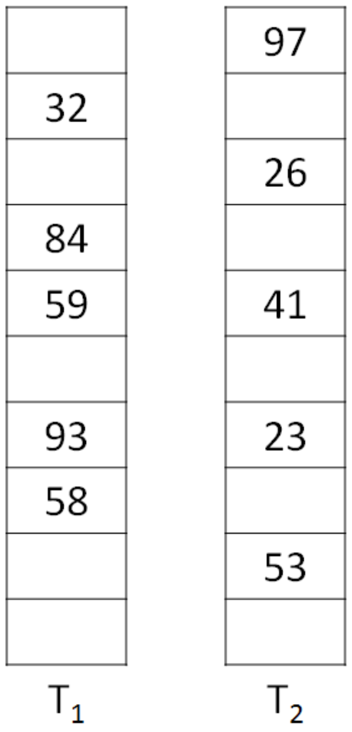
Cuckoo Hashing

Cuckoo hashing is an easy and at the same time efficient method for resolving collisions in hash tables. It was invented in 2001 by Rasmus Pagh and Flemming Friche Rodler. The original paper is available here (or https://www.brics.dk/RS/01/32/BRICS-RS-01-32.pdf). The lookup time is constant O(1), thus deletion is done in O(1), and it has O(1) amortized insertion (for more, read at https://en.wikipedia.org/wiki/Cuckoo_hashing). The basic idea is to use two hash functions instead of only one. This provides two possible locations in the hash table for each key.

There are several versions of cuckoo hashing. The one that we discussed in the class uses two hash tables, each with m slots, and two hash functions, h_1, h_2: U →{0, 1, 2, ..., m-1}. Each element x will have exactly one “nest” (hash location) in each table, either $h_1(x)$or $h_2(x)$.



The two operations are inserting an element in the hash table(s) and retrieving an element from the hash table.

Inserting an Element

To insert a new element, we use a greedy method. The new element is inserted in one of its two possible locations, “kicking out” any element that might already be in that location. This displaced element needs to find a place in the hash table(s) so it is is then inserted in its alternative location, again “kicking out” any key that might reside there, until a vacant position is found, or the procedure would enter an infinite loop. In the latter case, the hash table is rebuilt in-place using new hash functions. The steps are detailed next:

Insertion(x):

1. Insert x by looking first into Table 1, T1
2. If h1 (x) is empty, insert x there (into T1)
3. Otherwise place x there (into T1) and evict the old element y by trying to place it into table 2, T2
4. If h2 (y) is empty, insert y there (into T2)
5. Otherwise place y there (into T2) and evict the old element z by trying to place it into table 1, T1
6. Repeat this process, oscillating between the two tables, until all elements stabilize (have slots)

Under reasonable assumptions, this process will terminate in O(1) time but Insertion does not work if the algorithm runs in a cycle. A cycle only occurs if the revisit the same slot with the same element to be inserted. Please note that it is possible for a successful insertion to revisit the same slot.
In case you run into a cycle, execute rehashing by choosing new h1 (x) and h2 (x) hash functions and inserting all the elements back into the hash tables. Multiple rehashes might be necessary before this will succeed.

As long as the number of keys is kept below half of the capacity of the hash table, i.e., the load factor is below 50%, insertions succeed in expected constant time, even considering the possibility of having to rebuild the table.

To analyze the running time of insertion, we will use a Cuckoo graph, detailed in the next section. A Cuckoo graph is a bipartite graph derived from a cuckoo hash table.

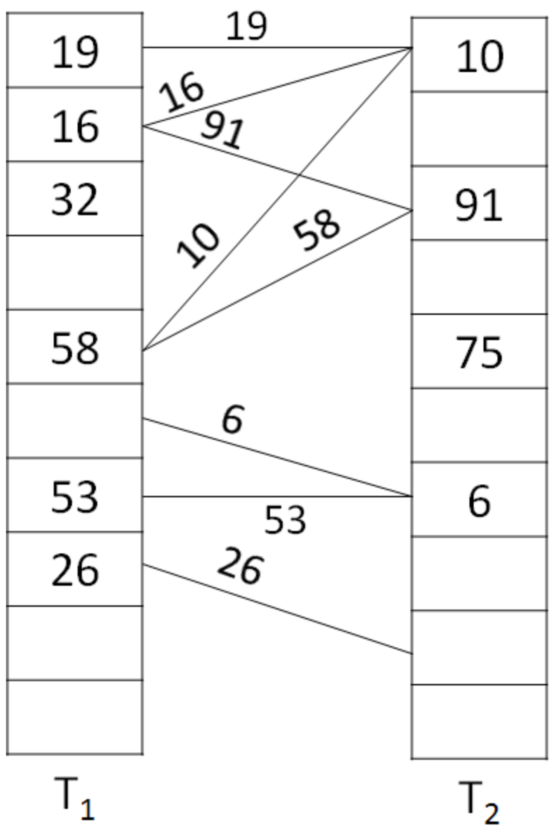
Cuckoo graph

The cuckoo graph is a bipartite graph in which:

- Each table slot is a node
- Each element to be inserted is an edge

Edges links slots where an element can be either because it was inserted successfully there or because it was inserted there after being evicted from somewhere else.
Each insertion inserts a new edge in the graph.
Therefore each insertion traces a path in the cuckoo graph. An insertion succeeds if and only if the connected component associated with the inserted value contains at most one cycle. In the example

below, we have two successful insertions and one unsuccessful insertion (one cycle). (For more details, please visit https://web.stanford.edu/class/cs166/lectures/13/Small13.pdf).


19
16
32
58
53
26
T1
10
91
75
6
T2
19
16
91
10
58
6
53
26