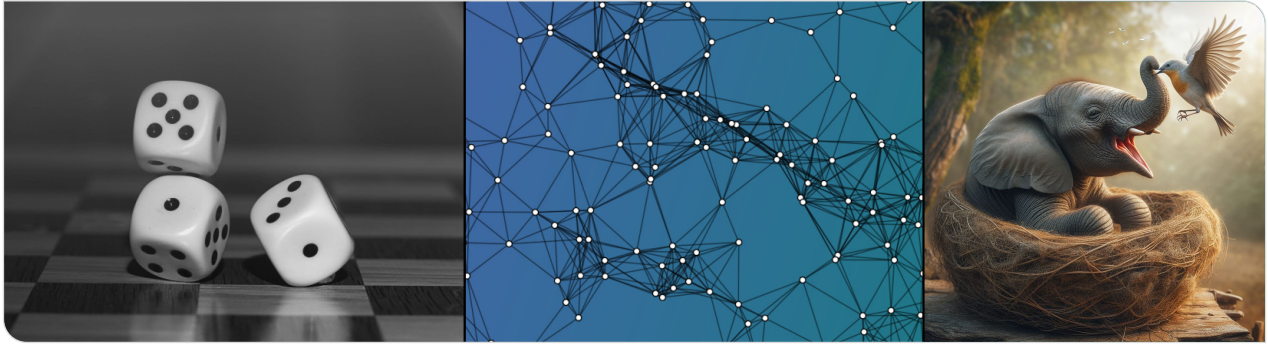
Karlsruhe Institute of Technology

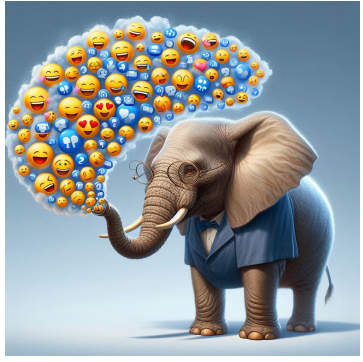
Algorithm Engineering

Probability and Computing – Cuckoo Hashing

Stefan Walzer | WS 2025/2026



Evaluation



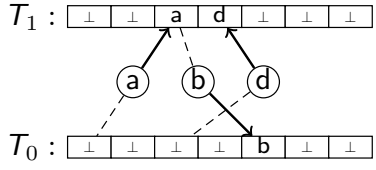
https://onlineumfrage.kit.edu/evasys/online.php?p=RF73W

Content

Classic Cuckoo Hashing

Setup

$S \subseteq D$	key set of size n
T_0, T_1	two tables of size m
$h_0, h_1 \sim \mathcal{U}([m]^D)$	two hash functions (SUHA)
$\frac{n}{m} = 1 - \beta$	for some $\beta > 0$
	($\triangle$! load factor $\alpha = \frac{n}{2m}$)



Algorithm lookup(x):
  return $x \in \{T_0[h_0(x)], T_1[h_1(x)]\}$

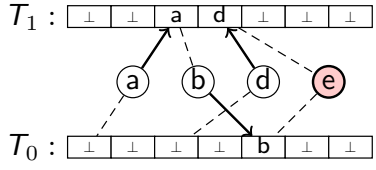
Algorithm delete(x):
  if $T_0[h_0(x)] = x$ **then**
    $T_0[h_0(x)] \leftarrow \perp$
  else if $T_1[h_1(x)] = x$ **then**
    $T_1[h_1(x)] \leftarrow \perp$

Algorithm insert(x):
  for $i = 0$ **to** LIMIT **do**
    $b \leftarrow i \bmod 2$
    swap(x, $T_b[h_b(x)]$)
    if $x = \perp$ **then**
      return SUCCESS
  return FAILURE

Classic Cuckoo Hashing

Setup

$S \subseteq D$	key set of size n
T_0, T_1	two tables of size m
$h_0, h_1 \sim \mathcal{U}([m]^D)$	two hash functions (SUHA)
$\frac{n}{m} = 1 - \beta$	for some $\beta > 0$
	($\triangle$! load factor $\alpha = \frac{n}{2m}$)



Algorithm lookup(x):
  return $x \in \{T_0[h_0(x)], T_1[h_1(x)]\}$

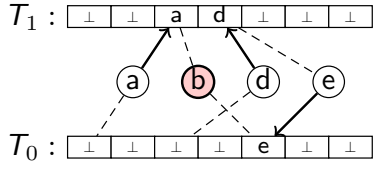
Algorithm delete(x):
  if $T_0[h_0(x)] = x$ **then**
    $T_0[h_0(x)] \leftarrow \perp$
  else if $T_1[h_1(x)] = x$ **then**
    $T_1[h_1(x)] \leftarrow \perp$

Algorithm insert(x):
  for $i = 0$ **to** LIMIT **do**
    $b \leftarrow i \bmod 2$
    swap(x, $T_b[h_b(x)]$)
    if $x = \perp$ **then**
      return SUCCESS
  return FAILURE

Classic Cuckoo Hashing

Setup

$S \subseteq D$	key set of size n
T_0, T_1	two tables of size m
$h_0, h_1 \sim \mathcal{U}([m]^D)$	two hash functions (SUHA)
$\frac{n}{m} = 1 - \beta$	for some $\beta > 0$
	($\triangle$! load factor $\alpha = \frac{n}{2m}$)



Algorithm lookup(x):
  return $x \in \{T_0[h_0(x)], T_1[h_1(x)]\}$

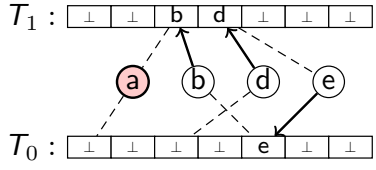
Algorithm delete(x):
  if $T_0[h_0(x)] = x$ **then**
    $T_0[h_0(x)] \leftarrow \perp$
  else if $T_1[h_1(x)] = x$ **then**
    $T_1[h_1(x)] \leftarrow \perp$

Algorithm insert(x):
  for $i = 0$ **to** LIMIT **do**
    $b \leftarrow i \bmod 2$
    swap(x, $T_b[h_b(x)]$)
    if $x = \perp$ **then**
      return SUCCESS
  return FAILURE

Classic Cuckoo Hashing

Setup

$S \subseteq D$	key set of size n
T_0, T_1	two tables of size m
$h_0, h_1 \sim \mathcal{U}([m]^D)$	two hash functions (SUHA)
$\frac{n}{m} = 1 - \beta$	for some $\beta > 0$
	($\triangle$! load factor $\alpha = \frac{n}{2m}$)



Algorithm lookup(x):
  return $x \in \{T_0[h_0(x)], T_1[h_1(x)]\}$

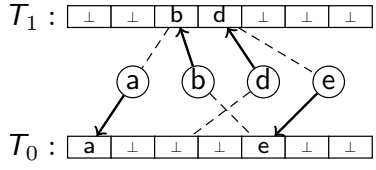
Algorithm delete(x):
  if $T_0[h_0(x)] = x$ **then**
    $T_0[h_0(x)] \leftarrow \perp$
  else if $T_1[h_1(x)] = x$ **then**
    $T_1[h_1(x)] \leftarrow \perp$

Algorithm insert(x):
  for $i = 0$ **to** LIMIT **do**
    $b \leftarrow i \bmod 2$
    swap(x, $T_b[h_b(x)]$)
    if $x = \perp$ **then**
      return SUCCESS
  return FAILURE

Classic Cuckoo Hashing

Setup

$S \subseteq D$	key set of size n
T_0, T_1	two tables of size m
$h_0, h_1 \sim \mathcal{U}([m]^D)$	two hash functions (SUHA)
$\frac{n}{m} = 1 - \beta$	for some $\beta > 0$
	($\triangle$! load factor $\alpha = \frac{n}{2m}$)



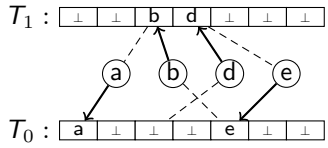
Algorithm lookup(x):
  return $x \in \{T_0[h_0(x)], T_1[h_1(x)]\}$

Algorithm delete(x):
  if $T_0[h_0(x)] = x$ **then**
    $T_0[h_0(x)] \leftarrow \perp$
  else if $T_1[h_1(x)] = x$ **then**
    $T_1[h_1(x)] \leftarrow \perp$

Algorithm insert(x):
  for $i = 0$ **to** LIMIT **do**
    $b \leftarrow i \bmod 2$
    swap(x, $T_b[h_b(x)]$)
    if $x = \perp$ **then**
      return SUCCESS
  return FAILURE

Cuckoo Hashing Theorem

Algorithm insert(x)**:**
- **for** $i = 0$ **to** LIMIT **do**
  - $b \leftarrow i \bmod 2$
  - swap(x, $T_b[h_b(x)]$)
  - **if** $x = \perp$ **then**
    - **return** SUCCESS
- **return** FAILURE



Theorem (Analysis with LIMIT $= \infty$)

Assume we insert all $x \in S$ and then another key y. Let E be the event that this succeeds and

$$T = \begin{cases} \text{insertion time of } y & \text{if } E \text{ occurs} \\ 0 & \textit{otherwise} \end{cases}$$

Then (i) $\Pr[E] = 1 - \mathcal{O}(1/m)$ and (ii) $\mathbb{E}[T] = \mathcal{O}(1)$.

Theorem (full analysis, not here)

If we
- set LIMIT $= \Omega(\log n)$ appropriately
- rebuild the table with fresh hash functions when LIMIT is reached

we obtain a hash table where lookup and delete take $\mathcal{O}(1)$ time and insert takes *expected* $\mathcal{O}(1)$ time.

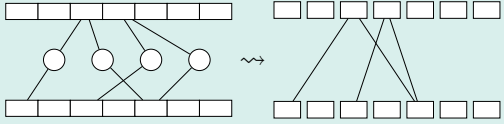
Proof of i: Success probability is $1 - \mathcal{O}(1/m)$

The Cuckoo Graph

Consider the bipartite *cuckoo graph*

$$G = ([m], [m], \{(h_0(x), h_1(x)) \mid x \in S\})$$

the key x corresponds to the edge $(h_0(x), h_1(x))$ and each table position to a vertex.



// Duplicate edges possible, don't worry about it.

Connection to Erdős-Renyi Graphs

G is a bipartite Erdős-Renyi variant

- much like $G^{\text{UE}}(2m, n)$ // uniform endpoint model
- a bit like $G(2m, n)$ // original Erdős-Renyi
- a bit like $G(2m, n/(2m^2))$ // Gilbert

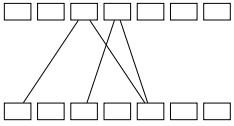
Confusing: n is a number of edges and $2m$ a number of vertices.

Exercise

Design a variant of cuckoo hashing such that the "Sudden Emergence" result for the $G^{\text{UE}}(m, n)$ model implies success for load factor $\alpha < \frac{1}{2}$.

Next: Completely self-contained analysis without reference to Erdős-Renyi.

Proof of i: Success probability is $1 - \mathcal{O}(1/m)$



Keys and buckets in the infinite loop

Assume $\bar{E}$ occurs, i.e. an insertion fails due to an infinite loop. Let $G^* = (V^*, E^*)$ be the subgraph of G with

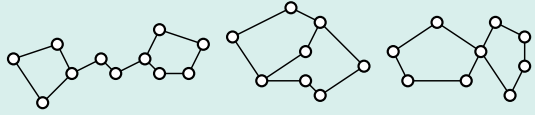
- V^*: table positions touched in the infinite loop
- E^*: keys touched in the infinite loop.

Properties of G^*:

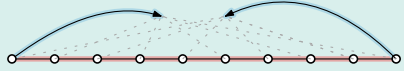
- connected
- $|E^*| = |V^*| + 1$ // can you see why?
- $\deg_{E^*}(v) \geq 2$ for $v \in V^*$.

Possibilities for G^*

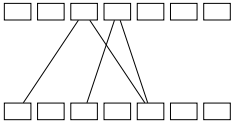
There are three options:



In all three cases: Simple path through $|V^*|$ and two extra edges connecting inwards:



Proof of i: Success probability is $1 - \mathcal{O}(1/m)$



Keys and buckets in the infinite loop

Assume $\bar{E}$ occurs, i.e. an insertion fails due to an infinite loop. Let $G^* = (V^*, E^*)$ be the subgraph of G with

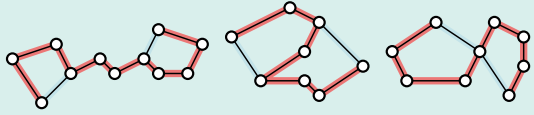
- V^*: table positions touched in the infinite loop
- E^*: keys touched in the infinite loop.

Properties of G^*:

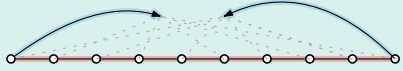
- connected
- $|E^*| = |V^*| + 1$ // can you see why?
- $\deg_{E^*}(v) \geq 2$ for $v \in V^*$.

Possibilities for G^*

There are three options:

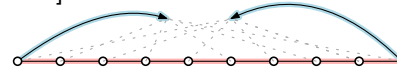


In all three cases: Simple path through $|V^*|$ and two extra edges connecting inwards:



Proof of i: Success probability is $1 - \mathcal{O}(1/m)$

$$\begin{aligned}
\Pr[\bar{E}] &= \Pr[\exists \text{path as shown}] \\
&= \Pr[\exists k \in \mathbb{N} : \exists x_0, \ldots, x_{k+1} \in S : x_0, \ldots, x_{k+1} \text{ form a path as shown}] \\
&\overset{\text{union bound}}{\leq} \sum_{k=1}^{n} \sum_{x_0,\ldots,x_{k+1} \in S} \Pr[x_0, \ldots, x_{k+1} \text{ form a path as shown}] \\
&\leq \sum_{k=1}^{n} \underbrace{n^{k+2}}_{\text{a}} \cdot \underbrace{2}_{\text{b}} \cdot \underbrace{\tfrac{1}{m^{k+1}}}_{\text{c}} \cdot \underbrace{\left(\tfrac{k+1}{2m}\right)^2}_{\text{d}} \\
&\leq \tfrac{1}{2} \sum_{k=1}^{n} m^{k+2-k-1-2} (1-\beta)^{k+2} (k+1)^2 \\
&\leq \tfrac{1}{2m} \sum_{k=1}^{\infty} (1-\beta)^{k+2} (k+1)^2 = \tfrac{1}{m} \cdot \mathcal{O}\left(\tfrac{1}{\beta^3}\right) = \mathcal{O}\left(\tfrac{1}{m}\right) \quad \square
\end{aligned}$$



- a Choose sequence of $k+2$ keys.
- b Choose to start in top or bottom table.
- c Neighbouring keys share a hash.
- d Two bordering keys connect back inward.

Proof of i: Success probability is $1 - \mathcal{O}(1/m)$

$$\Pr[\bar{E}] = \Pr[\exists \text{path as shown}]$$

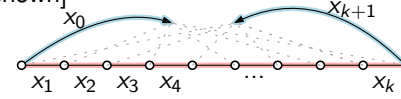
$$= \Pr[\exists k \in \mathbb{N} : \exists x_0, \ldots, x_{k+1} \in S : x_0, \ldots, x_{k+1} \text{ form a path as shown}]$$

$$\overset{\text{union bound}}{\leq} \sum_{k=1}^{n} \sum_{x_0,\ldots,x_{k+1} \in S} \Pr[x_0, \ldots, x_{k+1} \text{ form a path as shown}]$$

$$\leq \sum_{k=1}^{n} \underbrace{n^{k+2}}_{\text{a}} \cdot \underbrace{2}_{\text{b}} \cdot \underbrace{\tfrac{1}{m^{k+1}}}_{\text{c}} \cdot \underbrace{\left(\tfrac{k+1}{2m}\right)^2}_{\text{d}}$$

$$\leq \tfrac{1}{2} \sum_{k=1}^{n} m^{k+2-k-1-2}(1-\beta)^{k+2}(k+1)^2$$

$$\leq \tfrac{1}{2m} \sum_{k=1}^{\infty} (1-\beta)^{k+2}(k+1)^2 = \tfrac{1}{m} \cdot \mathcal{O}\left(\tfrac{1}{\beta^3}\right) = \mathcal{O}\left(\tfrac{1}{m}\right) \quad \square$$



- a Choose sequence of $k+2$ keys.
- b Choose to start in top or bottom table.
- c Neighbouring keys share a hash.
- d Two bordering keys connect back inward.

Proof of i: Success probability is $1 - \mathcal{O}(1/m)$

$$\Pr[\bar{E}] = \Pr[\exists \text{path as shown}]$$

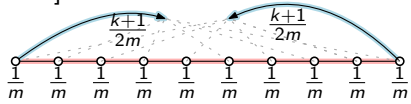
$$= \Pr[\exists k \in \mathbb{N} : \exists x_0, \ldots, x_{k+1} \in S : x_0, \ldots, x_{k+1} \text{ form a path as shown}]$$

$$\overset{\text{union bound}}{\leq} \sum_{k=1}^{n} \sum_{x_0,\ldots,x_{k+1} \in S} \Pr[x_0, \ldots, x_{k+1} \text{ form a path as shown}]$$

$$\leq \sum_{k=1}^{n} \underbrace{n^{k+2}}_{\text{a}} \cdot \underbrace{2}_{\text{b}} \cdot \underbrace{\tfrac{1}{m^{k+1}}}_{\text{c}} \cdot \underbrace{\left(\tfrac{k+1}{2m}\right)^2}_{\text{d}}$$

$$\leq \tfrac{1}{2} \sum_{k=1}^{n} m^{k+2-k-1-2}(1-\beta)^{k+2}(k+1)^2$$

$$\leq \tfrac{1}{2m} \sum_{k=1}^{\infty} (1-\beta)^{k+2}(k+1)^2 = \tfrac{1}{m} \cdot \mathcal{O}\left(\tfrac{1}{\beta^3}\right) = \mathcal{O}\left(\tfrac{1}{m}\right) \quad \square$$



- a Choose sequence of $k + 2$ keys.
- b Choose to start in top or bottom table.
- c Neighbouring keys share a hash.
- d Two bordering keys connect back inward.

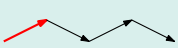
Proof of ii: Expected insertion time is $\mathcal{O}(1)$

Karlsruhe Institute of Technology

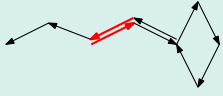
Lemma

If the insertion of y takes $t \in \mathbb{N}$ steps then the cuckoo graph G contained (previously) a path of length $\lceil (t-2)/3 \rceil$ starting from $h_0(y)$ or from $h_1(y)$.

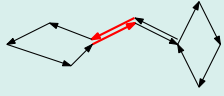
Proof.



no turning back
$\rightsquigarrow$ path of length $t-1$
starting from $h_0(y)$



turn back once
$\rightsquigarrow$ path of length $\lceil (t-2)/3 \rceil$
starting from $h_0(y)$ or $h_1(y)$



turn back twice
impossible: insertion would fail

□

Proof of ii: Expected insertion time is $\mathcal{O}(1)$ (continued)

$$\mathbb{E}[T] = \sum_{t\geq 1} \Pr[T \geq t] \qquad \text{tail sum formula}$$

$$\leq \sum_{t\geq 1} \Pr[\exists \text{path of length } \lceil (t-2)/3 \rceil \text{ starting from } h_0(y) \text{ or } h_1(y)] \qquad \text{by Lemma}$$

$$\leq 2 \cdot \sum_{t\geq 1} \Pr[\exists \text{path of length } \lceil (t-2)/3 \rceil \text{ starting from } h_0(y)] \qquad \text{union bound + symmetry}$$

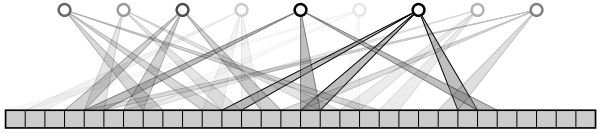
$$\leq 2\left(2 + 3 \cdot \sum_{t\geq 1} \Pr[\exists \text{path of length } t \text{ starting from } h_0(y)]\right) \qquad \sum_{i\geq 1} f(\lceil t/3 \rceil) = 3 \cdot f(1) + 3 \cdot f(2) + \ldots$$

$$\leq 4 + 6 \cdot \sum_{t\geq 1} \sum_{x_1,\ldots,x_t \in S} \Pr[x_1, \ldots, x_t \text{ form path starting from } h_0(y)] \qquad \text{union bound}$$

$$\leq 4 + 6 \cdot \sum_{t\geq 1} n^t m^{-t} \leq 6 \sum_{t\geq 0} (1-\beta)^t = 6/\beta = \mathcal{O}(1). \quad \square$$

Content

Cuckoo Hashing with one table and k hash functions



$n \in \mathbb{N}$ keys

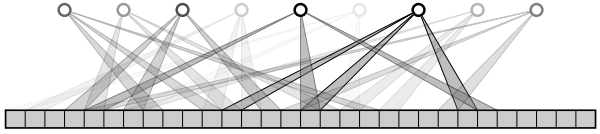
$m \in \mathbb{N}$ table size

$\alpha = \frac{n}{m}$ load factor

$h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$ hash functions

$\hookrightarrow$ Could also use a separate table per hash function.

Cuckoo Hashing with one table and k hash functions



$n \in \mathbb{N}$	keys
$m \in \mathbb{N}$	table size
$\alpha = \frac{n}{m}$	load factor
$h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$	hash functions

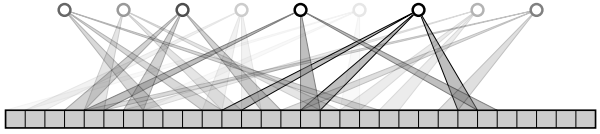
↪ Could also use a separate table per hash function.

randomWalkInsert(x)

while $x \neq \perp$ **do** // TODO: limit
  sample $i \sim \mathcal{U}([k])$
  swap$(x, T[h_i(x)])$

(some improvements possible)

Cuckoo Hashing with one table and k hash functions



$n \in \mathbb{N}$	keys
$m \in \mathbb{N}$	table size
$\alpha = \frac{n}{m}$	load factor
$h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$	hash functions

$\hookrightarrow$ Could also use a separate table per hash function.

randomWalkInsert(x)

while $x \neq \perp$ **do** // TODO: limit
  sample $i \sim \mathcal{U}([k])$
  swap($x, T[h_i(x)]$)

(some improvements possible)

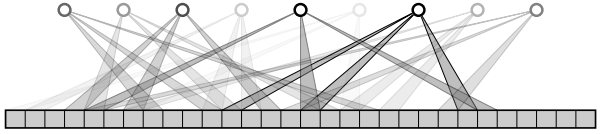
Theorem (without proof)

For each $k \in \mathbb{N}$ there is a **threshold** c_k^* such that:

- if $\alpha < c_k^*$ all keys can be placed with probability $1 - \mathcal{O}(\frac{1}{m})$.
- if $\alpha > c_k^*$ **not** all keys can be placed with probability $1 - \mathcal{O}(\frac{1}{m})$.

$c_2^* = \frac{1}{2}, \quad c_3^* \approx 0.92, \quad c_4^* \approx 0.98, \ldots$

Cuckoo Hashing with one table and k hash functions



$n \in \mathbb{N}$ keys
$m \in \mathbb{N}$ table size
$\alpha = \frac{n}{m}$ load factor
$h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$ hash functions
↪ Could also use a separate table per hash function.

randomWalkInsert(x)

while $x \neq \perp$ **do** // TODO: limit
  sample $i \sim \mathcal{U}([k])$
  swap($x, T[h_i(x)]$)

(some improvements possible)

Theorem (without proof)

For each $k \in \mathbb{N}$ there is a **threshold** c_k^* such that:

- if $\alpha < c_k^*$ all keys can be placed with probability $1 - \mathcal{O}(\frac{1}{m})$.
- if $\alpha > c_k^*$ **not** all keys can be placed with probability $1 - \mathcal{O}(\frac{1}{m})$.

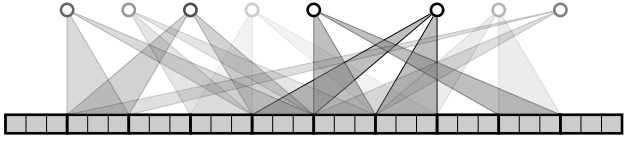
$c_2^* = \frac{1}{2}$, $c_3^* \approx 0.92$, $c_4^* \approx 0.98, \ldots$

Theorem (Bell, Frieze, 2024; retracted in 2025; re-announced for 2026)

If $k \geq 3$ and $\alpha < c_k^*$ then, conditioned on a high probability event[a], the expected insertion time is $\mathcal{O}(1)$.

[a] Without this conditioning, randomWalkInsert might be trapped in an infinite loop.

Cuckoo Hashing with k buckets of size ℓ

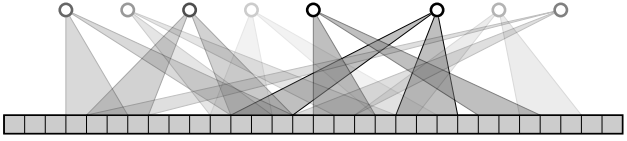


- picture illustrates $k = 2, \ell = 3$
- $k = 2$ has best cache efficiency
- larger ℓ improves space efficiency

Thresholds for the load factor

$\ell \backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.8970118682	0.9882014140	0.9982414840	0.9997243601	0.9999568737	0.9999933439
3	0.9591542686	0.9972857393	0.9997951434	0.9999851453	0.9999989795	0.9999999329
4	0.9803697743	0.9992531564	0.9999720661	0.9999990737	0.9999999721	0.9999999992
5	0.9895513619	0.9997746588	0.9999958681	0.9999999374	0.9999999992	≈ 1
6	0.9940727066	0.9999281468	0.9999993570	0.9999999956	≈ 1	≈ 1

Cuckoo Hashing with k unaligned blocks of size ℓ



- picture illustrates $k = 2, \ell = 3$
- note: unaligned block may cross cache line boundary

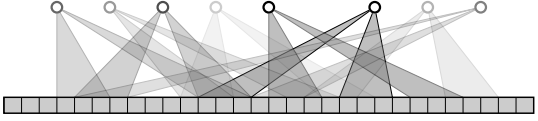
Thresholds for the load factor (slightly better than for buckets)

$\ell \backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.9649949234	0.9968991072	0.9996335076	0.9999529036	0.9999937602	0.9999991631
3	0.9944227538	0.9998255112	0.9999928198	0.9999996722	0.9999999843	0.9999999992
4	0.9989515932	0.9999896830	0.9999998577	0.9999999977	≈ 1	≈ 1
5	0.9997922174	0.9999993863	0.9999999972	≈ 1	≈ 1	≈ 1
6	0.9999581007	0.9999999635	0.9999999999	≈ 1	≈ 1	≈ 1

Back to Approximate Membership: Cuckoo Filters

General Idea

- construct cuckoo table for key set $S \subseteq D$
- store $\text{fp}(x)$ instead of $x \in S$ for random *fingerprint function* $\text{fp} : D \to \{0,1\}^r$
- query(x) checks for $\text{fp}(x)$ in positions associated with x



State of the Art Variant[1]

- uses $k = 2$ unaligned blocks of size $\ell = 2$
  - threshold ≈ 0.965
  - queries check 4 positions
- false positive probability $\varepsilon \leq 4 \cdot 2^{-r} = 2^{-r+2}$
- space $\approx \frac{r}{0.965} n = 1.04(\log_2(1/\varepsilon) + 2)n$ bits
  // compared to Bloom $\approx 1.44 \log_2(1/\varepsilon)n$ bits

Supporting Insertions and Deletions

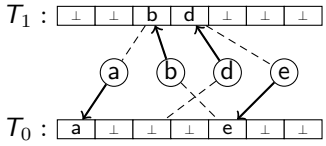
Complication: Need to evict fingerprints $\text{fp}(x)$ without knowing x.
Solution: Positions and fingerprints are related. Tricky details.

[1] Schmitz, Zentgraf, Rahman: Smaller and More Flexible Cuckoo Filters, ALENEX 2026.

Conclusion

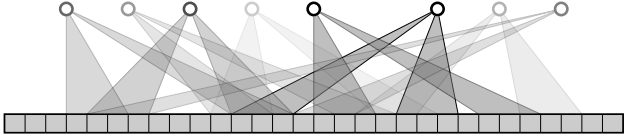
Classic Cuckoo Hashing

- hash table with *worst case* constant access times
- analysis considers paths in graphs similar to the Erdős-Renyi model



Practical Cuckoo Hashing

- uses buckets (or unaligned blocks) e.g. $k = 2$ buckets of size $\ell = 8$
- better than conventional hash tables, if high load factors are needed
- cuckoo filters are state of the art dynamic AMQ data structures
- *very* difficult to analyse



Anhang: Mögliche Prüfungsfragen I

- Was ist und was kann Cuckoo Hashing?
  - Was ist die Grundidee? Wie funktionieren die Operationen?
  - Worauf ist bei der Wahl der Tabellengröße / beim Load Factor zu achten?
  - Was kann man über die Laufzeit der Operationen sagen?
  - Welche Vorteile und Nachteile ergeben sich im Vergleich zu anderen Techniken wie linearem Sondieren?
- Analyse:
  - Eine Einfügung, die fehlschlägt, entspricht gewissen Strukturen im Cuckoo-Graphen. Welchen?
  - Wie haben wir gezeigt, dass solche Strukturen unwahrscheinlich sind?
  - Wie haben wir die erwartete Einfügezeit abgeschätzt?