Datastructuren

Data Structures

Hendrik Jan Hoogeboom
Mark van den Bergh

Informatica – LIACS
Universiteit Leiden

najaar 2024

Table of Contents I

Contents

ADT map, dictionary

associative array, [_hash-_]_map_, _symbol table_, or _dictionary_ wiki

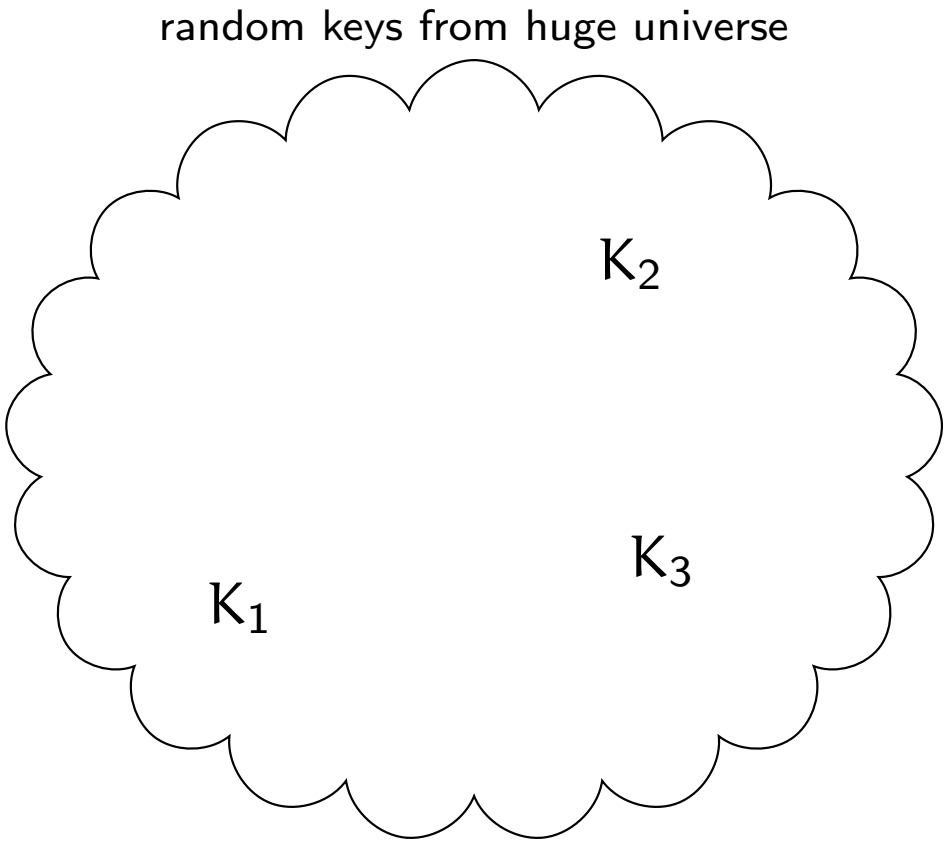
is composed of a collection of (key, value) pairs;
each possible key appears at most once

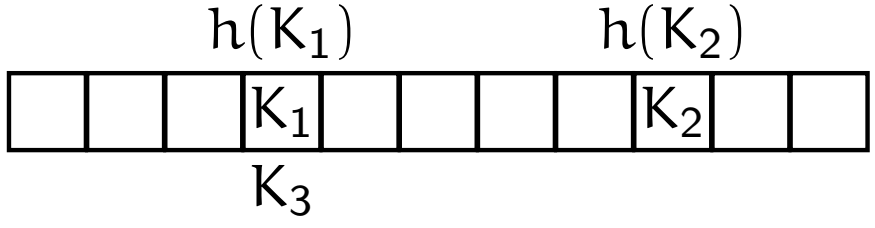
	find		*insert*		*delete*		
	av	wc	av	wc	av	wc	order
unordered list		n		1		n	no
bin tree	$\log n$	n	$\log n$	n	$\log n$	n	yes
balanced		$\log n$		$\log n$		$\log n$	yes
hash table	1	n	1	n	1	n	no

worst case

av=average, wc=worst case

hashing

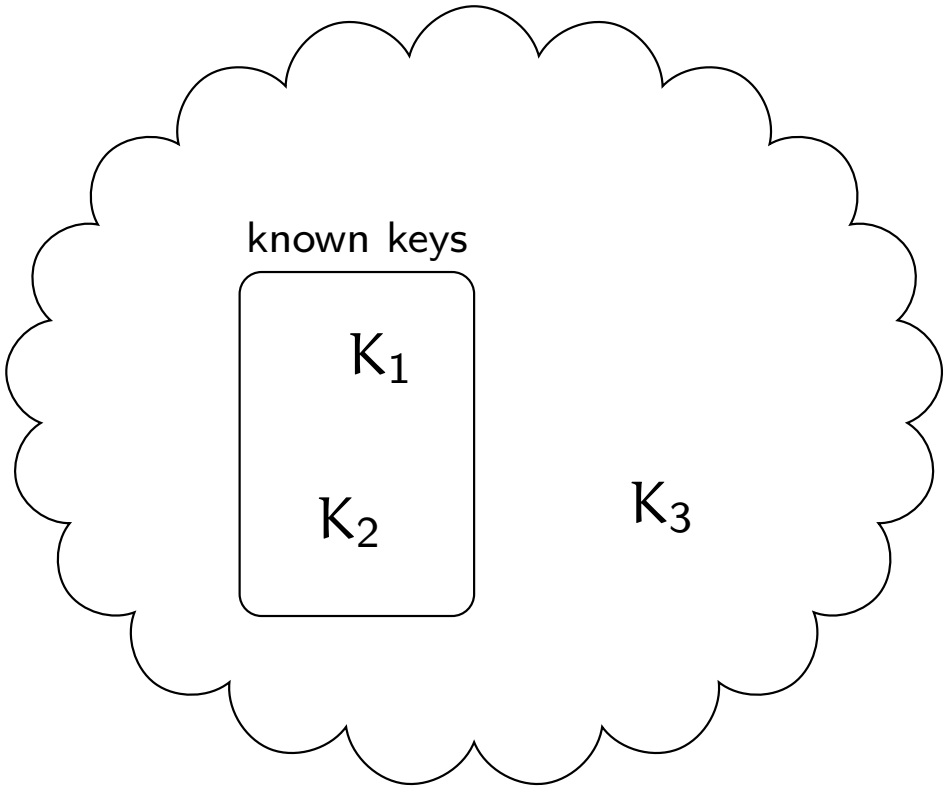


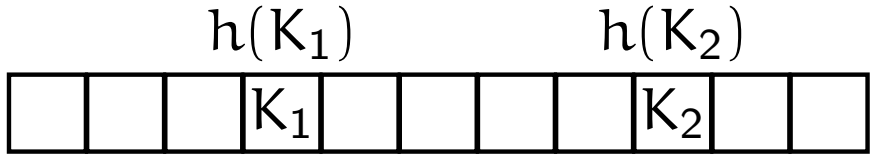


$$h(K_3) = h(K_1)$$

collision

perfect hashing





design h without collisions

hashing basics

- store keys of arbitrary size (usually large domains) in table of fixed size (usually small)
- store passes, checksums (MD5, CRC32)
- implement ADT unordered set: find, insertion and deletion in (avg) constant time

- *hash function* / *address function* calculates position in table: $\mathrm{h}(\mathrm{K})$ mod T (TableSize)
- *collision*: attempt to store key K when $\mathrm{h}(\mathrm{K})$ is occupied

hashing

- **perfect hashing**
  keys are known a-priori; can avoid collision
- **open addressing**[2]
  collision: store key elsewhere in same array
  linear, *quadratic*, *double* hashing
  primary, *secundary clustering*
- **chained hashing**
  store multiple keys at the same address
  (i.e. table entries are linked lists of items with same hash)

[2] also called closed hashing :)

Contents

keywords Pascal [Cichelli]

$$h(w) = |w| + v(\text{first}(w)) + v(\text{last}(w)),$$

α	a	b	c	f	g	h	i	l	m	n	p	r	s	t	u	v	w	y	other
$v(\alpha)$	11	15	1	15	3	15	13	15	15	13	15	14	6	6	14	10	6	13	0

3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
end	else	case	downto	goto	to	otherwise	type	while	const	div	and	set	or	of	mod	file	record	packed	not	then	procedure	with	repeat	var	in	array	if	nil	for	begin	until	label	

$$h(\text{case}) = |\text{case}| + v(\text{c}) + v(\text{e}) = 4 + 1 + 0 = 5$$
$$h(\text{help}) = |\text{help}| + v(\text{h}) + v(\text{p}) = 4 + 15 + 15 = 34$$

Contents

open adressing

- store in array *table size* T
- upon collision, attempt to store hash elsewhere
- hash function: $h(K, i) = (h(K) - p(i)) \bmod T$
  address function, *step function*
  - *linear probing*: $p(i)$ is linear, e.g., $p(i) = i$
  - *quadratic probing*: $p(i)$ is quadratic, e.g., $p(i) = \pm i^2$
  - *double hashing*: both depend on K $p(i) = i \cdot p(K)$

operations

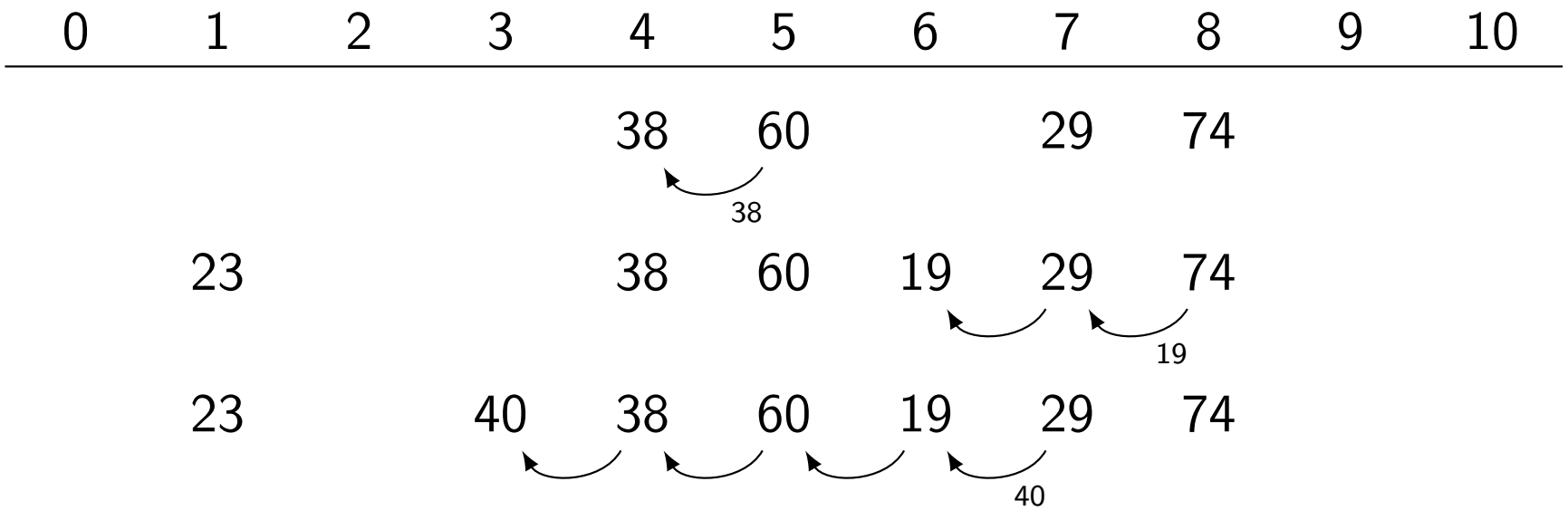
- *always* probe at $h(K, 0), h(K, 1), h(K, 2), \ldots$
  INSERT: until we find empty spot
  FIND: until we find K (success), or an empty spot (failure)
  DELETE: keep tag for each cell: active/deleted/empty

linear hashing

keys $60_5, 29_7, 74_8, 38_5, 19_8, 23_1$ and 40_7

address function $h(K) = K \bmod 11$

probe function $f(i) = i$



linear step size

- *primary clustering*: "nearby" keys in same cluster
- careful! step size *coprime* to table size
  if not, inserts can fail even if table is not full: not all positions are probed

$h(K, i) = K - 3i \bmod 10$

0	1	2	3	4	5	6	7	8	9
65		32	43		55	72			19

neighbours (relative to step size 3):

0	3	6	9	2	5	8	1	4	7
65	43	72	19	32	55				

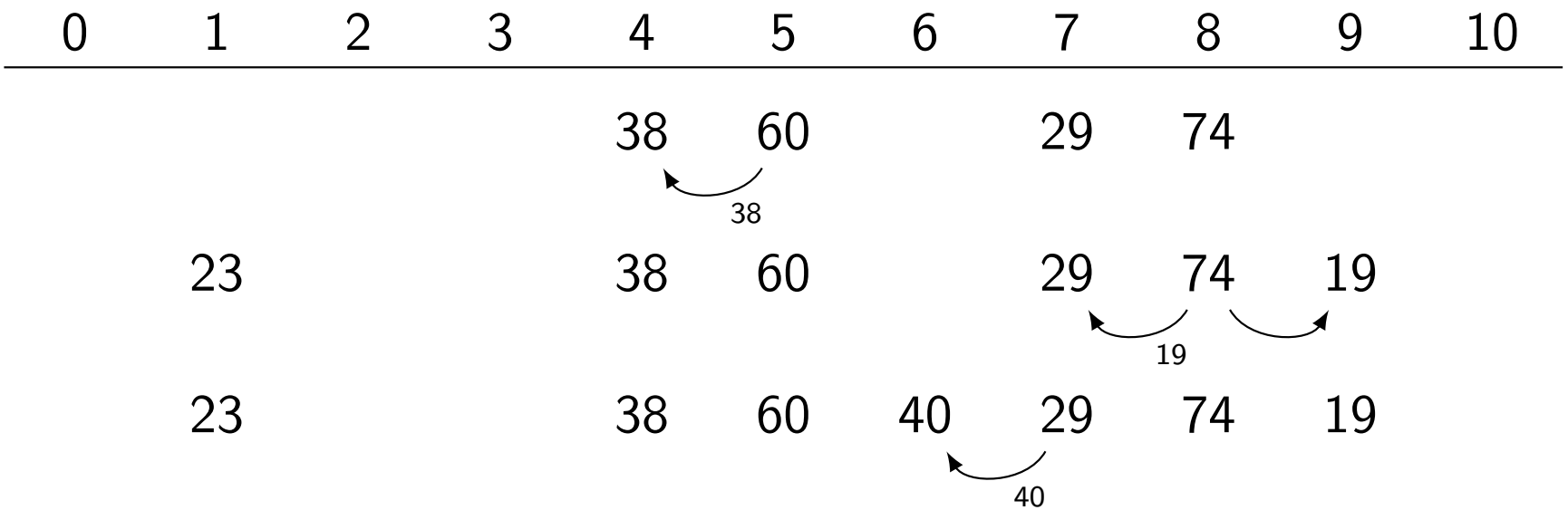
quadratic hashing

keys $60_5, 29_7, 74_8, 38_5, 19_8, 23_1$ and 40_7

address function $h(K) = K \bmod 11$

hash function $h(K, i) = h(K) \mp i^2 \bmod 11$

probes at $h(K) \mp 1, h(K) \mp 4, h(K) \mp 9, \ldots$



- *secondary clustering*: keys with same hash can cluster

double hashing

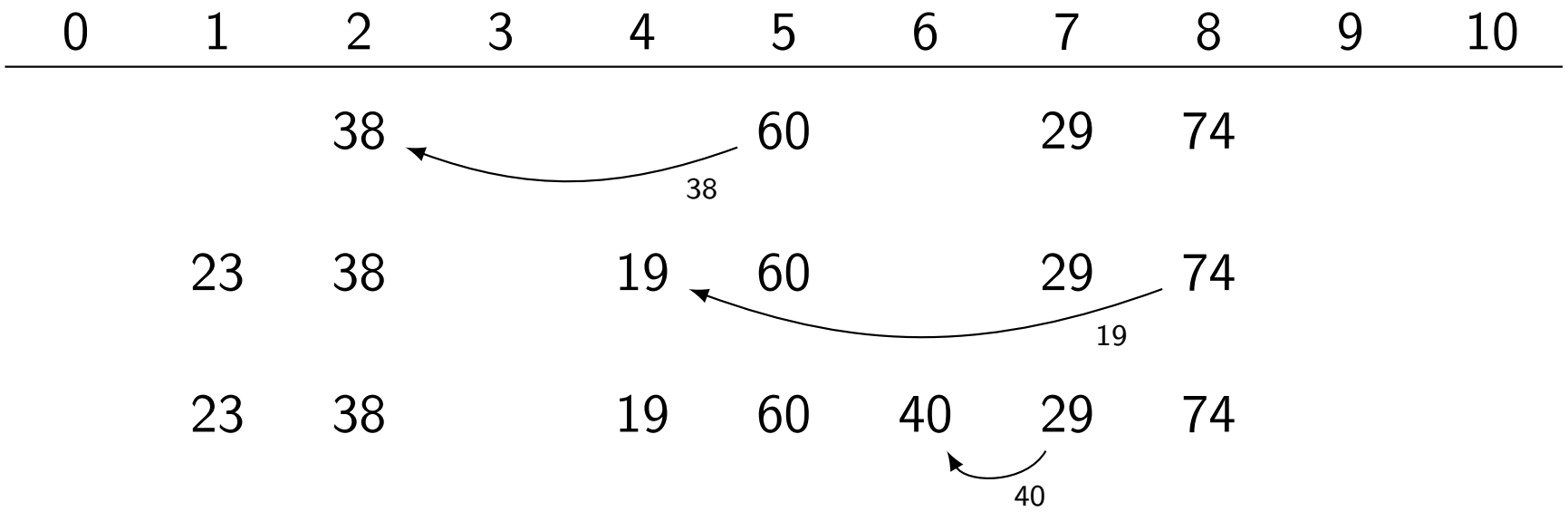
keys 60_5, 29_7, 74_8, 38_5, 19_8, 23_1 and 40_7

address function $h(K) = K \bmod 11$

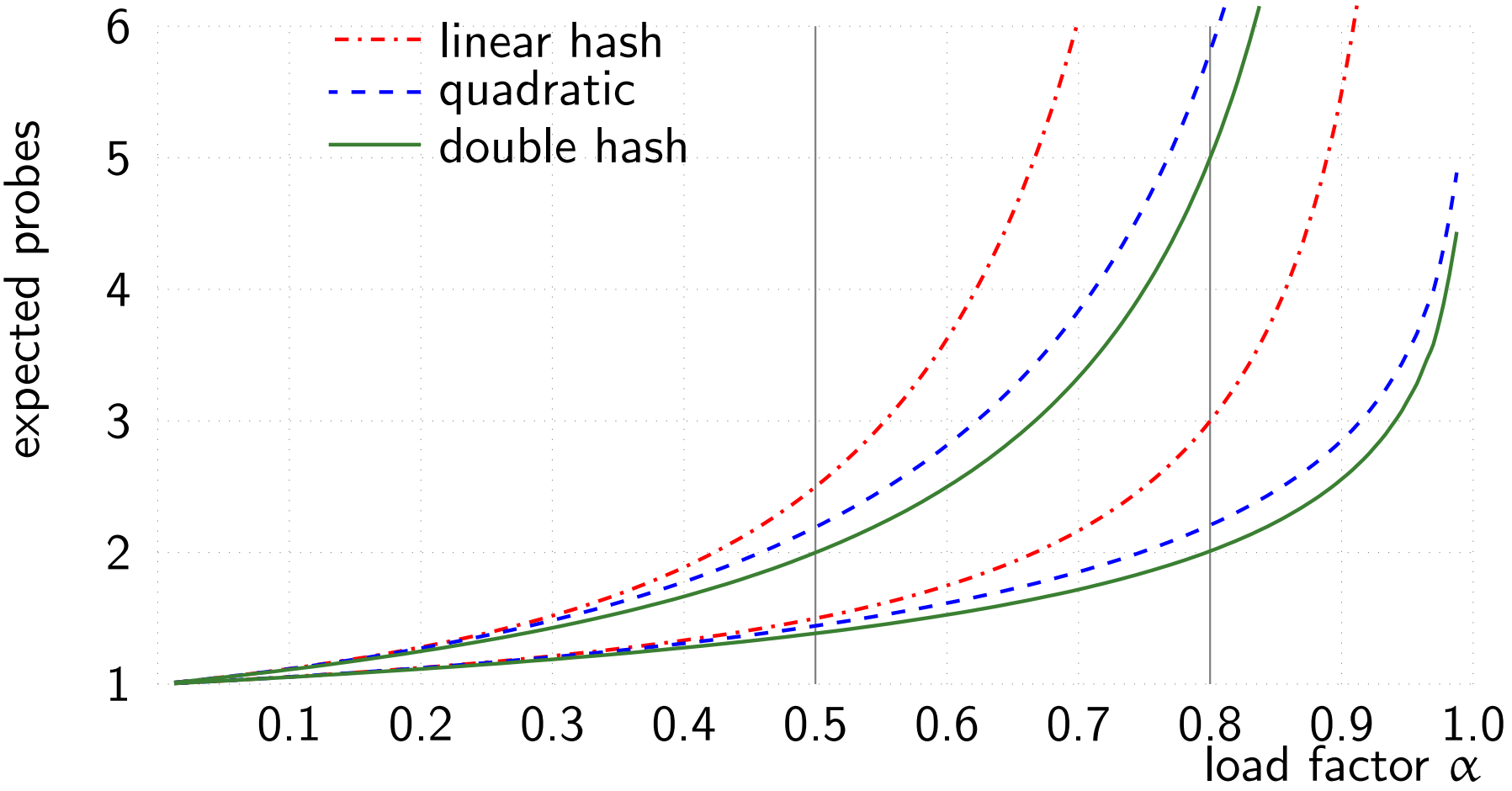
probe function $p(K) = (K \bmod 4) + 1$

NB. (1) coprime (2) independent

hash function $h(K, i) = h(K) - i \cdot p(K) \bmod 11$



	find / successful		add / unsuccessful	
linear	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	1.5–3	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$	2.5–13
secondary	$1+\ln(\frac{1}{1-\alpha})-\frac{\alpha}{2}$	1.4–2.2	$\frac{1}{1-\alpha}-\alpha+\ln(\frac{1}{1-\alpha})$	2.2–5.8
double	$\frac{1}{\alpha}\ln(\frac{1}{1-\alpha})$	1.4–2.0	$\frac{1}{1-\alpha}$	2–5

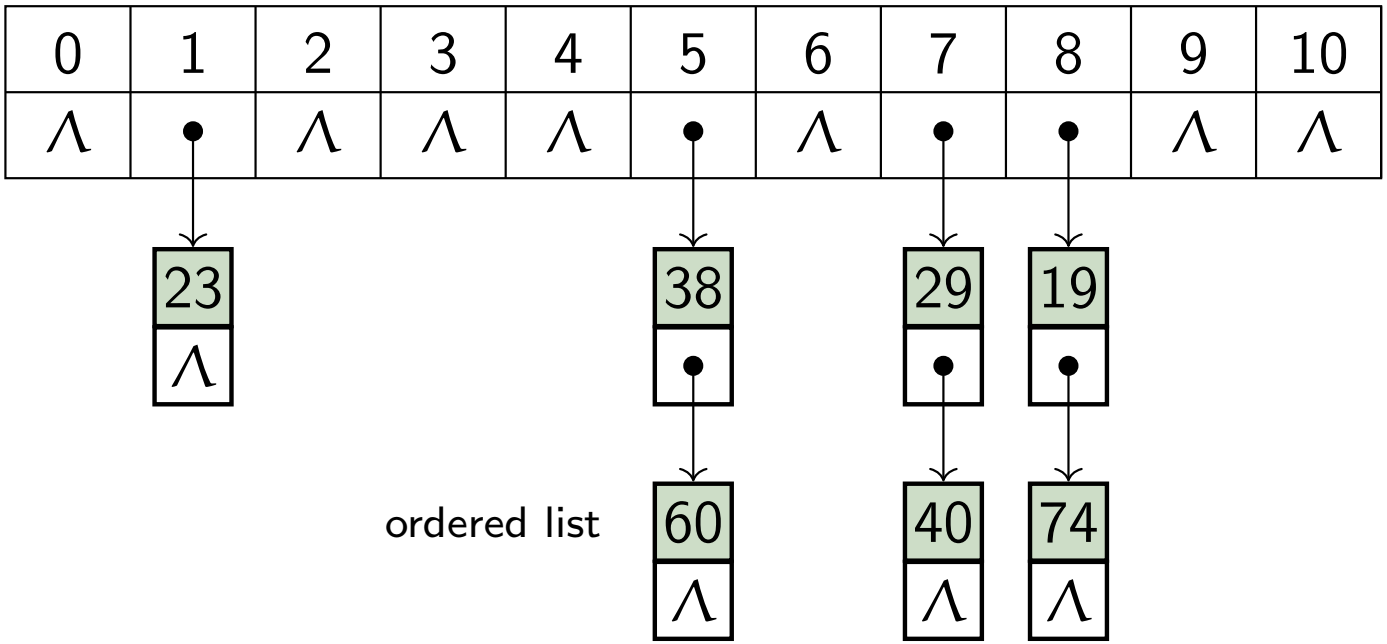


Contents

chaining

keys 60_5, 29_7, 74_8, 38_5, 19_8, 23_1 and 40_7

address function $h(K) = K \bmod 11$



Contents

choosing a hash function

A good hash function $\mathrm{h}(\mathrm{K})$ should

- be fast to compute,
- evenly distribute the keys over the table, and
- depend on all 'distinctive bits' of the key.

Techniques:

- extraction selected bits of key
- division modulo TableSize choose carefully (prime)
- folding chop key, combine (add,xor)
- mid-squaring square key, take middle bits $\sim$ random
- radix (base) conversion

MurmurHash

```
Murmur3_32(key, len, seed)
    // integer arithmetic with unsigned 32 bit integers.
    c1 := 0xcc9e2d51
    c2 := 0x1b873593
    r1 := 15
    r2 := 13
    m := 5
    n := 0xe6546b64

    hash := seed
    for each fourByteChunk of key
        k := fourByteChunk
        k := k * c1
        k := (k << r1) OR (k >> (32-r1))
        k := k * c2
        hash := hash XOR k
        hash := (hash << r2) OR (hash >> (32-r2))
        hash := hash * m + n

    with any remainingBytesInKey
        \\ (also do Endian swapping on big-endian machines.)
        remainingBytes := remainingBytesInKey * c1
        remainingBytes := (remainingBytes << r1) OR (remainingBytes >> (32 - r1))
        remainingBytes := remainingBytes * c2
        hash := hash XOR remainingBytes

    hash := hash XOR len
    hash := hash XOR (hash >> 16)
    hash := hash * 0x85ebca6b
    hash := hash XOR (hash >> 13)
    hash := hash * 0xc2b2ae35
    hash := hash XOR (hash >> 16)
```

end.