```
1       /**
2        * Remove the smallest item from the priority queue.
3        * @return the smallest item, or throw UnderflowException, if empty.
4        */
5       public AnyType deleteMin( )
6       {
7           if( isEmpty( ) )
8               throw new UnderflowException( );
9
10          AnyType minItem = findMin( );
11          array[ 1 ] = array[ currentSize-- ];
12          percolateDown( 1 );
13
14          return minItem;
15      }
16
17      /**
18       * Internal method to percolate down in the heap.
19       * @param hole the index at which the percolate begins.
20       */
21      private void percolateDown( int hole )
22      {
23          int child;
24          AnyType tmp = array[ hole ];
25
26          for( ; hole * 2 <= currentSize; hole = child )
27          {
28              child = hole * 2;
29              if( child != currentSize &&
30                      array[ child + 1 ].compareTo( array[ child ] ) < 0 )
31                  child++;
32              if( array[ child ].compareTo( tmp ) < 0 )
33                  array[ hole ] = array[ child ];
34              else
35                  break;
36          }
37          array[ hole ] = tmp;
38      }
```

图 6-12 在二叉堆中执行 deleteMin 的方法

对整个堆进行线性搜索，是没有办法找出任何特定的关键字的。为说明这一点，考虑图 6-13 所示的大型堆结构(具体元素没有标出)，我们在这里看到，关于最大值的元素所知道的唯一信息是：该元素在树叶上。但是，半数的元素位于树叶上，因此该信息是没什么价值的。由于这个原因，如果重要的是要知道元素都在什么地方，那么除堆之外，还必须用到诸如散列表等某些其他数据结构(回忆：该模型并不允许查看堆内部)。