

# 11C 排解冲突(1)

#数据结构邓神

## 一山二虎 排解冲突 | 预案

### multiple slots



但是会不会一个槽位溢出了怎么办?

❖ 只要槽位数目不多

依然可以保证  $O(1)$  的时间效率

❖ 但是, 需要为每个桶配备多少个槽, 方能保证  $O(1)$ ?

//难以预测

预留过多, 空间浪费 ✓

无论预留多少, 极端情况下仍有可能不够

## 泾渭分明 | 独立链条

### 改用链表

```
vector<list<elemType>> hash
```

但是这些都是没有办法优化散列函数的解决办法

最好的办法是竭尽全力优化散列函数, 尽可能避免重复, 提高均匀度, 避免重复

## 独立链

❖ linked-list chaining / separate chaining

每个桶存放一个指针

冲突的词条，组织成列表

❖ 优点 ✓ 无需为每个桶预备多个槽位

✓ 任意多次的冲突都可解决

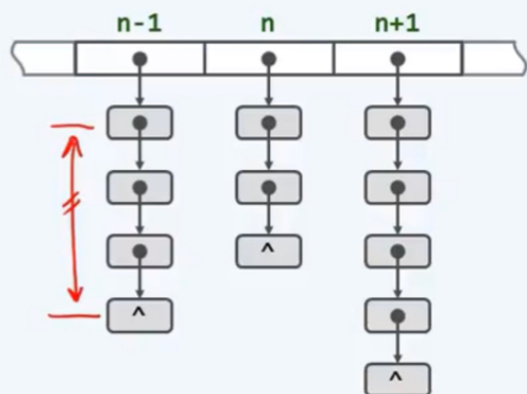
✓ 删除操作实现简单、统一

LIST

❖ 但是 指针需要额外空间

节点需要动态申请

更重要的是...



动态申请内存空间的时间效率要高出两个数量级

最大缺陷：空间未必连续分布，系统缓存失去效果

(系统无法预测你的访问方向，无法通过缓存加速你的访问)

我们如何充分利用缓存呢

开放定址 open addressing

独立链表法：closed addressing

## 开放定址

❖ open addressing ~ closed hashing

为每个桶都事先约定若干备用桶

它们构成一个查找链 probing sequence/chain

不用申请额外的空间，尝试使用系统缓存加速

线性试探

这种方法看起来慢，但是线性访问可以充分利用系统缓存加速！

## 线性试探

❖ Linear probing 一旦冲突，则试探后一紧邻桶单元；

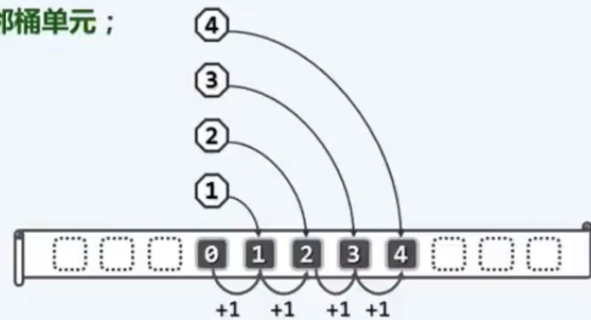
$[\text{hash}(\text{key}) + 1] \% M$

$[\text{hash}(\text{key}) + 2] \% M$

$[\text{hash}(\text{key}) + 3] \% M$

...

直到命中成功，或抵达空桶失败



❖ 优点：无需附加的（指针、链表或溢出区等）空间

查找链具有局部性，可充分利用系统缓存，有效减少I/O

❖ 但是：操作时间  $> O(1)$

冲突增多——以往的冲突，会导致后续的冲突 clustering

有可能存在一大片连续而都被使用的单元，导致插入单元被阻挡半天

反过来查找也可能导致这种情况，而且这种情况会愈演愈烈，情况会不断劣化。

## 懒惰删除

因为一旦查找到空桶就会立刻停止，所以其中不能有任何空隙

那么如何删除词条

### 懒惰删除

❖ 按照开放定址策略：先后插入、相互冲突的一组词条，将存放于同一查找链中

❖ 若需删除其中某一词条，应如何实现？

❖ 直接删除：清除词条，回收空桶？

问题：查找链被切断，后续词条将丢失——明明存在，却访问不到

❖ lazy removal：仅做删除标记，查找链不必续接