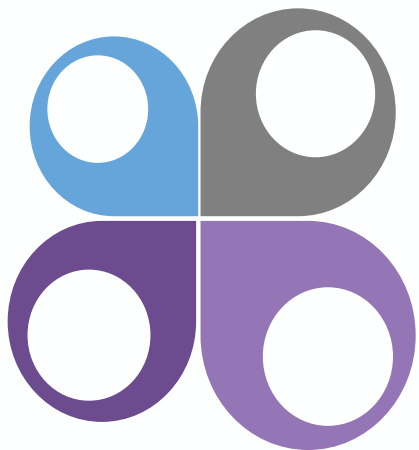


# 第8章 查找

柳银萍

ypliu@cs.ecnu.edu.cn



一、顺序查找（线性查找）



二、折半查找（二分或对分查找）



三、分块查找

# ▶▶▶ 折半查找的性能分析



## 查找过程

每次将待查记录所在区间缩小一半，比顺序查找效率高,时间复杂度  $O(\log_2 n)$



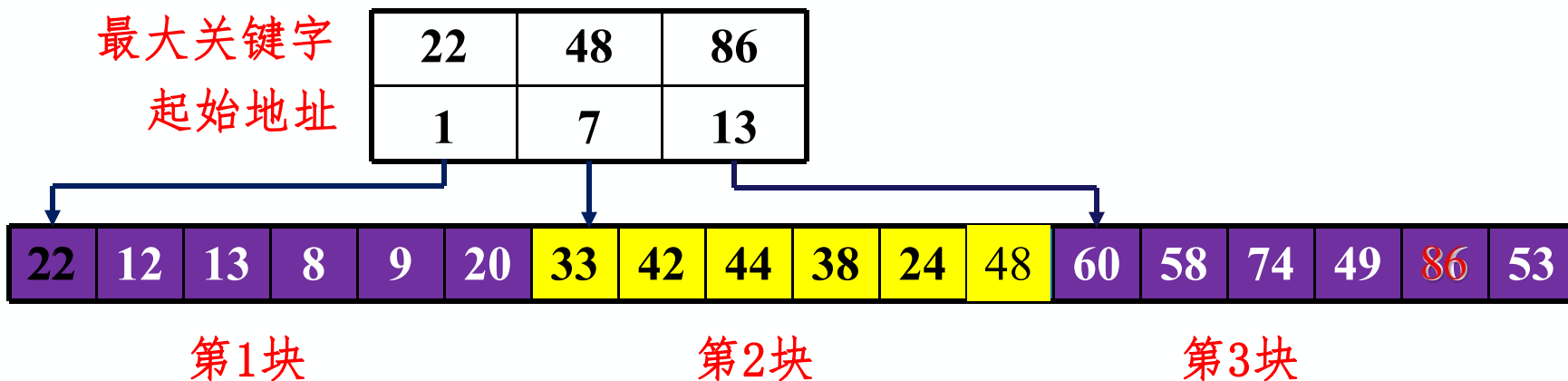
## 适用条件

采用顺序存储结构的有序表，不宜用于链式结构

## 分块查找（块间有序，块内无序）

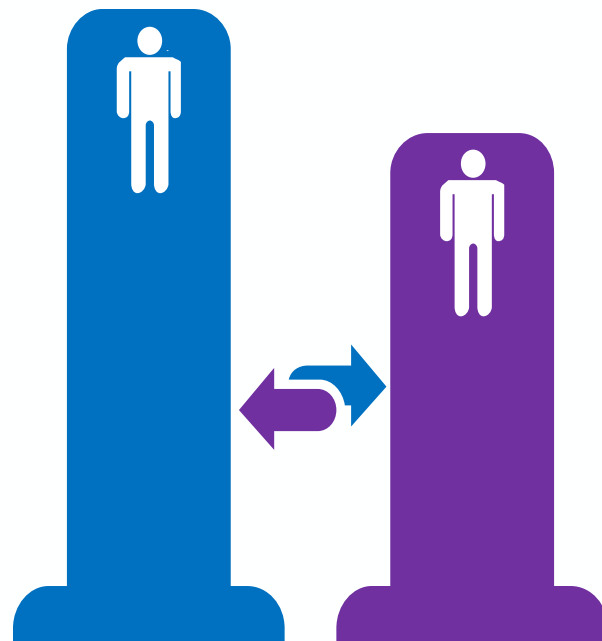
- ◆ **分块有序**，即分成若干子表，要求每个子表中的数值都比后一块中数值小（但子表内部未必有序）。
- ◆ 然后将各子表中的最大关键字构成一个**索引表**，表中还要包含每个子表的起始地址（即头指针）。

索引表



## 分块查找过程

- 对索引表使用折半查找法（因为索引表是有序表）；
- 确定了待查关键字所在的子表后，在子表内采用顺序查找（因为各子表内部是无序表）；



## 分块查找性能分析

查找效率：  $ASL = L_b + L_s$

对索引表查找的ASL

对块内查找的ASL

$$ASL_{bs} \cong \log_2 \left( \frac{n}{s} + 1 \right) + \frac{s}{2} \quad \left( \log_2 n \leq ASL_{bs} \leq \frac{n+1}{2} \right)$$

s为每块内的记录个数，n/s块的数目

例：当n=9，s=3时， $ASL_{bs}=3.5$ ，而折半法为3.1，顺序法为5。

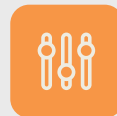
## 分块查找优缺点



**优点：**插入和删除比较容易，无需进行大量移动。



**缺点：**要增加一个索引表的存储空间并对初始索引表进行排序运算。



**适用情况：**如果线性表既要快速查找又经常动态变化，则可采用分块查找。

## ▶▶▶ 树表的查找

表结构在查找过程中动态生成

对于给定值key

若表中存在，则成功返回；

否则插入关键字等于key的记录



二叉排序树

平衡二叉树

B-树

B<sup>+</sup>树

键树



二叉排序树或是空树，或是满足如下性质的二叉树：



若其左子树非空，则左子树上所有结点的值均小于根结点的值；



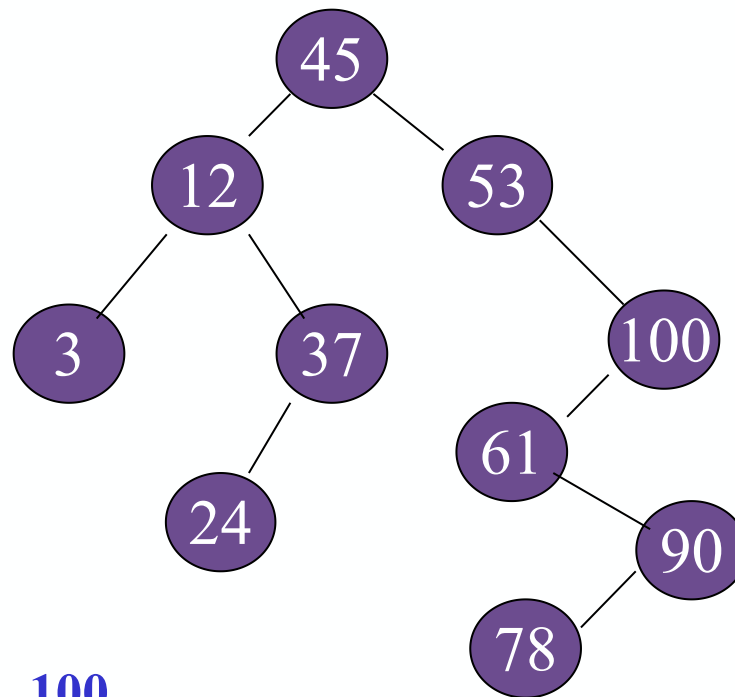
若其右子树非空，则右子树上所有结点的值均大于等于根结点的值；



其左右子树本身又各是一棵二叉排序树。

## 练习

中序遍历二叉排序树后的结果有什么规律？



3, 12, 24, 37, 45, 53, 61, 78, 90, 100

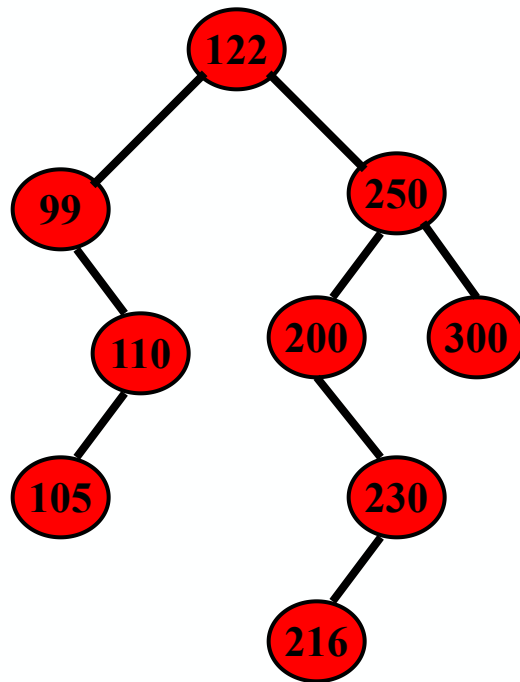
递增

得到一个关键字的递增有序序列

## ▶▶▶ 二叉排序树的操作—查找

- 若查找的关键字 **等于** 根结点，**成功**
- 否则
  - ✓ 若 **小于** 根结点，查其 **左子树**
  - ✓ 若 **大于** 根结点，查其 **右子树**
- 在左右子树上的操作类似；

查找效率：树的高度



## ►►► 查找的性能分析

**问题：**如何提高二叉排序树的查找效率？

尽量让二叉树的形状均衡




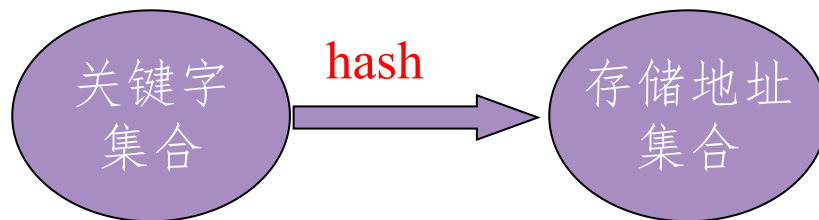
平衡二叉树

- 左、右子树是平衡二叉树；
- 所有结点的左、右子树深度之差的绝对值 $\leq 1$

**平衡因子：**该结点左子树与右子树的高度差

## ▶▶▶ 哈希表的查找

- 基本思想：记录的存储位置与关键字之间存在对应关系，
- $Loc(i)=H(key_i)$   哈希函数



- 优点：查找速度极快 **$O(1)$** ，查找效率与元素个数 **$n$** 无关。

## ▶▶▶ 哈希函数例

数据元素序列(14, 23, 39, 9, 25, 11), 若规定每个元素 $k$ 的存储地址 $H(k)=k$ , 请画出存储结构图。

地址	...	9	...	11	...	1	...	2	24	25	...	39	...
内容		9		11		14		23		25		39	

根据哈希函数 $H(k)=k$ , 查找 $key=9$ , 则访问 $H(9)=9$ 号地址, 若内容为9则成功;

若查不到, 则返回一个特殊值, 如空指针或空记录。

## ▶▶▶ 有关术语

哈希表(杂凑表): 按上述思想构造的表

地址	...	9	...	11	...	14	...	23	24	25	...	39	...
内容		9		11		1		23		25		39	

冲突: 不同的关键码映射到同一个哈希地址

$\text{key1} \neq \text{key2}$ , 但  $H(\text{key1}) = H(\text{key2})$

同义词: 具有相同函数值的两个关键字

## 冲突举例

(14, 23, 39, 9, 25, 11)

哈希函数:  $H(k) = k \bmod 7$

0	1	2	3	4	5	6
14		23		39		

6个元素用7个  
地址应该足够!

$H(14) = 14 \% 7 = 0$

9  
25  
11

有冲突

$H(25) = 25 \% 7 = 4$   
 $H(11) = 11 \% 7 = 4$   
同义词



## 如何减少冲突

冲突是不可能避免的



构造好的哈希函数



制定一个好的解决冲突方案

## ▶▶▶ 哈希函数的构造方法

- 根据元素集合的特性构造
- 地址空间尽量小
- 均匀



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

## ▶▶▶ 直接定址法

$$\text{Hash}(\text{key}) = a \cdot \text{key} + b \quad (a、b \text{ 为常数})$$

优点：

以关键码`key`的某个线性函数值为哈希地址，不会产生冲突。

缺点：

要占用连续地址空间，空间效率低。

## ▶▶▶ 直接定址法

例： {100, 300, 500, 700, 800, 900},

哈希函数  $\text{Hash}(\text{key}) = \text{key} / 100$

0	1	2	3	4	5	6	7	8	9
	100		300		500		700	800	900

# 除留余数法

最常用重点掌握!

$\text{Hash}(\text{key}) = \text{key} \bmod p$  ( $p$ 是一个整数)

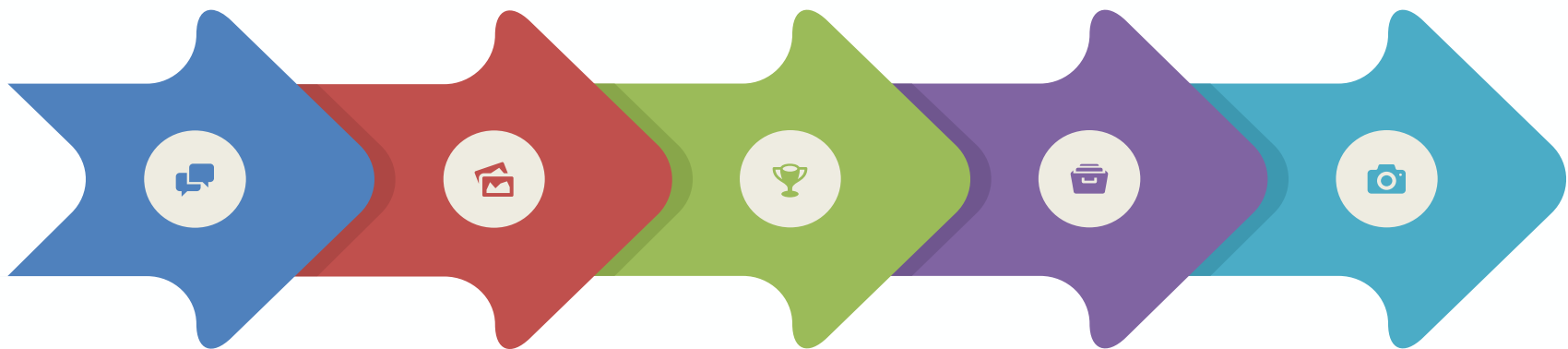
关键:

如何选取合适的 $p$ ?

技巧:

设表长为 $m$ , 取 $p \leq m$   
且为质数

## 构造哈希函数考虑的因素



① 执行速度  
(即计算哈希  
函数所需时间)

② 关键字  
的长度

③ 哈希  
表的大小

④ 关键字  
的分布情  
况

⑤ 查  
找频率

## ▶▶▶ 处理冲突的方法



1.开放定址法



2.链地址法

## ▶▶▶ 开放定址法

**基本思想：**有冲突时就去寻找下一个空的哈希地址，只要哈希表足够大，空的哈希地址总能找到，并将数据元素存入。



线性探测法



二次探测法



伪随机探测法



## 线性探测法

$$H_i = (\text{Hash}(\text{key}) + d_i) \bmod m \quad (1 \leq i < m)$$

其中：m为哈希表长度

$d_i$  为增量序列 1, 2, ..., m-1, 且  $d_i = i$

一旦冲突，就找下一个空地址存入

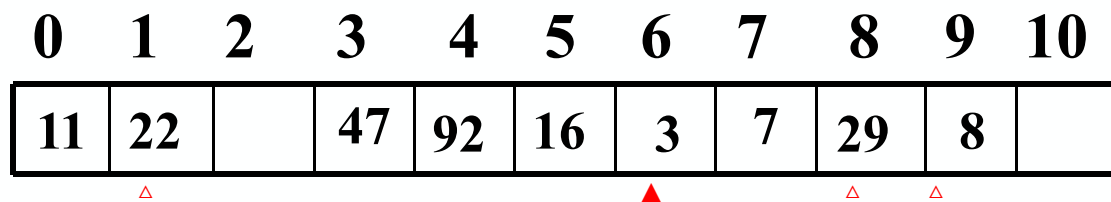
## 线性探测法

关键码集为 {47, 7, 29, 11, 16, 92, 22, 8, 3},

设：哈希表表长为  $m=11$ ;

哈希函数为  $\text{Hash}(\text{key}) = \text{key} \bmod 11$

0	1	2	3	4	5	6	7	8	9	10
11	22		47	92	16	3	7	29	8	





① 47、7、11、16、92没有冲突



②  $\text{Hash}(29)=7$ ，有冲突，由  $H_1=(\text{Hash}(29)+1) \bmod 11=8$ ，哈希地址8为空，因此将29存入



③ 3 连续移动了3次

## 线性探测法的特点

### 优点:

只要哈希表未被填满，保证能找到一个空地址单元存放有冲突的元素。

### 缺点:

可能使第 $i$ 个哈希地址的同义词存入第 $i+1$ 个地址，这样本应存入第 $i+1$ 个哈希地址的元素变成了第 $i+2$ 个哈希地址的同义词，...，产生“聚集”现象，降低查找效率。

解决方案：二次探测法

## 平方探测法

关键码集为  $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$ ,

**设： 哈希函数为  $\text{Hash}(\text{key}) = \text{key} \bmod 11$**

$$H_i = (\text{Hash}(\text{key}) \pm d_i) \bmod m$$

其中： $m$ 为哈希表长度， $m$ 要求是某个 $4k+3$ 的质数；

$d_i$ 为增量序列  $1^2, -1^2, 2^2, -2^2, \dots, q^2$

0	1	2	3	4	5	6	7	8	9	10
11	22	3	47	92	16		7	29	8	

△
▲
△
△

Hash(3)=3, 哈希地址冲突, 由 $H_1=(\text{Hash}(3)+1^2) \bmod 11=4$ , 仍然冲突;  $H_2=(\text{Hash}(3)-1^2) \bmod 11=2$ , 找到空哈希地址存入。

## 伪随机探测法

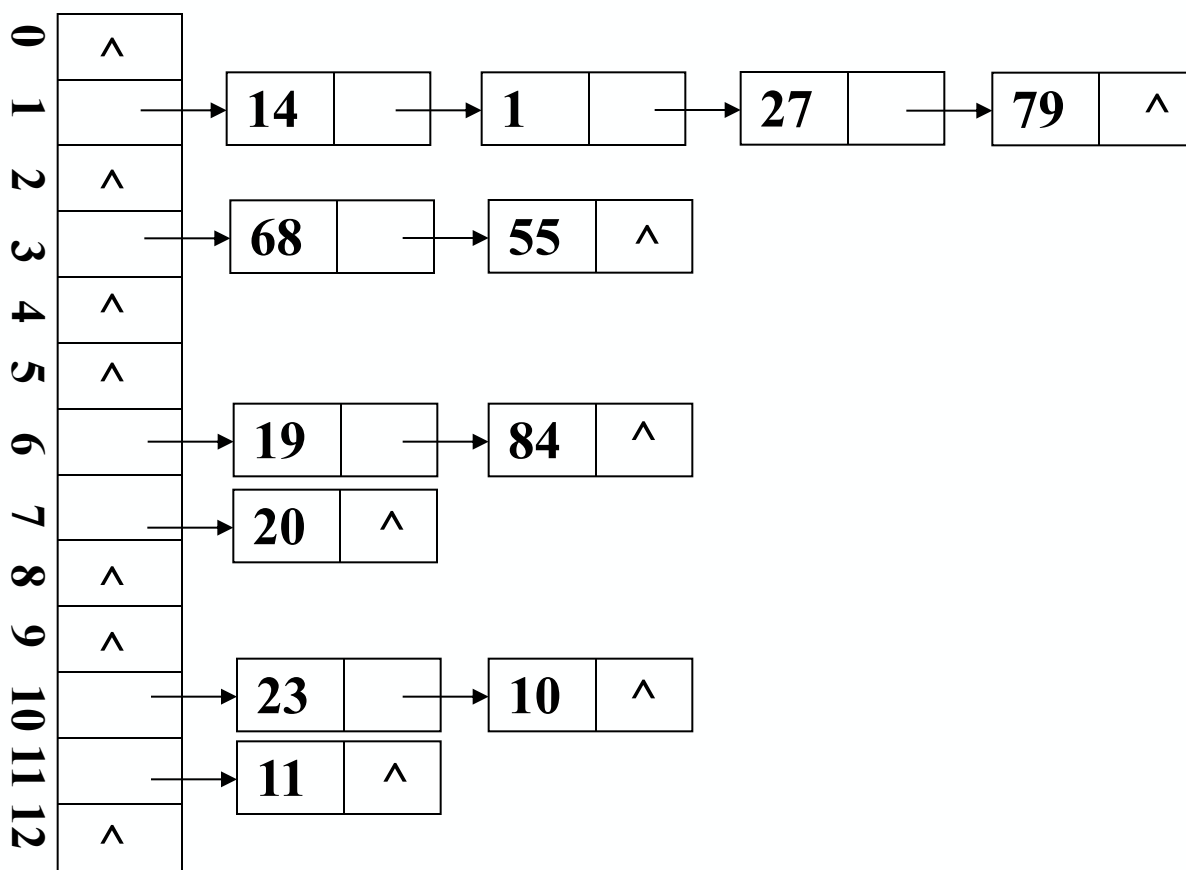
$$H_i = (\text{Hash}(\text{key}) + d_i) \bmod m \quad (1 \leq i < m)$$

其中：  $m$  为哈希表长度

$d_i$  为随机数

## 链地址法

**基本思想：**相同哈希地址的记录链成一单链表，**m个哈希地址就设m个单链表**，然后用一个数组将m个单链表的表头指针存储起来，形成一个动态的结构



## ▶▶▶ 链地址法的优点



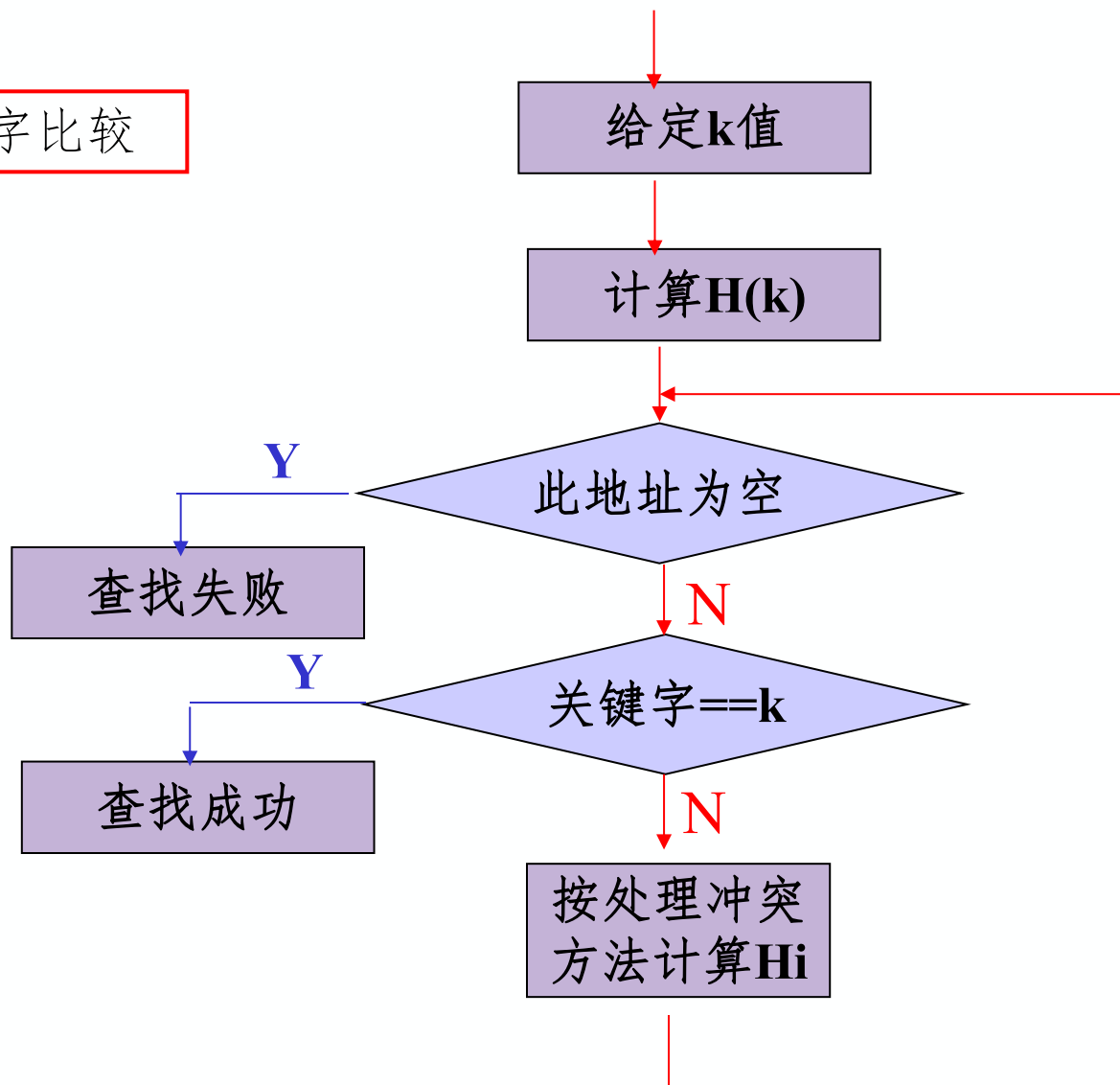
非同义词不会冲突，无“聚集”现象



链表上结点空间动态申请，更适合于表长不确定的情况

## 哈希表的查找

给定值与关键字比较





## ►►► 哈希表的查找

$$ASL=(1*6+2+3*3+4+9)/12=2.5$$

已知一组关键字(19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79)

哈希函数为：  $H(\text{key})=\text{key} \bmod 13$ ，哈希表长为  $m=16$ ，  
设每个记录的查找概率相等

(1) 用线性探测再散列处理冲突，即  $H_i=(H(\text{key})+d_i) \bmod m$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	14	1	68	27	55	19	20	84	79	23	11	10			
	1	2	1	4	3	1	1	3	9	1	1	3			

$$H(19)=6$$

$$H(14)=1 \quad H(23)=10$$

$$H(1)=1 \quad \text{冲突, } H_1=(1+1) \bmod 16=2$$

$$H(68)=3 \quad H(20)=7$$

$$H(27)=1 \quad \text{冲突, } H_1=(1+1) \bmod 16=2$$

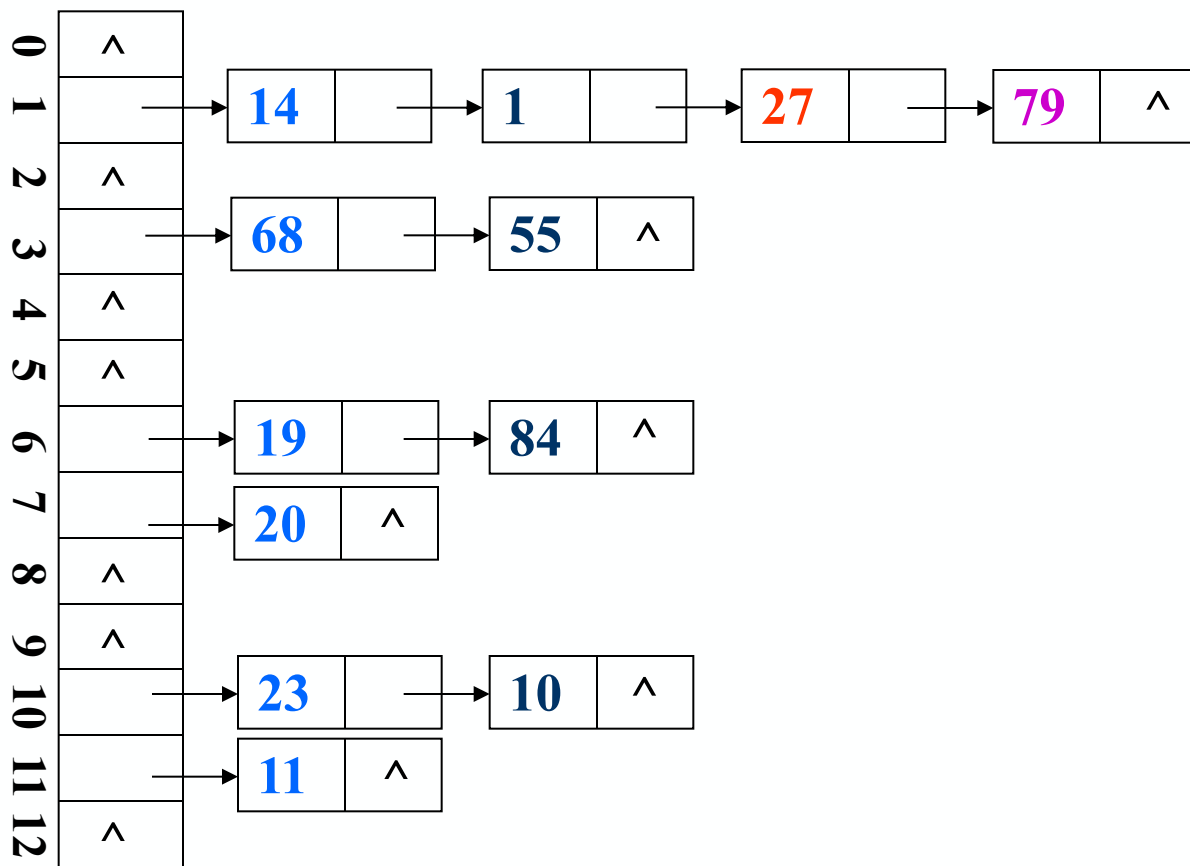
$$\text{冲突, } H_2=(1+2) \bmod 16=3$$

$$\text{冲突, } H_3=(1+3) \bmod 16=4$$

## ►►► 哈希表的查找效率

$$ASL=(1*6+2*4+3+4)/12=1.75$$

(2) 用链地址法处理冲突      关键字(19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79)



## ►►► 哈希表的查找效率分析

使用平均查找长度ASL来衡量查找算法，ASL取决于

- ✓ 哈希函数
- ✓ 处理冲突的方法
- ✓ 哈希表的装填因子

$$\alpha = \frac{\text{表中填入的记录数}}{\text{哈希表的长度}}$$

$\alpha$  越大，表中记录数越多，说明表装得越满，发生冲突的可能性就越大，查找时比较次数就越多。