```
void mark(ptr p) {
   if ((b = isPtr(p)) == NULL)
     return;
   if (blockMarked(b))
     return;
   markBlock(b);
   len = length(b);
   for (i=0; i < len; i++)
     mark(b[i]);
   return;
}
```

a）mark 函数

```
void sweep(ptr b, ptr end) {
   while (b < end) {
      if (blockMarked(b))
         unmarkBlock(b);
      else if (blockAllocated(b))
         free(b);
      b = nextBlock(b);
   }
   return;
}
```

b）sweep函数

图 9-51　mark 和 sweep 函数的伪代码

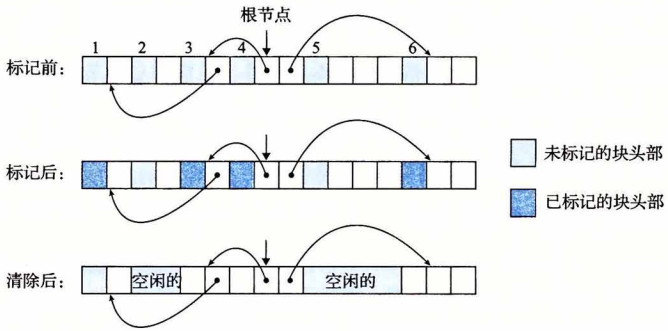

图 9-52　Mark&Sweep 示例。注意这个示例中的箭头表示内存引用，而不是空闲链表指针

初始情况下，图 9-52 中的堆由六个已分配块组成，其中每个块都是未分配的。第 3 块包含一个指向第 1 块的指针。第 4 块包含指向第 3 块和第 6 块的指针。根指向第 4 块。在标记阶段之后，第 1 块、第 3 块、第 4 块和第 6 块被做了标记，因为它们是从根节点可达的。第 2 块和第 5 块是未标记的，因为它们是不可达的。在清除阶段之后，这两个不可达块被回收到空闲链表。

9.10.3　C 程序的保守 Mark & Sweep

Mark&Sweep 对 C 程序的垃圾收集是一种合适的方法，因为它可以就地工作，而不需要移动任何块。然而，C 语言为 isPtr 函数的实现造成了一些有趣的挑战。

第一，C 不会用任何类型信息来标记内存位置。因此，对 isPtr 没有一种明显的方式来判断它的输入参数 p 是不是一个指针。第二，即使我们知道 p 是一个指针，对 isPtr 也没有明显的方式来判断 p 是否指向一个已分配块的有效载荷中的某个位置。

对后一问题的解决方法是将已分配块集合维护成一棵平衡二叉树，这棵树保持着这样一个属性：左子树中的所有块都放在较小的地址处，而右子树中的所有块都放在较大的地址处。如图 9-53 所示，这就要求每个已分配块的头部里有两个附加字段（left 和 right）。每个字段指向某个已分配块的头部。isPtr(ptr p)函数用树来执行对已分配块的二分查找。在每一步中，它依赖于块头部中的大小字段来判断 p 是否落在这个块的范围之内。