散列表（哈希表）

散列表也叫哈希表，是一种高效的存储和搜索方法。

我们可以通过元素的关键字直接通过一个散列函数`addr=Hash(key)`获得元素的存储地址，从而快速地获得这个元素。

散列函数是一种压缩映像函数，关键码的集合比散列表集合大很多，这种对应关系就类似于我有100个元素，分别是100,200,300,400，...，10000。我们可以通过`addr=key/100`这个函数把这100个元素映射到地址为1~100的范围内。这便是哈希表的一种简单情况。

在很多时候，根据哈希函数的不同，两个不同的元素可能映射在同一个散列地址上，这就是**冲突**。举个例子，就拿上面的那个哈希函数`addr=key/100`来说，如果我还有几个元素，120,130，根据计算hash(100)=hash(120)=hash(130)，也就是说三个元素被映射到了同一个散列地址上，起了冲突。冲突是散列函数不可避免的，所以解决冲突是很重要的。

那么我们便可以知道设计散列方法的步骤：

1. 设计一个合适的哈希函数，让元素分散均匀并且冲突比较少

   构造的散列函数需要尽可能简单便捷，定义域是所有关键码，值域是散列表允许的散列地址范围。

2. 设计出现冲突的处理办法

构造散列函数的方法

直接定址法

利用关键码的线性函数

`Hash(key)=a*key+b`

线性函数是1v1映射，不会产生冲突。（上面我举的例子就是a=1/100，b=0的时候的直接定址法哈希函数）

特点是散列表的地址空间需要大于等于集合的大小。

数字分析法

设有n个d位数，每一位可能有r种不同的符号，这r种不同的符号在每一位上出现的频率不一定相同。根据散列表的大小，选择其中符号均匀的若干位作为散列地址。

计算符号均匀度的公式 $$\lambda_k = \sum_{i=1}^{r} (\alpha_i^k - n/r)^2$$

其中 α_i^k 表示的是第i个符号在第k位上出现的次数，n/r表示各种符号在n个数中间均匀出现的期望值。

比如一组数

942148 941269 940527 941630 941805 941558 942047 940001

我们对这一组数字进行分析，一共有10个数，每个数出现概率相等，一共出现8次，那么每个数出现的期望是8/10，n/r=8/10

对于第一位 全部八个数都是9，λ1=(8-8/10)^2+(0-8/10)^2=56.60

对于第二位 全部八个数都是4，λ2=(8-8/10)^2+(0-8/10)^2=56.60

对于第三位，2出现2次，1出现4次，0出现2次，所以λ3=(4-8/10)^2+2 * (2-8/10)^2+7 * (0-8/10)^2=16.60

对于第四五六位，都是有两个数出现两次，有三个数出现一次

λ4=λ5=λ6=2 * (2-8/10)^2+3 * (1-8/10)^2+5*(0-8/10)^2=5.60

所以，如果散列地址有3位的话，应当取关键码的4,5,6位作为散列地址。

除留余数法

如果散列表允许的地址数目为m，取**一个不大于m或者等于m的质数p**作为除数利用关键码除以p的余数作为散列地址。（同时要求p不接近2的幂次）

`Hash(key)=key%p，p<m`

需要注意的一点是，如果散列表的大小是m大于哈希函数里面的p，那么在计算散列地址的时候大于等于p的部分是无法到达的，只有在处理冲突的时候可以到达。那么**在计算ASL不成功的时候，我们应当取散列地址为0-p的p个地址，而p-m这几个是不可以的。**

平方取中法

这是一种常用的哈希函数构造方法。这个方法是先取标识符内码的平方，然后根据可使用空间的大小，选取平方数是中间几位为哈希地址。

哈希函数 **H(key)=“key2二进制表示的中间几位”**因为这种方法的原理是通过取平方扩大差别，平方值的中间几位和这个数的
每一位都相关，则对不同的关键字得到的哈希函数值不易产生冲突，由此产生的哈希地址也较为均匀。

折叠法

这个方法把关键码从左到右分成位数相等的几部分，每一部分的位数和散列地址的位数相同，最后一部分如果不够可以短一点。把这些部分数据叠加起来就可以得到具有关键码的记录的散列地址。

两种叠加方法：

- 移位法：八个部分的**最后一位**对齐相加
- 分界法：各部分不折断，按照分界来回折叠然后对其相加，就像叠围巾一样。

  举个例子，比如对于key=23938587841，假设散列地址是3位，那么把key分成239 385 878 41

  利用移位法239+385+878+041=(1)543，那么散列地址是543

  利用分界法，折毛巾一样的239+583+878+14=(1)714

处理冲突的方法

闭散列方法

如果散列表元素经常增删变化的话，尽量不要用闭散列方法

线性探查法

Linear Probing

先利用散列函数计算散列地址，如果发生冲突就向后找……

通俗一点来说，就是给所有相同特征（有相同的散列地址）的人每人一张同一个座位号的车票，如果你上车之后发现那个座位已经被占了，那么就往后找空的座，有空就占，这样找你的时候呢，从你车票上的座位号开始往后找，也不会离太远。

线性探查法根据装载因子的平均搜索长度计算公式是

$$ASL_{succ} = (1 + 1/1 - \alpha)/2$$

$$ASL_{unsucc} = (1 + (1/1 - \alpha)^2)/2$$

二次探查法

Quadratic Probing

Hi=(H0+i^2)%m，m是散列表大小，散列表大小必须满足4k+3的质数

当表的长度为质数并且表的装载因子α不超过0.5的时候，新的表项一定能够插入，并且任何一个位置不会被探查两次，只要表中至少有一半空的就不会有表满的问题，搜索时不考虑表满的情况，但是在插入时候装载因子不超过0.5，超出的话应当把表长度扩大一倍，进行表的分裂。

注意：二次探查法对散列表（m）的大小有严格要求，必须是一个大小为 `4k+3` 的素数，并且对于装载因子也有严格要求，α必须小于0.5

双散列法

Hi=(H0+i * rehash(key))%m

开散列方法

链地址法

先用散列函数计算散列地址，那么把地址相同的“同义词”（桶）利用链表串起来，叫做同义词表

这种方法就是让具有相同特征的人坐一个车厢，找你的时候只需要你在哪个车厢就好了。

开散列法可以解决随着装载因子变大而ASL变长的问题，如果α比较大可以考虑开散列法，并且开散列法的装载因子可以超过1

解决冲突时候ASL

根据装载因子α

	ASL_succ	ASL_unsucc
线性探查法	(1+1/(1-α))/2	(1+1/(1-α)^2)/2
伪随机探查，二次探查，双散列法	-(1/α)loge(1-α)	1/1-α
开散列法	1+α/2	α

1.当装载因子接近1的时候，为什么线性探查法接近于顺序搜索?

因为线性探查法装载因子接近1的时候，出现大量的堆积表项，每次搜索表项的时候都要顺序搜索很多表项，如果搜索失败的话就是搜索了整个散列表。

2.为什么装在因子比较低的时候，线性探查法复杂度是常数级?

因为根据公式，搜索成功的ASL和搜索不成功的ASL都是比较小的常数（2和6）

3.装载因子没有强制性要求。

4.计算出地址分布最均匀的是除留余数法

5.散列表的平均搜索长度和表的长度无关，但是和表的装载因子还有解决冲突的办法有关

6.闭散列的**开地址法**，当冲突发生的时候,通过查找数组的一个空位,并将数值填充进去,而不再是使用哈希函数得到的数组下标,就叫做开放地址法

7.搜索已有表项ASL就是搜索成功ASL，插入表中没有表项的ASL就是搜索不成功ASL

8.二次探查法应该先搜索 + 再搜索 -

9.保证在删除的时候不中断搜索链，可以对被删记录进行逻辑删除，标记每个表项有三个状态，正在使用，删除，空闲