算法与数据结构

华东理工大学 叶琪

五

算法与数据结构

第五章 散列查找

CONTENTS

目录

5.1 引言

❖查找方法：

顺序查找	$O(N)$
二分查找（静态查找）	$O(\log_2 N)$
二叉搜索树	$O(h)$

$O(\log_2 N)$ 已经是相当不错的时间复杂度！

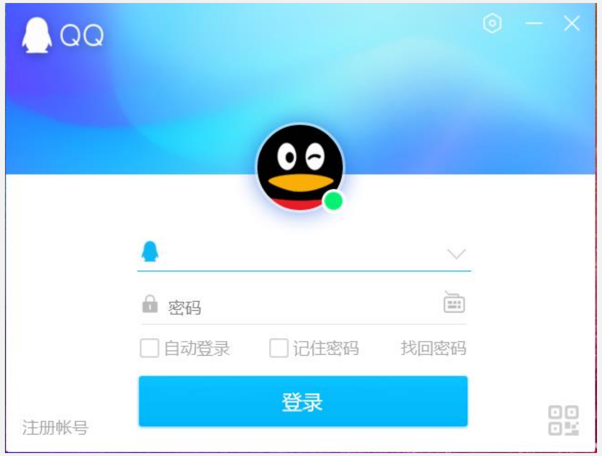
到底还有没有其他适应性广而速度又快的查找方法呢？

[例5.1] 在登录QQ的时候，QQ服务器如何**核对你的身份**，以确定你就是该号码的主人？

【分析】看看是否可以用**二分法**查找。

- 查找：十亿（$10^9 \approx 2^{30}$）有效用户，用二分查找30次。
- 存储：十亿（$10^9 \approx 2^{30}$）$\times$ 1K ≈ 1024G，1T连续空间。
- 插入/删除：按有效QQ号大小有序存储：在连续存储空间中，**插入和删除**一个新QQ号码将需要**移动大量数据**。

用不了二分查找，
该怎么办？



[例5.2] 查英文字典的过程——查询英文单词“zoo”，为什么不用二分法，而直接从字典的后面找?

- 根据要查找的关键词“zoo”在脑子里经过了“计算”，得出该关键词所在的大致位置，这样就能更快地找到它。这个“计算”过程非常类似于本章将要介绍的散列查找中的“散列函数计算”。
- 查字典的过程结合了散列查找（用于初步定位）、二分查找（一般不是准确二分）和顺序查找（当很接近关键词的时候）等几种查找方法。

【问题】如何能够在极短的时间内索到需要的关键词？

【答案】利用关键字直接映射到存储地址——散列查找法

❖期望查找的时间复杂度好于 $O(\log_2 N)$ ，

——几乎是常量：*O（1），即查找时间与问题规模无关！*

❖ 散列查找法的两项基本工作：

➢构造散列函数：确定关键词所在的存储位置的计算方法；

➢解决冲突：当多个关键词所在的存储位置相同时的解决方法。

5.2 基本概念

❖**基本思想：**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映射关系；一次存取就能得到所查元素的查找方法。

❖以数据对象的关键字*key*为自变量，通过一个确定的函数*h*，计算对应函数值*h(key)*，这个值为数据对象的存储地址，并按此存放，存储地址=*h(key)*

❖ 计算函数 *h* 称为“**散列函数**” （也称哈希函数），按这个思想构造的表称为“**散列表**”。

❖抽象数据类型描述

类型名称:符号表（SymbolTable）

数据对象集：符号表是“名字(Name)-属性(Attribute)”对的集合。

操作集：

1、SymbolTable InitializeTable(int TableSize)：创建一个长度为TableSize的符号表；

2、Boolean IsIn(SymbolTable Table, NameType Name)：判断Name是否在符号中；

3、AttributeType Find(SymbolTable Table, NameType Name)：
获取Table中**查找**指定名字Name对应的属性；

4、SymbolTable Modefy(SymbolTable Table, NameType Name, AttributeType Attr)：
将Table中指定名字Name的属性修改为Attr；

5、SymbolTable Insert(SymbolTable Table, NameType Name, AttributeType Attr)：
向Table中**插入**一个新名字Name及其属性Attr；

6、SymbolTable Delete(SymbolTable Table, NameType Name)：
从Table中**删除**一个名字Name及其属性。

❖ 查找方法：以数据对象的关键词*key*为自变量，通过一个确定的函数关系*h*，利用“**存储位置 = *h*(*key*)**”计算出地址，将*key*与该地址单元中数据对象关键字进行比较，确定查找是否成功。

❖ 可能将**不同的关键字映射到同一个散列地址**上，即$h(key_i) = h(key_j)$（当$key_i \neq key_j$），这种现象称为“**冲突(Collision)**”，key_i 和key_j称为“**同义词（synonym）**”。

❖通常关键词的**值域**（允许取值的范围）远远大于表空间的**地址集**，所以说，**冲突不可能避免，只能尽可能减少。**

[例] 有n = 11个数据对象的集合，关键词是正整数，分别为 18，23，11，20，2，7，27，30，42，15，34。如果符号表的大小用TableSize = 17（通常用一个素数），选取散列函数h如下：

$$h(key) = key \bmod TableSize$$

其中mod 是求余运算，相当于C语言中的%运算。散列表如下：

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
关键词	34	18	2	20			23	7	42		27	11		30		15	

查找：

- 查找：key = 34, 地址是34%17 = 0，地址为0中存放34，查找成功。
- 查找：key = 22, 地址是22%17 = 5，地址为空，查找失败。
- 查找：key = 40, 地址是40%17 = 6，该地址存的是23，40 ≠ 23，无法判断，还要采用解决冲突的策略才能确定。

【定义】 设散列表空间大小为m，填入表中的元素个数是n，则称$\alpha = n / m$为散列表的“装填因子（Loading Factor）”。

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
关键词	34	18	2	20			23	7	42		27	11		30		15	

- 装填因子$\alpha = 11 / 17 \approx 0.65$。
- 实用时，常将散列表大小设计使得 $\alpha = 0.5 \sim 0.85$为宜。

[例] 将给定的10个C语言中的关键词(保留字或标准函数名)顺次存入一张散列表 …… char、atan、ceil、…… 数组Table[26][2]，2 列分别……

当冲突或溢出不可避免时，如何处理使得表中没有空单元被浪费，同时插入、删除、查找等操作都能正确完成?

如何设计散列函数，使得冲突的概率尽可能小?

装填因子 α = ?　　10 / 52 = 0.19

设计散列函数 $h\,(key\,)$ = key[0]-’a’

acos define float exp char

atan ceil floor clock ctime

	槽 0	槽 1
0	acos	atan
1		
2	char	ceil
3	define	
4	exp	
5	float	floor
6		
……		
25		

如果没有溢出，算法的时间复杂度：

$T_{查询} = T_{插入} = T_{删除} = O(1)$

❖ **一个“好”的散列函数一般应考虑下列因素：**

（1）**计算简单**，以便提高转换速度；

（2）关键词对应的**地址空间分布均匀**，以尽量减少冲突；

(其原则是尽可能地使任意一组关键字的哈希地址**均匀地**分布在整个地址空间中，即用任意关键字作为哈希函数的自变量其计算结果**随机分布**，以尽可能少产生冲突)

（3）找到一种“**处理冲突**”的方法。

5.3 常见的散列函数

对数字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

若是非数字关键字(字符串)，则需先对其进行数字化处理。

①直接定址法

散列函数为关键字的线性函数

H(key) = key　**或者**　H(key) = a × key + b

地址h(key)	出生年份(key)	人数(attribute)
0	1990	1285万
1	1991	1281万
2	1992	1280万
…	……	……
10	2000	1250万
…	……	……
21	2011	1180万

H(key) = key-1990

优点：函数计算简单，分布均匀，不会产生冲突；
缺点：要求地址集合与关键词集合大小相同，不适合较大的关键词集合

②除留余数法（最常用）

构造：取关键字被某个不大于散列表表长 *TableSize* 的数 *p* 除后所得余数作散列地址，即 H(key) = key MOD *p*。

特点：简单，可与其他几种方法结合使用。

***p* 的选取很重要；*p* 选得不好，容易产生同义词。**

***p* 应为不大于 *TableSize* 的素数或不含 20 以下的质因子的合数。**

[例5.4] 有n = 11个数据对象的集合，关键词是正整数，分别为 18，23，11，20，2，7，27，30，42，15，34。

$$h(key) = key \ \% \ 17$$

地址h(key)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
关键词key	34	18	2	20			23	7	42		27	11		30		15	

- p的取值：p = Tablesize = 17；或者选择p ≤ TableSize的某个最大素数；
- 表长的取值：TableSize = n/α； n为集合的大小，α为装填因子的上限；

TableSize	8	16	32	64	128	256	512	1024
p	7	13	31	61	127	251	503	1019

为什么要对 p 加限制?

例如： 给定一组关键字：12, 39, 18, 24, 33, 21，若取 p = 9, 则他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。

③数字分析法（数字选择法）

atoi字符串转换为整数的处理函数

- 取11位手机号码key的后4位作为地址：

  散列函数为：h(key) = atoi(key+7)

- 若key集合大小$n \le 10000$, 则比较合适。若取装填因子上限$\alpha=0.8$，那么地址空间（表长）TableSize = $n/\alpha=12500$

- 若key集合大小$n \le 100$, 则可以作如下二选一：

  a、若取装填因子上限$\alpha=0.8$，那么地址空间（表长）TableSize = $n/\alpha \approx 128$，取p=127，h(key) = atoi(key+7) % p

  b、直接取后2位作为地址：

  h(key) = atoi(key+9)，而表长TableSize = $n/\alpha \approx 127$。

顺丰速递查找快递单的方法

选择6,10,14,16,17,18
参与散列计算

例如：如果关键词 key 是18位的身份证号码：

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	3	0	1	0	6	1	9	9	0	1	0	0	8	0	4	1	9
省		市		区（县）下属辖区编号		（出生）年份				月份		日期		该辖区中的序号			校验

$$h_1(key) = (key[6]-‘0’)\times 10^4 + (key[10]-‘0’)\times 10^3 + (key[14]-‘0’)\times 10^2 + (key[16]-‘0’)\times 10 + (key[17]-‘0’)$$

$h(key) = h_1(key)\times 10 + 10$ （当 $key[18] = ‘x’$时）

或 $h(key) = h_1(key)\times 10 + key[18]-‘0’$ （当 $key[18]$ 为‘0’ ~ ‘9’时）

特点：适用于关键词的位数较多，且其中部分位比较随机的情况

④平方取中法（较常用）

构造：以关键字的平方值的中间几位作为散列地址。

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。

⑤折叠法

构造：将关键字分割成位数相同的几部分，然后取这几部分的叠加和（舍去进位）做散列地址。

移位叠加：将分割后的几部分低位对齐相加。

间界叠加：从一端沿分割界来回折叠，然后对齐相加。

特点：适于关键字位数很多，且每一位上数字分布大致均匀情况。

例：关键字为：0442205864，散列地址位数为 4。

```
  5 8 6 4
  4 2 2 0
      0 4
---------
1 0 0 8 8
H(key)=0088
```

移位叠加

```
5 8 6 4
0 2 2 4
    0 4
-------
6 0 9 2
H(key)=6092
```

间界叠加

⑥随机数法

构造：取关键字的随机函数值作散列地址，即：

$$H(key) = Random(key)$$

其中，*Random* 为伪随机函数。

特点：适于关键字长度不等的情况

❖字符关键词的散列函数构造

1．一个简单的散列函数——ASCII码加和法

【例】对字符型关键词key定义散列函数如下：

$$h(key) = (\Sigma key[i]) \bmod TableSize$$

冲突严重：a3、 b2、c1；
eat、 tea；

2．简单的改进——前3个字符移位法。

【例】$h(key)=(key[0] + key[1]\times 27 + key[2]\times 27^2)\bmod TableSize$

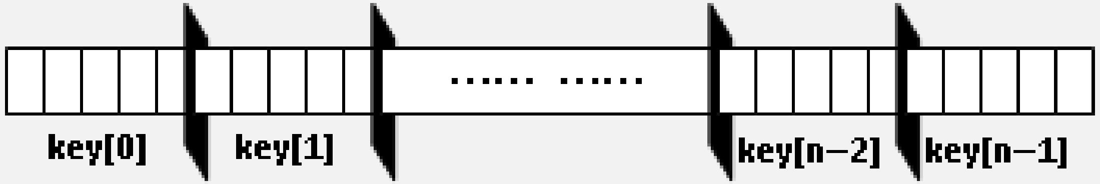
仍然冲突：string、 street、
strong、structure等等；
空间浪费：$3000/26^3 \approx 30\%$

3．好的散列函数——移位法

【例】

$$h(key) = \left(\sum_{i=0}^{n-1} key[n-i-1] \times 32^i\right) \bmod TableSize$$

每位字符占5位二进制（即$2^5 = 32$）。实现时不需要做乘法运算，只需一次左移5位来完成。




```
Index Hash ( const char *Key, int TableSize )
{
    unsigned int h = 0;      /* 散列函数值，初始化为0 */
    while ( *Key != '\0')    /* 位移映射 */
        h = ( h << 5 ) + *Key++;
    return  h % TableSize;
}
```

对32位字长的无符号整数作为h，只有7个字符起作用：$32 \le 7 \times 5$。

5.4 处理冲突的方法

常用的处理冲突的方法有两种：

开放地址法和链地址法

1.开放定址法（Open Addressing）

【定义】所谓开放定址法，就是一旦产生了冲突，即该地址已经存放了其它数据元素，就去寻找另一个空的散列地址。

- 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：$h_i(key) = (h(key)+d_i) \bmod TableSize \quad (1 \le i < TableSize)$
- d_i 决定了不同的解决冲突方案：线性探测、二次探测、双散列。

$d_i = i$

$d_i = \pm i^2$

$d_i = i*h_2(key)$

①线性探测法（Linear Probing）

- 即线性探测法以增量序列 1，2，……，（TableSize -1）循环试探下一个存储地址。

[例5.6] 设关键词序列为 {47，7，29，11，9，84，54，20，30}，

- 散列表表长TableSize =13，
- 装填因子 $\alpha = 9/13 \approx 0.69$；
- 散列函数为：h(key) = key mod 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(key)	3	7	7	0	9	7	10	9	8
冲突次数	0	0	1	0	0	3	1	3	6

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	$d_6 = 6$

“**一次聚集**”现象：很多元素在相邻的散列地址上“堆积”起来的现象，需要经过很**多次冲突**才找到空位置。

在散列表中查找数据对象的平均查找长度（ASL）：分成成功查找的ASL(ASLs)和不成功查找的ASL (ASLu) 。

【分析】ASLs：假设要查找的关键词一定在散列表中存在。ASLs是查找表中的每个关键词的比较次数之和/关键词的个数。其中每个关键词的比较次数是其冲突次数加1。

ASLs= （1+7+1+1+2+1+4+2+4）/ 9 = 23/9 ≈ 2.56

【分析】ASLu：假设要查找的关键词一定不在散列表中，并且所有关键词对应的散列地址均匀分布在地址空间上。每个地址的探测次数加和/地址空间大小就是ASLu（插入元素的ASL）：

ASLu= （3+2+1+2+1+1+1+9+8+7+6+5+4）/ 13 = 50/13 ≈ 3.85

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

②平方探测法（Quadratic Probing）

- 即**平方探测法**以增量序列1^2，-1^2，2^2，-2^2，…，q^2，$-q^2$且 $q \le \lfloor TableSize/2 \rfloor$循环试探下一个存储地址。

[例5.7] 设关键词序列为 {47，7，29，11，9，84，54，20，30}

- 散列表表长TableSize = 11（即满足4×2+3形式的素数）
- 装填因子 $\alpha = 9/11 \approx 0.82$
- 散列函数为：h(key) = key mod 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

平方探测法构建散列表的过程

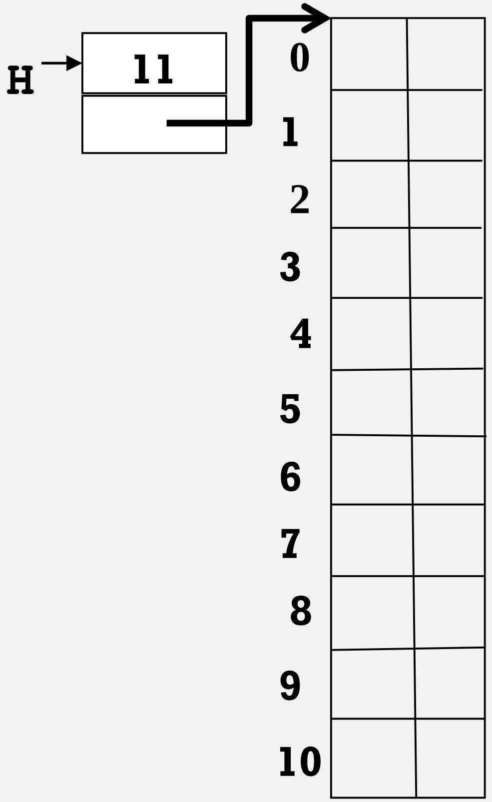
“二次聚集”现象：散列到同一地址的那些数据对象将探测相同的备选单元。

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	$d_3 = 4$

➢散列表初始化函数

```
HashTable  InitializeTable( int TableSize )
{
    HashTable H;
    int i;
    if ( TableSize < MinTableSize ){
        Error( "散列表太小" );
        return NULL;
    }
    /* 分配散列表 */
    H = malloc( sizeof( struct HashTbl ) );
     /*保证散列表最大长度是素数 */
    H->TableSize = NextPrime( TableSize );
      /*声明单元数组 */
    H->Cells = malloc( sizeof(Cell)*H->TableSize );
    /*初始化单元状态为'空单元'*/
    for( i = 0; i < H->TableSize; i++ )
        H->Cells[i].Info = Empty;
    return H;
}
```

```
typedef struct HashTbl *HashTable;
struct HashTbl{
    int TableSize;
    Cell *Cells;
}H ;
```



➢平方法探测法的查找函数

```
Position  Find( ElementType Key, HashTable H )
 {   Position CurrentPos, NewPos;
     int CNum;          /*   记录冲突次数   */
     CNum = 0;
     NewPos = CurrentPos = Hash( Key, H->TableSize ); //计算存储位置
     while( H->Cells[NewPos].Info!=Empty &&
         H->Cells[NewPos].Element != Key )
        {
         /*平方探测法*/
         if(++CNum%2){ /*判断冲突的奇偶次  */
           NewPos = CurrentPos + (CNum+1)*(CNum+1)/4;
           while( NewPos >= H->TableSize ) /*地址越界，调整为合法地址*/
             NewPos -= H->TableSize;
         } else {
             NewPos = CurrentPos - CNum * CNum/4;
             while( NewPos < 0 )          /*地址越界，调整为合法地址*/
               NewPos += H->TableSize;
         }
     }
   return NewPos;
}
```

➢平方法探测法的插入函数

```
void  Insert( ElementType Key, HashTable H )
{   /* 插入操作 */
    Position Pos;
    Pos = Find(Key,H);
    if( H->Cells[Pos].Info != Legitimate ) {
            /* 确认在此插入 */
         H->Cells[Pos].Info = Legitimate;
         H->Cells[Pos].Element = Key;
           /*字符串类型的关键词需要 strcpy 函数!! */
            }
        }
```

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

- d_i 选为$i*h_2(key)$，其中$h_2(key)$是另一个散列函数。我们把它叫做双散列探测法。由此，探测序列成了：$h_2(key)$，$2h_2(key)$，$3h_2(key)$，…
- 要求：对任意的key，$h_2(key) \neq 0$，且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：
  - $h_2(key) = p - (key \bmod p)$

其中：$p < TableSize$，p、TableSize都是素数。

④再散列（Rehashing）

- 开放地址法的装填因子 α会严重影响查找效率，由于表长在一定时间内是定值，α与“填入表中的元素个数”成正比。
- 当装填因子过大时，解决的方法是加倍扩大散列表，这样α可以减小一半，这个过程叫做“再散列（Rehashing）”。当然，装填因子过小时（比如 $\alpha<0.3$），会浪费空间，此时散列表大小可以减半。
- 再散列具有偶然性，在交互系统中会有“停顿”现象。

2.分离链接法（Separate Chaining）

【定义】分离链接法是解决冲突的另一种方法，其做法是将所有关键词为同义词的数据对象通过结点链接存储在同一个单链表中。

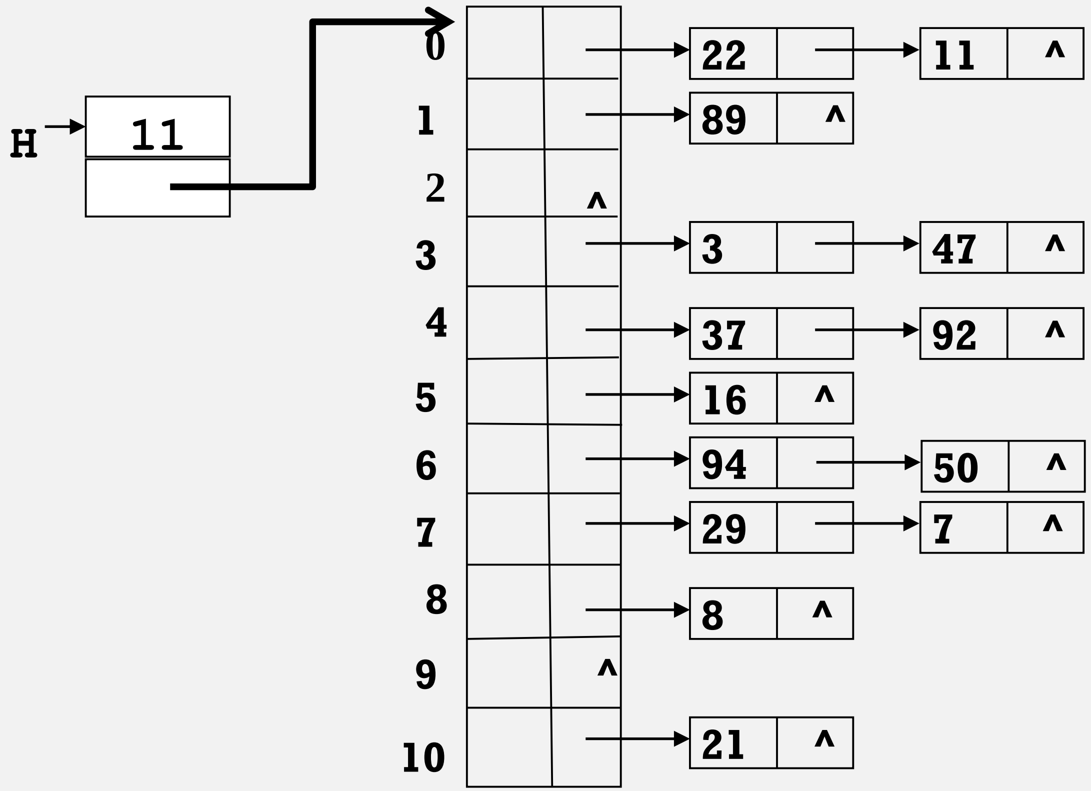
【例】 设关键字序列为 47, 7, 29, 11, 16, 92, 22, 8, 3, 50, 37, 89, 94, 21;

- 散列函数取为：h(key) = key mod 11；
- 用分离链接法处理冲突。

2.分离链接法（Separate Chaining）

➢结构定义：

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



➢该表中有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为：

ASL s=（9+5*2）/ 14 ≈ 1.36。

ASLu估算比较复杂。

```c
struct ListNode;
typedef struct ListNode *Position, *List;
struct HashTbl;
typedef struct HashTbl *HashTable;
struct ListNode
{
    ElementType Element;
    Position    Next;
};
```

```c
Position  Find( ElementType Key, HashTable H )
{
     Position P;
     List L;
     L = &( H->TheLists[Hash(Key, H->TableSize )]);
                                           //哈希函数找到头结点
     P = L->Next;
     while( P != NULL && strcmp(P->Element, Key) )
          P = P->Next;
     return P;
}
```

5.5 散列表的性能分析

- 平均查找长度（ASL）用来度量散列表查找效率。
- 关键词的比较次数，取决于产生冲突的多少。
- 影响产生冲突多少有以下三个因素：

（1）散列函数是否均匀；

（2）处理冲突的方法；

（3）散列表的装填因子α。

- 主要分析后两者对查找效率的影响。

下图表示了期望探测次数与装填因子α的关系。

合理的的最大装入因子α应该**不超过0.85**。

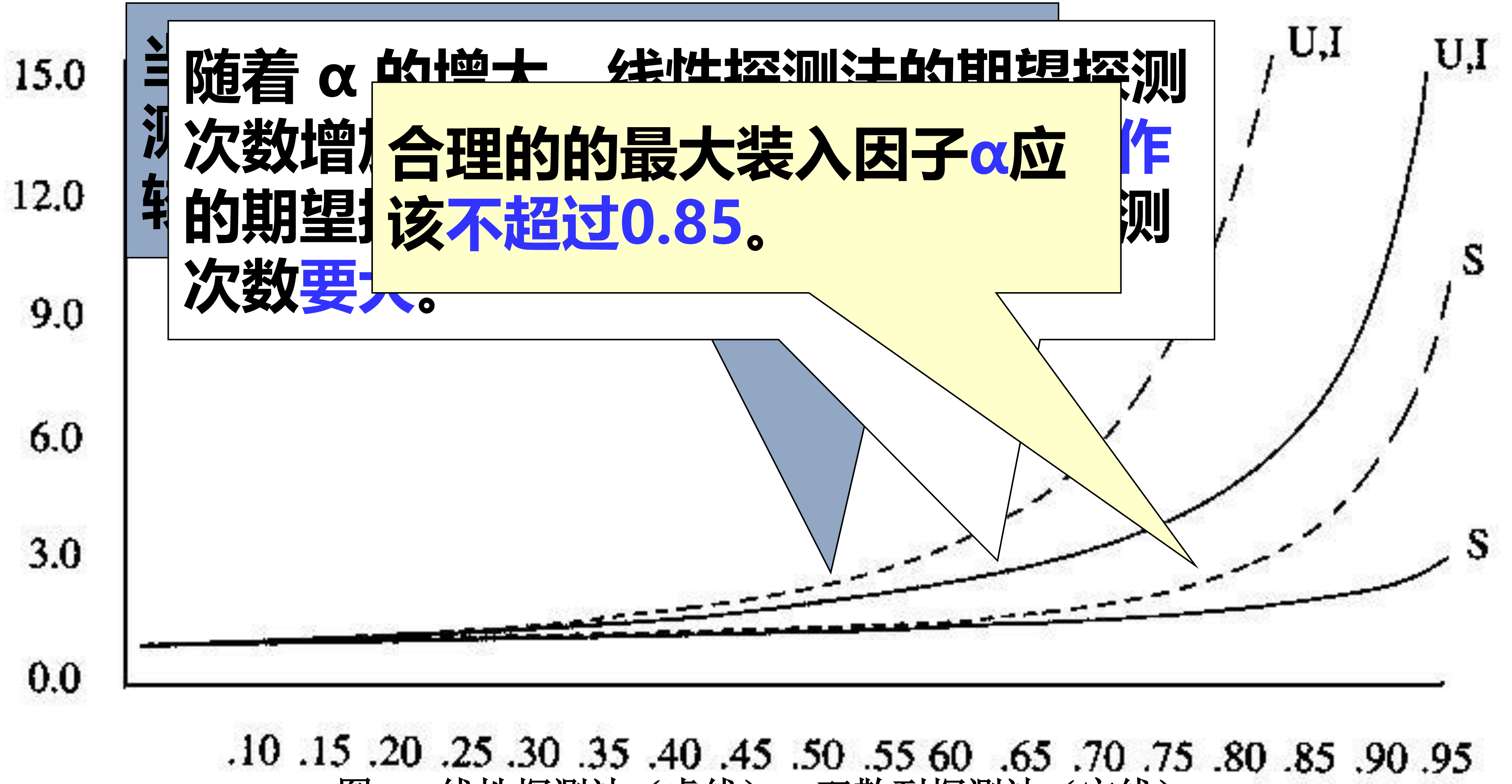


图5.7 线性探测法（虚线）、双散列探测法（实线）
U表示不成功查找，I表示插入，S表示成功查找