14. Hashing

Hash Tables, Birthday Paradoxon, Hash functions, Perfect and Universal Hashing, Resolving Collisions with Chaining, Open Addressing, Probing

[Ottman/Widmayer, Kap. 4.1-4.3.2, 4.3.4, Cormen et al, Kap. 11-11.4]

Motivation

Gloal: Table of all n students of this course

Requirement: fast access by name

Naive Ideas

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

$$k(s) = \sum_{i=1}^{l_s} s_i \cdot b^i$$

b large enough such taht different names map to different keys.

Store each data set at its index in a huge array.

Example with $b = 100$. Ascii-Values s_i.

Anna $\mapsto 71111065$
Jacqueline $\mapsto 102110609021813999774$

Unrealistic: requires too large arrays.

Better idea?

Allocation of an array of size m ($m > n$).

Mapping Name s to

$$k_m(s) = \left(\sum_{i=1}^{l_s} s_i \cdot b^i\right) \bmod m.$$

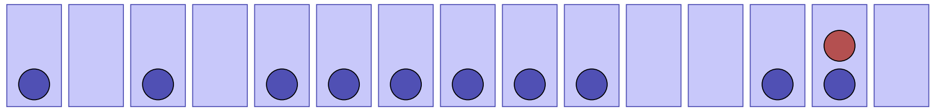
Different names can map to the same key ("Collision"). And then?

Estimation

Maybe collision do not really exist? We make an estimation ...

Abschätzung

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



What is the collision probability?

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

Estimation

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

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

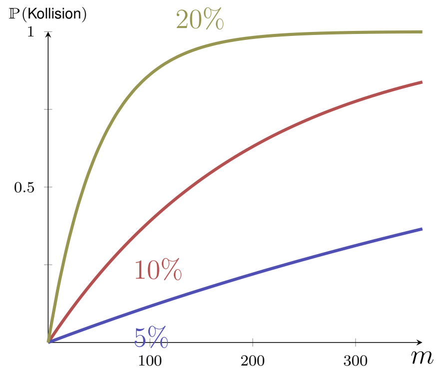
$$\mathbb{P}(\text{Kollision}) = 1 - e^{-\frac{n(n-1)}{2m}}.$$

Puzzle answer: with 23 people the probability for a birthday collision is 50.7%. Derived from the slightly more accurate Stirling formula.

With filling degree:

With filling degree $\alpha := n/m$ it holds that (simplified further)

$$\mathbb{P}(\text{collision}) \approx 1 - e^{-\alpha^2 \cdot \frac{m}{2}}.$$



The maximal filling degree should be chosen according to the ratio n^2/m.

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.

Examples of Good Hash Functions

- $h(k) = k \bmod m$, m prime
- $h(k) = \lfloor m(k \cdot r - \lfloor k \cdot r \rfloor) \rfloor$, r irrational, paritcularly good: $r = \frac{\sqrt{5}-1}{2}$.

Perfect Hashing

Is the set of used keys known up front? Then the hash function can be chosen perfectly. The practical construction is non-trivial.

Example: table of key words of a compiler.

Universal Hashing

- $|\mathcal{K}| > m \Rightarrow$ Set of “similar keys” can be chose such that a large number of collisions occur.
- Impossible to select a “best” hash function for all cases.
- Possible, however[14]: randomize!

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

$$\forall k_1 \neq k_2 \in \mathcal{K} : |\{h \in \mathcal{H} | h(k_1) = h(k_2)\}| \leq \frac{1}{m}|\mathcal{H}|.$$

[14] Similar as for quicksort

Universal Hashing

Theorem

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.

Universal Hashing

Initial remark for the proof of the theorem:

Define with $x, y \in \mathcal{K}$, $h \in \mathcal{H}$, $Y \subseteq \mathcal{K}$:

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

$$\delta(x, Y, h) = \sum_{y \in Y} \delta(x, y, h),$$

$$\delta(x, y, \mathcal{H}) = \sum_{h \in \mathcal{H}} \delta(x, y, h).$$

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

Universal Hashing

Proof of the theorem

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

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

■

Universal Hashing is Relevant!

Let p be prime and $\mathcal{K} = \{0, \ldots, p-1\}$. With $a \in \mathcal{K} \setminus \{0\}$, $b \in \mathcal{K}$ define

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

Then the following theorem holds:

Theorem

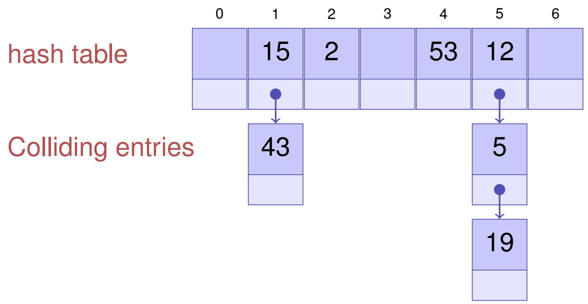
The class $\mathcal{H} = \{h_{ab} | a, b \in \mathcal{K}, a \neq 0\}$ is a universal class of hash functions.

Resolving Collisions

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

Keys 12 , 53 , 5 , 15 , 2 , 19 , 43

Chaining the Collisions

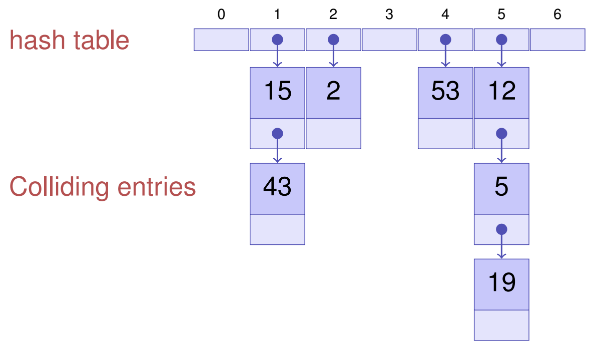


Resolving Collisions

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

Keys 12 , 53 , 5 , 15 , 2 , 19 , 43

Direct Chaining of the Colliding entries



Algorithm for Hashing with Chaining

- `search`(k) Search in list from position $h(k)$ for k. Return true if found, otherwise false.
- `insert`(k) Check if k is in list at position $h(k)$. If no, then append k to the end of the list.
- `delete`(k) Search the list at position $h(k)$ for k. If successful, remove the list element.

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

Advantages

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

Disadvantages

- Memory consumption of the chains-

Open Addressing

Store the colliding entries directly in the hash table using a *probing function* $s(j, k)$ ($0 \leq j < m$, $k \in \mathcal{K}$)

Key table position along a *probing sequence*

$$S(k) := (h(k) - s(0, k) \bmod m, \dots, (h(k) - (m - 1, k)) \bmod m$$

Algorithms for open addressing

- `search`(k) Traverse table entries according to $S(k)$. If k is found, return true. If the probing sequence is finished or an empty position is reached, return false.
- `insert`(k) Search for k in the table according to $S(k)$. If k is not present, insert k at the first free position in the probing sequence.[15]
- `delete`(k) Search k in the table according to $S(k)$. If k is found, mark the position of k with a `deleted` flag

[15] A position is also free when it is non-empty and contains a `deleted` flag.

Linear Probing

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

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.
Key 12 , 53 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2	5	53	12	

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!

? Disadvantage of the method?

! *Primary clustering:* simular hasht addresses have similar probing sequences $\Rightarrow$ long contiguous areas of used entries.

Quadratic Probing

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

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \mod m$.
Keys 12 , 53 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2		53	12	5

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

ⓘ 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 $h'(k)$. $s(j,k) = j \cdot h'(k)$.

$S(k) = (h(k) - h'(k), h(k) - 2h'(k), \ldots, h(k) - (m-1)h'(k)) \mod m$

Example:

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

Keys 12 , 53 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2	5	53	12	

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 independent of h (avoiding secondary clustering)

Independence:

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

Independence fulfilled by $h(k) = k \mod m$ and $h'(k) = 1 + k \mod (m-2)$ (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:

$$C_n \approx 1 + \frac{\alpha}{2} + \frac{\alpha^3}{4} + \frac{\alpha^4}{15} - \frac{\alpha^5}{18} + \cdots < 2.5$$

Overview

	$\alpha = 0.50$		$\alpha = 0.90$		$\alpha = 0.95$	
	C_n	C'_n	C_n	C'_n	C_n	C'_n
Separate Chaining	1.250	1.110	1.450	1.307	1.475	1.337
Direct Chaining	1.250	0.500	1.450	0.900	1.475	0.950
Linear Probing	1.500	2.500	5.500	50.500	10.500	200.500
Quadratic Probing	1.440	2.190	2.850	11.400	3.520	22.050
Double Hashing	1.39	2.000	2.560	10.000	3.150	20.000