写一个程序，在一棵二叉树中查找最近共同祖先

【解】假定二叉树采用顺序存储。我们在第5章实现的顺序存储的二叉树类中增加了一个求最近共同祖先的函数LCA。函数的原型为

```
T LCA(T x, T y);
```

该函数的两个参数是树上的两个元素值，返回这两个元素的最近共同祖先。

我们采用递归的思想来解决这个问题。如果 x 和 y 是同一元素，最近共同祖先就是自己。否则找层次比较低的结点的父结点和另一结点的最近共同祖先。例如在图 11-4 的二叉树上找 0 和 4 的最近共同祖先。先检查 0 和 4 是否同一结点，显然这是两个结点。接着把问题转换成找层次比较低的结点的父结点和另一个结点的最近共同祖先。在当前的例子中，层次比较低的是 0，于是就找 0 的父结点 1 和 4 的最近共同祖先。1 和 4 也不是同一结点，但层次是相同的。我们任意取一个结点的父结点，例如取 4 的父结点 3，接着找 1 和 3 的最近共同祖先。同样将问题转换成找 1 的父结点和 3 的最近共同祖先。1 的父结点是 3，于是可得 0 和 4 的最近共同祖先是 3。

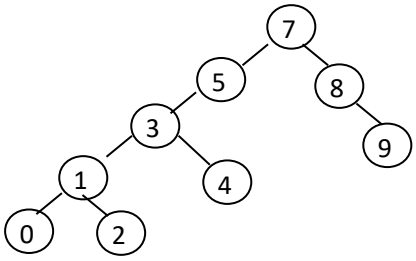

图 11-4 二叉树示例

这个过程在顺序存储中很容易实现，因为找父结点的操作很容易实现。序号为 i 的结点的父结点的序号为 i/2。在顺序存储中，决定两个结点的层次谁深谁浅也很容易，只要判断序号就可以了。序号大的结点的层次肯定大于等于序号小的结点的层次。根据这个思想定义了一个私有的 LCA 函数。函数的原型为

```
int LCA(int i, int j);
```

私有的 LCA 函数寻找序号为 i 的结点和序号为 j 的结点的最近共同祖先，返回共同祖先的序号。这个函数的定义见代码清单 11-8。

代码清单 11-8 私有的 LCA 函数的实现

```
1.  template <class T>
2.  int BinaryTree<T>::LCA(int i, int j)
3.  {  if (i == j) return i;
4.     if (i > j) return LCA(i / 2, j);
5.        else return LCA(i, j / 2);
6.  }
```

公有的 LCA 函数调用私有的 LCA 函数实现查找。公有的 LCA 函数先查找元素 x 和 y 的序号，然后调用私有的 LCA，得到最近共同祖先的序号，返回该序号的元素值。具体定义见代码清单 11-9。

代码清单 11-9 公有的 LCA 函数的实现

```
7.  template <class T>
8.  T BinaryTree<T>::LCA(T x, T y)
9.  {    int xi, yi, i;  // xi 是 x 的下标， yi 是 y 的下标
10.      // 查找 x 的下标
11.      for (i = 1; i <= size; ++i)
12.          if (array[i] == x) break;
13.      if ( i <= size) xi = i;
14.      else {
15.          cout << "树上没有元素 " << x << endl;
16.          return x;
17.      }
18.
19.      // 查找 y 的下标
20.      for (i = 1; i <= size; ++i)
21.          if (array[i] == y) break;
22.      if ( i <= size) yi = i;
23.      else {
24.          cout << "树上没有元素 " << y << endl;
25.          return x;
26.      }
27.
28.      return array[LCA(xi, yi)];
29. }
```

这个实现实际上是借用了并查集的 find 操作的实现思想。假如树是以二叉链表的方式实现，该算法可以借助于树的后序遍历来实现。程序设计题 4 给出的就是链接实现的实例。