北京大学信息学院

2006 年《数据结构与算法》期中考试答案

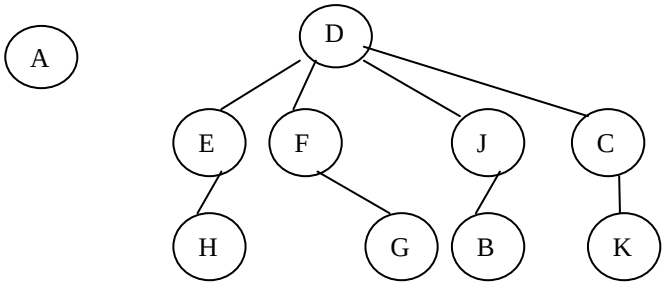
——张铭、赵海燕、王腾蛟

一、填空（30 分，每空 3 分）

1. n－i＋1
2. SaSbXbScSdXdXcSeSfXfXeXa
3. 165
4. n_2+1
5. $\lfloor (i-1)/2 \rfloor$，2i+1
6. 5，120 （4, 96 是有向无序树的情况，可以各给 1.5 分）
7. n(k-1)+1
8. -1， 0，0，1，2，3，4

二、辨析题（20 分，每题分值不等）

1. **(4 分)** 宜采用链式存储结构。采用链表，插入和删除操作只需要改变指针，时间复杂度为 0（1）。而采用顺序存储结构的线性表插入和删除操作涉及到数据地大量移动，时间复杂度为 0（n）。
2. **(4 分)** 把栈 S1 逆置到栈 tmp，然后复制到栈 S2 并且复制回 S1 中。
3. **(5 分)** 关键是 F、J、C 直接指向 D（父子关系正确就可以），E、F、J、C 四棵子树的相对位置没有关系。



4. **(共 7 分)** 算法有错误**（1 分）**。

错误的原因**（2 分）**：第 11 行至第 17 行访问具有非空右兄弟的结点，并把该右兄弟的第一个左孩子入队列；最后一个具有空右兄弟的结点没有处理（独子的情况也一样）。

改法 1：应该在第 17-第 18 行之间加入**（4 分）**

```
if (pointer->LeftMostChild())                    /* 第 17-1 行*/
    aQueue.push(pointer->LeftMostChild());       /* 第 17-2 行 */
```

改法 2：

（1）删除 10、16 行；

（2）修改 11 行为：while(pointer)

（3）在 12、13 行之间加入语句：visit(pointer->value());

三、算法填空（15 分）

参考答案：每个空格 3 分。大小写不符不扣分。

//？1 空一：Cstack.push(cur)

//？2 空二：j < nodes[i].degree() （如果写 <=，扣 1 分）

//？3 空三：temp1.setSibling(temp2); （写反了不给分，写成 temp1->pSilbling = temp2，扣 1 分）

//？4 空四：cur->setChild (temp2); 或 cur->setChild (temp1);（写成 cur->pChild = temp2，扣 1 分）

这里对 temp1 和 temp2 操作是一致的，因为 for 循环最后有 temp2=temp1;这一句。

//？5 空五：cur->setSibling(temp2);（写成 cur->pSilbling = temp2，扣 1 分）

四、算法设计（35 分）（写算法思想、注释、代价分析）

1. **（6 分）**已知 Q 是一个非空队列，S 是一个空栈，请用自然语言描述使用栈 S 将队列 Q 中的所有元素逆置的算法思想。（**不需要写具体的算法实现伪码**）

参考答案：本题只写思想就可以了。两个主要步骤，每个 2 分。请酌情给成绩。

（1）顺序取出队列中的元素，将其入栈；

（2）所有元素入栈后，再从栈中逐个取出，入队。

不要求写下面的代码

```
Template <class T>
Func1(stack &S, queue &Q) {
        T x;    // T 是栈中的元素类型
        InitStack(tmp);
        InitStack(S2);
        while (!Q.empty) {                  // 顺序取出队列中的元素，将其入栈
                x = Q.front();      Q.pop();
                S.push(x);
        }
        while (!S.empty) {                  // 所有元素入栈后，再从栈中逐个取出，入队
                x = S.top();        S.pop();
                Q.push(x);
        }
}
```

代价分析 2 分：设结点个数为 n，则该算法把每个结点都进行过一组出队/进栈、出栈/入队操作，代价为 O(4n)，即 O(n)。

2. **（15 分）**编写一个递归函数 PrintRange，传入参数为一个 BST，一个较小值和一个较大值，按照顺序打印出介于两个值之间的所有结点。

参考答案：

评分标准：算法思想占 2 分（如果不写思想则扣他 5 分），注释占 2 分，算法框架 9 分，代价分析 2 分。

注意：本题没有要求考虑关键码重复的情况，因此学生的算法考虑和不考虑重复关键码都不扣分。但如果考虑重复的情况，应该规定重复关键码是插入在左子树中，或者规定关键码是插入在右子树中，这样检索时就可以顺一个分枝进行。

另，[min, max]区间是否开闭也不扣分，只要在范围之内就可以。

解法 1：考虑重复关键码，而且假设把重复关键码插入在左子树（curr_root->value( ) <= key，插入左；否则插入右）。中序遍历，若某个结点的值<min,那么此结点左子树的值都<min，不需要遍历左子树（剪枝），但要遍历右子树。若某个结点的值=min，因为有重复码，那么需要遍历左子树。实际上=min 时，<max，因此也需要遍历右字树；所以需要遍历左，右子树。若某个结点的值>=max,那么此结点右子树的值都>max，不需遍历右子树，但要遍历左子树。

```
template <class T>
void printRange(BinaryTreeNode <T> *root, T min, T max);
{
```

```
        if (root != NULL)
        {
                // 若当前结点的左子树不为空，并且根结点的值>=min
                if (root -> value() >= min) // 也可加 &&root -> leftchild() != NULL
                        printRange(root -> leftchild(),min,max);
                // 若当前结点在要求范围内
                if (root -> value() >= min && root -> value() <= max) {
                        cout << root -> value();
                // 若当前结点的右子树不为空，并且根结点的值< max
                if (root -> value() < max) // 也可加 &&root -> rightchild() != NULL
                        printRange(root -> rightchild(),min,max);
        }
}
```

注释：如果把**重复关键码插入在右子树，**那么上面的“>=min”修改为“>min”，“<max”修改为“<=max”。

代价分析：设二叉树结点数为 n，处于[min, max]之间的结点个数为 m，则该算法把这些的结点以及经过的祖先访问且只访问一次，代价为 O（m），m<=n。

解法 2：递归周游，判断本结点与[min, max]区间的关系，只进入处于区间内的左右分枝，裁剪处于区间外的分枝。本算法没有考虑重复关键码。代价分析同解法 1 。

```
template <class T>
void BinarySearchTree<T>::RangeQuery(BinaryTreeNode<T> *root, T min, T max)
{
    if ( root == NULL )                         // 结点空，返回
            return;
    if ( root->value() < min )                  // 剪左枝
            RangeQuery( root->rightchild() , min , max );
    else if ( root->value() >= min && root->value() <= max )// 左右递归
    {
            RangeQuery( root->leftchild() , min , max );
            AfxMessageBox( root->value() );
            RangeQuery( root->rightchild() , min , max );
    }
    else if ( root->value() > max )             // 剪右枝
            RangeQuery( root->leftchild() , min , max );
}
```

下面这个算法与上面的等价，代码有些冗余

```
template <class T>
void BinarySearchTree<T>::RangeQuery(BinaryTreeNode<T> *root, T min, T max)
{
    if ( root == NULL )                         // 结点空，返回
            return;
/*  判断叶结点的情况可以不考虑，考虑了也正确，不扣分。
    if ( root->isLeaf() )
    {
            if ( root->value() >= min && root->value() <= max )
                    AfxMessageBox( root->value() );
            return;
    }*/
    if ( root->value() < min )                  // 剪左枝
    {
            RangeQuery( root->rightchild() , min , max );
    }
    else if ( root->value() == min )            // 结点值为 min，打印，并剪掉左
    {
            AfxMessageBox( root->value() );
            RangeQuery( root->rightchild() , min , max );
```

```
    }
    else if ( root->value() > min && root->value() <= max )// (min, max]，左右递归
    {
            RangeQuery( root->leftchild() , min , max );
            AfxMessageBox( root->value() );
            RangeQuery( root->rightchild() , min , max );
    }
    else if ( root->value() > max )                    // 剪右枝
    {
            RangeQuery( root->leftchild() , min , max );
    }
}
```

解法 3：按照中序周游整个二叉树，在那个 visit 语句处，打印满足条件的值。此方法效率不高，但题目并没有限制，因此扣两分，最高分只给 13 分

```
template<class T>
void BinaryTree<T>::DepthOrder (BinaryTreeNode<T>* root, T min, T max) {
    if(root!=NULL){
        DepthOrder(root->leftchild());                    // 递归访问左子树
        if  ( root->value() >= min && root->value() <= max )
            Printout(root->value());          // 打印在范围内的当前结点
        DepthOrder(root->rightchild());       // 递归访问右子树
}
```

代价分析：设二叉树的结点个数为 n，则该算法把二叉树的每个结点访问且只访问一次，代价为 O（n）。

3. **（14 分）**试编写一个算法，计算一棵给定二叉树的单孩子结点数。

评分标准：**算法思想占 2 分（如果不写思想则扣他 5 分），注释占 2 分，算法框架 8 分，代价分析 2 分。**

参考答案：

如果学生算法正确，不需要计较 left 和 right 的用法，因为这里的没有定义类，也没有规定成员名。例如 root->left()和 root->right()，root->leftChild()、root->rightChild()都可以。

解法 1：用递归的 DFS 框架，

模型如下：

$$f(b)=\begin{cases} 0 & \text{若 b=NULL} \\ 1+ f(b->left) + f(b->right) & \text{若 b->left=NULL 且 b->right≠NULL 或 b->left≠NULL 且 b->right=NULL} \\ f(b->left)+ f(b->right) & \text{其它情况} \end{cases}$$

因此本题算法如下：

```
int onechild(BinTree *b)
{
    int num1, num2, n=0;
    if (b==NULL)                                          // 空结点返回 0
        return 0;
    else if( (b->left==NULL && b->right!=NULL) ||         // 单亲的结点
        b->left!=NULL && b->right==NULL) )
            n=1;
    num1 = onechild(b->left);                             // 递归计算左边的情况
    num2 = onechild(b->right);                            // 递归计算右边的情况
    retrun (num1+num2+n);                                 // 返回左右和自身的和
}
```

注释：本算法也可以修改为 void onechild(BinTree *b, int &n)的形式，通过引用形的参数来返回单亲数目

代价分析：从判断是否为单亲的核心语句来看，算法把二叉树的每个结点访问且只访问一次，代价为O（n）。

解法 2：用 BFS 框架，宽度优先周游，在访问每个结点时，判断“是否单亲结点则累”，若是则单亲结点数目增加 1。

```
template<class T>
int BinaryTree<T>::LevelOrder
(BinaryTreeNode<T>* root)
{
        using std::queue;     //使用 STL 的队列
        queue<BinaryTreeNode<T>*> aQueue;
        int count=0;          //单亲结点计数
        BinaryTreeNode<T>* pointer=root;
        if(pointer)
                aQueue.push(pointer);
        while(!aQueue.empty()) {
                //取队列首结点
                pointer=aQueue.front();
                //对本结点判断是否单亲结点
                if ( (pointer -> leftchild()==NULL
                                && pointer -> leftchild()!=NULL)
                        || pointer -> leftchild()!=NULL
                                && pointer -> leftchild()==NULL) )
                        count++;
                aQueue.pop();
                if(pointer->leftchild())            //左子树进队列
                        aQueue.push(pointer->leftchild());
                if(pointer->rightchild())           //右子树进队列
                        aQueue.push(pointer-> leftchild());
        }//end while
        return count;
}
```

解法 3： 利用填空 4 的结论，周游二叉树时，数总数 n 和叶结点个数 n0，可知有两个孩子的结点为 n0-1，单亲结点就是 n-(2*n0-1)个。代价为 O(n)