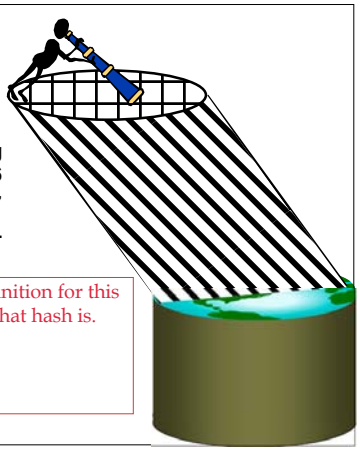
Hash-based Indexes

CS 186, Spring 2006

Lecture 7

R &G Chapter 11



HASH, x. There is no definition for this word -- nobody knows what hash is.

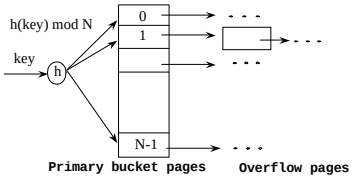
Ambrose Bierce,
"The Devil's Dictionary", 1911

Introduction

- ***As for any index, 3 alternatives for data entries*** k*:
  - ① Data record with key value **k**
  - ② <**k**, rid of data record with search key value **k**>
  - ③ <**k**, list of rids of data records with search key **k**>
  - – Choice orthogonal to the *indexing technique*
- ***Hash-based*** **indexes are best for** ***equality selections.*** *Cannot* **support range searches.**
- **Static and dynamic hashing techniques exist; trade-offs similar to ISAM vs. B+ trees.**

Static Hashing

- **# primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.**
- h(***k***) **MOD N= bucket to which data entry with key *k* belongs. (N = # of buckets)**



Static Hashing (Contd.)

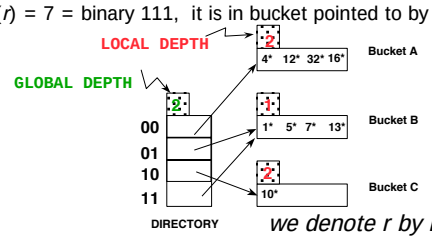
- **Buckets contain *data entries*.**
- **Hash fn works on *search key* field of record *r*. Use its value MOD N to distribute values over range 0 ... N-1.**
  - **h**(*key*) = (a * *key* + b) usually works well.
  - a and b are constants; lots known about how to tune **h**.
- **Long overflow chains can develop and degrade performance.**
  - *Extendible* and *Linear Hashing*: Dynamic techniques to fix this problem.

Extendible Hashing

- **Situation: Bucket (primary page) becomes full. Why not re-organize file by *doubling* # of buckets?**
  - Reading and writing all pages is expensive!
- ***Idea*****: Use** ***directory of pointers to buckets,*** **double # of buckets by *doubling the directory,* splitting just the bucket that overflowed!**
  - Directory much smaller than file, so doubling it is much cheaper. Only one page of data entries is split. *No overflow page*!
  - Trick lies in how hash function is adjusted!

Example

- **Directory is array of size 4.**
- **Bucket for record *r* has entry with index = `*global depth*' least significant bits of h(*r*);**
  - If **h**(*r*) = 5 = binary 101, it is in bucket pointed to by 01.
  - If **h**(*r*) = 7 = binary 111, it is in bucket pointed to by 11.



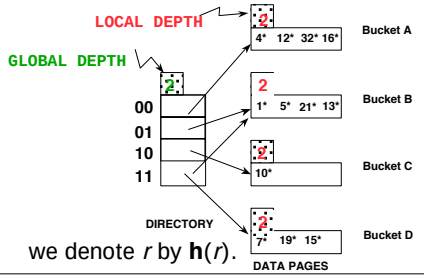
we denote r by ***h****(r).*

Handling Inserts

- Find bucket where record belongs.
- If there's room, put it there.
- Else, if bucket is full, *split* it:
  - increment local depth of original page
  - allocate new page with new local depth
  - re-distribute records from original page.
  - add entry for the new page to the directory

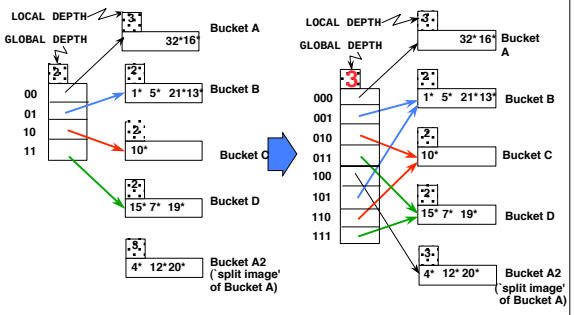
Example: Insert 21, then 19, 15

- **21 = 10101**
- **19 = 10011**
- **15 = 01111**



we denote *r* by **h**(*r*).

Insert **h**(r)=20 (Causes Doubling)

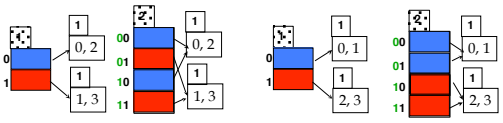


Points to Note

- **20 = binary 10100. Last 2 bits (00) tell us *r* belongs in either A or A2. Last 3 bits needed to tell which.**
  - *Global depth of directory*: Max # of bits needed to tell which bucket an entry belongs to.
  - *Local depth of a bucket*: # of bits used to determine if an entry belongs to this bucket.
- **When does bucket split cause directory doubling?**
  - Before insert, *local depth* of bucket = *global depth*. Insert causes *local depth* to become > *global depth*; directory is doubled by *copying it over* and `fixing' pointer to split image page.

Directory Doubling

Why use least significant bits in directory? Allows for doubling by copying the directory and appending the new copy to the original.



Least Significant vs. Most Significant

Comments on Extendible Hashing

- **If directory fits in memory, equality search answered with one disk access; else two.**
  - 100MB file, 100 bytes/rec, 4K pages contains 1,000,000 records (as data entries) and 25,000 directory elements; chances are high that directory will fit in memory.
  - Directory grows in spurts, and, if the distribution *of hash values* is skewed, directory can grow large.
  - Multiple entries with same hash value cause problems!
- Delete: **If removal of data entry makes bucket empty, can be merged with `split image'. If each directory element points to same bucket as its split image, can halve directory.**

Administrivia - Exam Schedule Change

- **Exam 1 will be held in class on Tues 2/21 (not on the previous thurs as originally scheduled).**
- **Exam 2 will remain as scheduled Thurs 3/23 (unless you want to do it over spring break!!!).**

Linear Hashing

- A dynamic hashing scheme that handles the problem of long overflow chains without using a directory.
- Directory avoided in LH by using *temporary* overflow pages, and choosing the bucket to split in a *round-robin* fashion.
- When *any* bucket overflows split the bucket that is currently pointed to by the "*Next*" pointer and then increment that pointer to the next bucket.

Linear Hashing – The Main Idea

- **Use a family of hash functions h_0, h_1, h_2, ...**
- $h_i(key) = h(key) \bmod (2^i N)$
  - N = initial # buckets
  - **h** is some hash function
- h_{i+1} doubles the range of h_i (similar to directory doubling)

Linear Hashing (Contd.)

- Algorithm proceeds in `rounds'. Current round number is "*Level*".
- There are N_{Level} (= N * 2^{Level}) buckets at the beginning of a round
- Buckets 0 to *Next-1* have been split; *Next* to N_{Level} have not been split yet this round.
- Round ends when all initial buckets have been split (i.e. *Next* = N_{Level}).
- To start next round:

  Level++;

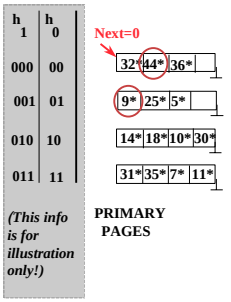
  Next = 0;

LH Search Algorithm

- **To find bucket for data entry *r*, find $h_{Level}(r)$:**
  - If $h_{Level}(r)$ >= Next (i.e., $h_{Level}(r)$ is a bucket that hasn't been involved in a split this round) then *r* belongs in that bucket for sure.
  - Else, r could belong to bucket $h_{Level}(r)$ ***or*** bucket $h_{Level}(r) + N_{Level}$ must apply $h_{Level+1}(r)$ to find out.

Example: Search 44 (11100), 9 (01001)

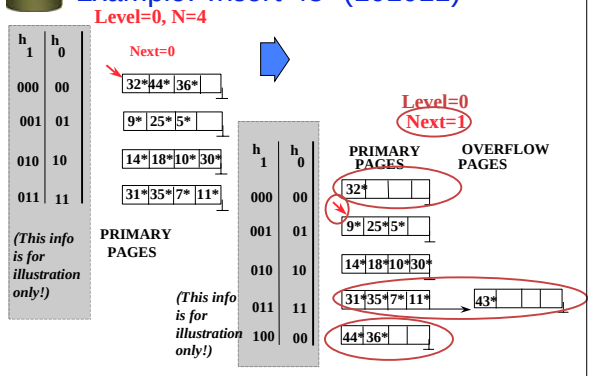
Level=0, Next=0, N=4



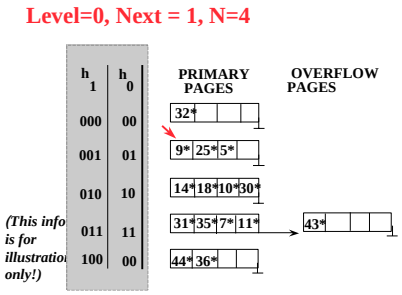
Linear Hashing - Insert

- **Find appropriate bucket**
- **If bucket to insert into is full:**
  - Add overflow page and insert data entry.
  - Split *Next* bucket and increment *Next*.
    - Note: This is likely NOT the bucket being inserted to!!!
    - to split a bucket, create a new bucket and use $\mathbf{h}_{\text{Level+1}}$ to re-distribute entries.
- Since buckets are split round-robin, long overflow chains don't develop!

Example: Insert 43 (101011)

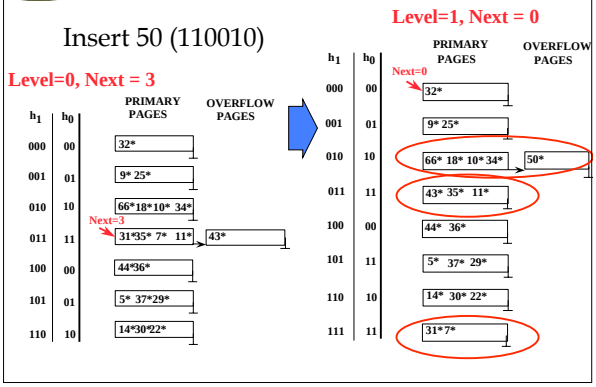


Example: Search 44 (11100), 9 (01001)



Example: End of a Round

Insert 50 (110010)



LH Described as a Variant of EH

- **The two schemes are actually quite similar:**
  - Begin with an EH index where directory has *N* elements.
  - Use overflow pages, split buckets round-robin.
  - First split is at bucket 0. (Imagine directory being doubled at this point.) But elements <1,*N*+1>, <2,*N*+2>, ... are the same. So, need only create directory element *N*, which differs from 0, now.
    - When bucket 1 splits, create directory element *N*+1, etc.
- **So, "directory" can double gradually. Also, primary bucket pages are created in order. If they are *allocated* in sequence too (so that finding i'th is easy), we actually don't need a directory! Voila, LH.**

Summary

- **Hash-based indexes: best for equality searches, cannot support range searches.**
- **Static Hashing can lead to long overflow chains.**
- **Extendible Hashing avoids overflow pages by splitting a full bucket when a new data entry is to be added to it. (*Duplicates may require overflow pages.*)**
  - Directory to keep track of buckets, doubles periodically.
  - Can get large with skewed data; additional I/O if this does not fit in main memory.

Summary (Contd.)

- **Linear Hashing avoids directory by splitting buckets round-robin, and using overflow pages.**
  - Overflow pages not likely to be long.
  - Space utilization could be lower than Extendible Hashing, since splits not concentrated on `dense' data areas.
  - Can tune criterion for triggering splits to trade-off slightly longer chains for better space utilization.
- **For hash-based indexes, a *skewed* data distribution is one in which the *hash values* of data entries are not uniformly distributed!**

Administrivia - Exam Schedule Change

- **Exam 1 will be held in class on Tues 2/21 (not on the previous thurs as originally scheduled).**
- **Exam 2 will remain as scheduled Thurs 3/23 (unless you want to do it over spring break!!!).**