数据结构作业(第六次)

PB20111686 黄瑞轩

5.34

这里使用下面所述的储存方式。这里认为当表是空表时，整个表为`NULL`。

```
typedef enum { ATOM, LIST }ElemTag;
typedef struct GLNode {
    ElemTag tag;
    union {
        AtomType atom;
        struct GLNode* hp;
    };
    struct GLNode* tp;
}*GList;
```

算法为：

```
void Reverse(GList L) {
    if (L == NULL) return;       // 空表
    if (L->tp == NULL) {
        if (L->tag) {
            Reverse(L->hp);
        }
        return;
    }
    GList p = NULL, r = L, n = L->tp;
    while (1) {
        if (n->tp == NULL) {
            if (r->tag)Reverse(r);
            r->tp = p;
            L->hp = n;
            if (n->tag)Reverse(n);
            break;
        }
        if (r->tag)Reverse(r);
        r->tp = p;
        p = r;
        r = n;
        n = n->tp;
    }
    // 在2.22的逆置函数上多加了一步，即在遍历的时候，每一次都判断是不是子表，如果是子表则先进入递归，直到判断出不是子表，按原逆置函数逆转，回退到上一步进行逆转。
}
```

6.36

二叉链表储存结构如下：

```
typedef struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
}* BT;
```

算法如下：

```
bool Sim(BT a, BT b) {
    if (a == nullptr && b == nullptr)return true;
    if (a == nullptr && b != nullptr || a != nullptr && b == nullptr)return false;
    return Sim(a->left, b->left) && Sim(a->right, b->right);
}
```

6.43

```
void Exchange(BT t) {
    if (t == nullptr)return;
    BT tmp = t->left;
    t->left = t->right;
    t->right = tmp;
    Exchange(t->left);
    Exchange(t->right);
}
```

6.48

```
bool Find(BT t, BT p, BT q) {
    //看以t为根节点的二叉树中是否有p或q
    if (t == nullptr)return false;
    //如果t是空则返回否
    if (t == p || t == q)return true;
    //如果t本身就是p或q则返回是
    return Find(t->left, p, q) || Find(t->right, p, q);
    //在t的左子树和右子树中分别查找
}
BT Ancestor(BT root, BT p, BT q) {
    //返回以root作为根节点的二叉树中距离p和q最近的公共祖先（也即深度最大）
    if (root == nullptr)return nullptr;
    //如果本身是空，则返回空
    if (Find(root->left, p, q) && Find(root->right, p, q))return root;
    //如果root的左子树有p或q，右子树也有，则root就是最近的公共祖先
    if ((root == p || root == q) && (Find(root->left) || Find(root->right)))return root;
    //如果root本身就是p或q，且root的一侧子树中有p或q，则root就是最近的公共祖先
    return Ancestor(root->left, p, q) | Ancestor(root->right, p, q);
    //如果以上情况都不满足，就说明当前节点一定不是我们要找的节点，递归搜索root的左右子树是否满足上述情况。如果一侧子树不满足条件将会返回nullptr，因此两侧只会返回一个有效指针，做|运算的结果一定是这个有效指针。
}
```