Cuckoo Hashing & Finding Loop

Nitesh Gupta
Dept. of Computer Science,
OPJS University, Churu,
Rajasthan, India

Dr. Om Prakash
Dept. of Computer Science,
OPJS University, Churu,
Rajasthan, India

Abstract: We will present a simple cuckoo hashing with an example of how it works. Along with it, we will present an algorithm to find a loop in cuckoo hashing.

Keywords: Hashing, Collision, Cuckoo Hashing, Memory Management.

I. INTRODUCTION
Cuckoo Hashing is an algorithm for resolving hash Collision of the values of the hash functions in the table and enhance the worst case lookup time.

Algorithm
1. Let F (n) used for hashing is a hash function and initial assignment is F (1).
2. Hash key (HK) with Function F (n).
3. Check if the place is empty or not.
4. If the place is not empty push the existing entry into TEMP.
5. Place HK into the location.
6. Check which function is used to place TEMP into the location.
7. Assign TEMP back to HK.
8. Assign alternate hash function into F (n).
9. Repeat from step 2 until there is no collision.

Now let’s see how this algorithm solves the above collision problem. Let’s say first all the keys are placed in empty locations (except JS). Now key JS came for the entry. It is hashed with function 1 and location is derived as 03. Now when the location is checked if it is free or not, it says that there is one entry stored at that location. Then entry (NG) is pushed out of the location and JS is stored at that location. Then this NG has hashed again and a new location is derived as 11. Now, this new location is checked if it is free or not. Since location 11 is free, NG is placed at this location.
II. SELF-EVOLUTION

Cuckoo Hashing use two hash functions which guarantee that a key must be present in one of the two memory locations derived by the two functions. Since we know that this algorithm has the problem of the infinite loop because of which it cannot be used in any of the stable systems. Let’s discuss the algorithm that can be used to identify such loop.
Identification of Loop

To evolve the algorithm, we first need to specify when a loop can occur in the algorithm. This can be identified easily by tracking what all elements are being replaced at the time of any insertion. If the number of replacement of any element is more than the number of functions present at that particular time, which means it went into an infinite loop.

Now let’s see understand this algorithm with an example. Let’s say first all the keys are placed in empty locations.

![Fig. 1 – Initial placement](image1)

Table I

<table>
<thead>
<tr>
<th>Replacement Key</th>
<th># of Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>NG</td>
<td></td>
</tr>
<tr>
<td>SJ</td>
<td></td>
</tr>
<tr>
<td>VE</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
</tr>
</tbody>
</table>

Now JS came for the entry. We hash JS with function 1 and get location 03. Since 03 is already filled by NG, NG is moved out and JS is placed.

![Fig. 2 – JS kicked out NG](image2)

Table II

<table>
<thead>
<tr>
<th>Replacement Key</th>
<th># of Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>NG</td>
<td></td>
</tr>
<tr>
<td>JS</td>
<td></td>
</tr>
<tr>
<td>SJ</td>
<td></td>
</tr>
<tr>
<td>VE</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
</tr>
</tbody>
</table>

Now NG is hashed with function 2 and a new location 04 is derived. Since VE is already at location 04, we will remove VE and NG is placed at 04.
Table III

<table>
<thead>
<tr>
<th>Replacement Key</th>
<th>NG</th>
<th>VE</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Replacement</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Now VE has hashed with function 2 and a new location is 08 derived. Since SJ is already at location 08, we will remove SJ and VE are placed at 08.

Table IV

<table>
<thead>
<tr>
<th>Replacement Key</th>
<th>NG</th>
<th>VE</th>
<th>SJ</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Replacement</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Now SJ is hashed with function 2 and a new location is 03 derived. Since JS is already at location 03, we will remove JS and SJ is placed at 03.
Now JS has hashed with function 2 and a new location is 08 derived. Since VE is already at location 08, we will remove VE and JS is placed at 08.

Now VE is hashed with function 1 and a new location is 04 derived. Since NG is already at location 04, we will remove NG and VE is placed at 04.

Now NG is hashed with function 1 and a new location is 03 derived. Since SJ is already at location 03, we will remove SJ and NG is placed at 04.
Now SJ has hashed with function 1 and a new location is 08 derived. Since JS is already at location 04, we will remove JS and SJ is placed at 04.

Now JS is hashed with function 1 and a new location is 03 derived. Since NG is already at location 03, we will remove NG and JS is placed at 03.
Now since NG is replaced 3 times, and functions for cuckoo hashing is 2, hence we can easily conclude that it’s an infinite loop.

CONCLUSIONS
In this paper, we discussed the basic concept of Cuckoo Hashing. Then we discussed the derived algorithm that can be used to detect a loop while insertion. We can finally conclude that this algorithm can be the best possible way in which we can find out the loop and can terminate the infinite running of the algorithm.

ACKNOWLEDGMENT
Nitesh Gupta wishes to acknowledge Dr. Om Prakash and other contributors for continuous encouragement and helping me in writing this paper.

REFERENCES