

CMU SCS

Carnegie Mellon Univ.
 Dept. of Computer Science
 15-415/615 – DB Applications

Faloutsos & Pavlo
 Lecture#11 (R&G ch. 11)
Hashing

CMU SCS

Outline

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

Faloutsos - Pavlo CMU SCS 15-415/615 2

CMU SCS

(Static) Hashing

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 3

CMU SCS

Hashing

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 4

CMU SCS **Review**

Hashing

Since space is NOT free:

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

#0 page
#123 page
#999,999,999

123; Smith; Main str

Faloutsos - Pavlo CMU SCS 15-415/615 5

CMU SCS **Review**

Hashing

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

#0 page
#h(123)
 M

123; Smith; Main str

Faloutsos - Pavlo CMU SCS 15-415/615 6

CMU SCS

Hashing

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

#0 page
#h(123)
 M

123; Smith; Main str.

Faloutsos - Pavlo CMU SCS 15-415/615 7

CMU SCS

Hashing


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

#0 page
#h(123)
 M

EMP file

234; Johnson; Forbes ave
123; Smith; Main str.
345; Tompson; Fifth ave

Faloutsos - Pavlo CMU SCS 15-415/615 8




CMU SCS

Indexing- overview

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

Faloutsos - Pavlo CMU SCS 15-415/615 9



CMU SCS

Design decisions

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 10




CMU SCS

Design decisions - functions

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 11



CMU SCS


Division hashing

Review

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

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

Faloutsos - Pavlo CMU SCS 15-415/615 12




CMU SCS

Review

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 - Pavlo CMU SCS 15-415/615 13



CMU SCS


Review

Multiplication hashing

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

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

Faloutsos - Pavlo CMU SCS 15-415/615 14




CMU SCS

Review

Other hashing functions

- quadratic hashing (bad)
- ...

Faloutsos - Pavlo CMU SCS 15-415/615 15




CMU SCS

Review

Other hashing functions

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

Faloutsos - Pavlo CMU SCS 15-415/615 16




CMU SCS

Design decisions

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 17




CMU SCS

Size of hash table

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 18



CMU SCS


Size of hash table

Review

- 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 - Pavlo CMU SCS 15-415/615 19



CMU SCS

Design decisions

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 20

CMU SCS

Collision resolution

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 21

CMU SCS

Collision resolution

Review

Faloutsos - Pavlo CMU SCS 15-415/615 22

CMU SCS

Collision resolution

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 23

CMU SCS

Collision resolution

Review

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

Faloutsos - Pavlo CMU SCS 15-415/615 24

CMU SCS **Review**

Collision resolution

linear probing:

123; Smith; Main str.

#0 page

#h(123)

M

Faloutsos - Pavlo CMU SCS 15-415/615 25

CMU SCS **Review**

Collision resolution

re-hashing

123; Smith; Main str.

#0 page

#h(123)

M

Faloutsos - Pavlo CMU SCS 15-415/615 26

CMU SCS **Review**

Collision resolution

separate chaining

123; Smith; Main str.

#0 page

#h(123)

M

Faloutsos - Pavlo CMU SCS 15-415/615 27

CMU SCS **Review**

Design decisions - conclusions

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

Faloutsos - Pavlo CMU SCS 15-415/615 28

CMU SCS

Outline

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

Faloutsos - Pavlo CMU SCS 15-415/615 29

CMU SCS

Problem with static hashing

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

Faloutsos - Pavlo CMU SCS 15-415/615 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 - Pavlo CMU SCS 15-415/615 31

CMU SCS

Extendible hashing

Faloutsos - Pavlo CMU SCS 15-415/615 32

CMU SCS

Extendible hashing

solution: don't overflow – instead:
SPLIT the bucket in two

123; Smith; Main str.

#0 page
#h(123)
M

Faloutsos - Pavlo CMU SCS 15-415/615 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 - Pavlo CMU SCS 15-415/615 34

CMU SCS

Extendible hashing

directory

00...		0001...
01...		0111...
10...		10101...
11...		10011...
		10110...
		1101...

101001...

Faloutsos - Pavlo CMU SCS 15-415/615 35

CMU SCS

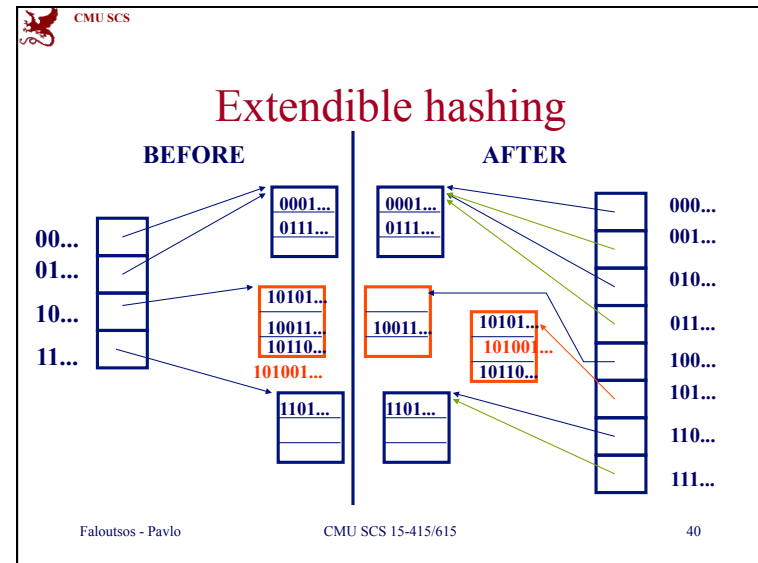
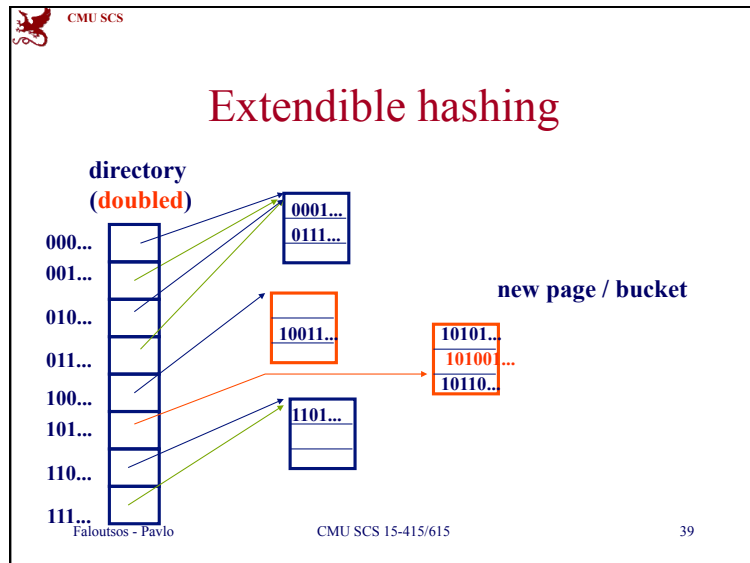
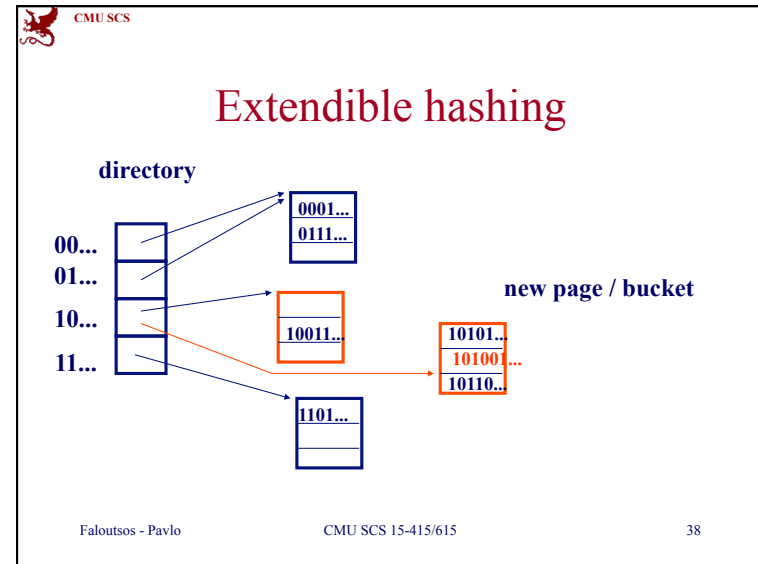
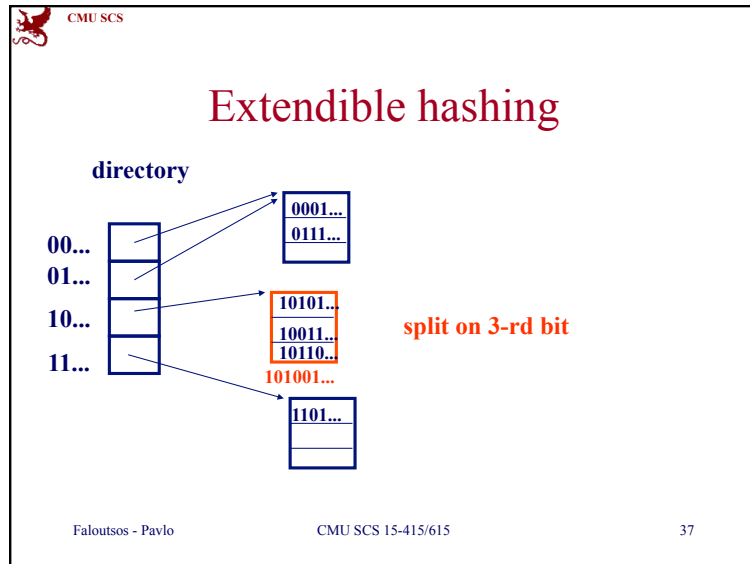
Extendible hashing

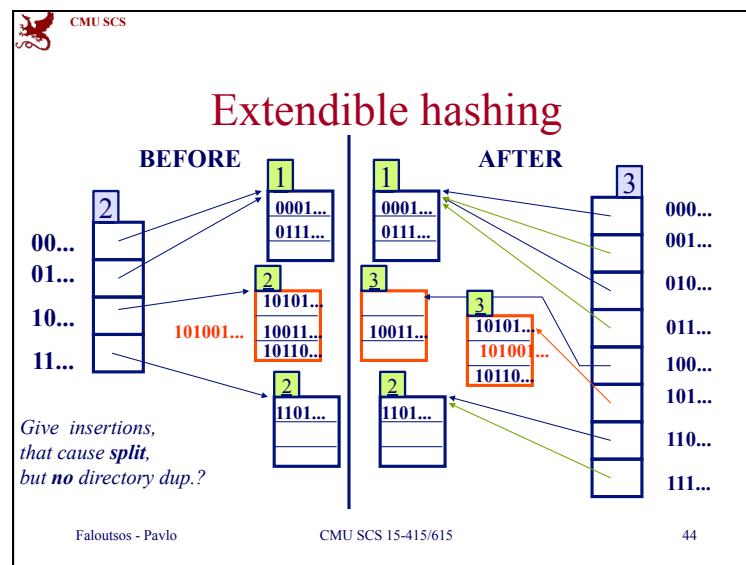
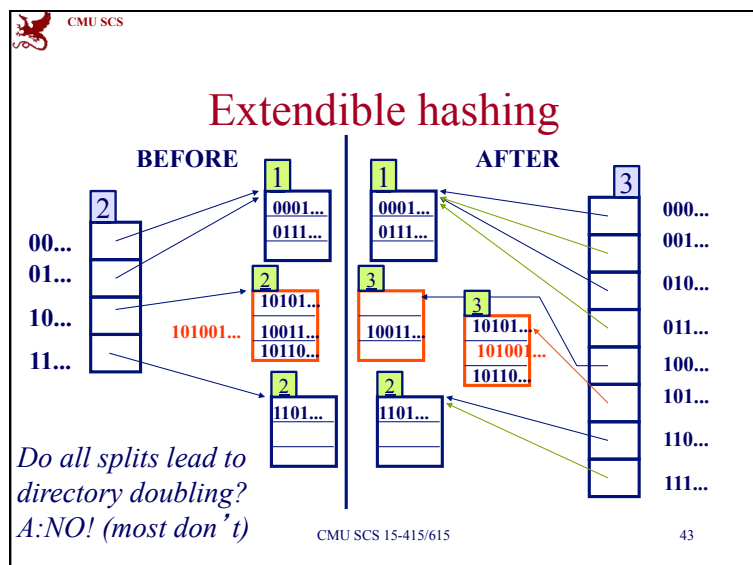
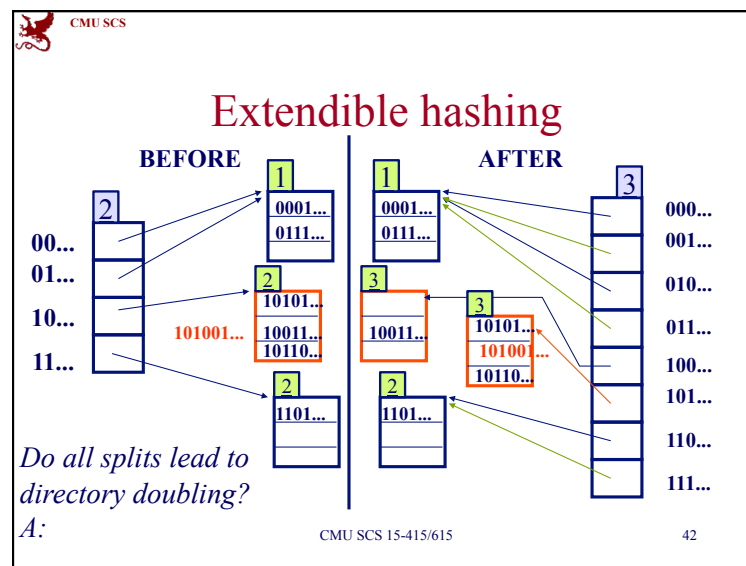
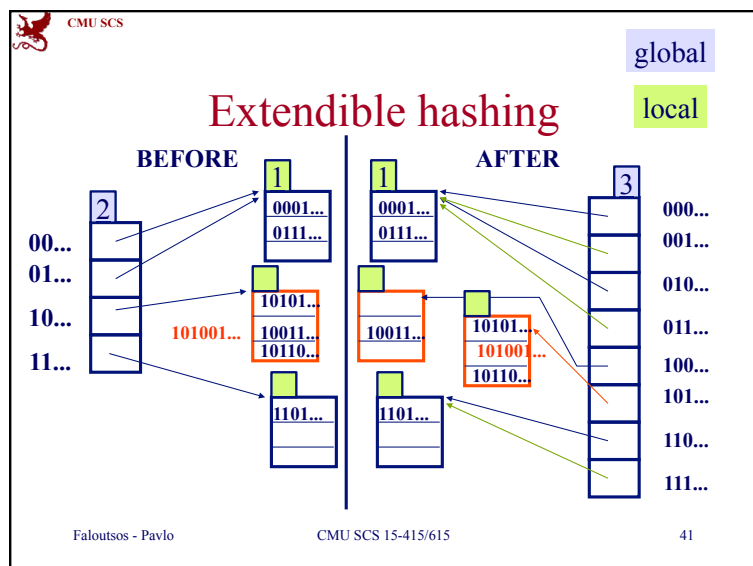
directory

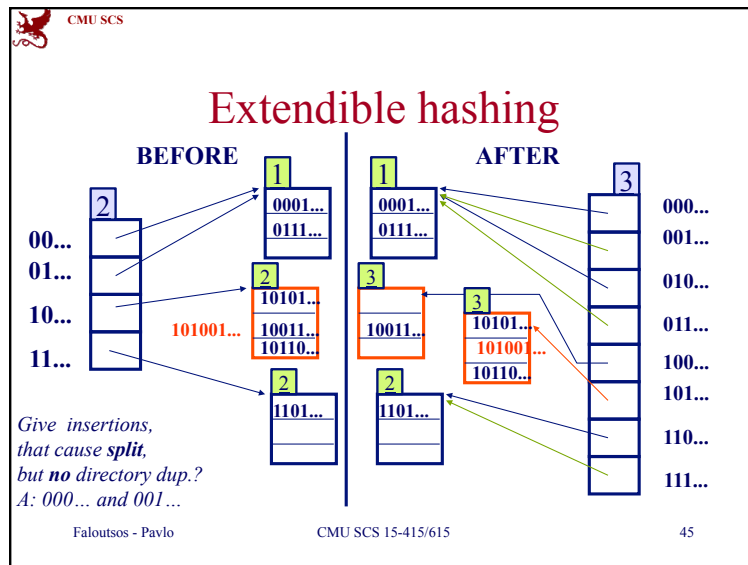
00...		0001...
01...		0111...
10...		10101...
11...		10011...
		10110...
		1101...

101001...

Faloutsos - Pavlo CMU SCS 15-415/615 36







(Note)

- The **textbook** uses the signature bits from **right to left**
- For HW4, use the **textbook's** approach.

Faloutsos - Pavlo CMU SCS 15-415/615 46

Extensible hashing

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

Faloutsos - Pavlo CMU SCS 15-415/615 47

Outline

- (static) hashing
- extensible hashing
- ➔ linear hashing
- Hashing vs B-trees

Faloutsos - Pavlo CMU SCS 15-415/615 48

CMU SCS

Linear hashing - overview

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

Faloutsos - Pavlo CMU SCS 15-415/615 49

CMU SCS

Linear hashing

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

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

Faloutsos - Pavlo CMU SCS 15-415/615 50

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 - Pavlo CMU SCS 15-415/615 51

CMU SCS

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	6	7	11
		13				

Faloutsos - Pavlo CMU SCS 15-415/615 52

CMU SCS

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	6	7	11
		13				

↓

Faloutsos - Pavlo CMU SCS 15-415/615 53

CMU SCS

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	6	7	11
		13				

↓

Faloutsos - Pavlo CMU SCS 15-415/615 54

CMU SCS

Linear hashing

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

Split #0, anyway!!!

Q: But, how?

17

bucket- id 0 1 2 3

4	8	5	9	6	7	11
		13				

↓

Faloutsos - Pavlo CMU SCS 15-415/615 55

CMU SCS

Linear hashing

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

$h1(x) = x \bmod (2*N)$

17

bucket- id 0 1 2 3

4	8	5	9	6	7	11
		13				

↓

Faloutsos - Pavlo CMU SCS 15-415/615 56

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 - Pavlo CMU SCS 15-415/615 57

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 - Pavlo CMU SCS 15-415/615 58

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 - Pavlo CMU SCS 15-415/615 59

CMU SCS

Linear hashing - overview

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

Faloutsos - Pavlo CMU SCS 15-415/615 60

CMU SCS

Linear hashing - searching?

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

Faloutsos - Pavlo CMU SCS 15-415/615 61

CMU SCS

Linear hashing - searching?

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

Q3: key '8'?

Faloutsos - Pavlo CMU SCS 15-415/615 62

CMU SCS

Linear hashing - searching?

Algo to find key 'k':

```

compute b = h0(k);    // original slot
if b < split_ptr     // has been split
    compute b = h1(k)
search bucket b
    
```


Faloutsos - Pavlo CMU SCS 15-415/615 63

CMU SCS

Linear hashing - overview

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

Faloutsos - Pavlo CMU SCS 15-415/615 64




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 - Pavlo CMU SCS 15-415/615 65




CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

Faloutsos - Pavlo CMU SCS 15-415/615 66



CMU SCS


Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

A1: space utilization \geq u-max

Faloutsos - Pavlo CMU SCS 15-415/615 67



CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

A1: space utilization \geq u-max

A2: avg length of ovf chains $>$ max-len

A3:

Faloutsos - Pavlo CMU SCS 15-415/615 68

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 - Pavlo CMU SCS 15-415/615 69

CMU SCS

Linear hashing - split now?

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

split ptr

Faloutsos - Pavlo CMU SCS 15-415/615 70

CMU SCS

Linear hashing - split now?

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

split ptr

Faloutsos - Pavlo CMU SCS 15-415/615 71

CMU SCS

Linear hashing - split now?

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


split ptr

Faloutsos - Pavlo CMU SCS 15-415/615 72

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 - Pavlo CMU SCS 15-415/615 73

CMU SCS

Linear hashing - split now?

this state is called 'full expansion'

split ptr


0	1	2	3	4	5	6	7

Faloutsos - Pavlo CMU SCS 15-415/615 74

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 - Pavlo CMU SCS 15-415/615 75

CMU SCS

Linear hashing - overview

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

Faloutsos - Pavlo CMU SCS 15-415/615 76

CMU SCS

Linear hashing - deletion?

- reverse of insertion:

Faloutsos - Pavlo CMU SCS 15-415/615 77

CMU SCS

Linear hashing - deletion?

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

Faloutsos - Pavlo CMU SCS 15-415/615 78

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 - Pavlo CMU SCS 15-415/615 79

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 - Pavlo CMU SCS 15-415/615 80

CMU SCS

Linear hashing - how to contract?

$h_0(x) = \text{mod } N$ (for the un-split buckets)
 $h_1(x) = \text{mod } (2*N)$ (for the splitted ones)

split ptr

Faloutsos - Pavlo CMU SCS 15-415/615 81

CMU SCS

Linear hashing - how to contract?

$h_0(x) = \text{mod } N$ (for the un-split buckets)
 $h_1(x) = \text{mod } (2*N)$ (for the splitted ones)

split ptr

Faloutsos - Pavlo CMU SCS 15-415/615 82

CMU SCS

Linear hashing - how to contract?

$h_0(x) = \text{mod } N$ (for the un-split buckets)
 $h_1(x) = \text{mod } (2*N)$ (for the splitted ones)

split ptr


Faloutsos - Pavlo CMU SCS 15-415/615 83

CMU SCS

Outline

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


Faloutsos - Pavlo CMU SCS 15-415/615 84



CMU SCS

Hashing - pros?

Faloutsos - Pavlo CMU SCS 15-415/615 85




CMU SCS

Hashing - pros?

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


Faloutsos - Pavlo CMU SCS 15-415/615 86



CMU SCS

B(+)-trees - pros?

Faloutsos - Pavlo CMU SCS 15-415/615 87




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 - Pavlo CMU SCS 15-415/615 88




CMU SCS

B(+)-trees vs Hashing

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

Faloutsos - Pavlo CMU SCS 15-415/615 89



CMU SCS

Conclusions

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

Faloutsos - Pavlo CMU SCS 15-415/615 90