数据结构 A
作业 8 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：树
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 多选题

在一棵度为 4 的树 T 中，若有 20 个度为 4 的结点，10 个度为 3 的结点，1 个度为 2 的结点，10 个度为 1 的结点，则树 T 的叶节点个数是（　　）。

A．41
B．82
C．113
D．122

答案：B

解析：题目说树 T 的度为 4，因此每个节点度的大小不会超过 4。根据教材 7.1.4，**树的节点个数等于所有节点度数和加一**。也就是说，除了根节点以外，每个节点有一个唯一的父节点。那么一个度对应一个**子节点**，根节点没有父节点，所以要额外加 1。

因此答案应该是 $20 \times 4 + 10 \times 3 + 1 \times 2 + 10 \times 1 + 1 = 82$。

习题 2 多选题

具有 12 个叶节点的二叉树中有（　　）个度为 2 的节点

A. 8
B. 9
C. 10
D. 11

答案：D

解析：二叉树意味着每个节点最多有两个子节点。分类讨论一下，

- 如果一个节点有 0 个子节点，就说明它是叶子；

- 如果有 1 个子节点，说明它不会影响叶子节点；
- 如果有 2 个子节点，它会增加一个分支数量，也就是会多一个叶子节点。

考虑到根节点在没有子节点的时候就是 1 个叶子，每多一个度为 2 的点就会增加一个叶子节点，因此叶子节点的数量是度为 2 的节点数量 +1。所以具有 12 个叶节点的二叉树中有 11 个度为 2 的节点。

习题 3 多选题

已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶结点，则完全二叉树的结点个数最多是（　　）

A. 39

B. 52

C. 111

D. 119

答案：C

解析：完全二叉树在第六层的节点数不会超过 $2^5 = 32$。如果此时叶子节点个数并未到达 32 个，说明这棵完全二叉树为 **6 或 7** 层。

当这棵完全二叉树为 6 层时，最左边 8 个叶子节点在第 6 层，加上前五层是 $8 + (2^5 - 1) = 41$ 个节点。

当这棵完全二叉树为 7 层时，第 6 层最右边 8 个是叶子节点，因此左边 $32-8 = 24$ 个是非叶子节点，那么第 7 层有 48 个节点，加上前六层共 $48 + (2^6 - 1) = 111$ 个节点。

习题 4 多选题

已知 n 个结点的二叉树具有最小路径长度时，其深度为 k，那么第 k 层上的结点数为（　　）

A. $n - 2^{k-2} + 1$

B. $n - 2^{k-1} + 1$

C. $n - 2^k + 1$

D. $n - 2^{k-1}$

答案：B

解析：二叉树具有最小路径长度时，说明此时**深度最低**。也就是尽可能将二叉树的形态构建为完全二叉树，当深度为 k 就要尽可能保证前 $k-1$ 层全满。

前 $k-1$ 层的节点个数为 $\sum_{i=1}^{k-1} 2^{i-1} = 2^k - 1$。因此第 k 层剩余 $n - 2^{k-1} + 1$ 个节点。

习题 5 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。

答案:

```cpp
#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
```

```
58                  if (r == NULL)          // 如果根节点为空，则将p设为根节点
59                      r = p;
60                  else // 否则进行查找，将p插入子节点
61                  {
62                      if (flag && !st.empty()) // flag=true表示p插入左子节点
63                          st.top()->lchild = p;
64                      else if (!st.empty()) // 否则将p插入右子节点
65                          st.top()->rchild = p;
66                  }
67                  break;
68              }
69              i++; // 进行下一个字符的读取
70          }
71      }
72
73      void DispBTree() // 输出二叉树，从根节点开始遍历
74      {
75          DispBTree1(r);
76      }
77      void DispBTree1(BTNode *b) // 作为DispBTree的搜索函数
78      {
79          if (b != NULL)
80          {
81              cout << b->data; // 输出当前节点（先序遍历）
82              if (b->lchild != NULL || b->rchild != NULL)
83              {
84                  cout << "(";             // 输出子节点前打"("
85                  DispBTree1(b->lchild); // 递归输出左子节点
86                  if (b->rchild != NULL)
87                      cout << ",";        // 如果有右子节点，在左右子节点之间输出","
88                  DispBTree1(b->rchild); // 递归输出右子节点
89                  cout << ")";            // 输出子节点后打")"
90              }
91          }
92      }
93      void LeftToRight() // 从左往右输出叶子节点
94      {
95          LeftToRight1(r);
96      }
97      void LeftToRight1(BTNode *b) // 作为LeftToRight()的搜索函数
98      {
99          if (b != NULL)
100         {
101             if (b->lchild == NULL && b->rchild == NULL) // 判断是叶子节点，就输出
102                 cout << b->data << " ";
103             else if (b->lchild != NULL)
104                 LeftToRight1(b->lchild);
```

```
105             LeftToRight1(b->rchild);
106         }
107     }
108 
109     void RightToLeft() // 从右往左输出叶子节点
110     {
111         RightToLeft1(r);
112     }
113     void RightToLeft1(BTNode *b) // 作为RightToLeft的辅助函数
114     {
115         if (b != NULL)
116         {
117             if (b->lchild == NULL && b->rchild == NULL) // 如果是叶子节点就输出
118                 cout << b->data << " ";
119             else if (b->rchild != NULL) // 先去遍历右子节点
120                 RightToLeft1(b->rchild);
121             RightToLeft1(b->lchild);
122         }
123     }
124     void AllLevel() // 逐层输出节点
125     {
126         AllLevel1(r);
127     }
128     void AllLevel1(BTNode *b) // 作为AllLevel的辅助函数
129     {
130         BTNode *p;
131         queue<BTNode *> qu;
132         qu.push(b);
133         while (!qu.empty())
134         {
135             p = qu.front();
136             qu.pop();                // 将p从队列中弹出
137             cout << p->data << " "; // 输出p的数据
138             if (p->rchild != NULL)  // 将右子节点压入队列
139                 qu.push(p->rchild);
140             if (p->lchild != NULL) // 将左子节点压入队列
141                 qu.push(p->lchild);
142         }
143     }
144 };
145 int main()
146 {
147     string s;
148     BTree bt;
149     vector<string> v;
150     bool result;
151     ifstream infile("in.txt");
```

```
152     if (!infile)
153     {
154         cerr << "Error opening file";
155         exit(EXIT_FAILURE);
156     }
157     while (getline(infile, s))
158     {
159         v.push_back(s);
160     }
161     infile.close();
162     infile.clear();
163     bt.CreateBTree(v[0]);
164     bt.LeftToRight();
165     cout << endl;
166     bt.RightToLeft();
167     cout << endl;
168     bt.AllLevel();
169 }
```

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**当前处理的节点位置、当前节点的父节点、当前节点的左右孩子节点**。

总结

题目：树

日期：2024 年 5 月 13 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：需要熟练掌握度的概念。

习题 2：注意对二叉树的理解和分类讨论。

习题 3：完全二叉树在同一层可能有两种形态：全满或未满。在讨论这种问题时需要对两种都注意到。

习题 4：弄明白二叉树具有最小路径长度代表的含义，接着对二叉树进行数学统计和计算即可。

习题 5：对于这种题目，需要熟练应用字符串和栈/递归。对于树的遍历，建议熟练掌握递归和非递归的方法。

本次作业经查重，有两组同学的代码有相似情况，已扣除 20% 的分数。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。