```
36      private static final int DEFAULT_TREES = 1;
37
38      private int currentSize;              // # items in priority queue
39      private Node<AnyType> [ ] theTrees;  // An array of tree roots
40
41      private void expandTheTrees( int newNumTrees )
42        { /* See online code */ }
43      private Node<AnyType> combineTrees( Node<AnyType> t1, Node<AnyType> t2 )
44        { /* Figure 6.54 */ }
45
46      private int capacity( )
47        { return ( 1 << theTrees.length ) - 1; }
48      private int findMinIndex( )
49        { /* See online code */ }
50  }
```

图 6-52 （续）

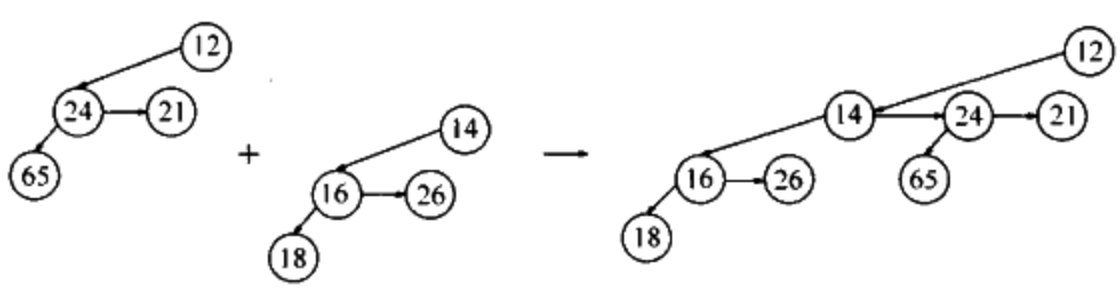

图 6-53 合并两棵二项树

```
1       /**
2        * Return the result of merging equal-sized t1 and t2.
3        */
4       private Node<AnyType> combineTrees( Node<AnyType> t1, Node<AnyType> t2 )
5       {
6           if( t1.element.compareTo( t2.element ) > 0 )
7               return combineTrees( t2, t1 );
8           t2.nextSibling = t1.leftChild;
9           t1.leftChild = t2;
10          return t1;
11      }
```

图 6-54 合并同样大小的两棵二项树的例程

现在我们介绍 merge 例程的简单实现。H_1 由当前的对象表示而 H_2 则用 rhs 表示。该例程将 H_1 和 H_2 合并，把合并结果放入 H_1 中，并清空 H_2。在任意时刻我们在处理的是秩(rank)为 i 的那些树。t_1 和 t_2 分别是 H_1 和 H_2 中的树，而 carry 是从上一步得来的树(它可能是 null)。从秩为 i 的树以及秩为 $i+1$ 的 carry 的树所形成的树，依赖于 8 种可能情形中的每一种。程序见图 6-55。对程序的改进在练习 6.35 中提出。

二项队列的 deleteMin 例程在图 6-56 中给出。

我们可以将二项队列扩展到支持二叉堆所允许的某些非标准的操作，诸如 decreaseKey 和 delete 等，前提是受到影响的元素的位置已知。decreaseKey 是一个 percolateUp，如果我们将一个域加到每个节点上存储其父链，那么这个操作可以在时间 $O(\log N)$ 内完成。一次任意的