为3。图13-8中针对一幅图像的一个4×4区域说明了这一思想。原始未压缩的图像如图13-8a所示，该图中每个取值是一个24位的值，每8位给出红、绿和蓝的强度。GIF图像如图13-8b所示，该图中每个取值是一个进入调色板的8位索引。调色板作为图像文件的一部分存放，如图13-8c所示。实际上，GIF算法的内容比这要多，但是思想的核心是表查找。

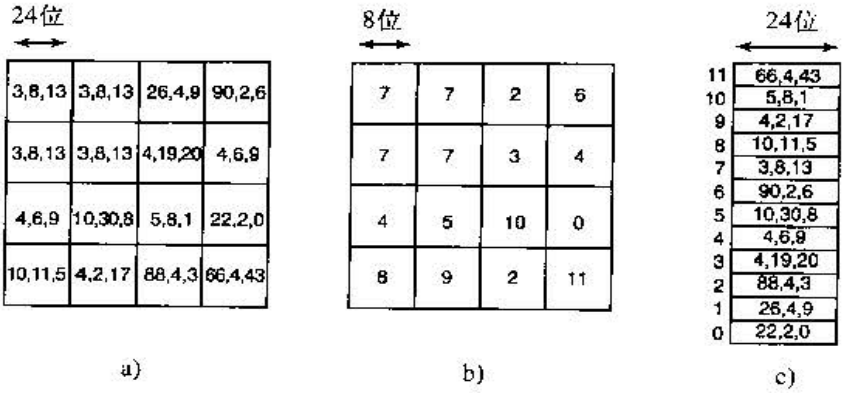

图13-8 a) 每个像素24位的未压缩图像的局部，b) 以GIF压缩的相同局部，每个像素8位，c) 调色板

存在减少图像大小的另一种方法，并且这种方法说明了一种不同的权衡。PostScript是一种程序设计语言，可以用来描述图像。（实际上，任何程序设计语言都可以描述图像，但是PostScript专为这一目的进行了调节。）许多打印机具有内嵌的PostScript解释器，能够运行发送给它们的PostScript程序。

例如，如果在一幅图像中存在一个像素矩形块具有相同的颜色，用于该图像的PostScript程序将携带指令，用来将一个矩形放置在一定的位置并且用一定的颜色填充该矩形。只需要少数几个位就可以发出此命令。当打印机接收图像时，打印机中的解释器必须运行程序才能绘制出图像。因此，PostScript以更多的计算为代价实现了数据压缩，这是与表查找不同的一种权衡，但是当内存或带宽不足时是颇有价值的。

其他的权衡经常牵涉数据结构。双向链表比单向链表占据更多的内存，但是经常使得访问表项速度更快。散列表甚至更浪费空间，但是要更快。简而言之，当优化一段代码时要考虑的重要事情之一是：使用不同的数据结构是否将产生最佳的时间-空间平衡。

13.4.4 高速缓存

用于改进性能的一种众所周知的技术是高速缓存。在任何相同的结果可能需要多次的情况下，高速缓存都是适用的。一般的方法是首先做完整的工作，然后将结果保存在高速缓存中。对于后来的尝试，首先要检查高速缓存。如果结果在高速缓存中，就使用它。否则，再做完整的工作。

我们已经看到高速缓存在文件系统内部的运用，在高速缓存中保存一定数目最近用过的磁盘块，这样在每次命中时就可以省略磁盘读操作。然而，高速缓存还可以用于许多其他目的。例如，解析路径名就代价高昂得令人吃惊。再次考虑图4-35中UNIX的例子。为了查找/usr/ast/mbox，需要如下的磁盘访问：

1) 读入根目录的i节点（i节点1）。
2) 读入根目录（磁盘块1）。
3) 读入/usr的i节点（i节点6）。
4) 读入/usr目录（磁盘块132）。
5) 读入/usr/ast的i节点（i节点26）。
6) 读入/usr/ast目录（磁盘块406）。

只是为了获得文件的i节点号就需要6次磁盘访问。然后必须读入i节点本身以获得磁盘块号。如果文件小于块的大小（例如1024字节），那么需要8次磁盘访问才读到数据。

某些系统通过对（路径，i节点）的组合进行高速缓存来优化路径名的解析。对于图4-35的例子，在解析/usr/ast/mbox之后，高速缓存中肯定会保存图13-9的前三项。最后三项来自解析其他路径。

路　径	i节点号
/usr	6
/usr/ast	26
/usr/ast/mbox	60
/usr/ast/books	92
/usr/bal	45
/usr/bal/paper.ps	85

图13-9 图4-35的i节点高速缓存的局部