Cuckoo hash

0. 基本思想

- 2个hash函数，映射到不同位置

1. 增删改查

get

- 只可能在`h1(key)`或`h2(key)`，只要探测这两个位置

put

插入`h1(key)`

- 如果冲突，插入`h2(key)`
  - 若`h2(key)`非空，提出`h1(key)`的原来元素`key0`
  - `key0`查看`h2(key0)`是否为空
  - 非空继续踢出

kick

```
KeyType evicted = key; // 当前被提出的
int which = 0;
int idx = hash1(evicted);
while(Table[idx] != 0){ // 如果需要插入的位置非空
    swap(&evicted, &Table[idx]);
    which = 1 - which; // 更换hash函数
    idx = (which == 0)? hash1(evicted): hash2(evicted);
}
Table[idx] = evicted;
```

kick循环

有当一个键的两个可选位置**都各自形成一个环结构**时，才会导致整个过程无法终止

检测：设置一个阈值，超过则认为产生了循环，进行`rehash`

2. 并行

get不改变数据，多个get可以并行

并行put

在需要kick的else语句，整块锁住

3. 基于回溯的实现

串行put+并行get

- 用于同步put和get
- 逆转踢出序列，用于保证被踢元素一直存在在table中
- 用递归回溯路径

```
void Cuckoo::bt_evict(const KeyType &key, int which, int pre_pos){ // 需要记录pre_pos
    int idx = (which == 0)? hash1(key) : hash2(key);

    if(T[idx] == 0){ // 递归终止条件
        T[idx] = key;
        return;
    }

    KeyType cur = T[idx];
    bt_evict(cur, 1 - which, idx); // 先递归踢出路径
    T[idx] = key; // 回溯后再修改值
}
```