# 2019 年软件工程优才夏令营测试题

题目：哈希表与哈希表

哈希表由于其拥有平均O(1)的插入和查找时间复杂度，被广泛应用于保存键值类数据。然而恰恰因为其时间复杂度不固定，有多种哈希表的实现方法来提升哈希表在各种情况下的表现。

本次题目中，你将实现并评测基于以下两种方法的哈希表:

Linear hashing

Cuckoo hashing

## A. 原理描述

为简化题目，题目中的键值(key,value)均表示为32位无符号整数。具体接口如下：

- Set(key,value) 将键值对(key,value)插入到哈希表中，若key已存在，则替换当前的value。
- Get(key) 给出指定key做对应的value，若key不在表中，则返回null。
- Delete(key) 将指定key和其对应的value从哈希表中删除，若key不在表中，则不做任何操作。

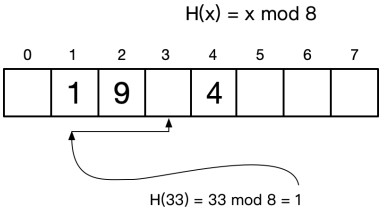
下面将分别对这两种方法的哈希表的原理进行描述。注意，为简单起见，所给的示例图中仅显示了key，在实际实现中应保存 (key,value)对。

1. Linear hashing

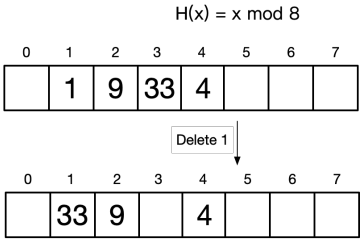
又称线性探测(linear probing)，值从哈希的目标地址开始依次向后探测以解决哈希冲突问题。若探测到哈希表末尾，则返回到哈希表开头继续探测。

**查找：**从哈希的目标位置开始，依次向后面搜索(最后一个位置的后面会回到第一个位置)，直到被目标键被找到，则返回目标键值对；若遇到一个空位，则表示目标键不在表中。

**插入：**当哈希的目标地址被占用时，依次向后面的位置进行查找，当遇到空位时可将键值对插入空位。




**删除：**找到目标键值对，将其删除。同时为了保证此后操作的正确性，可能需使用后面的元素向前移动。例如下图删除1后，将33挪到1号位置。

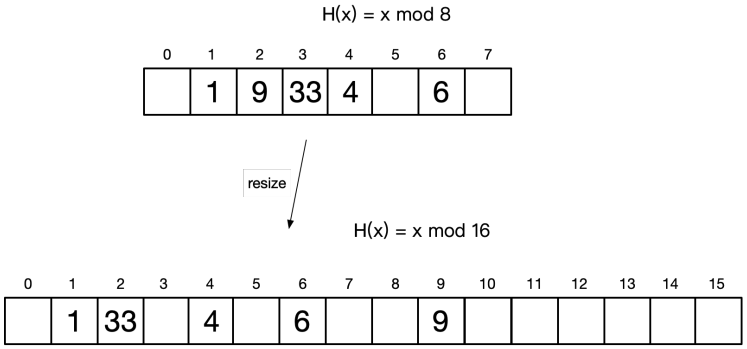



**扩容：**为了保证哈希表的性能，当哈希表中的元素个数超过哈希表位置数的一半时，需要触发一次哈希表扩容（本题目中不考虑使哈希表总容量减小的操作）：

1）分配一块两倍大的内存，作为新的哈希表；

2）将原哈希表中的所有元素重新插入到新的哈希表中。由于总容量改变，需要将哈希函数随着更改；

3）释放原哈希表空间。



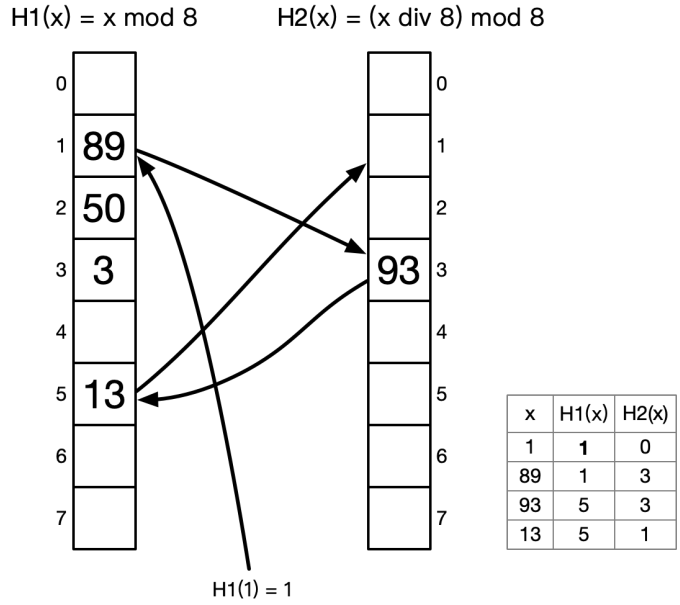
2. Cuckoo hashing

在Cuckoo中使用两个哈希表，分别对应两个不同的哈希函数H1和H2，因而每个键k有两个位置可以存放H1(k)和H2(k)。

**查找：**在两个哈希表中分别检查H1(k)和H2(k)，若其中有一个为目标键，则找到目标；若均不是目标键，则说明目标键不在哈希表中。

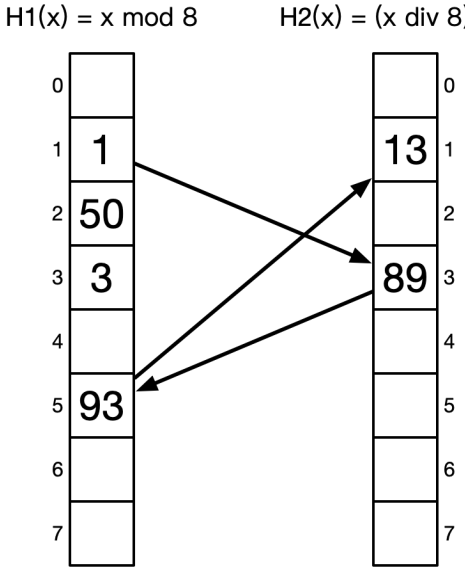
**插入：**在第一个哈希表中检查H1(k)所指定的位置，若H1(k)是空位，则插入到此空位；否则，尝试将H1(k)中所存放的键值对移动到其另外一个位置（即其H2所对应的位置），若另外一个位置有值，则继续移动，直到一个空位被发现，或者替换路径出现了一个环。

下图为一个具体的例子。对于要插入的键1，H1(1)=1但左边第一个位置被89占据，则需将89踢到右边的哈希表中。H2(89)=3但是右边哈希表第3个位置被93占据，则需继续将93踢到左边的哈希表中。H1(93)=5但是左边哈希表第5个位置被13占据，则需继续将13踢到右边的哈希表中。H2(13)=1，恰好右边第一个位置为空，则可将13放入此空位，并以此将前面踢到的键值对进行移动，最后键1可以顺利的插入到左边哈希表的第一个位置。



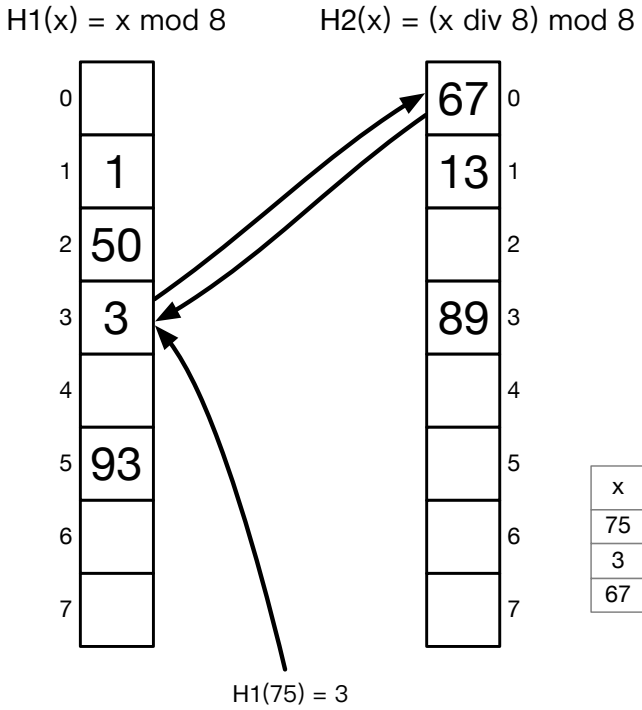

| x | H1(x) | H2(x) |
|---|---|---|
| 1 | **1** | 0 |
| 89 | 1 | 3 |
| 93 | 5 | 3 |
| 13 | 5 | 1 |

下图为插入之后的哈希表状态，且交换路径已经被用箭头标出。




| x | H1(x) | H2(x) |
|---|---|---|
| 1 | **1** | 0 |
| 89 | 1 | 3 |
| 93 | 5 | 3 |
| 13 | 5 | 1 |

下图中的例子表示一个交替路径产生环的情况，此处在插入键75之前，需先对哈希表进行扩容。（本题目中不考虑使哈希表总容量减小的操作）




| x | H1(x) | H2(x) |
|---|---|---|
| 75 | **3** | 1 |
| 3 | 3 | 0 |
| 67 | 3 | 0 |

**删除：**先查找到指定键值对，之后将其从其位置上删除。

**扩容：**当插入时交替路径出现环，而导致无法成功插入新的键值对时，cuckoo哈希表需要进行扩容操作。其扩容方法与linear hashing类似，即分配双倍的空间，并将所有的键值对重新插入到新的cuckoo哈希表中，最后释放原空间。扩容的同时哈希函数也应该进行相应的变化。

## B. 题目要求

题目包括三部分：正确性测试、性能评测、以及图形化展示。

1. 正确性测试

请大家按照上述两种方法实现哈希表。在实现时请遵循如下表格。

| 哈希表 | 哈希表容量S的初始值 | 哈希函数 |
| --- | --- | --- |
| Linear Hashing | 8 | H(key) = key mod S |
| Cuckoo Hashing | 两个哈希表均为8 | H1(key) = key mod S<br>H2(key) = (key div S) mod S |

为了保证大家写的哈希表的正确性，题目提供两组测试数据。其中一组规模较小，可用于调试；另外一组规模较大，可以更完备地测试哈希表的正确性。由于两种方法实现的哈希表仅内部实现不同，接口和对外功能均一样，两组测试用例均可用于两种哈希表中。

测试输入文件格式：“Set<空格>key<空格>value”，

“Get<空格>key”，

“Del<空格>key”。

样例输入如下：

```
Set 5 12
Get 5
Set 12 7
Set 22 9
Set 11 99
Get 22
Set 22 1
Get 22
Del 22
Get 22
```

对于其中个每个Get请求，程序应输出Get请求的结果，每个结果占一行。

因此上述样例的正确输出应为

```
12
9
1
null
```

每组样例提供一个.in文件表示输入，一个.ans文件表示正确答案。程序的输出与.ans文件相符则表明通过测试。提示：可使用diff（Linux上）、fc（Windows上）、comp（Windows上）等工具进行文件内容比较。

2. 性能评测

此部分中请大家设计并实现测试程序，对两种哈希表进行性能评测和对比。

具体过程如下：

a) 测试过程：

i. 初始化：新建空白哈希表，产生10000条**随机键**和值并向表中插入

ii. 测试0：记录初始化过程中，前1000次插入各自所需的延迟。

iii. 测试1：（初始化后）**Get请求**的性能，具体指标包括

1. 吞吐量：平均请求次数/s；
2. 测试时段内请求的最小、最大、平均延迟；

iv. 测试2：（初始化后）当**请求中Get和Set各占一半**时的系统性能，指标同上。

b) 通过上述方法得到的性能数据作图（可使用excel等任意画图工具）

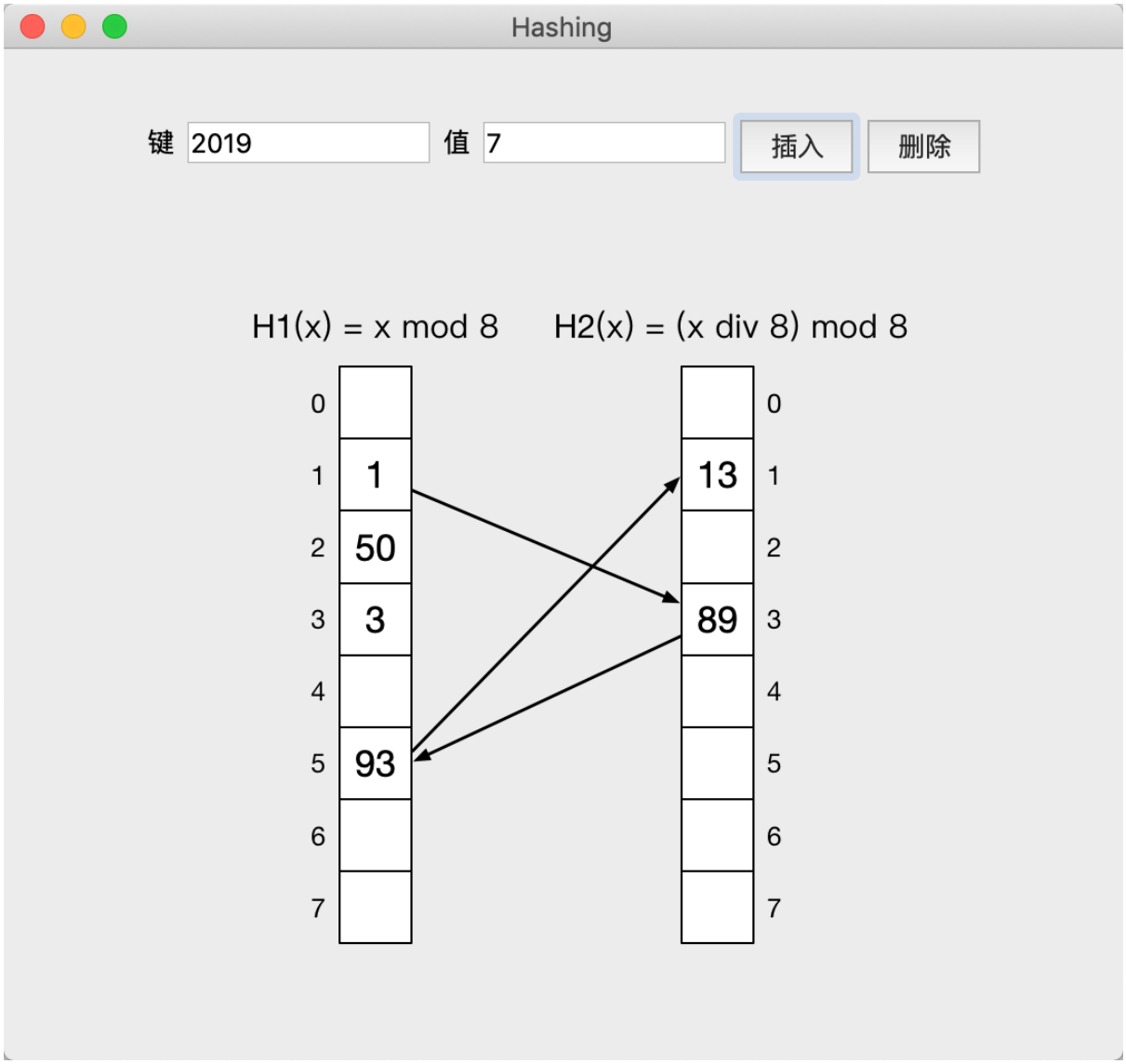
i. 一张折线图：表示两种哈希表在前1000次插入的延迟变化。

ii. 一张延迟柱状图：表示两种哈希表全为Get请求以及Get和Set混合时的延迟对比。请根据平均延迟画出柱状图，并使用最大和最小延迟画出误差区间。

iii. 一张吞吐量柱状图：表示两种哈希表全为Get请求以及Get和Set混合时的吞吐量对比。

3. 图形化演示

设计实现使用图形化的方式，展示cuckoo hashing哈希表的插入或删除的过程，只需展示两个哈希容量为8时的修改过程，不包括扩容，界面如下图所示（界面仅需显示键值对中的键即可）。




初始时哈希表为空，通过在输入框中输入新的键值对，并按“插入”按钮，可以将一个键值对插入到下方所示的哈希表中。当因哈希冲突发生替换时，应通过箭头的方式将替换的位置标明，如上图中的箭头表示，在插入1时，因此将89、93和13踢到了另外一个位置。进行删除时，输入需要删除的键（不需要输入值），并按“删除”按钮，可以执行删除操作。

## C. 考核标准：

1. （40分）分别使用三种方法实现哈希表，并通过给出的正确性测试。
   c） Linear hashing：小规模数据占5分，大规模数据占10分；
   d） Cuckoo hashing：小规模数据占10分，大规模数据占15分；
2. （30分）设计程序/测试用例，测试两种哈希表的优劣。请准确测试数据， 以保证其结果可被合理解释。
   a） （5分）获得1000次插入的延迟数据；
   b） （8分）全部Get请求下的：吞吐量、三种延迟（最大、最小、平均）；
   c） （8分）一半Set一半Get请求下的：吞吐量、三种延迟（最大、最小、平均）；
   e） （9分）得出对比图，每张图3分。
3. （30分）图形化展示。
   a） （10分）绘制出相应界面。
   b） （15分）实现事件响应，并正确实现哈希表的动态显示。
   c） （5分）正确展现cuckoo哈希的替换路径。