



9.3 哈 希 表

- 哈希表是什么?
- 哈希函数的构造方法
- 处理冲突的方法
- 哈希表的查找及其分析



哈希表是什么？

- 之前讨论的表示查找表的各种结构（线性表、二叉排序树，B-树等）的共同特点：

(1) 数据在查找表中的位置和它的关键字之间不存在一个确定的关系

(2) 查找的过程为给定值依次和数据元素集合中各个关键字进行比较

(3) 查找的效率取决于和给定值进行比较的关键字个数

不同的表示方法，其差别仅在于：关键字和给定值进行比较的顺序不同



哈希表是什么？

- **希望**：数据元素在表中存放位置和其关键字之间存在一种确定的关系→预先知道所查关键字在表中的存放位置
- **理想**：不进行关键字的比较，直接根据关键字的值确定存储位置，到相应位置上查找
- **确定**“关键字的值和存储位置的对应关系”的函数 h ---称为哈希函数

理想是否能实现？实现到什么程度呢？……



哈希表示例

- 例如：为每年招收的 1000 名新生建立一张查找表，其关键字为学号，其值的范围为 **xx000 --xx999** (前两位为年份)。若以下标为 **000--999** 的顺序表表示之。
- **存放位置为学号的后3位——哈希函数**
- 则查找过程可以简单进行：取给定值（学号）的后三位，不需要经过比较便可直接从顺序表中找到待查关键字



哈希表示例

- 关键字序列：3，15，22，24，表长=5。
- 地址计算公式（**哈希函数**）： $h(k)=k\%5$

■ 0 1 2 3 4

15		22	3	24
----	--	----	---	----



{Zhao, Qian, Sun, Li, Wu, Chen, Han, Ye, Dei}

- 设哈希函数 $h(\text{key}) = \lfloor (\text{Ord}(\text{第一个字母}) - \text{Ord}('A') + 1) / 2 \rfloor$

0 1 2 3 4 5 6 7 8 9 10 11 12 13

	Chen	Dei		Han		Li		Qian	Sun		Wu	Ye	Zhao
--	------	-----	--	-----	--	----	--	------	-----	--	----	----	------

哈希表示例

问题：若添加关键字 **Zhou** , 怎么办 ? 

能否找到另一个哈希函数 ?



哈希函数

- 哈希函数是一个映象，即：将关键字的集合映射到某个地址集合上，它的设置很灵活，只要这个地址集合的大小不超出允许范围即可
- 由于哈希函数是一个压缩映象，因此，在一般情况下，很容易产生“冲突”现象，即：
 $key_1 \neq key_2$ ，而 $h(key_1) = h(key_2)$ 。
- 发生冲突的两个数据 key_1, key_2 称为“同义词”
- 上例中“Zhao”和“Zhou”为同义词



哈希函数

- 实际应用中不保证总能找到一个不产生冲突的哈希函数。一般情况下，只能选择恰当的哈希函数，使冲突尽可能少地产生。
- 因此，在构造这种特殊的“查找表”时，除了需要选择一个“**好**” (尽可能少产生冲突)的哈希函数之外；还需要找到一种“**处理冲突**”的方法。



哈希表是什么？

- 根据设定的哈希函数 $h(key)$ 和所选中的**处理冲突**的方法，将一组关键字映射到一个有限的、地址连续的地址集(区间)上，并以关键字在地址集中的“象”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**”。



构造哈希函数的方法

■ 对数字的关键字可有下列构造方法：

1. 直接定址法
2. 数字分析法
3. 平方取中法
4. 折叠法
5. 除留余数法
6. 随机数法

说明：若是非数字关键字，则需先对其进行数字化处理。



1. 直接定址法

- 哈希函数为关键字的线性函数

$$h(key) = key \quad \text{或者}$$

$$h(key) = a \times key + b$$

- 此法仅适合于：地址集合的大小 = 关键字集合的大小



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, \dots, u_s) ，分析关键字集中的全体，并从中提取分布均匀的若干位或它们的组合作为地址。
- 此方法仅适合于：
- 能预先估计出全体关键字的每一位上各种数字出现的频度。

例：一组关键字如下

■	3	4	7	0	5	2	4
■	3	4	9	1	4	8	7
■	3	4	8	2	6	9	6
■	3	4	8	5	2	7	0
■	3	4	8	6	3	0	5
■	3	4	9	8	0	5	8
■	3	4	7	9	7	7	1
■	3	4	7	3	9	1	9
■	<hr/>						
■	①	②	③	④	⑤	⑥	⑦

地址空间：000--999

$$h(hey) = ([key / 1000] \% 10) * 100 +$$
$$([key / 100] \% 10) * 10 +$$
$$key \% 10$$



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。求“关键字的平方值”的目的是“扩大差别”，同时平方值的中间各位又能受到整个关键字中各位的影响。
- 此方法适合于：关键字中的每一位都有某些数字重复出现频度很高的现象。



4. 折叠法

- 将关键字（自左到右，自右到左）分成位数相等的几部分，最后一部分位数可以短些，然后将这几部分叠加求和作为哈希地址。
- 有两种叠加方法：
 1. **移位法** — 将各部分的最后一位对齐相加。
 2. **间界叠加法** — 从一端向另一端沿各部分分界来回折叠后，最后一位对齐相加。
- 加法：进位相加，不进位相加

此方法适合于：关键字的数字位数相对于地址位数特别多。

$k=0442205864$, 地址4位

04|4220|5864

0442|2058|64

■ 5864

■ 4220

■ +) 04 移位相加，进位相加，取和后4为存放地址

■ -----

■ 10088

■ $h(0442205864)=0088$

5864

4220

+) 04

9088

$h(0442205864)=9088$

移位相加，不进位相加，和为存放地址

5864

0224

+) 04

----- 进位叠加，进位相加，取和后4为存放地址

6092

$h(0442205864)=6092$



5. 除留余数法

为什么要对 p 加限制?

设定哈希函数为: $h(\text{key}) = \text{key} \bmod p$,

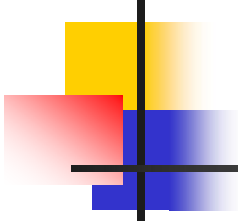
$p \leq m$ (m 为表长), p 应为不大于 m 的素数或是不含 20 以下的质因子.

- 给定一组关键字为: 12, 39, 18, 24, 33, 21, 若取 $p=9$, 则对应的哈希函数值将为: 3, 3, 0, 6, 6, 3
- 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。



6. 随机数法

- 设定哈希函数为: $h(\text{key}) = \text{Random}(\text{key})$
- 其中, Random 为伪随机函数
- 通常, 此方法用于对长度不等的关键字构造哈希函数



实际造表时，采用何种构造哈希函数的方法取决于建表的关键字集合的情况(包括关键字的范围和形态)，总的原则是使产生冲突的可能性降到尽可能地小。

- (1) 计算哈希函数的时间
- (2) 关键字的长度
- (3) 哈希表的大小
- (4) 关键字的分布
- (5) 记录的查找频率



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哈希地址。

设哈希表长度为 m ，存放地址 $0, 1, \dots, m-1$ 。 h 为哈希函数存放关键字为 key 的数据时，其哈希地址为 $h(key)$ ，但该地址已经被与他是**同义词**的其他数据占用，此时发生**冲突**。 key 不能存放于 $h(key)$ 了，要为其另找一个存放位置---**解决冲突**

1. 开放定址法

2. 再哈希法

3. 链地址法

4. 建立一个公共溢出区



1. 开放定址法

- 在地址 $h(key)$ 产生冲突，为解决冲突，设定一个地址探测序列 h_1, h_2, \dots, h_k , $1 \leq k \leq m-1$ ，按照这个探测序列顺序为发生冲突的数据 key 寻找存放位置。

其中： $h_i = (h(key) + d_i) \text{ MOD } m$, $i=1, 2, \dots, k$

- d_i 称为增量，有三种取法：

1) 线性探测再散列： $d_i = i$

2) 平方(二次)探测再散列： $d_i = 1^2, -1^2, 2^2, -2^2, 3^2, -3^2, \dots$,

3) 随机探测再散列： d_i 是一组伪随机数列

冲突地址 h , 表长 m . 线性探测： $h+1, h+2, \dots, m-1, 0, 1, \dots, h-1$

冲突地址 h , 表长 m . 二次探测： $h+1, h-1, h+4, h-4, \dots$,

冲突地址 h ,表长 m . 线性探测: $h+1, h+2, \dots, m-1, 0, 1, \dots, h-1$



{19, 01, 23, 14, 55, 68, 11, 82, 36}

设定哈希函数 $h(\text{key}) = \text{key} \text{ MOD } 11$ (表长 $m=11$)

若采用线性探测再散列处理冲突

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

$ASL = (1+1+2+1+3+6+2+5+1)/9$ 查找每个数据元素需要的比较次数

理想是通过关键字的值确定存放位置, 查找不进行数据元素的比较。
但由于冲突的存在还是要进行比较。只是比较次数少了

二次聚集: 哈希地址不同争夺同一个后继哈希地址, 比如: 23与68



{19, 01, 23, 14, 55, 68, 11, 82, 36}

设定哈希函数 $h(key) = key \text{ MOD } 11$ (表长 $m=11$)

若采用二次探测再散列处理冲突

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

查找每个数据元素需要的比较次数

$$ASL = (1+1+2+1+2+1+4+1+3)/9$$



增量 d_i 应具有“完备性”

- 产生的 h_i 均不相同，且所产生 $(m-1)$ 个 h_i 值能覆盖哈希表中所有地址。
- 平方探测时的表长 m 必为形如 $4j+3$ 的素数（如：7, 11, 19, 23, ... 等）；
- 随机探测时的 m 和 d_i 没有公因子

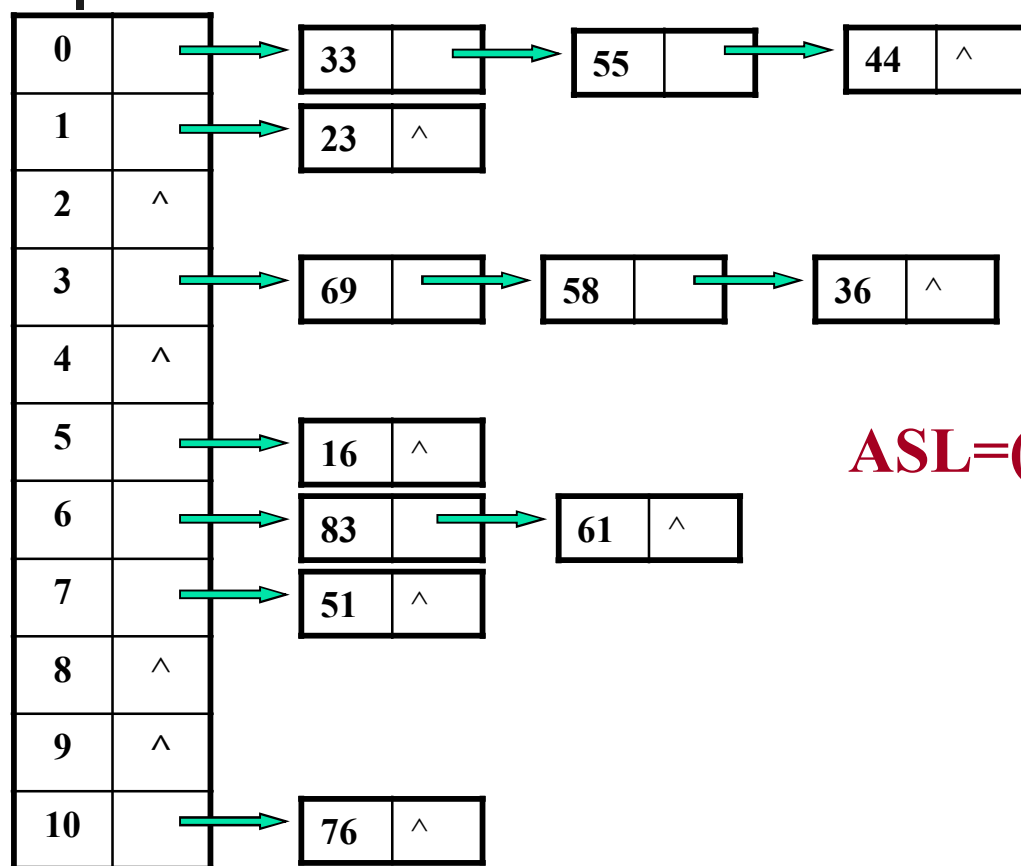


2. 再哈希法

- 为产生冲突的地址 $h(key)$ 求得一个地址序列：
 $h_1, h_2, \dots, h_k, \quad 1 \leq k \leq m-1$, 其中：
- $h_i(key) = Rh_i(key), i=1, 2, \dots, k$
- Rh_i 是不同的哈希函数
- 不宜产生“聚集”
- 计算费时

3. 链地址法 将所有哈希地址相同的数据都链接在同一链表中。

{36,51,44,58,23,55,61,76,83,33,16,69}



例如:哈希函数为

$$h(key) = key \text{ MOD } 11$$

$$ASL = (7 \times 1 + 3 \times 2 + 2 \times 3) / 12 = 19 / 12$$



4. 建立一个公共溢出区

- 将发生冲突的数据元素顺序存放于一个公共的溢出区



哈希表的查找

■ 查找过程和造表过程一致

- 例1：建表时哈希函数为 $H(key)=key \text{ MOD } p$ ，处理冲突采用线性探测再散列，那么在已经建好的表长为 m 的哈希表中查找 x 的步骤如下：
 1. 计算 $h=x \text{ MOD } p$
 2. 若哈希表中，地址 h 处为空闲，则 **查找失败，返回**；否则将 x 与哈希表中 h 处的数据元素比较，若相等，则 **查找成功，返回**；否则转3
 3. 按照线性探测再散列的地址探查序列： $h+1, h+2, \dots, m-1, 0, 1, \dots, h-1$ ，从： $h+1$ 开始查看。若当前查看的位置为空闲，则 **查找失败，返回**；否则将 x 与当前查看位置的数据元素进行比较，相等则 **查找成功，返回**；不相等，则按照地址探查序列继续查看下一位置。若整个地址探查序列查看一遍均没找到，则 **查找失败，返回**。



{19, 01, 23, 14, 55, 68, 21, 84, 32}

设定哈希函数 $H(key) = key \text{ MOD } 11$ (表长 $m=11$)

若采用线性探测再散列处理冲突

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	68	32		84	19		21
1	1	2	1	3	7		1	1		1

$ASL = (1+1+2+1+3+7+1+1+1)/9$ 查找每个数据元素需要的比较次数

二次聚集: 哈希地址不同争夺同一个后继哈希地址



哈希表查找的分析

- 从查找过程得知，哈希表查找的平均查找长度实际上并不等于零。
- 决定哈希表查找的ASL的因素：
 - 1) 选用的哈希函数；
 - 2) 选用的处理冲突的方法；
 - 3) 哈希表饱和的程度，装载因子 $\alpha = n/m$ 值的大小（ n —数据数， m —表的长度）



哈希表查找的分析

一般情况下，可以认为选用的哈希函数是“均匀”的，则在讨论ASL时，可以不考虑它的因素。

因此，哈希表的ASL是处理冲突方法和装载因子的函数。

例如：前述例子

线性探测处理冲突时， $ASL = 22/9$

双散列探测处理冲突时， $ASL = 14/9$

链地址法处理冲突时， $ASL = 13/9$



可以证明：查找成功时有下列结果：

线性探测再散列

$$S_{nl} \approx \frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$$

随机探测再散列

$$S_{nr} \approx -\frac{1}{\alpha} \ln(1-\alpha)$$

链地址法

$$S_{nc} \approx 1 + \frac{\alpha}{2}$$



哈希表

- 从以上结果可见：
 - 哈希表的平均查找长度是 α 的函数，而不是 n 的函数。
 - 这说明，用哈希表构造查找表时，可以选择一个适当的装填因子 α ，使得平均查找长度限定在某个范围内。
- 这是哈希表所特有的特点。



哈希表的删除操作

- 开放定址法：保证删除后查找表的操作正常进行
- 链地址法：直接删除
- 再哈希法：保证删除后查找表的操作正常进行
- 建立一个公共溢出区：保证删除后查找表的操作正常进行



哈希表的删除操作

- 开放定址法：删除一个数据，为保证查找工作的正常进行不能真删----加删除标志

若采用线性探测再散列处理冲突，删除时**只加删除标记**，以保证删除后的哈希表的操作（查找，插入，删除）能正常进行

{19, 01, 23, 14, 55, 68, 11, 82, 36}

设定哈希函数 $h(key) = key \text{ MOD } 11$ (表长 $m=11$)

0 1 2 3 4 5 6 7 8 9 10

55	01	23	14	68	11	82	36	19		
----	----	----	----	----	----	----	----	----	--	--

删除68后

0 1 2 3 4 5 6 7 8 9 10

55	01	23	14		11	82	36	19		
----	----	----	----	--	----	----	----	----	--	--

插入操作遇到加删除标记的位置，当成空闲的空间，直接放入插入的数据；查找操作遇到加删除标记的位置，当成非空闲空间，但是不做比较操作。

探测再散列处理冲突：删除操作也**只加删除标记**



哈希表也可以用来构造静态查找表

- 对静态查找表，有时可以找到不发生冲突的哈希函数。即，此时的哈希表称此类哈希函数为理想（perfect）的哈希函数。