


15-826: Multimedia Databases and Data Mining


Lecture#3: Primary key indexing – hashing
C. Faloutsos



Reading Material

- [Litwin] Litwin, W., (1980), *Linear Hashing: A New Tool for File and Table Addressing*, VLDB, Montreal, Canada, 1980
- textbook, Chapter 3
- Ramakrinshan+Gehrke, Chapter 11

15-826 Copyright: C. Faloutsos (2009) 2



Outline

Goal: 'Find similar / interesting things'

- Intro to DB
- ➔ Indexing - similarity search
- Data Mining

15-826 Copyright: C. Faloutsos (2009) 3

CMU SCS

Indexing - Detailed outline

- primary key indexing
 - B-trees and variants
 - ➔ – (static) hashing
 - extendible hashing
- secondary key indexing
- spatial access methods
- text
- ...

15-826 Copyright: C. Faloutsos (2009) 4

CMU SCS

(Static) Hashing

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

15-826 Copyright: C. Faloutsos (2009) 5

CMU SCS

Hashing

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

15-826 Copyright: C. Faloutsos (2009) 6

CMU SCS

Hashing

Since space is NOT free:

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

15-826 Copyright: C. Faloutsos (2009) 7

CMU SCS

Hashing

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

15-826 Copyright: C. Faloutsos (2009) 8

CMU SCS

Hashing

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

15-826 Copyright: C. Faloutsos (2009) 9

CMU SCS

Hashing

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

15-826 Copyright: C. Faloutsos (2009) 10

CMU SCS

Hashing - design decisions?

- eg., IRS, 200M tax returns, by SSN

15-826 Copyright: C. Faloutsos (2009) 11

CMU SCS

Indexing- overview

- B-trees
- hashing
 - ➔ - hashing functions
 - size of hash table
 - collision resolution
- Hashing vs B-trees
- Indices in SQL

15-826 Copyright: C. Faloutsos (2009) 12

CMU SCS

Design decisions

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

15-826 Copyright: C. Faloutsos (2009) 13

CMU SCS

Design decisions - functions

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

15-826 Copyright: C. Faloutsos (2009) 14

CMU SCS

Division hashing

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

- 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?)

15-826 Copyright: C. Faloutsos (2009) 15

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!

15-826 Copyright: C. Faloutsos (2009) 16

CMU SCS

Multiplication hashing

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

- 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

15-826 Copyright: C. Faloutsos (2009) 17

CMU SCS

Other hashing functions

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

15-826 Copyright: C. Faloutsos (2009) 18

CMU SCS

Design decisions

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

15-826 Copyright: C. Faloutsos (2009) 19

CMU SCS

Size of hash table

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

15-826 Copyright: C. Faloutsos (2009) 20

CMU SCS

Size of hash table

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

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

15-826 Copyright: C. Faloutsos (2009) 21

CMU SCS

Design decisions

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

15-826 Copyright: C. Faloutsos (2009) 22

CMU SCS

Collision resolution

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

15-826 Copyright: C. Faloutsos (2009) 23

CMU SCS

Collision resolution

123; Smith; Main str.

15-826 Copyright: C. Faloutsos (2009) 24

CMU SCS

Collision resolution

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

15-826 Copyright: C. Faloutsos (2009) 25

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'

15-826 Copyright: C. Faloutsos (2009) 26

CMU SCS

Collision resolution

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

15-826 Copyright: C. Faloutsos (2009) 27

CMU SCS

Collision resolution

linear probing:

#0 page
#h(123)
 M

15-826 Copyright: C. Faloutsos (2009) 28

CMU SCS

Collision resolution

re-hashing

#0 page
#h(123)
 M

15-826 Copyright: C. Faloutsos (2009) 29

CMU SCS

Collision resolution

separate chaining

#0 page
#h(123)
 M

15-826 Copyright: C. Faloutsos (2009) 30

CMU SCS

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

15-826 Copyright: C. Faloutsos (2009) 31

CMU SCS

Indexing- overview

- B-trees
- hashing
 - ➔ – Hashing vs B-trees
- Indices in SQL
- extendible hashing

15-826 Copyright: C. Faloutsos (2009) 32

CMU SCS

Hashing vs B-trees:

Hashing offers

- speed ! ($O(1)$ avg. search time)

..but:

15-826 Copyright: C. Faloutsos (2009) 33

CMU SCS

Hashing vs B-trees:

..but B-trees give:

- key ordering:
 - range queries
 - proximity queries
 - sequential scan
- $O(\log(N))$ guarantees for search, ins./del.
- graceful growing/shrinking

15-826 Copyright: C. Faloutsos (2009) 34

CMU SCS

Hashing vs B-trees:

thus:

- B-trees are implemented in most systems

footnotes:

- hashing is rarely implemented (why not?)
- 'dbm' and 'ndbm' of UNIX: offer one or both

15-826 Copyright: C. Faloutsos (2009) 35

CMU SCS

Indexing- overview

- B-trees
- hashing
 - Hashing vs B-trees
- ➔ • Indices in SQL
- extendible hashing

15-826 Copyright: C. Faloutsos (2009) 36

CMU SCS

Indexing in SQL

- create index **<index-name>** on **<relation-name>** (**<attribute-list>**)
- create unique index **<index-name>** on **<relation-name>** (**<attribute-list>**)
- drop index **<index-name>**

15-826 Copyright: C. Faloutsos (2009) 37

CMU SCS

Indexing in SQL

- eg.,
create index ssn-index
on STUDENT (ssn)
- or (eg., on *TAKES(ssn,cid, grade)*):
create index sc-index
on TAKES (ssn, c-id)

15-826 Copyright: C. Faloutsos (2009) 38

CMU SCS

Indexing- overview

- B-trees
- hashing
- Indices in SQL
- extensible hashing
- ➔ – ‘extendible’ hashing [Fagin, Pipenger +]
- ‘linear’ hashing [Litwin]

15-826 Copyright: C. Faloutsos (2009) 39

CMU SCS

Problem with static hashing

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

15-826 Copyright: C. Faloutsos (2009) 40

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]

15-826 Copyright: C. Faloutsos (2009) 41

CMU SCS

Extendible hashing

123; Smith; Main str.

→

#0 page

↕

FULL

↕

#h(123)

↕

↕

M

15-826 Copyright: C. Faloutsos (2009) 42

CMU SCS

Extendible hashing

solution:
split the bucket in two

123; Smith; Main str.

→

FULL

#0 page

#h(123)

M

15-826 Copyright: C. Faloutsos (2009) 43

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:

15-826 Copyright: C. Faloutsos (2009) 44

CMU SCS

Extendible hashing

directory

00...

01...

10...

11...

0001...

0111...

10101...

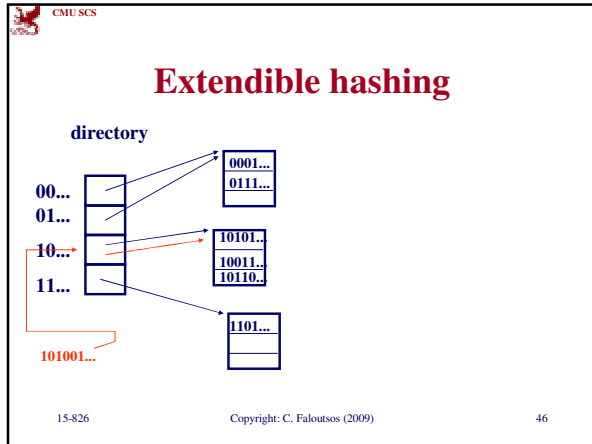
10011...

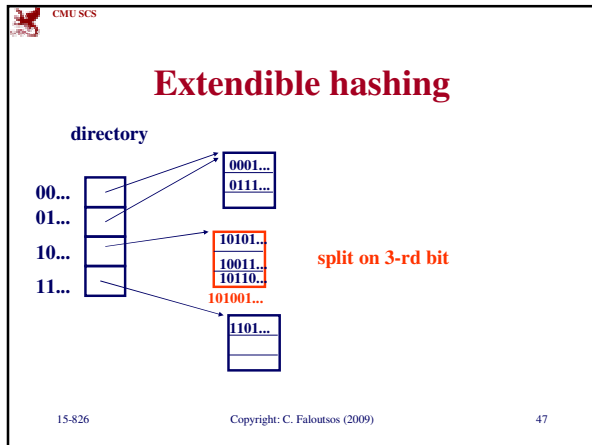
10110...

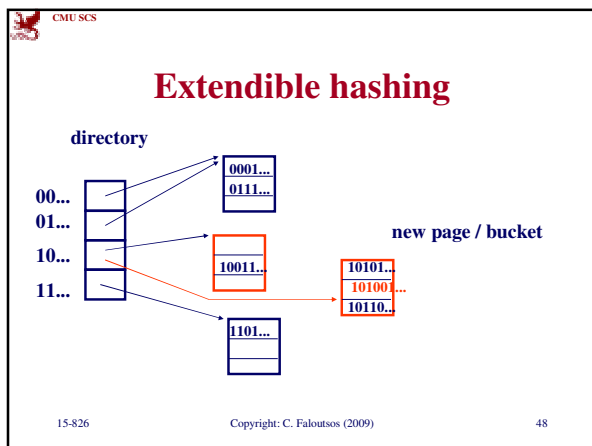
1101...

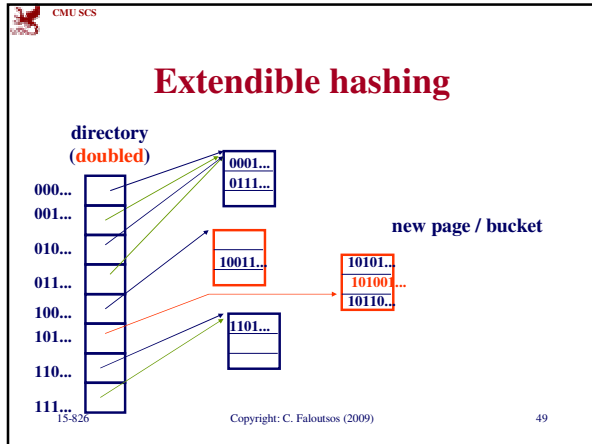
101001...

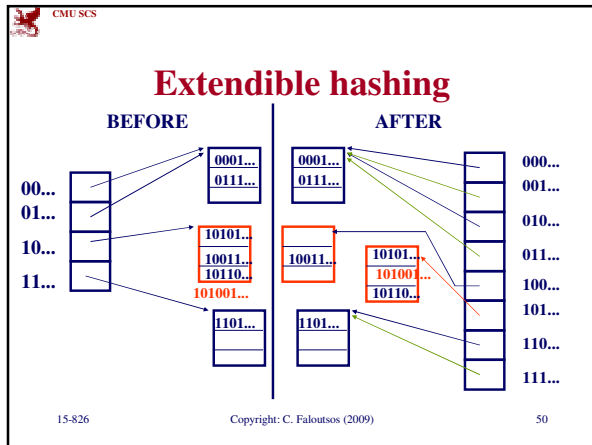
15-826 Copyright: C. Faloutsos (2009) 45











Extendible hashing

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

15-826 Copyright: C. Faloutsos (2009) 51

CMU SCS

Indexing- overview

- B-trees
- hashing
- Hashing vs B-trees
- Indices in SQL
- extendible hashing
 - ‘extensible’ hashing [Fagin, Pipenger +]
 - ➔ – ‘linear’ hashing [Litwin]

15-826 Copyright: C. Faloutsos (2009) 52

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion
- performance analysis
- variations

15-826 Copyright: C. Faloutsos (2009) 53

CMU SCS

Linear hashing

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

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

15-826 Copyright: C. Faloutsos (2009) 54

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.:

15-826 Copyright: C. Faloutsos (2009) 55

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				

15-826 Copyright: C. Faloutsos (2009) 56

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				

15-826 Copyright: C. Faloutsos (2009) 57

CMU SCS

Linear hashing

Initially: $h(x) = x \bmod N$ (N=4 here)
 overflow of bucket#1

17
↓

bucket- id 0 1 2 3

4	8	5	9	6	7	11
		13				

Split #0, anyway!!!

15-826 Copyright: C. Faloutsos (2009) 58

CMU SCS

Linear hashing

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

17
↓

bucket- id 0 1 2 3

4	8	5	9	6	7	11
		13				

Q: But, how?

15-826 Copyright: C. Faloutsos (2009) 59

CMU SCS

Linear hashing

A: use two h.f.: $h_0(x) = x \bmod N$
 $h_1(x) = x \bmod (2*N)$

17
↓

bucket- id 0 1 2 3

4	8	5	9	6	7	11
		13				

15-826 Copyright: C. Faloutsos (2009) 60

CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \text{ mod } N$
 $h1(x) = x \text{ mod } (2*N)$

bucket- id 0 1 2 3 4

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

17

15-826 Copyright: C. Faloutsos (2009) 61

CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \text{ mod } N$
 $h1(x) = x \text{ mod } (2*N)$

bucket- id 0 1 2 3 4

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

↓
17 overflow

15-826 Copyright: C. Faloutsos (2009) 62

CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \text{ mod } N$
 $h1(x) = x \text{ mod } (2*N)$

↓ split ptr

bucket- id 0 1 2 3 4

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

↓
17 overflow

15-826 Copyright: C. Faloutsos (2009) 63

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- ➔ • search algo
- insertion/split algo
- deletion
- performance analysis
- variations

15-826 Copyright: C. Faloutsos (2009) 64

CMU SCS

Linear hashing - searching?

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

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

↓ split ptr

17	overflow
----	----------

15-826 Copyright: C. Faloutsos (2009) 65

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 13	6	7 11	4

↓ split ptr

17	overflow
----	----------

15-826 Copyright: C. Faloutsos (2009) 66

CMU SCS

Linear hashing - searching?

Algo to find key 'k':

- compute $b = h_0(k)$;
 - if $b < \text{split-ptr}$, compute $b = h_1(k)$
- search bucket b

15-826 Copyright: C. Faloutsos (2009) 67

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- ➔ insertion/split algo
- deletion
- performance analysis
- variations

15-826 Copyright: C. Faloutsos (2009) 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 ++ (*)


15-826 Copyright: C. Faloutsos (2009) 69

 CMU SCS

Linear hashing - insertion?

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


15-826 Copyright: C. Faloutsos (2009) 70

 CMU SCS

Linear hashing - insertion?

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

15-826 Copyright: C. Faloutsos (2009) 71

 CMU SCS

Linear hashing - insertion?

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

15-826 Copyright: C. Faloutsos (2009) 72

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??

15-826 Copyright: C. Faloutsos (2009) 73

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

0 1 2 3 4 5 6

15-826 Copyright: C. Faloutsos (2009) 74

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

0 1 2 3 4 5 6 7

15-826 Copyright: C. Faloutsos (2009) 75

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

0 1 2 3 4 5 6 7

--	--	--	--	--	--	--	--

15-826 Copyright: C. Faloutsos (2009) 76

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

0 1 2 3 4 5 6 7

--	--	--	--	--	--	--	--

15-826 Copyright: C. Faloutsos (2009) 77

CMU SCS

Linear hashing - split now?

this state is called 'full expansion'

split ptr

0 1 2 3 4 5 6 7

--	--	--	--	--	--	--	--

15-826 Copyright: C. Faloutsos (2009) 78

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.)

15-826 Copyright: C. Faloutsos (2009) 79

CMU SCS

Linear hashing - overview

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

15-826 Copyright: C. Faloutsos (2009) 80

CMU SCS

Linear hashing - deletion?

- reverse of insertion:

15-826 Copyright: C. Faloutsos (2009) 81

CMU SCS

Linear hashing - deletion?

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

15-826 Copyright: C. Faloutsos (2009) 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
↓

0 1 2 3 4 5 6

15-826 Copyright: C. Faloutsos (2009) 83

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
↓

0 1 2 3 4 5

15-826 Copyright: C. Faloutsos (2009) 84

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion
- ➔ performance analysis
- variations

15-826 Copyright: C. Faloutsos (2009) 85

CMU SCS

Linear hashing - performance

- [Larson, TODS 1982]

search-time (avg # of d.a.)

split: if $u > u_0$
(say $u_0 = .85$)

1.01 d.a.

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 86

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

search-time (avg # of d.a.)

split: if $u > u_0$
(say $u_0 = .85$)

1.01 d.a.

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 87

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

search-time (avg # of d.a.)

split: if $u > u_0$
(say $u_0 = .85$)

1.01 d.a.

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 88

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

search-time (avg # of d.a.)

split: if $u > u_0$
(say $u_0 = .85$)

1.01 d.a.

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 89

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

search-time (avg # of d.a.)

split: if $u > u_0$
(say $u_0 = .85$)

1.01 d.a.

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 90

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

search-time (avg # of d.a.)

eg., 1.3 d.a.

eg., 1.01 d.a.

split: if $u > u_0$
(say $u_0 = .85$)

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 91

CMU SCS

Linear hashing - performance

- Q: How to shorten the maximum?

search-time

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 92

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion
- performance analysis
- ➡ variations

15-826 Copyright: C. Faloutsos (2009) 93

CMU SCS

Linear hashing - performance

- Q: How to shorten the maximum?
- A: 2-3 splits - partial expansions!

15-826 Copyright: C. Faloutsos (2009) 94

CMU SCS

Linear hashing - performance

- Q: How to shorten the maximum?
- A: 2-3 splits - partial expansions!

15-826 Copyright: C. Faloutsos (2009) 95

CMU SCS

Linear hashing - variations

Two split pointers! On split:

15-826 Copyright: C. Faloutsos (2009) 96

CMU SCS

Linear hashing - variations

Two split pointers! On split:

The diagram shows a horizontal array of five slots labeled 0, 1, 2, 3, and 4. Above the first three slots (0, 1, 2) are three red arrows pointing down. Above the fourth slot (3) is a red arrow pointing down. The fourth slot (3) is highlighted with a red border. A blue curved arrow starts from the bottom of the first slot (0) and points to the bottom of the fourth slot (3). Another blue curved arrow starts from the bottom of the second slot (1) and points to the bottom of the fourth slot (3).

15-826 Copyright: C. Faloutsos (2009) 97

CMU SCS

Linear hashing - variations

2nd split:

The diagram shows a horizontal array of six slots labeled 0, 1, 2, 3, 4, and 5. Above the second slot (1) and the fourth slot (3) are two red arrows pointing down. The fifth slot (5) is highlighted with a red border. A blue curved arrow starts from the bottom of the second slot (1) and points to the bottom of the fifth slot (5). Another blue curved arrow starts from the bottom of the third slot (2) and points to the bottom of the fifth slot (5).

15-826 Copyright: C. Faloutsos (2009) 98

CMU SCS

Linear hashing - variations

2nd split: Partial expansion! (50% larger table)

The diagram shows a horizontal array of six slots labeled 0, 1, 2, 3, 4, and 5. Above the first slot (0) and the third slot (2) are two red arrows pointing down. The fifth slot (5) is highlighted with a red border. A blue curved arrow starts from the bottom of the first slot (0) and points to the bottom of the fifth slot (5). Another blue curved arrow starts from the bottom of the second slot (1) and points to the bottom of the fifth slot (5).

15-826 Copyright: C. Faloutsos (2009) 99

CMU SCS

Linear hashing - variations

Q: how to do the third split?

0 1 2 3 4 5

15-826 Copyright: C. Faloutsos (2009) 100

CMU SCS

Linear hashing - variations

Q: how to do the third split?

A: 3-to-4 splits now!

0 1 2 3 4 5

15-826 Copyright: C. Faloutsos (2009) 101

CMU SCS

Linear hashing - performance

- Q1: Which of the two red peaks is higher?
- Q2: Why?

search-time

R 2R # records

15-826 Copyright: C. Faloutsos (2009) 102

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion
- performance analysis
- ➔ • variations

15-826 Copyright: C. Faloutsos (2009) 103

CMU SCS

Other hashing variations

- ‘order preserving’
- ‘perfect hashing’ (no collisions!) [Ed. Fox, et al]

15-826 Copyright: C. Faloutsos (2009) 104

CMU SCS

Primary key indexing - conclusions

- hashing is $O(1)$ on the average for search
- linear hashing: elegant way to grow a hash table
- B-trees: major contenders for primary-key indexing ($O(\log(N))$ w.c.!)

15-826 Copyright: C. Faloutsos (2009) 105

CMU SCS

References for primary key indexing

- [Fagin+] Ronald Fagin, Jürg Nievergelt, Nicholas Pippenger, H. Raymond Strong: Extendible Hashing - A Fast Access Method for Dynamic Files. TODS 4(3): 315-344(1979)
- [Fox] Fox, E. A., L. S. Heath, Q.-F. Chen, and A. M. Daoud. "Practical Minimal Perfect Hash Functions for Large Databases." Communications of the ACM 35.1 (1992): 105-21.

15-826 Copyright: C. Faloutsos (2009) 106

CMU SCS

References, cont'd

- [Knuth] D.E. Knuth. The Art Of Computer Programming, Vol. 3, Sorting and Searching, Addison Wesley
- [Larson] Per-Ake Larson Performance Analysis of Linear Hashing with Partial Expansions ACM TODS, 7,4, Dec. 1982, pp 566--587
- ➔ [Litwin] Litwin, W., (1980), Linear Hashing: A New Tool for File and Table Addressing, VLDB, Montreal, Canada, 1980

15-826 Copyright: C. Faloutsos (2009) 107
