第5章　散　　列

我们在第4章讨论了查找树 ADT，它允许对元素的集合进行各种操作。本章讨论**散列表**(hash table)ADT，不过它只支持二叉查找树所允许的一部分操作。

散列表的实现常常叫做**散列**(hashing)。散列是一种用于以常数平均时间执行插入、删除和查找的技术。但是，那些需要元素间任何排序信息的树操作将不会得到有效的支持。因此，诸如findMin、findMax 以及以线性时间将排过序的整个表进行打印的操作都是散列所不支持的。

本章的中心数据结构是散列表。我们将

- 看到实现散列表的几种方法。
- 解析地比较这些方法。
- 介绍散列的多种应用。
- 将散列表和二叉查找树进行比较。

5.1　一般想法

理想的散列表数据结构只不过是一个包含一些项(item)的具有固定大小的数组。第4章讨论过，通常查找是对项的某个部分(即数据域)进行的。这部分就叫做**关键字**(key)。例如，项可以由一个串(它可以作为关键字)和其他一些数据域组成(例如，姓名是大型雇员结构的一部分)。我们把表的大小记作 *TableSize*，并将其理解为散列数据结构的一部分，而不仅仅是浮动于全局的某个变量。通常的习惯是让表从0到 *TableSize* − 1 变化；稍后我们就会明白为什么要这样做。

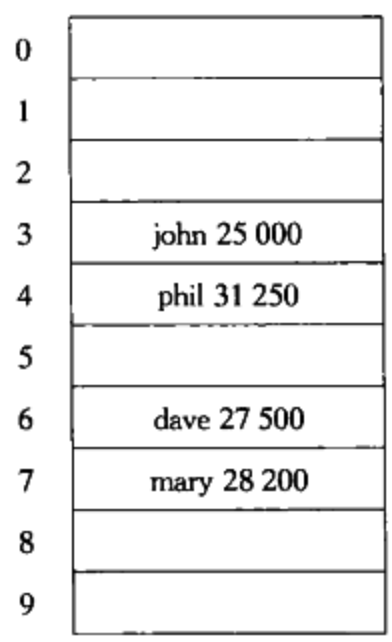

图 5-1　一个理想的散列表

每个关键字被映射到从0到 *TableSize* − 1 这个范围中的某个数，并且被放到适当的单元中。这个映射就叫做散列函数(hash function)，理想情况下它应该计算起来简单，并且应该保证任何两个不同的关键字映射到不同的单元。不过，这是不可能的，因为单元的数目是有限的，而关键字实际上是用不完的。因此，我们寻找一个散列函数，该函数要在单元之间均匀地分配关键字。图 5-1 是完美情况的一个典型。在这个例子中，john 散列到 3，phil 散列到 4，dave 散列到 6，mary 散列到 7。

这就是散列的基本想法。剩下的问题就是要选择一个函数，决定当两个关键字散列到同一个值的时候(这叫做**冲突**(collision))应该做什么以及如何确定散列表的大小。

5.2　散列函数

如果输入的关键字是整数，则一般合理的方法就是直接返回 Key mod *Tablesize*，除非 Key 碰巧具有某些不合乎需要的性质。在这种情况下，散列函数的选择需要仔细地考虑。例如，若表的大小是10而关键字都以0为个位，则此时上述标准的散列函数就是一个不好的选择。其原因我