

Indexing: Part III

CPS 216
Advanced Database Systems

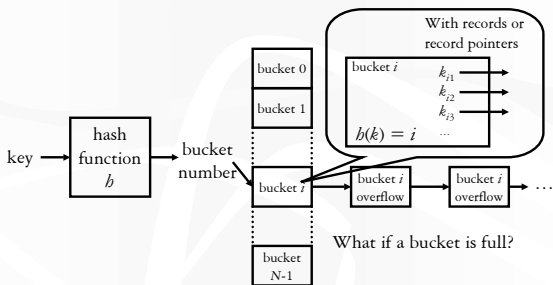
Announcements (February 15)

2

- ❖ Homework #1 graded
 - Verify your grades on Blackboard
- ❖ Homework #2 assigned today
 - Due in 2½ weeks
- ❖ Reading assignments for this and next week
 - “The” query processing survey by Graefe
 - Due next Wednesday
- ❖ Midterm and course project proposal in 3½ weeks

Static hashing

3



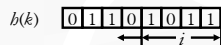
Does it make sense to use a hash-based index as a sparse index on a sorted table?

Performance of static hashing 4

- ❖ Depends on the quality of the hash function!
 - Best (hopefully average) case: one I/O!
 - Worst case: all keys hashed into one bucket!
 - See Knuth vol. 3 for good hash functions
- ❖ Rule of thumb: keep utilization at 50%-80%
- ❖ How do we cope with growth?
 - Extensible hashing
 - Linear hashing

Extensible hashing (*TODS* 1979) 5

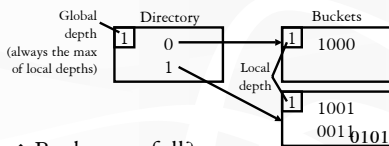
- ❖ Idea 1: use i bits of output by hash function and dynamically increase i as needed



- ❖ Problem: $++i =$ double the number of buckets!
 - ❖ Idea 2: use a directory
 - Just double the directory size
 - Many directory entries can point to the same bucket
 - Only split overflowed buckets
- “One more level of indirection solves everything!”

Extensible hashing example (slide 1) 6

- ❖ Insert k with $b(k) = 0101$

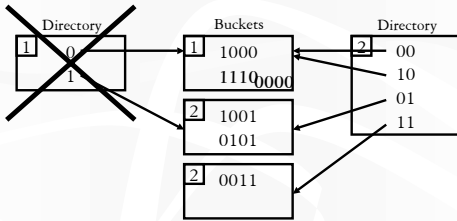


- ❖ Bucket too full?
 - $++$ local depth, split bucket, and $++$ global depth (double the directory size) if necessary
 - Allowing some overflow is fine too

Extensible hashing example (slide 2)

7

❖ Insert 1110, 0000



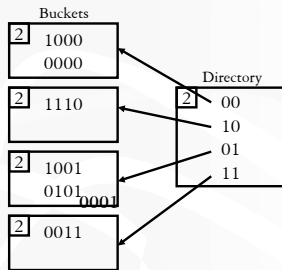
❖ Split again

- No directory doubling this time

Extensible hashing example (slide 3)

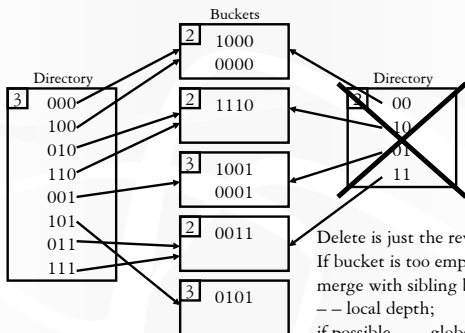
8

❖ Insert 0001



Extensible hashing example (slide 4)

9



Delete is just the reverse:
 If bucket is too empty,
 merge with sibling bucket,
 -- local depth;
 if possible, -- global depth
 and half the directory

Summary of extensible hashing 10

- ❖ Pros
 - Handles growing files
 - No full reorganization
- ❖ Cons

Linear hashing (VLDB 1980) 11

- ❖ Grow only when utilization exceeds a given threshold
- ❖ No extra indirection
 - Some extra math to figure out the right bucket

Insert 0101
Threshold exceeded; grow!

0	1
0000 1010	1111 0101

$i = 1$ Number of bits in use = $\lceil \log_2 n \rceil$
 $n = 2$ Number of primary buckets

Linear hashing example (slide 2) 12

- ❖ Grows linearly (hence the name)
- ❖ Always split the $(n - 2^{\lceil \log_2 n \rceil})$ -th bucket (0-based index)
 - Intuitively, the first bucket with the lowest depth
 - Not necessarily the bucket being inserted into!

Insert 0001 Insert 1100
Threshold exceeded; grow!

00	1	10
0000 1100	1111 0101	1010

$i = 2$
 $n = 3$

0001

Linear hashing example (slide 3)

13

Insert 1110
Threshold exceeded; grow!

00	01	10	11
0000 1100	0001 0101	1010 1110	1111

$i = 2$
 $n = 4$

Linear hashing example (slide 4)

14

❖ Look up 1110

- Bucket 110 (6-th bucket) is not here
- Then look in the $(6 - 2^{\lfloor \log_2 n \rfloor})$ -th bucket (= 2nd)

000	01	10	11	100
0000 1100	0001 0101	1010 1110	1111	1100

$i = 3$
 $n = 5$

Summary of linear hashing

15

❖ Pros

- Handles growing files
- No full reorganization

❖ Cons

Hashing versus B-trees

- ❖ Hashing is faster on average, but the worst case can be really bad
- ❖ B-trees provide performance guarantees, and they are not that tall in practice
- ❖ Hashing destroys order!
- ❖ B-trees provide order and support range queries
