1-6 哈希表

在哈希表这种数据结构中，使用将在5-3节讲解的“哈希函数”，可以使数据的查询效率得到显著提升。

参考：5-3 哈希函数

哈希表存储的是由键（key）和值（value）组成的数据。例如，我们将每个人的性别作为数据进行存储，键为人名，值为对应的性别。

为了和哈希表进行对比，我们先将这些数据存储在数组中（数组的详细讲解在1-3节）。

参考：1-3 数组

此处准备了6个箱子（即长度为6的数组）来存储数据。假设我们需要查询Ally的性别，由于不知道Ally的数据存储在哪个箱子里，所以只能从头开始查询。这个操作便叫作“线性查找”（线性查找的讲解在3-1节）。

参考：3-1 线性查找

一般来说，我们可以把键当成数据的标识符，把值当成数据的内容。

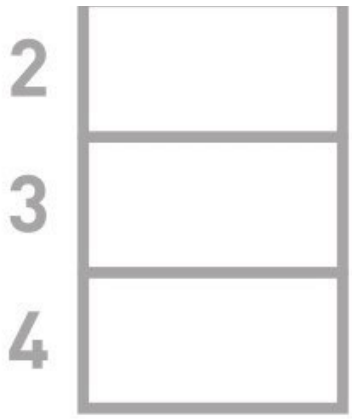
0号箱子中存储的键是Joe而不是Ally。

1号箱子中的也不是Ally。

同样，2号、3号箱子中的也都不是Ally。

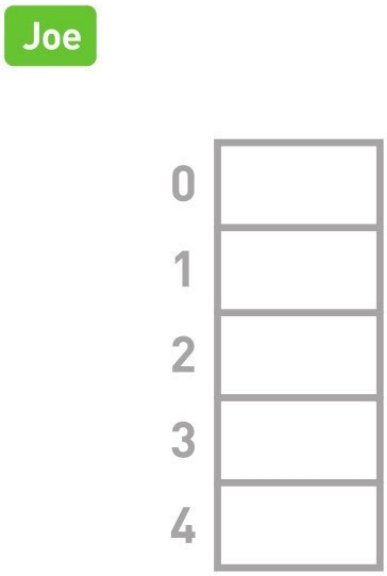
查找到4号箱子的时候，发现其中数据的键为Ally。把键对应的值取出，我们就知道Ally的性别为女（F）了。

数据量越多，线性查找耗费的时间就越长。由此可知：由于数据的查询较为耗时，所以此处并不适合使用数组来存储数据。



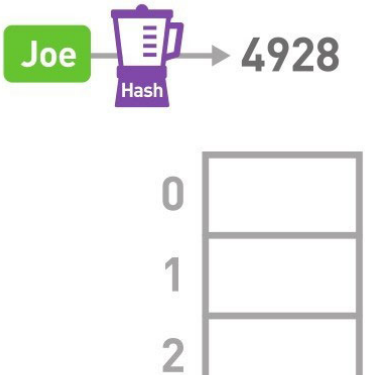
但使用哈希表便可以解决这个问题。首先准备好数组，这次我们用5个箱子的数组来存储数据。

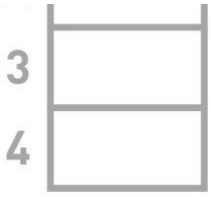
10



尝试把Joe存进去。

11

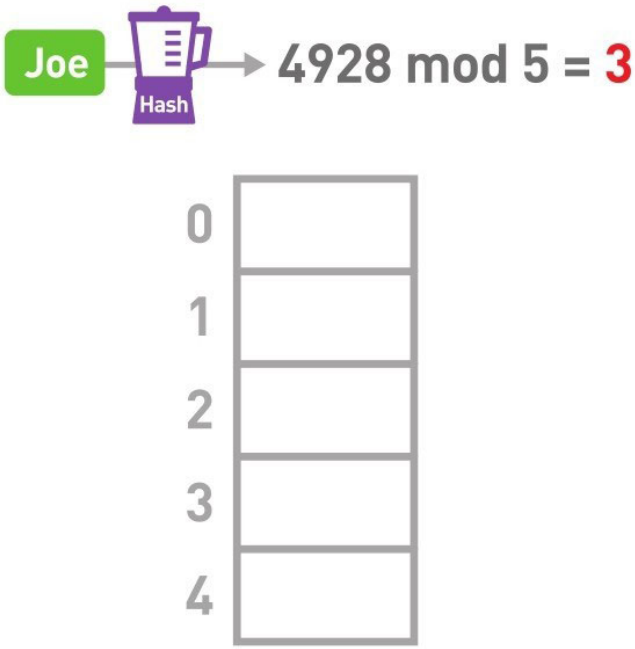




使用哈希函数（Hash）计算Joe的键，也就是字符串“Joe”的哈希值。得到的结果为4928（哈希函数的详细说明在5-3节）。

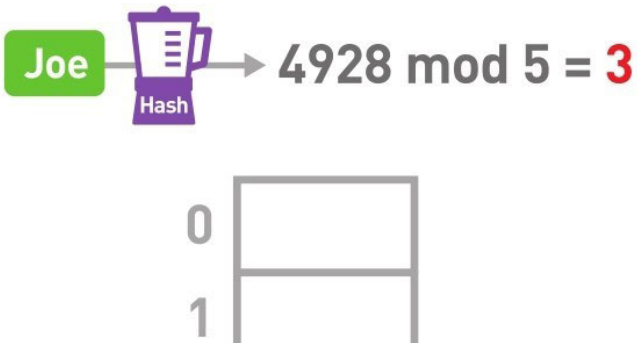
参考：5-3 哈希函数

12



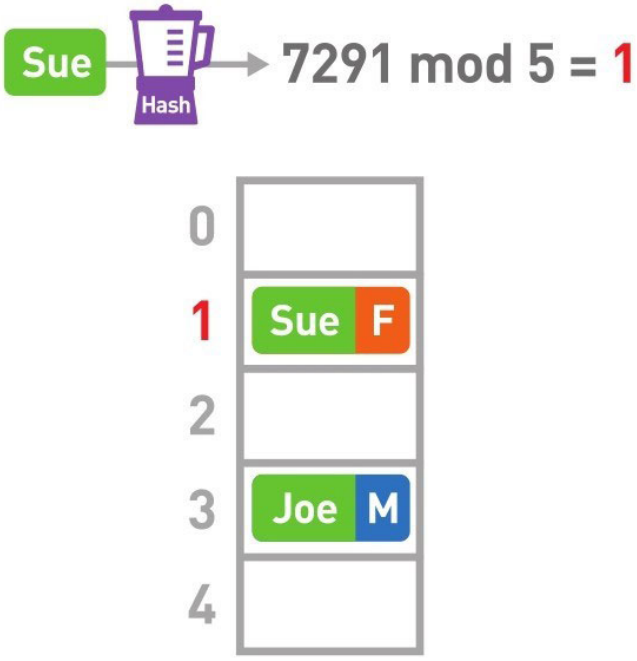
将得到的哈希值除以数组的长度5，求得其余数。这样的求余运算叫作“mod运算”。此处mod运算的结果为3。

13



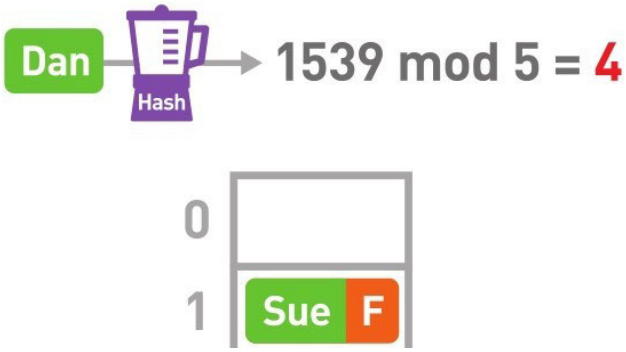
因此，我们将Joe的数据存进数组的3号箱子中。重复前面的操作，将其他数据也存进数组中。

14



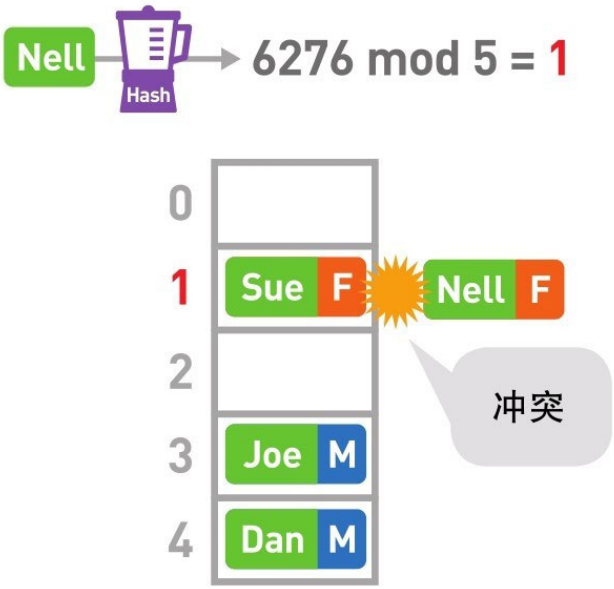
Sue键的哈希值为7291, mod 5的结果为1，将Sue的数据存进1号箱中。

15



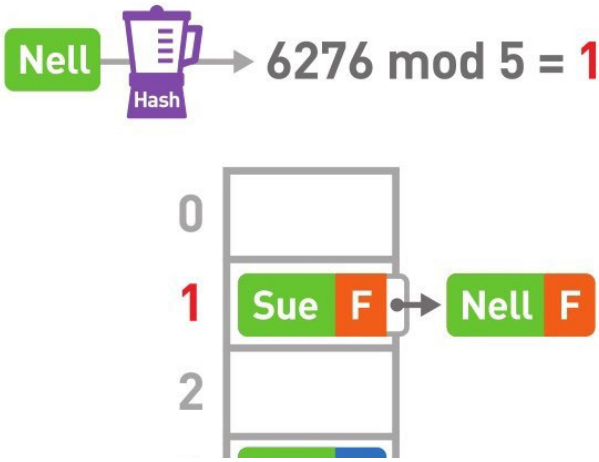
Dan键的哈希值为1539, mod 5的结果为4，将Dan的数据存进4号箱中。

16



Nell键的哈希值为6276, mod 5的结果为1。本应将其存进数组的1号箱中，但此时1号箱中已经存储了Sue的数据。这种存储位置重复了的情况便叫作“冲突”。

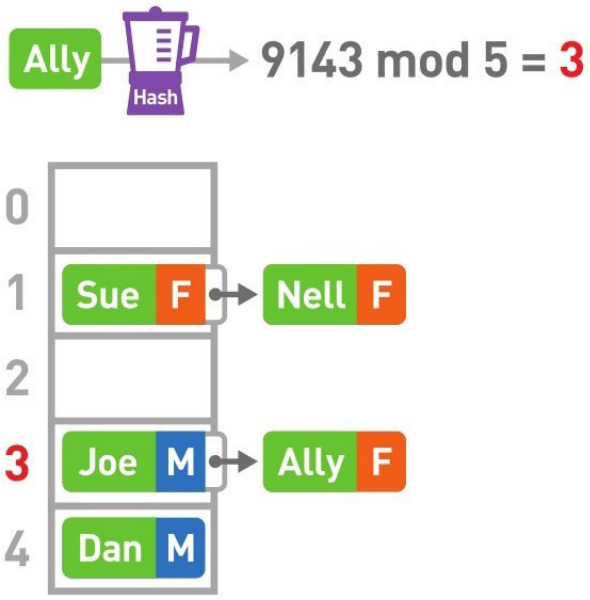
17



遇到这种情况，可使用链表在已有数据的后面继续存储新的数据。关于链表的详细说明请见1-2节。

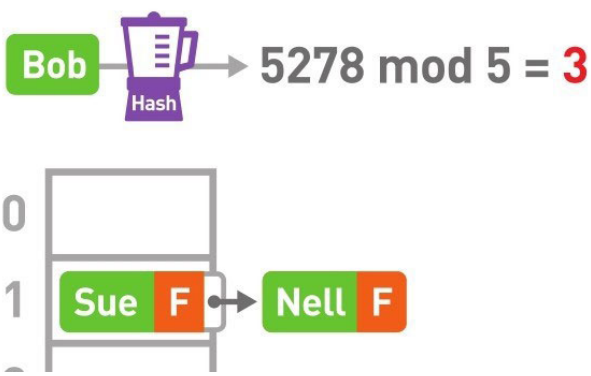
参考：1-2 链表

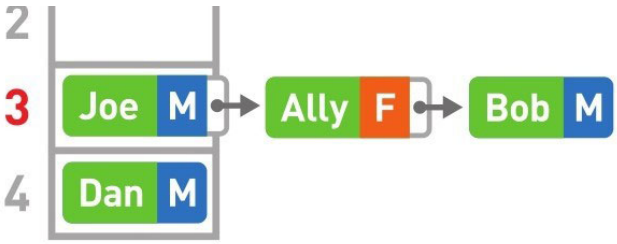
18



Ally键的哈希值为9143, mod 5的结果为3。本应将其存储在数组的3号箱中，但3号箱中已经有了Joe的数据，所以使用链表，在其后面存储Ally的数据。

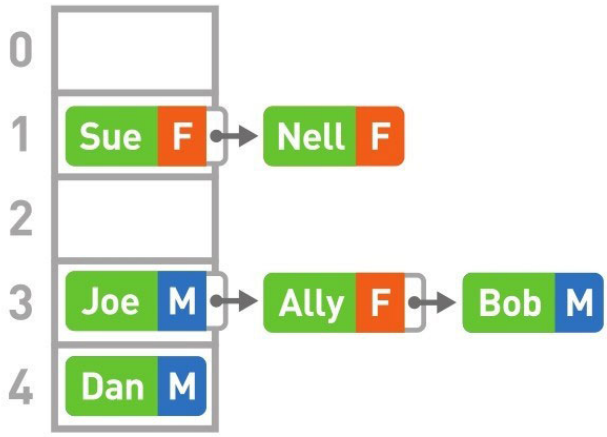
19





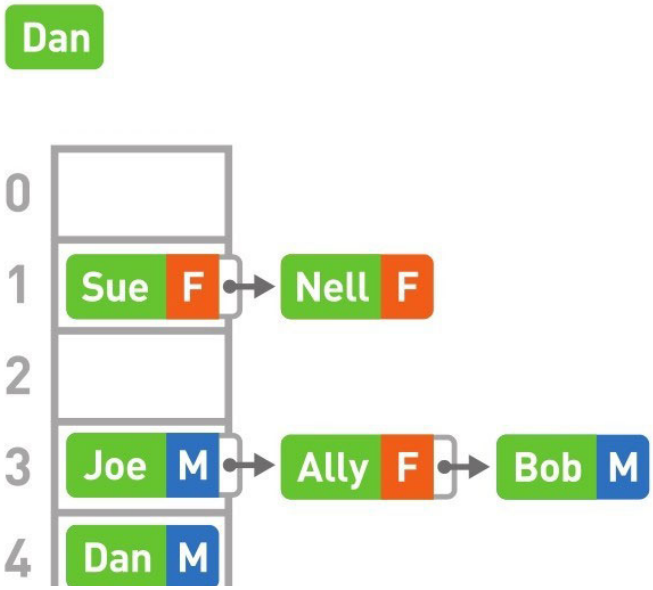
Bob键的哈希值为5278, mod 5的结果为3。本应将其存储在数组的3号箱中，但3号箱中已经有了Joe和Ally的数据，所以使用链表，在Ally的后面继续存储Bob的数据。

20



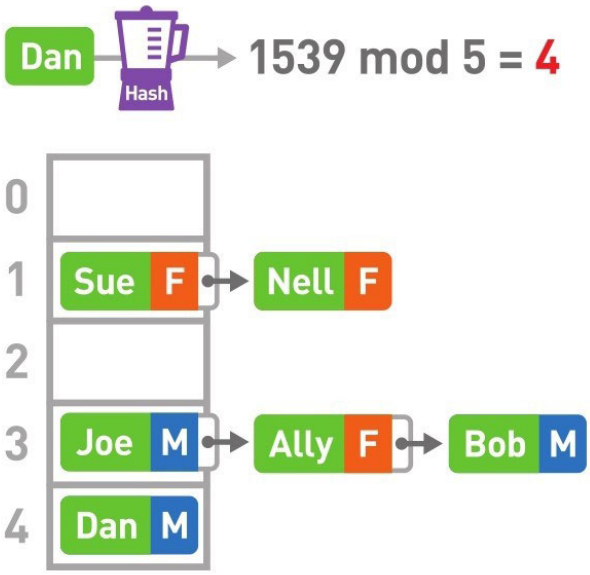
像这样存储完所有数据，哈希表也就制作完成了。

21



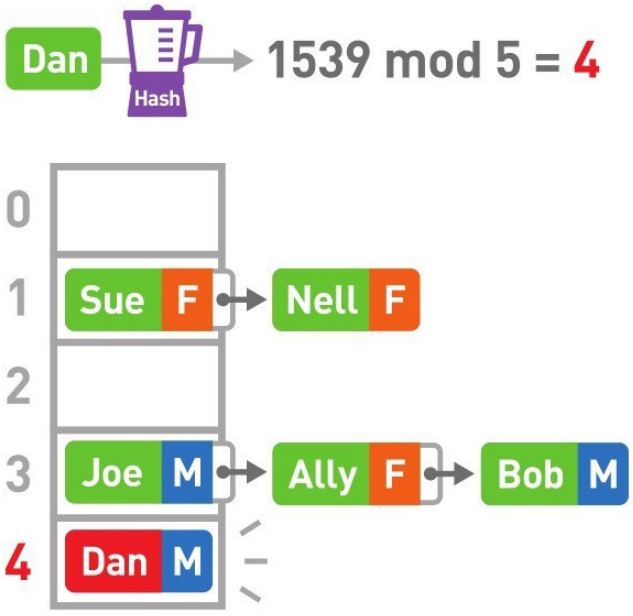
接下来讲解数据的查询方法。假设我们要查询Dan的性别。

22



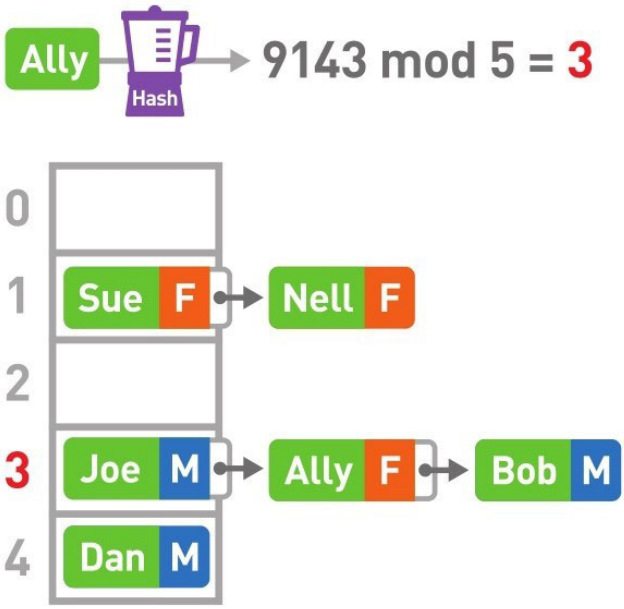
为了知道Dan存储在哪个箱子里，首先需要算出Dan键的哈希值，然后对其进行mod运算。最后得到的结果为4，于是我们知道了它存储在4号箱中。

23



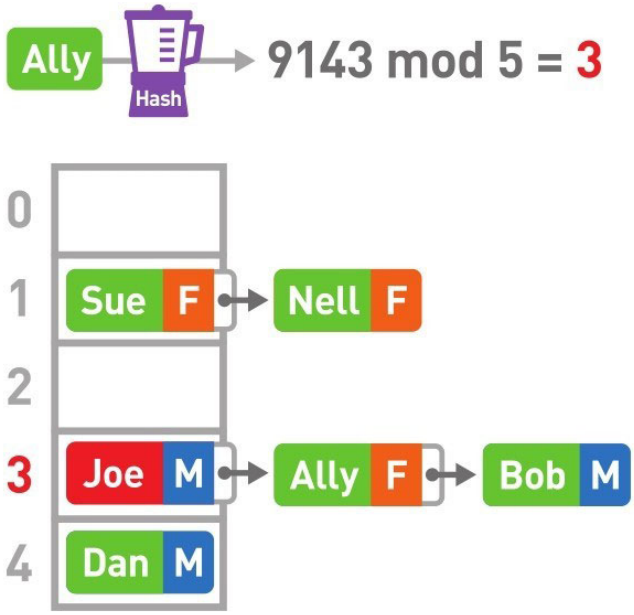
查看4号箱可知，其中的数据的键与Dan一致，于是取出对应的值。由此我们便知道了Dan的性别为男（M）。

24



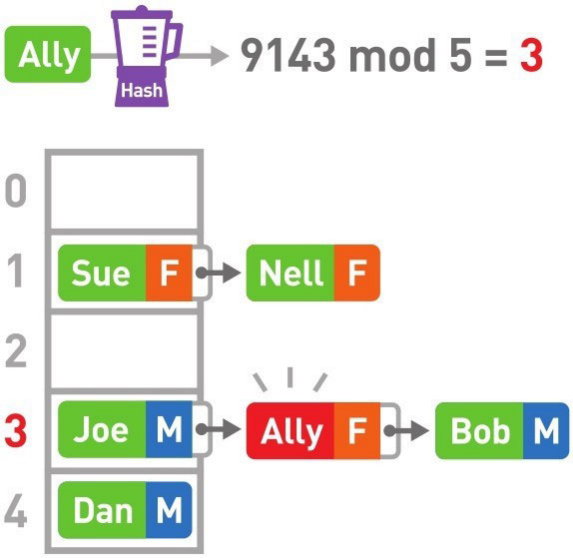
那么，想要查询Ally的性别时该怎么做呢？为了找到它的存储位置，先要算出Ally键的哈希值，再对其进行mod运算。最终得到的结果为3。

25



然而3号箱中数据的键是Joe而不是Ally。此时便需要对Joe所在的链表进行线性查找。

26



于是我们找到了键为Ally的数据。取出其对应的值，便知道了Ally的 性别为女（F）。

解说

在哈希表中，我们可以利用哈希函数快速访问到数组中的目标数据。如果发生哈希冲突，就使用链表进行存储。这样一来，不管数据量为多少，我们都能够灵活应对。

如果数组的空间太小，使用哈希表的时候就容易发生冲突，线性查找的使用频率也会更高；反过来，如果数组的空间太大，就会出现很多空箱子，造成内存的浪费。因此，给数组设定合适的空间非常重要。

补充说明

在存储数据的过程中，如果发生冲突，可以利用链表在已有数据的后面插入新数据来解决冲突。这种方法被称为“链地址法”。

除了链地址法以外，还有几种解决冲突的方法。其中，应用较为广泛的是“开放地址法”。这种方法是指当冲突发生时，立刻计算出一个候补地址（数组上的位置）并将数据存进去。如果仍然有冲突，便继续计算下一个候补地址，直到有空地址为止。可以通过多次使用哈希函数或“线性探测法”等方法计算候补地址。

另外，本书在5-3节关于哈希函数的说明中将会提到“无法根据哈希值推算出原值”这个条件。不过，这只是在把哈希表应用于密码等安全方面时需要留意的条件，并不是使用哈希表时必须要遵守的规则。

因为哈希表在数据存储上的灵活性和数据查询上的高效性，编程语言的关联数组等也常常会使用它。

下一章