14. Hashing

Hash Tables, Pre-Hashing, Hashing, Resolving Collisions using Chaining, Simple Uniform Hashing, Popular Hash Functions, Table-Doubling, Open Addressing: Probing, Uniform Hashing, Universal Hashing, Perfect Hashing [Ottman/Widmayer, Kap. 4.1-4.3.2, 4.3.4, Cormen et al, Kap. 11-11.4]

Motivating Example

Gloal: Efficient management of a table of all n ETH-students of
Possible Requirement: fast access (insertion, removal, find) of a dataset by name

Dictionary

Abstract Data Type (ADT) D to manage items[16] i with keys $k \in \mathcal{K}$ with operations

- **`D.insert`**(i): Insert or replace i in the dictionary D.
- **`D.delete`**(i): Delete i from the dictionary D. Not existing $\Rightarrow$ error message.
- **`D.search`**(k): Returns item with key k if it exists.

[16] Key-value pairs (k, v), in the following we consider mainly the keys

Dictionary in $C{+}{+}$

Associative Container `std::unordered_map<>`

```cpp
// Create an unordered_map of strings that map to strings
std::unordered_map<std::string, std::string> u = {
  {"RED","#FF0000"}, {"GREEN","#00FF00"}
};

u["BLUE"] = "#0000FF"; // Add

std::cout << "The HEX of color RED is: " << u["RED"] << "\n";

for( const auto& n : u ) // iterate over key-value pairs
  std::cout << n.first << ":" << n.second << "\n";
```

Motivation / Use

Perhaps **the** most popular data structure.

- Supported in many programming languages (C++, Java, Python, Ruby, Javascript, C# ...)
- Obvious use
  - Databases, Spreadsheets
  - Symbol tables in compilers and interpreters
- Less obvious
  - Substrin Search (Google, grep)
  - String commonalities (Document distance, DNA)
  - File Synchronisation
  - Cryptography: File-transfer and identification

1. Idea: Direct Access Table (Array)

Index	Item
0	-
1	-
2	-
3	[3,value(3)]
4	-
5	-
⋮	⋮
k	[k,value(k)]
⋮	⋮

Problems

1. Keys must be non-negative integers
2. Large key-range $\Rightarrow$ large array

Solution to the first problem: Pre-hashing

Prehashing: Map keys to positive integers using a function $ph : \mathcal{K} \to \mathbb{N}$

- Theoretically always possible because each key is stored as a bit-sequence in the computer
- Theoretically also: $x = y \Leftrightarrow ph(x) = ph(y)$
- Practically: APIs offer functions for pre-hashing. (Java: **`object.hashCode()`**, C++: **`std::hash<>`**, Python: **`hash(object)`**)
- APIs map the key from the key set to an integer with a restricted size.[17]

[17] Therefore the implication $ph(x) = ph(y) \Rightarrow x = y$ does **not** hold any more for all x,y.

Prehashing Example : String

Mapping Name $s = s_1 s_2 \dots s_{l_s}$ to key

$$ph(s) = \left(\sum_{i=0}^{l_s - 1} s_{l_s - i} \cdot b^i \right) \bmod 2^w$$

b so that different names map to different keys as far as possible.

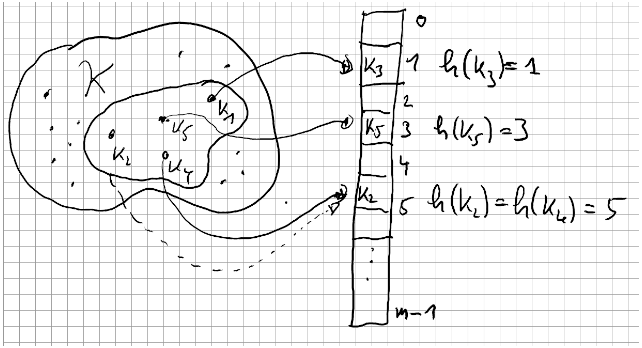
b Word-size of the system (e.g. 32 or 64)

Example (Java) with $b = 31$, $w = 32$. Ascii-Values s_i.

Anna $\mapsto 2045632$
Jacqueline $\mapsto 2042089953442505 \bmod 2^{32} = 507919049$

Lösung zum zweiten Problem: Hashing

Reduce the universe. Map (hash-function) $h : \mathcal{K} \rightarrow \{0, ..., m-1\}$ ($m \approx n =$ number entries of the table)



Collision: $h(k_i) = h(k_j)$.

Nomenclature

Hash funtion h: Mapping from the set of keys $\mathcal{K}$ to the index set $\{0, 1, \ldots, m-1\}$ of an array (**hash table**).

$$h : \mathcal{K} \rightarrow \{0, 1, \ldots, m-1\}.$$

Normally $|\mathcal{K}| \gg m$. There are $k_1, k_2 \in \mathcal{K}$ with $h(k_1) = h(k_2)$ (**collision**).

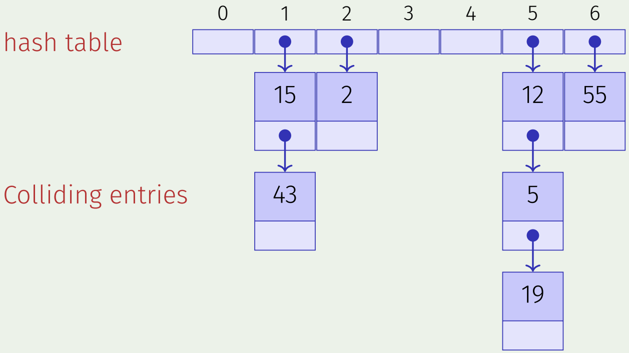
A hash function should map the set of keys as uniformly as possible to the hash table.

Resolving Collisions: Chaining

$m = 7, \mathcal{K} = \{0, \ldots, 500\}, h(k) = k \bmod m.$

Keys 12 , 55 , 5 , 15 , 2 , 19 , 43

Direct Chaining of the Colliding entries



Algorithm for Hashing with Chaining

- **`insert`**(i) Check if key k of item i is in list at position $h(k)$. If no, then append i to the end of the list. Otherwise replace element by i.
- **`find`**(k) Check if key k is in list at position $h(k)$. If yes, return the data associated to key k, otherwise return empty element **`null`**.
- **`delete`**(k) Search the list at position $h(k)$ for k. If successful, remove the list element.

Worst-case Analysis

Worst-case: all keys are mapped to the same index.

$\Rightarrow \Theta(n)$ per operation in the worst case. ☹

Simple Uniform Hashing

Strong Assumptions: Each key will be mapped to one of the m available slots

- with equal probability (Uniformity)
- and independent of where other keys are hashed (Independence).

Simple Uniform Hashing

Under the assumption of simple uniform hashing:
Expected length of a chain when n elements are inserted into a hash table with m elements

$$\mathbb{E}(\text{Länge Kette j}) = \mathbb{E}\left(\sum_{i=0}^{n-1} \mathbb{1}(k_i = j)\right) = \sum_{i=0}^{n-1} \mathbb{P}(k_i = j)$$
$$= \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m}$$

$\alpha = n/m$ is called **load factor** of the hash table.

Simple Uniform Hashing

Theorem 16

Let a hash table with chaining be filled with load-factor $\alpha = \frac{n}{m} < 1$. Under the assumption of simple uniform hashing, the next operation has expected costs of $\leq 1 + \alpha$.

Consequence: if the number slots m of the hash table is always at least proportional to the number of elements n of the hash table, $n \in \mathcal{O}(m) \Rightarrow$ Expected Running time of Insertion, Search and Deletion is $\mathcal{O}(1)$.

Further Analysis (directly chained list)

1. Unsuccesful search. The average list lenght is $\alpha = \frac{n}{m}$. The list has to be traversed completely.
   $\Rightarrow$ Average number of entries considered

$$C'_n = \alpha.$$

2. Successful search Consider the insertion history: key j sees an average list length of $(j-1)/m$.
   $\Rightarrow$ Average number of considered entries

$$C_n = \frac{1}{n}\sum_{j=1}^{n}(1+(j-1)/m)) = 1 + \frac{1}{n}\frac{n(n-1)}{2m} \approx 1 + \frac{\alpha}{2}.$$

Advantages and Disadvantages of Chaining

Advantages

- Possible to overcommit: $\alpha > 1$ allowed
- Easy to remove keys.

Disadvantages

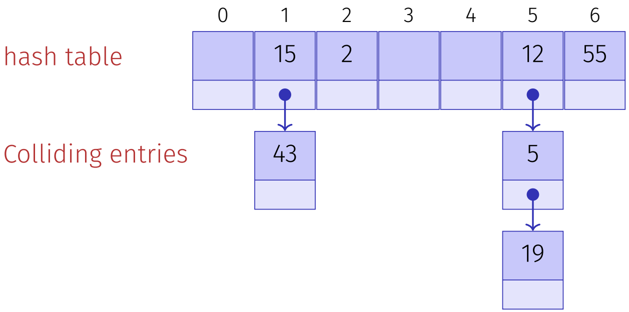
- Memory consumption of the chains-

[Variant:Indirect Chaining]

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2 , 19 , 43

Indirect chaining the Collisions



Examples of popular Hash Functions

$$h(k) = k \bmod m$$

Ideal: m prime, not too close to powers of 2 or 10
But often: $m = 2^k - 1$ ($k \in \mathbb{N}$)

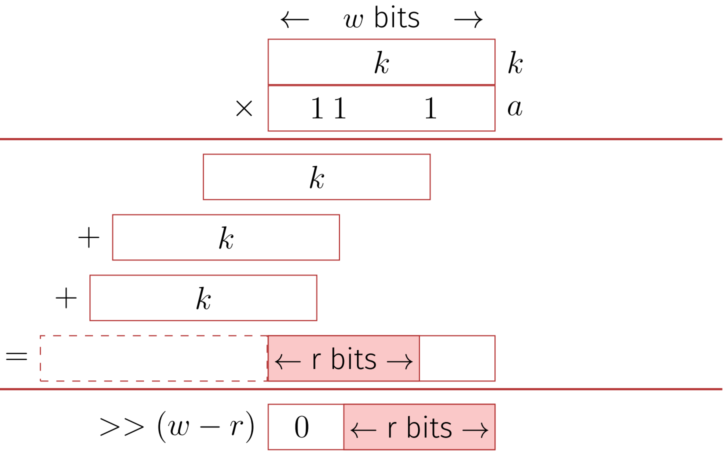
Examples of popular Hash Functions

Multiplication method

$$h(k) = \left\lfloor (a \cdot k \bmod 2^w)/2^{w-r} \right\rfloor \bmod m$$

- $m = 2^r$, $w =$ size of the machine word in bits.
- Multiplication adds k along all bits of a, integer division with 2^{w-r} and $\mathrm{mod} m$ extract the upper r bits.
- Written as code `a * k >> (w-r)`
- A good value of a: $\left\lfloor \frac{\sqrt{5}-1}{2} \cdot 2^w \right\rfloor$: Integer that represents the first w bits of the fractional part of the irrational number.

Illustration



$\leftarrow$ w bits $\rightarrow$
k
k
$\times$
1 1 1
a
k
$+$ k
$+$ k
$=$
$\leftarrow$ r bits $\rightarrow$
$>> (w - r)$
0
$\leftarrow$ r bits $\rightarrow$

Table size increase

- We do not know beforehand how large n will be
- Require $m = \Theta(n)$ at all times.

Table size needs to be adapted. Hash-Function changes $\Rightarrow$ **rehashing**

- Allocate array A' with size $m' > m$
- Insert each entry of A into A' (with re-hashing the keys)
- Set $A \leftarrow A'$.
- Costs $\mathcal{O}(n + m + m')$.

How to choose m'?

Table size increase

- 1.Idea $n = m \Rightarrow m' \leftarrow m + 1$
  Increase for each insertion: Costs $\Theta(1 + 2 + 3 + \cdots + n) = \Theta(n^2)$ ☹
- 2.Idea $n = m \Rightarrow m' \leftarrow 2m$ Increase only if$m = 2^i$:
  $\Theta(1 + 2 + 4 + 8 + \cdots + n) = \Theta(n)$
  Few insertions cost linear time but on average we have $\Theta(1)$ ☺

Jede Operation vom Hashing mit Verketten hat erwartet amortisierte Kosten $\Theta(1)$.
($\Rightarrow$ Amortized Analysis)

Open Addressing

Store the colliding entries directly in the hash table using a **probing function** $s : \mathcal{K} \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$
Key table position along a **probing sequence**

$$S(k) := (s(k,0), s(k,1), \ldots, s(k, m-1)) \mod m$$

Probing sequence must for each $k \in \mathcal{K}$ be a permutation of $\{0, 1, \ldots, m-1\}$

Notational clarification: this method uses **open addressing**(meaning that the positions in the hashtable are not fixed) but it is a **closed hashing** procedure (because the entries stay in the hashtable)

Algorithms for open addressing

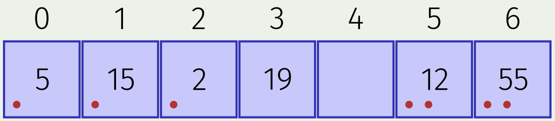
- **`insert`**(i) Search for kes k of i in the table according to $S(k)$. If k is not present, insert k at the first free position in the probing sequence. Otherwise error message.
- **`find`**(k) Traverse table entries according to $S(k)$. If k is found, return data associated to k. Otherwise return an empty element **`null`**.
- **`delete`**(k) Search k in the table according to $S(k)$. If k is found, replace it with a special key **`removed`**.

Linear Probing

$$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \dots, h(k)+m-1) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Key 12 , 55 , 5 , 15 , 2 , 19



[Analysis linear probing (without proof)]

1. Unsuccessful search. Average number of considered entries

$$C'_n \approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

2. Successful search. Average number of considered entries

$$C_n \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right).$$

Discussion

Example $\alpha = 0.95$

The unsuccessful search consideres 200 table entries on average! (here without derivation).

Disadvantage of the method?

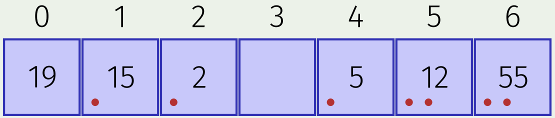
Primary clustering: similar hash addresses have similar probing sequences $\Rightarrow$ long contiguous areas of used entries.

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2 , 19



[Analysis Quadratic Probing (without Proof)]

1. Unsuccessful search. Average number of entries considered

$$C'_n \approx \frac{1}{1-\alpha} - \alpha + \ln\left(\frac{1}{1-\alpha}\right)$$

2. Successful search. Average number of entries considered

$$C_n \approx 1 + \ln\left(\frac{1}{1-\alpha}\right) - \frac{\alpha}{2}.$$

Discussion

Example $\alpha = 0.95$

Unsuccessfuly search considers 22 entries on average (here without derivation)

Problems of this method?

Secondary clustering: Synonyms k and k' (with $h(k) = h(k')$) travers the same probing sequence.

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \dots, h(k) + (m-1)h'(k)) \mod m$

$m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Keys 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12 ● ●	55 ●

Double Hashing

- Probing sequence must permute all hash addresses. Thus $h'(k) \neq 0$ and $h'(k)$ may not divide m, for example guaranteed with m prime.
- h' should be as independent of h as possible (to avoid secondary clustering)

Independence:

$$\mathbb{P}((h(k) = h(k')) \wedge (h'(k) = h'(k'))) = \mathbb{P}(h(k) = h(k')) \cdot \mathbb{P}(h'(k) = h'(k')).$$

Independence largely fulfilled by $h(k) = k \bmod m$ and $h'(k) = 1 + k \bmod (m-2)$ (m prime).

[Analysis Double Hashing]

Let h and h' be independent, then:

1. Unsuccessful search. Average number of considered entries:

$$C'_n \approx \frac{1}{1-\alpha}$$

2. Successful search. Average number of considered entries:

$$C_n \approx \frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$$

Uniform Hashing

Strong assumption: the probing sequence $S(k)$ of a key l is equaly likely to be any of the $m!$ permutations of $\{0, 1, \ldots, m-1\}$

(Double hashing is reasonably close)

Analysis of Uniform Hashing with Open Addressing

Theorem 17

Let an open-addressing hash table be filled with load-factor $\alpha = \frac{n}{m} < 1$. *Under the assumption of uniform hashing, the next operation has expected costs of* $\leq \frac{1}{1-\alpha}$.

Analysis of Uniform Hashing with Open Addressing

Proof of the Theorem: Random Variable X: Number of probings when searching without success.

$$\mathbb{P}(X \geq i) \overset{*}{=} \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2}$$
$$\overset{**}{\leq} \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}. \qquad (1 \leq i \leq m)$$

*: A_j:Slot used during step j.
$\mathbb{P}(A_1 \cap \cdots \cap A_{i-1}) = \mathbb{P}(A_1) \cdot \mathbb{P}(A_2|A_1) \cdot ... \cdot \mathbb{P}(A_{i-1}|A_1 \cap \cdots \cap A_{i-2})$,
**: $\frac{n-1}{m-1} < \frac{n}{m}$ because[18] $n < m$.
Moreover $\mathbb{P}(x \geq i) = 0$ for $i \geq m$. Therefore

$$\mathbb{E}(X) \overset{\text{Appendix}}{=} \sum_{i=1}^{\infty} \mathbb{P}(X \geq i) \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha}.$$

[18] $\frac{n-1}{m-1} < \frac{n}{m} \Leftrightarrow \frac{n-1}{n} < \frac{m-1}{m} \Leftrightarrow 1 - \frac{1}{n} < 1 - \frac{1}{m} \Leftrightarrow n < m$ ($n > 0, m > 0$)

[Successful search of Uniform Open Hashing]

Theorem 18

Let an open-addressing hash table be filled with load-factor $\alpha = \frac{n}{m} < 1$. Under the assumption of uniform hashing, the successful search has expected costs of $\leq \frac{1}{\alpha} \cdot \log \frac{1}{1-\alpha}$.

Proof: Cormen et al, Kap. 11.4

Overview

	$\alpha = 0.50$		$\alpha = 0.90$		$\alpha = 0.95$	
	C_n	C'_n	C_n	C'_n	C_n	C'_n
(Direct) Chaining	1.25	0.50	1.45	0.90	1.48	0.95
Linear Probing	1.50	2.50	5.50	50.50	10.50	200.50
Quadratic Probing	1.44	2.19	2.85	11.40	3.52	22.05
Uniform Hashing	1.39	2.00	2.56	10.00	3.15	20.00

: C_n: Anzahl Schritte erfolgreiche Suche, C'_n: Anzahl Schritte erfolglose Suche, Belegungsgrad α.

Universal Hashing

- $|\mathcal{K}| > m \Rightarrow$ Set of "similar keys" can be chosen such that a large number of collisions occur.
- Impossible to select a "best" hash function for all cases.
- Possible, however[19]: randomize!

Universal hash class $\mathcal{H} \subseteq \{h : \mathcal{K} \to \{0, 1, \ldots, m-1\}\}$ is a family of hash functions such that

$$\forall\, k_1 \neq k_2 \in \mathcal{K} \text{ it holds that } |\{h \in \mathcal{H} \text{ with } h(k_1) = h(k_2)\}| \leq \frac{|\mathcal{H}|}{m}.$$

[19]Similar as for quicksort

Universal Hashing

Theorem 19

A function h randomly chosen from a universal class $\mathcal{H}$ of hash functions randomly distributes an arbitrary sequence of keys from $\mathcal{K}$ as uniformly as possible on the available slots.

When using hashing with chaining, the expected chain length for an element that is not contained in the table is $\leq \alpha = n/m$. The expected chain length for an element contained is $\leq 1 + \alpha$.

Universal Hashing

Initial remark for the proof of the theorem:
Define with $x, y \in \mathcal{K}$, $h \in \mathcal{H}$, $Y \subseteq \mathcal{K}$:

$$\delta(h, x, y) = \begin{cases} 1, & \text{if } h(x) = h(y) \\ 0, & \text{otherwise,} \end{cases} \qquad \text{is } h(x) = h(y) \text{ (0 or 1)?}$$

$$\delta(h, x, Y) = \sum_{y \in Y} \delta(x, y, h), \qquad \text{for how many } y \in Y \text{ is } h(x) = h(y)?$$

$$\delta(\mathcal{H}, x, y) = \sum_{h \in \mathcal{H}} \delta(x, y, h) \qquad \text{for how many } h \in \mathcal{H} \text{ is } h(x) = h(y)?.$$

$\mathcal{H}$ is universal if for all $x, y \in \mathcal{K}$, $x \neq y : \delta(\mathcal{H}, x, y) \leq |\mathcal{H}|/m$.

Universal Hashing

Proof of the theorem

$S \subseteq \mathcal{K}$: keys stored up to now. x is added now: ($x \notin S$)

Expected number of collisions of x with S

$$\begin{aligned}
\mathbb{E}_{\mathcal{H}}(\delta(h,x,S)) &= \sum_{h\in\mathcal{H}} \delta(h,x,S)/|\mathcal{H}| \\
&= \frac{1}{|\mathcal{H}|} \sum_{h\in\mathcal{H}} \sum_{y\in S} \delta(h,x,y) = \frac{1}{|\mathcal{H}|} \sum_{y\in S} \sum_{h\in\mathcal{H}} \delta(h,x,y) \\
&= \frac{1}{|\mathcal{H}|} \sum_{y\in S} \delta(\mathcal{H},x,y) \\
&\le \frac{1}{|\mathcal{H}|} \sum_{y\in S} \frac{|\mathcal{H}|}{m} = \frac{|S|}{m} = \alpha.
\end{aligned}$$

■

Universal Hashing

$S \subseteq \mathcal{K}$: keys stored up to now, now $x \in S$.

Expected number of collisions of x with S

$$\begin{aligned}
\mathbb{E}_{\mathcal{H}}(\delta(x,S,h)) &= \sum_{h\in\mathcal{H}} \delta(x,S,h)/|\mathcal{H}| \\
&= \frac{1}{|\mathcal{H}|} \sum_{h\in\mathcal{H}} \sum_{y\in S} \delta(h,x,y) = \frac{1}{|\mathcal{H}|} \sum_{y\in S} \sum_{h\in\mathcal{H}} \delta(h,x,y) \\
&= \frac{1}{|\mathcal{H}|} \left(\delta(\mathcal{H},x,x) + \sum_{y\in S-\{x\}} \delta(\mathcal{H},x,y) \right) \\
&\leq \frac{1}{|\mathcal{H}|} \left(|\mathcal{H}| + \sum_{y\in S-\{x\}} |\mathcal{H}|/m \right) = 1 + \frac{|S|-1}{m} = 1 + \frac{n-1}{m} \leq 1 + \alpha.
\end{aligned}$$

■

Construction Universal Class of Hashfunctions

Let key set be $\mathcal{K} = \{0, \dots, u-1\}$ and $p \geq u$ be prime. With $a \in \mathcal{K} \setminus \{0\}$, $b \in \mathcal{K}$ define

$$h_{ab} : \mathcal{K} \to \{0, \dots, m-1\}, h_{ab}(x) = ((ax + b) \bmod p) \bmod m.$$

Then the following theorem holds:

Theorem 20

The class $\mathcal{H} = \{h_{ab} | a, b \in \mathcal{K}, a \neq 0\}$ is a universal class of hash functions.

(Here without proof, see e.g. Cormen et al, Kap. 11.3.3)

Perfect Hashing

If the set of used keys is known up-front, the hash function can be chosen perfectly, i.e. such that there are no collisions.

Example: table of key words of a compiler.

Observation (Birthday Paradox Reversed)

- h be chosen at random from universal hashclass $\mathcal{H}$.
- n keys $S \subset \mathcal{K}$
- Random variable X : number collisionsof the n keys from$S$

$\Rightarrow$

$$\mathbb{E}(X) = \mathbb{E}\left(\sum_{i \neq j} \mathbb{1}(h(k_i) = h(k_j)\right) = \sum_{i \neq j} \mathbb{E}(\mathbb{1}(h(k_i) = h(k_j))$$
$$\overset{*}{=} \binom{n}{2}\frac{1}{m} \leq \frac{n^2}{2m}$$

* # Unordered Pairs

$\sum_{i \neq j} 1 = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} (n-1-i) = n(n-1) - n(n-1)/2 = n(n-1)/2$

Perfect Hashing with memory space $\Theta(n^2)$

if $m = n^2 \Rightarrow \mathbb{E}(X) \leq \frac{1}{2}$.

Markov-Inequality[20] $\mathbb{P}(X \geq 1) \leq \frac{\mathbb{E}(X)}{1} \leq \frac{1}{2}$

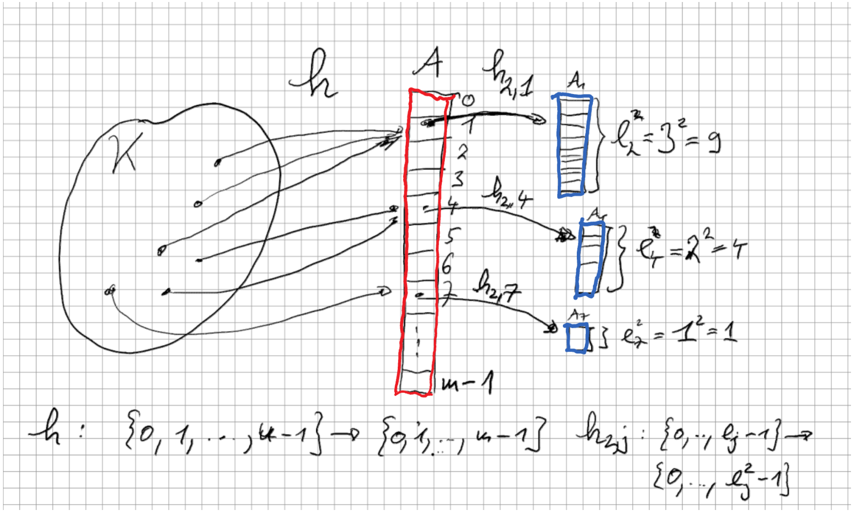
Thus

$$\mathbb{P}(X < 1) = \mathbb{P}(\text{no Collision}) \geq \frac{1}{2}.$$

Consequence: for n keys, in expected $2 \cdot n$ steps, a collision free hash-table of size $m = n^2$ can be constructed by choosing from a universal hash class at random.

[20] Appendix

Perfect Hashing Idea



h
A
$h_{2,1}$
A_1
K
0
1
2
3
4
5
6
7
m−1
$\ell_1^2 = 3^2 = 9$
$h_{2,4}$
A_4
$\ell_4^2 = 2^2 = 4$
$h_{2,7}$
A_7
$\ell_7^2 = 1^2 = 1$
$h : \{0, 1, \ldots, u-1\} \to \{0, 1, \ldots, m-1\}$
$h_{2,j} : \{0, \ldots, \ell_j - 1\} \to \{0, \ldots, \ell_j^2 - 1\}$

Perfect Hashing with $\Theta(n)$ memory consumption.

Two-level hashing

1. Choose $m = n$ and $h : \{0, 1, \ldots, u-1\} \to \{0, 1, \ldots, m-1\}$ from a universal hash-class. Insert all n keys into the hash table using chaining. Let l_i be the length of a chain at index i.
   If $\sum_{i=0}^{m-1} l_i^2 > 4n$, then repeat this step 1.
2. For each index $i = 1, \ldots, m-1$ with $l_i > 0$ construct, for the l_i contained keys, hash tables of length l_i^2 using universal hashing (hash function $h_{2,i}$) until there are no collisions.

Memory consumption $\Theta(n)$.

Expected Running times

- For Step 1: hash table of size $m = n$.
  We show on the next page that $\mathbb{E}\left(\sum_{j=0}^{m-1} l_j^2\right) \leq 2n$. Consequently (Markov):
  $\mathbb{P}\left(\sum_{j=0}^{m-1} l_j^2 \geq 4n\right) \leq \frac{2n}{4n} = \frac{1}{2}$.
  $\Rightarrow$ Expected two retries of step 1.
- For Step 2: $\sum l_i^2 \leq 4n$. For each i expected two trials with running time l_i^2.
  Overal $\mathcal{O}(n)$

$\Rightarrow$ The perfect hash tables can be constructed in expected $\mathcal{O}(n)$ steps.

Expected Memory Space 2nd Level Hash Tables

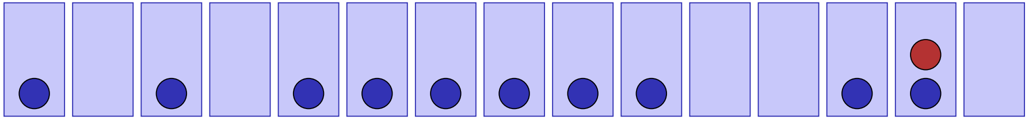
$$\begin{aligned}
\mathbb{E}\left(\sum_{j=0}^{m-1} l_j^2\right) &= \mathbb{E}\left(\sum_{j=0}^{m-1}\sum_{i=0}^{n-1}\sum_{i'=0}^{n-1} \mathbb{1}(h(k_i) = h(k_{i'}) = j)\right) \\
&= \mathbb{E}\left(\sum_{i=0}^{n-1}\sum_{i'=0}^{n-1} \mathbb{1}(h(k_i) = h(k_{i'}))\right) \\
&= \mathbb{E}\left(\sum_{i=i'} \mathbb{1}(h(k_i) = h(k_{i'})) + 2 \cdot \sum_{i \neq i'} \mathbb{1}(h(k_i) = h(k_{i'}))\right) \\
&= n + 2 \cdot \sum_{i \neq i'} \mathbb{E}(\mathbb{1}(h(k_i) = h(k_{i'}))) \\
&= n + 2\binom{n}{2}\frac{1}{m} \overset{m=n}{=} 2n - 1 \leq 2n.
\end{aligned}$$

14.9 Appendix

Some mathematical formulas

[Birthday Paradox]

Assumption: m urns, n balls (wlog $n \leq m$).
n balls are put uniformly distributed into the urns



What is the collision probability?

Birthdayparadox: with how many people (n) the probability that two of them share the same birthday ($m = 365$) is larger than 50%?

[Birthday Paradox]

$\mathbb{P}(\text{no collision}) = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-n+1}{m} = \frac{m!}{(m-n)! \cdot m^m}$.

Let $a \ll m$. With $e^x = 1 + x + \frac{x^2}{2!} + \ldots$ approximate $1 - \frac{a}{m} \approx e^{-\frac{a}{m}}$. This yields:

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{m}\right) \approx e^{-\frac{1+\cdots+n-1}{m}} = e^{-\frac{n(n-1)}{2m}}.$$

Thus

$$\mathbb{P}(\text{Kollision}) = 1 - e^{-\frac{n(n-1)}{2m}}.$$

Puzzle answer: with 23 people the probability for a birthday collision is 50.7%. Derived from the slightly more accurate Stirling formula. $n! \approx \sqrt{2\pi n} \cdot n^n \cdot e^{-n}$

[Formula for Expected Value]

$X \geq 0$ discrete random variable with $\mathbb{E}(X) < \infty$

$$\begin{aligned}\mathbb{E}(X) &\overset{(def)}{=} \sum_{x=0}^{\infty} x\mathbb{P}(X = x) \\ &\overset{\text{Counting}}{=} \sum_{x=1}^{\infty}\sum_{y=x}^{\infty} \mathbb{P}(X = y) \\ &= \sum_{x=0}^{\infty} \mathbb{P}(X > x)\end{aligned}$$

[Markov Inequality]

discrete Version $X \geq 0, a > 0$:

$$\begin{aligned}\mathbb{E}(X) &= \sum_{x=0}^{\infty} x\mathbb{P}(X = x) \\ &\geq \sum_{x=a}^{\infty} x\mathbb{P}(X = x) \\ &\geq a \sum_{x=a}^{\infty} \mathbb{P}(X = x) \\ &= a \cdot \mathbb{P}(X \geq a)\end{aligned}$$

$\Rightarrow$

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}(X)}{a}$$