用链。图 6-3 中的数组对应图 6-2 中的堆。

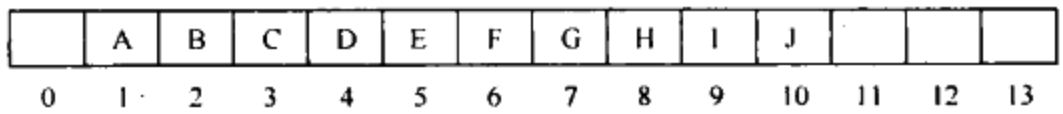

图 6-3 完全二叉树的数组实现

对于数组中任一位置 i 上的元素，其左儿子在位置 $2i$ 上，右儿子在左儿子后的单元 $(2i+1)$ 中，它的父亲则在位置 $\lfloor i/2 \rfloor$ 上。因此，这里不仅不需要链，而且遍历该树所需要的操作极简单，在大部分计算机上运行很可能非常快。这种实现方法的唯一问题在于，最大的堆大小需要事先估计，但一般这并不成问题(而且如果需要,我们可以重新调整大小)。在图 6-3 中，堆大小的限界是 13 个元素。该数组有一个位置 0，后面将详细叙述。

因此，一个堆结构将由一个(Comparable 对象的)数组和一个代表当前堆的大小的整数组成。图 6-4 显示一个优先队列的架构。

```
1   public class BinaryHeap<AnyType extends Comparable<? super AnyType>>
2   {
3       public BinaryHeap( )
4         { /* See online code */ }
5       public BinaryHeap( int capacity )
6         { /* See online code */ }
7       public BinaryHeap( AnyType [ ] items )
8         { /* Figure 6.14 */ }
9
10      public void insert( AnyType x )
11        { /* Figure 6.8 */ }
12      public AnyType findMin( )
13        { /* See online code */ }
14      public AnyType deleteMin( )
15        { /* Figure 6.12 */ }
16      public boolean isEmpty( )
17        { /* See online code */ }
18      public void makeEmpty( )
19        { /* See online code */ }
20
21      private static final int DEFAULT_CAPACITY = 10;
22
23      private int currentSize;      // Number of elements in heap
24      private AnyType [ ] array;    // The heap array
25
26      private void percolateDown( int hole )
27        { /* Figure 6.12 */ }
28      private void buildHeap( )
29        { /* Figure 6.14 */ }
30      private void enlargeArray( int newSize )
31        { /* See online code */ }
32  }
```

图 6-4 优先队列的类架构

本章我们将始终把堆画成树，这意味着具体的实现将使用简单的数组。