

❖ 处理冲突的方法

常用处理冲突的思路:

- 换个位置: 开放地址法
- 同一位置的冲突对象组织在一起: 链地址法

❖ 开放定址法 (Open Addressing)

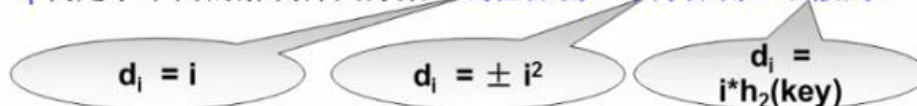
一旦产生了冲突 (该地址已有其它元素), 就按某种规则去寻找另一空地址

❖ 开放定址法 (Open Addressing)

✎ 若发生了第 i 次冲突, 试探的下一个地址将增加 d_i , 基本公式是:

$$h_i(\text{key}) = (h(\text{key}) + d_i) \bmod \text{TableSize} \quad (1 \leq i < \text{TableSize})$$

✎ d_i 决定了不同的解决冲突方案: 线性探测、平方探测、双散列。



1. 线性探测法 (Linear Probing)

❖ 线性探测法: 以增量序列 $1, 2, \dots, (\text{TableSize} - 1)$ 循环试探下一个存储地址。

[例] 设关键词序列为 $\{47, 7, 29, 11, 9, 84, 54, 20, 30\}$,

□ 散列表表长 $\text{TableSize} = 13$ (装填因子 $\alpha = 9/13 \approx 0.69$);

□ 散列函数为: $h(\text{key}) = \text{key} \bmod 11$ 。

用线性探测法处理冲突, 列出依次插入后的散列表, 并估算查找性能

关键词 (key)	47	7	29	11	9	84	54	20	30
散列地址 $h(\text{key})$	3	7	7	0	9	7	10	9	8

关键词 (key)	47	7	29	11	9	84	54	20	30
散列地址 h(key)	3	7	7	0	9	7	10	9	8

地址 操作	0	1	2	3	4	5	6	7	8	9	10	11	12	说明
插入47				47										无冲突
插入7				47				7						无冲突
插入29				47				7	29					$d_1 = 1$
插入11	11			47				7	29					无冲突
插入9	11			47				7	29	9				无冲突
插入84	11			47				7	29	9	84			$d_3 = 3$
插入54	11			47				7	29	9	84	54		$d_1 = 1$
插入20	11			47				7	29	9	84	54	20	$d_3 = 3$
插入30	11	30		47				7	29	9	84	54	20	$d_6 = 6$

散列表查找性能分析

- 成功平均查找长度(ASLs)
- 不成功平均查找长度 (ASLu)

散列表:

H(key)	0	1	2	3	4	5	6	7	8	9	10	11	12
key	11	30		47				7	29	9	84	54	20
冲突次数	0	6		0				0	1	0	3	1	3

【分析】

ASLs: 查找表中关键词的平均查找比较次数 (其冲突次数加1)

$$ASL\ s = (1+7+1+1+2+1+4+2+4) / 9 = 23/9 \approx 2.56$$

ASLu: 不在散列表中的关键词的平均查找次数 (不成功)

一般方法: 将不在散列表中的关键词分若干类。

如: 根据H(key)值分类

$$ASL\ u = (3+2+1+2+1+1+1+9+8+7+6) / 11 = 41/11 \approx 3.73$$

散列函数 $h(key) = key \bmod 11$

因此, 分为11类分析

中国八

Wu Wu Wu

ASLs: 表中关键词的平均查找比较次数

ASLu: 不在散列表中的关键词的平均查找次数（不成功）

$$\text{ASL}_u = (9+8+7+6+5+4+3+2+1 \cdot 18) / 26 = 62/26 \approx 2.38$$
 清华大学

❖ 平方探测法：以增量序列 $1^2, -1^2, 2^2, -2^2, \dots, q^2, -q^2$ 且 $q \leq \lfloor \text{TableSize}/2 \rfloor$ 循环试探下一个存储地址。

□ 散列表表长 $\text{TableSize} = 11$,

用平方探测法处理冲突，列出依次插入后的散列表，并估算ASLs。

[illegible]

关键词 key	47	7	29	11	9	84	54	20	30
散列地址 h(key)	3	7	7	0	9	7	10	9	8
冲突次数	0	0	1	0	0	2	0	3	3

$$ASL_s = (1+1+2+1+1+3+1+4+4) / 9 = 18/9 = 2$$

地址 操作	0	1	2	3	4	5	6	7	8	9	10	说明
插入47				47								无冲突
插入7				47				7				无冲突
插入29				47				7	29			$d_1 = 1$
插入11	11			47				7	29			无冲突
插入9	11			47				7	29	9		无冲突
插入84	11			47			84	7	29	9		$d_2 = -1$
插入54	11			47			84	7	29	9	54	无冲突
插入20	11		20	47			84	7	29	9	54	$d_3 = 4$
插入30	11	30	20	47			84	7	29	9	54	$d_3 = 4$

2. 平方探测法 (Quadratic Probing)

是否有空间，平方探测(二次探测)就能找得到？

5	6	7		
0	1	2	3	4

$$h(k) = k \bmod 5$$

插入 11, $h(11)=1$

探测序列: $1+1=2$, $1-1=0$, $(1+2^2)\bmod 5=0$, $(1-2^2)\bmod 5=2$,
 $(1+3^2)\bmod 5=0$, $(1-3^2)\bmod 5=2$, $(1+4^2)\bmod 5=2, \dots$

有定理显示：如果散列表长度 TableSize 是某个 $4k+3$ (k 是正整数) 形式的素数时，平方探测法就可以探查整个散列表空间。

在开放地址散列表中，删除操作要很小心。通常只能“懒惰删除”，即需要增加一个“删除标记(Deleted)”，而并不是真正删除它。以便查找时不会“断链”。其空间可以在下次插入时重用。

3. 双散列探测法 (Double Hashing)

双散列探测法: d_i 为 $i \cdot h_2(\text{key})$, $h_2(\text{key})$ 是另一个散列函数
探测序列成: $h_2(\text{key}), 2h_2(\text{key}), 3h_2(\text{key}), \dots$

对任意的 key , $h_2(\text{key}) \neq 0$!

探测序列还应该保证所有的散列存储单元都应该能够被探测到。
选择以下形式有良好的效果:

$$h_2(\text{key}) = p - (\text{key} \bmod p)$$

其中: $p < \text{TableSize}$, p 、 TableSize 都是素数。

4. 再散列 (Rehashing)

当散列表元素太多 (即装填因子 α 太大) 时, 查找效率会下降;

实用最大装填因子一般取 $0.5 \leq \alpha \leq 0.85$

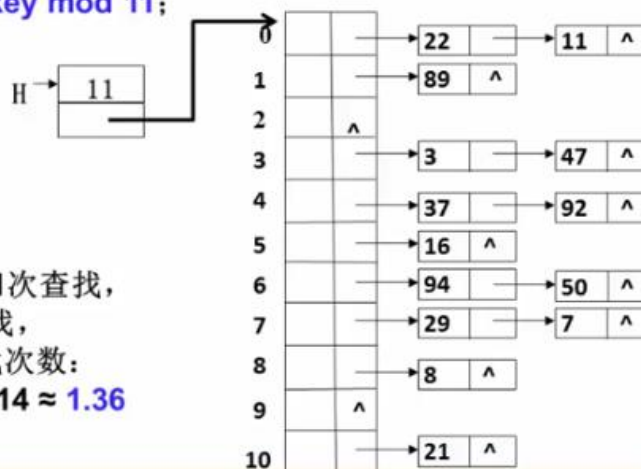
当装填因子过大时, 解决的方法是加倍扩大散列表, 这个过程叫做 “再散列 (Rehashing)”

❖ 分离链接法 (Separate Chaining)

分离链接法: 将相应位置上冲突的所有关键词存储在同一个单链表中

【例】设关键字序列为 47, 7, 29, 11, 16, 92, 22, 8, 3, 50, 37, 89, 94, 21;
散列函数取为: $h(\text{key}) = \text{key} \bmod 11$;
用分离链接法处理冲突。

```
struct HashTbl {
    int TableSize;
    List TheLists;
} *H;
```



表中有 9 个结点只需 1 次查找,
5 个结点需要 2 次查找,
查找成功的平均查找次数:
 $ASL_s = (9 + 5 \cdot 2) / 14 \approx 1.36$