CMU SCS

15-826: Multimedia Databases and Data Mining

Primary key indexing – hashing

C. Faloutsos

CMU SCS

Outline

Goal: ‘Find similar / interesting things’

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

CMU SCS

Indexing - Detailed outline

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

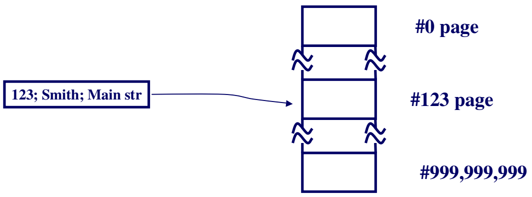
(Static) Hashing

Problem: "*find EMP record with ssn=123*"

What if disk space was free, and time was at premium?

Hashing

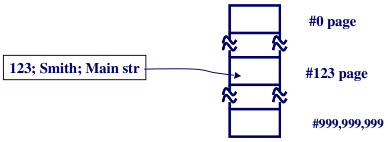
A: Brilliant idea: key-to-address transformation:



Hashing

Since space is NOT free:

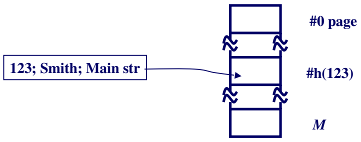
- use *M,* instead of 999,999,999 slots
- hash function: *h(key) = slot-id*



CMU SCS

Hashing

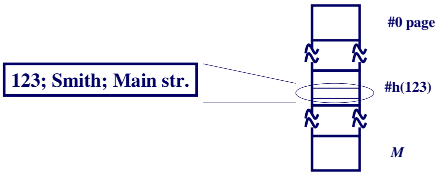
Typically: each hash bucket is a page, holding many records:



CMU SCS

Hashing

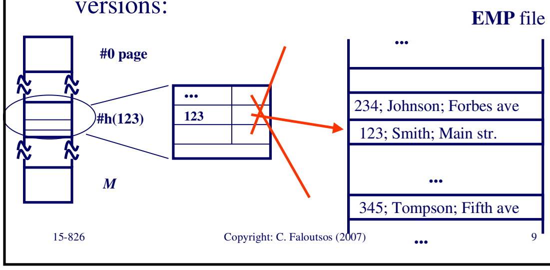
Notice: could have **clustering**, or non-clustering versions:



CMU SCS

Hashing

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



CMU SCS

Hashing - design decisions?

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

CMU SCS

Indexing- overview

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

CMU SCS

Design decisions

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

CMU SCS

Design decisions - functions

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

CMU SCS

Division hashing

*h(x) = (a*x+b) mod M*

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

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!

CMU SCS

Multiplication hashing

*h(x) = [fractional-part-of (x * φ)] * 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

CMU SCS

Other hashing functions

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

CMU SCS

Design decisions

1) formula *h()* for hashing function

➡ 2) size of hash table *M*

3) collision resolution method

CMU SCS

Size of hash table

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

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= closest prime to 50,000/10 / 0.9 = 5,555

CMU SCS

Design decisions

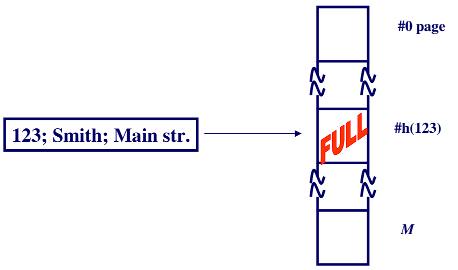
1) formula *h()* for hashing function
2) size of hash table M
3) collision resolution method

Collision resolution

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

CMU SCS

Collision resolution



CMU SCS

Collision resolution

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

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'

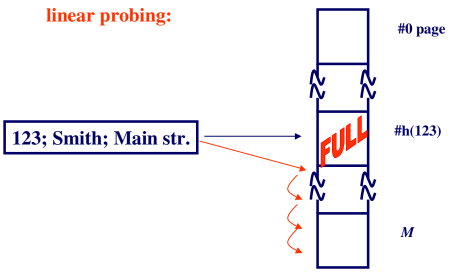
CMU SCS

Collision resolution

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

CMU SCS

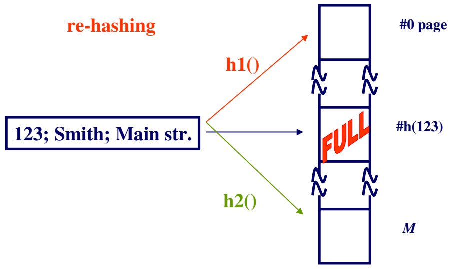
Collision resolution



CMU SCS

Collision resolution

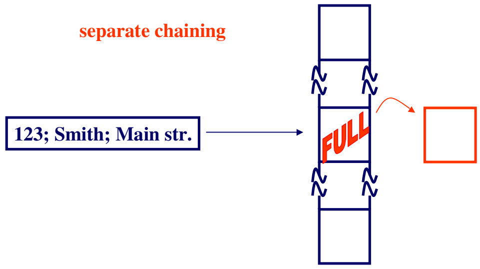
re-hashing



CMU SCS

Collision resolution

separate chaining



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

Indexing- overview

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

Hashing vs B-trees:

Hashing offers

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

..but:

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

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

CMU SCS

Indexing- overview

- B-trees
- hashing
  - Hashing vs B-trees
- ➡ Indices in SQL
- extendible hashing

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>**

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)

CMU SCS

Indexing- overview

- B-trees
- hashing
- Indices in SQL
- extensible hashing
  - 'extendible' hashing [Fagin, Pipenger +]
  - **'linear' hashing** [Litwin]

CMU SCS

Problem with static hashing

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

CMU SCS

Solution: Dynamic/extendible hashing

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

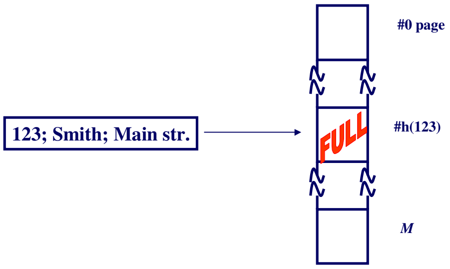
Details: how to grow gracefully, on overflow?

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

15-826 Copyright: C. Faloutsos (2007) 40

CMU SCS

Extendible hashing



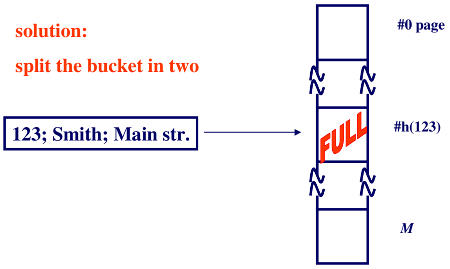
15-826 Copyright: C. Faloutsos (2007) 41

CMU SCS

Extendible hashing

solution:

split the bucket in two



15-826 Copyright: C. Faloutsos (2007) 42

CMU SCS

Extendible hashing

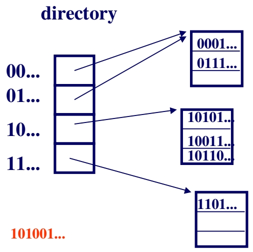
in detail:

- 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:

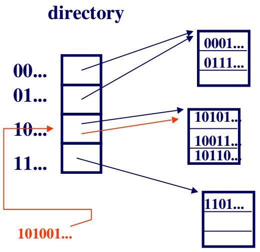
CMU SCS

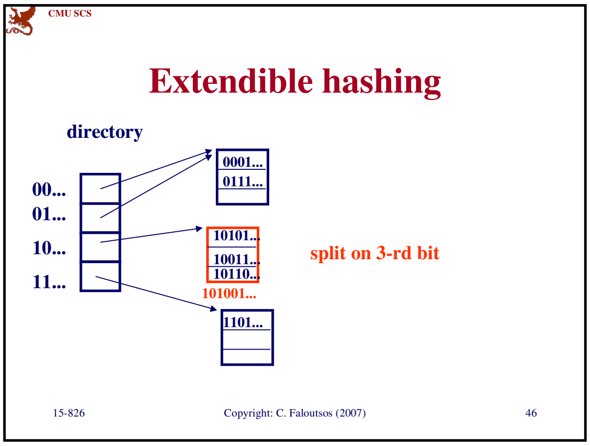
Extendible hashing

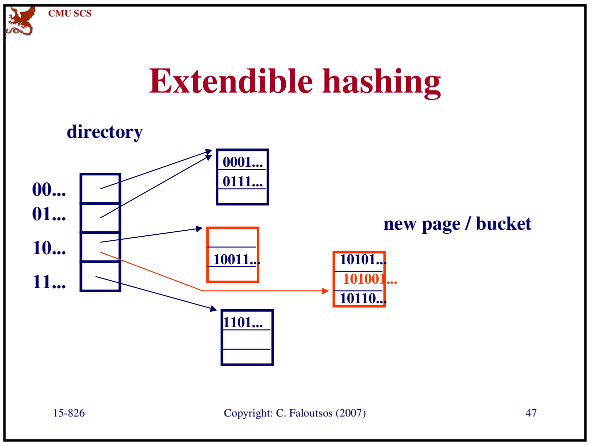


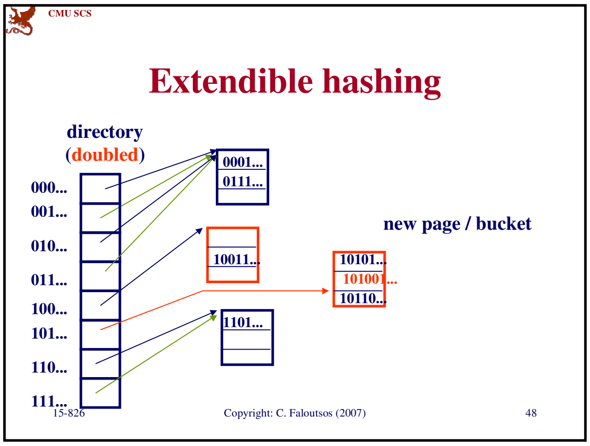
CMU SCS

Extendible hashing



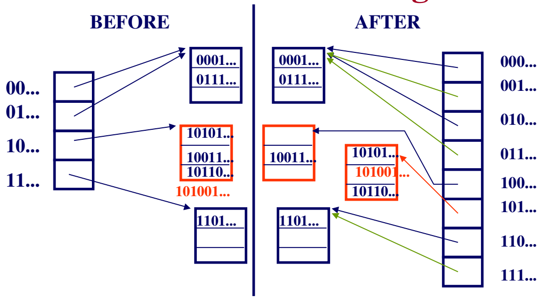

CMU SCS
Extendible hashing
directory
00...
01...
10...
11...
0001...
0111...
10101...
10011...
10110...
101001...
1101...
split on 3-rd bit
15-826
Copyright: C. Faloutsos (2007)
46


CMU SCS
Extendible hashing
directory
00...
01...
10...
11...
0001...
0111...
10011...
new page / bucket
10101...
101001...
10110...
1101...
15-826
Copyright: C. Faloutsos (2007)
47


CMU SCS
Extendible hashing
directory
(doubled)
000...
001...
010...
011...
100...
101...
110...
111...
0001...
0111...
10011...
new page / bucket
10101...
101001...
10110...
1101...
15-826
Copyright: C. Faloutsos (2007)
48

CMU SCS

Extendible hashing



15-826 Copyright: C. Faloutsos (2007) 49

CMU SCS

Extendible hashing

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

15-826 Copyright: C. Faloutsos (2007) 50

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 (2007) 51

Linear hashing - overview

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

Linear hashing

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

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

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.:

Linear hashing

Initially: *h(x) = x mod 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

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)

overflow of bucket#1

17 → bucket 1

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

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)

overflow of bucket#1

Split #0, anyway!!!

17 → bucket 1

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

CMU SCS

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)

Split #0, anyway!!!

Q: But, how?

17 → (arrow to bucket 1)

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

15-826 Copyright: C. Faloutsos (2007) 58

CMU SCS

Linear hashing

A: use two h.f.: *h0(x) = x mod N*

*h1(x) = x mod (2*N)*

17 → (arrow to bucket 1)

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

15-826 Copyright: C. Faloutsos (2007) 59

CMU SCS

Linear hashing - after split:

A: use two h.f.: *h0(x) = x mod N*

*h1(x) = x mod (2*N)*

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

17

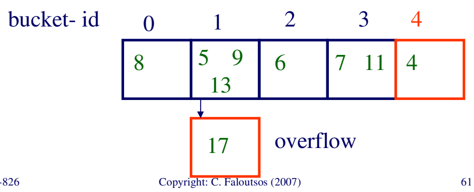
15-826 Copyright: C. Faloutsos (2007) 60

CMU SCS

Linear hashing - after split:

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

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

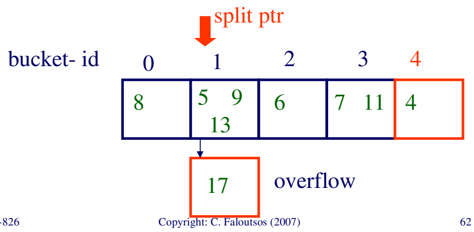


CMU SCS

Linear hashing - after split:

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

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



CMU SCS

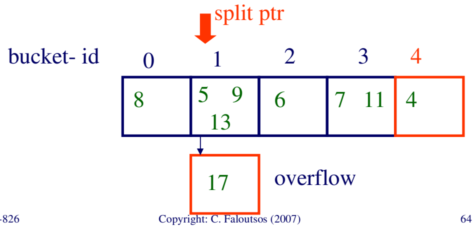
Linear hashing - overview

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

CMU SCS

Linear hashing - searching?

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

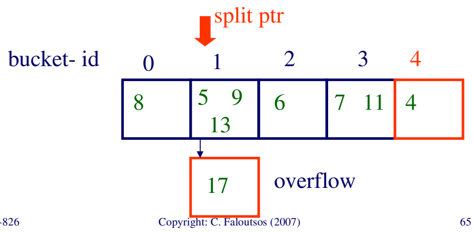


CMU SCS

Linear hashing - searching?

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

Q3: key '8'?



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*

CMU SCS

Linear hashing - overview

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

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 ++ (*)

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

A1: space utilization >= u-max

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: ....

CMU SCS

Linear hashing - insertion?

Algo: insert key '*k*'

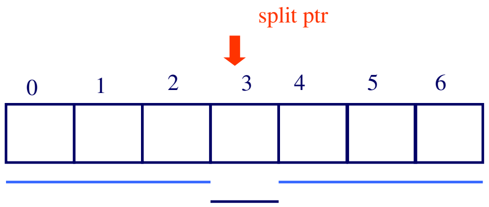
- compute appropriate bucket '*b*'
- if the **overflow criterion** is true
  - split the bucket of 'split-ptr'
  - split-ptr ++ (*)

what if we reach the right edge??

CMU SCS

Linear hashing - split now?

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

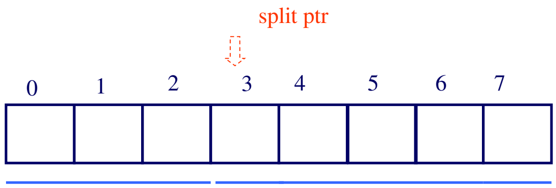


15-826 Copyright: C. Faloutsos (2007) 73

CMU SCS

Linear hashing - split now?

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

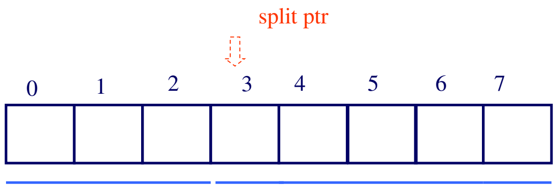


15-826 Copyright: C. Faloutsos (2007) 74

CMU SCS

Linear hashing - split now?

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



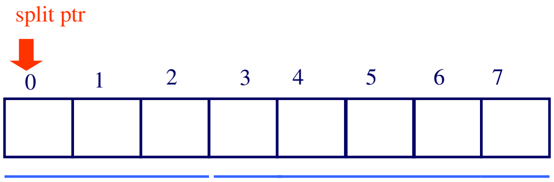
15-826 Copyright: C. Faloutsos (2007) 75

CMU SCS

Linear hashing - split now?

~~*h0(x) = x mod N (for the un-split buckets)*~~

*h1(x) = x mod (2*N) (for the splitted ones)*

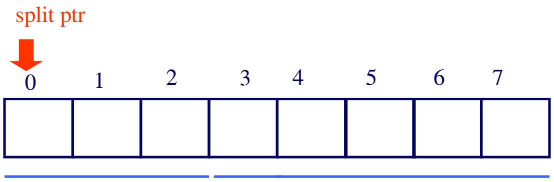


15-826 Copyright: C. Faloutsos (2007) 76

CMU SCS

Linear hashing - split now?

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



15-826 Copyright: C. Faloutsos (2007) 77

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 (2007) 78

CMU SCS

Linear hashing - overview

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

CMU SCS

Linear hashing - deletion?

- reverse of insertion:

CMU SCS

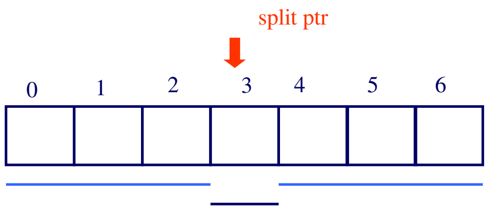
Linear hashing - deletion?

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

CMU SCS

Linear hashing - how to contract?

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

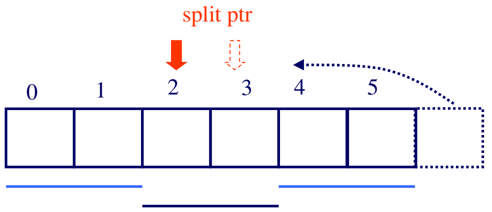


15-826 Copyright: C. Faloutsos (2007) 82

CMU SCS

Linear hashing - how to contract?

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



15-826 Copyright: C. Faloutsos (2007) 83

CMU SCS

Linear hashing - overview

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

15-826 Copyright: C. Faloutsos (2007) 84

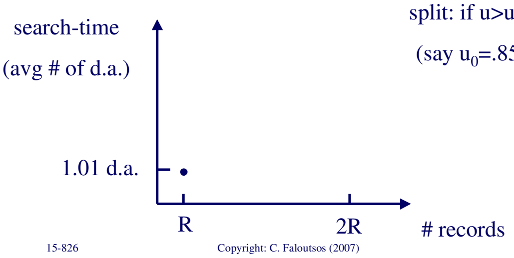
CMU SCS

Linear hashing - performance

- [Larson, TODS 1982]

split: if $u>u_0$

(say u_0=.85)



15-826 Copyright: C. Faloutsos (2007) 85

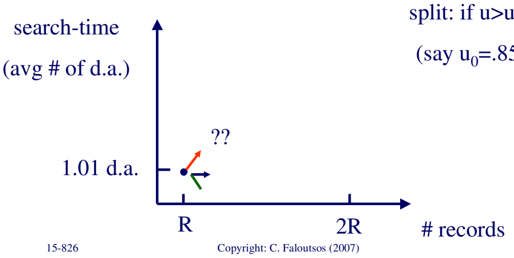
CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

split: if $u>u_0$

(say u_0=.85)



15-826 Copyright: C. Faloutsos (2007) 86

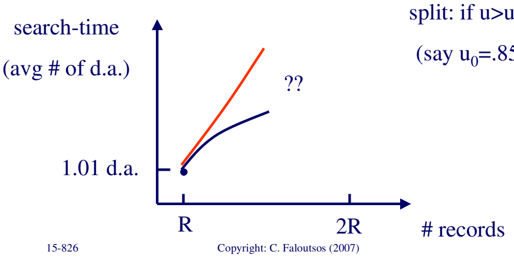
CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

split: if $u>u_0$

(say u_0=.85)

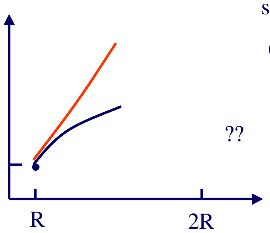


15-826 Copyright: C. Faloutsos (2007) 87

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

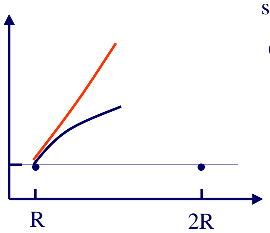


15-826 Copyright: C. Faloutsos (2007) 88

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

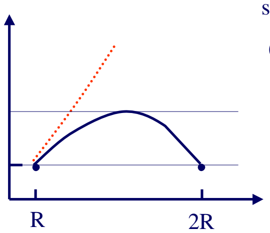


15-826 Copyright: C. Faloutsos (2007) 89

CMU SCS

Linear hashing - performance

- [Larson, TODS 1983]

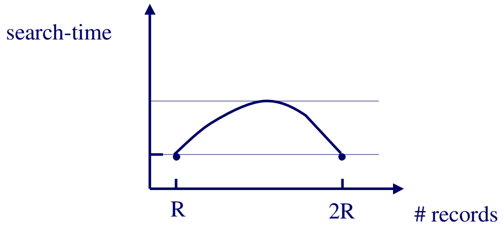


15-826 Copyright: C. Faloutsos (2007) 90

CMU SCS

Linear hashing - performance

- Q: How to shorten the maximum?



CMU SCS

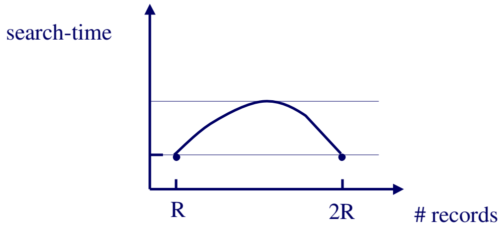
Linear hashing - overview

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

CMU SCS

Linear hashing - performance

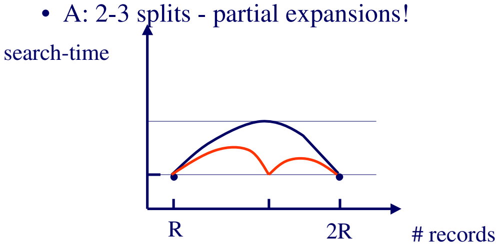
- Q: How to shorten the maximum?
- A: 2-3 splits - partial expansions!



CMU SCS

Linear hashing - performance

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

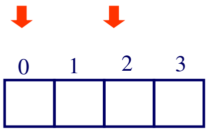


15-826 Copyright: C. Faloutsos (2007) 94

CMU SCS

Linear hashing - variations

Two split pointers! On split:

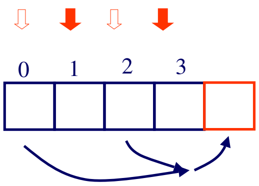


15-826 Copyright: C. Faloutsos (2007) 95

CMU SCS

Linear hashing - variations

Two split pointers! On split:

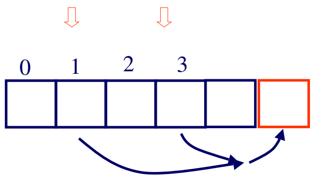


15-826 Copyright: C. Faloutsos (2007) 96

CMU SCS

Linear hashing - variations

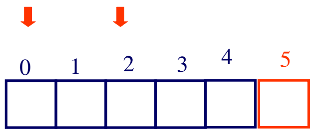
2nd split:



CMU SCS

Linear hashing - variations

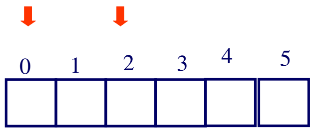
2nd split: Partial expansion! (50% larger table)



CMU SCS

Linear hashing - variations

Q: how to do the third split?

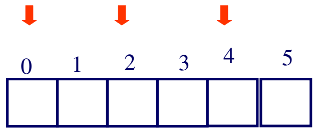


CMU SCS

Linear hashing - variations

Q: how to do the third split?

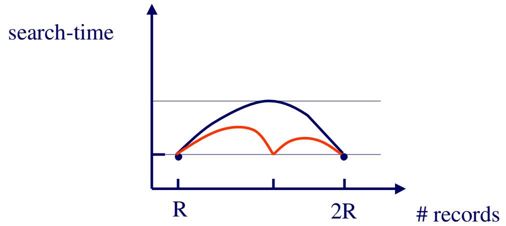
A: 3-to-4 splits now!



CMU SCS

Linear hashing - performance

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



CMU SCS

Linear hashing - overview

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

CMU SCS

Other hashing variations

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

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.!)

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.

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

