Institute of
Pervasive Computing

Hashing

Algorithms and Data Structures 2, 340300
Lecture – 2023W

Univ.-Prof. Dr. Alois Ferscha, teaching@pervasive.jku.at

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Motivation

Dictionary data structures until now:

- ordered storage with usage of keys k ∈ K
  At a time only a small amount of keys K from the amount of all possible keys ***K*** is in use (k ∈ ***K***)
- **search**, **remove**, **insert** always requires a series of key comparisons

Hashing:

- Try to do this **without key comparisons**, i.e. determine by calculation where a data set with key k ∈ ***K*** is stored.

Hashtable:

- Data set are stored in an array A[0..N-1]

Hashfunction:

- h: ***K*** → {0, ..., N-1} assigns a **hash address** to each key k (= index in the hash table)
  $0 \leq h(k) \leq N-1$
- Since N is generally much smaller than K, h() is generally not injective
- Example: Symbol table: 51 reserved words in Java with more than 62^{80} allowed identifiers with ≤ 80 digits.

Motivation

Synonyms

- Keys k, k‘∈ K are **synonymous** if h(k) = h(k‘)

Address collision

- The same hash address is assigned to synonyms
- No synonyms, no collision
- Address collision requires special handling

Occupancy factor

- For a hash table of size N, that currently stores n keys, we specify $\alpha =$ **n/N** as the occupancy factor.

Two requirements on hashing methods:

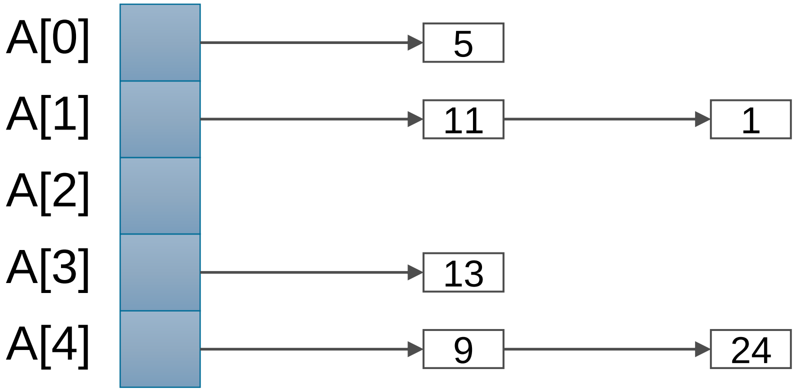
1. Choose h() in a way, so that **as few collisions as possible** occur = Selection of a “good” hash function
2. Address collisions should be **resolved** as **efficiently** as possible

Hash Tables

Hash tables:

Efficient implementation of a dictionary with regard to storage space and complexity of **search**, **insert** and **remove** operations (usually better than implementations based on key comparisons)

- **Key-value pairs** are stored in an array of size N
- **Index** is calculated from the **hash function value** of the **key h(k)**.
  Aim: store ***item(k,e)*** at *A[h(k)]*
- Example: Use ***key k modulo array size*** as index and use ***chaining***,
  if two keys are mapped to the same index (collision)



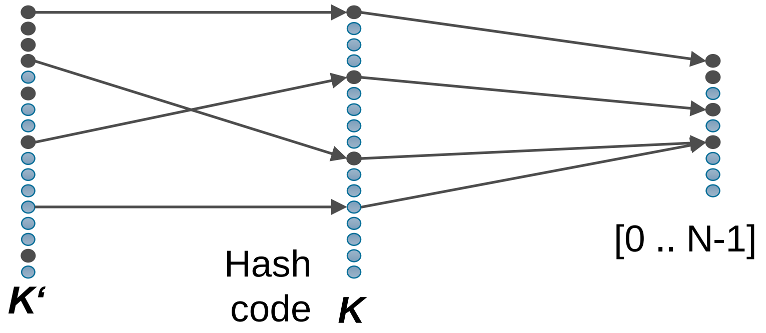
Chaining:

Keys with same index are stored in a list.

Hash Function

h() often consist of **two mapping functions**: **$h = h_1 \circ h_2$**

- **hash code**: h_1 :key $\rightarrow$ integer (k now integer = Hashcode)
- **compression map**: h_2 :k $\rightarrow$ [0 .. N-1]



k= h_1(s) ... Hashcode (-address)

$h_2(k) = h_2(k') \Leftrightarrow$ k and k´ are synonyms

K' is the key set in any data type (e.g. s $\in$ ***K'*** string)
K is the key set where k $\in$ ***K*** integer
(but often the key k can also be used directly as hash code)

„Good" hash function:

- Easy to calculate
- Possible keys distributed as evenly as possible across indexes
- Probability of a collision should be minimized

Hash Codes in Java

First part of the hash function (h_1) assigns an integer to any key k
= Hash code or hash value

in Java: **hashCode()** method returns 32 bit int (!) for each object

- (in many Java implementations, however, this is only the memory address of the object, i.e. a bad distribution => bad hash codes => **overload** with better method)

Example: Integer cast

- for numeric data types with 32 bits or less, the bits can be interpreted as int: **typecast** of byte, short, int, char

Exmaple: Component sum

- for numeric types with more than 32 bits (long, double) add 32-bit components

```
public static int hashCode(long i) {
            return (int) ((i >> 32) + (int) i);
}
```

(upper 32 bit) + (lower 32 bit)

Hash Codes: Polynomial Accumulation

Consider binary representation of the key as (x_0, x_1, x_2, ... x_{k-1}):
simple accumulation results in bad hash code because e.g. „spot“, „stop“, „tops“ ... Collide.

for (Java-)Strings therefore:
Consider the character values (ASCII or unicode) $x_0 x_1 ... x_{n-1}$ as **coefficients of a polynomial**

$$x_0 a^{k-1} + x_1 a^{k-2} + ... + x_{k-2} a + x_{k-1}$$

Calculation according to **Horner scheme** (overflows are ignored) for certain value $a \neq 1$

$$x_{k-1} + a (x_{k-2} + a (x_{k-3} + ... + a (x_1 + a x_0)...))$$

For e.g. a=33, 37, 39, or 41 there are only 6 collisions in a vocabulary of 50.000 (english) words

```
public static int hashCode(String s) {
          int h = 0;
          for (int i = 0; i < s.length(); i++) {
                    h = (h << 5) | (h >>> 27);   // 5-bit cyclic shift
                    h += (int) s.charAt(i);      // add next character
          }
          return h;
}
```

Compression Maps – Hash Functions

Division-Reminder-Method

- h(k) = **|k| mod N**
- Choice of N **even** (**odd**), then h(k) also **even** (**odd**)
  - Bad if e.g. the last bit expresses a fact
    (e.g. 0 = male, 1 = female)
- Choice of N = $\mathbf{2^p}$
  - h(k) returns the **p lowest dual digits of k**: bad because remaining bits are neglected
- Choice of N as prime number: $N \neq r^i \pm j$, $0 \leq j \leq r-1$, ... r = radix
  (proves best in practice, empirically best results)

MAD - Multiply, Add, and Divide

- h(k) = **|ak+b| mod N** ... N prime, a, b $\geq$ 0, a mod N $\neq$ 0
- eliminates „**patterns**“ in keys of the form **iN+j**
- Collision probability for two keys $\leq$ 1/N
- the same formula is also used in linear congruent (pseudo)random number generators

Compression Maps – Hash Functions

Multiplicative Method - Requirement [Turan Sos]:

Let Ψ be an **irrational number**. If you place n points

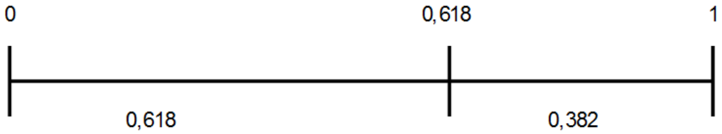
$$\Psi - \lfloor \Psi \rfloor, \quad 2\Psi - \lfloor 2\Psi \rfloor, \quad 3\Psi - \lfloor 3\Psi \rfloor, \quad \ldots, \quad n\Psi - \lfloor n\Psi \rfloor$$

in the interval [0,1], then the resulting n+1 intervals have **at most three different lengths**.

Compression Maps – Hash Functions

Multiplicative Method - Requirement [Turan Sos]:

$$\Psi = \frac{\sqrt{5} - 1}{2} \approx 0{,}618$$



$0{,}618$

$\Psi - \lfloor \Psi \rfloor$

Compression Maps – Hash Functions

Multiplicative Method - Requirement [Turan Sos]:

Let Ψ be an **irrational number**. If you place n points

$$\Psi - \lfloor\Psi\rfloor, \quad 2\Psi - \lfloor 2\Psi\rfloor, \quad 3\Psi - \lfloor 3\Psi\rfloor, \quad \ldots, \quad n\Psi - \lfloor n\Psi\rfloor$$

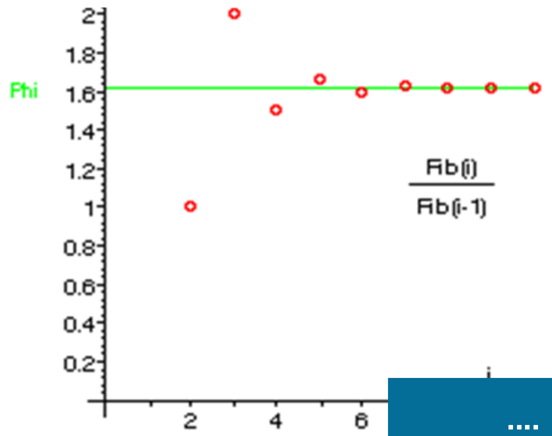
in the interval [0,1], then the resulting n+1 intervals have **at most three different lengths**.

If you divide further, the next point $(n+1)\Psi - \lfloor (n+1)\Psi\rfloor$ falls into the **largest partial interval**.

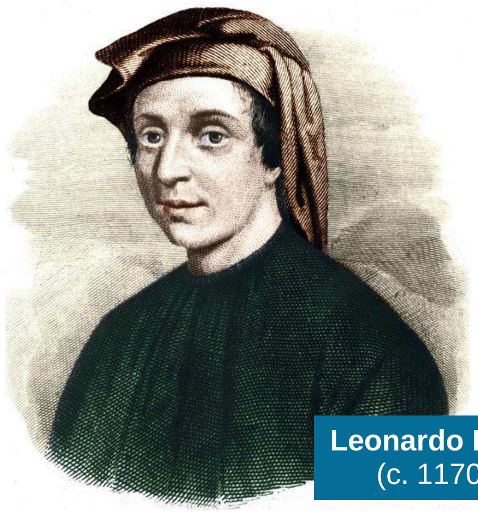
Of all numbers $0 \leq \Psi \leq 1$ the **golden ration** $\Psi = (\sqrt{5} - 1)/2$ leads to the most balanced intervals.

$h(k) = \lfloor N (k\Psi - \lfloor k\Psi \rfloor) \rfloor$ forms exactly the permutation for N=10
h(1) = 6, h(2) = 2, h(3)= 8, =4, =0, =7, =3, =9, =5, h(10)=1
(.. and always divides exactly in the golden ratio)

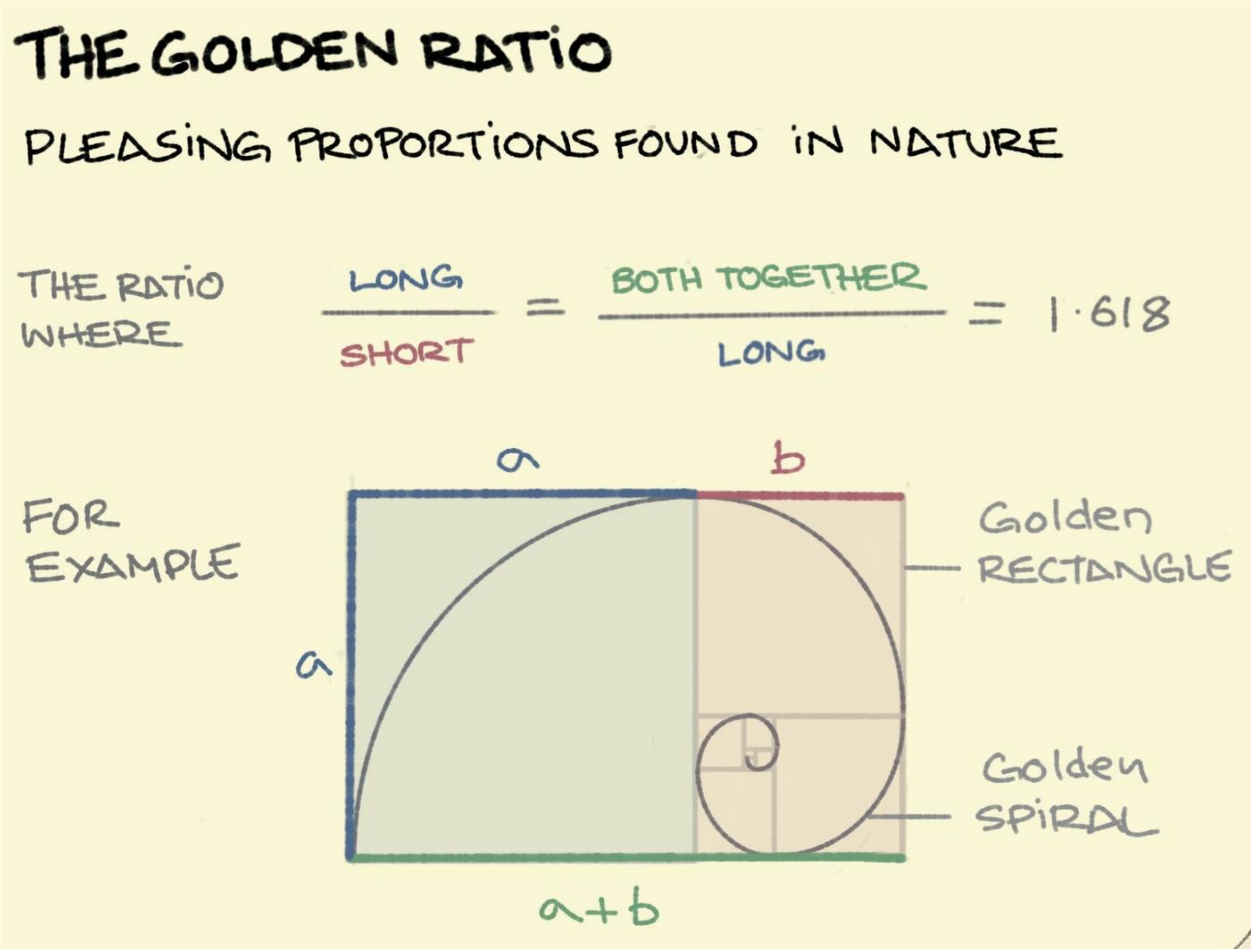
The “Golden Number”



.... 1,618034



Leonardo Pisano Fibonacci
(c. 1170 – c. 1240–50)



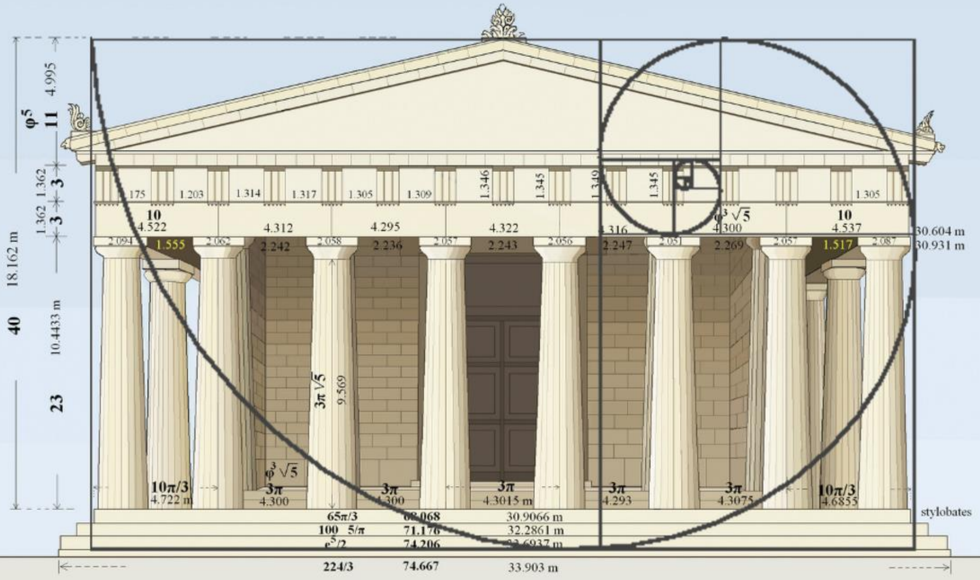
The “Golden Ratio”

$\frac{x}{1} = \frac{1}{x-1}$, thus $x^2 - x = 1$ or: $x^2 - x - 1 = 0$

with the solutions

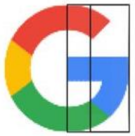
$X_1 = (1 + \sqrt{5}\,)/2) = 1{,}61803$

$X_2 = (\sqrt{5} - 1\,)/2) = 0{,}61803$

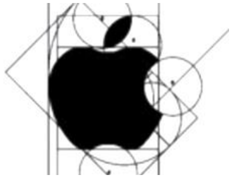


Parthenon in Athens, Greece

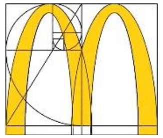
Logo Design based on the Golden Ratio



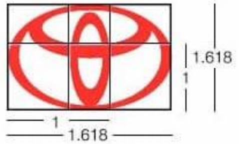
Google



Apple

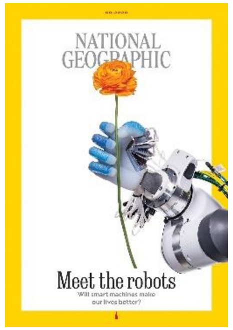


McDonald’s



Toyota

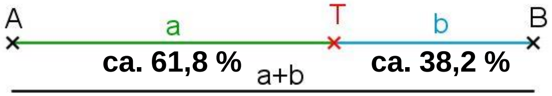
1



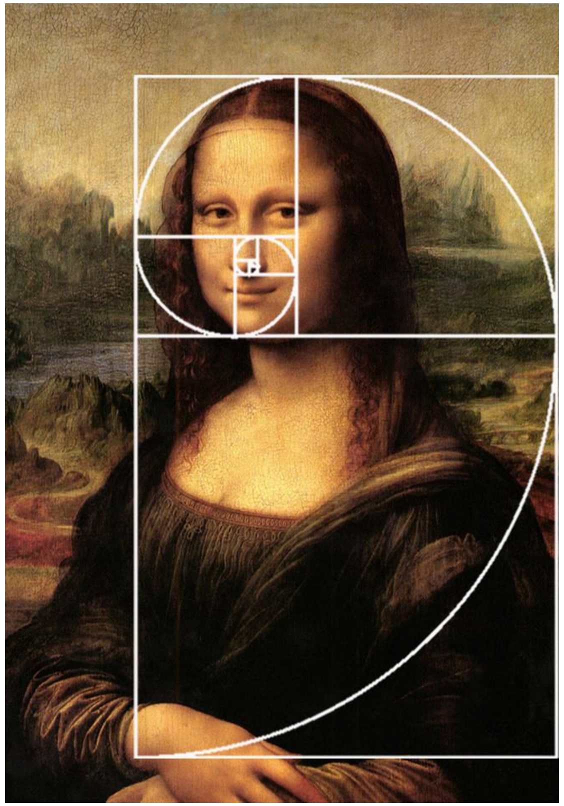
1.618

National Geographic

NATIONAL GEOGRAPHIC



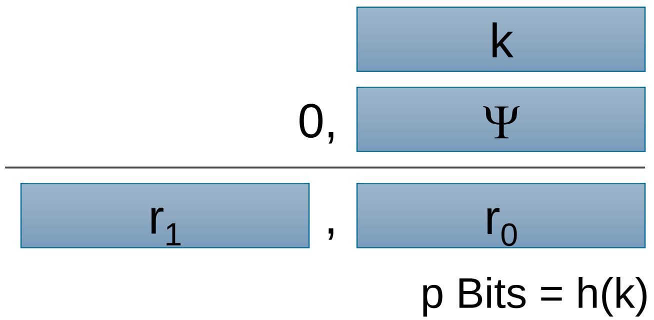
$$\frac{a+b}{a} = \frac{a}{b} \approx 1.618$$



Compression Maps – Hash Functions

Multiplicative Method

- Choose constant Ψ with $0 < \Psi < 1$
- Calculate $k \Psi \bmod 1 = k \Psi - \lfloor k \Psi \rfloor$
- $h(k) = \lfloor N(k \Psi \bmod 1) \rfloor$
- Choice of N not critical
  with $N = 2^p$ the
  calculation of h(k) can be accelerated.



Example:

$\Psi = (\sqrt{5} - 1)/2 \approx 0{,}6180339...$

$k = 123456$

$N = 1024$

$h(k) = \lfloor 1024(123456 \cdot 0{,}6180339... \bmod 1) \rfloor$

$= \lfloor 1024(76300{,}0041151... \bmod 1) \rfloor$ can be implemented as shift

$= \lfloor 4{,}213... \rfloor = 4$

$= \lfloor 41{,}151... \rfloor = 41$ .. Calculation error

Perfect Hashing

If the number of keys to be stored is known and $|K| \leq N$, collision-free storing is always possible!

Form the **injective** mapping h: $K \rightarrow \{0, ..., N\text{-}1\}$:

1. arrange the keys $k \in K$ in lexicographic order
2. assign (unique) order numbers to the keys

Collisions are completely avoided: **perfect hashing**

Application example:

Keywords of a programming language are assigned to fixed places in a symbol table.

Analysis of Ideal Hashing

Assumptions

- **n** data items inserted into a memory with **N** places
- there have been no deletions
- All configrations of **n occupied** and **N-n nonoccupied** storage locations have the same probability

If P_r is the probability that exactly r places must be tested in the unsuccessful search, then we have:

$$P_r = \begin{cases} \frac{\binom{N-r}{n-(r-1)}}{\binom{N}{n}} & 1 \leq r \leq N \\ 0 & r > N \end{cases}$$

- the first r-1 places are occupied, the r^{th} place is free
- on the remaining m-r places the other n-(r-1) occupied places can be distributed arbitrarily

Analysis of Ideal Hashing

Expected number of searched items if **search failed**

$$C'_n = \sum_{r=1}^{N} r P_r = \frac{N+1}{N-n+1} \approx \frac{N}{N-n} = \frac{1}{1-\alpha}$$

Expected number of searched items for **successful search**

$$C_n = \frac{1}{n} \sum_{k=0}^{n-1} \frac{N+1}{N-k+1} = \frac{N+1}{n}(H(N+1) + H(N-n+1))$$

$$\approx \frac{N+1}{n} \ln \frac{N+1}{N-n+1} \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

with H(N) = 1 + 1/2 + 1/3 + ... + 1/N ≈ **ln N**

Universal Hashing (Randomisation)

Observation:
current key set $K \subset \boldsymbol{K}$ is generally not “equally distributed” from the universe of keys $\boldsymbol{K}$
(example: programmers’ preference for variables i, i1, i2, i3, ...)

Problem: If fixed **h** is chosen $\Rightarrow K \subset \boldsymbol{K}$ can be constructed with arbitrarily many collisions!

Idea: ***Universal Hashing***

- Choose hash function **h randomly from** a finite number of hash functions **H**
  $h \in H : \boldsymbol{K} \rightarrow \{ 0, ..., N-1\}$
- Definition: **H is universal**, if for arbitrary x, y $\in$ K we have: $$\frac{|\{h \in H | h(x) = h(y)\}|}{|H|} \leq \frac{1}{N}$$
- Conclusion: x, y $\in \boldsymbol{K}$ arbitrary, H universal, h $\in$ H random

$$Pr_H(h(x) = h(y)) \leq \frac{1}{N}$$

- (Probability that x, y is mapped to the same hash address of a random h $\in$ H)

Example for Universal Hashing

In other words:
H is **universal** if the number of hash functions to which **h(k)=h(l)** applies **is at maximum** equal to **|H|/N** for each pair of different keys.

Universal Hash functions exist / and are „easy“ to create:

Hash table A of size N=3 and p=5 (prime number) Keys ***K*** = {0, 1, 2, 3, 4}

20 hash functions H = $h_{i,j}(x)$ = ((ix + j) mod 5) mod 3 $1 \le i < p,\ 0 \le j < p$

1x+0	2x+0	3x+0	4x+0
1x+1	2x+1	3x+1	4x+1
1x+2	2x+2	3x+2	4x+2
1x+3	2x+3	3x+3	4x+3
1x+4	2x+4	3x+4	4x+4

each (mod 5) (mod 3)

Example: Consider e.g. keys 1 and 4
h(1) = h(4) occurs in 4 of 20 hash functions (x+0, x+4, 4x+0, 4x+4)
$(1\cdot 1 + 0) \bmod 5 \bmod 3 = 1 = (1\cdot 4 + 0) \bmod 5 \bmod 3$
$(1\cdot 1 + 4) \bmod 5 \bmod 3 = 0 = (1\cdot 4 + 4) \bmod 5 \bmod 3$
$(4\cdot 1 + 0) \bmod 5 \bmod 3 = 1 = (4\cdot 4 + 0) \bmod 5 \bmod 3$
$(4\cdot 1 + 4) \bmod 5 \bmod 3 = 0 = (4\cdot 4 + 4) \bmod 5 \bmod 3$
i.e. $Pr_H (h_{i,j}(x) = h_{i,j}(y)) \le 4/20=1/5$ for all hash functions $h_{i,j}(x) \in H$
i.e. H is universal

For two randomly chosen keys 1, 4 there is **one** collision in 4 of the 20 hash functions

in the other 16 there are **0** collisions

Universal Hashing

Recommended approach:

Known:

The number of keys |K| which has to be mapped to N hash addresses.

Choose:

1. a **prime number p** which is greater than or equal to |K|
2. **two numbers i, j** in the range $1 \leq i < p$, $0 \leq j < p$

Then:

$h(x) = ((ix + j) \bmod p) \bmod N$

is a "good" hash function

Universal Hashing

Definition

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

δ shows if collisions occur for two keys from ***K*** regarding h()

Extension of δ to a set Y $\subseteq$ ***K*** and H

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

$$\delta(x,y,H) = \Sigma_{h \in H}\, \delta(x,y,h)$$

H is universal, if for two arbitrary x,y $\in$ ***K*** ($x \neq y$) we have

$$\delta(x,y,H) \leq \frac{|H|}{N}$$

Universal Hashing

Unknown: number of $|K|$

Known:

$H : \boldsymbol{K} \rightarrow \{ 0, ..., N-1\}$ a universal set of hash functions
$h \in H$ a randomly chosen hash function used for all insertions,
then the place $h(x)$ can already be occupied for the insertion attempt x.
For the insertion attempt x there are already S keys stored:

$$\begin{aligned}
\mathrm{E}[\delta(x,S,h)] &= \Sigma_{h \in H}\, \delta(x,S,h) \,/\, |H| \\
&= 1/|H|\, \Sigma_{h \in H}\, \Sigma_{y \in S}\, \delta(x,y,h) \\
&= 1/|H|\, \Sigma_{y \in S}\, \Sigma_{h \in H}\, \delta(x,y,h) \\
&= 1/|H|\, \Sigma_{y \in S}\, \delta(x,y,H) \\
&\leq 1/|H|\, \Sigma_{y \in S}\, |H| \,/\, N \\
&= |S| \,/\, N
\end{aligned}$$

i.e. the expected number of already inserted elements which probably have collided with x is $|S| / N$

This means that an arbitrarily chosen hash function h from a universal set H, will map sequences of keys (no matter how unilaterally they are) to available hash addresses as evenly as possible.

Overview :: Collision Handling

Inserting a synonym k‘, if key k is already stored: Collision (place h(k) = h(k‘) is already occupied)

h(k‘) is referred to as ***overflow***

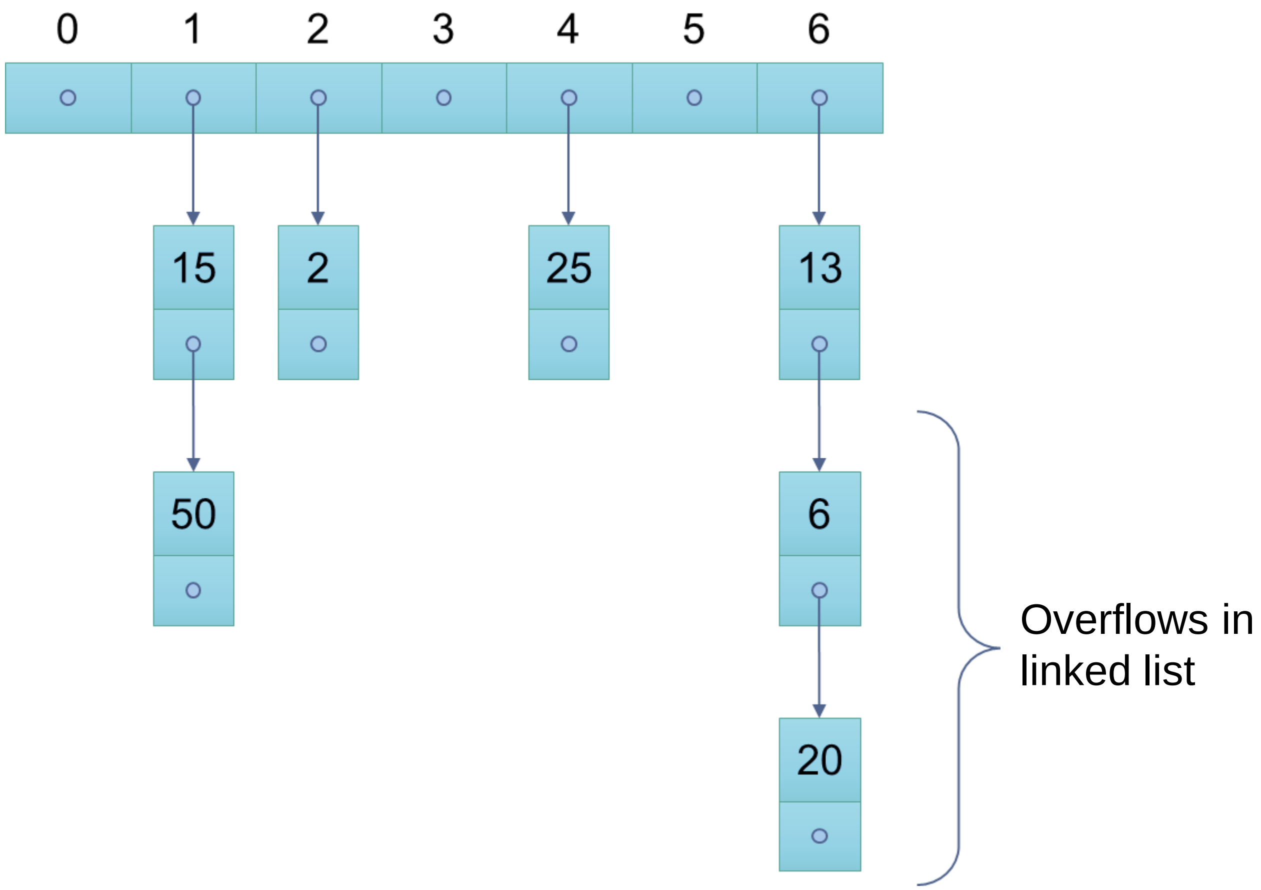
Solution 1: **Overflow chaining**

- Keys with the same index are stored in an overflow list at the corresponding index position.
- Is also known as ***closed hash procedure***.

Solution 2: **Open hashing**

- If a key is to be inserted at a position that is already occupied, **another free** (vacant) position is selected and the key is stored at this position.
- More details regarding this follow later.

Chaining



Example:
Insert sequence:
25, 2, 15, 50, 13, 6, 20

Hash Function:
h(k) = k mod 7

Method:
Each element of the hash table is a reference to an overflow chain.

Operations in Hash Tables with Chaining

Search for key k

- **calculate h(k)** and reference A[h(k)] in the overflow list
- **search for k** in the overflow list until it is found or the end of the list is reached (not found)

Insert a key k

- **search for k** as described above (ends unsuccessfully – otherwise it will not be inserted)
- **create** list element for k and **insert** it in the overflow list

Remove a key k

- **search for k as** described above
- if successful, **remove** from overflow list

All operations are based on **pure list operations**.

Analysis of Hash Tables with Chaining

Uniform Hashing assumption:

- all hash addresses are chosen with equal probability: $P_r\ (\ h(k_i) = j\ ) = 1/N$
- Independent from operation to operation (above P_r for each $0 \le j \le N\text{-}1$)

Average overflow list length for n entries (also: occupancy factor)

- $n/N = \alpha$

Complexity of the **search**: (new keys are always added to the end of the overflow list)

C'_n expected number of searched position for **unsuccessful search**

$C'_n = n/N = \alpha$

C_n expected number of searched positions for **successful search**

$C_n = 1/n \quad \Sigma_{j=1..n}\ (1+(j-1)/N) = 1 + (n-1) / 2m \approx \mathbf{1 + \alpha / 2}$

Open Hashing

Idea:

- Placement of overflows k‘ (h(k‘) = h(k)) **at vacant position** in the hash table
- According to the rule: if A[h(k)] **occupied**, **search other position** for k‘
- Sequence of chosen positions: ***probing sequence***
- Basic problem: Selection of a suitable ***probing sequence***

Example: Consider entry with next smaller index (h(k) – 1) mod N
Problem: Recovery of k‘ if k is removed in the meantime

Generalization: consider entry with

- (h(k) **- s(j,k)**) mod N   j = 0, ..., N-1 for a given function s(j,k)

Common variants of s(j,k):

- **linear probing:**   **s(j,k) = j**
- **quadratic probing:**   **s(j,k) =** $(-1)^j \cdot \lceil j/2 \rceil^2$

Open Hashing

Properties of **s(j,k)**

Sequence

h(k) - s(0,k) mod N

h(k) - s(1,k) mod N

...

h(k) - s(N-2,k) mod N

h(k) - s(N-1,k) mod N

Is a **permutation** of the hash addresses 0, ..., N-1

e.g. quadratic probing

h(11) = 4

$s(j,k) = (-1)^j \cdot \lceil j/2 \rceil^2 = 0, -1, 1, -4, 4, -9, 9$

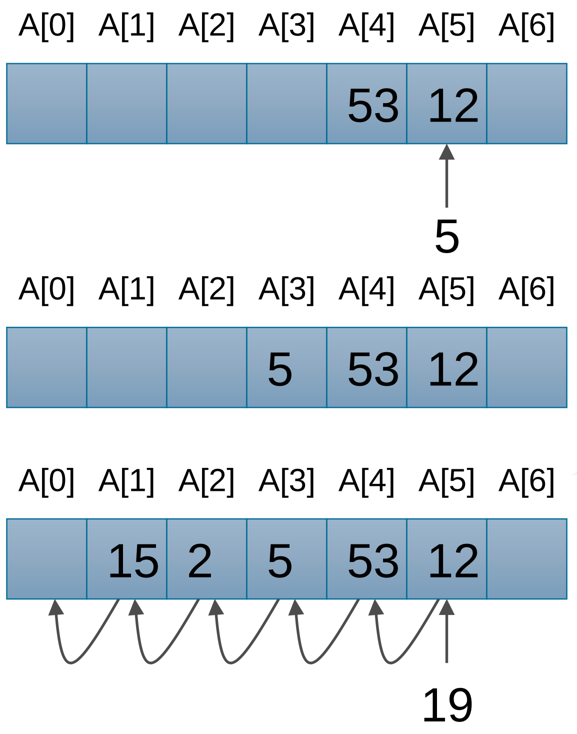
0	1	2	3	4	5	6

Open Hashing :: Linear Probing

Probe function: s(j,k) = j

Probe sequence: h(k), h(k)-1, h(k) –2, ... 0, N-1, ..., h(k)+1

Example: N=7, ***K*** = {0, 1, ..., 500}, h(k) = k mod N, Keys: 12, 53, 5, 15, 2, 19



6 inspections!

„coalescing“

long (occupied) parts tend to grow further

"primary accumulation“

Efficiency gets worse drastically near α=1

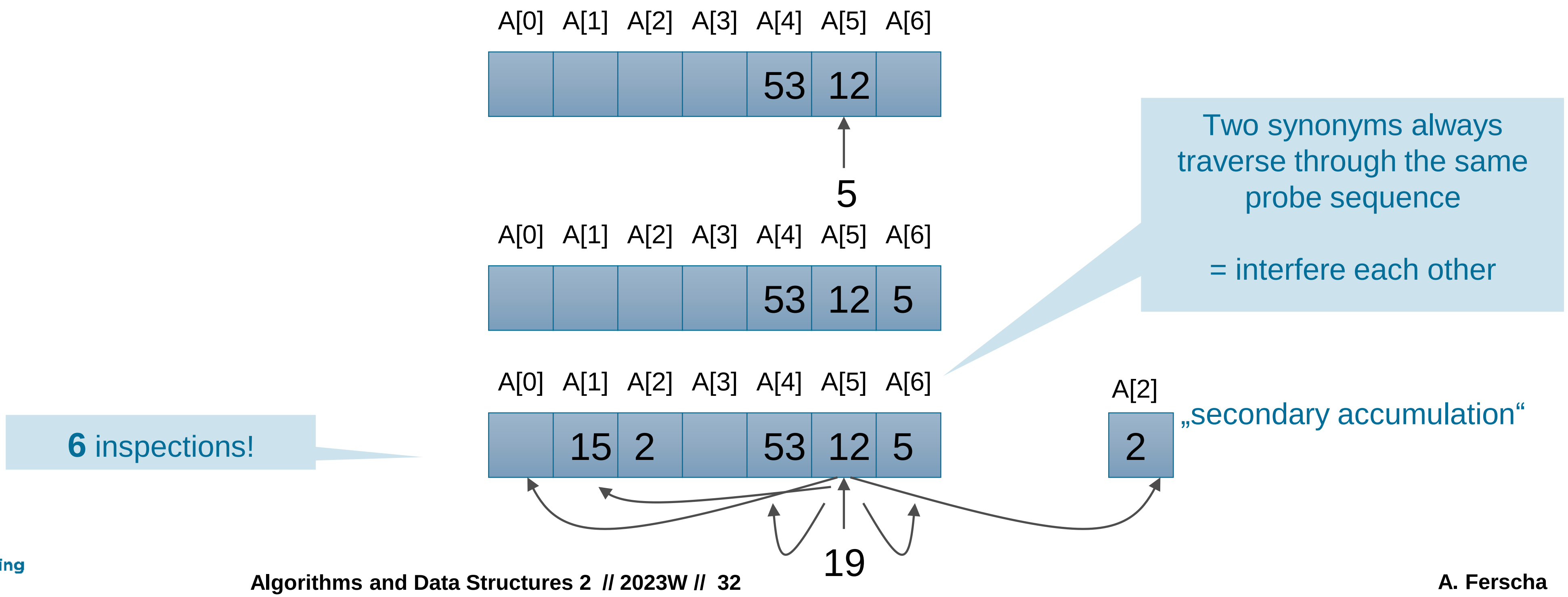
Open Hashing :: Quadratic Probing

Aim: avoid „primary accumulation“

Probe function: $s(j,k) = (-1)^j \cdot \lceil j/2 \rceil^2 = 0, -1, 1, -4, 4, -9, 9$

Probe sequence: h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, ...

Example: N=7, ***K*** = {0, 1, ..., 500}, h(k) = k mod N, Keys: 12, 53, 5, 15, 2, 19



Open Hashing :: Uniform Probing

Aim:	avoid “primary“ and “secondary accumulation“
Reason for accumulation:	Probing function is **independent of k** ! (probe function is the same for all synonyms)
Probing function:	s(j,k) for j = 0, ..., N-1 is a permutation of the hash addresses that depends only on k, where each of the N! possible permutations is used with **equal probability** (Uniform probing) + asymptotically optimal! - practically very difficult to realize
Random Probing:	s(j,k) chooses a hash address **randomly** Contrary to Uniform Probing: A value chosen for s(j,k) can be “drawn” again later (j‘ > j)

Analysis Open Hashing

Linear Probing

- Probe sequence: h(k), h(k)-1, h(k-2), ...
- Problem: primary clustering
- $C'_n \approx (1 + 1/(1-\alpha)^2)$ $\quad C_n \approx (1 + 1/(1-\alpha))$

Quadratic Probing

- Probe sequence: h(k), h(k)-1, h(k)+1, h(k)-4, h(k)+4, ...
- Permutation, if N = 4i+3, prime
- Problem: secondary clustering
- $C'_n \approx 1/(1-\alpha) - \alpha + \ln(1/(1-\alpha))$ $\quad C_n \approx 1 - \alpha/2 + \ln(1/(1-\alpha))$

Uniform Probing

- $s(j,k) = \pi_k(j)$ $\quad \pi_k$ one of N! permutations of {0,...,N-1}
- Each permutation has equal probability
- $C'_n \leq 1/(1-\alpha)$ $\quad C_n \approx 1/\alpha \cdot \ln(1/(1-\alpha))$

Random Probing

- s(j,k) = random number dependent on k
- s(j,k) = s(j´,k) possible, but unlikely

Open Hashing

Idea:

- Placement of overflows k‘ (h(k‘) = h(k)) **at vacant position** in the hash table
- According to the rule: if A[h(k)] **occupied**, **search other position** for k‘
- Sequence of chosen positions: ***probing sequence***
- Basic problem: Selection of a suitable ***probing sequence***

Example: Consider entry with next smaller index (h(k) – 1) mod N

Problem: Recovery of k‘ if k is removed in the meantime

Generalization: consider entry with

- (h(k) **- s(j,k)**) mod N   j = 0, ..., N-1 for a given function s(j,k)

Common variants of s(j,k):

- **Linear Probing:** **s(j,k) = j**
- **Quadratic Probing:** **s(j,k) =** $(-1)^j \cdot \lceil j/2 \rceil^2$
- **Double Hashing:** **s(j,k) =** $j \cdot h_2(k)$

Hashing

Algorithms and Data Structures 2, 340300
Lecture – 2023W
Univ.-Prof. Dr. Alois Ferscha, teaching@pervasive.jku.at

Institute of
Pervasive Computing

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at