词典

本质

按值寻找 call by value.

关键码不需要比较大小，只需要能够比较是否相同。

散列

本质 / 原理

两项基本任务：

- 精心设计散列表，尽可能降低冲突的概率。
- 制定可行的预案，尽快排解冲突。

散列表

散列方法的底层基础，逻辑上是一系列存储单元（称为桶）组成，在物理空间上也是相邻按顺序排布的。

值域：$\mathcal{R}$，可能用到事实上是 $\mathcal{N}$；容量：$\mathcal{M}$；

这三者满足 $\mathcal{N} < \mathcal{M} \ll \mathcal{R}$

装填因子 / 空间利用率：$\lambda = \mathcal{N}/\mathcal{M}$

散列函数

本质：`hash() : key -> &entry`

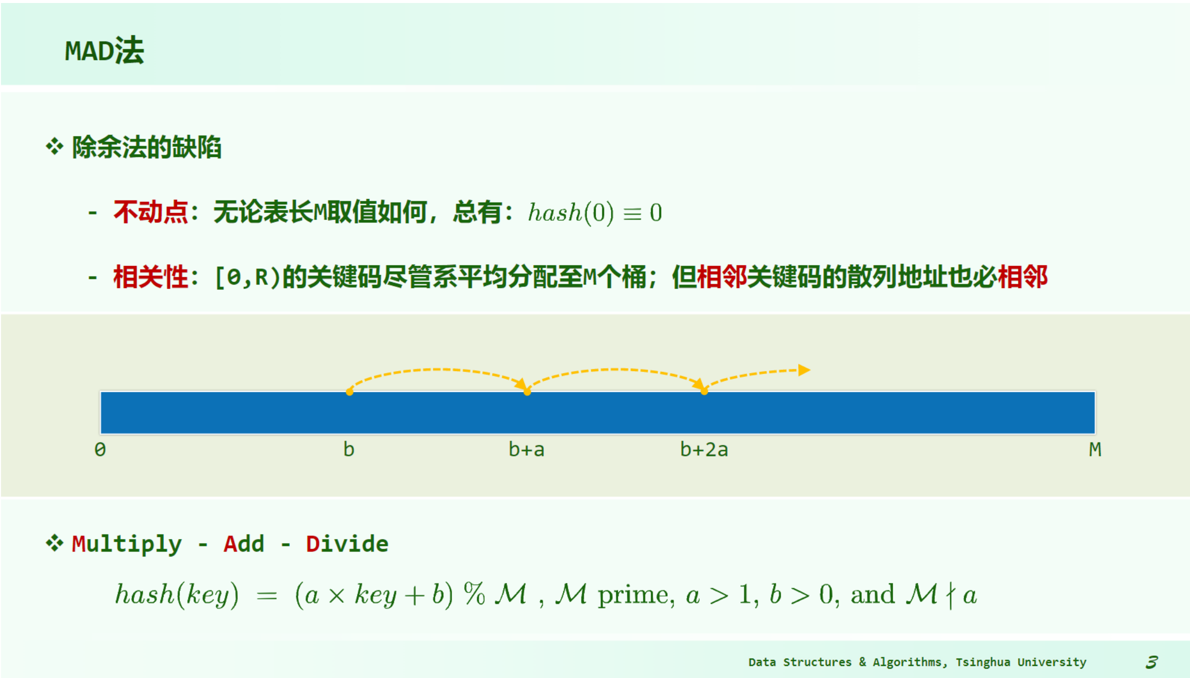
理论上应该是 expected $O(1)$，并不是 $O(1)$

评价方法：确定、快速、满射、均匀（避免 clustering）。

除余法

`hash(key) = key % M`

缺点：不动点、相关性，因此引入MAD法



MAD法

Multiply Add Divide

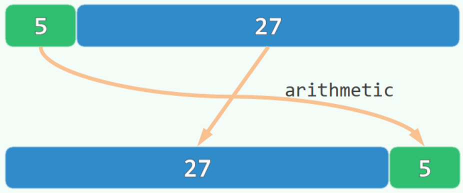
$\text{hash}(key) = (a \times \text{key} + b)\%\mathcal{M}$ $\mathcal{M}$ prime, $a > 1, b > 0,$ and $M \nmid a$

其中M和a互质很重要，$g = gcd(M, a)$，则散列表利用率只为$\frac{1}{g}$，也就是每个关键码都大约和g个关键码冲突（习题9-6）

多项式法

String/Object To Integer

```
static Rank hashCode( char s[] ) {
   Rank n = strlen(s); Rank h = 0;
   for ( Rank i = 0; i < n; i++ ) {
      h = (h << 5) | (h >> 27);
      h += s[i];
   } //乘以32，加上扰动，累计贡献
   return h;
} //有必要如此复杂吗？能否使用更简单的散列，比如...
```



$$
\begin{aligned}
& hashCode(\text{“}x_{n-1}\ldots x_3x_2x_1x_0\text{”}) \\
= \ & x_{n-1}\cdot a^{n-1} \ + \ \ldots \ + \ x_2\cdot a^2 \ + \ x_1\cdot a^1 \ + \ x_0 \\
= \ & (\ldots((x_{n-1}\cdot a \ + \ x_{n-2})\cdot a) \ + \ \ldots \ + \ x_1)\cdot a \ + \ x_0
\end{aligned}
$$

乱七八糟的散列方法

数字分析

平方取中

folding

xor

伪随机数法

散列冲突排解

开放散列

多槽位（multiple slots）

很多个槽位。浪费空间，极端情况下还是不够。

❖ Multiple Slots

- 桶单元细分成若干**槽位**
- 存放（与同一单元）冲突的词条

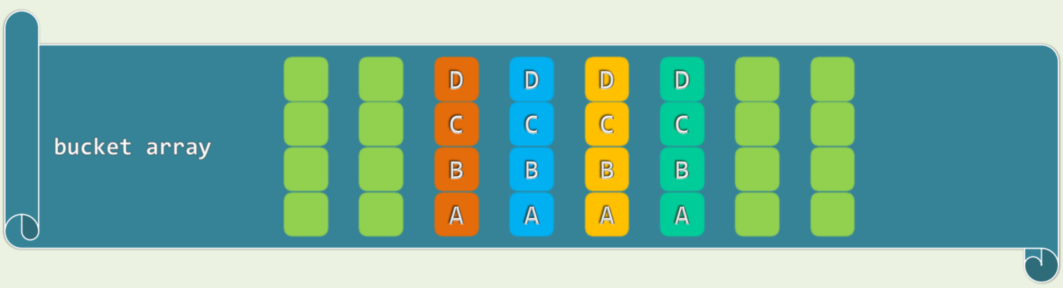
❖ 只要槽位数目不太多

依然可以保证$\mathcal{O}(1)$的时间效率

❖ 但是，究竟需要细分到什么程度？

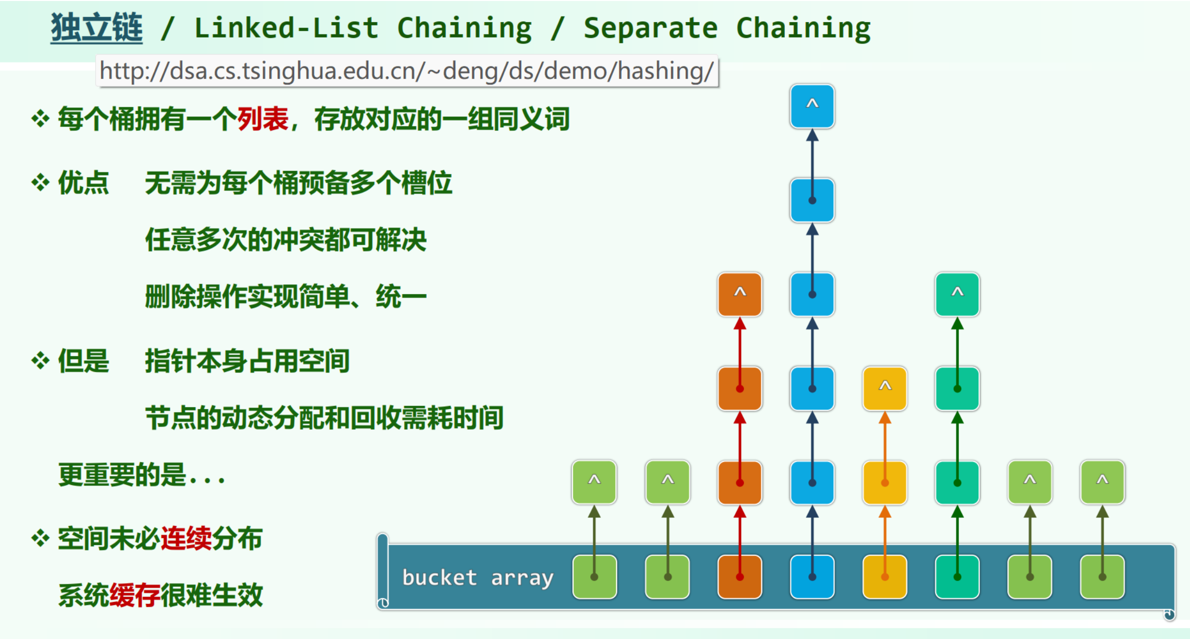
难以预测！

- 过细，空间**浪费**；反过来
- 无论多细，极端情况下仍可能**不够**



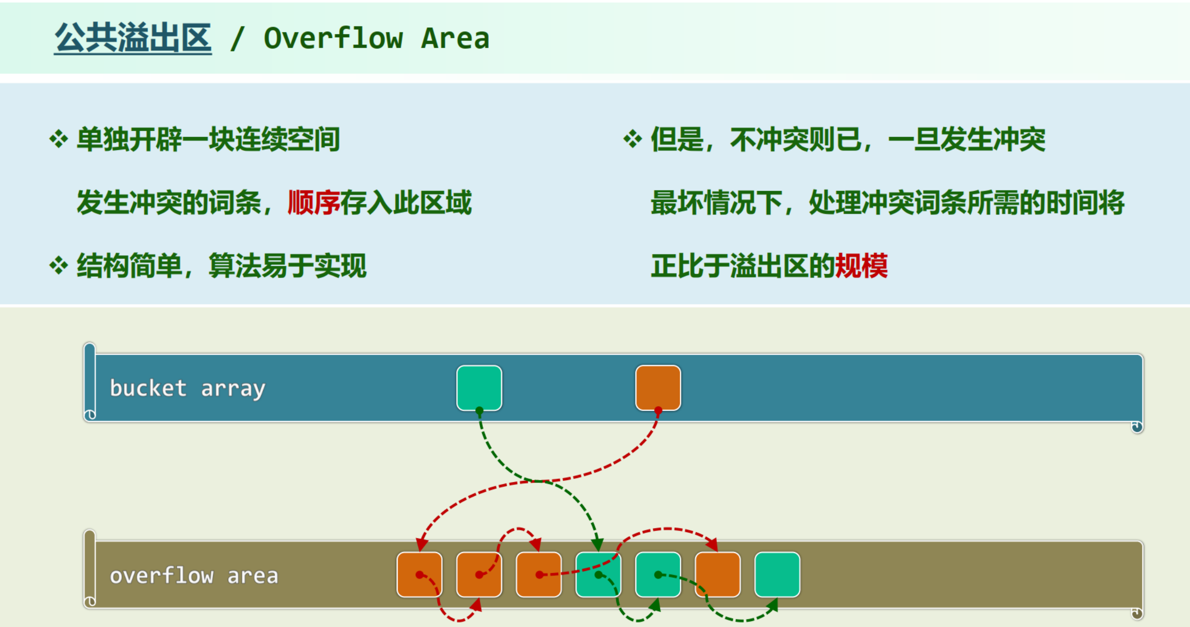
独立链（separate chaining）

每个桶拥有一个列表。但利用不上缓存



公共溢出区（overflow area）

将发生冲突的词条，顺序存入该区域。但时间复杂度可能恶化。



封闭散列

只要有必要，任一散列桶可以接受任意词条。

需要给出词条可能依赖的备用桶，概率/优先级逐次下降。

查找算法：沿着试探链，逐个转向下一个桶单元，直到命中（成功）；或者抵达一个空桶（失败）

线性试探（linear probing）

不断向后一个一个试探。

优势：简洁。一定能找到。局部性良好。

劣势：会堆积。试探链重叠后会更场。

懒惰删除

删除不能直接删除，只能标记一下不存在，在插入的时候再次使用即可。注意对闭散列而言，查找链可能重叠。

平方试探、双向平方试探（quadratic probing）

以平方数为距离确定下一个试探单元。双向平方试探是交替方向以平方数为距离确定下一个试探单元。【只有查找链很长的时候才会产生IO上不局部性的影响。】

能否全部用上？

> 若 $\mathcal{M}$ 是素数，且 $\lambda \leq 0.5$ ，那么平方试探法就一定能找出空桶。（习题9-14）
>
> 反之$\mathcal{M}$非素数，容易在前$\lceil \mathcal{M}/2 \rceil$ 次就循环至起点，但这也并非绝对，只是概率大大提升

再散列 & 重散列

随着装填因子过大，冲突概率和排解难度都将激增，如此，不如集体“搬迁”至更大的散列表。

插入时当$\lambda = \frac{N+L}{M}$达到一定上界时触发重散列，一般设为50%

删除时当$\frac{N}{L}$达到一定上界时触发重散列，一般设为33%

重散列时$\mathcal{M} = 4N$，注意和L无关，所以重散列也可能导致缩容。

应用

桶排序

对 $[0, m)$ 内的 n 个互异整数，借助散列表 $\mathcal{H}$ 做排序。

空间：$O(m)$ 时间：$O(n)$ 。

> 允许重复的时候，就把每一组同义词开一个链表。

最大间隙问题

找到最左侧的节点，最右侧节点；

将有效范围均匀地划分为 $n-1$ 段；

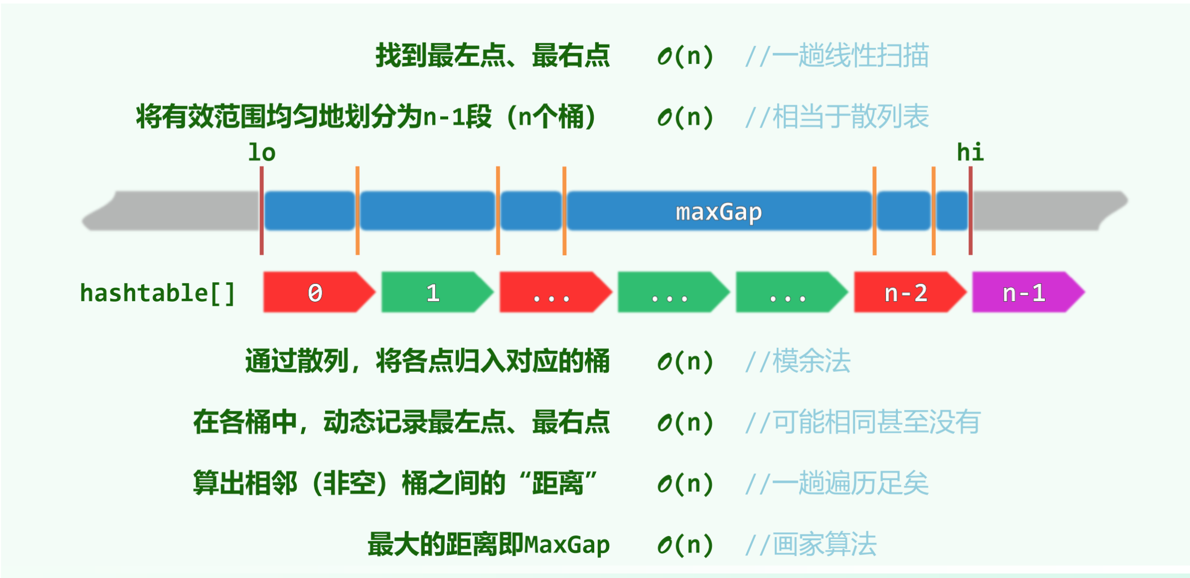
> 最大间隔必然横跨一个段的分割。

通过【散列】（除余法）将各个点归入对应的桶；

在各桶中，动态记录最左点、最右点；

算出相邻（非空）桶之间的距离，最大的距离即 要求的 最大间隙。

> 那 mingap 问题呢？是否有如此的对称性？
>
> > 感觉是没有的，至少我想不出来。



基数排序

低位优先的多次桶排序.

设各字段取值范围为 $[0, M_i), 1 \leq i \leq t$ ，$M = \max\{m_1, \cdots, m_t\}$ ，则时间复杂度为 $O(t \times (n + M))$ 。

如整数排序（常对数密度的整数集：$[0, n^d]$内n个整数。先转为n进制，再做基数排序（d次桶排序）

计数排序

跳转表

William Pugh, 1989.

思想

分层耦合的多个列表。

基本概念：层和塔。

高层列表是低层的子集。

往上生长的策略，每次上面都只留下(随机) 1/2 的底层节点。

期望塔高：2

操作

查找

由高到低，由粗到细。

纵向跳转 ~ 层高（$O(\log n)$）

横向跳转 ~ 紧邻塔顶（不会跑到下一个更高塔那里去）。

$Pr(Y=k)=(1-p)^k p$，$p=1/2$，$E(Y)=(1-p)/p=1$。因此每层期望 $O(1)$ ，横向跳转次数就是 $O(\log n)$

插入

按照 1/2 概率上升即可。

删除

硬删。

复杂度分析

总体空间复杂度是 expected $O(2n)$

单次操作时间复杂度是 expected $O(\log n)$