Hash

吉林大学计算机学院
谷方明
fmgu2002@sina.com

直接寻址法的启示

- 迄今讨论的查找方法都要对表作搜索。
- 直接寻址法
  - 例：2 4 6 9
  - 直接寻址：数组保存，**位置对应关键字**
  - 效率：O(1)
  - 失效：当关键字的范围过大、实际的关键字数目较少时。
  - 散列方法是直接寻址法的一种有效替代。

散列思想

- 例：5个数1000000，2000000，3000000，4000000，5000000
  - ✓ h(K) = K % 7
  - ✓ 查找：O(1)
  - ✓ 插入：O(1)
  - ✓ 删除：O(1)

0	
1	1000000
2	2000000
3	3000000
4	4000000
5	5000000
6	

- 散列思想：按关键字编址，即给出关键字K，直接计算存储地址 。

散列方法

- 散列方法是按关键字编址的一项技术。以关键字 ***K*** 为自变量, 通过函数 ***h*(*K*)** 计算地址。 ***h*(*K*)**称为散列函数。
- 散列方法不仅是一种快速的查找方法，也是一种重要的存储方式。
- 按散列方式构造的存储结构被称为散列表，散列表中的一个位置也被称为槽。

散列的含义

- 设散列表长度为M，散列函数 *h* 的值域为{0, 1, 2, …, M–1}。
- *h* 通常要把变化范围很大并且其中又有些关键词*K*十分靠近的数据元素尽可能地“混杂搅乱”，使它们的*h* (*K*) 值在区间 [0, M–1] 中尽可能地“散开”。（hash、杂凑、哈希）

散列方法的核心

- 散列函数： ***h*(*K*)**

- 冲突： **K1 ≠ K2，有h(K1)=h(K2).**
  - 没有冲突的散列函数 h 是很不好找的(教材：31个常用单词映射到41个整数值)；对于动态数据困难更大。
  - 幸运的是，我们能找到有效的方法解决冲突。

- 散列方法核心：散列函数和冲突消解方法

散列函数的设计

- 散列函数是<u>均匀的</u>：设 *K* 是从关键词集合中随机选取的一个关键词，则 *h*(*K*) 以同等概率取区间[0, *M*–1]中的每一个值，并与其它关键字已散列到哪个槽位无关。
- 通常应使 h 与组成 K 的所有符号有关

- 散列函数的分类
  - ✓ 数字和字符串
  - ✓ 按使用用的主要运算或方法

除法散列函数

- $h(K)=K \bmod M$.

- 除数的大小选择得当；
- 除数的性质与 h (K) 的值在给定区域中的分布情况和冲突产生的数量均密切相关。
- 一般取略大于元素个数的素数；
- 除法散列函数是一种简单、常用的构造散列的方法，并且不要求事先知道关键词的分布；

乘法散列函数

- $h(K) = \lfloor M(K\theta \bmod 1) \rfloor$　　实数θ, $0<\theta<1$
- θ值接近 0 或 1将导致散列地址集中在表的末端
- $\theta \approx 0.618$ 实验效果较好

压缩法

- 把关键词的二进制串分割成若干个子串，然后按某种方式把这些子串合并形成该关键词的地址。

- 例：关键词是英文单词，则可将单词的每个字母对应的二进制串看成是分割得到的子串，然后用某种运算，譬如异或运算进行合并.
- 但异或满足交换律，**a XOR b=b XOR a**，相同字母组成的不同单词具有相同的地址，**h1(STEAL) = h1(STALE)=h1(TALES)=h1(LEAST).**

散列函数的例子

- 在表8.4（325页）中示出了三个散列函数，其中:
- h_1为字母的异或;
- h_2是除法散列函数, $h_2(WORD) = WORD \bmod 31$;
- h_3 是乘法散列函数, $h_3(WORD)=1+\lfloor 30\times[(\theta\times WORD)\bmod 1]\rfloor$其中，$\theta=0.6125423371$，$M = 30$.
- 将表8.4中 30 个英文单词的每个字母都表成一个 5 位二进制数，如将A , B , C , … , Z 分别表成 00001, 00010 , 00011 , … , 11010 . 故每个单词是一个二进制串,

WORD		h1(WORD)		h2(WORD)		h3(WORD)	
THE	BE	25	7	2	7	23	8
OF	AT	9	21	21	21	21	26
AND	BY	11	27	19	27	4	16
TO	I	27	9	4	9	7	16
A	THIS	1	6	1	25	19	23
IN	HAD	7	13	23	13	30	30
THAT	NOT	9	21	18	18	17	21
IS	ARE	26	22	28	24	2	4
WAS	BUT	5	3	12	12	26	11
HE	FROM	13	22	13	21	27	3
FOR	OR	27	29	8	2	16	2
IT	HAVE	29	26	29	5	20	26
WITH	AN	2	15	29	15	7	6
AS	THEY	18	0	20	27	8	1
HIS		18		5		30	
ON		1		29		18	

平方取中法

- 取 K^2 的中间部分作为$h(K)$的值；
- 虽然只取了K×K 的中间几位，但这些位与乘数 K 的每一位都相关，故散列值还是比较均匀的。
- 取中一般通过位运算来实现

关键词值的内码	内码的平方	内码的平方取 ω 位 二进制取18位 八进制取6位	散列函数值 二进制右移9位 八进制右移3位
0100	0010000	010000	010
1100	1210000	210000	210
1200	1440000	440000	440

关键词 $1100_8=576_{10}$

关键词平方 $1100_8\times1100_8= 1210000_8$

从右往左取6位 210000_8

右移 3 位 210_8

散列函数最大取值 $777_8=7\times8^2+7\times8^1+7\times8^0=511< M$

- 通常用在事先不知道关键词的分布且位数不是很大的情况。有些编译器采用。

- 中间部分的长度或位数取决于M 的大小

抽取法

- 从与关键词对应的二进制串中抽取几个分散的代码，然后合并这几个代码而形成一个地址.
- 例：将**A , B , C , … , Z** 分别表成 **00001, 00010 , 00011 , … , 11010 .** 故每个单词是一个二进制串

单词	二进制串	取第三位和最后两位
THE	101000100000101	$(101)_2=5$
OF	0111100110	$(110)_2=6$
AND	000010111000100	$(000)_2=0$
TO	1010001111	$(111)_2=7$
A	00001	$(001)_2=1$
IN	0100101110	$(010)_2=2$
THAT	10100010000000110100	$(100)_2=4$
IS	0100110011	$(011)_2=3$

- 这种方法容易出现群集，出现这种现象的原因, 是因为散列函数值仅依赖二进制串的部分代码, 而不是依赖整个二进制串.

冲突消解(Collision Resolution)

- 冲突消解, 也称“溢出”处理技术，是一个重要问题。

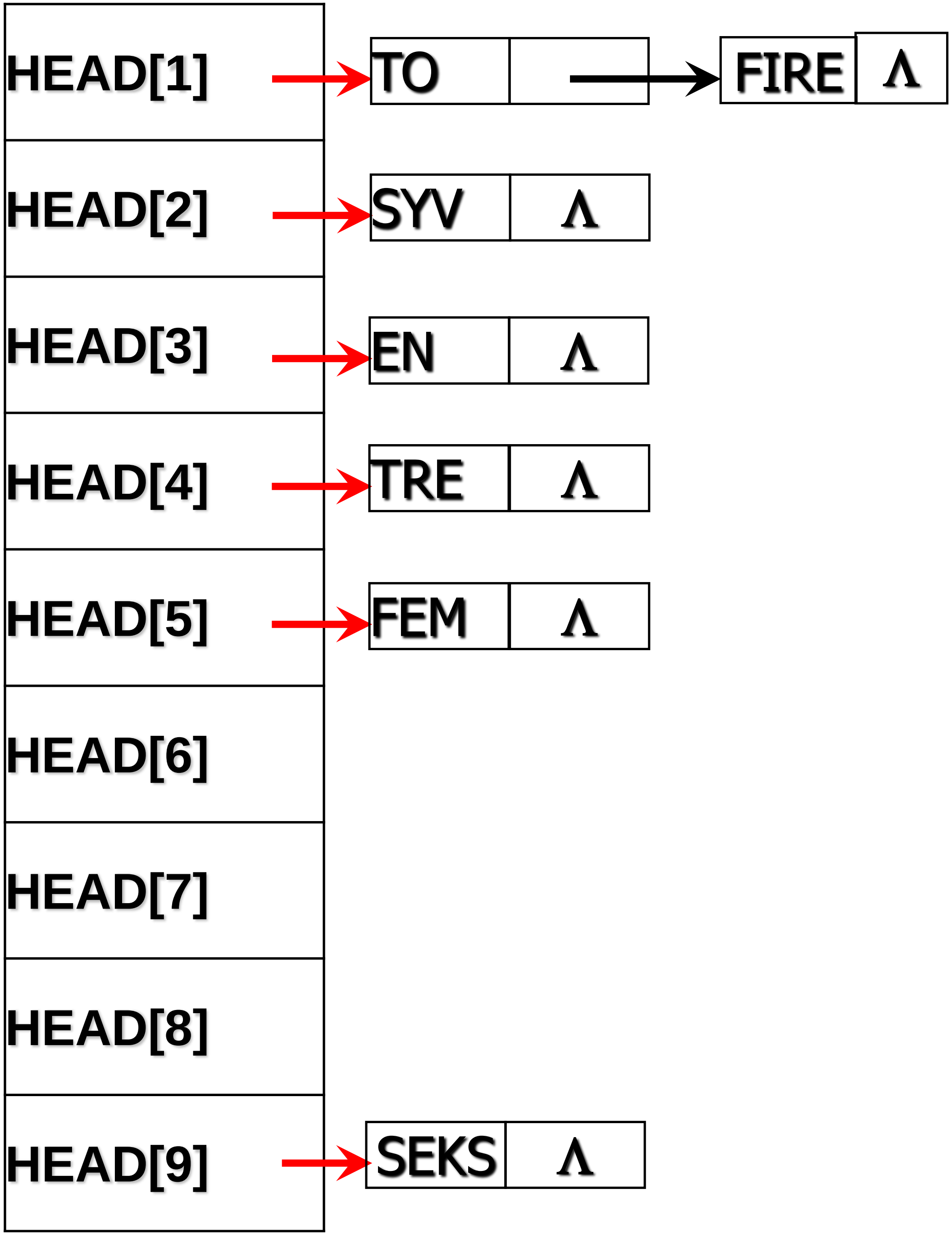
- 常用的两类冲突消解方法
  - ✓ 拉链方法（chaining）
  - ✓ 开地址法（open addressing）

拉链法

- 对 h 值域 $[0, M-1]$ 中的每个值保持一个链表。

- 每个链表中存放一组关键词互相冲突的记录，该组关键词有 $h(K_1)=h(K_2)=\ldots=h(K_t)$.

- 每个链表LIST[i]有一个表头HEAD[i] , $i=h(K)+1$. 同一个链表中的记录按某种次序链接在一起.



K	h(K)
EN	2
TO	0
TRE	3
FIRE	0
FEM	4
SEKS	8
SYV	1

拉链法平均查找长度

- $ASL_{succ} = 8/7$

- $ASL_{unsucc} = 7/9$（与参考教材一致）

拉链法的实现

- **插入**
  - 在链表T[h(x.key)]的头部插入x；
  - 时间复杂度O(1)
- **查找**
  - 在链表T[h(x.key)]查找x是否出现；
  - 最坏时间复杂度O(n);
- **删除**
  - 从链表T[h(x.key)]删除x；
  - 时间复杂度O(1)，假设已知x的位置；使用双向链表删除方便

拉链法分析

- 装载因子(load factor): 给定一个能存放N个元素、具有M个槽位的散列表T，定义N/M为T的装载因子。通常记为*λ* 或 *α* .
- *λ*可以小于、等于或大于1.
- 使用拉链法，装载因子就是一个链平均长度。即对于j = 0,1,…, M-1，链表T[j]的长度用n_j表示，于是有
  - ✓ $N = n_0 + n_1 + \ldots + n_m$
  - ✓ $E(n_j) = \lambda$

定理1：

- 对于均匀散列和拉链法解决冲突的散列表，一次不成功查找的平均时间为**O(1+ λ)**
- 证明：

均匀散列假设下，一个不在表中的关键字**k**等可能的散列到任意一个槽中。因此，当查找一个关键字不成功的情况下，查找的期望时间就是查找到链表**T[h[k]]**尾部的时间。这一时间的期望就是链表**T[h[k]]**的长度的期望，即$E(n_j) = \lambda$。包括计算**h**的时间，一次不成功查找的平均时间为**O(1+ λ)**。

定理2

- 对于均匀散列和拉链法解决冲突的散列表，一次成功查找的平均时间为O(1+ λ)
- 证明：

假定要查找的元素是表中的N个元素中任何一个，且是等可能的。在对元素x的一次成功查找中，所检查的元素数目就是x所在的链表中x前面的元素数多1。由于采用头插法，所以x之前的元素都是在x之后插入的。因此所检查元素的期望数目，就是x所在链表、在x之后插入到该链表中的期望元素数目加1，再对表中的N个元素取平均。

设x_i表示插入散列表中的第i个元素，$k_i = x_i.key$。

定义指示器随机变量$X_{ij} = I\{h(k_i)=h(k_j)\}$。

均匀散列假设下，有$P\{h(k_i)=h(k_j)\} = 1/M$，从而 $E[X_{ij}] = 1/M$。一次成功查找检查元素的期望数目：

$$E\left[\frac{1}{N}\sum_{i=1}^{N}\left(1+\sum_{j=i+1}^{N}X_{ij}\right)\right]$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(1+\sum_{j=i+1}^{N}E[X_{ij}]\right)$$

$= 1+(N-1)/2M = 1+ \lambda/2 + \lambda/2N = O(1+\lambda)$

拉链法小结

- **期望时间复杂度：O(1+λ)**
  - 若散列表的槽数与元素数成正比，则λ=O(1)，从而查找操作平均时间为常数。
  - 采用双向链表，删除操作最坏时间复杂度为O(1)
  - 采用头插法插入的最坏时间复杂度为O(1)
- **空间复杂度：*M*个表头和N个链接，共N+M个指针。**
- **为保证速度，M应很大；M太大，浪费空间。**

拉链法的修改（合并拉链表）

- 散列表中结点用**TABLE [*i*]** 表示,包含关键词域***K*****EY[*i*]**，链接域**LIN*****K*****[*i*]**等, **0 ≤ *i* ≤ *M*** ．它们有**:** 空的或已占用两种状态**.**

- 算法**C**检索***M***个结点的表**.**
- 若***K***不在表中且表不满，则插入***K*****.**
- 算法**C**下标从**1**开始，因此从**h[k]+1**开始查找。如果下标从**0**开始，从**h[k]**开始查找

TABLE[1]	TO	
TABLE[2]	SYV	Λ
TABLE[3]	EN	Λ
TABLE[4]	TRE	Λ
TABLE[5]	FEM	Λ
TABLE[6]		
TABLE[7]		
TABLE[8]	SEKS	Λ
TABLE[9]	FIRE	

算法C应用于关键词的序列:

EN,TO,TRE,FIRE,FEM,

SEKS,SYV;

假定散列函数 h 作用于以上 7 个关键词依序分别有值:

2, 0, 3, 0, 4, 8, 1;

故对$h(K)+1$我们又有:

3, 1, 4, 1, 5, 9, 2;

并假定 9 个链表初始皆空

算法C允许几个链表相结合，所以记录被插入表中后不需要移动它们.

开地址法

- 不建立链表，也称空缺编址法。
- 插入关键词值为*K*的新元素的方法是：从地址*h*(*K*)开始, 按照某种次序探查插入新元素的空位置。其被检查的位置序列称为探查序列。
  - ✓ 线性探查
  - ✓ 伪随机探查
  - ✓ 二次探查
  - ✓ 双散列

线性探查法

- 线性探查法是一种最简单的开地址法.

- 使用如下循环探查序列:

$$h(K),\ h(K)+1, \cdots, M-2, M-1, 0, \cdots, h(K)-1$$

TABLE[0]	TO
TABLE[1]	FIRE
TABLE[2]	EN
TABLE[3]	TRE
TABLE[4]	FEM
TABLE[5]	SYV
TABLE[6]	
TABLE[7]	
TABLE[8]	SEKS

算法L应用于关键词的序列: EN,TO,TRE,FIRE,FEM, SEKS,SYV;

假定散列函数 *h* 作用于以上 7 个关键词依序分别有值:

2, 0, 3, 0, 4, 8, 1;

线性探查法分析

- 时间效率

$$S(\lambda) \approx 0.5\left(1 + 1/(1-\lambda)\right)$$
$$U(\lambda) \approx 0.5\left(1 + 1/(1-\lambda)^2\right)$$

- 优点：简单
- 缺点：基本聚集（容易使许多元素在散列表中连成一片，从而使探查的次数增加，影响查找效率）
- 经验：**M**一般取**N**的**5**倍

线性探查法：参考实现

```
int hash(int x){ return x % P; }

void makenull() { for(int i=0;i<P;i++) h[i]=EMP; }

int loc(int x) {
    int ori=hash(x);
    int i=0;
    while( i < P && h[(ori+i)%P] != x &&
    h[(ori+i)%P] != EMP)  i++;

    return (ori+i)%P;
}
```

```
void insert(int x) {
    int pos=loc(x);
    h[pos]=x;
}

int find(int x){
    int pos=loc(x);
    return h[pos]==x;
}
```

线性探查法的删除

- 不能简单地将一个元素清除，这会隔离探查序列后面的元素，从而影响后面元素的查找过程。
- 处理策略
  - ✓ 每个元素增设一个标识位 ；表中的项有三类：空的、已占用的以及已删除的(**Deleted**)；仅当删除是非常稀少时才是可行的
  - ✓ 真删：教材上的算法R。仅用于线性探查法。

算法R (TABLE[ ]，M ， i . TABLE[ ])

/* TABLE用线性探查法构造，本算法删除TABLE[i] */

R1. [清空i] 置TABLE[i] 为空，$j \leftarrow i$.

R2. [i增值] 置$i \leftarrow (i + 1) \bmod M$.

R3. [检查TABLE[i]] 如果TABLE[i] 为空，则算法结束. 否则置$r \leftarrow h(\mathrm{KEY}(i))$，这个关键词原来的散列地址现在存在位置$i$中. 如果$j < r \le i$或如果$i< j <r$或$r \le i< j$（换言之，如果r循环地位于$i$和$j$之间），则转回到R2 .

R4. [移动一个记录] 置TABLE [j] $\leftarrow$ TABLE [i] 并返回步骤R1 ▌

伪随机探查法

- 伪随机探查的基本思想是：建立一个伪随机数生成器，当发生冲突时，就利用伪随机数生成器计算出下一个探查的位置. 有很多伪随机数生成器，现介绍较简单的一种。其计算公式为：

$$y_0=h(key) \qquad y_{i+1}=(y_i+p) \bmod M \ (\ i=0,\ 1,\ 2,\ \ldots\ )$$

- 式中，**y0**为伪随机数生成器的初值，***M*** 为散列表的长度，***p*** 为与 ***M*** 接近的素数。

二次探查法

消除基本聚集的另一种方法是二次探查法。二次探查法使用下列探查序列： $h(\text{key}), h_1(\text{key}), h_2(\text{key}), \ldots, h_{2i-1}(\text{key}), h_{2i}(\text{key}), \ldots$

其中，$h_{2i-1}(\text{key})$ 和$h_{2i}(\text{key})$的计算公式如下：

$$h_{2i-1}(key) = (h(key) + i^2) \bmod M,$$
$$i = 1, 2, 3, \ldots, (\mathrm{M} - 1)/2$$
$$h_{2i}(key) = (h(key) - i^2) \bmod M,$$
$$i = 1, 2, 3, \ldots, (\mathrm{M} - 1)/2$$

其中M（表的大小）应该是一个4k+3形式的素数，如503、1019等

双重杂凑法（再散列法）

- 使用两个散列函数$h_1(K)$ 和$h_2(K)$ 对表进行探查。最好的探测方法之一

- 探测序列：$\mathbf{h}(K, \mathbf{i}) = (h_1(K) + \mathbf{i} * h_2(K)) \bmod M$
  - ✓ h_1 的值域为$0 \le h_1(K) < M$；
  - ✓ h_2 的值必须是一个与M 互质的、属于区间 $[1, M–1]$ 的整数
  - ✓ 若M是质数，则$h_2(K)$可以是1和$M–1$之间的任何值；
  - ✓ 若$M = 2^m$，则$h_2(K)$可取区间 [1，$2^m–1$]中的任何一个奇数值

散列小结

- 散列方法多用于数据的快速插入和查找。