书籍推荐

2024年5月6日 9:26

数据结构和算法

入门：《图解算法》，Hello算法（K神）

中级：《算法导论》

- 数学（概率论）
- 伪代码

（证明 → 分析算法）

《Algorithms Illustrated》四卷（强推）

算法详解（一，二）

— 伪代码

— 分析

高级：《TAOCP》"计算机编程艺术" —— 高德纳 → TeX

（七卷）（四卷）

图灵奖

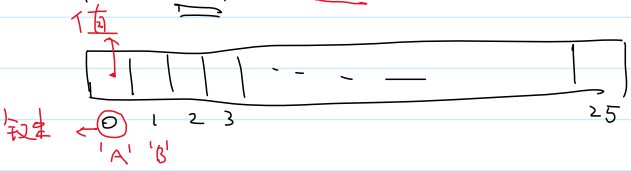
哈希表（~~xxxxxxx~~）

2024年5月6日 9:40

CPU 1% 哈希表的运算

#1. 引入（为什么需要哈希表）

Q：统计一个文件中，字母出现的次数（不区分大小写）。



'A'：42 → 键值对 key-value

'B'：18

⋮

'Z'：22

限制：① 键的取值范围很小。

② 键可以很容易地转换成数组的下标。

key-value：

单词：释义

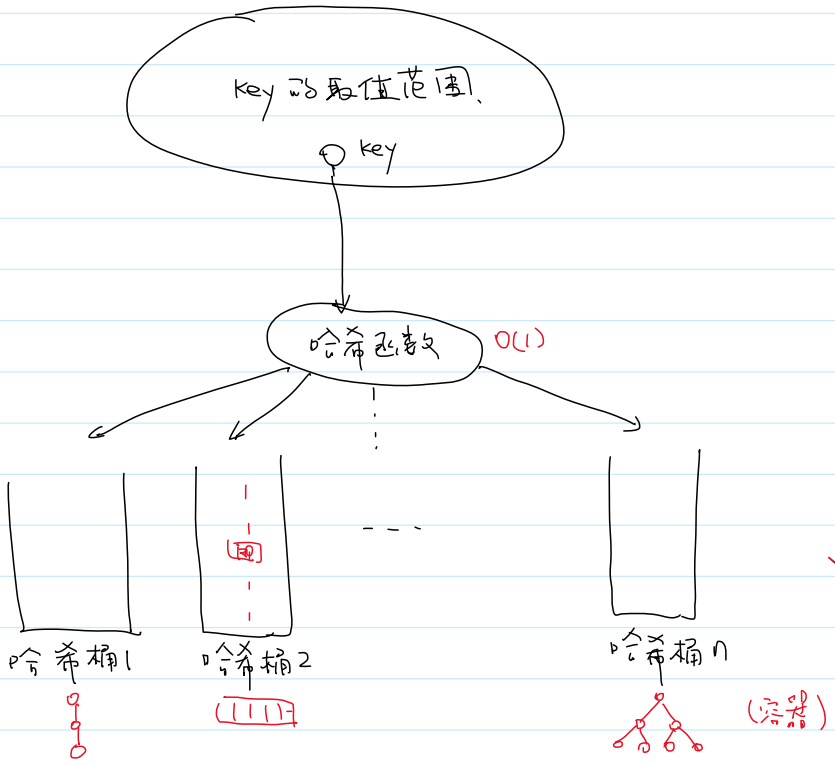
账号：账号信息

关键字：一串相关网页

⋮

核心问题：如果不满上述两个限制条件，该如何表示 key-val 数据 ⇒ 哈希表！

#2. 模型



同一个哈希表，哈希桶的数据结构可以不一样。

key 是唯一的！

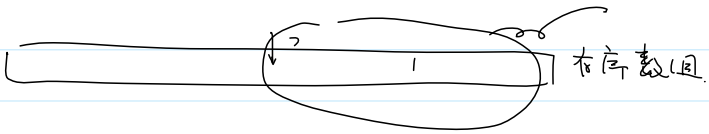
#3. 基本操作.

增 put(key, val);

删 delete(key)

查 val = get(key)

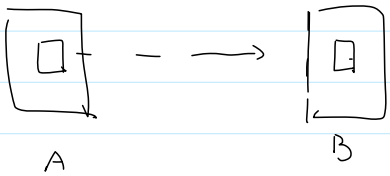
遍历. 依次遍历每一个哈希桶.



#4. 实现.

A. 哈希函数 (数据的指纹)

← 性能非常好的哈希函数.



md5 → md5码(256bits) → 指纹

md5 → md5码 (256bits) → 指纹

性能非常好的哈希函数:

哈希表
① 速度快
② 哈希值平均分布.

安全
③ 哈希碰撞的概率低 $\frac{1}{2^{256}}$
(data1, data2)
④ 对数据非常敏感
(data1, data2)
⑤ 逆向非常困难
hash值 ✗→ data
md5码 → data

定义域 ∞

完美的"圆"

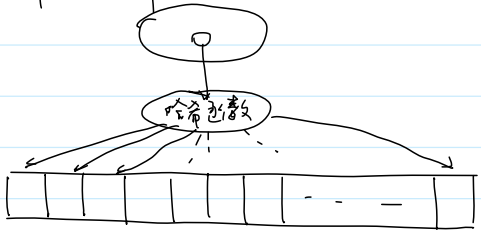
模拟、等概率随机映射 data

256-bits 值域 (2^{256})

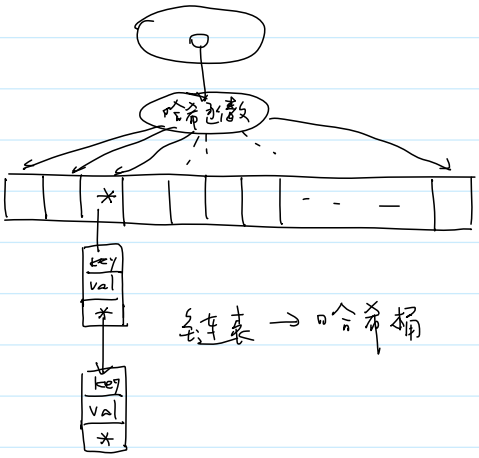
算法 → 固定的步骤

↘ 有规律的

B. 哈希桶 (解决哈希冲突)



拉链法.

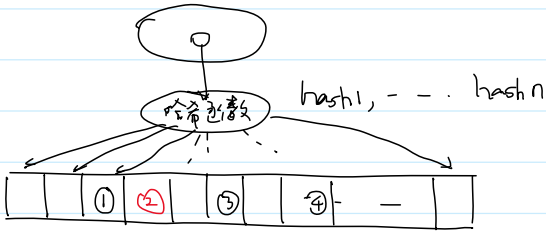


链表 → 哈希桶

C++: unordered_map

Java: HashMap

开放地址法:



线性探测:

平方探测:

再散列法.

哈希桶 → "逻辑结构"

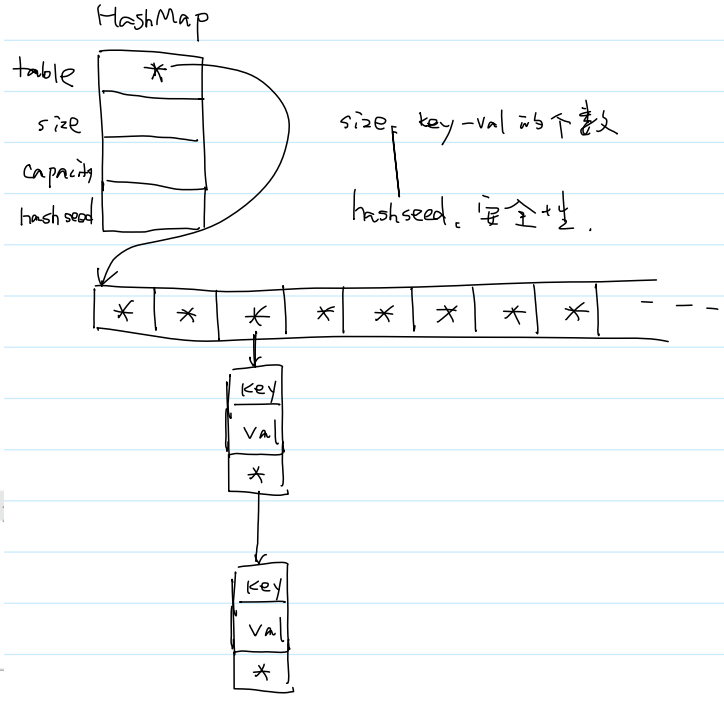
```
// HashMap.h
typedef char* K;
typedef char* V;

typedef struct node {
    K key;
    V val;
    struct node* next;
} Node;

typedef struct {
    Node** table;
    int size;
    int capacity;
    uint32_t hashseed;
} HashMap;

HashMap* hashmap_create();
void hashmap_destroy(HashMap* map);

V hashmap_put(HashMap* map, K key, V val);
V hashmap_get(HashMap* map, K key);
void hashmap_delete(HashMap* map, K key);
```



#5. 分析哈希表的性能 L: 链表的平均长度 $L = \frac{size}{capacity}$

#5 分析哈希表的性能　　L：链表的平均长度　　$L = \frac{size}{capacity}$

不超过一个常数（$L \le 0.75$）

加载因子 0.75（Load Factor）

get　O(L)

put　O(L)　扩容

delete　O(L)

哈希表：用空间换时间

#6 应用

存储键值对数据

Redis（C语言）→ 内存数据库 ⟶（缓存）

（键值对数据库）

↓

大量使用哈希表

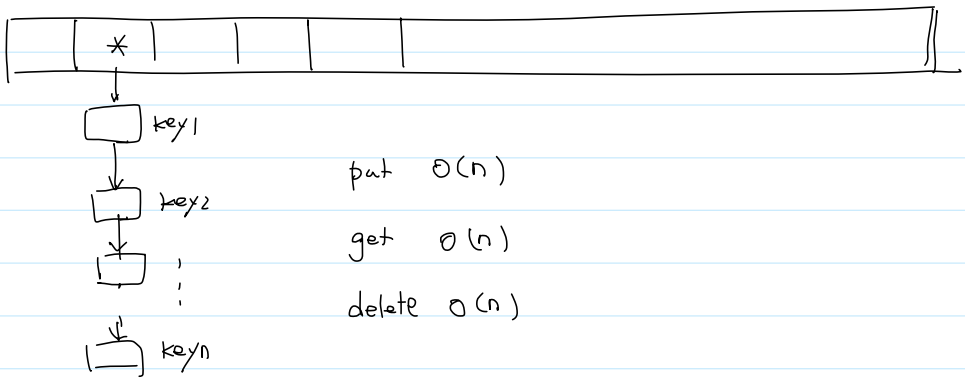
哈希表的扩容

2024年5月6日 15:05

hash值相同
key1
key2
⋮
keyn

hash

1. 安全性 (hashseed)



put O(n)
get O(n)
delete O(n)

引入 hashseed, 引入随机性.

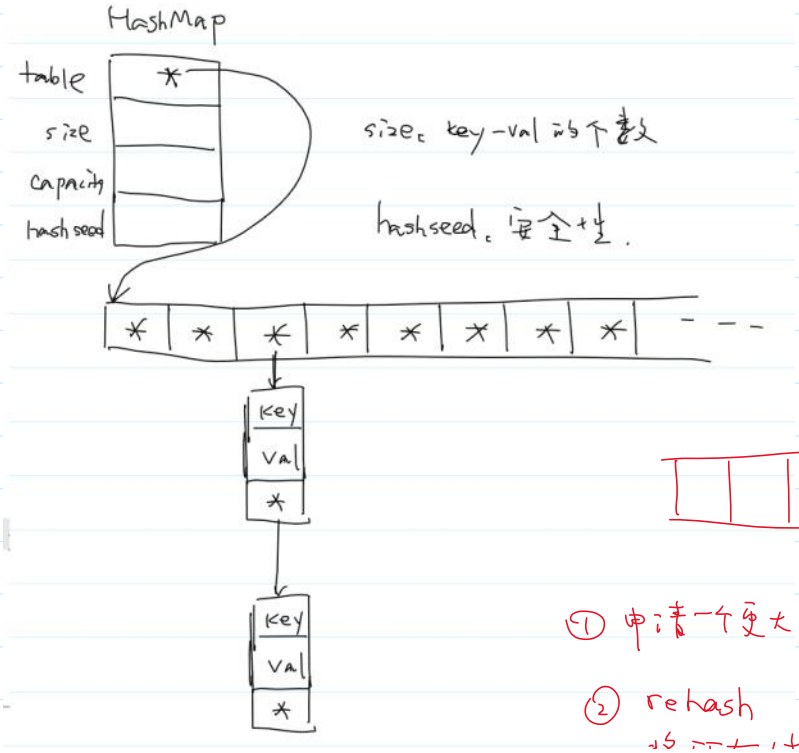
key1
key2
⋮
keyn

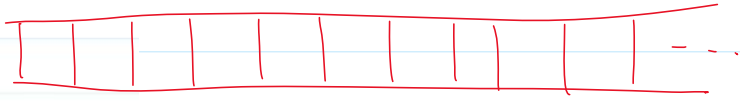
seed 改变

hash (key, len, seed)

在扩容时, 更改 hashseed. 安全

2. 如何扩容.





① 申请一个更大的数组, calloc

② rehash

② rehash
将所有结点挂载到新数组中

③ 释放原数组.

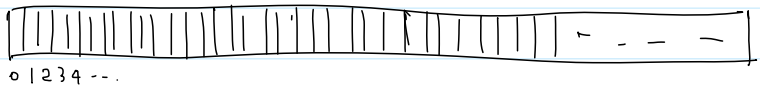
位图

2024年5月6日 15:23

#1. 模型

位的数组。

内存紧凑的数据结构



Q: 为什么需要构建一个专门的数据结构来表示位的数组?

计算机最小的寻址单位: 字节

而不是位.

#2. 基本操作.

增 set: 将某一位设置为 1

删 unset: 将某一位设为 0

查 isset: 判断某一位是不是 1

遍历

#3. 实现

① 尽可能少地申请内存空间

② 块

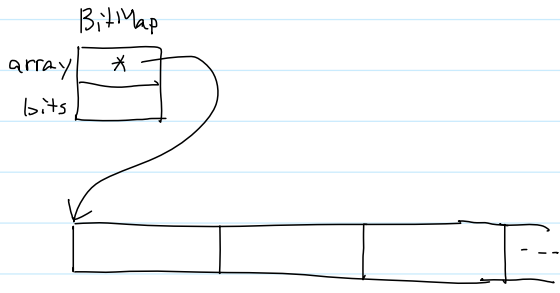
```c
// BitMap.h
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>

typedef struct {
    uint32_t* array;
    size_t bits;        // number of bits in the array
} BitMap;

BitMap* bitmap_create(size_t bits);
void bitmap_destroy(BitMap* bm);

void bitmap_set  (BitMap* bm, size_t n);    // n is a bit index
void bitmap_unset(BitMap* bm, size_t n);
bool bitmap_isset(BitMap* bm, size_t n);
void bitmap_clear(BitMap* bm);
```

位图的长度



每一个字节的值.

```c
memset(bm->array + BITMAP_SIZE(bm->bits), 0, bytes);
```

一个字节一个字节地设置

起始地址

长度

0 ~ 0xFF

总结：

n * 32　等价　n << 5

n / 32　等价　n >> 5

↳ SHIFT

n % 32　等价　n & 0x1F

↳ MASK

```
BitMap* bm = bitmap_create(100);

bitmap_set(bm, 9);
bitmap_set(bm, 5);
bitmap_set(bm, 2);
bitmap_set(bm, 7);
```

0x016B4AF0　a4 02 00 00

7 5 2　　9

1010 0100　　0000 0010

低 → 高

#4. 应用.

① 问题是：10亿 QQ 用户　存储每一个用户的在线状态　（内存 ≤ 1G）

↑ 是否

$10^9 \approx 2^{30}$

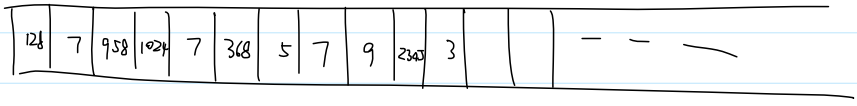
int 数组：$4 \times 2^{30} = 4G$

位图：$\frac{1}{8} \times 2^{30} = 0.125\ G$

位图，内存紧凑

存储两种状态

② 排序并去重.



思路一：排序 O(nlogn)

去重 O(n)

思路二：位图

时间：O(n)