



CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture#11:
Hashing (R&G ch. 11)



CMU SCS

Outline

➡ • (static) hashing
• extendible hashing
• linear hashing
• Hashing vs B-trees

Faloutsos CMU SCS 15-415 2



CMU SCS

(Static) Hashing

Problem: “*find EMP record with ssn=123*”
What if disk space was free, and time was at premium?

Faloutsos CMU SCS 15-415 3

CMU SCS

Hashing

A: Brilliant idea: key-to-address transformation:

123; Smith; Main str

#0 page

#123 page

#999,999,999

Faloutsos CMU SCS 15-415 4

CMU SCS

Hashing

Since space is NOT free:

- use M , instead of 999,999,999 slots
- hash function: $h(key) = slot-id$

123; Smith; Main str

#0 page

#123 page

#999,999,999

Faloutsos CMU SCS 15-415 5

CMU SCS

Hashing

Typically: each hash bucket is a page, holding many records:

123; Smith; Main str

#0 page

#h(123)

M

Faloutsos CMU SCS 15-415 6

CMU SCS

Hashing

Notice: could have **clustering**, or non-clustering versions:

Faloutsos CMU SCS 15-415 7

CMU SCS

Hashing

Notice: could have clustering, or **non-clustering** versions:

Faloutsos CMU SCS 15-415 8

CMU SCS

Indexing- overview

- hashing
 - ➡ – hashing functions
 - size of hash table
 - collision resolution
- extendible hashing
- Hashing vs B-trees

Faloutsos CMU SCS 15-415 9



CMU SCS

Design decisions

- 1) formula $h()$ for hashing function
- 2) size of hash table M
- 3) collision resolution method

Faloutsos CMU SCS 15-415 10



CMU SCS

Design decisions - functions

- Goal: uniform spread of keys over hash buckets
- Popular choices:
 - Division hashing
 - Multiplication hashing

Faloutsos CMU SCS 15-415 11



CMU SCS

Division hashing

$$h(x) = (a \cdot x + b) \bmod M$$

- eg., $h(ssn) = (ssn) \bmod 1,000$
 - gives the last three digits of ssn
- M : size of hash table - choose a prime number, defensively (why?)

Faloutsos CMU SCS 15-415 12



CMU SCS

Division hashing

- eg., $M=2$; hash on driver-license number (dln), where last digit is 'gender' (0/1 = M/F)
- in an army unit with predominantly male soldiers
- Thus: avoid cases where M and keys have common divisors - prime M guards against that!

Faloutsos CMU SCS 15-415 13



CMU SCS

Multiplication hashing

$$h(x) = [\text{fractional-part-of} (x * \phi)] * M$$

- ϕ : golden ratio ($0.618... = (\sqrt{5}-1)/2$)
- in general, we need an irrational number
- advantage: M need not be a prime number
- but ϕ must be irrational

Faloutsos CMU SCS 15-415 14



CMU SCS

Other hashing functions

- quadratic hashing (bad)
- ...

Faloutsos CMU SCS 15-415 15



CMU SCS

Other hashing functions

- quadratic hashing (bad)
- ...
- conclusion: use division hashing

Faloutsos CMU SCS 15-415 16



CMU SCS

Design decisions

- 1) formula $h()$ for hashing function
- ➡ 2) size of hash table M
- 3) collision resolution method

Faloutsos CMU SCS 15-415 17



CMU SCS

Size of hash table

- eg., 50,000 employees, 10 employee-records / page
- Q: $M = ??$ pages/buckets/slots

Faloutsos CMU SCS 15-415 18



CMU SCS

Size of hash table

- eg., 50,000 employees, 10 employees/page
- Q: $M = ??$ pages/buckets/slots
- A: utilization $\sim 90\%$ and
 - M : prime number

Eg., in our case: $M = \text{closest prime to } 50,000/10 / 0.9 = 5,555$

Faloutsos CMU SCS 15-415 19



CMU SCS

Design decisions

- 1) formula $h()$ for hashing function
- 2) size of hash table M
- ➡ 3) collision resolution method

Faloutsos CMU SCS 15-415 20



CMU SCS

Collision resolution

- Q: what is a 'collision'?
- A: ??

Faloutsos CMU SCS 15-415 21

CMU SCS

Collision resolution

123; Smith; Main str.

#0 page

#h(123)

M

Faloutsos CMU SCS 15-415 22

CMU SCS

Collision resolution

- Q: what is a 'collision'?
- A: ??
- Q: why worry about collisions/overflows?
(recall that buckets are ~90% full)
- A: 'birthday paradox'

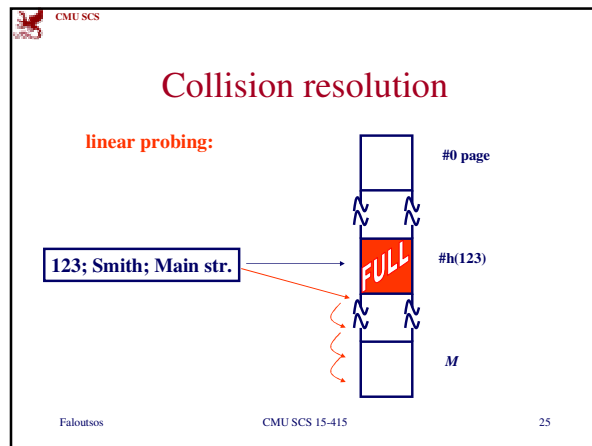
Faloutsos CMU SCS 15-415 23

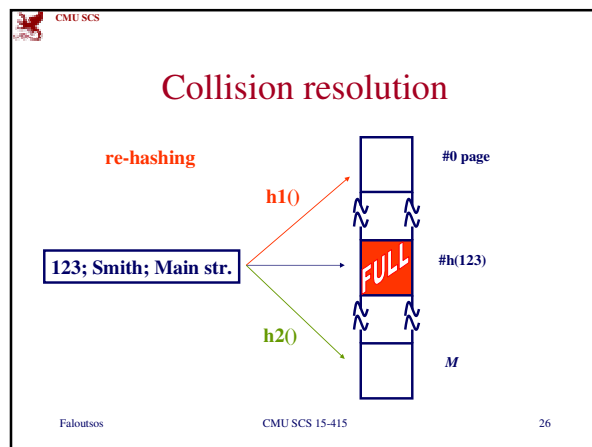
CMU SCS

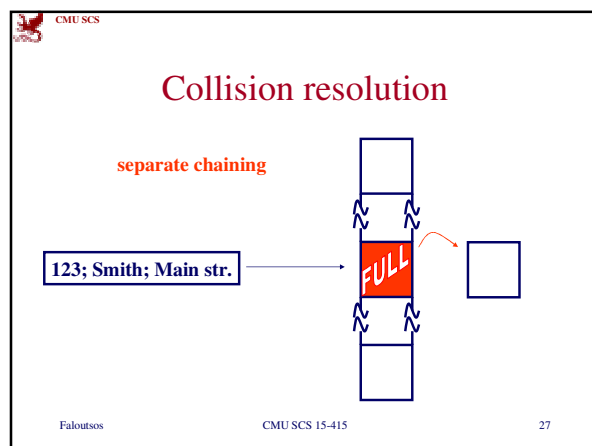
Collision resolution

- open addressing
 - linear probing (ie., put to next slot/bucket)
 - re-hashing
- separate chaining (ie., put links to overflow pages)

Faloutsos CMU SCS 15-415 24









CMU SCS

Design decisions - conclusions

- function: division hashing
 - $h(x) = (a * x + b) \bmod M$
- size M : ~90% util.; prime number.
- collision resolution: separate chaining
 - easier to implement (deletions!);
 - no danger of becoming full

Faloutsos CMU SCS 15-415 28



CMU SCS

Outline

- (static) hashing
- ➡ • extendible hashing
- linear hashing
- Hashing vs B-trees

Faloutsos CMU SCS 15-415 29



CMU SCS

Problem with static hashing

- problem: overflow?
- problem: underflow? (underutilization)

Faloutsos CMU SCS 15-415 30

CMU SCS

Solution: Dynamic/extendible hashing

- idea: shrink / expand hash table on demand..
- ..dynamic hashing

Details: how to grow gracefully, on overflow?

Many solutions - One of them: 'extendible hashing' [Fagin et al]

Faloutsos CMU SCS 15-415 31

CMU SCS

Extendible hashing

123; Smith; Main str. →

#0 page

#h(123)

M

FULL

Faloutsos CMU SCS 15-415 32

CMU SCS

Extendible hashing

solution:
split the bucket in two

123; Smith; Main str. →

#0 page

#h(123)

M

FULL

Faloutsos CMU SCS 15-415 33


CMU SCS

Extendible hashing

in detail:

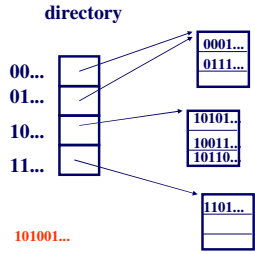
- keep a directory, with ptrs to hash-buckets
- Q: how to divide contents of bucket in two?
- A: hash each key into a very long bit string; keep only as many bits as needed

Eventually:

Faloutsos
CMU SCS 15-415
34


CMU SCS

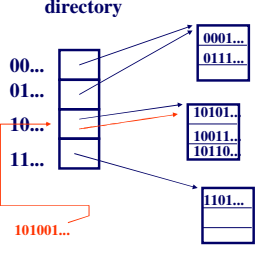
Extendible hashing



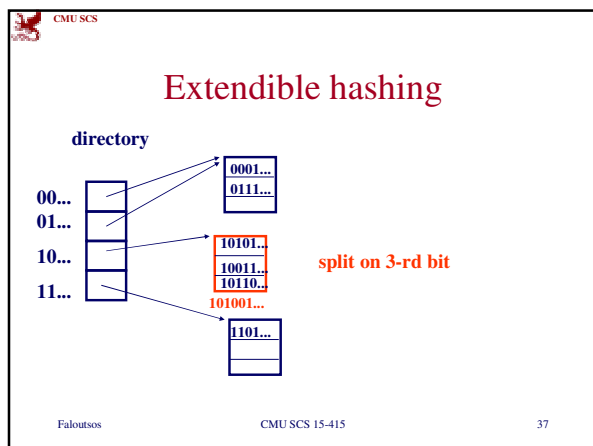
Faloutsos
CMU SCS 15-415
35

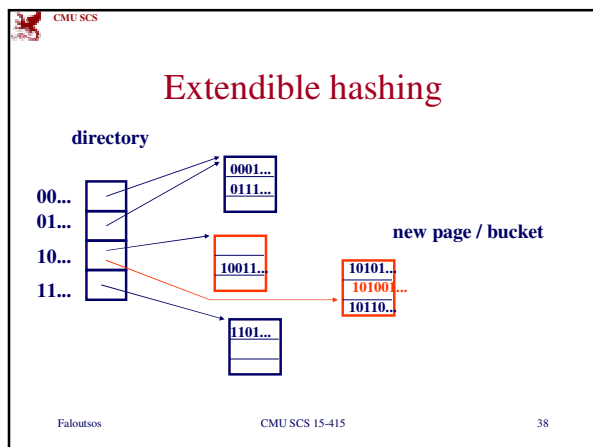

CMU SCS

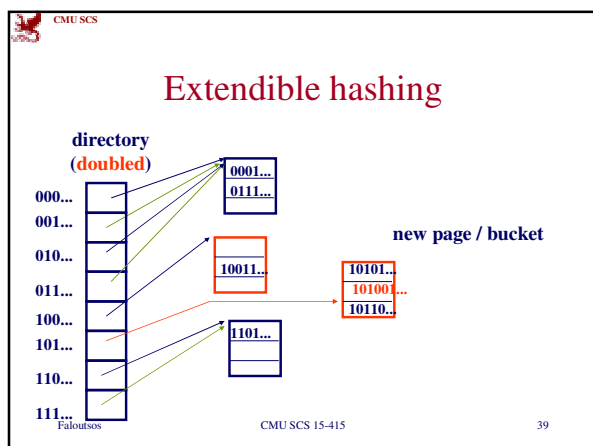
Extendible hashing

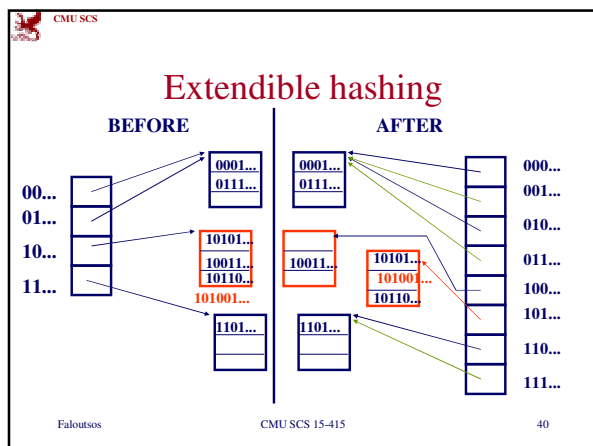


Faloutsos
CMU SCS 15-415
36









CMU SCS

Extendible hashing

- Summary: directory doubles on demand
- or halves, on shrinking files
- needs 'local' and 'global' depth

Faloutsos CMU SCS 15-415 41

CMU SCS

Outline

- (static) hashing
- extendible hashing
- ➡ • linear hashing
- Hashing vs B-trees

Faloutsos CMU SCS 15-415 42



CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion

Faloutsos CMU SCS 15-415 43



CMU SCS

Linear hashing

Motivation: ext. hashing needs directory
etc etc; which doubles (ouch!)

Q: can we do something simpler, with
smoother growth?

Faloutsos CMU SCS 15-415 44



CMU SCS

Linear hashing

Motivation: ext. hashing needs directory
etc etc; which doubles (ouch!)

Q: can we do something simpler, with
smoother growth?

A: split buckets from left to right,
regardless of which one overflowed
(‘crazy’, but it works well!) - Eg.:

Faloutsos CMU SCS 15-415 45

Linear hashing

Initially: $h(x) = x \bmod N$ ($N=4$ here)

Assume capacity: 3 records / bucket

Insert key '17'

bucket- id	0	1	2	3
	4 8	5 9 13	6	7 11

Faloutsos CMU SCS 15-415 46

Linear hashing

Initially: $h(x) = x \bmod N$ ($N=4$ here)

17 overflow of bucket#1

bucket- id	0	1	2	3
	4 8	5 9 13	6	7 11

Faloutsos CMU SCS 15-415 47

Linear hashing

Initially: $h(x) = x \bmod N$ ($N=4$ here)

17 overflow of bucket#1

Split #0, anyway!!!

bucket- id	0	1	2	3
	4 8	5 9 13	6	7 11

Faloutsos CMU SCS 15-415 48

CMU SCS

Linear hashing

Initially: $h(x) = x \bmod N$ ($N=4$ here)
Split #0, anyway!!!

Q: But, how?

bucket- id 0 1 2 3

4	8	5 9 13	6	7 11
---	---	-----------	---	------

Faloutsos CMU SCS 15-415 49

CMU SCS

Linear hashing

A: use two h.f.: $h0(x) = x \bmod N$
 $h1(x) = x \bmod (2*N)$

bucket- id 0 1 2 3

4	8	5 9 13	6	7 11
---	---	-----------	---	------

Faloutsos CMU SCS 15-415 50

CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \bmod N$
 $h1(x) = x \bmod (2*N)$

bucket- id 0 1 2 3 4

8	5 9 13	6	7 11	4
---	-----------	---	------	---

17

Faloutsos CMU SCS 15-415 51

CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \bmod N$
 $h1(x) = x \bmod (2*N)$

bucket- id 0 1 2 3 4

8	5 9 13	6	7 11	4
---	-----------	---	------	---

↓

17

overflow

Faloutsos CMU SCS 15-415 52

CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \bmod N$
 $h1(x) = x \bmod (2*N)$

split ptr

bucket- id 0 1 2 3 4

8	5 9 13	6	7 11	4
---	-----------	---	------	---

↓

17

overflow

Faloutsos CMU SCS 15-415 53

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- ➡ • search algo
- insertion/split algo
- deletion

Faloutsos CMU SCS 15-415 54

CMU SCS

Linear hashing - searching?

$h0(x) = x \bmod N$ (for the un-split buckets)
 $h1(x) = x \bmod (2*N)$ (for the splitted ones)

bucket-id 0 1 2 3 4

8 5 9 6 7 11 4

13

17 overflow

Faloutsos CMU SCS 15-415 55

CMU SCS

Linear hashing - searching?

Q1: find key '6'? Q2: find key '4'?
 Q3: key '8'?

bucket-id 0 1 2 3 4

8 5 9 6 7 11 4

13

17 overflow

Faloutsos CMU SCS 15-415 56

CMU SCS

Linear hashing - searching?

Algo to find key 'k':

- compute $b = h0(k)$;
 - if $b < split_ptr$, compute $b = h1(k)$
- search bucket b

Faloutsos CMU SCS 15-415 57



CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- ➡ • insertion/split algo
- deletion

Faloutsos CMU SCS 15-415 58



CMU SCS

Linear hashing - insertion?

Algo: insert key ' k '

- compute appropriate bucket ' b '
- if the **overflow criterion** is true
 - split the bucket of 'split-ptr'
 - split-ptr ++ (*)

Faloutsos CMU SCS 15-415 59



CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!
Q: suggestions?

Faloutsos CMU SCS 15-415 60

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!
 Q: suggestions?
 A1: space utilization $\geq u\text{-max}$

Faloutsos CMU SCS 15-415 61

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!
 Q: suggestions?
 A1: space utilization $> u\text{-max}$
 A2: avg length of ovf chains $> \text{max-len}$
 A3:

Faloutsos CMU SCS 15-415 62

CMU SCS

Linear hashing - insertion?

Algo: insert key ' k '

- compute appropriate bucket ' b '
- if the **overflow criterion** is true
 - split the bucket of 'split-ptr'
 - split-ptr ++ (*)

what if we reach the right edge??

Faloutsos CMU SCS 15-415 63

CMU SCS

Linear hashing - split now?

$h0(x) = x \bmod N$ (for the un-split buckets)
 $h1(x) = x \bmod (2*N)$ (for the splitted ones)

split ptr

0 1 2 3 4 5 6

Faloutsos CMU SCS 15-415 64

CMU SCS

Linear hashing - split now?

$h0(x) = x \bmod N$ (for the un-split buckets)
 $h1(x) = x \bmod (2*N)$ (for the splitted ones)

split ptr

0 1 2 3 4 5 6 7

Faloutsos CMU SCS 15-415 65

CMU SCS

Linear hashing - split now?

~~$h0(x) = x \bmod N$ (for the un-split buckets)~~
 ~~$h1(x) = x \bmod (2*N)$ (for the splitted ones)~~

split ptr

0 1 2 3 4 5 6 7

Faloutsos CMU SCS 15-415 66

CMU SCS

Linear hashing - split now?

~~$h_0(x) = x \bmod N$ (for the un-split buckets)~~
 $h_1(x) = x \bmod (2 * N)$ (for the splitted ones)

split ptr


0	1	2	3	4	5	6	7

Faloutsos CMU SCS 15-415 67

CMU SCS

Linear hashing - split now?

this state is called '**full expansion**'

split ptr


0	1	2	3	4	5	6	7

Faloutsos CMU SCS 15-415 68

CMU SCS

Linear hashing - observations

In general, at any point of time, we have at **most two** h.f. active, of the form:

- $h_n(x) = x \bmod (N * 2^n)$
- $h_{n+1}(x) = x \bmod (N * 2^{n+1})$

(after a full expansion, we have only one h.f.)

Faloutsos CMU SCS 15-415 69



CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- ➡ • deletion

Faloutsos CMU SCS 15-415 70



CMU SCS

Linear hashing - deletion?

- reverse of insertion:

Faloutsos CMU SCS 15-415 71



CMU SCS

Linear hashing - deletion?

- reverse of insertion:
- if the underflow criterion is met
 - contract!

Faloutsos CMU SCS 15-415 72

CMU SCS

Linear hashing - how to contract?

$h0(x) = \text{mod } N$ (for the un-split buckets)
 $h1(x) = \text{mod } (2*N)$ (for the splitted ones)

split ptr

Faloutsos CMU SCS 15-415 73

CMU SCS

Linear hashing - how to contract?

$h0(x) = \text{mod } N$ (for the un-split buckets)
 $h1(x) = \text{mod } (2*N)$ (for the splitted ones)

split ptr

Faloutsos CMU SCS 15-415 74

CMU SCS

Outline

- (static) hashing
- extendible hashing
- linear hashing
- ➡ • Hashing vs B-trees

Faloutsos CMU SCS 15-415 75



CMU SCS

Hashing - pros?

Faloutsos CMU SCS 15-415 76



CMU SCS

Hashing - pros?

- Speed,
 - on exact match queries
 - on the average

Faloutsos CMU SCS 15-415 77



CMU SCS

B(+)-trees - pros?

Faloutsos CMU SCS 15-415 78



CMU SCS

B(+)-trees - pros?

- Speed on search:
 - exact match queries, worst case
 - range queries
 - nearest-neighbor queries
- Speed on insertion + deletion
- smooth growing and shrinking (no re-org)

Faloutsos CMU SCS 15-415 79



CMU SCS

Conclusions

- B-trees and variants: in all DBMSs
- hash indices: in some
 - (but hashing is useful for joins - later...)

Faloutsos CMU SCS 15-415 80
