数据结构 A
第七章参考答案

题 目 范 围 ：树和二叉树

授 课 老 师 ：何璐璐

助 教 ：王思怡（撰写人），赵守玺

联 系 邮 箱 ：2021302111095@whu. edu. cn

目录

1. 树的简单问题

【问题描述】

假设二叉树中的每个结点值为单个整数，采用二叉链结构存储，假定每颗二叉树不超过2000个节点。设计算法完成

（1）按从左到右的顺序输出二叉树的叶子结点

（2）按从右到左的顺序输出二叉树的叶子结点

(3) 输出二叉树所有的节点，按照从根节点开始，逐层输出，同一层按照从右向左的顺序

【输入形式】

每个测试是一颗二叉树的括号表示法字符串

【输出形式】

第一行是按从左到右的顺序输出二叉树的叶子结点，结点之间用空格隔开

第二行是按从右到左的顺序输出二叉树的叶子结点，结点之间用空格隔开

第三行是输出二叉树所有的节点，按照从根节点开始，逐层输出，同一层按照从右向左的顺序，结点之间用空格隔开

【样例输入】

1(2(4,5),3)

【样例输出】

4 5 3

3 5 4

1 3 2 5 4

【样例说明】

测试数据的文件名为in. txt

【评分标准】

该题目有10个测试用例，每通过一个测试用例，得10分。

参考答案：

```
#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <stack>
#include <fstream>
#include <algorithm>

using namespace std;

/* Definition for a binary tree node.*/
struct BTNode
{
   int data;
   BTNode *lchild;
   BTNode *rchild;
   BTNode() : lchild(NULL), rchild(NULL) {}
   BTNode(int x) : data(x), lchild(NULL), rchild(NULL) {}
};
class BTree
{
   BTNode *r;

public:
   BTree()
   {
      r = NULL;
   }
   void CreateBTree(string str) // 从 str 创建二叉树
   {
      stack<BTNode *> st; // 定义名为 st 的栈
      BTNode *p;
      bool flag;
      int i = 0;
      while (i < str.length()) // 用 i 从前到后遍历 str 的每个字符
      {
         switch (str[i])
         {
         case '(':
            st.push(p);  // 如果是'(', 则将 p 压入栈中
            flag = true; // flag=true 说明去找左子节点
            break;
         case ')':
```

```
            st.pop(); // 否则出栈
            break;
        case ',':
            flag = false; // 如果遇到逗号就将flag 置为false，去找右子节点
            break;
        default: // 遇到数字就进行读入操作
            int sum = 0;
            while (str[i] <= '9' && str[i] >= '0')
            {
                sum = sum * 10 + str[i] - '0';
                i++;
            }
            i--;
            p = new BTNode(sum); // 将读到的数字存入当前节点p
            if (r == NULL)       // 如果根节点为空，则将p 设为根节点
                r = p;
            else // 否则进行查找，将p 插入子节点
            {
                if (flag && !st.empty()) // flag=true 表示p 插入左子节点
                    st.top()->lchild = p;
                else if (!st.empty()) // 否则将p 插入右子节点
                    st.top()->rchild = p;
            }
            break;
        }
        i++; // 进行下一个字符的读取
    }
}

void DispBTree() // 输出二叉树，从根节点开始遍历
{
    DispBTree1(r);
}
void DispBTree1(BTNode *b) // 作为DispBTree 的搜索函数
{
    if (b != NULL)
    {
        cout << b->data; // 输出当前节点（先序遍历）
        if (b->lchild != NULL || b->rchild != NULL)
        {
            cout << "(";           // 输出子节点前打"("
            DispBTree1(b->lchild); // 递归输出左子节点
            if (b->rchild != NULL)
                cout << ",";       // 如果有右子节点，在左右子节点之间输出","
            DispBTree1(b->rchild); // 递归输出右子节点
            cout << ")";           // 输出子节点后打")"
```

```
        }
    }
}
void LeftToRight() // 从左往右输出叶子节点
{
    LeftToRight1(r);
}
void LeftToRight1(BTNode *b) // 作为LeftToRight()的搜索函数
{
    if (b != NULL)
    {
        if (b->lchild == NULL && b->rchild == NULL) // 判断是叶子节点，就输出
            cout << b->data << " ";
        else if (b->lchild != NULL)
            LeftToRight1(b->lchild);
        LeftToRight1(b->rchild);
    }
}

void RightToLeft() // 从右往左输出叶子节点
{
    RightToLeft1(r);
}
void RightToLeft1(BTNode *b) // 作为RightToLeft 的辅助函数
{
    if (b != NULL)
    {
        if (b->lchild == NULL && b->rchild == NULL) // 如果是叶子节点就输出
            cout << b->data << " ";
        else if (b->rchild != NULL) // 先去遍历右子节点
            RightToLeft1(b->rchild);
        RightToLeft1(b->lchild);
    }
}
void AllLevel() // 逐层输出节点
{
    AllLevel1(r);
}
void AllLevel1(BTNode *b) // 作为AllLevel 的辅助函数
{
    BTNode *p;
    queue<BTNode *> qu;
    qu.push(b);
    while (!qu.empty())
    {
        p = qu.front();
```

```
            qu.pop();                // 将 p 从队列中弹出
            cout << p->data << " "; // 输出 p 的数据
            if (p->rchild != NULL)  // 将右子节点压入队列
               qu.push(p->rchild);
            if (p->lchild != NULL) // 将左子节点压入队列
               qu.push(p->lchild);
         }
      }
};
int main()
{
   string s;
   BTree bt;
   vector<string> v;
   bool result;
   ifstream infile("in.txt");
   if (!infile)
   {
      cerr << "Error opening file";
      exit(EXIT_FAILURE);
   }
   while (getline(infile, s))
   {
      v.push_back(s);
   }
   infile.close();
   infile.clear();
   bt.CreateBTree(v[0]);
   bt.LeftToRight();
   cout << endl;
   bt.RightToLeft();
   cout << endl;
   bt.AllLevel();
}
```

解析：

本题难点在于解析输入的代表二叉树的字符串。处理需要关注的问题主要是：遇到左括号就深入一层，遇到右括号就撤回一层。

这个过程可以用栈来模拟，也可以把所有节点的父子关系都保存下来。我们在读取字符串的过程中需要知道当前处理节点的位置、父节点和左右孩子节点。

2. 树的中等题目

【问题描述】

假设二叉树中的每个结点值为单个整数，采用二叉链结构存储，设计算法判断给定的二叉树是否是完全二叉树。假定每棵二叉树节点不超过2000个。

【输入形式】

每个测试是一颗二叉树的括号表示法字符串

【输出形式】

如果是完全二叉树，输出“1”；如果不是完全二叉树，输出“0”

【样例输入】

1(2(4,5),3)

【样例输出】

1

【样例说明】

测试数据的文件名为in. txt

【评分标准】

该题目有10个测试用例，每通过一个测试用例，得10分。

参考答案：

```
#include <iostream>
#include <vector>
#include <stack>
#include <queue>
#include <string>
#include <fstream>

using namespace std;

/* Definition for a binary tree node.*/
struct BTNode
{
    int data;
    BTNode *lchild;
    BTNode *rchild;
```

```
    BTNode() : lchild(NULL), rchild(NULL) {}
    BTNode(int x) : data(x), lchild(NULL), rchild(NULL) {}
};
class BTree
{
    BTNode *r;

public:
    BTree()
    {
        r = NULL;
    }
    void CreateBTree(string str) // 从 str 创建二叉树
    {
        stack<BTNode *> st; // 定义名为 st 的栈
        BTNode *p;
        bool flag;
        int i = 0;
        while (i < str.length()) // 用 i 从前到后遍历 str 的每个字符
        {
            switch (str[i])
            {
            case '(':
                st.push(p);  // 如果是'(', 则将 p 压入栈中
                flag = true; // flag=true 说明去找左子节点
                break;
            case ')':
                st.pop(); // 否则出栈
                break;
            case ',':
                flag = false; // 如果遇到逗号就将 flag 置为 false, 去找右子节点
                break;
            default: // 遇到数字就进行读入操作
                int sum = 0;
                while (str[i] <= '9' && str[i] >= '0')
                {
                    sum = sum * 10 + str[i] - '0';
                    i++;
                }
                i--;
                p = new BTNode(sum); // 将读到的数字存入当前节点 p
                if (r == NULL)       // 如果根节点为空, 则将 p 设为根节点
                    r = p;
                else // 否则进行查找, 将 p 插入子节点
                {
                    if (flag && !st.empty()) // flag=true 表示 p 插入左子节点
```

```
                    st.top()->lchild = p;
                else if (!st.empty()) // 否则将p插入右子节点
                    st.top()->rchild = p;
            }
            break;
        }
        i++; // 进行下一个字符的读取
    }
}

void DispBTree() // 输出二叉树，从根节点开始遍历
{
    DispBTree1(r);
}
void DispBTree1(BTNode *b) // 作为DispBTree的搜索函数
{
    if (b != NULL)
    {
        cout << b->data; // 输出当前节点（先序遍历）
        if (b->lchild != NULL || b->rchild != NULL)
        {
            cout << "(";           // 输出子节点前打"("
            DispBTree1(b->lchild); // 递归输出左子节点
            if (b->rchild != NULL)
                cout << ",";       // 如果有右子节点，在左右子节点之间输出","
            DispBTree1(b->rchild); // 递归输出右子节点
            cout << ")";           // 输出子节点后打")"
        }
    }
}
bool CompBTree()
{
    bool result = CompBTree1(r);
    cout << result;
}
bool CompBTree1(BTNode *b)
{
    queue<BTNode *> qu; //定义一个队列用来做bfs（按层遍历）
    BTNode *p;
    bool cm = true; // cm表示比较的结果
    bool bj = true; // bj表示是否已经达到边界
    if (b == NULL)
        return true;    //如果传入数据是空树，直接返回true
    qu.push(b);         //将根节点入队
    while (!qu.empty()) //进行循环，遍历整个树
    {
```

```
            p = qu.front();
            qu.pop();                //将队首元素出队
            if (p->lchild == NULL) // p 的左孩子如果为空，则说明达到边界
            {
                bj = false;
                if (p->rchild != NULL) // p 的右孩子不为空，说明不是完全二叉树
                    cm = false;
            }
            else // p 左孩子不为空，继续判断
            {
                if (!bj)
                    cm = false; // 如果已经达到边界仍有左孩子，说明不是完全二叉树
                qu.push(p->lchild);
                if (p->rchild == NULL)  //右孩子为空说明达到边界
                    bj = false;
                else                    //右孩子不为空则继续遍历
                    qu.push(p->rchild);
            }
        }
        return cm;
    }
};
int main()
{
    string s;
    BTree bt;
    vector<string> v;
    bool result;
    ifstream infile("in.txt");
    if (!infile)
    {
        cerr << "Error opening file";
        exit(EXIT_FAILURE);
    }
    while (getline(infile, s))
    {
        v.push_back(s);
    }
    infile.close();
    infile.clear();
    bt.CreateBTree(v[0]);
    bt.CompBTree();
}
```

解析：

参考教材7. 2. 1，考虑完全二叉树的定义，完全二叉树仅有最后两层可能有度小于2

的节点。那么可以按层从左到右遍历所有节点，保证度数要么是222..222000..000的排布，要么是222..2221000..000的排布。并且如果出现了这个度为1的节点，要保证它出现的是左孩子。

3. 由先序和中序序列产生后序序列

【问题描述】

由二叉树的先序序列和中序序列构造二叉树并求其后序序列

【输入形式】

每个测试用例的第一行包含一个整数n（1<=n<=1000）表示二叉树的节点个数，所有节点的编号为1~n，后面两行分别给出先序序列和中序序列。可以假设构造出的二叉树是唯一的。

【输出形式】

对于每个测试用例，输出一行表示其后序序列。

【样例输入】

9

1 2 4 7 3 5 8 9 6

4 7 2 1 8 5 9 3 6

【样例输出】

7 4 2 8 9 5 6 3 1

【样例说明】

测试数据的文件名为in. txt

【评分标准】

该题目有10个测试用例，每通过一个测试用例，得10分。

参考答案：

```
1. #include<iostream>
2. using namespace std;
```

```
3. const int MAXN=1005;
4. int pres[MAXN],ins[MAXN];
5. bool first;
6.
7. void Getposts(int *pres,int * ins,int n)
8. {
9.  if (n<=0)
10.   return;
11. if(n==1)
12. {
13.  if(first)
14.  { printf("%d ",*pres);
15.    first=false;
16.  }
17.  else
18.    printf("%d ",*pres);
19.  return;
20. }
21. int d=*pres;
22. int *p=ins;
23. while(*p!=d) p++;
24. int k=p-ins;
25. Getposts(pres+1,ins,k);
26. Getposts(pres+k+1,p+1,n-k-1);
27. if(first)
28. { printf("%d ",d) ;
29.   first=false;
30.    }
31.    else
32.      printf("%d ",d);
33.}
34.
35.int main()
36.{
37. int n;
38. freopen("in.txt","r",stdin);
39. while(scanf("%d",&n)!=EOF)
40.    { for(int i=0;i<n;i++)
41.     scanf("%d",&pres[i]);
42.   for(int i=0;i<n;i++)
43.     scanf("%d",&ins[i]);
44.   first=true;
45.   Getposts(pres,ins,n);
46.   printf("\n");
47.    }
48.    return 0;
```

```
49.}
```

解析：

如果已知先序遍历，那么先序遍历的第一个位置就是根节点，此时在中序遍历中找到这一根节点，将中序遍历左边部分划分为左子树，右边部分划分为右子树。这时先序遍历也可以按照子树大小进行划分，递归执行上述内容直到划分完毕。

4. 最大二叉树

【问题描述】

给定一个不含重复元素的整数数组，一个以此数组构建的最大二叉树定义如下：

① 二叉树的根是数组中的最大元素。

② 左子树是通过数组中最大值左边部分构造出的最大二叉树。

③ 右子树是通过数组中最大值右边部分构造出的最大二叉树。

要求通过给定的数组构建最大二叉树，并且用括号表示法输出构建好的最大二叉树，假设给定的数组的大小在[1, 1000]之间。

【输入形式】

在一行中输入整数数组，用空格分隔开

【输出形式】

输出生成的二叉树的树根节点的值。

【样例输入】

3 2 1 6 0 5

【样例输出】

6(3(,2(,1)),5(0))

【样例说明】

测试数据的文件名为in. txt

【评分标准】

该题目有10个测试用例，每通过一个测试用例，得10分。

参考答案：

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

/* Definition for a binary tree node.*/
struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution
{
public:
    TreeNode *constructMaximumBinaryTree(vector<int> &nums)
    {
        if (nums.size() == 0)
            return NULL;
        return CreateBTree(nums, 0, nums.size() - 1);
    }
    TreeNode *CreateBTree(vector<int> &nums, int l, int r)
    {
        if (l > r) //如果区间不满足条件就返回NULL，表示不合法
            return NULL;
        int maxi = l;
        //在nums[l..r]区间里定位出nums[maxi]
        for (int i = l + 1; i <= r; i++)
            if (nums[i] > nums[maxi])
                maxi = i;
        TreeNode *b = new TreeNode(nums[maxi]);     //用maxi 的位置创建新节点
        b->left = CreateBTree(nums, l, maxi - 1);  //递归用左边区间创建左子树
        b->right = CreateBTree(nums, maxi + 1, r); //递归用右边区间创建右子树
        return b;
    }
    void DispBTree(TreeNode *b) //用DispBTree 函数输出整棵树
    {
        if (b != NULL)
        {
```

```
            cout << b->val; //先序遍历输出根节点
            if (b->left != NULL || b->right != NULL)
            {
                cout << "(";         //如果有子树就要先输出"("
                DispBTree(b->left); //然后输出左子树
                if (b->right != NULL)
                    cout << ",";     //如果有右子树就输出","
                DispBTree(b->right); //输出右子树
                cout << ")";         //输出完毕后用")"结尾
            }
        }
    }
};

int main()
{
    TreeNode *b;
    Solution sol;
    vector<int> v;
    freopen("in.txt", "r", stdin);
    int count, n;
    count = 0;
    while (~scanf("%d", &n))
    {
        v.push_back(n);
        count++;
    }
    if (count == 0)
        printf("???????");
    else
    {
        b = sol.constructMaximumBinaryTree(v);
        sol.DispBTree(b);
        // printf("%d",b->val);
    }
    return 0;
}
```

解析：

本题核心思路是每次找到数组中的最大值，然后分裂数组，递归进行左右子树的构建。通过控制传入的参数判断当前需要处理的区间位置。