Cuckoo Hashing

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 other methods of probing with open addressing weren't 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]. We found at least 15 papers on the web concerning cuckoo hashing and improvements on it.

How cuckoo hashing got its name

This 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. The young cuckoo then gets the undivided attention of its foster parents, who will feed and nurture it.[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, the removed `egg1` can't be eaten but must be placed in a different bucket.

A simple variant of cuckoo hashing

A 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).

Both search and remove take constant time in the worst case! How can that be! Because collisions won't occur. The method to insert `e` into the hash table does a lot of work to ensure the absence of collisions. It is shown on the next page.

Method `insert` works as follows. Suppose e_0 is to be inserted, and let `n` be the table size: `b.length`. Remember, two hash functions are used to determine two possible buckets for e_0. Below, e_k *bucket* refers to one or the other of the two buckets to which e_k hashes.

If an e_0 bucket is null, store e_0 there and return; else, kick a value e_1 out of an e_0 bucket and store e_0 there.
If an e_1 bucket is null, store e_1 there and return; else, kick a value e_2 out of an e_1 bucket and store e_1 there.
If an e_2 bucket is null, store e_2 there and return; else, kick a value e_3 out of an e_2 bucket and store e_2 there.
...
If an e_{n-1} bucket is null, store e_n there and return; else, kick a value e_n out of an e_{n-1} bucket and store e_n there.

A total of `n` values have been kicked out of a bucket without finding a place for all values. Since `n` is the table size, there is clearly a loop. Create two new hash functions, `h1` and `h2`, and rehash all values using the new functions. This includes all values in the table, but note that e_n is not in the table.
Insert e_n into the table (a recursive call).

Method insert appears on the next page.

---

[1] R. Pagh. *Cuckoo hashing for undergraduates*. IT University of Copenhagen, 2006. Obtain a pdf file of this paper from JavaHyperText entry *hashing*.

[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.

Method `insert(e)` appears to the right. *All indexes are taken modulo the table size*; to save space, we leave that implicit. We provide some explanation.

```
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.}
```

Line 2. If either of the two buckets to which `e` hashes contains `e`, there is no need to insert `e`, so the method returns.

Lines 3..10. The loop has the invariant:

`e` has to be inserted at `b[p]`

The initialization, setting `p` to `h1(e)`, truthifies it. You can check the repetend yourself to see that it maintains the invariant. Also see that the repetend returns if it is able to place `e` at `b[p]`.

Line 11. If `n` iterations of the loop are performed, then it is time to rehash the whole table, choosing two new hash functions `h1` and `h2`. After that, `n` has to be inserted into the table. Note that this is a recursive call on method `insert`.

The expected amortized time of insert is O(1), provided the load factor is ≤ .5. The worst case is O(table size).

Cuckoo hashing relies on the existence of a family of high-quality hash functions from which new ones can be chosen. They need to be uniformly distributed, and if two values have colliding hash codes with one function, they won't have them with most other functions in the family. This notion of "families of independent hash functions" has been studied, but the topic is not covered in JavaHyperText.

You will have questions about this. How are the new hash functions created? How do we prove that method `insert` is expected constant time? Our purpose here is not to answer all such questions but simply to give you an overview of this idea. Some of these questions are answered in the paper *Cuckoo hashing for undergraduates* mentioned in the footnote on the previous page. Also, if you are really interested in cuckoo hashing and want to try implementing it yourself, look at other papers on the topic.