

数据结构 A

作业 9 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 单选题

设有 13 个值，用它们组成一棵哈夫曼树，则该哈夫曼树共有（ ）个结点

- A.13
- B.12
- C.26
- D.25

答案：D

解析：若有 13 个值，说明有 13 个叶子节点。每次合并两个叶子节点会增加一个节点并减少一个联通块。一开始是 13 个联通块，需要合并 12 次到 1 个联通块。

因此需要增加 $13 - 1 = 12$ 个节点，所以哈夫曼树共有 $13 + 12 = 25$ 个节点。

习题 2 单选题

某二叉树的先序序列和后序序列正好相反，则该二叉树一定是（ ）

- A. 空或者只有一个结点
- B. 完全二叉树
- C. 二叉排序树
- D. 高度等于其结点数

答案：D

解析：先序序列的遍历顺序是：【当前、左子树、右子树】；后序序列的遍历顺序是：【左子树、右子树、当前】。

为了完全倒转过来，我们需要让【当前、左子树、右子树】变成【右子树倒序、左子树倒序、当前】，并和【左子树、右子树、当前】保持一致。

所以只要保证左右子树只有其中一棵,就可以了,此时除了叶子节点,所有点的度数都为 1,高度等于其结点数。

习题 3 单选题

一颗完全二叉树上有 1001 个结点,其中叶子结点的个数是 ()

- A.250
- B.501
- C.254
- D.505

答案: B

解析: 考虑完全二叉树的定义,每一层、每一个节点一定要填满了才能开始填下一层、下一个节点。那么第 i 层一定会有 2^{i-1} 个节点。

第一层 1 个,第二层 2 个,第三层 4 个,……,第 k 层 2^{k-1} 个。

观察等比数列得到, $1 + 2 + 2^2 + \cdots + 2^9 = 1023$, 所以 1001 个节点的完全二叉树的高度为 9。此时撤掉最后的 $1023 - 1001 = 22$ 个节点,为第 8 行空出 11 个叶子,此时第 9 行剩余 $512 - 22 = 490$ 个叶子。

因此叶子数量为 501。

习题 4 单选题

如果将一棵有序树 T 转换为二叉树 B , 那么 T 中结点的层次序列对应 B 的 () 序列

- A. 先序遍历
- B. 中序遍历
- C. 层次遍历
- D. 以上都不对

答案: D

解析: 参考教材 7.8.1。此题可以用排除法进行判断

二叉树无论先序遍历、中序、后序,都会分离左右两棵子树。而有序树是有层次的,左右子树可能是混合连接的,所以要先排除 A、B 两项。

当一个节点有很多孩子时,它的孩子会顺着当前的右子树逐层加深排布,因此也无法满足层次遍历。所以选 D。

习题 5 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 ($1 \leq n \leq 1000$) 表示二叉树的节点个数，所有节点的编号为 $1 \sim n$ ，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。

答案：

```
1 #include<iostream>
2 #include<fstream>
3 #include<vector>
4 using namespace std;
5 vector<int> pre,ins;
6 void suc(int l1,int r1,int l2,int r2)
7 {
8     if(l1>r1||l2>r2)//递归边界，此时树空
9         return;
10    //考虑先序遍历第一个一定是「当前的根节点」
11    //也就是在l2-r2中找到pre[l1]所在的位置
12    int i;
13    for(i=l2;i<=r2;i++)
14    {
15        if(ins[i]==pre[l1])
16            break;
17    }
18    //此时i就是中序遍历的根节点，前面为左孩子，右边为右孩子
19    //左孩子大小为i-l2
20    //右孩子大小为r2-i
21    //后序遍历为：左、右、当前
22    suc(l1+1,l1+i-l2,l2,i-1);
```

```
23     suc(r1-r2+i+1,r1,i+1,r2);
24     // 此处注意左右长度保持一致，右子树长度更好确定
25     cout<<pre[l1]<<" ";
26 }
27 int main()
28 {
29     ifstream in("in.txt");
30     int n,x;
31     in>>n; // 输入 n
32     for(int i=0;i<n;i++)
33     {
34         in>>x; // 输入当前数字并将其插入链表最后，作为数组
35         pre.push_back(x);
36     }
37     for(int i=0;i<n;i++)
38     {
39         in>>x; // 输入当前数字并将其插入链表最后，作为数组
40         ins.push_back(x);
41     }
42     // 该函数表示将先序遍历 l1-r1 与中序遍历 l2-r2 进行对应，并输出后序
43     suc(0,n-1,0,n-1);
44     return 0;
45 }
```

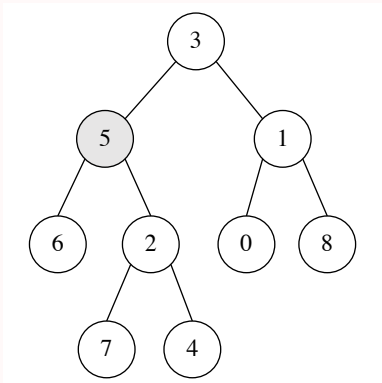
解析：如果已知先序遍历，那么先序遍历的第一个位置就是根节点，此时在中序遍历中找到这一根节点，将中序遍历左边部分划分为左子树，右边部分划分为右子树。这时先序遍历也可以按照子树大小进行划分，递归执行上述内容直到划分完毕。

习题 6 二叉树中距离为 k 的结点问题

【问题描述】

给你二叉树的根结点 $root$ 、树中一个结点 $target$ 和一个正整数 k ，求二叉树中距离 $target$ 结点为 k 的所有结点。假设二叉树中的所有结点值都唯一。输入为顺序存储方式表示的二叉树字符串，如果节点为空，则输入‘#’。

例如，输入 $root = [3, 5, 1, 6, 2, 0, 8, \#, \#, 7, 4]$, $target = 5$ $k = 2$ ，输出为 $[7, 4, 1]$ 。对应的二叉树如图所示。



其中离目标结点 5 的距离为 2 的结点是 7、4、1。假设给定的二叉树非空，树中每个结点有唯一值，结点值位于 0 到 500 之间，*target* 是其中的一个结点， $0 \leq k \leq 1000$ 。

要求设计如下成员函数：

```
1
2 class Solution {
3 public:
4     vector<int> distanceK(TreeNode* root, TreeNode*
        target, int k)
5     { ... }
6 };
```

【输入形式】

每个测试用例由三行，第一行是由一对方括号 [] 括起来的顺序存储的二叉树节点数据，每个节点用“,”隔开，如果是空节点，则存入“#”。第二行为目标节点数据，第三行为距离。

【输出形式】

用一对方括号 [] 将满足要求的节点括起来后输出，如有多个节点，用“,”隔开，节点输出顺序：优先输出目标节点的子孙节点，其次是目标节点兄弟节点的子孙节点，最后是目标节点的祖父节点（如果有多个祖父节点的子孙节点，则从最近的开始输出），如果在同一层从左边到右边输出。

【样例输入】

```
[1,2,3,4,5,6,7,8,9,10,11,12,13,#,#,14,15,16,17,18,19,#,#,#,#,#,#,#,#,#,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290,291,292,293,294,295,296,297,298,299,300,301,302,303,304,305,306,307,308,309,310,311,312,313,314,315,316,317,318,319,320,321,322,323,324,325,326,327,328,329,330,331,332,333,334,335,336,337,338,339,340,341,342,343,344,345,346,347,348,349,350,351,352,353,354,355,356,357,358,359,360,361,362,363,364,365,366,367,368,369,370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,385,386,387,388,389,390,391,392,393,394,395,396,397,398,399,400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,418,419,420,421,422,423,424,425,426,427,428,429,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,445,446,447,448,449,450,451,452,453,454,455,456,457,458,459,460,461,462,463,464,465,466,467,468,469,470,471,472,473,474,475,476,477,478,479,480,481,482,483,484,485,486,487,488,489,490,491,492,493,494,495,496,497,498,499,500]
```

【样例输出】

[20,21,22,23,24,25,26,27,10,11,3]

【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：

```
1 #include<iostream>
2 #include<fstream>
3 #include<vector>
4 #include<queue>
5 #include<stack>
6 using namespace std;
7 struct TreeNode {
8     int val;
9     TreeNode *left;
10    TreeNode *right;
11    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
12 };
13 int target,dis;
14 vector<int> tree;
15 queue<TreeNode*> q; // 存储每个节点，用于跳转Node
16 stack<pair<TreeNode*,int>> s; // 存储祖先节点指针和遍历方向
17 TreeNode* pos; // target所在的指针
18 bool printed=false; // 是否打印过，以此判断是输出逗号还是括号
19 void dfs2(TreeNode* x,int d) // d表示找距离当前深度为d的点
20 {
21     if(x->val==-1) // 空节点直接结束
22         return;
23     if(d==0)
24     {
25         if(!printed)
26         {
27             printed=1;
28             printf("[");
29         }
30         else
31             printf(",");
32         cout<<x->val;
33         return;
34     }
35     dfs2(x->left,d-1);
36     dfs2(x->right,d-1);
37 }
```

```
38 void dfs(TreeNode* x)
39 {
40     if(x==NULL || x->val==-1)
41         return;
42     if(x==pos)
43     {
44         dfs2(x,dis); //当前点子树中距离为dis的打印出来
45         //此时可以把它祖先全部进行遍历
46         while(!s.empty() && dis>0)
47         {
48             dis--; //每向上一层, 需要找的位置就少一个
49             if(dis==0)
50                 dfs2(s.top().first,0);
51             if(s.top().second)
52                 dfs2(s.top().first->left,dis-1);
53             if(s.top().first)
54                 dfs2(s.top().first->right,dis-1);
55             s.pop();
56         }
57         return;
58     }
59     s.push(make_pair(x,0));
60     dfs(x->left);
61     if(!s.empty()) //如果找到答案了就可能是空栈
62         s.pop();
63     s.push(make_pair(x,1));
64     dfs(x->right);
65     if(!s.empty()) //如果找到答案了就可能是空栈
66         s.pop();
67 }
68 int main()
69 {
70     ifstream in("in.txt");
71     string s;
72     in>>s;
73     in>>target>>dis;
74     int now=0; //现在的数字
75     for(int i=1;i<s.size();i++) //跳过[
76     {
77         if(s[i]>='0' && s[i]<='9')
78             now=now*10+s[i]-'0'; //进位读取当前数字
79         else if(s[i]=='#')
80             now=-1; //如果是空为止, 就放-1代表空
81         else
82         {
83             tree.push_back(now); //将当前数字存储到树上
84             now=0; //清空now

```

```
85     }
86 }
87 int fa=0,son=1; //依次找到每个节点对应的父子关系
88 TreeNode* root=new TreeNode(tree[0]); //创建根节点
89 pos=root; //target的位置 (初始化为root)
90 //如果son没有找到target,就说明它在root
91 q.push(root); //需要遍历的父节点
92 while(son<tree.size()) //遍历所有子节点,寻找父节点
93 {
94     TreeNode* Node=q.front(); //当前节点指针
95     q.pop();
96     Node->left=new TreeNode(tree[son]); //优先找左孩子
97     q.push(Node->left);
98     if(tree[son]==target) //如果遇到target就保存位置
99         pos=Node->left;
100     son++; //找下一个孩子
101     if(son<tree.size())
102     {
103         Node->right=new TreeNode(tree[son]);
104         q.push(Node->right);
105         if(tree[son]==target)
106             pos=Node->right;
107     }
108     son++;
109     fa++;
110 }
111 //从根节点开始dfs,用栈存储经历过的点,方便回溯
112 dfs(root);
113 printf("\n"); //格式收尾
114 return 0;
115 }
```

解析：本题可以采用 BFS 宽度优先搜索的方式构建输入的树，也可以用下标对树上的节点进行映射。

此后可以把整棵树的相连关系构建出来，从 *target* 出发走 k 步，输出对应的节点。但是要注意先输出自己子树中的点，再输出父节点、祖先节点。

或者按照上面答案中写的，用栈的形式，依次处理当前节点、父节点、祖先节点。对于当前节点，找到距离为 k 的点，对于父节点，找到另一棵子树中距离为 $k-1$ 的点，以此类推。

总结

题目：树 2

日期：2024 年 5 月 20 日

批改人：王骏骁

邮箱：wjyyy1@126.com

习题 1：考察哈夫曼树的定义。

习题 2：考察二叉树的（三种）遍历方式。

习题 3：对完全二叉树的掌握，主要是要构建出完全二叉树的形状。

习题 4：考察二叉树和多叉树的互换，以及二叉树的（三种）遍历方式。

习题 5：考察先序遍历的特点，以及树的构建和遍历。如果一边构建一边遍历不是很熟练的话，可以先构建再遍历。

习题 6：考察对二叉树的构建、逻辑关系处理，以及树的深度和广度优先搜索。

本次作业经查重，没有代码相似的同学。

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。