INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X

Impact factor: 4.295

*(Volume3, Issue2)*

# Complexity of Cuckoo Hashing

**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

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of a string representing the input. It is commonly expressed using big O notation which excludes constants, coefficients and lowers order terms. In other words, time taken by an algorithm to finish its execution is known as Time Complexity.

We usually deal with WORST CASE TIME COMPLEXITY, which defines the maximum time that algorithm can take for a particular set of input.

All the operations that we perform on Hash Table can be formed using two main operations:

1. INSERT
2. LOOKUP / SEARCH

Now let's see the Worst-Case Time Complexity for all the algorithms that we have discussed above.
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.

**Insert**

In Cuckoo Hashing, if the derived location is empty it is placed at that location. If not then the Key present at the location is removed and the new Key is placed at the location. The old Key is then hashed using another hash function to find the alternate location and repeated. This will go on until we find an empty location for the key. Hence, the algorithm will look like

a. Location = Hash of Key using Hash function Fn.
b. If location is not free, take out old Key and place new Key.
c. Key = old Key and Hash function Fn = alternate function.
d. Repeat step (a) until the location is empty.

This means, in worst case scenario, for every insert there can be n number of locations are looked up before any insertion. Hence, the Time Complexity for this algorithm is big O (n). But the probability of occurrence of this situation is least as compared to other algorithms.

**Lookup / Search**

In this algorithm, location is derived by hashing the Key using two Hash functions as a result Key can be found on exactly two locations.

a. Location = Hash of Key using Fn (1).
b. If Key if not present at location then Location = Hash of Key using Fn (2)

This means even in worst case scenario, Key must be present at one of the two locations. Hence, the Time Complexity for this algorithm is big O (1).

TABLE I
COMPLEXITY TABLE

| | **Insert** | **Lookup / Search** |
|---|---|---|
| **Bucketed Hashing** | O (1) | O (n) |
| **Linear Probing** | O (n) | O (n) |
| **Cuckoo Hashing** | O (n) | O (1) |

## CONCLUSIONS

In this paper, we discussed the basic concept of Cuckoo Hashing and how to calculate the complexity of cuckoo hashing in various scenarios: Insertion / Lookup / Search.

## ACKNOWLEDGMENT

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

## REFERENCES

[1] Pagh, Rasmus and Rodler, Flemming Friche (2001). "Cuckoo Hashing". Algorithms — ESA 2001. Lecture Notes in Computer Science 2161. pp. 121–133. doi:10.1007/3-540-44676-1_10. ISBN 978-3-540-42493-2.

[2] Knuth, Donald (1998). 'The Art of Computer Programming'. 3: Sorting and Searching (2nd ed.). Addison-Wesley. pp. 513–558.

[3] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). Introduction to Algorithms (3rd ed.). Massachusetts Institute of Technology. pp. 253–280. ISBN 978-0-262-03384-8.

[4] Cuckoo Hashing, Theory, and Practice (Part 1, Part 2 and Part 3), Michael Mitzenmacher, 2007.

[5] Algorithmic Improvements for Fast Concurrent Cuckoo Hashing, X. Li, D. Andersen, M. Kaminsky, M. Freedman. EuroSys 2014.