# 第18讲 散列表

信息学院（智能应用研究院）
欧 新 宇

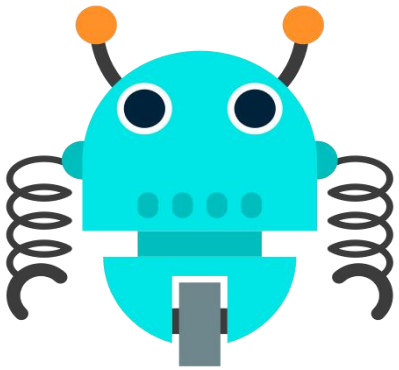
# 第7章 查找

## 考核要点

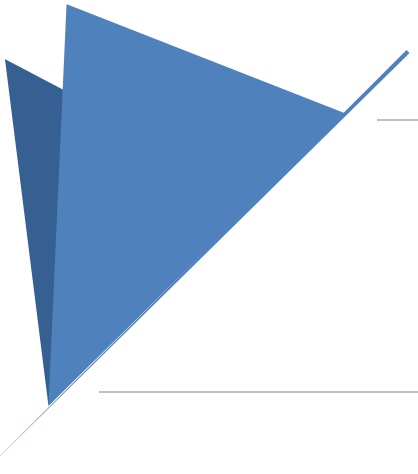
- **考核大纲**

查找的基本概念、顺序查找、分块查找、折半查找、二叉搜索树、平衡二叉树、红黑树、B树和B+树、散列表以及各种查找算法的分析及应用

- **复习要点**
  - 熟悉顺序查找和分块查找的基本方法
  - 掌握折半查找的过程、构造判定树、分析平均查找长度
  - 掌握二叉排序树、二叉平衡树和红黑树的概念、性质和相关操作
  - 掌握B树的插入、删除和查找操作的过程，了解B+树的基本概念和性质
  - 掌握散列表的构造和冲突处理以及散列查找的特征和查找成功、失败的性能分析



- 散列表的基本概念
- 散列函数的构造方法
- 处理冲突的方法
- 散列查找的性能分析



# 散列表的基本概念

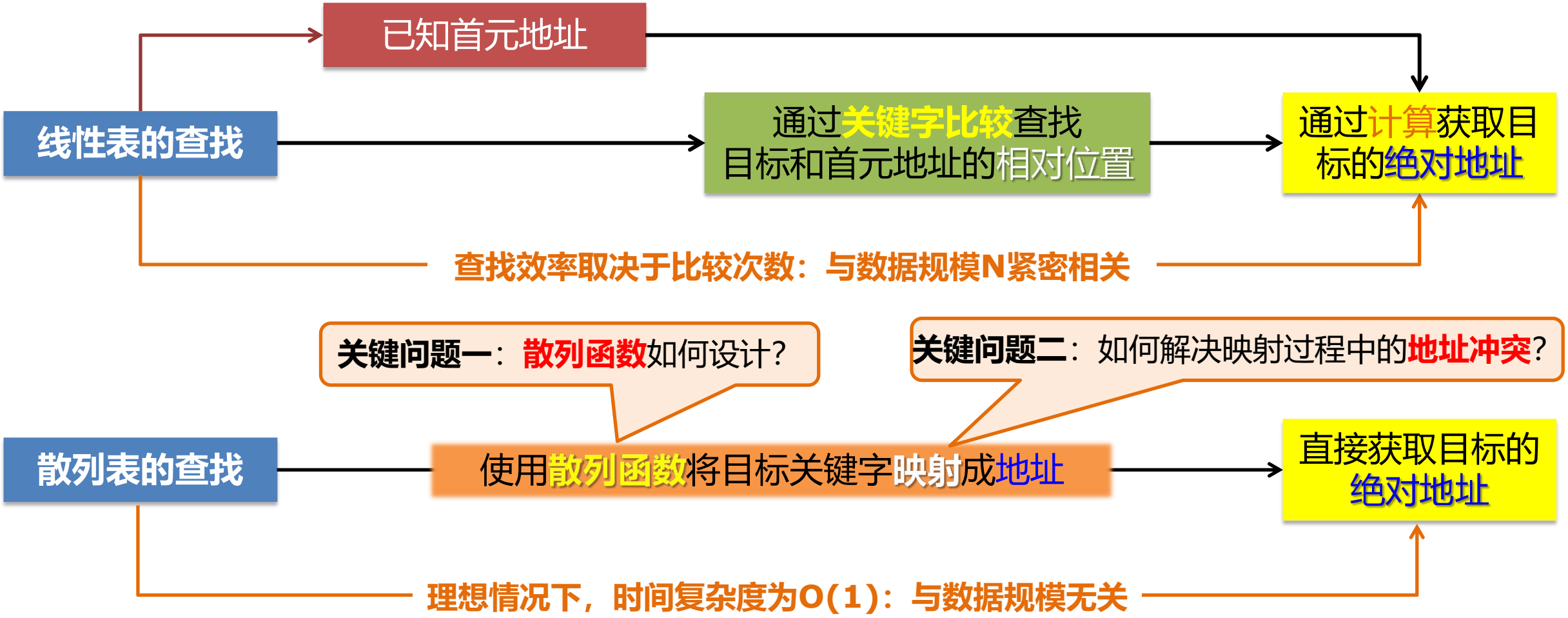
/ 什么是散列查找

/ 散列查找的基本过程

/ 散列表应用举例

# 散列表的基本概念

## 什么是散列表的查找



# 散列表的基本概念

## 什么是散列表的查找


**散列函数（Hashing Function, hashing, f）**：集合元素的关键字（key）与其存储位置（Loc）之间的关系函数，可以将关键字映射成地址，该地址也称为散列值。

**存储地址（Loc）**：关键字值为Key的集合元素的存储地址。

**散列表（Hash Table）**：又称为哈希表、杂凑表，是使用散列函数建立起来的表，用于存储地址集合元素。

**散列（Hash）**：又称哈希，指将关键字key使用哈希函数映射成存储地址Loc的过程。

**散列表查找的优点：**查找速度极快，**复杂度为O(1)**，查找效率与元素个数N无关。

# 散列表的基本概念

## 哈希表的应用举例一：基于哈希编码的数据访问

**学生信息表**

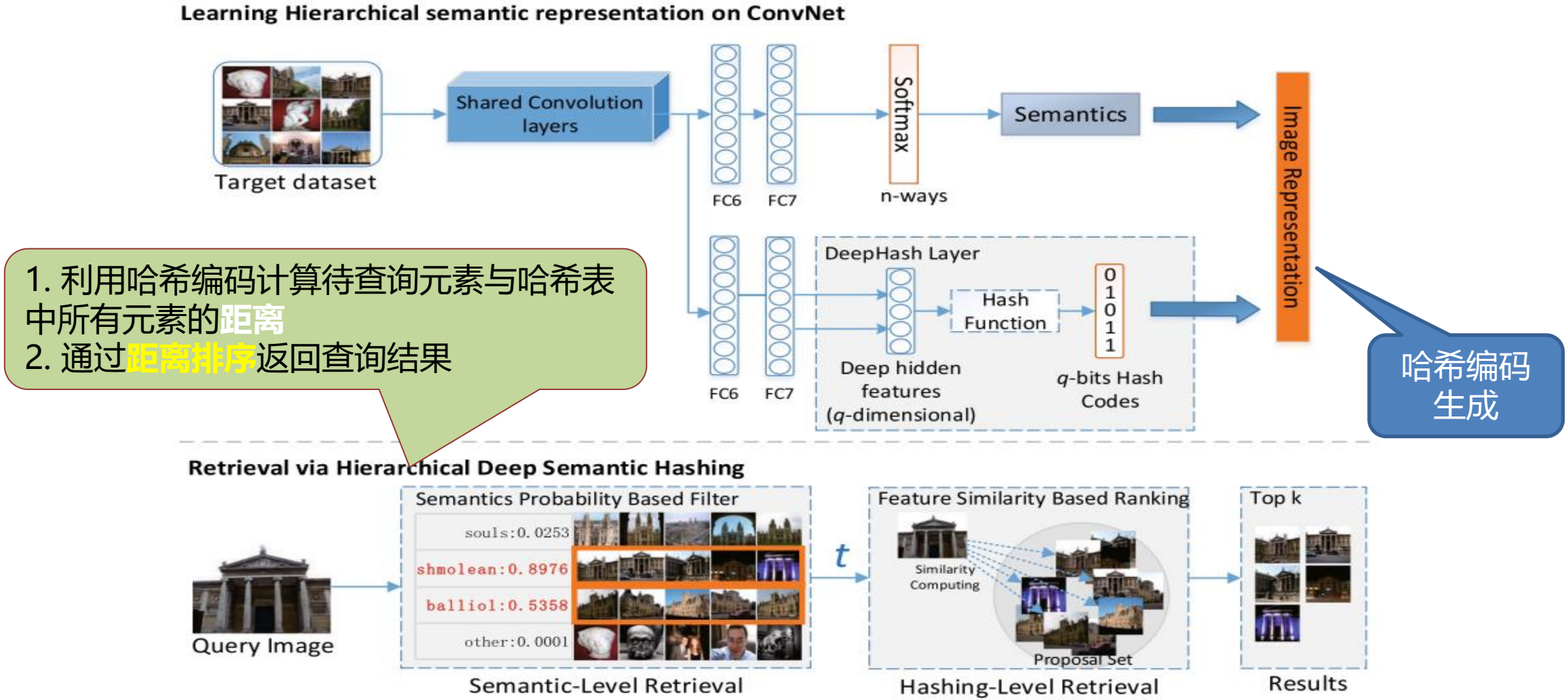
若将学生信息按如下方式存入计算机，如：

- ✓ 将20010118102**01**的所有信息存入V[01]单元；
- ✓ 将20010118102**02**的所有信息存入V[02]单元；
- ✓ ……
- ✓ 将20010118102**31**的所有信息存入V[31]单元。

查找20010118102**16**的信息，可直接访问V[16]！

# 散列表的基本概念

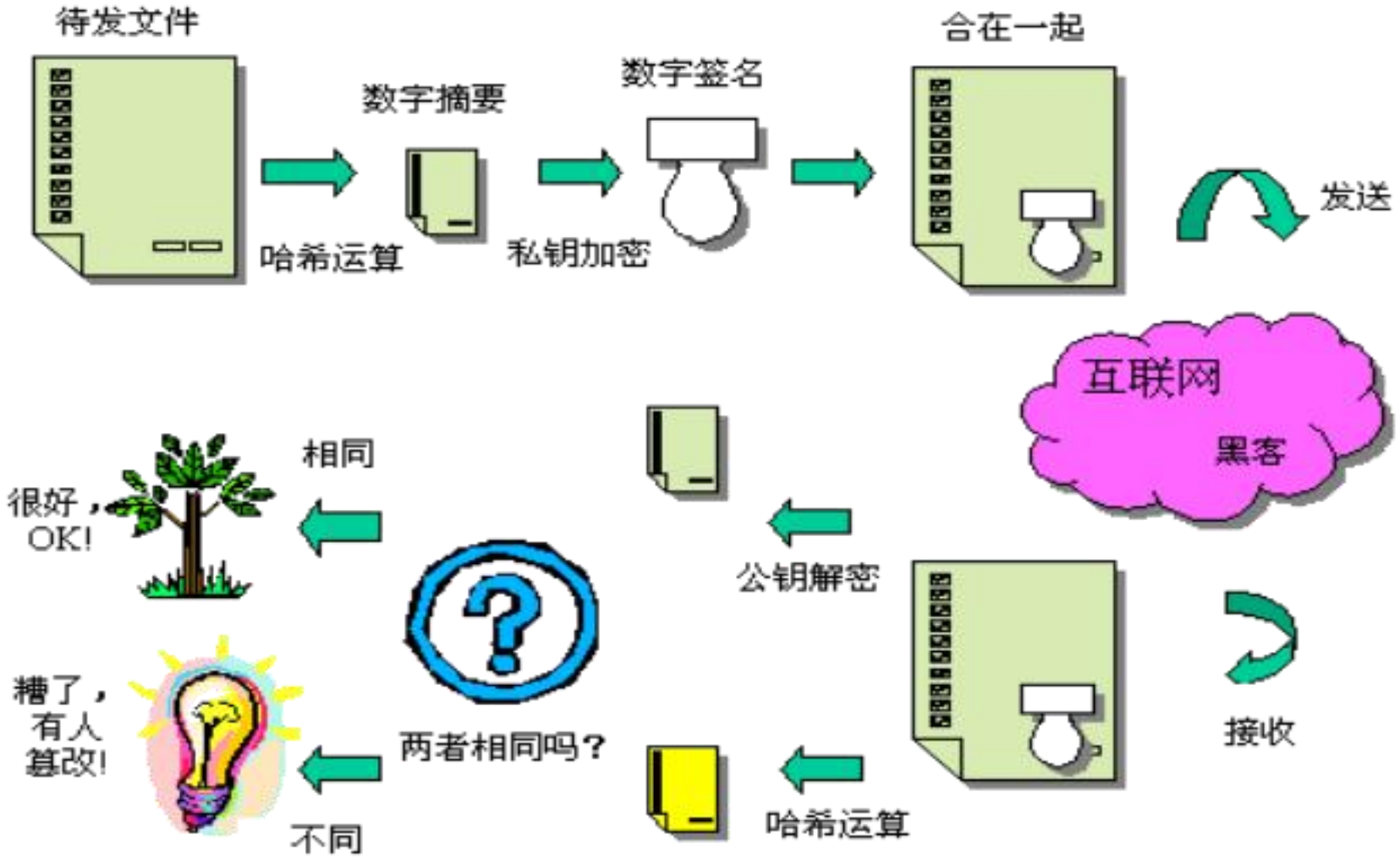
## 哈希表的应用举例二：基于哈希编码的图像检索




Xinyu Ou, Hefei Ling, Si Liu, Jie Lei. Hierarchical Deep Semantic Hashing for Fast Image Retrieval. *Multimedia Tools and Applications*. 2017. 76(20)

# 散列表的基本概念

## 哈希表的应用举例三：基于哈希编码的文件验证



# 散列表的基本概念

## 哈希表的应用举例四：省市人口统计表

- **省市人口统计表**

✓ 关键字：名称的汉语拼音表

✓ 散列表长度：30

✓ 编码：A~Z => 01~26

✓ 哈希函数$h_1$：$h_1$(Key) = 首字母编码

✓ 哈希函数$h_2$：$h_2$(Key) = (首字母编码+尾字母编码)%30

| Key | BeiJing | JiangSu | ShangHai | SiChuan | JiangXi | TianJin | ShanXi |
|---|---|---|---|---|---|---|---|
| $h_1$(key) | 02 | 10 | 19 | 19 | 10 | 20 | 19 |
| $h_2$(key) | 09 | 01 | 28 | 03 | 19 | 04 | 28 |

# 散列表的基本概念

## 哈希表的应用举例四：省市人口统计表

| Key | BeiJing | JiangSu | ShangHai | SiChuan | JiangXi | TianJin | ShanXi |
|---|---|---|---|---|---|---|---|
| $h_1$(key) | 02 | 10 | 19 | 19 | 10 | 20 | 19 |
| $h_2$(key) | 09 | 01 | 28 | 03 | 19 | 04 | 28 |

**发现一：**

- ✓ 对$h_1$：JiangSu与JiangXi，Shanghai、SiChuan与ShanXi都被映射到了相同位置，产生冲突
- ✓ 对$h_2$：ShangHai与ShanXi被映射到了相同的位置，产生冲突

**冲突：** Key1 ≠ Key2 =》 $h_1$(Key1) = $h_1$(Key2) **冲突是不允许发生的**

**同义词：** 给定一个哈希函数h，若两个不同关键字具有相同的散列值，则称这两个关键字为同义词

# 散列表的基本概念

## 哈希表的应用举例四：省市人口统计表

| Key | BeiJing | JiangSu | ShangHai | SiChuan | JiangXi | TianJin | ShanXi |
|---|---|---|---|---|---|---|---|
| $h_1$(key) | 02 | 10 | 19 | 19 | 10 | 20 | 19 |
| $h_2$(key) | 09 | 01 | 28 | 03 | 19 | 04 | 28 |

**发现二：**

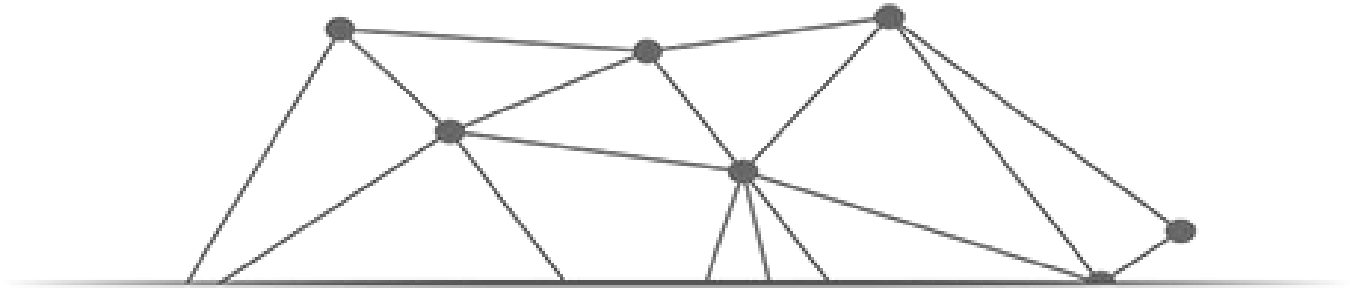
✓ 散列函数$h_1$(Key)发生冲突的次数明显少于散列函数$h_2$(Key)

$h_1$(Key) = 首字母编码

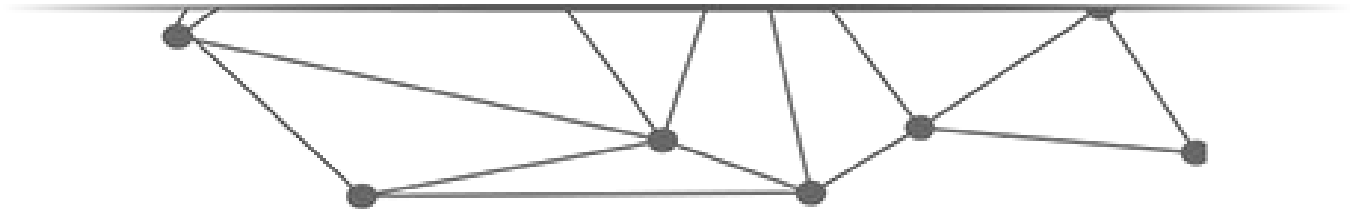
$h_2$(Key) = (首字母编码+尾字母编码)%30

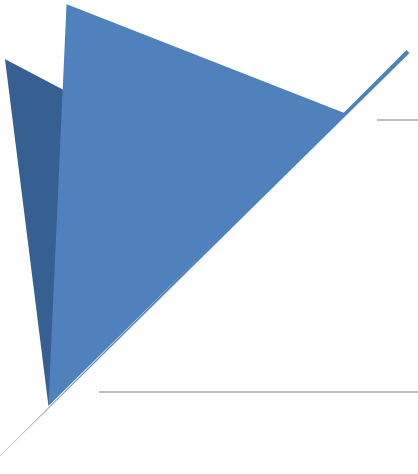
**冲突的发生频率与散列函数相关** ==》 1. 如何设计一个好的散列函数?

==》 2. 当冲突发生时，如何处理最好?



# 课堂互动





# 散列函数的构造方法

/ 什么是好的散列函数

/ 直接定址法　　/ 除留余数法　　/ 平方取中法

/ 折叠法　　/ 数字分析法　　/ 随机数法

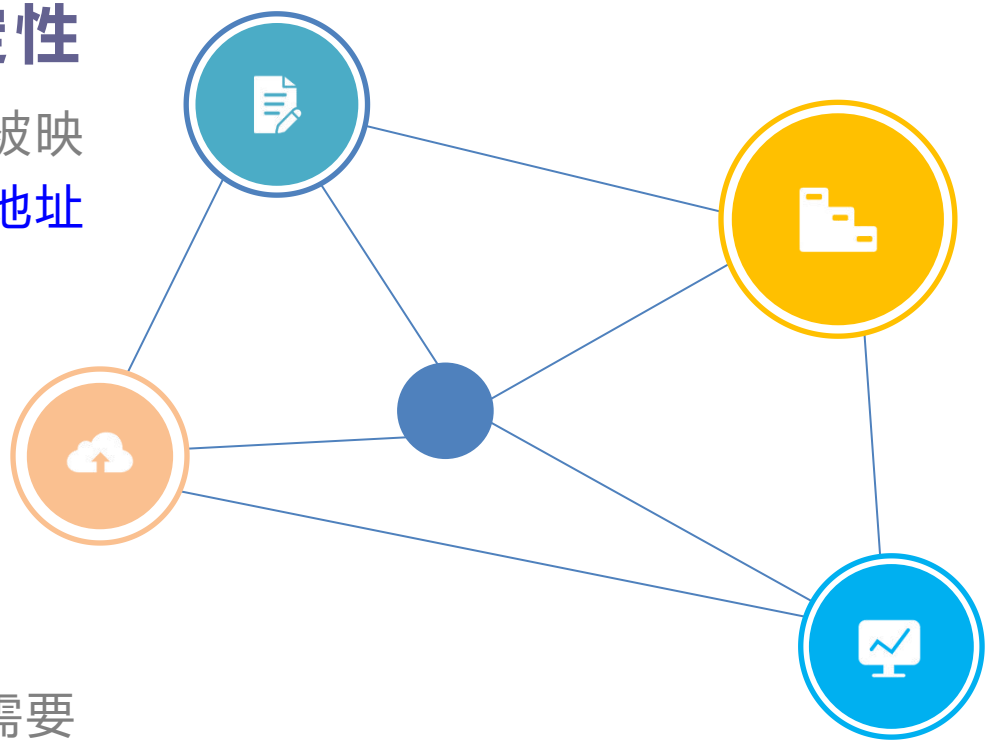
/ 均值哈希

# 散列函数的构造方法

## 什么是好的散列函数

### 确定性

同一个关键字始终被映射到同一个地址



### 快速、简单

散列函数尽量简单，能够保证在短时间内计算出任意关键字对应的散列地址。时间复杂度为O(1)。

### 满射

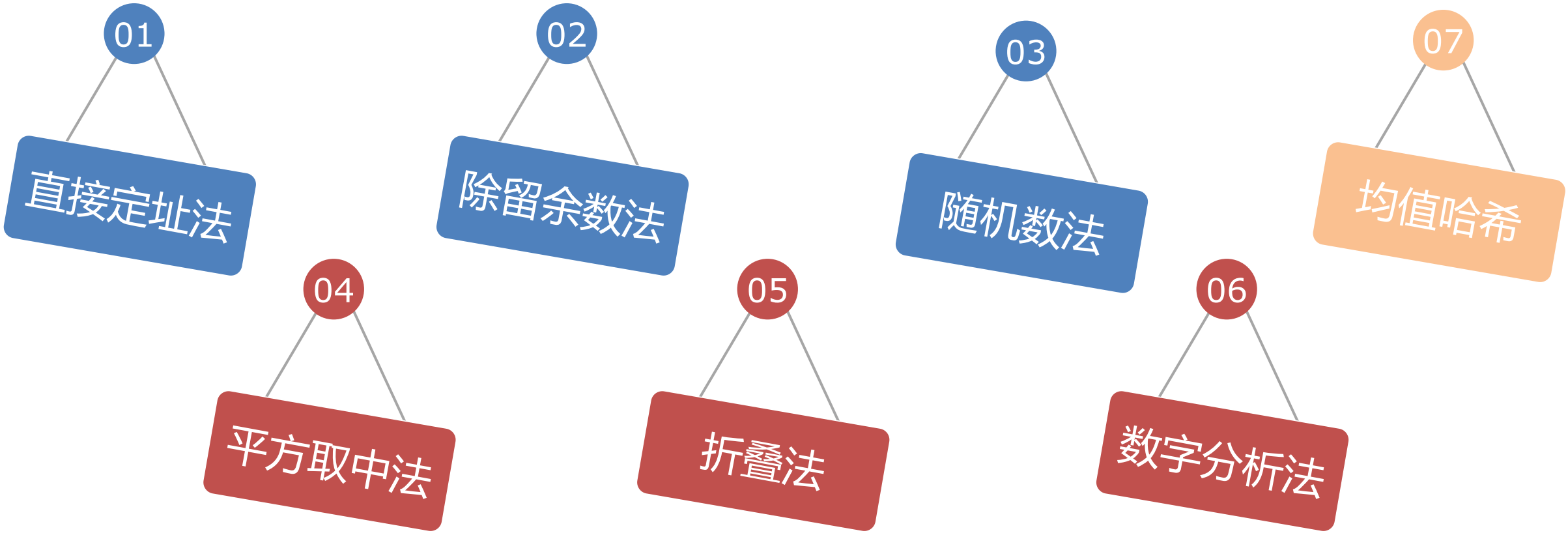
散列函数的定义域包含全部需要存储的关键字，经过唯一映射到值域后，能够覆盖整个值域

### 均匀分布

通过散列函数计算获得的各个地址，应接近等概率、均匀地分布在整个地址空间，避免多元素扎堆聚集，从而减少冲突

# 散列函数的构造方法

## 什么是好的散列函数



# 散列函数的构造方法

## 直接定址法

**直接定址法：**直接使用关键字的某个线性函数值作为散列地址。

- ✓ 优点：计算简单，不会产生冲突；
- ✓ 缺点：需要占用连续地址空间，空间利用率低。
- ✓ 适用场景：数据源的关键字基本呈连续分布。若关键字分布不连续，则空位较多，容易造成存储空间的浪费。
- ✓ 基本形式：

$$Hash(key) = a \times key + b \quad (a, b是常数)$$

**例1：试使用哈希函数Hash(key)=key/100对有限序列S = {100, 200, 400, 500, 800, 900, 1000}进行地址转换，并存储在对应数组中。**

空间利用率60%

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 100 | 200 | | 400 | 500 | | | 800 | 900 | 1000 |

# 散列函数的构造方法

## 除留余数法

**除留余数法：**假定散列表表长为m，取一个不大于m但接近或等于m的质数p，利用取余操作将关键字转换为散列地址。

- ✓ 优点：计算简单
- ✓ 缺点：存在不当点h(0)=0，与均匀分布相悖；相邻关键字散列到相邻地址。
- ✓ 基本形式：

$$Hash(key) = key \% p \quad (p \le m)$$

选择合适的模数p是关键。
实验证明，不大于表长但接近于表长的质数性能最佳。

**例2：假设散列表表长为1000，试求关键字的散列地址。**
**M = 1000**
**P = 997**

| 关键字 | ASCII编码 | 散列地址 |
|---|---|---|
| KEYA | 75698965 | 743 |
| KEYB | 75698966 | 744 |
| AKEY | 65756989 | 851 |
| BKEY | 66756989 | 860 |

相邻关键字散列到相邻地址

# 散列函数的构造方法

## 除留余数法的改进MAD

**改进的除留余数法：**

$$Hash(key) = (key \times a + b) \% p \quad (a, b > 0, p \le m)$$

**例3：假设散列表表长为1000，试求关键字的散列地址。**
**M = 1000， P = 997， a = 15，b = 87**

| 关键字 | ASCII编码 | 散列地址 |
|---|---|---|
| KEYA | 75698965 | 460 |
| KEYB | 75698966 | 475 |
| AKEY | 65756989 | 546 |
| BKEY | 66756989 | 538 |

b 是偏置量，用于消除不动点；

a 是间隔量，原本相邻地址变为间隔a

# 散列函数的构造方法

## 平方取中法

**平均取中法：**取关键字的平方值的中间几位作为散列值（冯·诺依曼）。

- ✓ 优点：计算方便，速度快，容易排查错误，且地址与关键字关系密切，分布比较均匀
- ✓ 缺点：种子值难以确定，无法保证足够的周期k
- ✓ 适用场景：关键字每位取值都不均匀
- ✓ 基本形式：

$$Hash(key) = key^2 \text{ 的中间若干k位} \quad (10^{k-1}<n<10^k)$$

**例4：若序列中集合元素个数 n = 765，则 $10^{k-1}<n<10^k$，于是有 k = 3。**

| 关键字 | ASCII编码 | $Key^2$ | 散列地址 |
|---|---|---|---|
| KEYA | 75698965 | 0573033302071220 | 330 |
| KEYB | 75698966 | 0573033453469160 | 345 |
| AKEY | 65756989 | 0432398160 2346120 | 160 |
| BKEY | 66756989 | 0445649558 0346120 | 558 |

位数不够
高位补零

# 散列函数的构造方法

**折叠法：**

1. 将关键字从左到右分割成位数相等的几个部分（最后一部分位数不够时可短一些）

2. 将几个部分进行叠加求和；或使用**回折叠法**进行叠加求和

3. 根据散列表表长k，取后k位作为散列地址

## 折叠法

**例5：设关键字 key=12323434567890，散列表表长为3位。**

将关键字key划分为5组，分别是：123 234 345 678 90

| | | |
|---|---|---|
| 123 | | 123 |
| 234 | | 432 |
| 345 | | 345 |
| 678 | | 876 |
| + 90 | 哈希值 | + 90 | 哈希值 |
| 1470 | 470 | 1866 | 866 |
| **折叠法** | | **回折叠法** | |

# 散列函数的构造方法

## 数字分析法

**数字分析法：**观察关键字编码组合规律，取分布均匀的若干位。

- ✓ 优点：分布均匀
- ✓ 缺点：做出良好的分析依赖于主观因素或分析算法，且关键字更新后需要重新构造散列函数

| 位 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 关键字 | 9 | 4 | 2 | 1 | 2 | 3 |
| | 9 | 4 | 2 | 3 | 5 | 4 |
| | 9 | 4 | 1 | 4 | 4 | 5 |
| | 9 | 5 | 2 | 5 | 1 | 7 |
| | 8 | 4 | 2 | 6 | 6 | 0 |
| | 9 | 4 | 2 | 8 | 8 | 9 |
| | 9 | 4 | 4 | 3 | 3 | 2 |

可以看出第4，5，6位取值分布均匀，

故取第4，5，6三位的值为散列函数值

# 散列函数的构造方法

## 随机数法

**随机数法：**取关键字的值作为随机函数的种子值来生成散列地址。

- ✓ 优点：构造简单、快速
- ✓ 缺点：做出良好的分析依赖于主观因素或分析算法，且关键字更新后需要重新构造散列函数
- ✓ 适用场景：当关键字长度不等时，随机数法更易于实现
- ✓ 基本形式：

$$Hash(key) = random(key)$$

# 散列函数的构造方法

## 基于特征值的均值哈希

原始数据

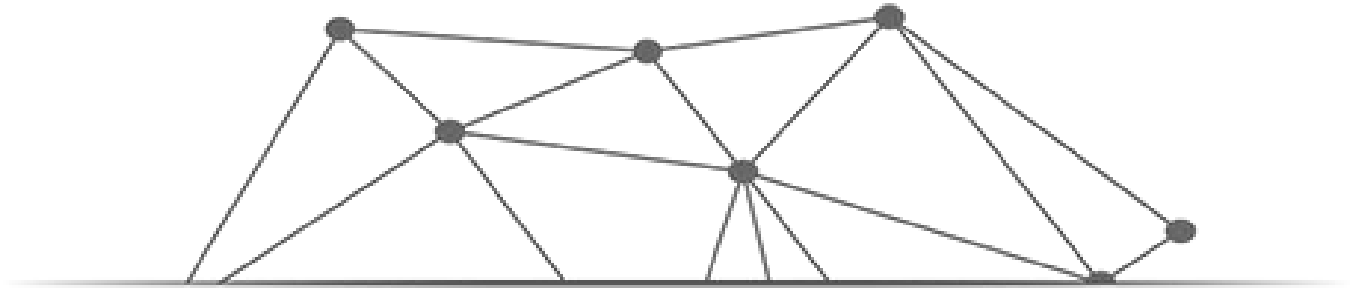
特征提取

| 特征值 |
| --- |
| 0.71 |
| 0.23 |
| 0.87 |
| 0.77 |
| 0.01 |
| 0.98 |
| 0.75 |
| 0.63 |
| … |
| 0.22 |
| 0.45 |
| 0.85 |

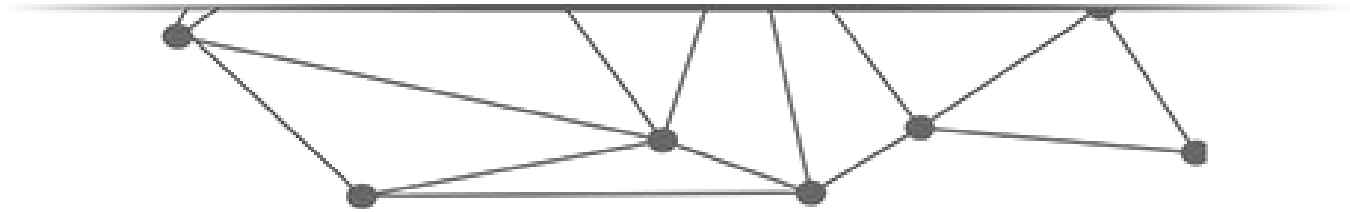
$$h(x)=\begin{cases}1 & x-avg(X)>0\\ 0 & x-avg(X)<0\end{cases}$$

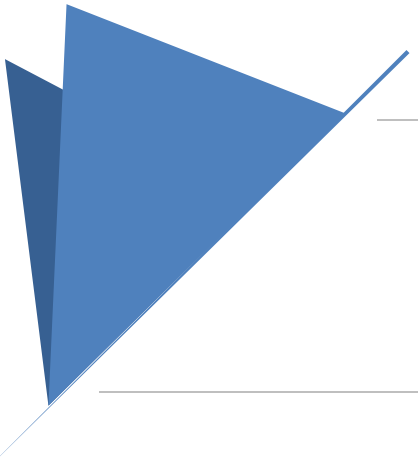
哈希变换

| 哈希编码 |
| --- |
| 1 |
| 0 |
| 1 |
| 1 |
| 0 |
| 1 |
| 1 |
| 1 |
| … |
| 0 |
| 0 |
| 1 |



# 课堂互动





# 处理冲突的方法

/ 伪随机探测法

/ 二次探测法

/ 拉链法

/ 线性探测法

/ 双散列法

# 处理冲突的方法


**开放定址法**：有冲突时，寻找下一个空的哈希地址，只要确保哈希表足够大，总能找到空的哈希地址将数据元素存入。

**链地址法**：相同哈希地址的记录链成一个单链表，m个哈希地址，就设置m个单链表，然后使用一个数组将所有单链表的表头指针存储起来，形成一个动态结构。

# 开放定址法

**Step1**：取数据元素的关键字key，并计算其哈希地址。

- ✓ 若该地址对应的存储空间还没有被占用，则将该元素存入；
- ✓ 否则，执行Step2解决冲突。


**Step2**：根据选择的冲突处理方法，计算关键字key的下一个存储地址。

- ✓ 若下一个存储地址仍被占用，则继续执行Step2；
- ✓ 直到找到能使用的空地址为止。

# 线性探测法

## 线性探测法的基本原理

**基本思想：**从基地址 h(key) 开始，按照线性探查序列查找元素。若 h(key) 被占用，则检查 h(key)+1；若也被占用，则再检查探测序列中的下一个位置；直到某个位置为空时，将关键字插入该位置。

**搜索成功：**在查询表中找到关键字值为key的元素

**查询失败：**1. 遇到空位置

2. 探查到 h(key) 前一个位置 （表满）

# 线性探测法

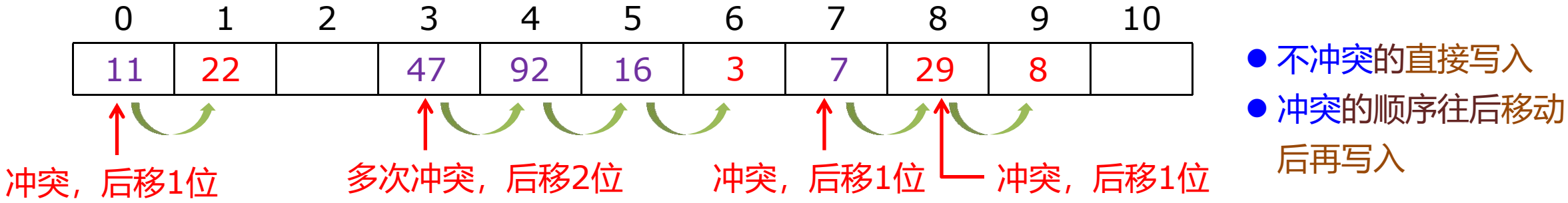
## 线性探测法的存储地址分配

**例5：若存在关键字码集 {47，7，29，11，16，92，22，8，3}，试将关键字存入数组 A[11] 中。**

根据题意，可知哈希表表长 m = 11，若哈希函数为 Hash(key) = key % 11，则可以得到关键字的初步哈希编码：

| key | 47 | 7 | 29 | 11 | 16 | 92 | 22 | 8 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| Hash(key) | 3 | 7 | 7 | 0 | 5 | 4 | 0 | 8 | 3 |

**存储空间分配：**




- 不冲突的直接写入
- 冲突的顺序往后移动后再写入

# 线性探测法

## 基于线性探测法的查找过程

**例6：试在散列表A中查找关键字值为47, 46, 3的位置。**

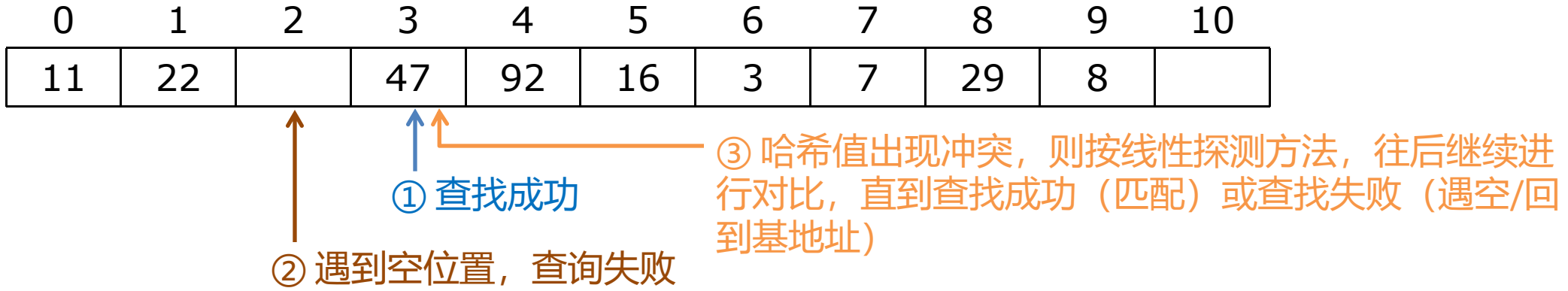
1. 计算关键字的哈希值

   ① hash[47] = 47%11 = 3

   ② hash[46] = 46%11 = 2

   ③ hash[3] = 3%11 = 3

2. 按照哈希值在查找表中查找目标值



# 线性探测法

## 基于线性探测法的删除操作

**例7：试在下列散列表中，先删除92，再查询3，最后再插入关键字35。**

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

1. 若不需要先执行删除92，则查询3的操作和例5相同。先求3%11=3，然后使用哈希值3按照线性探测法从地址3开始进行查询，最终能够在地址6完成查询。

2. 若先执行删除92操作，再查询3，结果是什么呢?

92%11 = 4

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

遇到空位置，搜索失败!

不匹配，后移1位

如何解决?

# 线性探测法

## 删除操作的改进

- **删除需要考虑的两个关键点：**

1. 不能仅仅简单删除元素，否则会隔离探查序列后面的元素，影响搜索；
2. 删除元素后，该位置能够重新使用。

- **解决办法：**

1. 为每个位置增加一个empty标志域，用于标识该位置是否使用过
2. 删除元素时，不改变标志位，仍保持False，元素值改为NeverUsed

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| key | 11 | 22 | Never Used | 47 | 92 | 16 | 3 | 7 | 29 | 8 | Never Used |
| empty | F | F | T | F | F | F | F | F | F | F | T |

# 线性探测法

## 删除操作的改进

**例7：试在下列散列表中，先删除92，再查询3，最后再插入关键字35。**

**删除元素的改进**

1. 按哈希码线性探查待删除元素
2. 若搜索成功，则将该位置的Key值设置为NeverUsed，但empty不作更改

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| key | 11 | 22 | Never Used | 47 | Never Used | 16 | 3 | 7 | 29 | 8 | Never Used |
| empty | F | F | T | F | F | F | F | F | F | F | T |

# 线性探测法

## 查询操作的改进

**例7：试在下列散列表中，先删除92，再查询3，最后再插入关键字35。**

**改进算法：** 从基地址h(key)开始查询，若遇到 empty=False 时，可继续按照线性探测方法进行查询

**搜索成功：** 找到关键字为key的元素

**搜索失败：** 1. 遇到一个从未被使用过的空位置(empty=T且key=NeverUsed)

2. 回到h(key) 表满

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| key | 11 | 22 | Never Used | 47 | Never Used | 16 | 3 | 7 | 29 | 8 | Never Used |
| empty | F | F | T | F | F | F | F | F | F | F | T |

3%11=3

多次冲突，后移2位(遇到 empty=False不必管)

# 线性探测法

## 插入操作的改进

**例7：试在下列散列表中，先删除92，再查询3，最后再插入关键字35。**

35%11 = 2

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| key | 11 | 22 | 35 | 47 | 92 | 16 | 3 | 7 | 29 | 8 | Never Used |
| empty | F | F | F | F | F | F | F | F | F | F | T |

**插入失败：** 1. 若搜索的目标地址存在重复元素，则插入失败

2. 如果未搜索到关键字Key，但表已满，则插入失败

**插入成功：** 若搜索不到，且表未满，则线性探查第一个NeverUsed的位置，并插入新元素，然后将empty设置为False

# 线性探测法

## 线性探查的缺点

- 表中的所有empty位置，会很快变为False
- 容易在表中产生线性聚集（h(key) = h(key) + 1）。随着探查次数的增加，影响搜索效率。

改进方法：1. 伪随机探测
2. 二次探查法
3. 双散列法

# 二次探查法

## 二次探查的基本思想

**基本思想：**为关键字的基地址定义一个偏移值，用于替代发生冲突时，使用基地址线性增长查找所带来的元素聚集问题。

**实现方法：**

1. 定义一个二次探测序列：$h(key), h_1(key), h_2(key), ..., h_{2i-1}(key), h_{2i}(key),...$

$$\begin{cases} h_{2i-1}(key) = (h(key)+i^2)\%m, & \text{奇数=》+系数平方} \\ h_{2i}(key) = (h(key)-i^2)\%m, & \text{偶数=》-系数平方} \end{cases}$$

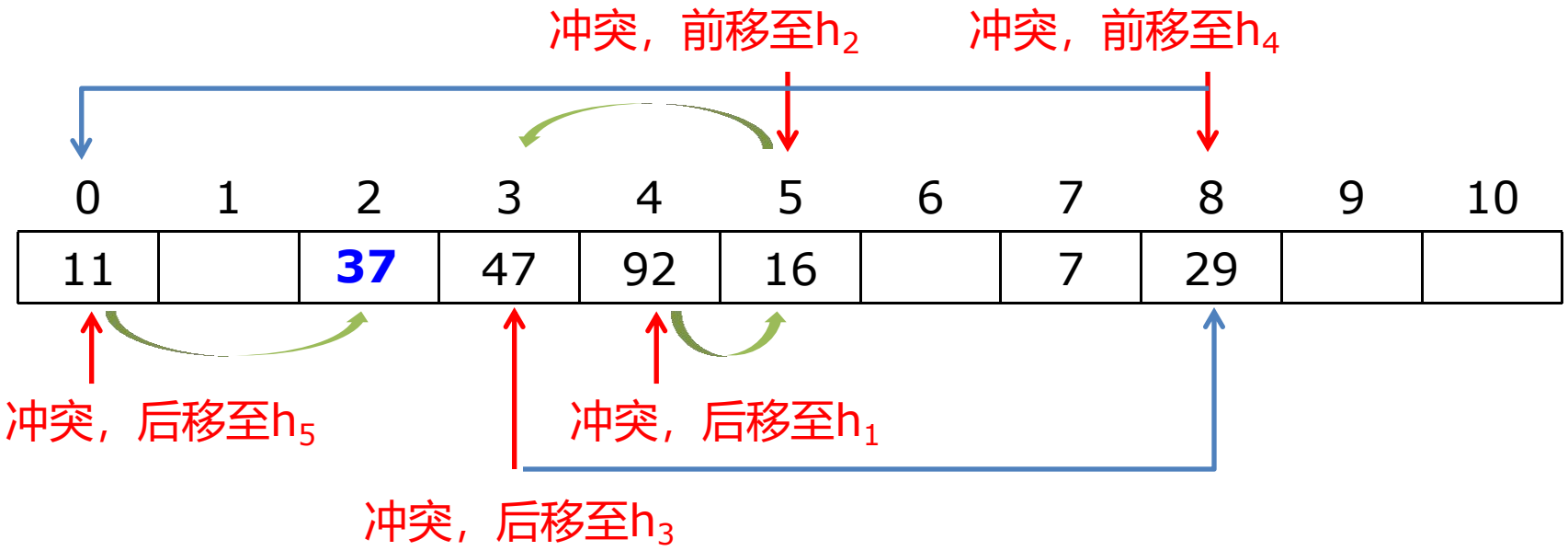
$i = 1, 2,..., (m-1)/2$

2. 当发生冲突时，按照二次探查序列依次探查

# 二次探查法

## 二次探查的基本思想

**例8：试在下列散列表中插入关键字37。**




$h(37) = 37\%11 = 4$

$h_1(37) = (h(37)+1^2)\%11 = 5$

$h_2(37) = (h(37)-1^2)\%11 = 3$

$h_3(37) = (h(37)+2^2)\%11 = 8$

$h_4(37) = (h(37)-2^2)\%11 = 0$

$h_5(37) = (h(37)+3^2)\%11 = 2$

$h_6(37) = (h(37)-3^2)\%11 = 6$

**二次探查法能改善"线性聚集"**

但**同义词**会有相同的探查序列，产生"**二次聚集**"

# 双散列法

## 双散列法的基本思想

**基本思想：**双散列法又称为**再散列法**，通过定义两个散列函数$h_1$和$h_2$，来对$h_1$散列值产生一个固定增量，以改善“二次聚集”，从而实现跳跃式探查。

**实现方法：**

1. 设存在散列函数 h(key) = key%M，则使用如下方式解决第i次冲突：

$$h(key) = (h(key) + i*h_2(key))\%M$$

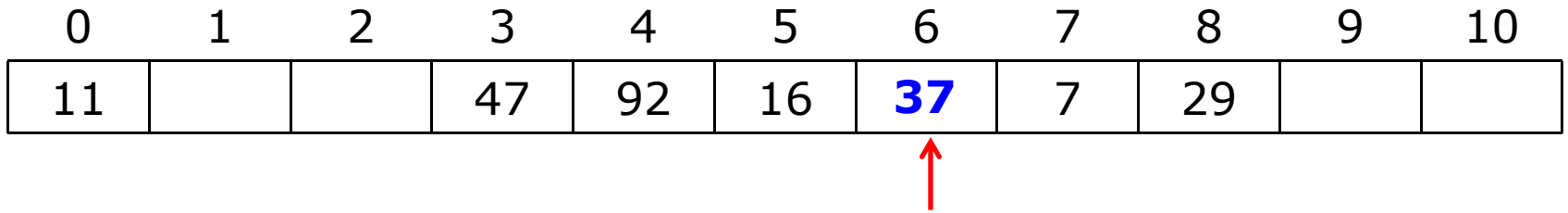
其中，$\begin{cases} h_2(key) = (key\%M)+1 \\ h_2(key) = (key\%M-2)+1 \end{cases}$，若M为素数

2. 当发生冲突时，按照双散列法生成的序列依次探查

# 双散列法

## 双散列法的基本思想

**例8：试在下列散列表中插入关键字37。**




h(key) = h(37) = key%M = 37%11 = 4

**Round1：**

$h_2(key) = h_2(37) = (key\%M-2)+1 = (37\%(11-2))+1 = 2$

$h(key) = h(37) = (h(key) + i*h_2(key))\%M = (4 + 1*2)\%11 = 6$

**Round2：**

...

# 伪随机探测法

## 伪随机探测法的基本思想

**基本思想：**为关键字的基地址增加一个随机偏移值，使得发生冲突时，不至于因为线性增长而带来元素聚集。

**实现方法：**

$$h(key) = (h(key) + d_i) \% m$$

$d_i$是随机函数生成的随机数

# 链地址法（拉链法）

**Step1：**取数据元素的关键字key，并计算其哈希地址。

- ✓ 若该地址对应的链表为空，则将该元素插入此链表；
- ✓ 否则，执行Step2解决冲突。


**Step2：**根据选择的冲突处理方法，计算关键字key的下一个存储地址。

- ✓ 若该地址对应的链表不为空，则利用链表的前插法或后插法将该元素插入此链表

# 链地址法（拉链法）

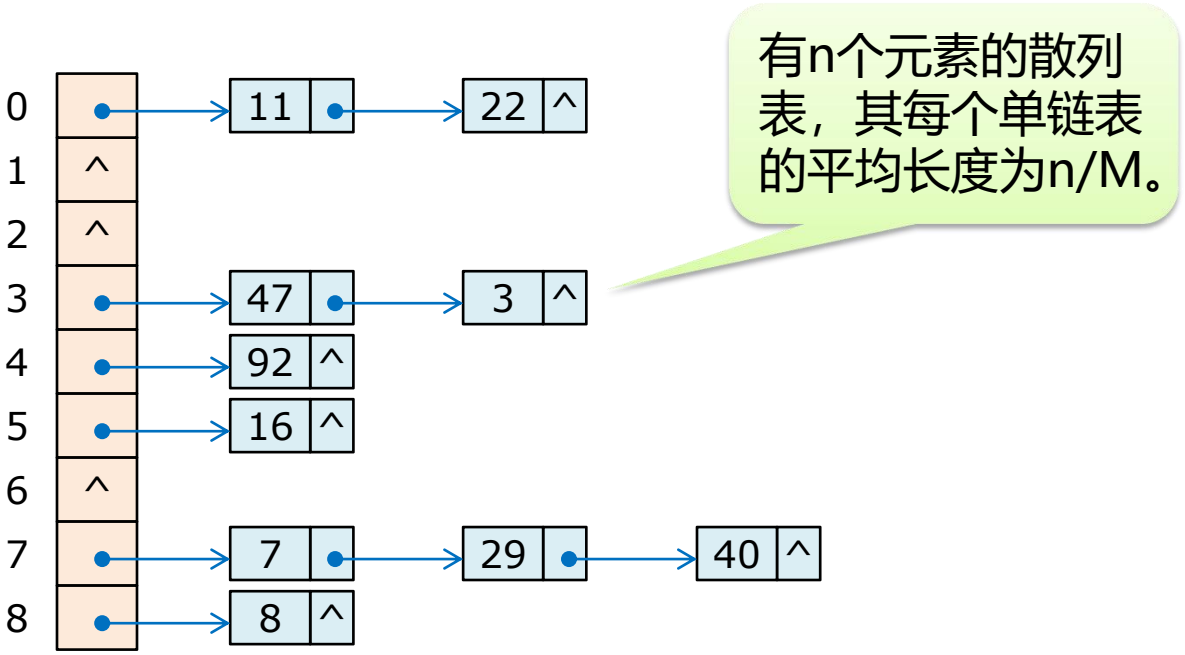
## 链地址法（拉链法）的基本思想

**例8：试将关键字集 A={47，7，29，11，16，92，22，8，3，40} 存入链表中。**

**基本思想：**

1. 所有关键字互为同义词的集合元素都存储在同一个单链上，形成动态结构；
2. 所有单链表的头指针存入一个长度为M的散列表；
3. 单链表的头指针在散列表中的存储位置通过散列函数计算得到。

**h(key)=key % 11**

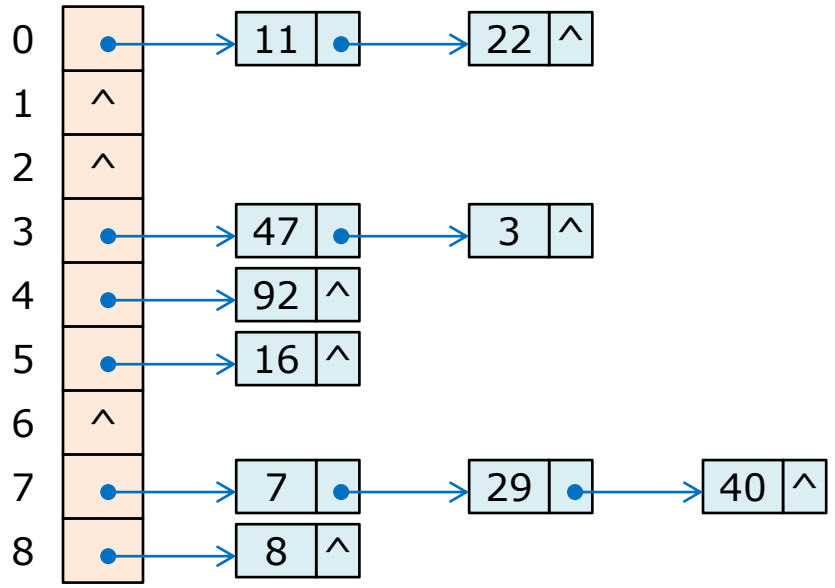


# 链地址法（拉链法）

## 链地址法（拉链法）的基本操作

**例9：在哈希链表中执行查找和插入操作。**

**h(key)=key % 11**

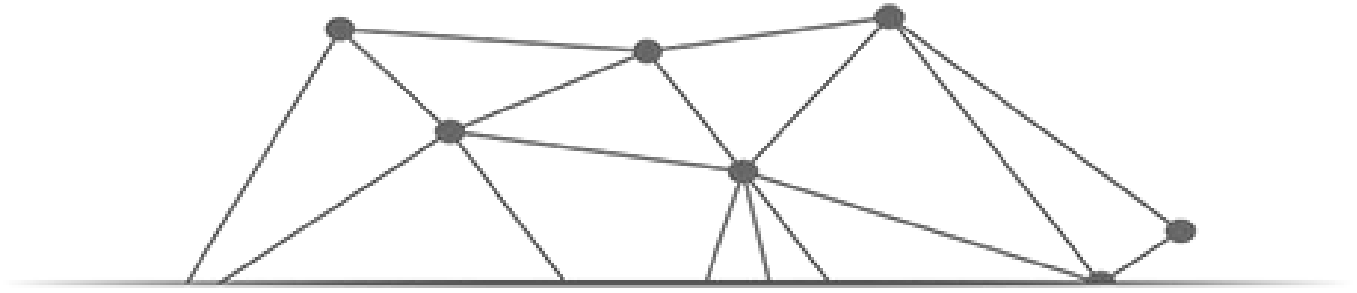



- **集合元素的查找操作**

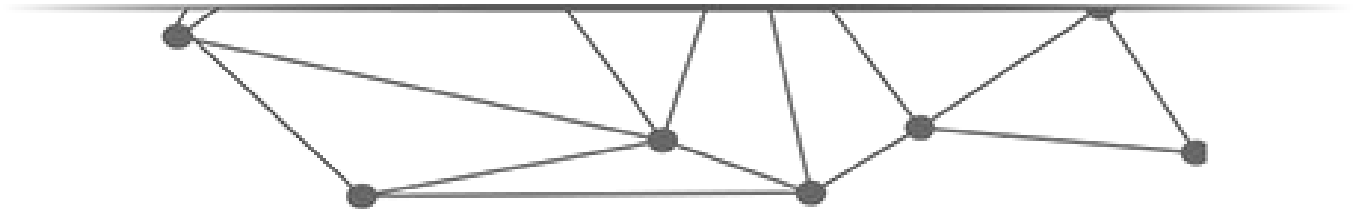
1. 计算散列值h(key)
2. 到散列表h(key)位置处取出单链表头指针
3. 遍历链表进行查找：比较key

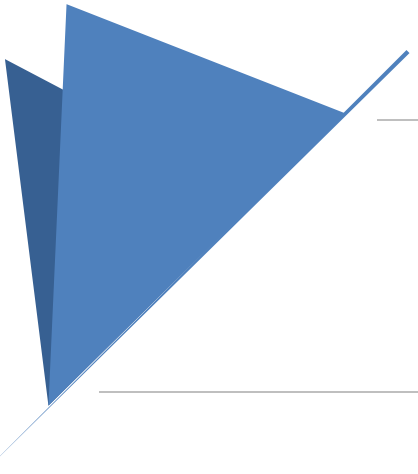
- **集合元素的插入操作**

1. 执行元素查找操作
2. 查找不成功，创建新结点
3. 执行单链表结点插入操作



# 课堂互动





# 散列查找的性能分析

/ 散列查找的基本过程

/ 散列查找的性能分析

# 散列查找的性能分析

## 散列查找的基本过程

- **准备阶段**

1.1 构造哈希函数

1.2 将查询源的数据使用哈希函数转换为哈希值

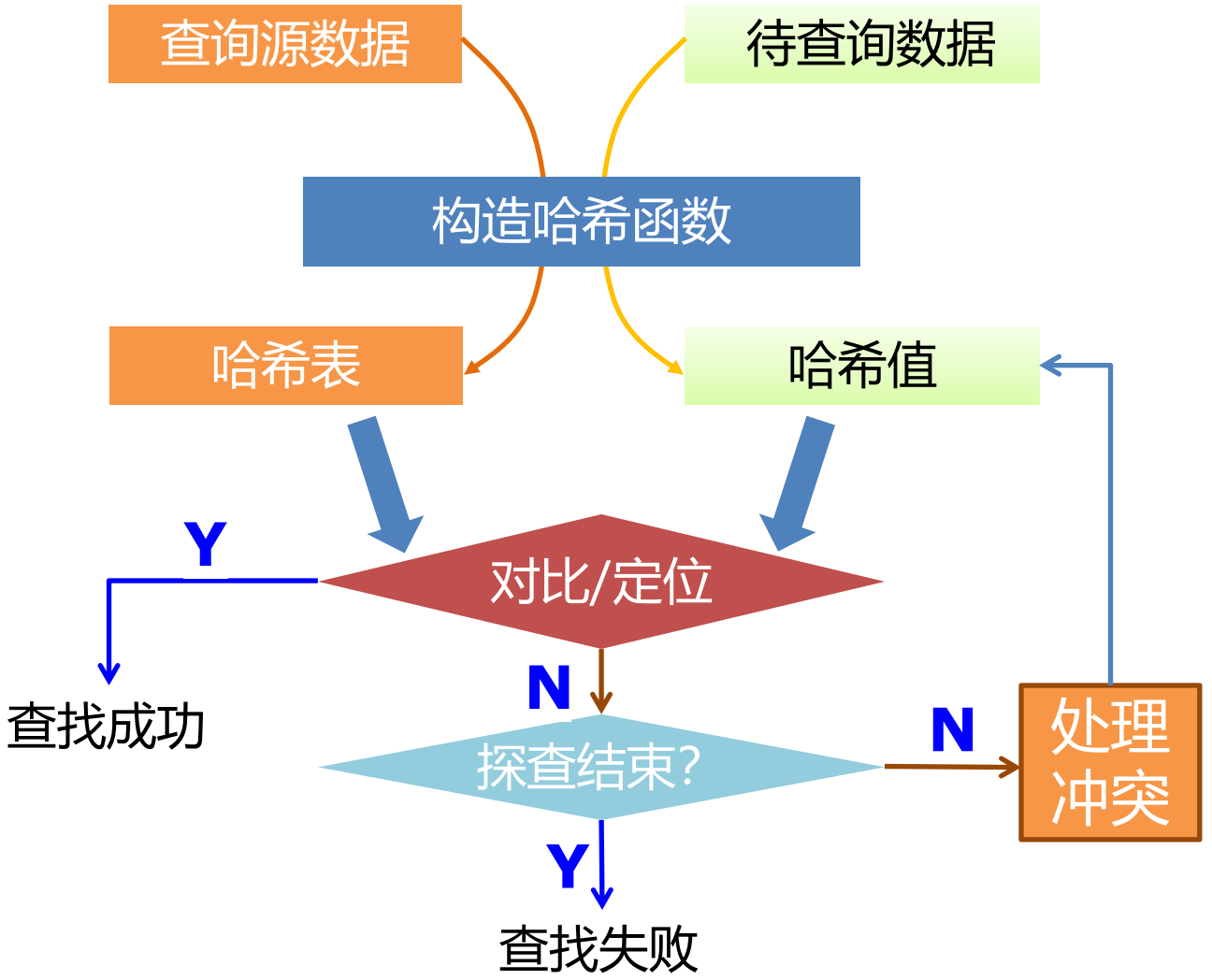
1.3 将所有元素的哈希值组合成哈希表进行存储

- **查询阶段**

2.1 将待查询数据使用哈希函数转换成哈希值

2.2 使用哈希值直接进行定位和对比

2.3 若对比成功则进行输出；若对比失败则进行冲突处理，直至查询成功或探查完整个待查哈希表



# 散列查找的性能分析

例10：已知一组关键字(19,14,23,1,68,20,84,27,55,11,10,79)，试对比线性探查法和拉链法的查找插入效率。设哈希函数为H(key) = key%13，哈希表长为13，并且假设每个记录的查找概率相等。

## 1. 线性探测法，即 $h_{i+1}(key) = h_i(key)+1$

| 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 | | | |

H(19) = 6
H(14) = 1
H(23) = 10
H(1) = 1 冲突
H(1) = H(1)+1=2
H(68) = 3
H(20) = 7

H(84) = 6 冲突
H(84) = H(84) + 1 = 7 冲突
H(84) = H(84) + 1 = 8
H(27) = 1 冲突
H(27) = H(27) + 1 = 2 冲突
H(27) = H(27) + 1 = 3 冲突
H(27) = H(27) + 1 = 4

H(55) = 3 -> 2次冲突
H(11) = 11 ->2次冲突
H(10) = 10
H(79) = 1 ->9次冲突

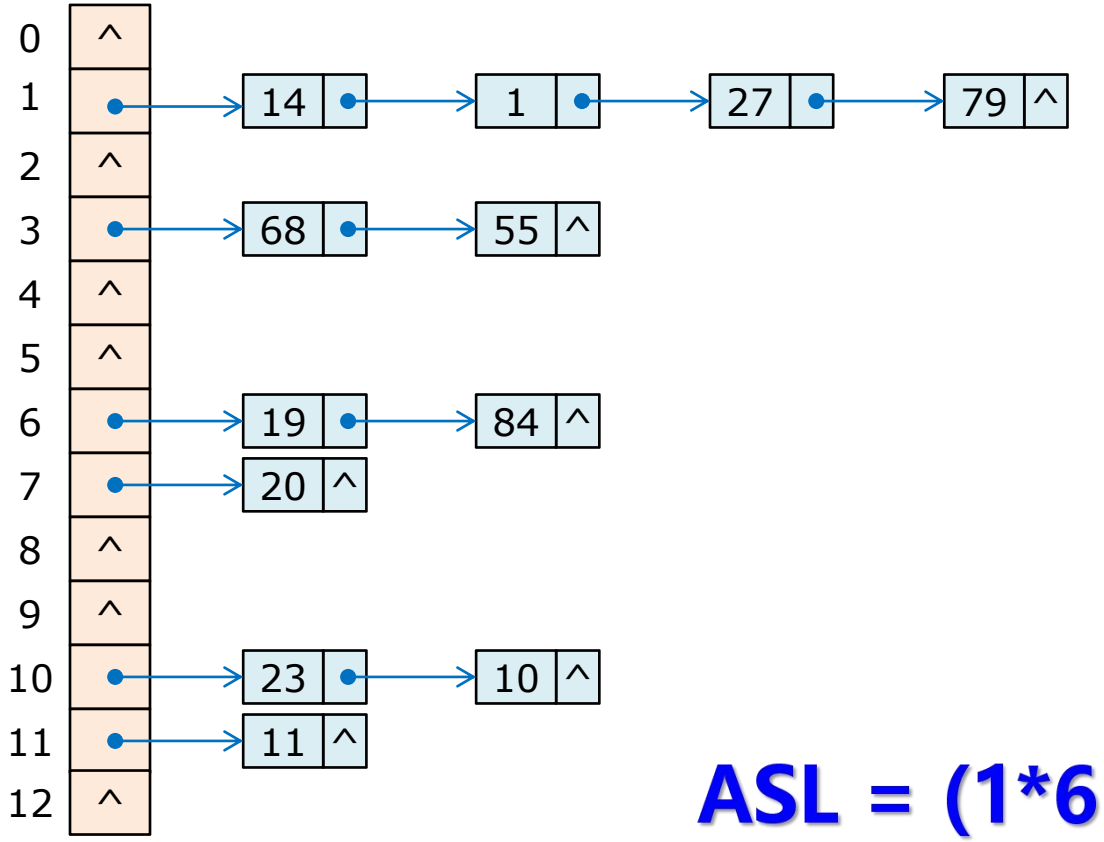
**ASL = (1*6+2+3*3+4+9)/12 = 2.5**

# 散列查找的性能分析

例10：已知一组关键字(19,14,23,1,68,20,84,27,55,11,10,79)，试对比线性探查法和拉链法的查找插入效率。设哈希函数为H(key) = key%13，哈希表长为13，并且假设每个记录的查找概率相等。

## 2. 拉链法

H(19) = 6
H(14) = 1
H(23) = 10
H(1) = 1
H(68) = 3
H(20) = 7
H(84) = 6
H(27) = 1
H(55) = 3
H(11) = 11
H(10) = 10
H(79) = 1




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

# 散列查找的性能分析

## 哈希表的查找效率分析


$$\alpha = \frac{\text{表中填入的记录数}n}{\text{散列表长度}m}$$

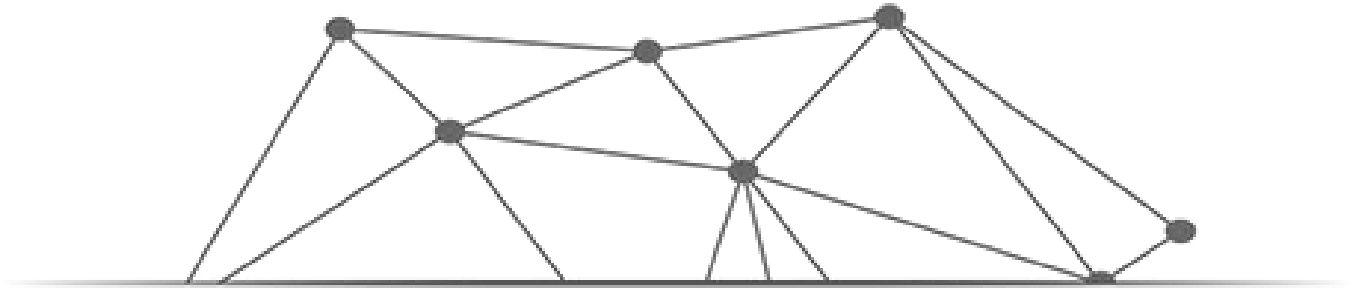
- ✓ $\alpha$越大，记录数越多，发生冲突的可能性越大，查找时比较次数越多。空间利用率越高。
- ✓ $\alpha$越小，空间利用率越低，发生冲突的可能性越小

# 散列查找的性能分析

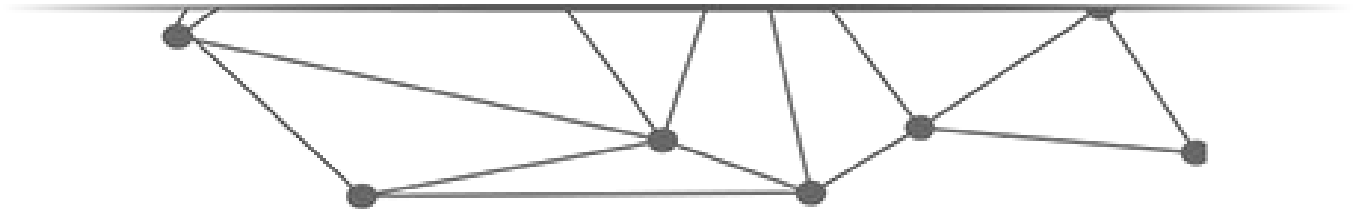
## 哈希表的查找效率分析

ASL与装填因子$\alpha$有关！O(1)≤ASL≤O(n)

| 处理冲突的方法 | *ASL*平均查找长度 | |
|---|---|---|
| | 查找成功 | 查找失败 |
| 拉链法 | $1+\frac{\alpha}{2}$ | $\alpha+e^{-\alpha}$ |
| 线性探测法 | $\frac{1}{2}(1+\frac{1}{1-\alpha})$ | $\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$ |
| 伪随机探测法<br>和二次探测法 | $-\frac{1}{\alpha}\ln(1-\alpha)$ | $\frac{1}{1-\alpha}$ |



# 课堂互动



# 第18讲 散列表

## 小　结

- 散列表是一种线性结构，它不以关键字比较为基础进行查找，而是通过散列函数把记录的关键字和其在表中的位置建立对应关系
- 除留余数法是最常用的哈希函数构建方法
- 散列码的查找长度与记录总数无关，可以通过调节装填因子来控制平均查找长度ASL
- 散列表采用专门的冲突处理方法解决哈希码冲突时的地址映射问题

| 比较项目 | | 处理方法 | |
|---|---|---|---|
| | | 开放地址法（线性探测） | 链地址法（拉链法） |
| 空间 | | 无指针域，存储效率较高 | 附加指针域，存储效率较低 |
| 时间 | 查找 | 有二次聚集现象，查找效率较低 | 无二次聚集现象，查找效率较高 |
| | 插入和删除 | 不易实现 | 易于实现 |
| 适用情况 | | 表的大小固定，<br>适于表长无变化的情况 | 结点动态生成，<br>适于表长经常变化的情况 |

# 欧老师的联系方式

读万卷书 行万里路 只为最好的修炼


QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Website: http://ouxinyu.cn

Tel：18687840023

地址：云南财经大学南院3楼 智能应用研究院A306-2