9.3动态查找表（续）

DS/SS

哈希表查找

哈希表查找=查找表+顺序表+Hash规则

基本思想：

在线性表和树结构中，元素在结构中的相对位置是随机的，和元素的关键字之间不存在确定的关系，在这些结构作用下的查找表中查找时需要进行一系列的和关键字的比较（包括“相等”、“不相等”、“大于”、“小于”、“等于”比较)。

哈希表查找的思路是希望不通过比较，一次存取便能得到所查元素。显然，前提是每个关键字和结构中一个唯一的存储位置相对应。这种对应关系为查找提供依据和线索。

9.3动态查找表（续）

哈希表中基本概念和术语

- 哈希函数(Hash Function)：由关键字到元素存储位置之间的对应关系H称为哈希表函数。它是一个映象，设置灵活，只要使得关键字由此所得的哈希函数值都落在表长允许范围内即可。H: Key ---> Address；key的哈希函数值记H(key)。
- 冲突(collision)：对不同关键字可能得到同一哈希地址的现象，即key1 != key2，而H(key1)=H(key2)。具有相同函数值的关键字对该哈希函数来说称作同义词(synonym)。
- 哈希表(Hash Table):根据设定的哈希函数H(key)和处理冲突的方法将一组关键字映象到一个有限的连续的地址集(区间)上，并以关键字在地址集中的“象”作为元素在表中的存储位置，这种表称为哈希表。这种映象过程称为哈希造表或散列，所得存储位置称为哈希地址或散列地址。

9.3动态查找表（续）

哈希函数构造方法

构造和选取哈希函数的原则：均匀性(Uniform)。

若对于关键字集合中的任一个关键字，经哈希函数映象到地址集合中任何一个地址的概率是相等的，则称此类哈希函数为均匀的哈希函数。这种均匀性可以使一组关键字的哈希地址均匀分布在整个地址区间中，从而减少冲突。

实际中选取哈希函数要考虑的因素包括：

*计算哈希函数所需要时间；

*关键字长度；

*哈希表大小；

*关键字分布情况

*记录的查找频率

9.3动态查找表（续）

DS/SS

- **直接定址法**：取关键字或关键字的某个线性函数值为哈希地址。即：H(key) = key 或H(key) = a*key+b (a,b为常数)。基于这种哈希函数的哈希表中不会发生冲突。但太理想化。
- **数字分析法**：假设关键字是以r为基的数（如：以10为基的十进制数），并且哈系表中可能出现的关键字都是事先知道的，则可取关键字的若干数位组成哈希地址。当然，所取数位应该能比较均匀地区分这些关键字。
- **平方取中法**:取关键字平方后的中间几位为哈希地址。因为一个数平方后的中间几位数和数的每一位都相关，由此使随机分布的关键字得到的哈希地址也是随机的。
- **折叠法**: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)作为哈希地址。关键字位数很多，而且关键字中每一位上数字分布大致均匀时，可以采用折叠法
- **随机数法**:选择一个随机函数，取关键字的随机函数值为它的哈希地址，即H(key)=random(key)。关键字长度不等时采用该方法构造哈希函数较恰当。

9.3动态查找表（续）

DS/SS

哈希函数构造方法

- **除留余数法**：取关键字被某个不大于哈希表表长m的数p除后所得余数为哈希地址。即：

  H(key) = key MOD p　p<=m

  可以对关键字直接取模，也可以在折叠、平方取中等运算后取模。

  一般情况下，p取质数或不包含小于20的质因数的合数。

9.3动态查找表（续）

处理冲突的方法

假设哈希表地址集为：0~(m-1)，冲突是指由关键字得到的哈希地址为j(0<=j<=m-1)的位置上已存有记录，则“处理冲突”就是为该关键字的元素找到另一个“空”的哈希地址。在处理冲突的过程中可能得到一个地址序列H_i, i= 1,2,…,k (H_i属于[0..m-1])。在处理冲突过程中，若H_1发生冲突，则求H_2；若H_2发生冲突，求H_3，依次类推，直到H_k不发生冲突为止。H_k为元素在表中的地址。

9.3动态查找表（续）

- 开放定址法： $H_i = (H(key) + d_i) \text{ MOD } m$　　$i=1,2,\ldots,k(k<=m-1)$

  其中：H(key)为哈希函数；m为哈希表表长；d_i 为增量序列.

  若：(1)$d_i = 1,2,3,\ldots,m-1$,称线性探测再散列；

  (2)$d_i = 1^2, -1^2, 2^2, 2^2, \ldots, +k^2, -k^2$,称二次探测再散列；

  (3)d_i = 伪随机数序列,称伪随机探测再散列。

H(key) = key MOD 11, 表中已有关键字17, 60, 29,现哈希关键字为38的元素.

线性探测：

0	1	2	3	4	5	6	7	8	9	10
					60	17	29	38		
					38	38	38	d_3=3		
					H(38)	d_1=1	d_2=2			

- 避免“二次聚集”
- 线性探测:空间足够时,可以最终完成
- 二次探测:当m=4j+3的素数时才能完成

二次探测：

0	1	2	3	4	5	6	7	8	9	10
				38	60	17	29			
				d_2=-1	38	38				
					H(38)	d_1=1				

9.3动态查找表（续）

- 再哈希法： $H_i = RH_i(key)$ i=1,2,…,k

  RH_i均是不同的哈希函数,即在同义词产生地址冲突时计算另一个哈希函数地址,直到冲突不再发生。

  这种方法不容易产生“聚集”，但增加了计算时间

- 公共溢出区法：

  设向量HachTable[0..m-1]为基本表，每个分量存放一个元素，另设向量OverTable[0..v]为溢出表。所有关键字和基本表中关键字为同义词的元素，不管它们由哈希函数得到的哈希地址是什么，一旦发生冲突，都填入溢出表。

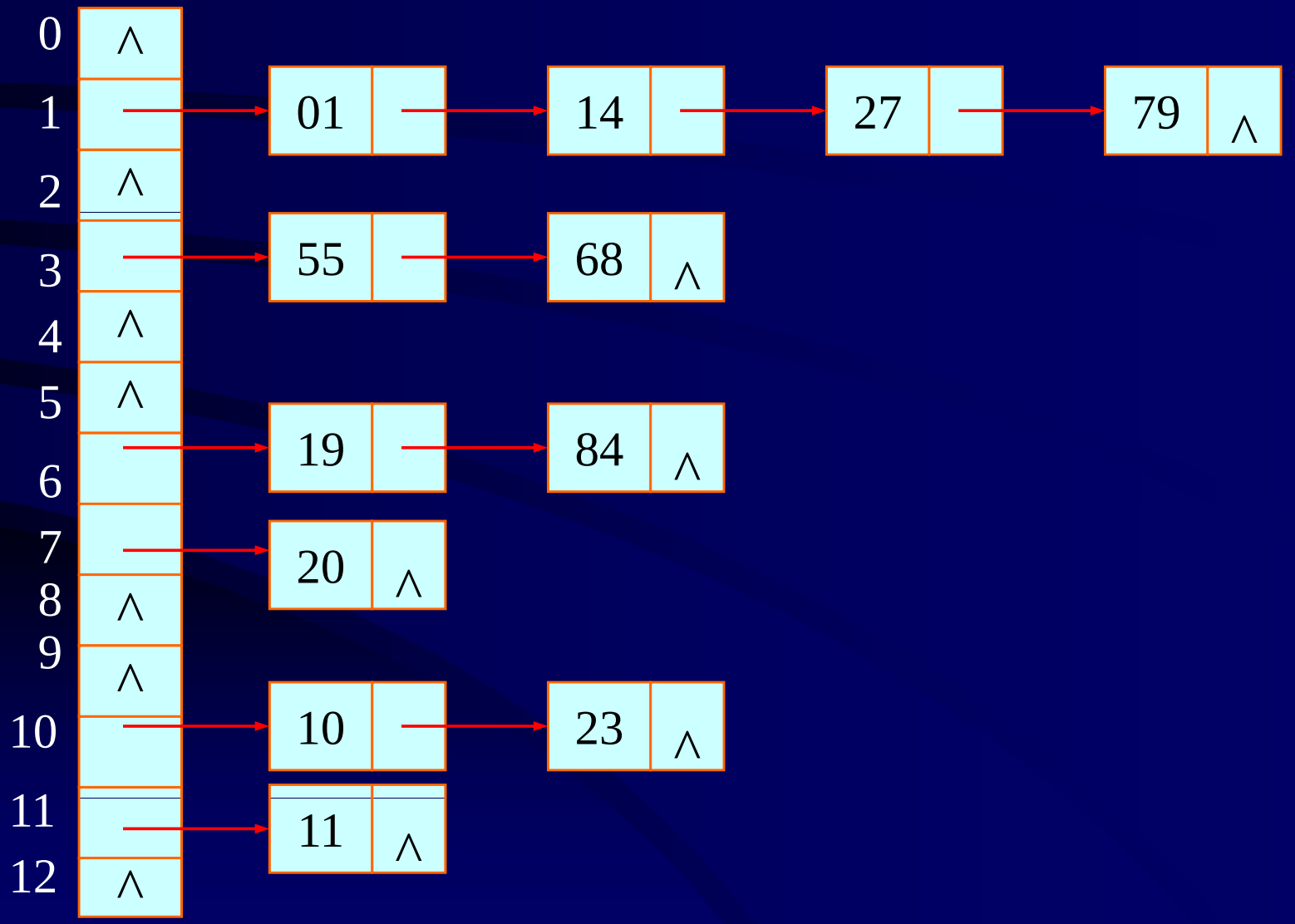
- 链地址法：

  将所有关键字为同义词的元素存储在同一线性链表中。

  用一个指针向量来表示：ChainHash[*m*]，凡是哈希地址为i的元素都插入到头指针为ChainHash[i]的链表中。

9.3动态查找表（续）

关键字集合：{19,14,23,01,68,20,84,27,55,11,10,79}，　H(key)= key MOD 13



9.3动态查找表（续）

DS/SS

哈希表上的查找及性能

哈希表查找过程：

由于哈希表为动态查找表，哈希造表的过程就是查找失败时元素插入的过程(在失败位置)。同样，根据哈希查找特征，哈希表上元素查找过程也类同于哈希造表过程。给定K值，根据造表时设定的哈希函数求得哈希地址，若表中此位置上没有元素，则查找不成功；否则比较关键字，若和给定值相等，则查找成功；否则根据造表时设定的处理冲突的方法找“下一地址”，直至哈希表某个位置为“空”或者表中所填元素的关键字等于给定值时为止。

哈希表查找性能取决于三个因素：

*哈希函数

*处理冲突的方法

*装填因子(表中填入元素个数/哈希表表长)

9.3动态查找表（续）

关键字集合：{19,14,23,01,68,20,84,27,55,11,10,79}

M=16; H(key) = key MOD 13,线性探测处理冲突

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	14	01	68	27	55	19	20	84	79	23	11	10			
	1	2	1	4	3	1	1	3	9	1	1	3			

性能分析：

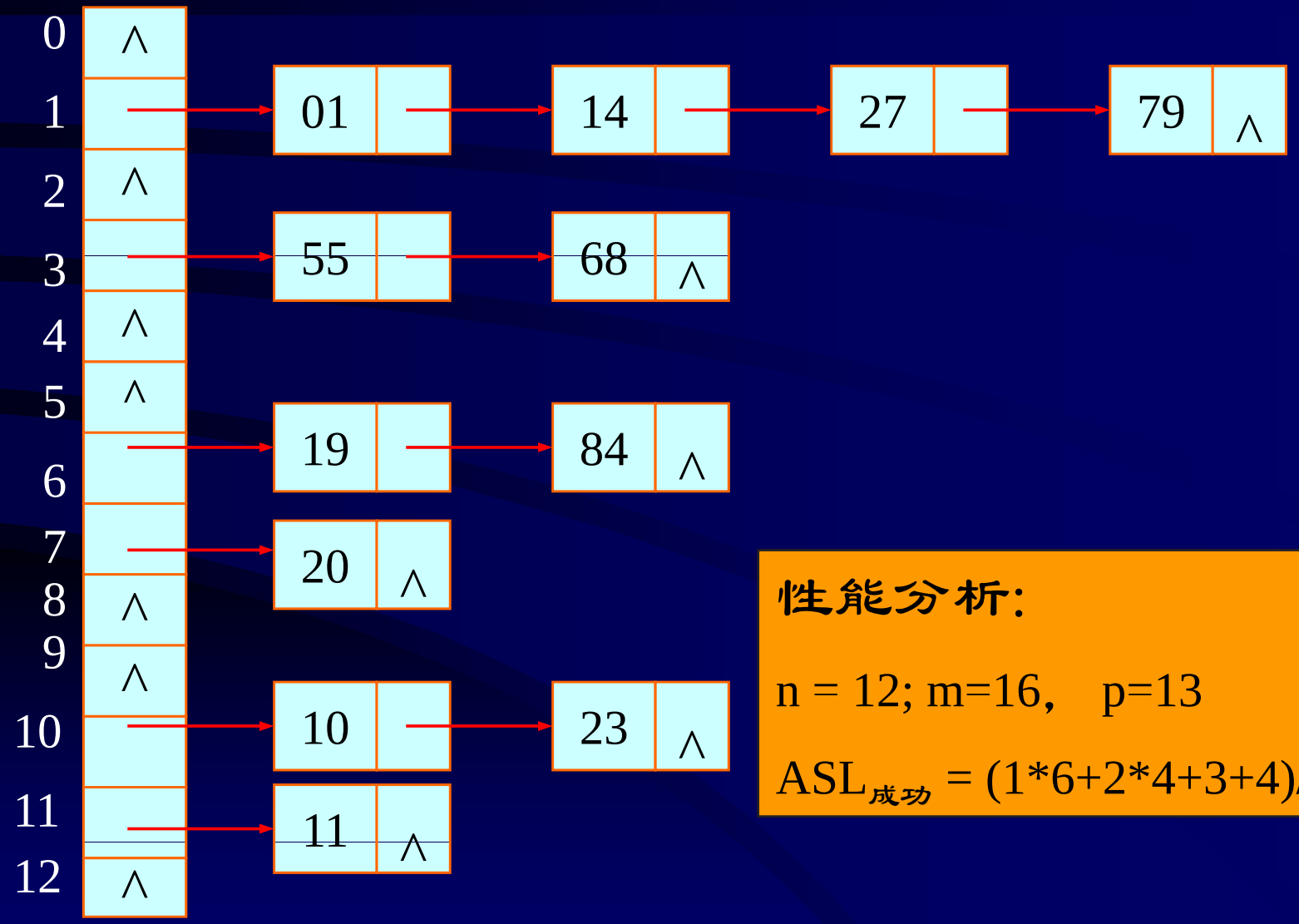
n = 12; m=16,　p=13

$ASL_{成功}$ = (1*6+2+3*3+4+9)/12=2.5

$ASL_{不成功}$ = (1+13 +12+11+10+9+8+7+6+5+4+3+2)/13=7

关键字集合：{19,14,23,01,68,20,84,27,55,11,10,79}

M=16; H(key) = key MOD 13,链地址法处理冲突



性能分析:

n = 12; m=16, p=13

$ASL_{成功}$ = (1*6+2*4+3+4)/12=1.75