计算机领域本科教育教学改革试点
工作计划（“101计划”）研究成果

数据结构

俞勇、张铭、陈越、韩文弢

上海交通大学、北京大学、浙江大学、清华大学

第 11 章 查找

11.4 散列查找

林劼

电子科技大学

提纲

11.4.1散列函数

平均比较次数

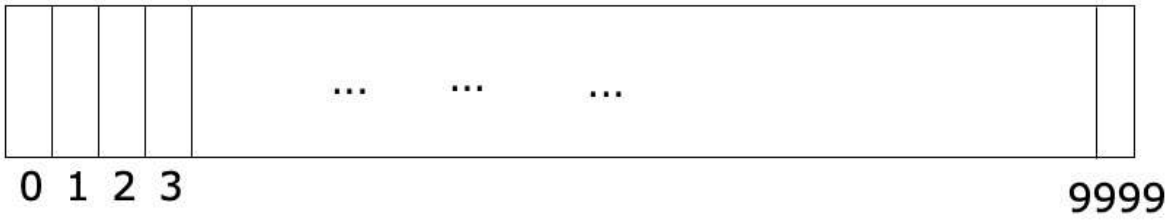
- 顺序查找：O(n)
- 二分查找：O(log(n))
- 二分查找树BST: O(d) --d是树的高度
- 平衡二叉树AVL：O(log(n))
- 索引查找：由索引表决定

有没有效率更高的算法?

11.4.1散列函数

问题：设所有学生的学号分布在区间[20220000, 20229999]内，什么数据结构方便记录学生信息，并实现**快速查询**？

- 最多有10000名学生，可用顺序表存储，顺序表的长度（容量）为10000

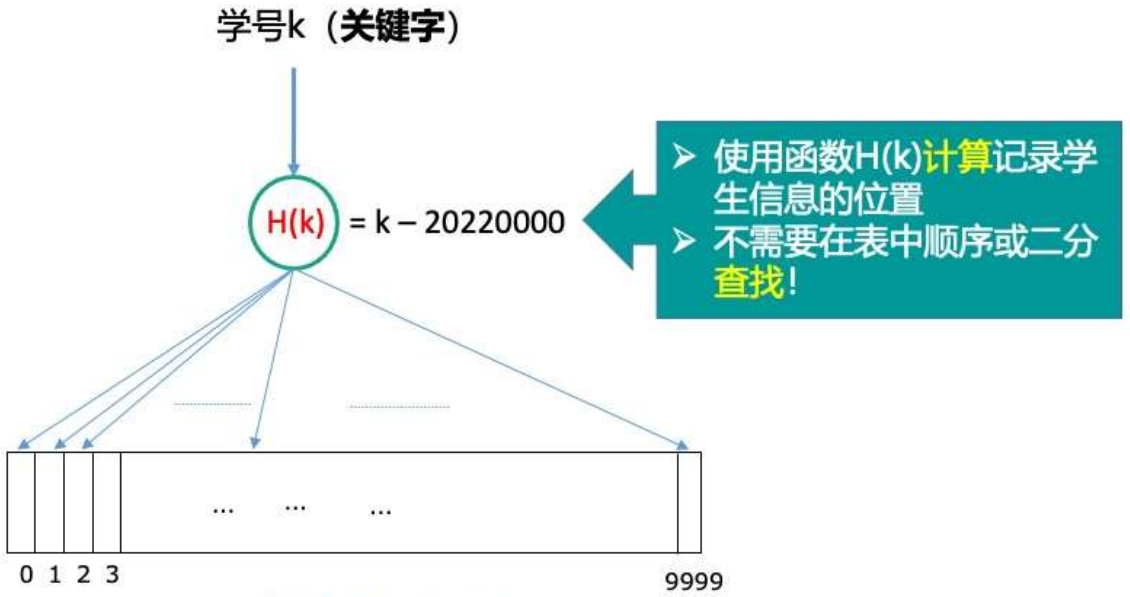


- 学号k的学生的信息记录在表中哪个位置，查询最简单、最方便?
  - 使用函数 H(k) = k – 20220000, 来**计算**表中记录学生k的位置（索引）

查询时间：O(1)

11.4.1散列函数

- 取出学号的后四位，**不需要经过比较**，便可**直接**从查找表中**找到**给定学生的记录。



11.4.1散列函数

散列函数定义

一般情况下，需在关键字与记录在表中的存储位置之间建立一个函数关系，以 H(key) 作为关键字为 key 的记录在表中的位置，通常称这个函数 H(key) 为散列函数。

11.4.1散列函数

1） 散列函数是一个**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；

11.4.1散列函数

1) 散列函数是一个**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；

2) 由于散列函数是一个**压缩映象**，因此，在一般情况下，很容易产生“冲突”现象，即：**key1≠ key2**，而 **H(key1) = H(key2)。**

如：H(key)=key % 10

H(2) = 2, H(12)=12, H(102)=2,....

个位相同的关键字全部冲突!

11.4.1散列函数

1） 散列函数是一个**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；

2）由于散列函数是一个**压缩映象**，因此，在一般情况下，很容易产生“冲突”现象，即： **key1≠ key2**，而 **H(key1) = H(key2)**。

3）**很难**找到一个不产生冲突的散列函数。一般情况下，**只能**选择恰当的散列函数，使冲突尽可能少地产生。

因此，散列查找需要做两方面事情：

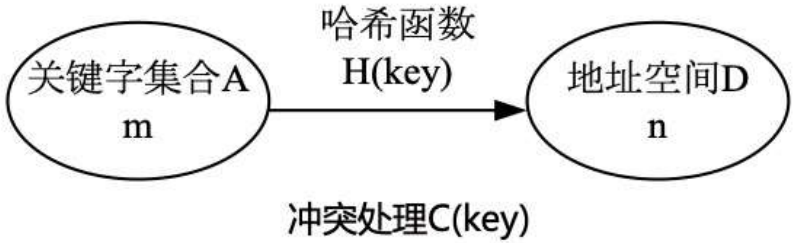
（1）选择一个“**好**”的散列函数；

（2）提供一种“**好**”的冲突处理方法。

11.4.1散列函数

散列表

根据设定的**散列函数 H(key)** 和提供的**处理冲突的方法**，将一组关键字**映象到**一个地址连续的地址空间上，并以关键字在地址空间中的**“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**散列表。**



地址空间存储的数据集合称为散列表

11

11.4.1散列函数

散列函数

一般来说，一个好的散列函数应满足下列两个条件：

(1) 计算简单

(2) 冲突少

散列函数

例： H(key) = key % 16

(1) 计算简单 ✓ 可用位运算：key & 15

(2) 冲突少 ✗ 哈希值只取决于关键字二进制值的最低四位!

散列函数

例：

算法：HashCode1(s, D)
输入：字符串s，正整数D
输出：计算字符串对应的哈希值

```
1. sum ← 0
2. for i ← 0 to Length(s)-1 do
3. |  sum ← sum + s[i]
4. end
5. return sum % D
```

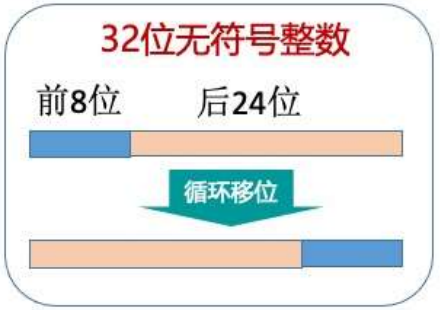
(1) 计算简单 ✓ 所有字符的数值总和

(2) 冲突少 ✗ 没有考虑字符的排列顺序!

散列函数

例：

```
算法：HashCode2(s, D)
输入：字符串s，正整数D
输出：计算字符串对应的哈希值
1. sum ← 0
2. for i ← 0 to Length(s)-1 do
3. |  sum ← (sum << 8) | (sum >> 24)   //cyclic shift
4. |  sum ← sum + s[i]
5. end
6. return sum % D
```



(1) 计算简单 — 所有字符的数值总和+循环移位

(2) 冲突少 — 结果受字符的排列顺序影响！

散列函数

常见的散列函数构造方法有：

- 直接散列函数
- 数字分析法
- 平方取中法
- 折叠法
- 除留余数法
- 随机数法

散列函数

1）直接散列函数：

- 取关键字本身或关键字的某个线性函数值作为散列地址，
  即：H(key) = key
  或：H(key) = a* key + b（a，b为常数）。

解放后每年出生人数的统计：

散列地址						
出生年份	1949	1950	1951	……	1970	……
出生人数	××××	××××	××××	……	××××	……

H(key) = key + (-1948)

散列函数

2）数字分析法

设n个d位数的关键字，由r个不同的符号组成，此r个符号在关键字各位出现的频率不一定相同：

- 在某些位上均匀分布，即每个符号出现的次数都接近于n / r次；
- 在另一些位上分布不均匀。

数学分析

则**选择其中分布均匀的s位作为散列地址**，即 H(key) = “key中数字均匀分布的s位”

								散列地址
8	1	3	4	6	5	3	2	→ 45
8	1	3	7	2	2	4	2	→ 72
8	1	3	8	7	4	2	2	→ 84
8	1	3	0	1	3	6	7	→ 03
8	1	3	2	2	8	1	7	→ 28
8	1	3	3	8	9	6	7	→ 39
8	1	3	5	4	1	5	7	→ 51
8	1	3	6	8	5	3	7	→ 65
8	1	4	1	9	3	5	5	→ 13

n=80, d=8, r=10, s=2:

- 第1，2，3，8位分布不均匀，不能取
- 第4，5，6，7位分布相对均匀

取第4、6两位组成的2位十进制数作为每个数据的散列地址

散列函数

3）平方取中法

取关键字平方后的中间几位作为散列地址，即散列函数为：

H（key）=“ key^2 的中间几位”

其中，所取的位数由散列表的大小确定

数据	关键字	(关键字)²	散列地址
A	0100	0010000	010
I	1100	1210000	210
J	1200	1440000	440
I0	1160	1370400	370
P1	2061	4310541	310
P2	2062	4314704	314
Q1	2161	4734741	734
Q2	2162	4741304	741
Q3	2163	4745651	745

散列函数

3）平方取中法

平方取中法思想

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**”和“**贡献均衡**”。

即：关键字的各位都在平方值的中间几位有所贡献，Hash值中应该有各位影子。

散列函数

关键字位数特别多，怎么办？

散列函数

4）折叠法

关键字位数较长时，可将关键字**分割成位数相等的几部分**（最后一部分位数可以不同），取这几部分的叠加和（舍去高位的进位）作为散列地址。位数由存储地址的位数确定。

- 叠加时有两种方法：
  - 移位叠加法，即将每部分的最后一位对齐，然后相加；
  - 边界叠加法，即把关键字看作一纸条，从一端向另一端沿边界逐次折叠，然后对齐相加。

$$
\begin{array}{r}
d_r\cdots\cdots\ d_2\ \ d_1 \\
d_{2r}\cdots\cdots d_{r+2}\ \ d_{r+1} \\
+)\ d_{3r}\cdots\cdots d_{2r+2}\ d_{2r+1} \\
\hline
S_r\cdots\cdots\ S_2\ \ S_1
\end{array}
$$

(a) 移位叠加法

$$
\begin{array}{r}
d_r\cdots\cdots\ d_2\ \ d_1 \\
d_{r+1}\cdots\cdots d_{2r-1}d_{2r} \\
+)\ d_{3r}\cdots\cdots d_{2r+2}\ d_{2r+1} \\
\hline
S_r\cdots\cdots\ S_2\ \ S_1
\end{array}
$$

(b) 边界叠加法

此方法适合于：关键字的数字位数特别多。

散列函数

5）除留余数法

取关键字被某个不大于散列表长度m的数p除后的余数作为散列地址，即：

H（key）=key MOD p（p≤m）

例 p=21

关键字	28	35	63	77	105
哈希地址	7	14	0	14	0

其中p的选择很重要，如果选得不好会产生很多冲突。

比如关键字都是10的倍数，而p=10

- 一般取小于表长的最大质数

散列函数

6） 随机数法

选择一个随机函数，取关键字的随机函数值作为散列地址，

即：H（key）=random（key）

其中random为随机函数。

实际工作中需根据不同的情况采用不同的散列函数。通常需要考虑的因素有：

计算散列函数所需时间；

关键字的长度；

散列表的大小；

关键字的分布情况；

记录的查找频率。

散列函数

常用的哈希函数

1. MD4（RFC 1320）是 MIT 的 Rivest 在 1990 年设计的，其输出为 128 位。MD4 已证明不够安全
2. MD5（RFC 1321）对 MD4 的改进版本。它对输入仍以 512 位分组，其输出是 128 位。MD5 比 MD4 复杂，并且计算速度要慢一点，更安全一些。MD5 已被证明不具备"强抗碰撞性"
3. SHA （Secure Hash Algorithm）是一个 Hash 函数族，由 NIST于 1993 年发布第一个算法。目前知名的 SHA-1 在 1995 年面世，它的输出为长度 160 位的 hash 值，因此抗穷举性更好。SHA-1 设计时基于和 MD4 相同原理，并且模仿了该算法。SHA-1 已被证明不具"强抗碰撞性"。

散列函数

- MD5的碰撞案例

```
import hashlib

#  两段HEX字节串，注意它们有细微差别
a =
bytearray.fromhex("0e306561559aa787d00bc6f70bbdfe3404cf03659e704f8534c00ffb659c4c8
740cc942feb2da115a3f4155cbb8607497386656d7d1f34a42059d78f5a8dd1ef")

b =
bytearray.fromhex("0e306561559aa787d00bc6f70bbdfe3404cf03659e744f8534c00ffb659c4c8
740cc942feb2da115a3f415dcbb8607497386656d7d1f34a42059d78f5a8dd1ef")

#  输出MD5，它们的结果一致
print(hashlib.md5(a).hexdigest())
print(hashlib.md5(b).hexdigest())
```

a和b输出结果都为：

cee9a457e790cf20d4bdaa6d69f01e41
cee9a457e790cf20d4bdaa6d69f01e41

散列函数

- SHA以及SHA1碰撞
  - SHA与MD5算法本质上是类似的，但安全性要领先很多——这种领先型更多的表现在碰撞攻击的时间开销更大，当然计算时间慢
  - SHA有SHA0、SHA1、SHA256、SHA384等等，它们的计算方式和计算速度都有差别。
  - SHA1是现在用途最广泛的一种算法。包括GitHub在内的版本控制工具以及各种云同步服务都是用SHA1来区别文件。长期以来，人们都认为SHA1是十分安全的，至少大家还没有找到一次碰撞案例。
  - 但在2017年2月，CWI和Google的研究人员们成功找到了一例SHA1碰撞，而且很厉害的是，发生碰撞的是两个真实的、可阅读的PDF文件。这两个PDF文件内容不相同，但SHA1值完全一样。
  - 所以，对于一些大的商业机构来说， MD5 和 SHA1 已经不够安全，推荐至少使用 SHA2-256 算法。

建立散列表如果遇到两个相同关键字的情况怎么办?

有可能有两个相同关键字吗?

11.4.2 冲突处理

冲突：

是指由关键字得到的Hash地址上已有其他记录。

好的散列函数可以减少冲突，但很难避免冲突。

冲突处理：

为出现散列地址冲突的关键字寻找下一个散列地址。

11.4.2 冲突处理

常见的冲突处理方法有：

- 开放地址法
- 再散列法
- 链地址法
- 公共溢出区法

11.4.2 冲突处理

1）开放地址法

为产生冲突的地址 H(key) 求得一个地址序列：

$H_0, H_1, H_2, \ldots, H_s$　　$1 \le s \le m-1$

其中：　$H_0 = H(key)$

$H_i = (H(key) + d_i) \text{ MOD } m$

$i=1, 2, \ldots, s$

其中：Hi 为第i次冲突的地址，$i=1, 2, \ldots, s$

H(key) 为Hash函数值

m 为Hash表表长

d_i 为增量序列

11.4.2 冲突处理

1）开放地址法

对增量 d_i 有三种取法：

1) 线性探测再散列 (linear probing)

$d_i = c \times i$　一般情况：$c=1$

2) 平方探测再散列

$d_i = 1^2, -1^2, 2^2, -2^2, \ldots$，或者 $d_i=1^2, 2^2, 3^2, \ldots$

3) 随机探测再散列

d_i 是一组伪随机数列

11.4.2 冲突处理

例:

- **表长为11的散列表中已填有关键字为17，60，29的记录**
- **哈希函数：H(key) = key MOD 11**
- **现增加第4个记录，其关键字为38，**

线性探测法

① 逻辑上，把哈希表当作首尾相连的循环结构

② 从散列地址开始“向下”逐个探寻，直到找到表中第一个空位置，然后填入数据

③ 容易造成大量数字聚集在一个区域的情况，降低插入和查找的效率!

0	1	2	3	4	5	6	7	8	9	10
			38	38	60	17	29	38		

按三种处理冲突的方法，将38填入散列表中

线性探测

(1) H(38) = 38 MOD 11 = 5 冲突
H_1 = (5+1) MOD 11 = 6 冲突
H_2 = (5+2) MOD 11 = 7 冲突
H_3 = (5+3) MOD 11 = 8 不冲突

平方探测

(2) H(38) = 38 MOD 11 = 5 冲突
H_1 = (5+1^2) MOD 11 = 6 冲突
H_2 = (5-1^2) MOD 11 = 4 不冲突

随机探测

(3) H(38) = 38 MOD 11 = 5 冲突
设伪随机数序列为9，则:
H_1 = (5+9) MOD 11 = 3 不冲突

高等教育出版社

11.4.2 冲突处理

2）再散列法

将n个不同散列函数排成一个序列，当发生冲突时，由RHi确定第i次冲突的地址Hi。

即：

Hi =RHi (key) i=1, 2, …, n

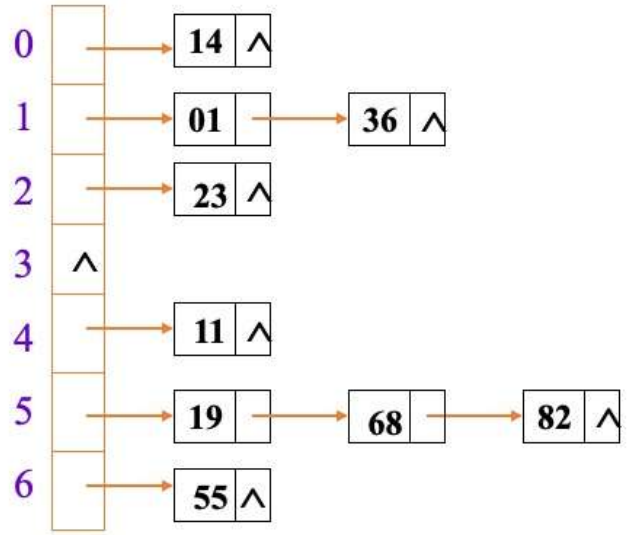
其中：RHi 为不同散列函数

这种方法不会产生“聚类”，但会增加计算时间。

11.4.2 冲突处理

3）链地址法：

将所有散列地址相同的记录都链接在同一链表中。



关键字集合为

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

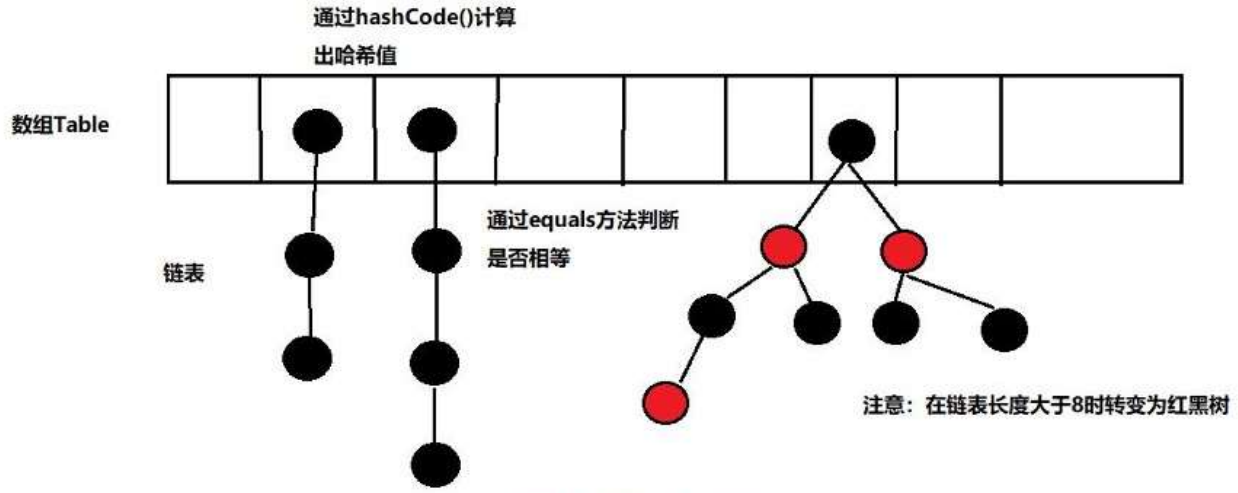
散列函数为 H(key)=key MOD 7

11.4.2 冲突处理

3）链地址法：

将所有散列地址相同的记录都链接在同一链表中。

- 应用：HashMap (JDK1.8)



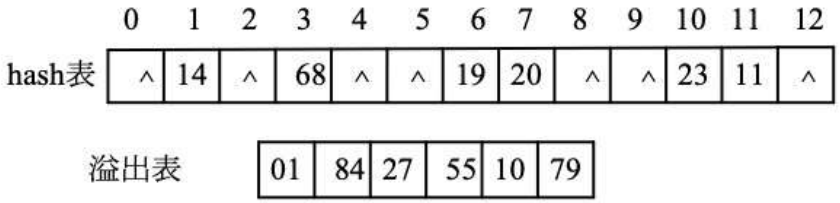
11.4.2 冲突处理

4)公共溢出区法

•假设某散列函数的值域[0, m-1]，

•向量HashTable[0, m-1]为基本表，每个分量存放一个记录，另设一个向量OverTable[0, v]为溢出表。将与基本表中的关键字发生冲突的所有记录都填入溢出表中。

•如一组关键字序列为{19，14，23，01，68，20，84，27，55，11，10，79}，散列函数为H（key）= key mod 13，采用公共溢出区法得到的结果为：



11.4.3 散列表查找算法

在散列表上查找的过程和散列造表的构造过程基本一致。

1）给定K值，根据构造表时所用的散列函数求散列地址 j

2）若此位置无记录，则查找不成功

3）如果有记录，比较关键字

4）如果和给定的关键字相等则成功

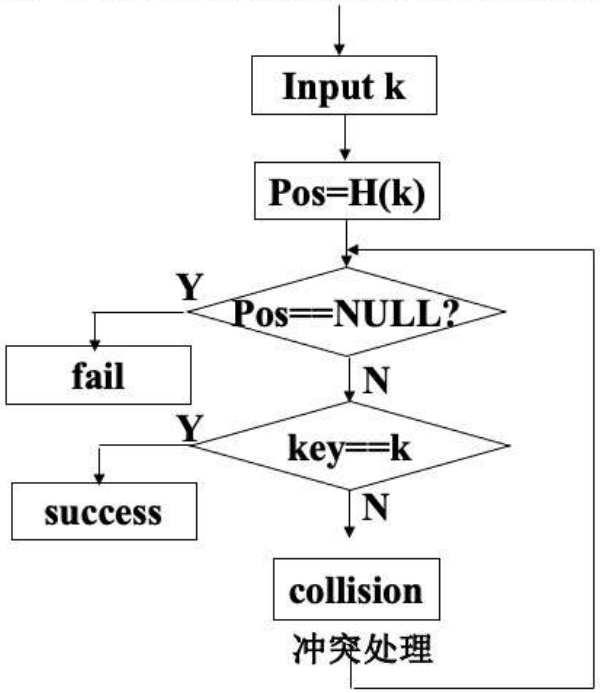
5）否则根据构造表时设定的冲突处理的方法计算“下一地址”， 重复2）

可能需要查重，避免在表装满或重复比较同一个关键字！

如果散列表始终留有空位，可以不用查重（线性探测？）

11.4.3 散列表查找算法

存在冲突检测与处理的散列查找流程图



11.4.3 散列表查找算法

散列表查找与插入算法举例

关键字序列为：

{19，14，23，01，68，20，84，27，55，11，10，79}

散列函数为H（key）= key mod 13

采用线性探测处理冲突

建立散列查找表如下：请查找关键字为84的记录

0	1	2	3	4	5	6	7	8	9	10	11	12
	14	01	68	27	55	19	20	84	79	23	11	10

Key=84

散列地址H(84)=6，因为e.data[6]不空，且e.data[6].key=19≠84，冲突

冲突处理H1=(6+1)MOD13=7，e.data[7]不空，且e.data[7].key=20≠84, 冲突

冲突处理H2=(6+2)MOD13=8，e.data[8]不空，且e.elem[8].key=84，查找成功，返回数据在散列表中的序号8。

11.4.3 散列表查找算法

散列表查找与插入算法举例

关键字序列为：

{19，14，23，01，68，20，84，27，55，11，10，79}

散列函数为H（key）= key mod 13

采用线性探测处理冲突

建立散列查找表如下：请查找关键字为38的记录

0	1	2	3	4	5	6	7	8	9	10	11	12
	14	01	68	27	55	19	20	84	79	23	11	10

Key=38

散列地址H(38)=12，因为e.data[12]不空，且e.data[12].key=10≠38, 冲突

冲突处理H1=(12+1)MOD13=0，由于e.data[0]没有存放数据，表明散列表中不存在关键字为38的记录，查找失败。

11.4.4 查找性能分析

例题：关键字集合

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

设定散列函数 H(key) = key MOD 11 (表长=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		

使用线性探测法解决冲突

(1) 求查找成功的ASL

(2) 查找失败的ASL

11.4.4 查找性能分析

例题：关键字集合

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

设定散列函数 H(key) = key MOD 11 (表长=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(成功)=(1+1+2+1+3+6+2+5+1) / 9 = 22/9

11.4.4 查找性能分析

例题：关键字集合

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

设定散列函数 H(key) = key MOD 11 (表长=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(失败)=？ 如果查找数据的散列地址是9和10，无需移动，如果地址是其它值，要移动至9

ASL(失败) = (10+9+8+7+6+5+4+3+2+1+1) / 11=56/11

11.4.4 查找性能分析

总 结 ——映射的散列函数

关键字范围广 —散列函数→ 存储空间范围小

冲突不可避免，不同解决冲突的策略的ASL不同

查找表大小与解决冲突策略和ASL范围相关

选择散列函数 → 根据冲突策略与ASL计算散列表大小 → 建立查找表

11.4.5分布式散列表

当面对分布式系统时，传统用于单机系统的散列表数据结构已无法支撑数据存储应用需求。此时，需要采用能够支撑分布式系统的散列数据结构，即分布式散列列表DHT。