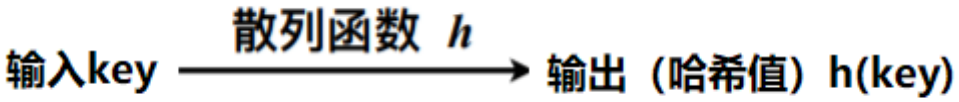
字典

什么是字典

字典是以键值对的形式保存数据的一种数据结构，可以用哈希表实现（比如Python的dict），也可以用其他方式实现（比如C++的map是用红黑树实现的）。只要给出一个关键码，字典就可以返回对应的数据项。这看起来和数组一样，但对于字典来说，其关键码不局限于整数，它可以是任何元素类型，比如字符串，而且关键码之间允许没有大小次序。

用散列表实现字典

什么是散列



散列（也叫杂凑、哈希、Hash）指的是一种能够把任意长度的输入通过散列函数变换成固定长度的输出（散列值）的算法方式。这种转换是一种压缩映射，即散列值的空间远远小于输入空间，不同的输入可能会散列成相同的输出，因此不可能从散列值来确定唯一的输入值。散列也可以认为是一种思想，使用散列算法可以提高存储空间的利用率和数据的查询效率，也可以做数字签名来保障数据传递的安全性。

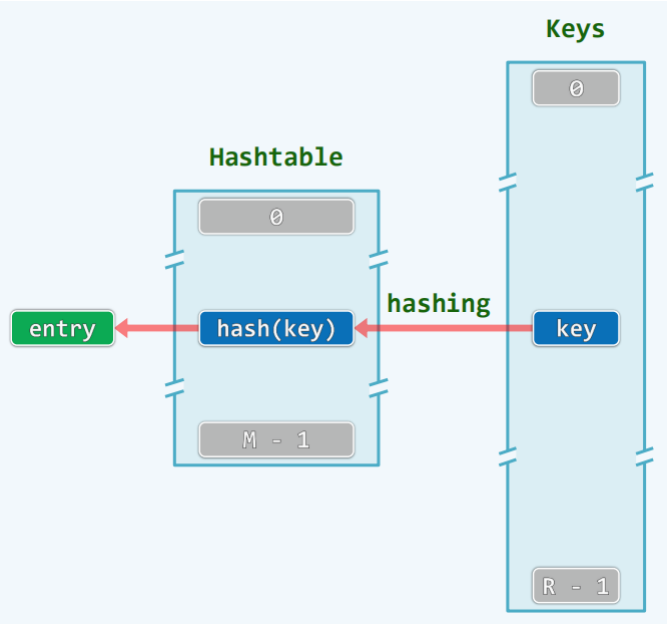
散列函数没有一个固定的公式，只要符合散列特征的算法都可以被称为是散列函数，以下是散列函数的特征：

1. 输出的哈希值数据长度不变；
2. 如果输入的数据相同，那么输出的哈希值也必定相同；
3. 即使输入的数据相似，但哪怕它们只有一比特的差别，那么输出的哈希值也会有很大的差异；
4. 即使输入的两个数据完全不同，输出的哈希值也有可能是相同的。这种情况叫作哈希冲突，而具有相同哈希值的关键字就称作同义词；
5. 不可能从哈希值反向推算出原本的数据。输入和输出不可逆这一点和加密有很大不同；
6. 求哈希值的计算相对容易。

什么是散列表

由散列算法直接实现的数据结构就是散列表。散列表中的每一个存储单元都叫做桶，桶直接存放或间接指向一个词条（关键码+数值），所以散列表也叫做桶数组。每个桶在散列表中的入口地址就是词条关键码通过散列后的哈希值，因此每当存储词条时，都将关键码作为输入通过散列函数计算出对应的桶地址来进行存储；而当访问词条时，也通过同样的散列函数将关键码散列成桶地址，然后根据桶地址从散列表取出对应的词条（可见复杂度只有O(1)）。关键码的集合R通常远远大于散列表长度M（散列值域）的，而能够实现这一现象的原因在于散列是一种压缩映射，它可以将无限的关键码映射到有限的散列值上。

1. 假设待存放词条的总数为N，则称N/M为装填因子，反映了散列表空间利用效率。
2. N、R、M三者的关系：R>>M>N。
3. 散列表的空间复杂度为Θ(N)。



散列表的访问方式

> 假设有一个货架，上面摆满了商品：
>
> 1. 循秩访问：每个商品都按顺序摆放并打上序号，因此顾客可向店员说"我要第3个商品"来获得摆放在第三个位置上的商品。
> 2. 循位置访问：顾客向店员说的话改为"我要你当前指向商品的右边的右边的下边的左边的那一个商品"。
> 3. 循关键码访问：每个商品都用黑盒子装着，如果不取出来根本不知道里面装的是啥，但每个商品都有自己的条形码。当顾客拥有某件商品的条形码时，只需将条形码交给店员，店员就会一个个地取出每个盒子中所存放的商品，然后将该商品的条形码与顾客所给的条形码进行比对，比对相同的就是顾客所需要的商品（为了加快比对速率，使用了二分查找树结构）。
> 4. 循值访问：每个商品都随意摆放并打上序号，此外还有一张表，里面记录了每个商品的名字及其在货架上的序号。因此顾客只要对店员说"我要那个叫某某某的商品"，店员就会根据某某某这个名字在表中找到该商品在货架上的序号，然后就可以按照序号直接从货架上取下该商品给顾客。

可以看出，散列表采用的是循值访问，对应于上面的例子来说，关键码就是商品的名字，而散列函数就是那张表。

散列表的冲突

如前述所言，两个关键码完全不同，输出的散列地址也可能相同。这种情况就叫散列冲突，而具有相同散列值的关键码就称作同义词。

> 散列函数实际上就是从词条空间到地址空间的一种映射，而前者远远大于后者。所以散列属于一种压缩映射，它试图将一个更大的集合映射到一个更小的集合上，因此难以避免会出现散列值相同的关键码。
>
> 所以说散列冲突是不可避免的，其应对方法只能是想法子降低哈希冲突的概率以及当发生哈希冲突时如何进行排解。这两种方法也是设计散列函数的基本策略。

散列函数的设计准则和评价标准：

确定（determinism）：同一关键码总是被映射至同一地址

快速（efficiency）：expected-$O(1)$

满射（surjection）：尽可能充分地覆盖整个散列空间

均匀（uniformity）：关键码映射到散列表各位置的概率尽量接近

可有效避免聚集（clustering）现象

散列函数

除余法

hash(key) = key % M（M为散列表长度）

对于理想随机的序列，M取啥都无关紧要。但在程序中，由于程序具有局部性（for循环等），其访问地址以某个步长递增，此时当M取值为质数时，数据对散列表的覆盖最充分，分布也最均匀。

MAD法

数字分析

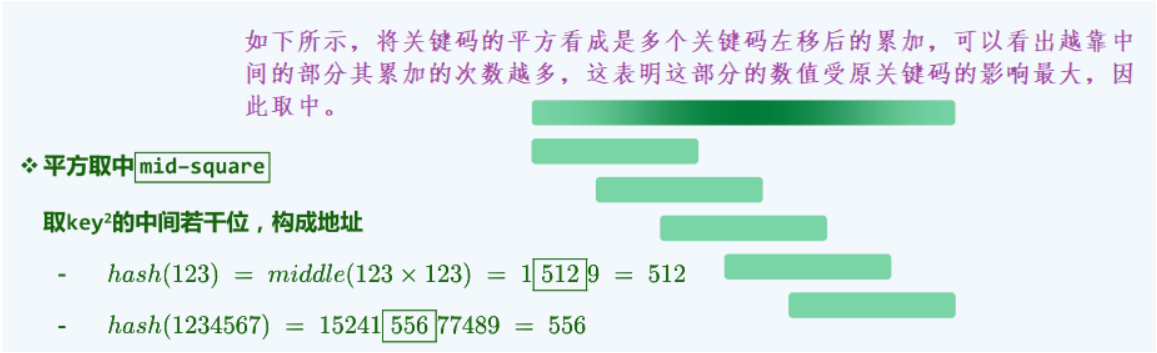
不太均匀。

❖ 数字分析 selecting digits

抽取key中的某几位，构成地址

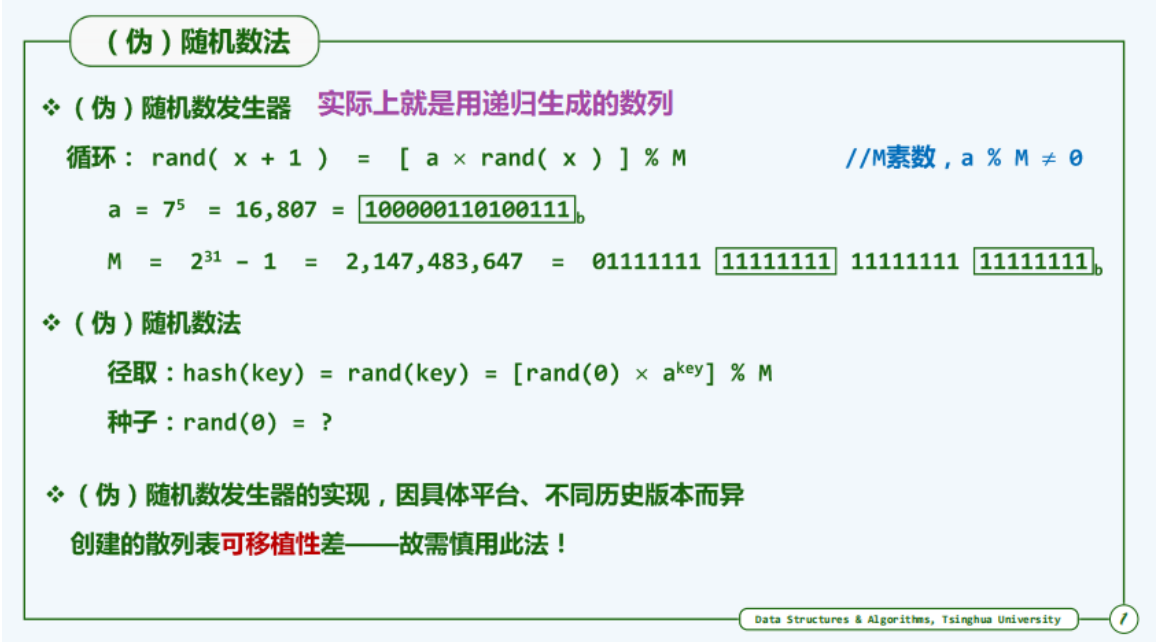
- 比如，取十进制表示的奇数位
- $hash(\begin{smallmatrix}3&4&5&2&5\\1&1&9&6&4\end{smallmatrix}) = 34525$

平方取中



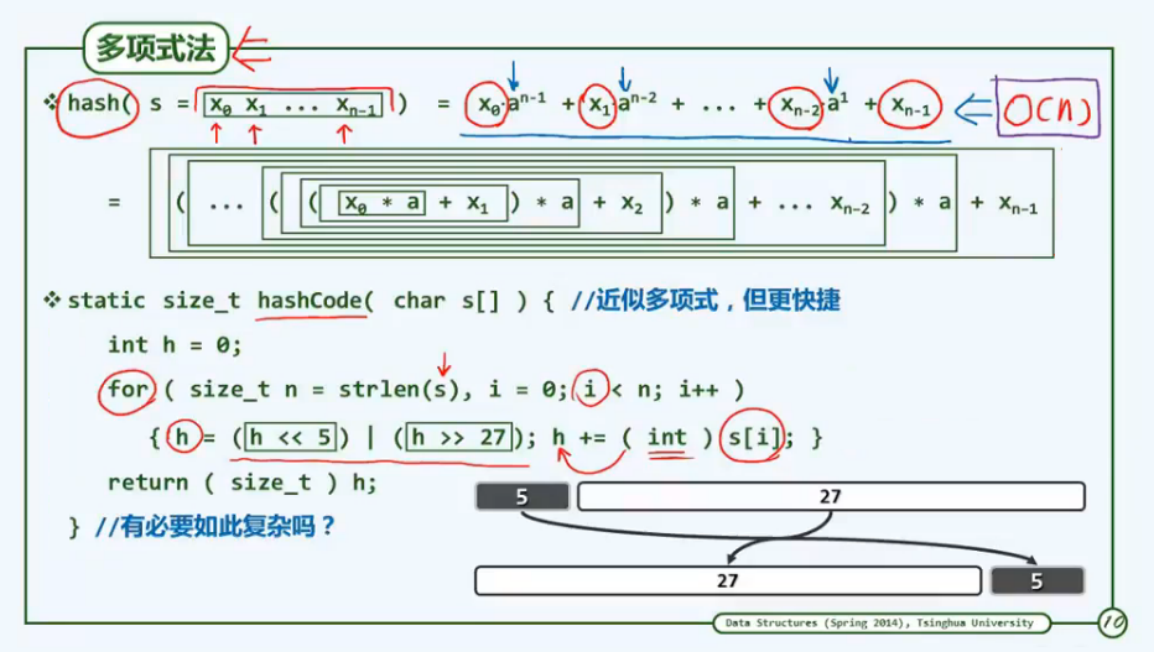
折叠汇总

伪随机数法



多项式法

多应用于关键码为字符串的场合，其过程如下：先将字符串的每个字符转化为对应的数值，然后将这些数值代入多项式进行计算，得出的散列值作为词条地址。



拓展问题：将字符串转化为数值后有必要进行多项式计算吗？不能直接进行相加，将累计总和作为散列地址吗？

	Tom Marvolo Riddle																	
string	T	o	m		M	a	r	v	o	l	o		R	i	d	d	l	e
code	20	15	13		13	1	18	22	15	12	15		18	9	4	4	12	5
hash	196																	

	I am Lord Voldemort																	
string	I		a	m		L	o	r	d		V	o	l	d	e	m	o	r	t
code	9		1	13		12	15	18	4		22	15	12	4	5	13	15	18	20
hash	196																		

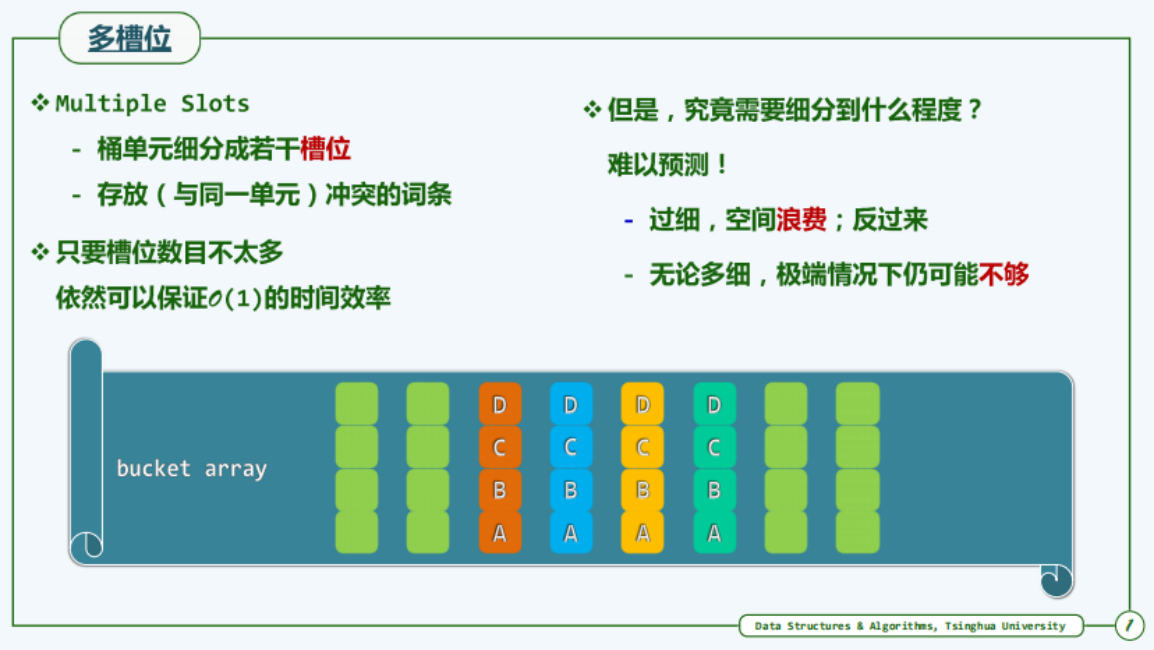
	He's Harry Potter																		
string	H	e	'	s		H	a	r	r	y		P	o	t	t	e	r		
code	8	5		19		8	1	18	18	25		16	15	20	20	5	18		

可以看出，不管是字符相同而次序不同的字符串，还是字符不同的字符串，仅仅通过累计总和的方式计算出来的散列值发生冲突的概率非常大。因此非常有必要进行多项式计算。

排解冲突的方法

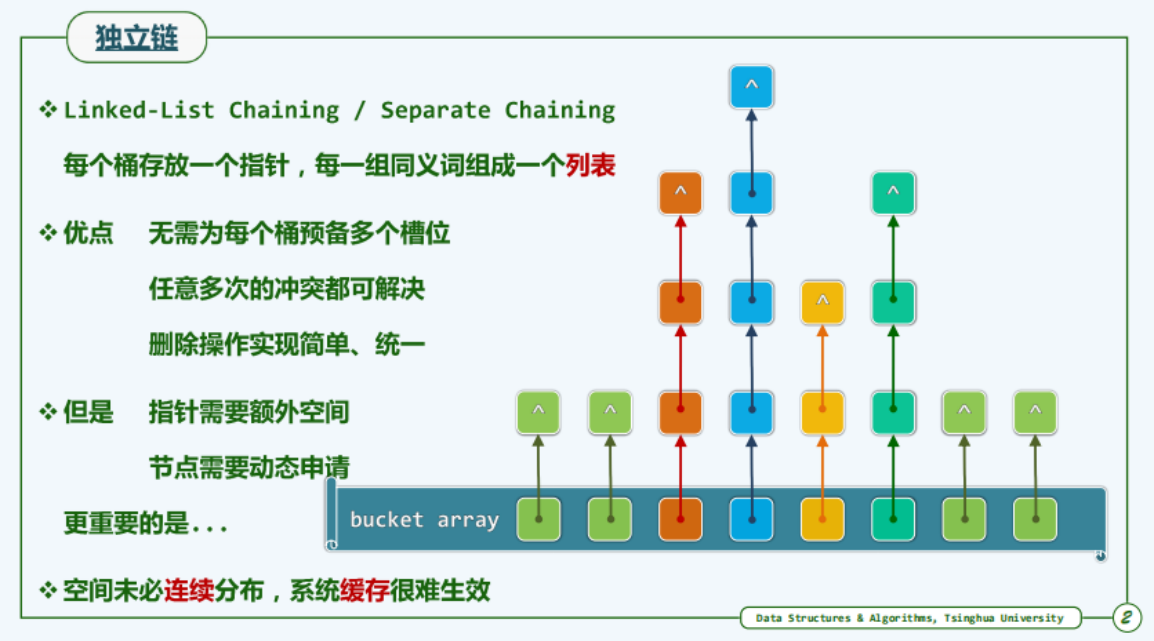
多槽位

每个桶单元存放固定长度的向量。



独立链

每个桶单元存放长度可变的链表。



公共溢出区

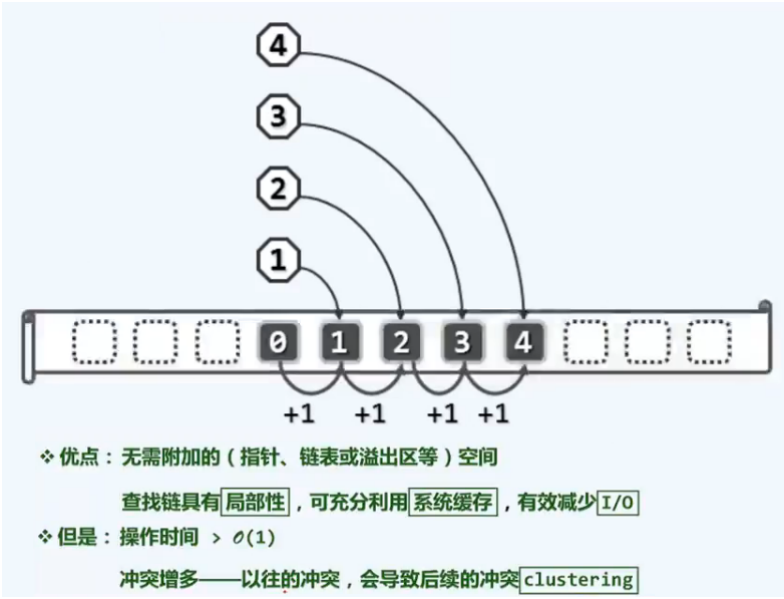
使用一个公共存储空间专门存放冲突的词条。

开放定址

前述方法都是封闭定址，即每个词条都只能存放在各自散列地址对应的桶中，发生冲突的词条也不得占用其它词条的桶子，转而只能另辟空间进行存放。而开放定址指的就是散列表的每个桶子都可以存放任意的词条，每个词条存放在哪个桶子中完全由该词条的散列地址和优先级决定。在查找某个词条时，会依据该词条的查找链（可能存在该词条的一个桶序列）逐个进行查找，直到命中，或抵达一个空桶说明失败。

将决定词条存放在哪个桶子的过程称为试探（也是查找词条的过程），以下是几种试探方法：

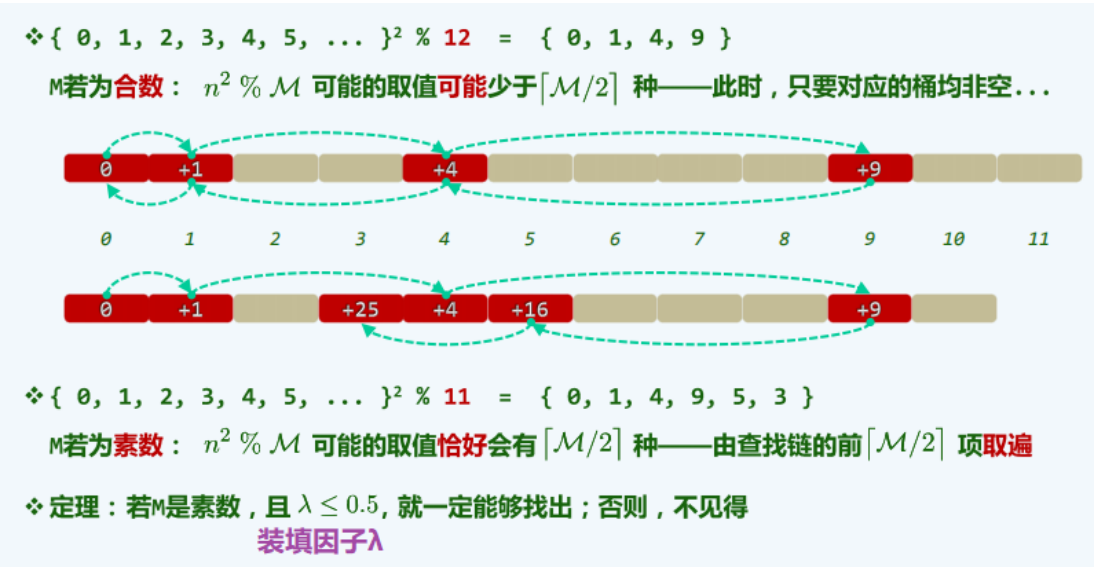
1. 线性试探：假如要存放一个词条，先试探其散列地址对应的桶，如果冲突则试探后一紧邻的桶，以此类推，直到有一个空桶可以存放该词条。



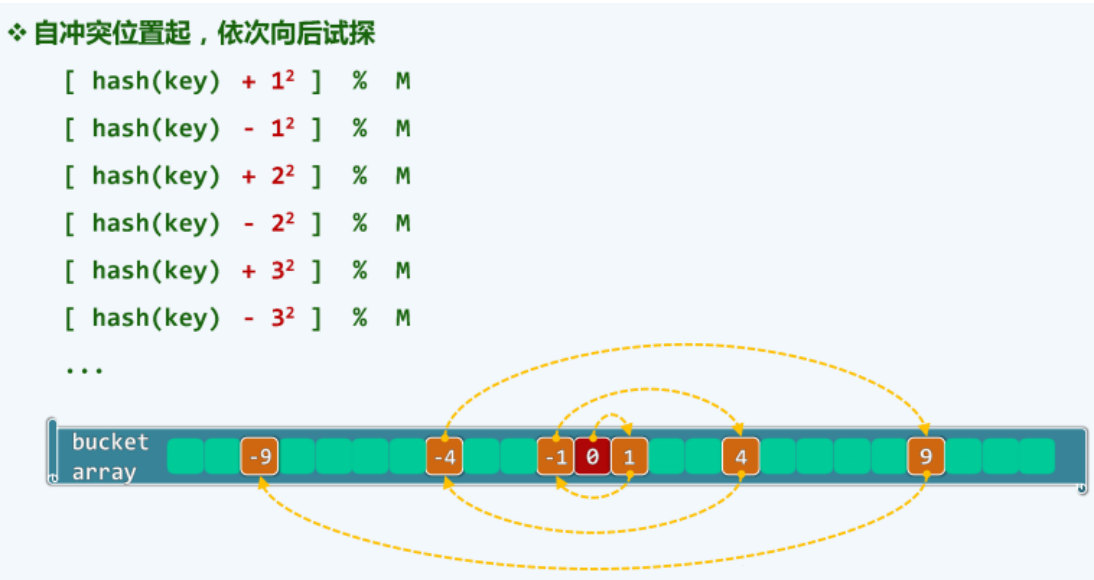
问题及解决方法：如何要删除一个词条，不能直接删除词条，因为它会使后续词条的查找链被切断，使得后续词条处在存在却访问不到的情况。一种解决方法就是懒惰删除，如下图所示：

2. 平方试探：线性试探的缺点在于试探位置间接太近，所以解决方法就是适当增加间距。

问题：在查找链上，无法遍历由所有的空桶，如下图所示：



3. 双向平方试探：平方试探的缺点在于一次试探只能遍历前[M/2]项，为了能够使后[M/2]项，就可以采用双向平方试探。



问题：查找链可能还是只能遍历一半的桶子。

❖ **正向和反向的子查找链，各自包含$\lceil \mathcal{M}/2 \rceil$个互异的桶**

$$\underbrace{-\lfloor \mathcal{M}/2 \rfloor, \ldots, -3, -2, -1}, 0, \underbrace{1, 2, 3, \ldots, \lfloor \mathcal{M}/2 \rfloor}$$

M \ $\pm i^2$	-36	-25	-16	-9	-4	-1	0	1	4	9	16	25	36
5					1	4	0	1	4				
7				5	3	6	0	1	4	2			
11		8	6	2	7	10	0	1	4	9	5	3	
13	3	1	10	4	9	12	0	1	4	9	3	12	10

❖ **两类素数：** 3 5 7 11 13 17 19 23 29 31

❖ **表长取作素数$\mathcal{M} = 4 \cdot k + 3$，即必然可以保证查找链的前$\mathcal{M}$项均互异**

❖ **反之，$\mathcal{M} = 4 \cdot k + 1$就导致正反子查找链存在公共的桶**

散列的具体应用

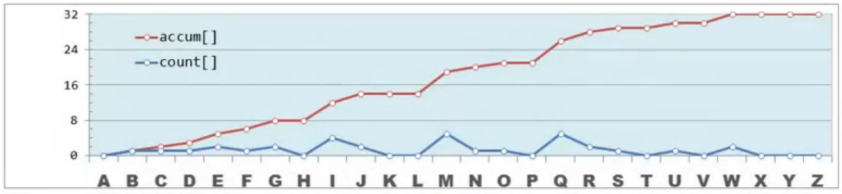
计数排序

计数排序通常用于数据规模n极大而数据取值范围M较小的情况。举一个例子说明其排序过程：

假设要给规模为n的字母序列进行排序，则首先创建一个散列表，里面的每个桶按序存放每个字母对应的词条，每个词条还包括两个内容：该字母在待排序序列中的计数count和在该字母前所有字母计数的总和accum（包括该字母本身的计数），计数排序的任务就是为所有字母词条填好这两项内容：

1. 对于该字母在待排序序列中的计数，可以通过遍历待排序序列来解决，所以只需花费O(n)；
2. 对于字母计数的累计值，可以通过遍历散列表来解决（每次将前者的积分加上自己的计数来得到自己的积分），所以只需花费O(M)。
3. 由此可以看出，总体复杂度为O(n+M)，但由于M<<n，所以复杂度可认为是O(n)。

有了这两项内容后，就可以根据它们进行排序，即每个字母在已排序序列中的区间就由count和accum决定，accum决定了该字母在序列中最末尾的位置，count指出了该字母有多少个。



		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
累计值	accum[]	0	1	2	3	5	6	8	8	12	14	14	14	19	20	21	21	26	28	29	29	30	30	32	32	32	32	O(M)
计数值	count[]	0	1	1	1	2	1	2	0	4	2	0	0	5	1	1	0	5	2	1	0	1	0	2	0	0	0	O(n)
	key/value	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	散列表

CRC	rank	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
2444	vector[]	Q	R	C	B	U	M	I	E	J	I	Q	J	M	Q	I	R	W	W	F	D	Q	M	Q	G	M	S	M	O	I	G	N	E	待排序序列
2444	Sorted[]	B	C	D	E	E	F	G	G	I	I	I	I	J	J	M	M	M	M	M	N	O	Q	Q	Q	Q	Q	R	R	S	U	W	W	已排序序列