块的头部(第 14 行)。最后，在很可能出现的前一个堆以一个空闲块结束的情况中，我们调用 coalesce 函数来合并两个空闲块，并返回指向合并后的块的块指针(第 17 行)。

4. 释放和合并块

应用通过调用 mm_free 函数(图 9-46)，来释放一个以前分配的块，这个函数释放所请求的块(bp)，然后使用 9.9.11 节中描述的边界标记合并技术将之与邻接的空闲块合并起来。

code/vm/malloc/mm.c

```
1   void mm_free(void *bp)
2   {
3       size_t size = GET_SIZE(HDRP(bp));
4
5       PUT(HDRP(bp), PACK(size, 0));
6       PUT(FTRP(bp), PACK(size, 0));
7       coalesce(bp);
8   }
9
10  static void *coalesce(void *bp)
11  {
12      size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
13      size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
14      size_t size = GET_SIZE(HDRP(bp));
15
16      if (prev_alloc && next_alloc) {            /* Case 1 */
17          return bp;
18      }
19
20      else if (prev_alloc && !next_alloc) {      /* Case 2 */
21          size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
22          PUT(HDRP(bp), PACK(size, 0));
23          PUT(FTRP(bp), PACK(size,0));
24      }
25
26      else if (!prev_alloc && next_alloc) {      /* Case 3 */
27          size += GET_SIZE(HDRP(PREV_BLKP(bp)));
28          PUT(FTRP(bp), PACK(size, 0));
29          PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
30          bp = PREV_BLKP(bp);
31      }
32
33      else {                                     /* Case 4 */
34          size += GET_SIZE(HDRP(PREV_BLKP(bp))) +
35              GET_SIZE(FTRP(NEXT_BLKP(bp)));
36          PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
37          PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
38          bp = PREV_BLKP(bp);
39      }
40      return bp;
41  }
```

code/vm/malloc/mm.c

图 9-46　mm_free：释放一个块，并使用边界标记合并将之与所有的邻接空闲块在常数时间内合并