# 华中科技大学

# 大数据存储系统与管理报告


姓　名：　史佳鑫

学　院：　计算机科学与技术学院

专　业：　计算机科学与技术

班　级：　CS2104

学　号：　U202111035

指导教师：　施展


| 分数 | |
|---|---|
| 教师签名 | |

2024 年 4 月 21 日

# 目 录

# 1 实验背景

Cuckoo 哈希表是一种用于实现关联数组（Associative Array）的数据结构，它提供了一种高效的查找、插入和删除操作。Cuckoo 哈希的特点是使用了两个不同的哈希函数和一个桶，并且每个键值对只存储在其中一个桶中。当发生哈希冲突时，Cuckoo 哈希表使用另一个哈希函数将冲突的元素移动到另一个桶中，以解决冲突。

Cuckoo 哈希表的工作原理可以简述如下：

初始化：创建两个哈希表，每个哈希表有一组哈希函数，和一定数量的桶。初始化时，所有桶都为空。

插入操作：当要插入一个键值对时，首先使用第一个哈希函数计算键的哈希值，然后将键值对插入到对应的桶中。如果发生哈希冲突，即目标桶已经被占用，那么就使用第二个哈希函数计算新位置，并将原来的元素移动到新位置。如果新位置也已经被占用，就将占用该位置的元素移到其对应的位置，直到找到一个空位置或者达到最大迁移次数为止。

查找操作：通过两个哈希函数计算出键的哈希值，分别在两个哈希表中查找键，如果其中一个哈希表中找到了对应的键，则返回对应的值，否则键不存在。

删除操作：删除操作与查找操作类似，首先查找要删除的键，如果存在，则将对应的桶中的元素删除即可。

Cuckoo 哈希表的优点是在理想情况下，插入、查找和删除操作的时间复杂度都是 O(1)。但是它也有一些缺点，比如可能会出现无限循环的情况，需要进行一定的处理来解决。此外，Cuckoo 哈希表的性能在面对哈希冲突时可能会有所下降，因为需要进行元素的迁移操作。

# 2 实验目的

通过本实验，旨在研究并确定在 Cuckoo 哈希表的操作中，降低无限循环的概率以及提高有效存储的方法。具体目标包括：

理解 Cuckoo 哈希表的原理和操作流程。

研究不同的冲突解决策略，以减少无限循环的概率。

探索合适的哈希函数设计，以降低冲突发生的可能性。

分析并比较不同桶大小和哈希表大小对存储效率的影响。

设计实验测试用例，评估各种改进策略对 Cuckoo 哈希表性能的影响。

通过实验数据和性能指标，确定降低无限循环概率和提高有效存储的最佳方法。

# 3 实验内容

## 3.1 数据结构设计

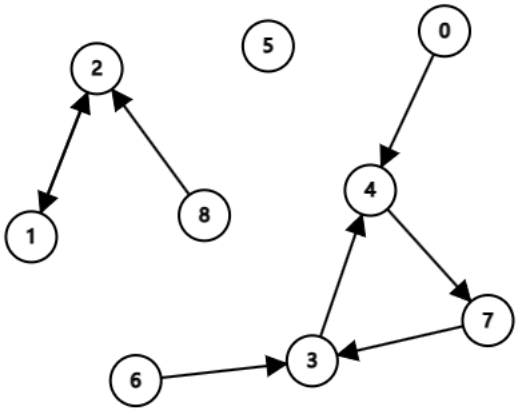
在本次实验中，我们作出了适当的情形简化，具体分析的是更为简单的二个哈希函数和一个一维桶的情况，而不是两个哈希函数和四路哈希桶的复杂情况。

首先根据 Cuckoo 哈希表进行数据结构设计，可以直接使用一个一维数组进行存储，命名为 bucket。

之后进行辅助数据结构设计。在本次实验中，我采用了基于图论算法的优化策略，故需要对图中的信息进行储存。由于每个表单元中只能存储一个元素，这意味着每个元素的出度一定会小于等于一，故采用一维邻接表进行边的存储，将该数组命名为 table。该数组的每个位置上存储着该边的的终点，当不存在出度时表示该点未被选择。示例如图表 1，当前共存储 9 个数据，编号从 0 开始依次为 0、1、……、8，此时表示的有向图如图片 1 所示。（注：该图缺少 5 上指向自己的一个环）

| 4 | 2 | 1 | 4 | 7 | 5 | 3 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|

**图表 1：图存储示例**




图片 1：图存储示例图

## 3.2 操作流程分析

在实验中，我们最关心如何优化插入、查找和删除操作以达到更小的系统开销和更好的性能。以下依次对各项操作进行流程分析。

### 3.2.1 插入操作

在插入操作中我们采用基于图论算法的优化操作，具体操作过程如下：

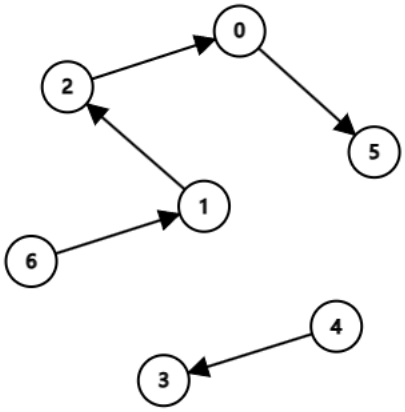
在进行插入操作时，首先计算 hash1(key)、hash2(key)，在桶内查找 hash1(key)和 hash2(key)上的位置是否为空，若存在空位置则直接插入，并产生一条从 hash1(key) 指向 hash2(key)的单向边，该单向边代表着当前位发生 kick-out 操作时当前元素应该被踢出的位置，以维护当前图中信息。若不为空则进行特殊处理。

当 hash1(key)和 hash2(key)均不为空时，则此时需要产生 kick-out 操作，对当前哈希表中的元素进行重新排列。排列策略如下：

在当前图中已存在多条单向边，当一个顶点的出度为一时，代表该点已被某个元素占用，且该单向边的终点为当前元素被踢出时应该去往的位置，例子如图片 2 所示。该图表示分别在哈希表的 0、1、2、4、6 位置，且当前元素发生 kick-out 操作时需要前往的位置。

| 5 | 2 | 0 | -1 | 3 | -1 | 1 |
|---|---|---|---|---|---|---|

图表 2：插入示例 1



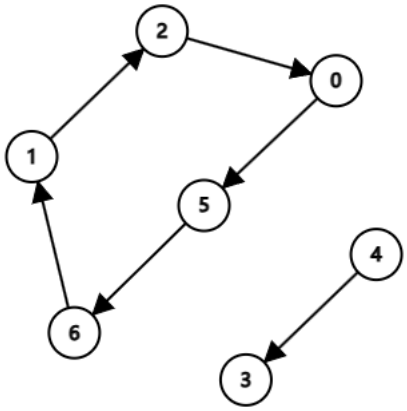

图片 2：插入示例 1

以下依次加入 hash1(key1)=5,hash2(key1)=6 的元素 key1，和 hash1(key2)=5,hash2(key2)=4 的元素 key2 为例继续优化操作。

当加入元素 key1 时，发现虽然当前哈希表的 6 位置被占用，但是 5 位置的出度为 0 表示 5 位置上为空，故将新元素 key1 放在哈希表 5 位置上，并产生一条从 5 指向 6 的单向边，更新后的图如图片 3 所示。

| 5 | 2 | 0 | -1 | 3 | 6 | 1 |
|---|---|---|---|---|---|---|

图表 3：插入 key1 后示例



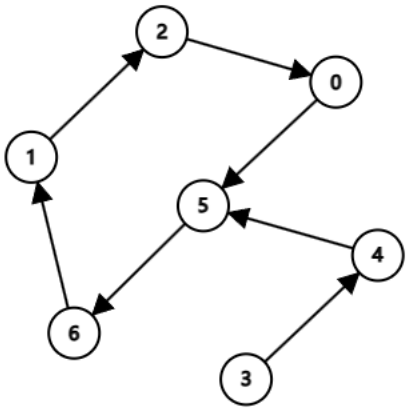

图片 3：插入 key1 后示例

在加入元素 key1 后，再插入 hash1(key2)=5,hash2(key2)=4 的元素 key2。此时会发现哈希表中位置 5 和位置 4 都已被占用，此时应该选择相应的替换策略

以减小在进行 kick-out 操作中的时间开销。通过图论中的判环算法，可以判断两种替换方法中是否存在无限循环的情况。通过查找发现，当插入位置 5 时会发生无限循环，因为存在环，而该环会使所有元素不断发生 kick-out 操作。而以元素 4 为插入位置则不存在环，也就意味着可以进行插入。故在位置 4 中进行插入操作。在插入过程中，首先需要添加一条由 4 指向 5 的边，然后将 kick-out 操作路径上所有经过的有向边依次取反方向，插入后图如图片 4 所示，此时可以根据 hash2(key2)=4 找到该元素 key2，保证了插入的有效性。

| 5 | 2 | 0 | 4 | 5 | 6 | 1 |
|---|---|---|---|---|---|---|

图表 4：插入 key2 后示例




图片 4：插入 key2 后示例

### 3.2.2 查找操作

查找操作的逻辑很简单，可以直接将待查询元素 key 进行哈希运算，获得 hash1(key)、hash2(key)后直接判断当前哈希表中是否存储了当前待查询元素即可，若成功查找到则返回位置，若未找到返回-1。具体代码如下：

```
int search(int key) {
    int index1 = hash1(key);
    int index2 = hash2(key);
    if(bucket[index1]==key) return index1;
    if(bucket[index2]==key) return index2;
    return ERROR;
}
```

### 3.2.3 删除操作

删除操作类似查找操作，首先根据元素 key 进行查找操作，若当前元素不存在则返回-1 表示删除失败；若当前元素存在，则首先将哈希表中当前元素删除，并将当前元素产生的单向边删除，以维护图中的信息。具体代码如下：

```
int delete(int key) {
    int index = search(key);
    if(index==ERROR) return ERROR;
    table[index] = -1;
    bucket[index] = -1;
    return OK;
}
```

## 3.3 理论分析

由于我们主要关心插入操作时如何选择插入策略以避免无限循环和增大有限存储，所以理论分析部分主要对插入操作进行时间复杂度的分析，而查找操作和删除操作较为简单，均为 O(1)时间复杂度。

在插入过程中我们根据当前图的状态在线进行插入策略的调整，保证了在每次插入过程中都能保证能够准确将要插入的元素插入到合适位置。而具体的时间花销在于发生两个哈希函数得出的位置均被占位时，进行的图维护和深度搜索过程。

在进行深度搜索的过程中，需要根据当前元素依次向下进行搜索，直到判断出环或非环。当得到环或非环的信息后，需要进行图维护操作，将当前元素插入，并将路径上的单向边反转以正确维护图信息。该方法的时间复杂度均为 O(L)，L 为该环或链的长度，最坏情况下时间复杂度为 O(n)，最好情况下时间复杂度为 O(1)。

可以优化的部分是暴力的判环算法。该算法的时间复杂度为环的长度，因此在某些极端情况下并不能很好的工作，最坏的情况下所有元素构成一个大环，此时判环时间开销将为整个哈希表的大小。可以引入记忆化算法进行优化，使判环的时间复杂度尽可能的减小，优化后该算法的时间复杂度可以降为 O(1)。优化方法此处省略。

# 4 实验总结和课程建议

大数据存储系统与管理这门课程设计的很好，我在本门的理论课程中学到了关于固态存储、数据去重、元数据管理等一系列先进存储知识，扩充了我的知识面。同时通过课程实验报告的撰写，我自主探索了关于 Cuckoo 哈希表插入策略的更优解，通过特殊的数据结构优化，降低了在进行 Cuckoo 哈希表插入时无限循环的概率，并提高了有效存储的容量，我觉得成就感满满。

在课堂上我认为华老师学术能力很强，课程内有很多存储领域相关的学术词汇，这很好地扩展了我们的学术视野。但大部分的词语我们并没有听过，存在着知识上的差距。同时由于缺乏必要的教程推荐，理论课让我感觉更像是在听一场学术讲座，实际的上课感受并不理想。希望华宇老师能增加关于存储方面的教材推荐，在教学用的 ppt 内添加一些定义和阐述，一定能有效增加教学的效果。