Cuckoo hashing

In cuckoo hashing we have two hash tables of the same size and each hash table has its hash function.

- For each element we compute two positions: one from the first hash table and one from the second.

In case of cuckoo hashing, it is guaranteed that an element will be on one of these positions.

- Search is simple, because we only have to look at these two positions.
- Delete is simple, because we only have to look at these two positions and set to empty the one where we find the element.

h1(k) = k mod 11

e.g.:

Position	0	1	2	3	4	5	6	7	8	9	10
T		100		36			50			75	

Position	0	1	2	3	4	5	6	7	8	9	10
T	3	20		39	53		67			105	

h2(k) = (k div 11)mod 11

Cuckoo hashing

Think about:

Assume that we have two hash tables, with m = 11 positions and the following hash functions:

- h1(k) = k mod 11
- h2(k) = (k div 11) mod 11

Insert: 20, 50, 53, 75, 100, 67, 105, 3, 36, 39 , …

Cuckoo hashing

Adding an element:

- When we want to insert a new element we will compute its position in the first hash table.
- If the position is empty, we will place the element there.
- If the position in the first hash table is not empty, we will kick out the element that is currently there, and place the new element into the first hash table.
  - The element that was kicked off, will be placed at its position in the second hash table.
  - If that position is occupied, we will kick off the element from there and place it into its position in the first hash table.

We repeat the above process until

- we will get an empty position for an element

- or, if we get back to the same location with the same key, we have a cycle we cannot add this element ➔ resize, rehash (until no collisions occur)

Cuckoo hashing

Remark:

- While in some situation insert moves a lot of elements, it can be shown that if the load factor of the tables is below 0.5, the probability of a cycles is low and it is very unlikely that more than $O(\log_2 n)$ elements will be moved.