13.4.3 空间-时间的权衡

改进性能的一种一般性的方法是权衡时间与空间。在一个使用很少内存但是速度比较慢的算法与一个使用很多内存但是速度更快的算法之间进行选择，这在计算机科学中是经常发生的事情。在做出重要的优化时，值得寻找通过使用更多内存加快了速度的算法，或者反过来通过做更多的计算节省了宝贵的内存的算法。

一种常用而有益的技术是用宏来代替小的过程。使用宏消除了通常与过程调用相关联的开销。如果调用出现在一个循环的内部，这种获益尤其显著。例如，假设我们使用位图来跟踪资源，并且经常需要了解在位图的某一部分中有多少个单元是空闲的。为此，我们需要一个过程bit_count来计数一个字节中值为1的位的个数。图13-7a中给出了简单明了的过程。它对一个字节中的各个位循环，每次计数它们一次。

```
#define BYTE_SIZE 8                          /* 一个字节包含8个位 */
int bit_count(int byte)
{                                            /* 对一个字节中的位进行计数 */
    int i, count = 0;
    for (i = 0; i < BYTE_SIZE; i++)          /* 对一个字节中的各个位循环 */
        if ((byte >> i) & 1) count++;        /* 如果该位是1，计数加1 */
    return(count);                           /* 返回和 */
}
```

a)

```
/* 将一个字节中的位相加并且返回和的宏 */
#define bit_count(b)((b&1) + ((b>>1)&1) + ((b>>2)&1) + ((b>>3)&1) + \
                     ((b>>4)&1) + ((b>>5)&1) + ((b>>6)&1) + ((b>>7)&1))
```

b)

```
/* 在一个表中查找位计数的宏 */
char bits[256] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3, 3, ...};
#define bit_count(b) (int) bits[b]
```

c)

图13-7 a) 对一个字节中的位进行计数的过程；b) 对位进行计数的宏；c) 在表中查找位计数

该过程有两个低效的根源。首先，它必须被调用，必须为它分配栈空间，并且必须返回。每个过程调用都有这个开销。第二，它包含一个循环，并且总是存在与循环相关联的某些开销。

一种完全不同的方法是使用图13-7b中的宏。这个宏是一个内联表达式，它通过对参数连续地移位，屏蔽除低位以外的其他位，并且将8个项相加，这样来计算位的和。这个宏决不是一件艺术作品，但是它只在代码中出现一次。当这个宏被调用时，例如通过

```
sum = bit_count(table[i]);
```

这个宏调用看起来与过程调用等同。因此，除了定义有一点凌乱以外，宏中的代码看上去并不比过程中的代码要差，但是它的效率更高，因为它消除了过程调用的开销和循环的开销。

我们可以更进一步研究这个例子。究竟为什么计算位计数？为什么不在一个表中查找？毕竟只有256个不同的字节，每个字节具有0到8之间的惟一的值。我们可以声明一个256项的表bits，每一项（在编译时）初始化成对应于该字节值的位计数。采用这一方法在运行时根本就不需要计算，只要一个变址操作就可以了。图13-7c中给出了做这一工作的宏。

这是用内存换取计算时间的明显的例子。然而，我们还可以再进一步。如果需要整个32位字的位计数，使用我们的bit_count宏，每个字我们需要执行四次查找。如果将表扩展到65 536项，每个字查找两次就足够了，代价是更大的表。

在表中查找答案可以用在其他方面。例如，在第7章中，我们看到了JPEG图像压缩是怎样工作的，它使用了相当复杂的离散余弦变换。另一种压缩技术GIF使用表查找来编码24位RGB图像。然而，GIF只对具有256种颜色或更少颜色的图像起作用。对于每幅要压缩的图像，构造一个256项的调色板，每一项包含一个24位的RGB值。压缩过的图像于是包含每个像素的8位索引，而不是24位颜色值，增益因子