电 子 科 技 大 学 实 验 报 告

课程名称： Python语言程序设计及其应用

实验地点： 科A229

指导教师： 张勇

评 分：

完成实验学生信息：

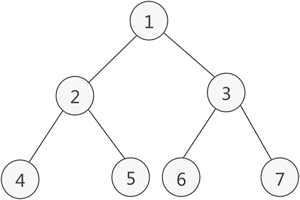
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序** | **姓名** | **学号** | **选课**  **序号** | **贡献**  **百分比(%)** | **备注**  **（主要工作**） |
| 1 | 苏徐涛 | 2022080401011 |  | 30 | 1.提供算法思路和二叉树建立思路  2.提出宽度优先搜索算法实现方法 |
| 2 | 付文亮 | 2022080401004 |  | 30 | 1.分析代码运行结果及代码的优劣问题  2.参与二叉树问题的讨论，提供解题思路 |
| 3 | 傅若山 | 2022080401023 |  | 40 | 1.实现代码设计  2.对代码进行优化改进 |
| 4 |  |  |  |  |  |

1. 学生人数按照任课教师要求限定；
2. 对于“评价、改进、总结和体会”都要认真填写，和其他内容是评价实验成绩的重要参考。

**实验题目名称: 二叉树问题**

1. **实验内容**

**（1）设计一个二叉树类**，构造一颗二叉树存储下面的数据；并用宽度优先搜素搜索节点3.

1. 

**（2）改进该二叉树类：**让其具有插入元素排序功能，插入元素方法名称为add；且设计一个查找方法find。并设计实验验证编程设计是正确。

1. **实验目的**

创建具有插入元素排序功能的二叉树，并实现用宽度优先搜索方法搜索树中元素的功能。

1. **实验过程**

目 录

**[1.](#_Toc133356505)****[实验1：二叉树问题](#_Toc133356505)** [4](#_Toc133356505)

[1.1. 问题分析 4](#_Toc133356506)

[1.2. 系统设计与算法设计 4](#_Toc133356507)

[1.3. 编写程序 4](#_Toc133356508)

[1.4. 运行结果 6](#_Toc133356509)

[1.5. 实验结果分析 7](#_Toc133356510)

[1.6. 优缺点及改进方向 8](#_Toc133356511)

[1.7. 心得体会与总结 8](#_Toc133356512)

**[2.](#_Toc133356513)****[问题重述](#_Toc133356513)** [8](#_Toc133356513)

**[3.](#_Toc133356514)****[对本次实验的设计提出改进意见](#_Toc133356514)** [9](#_Toc133356514)

**[4.](#_Toc133356515)****[附件](#_Toc133356515)** [9](#_Toc133356515)

[附件1.二叉树的程序 9](#_Toc133356516)

[附件2.二叉树程序的输出结果 11](#_Toc133356517)

# **实验1：二叉树问题**

## 问题分析

首先应该了解怎样建立一个二叉树，接着要确定元素以怎样的顺序在二叉树中进行排列，然后写出排序算法使得元素出现在其应处位置，最后用宽度优先搜索方法实现对二叉树中元素的搜索。

## 系统设计与算法设计

功能设计:设计了添加元素功能,使得二叉树能够建立，设计了查找元素功能，使得程序可以在二叉树中寻找一个元素。

算法设计:查找元素功能时采用宽度优先搜索方法，提高搜索效率；使用排序算法，在添加元素时会将新插入元素放到预计出现的位置上。

## 编写程序

class Node:

def \_\_init\_\_(self, val: int, left=None, right=None):

self.left = left

self.right = right

self.val = val

def cmp(ob: Node) -> int:

return ob.val

class Tree:

def \_\_init\_\_(self):

self.tree\_list = [Node(0)]

self.ele\_num = 0

def add(self, val: int):

self.add1(val)

self.sort()

def add1(self, val: int):

self.tree\_list.append(Node(val))

self.ele\_num += 1

if self.ele\_num == 1:

return

if self.ele\_num % 2 == 0:

self.tree\_list[self.ele\_num // 2].left = self.tree\_list[self.ele\_num]

else:

self.tree\_list[self.ele\_num // 2].right = self.tree\_list[self.ele\_num]

# check if val is in the tree

def find(self, val: int):

que = Queue()

if self.ele\_num == 0:

return False

que.push\_front(self.tree\_list[1])

while True:

l = que.size

for i in range(l):

if que.queue[i].val == val:

return True,

for i in range(l):

if que.queue[i].left is None:

for i in range(l, que.size):

if que.queue[i].val == val:

return True

return False

que.push\_front(que.queue[i].left)

if que.queue[i].right is None:

for i in range(l, que.size):

if que.queue[i].val == val:

return True

return False

que.push\_front(que.queue[i].right)

for i in range(l):

que.pop(0)

def clear(self):

self.tree\_list.clear()

self.ele\_num = 0

# sort in the order of the index (the order of bfs)

def sort(self):

copy = self.tree\_list[:]

copy.sort(key=cmp)

self.clear()

self.tree\_list.append(Node(0))

for i in copy[1:]:

self.add1(i.val)

class Queue:

def \_\_init\_\_(self):

self.queue = []

self.size = 0

def push\_front(self, node: Node):

self.queue.append(node)

self.size += 1

def pop(self, ind: int):

self.queue.pop(ind)

self.size -= 1

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

tree = Tree()

print("if you choose to find a value, input 1; else if you choose to add a value, input 2; if you want to exit, input 0 please")

while True:

a = int(input())

if a == 1:

b = int(input())

if tree.find(b):

print("exist")

else:

print("non-exist")

elif a == 2:

tree.add(int(input()))

print("succeed")

elif a == 0:

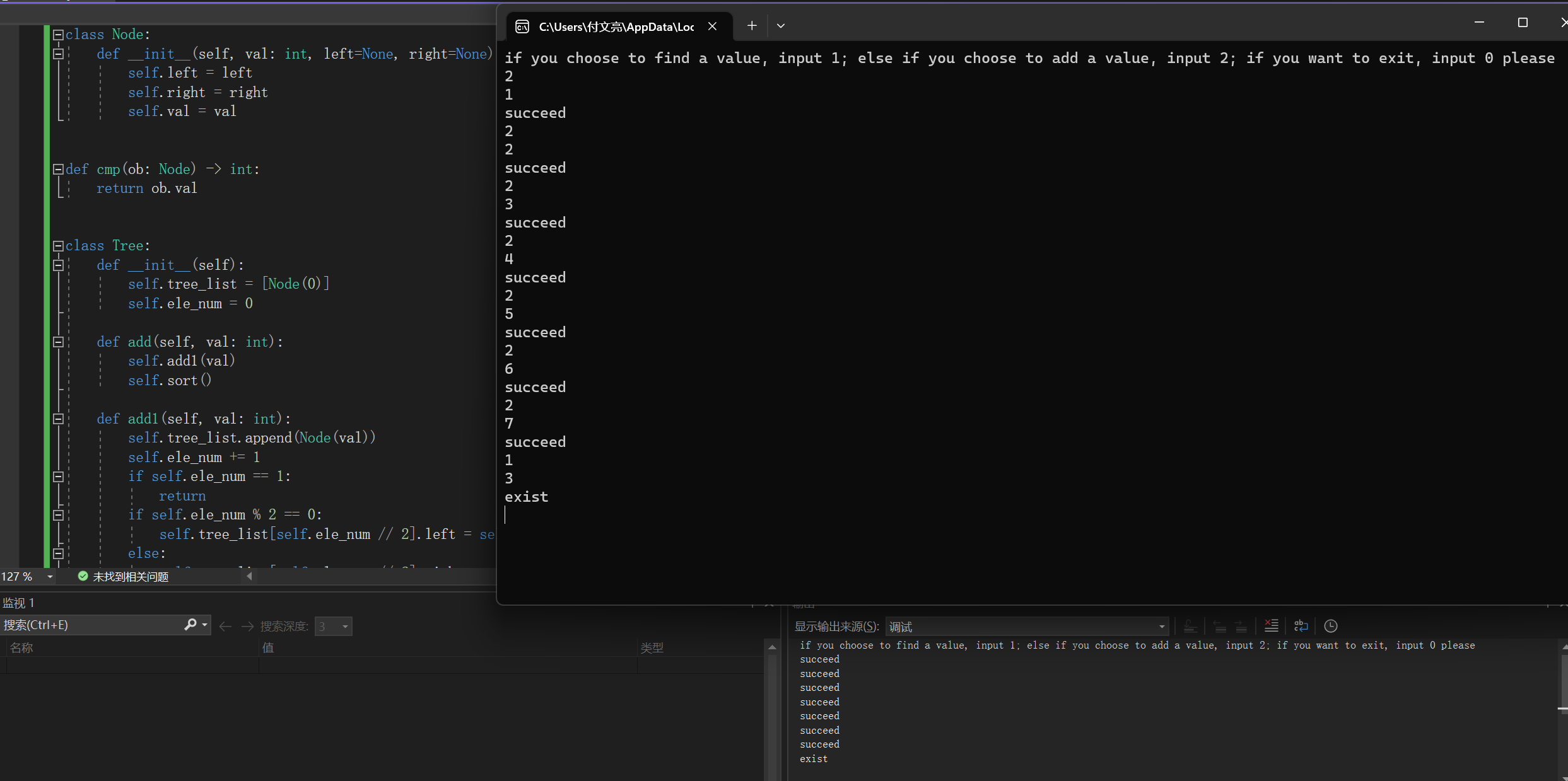
print("over")

break;

else:

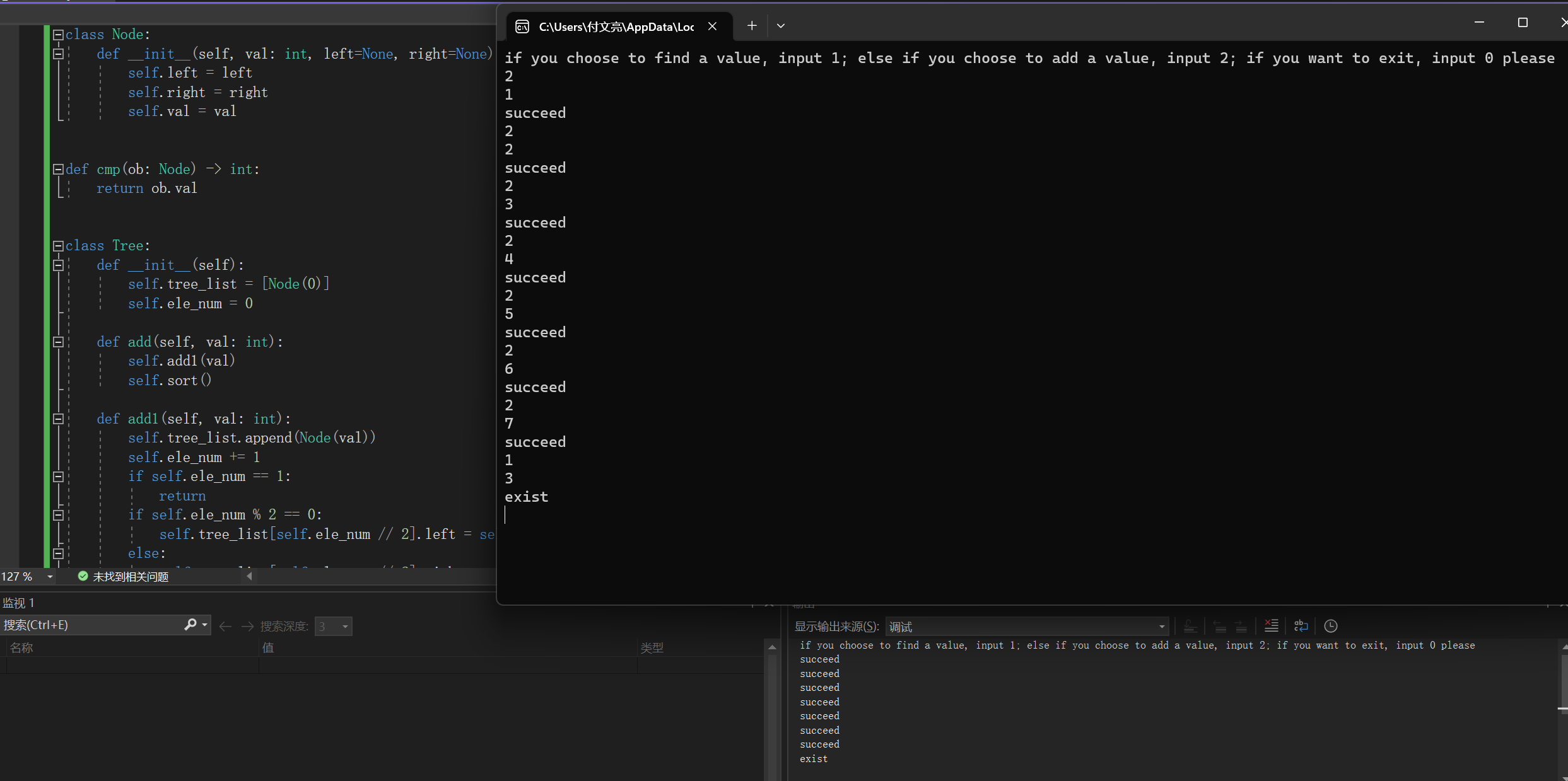
print("input error")

## 运行结果

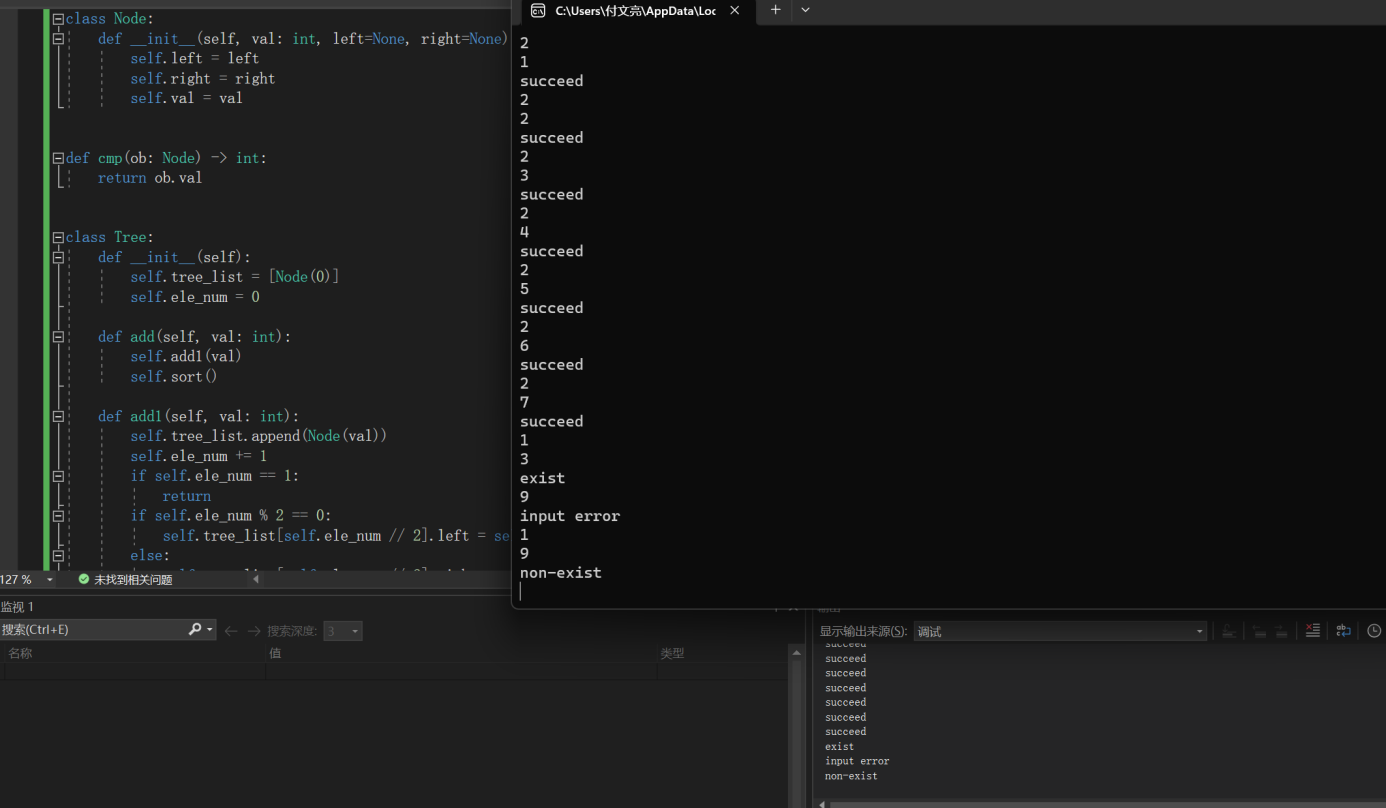


建立一个1~7的二叉树并在其中查找元素3得到存在结果

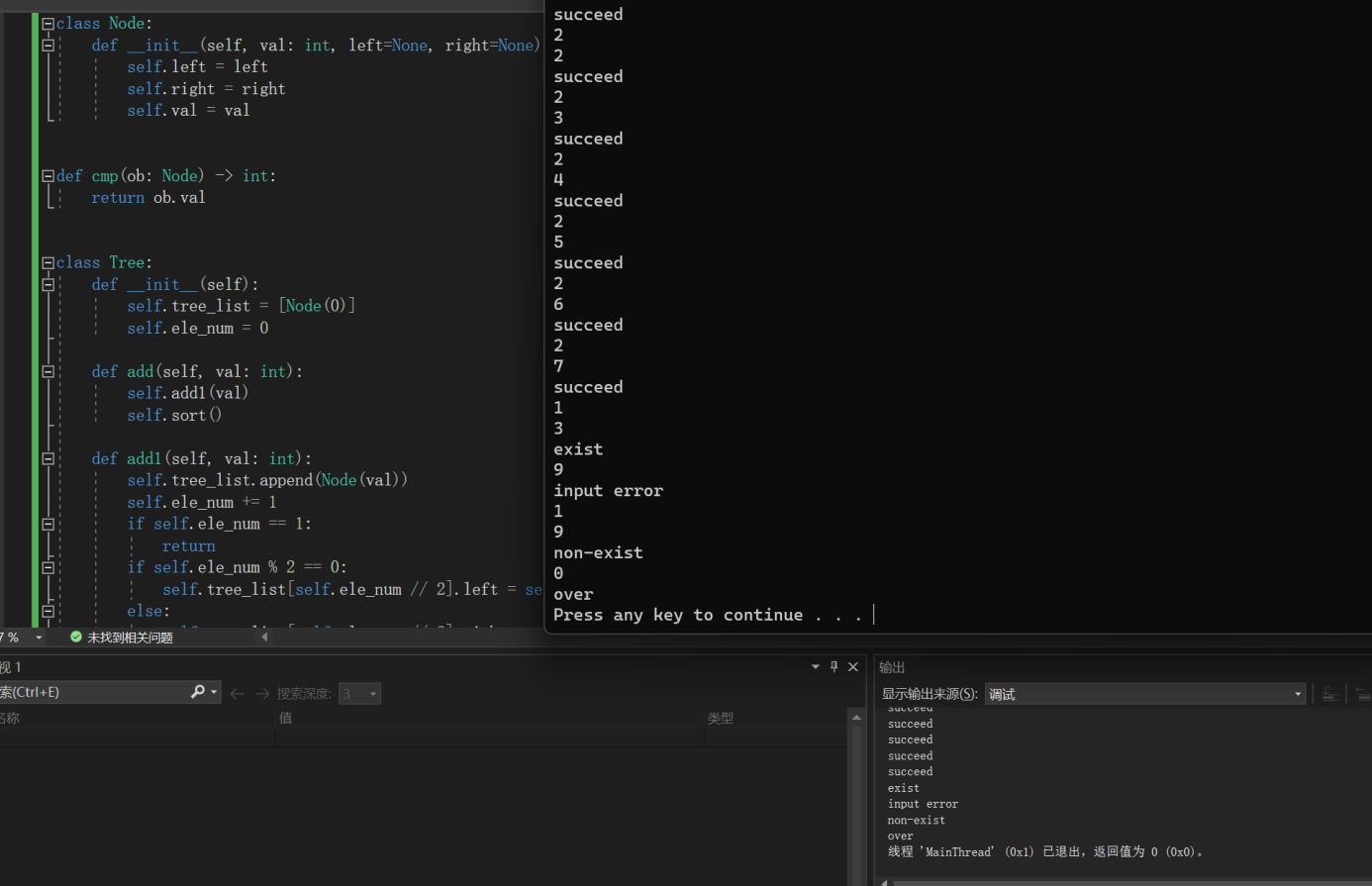
## 实验结果分析



逐步添加1~7元素到二叉树中均获得成功，创建完二叉树后使用查找元素查找3获得成功。



输入错误时，没有调用添加或查找函数会提示错误并允许重新输入。正确输入后查找元素9发现不存在，符合事实条件。



输入0时，程序成功退出。

## 优缺点及改进方向

优点:实现了二叉树的建立、插入元素排序、搜索二叉树中元素的功能，使用宽度优先搜索提高了数据搜索效率。

缺点:代码注释太少，不易让他人看懂；插入元素排序功能需要遍历所有数据，运行效率低下。

改进方向:代码中应出现注释的地方添加注释，便于别人理解，优化插入元素排序功能，减少遍历的元素数量，提高运行效率。

## 心得体会与总结

心得体会:知道了二叉树的作用以及如何创建二叉树并对其中的元素进行查找，感受到了二叉树对提高元素搜索效率提高的重要作用，学会了怎样使用宽度优先搜索方法搜索元素。

总结:当遇到错综复杂的数据时可以采用二叉树对这些数据进行存储和查询，查询二叉树中元素的方法可以采用宽度优先搜索方法。建立二叉树时也要考虑数据的存储方式，不能盲目的存储数据。

# **问题重述**

二叉树作为基础数据结构之一，熟练掌握二叉树是程序员的基本功之一。该实验是基本的二叉树问题，可引申到数据存储的优化问题中，使得数据结构更加简洁，提高数据的有序性进而提高数据搜索速率，从而提高程序的执行速度。二叉树问题的关键在于以怎样的顺序存储和查找数据，使得数据得到高效利用。

# **对本次实验的设计提出改进意见**

让学生设计的二叉树应用到实际问题中或增添实际问题中蕴含的二叉树问题让学生实现，让学生了解到二叉树的强大功能。设计具体要求输出来体现学生使用了宽度优先搜索方法搜索数据。

# **附件**

## 附件1.二叉树的程序

class Node:

def \_\_init\_\_(self, val: int, left=None, right=None):

self.left = left

self.right = right

self.val = val

def cmp(ob: Node) -> int:

return ob.val

class Tree:

def \_\_init\_\_(self):

self.tree\_list = [Node(0)]

self.ele\_num = 0

def add(self, val: int):

self.add1(val)

self.sort()

def add1(self, val: int):

self.tree\_list.append(Node(val))

self.ele\_num += 1

if self.ele\_num == 1:

return

if self.ele\_num % 2 == 0:

self.tree\_list[self.ele\_num // 2].left = self.tree\_list[self.ele\_num]

else:

self.tree\_list[self.ele\_num // 2].right = self.tree\_list[self.ele\_num]

# check if val is in the tree

def find(self, val: int):

que = Queue()

if self.ele\_num == 0:

return False

que.push\_front(self.tree\_list[1])

while True:

l = que.size

for i in range(l):

if que.queue[i].val == val:

return True,

for i in range(l):

if que.queue[i].left is None:

for i in range(l, que.size):

if que.queue[i].val == val:

return True

return False

que.push\_front(que.queue[i].left)

if que.queue[i].right is None:

for i in range(l, que.size):

if que.queue[i].val == val:

return True

return False

que.push\_front(que.queue[i].right)

for i in range(l):

que.pop(0)

def clear(self):

self.tree\_list.clear()

self.ele\_num = 0

# sort in the order of the index (the order of bfs)

def sort(self):

copy = self.tree\_list[:]

copy.sort(key=cmp)

self.clear()

self.tree\_list.append(Node(0))

for i in copy[1:]:

self.add1(i.val)

class Queue:

def \_\_init\_\_(self):

self.queue = []

self.size = 0

def push\_front(self, node: Node):

self.queue.append(node)

self.size += 1

def pop(self, ind: int):

self.queue.pop(ind)

self.size -= 1

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

tree = Tree()

print("if you choose to find a value, input 1; else if you choose to add a value, input 2; if you want to exit, input 0 please")

while True:

a = int(input())

if a == 1:

b = int(input())

if tree.find(b):

print("exist")

else:

print("non-exist")

elif a == 2:

tree.add(int(input()))

print("succeed")

elif a == 0:

print("over")

break;

else:

print("input error")

## 附件2.二叉树程序的输出结果

