**3.4 散列表**

使用散列的查找算法分为两步：1，用散列函数将被查找的键转化为数组的一个索引；2，处理碰撞冲突

散列表是算法在时间和空间上做出权衡的经典例子。

使用散列表，你可以实现在一般应用中拥有（均摊后）常数级别的查找和插入操作的符号表。

这使得它在很多情况下成为实现简单符号表的最佳选择。

**3.4.1 散列函数**

我们要找的散列函数应该易于计算并且能够均匀分布所有的键，即对于任意键，0到M-1之间的每个整数都有相等的可能性与之相关（与键无关）。

散列函数和键的类型有关。严格地说，对于每种类型的键我们都需要一个与之对应的散列函数。

正整数：

将整数散列最常用方法是除留余数法。选择大小为素数M的数组，对于任意正整数k，计算k除以M的余数。

浮点数：

如果键是0到1之间的实数，我们可以将它乘以M并四舍五入得到一个0至M-1之间的索引值。

或者将键表示为二进制数然后再使用除留余数法。

字符串：

除留余数法也可以处理较长的键，例如字符串，我们只需将它们当作大整数即可。

组合键：

如果键的类型含有多个类型变量，我们可以和String类型一样将它们混合起来。

Java约定每一种数据类型的hashCode()方法都必须和equals()方法一致。

Java为很多常用的数据类型重写了hashCode方法（包括String,Integer,Double,File和URL），默认散列函数会返回对象的内存地址。

要为一个数据类型实现一个优秀的散列方法需要满足三个条件：一致性，高效性和均匀性

**3.4.2 基于拉链法的散列表**

碰撞处理也就是处理两个或多个键的散列值相同的情况。

一种方法是将大小为M的数组中的每个元素都指向一条链表，链表中的每个结点都存储了散列值为该元素的索引的键值对。这种方法称为拉链法。

拉链法的一种实现方法是使用原始的链表数据类型，另一种更简单的方法（效率稍低），为M个元素分别构建符号表来保存散列到这里的键。

编程：基于拉链法的散列表

在一张含有M条链接和N个键的散列表中，任意一条链表中的键的数量均在N/M的常数因子范围内的概率无限趋向于1.

在一张含有M条链接和N个键的散列表中，未命中查找和插入操作所需的比较次数为~N/M。

在键的顺序并不重要的应用中，拉链法可能是最快的符号表实现。

**3.4.3 基于线性探测法的散列表**

实现散列表的另一种方式就是用大小为M的数组保存N个键值对，其中M>N。我们需要依靠数组中的空位解决碰撞冲突。基于这种策略的所有方法称为开放地址散列表。

开放地址散列表中最简单的方法叫做线性探测法。当发生碰撞时，我们直接检查散列表中的下一个位置。

我们用散列函数找到键在数组中的索引，检查其中的键和被查找的键是否相同。如果不同则继续查找，直到找到该键或者遇到一个空元素。

编程：基于线性探测的符号表

删除操作：

如何从基于线性探测的散列表中删除一个键？直接将该键所在的位置设为null是不行的，因为这会使得此位置之后的元素无法被查找。

我们需要将簇中被删除键的右侧的所有键重新插入散列表。

编程：删除元素

a是表中已被占用空间的比例，它是不可能大于1的。为了保证性能，我们会动态调整数组大小保证使用率在1/8到1/2之间。

线性探测的平均成本取决于元素在插入数组后聚集成的一组连续的条目，叫做键簇。

在一张大小为M并含有N=aM个键的基于线性探测的散列表中，基于假设J，命中和未命中的查找所需的探测次数分别为：

~1/2(1+1/(1-a))和~1/2(1+1/(1-a)的平方)

当散列表快满的时候查找所需的探测次数是巨大的，当使用率小于1/2时探测的预计次数在1.5到2.5之间。为此，我们考虑动态调整散列数组的大小。

**3.4.4 调整数组大小**

编程：调整线性探测散列表

**3.4.5 内存使用**