It's interesting that the development of open addressing using linear probing, quadratic probing, and double hashing took place in the 1960's and early-middle 1970's but that different methods of probing with open addressing were not discovered until 30-40 years later. Here, we describe a technique call *cuckoo hashing*, which was first described in a paper by R Pagh and F. Rodler in 2001. This description is based on a simplified presentation written by R.Pagh in 2006[[1]](#footnote-1). We found at least 15 papers on the web concerning Cuckoo hashing and improvements on it.

**How cuckoo hashing got its name**

A picture containing outdoor object, aerie

Description automatically generatedThis version of probing is named after the European cuckoo. When a female cuckoo is ready to lay her eggs, she finds a nest of some other bird, waits for the host bird to leave the nest, picks up an egg that's in the nest, lays one of her eggs there, and flies off, eating the egg she picked up. The host bird usually accepts the cuckoo egg and incubates it.

There's more. The cuckoo egg usually hatches before the other eggs. The hatchling cuckoo, with its eyes still closed, pushes the unhatched host eggs from the nest. Then, the young cuckoo gets the undivided attention of its foster parents, who will feed and nurture it.[[2]](#footnote-2)

In a nutshell (or a bird nest?), here is how cuckoo hashing works. Suppose value egg is to be inserted in the table and it hashes to h (taken mod the table size, as usual). If bucket h is null, fine, egg is placed there. But if another value egg1 is in bucket h, it is removed and egg is placed there! Of course, now, a different bucket must be found to place egg1.

**A simple variant of cuckoo hashing**

The simple variant of cuckoo hashing uses two hash functions h1 and h2. To determine whether a value e is in the table, check the two positions b[h1(e)] and b[h2(e)] (taken modulo the table size, of course). If neither contains e, then e is not in the table; there is no need to worry about collisions. This is worst-case time O(1).

Similarly, to remove e from the table, look at those two buckets. If neither is e, then e is not in the set and nothing need be removed. If one of them is e, then set that bucket to null. This is worst-case time O(1).

1. void insert(e) {

2. if (b[h1(e)] == e || b[h2(e)] == e) return;

3. int p= h1(e);

4. // inv: Trying to insert e at b[p]

5. loop n times: {

6. if b[p] == null { b[p]= e; return; }

7. Swap e and b[p];

8. if (p == h1(e)) p= h2(e);  
9. else p= h1(e);

10. }

11. rehash; insert(e);

12.}

Both search and remove take constant time in the worst case! How can that be! Because collisions won't occur. All the work to eliminate collisions takes place in the method to insert e into the hash table. It is shown to the right (look at it after reading everything below). *All indexes are taken module the table size*; to save space, we leave that implicit.

Method insert works as follows. Suppose e0 is to be inserted. Remember, two hash functions are used to produce two possible buckets for e0. Value n, the maximum number of iterations, is the table size: b.length.

If a bucket is null, store e0 there and return; else, kick some value e1 out of a bucket and store e0 there.

If a bucket is null, store e1 there and return; else, kick some value e2 out of a bucket and store e1 there.

...

If a bucket is null, store en-1 there and return; else, kick some value en out of a bucket and store en-1 there.

Create two new hash functions, h1 and h2, and rehash all values using the new functions.

Now, insert en into the table (a recursive call).

1. R. Pugh. *Cuckoo hashing for undergraduates*. IT University of Copenhagen, 2006. Obtain a pdf file of this paper from JavaHyperText entry *hashing*. [↑](#footnote-ref-1)
2. The image is taken from this website: http://akermariano.blogspot.com/2006/12/european-cuckoo.html. The discussion is a paraphrase of the text found there. [↑](#footnote-ref-2)