14. Hashing

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

Introductory Example

Gloal: Efficient management of a table of all n ETH students.

Requirement: Fast access (insertion, removal, find) of a dataset by name.

Dictionary

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

- $\texttt{insert}(D, i)$: Insert or replace i in the dictionary D.
- $\texttt{delete}(D, i)$: Delete i from the dictionary D. Not existing $\Rightarrow$ error message.
- $\texttt{search}(D, k)$: 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<>`

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

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

std::cout << "The hex value of color red is: "
          << colours["RED"] << "\n";

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

Motivation/Applications

Perhaps *the* most popular data structure.

- Supported in many programming languages (C++, Python, Javascript, Java, C#, Ruby, ...)
- Obvious use
  - Databases
  - Symbol tables in compilers and interpreters
  - Objects in dynamically typed languages, e.g. Python, Javascript
- Less obvious
  - Substring search (z.B. Rabin-Karp)
  - String similarity (e.g. comparing documents, DNA)
  - File synchronisation (e.g. git, rsync)
  - Cryptography (e.g. identification, authentification)

Idea: Keys as Indices

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: Prehashing

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: $x = y \Leftrightarrow ph(x) = ph(y)$
- In practice: 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 $s = s_1 s_2 \ldots s_{l_s}$ to key

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

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

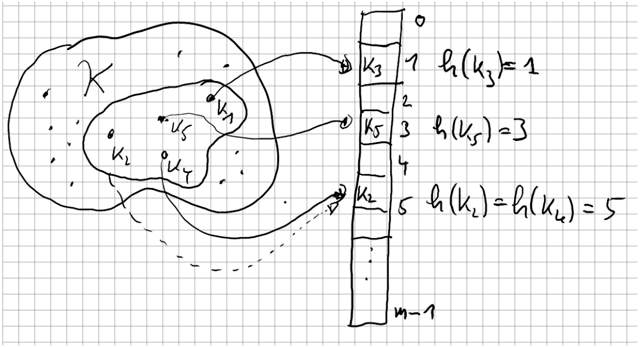
b Word-size of the system (e.g. 32 or 64)

Example with $b = 31$, $w = 32$, ASCII values s_i

Anna $\mapsto 92966272$
Anne $\mapsto 96660356$
Heinz-Harald $\mapsto 81592996699304236533 \bmod 2^{32} = 631641589$

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

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

Usually $|\mathcal{K}| \gg m$. 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 popular Hash Functions

Division method

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

Ideal: m prime number, not too close to powers of 2 or 10
(see e.g. Cormen et al. “Introduction to Algorithms”, Donald E. Knuth “The Art of Computer Programming”).

But often: $m = 2^r - 1$ ($r \in \mathbb{N}$), due to growing tables by doubling (more later).

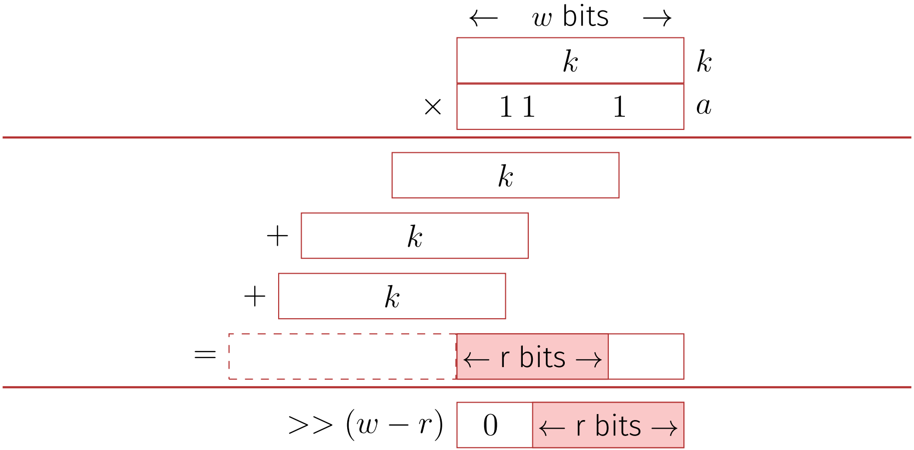
Examples of popular Hash Functions

Multiplication method

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

- 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.
- Table size $m = 2^r$, $w =$ size of the machine word in bits.
- Multiplication adds k along all bits of a, integer division by 2^{w-r} and $\bmod m$ extract the upper r bits.
- Written as code very simple: `a * k >> (w-r)`

Illustration

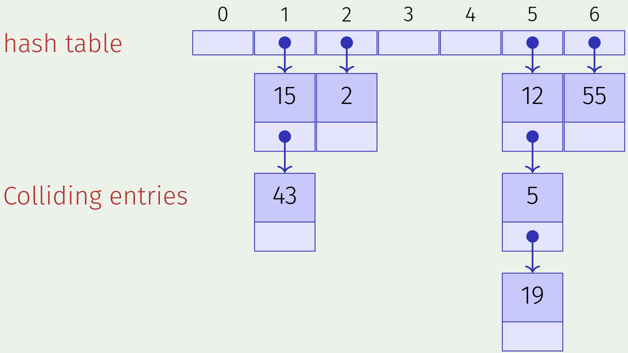


Resolving Collisions: Chaining

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

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

Direct Chaining of the Colliding entries



Algorithm for Hashing with Chaining

Let H be a hash table with collision lists.

- **`insert`**(H, 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`**(H, 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`**(H, 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

$$\mathbb{E}(\text{Length of Chain j}) = \mathbb{E}\left(\sum_{i=0}^{n-1} \mathbb{1}(h(k_i) = j)\right) = \sum_{i=0}^{n-1} \mathbb{P}(h(k_i) = j)$$

$$= \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m}$$

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

Simple Uniform Hashing

Theorem 17

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* $\Theta(1 + \alpha)$.

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 entirely.
   $\Rightarrow$ 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:

- Load factor greater 1 possible (more entries than hash table slots)
- Removing keys is straightforward (relative to alternative introduced later)

Disadvantages:

- Linear runtime in case of degenerated hash tables with long collision chains
- (Memory consumption of the chains)

Better: reduce probability of collisions

[Variant:Indirect Chaining]

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

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

Indirect chaining of colliding entries

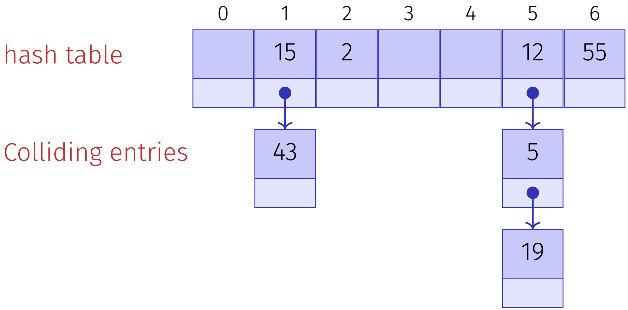


Table size increase

- We do not know beforehand how large n will be
- We would like $m = \Theta(n)$ at all times (hash table size m linearly dependent on no. of entries n, i.e. not arbitrarily large)

Adjust table size $\rightarrow$ 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 $\Theta(n + m + m')$

How to choose m'?

Table size increase

Double the table size, depending on the load factor.

$\Rightarrow$ Amortized analysis yields: Each operation of hashing with chaining has expected amortized costs $\Theta(1)$.

Open Addressing

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

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

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

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

Algorithms for open addressing

Let H be a hash table (without collision lists).

- **`insert`**(H, 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`**(H, k) Traverse table entries according to $S(k)$. If k is found, return data associated to k. Otherwise return an empty element **`null`**.
- **`delete`**(H, k) Search k in the table according to $S(k)$. If k is found, replace it with a special key **`removed`**.

Linear Probing

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

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

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

0	1	2	3	4	5	6
5	15	2	19		12	55

[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

0	1	2	3	4	5	6
19	15	2		5	12	55

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

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

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 18

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: Proof of the theorem

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

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

* : Event 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 $n < m$: $\frac{n-1}{m-1} < \frac{n}{m} \Leftrightarrow \frac{n-1}{n} < \frac{m-1}{m} \Leftrightarrow 1 - \frac{1}{n} < 1 - \frac{1}{m} \Leftrightarrow n < m$
$(n > 0, m > 0)$

Moreover $\mathbb{P}(x \geq i) = 0$ for $i \geq m$. Therefore

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

[Successful search of Uniform Open Hashing]

Theorem 19

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

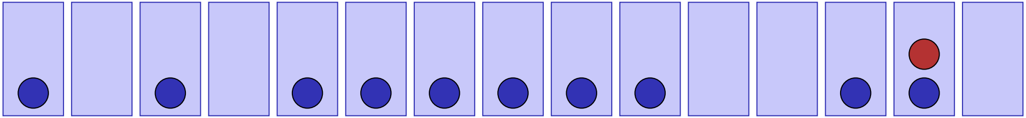
α: load factor.
C_n: Number steps successful search,
C'_n: Number steps unsuccessful search

14.8 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]

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

Let $a \ll m$. With $e^x = 1 + x + \frac{x^2}{2!} + \dots$ 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 $\mathbb{E}(X) < \infty$

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