Probability and Computing – Cuckoo Hashing

Stefan Walzer, Maximilian Katzmann | WS 2023/2024
Probability and Computing – Cuckoo Hashing

Stefan Walzer, Maximilian Katzmann | WS 2023/2024
Probability and Computing – Cuckoo Hashing

Stefan Walzer, Maximilian Katzmann | WS 2023/2024
1. Cuckoo Hashing
   - Algorithm
   - Analysis
Cuckoo Hashing

Setup

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

\[
\begin{align*}
T_1 : & \quad \perp \perp a \quad d \quad \perp \perp \perp \\
T_0 : & \quad \perp \perp \perp \perp \perp b \quad \perp \perp
\end{align*}
\]

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
  - \( b \leftarrow i \mod 2 \)
  - swap(\( x, T_b[h_b(x)] \))
  - if \( x = \perp \) then
    - return SUCCESS
  - return FAILURE
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 \( \beta > 0 \)
- (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
  - \( T_0[h_0(x)] \leftarrow \bot \)
- else if \( T_1[h_1(x)] = x \) then
  - \( T_1[h_1(x)] \leftarrow \bot \)

Algorithm insert(\( x \)):

- for \( i = 0 \) to LIMIT do
  - \( b \leftarrow i \mod 2 \)
  - swap(\( x, T_b[h_b(x)] \))
  - if \( x = \bot \) then
    - return SUCCESS
  - return FAILURE
Cuckoo Hashing

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

**Algorithm delete(x):**
\[
\begin{align*}
&\text{if } T_0[h_0(x)] = x \text{ then} \\
&T_0[h_0(x)] \leftarrow \bot
\end{align*}
\]
\[
\begin{align*}
&\text{else if } T_1[h_1(x)] = x \text{ then} \\
&T_1[h_1(x)] \leftarrow \bot
\end{align*}
\]

**Algorithm insert(x):**
\[
\begin{align*}
&\text{for } i = 0 \text{ to LIMIT do} \\
&b \leftarrow i \mod 2 \\
&\text{swap}(x, T_b[h_b(x)]) \\
&\text{if } x = \bot \text{ then} \\
&\text{return SUCCESS} \\
&\text{return FAILURE}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \subseteq D ) key set of size ( n )</td>
</tr>
<tr>
<td>( T_0, T_1 ) two tables of size ( m )</td>
</tr>
<tr>
<td>( h_0, h_1 \sim \mathcal{U}(\lbrack m \rbrack^D) ) two hash functions (SUHA)</td>
</tr>
<tr>
<td>( \frac{n}{m} = 1 - \beta ) for some ( \beta &gt; 0 )</td>
</tr>
<tr>
<td>(load factor ( \alpha = \frac{n}{2m} ))</td>
</tr>
</tbody>
</table>

- **\( T_0 \):**
  - \( \bot \quad \bot \quad \text{a} \quad \text{d} \quad \bot \quad \bot \quad \bot \)

- **\( T_1 \):**
  - \( \text{a} \quad \text{b} \quad \text{d} \quad \text{e} \)
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 ∈ \{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)] ← ⊥
else if T_1[h_1(x)] = x then
  T_1[h_1(x)] ← ⊥
```

**Algorithm insert(x):**

```
for i = 0 to LIMIT do
  b ← i mod 2
  swap(x, T_b[h_b(x)])
  if x = ⊥ then
    return SUCCESS
return FAILURE
```
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 $\beta > 0$

\[ \Delta \text{ load factor } \alpha = \frac{n}{2m} \]

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
  - $b \leftarrow i \mod 2$
  - 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

$b \leftarrow i \mod 2$

swap($x$, $T_b[h_b(x)]$)

if $x = \bot$ then

\[ \text{return } \text{SUCCESS} \]

return FAILURE

### Theorem (Analysis with $\text{LIMIT} = \infty$)

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

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

Then

\[ \mathbb{P}[E] = 1 - O(1/m) \]

and

\[ \mathbb{E}[T] = O(1) \]

### Theorem (full analysis, not here)

If we

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

we obtain a hash table where lookup and delete take $O(1)$ time and insert takes expected $O(1)$ time.
The Cuckoo Graph

Consider the bipartite *cuckoo graph*

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

the key \(x\) corresponds to the edge \((h_0(x), h_1(x))\) and each table position to a vertex.
Proof of $i$: Success probability is $1 - \mathcal{O}(1/m)$

Assume $\tilde{E}$ 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 infinitely often
- $E^*$: keys touched infinitely often.

Properties of $G^*$:

- connected
- $|E^*| = |V^*| + 1$ can you see why?
- $\text{deg}_{E^*}(v) \geq 2$.

Possibilities for $G^*$

There are three options:

In all three cases: Simple path through $|V^*|$ and two extra edges connecting inwards:
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 infinitely often
- $E^*$: keys touched infinitely often.

Properties of $G^*$:
- connected
- $|E^*| = |V^*| + 1$ can you see why?
- $\deg_{E^*}(v) \geq 2$.

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 $1 - \mathcal{O}(1/m)$

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

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

union bound

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

$\leq \sum_{k=1}^{n} n^{k+2} \cdot 2 \cdot \frac{1}{m^{k+1}} \cdot \left( \frac{k+1}{2m} \right)^2$

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

$\leq \frac{1}{2m} \sum_{k=1}^{\infty} (1 - \beta)^{k+2} (k + 1)^2 = \frac{1}{m} \cdot \mathcal{O}(\frac{1}{\beta^3}) = \frac{1}{m} \cdot \mathcal{O}(1)$
Proof of (i): Success probability is $1 - \mathcal{O}(1/m)$

\[
\Pr[\tilde{E}] = \Pr[\exists \text{path as shown}]
= \Pr[\exists k \in \mathbb{N} : \exists x_0, \ldots, x_{k+1} \in S : x_0, \ldots, x_{k+1} \text{ form a path as shown}]
\leq \sum_{k=1}^{n} \sum_{x_0, \ldots, x_{k+1} \in S} \Pr[x_0, \ldots, x_{k+1} \text{ form a path as shown}]
\leq \sum_{k=1}^{n} n^{k+2} \cdot \frac{2}{m^{k+1}} \cdot \left( \frac{k+1}{2m} \right)^2
\leq \frac{1}{2} \sum_{k=1}^{n} m^{k+2-k-1-2} (1 - \beta)^{k+2}(k + 1)^2
\leq \frac{1}{2m} \sum_{k=1}^{\infty} (1 - \beta)^{k+2}(k + 1)^2 = \frac{1}{m} \cdot \mathcal{O}(\frac{1}{\beta^3}) = \frac{1}{m} \cdot \mathcal{O}(1)
\]

- **a**: Choose sequence of $k + 2$ keys.
- **b**: Choose to start in left or right table.
- **c**: Neighbouring keys share a hash.
- **d**: Two bordering keys connect back inward.
Proof of \( i \): Success probability is \( 1 - \mathcal{O}(1/m) \)

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

union bound

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

\[
\leq \sum_{k=1}^{n} n^{k+2} \cdot \frac{2}{m^{k+1}} \cdot \left( \frac{k+1}{2m} \right)^2
\]

\[
\leq \frac{1}{2} \sum_{k=1}^{n} m^{k+2-k-1-2}(1 - \beta)^{k+2}(k + 1)^2
\]

\[
\leq \frac{1}{2m} \sum_{k=1}^{\infty} (1 - \beta)^{k+2}(k + 1)^2 = \frac{1}{m} \cdot \mathcal{O}(\frac{1}{\beta^3}) = \frac{1}{m} \cdot \mathcal{O}(1)
\]

\( \square \)
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 \( \boxed{\text{ii}} \): Expected insertion time is \( \mathcal{O}(1) \) (continued)

\[
\mathbb{E}[T] = \sum_{t \geq 1} \Pr[T \geq t]
\]
\[
\leq \sum_{t \geq 1} \Pr[\exists \text{path of length } \lceil (t - 2)/3 \rceil \text{ starting from } h_0(y) \text{ or } h_1(y)] \quad \text{by Lemma}
\]
\[
\leq 2 \cdot \sum_{t \geq 1} \Pr[\exists \text{path of length } \lceil (t - 2)/3 \rceil \text{ starting from } h_0(y)] \quad \text{union bound + symmetry}
\]
\[
\leq 2 \left( 2 + 3 \cdot \sum_{t \geq 1} \Pr[\exists \text{path of length } t \text{ starting from } h_0(y)] \right)
\]
\[
\leq 4 + 6 \cdot \sum_{t \geq 1} \Pr[x_1, \ldots, x_t \text{ form path starting from } h_0(y)] \quad \text{union bound}
\]
\[
\leq 4 + 6 \cdot \sum_{t \geq 1} n^t m^{-t} = 6 \sum_{t \geq 0} (1 - \beta)^t = \mathcal{O}(1/\beta) = \mathcal{O}(1).
\]
Conclusion

Cuckoo Hashing

- hash table with *worst case* constant access times
- analysis considers path in graphs similar to the Erdős-Rényi model
- many variations and spin-offs (not discussed here)
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?