Indexing: Part III

CPS 216
Advanced Database Systems

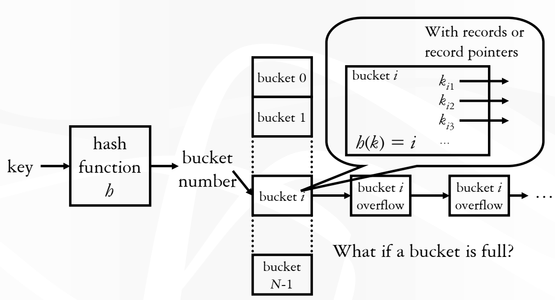
Announcements

2

- Homework #2 due in one week (February 26)
- Recitation session this Friday (February 21)
  - Homework #1 sample solution and graded assignments
  - Homework #1 common problems
  - Homework #2 Q&A
- Reading assignment
  - "A Study of Index Structures for Main Memory Database Management Systems," by Lehman and Carey, *VLDB* 1986

Static hashing

3



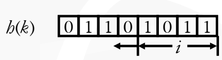
What if a bucket is full?

Performance of static hashing

- 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)

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

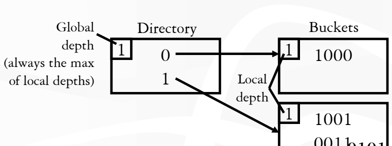


- Problem: $++i$ = double the number of buckets!
- Idea 2: use a directory
  - Just double the directory size
  - Many directory entries can point to the same bucket
  - Only split overflowed buckets

  "One more level of indirection solves everything!"

Extensible hashing example (slide 1)

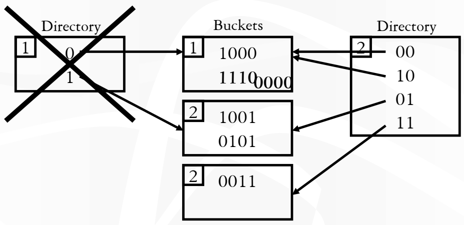
- Insert k with $h(k)$ = 0101



- 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)

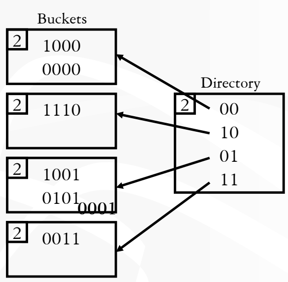
- Insert 1110, 0000



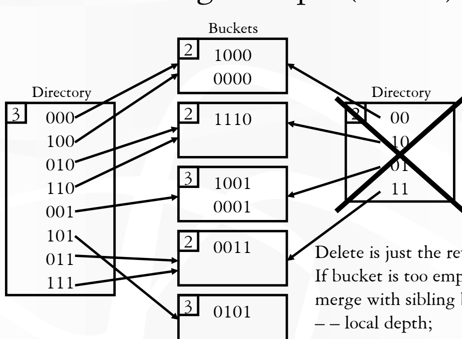
- Split again
  - No directory doubling this time

Extensible hashing example (slide 3)

- Insert 0001



Extensible hashing example (slide 4)



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

10

Summary of extensible hashing

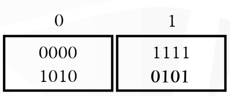
- Pros
  - Handles growing files
  - No full reorganization
- Cons
  - One more level of indirection
  - Directory size still doubles
  - Sometimes doubling is not enough!

11

Linear hashing (*VLDB* 1980)

- 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!



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

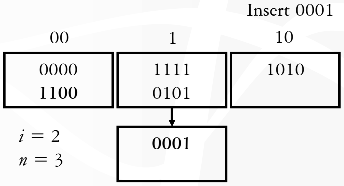
12

Linear hashing example (slide 2)

- Grows linearly (hence the name)
- Always split the $(n - 2^{\lfloor \log_2 n \rfloor})$-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!



$i = 2$
$n = 3$

Linear hashing example (slide 3)

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)

- 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	0001 0101	1010 1110	1111	1100

$i = 3$
$n = 5$

Summary of linear hashing

- Pros
  - Handles growing files
  - No full reorganization
  - No extra level of indirection
- 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