第九章 查找

9.1 静态查找表

9.2 动态查找表

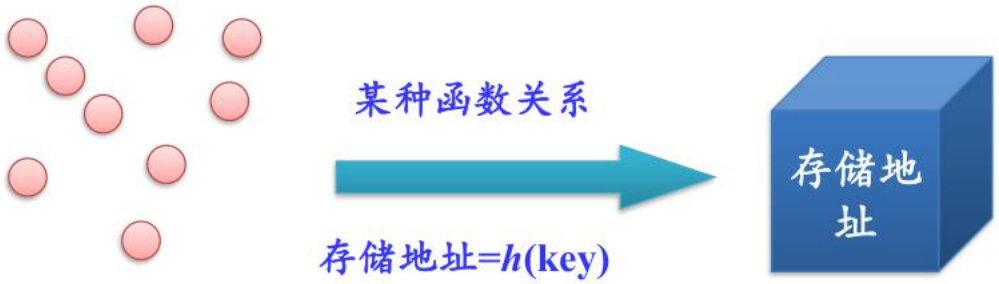
9.3 哈希表

9.3 哈希表

一. 概念

■ 哈希查找

- 哈希存储结构（散列存储）：在记录的存储地址和它的关键字之间建立一个确定的对应关系



- 目的：理想情况下，查找时可以不经过关键字比较，一次存取就能得到所查记录。

9.3 哈希表

一. 概念

- 哈希查找

  - **哈希函数**：记录的关键字与记录的存储地址之间建立的一种对应关系，是从关键字空间到存储地址空间的一种映象。

    可写成：

$$addr(a_i) = H(key_i)$$

其中，$H(\cdot)$为哈希函数，key_i是记录a_i的关键字，$addr(a_i)$是存储地址，即**哈希地址**（或散列地址）

9.3 哈希表

一. 概念

- 哈希冲突
  - 不同的记录，其关键字通过哈希函数的计算，可能得到相同的地址，把不同的记录映射到同一个散列地址上，这种现象称为**冲突**

    即对于不同的关键字k_i、k_j，若$k_i \neq k_j$，但$H(k_i)=H(k_j)$
  - 具有相同函数值的关键字对该哈希函数来说称做**同义词**

9.3 哈希表

一. 概念

- 哈希冲突举例

我国省名称列表，以名称拼音的首字母为关键字，哈希函数是字母ASCII码之和，取八进制值作为十进制值，模30后的值为地址

- 河南，拼音HE NAN，关键字HN，ASCII码和为八进制的226，不转换，作为十进制的226，模30的值为16，即地址为16
- 四川，拼音SI CHUAN，关键字SC，ASCII码和为八进制的226，不转换，作为十进制的226，模30的值为16，即地址为16

- 关键字HN和SC不同，但计算出来的地址相同，因此冲突

9.3 哈希表

一. 概念

- 哈希表

根据设定的哈希函数 H(key) 和所选中的处理冲突的方法，将一组关键字映像到一个有限的的连续的地址集（区间）上，并以关键字在地址集中的“像”作为相应记录在表中的存储位置，如此构造所得的查找表称之为哈希表，这一映像过程称为哈希造表或散列，所得存储位置称为哈希地址或散列地址。

9.3 哈希表

二. 哈希函数

- 哈希函数的设计要求
  - 哈希函数实现的一般是从一个大的集合（部分元素，空间位置上一般不连续）到一个小的集合（空间连续）的映射，一般情况下，哈希函数是一个**压缩映像**。
  - 哈希函数应是**简单**的，能在较短的时间内计算出结果。
  - 哈希函数的定义域必须包括需要存储的**全部关键字**。
  - **均匀性**：哈希函数计算出来的地址应能均匀分布在整个地址空间中，即关键字经过哈希函数得到一个“随机的地址”。

9.3 哈希表

二. 哈希函数

- 哈希函数的设计方法：
  - 直接定址法
  - 数字分析法
  - 平方取中法
  - 折叠法
  - 除留余数法

9.3 哈希表

二. 哈希函数

1、直接定址法

- 哈希函数取关键字的线性函数

H(key) = a * key + b，其中a和b为常数

例如，H(key) = key – 2005131000，用于下表

003	2005131003	朱嘉成	男	信息工程学院
005	2005131005	陈乾	男	信息工程学院
006	2005131006	桂许升	男	信息工程学院
007	2005131007	罗杨洋	男	信息工程学院
008	2005131008	叶建行	男	信息工程学院
009	2005131009	曹亚仑	男	信息工程学院
012	2005131012	欧东	男	信息工程学院

9.3 哈希表

二. 哈希函数

- 直接定址法的特性
  - 直接定址法仅适合于地址集合的大小与关键字集合的大小相等的情况
  - 对于不同的关键字不会发生冲突
  - 当a=1，b=0时，H(key)=key，即用关键字作地址，在实际应用中能使用这种哈希函数的情况很少

9.3 哈希表

二. 哈希函数

- 2、数字分析法
  - 假设关键字集合中的每个关键字都是由 s 位数字组成 $(u_1, u_2, \ldots, u_s)$
  - 分析关键字集，从中提取分布均匀的若干位或它们的组合作为地址。

9.3 哈希表

二. 哈希函数

- 数字分析法举例

有80个记录，关键字为8位十进制数，要求哈希地址为2位十进制数。

①	②	③	④	⑤	⑥	⑦	⑧
8	1	3	4	6	5	3	2
8	1	3	7	2	2	4	2
8	1	3	8	7	4	2	2
8	1	3	0	1	3	6	7
8	1	3	2	2	8	1	7
8	1	3	3	8	9	6	7
8	1	3	6	8	5	3	7
8	1	4	1	9	3	5	5

分析： 第①、②位只取8、1， 第③位只取3、4， 第⑧位只取2、7、5，不可取。第 ④⑤⑥⑦位数字分布近乎随机

所以：取④⑤⑥⑦任意两位或两位与另两位的叠加作哈希地址

9.3 哈希表

二. 哈希函数

- 数字分析法的特性
  - 数字分析法仅适用于事先明确知道表中所有关键码每一位数值的分布情况
  - 数字分析法完全依赖于关键码集合，如果换一个关键码集合，选择哪几位要重新决定。

9.3 哈希表

二. 哈希函数

- 3、平方取中法
  - 以关键字的平方值的中间几位作为存储地址。
  - 平方值的中间各位与关键字的各位均有关系
  - 取的位数与哈希表的表长有关

9.3 哈希表

二. 哈希函数

- 平方取中法举例

标识符的八进制内码表示及其平方值，哈希表表长2^9

A	*B*	*C*	…	*Z*	*0*	*1*	…	*9*
01	02	03	…	32	60	61	…	71

标识符	内码	内码的平方	散列地址
A	0100	0<u>010</u>000	010
P2	2062	4<u>314</u>704	314
Q1	2161	4<u>734</u>741	734

9.3 哈希表

二. 哈希函数

- 平方取中法的特性
  - 平方取中法是较常用的构造哈希函数的方法
  - 适合于关键字中的每一位都有某些数字重复出现且频度很高的情况
  - 中间所取的位数，由哈希表长决定

9.3 哈希表

二. 哈希函数

4、折叠法

- 将关键字分割成位数相同的若干部分(最后部分的位数可以不同)，然后取它们的叠加和(舍去进位)为哈希地址

① 移位叠加：将分割后的几部分低位对齐相加

② 间界叠加：从一端向另一端沿分割界来回折叠，然后对齐相加

9.3 哈希表

二. 哈希函数

- 折叠法举例
  - 例如图书都有国际标准图书编号ISBN号，当图书种类不超过10000种时，可以采用折叠法构造一个四位数的哈希函数。

关键字为：0442205864，哈希地址位数为4

移位叠加	间界叠加
5864	5864
4220	0224
04	04
10088	6092
H(key)=0088	H(key)=6092

9.3 哈希表

二. 哈希函数

- 折叠法特性
  - 折叠法适合于关键字的数字位数特别多，而且每一位上数字分布大致均匀的情况

9.3 哈希表

二. 哈希函数

5、除留余数法

- 取关键字除以某个不大于哈希表表长 m 的数 p 后所得余数为哈希地址

$$H(key) = key \ MOD \ p \quad (p \leqslant m)$$

其中，m为哈希表表长

p为不大于m的素数或是不含20以下的质因子的合数

9.3 哈希表

二. 哈希函数

- 除留余数法举例
  - 给定一组关键字为： 12，39，18，24，33，21

H(key) = key MOD p （ p≤m ）

若取 p=9，则它们对应的哈希函数值将为：

3， 3， 0， 6， 6， 3

可见，若p中含质因子3，则所有含质因子3的关键字均映射到“3的倍数”的地址上，从而增加了“冲突”的可能。

9.3 哈希表

二. 哈希函数

- 除留余数法特性
  - 除留余数法是一种最简单、最常用的构造哈希函数的方法
  - 不但可以对关键字直接取模(MOD)，也可在折叠、平方取中等运算之后取模

9.3 哈希表

二. 哈希函数

- 设计哈希函数应考虑的因素有：
  - 计算哈希函数所需的时间
  - 关键字的长度
  - 哈希表的大小
  - 关键字的分布情况
  - 记录的查找频率

9.3 哈希表

三. 处理冲突的方法

- “处理冲突” 的实际含义是：为产生冲突的地址寻找下一个“空”的哈希地址。
- 方法主要有：

1. 开放定址法
2. 再哈希法
3. 链地址法（拉链法）
4. 建立公共溢出区

9.3 哈希表

三. 处理冲突的方法

1、开放定址法

为产生冲突的地址 H(key) 求得一个地址序列： H_0, H_1, H_2, …, H_s, $1 \leqslant s \leqslant m-1$，沿此地址序列逐个探测，直到找出一个空闲空间，将冲突记录存于此处。只要哈希表不满，总是可行的。

$$H_i = [\ H(key) + d_i\ ]\ MOD\ m \qquad i=1,2,\ldots,s$$

H(key)为哈希函数，m为哈希表表长， d_i为增量序列

a) 当d_i取1, 2, 3, …, m-1时，称为线性探测再散列

b) 当d_i取= $+1^2, -1^2, +2^2, -2^2, +3^2, -3^2$, …, $\pm k^2$（$k \leqslant m/2$）时，称为二次探测再散列

c) 当d_i取伪随机数序列，称为伪随机探测再散列

9.3 哈希表

三. 处理冲突的方法

- 开放定址法举例
  - 给定关键字集合{19, 01, 23, 14, 55, 68, 11, 82, 36}，设定哈希函数H(key)= key MOD 11 （表长=11）
  - 采用线性探测再散列解决冲突

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	68	11	82	36	19		

  - 前面19、01依次插入，23的哈希地址为1，与01冲突，地址+1寻找下一地址，不冲突则插入
  - 14、55都无冲突，直接插入
  - 68与23冲突，共查找3次后插入；11与55冲突，共查找6次后插入
  - 82与11冲突，共查找2次后插入；36与14冲突，共查找5次后插入

9.3 哈希表

三. 处理冲突的方法

- 开放定址法举例
  - 给定关键字集合{19, 01, 23, 14, 55, 68, 11, 82, 36}，设定哈希函数H(key)=key MOD 11 （表长=11）
  - 采用二次探测再散列

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	36	82	68		19		11

  - 前面19、01依次插入，23的哈希地址为1，与01冲突，原地址+1为空插入，14、55都无冲突，直接插入
  - 68与23冲突，原地址+1、-1都冲突，原地址+4为空插入
  - 11与55冲突，原地址+1冲突，原地址-1为空，共查找3次后插入
  - 82无冲突直接插入
  - 36与14冲突，原地址+1为空, 插入

9.3 哈希表

三. 处理冲突的方法

- 开放定址法举例
  - 给定关键字集合{19, 01, 23, 14, 55, 68, 11, 82, 36}，设定哈希函数H(key)=key MOD 11 (表长=11)
  - 采用伪随机探测再散列，采用随机数列：1、8、4、7.....

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	11	82		36	19		68

  - 前面19、01依次插入，23的哈希地址为1，与01冲突，原地址+1为空插入，14、55、82都无冲突，直接插入
  - 68与23冲突，原地址+1冲突，+8为空
  - 11与55冲突，原地址+1、+8都冲突，+4为空
  - 36与14冲突，原地址+1、+8都冲突，+4为空

9.3 哈希表

三. 处理冲突的方法

- 开放定址法的特点
  - 优点：充分利用了哈希表的空间，只要哈希表中有空位置，总能找到一个不发生冲突的地址。
  - 缺点：易产生“二次聚集”，即在处理同义词的冲突过程中，又添加了非同义词的冲突，对查找不利。

9.3 哈希表

三. 处理冲突的方法

2、再哈希法

构造若干个哈希函数，当发生冲突时，计算下一个哈希地址，直到冲突不再发生，即：

$H_i = Rh_i(key)$　　$i=1, 2, \ldots\ldots k$

Rh_i—不同的哈希函数

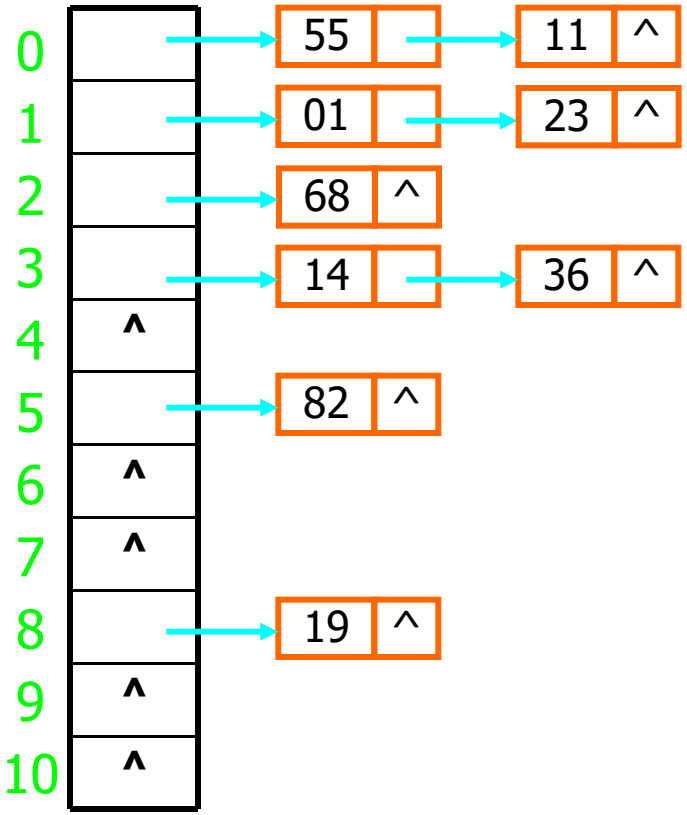
特点：不易产生聚集，但增加计算时间。

9.3 哈希表

三. 处理冲突的方法

3、链地址法

- 将所有哈希地址相同（即关键字为同义词）的记录都存储在一个单链表中，而哈希表中只存储所有这些单链表的头指针。
- 无论有多少个冲突，采用链地址法，都只是在当前位置给单链表增加结点而已，当然也带来了查找时需要遍历单链表的性能损耗。



例如：0下标→55→111,

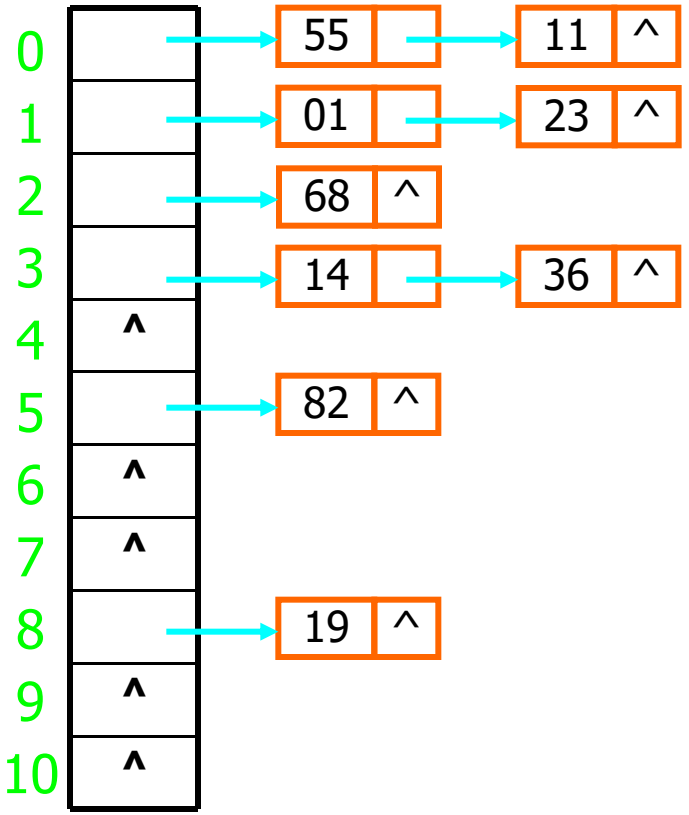
1下标→01→23

9.3 哈希表

三. 处理冲突的方法

3、链地址法（表后插入）

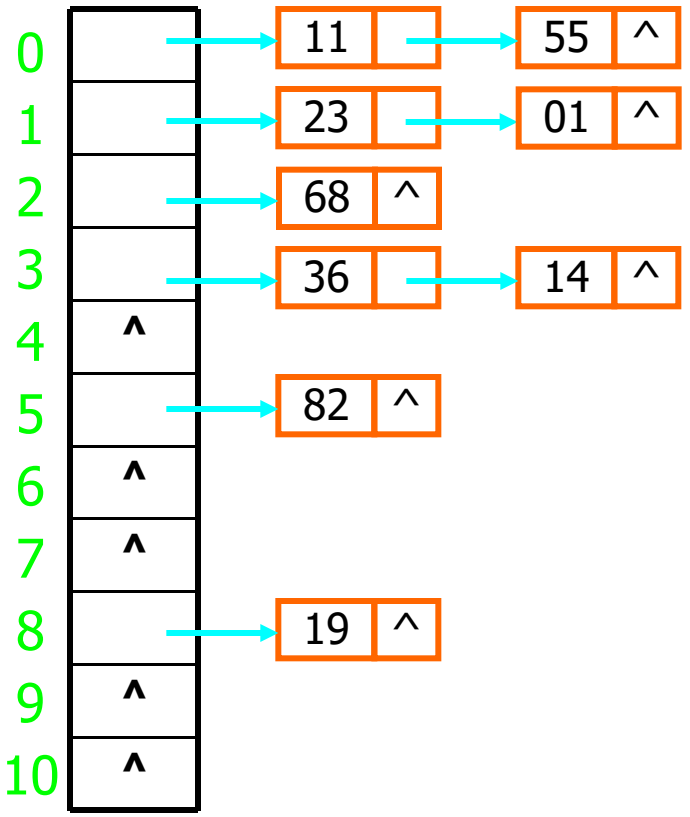
- 将所有哈希地址相同的记录都链接在同一链表中
- 举例：给定关键字集合{19, 01, 23, 14, 55, 68, 11, 82, 36}，设定哈希函数H(key)=key MOD 11 (表长=11) *[表后插入]*
- 查找不成功，再插入新结点时，用表后插入方法较好



9.3 哈希表

三. 处理冲突的方法

- 链地址法（表头插入）
  - 将所有哈希地址相同的记录都链接在同一链表中
  - 举例：给定关键字集合{19, 01, 23, 14, 55, 68, 11, 82, 36}，设定哈希函数H(key)=key MOD 11 (表长=11)
  - 给定关键字集合，逐步生成哈希表时，用表头插入方法较好



9.3 哈希表

三. 处理冲突的方法

- 建立公共溢出区

在基本散列表之外，另外设立一个溢出表保存与基本表中记录冲突的所有记录。

设散列表长为m，设立基本散列表hashtable[m]，每个分量保存一个记录；溢出表overtable[m]，一旦某个记录的散列地址发生冲突，都填入溢出表中。

9.3 哈希表

三. 处理冲突的方法

- 建立公共溢出区

例： 已知一组关键字(15, 4, 18, 7, 37, 47)，散列表长度为7，哈希函数为：H(key)=key MOD 7，用建立公共溢出区法处理冲突。得到的基本表和溢出表如下：

Hashtable表：

散列地址	0	1	2	3	4	5	6
关键字	7	15	37		4	47	

overtable表：

溢出地址	0	1	2	3	4	5	6
关键字	18						

9.3 哈希表

四. 哈希表的实现

- 假设哈希函数为关键字求模运算，哈希表用链地址法解决冲突
- 数据结构可以定义如下：

```
#define LEN 31                  // 表长LEN最好为质数

typedef struct node {
    int data;
    struct node *next;
} node;

struct node HashTab[LEN];
```

9.3 哈希表

四. 哈希表的实现

- 哈希函数可以定义如下
  - 返回值为哈希表地址

```
int hash(int key) {

        retAddr = key MOD LEN;

        return(retAddr);

}
```

9.3 哈希表

五、哈希查找

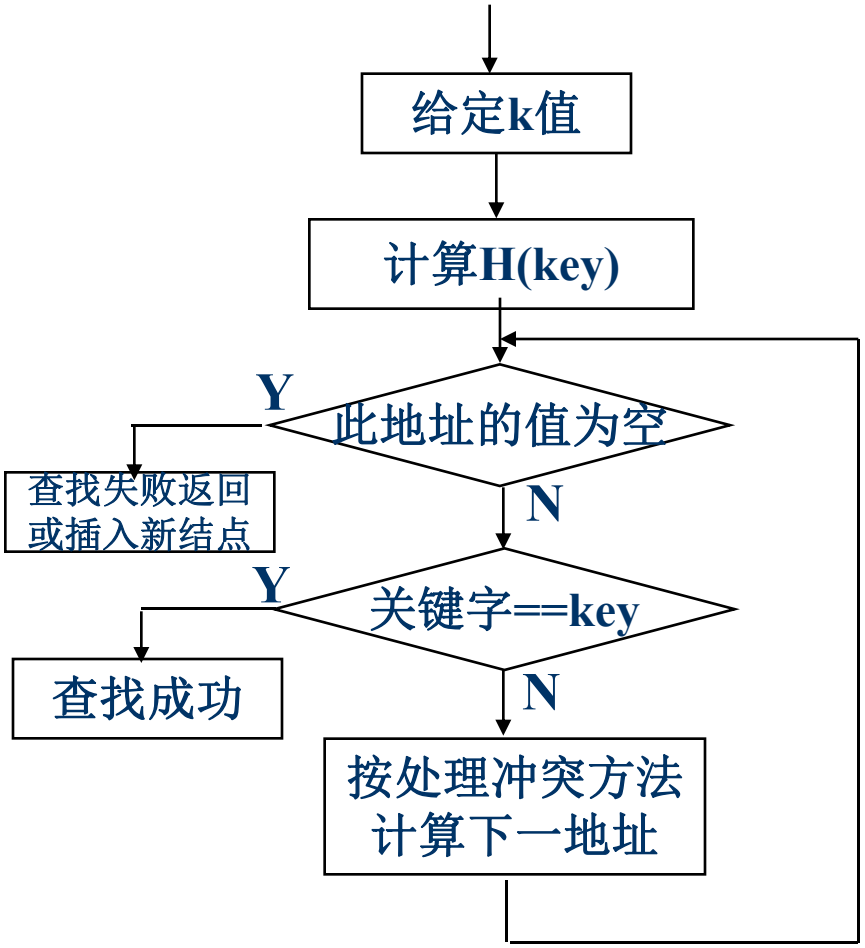
- **哈希查找，**也叫散列查找，是利用哈希函数和处理冲突的方法进行查找的过程。

首先利用关键字及哈希函数计算出哈希地址，然后到指定地址进行查找。

9.3 哈希表

五. 哈希查找

- 查找过程如下
  - 对于给定值key，计算哈希地址 p=hash(Key)
  - 若HashTab[p].next=NULL，则查找不成功
  - 若HashTab[p].next.data = key，则查找成功
  - 否则 “求下一地址”，再进行比较



9.3 哈希表

五. 哈希查找

- 查找函数实现如下
  - 若找到key，返回其结点指针；否则将其插入表中再返回其结点指针[链地址法解决冲突，表头插入]

```
node *search(int key) {
    i = hash(key);
    p = HashTab[i].next;
    while (p != NULL)) {
       if (p.data == key) return(p);
              p = p.next;}
    q = malloc(sizeof(node));    //表头插入
    q.data = key; q.next = HashTab[i].next;
    HashTab[i].next = q;
    return(q);
}
```

9.3 哈希表

哈希查找的性能分析

- 哈希表在关键字与记录的存储位置之间建立了直接映象，如果没有冲突，散列查找是效率最高的，因为它的时间复杂度为O(1)。

然而，“冲突”不可避免的，由于“冲突”的产生，使得哈希表的查找过程仍然是一个给定值和关键字进行比较的过程，因此，仍须以平均查找长度(ASL)作为衡量哈希表的查找效率的量度。

9.3 哈希表

五. 哈希查找的性能分析

- 线性探测再散列的性能分析

假设每个关键字的查找概率相同，则：

$$ASL = \frac{1x4 + 2x2 + 3x1 + 5x1 + 6x1}{9} = 22/9$$

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	68	11	82	36	19		
比较次数 1	1	2	1	3	6	2	5	1		

9.3 哈希表

五. 哈希查找的性能分析

- 二次探测再散列的性能分析

假设每个关键字的查找概率相同，则：

$$ASL = \frac{1x5 + 2x2 + 3x1 + 4x1}{9} = 16/9$$

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	36	82	68		19		11
比较次数 1	1	2	1	2	1	4		1		3

9.3 哈希表

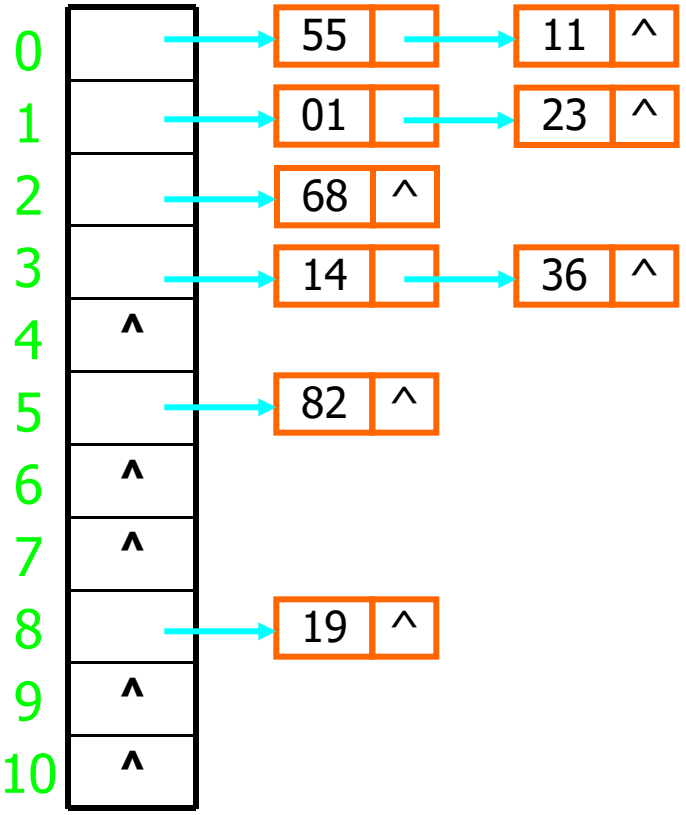
五. 哈希查找的性能分析

- 链地址的性能分析

假设每个关键字的查找概率相同，则：

$$ASL = \frac{1x6 + 2x3}{9}$$

$$= 12/9$$



9.3 哈希表

- 哈希表的**装填因子**是哈希表中填入的记录数与哈希表的长度的比值，即：

  α = 哈希表中填入的记录数 / 哈希表的长度

- 装填因子 α 标志哈希表的装满程度，直观来看：
  - 装填因子 α 越小，发生冲突的可能性就越小
  - 装填因子 α 越大，发生冲突的可能性就越大

9.3 哈希表

五. 哈希查找的性能分析

- 决定哈希表查找的ASL的因素：
  - 选用的哈希函数
  - 选用的处理冲突的方法
  - 哈希表的装填因子

9.3 哈希表

- 哈希表的ASL
  - 线性探测再散列的哈希表查找成功时：
    $ASL \approx (1/2)(1 + 1/(1-\alpha))$
  - 二次探测再散列的哈希表查找成功时：
    $ASL \approx -(1/\alpha)\ln(1-\alpha)$
  - 链地址法处理冲突的哈希表查找成功时：
    $ASL \approx (1 + \alpha/2)$
  - 哈希表的平均查找长度ASL是**α**的函数，而不是表中数据元素个数**n**的函数。

练习

给定关键字集合33、15、88、31、28、64、44、58、77，设定哈希函数 H(key)=key MOD 13 (表长=13)

- 若采用线性探测再散列方式构造哈希表，求哈希表，并给出每个记录的查找次数及平均查找长度；
- 若采用二次探测再散列方式构造，求哈希表，并给出每个记录的查找次数及平均查找长度；
- 若链地址法，表头插入方式构造，求哈希表及平均查找长度。

第九章总结

- 静态查找、动态查找、哈希查找三者概念与区别
- 静态查找：顺序查找、折半查找、索引顺序查找
  - 三种查找的原理、判定树的概念
  - 掌握三种查找方法的查找成功ASL计算
- 动态查找：二叉排序树、平衡二叉树
  - 二叉排序树的概念、查找、插入、删除，中序遍历输出有序序列
  - 平衡二叉树的概念、四种平衡化处理、插入和删除
  - B-和B+树的概念
- 哈希查找：哈希函数、哈希冲突、哈希表，哈希又称为散列
  - 哈希函数的设计方法：
    - 直接定址法、数字分析法、平方取中法、折叠法、除留余数法
  - 处理冲突的方法
    - 开放定址法、再哈希法、链地址法
    - 哈希查找成功的ASL计算