9.词典

(d1) 散列：排解冲突(1)

Every mistake I've ever made

Has been rehashed and then replayed

As I got lost along the way.

邓俊辉

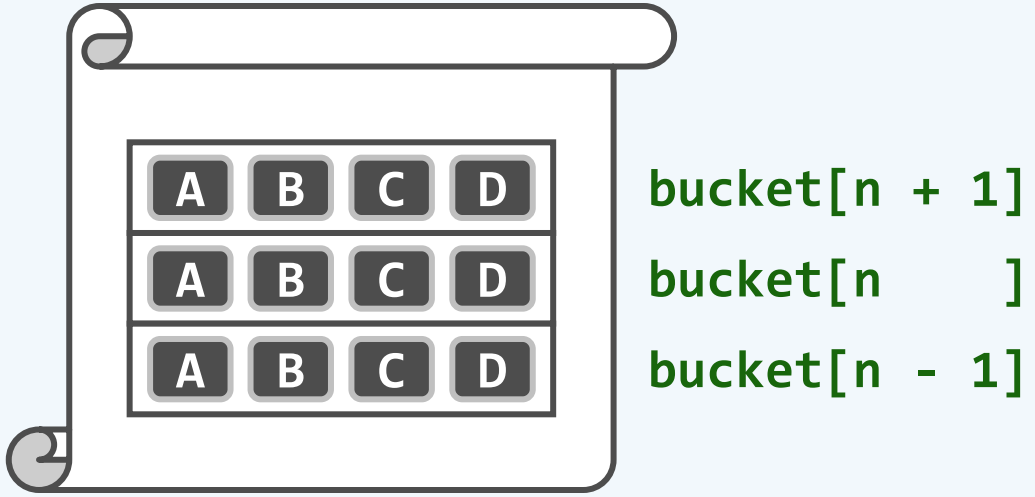
deng@tsinghua.edu.cn

多槽位

❖ multiple slots

桶单元细分成若干槽位slot

存放（与同一单元）冲突的词条



❖ 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

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

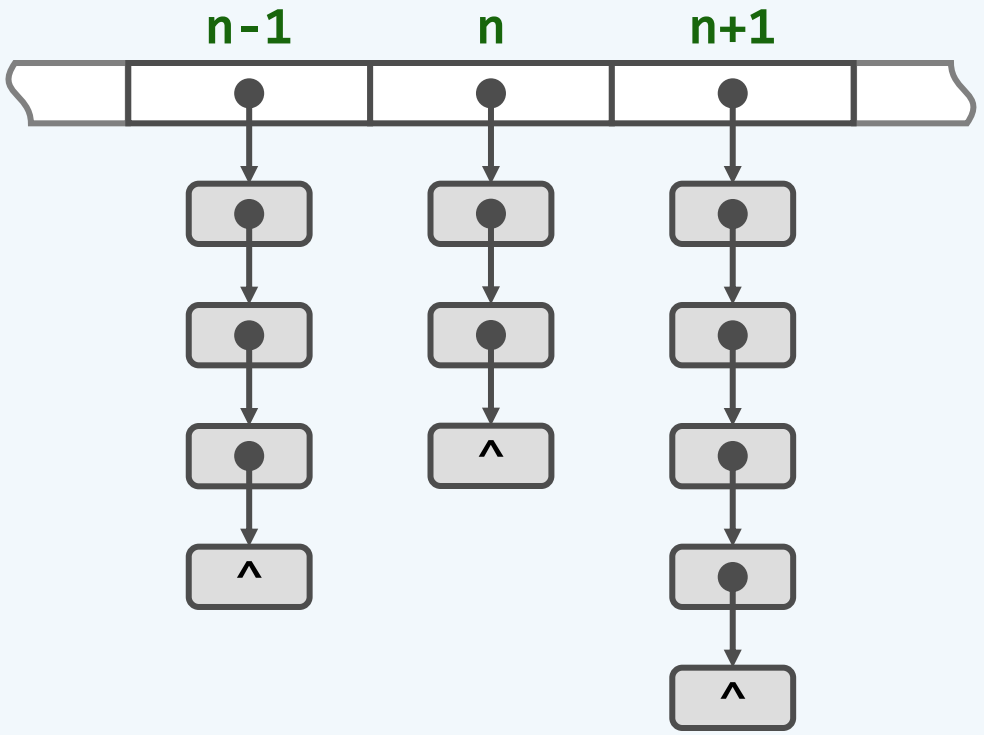
删除操作实现简单、统一

❖ 但是　指针需要额外空间

节点需要动态申请

更重要的是...

❖ 空间未必连续分布，系统缓存几乎失效



公共溢出区

- overflow area

  单独开辟一块连续空间

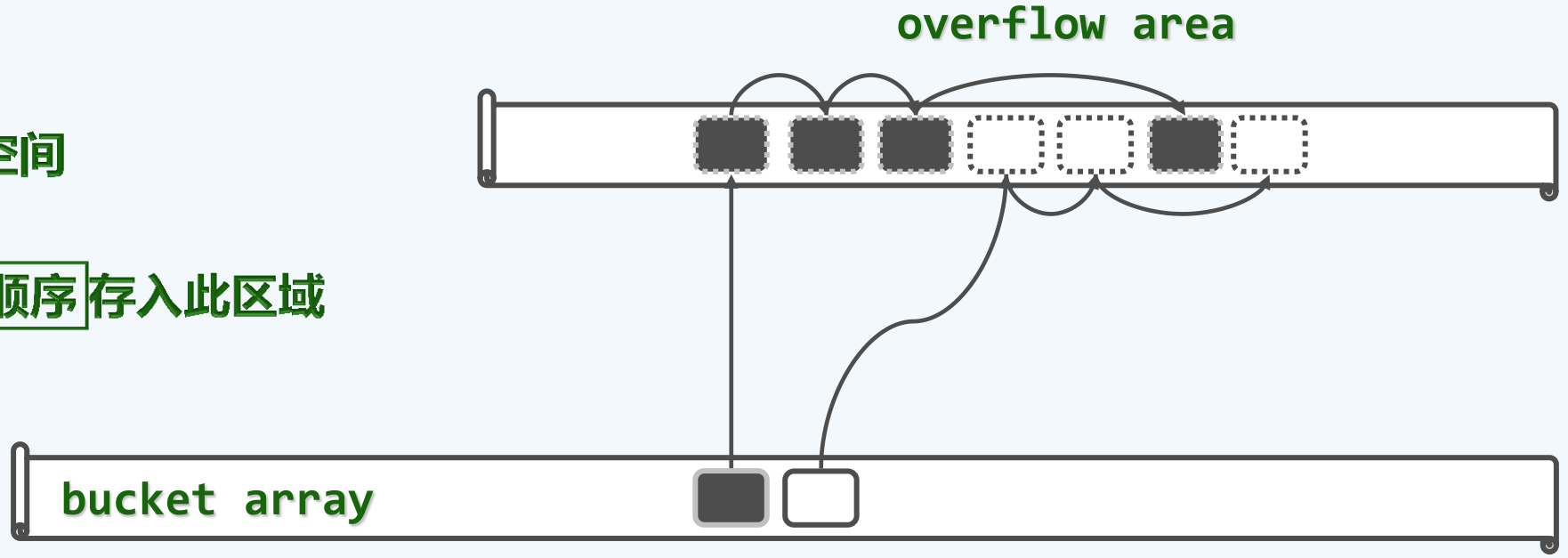
  发生冲突的词条，顺序存入此区域

- 结构简单

  算法易于实现

- 但是，不冲突则已，一旦发生冲突

  最坏情况下，处理冲突词条所需的时间正比于溢出区的规模



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

- 查找： 沿查找链，逐个转向下一桶单元，直到...

  命中成功，或者

  抵达一个空桶（已遍历所有冲突的词条）失败

- 具体地，查找链应如何约定？

开放定址

- 插入： 新词条若尚不存在，则存入查找终止处的空桶
- 删除： 简单地清除命中的桶？

  某条查找链可能因此被切断...

  多条查找链可能因此被切断...

- 优点： 结构本身保持简洁

  在散列表内部解决冲突，无需附加空间

- 缺点： 冲突之后，可能引发本可避免的新冲突 //举个例子吧...

线性试探

❖ Linear probing 一旦冲突，则试探后一紧邻桶单元；

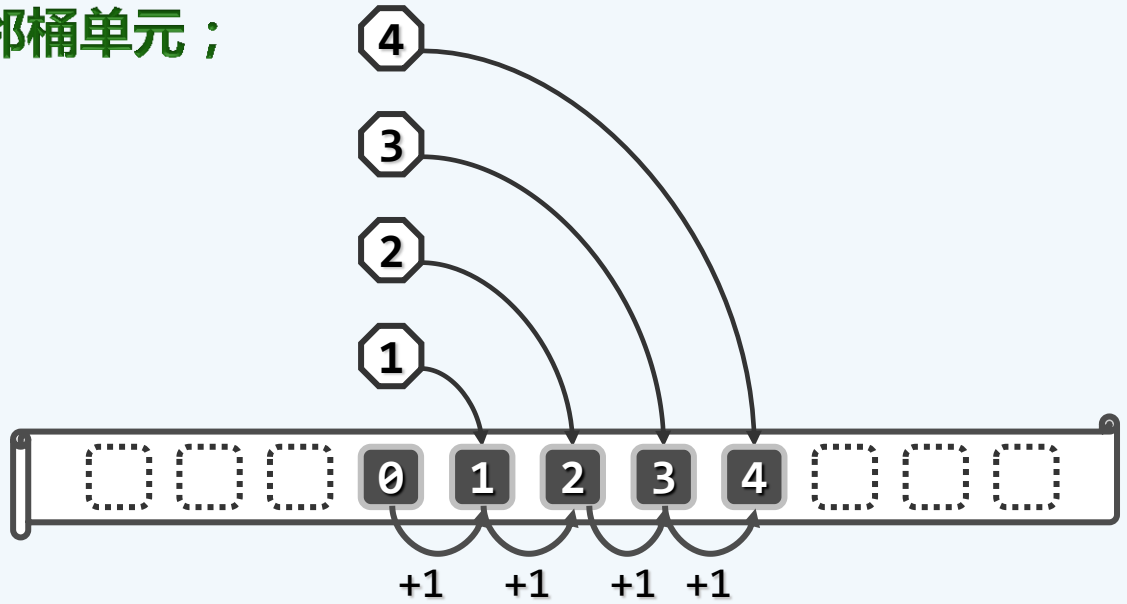
[hash(key) + 1] % M

[hash(key) + 2] % M

[hash(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

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
- 此后，带有删除标记的桶所扮演的角色，因具体的操作类型而异

  1）查找词条时，被视作“必不匹配的非空桶”，查找链在此得以延续

  2）插入词条时，被视作“必然匹配的空闲桶”，可以用来存放新词条

  具体过程的实现...

懒惰删除

```
❖template <typename K, typename V> int Hashtable<K, V>::probe4Hit(const K& k) {
    int r = hashCode(k) % M; //从首个桶起沿查找链，跳过所有冲突的和被懒惰删除的桶
    while ( ht[r] && ( k != ht[r]->key ) || !ht[r] && lazilyRemoved(r) )
        r = ( r + 1 ) % M; //线性试探（注意并列判断的次序，命中可能性更大者前置）
    return r; //调用者根据ht[r]是否为空，即可判断查找是否成功
  }

❖template <typename K, typename V> int Hashtable<K, V>::probe4Free(const K& k) {
    int r = hashCode(k) % M; //从首个桶起
    while ( ht[r] ) r = (r + 1) % M; //沿查找链找到第一个空桶（无论是否懒惰删除）
    return r; //调用者根据ht[r]是否为空，即可判断查找是否成功
  }
```