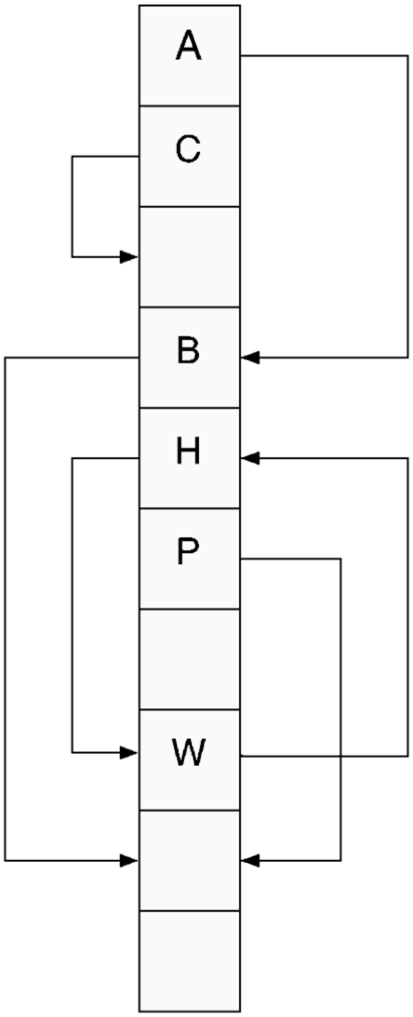
en.wikipedia.org /wiki/Cuckoo_hashing

Cuckoo hashing

Contributors to Wikimedia projects ⁝ 15-19 minutes ⁝ 2/9/2006
DOI: 10.1007/3-540-44676-1_10, Show Details

From Wikipedia, the free encyclopedia



Cuckoo hashing example. The arrows show the alternative location of each key. A new item would be inserted in the location of A by moving A to its alternative location, currently occupied by B, and moving B to its alternative location which is currently vacant. Insertion of a new item in the location of H would not succeed: Since H is part of a cycle (together with W), the new item would get kicked out again.

Cuckoo hashing is a scheme in computer programming for resolving hash collisions of values of hash functions in a table, with worst-case constant lookup time. The name derives from the behavior of some species of cuckoo, where the cuckoo chick pushes the other eggs or young out of the nest when it hatches in a variation of the behavior referred to as brood parasitism; analogously, inserting a new key into a cuckoo hashing table may push an older key to a different location in the table.

Cuckoo hashing was first described by Rasmus Pagh and Flemming Friche Rodler in a 2001 conference paper.[1] The paper was awarded the European Symposium on Algorithms Test-of-Time award in 2020.[2]:122

Cuckoo hashing is a form of open addressing in which each non-empty cell of a hash table contains a key or key–value pair. A hash function is used to determine the location for each key, and its presence in the table (or the value associated with it) can be found by examining that cell of the table. However, open addressing suffers from collisions, which happens when more than one key is mapped to the same cell. The basic idea of cuckoo hashing is to resolve collisions by using two hash functions instead of only one. This provides two possible locations in the hash table for each key. In one of the commonly used variants of the algorithm, the hash table is split into two smaller tables of equal size, and each hash function provides an index into one of these two tables. It is also possible for both hash functions to provide indexes into a single table.[1]:121-122

Cuckoo hashing uses two hash tables, T_1 and T_2. Assuming r is the length of each table, the hash functions for the two tables is defined as, $h_1,\, h_2\ :\ \cup \rightarrow \{0, \dots, r-1\}$ and $\forall x \in S$ where x is the key and S is the set whose keys are stored in $h_1(x)$ of T_1 or $h_2(x)$ of T_2. The lookup operation is as follows:[1]:124

```
function lookup(x) is
    return $T_1[h_1(x)] \ = \ x \vee T_2[h_2(x)] = x$
end function
```

The logical or ($\vee$) denotes that, the value of the key x is found in either T_1 or T_2, which is $O(1)$ in worst case.[1]:123

Deletion is performed in $O(1)$ time since probing is not involved. This ignores the cost of the shrinking operation if the table is too sparse.[1]:124-125

When inserting a new item with key x, the first step involves examining if slot $h_1(x)$ of table T_1 is occupied. If it is not, the item is inserted in that slot. However, if the slot is occupied, the existing item x' is removed and x is inserted at $T_1[h_1(x)]$. Then, x' is inserted into table T_2 by following the same procedure. The process continues until an empty position is found to insert the key.[1]:124-125 To avoid an infinite loop, a threshold Max-Loop is specified. If the number of iterations exceeds this fixed threshold, both T_1 and T_2 are rehashed with new hash functions and the insertion procedure repeats. The following is pseudocode for insertion:[1]:125

On lines 10 and 15, the "cuckoo approach" of kicking other keys which occupy $T_{1,2}[h_{1,2}(x)]$ repeats until every key has its own "nest", i.e. item x is inserted into an empty slot in either of the two tables. The notation $x \leftrightarrow y$ expresses swapping x and y.[1]:124-125

Insertions succeed in expected constant time,[1] even considering the possibility of having to rebuild the table, 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%.

One method of proving this uses the theory of random graphs: one may form an undirected graph called the "cuckoo graph" that has a vertex for each hash table location, and an edge for each hashed value, with the endpoints of the edge being the two possible locations of the value. Then, the greedy insertion algorithm for adding a set of values to a cuckoo hash table succeeds if and only if the cuckoo graph for this set of values is a pseudoforest, a graph with at most one cycle in each of its connected components. Any vertex-induced subgraph with more edges than vertices corresponds to a set of keys for which there are an insufficient number of slots in the hash table. When the hash function is chosen randomly, the cuckoo graph is a random graph in the Erdős–Rényi model. With high probability, for load factor less than 1/2 (corresponding to a random graph in which the ratio of the number of edges to the number of vertices is bounded below 1/2), the graph is a pseudoforest and the cuckoo hashing algorithm succeeds in placing all keys. The same theory also proves that the expected size of a connected component of the cuckoo graph is small, ensuring that each insertion takes constant expected time. However, also with high probability, a load factor greater than 1/2 will lead to a giant component with two or more cycles, causing the data structure to fail and need to be resized.[3]

Since a theoretical random hash function requires too much space for practical usage, an important theoretical question is which practical hash functions suffice for Cuckoo hashing. One approach is to use k-independent hashing. In

2009 it was shown[4] that $O(\log n)$-independence suffices, and at least 6-independence is needed. Another approach is to use tabulation hashing, which is not 6-independent, but was shown in 2012[5] to have other properties sufficient for Cuckoo hashing. A third approach from 2014[6] is to slightly modify the cuckoo hashtable with a so-called stash, which makes it possible to use nothing more than 2-independent hash functions.

In practice, cuckoo hashing is about 20–30% slower than linear probing, which is the fastest of the common approaches.[1] The reason is that cuckoo hashing often causes two cache misses per search, to check the two locations where a key might be stored, while linear probing usually causes only one cache miss per search. However, because of its worst case guarantees on search time, cuckoo hashing can still be valuable when real-time response rates are required.

The following hash functions are given (the two least significant digits of k in base 11):

$$h\left(k\right) = k \bmod 11$$
$$h'\left(k\right) = \left\lfloor \frac{k}{11} \right\rfloor \bmod 11$$

The following two tables show the insertion of some example elements. Each column corresponds to the state of the two hash tables over time. The possible insertion locations for each new value are highlighted. The last column illustrates a failed insertion due to a cycle, details below.

Table 1: uses h(k)

		Steps									
Step number		1	2	3	4	5	6	7	8	9	10
Key inserted		53	50	20	75	100	67	105	3	36	45
h(k)		9	6	9	9	1	1	6	3	3	1
Hash table entries	**0**										
	1					100	67	67	67	67	45
	2										
	3								3	36	36
	4										
	5										

6		50	50	50	50	50	105	105	105	105
7										
8										
9	53	53	20	75	75	75	53	53	53	53
10										

Table 2: uses h′(k)

		Steps									
Step number		1	2	3	4	5	6	7	8	9	10
Key inserted		53	50	20	75	100	67	105	3	36	45
h′(k)		4	4	1	6	9	6	9	0	3	4
	0									3	3
	1				20	20	20	20	20	20	20
	2										
	3										
	4			53	53	53	53	50	50	50	50
Hash table entries	**5**										
	6							75	75	75	75
	7										
	8										
	9						100	100	100	100	100
	10										

If you attempt to insert the element 45, then you get into a cycle, and fail. In the last row of the table we find the same initial situation as at the beginning again.

$$h\left(45\right) = 45 \bmod 11 = 1$$

$$h'\left(45\right) = \left\lfloor \frac{45}{11} \right\rfloor \bmod 11 = 4$$

Table 1	**Table 2**
45 replaces 67 in cell 1	67 replaces 75 in cell 6
75 replaces 53 in cell 9	53 replaces 50 in cell 4
50 replaces 105 in cell 6	105 replaces 100 in cell 9
100 replaces 45 in cell 1	45 replaces 53 in cell 4
53 replaces 75 in cell 9	75 replaces 67 in cell 6
67 replaces 100 in cell 1	100 replaces 105 in cell 9

105 replaces 50 in cell 6 50 replaces 45 in cell 4
45 replaces 67 in cell 1 67 replaces 75 in cell 6

Several variations of cuckoo hashing have been studied, primarily with the aim of improving its space usage by increasing the load factor that it can tolerate to a number greater than the 50% threshold of the basic algorithm. Some of these methods can also be used to reduce the failure rate of cuckoo hashing, causing rebuilds of the data structure to be much less frequent.

Generalizations of cuckoo hashing that use more than two alternative hash functions can be expected to utilize a larger part of the capacity of the hash table efficiently while sacrificing some lookup and insertion speed. Using just three hash functions increases the load to 91%.[7]

Another generalization of cuckoo hashing called *blocked cuckoo hashing* uses more than one key per bucket and a balanced allocation scheme. Using just 2 keys per bucket permits a load factor above 80%.[8]

Another variation of cuckoo hashing that has been studied is *cuckoo hashing with a stash*. The stash, in this data structure, is an array of a constant number of keys, used to store keys that cannot successfully be inserted into the main hash table of the structure. This modification reduces the failure rate of cuckoo hashing to an inverse-polynomial function with an exponent that can be made arbitrarily large by increasing the stash size. However, larger stashes also mean slower searches for keys that are not present or are in the stash. A stash can be used in combination with more than two hash functions or with blocked cuckoo hashing to achieve both high load factors and small failure rates.[9] The analysis of cuckoo hashing with a stash extends to practical hash functions, not just to the random hash function model commonly used in theoretical analysis of hashing.[10]

Some people recommend a simplified generalization of cuckoo hashing called skewed-associative cache in some CPU caches.[11]

Another variation of a cuckoo hash table, called a cuckoo filter, replaces the stored keys of a cuckoo hash table with much shorter fingerprints, computed by applying another hash function to the keys. In order to allow these fingerprints to be moved around within the cuckoo filter, without knowing the keys that they came from, the two locations of each fingerprint may be computed from each other by a bitwise exclusive or operation with the fingerprint, or with a hash of the fingerprint. This data structure forms an approximate set membership data structure with much the same properties as a Bloom filter: it can store the members of a set of keys, and test whether a query key is a member, with some chance of false positives (queries that are incorrectly reported as being part of the set) but no false negatives. However, it improves on a Bloom filter in multiple respects: its memory usage is smaller by a constant factor, it has better locality of reference, and (unlike Bloom filters) it allows for fast deletion of set elements with no additional storage penalty.[12]

A study by Zukowski et al.[13] has shown that cuckoo hashing is much faster than chained hashing for small, cache-resident hash tables on modern processors. Kenneth Ross[14] has shown bucketized versions of cuckoo hashing (variants that use buckets that contain more than one key) to be faster than conventional methods also for large hash tables, when space utilization is high. The performance of the bucketized cuckoo hash table was investigated further by Askitis,[15] with its performance compared against alternative hashing schemes.

A survey by Mitzenmacher[7] presents open problems related to cuckoo hashing as of 2009.

Cuckoo hashing is used in TikTok's recommendation system to solve the problem of "embedding table collisions", which can result in reduced model quality. The TikTok recommendation system "Monolith" takes advantage cuckoo hashing's collision resolution to prevent different concepts from being mapped to the same vectors.[16]

- Perfect hashing
- Double hashing
- Quadratic probing
- Hopscotch hashing

1. ^ Jump up to: ***a b c d e f g h i j*** *Pagh, Rasmus; Rodler, Flemming Friche (2001). "Cuckoo Hashing". Algorithms — ESA 2001. Lecture Notes in Computer Science. Vol. 2161. CiteSeerX 10.1.1.25.4189. doi:10.1007/3-540-44676-1_10. ISBN 978-3-540-42493-2.*
2. **^** *"ESA - European Symposium on Algorithms: ESA Test-of-Time Award 2020". esa-symposium.org. Award committee: Uri Zwick, Samir Khuller, Edith Cohen. Archived from the original on 2021-05-22. Retrieved 2021-05-22.*{{cite web}}: CS1 maint: others (link)

3. ^ *Kutzelnigg, Reinhard (2006). Bipartite random graphs and cuckoo hashing (PDF). Fourth Colloquium on Mathematics and Computer Science. Discrete Mathematics and Theoretical Computer Science. Vol. AG. pp. 403–406.*
4. ^ Cohen, Jeffrey S., and Daniel M. Kane. "Bounds on the independence required for cuckoo hashing." ACM Transactions on Algorithms (2009).
5. ^ Pătraşcu, Mihai, and Mikkel Thorup. "The power of simple tabulation hashing." Journal of the ACM (JACM) 59.3 (2012): 1-50.
6. ^ Aumüller, Martin, Martin Dietzfelbinger, and Philipp Woelfel. "Explicit and efficient hash families suffice for cuckoo hashing with a stash." Algorithmica 70.3 (2014): 428-456.
7. ^ Jump up to: *a b Mitzenmacher, Michael (2009-09-09). "Some Open Questions Related to Cuckoo Hashing" (PDF). Proceedings of ESA 2009. Retrieved 2010-11-10.*
8. ^ *Dietzfelbinger, Martin; Weidling, Christoph (2007). "Balanced allocation and dictionaries with tightly packed constant size bins". Theoret. Comput. Sci. **380** (1–2): 47–68. doi:10.1016/j.tcs.2007.02.054. MR 2330641.*
9. ^ *Kirsch, Adam; Mitzenmacher, Michael D.; Wieder, Udi (2010). "More robust hashing: cuckoo hashing with a stash". SIAM J. Comput. **39** (4): 1543–1561. doi:10.1137/080728743. MR 2580539.*
10. ^ *Aumüller, Martin; Dietzfelbinger, Martin; Woelfel, Philipp (2014). "Explicit and efficient hash families suffice for cuckoo hashing with a stash". Algorithmica. **70** (3): 428–456. arXiv:1204.4431. doi:10.1007/s00453-013-9840-x. MR 3247374. S2CID 1888828.*
11. ^ "Micro-Architecture".
12. ^ *Fan, Bin; Andersen, Dave G.; Kaminsky, Michael; Mitzenmacher, Michael D. (2014), "Cuckoo filter: Practically better than Bloom", Proc. 10th ACM Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT '14), pp. 75–88, doi:10.1145/2674005.2674994*
13. ^ *Zukowski, Marcin; Heman, Sandor; Boncz, Peter (June 2006). "Architecture-Conscious Hashing" (PDF). Proceedings of the International Workshop on Data Management on New Hardware (DaMoN). Retrieved 2008-10-16.*
14. ^ *Ross, Kenneth (2006-11-08). Efficient Hash Probes on Modern Processors (PDF) (Research Report). IBM. RC24100. Retrieved 2008-10-16.*
15. ^ *Askitis, Nikolas (2009). "Fast and Compact Hash Tables for Integer Keys". Proceedings of the 32nd Australasian Computer Science Conference (ACSC 2009) (PDF). Vol. 91. pp. 113–122. ISBN 978-1-*

920682-72-9. Archived from the original (PDF) on 2011-02-16. Retrieved 2010-06-13.

16. ^ *"Monolith: The Recommendation System Behind TikTok". gantry.io. Retrieved 2023-05-30.*

- A cool and practical alternative to traditional hash tables Archived 2019-04-07 at the Wayback Machine, U. Erlingsson, M. Manasse, F. Mcsherry, 2006.
- Cuckoo Hashing for Undergraduates, 2006, R. Pagh, 2006.
- Cuckoo Hashing, Theory and Practice (Part 1, Part 2 and Part 3), Michael Mitzenmacher, 2007.
- *Naor, Moni; Segev, Gil; Wieder, Udi (2008). "History-Independent Cuckoo Hashing". International Colloquium on Automata, Languages and Programming (ICALP). Reykjavik, Iceland. Retrieved 2008-07-21.*
- Algorithmic Improvements for Fast Concurrent Cuckoo Hashing, X. Li, D. Andersen, M. Kaminsky, M. Freedman. EuroSys 2014.

- Concurrent high-performance Cuckoo hashtable written in C++
- Cuckoo hash map written in C++
- Static cuckoo hashtable generator for C/C++
- Cuckoo hash table written in Haskell
- Cuckoo hashing for Go