

数据结构与算法

查找

计算机学院

朱晨阳 副教授

zhuchenyang07@nudt.edu.cn



查找

- 基本概念
- 顺序查找
- 折半查找
- 分块查找
- 散列查找



查找

- 查找是在给定的数据结构中搜索满足条件的结点。
- 查找也称为检索。
- 衡量一个查找算法好坏的依据主要是查找过程中需要执行的平均比较次数，或称为**平均查找长度**。
- 假设：结点是等长的，查找是**基于关键码的**，**关键码都是整数**。
- 线性表的查找（第6章），树的查找（第7/8章），图的查找（第9章）



顺序查找

- 逐个将每个结点的关键码和待查的关键码值进行比较，直到找出相等的结点或者找遍了所有的结点。
- 对存储方式和排序是否有要求？



顺序查找

- 逐个将每个结点的关键码和待查的关键码值进行比较，直到找出相等的结点或者找遍了所有的结点。
- 执行顺序查找算法时，被查找的线性表可以是顺序存储或链接存储，对结点没有排序要求。



顺序查找

- 复杂性
 - 最好情况下，第一个结点就是要查找的结点，时间复杂性为 $O(1)$;
 - 最坏情况下，最后一个结点是要查找的结点，或者表中没有符合条件的结点，时间复杂性为 $O(n)$;
 - 一般每个结点都有相同的查找概率，此时顺序查找的平均长度为 $n/2$,时间复杂性为 $O(n)$ 。



查找

- 如果是排序的？
- 这个已知条件有什么用？



折半查找

- 折半查找，首先找到表的中间结点，将其关键码与给定的要查找的值进行比较，若相等，则查找成功；若大于要查找的值，则继续在表的前半部分折半查找，否则继续在表的后半部分进行折半查找。
- 对存储和排序是否有要求？



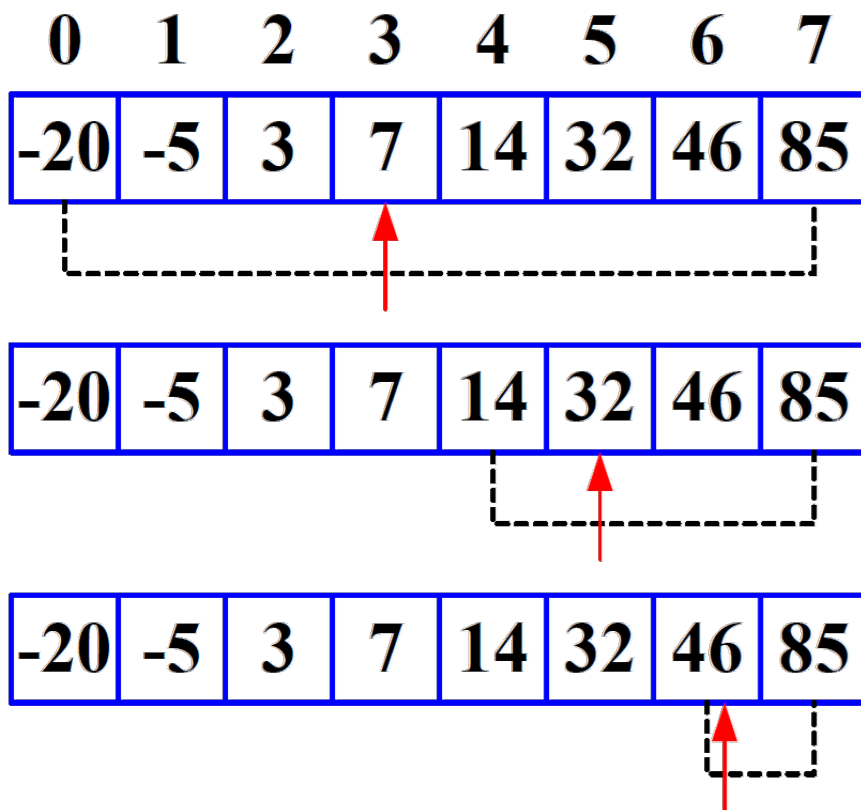
折半查找

- 折半查找，首先找到表的中间结点，将其关键码与给定的要查找的值进行比较，若相等，则查找成功；若大于要查找的值，则继续在表的前半部分折半查找，否则继续在表的后半部分进行折半查找。
- 顺序存储且结点排序



折半查找

- 例，8个结点组成的线性表，其关键码依次为-20，-5，3，7，14，32，46，85中，采用折半查找算法查找关键码为46的结点。





折半查找

- 0 1 2 3 4 5 6 7
- -20 -5 3 7 14 32 46 85
- 在 $A[0:7]$ 中查找 46 $low=0, high=7, m=3$
- $46 > A[3]$
- 在 $A[4:7]$ 中查找 46 $low=4, high=7, m=5$
- $46 > A[5]$
- 在 $A[6:7]$ 中查找 46 $low=6, high=7, m=6$
- $46 = A[6]$, 成功



折半查找

- 0 1 2 3 4 5 6 7
- -20 -5 3 7 14 32 46 85
- 在 $A[0:7]$ 中查找 15 $\text{low}=0, \text{high}=7, \text{m}=3$
- $15 > A[3] = 7$
- 在 $A[4:7]$ 中查找 15 $\text{low}=4, \text{high}=7, \text{m}=5$
- $15 < A[5] = 32$
- 在 $A[4:4]$ 中查找 15 $\text{low}=4, \text{high}=4, \text{m}=4$
- $15 > A[4]$
- 在 $A[5:4]$ 中查找 $\text{low}=5, \text{high}=4$
- 因为 $5 > 4$, 因此失败



折半查找

● 复杂性分析

- ◆ 最好情况，只需比较一次就找到对应结点，时间复杂度为
- ◆ 最坏情况，找不到对应结点，需要 $\log_2 n$ 次比较，时间复杂度为
- ◆ 平均时间复杂度为



对比

查找方法	查找速度	排序	适应结点动态变化
顺序查找			
折半查找			



对比

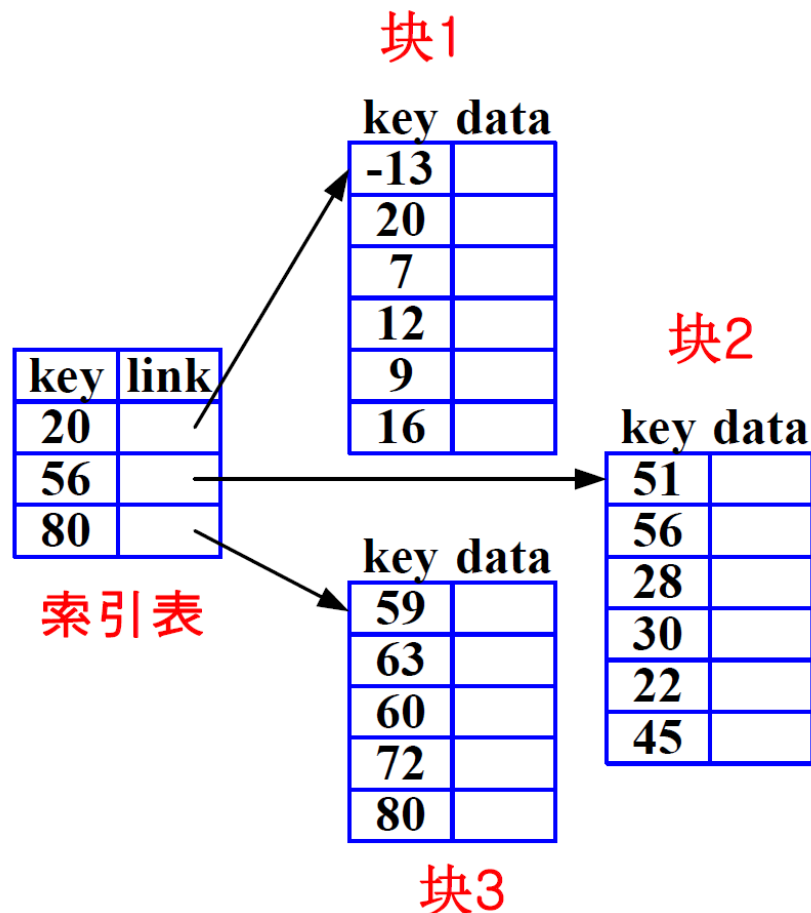
查找方法	查找速度	排序	适应结点动态变化
顺序查找	慢	不需要	好
折半查找	快	全部结点排序	差

如何各取优点？



分块查找

- 例：设有一个线性表包含17个结点，将其分为3块，每块的结点数分别为6、6、5，各块采用顺序存储，分别放在三个连续的内存空间中，索引表包含三个索引项，每个索引项包含它对应的块的最大关键码值，和该块的起始地址。





分块查找

- 如果既要有较快的查找速度，又要满足元素动态变化的要求，可以采用分块查找算法。
- 分块查找将一个大的线性表划分成若干块(如何分块？)，块内不排序，块之间排序(假设非递减)。建立一个索引表，把每块中的最大关键码值作为索引表的关键码值，且非递减排序。
- 查找某结点时，先在索引表中顺序查找或者折半查找，找到该结点对应的块，然后在块内顺序查找。



分块查找

- 复杂性
- 分块查找的平均查找长度(比较次数)由对索引表的平均查找长度和对块的平均查找长度组成。
 - 设线性表有 n 个结点，等分成 b 块，每块有 $s=n/b$ 个结点。假设对索引表和块都采用顺序查找，假定对每个结点的查找概率相同。则平均查找长度为 $b/2 + s/2 = (n/s + s)/2$ ，当 $s^2=n$ 时取得最小值 $O(n^{0.5})$ ——指导分块查找算法如何分块。



分块查找

查找方法	查找速度	排序	适应结点动态变化
顺序查找	慢	不需要	好
折半查找	快	全部结点排序	差
分块查找	中	块之间排序	中

- 分块查找中，当增加或减少结点以及结点的关键码改变时，只需要调整结点所在的块即可。
- 当结点变化频繁，导致块与块之间结点数相差很大时，查找效率会下降！
- 如何应对？对块的存储方式有何考虑？



查找

- 有没有可能有更快的查找算法?
- 比如 $O(1)$?



散列查找

- 什么是散列函数？什么是散列表？
- 什么是冲突？什么是同义词？
- 什么是负载因子？设计散列函数要考虑什么？
- 常用的散列函数有哪些？
- 怎么处理冲突？
- 什么是开放地址法？什么是链表地址法？



散列查找

- 是否存在时间复杂度为 $O(1)$ 的查找算法?
- 假设一组结点（例如10个结点），最小最大关键码分别为0和79999，建立一个长度为80000的数组，每个结点存放在关键码对应的数组位置。
- 空间闲置，内存空间不够(不可行)。
- 用长度 n 为10的数组来存放。关键码为 k 的
- 结点存放在 $k\%10$ 号数组单元。
 - $k \rightarrow k\%10$



散列查找

- 散列函数(哈希函数), 将分散在一个大区间(例如 $[0, 79999]$)的密钥值映射到一个较小的区间(例如 $[0, 9]$), 用映射后的值做为访问结点的下标。
例如

```
int Hash(int key)
{   return key%10; }
```



散列查找

- 与散列函数相关联的是一个长度为 n 的表，称为散列表或哈希表(Hash Table)，用来存放结点的数据或数据的引用。散列函数将关键码值映射到 $[0, n-1]$ 范围内的整数值。
- 会出现什么问题？



散列查找

- 散列函数经常是多对一的，导致**冲突(碰撞)**，具有相同散列值的关键码值称为同义词。
- 两个结点不能占据同一个位置，需要一种冲突解决策略。为了讨论冲突及其解决办法，引入**负载因子 α** 的概念。
- $\alpha = \text{填入表中的结点数} / \text{散列表长度}$
- **设计散列函数函数应考虑什么？**



散列查找

- 设计散列函数需考虑：应该能够**有效减少冲突**；必须具有**很高的执行效率**。
- 因为任何不是整数的关键码都可以转换为整数，所以下面的讨论都假定关键码是整数。



散列查找

- 除留余数法：利用余数运算将整数型的关键码值映射到 $0 \sim n-1$ 的范围内。选择一个适当的正整数 p ，用 p 去除关键码值，所得余数作为该关键码的散列值。
- 除留余数法的关键是 p 的选取。
 - ✓一般取 p 为小于等于 n 的最大素数
 - ✓为什么？



散列查找

- 除留余数法：利用余数运算将整数型的关键码值映射到 $0 \sim n-1$ 的范围内。选择一个适当的正整数 p ，用 p 去除关键码值，所得余数作为该关键码的散列值。
- 除留余数法的关键是 p 的选取。
 - ✓ 一般取 p 为小于等于 n 的最大素数
 - ✓ 对于素数 p , $0 < k < p$, 任意 $0 \leq i < j < p$, $k*i$ 与 $k*j$
 - ✓ 不同余, 即不碰撞; 例如 $p=11$, $k=10$, $i=3$, $j=9$
 - ✓ 对于非素数, $k*i$ 与 $k*j$ 可能同余; 例如 $p=12$



散列查找

- 数字分析法：当关键码的位数很多时，可以通过对关键码的各位进行分析，**丢掉分布不均匀的位，留下分布均匀的位**作为散列值。
- 例：key hash(key) 取分布均匀的3位
 - 100**3**19426 **326**
 - 000**7**18309 **709**
 - 000**6**29443 **643**
 - 100**7**58615 **715**
 - 000**9**19697 **997**
 - 000**3**10329 **329**
- 只适合**静态的关键码**值集合。



散列查找

- 平方取中法：计算关键码值的平方，从平方的中间位置取连续若干位，将这些位构成的数作为散列值。

● 例，hash(key)取平方的中间两位

key	key ²	hash(key)
319426	102032969476	29
718309	515967819481	78
629443	396198480249	84
758615	575496718225	67



散列查找

- 随机乘法法：使用一个随机实数 $f(0 \leq f < 1)$ ， $f * \text{key}$ 的小数部分与散列表长度 n 相乘，将乘积的整数部分作为 key 的散列值。

● 例，设随机数 $f=0.103149002$ ，散列表长 $n=101$ 。

key	$f * \text{key}$	$(f * \text{key})$ 的小数部分 $* n$	hash(key)
319426	32948.47311	47.78411	47
718309	74092.85648	86.50448	86
629443	64926.41727	42.14427	42



散列查找

- 折叠法：将关键码值分成若干段，其中至少有一段的长度等于散列表长度值的位数，把这些多段数相加，并舍弃可能产生的进位，所得整数作为散列值。
- 关键码值的位数比散列表长度值的位数多出很多时，可以采用折叠法。
- 例，关键码key=852422241，散列表长度为4000。

85|2422|241

8	5		
	2	4	1
2	4	2	2
<hr/>			

1 1 1 6 3

852|4222|41

8	5	2	
	4	1	
4	2	2	2
<hr/>			

5 1 1 5



散列查找

- 当冲突发生时怎么办?
 1. 将引起冲突的新数据项存放在表中另外的位置。——**开放地址法**
 2. 为每个散列值单独建立一个表以存放具有相同散列值的所有数据项。——**链表地址法**



散列查找

- 开放地址法：散列表的每个表项有一个表示该表项是否被占用的标志，当试图加入新的数据项到散列表中时，首先判断散列值指定的表项是否被占用，**如果被占用，则依据一定的规则在表中寻找其它空闲的表项。**
- 探测空闲表项的最简单的方法是线性探测法：当冲突发生时，顺序地探测下一个表项是否空闲。若 $\text{Hash}(\text{key})=d$ ，而第 d 表项已被占用，则依次探测 $d+1, d+2, \dots, n-1, 0, 1, \dots, d-1$ 。**（这会导致一个新问题！）**



散列查找

- 线性探测法解决冲突可能产生堆积。使用散列函数计算出散列值后，散列表的对应表项可能已经被非同义词的结点占用。
- 如何改善堆积？



散列查找

- 可采用双散列函数探测法改善堆积现象：使用2个散列函数Hash1和Hash2，其中Hash1以键码值为自变量，产生一个 $0 \sim n-1$ 之间的数。Hash1用来产生基本的散列值，当发生冲突时，利用Hash2计算探测序列。当 $\text{Hash1}(\text{key}) = d$ 时发生冲突，则再计算 $k = \text{Hash2}(\text{key})$ ，得到探测序列为 $(d+k)\%n$, $(d+2*k)\%n$, $(d+3*k)\%n$, ...



散列查找

- 双散列函数可以使探测序列跳跃地分散到整个存储区域里，从而有助于减少“堆积”的产生
- 不能随便删除散列表中的表项目，因为删除一个表项可能使得同义词序列断开



散列查找

- 链表地址法：为散列表的每个表项建立一个单链表，用于链接同义词子表，每个表项需增加一个指针域。
 - **独立链表地址法**：在散列表的基本存储区域外开辟一个新的区域用于存储同义词子表。
 - **公共链表地址法**：将同义词子表存储在散列表所在基本存储区域里。例如，在基本存储区域探测空闲表项，找到后将其链接到同义词子表中。



散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度
为11，散列函数为除留
余数法
 $h(key)=key\%11$

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10	54	



散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1		
2		
3		
4		
5		
6		
7		
8		
9		
10	54	



散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1		
2		
3		
4		
5		
6	94	
7		
8		
9		
10	54	



散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3		
4		
5		
6	94	
7		
8		
9		
10	54	



散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3	14	
4		
5		
6	94	
7		
8		
9		
10	54	



散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
$$h(\text{key}) = \text{key} \% 11$$

0	77	
1	89	→ 45
2		
3	14	
4		
5		
6	94	
7		
8		
9		
10	54	



散列查找

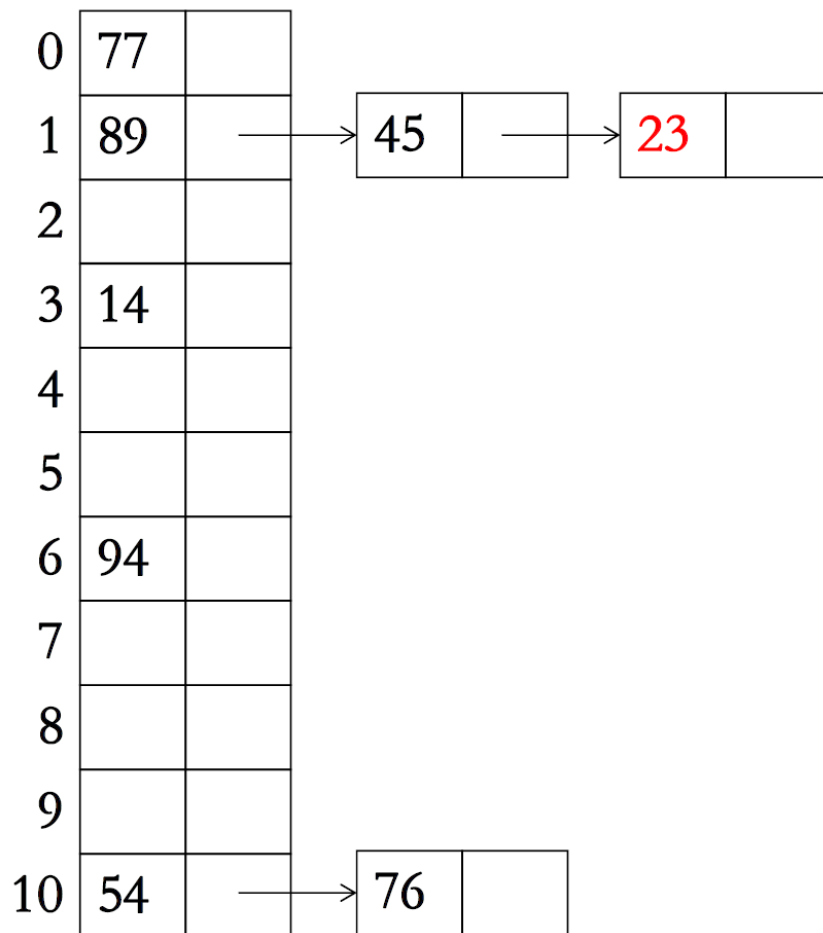
- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77		
1	89	→	45
2			
3	14		
4			
5			
6	94		
7			
8			
9			
10	54	→	76



散列查找

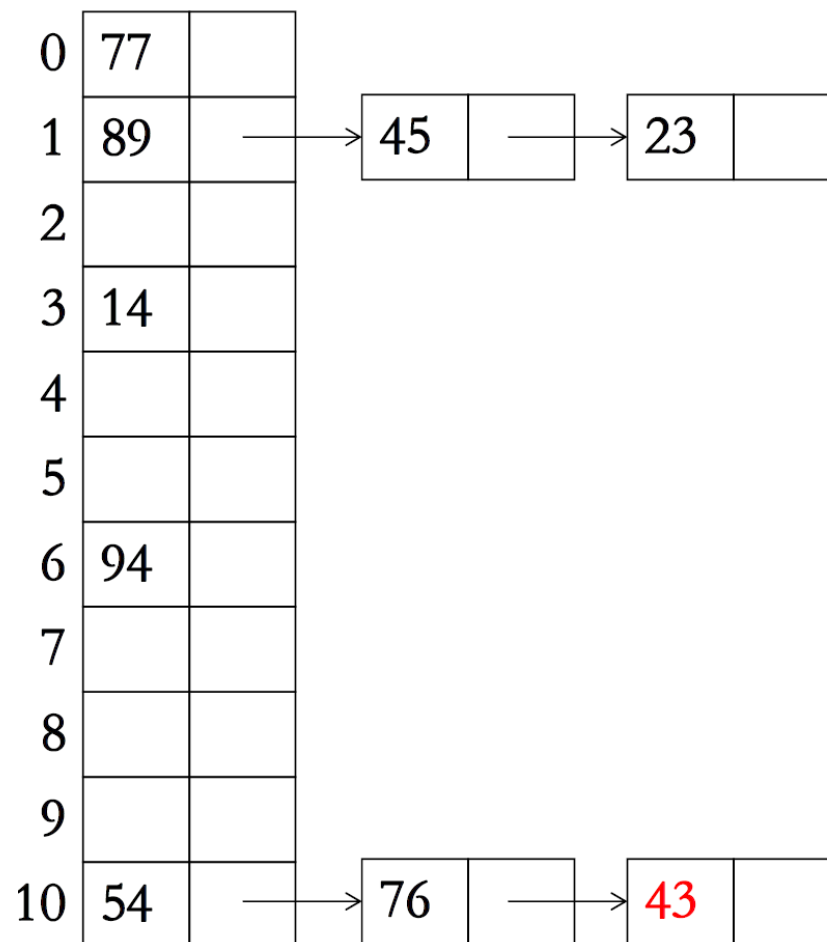
- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$





散列查找

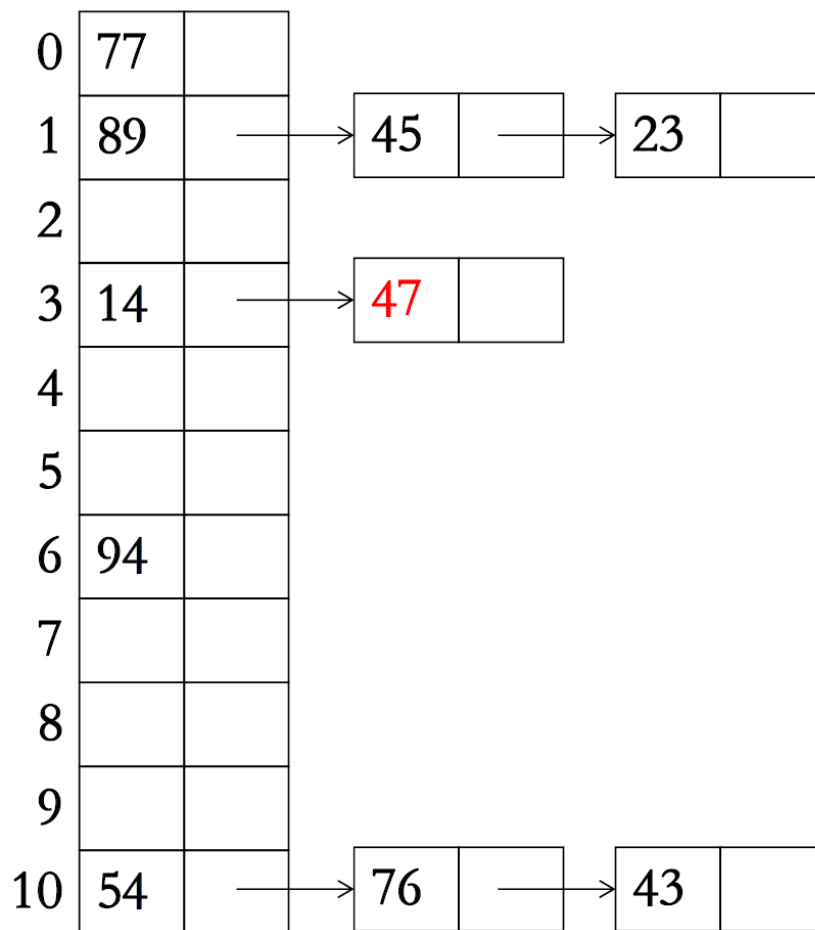
- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$





散列查找

- 独立链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$





散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

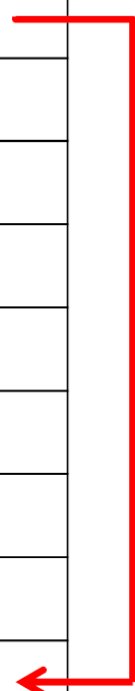
0	77	
1	89	
2		
3	14	
4		
5		
6	94	
7		
8		
9		
10	54	



散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(\text{key}) = \text{key} \% 11$

0	77	
1	89	
2		
3	14	
4		
5		
6	94	
7		
8		
9	45	
10	54	





散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3	14	
4		
5		
6	94	
7		
8	76	
9	45	
10	54	

Diagram illustrating the mapping of keys to indices in a hash table using the division method. The keys are mapped to indices as follows:

- Key 89 maps to index 1 (89 % 11 = 8)
- Key 76 maps to index 8 (76 % 11 = 10)
- Key 45 maps to index 9 (45 % 11 = 1)



散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3	14	
4		
5		
6	94	
7	23	
8	76	
9	45	
10	54	



散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3	14	
4		
5	43	
6	94	
7	23	
8	76	
9	45	
10	54	



散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3	14	
4		
5	43	
6	94	
7	23	
8	76	
9	45	
10	54	



散列查找

- 公共链表地址法
- 例，10个数据项，关键码分别为
54,77,94,89,14,45,76,
23,43,47，散列表长度为11，散列函数为除留
余数法
 $h(key)=key\%11$

0	77	
1	89	
2		
3	14	
4	47	
5	43	
6	94	
7	23	
8	76	
9	45	
10	54	



散列查找

- 开放地址法的表长是固定的，独立链表地址法的表项是动态分配的。链表法的缺点是要为每个表项设立指针域。
- 开放地址法
 - 线性探测法
 - 双散列函数探测法
- 链表地址法
 - 独立链表地址法
 - 公共链表地址法

独立链表地址法是查找效率最好的解决冲突的方法，是解决冲突的首选方法。



散列查找

- 平均查找长度 (ASL)
 - 查找成功的ASL
 - 查找不成功的ASL

关键字序列: (7、8、30、11、18、9、14)

散列函数:

$$H(\text{Key}) = (\text{key} \times 3) \text{ MOD } 7$$

装载因子:

0.7



散列查找

地址	0	1	2	3	4	5	6	7	8	9
key	7	14		8		11	30	18	9	

第一个元素7，带入散列函数，计算得0。

第二个元素8，带入散列函数，计算得3。

第三个元素30，带入散列函数，计算得6。

第四个元素11，带入散列函数，计算得5。

第五个元素18，带入散列函数，计算得5；和11冲突，使用线性探测法，得7。

第六个元素9，带入散列函数，计算得6；和30冲突，使用线性探测法，得8。

第七个元素14，带入散列函数，计算得0；和7冲突，使用线性探测法，得1。