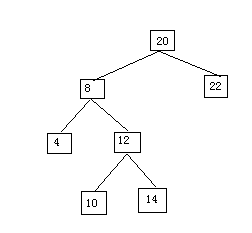
[题目]：已知二元搜索树（Binary Search Tree）上两个结点的值，请找出它们的公共祖先。你可以假设这两个值肯定存在。这个函数的调用接口如下所示：  
      int FindLowestCommonAncestor(node \*root, int value1, int value2);

    根据树的存储结构，我们可以立刻得到一个这样的算法：从两个给定的结点出发回溯，两条回溯路线的交点就是我们要找的东西。这个算法的具体实现办法是：从根结点开始，先用这两个结点的全体祖先分别生成两个链表，再把这两个链表第一次出现不同结点的位置找出来，则它们的前一个结点就是我们要找的东西。  
    这个算法倒没有什么不好的地方，但是它没有利用二元搜索树的任何特征，其他类型的树也可以用这个算法来处理。二元搜索树中左结点的值永远小于或者等于当前结点的值，而右结点的值永远大于或者等于当前结点的值。仔细研究，4和14的最低公共祖先是8，它与4和14的其他公共祖先是有重要区别的：其他的公共祖先或者同时大于4和4，或者同时小于4和14，只有8介于4和14之间。利用这一研究成果，我们就能得到一个更好的算法。  
    从根结点出发，沿着两个给定结点的公共祖先前进。当这两个结点的值同时小于当前结点的值时，沿当前结点的左指针前进；当这两个结点的值同时大于当前结点的值时，沿当前结点的右指针前进；当第一次遇到当前结点的值介于两个给定的结点值之间的情况时，这个当前结点就是我们要找的最的最低公共祖先了。  
    这是一道与树有关的试题，算法也有递归的味道，用递归来实现这一解决方案似乎是顺理成章的事，可这里没有这个必要。递归技术特别适合于对树的多个层次进行遍历或者需要寻找某个特殊结点的场合。这道题只是沿着树结点逐层向下前进，用循环语句来实现有关的过程将更简单明了。  
     int FindLowestCommonAncestor(node \*root, int value1, int value2)  
     {  
      node \*curnode = root;  
      while(1)  
         {  
           if (curnode->data>value1&&curnode->data>value2)  
             curnode = curnode->left;  
           else if(curnode->data<value1&&curnode->data<value2)  
             curnode = curnode->right;  
           else  
             return curnode->data;  
          }  
      }