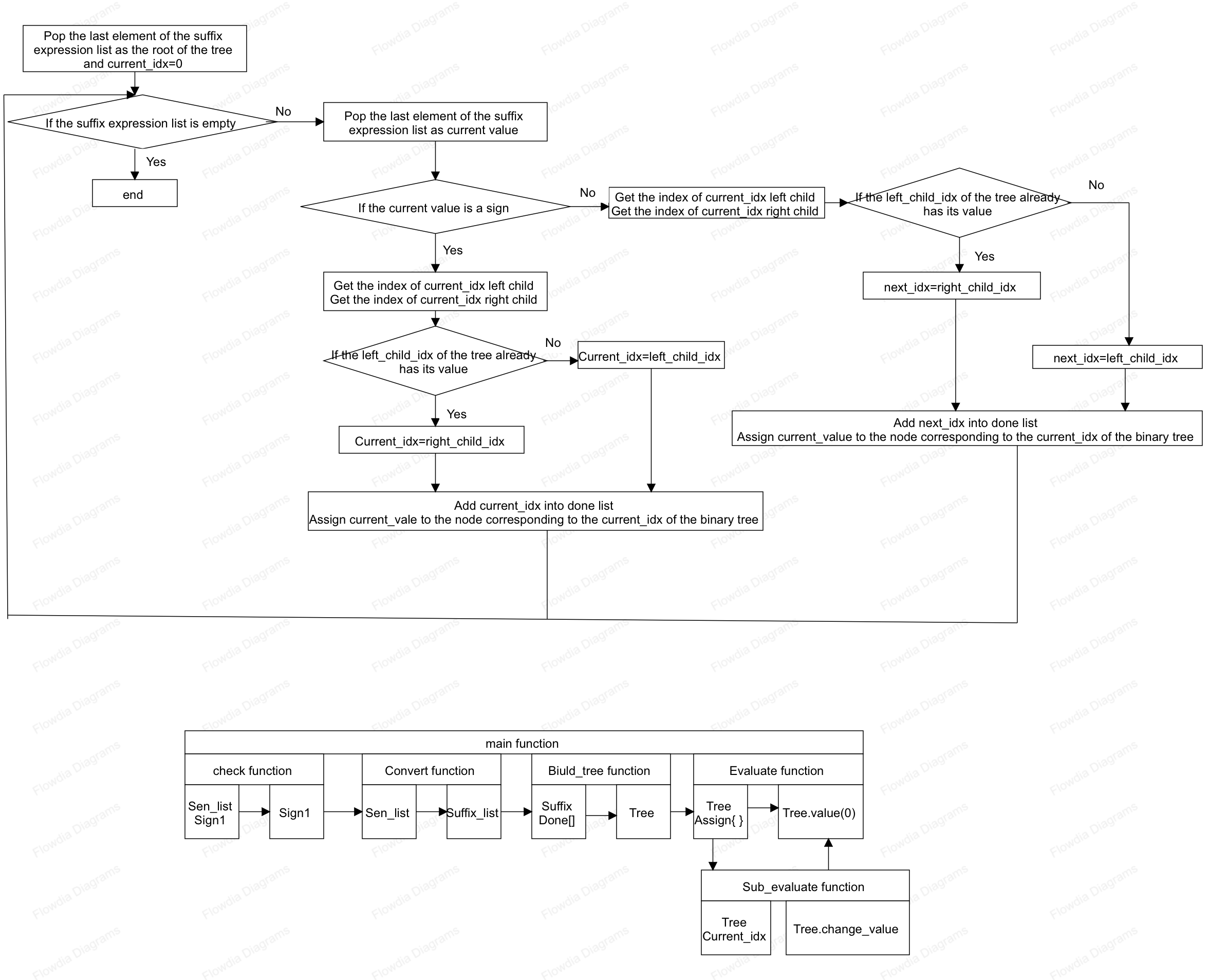
1. 计算二叉树

根据后缀表达所构成的二叉树的特点，将二叉树从下往上、从后往前计算。先获得最后一个叶节点的索引作为当前索引，整体通过while循环结构，在while循环内部，根据当前索引对应的二叉树的值的类型判断是否进行计算和确认下一步索引，当值为非符号时，存在数字、未知数、None三种情况，前两种情况直接返回其父节点的索引为当前索引，最后一种情况返回前一索引作为当前索引；当值为符号时，获得其左右节点的值放入子计算函数进行对应计算，将计算结果重新赋值给当前索引的值，后返回前一索引作为当前索引，当当前索引查找到小于0时，结束循环，根节点对应的值为计算结果。

计算二叉树设计思想流程图如图2.2所示。

### 3.2 设计表示

主要函数的调用关系如图2.3所示。

图2.3 主要函数调用关系图

1. check函数模块：输入已写入的中缀算数式列表，初始化布尔变量sign1为False，经过穷举法判断中缀算数式是否合法，若不合法，sign1变为True，最终该函数返回sign1.
2. Convert函数模块：输入已写入的中缀算数式列表，经过整理重新写入后，输出后缀表达式列表suffix\_list。
3. Build\_tree函数模块：输入后缀表达式列表suffix，生成临时列表done来记录已赋值的索引，经过算法后获得二叉树tree。
4. Evaluate函数模块：输入二叉树，生成临时字典assign来保存未知数赋值，经过算法后获得树的根节点的值。在该函数中，又引用sub\_evaluate函数对符号节点进行计算并更改其值。

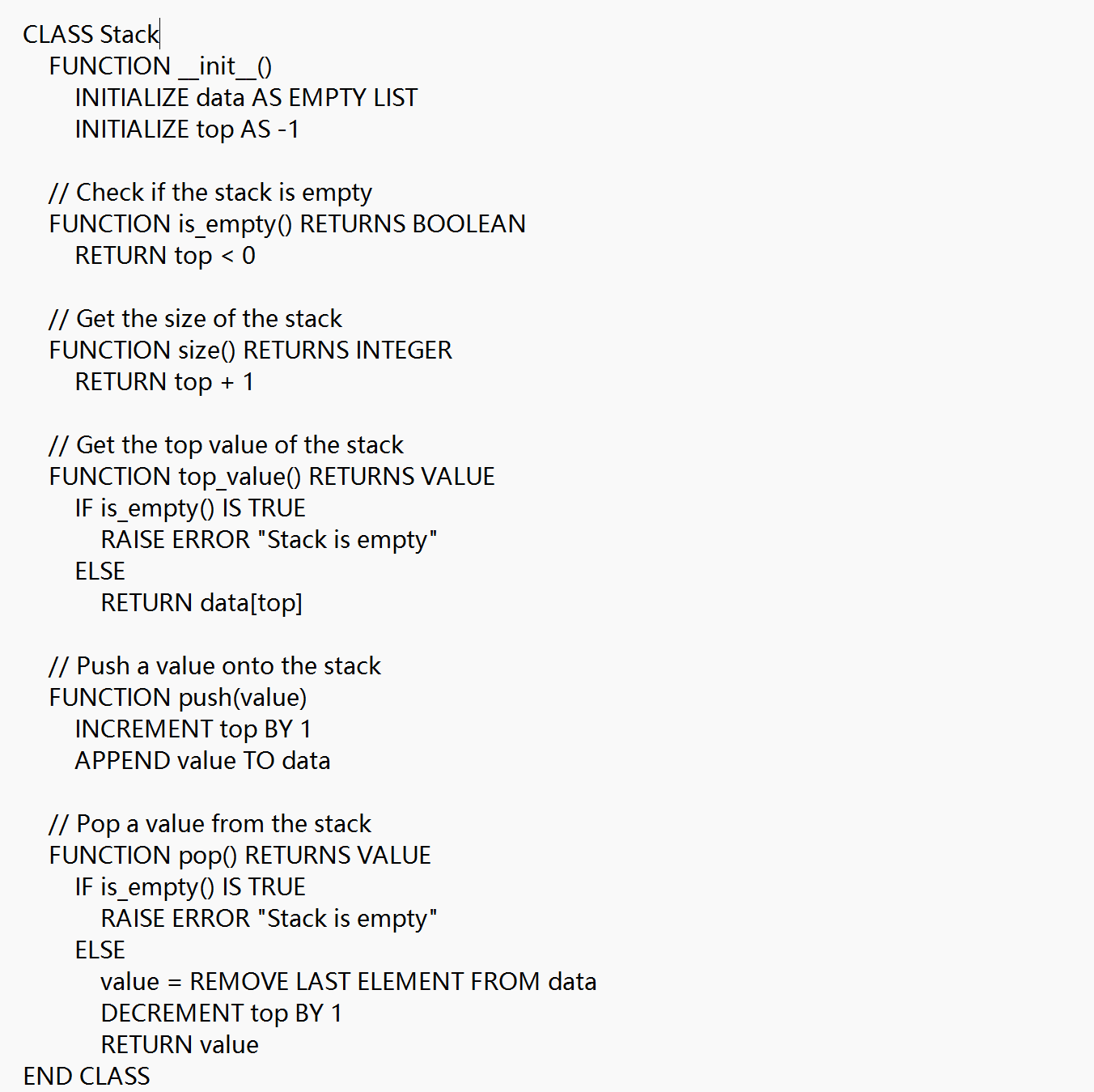
### 3.3 详细设计

#### 3.3.1 基本功能程序

（1）栈类设计

一个列表data，用于存储栈中的元素，一个整数top，表示栈顶元素的索引，初始化为-1。Is\_empty()函数用来判断栈是否为空，即判断top是否小于0，若小于0，则表示栈顶端序列号不存在，即为空，否则不为空，返回布尔值。Size()函数用来返回栈内元素数量，top表示最顶端元素序列号，即列表data最后一个元素的序列号，其加上1就是列表长度，也可以直接返回列表data的长度。Top\_value()函数用来返回当前顶端元素，即返回列表data的最后一个元素。Push(value)函数用来将value添加到栈中成为新的顶端，即将value添加到列表data的末端，并将表示顶端元素序列号的top值加一。Pop()函数用来将栈顶端元素去除并返回，即将列表data的末尾元素删除，并将表示顶端元素序列号的top值减一。

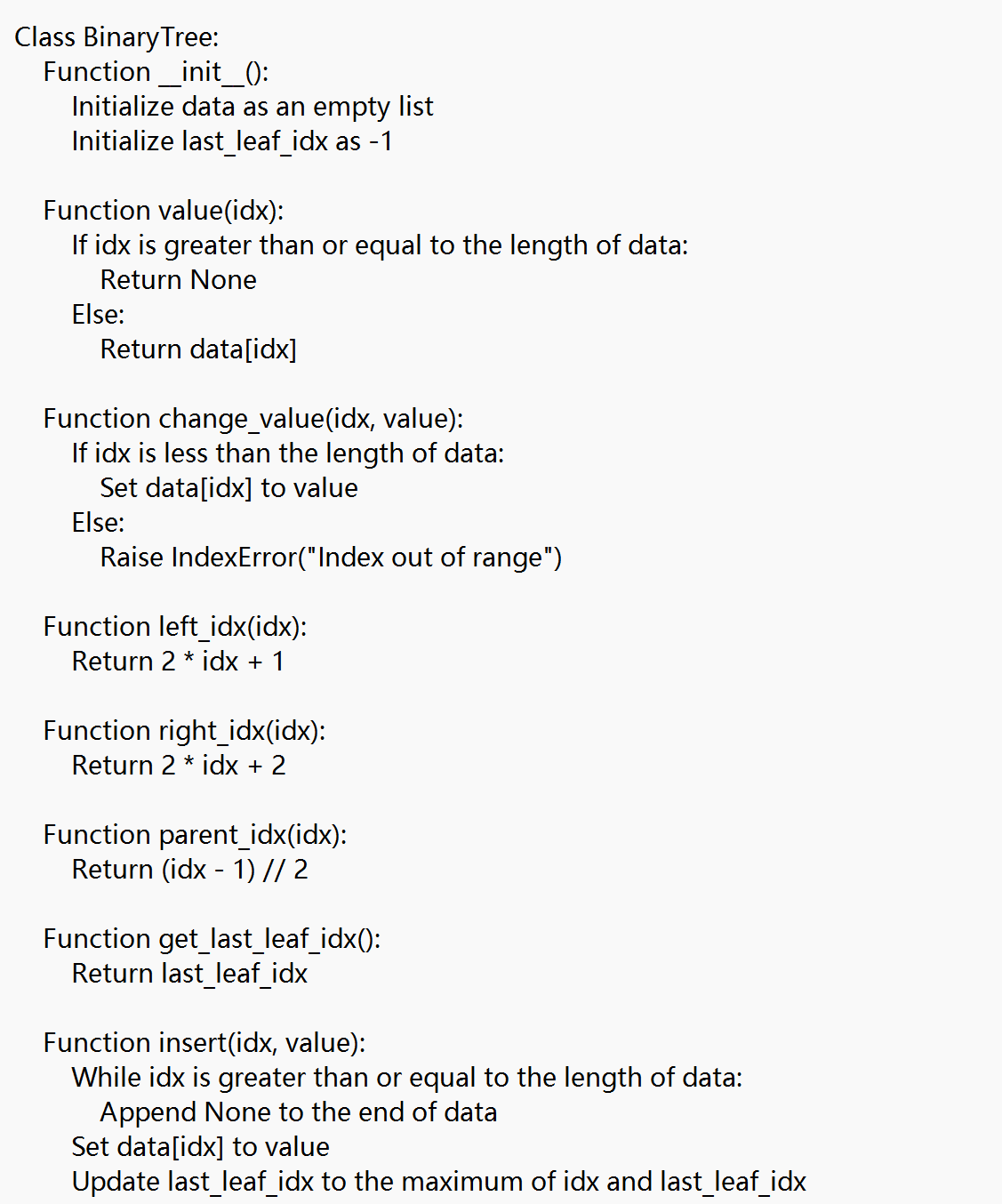
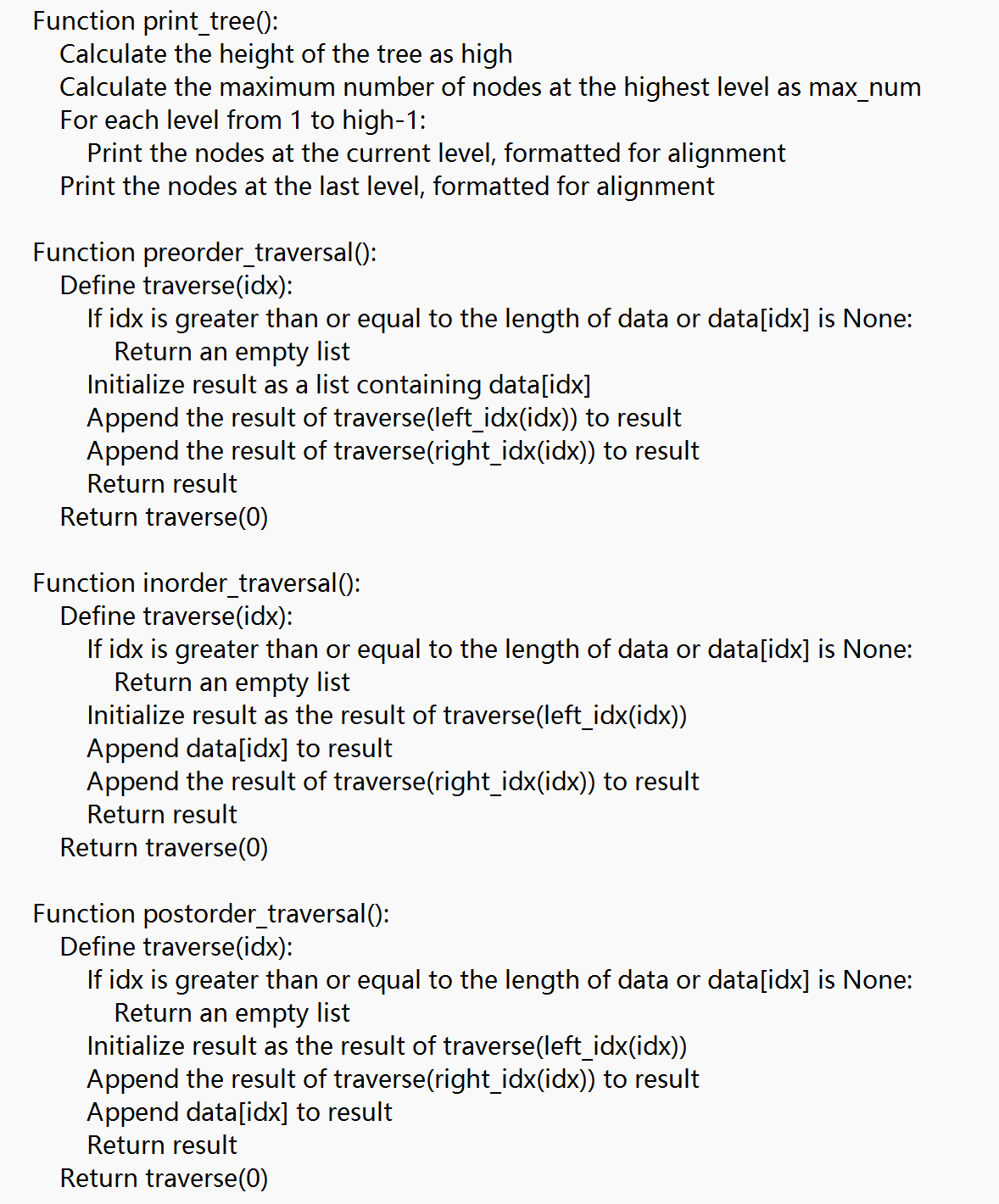
伪代码如图2.4所示。

图2.4 栈类设计伪代码

1. 二叉树类设计

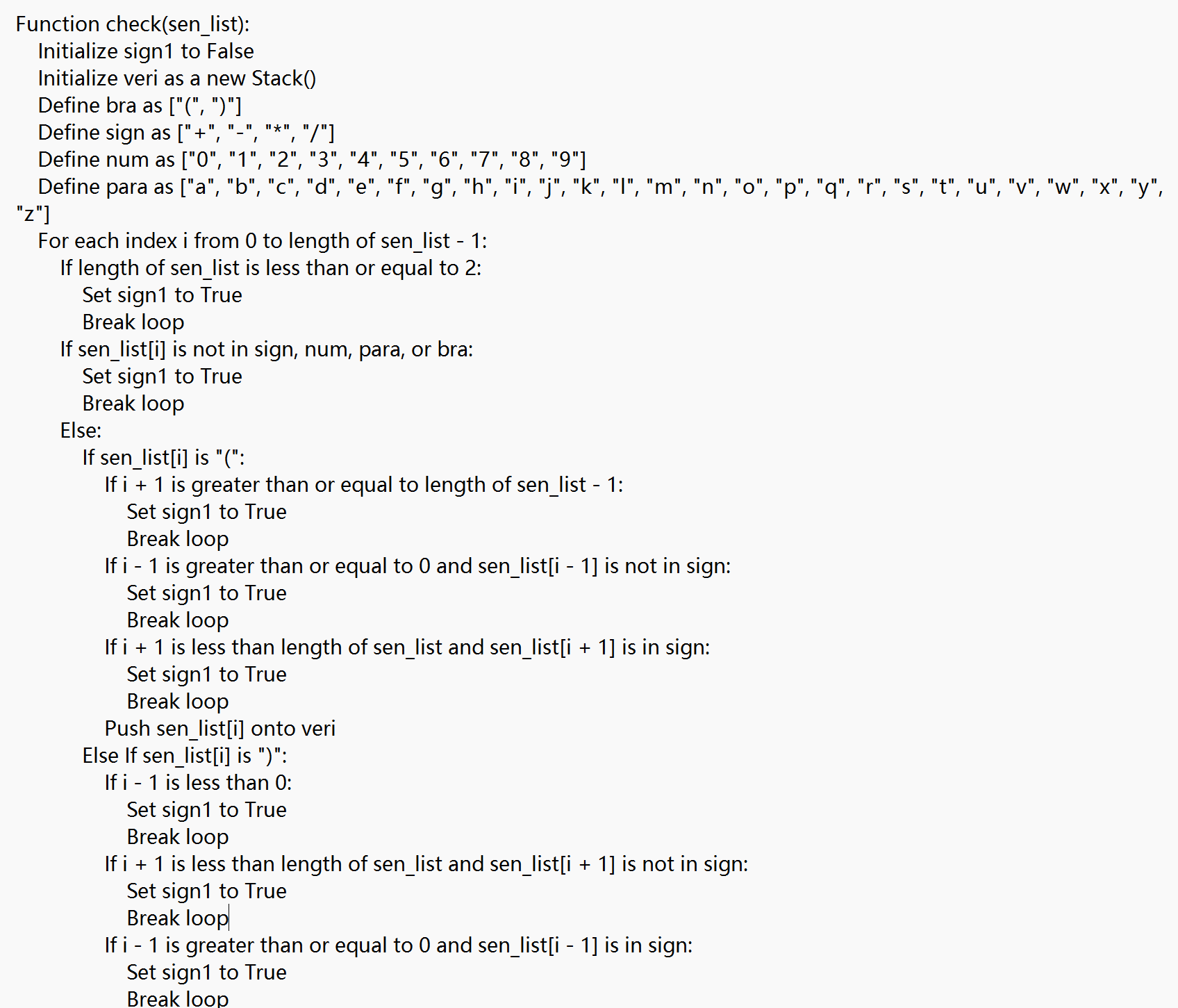
初始化二叉树，用列表存储二叉树，并初始化最后一个叶节点的索引为-1；value(idx)函数用来获得二叉树idx索引对应的值，直接返回该索引列表对应的值；change\_value(idx,value)函数是将二叉树idx索引处赋值为value，直接通过列表中的等号强制赋值；left\_idx(idx)和right\_idx(idx)函数分别求当前索引idx对应的左右节点的索引；print\_tree()函数用来打印二叉树，计算树的高度和每层高度对应的节点数，并按层级打印每个节点，处理对齐以便于可读性；preorder\_traversal()函数用来前序遍历二叉树，通过递归的算法，从根节点开始，依次访问左子树和右子树；inorder\_traversal()用来中序遍历二叉树，通过递归的算法，先访问左子树，再访问当前节点，最后访问右子树；postorder\_traversal()用来后序遍历二叉树，通过递归的算法，先访问左子树和右子树，最后访问当前节点。

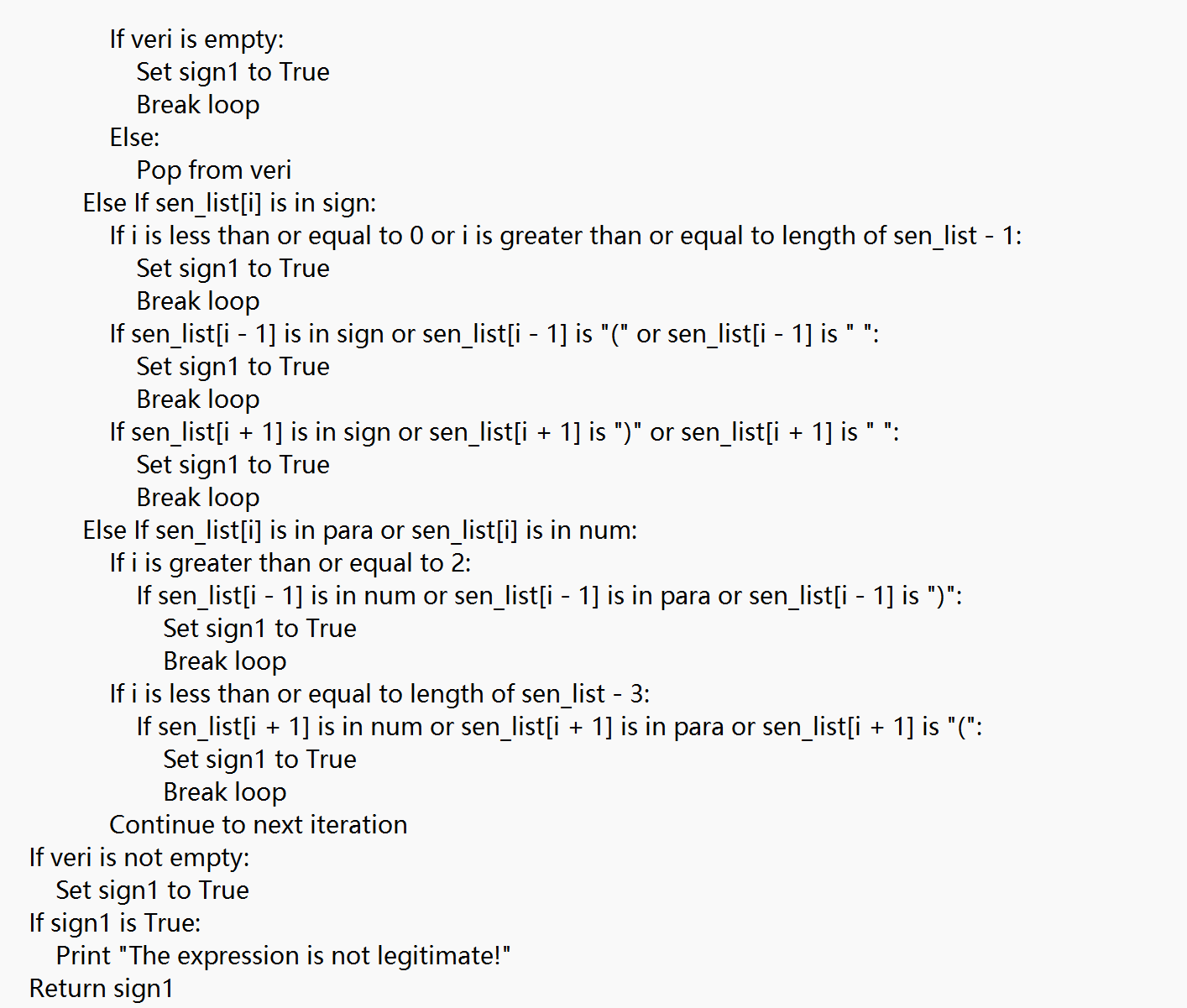
伪代码如图2.5所示。

图2.5 二叉树类设计伪代码

1. 合法性检验设计

check 函数用于验证一个算术表达式的语法正确性，它检查括号是否匹配、字符是否有效以及运算符是否放置正确。首先初始化一个布尔标志sign1，表示表达式是否无效，一个Stack 类的实例，用于跟踪开括号，以确保它们配对，列表bra, sign, num和 para分别定义了有效的括号、运算符、数字和变量。函数遍历 sen\_list 中的每个字符，如果表达式长度小于或等于 2，则无效，这样的表达式太短，无法形成有效的表达式；检查当前字符是否属于有效的运算符、数字、变量或括号，如果不属于这些类别，则也表示表达式无效；检查左括号是否位于表达式的末尾，是否有有效字符作为前导，并且后面不是紧跟着运算符有效的左括号被压入栈中以进行跟踪，否则无效；检查右括号是否是第一个字符，是否后面跟有有效字符，前面是否跟有运算符，如果栈为空时尝试关闭括号，表示括号不匹配；确保运算符没有出现在表达式的开头或结尾，并且前后不是紧跟着其他运算符或括号，运算符应被有效字符包围；确保变量和数字的放置正确，它们不能直接跟在其他数字、变量或右括号后面，并且它们后面必须跟有有效字符；最终检查栈中是否还有未匹配的括号，若有，则表示括号未完全匹配，也不合法。如果 sign1 被设置为 True，表示表达式无效，打印错误信息。返回 sign1，True 表示表达式无效，False 表示有效。

伪代码如图2.6所示。

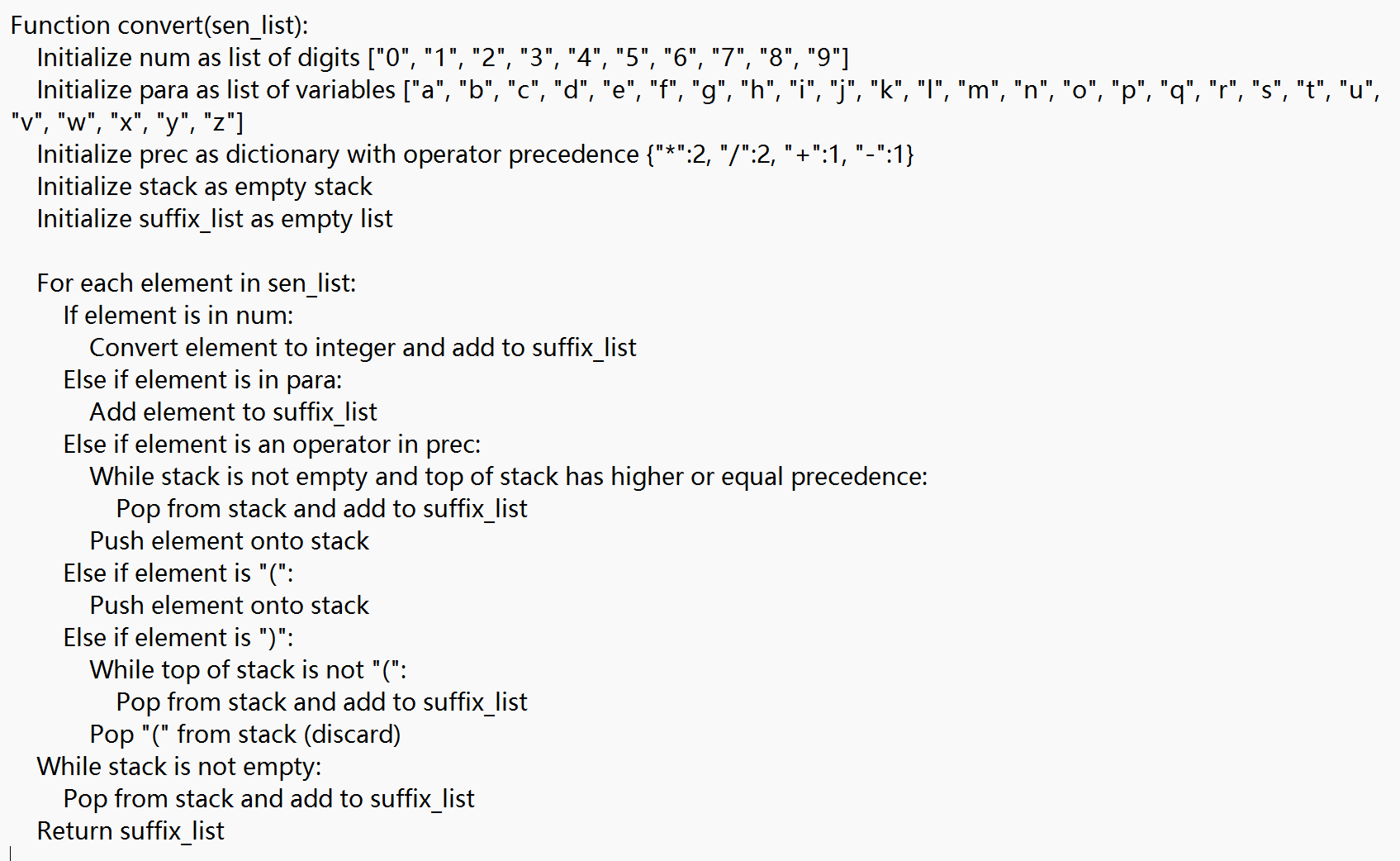
图2.6 合法性检验设计伪代码

1. 中缀表达式转后缀表达式设计

列表bra, num和 para分别定义了有效的括号、数字和变量，字典prec:定义操作符的优先级，乘法 \* 和除法 / 的优先级为 2，加法 + 和减法 - 的优先级为 1，栈 s用于存储操作符和括号，suffix\_list:用于存储转换后的后缀表达式。

遍历输入列表，如果当前元素是数字，将其转换为整数并添加到 suffix\_list 中；如果当前元素是未知数，直接添加到 suffix\_list 中；如果当前元素是操作符，要考虑其前后的符号，考虑括号和优先级，先分析栈s是否为空，如果栈 s 为空，则表示目前为无等待符号，直接将操作符压入栈中，如果栈顶元素是左括号，也将当前操作符压入栈中，如果栈顶元素是操作符 j，比较当前操作符与 j 的优先级，如果当前操作符优先级更高，将其压入栈中，如果当前操作符优先级不高，弹出栈顶操作符（优先级较高的操作符），将其添加到suffix\_list中，然后将当前操作符压入栈中；如果当前元素是左括号，将其压入栈中；如果当前元素是右括号，弹出栈中的元素直到遇到左括号，并将这些元素添加到 suffix\_list 中。当所有元素都处理完毕后，将栈中剩余的操作符依次弹出并添加到 suffix\_list 中。

伪代码如图2.7所示。

图2.7 中缀表达式转后缀表达式设计伪代码

1. 构建二叉树设计

列表sign定义了操作符，列表done用于存储已赋值的索引，current\_idx是当前插入节点的索引，初始化为 0。

先从 suffix 列表中弹出最后一个值作为树的根节点，并插入到树中，后进入while循环结构，处理后缀表达式中的每个元素，当当前元素为操作符时，计算当前操作符的左右子节点索引 left\_child\_idx 和 right\_child\_idx，根据 done 列表判断左右子节点是否已经被使用，如果左子节点未被使用，则设置 current\_idx 为左子节点的索引，如果左子节点已被使用，则设置 current\_idx 为右子节点的索引，将操作符插入到树中，并打印插入过程，记录当前节点索引为已使用；当当前值为操作数时，同样计算当前操作数的左右子节点索引，判断左右子节点是否已经被使用，如果左子节点未被使用，则将其作为新节点的索引，如果左子节点已被使用，则寻找右子节点，若右子节点也已被使用，则回溯到父节点，继续寻找合适的插入位置，将操作数插入到树中，并打印插入过程，记录新节点索引为已使用。最终返回返回构建完成的 tree 对象。

伪代码如图2.8所示。

图2.8 构建二叉树设计伪代码

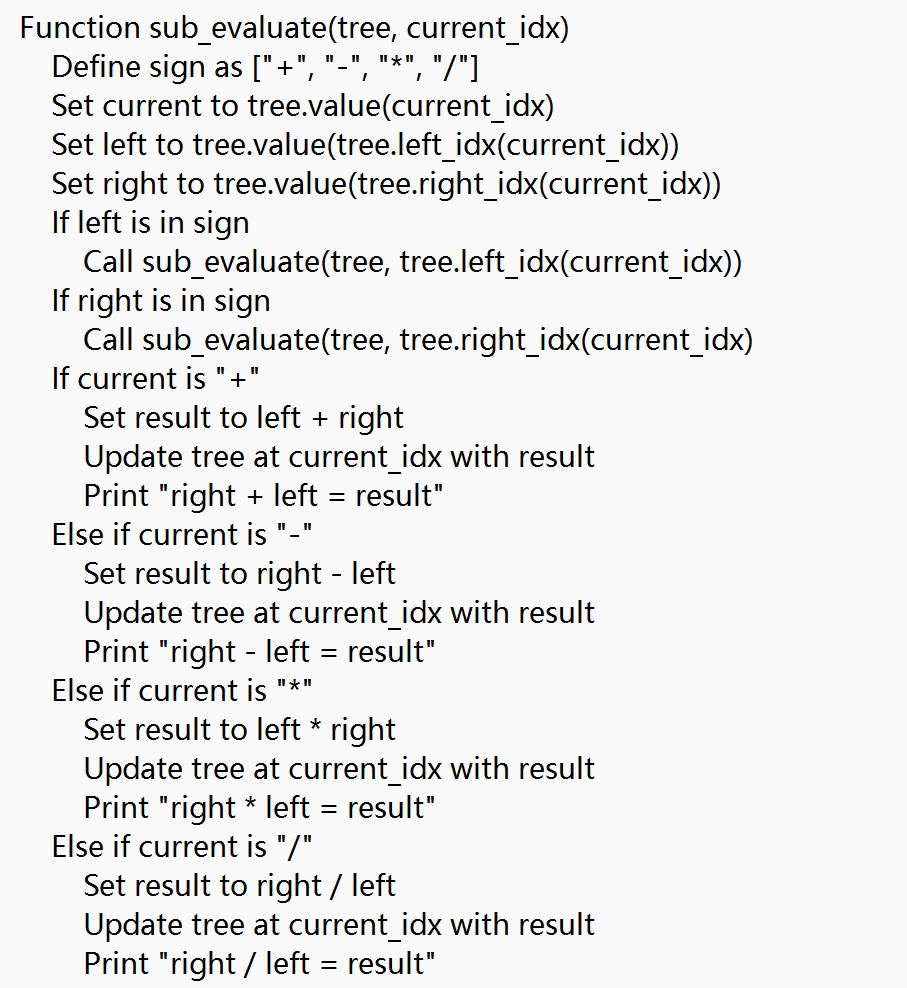
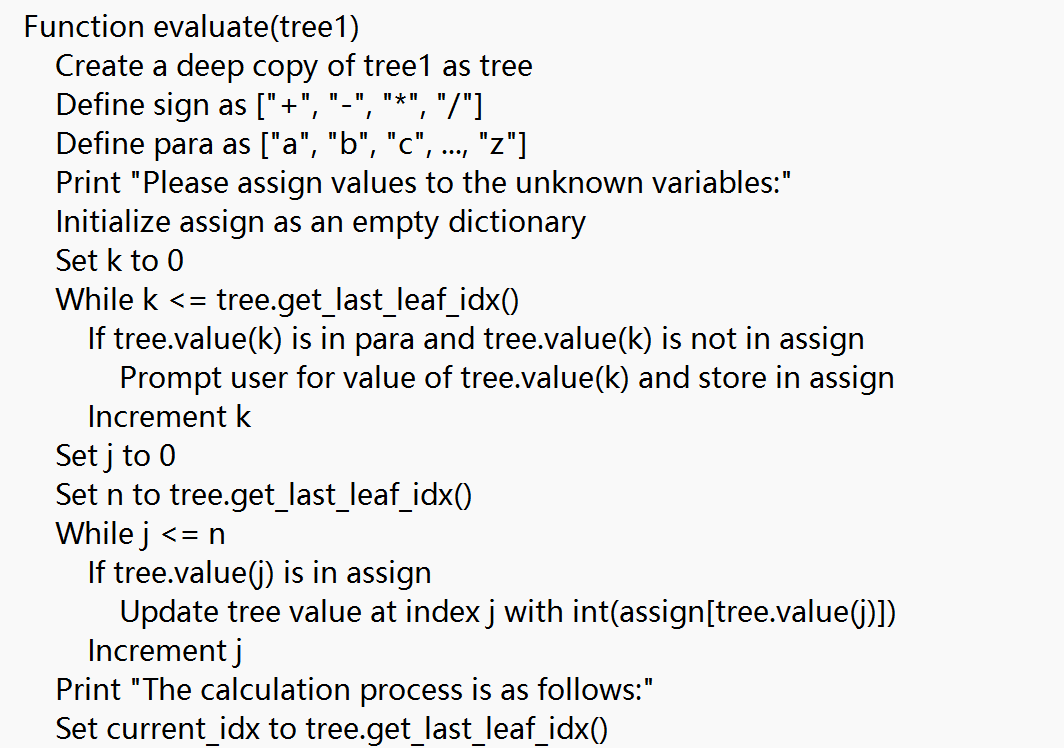
1. 计算二叉树

考虑到主函数中会多次引用该函数对二叉树中的未知数反复赋值进行计算，所以使用 copy.deepcopy 复制输入树 tree1，确保操作不会改变原始树。通过遍历二叉树找到未赋值的未知数，提示用户为每个未知变量赋值，并使用 assign 字典保存变量及其对应值。再遍历树中的所有节点，将用户输入的变量值更新到树中。

Evaluate函数计算二叉树时，根据后缀算数表达式的特点，及所构成二叉树的特点，从叶节点向根节点进行计算。先找到最后一个叶节点的索引作为current\_idx，从该节点索引向上遍历树，如果当前节点不是操作符，则检查其值是否为 None 或已被赋值，如果未被赋值，则索引减一，直到找到一个操作符，如果当前节点是操作符，则计算其左右子节点的值，如果子节点存在变量或操作符，递归调用 sub\_evaluate 计算其子节点值后再执行计算并更新当前节点的值，遍历树的过程中处理每个节点，直到计算完成。

Sub\_evaluate函数中current\_idx是evaluate函数中传入的当前节点的索引，读取当前节点的值 current，以及其左右子节点的值 left 和 right，因为还存在子节点的子节点依旧是操作符的情况，所以如果 left 或 right 是操作符，则递归调用 sub\_evaluate 计算它们的值，否则直接根据 current 的操作符，进行相应的运算并更新当前节点的值。

伪代码如图2.9所示。



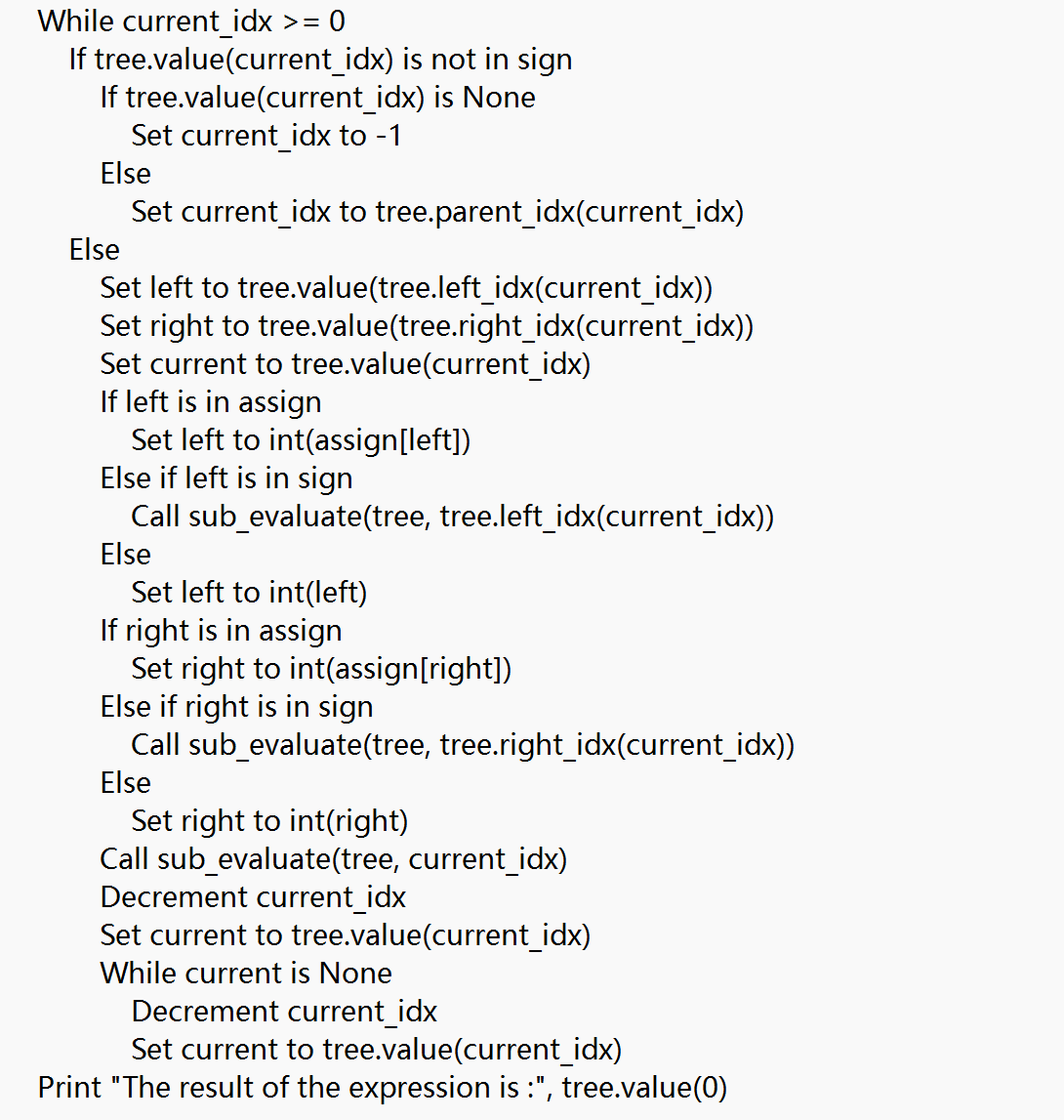


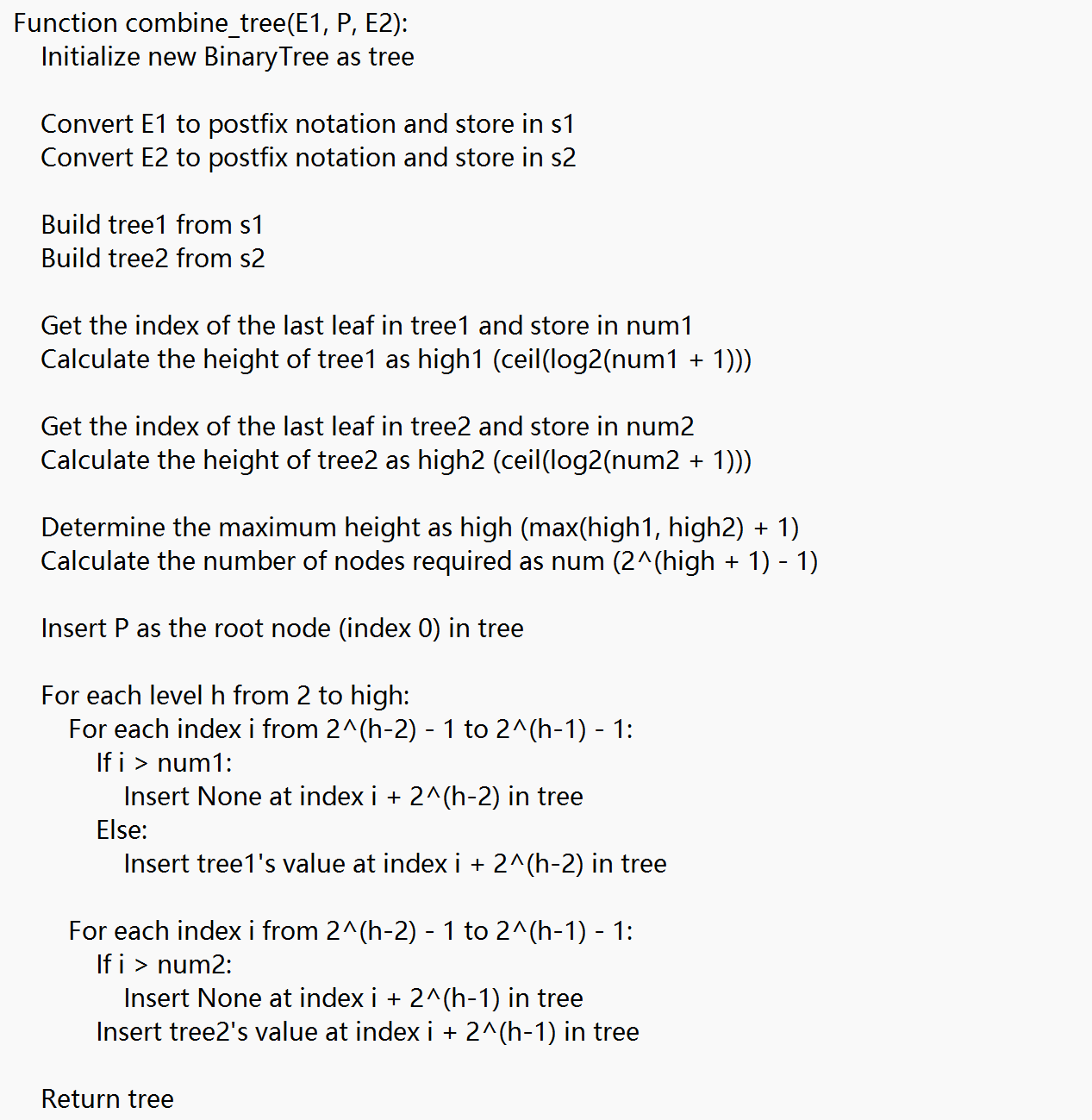
图2.9 计算二叉树设计伪代码

#### 3.3.2 拓展程序

扩展程序中，先将两个子式E1和E2分别从中缀算术式化成后缀表达式，后根据后缀表达式构建二叉树，再通过加入根节点P，将E1和E2的二叉树的根节点作为其左右节点依次连接。考虑到减法和除法存在操作数的前后顺序，在之前的evaluate函数中都是右节点操作左节点，这里将E1作为右子树，E2作为左子树。最后通过evaluate函数计算合二叉树得到计算结果。

Combine函数中，创建一个新的空二叉树 tree作为合二叉树，通过convert函数将表达式E1和E2转换为后缀表示法，结果分别存储在 s1 和 s2 中，使用后缀表示法s1和s2分别构建tree1和tree2。计算这两个树的高度high1、high2和节点数num1、num2，high是合二叉树的高度，是tree1和tree2高度的最大值加一， num 是新二叉树的节点总数。先将运算符 P 插入到新树的根节点位置，从第二层开始，遍历新树的每一层，根据树的高度和节点索引插入 tree1 和 tree2 的节点，对于每一层，首先将 tree1 的节点插入到新树的右子树部分，然后将 tree2 的节点插入到新树的左子树部分，如果 tree1 或 tree2 中没有足够的节点，则在相应位置插入 None。最终返回新的合二叉树tree。

合并二叉树设计伪代码如图2.10所示。

图2.10 合并二叉树设计伪代码

## 调试分析

（1）过程中的困难与解决：在中缀表达式的合法性检验中经常遇到缺漏考虑某种情况出现报错，经过多次尝试和补充后完善。在计算二叉树时，最开始没有考虑到多括号，含多个平等运算顺序子式的情况，总是报错显示符号不能进行运算，经过调试发现是在当前坐标运动到根节点时发生错误，又经过自己根据算法的多次推演后发现问题出现在不停的返回父节点索引导致其实只计算了其中一个子树，漏掉了同高度的符号节点，最终通过将符号节点索引减一和None节点减一来寻找下一节点的方式，解决了这个问题，使二叉树能够进行正确运算。

（2）算法的时间空间复杂度分析：check 函数的时间复杂度:O(n)，其中 n 是表达式的长度，这个函数主要遍历一次表达式，检查括号配对、操作符和操作数的位置；空间复杂度为 O(n)，主要用于存储操作数和括号在栈中的状态。 convert 函数的时间复杂度为O(n)，其中 n 是表达式的长度，此函数也遍历一次表达式，通过栈来管理操作符和括号，并生成后缀表达式；空间复杂度为O(n)，用于存储后缀表达式和栈中的操作符。 build\_tree 函数的时间复杂度为O(n)，其中 n 是后缀表达式的长度，函数对后缀表达式进行一次遍历，并根据操作符和操作数构建二叉树；空间复杂度为 O(n)，用于存储二叉树的节点以及后缀表达式。sub\_evaluate 函数的时间复杂度为O(n)，其中n是树的节点数，这个函数进行递归计算，遍历每个节点以进行计算；空间复杂度为 O(h)，其中 h 是树的高度，由于递归调用的堆栈深度与树的高度成正比。evaluate 函数的时间复杂度为O(n)，其中 n 是树的节点数，此函数首先将树的叶子节点的值进行替换，然后执行计算；空间复杂度: O(n)，主要用于存储树的节点以及赋值的字典。

（3）算法讨论：涉及while循环结构的算法都可以化为递归函数算法，比如计算二叉树时，可以直接通过从根节点进行sub\_evaluate，不停的向下进行子节点的计算，最终获得结果。

## 五、用户手册

根据提示输入中缀表达式，按回车键进行后缀表达式等多项题目需求输出，再根据提示给未知变量赋值，按回车键进行计算。最终根据提示进行输入选择再次赋值、再次输入表达式或者退出程序。

## 六、测试数据及测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | 测试目的 | 正确输出 | 程序输出 | 当前状态 |
| 3+2+ | 程序能否检测出中缀表达式的不合法输入 | The expression is not legitimate! | 图2.11.1 | 通过 |
| 8+3+2-1\*1 | 程序能否处理简单中缀表达式 | 12 | 图2.11.2 | 通过 |
| a+3\*2-1 | 程序能否正确进行赋值运算 | 8 | 图2.11.3 | 通过 |
| 2\*(a-b)+8/2 | 程序能否处理较复杂中缀表达式 | 12.0 | 图2.11.4 | 通过 |
| (1+3\*a)+(2-b/3) | 拓展程序能否正确处理两个分式计算 | 7.0 | 图1.11.5 | 通过 |

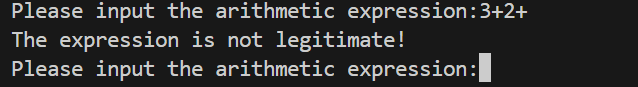


图2.11.1 用例1输出

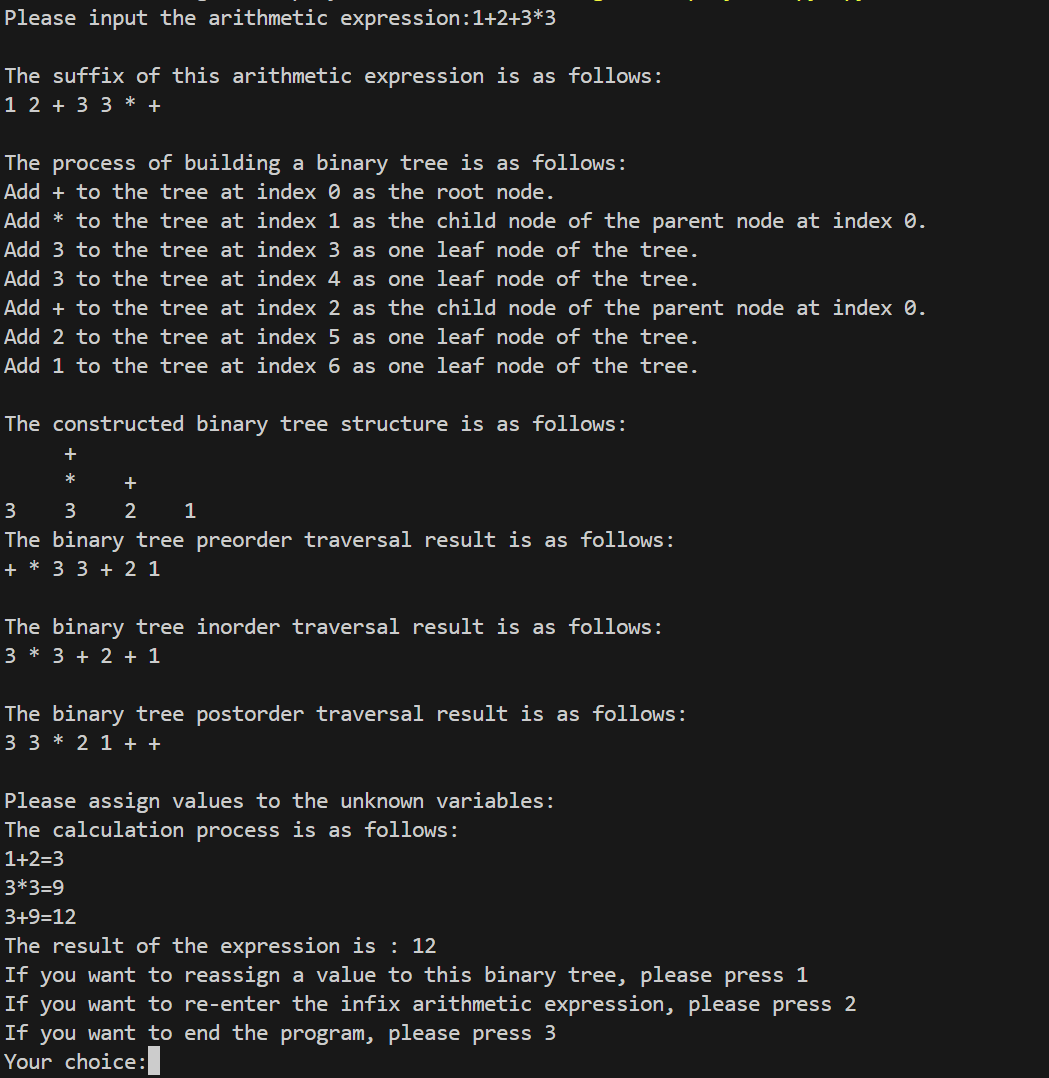
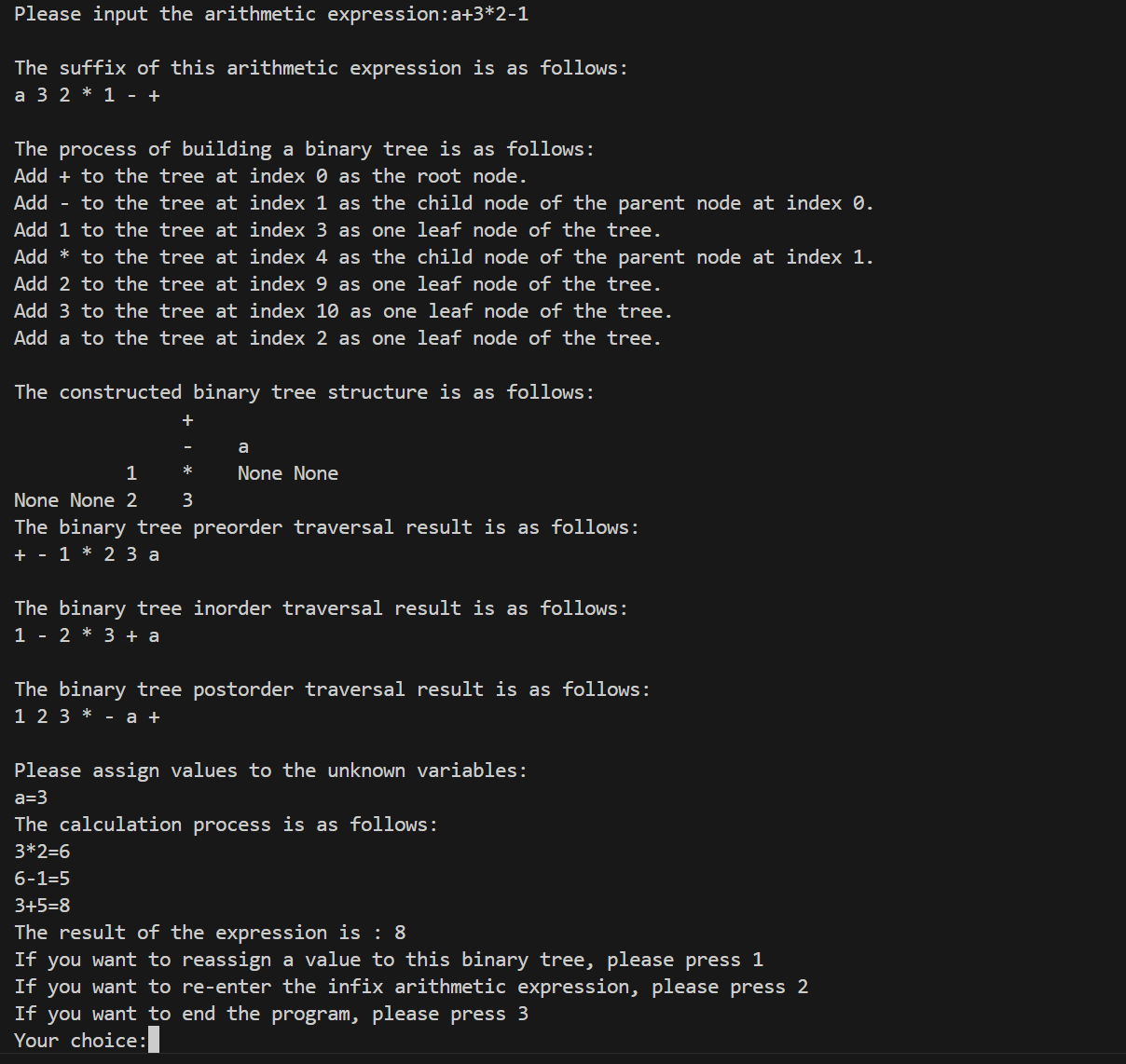


图2.11.2 用例2输出

图2.1013 用例3输出

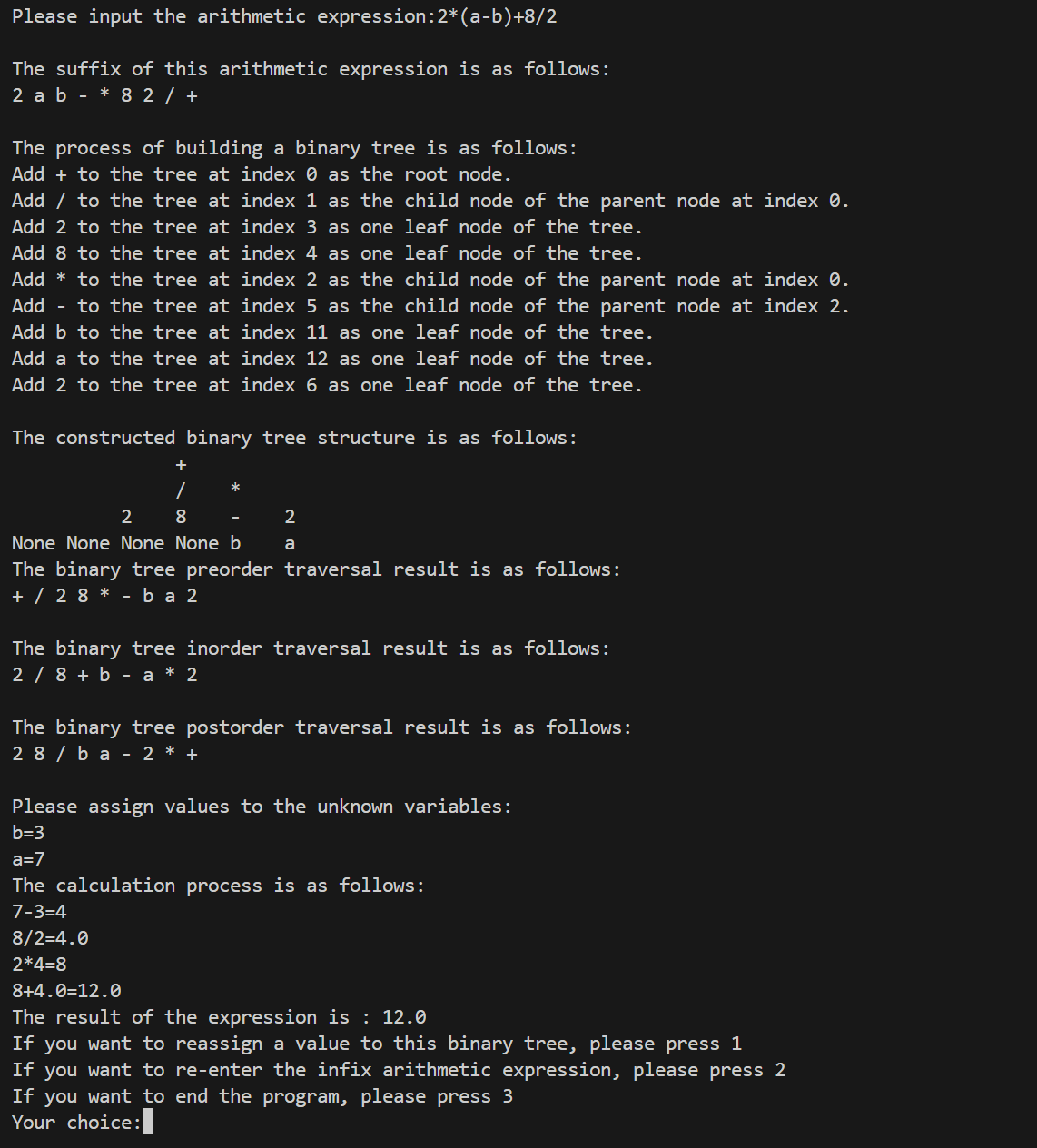


图2.11.4 用例4输出

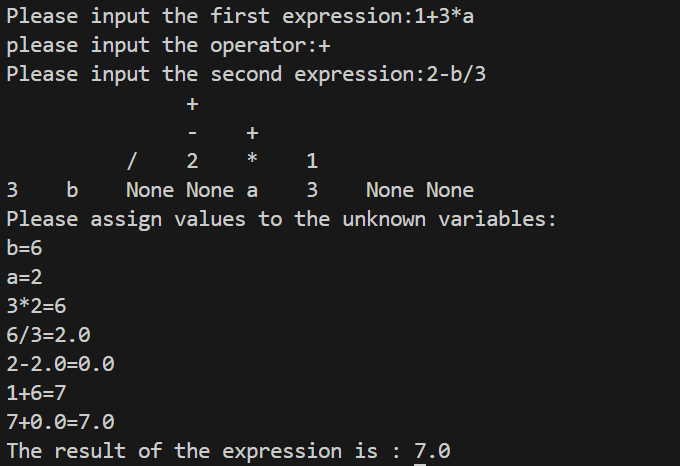


图2.11.5 用例5输出

## 七、源程序清单

（1）基本功能程序

import math

import copy

class Stack():

    def \_\_init\_\_(self):

        self.data=[]

        self.top=-1

    def is\_empty(self):

        return self.top<0

    def size(self):

        return self.top+1

    def top\_value(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            return self.data[self.top]

    def push(self,value):

        self.top+=1

        self.data.append(value)

    def pop(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            value=self.data.pop()

            self.top-=1

            return value

class BinaryTree:

    def \_\_init\_\_(self):

        self.data = []

        self.last\_leaf\_idx = -1

    def value(self, idx):

        if idx >= len(self.data):

            return None

        return self.data[idx]

    def change\_value(self, idx, value):

        if idx < len(self.data):

            self.data[idx] = value

        else:

            raise IndexError("Index out of range")

    def left\_idx(self, idx):

        return 2 \* idx + 1

    def right\_idx(self, idx):

        return 2 \* idx + 2

    def parent\_idx(self, idx):

        return (idx - 1) // 2

    def get\_last\_leaf\_idx(self):

        return self.last\_leaf\_idx

    def insert(self, idx, value):

        while idx >= len(self.data):

            self.data.append(None)

        self.data[idx] = value

        self.last\_leaf\_idx = max(self.last\_leaf\_idx, idx)

    def print\_tree(self):

        n=self.last\_leaf\_idx

        high=math.ceil(math.log((n+1),2))

        max\_num=2\*\*(high-1)

        for i in range(1,high):

            num=2\*\*(i-1)

            for k in range((max\_num-num)//2):

                print("     ",end="")

            for j in range(2\*\*(i-1)-1,2\*\*i-1):

                if self.data[j]==None:

                    print(self.data[j],end=" ")

                else:

                    print(self.data[j],end="    ")

            print("\n",end="")

        for i in range(2\*\*(high-1)-1,n+1):

            if self.data[i]==None:

                print(self.data[i],end=" ")

            else:

                print(self.data[i],end="    ")

    def preorder\_traversal(self):

        def traverse(idx):

            if idx >= len(self.data) or self.data[idx] is None:

                return []

            result = [self.data[idx]]

            result += traverse(self.left\_idx(idx))

            result += traverse(self.right\_idx(idx))

            return result

        return traverse(0)

    def inorder\_traversal(self):

        def traverse(idx):

            if idx >= len(self.data) or self.data[idx] is None:

                return []

            result = traverse(self.left\_idx(idx))

            result.append(self.data[idx])

            result += traverse(self.right\_idx(idx))

            return result

        return traverse(0)

    def postorder\_traversal(self):

        def traverse(idx):

            if idx >= len(self.data) or self.data[idx] is None:

                return []

            result = traverse(self.left\_idx(idx))

            result += traverse(self.right\_idx(idx))

            result.append(self.data[idx])

            return result

        return traverse(0)

#Legitimacy verification

#check if the parentheses are complete

def check(sen\_list):

    sign1=False

    veri=Stack()

    bra=["(",")"]

    sign=["+","-","\*","/"]

    num=["0","1","2","3","4","5","6","7","8","9"]

    para=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

    for i in range(len(sen\_list)):

        if len(sen\_list)<=2:

            sign1=True

            break

        elif sen\_list[i] not in sign and sen\_list[i] not in num and sen\_list[i] not in para and sen\_list[i] not in bra:

            sign1=True

            break

        else:

            if sen\_list[i]=="(":

                if i+1>=len(sen\_list)-1:

                    sign1=True

                    break

                if i-1>=0 and  sen\_list[i-1] not in sign:

                    sign1=True

                    break

                if i+1<len(sen\_list) and sen\_list[i+1] in sign:

                    sign1=True

                    break

                veri.push(sen\_list[i])

            elif sen\_list[i]==")":

                if i-1<0:

                    sign1=True

                    break

                if i+1<len(sen\_list) and sen\_list[i+1] not in sign:

                    sign1=True

                    break

                if i-1>=0 and sen\_list[i-1] in sign:

                    sign1=True

                    break

                if veri.is\_empty():

                    sign1=True

                    break

                else:

                    veri.pop()

            elif sen\_list[i] in sign:

                if i<=0 or i>=len(sen\_list)-1:

                    sign1=True

                    break

                if sen\_list[i-1] in sign or sen\_list[i-1]=="(" or sen\_list[i-1]==" ":

                    sign1=True

                    break

                if sen\_list[i+1] in sign or sen\_list[i+1]==")" or sen\_list[i+1]==" ":

                    sign1=True

                    break

            elif sen\_list[i] in para or sen\_list[i] in num:

                if i>=2:

                    if sen\_list[i-1] in num or sen\_list[i-1] in para or sen\_list[i-1]==")":

                        sign1=True

                        break

                if i<=len(sen\_list)-3:

                    if sen\_list[i+1] in num or sen\_list[i+1] in para or sen\_list[i+1]=="(":

                        sign1=True

                        break

                else:

                    continue

    if veri.is\_empty()==False:

        sign1=True

    if sign1==True:

        print("The expression is not legitimate!")

    return sign1

#convert the expression to the postfix expression

def convert(sen\_list):

    num=["0","1","2","3","4","5","6","7","8","9"]

    para=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

    prec={"\*":2,"/":2,"+":1,"-":1}

    s=Stack()

    suffix\_list=[]

    for i in range(len(sen\_list)):

        if sen\_list[i] in num:#if the current element is a number,turn it into an int form element here,which is convenient for subsequent calculations

            val=int(sen\_list[i])

            suffix\_list.append(val)

            continue

        elif sen\_list[i] in para:

            suffix\_list.append(sen\_list[i])

            continue

        else:

            if sen\_list[i] in prec:

                if s.is\_empty()==True:

                    s.push(sen\_list[i])

                else:

                    j=s.top\_value()

                    if j=="(":

                        s.push(sen\_list[i])

                    else:

                        if prec[sen\_list[i]]>prec[j]:

                            s.push(sen\_list[i])

                        elif prec[sen\_list[i]]<=prec[j]:

                            s.pop()

                            suffix\_list.append(j)

                            s.push(sen\_list[i])

            elif sen\_list[i]=="(":

                 s.push(sen\_list[i])

            elif sen\_list[i]==")":

                j=s.pop()

                while j!="(":

                    suffix\_list.append(j)

                    j=s.pop()

    while not s.is\_empty():

        j=s.pop()

        suffix\_list.append(j)

    return suffix\_list

def build\_tree(suffix):

    print("\nThe process of building a binary tree is as follows:")

    sign=["+","-","\*","/",]

    tree = BinaryTree()

    done=[]

    current\_idx=0

    current\_value=suffix.pop(-1)

    tree.insert(current\_idx,current\_value)

    print(f"Add {current\_value} to the tree at index {current\_idx} as the root node. ")

    done.append(current\_idx)

    while len(suffix)>0:

        current\_value=suffix.pop(-1)

        if current\_value in sign:

            left\_child\_idx=tree.left\_idx(current\_idx)

            right\_child\_idx=tree.right\_idx(current\_idx)

            if left\_child\_idx not in done:

                current\_idx=left\_child\_idx

            else:

                if right\_child\_idx not in done:

                    current\_idx=right\_child\_idx

            done.append(current\_idx)

            tree.insert(current\_idx,current\_value)

            print(f"Add {current\_value} to the tree at index {current\_idx} as the child node of the parent node at index {tree.parent\_idx(current\_idx)}.")

        else:

            left\_child\_idx=tree.left\_idx(current\_idx)

            right\_child\_idx=tree.right\_idx(current\_idx)

            if left\_child\_idx not in done:

                next\_idx=left\_child\_idx

            else:

                if right\_child\_idx not in done:

                    next\_idx=right\_child\_idx

                    current\_idx=tree.parent\_idx(current\_idx)

                else:

                    while right\_child\_idx in done:

                        next\_idx=tree.parent\_idx(next\_idx)

                        right\_child\_idx=tree.right\_idx(next\_idx)

                    next\_idx=right\_child\_idx

            done.append(next\_idx)

            tree.insert(next\_idx,current\_value)

            print(f"Add {current\_value} to the tree at index {next\_idx} as one leaf node of the tree.")

    return tree

def sub\_evaluate(tree,current\_idx):

    sign=["+","-","\*","/",]

    current=tree.value(current\_idx)

    left=tree.value(tree.left\_idx(current\_idx))

    right=tree.value(tree.right\_idx(current\_idx))

    if left in sign:

        sub\_evaluate(tree,tree.left\_idx(current\_idx))

    if right in sign:

        sub\_evaluate(tree,tree.right\_idx(current\_idx))

    if current=="+":

        tree.change\_value(current\_idx,left+right)

        print(f"{right}+{left}={left+right}")

    elif current=="-":

        tree.change\_value(current\_idx,right-left)

        print(f"{right}-{left}={right-left}")

    elif current=="\*":

        tree.change\_value(current\_idx,left\*right)

        print(f"{right}\*{left}={left\*right}")

    elif current=="/":

        tree.change\_value(current\_idx,right/left)

        print(f"{right}/{left}={right/left}")

def evaluate(tree1):

    tree = copy.deepcopy(tree1)

    sign=["+","-","\*","/",]

    para=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

    print("\nPlease assign values to the unknown variables:")

    k=0

    assign={}

    while k<tree.get\_last\_leaf\_idx()+1:

        if tree.value(k) in para and tree.value(k) not in assign.keys():

            info=input(f"{tree.value(k)}=")

            assign[tree.value(k)]=info

        k+=1

    j=0

    n=tree.get\_last\_leaf\_idx()

    while j<=n:

        if tree.value(j) in assign.keys():

            tree.change\_value(j,int(assign[tree.value(j)]))

        j+=1

    #find the index of the last leaf

    print("The calculation process is as follows:")

    current\_idx=tree.get\_last\_leaf\_idx()

    while current\_idx>=0:

        if tree.value(current\_idx) not in sign:#not a operator

            if tree.value(current\_idx)==None:

                current\_idx-=1

            else:

                current\_idx=tree.parent\_idx(current\_idx)#find the next one

        else:#a operator

            left=tree.value(tree.left\_idx(current\_idx))

            right=tree.value(tree.right\_idx(current\_idx))

            current=tree.value(current\_idx)

            if left in assign.keys():

                left=int(assign[left])

            elif left in sign:

                sub\_evaluate(tree,tree.left\_idx(current\_idx))

            else:

                left=int(left)

            if right in assign.keys():

                right=int(assign[right])

            elif right in sign:

                sub\_evaluate(tree,tree.right\_idx(current\_idx))

            else:

                right=int(right)

            sub\_evaluate(tree,current\_idx)

            current\_idx=current\_idx-1

            current=tree.value(current\_idx)

            while current==None:

                current\_idx=current\_idx-1#find the next one

                current=tree.value(current\_idx)#find the next one

    print("The result of the expression is :",tree.value(0))

if \_\_name\_\_=="\_\_main\_\_":

    get=2

    while get==2:

        sen=str(input("Please input the arithmetic expression:"))

        sen\_list=[]

        for i in sen:

            sen\_list.append(str(i))

        if not check(sen\_list):

            suffix = convert(sen\_list)

            print("\nThe suffix of this arithmetic expression is as follows:")

            print(" ".join(map(str, suffix)))

            tree = build\_tree(suffix)

            print("\nThe constructed binary tree structure is as follows:")

            tree.print\_tree()

            print("\nThe binary tree preorder traversal result is as follows:")

            preorder\_result = tree.preorder\_traversal()

            print(" ".join(map(str, preorder\_result)))

            print("\nThe binary tree inorder traversal result is as follows:")

            inorder\_result = tree.inorder\_traversal()

            print(" ".join(map(str, inorder\_result)))

            print("\nThe binary tree postorder traversal result is as follows:")

            postorder\_result = tree.postorder\_traversal()

            print(" ".join(map(str, postorder\_result)))

            evaluate(tree)

            print("If you want to reassign a value to this binary tree, please press 1\nIf you want to re-enter the infix arithmetic expression, please press 2\nIf you want to end the program, please press 3")

            choose=int(input("Your choice:"))

            while choose==1:

                evaluate(tree)

            get=choose

（2）拓展程序

import math

import copy

class Stack():

    def \_\_init\_\_(self):

        self.data=[]

        self.top=-1

    def is\_empty(self):

        return self.top<0

    def size(self):

        return self.top+1

    def top\_value(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            return self.data[self.top]

    def push(self,value):

        self.top+=1

        self.data.append(value)

    def pop(self):

        if self.is\_empty():

            raise KeyError("Stack is empty")

        else:

            value=self.data.pop()

            self.top-=1

            return value

class BinaryTree:

    def \_\_init\_\_(self):

        self.data = []

        self.last\_leaf\_idx = -1

    def value(self, idx):

        if idx >= len(self.data):

            return None

        return self.data[idx]

    def change\_value(self, idx, value):

        if idx < len(self.data):

            self.data[idx] = value

        else:

            raise IndexError("Index out of range")

    def left\_idx(self, idx):

        return 2 \* idx + 1

    def right\_idx(self, idx):

        return 2 \* idx + 2

    def parent\_idx(self, idx):

        return (idx - 1) // 2

    def get\_last\_leaf\_idx(self):

        return self.last\_leaf\_idx

    def insert(self, idx, value):

        while idx >= len(self.data):

            self.data.append(None)

        self.data[idx] = value

        self.last\_leaf\_idx = max(self.last\_leaf\_idx, idx)

    def print\_tree(self):

        n=self.last\_leaf\_idx

        high=math.ceil(math.log((n+1),2))

        max\_num=2\*\*(high-1)

        for i in range(1,high):

            num=2\*\*(i-1)

            for k in range((max\_num-num)//2):

                print("     ",end="")

            for j in range(2\*\*(i-1)-1,2\*\*i-1):

                if self.data[j]==None:

                    print(self.data[j],end=" ")

                else:

                    print(self.data[j],end="    ")

            print("\n",end="")

        for i in range(2\*\*(high-1)-1,n+1):

            if self.data[i]==None:

                print(self.data[i],end=" ")

            else:

                print(self.data[i],end="    ")

def check(sen\_list):

    sign1=False

    veri=Stack()

    bra=["(",")"]

    sign=["+","-","\*","/",]

    num=["0","1","2","3","4","5","6","7","8","9"]

    para=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

    for i in range(len(sen\_list)):

        if sen\_list[i] not in sign and sen\_list[i] not in num and sen\_list[i] not in para and sen\_list[i] not in bra:

            sign1=True

            break

        else:

            if sen\_list[i]=="(":

                if i-1>=0 and  sen\_list[i-1] not in sign:

                    sign1=True

                    break

                if i+1<len(sen\_list) and sen\_list[i+1] in sign:

                    sign1=True

                    break

                veri.push(sen\_list[i])

            elif sen\_list[i]==")":

                if i+1<len(sen\_list) and sen\_list[i+1] not in sign:

                    sign1=True

                    break

                if i-1>=0 and sen\_list[i-1] in sign:

                    sign1=True

                    break

                if veri.is\_empty():

                    sign1=True

                    break

                else:

                    veri.pop()

            elif sen\_list[i] in sign :

                if sen\_list[i-1] in sign or sen\_list[i-1]=="(":

                    sign1=True

                    break

                if sen\_list[i+1] in sign or sen\_list[i+1]==")":

                    sign1=True

                    break

    if veri.is\_empty()==False:

        sign1=True

    if sign1==True:

        print("The expression is not legitimate!")

    return sign1

def convert(sen\_list):

    num=["0","1","2","3","4","5","6","7","8","9"]

    para=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

    prec={"\*":2,"/":2,"+":1,"-":1}

    s=Stack()

    suffix\_list=[]

    for i in range(len(sen\_list)):

        if sen\_list[i] in num:#if the current element is a number,turn it into an int form element here,which is convenient for subsequent calculations

            val=int(sen\_list[i])

            suffix\_list.append(val)

            continue

        elif sen\_list[i] in para:

            suffix\_list.append(sen\_list[i])

            continue

        else:

            if sen\_list[i] in prec:

                if s.is\_empty()==True:

                    s.push(sen\_list[i])

                else:

                    j=s.top\_value()

                    if j=="(":

                        s.push(sen\_list[i])

                    else:

                        if prec[sen\_list[i]]>prec[j]:

                            s.push(sen\_list[i])

                        elif prec[sen\_list[i]]<=prec[j]:

                            s.pop()

                            suffix\_list.append(j)

                            s.push(sen\_list[i])

            elif sen\_list[i]=="(":

                 s.push(sen\_list[i])

            elif sen\_list[i]==")":

                j=s.pop()

                while j!="(":

                    suffix\_list.append(j)

                    j=s.pop()

    while not s.is\_empty():

        j=s.pop()

        suffix\_list.append(j)

    return suffix\_list

def build\_tree(suffix):

    sign=["+","-","\*","/",]

    tree = BinaryTree()

    done=[]

    current\_idx=0

    current\_value=suffix.pop(-1)

    tree.insert(current\_idx,current\_value)

    done.append(current\_idx)

    while len(suffix)>0:

        current\_value=suffix.pop(-1)

        if current\_value in sign:

            left\_child\_idx=tree.left\_idx(current\_idx)

            right\_child\_idx=tree.right\_idx(current\_idx)

            if left\_child\_idx not in done:

                current\_idx=left\_child\_idx

            else:

                if right\_child\_idx not in done:

                    current\_idx=right\_child\_idx

            done.append(current\_idx)

            tree.insert(current\_idx,current\_value)

        else:

            left\_child\_idx=tree.left\_idx(current\_idx)

            right\_child\_idx=tree.right\_idx(current\_idx)

            if left\_child\_idx not in done:

                next\_idx=left\_child\_idx

            else:

                if right\_child\_idx not in done:

                    next\_idx=right\_child\_idx

                else:

                    while right\_child\_idx in done:

                        next\_idx=tree.parent\_idx(next\_idx)

                        right\_child\_idx=tree.right\_idx(next\_idx)

                    next\_idx=right\_child\_idx

            done.append(next\_idx)

            tree.insert(next\_idx,current\_value)

    return tree

def combine\_tree(E1,P,E2):#put E2 at the left

    tree=BinaryTree()

    s1=convert(E1)

    s2=convert(E2)

    tree1=build\_tree(s1)

    tree2=build\_tree(s2)

    num1=tree1.get\_last\_leaf\_idx()

    high1=math.ceil(math.log((num1+1),2))

    num2=tree2.get\_last\_leaf\_idx()

    high2=math.ceil(math.log((num2+1),2))

    high=max(high1,high2)+1

    num=2\*\*(high+1)-1

    tree.insert(0,P)#the root node(high=1)

    for h in range(2,high+1):

        for i in range(2\*\*(h-2)-1,2\*\*(h-1)-1):

            if i>num1:

                tree.insert(i+2\*\*(h-2),None)

            else:

                tree.insert(i+2\*\*(h-2),tree1.value(i))

        for i in range(2\*\*(h-2)-1,2\*\*(h-1)-1):

            if i>num2:

                tree.insert(i+2\*\*(h-1),None)

            tree.insert(i+2\*\*(h-1),tree2.value(i))

    return tree

def sub\_evaluate(tree,current\_idx):

    sign=["+","-","\*","/",]

    current=tree.value(current\_idx)

    left=tree.value(tree.left\_idx(current\_idx))

    right=tree.value(tree.right\_idx(current\_idx))

    if left in sign:

        sub\_evaluate(tree,tree.left\_idx(current\_idx))

    if right in sign:

        sub\_evaluate(tree,tree.right\_idx(current\_idx))

    if current=="+":

        tree.change\_value(current\_idx,left+right)

        print(f"{right}+{left}={left+right}")

    elif current=="-":

        tree.change\_value(current\_idx,right-left)

        print(f"{right}-{left}={right-left}")

    elif current=="\*":

        tree.change\_value(current\_idx,left\*right)

        print(f"{right}\*{left}={left\*right}")

    elif current=="/":

        tree.change\_value(current\_idx,right/left)

        print(f"{right}/{left}={right/left}")

def evaluate(tree1):

    tree = copy.deepcopy(tree1)

    sign=["+","-","\*","/",]

    para=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]

    print("\nPlease assign values to the unknown variables:")

    k=0

    assign={}

    while k<tree.get\_last\_leaf\_idx()+1:

        if tree.value(k) in para and tree.value(k) not in assign.keys():

            info=input(f"{tree.value(k)}=")

            assign[tree.value(k)]=info

        k+=1

    j=0

    n=tree.get\_last\_leaf\_idx()

    while j<=n:

        if tree.value(j) in assign.keys():

            tree.change\_value(j,int(assign[tree.value(j)]))

        j+=1

    #find the index of the last leaf

    current\_idx=tree.get\_last\_leaf\_idx()

    while current\_idx>=0:

        if tree.value(current\_idx) not in sign:#not a operator

            if tree.value(current\_idx)==None:

                current\_idx-=1

            else:

                current\_idx=tree.parent\_idx(current\_idx)#find the next one

        else:#a operator

            left=tree.value(tree.left\_idx(current\_idx))

            right=tree.value(tree.right\_idx(current\_idx))

            current=tree.value(current\_idx)

            if left in assign.keys():

                left=int(assign[left])

            elif left in sign:

                sub\_evaluate(tree,tree.left\_idx(current\_idx))

            else:

                left=int(left)

            if right in assign.keys():

                right=int(assign[right])

            elif right in sign:

                sub\_evaluate(tree,tree.right\_idx(current\_idx))

            else:

                right=int(right)

            sub\_evaluate(tree,current\_idx)

            current\_idx=current\_idx-1

            current=tree.value(current\_idx)

            while current==None:

                current\_idx=current\_idx-1#find the next one

                current=tree.value(current\_idx)#find the next one

    print("The result of the expression is :",tree.value(0))

if \_\_name\_\_=="\_\_main\_\_":

    e1=str(input("Please input the first expression:"))

    E1=[]

    for i in e1:

        E1.append(i)

    P=str(input("please input the operator:"))

    e2=str(input("Please input the second expression:"))

    E2=[]

    for i in e2:

        E2.append(i)

    if check(E1)==False and check(E2)==False:

        tree=combine\_tree(E2,P,E1)

        tree.print\_tree()

        evaluate(tree)