哈尔滨工业大学(深圳)2022 年春《数据结构》

第二次作业 树型结构

学号	220110515	姓名	金正达	成绩	

1、简答题

1.1 假设一棵二叉树采用同高度完全二叉树顺序存储,如下表所示：

data		e	a	f		d		g			c	j			h	i					b
下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

请回答下列问题：

①画出该二叉树；

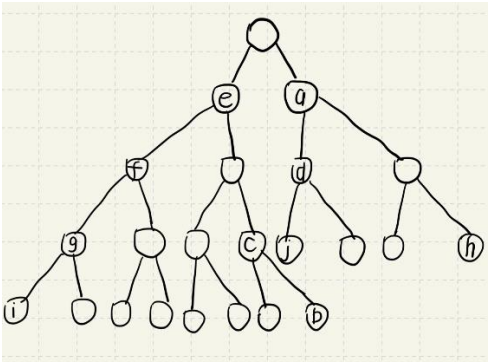
②写出二叉树的先序、中序和后序遍历结果；

③写出结点 c 的双亲结点和左、右孩子结点；

④画出此二叉树还原成的森林。

答：

①



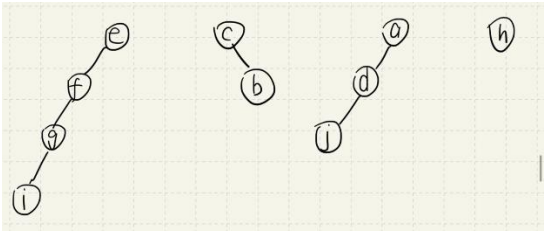
②先序：efgicbadjh

中序：igfecbjdah

后序：igfbcejdha

③双亲为空，左孩子为空，右孩子为 b

④



1.2 对于二叉树 T 的两个结点 n_1 和 n_2，我们应该选择二叉树 T 结点的前序、中序和后序中哪两个序列来判断结点 n_1 必定是结点 n_2 的祖先?
试给出判断的方法。(不需证明判断方法的正确性)

答：选择二叉树 T 结点的前序和后序序列来判断结点 n1 是否是结点 n2 的祖先，

1. 遍历 T 的前序序列，查找结点 n1 和结点 n2 的位置。
2. 遍历 T 的后序序列，查找结点 n1 和结点 n2 的位置。
3. 如果 n1 在 n2 之前出现在 T 的前序序列中，并且 n1 在 n2 之后出现在 T 的后序序列中，则 n1 是 n2 的祖先。否则，n1 不是 n2 的祖先。

1.3 证明在任意非空二叉树 T 上，分支结点数等于叶子结点数的条件是二叉树 T 上度为 1 的结点只有 1 个。

答：任意二叉树中，有 n0 = n2 + 1,
而 n1 = 1, 故有 n0 = n2 + n1, 命题得证.

1.4 假设用于通信的电文仅由 8 个字母组成，字母在电文中出现的频率分别为 0.07,0.19,0.02,0.06,0.32,0.03,0.21,0.10。
（1）试为这 8 个字母设计哈夫曼编码；
（2）试设计另一种由二进制表示的等长编码方案；
（3）对于上述实例，比较（1）和（2）两种方案的优缺点。

答：

（1）

0.19: 00
0.21: 01
0.32: 10
0.07: 1100
0.10: 1101
0.06: 1110
0.02: 11110
0.03: 11111

（2）

0.19: 000
0.21: 001
0.32: 010
0.07: 011
0.10: 100
0.06: 101
0.02: 110

0.03: 111

（3）

第一种：压缩率高，编码长度差异较大，降低了通信效率；

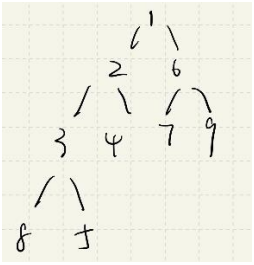
第二种：压缩率低，编码长度固定，编码解码速度较快。

1.5 (a)Draw the binary min heap that results from inserting 8, 7, 3, 2, 4, 6, 9, 5, 1 in that order into an initially empty binary min heap. You do not need to show the array representation of the heap. You are only required to show the final tree.

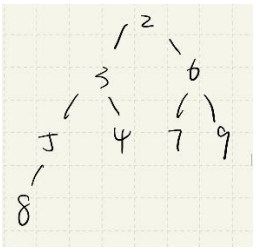
(b) Draw the result of one deletemin call on your heap drawn at the end of part (a).

答：

（a）



（b）



2、算法设计

针对本部分的每一道题，要求：

（1） 给出算法的基本设计思想；

（2） 根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释；

（3） 说明你所设计算法的时间复杂度和空间复杂度。

2.1 已知一棵完全二叉树按顺序方式存储在数组 int A[1..n]中。设计算法，求出下标分别为 i 和 j（i <=n，j <=n）的两个结点的最近的公共祖先结点的位置和值。

答：

```
//找出指定索引的深度
int FindDepth(int *a, int i) {
    int depth = 0;
    while (i >= 1) {
        depth += 1;
        i = i / 2;
    }
    return depth;
}
//返回最近的共同祖先的数组下标
int FindRoot(int* a, int i, int j) {
    int iDepth = FindDepth(i);
    int jDepth = findDepth(j);
    while (iDepth > jDepth) {
        i = i / 2;
        iDepth -= 1;
    }
    while (iDepth < jDepth) {
        j = j / 2;
        jDepth -= 1;
    }
    while (i != j) {
        i = i / 2;
        j = j / 2;
    }
    return i;
}
```

时间复杂度为 O(logN)，空间复杂度为 O(1).

2.2 假设二叉树 bt 采用二叉链表存储，在二叉树 bt 中查找值为 *x* 的结点，试编写算法打印值为 *x* 的结点的所有祖先，假设值为 *x* 的结点不多于一个。

答：

```
//由根向下递归查找，返回true就打印节点
bool find(BTree* p, int x) {
    if (p == NULL) {
        return false;
    }
    if (p->val == x) {
        return true;
    }
    bool flag;
    //先查找左子树
    flag = find(p->left, x);
    if (flag) {
        printf("%d", p->val);
        return true;
    }
    //再查找右子树
    flag = find(p->right, x)
    if (flag) {
        printf("%d", p->val);
        return true;
    }
}
```

时间复杂度 O(N), 空间复杂度为 O(1).

2.3 假设以顺序表 ST 表示一棵完全二叉树，ST.data[ST.last]中存放二叉树中各结点的数据元素。试设计算法由此顺序存储结构建立该二叉树的二叉链表 LT。

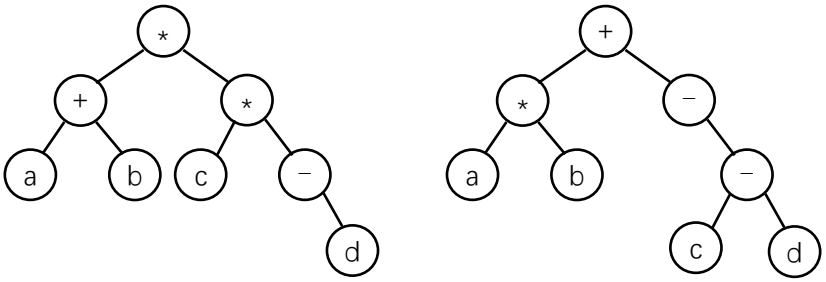
答：

```c
//创建树节点
BTree* createTreeNode(int val, BTree* left, BTree* right) {
    BTree* curr = (BTree*) (malloc(sizeof(BTree)));
    curr->val = val;
    curr->left = left;
    curr->right = right;
    return curr;
}

//使用队列来构建树，空节点值为-1，返回树根节点
BTree* createTreeWithLevelOrder(ST st) {
    QueuePtr qPtr = InitQueue();
    BTree* root = NULL;
    int i = 0;
    if (st.data[i] == -1) {
        return root;
    }
    root = createTreeNode(st.data[i], NULL, NULL);
    EnQueue(qPtr, root);
    while (!QueueEmpty(qPtr) && i < st.last) {
        TreeNodePtr T = DeQueue(qPtr);
        i++;
        int leftData;
        if (i >= st.last) {
            leftData = -1;
        } else {
            leftData = data[i];
        }
        T->left = createTreeNode(leftData, NULL, NULL);
        EnQueue(qPtr, T->left);
        i++;
        int rightData;
        if (i >= st.last) {
            rightData = -1;
        } else {
            rightData = data[i];
        }
        T->right = createTreeNode(rightData, NULL, NULL);
        EnQueue(qPtr, T->right);
    }
    return root;
}
```

时间复杂度 O(N), 空间复杂度 O(1).

2.4 请设计一个算法，将给定的表达式树（二叉树）转换为等价的中缀表达式（通过括号反映操作符的计算次序）并输出。例如，当下列两棵表达式树作为算法的输入时：



输出的等价中缀表达式分别为 (a+b)*(c*(-d)) 和 (a*b)+(-(c-d))。

二叉树结点定义如下：

```
typedef struct node
{    char  data[10];              // 存储操作数或操作符
     struct node *left, *right;
} BTree;
```

答：

```
//中序遍历的过程前后打印括号即可
void visit(BTree* root) {
    printf("(");
    visit(root->left);
    printf("%s", root->data);
    visit(root->right);
    printf(")");
}
```

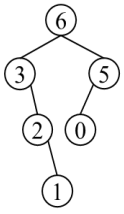
时间复杂度 O(N), 空间复杂度 O(1).

2.5 给定一个非空且无重复元素的整数数组 A，它对应的“最大二叉树”T(A)定义为：

① T(A)的根为 A 中最大值元素；

② T(A)左子树为 A 中最大值左侧部分对应的“最大二叉树”；

③ T(A)右子树为 A 中最大值右侧部分对应的“最大二叉树”。



例如：A={3, 2, 1, 6, 0, 5}对应的“最大二叉树”T (A)如右图所示。

设计一个“最大二叉树”的构建算法，并分析最好情况、最坏情况下的时间和空间复杂性。

答：

```c
//返回数组中最大值的索引
int Max(int* a, int N) {
    int t = 0;
    int max = a[t];
    for (int i = 1; i < N; i++) {
        if (a[i] > max) {
            t = i;
        }
    }
    return t;
}

BTree* CreatMaxBTree(int* a, int N) {
    int index = Max(a, N);
    //复制最大值左右的两个数组的副本
    int left[index];
    int right[N - index - 1];
    for (int i = 0; i < index; i++) {
        left[i] = a[i];
    }
    for (int i = 0; i < N - index - 1; i++) {
        right[i] = a[N - index - 1];
    }
    BTree* root = (BTree*)malloc(szieof(Btree));
    root->val = a[index];
    //递归调用构建子树
    root->left = CreatMaxBTree(left, index);
    root->right = CreatMaxBTree(right, N - index - 1);
}
```

最坏情况：时间复杂度 O(N * N), 空间复杂度 O(NlogN);

最好情况：时间复杂度为 O(N), 空间复杂度为 O(N).