Double 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

Remember :: Analysis Open Hashing

Linear Probing

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

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
- $C'_n \approx 1/(1-\alpha) - \alpha + \ln(1/(1-\alpha))$ $C_n \approx 1 - \alpha/2 + \ln(1/(1-\alpha))$

Uniform Probing

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

Random Probing

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

Remember :: Analysis Open Hashing

linear probing

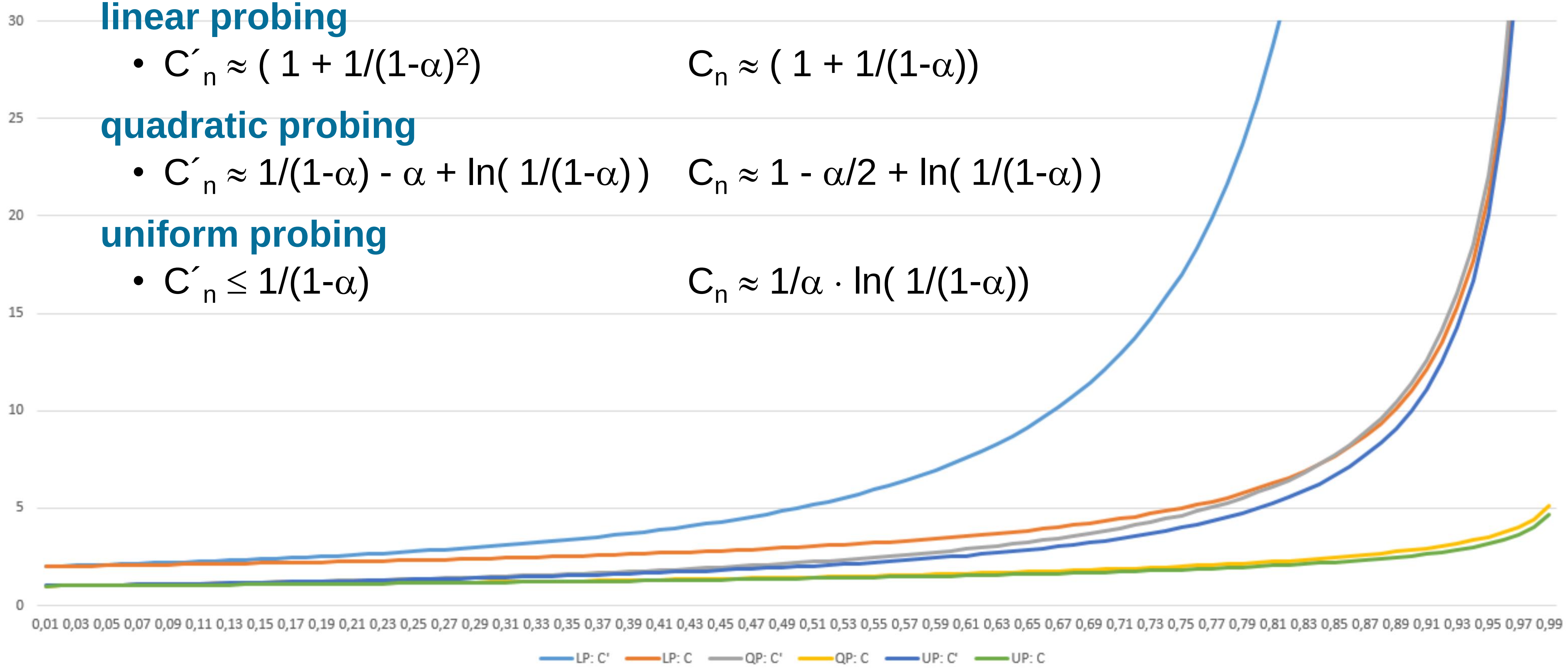
- $C'_n \approx (1 + 1/(1-\alpha)^2)$ $\quad$ $C_n \approx (1 + 1/(1-\alpha))$

quadratic probing

- $C'_n \approx 1/(1-\alpha) - \alpha + \ln(1/(1-\alpha))$ $\quad$ $C_n \approx 1 - \alpha/2 + \ln(1/(1-\alpha))$

uniform probing

- $C'_n \leq 1/(1-\alpha)$ $\quad$ $C_n \approx 1/\alpha \cdot \ln(1/(1-\alpha))$



Open Hashing :: Double Hashing

Uniform probing efficiency is already achieved if a second hash function is used instead of random permutation: **Double Hashing**

Use **two hash functions $h_1(k)$, $h_2(k)$**

```
double_hash_insert(k)
  if(table is full) error
        probe = h1(k)
        offset = h2(k)
        while(table[probe] occupied)
              probe = (probe + offest) mod m
        table[probe] = k
```

Keys are **distributed more equally** than with linear probing

- (≈) same efficiency as uniform probing (if h1(k), h2(k) independent)

Disadvantage of all open hashing procedures:

- Operations become slower and more complex (e.g. selecting or moving elements in remove operation)

Open Hashing :: Double Hashing

Probing function: $s(j,k) = j \cdot h_2(k)$

Probing sequence: $h_1(k), h_1(k) - h_2(k), h_1(k) - 2 \cdot h_2(k), ..., h_1(k) - (N-1) \cdot h_2(k)$
(each mod N)

h_2 must be chosen so that the probing sequence forms a permutation of the hash addresses $\Rightarrow$ N prime

Example: N=7, $\boldsymbol{K} = \{0, 1, ..., 500\}$, Keys: 12, 53, 5, 15, 2, 19

$h_1(k) = k \bmod N$, $h_2(k) = 1 + k \bmod (N-2)$

$h_1(k) = k \bmod 7$

$h_2(k) = 1 + k \bmod 5$

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
				53	12	

5

Insert of 5

$h_1(5) = 5 \bmod 7 = 5$
$h_2(5) = 1 + 5 \bmod 5 = 1$

Probe sequence: 5, 4, 3

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
			5	53	12	
19	15	2	5	53	12	

19

2 inspections!

Insert of 19

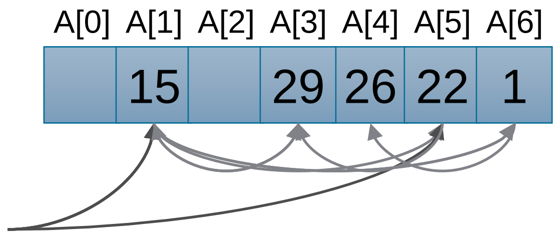
$h_1(19) = 19 \bmod 7 = 5$
$h_2(19) = 1 + 19 \bmod 5 = 5$

Probe sequence: 5, 0

Further Example for Double Hashing

Example

- **h_1(k)** = k mod 7, **h_2(k)** = 1 + k mod 5
  Key sequence 15, 22, 1, 29, 26



$h_1(15) = 1$

$h_1(22) = 1$ $\quad h_2(22) = 3$

$h_1(1) = 1$ $\quad h_2(1) = 2$

$h_1(29) = 1$ $\quad h_2(29) = 5$

$h_1(26) = 5$ $\quad h_2(26) = 2$

- Average search time: (1+2+2+2+5)/5 = 2.4

  Average search time (1+1+3+1+1+1+2)/6 = 1.5
  if in the order 53, 5, 15, 2, 19, 12 is inserted!!!

Improvement of the Successful Search

When inserting:

- k encounters k_{old} in A[i], i.e. $i = h(k) - s(j,k) = h(k_{old}) - s(j´,k_{old})$
- k_{old} already stored in A[i]
- Idea: Search vacant position for **k or k_{old}**

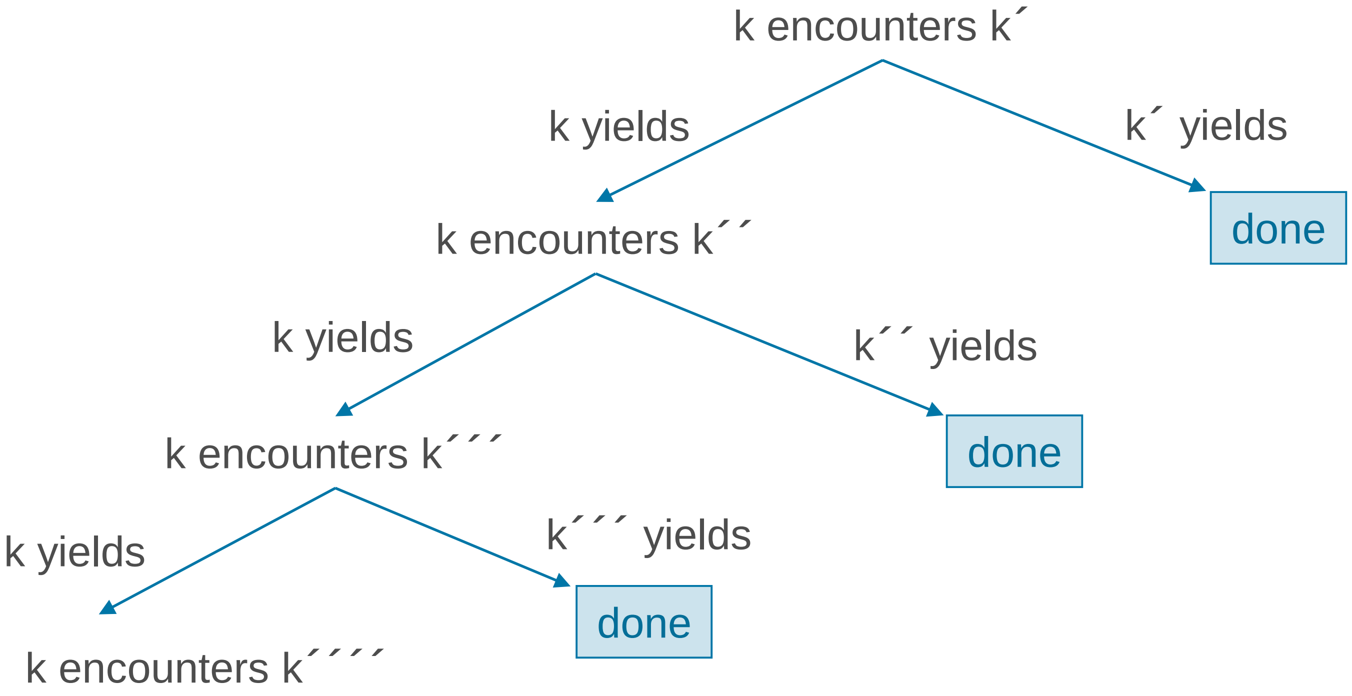
Two options:

M1 : k_{old} **remains** in A[i] and k tries insert position $h(k)- s(j+1,k)$
M2 : k **pushes** k_{old} to $h(k_{old}) - s(j´+1, k_{old})$

If M1 or M2 finds a vacant position:
then insert the appropriate key $\Rightarrow$ done
otherwise proceed with M1 or M2

Brent's Algorithm

Only follow M1



Time for unsuccessful search remains unchanged

$C_n' \approx 1 / (1-\alpha)$

Time for successful search is reduced to

$C_n^{Brent} \approx 1 + \alpha/2 + \alpha^3/4 + \alpha^4/15 + \alpha^5/18 + \ldots < 2.5$

Example for Brent's Algorithm

Example: N=7, $\boldsymbol{K}$ = {0, 1, ..., 500}, Keys: 12, 53, 5, 15, 2, 19

$h_1(k)$ = k mod 7, $h_2(k)$ = 1 + k mod 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
				53	12	

5

Insert of 5

$h_1(5)$ = 5 mod 7 = 5 occupied,

calculate
$h_2(5)$ = 1+ 5 mod 5 = 1
and
$h_2(12)$ = 1 + 12 mod 5 = 3

5: probe (5-1)mod7=4 occupied,
12: probe (5-3)mod7=2 vacant,
therfore 12 yields to 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
19	15	12		53	5	2

2 19

Insert of 2

$h_1(2)$ = 2 mod 7 = 2 occupied,

calculate
$h_2(2)$ = 1+ 2 mod 5 = 3
and
$h_2(12)$ = 1 + 12 mod 5 = 3

2: probe (2-3)mod7=6
vacant

Insert of 19

$h_1(19)$ = 19 mod 7 = 5 occupied,

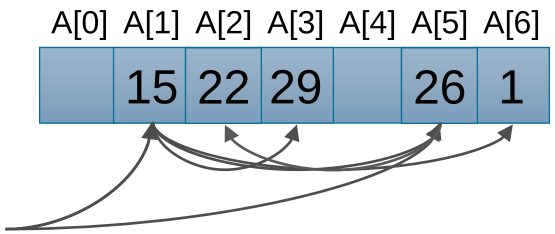
calculated
$h_2(19)$ = 1+ 19 mod 5 = 5
and
$h_2(5)$ = 1 + 5 mod 5 = 1

19: probe (5-5)mod7=0 vacant

Further Example for Brent's Algorithm

Example

$h_1(k) = k \bmod 7$, $h_2(k) = 1 + k \bmod 5$
key sequence 15, 22, 1, 29, 26



$h_1(15) = 1$

$h_1(22) = 1$ $h_2(22) = 3$

$h_1(1) = 1$ $h_2(1) = 2$

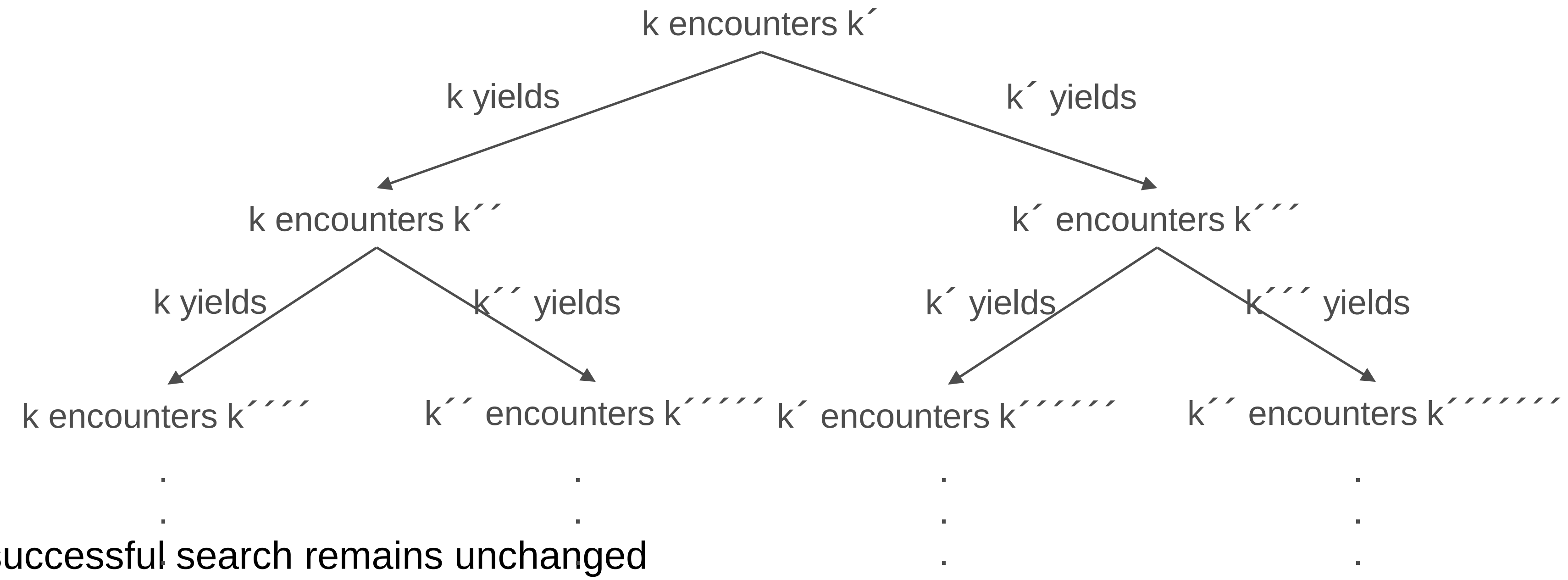
$h_1(29) = 1$ $h_2(29) = 5$

$h_1(26) = 5$ $h_2(26) = 2$ occupied
$h_2(22) = 3$ vacant

- Durchschnittliche Suchzeit = (1+2+2+2+2)/5 = 9/5 = 1.8

Binary Tree Probing

Follow both M1 as well as M2 simultaneously, until a vacant position is found in a subbranch



- Time for unsuccessful search remains unchanged
  - $C_n^{´} \approx 1 / (1-\alpha)$
- Time for successful search is reduced to
  - $C_n^{\text{binary tree}} < 2.2$

Example for Binary Tree Probing

Example:
N=7, ***K*** = {0, 1, ..., 500},
Keys: 12, 53, 5, 15, 2, 19, 21

$h_1(k)$ = k mod 7,
$h_2(k)$ = 1 + k mod 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
				53	12	

5 → A[5]

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
19	15	12		53	5	2

21 → A[0], 2 → A[2], 19 → A[5]

Insert of 5

$h_1(5)$ = 5 mod 7 = 5 occupied,
calculate
$h_2(5)$ = 1+ 5 mod 5 = 1
and
$h_2(12)$ = 1 + 12 mod 5 = 3

5: probe (5-1)mod7=4 occupied,
12: probe (5-3)mod7=2 vacant,
therefore 12 yields to 5

Insert of 2

$h_1(2)$ = 2 mod 7 = 2 occupied,
calculate
$h_2(2)$ = 1+ 2 mod 5 = 3
and
$h_2(12)$ = 1 + 12 mod 5 = 3

2: probe (2-3)mod7=6
vacant

Insert of 19

$h_1(19)$ = 19 mod 7 = 5 occupied,
calculate
$h_2(19)$ = 1+ 19 mod 5 = 5
and
$h_2(5)$ = 1 + 5 mod 5 = 1

19: probe (5-5)mod7=0 vacant

Insert of 21

$h_1(21)$ = 21 mod 7 = 0 occupied,
calculate
$h_2(21)$ = 1+ 21 mod 5 = 2 and $h_2(19)$ = 1 + 19 mod 5 = 5

probe 21: (0-2)mod7=5 occupied and 19: (0-5)mod7=2 occupied,
continue probing for 21 AND 19

Example for Binary Tree Probing

Example:
N=7, ***K*** = {0, 1, ..., 500},
Keys: 12, 53, 5, 15, 2, 19, 21

$h_1(k)$ = k mod 7,
$h_2(k)$ = 1 + k mod 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
19	15	12		53	5	2

21

Subbranch for 21

calculate
$h_2(21)$ = 1 + 21 mod 5 = 2
and
$h_2(5)$ = 1 + 5 mod 5 = 1

probe 21: (5-2)mod7=3 and 5: (5-1)mod7=4
Position 3 vacant, insert 21

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
19	15	12	21	53	5	2

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
21	15	12		53	5	2

19

Subbranch for 19
(assumption: 21 would have displaced 19)

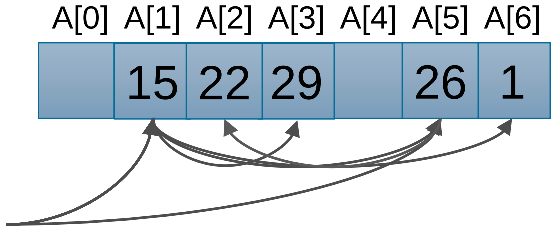
calculate
$h_2(19)$ = 1 + 19 mod 5 = 5
and
$h_2(12)$ = 1 + 12 mod 5 = 3

probe 19: (2-5)mod7=4 and 12: (2-3)mod7=6
both occupied

Further Example for Binary Tree Probing

Example

- $h_1(k) = k \bmod 7$, $h_2(k) = 1 + k \bmod 5$
  key sequence 15, 22, 1, 29, 26



$h_1(15) = 1$

$h_1(22) = 1$ $h_2(22) = 3$

$h_1(1) = 1$ $h_2(1) = 2$

$h_1(29) = 1$ $h_2(29) = 0$

$h_1(26) = 5$ $h_2(26) = 2$

$h_2(22) = 3$

- Durchschnittliche Suchzeit = (1+2+2+2+2)/5 = 9/5 = 1.8
- hier identisch mit Brent‘s Algorithmus, da jeweils im ersten Sondierungsschritt Lücke gefunden wird

Ordered Double Hashing

Aim: **Improvement** of **unsuccessful search**

Search:

- k´> k in probe sequence => search unsuccessful

Inserting:

- **Smaller keys displace larger keys** (ordered hashing)

Invariant:

- **all keys in the probe sequence before k are smaller than k**
  (but not necessarily sorted in ascending order)

Problems:

- Displacement can trigger "**chain reaction**"
- k´ displaced by k: Position of k´ in probe sequence? => (s(j,k) - s(j-1,k) = s(1,k)) $1 \leq j \leq N$

Example :: Ordered Double Hashing

Hash functions:

$h_1(k) = k \bmod 7,\ h_2(k) = 1 + k \bmod 5$

Key sequence: 15, 2, 43, 4, 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
15	8	2	43	4		

$h_1(15) = 1$

$h_1(2) = 2$

$h_1(43) = 1$ $\quad 43>15 \quad h_2(43) = 4 \quad (1-4) \bmod 7 = 4$

$h_1(4) = 4$ $\quad 4<43 \quad h_2(43) = 4 \quad (4-4) = 0$

$h_1(8) = 1$ $\quad 8<15 \quad h_2(15) = 1 \quad (1-1) = 0$

$\quad 15<43 \quad h_2(43) = 4 \quad (0-4) \bmod 7 = 3$

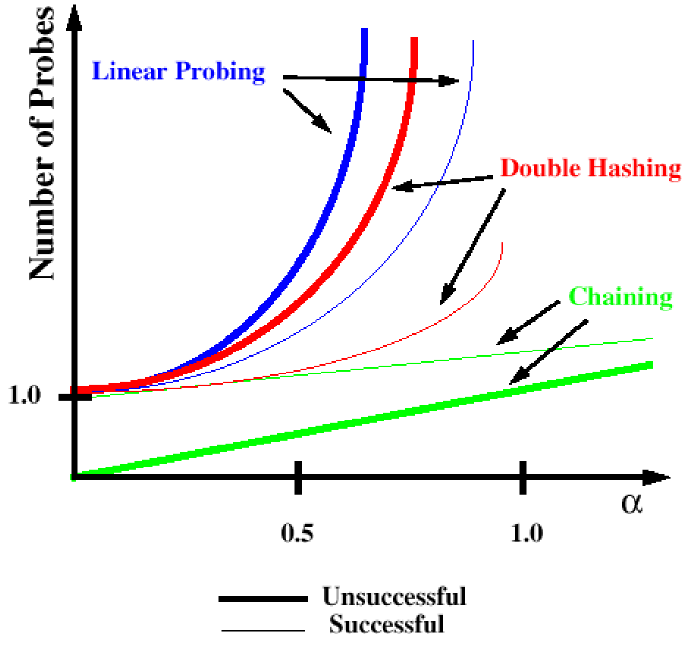
Hashing Complexity :: Summary

Occupancy factor α = average number of keys per array index

$$= \frac{\#\,stored\ keys}{size\ of\ hash\ table} = \frac{|S|}{N} = \frac{n}{N}$$

Complexity after probabilistic analysis

	Expected number of probes	
	unsuccessful	**successful**
Chaining	α	$1 + \alpha/2$
Linear Probing	$1 + 1/(1-\alpha)^2$	$1 + 1/(1-\alpha)$
Double Hashing	$1/(1-\alpha)$	$1/\alpha \cdot \ln 1/(1-\alpha)$



Double 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