数据结构课程设计

北京郵電大學

姓　　名　tobeApe6eHok

学　　院　人工智能学院

专　　业　智能科学与技术

班　　级　1111111111

学　　号　1111111111

班内序号　1

指导教师　周延泉

2022 年　4 月

目录

一、实验题目 3
二、题目分析与算法设计 3
三、数据结构描述 5
四、程序清单 6
五、程序复杂度分析 8
六、程序运行结果 9

数据结构课程设计实验四

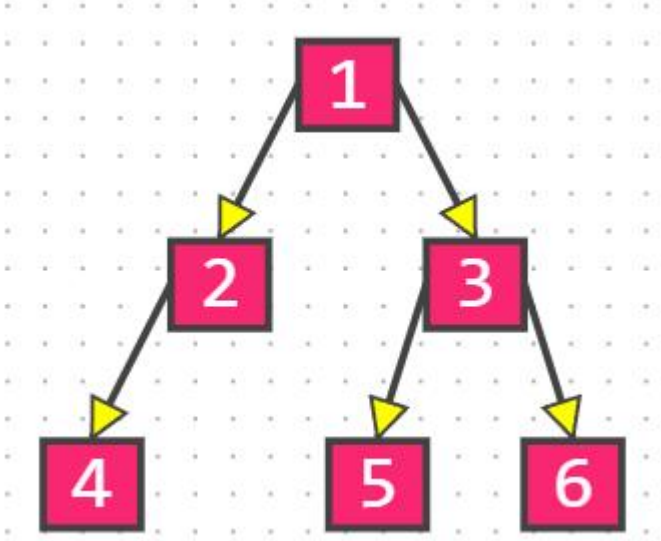
一、实验题目

设计二叉树的前序遍历的递归与非递归算法。

二、题目分析与算法设计

1、设计思路：

（1）在进行前序遍历前，要先创建一棵树，我在代码中创建了两个类：Node 和 Tree 分别表示树节点和树。使用一个层序遍历的列表创建树。比如下图中的树层序遍历为[1,2,3,4,None,5,6]：



（2）有两种方法可以实现二叉树的前序遍历：使用栈数据结构迭代和 Morris 遍历方式；而二叉树的递归前序遍历方式只需要递归遍历即可，实现较简单。

2、算法描述：

首先前序遍历的遍历顺序依次是：当前节点、左节点、右节点。

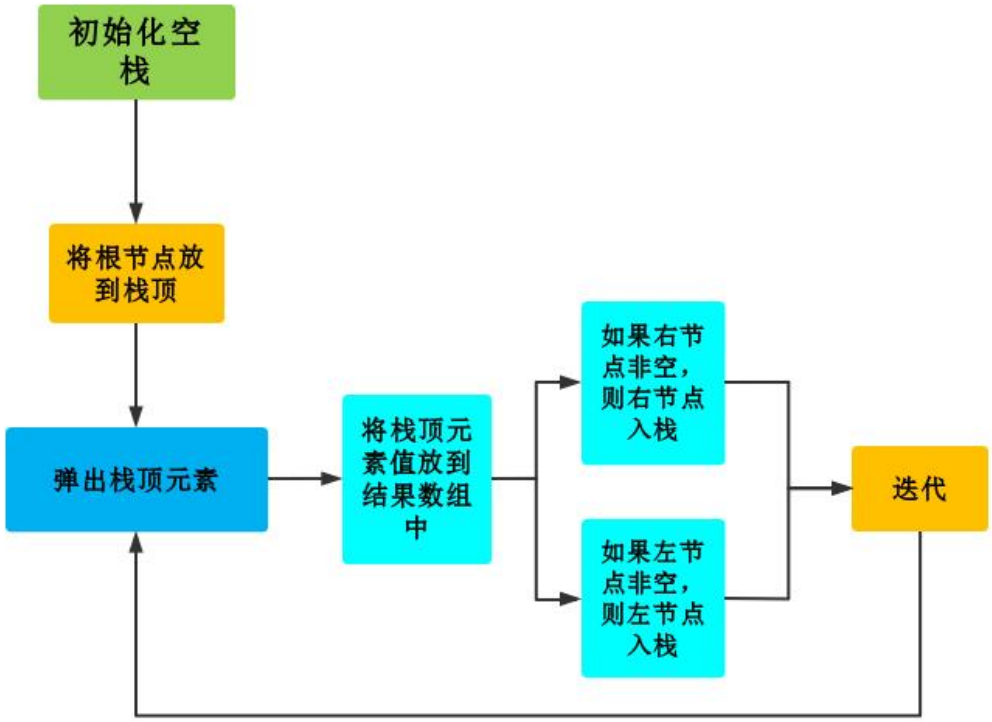
（1）递归算法：

前序遍历的过程具有很好的递归性质，我们定义一个递归函数，只需要模拟出在当前节点的遍历顺序即可。

（2）使用栈数据结构迭代的算法：

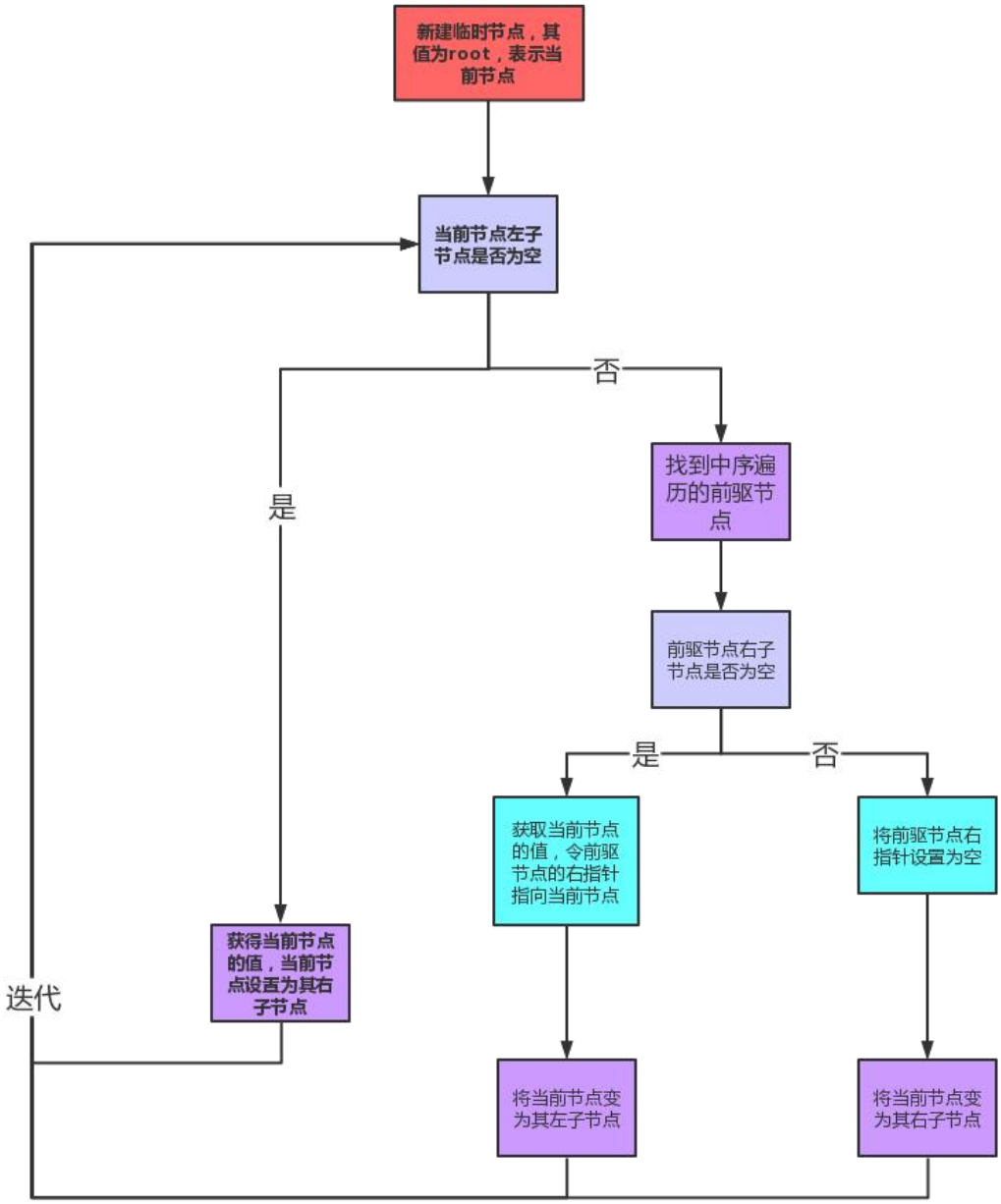
在递归方法中，每次的递归函数调用其实都是一个压栈的过程，我们可以直接使用栈的数据结构将其显式地模拟出来，两者是等价的。

具体的操作步骤是：



（3）Morris 遍历算法：

Morris 遍历算法是一种线性时间复杂度、只占常数空间的巧妙的非递归遍历算法。Morris 算法主要思想是利用二叉树大量的空闲指针，在遍历的同时改变二叉树的结构，控制遍历节点的顺序，同时也不断还原树的原始结构，以此来实现空间的极限缩减。其具体操作步骤是：



三、数据结构描述

1、数据结构的选择：

在递归实现的前序遍历算法和 Morris 算法中均不需要设置特殊的数据结构。在第一种非递归算法中，需要使用栈数据结构。

2、证明数据结构的正确性：

在递归实现的前序遍历中，每一次递归对应创建一次内存调用函数的栈帧序列，所以使用栈数据结构模拟出来是一种很自然、正确的

做法。递归时函数的调用顺序对应着当前节点的左右节点入栈的顺序，首先右子节点入栈，然后左子节点入栈，这对应着左子节点先出栈，右子节点后出栈，也对应着先遍历左子树，后遍历右子树。由此栈数据结构的选择是正确的。

3、数据结构的定义：

为了实现代码的复用和结构的简洁，我设计了 Node 类，定义每个节点的结构；还有一个 Tree 类，其中的 self.Root 是二叉树的根节点，NonRECURSIVEpre()是使用栈模拟递归实现的前序遍历函数，RECURSIVEpre()是使用递归算法实现的前序遍历函数，MORRISpre()是使用 Morris 算法实现的前序遍历函数，三者的返回值都是按照前序遍历顺序得到的节点值的列表。

四、程序清单

以下是我调试好可以正常运行的代码，实验环境是 WIN10+VS code + python 3.10.0。

```
class Node:

    def __init__(self,v) -> None:
        """定义每个节点的内部结构"""
        self.Val=v            # 定义 value 值
        self.Left=None        # 定义 Left 子节点的指针
        self.Right=None       # 定义 Right 子节点的指针

class Tree:
    def __init__(self,Nodes:list) -> None:
        """使用层序遍历结果 Nodes 初始化二叉树"""
        # 如果层序遍历结果为空,则说明二叉树为空,将根节点设置为 None 即可
        if len(Nodes)==0:
            self.Root=None
            return
        # 将根节点的 value 设置为层序遍历第一个值
        self.Root=Node(Nodes[0])
        # remain 是还未设置 Left 和 Right 的节点和其在 Nodes 中的位置
        remain=[[self.Root,0]]
        while len(remain)>0:
            node,place=remain.pop(0)
            # 得到左子节点、右子节点在层序遍历结果中的下标
            leftPlace=place*2+1
            rightPlace=leftPlace+1
            # 如果下标合法,则创建 Left 节点或者 Right 节点
            # 并放入 remain 中
            if leftPlace<len(Nodes) and Nodes[leftPlace]!=None:
                node.Left=Node(Nodes[leftPlace])
                remain.append([node.Left,leftPlace])
            if rightPlace<len(Nodes) and Nodes[rightPlace]!=None:
                node.Right=Node(Nodes[rightPlace])
                remain.append([node.Right,rightPlace])
```

```
        return
    def NonRECURSIVEpre(self)->list:
        """非递归实现前序遍历"""
        # result 是前序遍历的结果
        result=[]
        # 如果 Root 为空,则表示二叉树为空,返回空列表
        if self.Root==None:
            return result
        # remain 表示模拟的栈
        remain=[self.Root]
        # 只要栈中还有元素,就代表还未遍历完
        while len(remain)>0:
            # 最后一个元素出栈
            node=remain.pop(-1)
            # 获得子树根节点值
            result.append(node.Val)
            # 先让右子节点入栈,后让左子节点入栈
            if node.Right!=None:
                remain.append(node.Right)
            if node.Left!=None:
                remain.append(node.Left)
        return result

    def RECURSIVEpre(self)->list:
        """递归实现前序遍历"""
        # result 是前序遍历的结果
        result=[]
        # 如果 Root 为空,则表示二叉树为空,返回空列表
        if self.Root==None:
            return result
        # 递归函数闭包
        def recursive(node)->None:
            # 按照顺序递归遍历当前节点、左子树、右子树
            result.append(node.Val)
            if node.Left!=None:
                recursive(node.Left)
            if node.Right!=None:
                recursive(node.Right)
        recursive(self.Root)
        return result
    def MORRISpre(self)->list:
        """Morris 非递归算法"""
        # result 是前序遍历的结果
        result=[]
        # 如果 Root 为空,则表示二叉树为空,返回空列表
        if self.Root==None:
            return result
        # 将最初的当前节点设置为根节点
        node1=self.Root
        # 直到当前节点为 None,代表所有节点都已遍历
        while node1!=None:
            # 判断其左子节点是否为空
            node2=node1.Left
            # 若非空
            if node2!=None:
                # 则需要找到当前节点的中序遍历前驱节点
                # 因为在 Morris 算法中将所有节点都和前驱节点连起来一次,所以需要加上两个判断
条件
                while node2.Right!=None and node2.Right!=node1:
                    node2=node2.Right
```

```
                # 如果还未连接
                if node2.Right==None:
                    result.append(node1.Val)
                    # 连接前驱节点和当前节点
                    node2.Right=node1
                    # 将当前节点设置为其左子节点
                    node1=node1.Left
                    continue
                # 如果已经连接
                else:
                    # 以为在连接时,已经取过其值,故只需要遍历其右子树即可
                    node2.Right=None
            # 若为空,只需要先取其值,然后遍历其右子树即可
            else:
                result.append(node1.Val)
            # 默认将当前节点设置为其右子节点
            node1=node1.Right
        return result
def main(Nodes:list,num:int):
    """测试程序"""
    print('\n 测试 '+str(num)+' :')
    t=Tree(Nodes)
    print('递归前序遍历:')
    print('            '+str(t.NonRECURSIVEpre()))
    print('栈模拟递归算法:')
    print('             '+str(t.RECURSIVEpre()))
    print('Morris 算法:')
    print('              '+str(t.MORRISpre()))


if __name__=='__main__':
    a=[1,2,3,4,5,6,7,None,8,9,None,10,11,None,12]
    b=[1,2,3,4,5,None,6,None,7,8,9,None,None,10,None,None,None,11,12,None,None,13,14,
None,None,None,None,None,15]
    # 测试一
    main(a,1)
    # 测试二
    main(b,2)
```

五、程序复杂度分析

首先，假设要遍历的二叉树节点总数是 n。

1、时间复杂度：

（1）递归算法：

在递归过程中每个节点都恰好被遍历一次，所以时间复杂度是 O（n）。

（2）栈模拟非递归算法：

在遍历过程中，每个节点都被入栈、出栈、取值一次，所以时间复杂度也是 O(n).

（3）Morris 算法：

Morris 算法通过改变树的结构来控制访问节点的顺序：第一次遇

到节点时，取其值，并且将其和前驱节点连接；第二次遇到则将其与前驱节点断开连接。

所以，对于不同的节点来说，程序中被访问的次数不同（此处的次数还要考虑其父节点在寻找前驱节点时访问它的次数）：

是否有左子节点	是否有右子节点	访问的次数
是	否	3
是	是	3
否	否	2
否	是	2

由此，可得，每个节点最多被访问 3 次，所以时间复杂度也是 O（n）

2、空间复杂度：

（1）递归算法：

递归过程中栈的开销，平均情况下是树的深度，为 O（log（n）），最坏的情况下树的链状的，空间复杂度是 O（n）。

（2）栈模拟非递归算法：

因为是模拟递归算法，所以空间复杂度相同。

（3）Morris 算法：

在遍历过程中并未使用额外的空间存储节点，只是用了一个当前指针的临时变量，一次只需要常数空间，空间复杂度是 O（1）.

六、程序运行结果

1、实验环境：WIN10+VS code + python 3.10.0 64 位

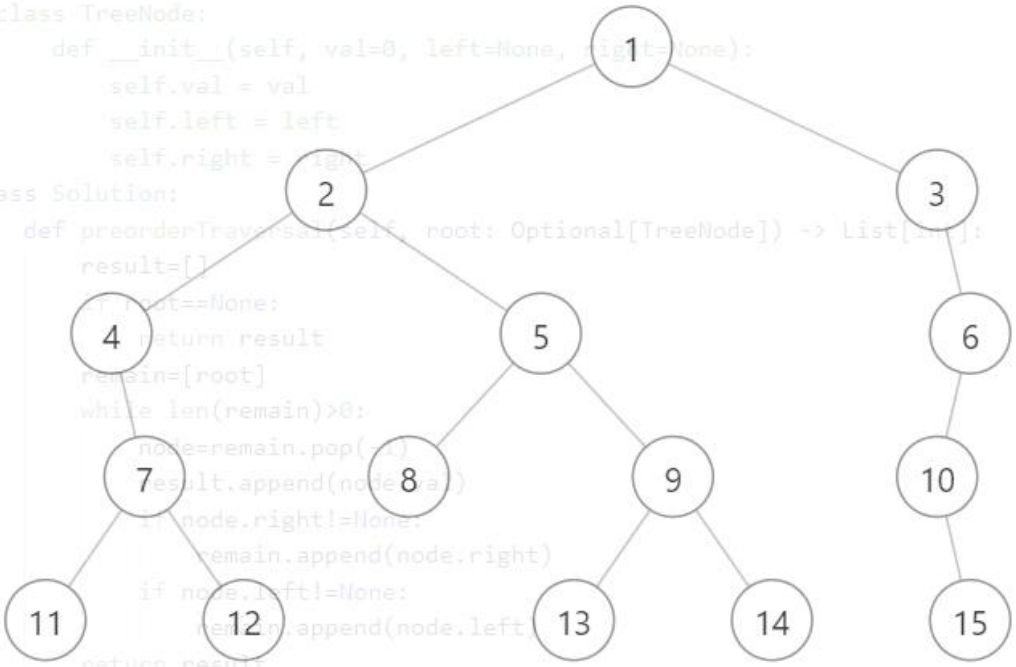
2、测试用例：

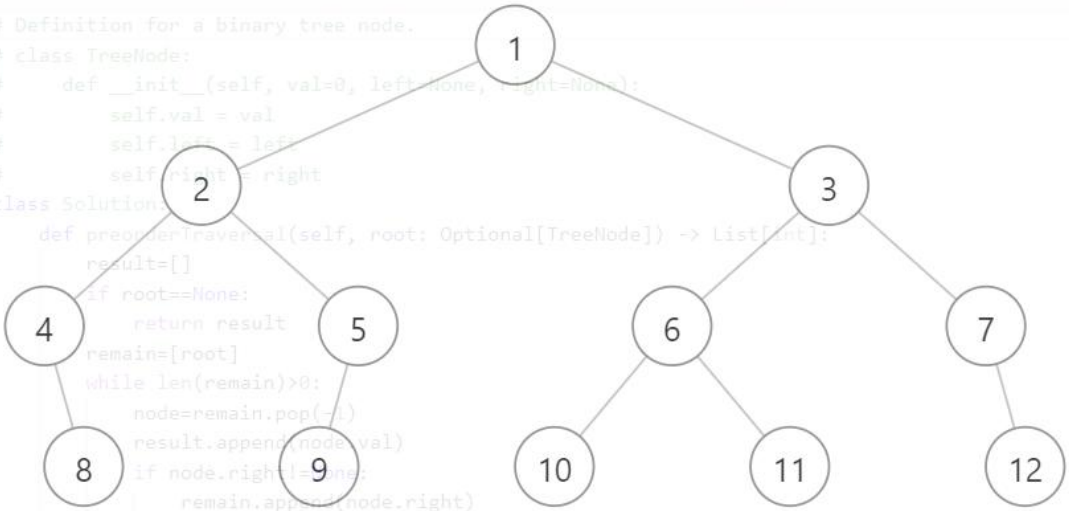
测试中我创建了两个不同结构的二叉树，层序遍历结果分别为：

(1)[1,2,3,4,5,6,7,None,8,9,None,10,11,None,12]

(2)[1,2,3,4,5,None,6,None,7,8,9,None,None,10,None,None,None,
11,12,None,None,13,14,None,None,None,None,None,15]

两棵树的结构分别为：





两者的前序遍历序列应该是：

1、[1, 2, 4, 8, 5, 9, 3, 6, 10, 11, 7, 12]

2、[1, 2, 4, 7, 11, 12, 5, 8, 9, 13, 14, 3, 6, 10, 15]

运行程序，可以得到结果如下，是正确的：

```
测试 1 :
递归前序遍历:
            [1, 2, 4, 8, 5, 9, 3, 6, 10, 11, 7, 12]
栈模拟递归算法:
            [1, 2, 4, 8, 5, 9, 3, 6, 10, 11, 7, 12]
Morris算法:
             [1, 2, 4, 8, 5, 9, 3, 6, 10, 11, 7, 12]

测试 2 :
递归前序遍历:
            [1, 2, 4, 7, 11, 12, 5, 8, 9, 13, 14, 3, 6, 10, 15]
栈模拟递归算法:
            [1, 2, 4, 7, 11, 12, 5, 8, 9, 13, 14, 3, 6, 10, 15]
Morris算法:
             [1, 2, 4, 7, 11, 12, 5, 8, 9, 13, 14, 3, 6, 10, 15]
```