6.1 哈希

一：引出

对于哈希，学到现在，大家至少对这个词不陌生了，在前边的课程，我给大家讲过，求一个字符串中每个字母的个数。还记得我讲过的比较简单的实现方式么？那个就是一个简单的哈希实现，那我们就来详细的介绍一下哈希。

1. 哈希查找具体含义讲解

1.1 简介

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的一种数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

记录的存储位置=f(关键字)

这里的对应关系 f 称为散列函数，又称为哈希（Hash 函数），采用散列技术将记录存储在一块连续的存储空间中，这块连续存储空间称为散列表或哈希表（Hash table）。

那我们之前讲的统计字母个数的hash函数是什么？：f(key)=key-'A';

1.2 原理

哈希表 hashtable(key，value) 就是把 Key 通过一个固定的算法函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将 value 存储在以该数字为下标的数组空间里。（或者：把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，而不可能从散列值来唯一的确定输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。） 而当使用哈希表进行查询的时候，就是再次使用

哈希函数将key 转换为对应的数组下标，并定位到该空间获取 value，如此一来，就可以充分利用到数组的定位性能进行数据定位。

整个散列过程其实就是两步：

(1)存储时，通过散列函数，根据记录，获取到对应的存储位置，将记录存储下来

(2)存储时，通过散列函数，根据记录，获取到对应的存储位置，获取记录

所以，散列，是一种存储方法们也是一种查找方法，与之前的数据结构不同的是，之前的数据结构元素之间都是存在逻辑关系的，而散列元素与元素之间不存在关系，散列的存储和查找只与关键字有关系，因此，散列主要面向查找。

二：具体实例讲解

1. 哈希函数的构造

比如数列{11,54,33,60,72}的存储，我们之前再数组中是怎么存的？

{11,54,33,60,72}顺序存储

11	54	33	60	72

基于这个存储，我们要查找60，怎么查？for (i=0;i<len;i++) 循环到第4次找到，那假如这个数组非常大呢？长度为1024，为1024*1024， 1024*1024*1024呢，那我们要找的数据如果刚好在后边是不是效率很低。

有没有效率高的方法呢？当然有，那就要用我们的哈希查找了。

现在我们在散列中存储，我们的哈希函数选择为f(key) = key%len;key为要存储的数据，len为哈希表的长度，f(key)求得的是key在哈希表中存储的数组下标。数列{11,54,33,60,72}在哈希表中的存储如下：

{11,54,33,60,72}数列存储，f(key) = key%len;

	60	11	72	33	54
f(key)	60%5=0	11%5=1	12%5=2	33%5=3	54%5=4

那基于哈希表的查找呢？要查找60，f(60) = 60%5 = 0，则通过hash[0](hash为hash表的名字，也就是存储这些数据的数组名)直接就取到了。取哈希表中的任何一个元素都是O(1)的。

思考：字母个数统计，用这中哈希函数构造方法是不是也可以？

2. 哈希冲突

刚才大家也看到了，我的这个数列比较特殊，通过f(key) = key%len映射到数组中，刚好存在了数组的不同位置，但我们遇到的数据，如果是{11,54,10,60,72}呢，f(10)=10%5=0；f(60)=60%5=0,那60和10同时对应了数组的一个位置，这就叫哈希冲突。

3. 解决hash冲突

方法 1：开发地址法

所谓开放地址方解释一旦发生了冲突，就去寻址下一个空的散列地址。只要 散列表足够大，空的散列地址总是能够找到，并且将其记录在内。 重新构造哈希函数公式 f(key) = (f(key) + d) % p (d = 1,2,3,4,...m - 1);

集合{12,67,56,16,25,37,22,29,15,47,48,34}表长为12，我们使用哈希函数: f(key) = key%12

存放前5个数据的时候，都是没有冲突的，直接存入

下标	0	1	2	3	4	5	6	7	8	9	10	11
关键字	12	25			16			67	56			

接下来是37，f(37)=37%12=1，数组下标为1的已经有数据25了，发生了冲突，用上边解决冲突的公式，f(37)=(f(37)+1)%12=2，数组下标为2的没有，直接存入。

下标	0	1	2	3	4	5	6	7	8	9	10	11
关键字	12	25	37		16			67	56			

接下来22,29,15,47都没有冲突，直接存入。

下标	0	1	2	3	4	5	6	7	8	9	10	11
关键字	12	25	37	15	16	29		67	56		22	47

下面是48，f(48)=0，与12位置冲突了，不要紧，f(48)=(f(48)+1)%12=1，又与25所在的位置冲突了，继续f(48)=(f(48)+2)%12=2，还是冲突，继续……一直到f(48)=(f(48)+6)%12=6，有空位了，存入。

下标	0	1	2	3	4	5	6	7	8	9	10	11
关键字	12	25	37	15	16	29	48	67	56		22	47

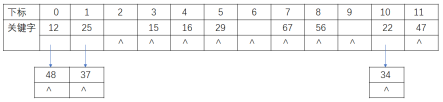
思考：往后找，找到尾了会怎样？会不会存在找不到的情况？

那大家思考一下，我用这种的方式解决冲突了，那我们查找的时候怎么做呢？按照一样的方式查找就可以了，找了一圈还没找到则就返回没找到。

这种解决方法，大家想一下，加入就按照上边的第二个图，37的存发，37应该在数组下标为1的位置，而通过我们解决冲突，占用了数组下标2的位置，后边如果来了真正的数组下标2位置的数据，我们把这种多个数据抢占一个位置的现象叫堆积。很显然，堆积的出现，导致我们不管存储还是查找，为了解决冲突，都会有额外的时间开销，会对我们的效率产生影响。

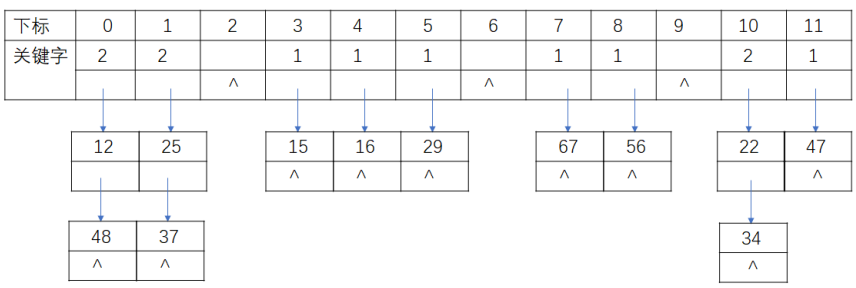
方法 2：链地址法

我们就想按照自己应该在的位置存储，不想像上边那样取解决冲突，可不可以？可以，用我们的链地址的方式就可以了。链地址的思路很简单，是将数组和链表结合起来用，就是数组下边挂链表，怎么用？我们看看上边的例子，怎么来存储。



上图，按照哈希函数找到对应的位置，这个位置还没有数据，则存储数据，有则在这个数据后边直接挂接。再有就接着往后挂接。我们上图是数组中也是保存了数据了，当然，我们也可以数组中并不存真正的数据，就保存指针，或者可以维护一下下边挂的节点的个数。

{12,67,56,16,25,37,22,29,15,47,48,34}



方法 3：公共溢出区

哈希冲突的解决办法，我们介绍了开放地址和链地址方法，这里再简单拓展介绍一个，“公共溢出区法”。这个字面就很好理解了，就是除了基本的哈希表，再建立一张公共溢出表，存放冲突数据。比如上边的，使用公共溢出的方式解决冲突：

基本表

下标	0	1	2	3	4	5	6	7	8	9	10	11
关键字	12	25		15	16	29		67	56		22	47

溢出表

下标	0	1	2	3	4	5	6	7	8	9	10	11
关键字	37	48	34									

在冲突数据比较少的时候，公共溢出区的方式还是很高效的，比我们开放地址法要高效。

三：哈希其他

刚刚用到的f(key)=key%len是哈希构造函数的一种，我们还有很多其他的构造函数，不管什么样的构造函数，遵循两个原则：1.函数值应在1至记录总数之间 2.尽可能的避免冲突。

简单介绍几个哈希函数的构造方法：

1.直接定址法

公式：f(key)=a*key+b;

我们之前计算字母个数的函数f(key)=key-'A';就是直接址法。

2.除留余数法

取关键字被某个不大于哈希表表长m的数p除后所得余数为哈希地址，即设定哈希函数为 Hash(key)=key mod p (p≤m)，其中，除数p称作模。

3.字符串哈希函数构造法

字符串的哈希也是经常会用到的，字符串的哈希就是先有算法把字符串转换成整数，然后再通过整数哈希。最简单的就是将字符串中所有字符的ascii码求和转换成整数。字符串的哈希，有很多已经很成熟很优秀的算法，我们就简单介绍两种种。

下列函数是取字符串前10个字符来设计的哈希函数

```
Int Hash _ char (char *X)
 {
  int I ,sum
  i=0;
  while (i 10 && X[i])
  Sum +=X[i++];
  sum%=N;     //N是记录的条数
  }
```

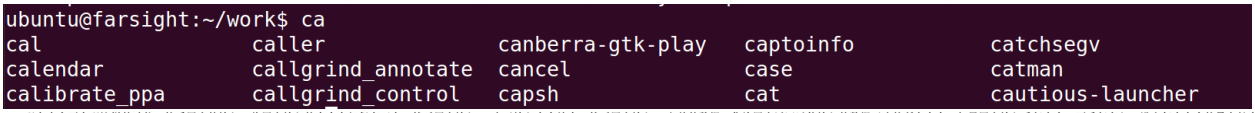
这种函数把字符串的前10个字符的ASCII值之和对N取摸作为Hash地址，只要N较小，Hash地址将较均匀分布[0，N]区间内，因此这个函数还是可用的。对于N很大的情形，可使用下列函数

```
int ELFhash (char *key )
{
    Unsigned long h=0,g;
    whie (*key)
    {
        h=(h<<4)+ *key;
        key++;
        g=h & 0 xF0000000L;
        if (g) h^=g>>24;
        h & =~g;
    }
    h=h % N
    return （h）;
}
```

ELFhash算法的核心在于"影响"，字符中每个字符，力求都对结果产生影响，从而减少冲突。

参考代码：hash_open.c hash_link.c

.*练习 我们学了linux基础知识，学了shell命令，都应该知道，shell命令中，我们有一个功能叫命令补全，就是你输入命令的一部分，然后两下tab，系统会把所有以你输入的这部分的内容开头的命令都列出来。比如下图，输入ca，然后两下tab之后的结果。



现在大家要实现这样的功能，接受用户的输入，将用户输入的命令全部保存下来，然后用户输入cnd表示输入命令结束，然后用户输入tab之前的数据，你给用户返回以他输入的数据开头的所有命令。如果用户输入重复命令，不重复存入。设定每个命令的最大长度不超过15个字符，并且都是小写字母组成。

例如：

用户输入：cal cat case tree man make mag end　　ca 回车

程序输出: cal cat case

参考代码：hash_command.c

百度面试题：

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的10个查询串，要求使用的内存不能超过1G。

1G = 1024M = 1024*1024K = 1024*1024*1024字节