Cuckoo Hashing is an embellished form of normal hashing for dictionaries that has constant lookup time and decreases the time in between rehashing. The implementation of Cuckoo Hashing is relatively easy. Two tables T1 and T2 , both of size n are needed as well as two hash fuctions H1 and H2 . The two tables always need to be the same size. When a value V1 is added, its hash value is computed with function H1 and then it is added to T1 . If there is already a value V2 in that specific position in T1 , then it is pushed out of its spot in T1 and H2 is used to compute the new hash value of V­2 and add it into T­2 . This effectively doubles the number of values that can be stored between the two tables and allows for more time between rehashes. Also, the load factor for the two tables can be increased because there is double the capacity for holding hashed values. When a key is looked up, Cuckoo Hash will return either H1 {key} or H2 {key} , whichever one hashes to a non-null value. Cuckoo Hashing favors putting values in T1 before putting them in T2 to reduce the time needed to find which table the value is in.

When Cuckoo Hashing is tested against Two-Way Chaining, Chained Hashing, and Linear Probing on a Linux system with an 800 MHz Intel Pentium III processor, it is quicker in some regards and slower in others. For data sets bigger than 16, Cuckoo Hashing is a better performing method when there is an unsuccessful lookup or a deletion by nearly 40 clock cycle. For successful lookups, Cuckoo hashing performed just as well as Two-way chaining which was about 40 clock cycles longer than Linear Probing and 40 clock cycles less than Chained Hashing. For insertion, Cuckoo Hashing only out-performed Two-way Chaining by about 20 clock cycles. Overall Cuckoo Hashing is a better method of dictionary management than these other methods.

Here is the pseudo code for Cuckoo Hashing:

procedure insert(x)

if lookup(x) then return;

loop MaxLoop times

if T1[h1(x)] = ⊥ then { T1[h1(x)] ← x; return; }

x ↔ T1[h1(x)];

if T2[h2(x)] = ⊥ then { T2[h2(x)] ← x; return; }

x ↔ T2[h2(x)];

end loop

rehash(); insert(x);

end;

Source: [http://link.springer.com/chapter/10.1007%2F3-540-44676-1\_10]