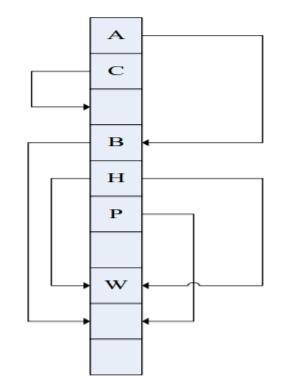
CuckooHash Lib详细说明书

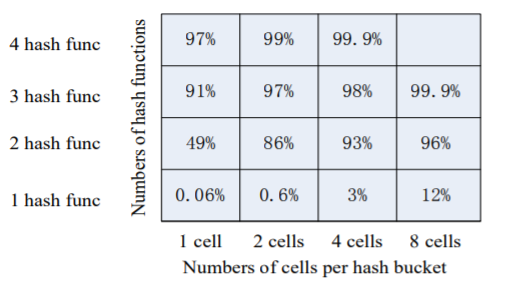
# Cuckoohash概述

## 简介

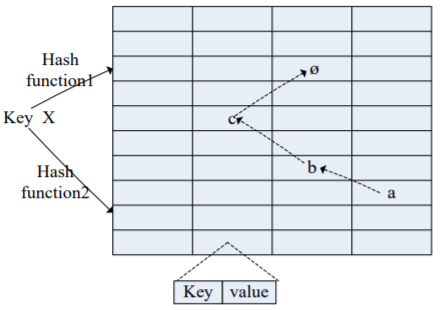
Cuckoo hashing 是一种哈希的冲突解决方式。它的名字源于一种的杜鹃的行为，杜鹃在孵化的时候，会将鸟巢里的其他的小鸡或者蛋推出去，把自己的蛋留下。类似的，当插入一个 key 到已经被占用的位置时，可以把那个占用这个位置的值踢出去，然后将 key 插入进去，被踢出去的值继续搜索下一个位置。可以通过下图来解释：



每个 key 有两个候选的位置用来插入，图中的箭头表示这个 key 的另一个候选 位置。如果来一个 key 值，想要插入到 A 这个位置，则将 A 踢出去，key 插入。 由于 A 的另一个候选位置被 B 占用，所以将 B 踢出，将 A 插入到 B 的位置，B 的 另一个候选位置是空的，所以 B 成功插入。 如果来一个 key，想要插入到 H 这个位置，则将 H 踢出，key 插入。发现 H 的另一个位置已经被 W 占用，需要将 W 移出，再将 H 插入。这时会发现 W 的另 一个候选位置正好是新插入的 key 的位置，按照 cuckoo hashing 的思想，又会把 key 踢出，将 W 插入。而 key 又会将 H 踢出，然后插入到 H 的位置，如此循环往 复，变成死循环，这个时候插入失败。 为了减少冲突，cuckoo hash 表有两种变形：1.增加 hash 函数的个数。2.每个 哈希桶存放多个 key，以这两个参数为参数，表的利用率为因变量，可以得到下表所示的结果。



cuckoo hash 表的优点在于具有高的内存使用率以及 O(1)的查找时间。使用 2 个哈希函数和 4 个 cell 的 cuckoo hash 表如下图所示：



2 个哈希函数 2 个 cell 的 cuckoo hash 表 当查找一个 key 的时候，首先通过两个哈希函数计算出对应的两个哈希桶， 首先查找哈希桶 1，如果查找成功，则结束。否则继续查找哈希桶 2。如果哈希桶 1 和哈希桶 2 都查找失败，则没有查找到对应的 key。 当插入一个 key 的时候，通过两个哈希函数计算出两个候选的可以被插入的 哈希桶，即每个 key 都有对应的两个可选哈希桶。如果这两个哈希桶中有一个有 空闲的 slot，则直接将 key 插入进去，如果这两个哈希桶都是满的，则意味着冲突 的发生，这个时候就会随机选取哈希桶中的一个 slot，将它的 key 踢出去，然后将 待插入的 key 放置在该位置，被踢出去的 key 查看它的另一个候选哈希桶是否满， 如果满了则重复上述的过程，直到成功的插入到一个含有空闲 slot 的哈希桶中。随着哈希表的负载越来越大，这个替换路径也会变的越来越长，因此在 cuckoo hash 表比较负载比较大的时候，插入性能就会大大降低。

# CuckooHash Lib库文件说明

* 1. cuckoohash\_map

定义cuckoohash map的类，以及插入、删除、查找等函数的实现。

* 1. libcuckoo\_bucket\_container

桶容器的定义，以及函数的实现。

* 1. cuckoohash\_config

哈希表默认参数的设置，如：

constexpr size\_t LIBCUCKOO\_DEFAULT\_SLOT\_PER\_BUCKET = 4;

每个bucket的键值对的个数为4个。

constexpr size\_t LIBCUCKOO\_DEFAULT\_SIZE =(1U << 16) \* LIBCUCKOO\_DEFAULT\_SLOT\_PER\_BUCKET;

默认哈希表的大小，此处为2^16\*4=262144

constexpr double LIBCUCKOO\_DEFAULT\_MINIMUM\_LOAD\_FACTOR = 0.05;

最小重载因子=0.05

#define LIBCUCKOO\_DEBUG 0

设置是否启用debug

* 1. cuckoohash\_util

主要存储关于cuckoohash的debug的一些设置及信息包括抛出异常等。

# CuckooHash Map类与函数

* 1. 属性定义

**private:**

using partial\_t = uint8\_t;

using buckets\_t = libcuckoo\_bucket\_container<Key, T, Allocator, partial\_t, SLOT\_PER\_BUCKET>;

**public:**

using key\_type = typename buckets\_t::key\_type;

using mapped\_type = typename buckets\_t::mapped\_type;

using value\_type = typename buckets\_t::value\_type;

using size\_type = typename buckets\_t::size\_type;

using difference\_type = std::ptrdiff\_t;

using hasher = Hash;

using key\_equal = KeyEqual;

using allocator\_type = typename buckets\_t::allocator\_type;

using reference = typename buckets\_t::reference;

using const\_reference = typename buckets\_t::const\_reference;

using pointer = typename buckets\_t::pointer;

using const\_pointer = typename buckets\_t::const\_pointer;

class locked\_table;

* 1. 函数定义

1. 构造函数

cuckoohash\_map(size\_type n = LIBCUCKOO\_DEFAULT\_SIZE, const Hash &hf = Hash(), const KeyEqual &equal = KeyEqual(), const Allocator &alloc = Allocator())

cuckoohash\_map(InputIt first, InputIt last, size\_type n = LIBCUCKOO\_DEFAULT\_SIZE, const Hash &hf = Hash(), const KeyEqual &equal = KeyEqual(), const Allocator &alloc = Allocator())

cuckoohash\_map(const cuckoohash\_map &other): cuckoohash\_map(other, std::allocator\_traits<allocator\_type>::select\_on\_container\_copy\_construction(other.get\_ allocator()))

cuckoohash\_map(const cuckoohash\_map &other, const Allocator &alloc)

cuckoohash\_map(cuckoohash\_map &&other)

cuckoohash\_map(cuckoohash\_map &&other, const Allocator &alloc)

cuckoohash\_map(std::initializer\_list<value\_type> init, size\_type n = LIBCUCKOO\_DEFAULT\_SIZE, const Hash &hf = Hash(), const KeyEqual &equal = KeyEqual(), const Allocator &alloc = Allocator())

1. 主要操作函数

bool empty() 返回哈希表是否为空

size\_type size() 返回表中元素的个数

bool find(const K &key, mapped\_type &val)查找键为key的值，如果找到则将找到的值赋给val并返回true，否则返回false

bool contains(const K &key)查找键key是否存在

bool update(const K &key, V &&val)将键为key的值更新为val

bool insert(K &&key, Args &&... val)将key-val键值插入表中，如果成功返回true，失败返回false

bool erase(const K &key)删除键为key的键值

bool rehash(size\_type n)重哈希

void clear()删除哈希表

# 编译环境要求

编译需要clang++ >= 3.3 且g++ >= 4.8, CMake version >= 3.1.0.并且支持c++11。本人实验过程中试用了win10（Clion，Visual Studio 2017），ubuntu16.04（Clion），综合来看使用ubuntu系统更好，win10下变成bug较多，并且intel DPDK需要在ubuntu上实现。

# 样例代码

1、使用python生成mac、port对，并且有5%的概率生成mac相同，port不如的数据，以测试更新效果。

*import* numpy *as* np  
  
np.random.seed(1)  
  
mac = 48 #mac地址位数  
port = 4 #端口号位数  
address = 4 \* 10\*\*7 #生成个数  
repeated = 1 #重复次数  
filename = "F:\\mac"+str(mac)+"\_port"+str(port)+"\_address"+str(address)+"\_repeated"+str(repeated)+".txt" #生成的文件  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 *with* open(filename, "w") *as* fo:  
 fo.write(str(address\*repeated)+"\n") #文件第一行 数据总数  
 *for* t *in* range(repeated):  
 np.random.seed(1)  
 *for* i *in* range(address):  
 ppp = np.random.rand()  
 tmac = int(ppp \* 2\*\*mac)  
 *if* ppp < 0.02: #2%的概率生成mac相同但是port不同的数据，用以测试update操作  
 ppp = np.random.rand()  
 tport = int(ppp \* 2 \*\* port)  
 fo.write(str(tmac)+" "+str(tport)+"\n")  
 *if*(i % address/10 == 0):  
 print(i)

1. hash.h

**using namespace** std;   
**using namespace** \_\_gnu\_cxx;//需要使用此命名空间  
  
**const char** file\_name[] = **"python保存的测试txt路径+文件名.txt"**;

**using** key = int64\_t;  
**using** value = **int**;

**void** print\_result(**const char** name[],time\_t total\_time,time\_t read\_time,time\_t find\_time,time\_t insert\_time,time\_t update\_time, **int** find\_num,**int** insert\_num,**int** update\_num)//打印查找函数的操作名称、总运行时间、读取时间、查找时间、插入时间、更新时间、查找次数、插入次数、更新次数。

**void** runCuckoohash()//运行布谷鸟哈希查表操作2次，第一次用于建表，第二次用于进行查表

**void** runMap()//运行Map查表操作2次，第一次用于建表，第二次用于进行查表

**void** runHashmap()//运行哈希查表操作2次，第一次用于建表，第二次用于进行查表

以下函数未完成

**void** \_runLockConcurrentCuckoohash(cuckoohash\_map<key , value>& Table)

**void** runLockConcurrentCuckoohash()

**void** runNonlockConcurrentCuckoohash()

2、CmakeList

cmake\_minimum\_required(VERSION 3.10)

project(concurrency)

**set(CMAKE\_CXX\_STANDARD 14)**

**add\_executable(concurrency main.cpp hash.h test.h)**

**include\_directories(/media/wangluobin/Story/研究生/路由器原理与设计/libcuckoo-master/libcuckoo)**

**find\_package(Threads REQUIRED)**

**target\_link\_libraries(concurrency Threads::Threads)**

其中加粗表示需要自己进行修改。

3、测试结果：

（1）mac24位，port4位，4e7个数据，不重复：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cuckoohash | | Map | | Hash | |
| Total time | 125.002 | 80.326 | 151.230 | 129.393 | 81.766 | 68.110 |
| Read time | 18.285 | 18.006 | 18.212 | 17.797 | 18.095 | 18.073 |
| Find time | 39.939 | 36.846 | 83.441 | 86.744 | 22.468 | 21.199 |
| Insert time | 36.287 | 0 | 19.491 | 0 | 9.920 | 0 |
| Update time | 0.338 | 0.489 | 464429 | 0 | 0.171 | 0.244 |
| Find num | 24303969 | 39318875 | 24303969 | 39318875 | 24303969 | 39318875 |
| Insert num | 15231602 | 0 | 15231602 | 0 | 15231602 | 0 |
| Update num | 464429 | 681125 | 464429 | 681125 | 464429 | 681125 |

（2）mac48位，port4位，4e7个数据，不重复：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cuckoohash | | Map | | Hash | |
| Total time | 273.485 | 82.814 | 201.716 | 139.790 | 104.956 | 69.777 |
| Read time | 23.247 | 19.482 | 19.615 | 19.682 | 19.490 | 19.532 |
| Find time | 50.434 | 38.717 | 87.439 | 95.055 | 22.876 | 23.235 |
| Insert time | 153.398 | 0 | 57.713 | 0 | 25.653 | 0 |
| Update time | 0 | 0 | 0 | 0 | 0 | 0 |
| Find num | 2 | 40000000 | 2 | 40000000 | 2 | 40000000 |
| Insert num | 39999998 | 0 | 39999998 | 0 | 39999998 | 0 |
| Update num | 0 | 0 | 0 | 0 | 0 | 0 |