


CMU SCS

15-826: Multimedia Databases and Data Mining

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



CMU SCS

Reading Material

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

15-826 Copyright: C. Faloutsos (2011) 2



CMU SCS

Outline

Goal: ‘Find similar / interesting things’

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

15-826 Copyright: C. Faloutsos (2011) 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 (2011) 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 (2011) 5

CMU SCS

Hashing

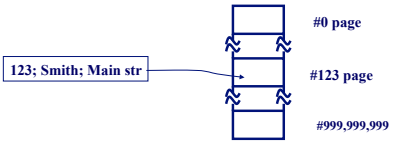
A: Brilliant idea: key-to-address transformation:

15-826 Copyright: C. Faloutsos (2011) 6

Hashing

Since space is NOT free:

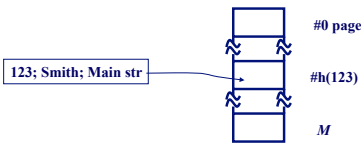
- use M , instead of 999,999,999 slots
- hash function: $h(\text{key}) = \text{slot-id}$



15-826 Copyright: C. Faloutsos (2011) 7

Hashing

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




15-826 Copyright: C. Faloutsos (2011) 8

Hashing - design decisions?

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

15-826 Copyright: C. Faloutsos (2011) 11




CMU SCS

Indexing- overview

- B-trees
- (static) hashing
 - ➔ – hashing functions
 - size of hash table
 - collision resolution
 - Hashing vs B-trees
 - Indices in SQL
- Extendible hashing

15-826 Copyright: C. Faloutsos (2011) 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 (2011) 13



CMU SCS

Design decisions

1) formula $h()$ for hashing function	Division hashing
2) size of hash table M	90% utilization
3) collision resolution method	Separate chaining

15-826 Copyright: C. Faloutsos (2011) 14


 CMU SCS

SKIP

Design decisions - functions

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

15-826 Copyright: C. Faloutsos (2011) 15

 CMU SCS


SKIP

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 (2011) 16


 CMU SCS

SKIP

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 (2011) 17



CMU SCS

SKIP

Design decisions

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

15-826
Copyright: C. Faloutsos (2011)
18



CMU SCS

SKIP

Size of hash table

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

15-826
Copyright: C. Faloutsos (2011)
19


CMU SCS

SKIP

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: $M = \text{closest prime to } 50,000/10 / 0.9 = 5,555$

15-826
Copyright: C. Faloutsos (2011)
20

CMU SCS

SKIP

Design decisions

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

15-826 Copyright: C. Faloutsos (2011) 21

CMU SCS

SKIP

Collision resolution

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


15-826 Copyright: C. Faloutsos (2011) 22

CMU SCS

SKIP

Collision resolution

15-826 Copyright: C. Faloutsos (2011) 23


 CMU SCS

SKIP

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 (2011) 24


 CMU SCS

SKIP

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 (2011) 25


 CMU SCS

SKIP

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 (2011) 26

CMU SCS

Collision resolution

linear probing:

123; Smith; Main str.

#0 page

#h(123)


M

FULL

15-826

Copyright: C. Faloutsos (2011)

27

CMU SCS

Collision resolution

re-hashing

123; Smith; Main str.

#0 page

#h(123)

M

FULL


h1()

h2()

15-826

Copyright: C. Faloutsos (2011)

28

CMU SCS

Collision resolution

separate chaining


123; Smith; Main str.

FULL

15-826

Copyright: C. Faloutsos (2011)

29




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

15-826 Copyright: C. Faloutsos (2011) 30




CMU SCS

Indexing- overview

- B-trees
- (static) hashing
 - hashing functions
 - size of hash table
 - collision resolution
- ➔ • Hashing vs B-trees
 - Indices in SQL
- Extendible hashing

15-826 Copyright: C. Faloutsos (2011) 31



CMU SCS


Hashing vs B-trees:

Hashing offers

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

..but:

15-826 Copyright: C. Faloutsos (2011) 32


 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 (2011) 33

 CMU SCS

Hashing vs B-trees:


thus:

- B-trees are implemented in most systems

footnotes:

- ‘dbm’ and ‘ndbm’ of UNIX: offer one or both


15-826 Copyright: C. Faloutsos (2011) 34

 CMU SCS

Indexing- overview

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

15-826 Copyright: C. Faloutsos (2011) 35




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 (2011) 36



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 (2011) 37




CMU SCS

Indexing- overview

- B-trees
- (static) Hashing
- extensible hashing

➡ – ‘linear’ hashing [Litwin]

15-826 Copyright: C. Faloutsos (2011) 38




CMU SCS

Problem with static hashing

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

15-826 Copyright: C. Faloutsos (2011) 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 – simplest: Linear hashing [Litwin]

15-826 Copyright: C. Faloutsos (2011) 41




CMU SCS

Indexing- overview

- B-trees
- Static hashing
- extendible hashing
 - ‘extensible’ hashing [Fagin, Pipenger +]
 - ➡ – ‘linear’ hashing [Litwin]

15-826 Copyright: C. Faloutsos (2011) 42




CMU SCS

Linear hashing - Detailed overview

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

15-826 Copyright: C. Faloutsos (2011) 43




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 (2011) 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.:

15-826 Copyright: C. Faloutsos (2011) 45

 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 13	6	7 11
---	---	-----------	---	------

15-826

Copyright: C. Faloutsos (2011)

46

 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 13	6	7 11
---	---	-----------	---	------

15-826

Copyright: C. Faloutsos (2011)

47

 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 13	6	7 11
---	---	-----------	---	------

15-826

Copyright: C. Faloutsos (2011)

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

15-826

Copyright: C. Faloutsos (2011)

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

15-826

Copyright: C. Faloutsos (2011)

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

15-826

Copyright: C. Faloutsos (2011)

51

Linear hashing - after split:

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

bucket- id 0 1 2 3 4

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

↓

17

overflow

15-826 Copyright: C. Faloutsos (2011) 52

Linear hashing - after split:

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

split ptr

bucket- id 0 1 2 3 4

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

↓

17

overflow

15-826 Copyright: C. Faloutsos (2011) 53

Linear hashing - overview

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

15-826 Copyright: C. Faloutsos (2011) 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

15-826 Copyright: C. Faloutsos (2011) 55

CMU SCS

Linear hashing - searching?

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

bucket-id 0 1 2 3 4

15-826 Copyright: C. Faloutsos (2011) 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

15-826 Copyright: C. Faloutsos (2011) 57




CMU SCS

Linear hashing - overview

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

15-826 Copyright: C. Faloutsos (2011) 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 ++ (*)

15-826 Copyright: C. Faloutsos (2011) 59




CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

15-826 Copyright: C. Faloutsos (2011) 60



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 (2011) 61



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 (2011) 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??

15-826 Copyright: C. Faloutsos (2011) 63

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

15-826

Copyright: C. Faloutsos (2011)

64

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

15-826

Copyright: C. Faloutsos (2011)

65

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

15-826


Copyright: C. Faloutsos (2011)

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

15-826 Copyright: C. Faloutsos (2011) 67

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 (2011) 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.)

15-826 Copyright: C. Faloutsos (2011) 69




CMU SCS

Linear hashing - overview

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

15-826 Copyright: C. Faloutsos (2011) 70




CMU SCS

Linear hashing - deletion?

- reverse of insertion:

15-826 Copyright: C. Faloutsos (2011) 71



CMU SCS

Linear hashing - deletion?

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

15-826 Copyright: C. Faloutsos (2011) 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

0 1 2 3 4 5 6

15-826 Copyright: C. Faloutsos (2011) 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

0 1 2 3 4 5

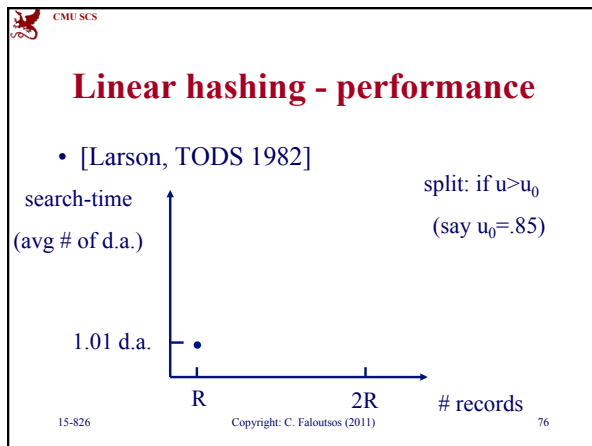
15-826 Copyright: C. Faloutsos (2011) 74

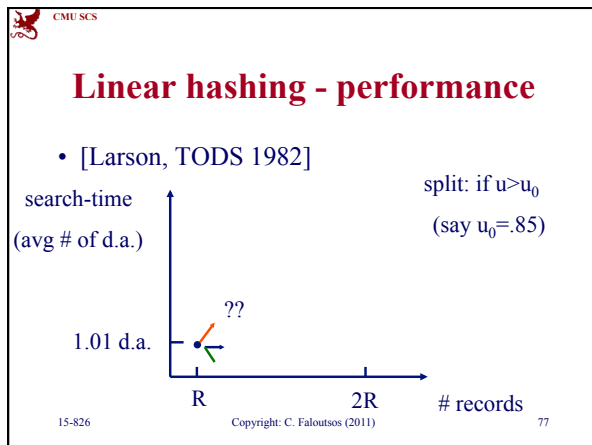
CMU SCS

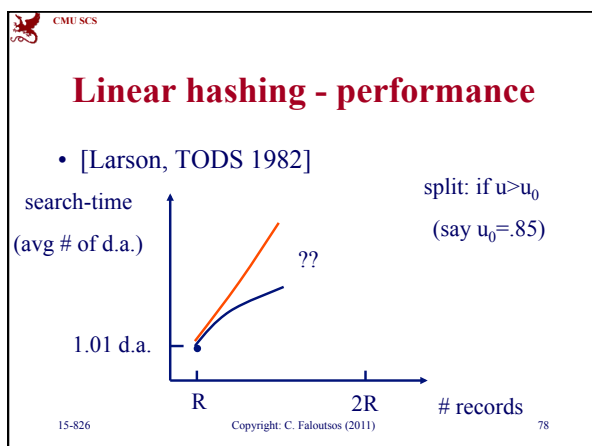
Linear hashing - overview

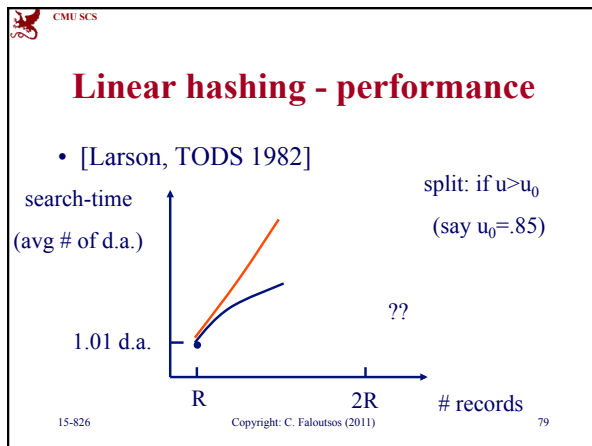
- Motivation
- main idea
- search algo
- insertion/split algo
- deletion
- ➡ • performance analysis
- variations

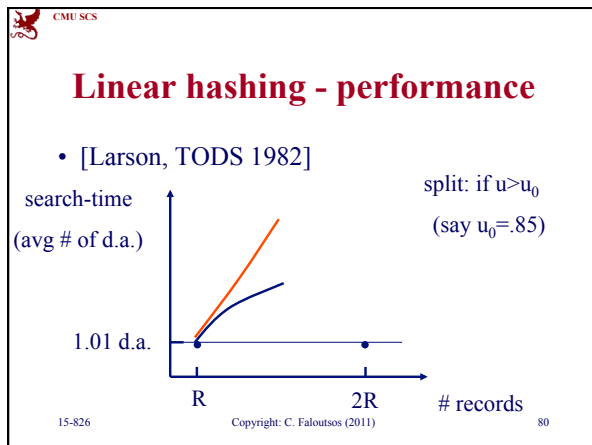
15-826 Copyright: C. Faloutsos (2011) 75

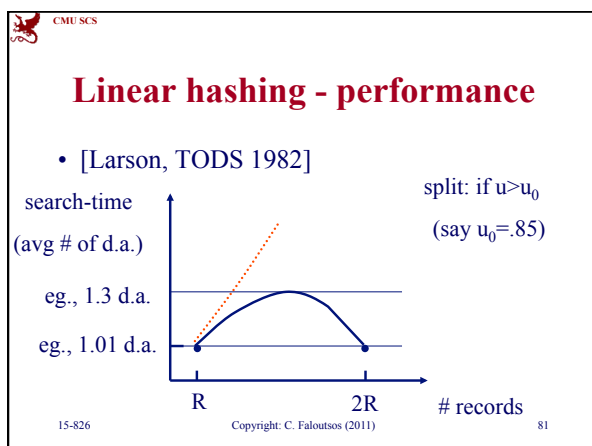












CMU SCS

Linear hashing - performance

- Q: How to shorten the maximum?

search-time

R 2R # records

15-826 Copyright: C. Faloutsos (2011) 82

CMU SCS

Linear hashing - overview

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


15-826 Copyright: C. Faloutsos (2011) 93

CMU SCS

Other hashing variations

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


15-826 Copyright: C. Faloutsos (2011) 94



Primary key indexing - conclusions

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


15-826 Copyright: C. Faloutsos (2011) 95



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 (2011) 96



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 (2011) 97
