

0628 go 中 HashMap 的实现

go 中声明并初始化 Hash Map:

```
map := make(map[string]string)
```

key 类型 value 类型

key 的种类可以是: 布尔、数字、string、指针、chan、interface、struct. 只包含上述类型的数组。

不能是: slice, map, function.

只要能支持 == 和 != 操作 就可以做为 key.

HashMap 基本原理: ① hash 函数 ② 如何解决冲突.

go 的实现: ① 在 runtime/algob 中定义了各种基础类的 hash func 与 equal func. 得到 hash 值后用与运算得到桶的编号

② 一般有开放地址法与-拉链法。go 中

用的是拉链法。

桶的结构 bmap 每个桶可以存 8 个 k-v 对.

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	tophash, 存 hash 的前几位
k_1		k_2		k_3		k_4		
k_5		k_6		k_7		k_8		
v_1		v_2		v_3		v_4		
v_5		v_6		v_7		v_8		

overflow * bmap 溢出桶 (8 个装不下可以再链一个桶)

常规桶的个数为 2^B , 一定是 2 的幂次。hash 值与 $(2^B - 1)$ 与运算后, 定位桶编号。与运算与取模效果一致, 但更快, 需要与 2 的幂次才可以。

如果 $B \geq 4$, 溢出桶的个数为 2^{B-4} , 否则无溢出桶
预分配

hmap {

count int

flags uint8

B uint8

noverflow uint8

hasho uint32

buckets unsafe.Pointer

old buckets unsafe.Pointer

nevacuate

uintptr (接下来迁移的桶)

extra

*mapextra

{ overflow *[]*bmap
old overflow *[]*bmap
next overflow *bmap

}

扩容规则

$$\frac{\text{count} / (2^B)}{\text{load factor}} > 6.5 \rightarrow \text{翻倍扩容}$$

$$\text{load factor} \leq 6.5 \rightarrow \text{等量扩容}$$

overflow 较多

$$B \leq 15, \text{ overflow} \geq 2^B$$

$$B > 15, \text{ overflow} \geq 2^{15}$$

扩容不是一步到位, 是一个个扩容, 逐步分推到多次哈希表操作

中: 渐进式扩容, 可以避免一次性扩容带来的性能

瞬时抖动。

06.27 红黑树, B树和B+树.

红黑树是二叉树, 大规模数据存储时红黑树很深, 所需IO很多, 效率低下, 对存储设备寿命的影响也很大。因此, 我们需要使用多叉的搜索树, 来降低树的高度, 提高效率。

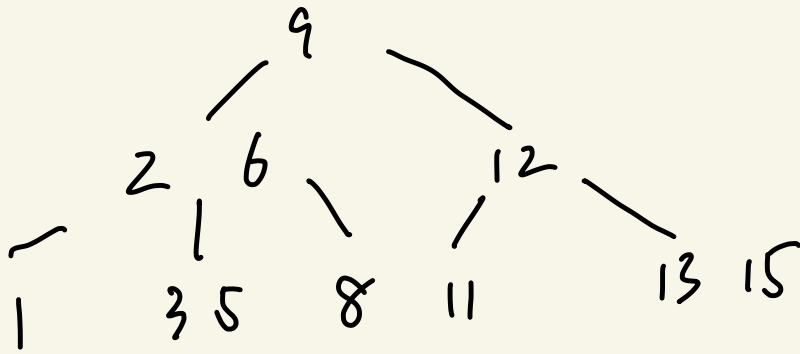
B树: m 叉的搜索树, m 称为B树的阶, m 取决于磁盘页的大小。 $m > 2$ 。

~~根结点的孩子树: $[2, m]$~~

~~非根非叶结点的孩子数: $[m/2, m]$~~

非叶结点的关键字个数 (k -val): 孩子数 - 1。

k 个关键字个数把节点拆成 $k+1$ 段, 指向 $k+1$ 个孩子,



所有叶子节点位于同一层。

与B树不同的点: 关键字分布在整棵树

任何关键字只出现一次

搜索可能停止在非叶节点

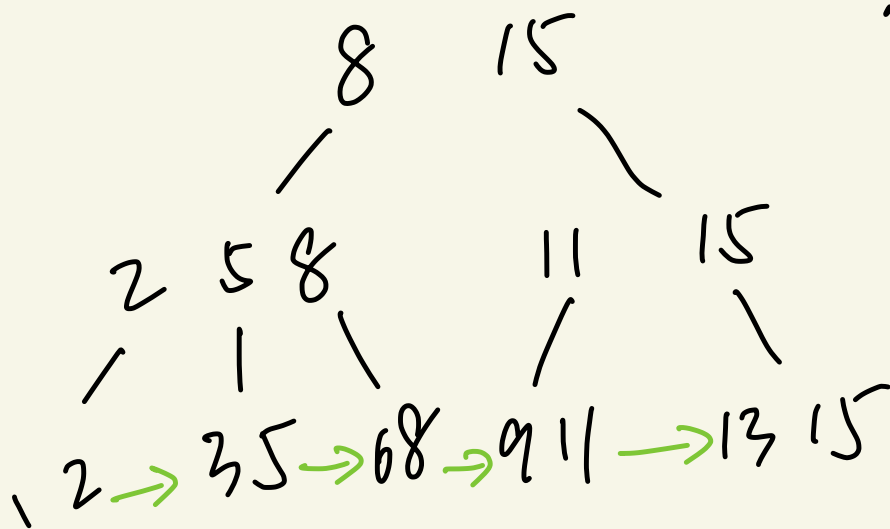
B+树: 一个节点 k 个关键字, 拆成 k 段, 指向 k 个孩子
(B树会差1), ^{根结点至少}

关键字是子树的最大元素, 一些关键字是出现多次的.

非叶节点不存储 V , 只作索引

叶节点才存 $k-V$

所有叶节点用链表相连, range-query 方便



B树B个数, 关键字个数:

B: 每个节点最多有 m 个孩子; 根节点最少 2 个孩子, 其他非叶节点最少 $\text{ceil}(m/2)$ 个孩子。非叶节点: 关键字个数 = 孩子数 - 1。

B+树: 孩子数的情况和B树一样, 关键字数 = 孩子数

B+树优点: 非叶节点不存储 V 的信息, 进一步压缩空间, m 可以进一步变大。range-query 方便 (B树只能中序便利)

B树: 可能可以不到叶节点就找到更快。 (B+树只能到叶节点找到 V , 但是搜索性能更稳定)。

06.26 python 内存管理特点

1. 引用计数

python 内部记录了对象有多少个引用。对象创建时会创建一个引用计数, 当引用计数为 0, 会被

GC.

① 分配新名称/放入容器, 引用计数增加

② del / 引用超出作用域 / 重新赋值 引用计数减少

③ `sys.getrefcount()` 获取引用计数

2. GC: 检查引用计数为 0 的对象清除内存空间

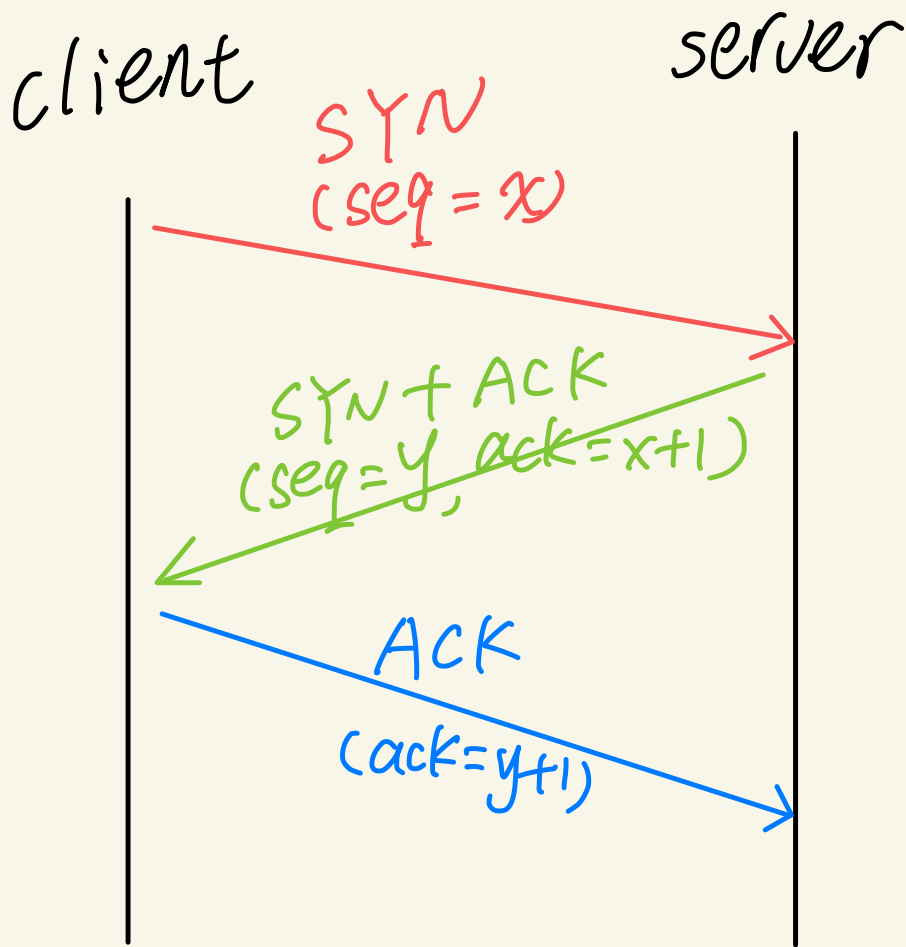
3. 内存池: 很多小内存申请释放频繁, 会影响效率。引入内存池机制, 用于管理小块内存, 能够减少内存碎片。

① GC 将内存返还给内存池而非操作系统

② 小于 256B 的对象使用 `Pymalloc` 分配器, 大的使用系统 `malloc`

③ `Int`, `Float`, `List` 等之间不共享内存池

06.23 TCP 3-way handshake process



TCP uses the 3-way handshake process to establish a secure and reliable communication between the client and server.

① client sends the SYN to the server with SYN Flag as 1, means client wants to establish a secure connection with the server. $seq = x$, which is a random number, assigned to the first bit of the data.

② Server receives the SYN from client, sends SYN + ACK to the client, with SYN and ACK Flag as 1. ack number is set as $x+1$, tell client that the server has successfully receives the previous SYN from the client. Another seq is set as y which is a random number, assigned to the first bit of the data.

③ Client receives the SYN+ACK from the server, send an ACK to the server, with ack number set as $y+1$.

After the 3-way handshake the TCP connection is established.

06.22

oov features,

06.21 Describe what happens when you click on a URL in a browser

- ① You type the URL, for example, `www.leetcode.com` into the browser.
- ② The browser, will contact the DNS (domain name system), to convert the human-readable URL into the numerical IP address.
- ③ With the IP address, the browser can send an HTTP request to the server.
- ④ If the server receives the HTTP request from the browser, it will parse

the request, and send an HTTP response to the browser.

⑤ After receiving the response, the browser will render the HTML encoded in the response. If there are additional resources embedden in the HTML, steps 3~5 will be repeated.

4 layer cache:

browser, os, router, ISP

recursively request the IP address

Root DNS

Top-level

.com

Second-level

microsoft.com

Third-level

download.microsoft.com

TCP connection, 3-way handshake

