

CSE 562 Database Systems

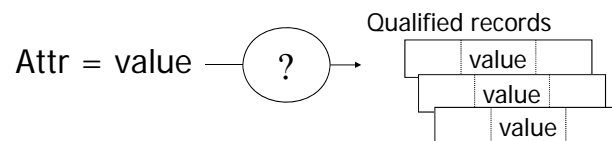
Indexing

Some slides are based or modified from originals by
Database Systems: The Complete Book,
Pearson Prentice Hall, 2nd Edition
© 2008 Garcia-Molina, Ullman, and Widom

.cse@buffalo

Goal of Indexing

Given condition(s) on attribute(s) find qualified records



Condition may also be

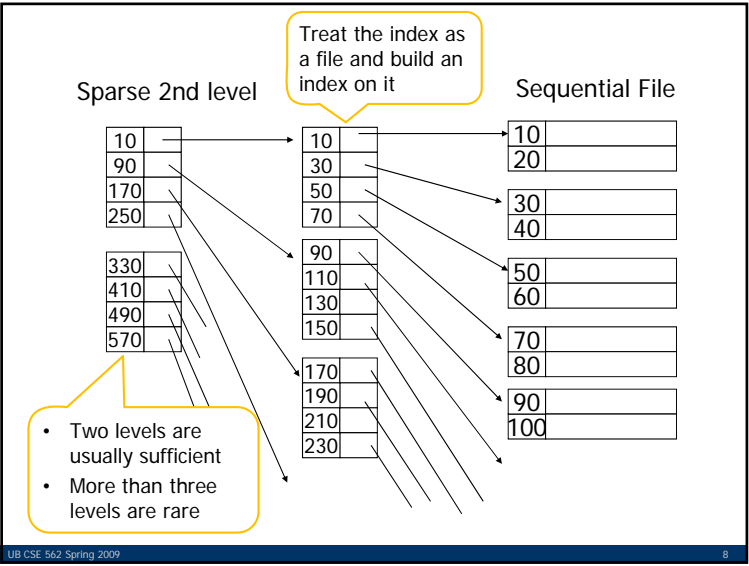
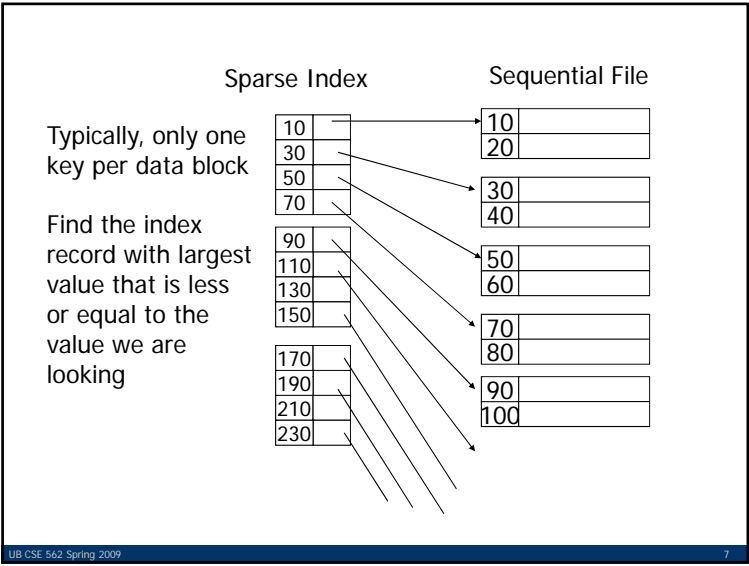
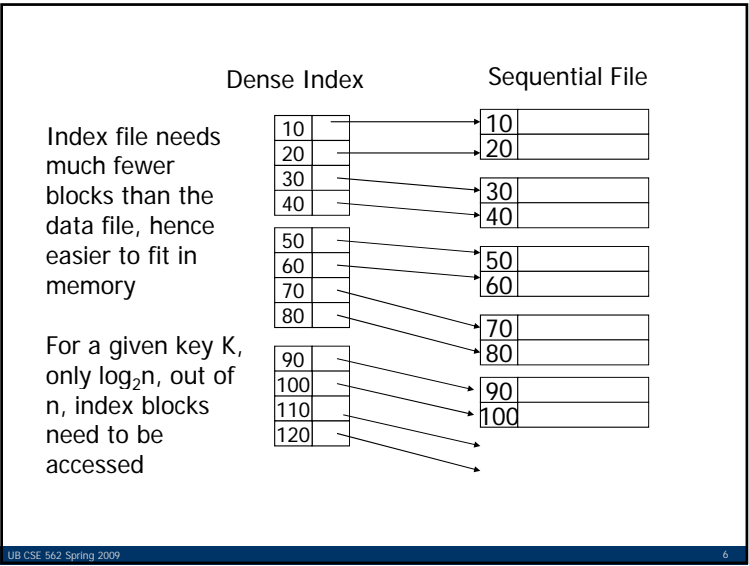
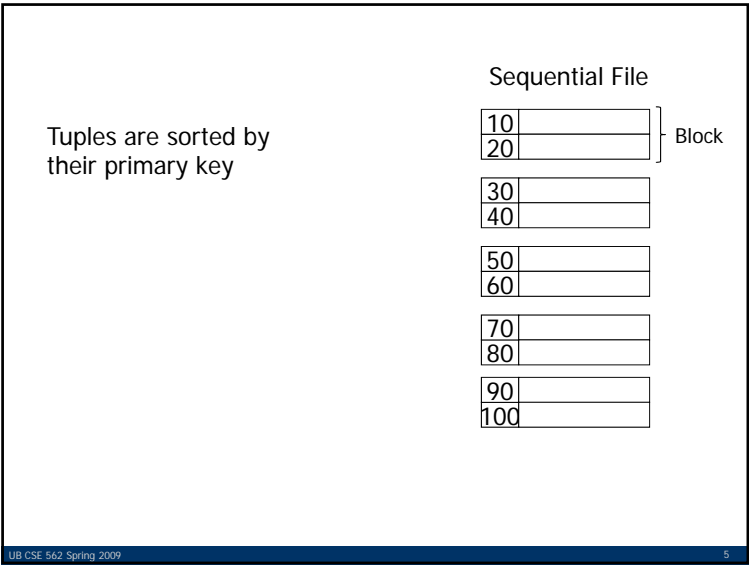
- Attr > value
- Attr >= value

Indexes (or Indices)

- Data Structures used for quickly locating tuples that meet a specific type of condition
 - *Equality* condition: Find Movie tuples where Director = X
 - Other conditions possible, e.g., *range* conditions: Find Employee tuples where Salary > 40 AND Salary < 50
- Many types of indexes. Evaluate them on:
 - Access time
 - Insertion/Deletion time
 - Condition types
 - Disk Space needed

Topics

- Conventional indexes
- B-Trees
- Hashing schemes



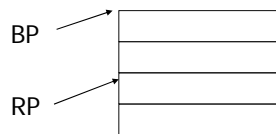
- Comment:
{FILE,INDEX} may be contiguous
or not (blocks chained)

Question:

- Can we build a dense, 2nd level index for a dense index?

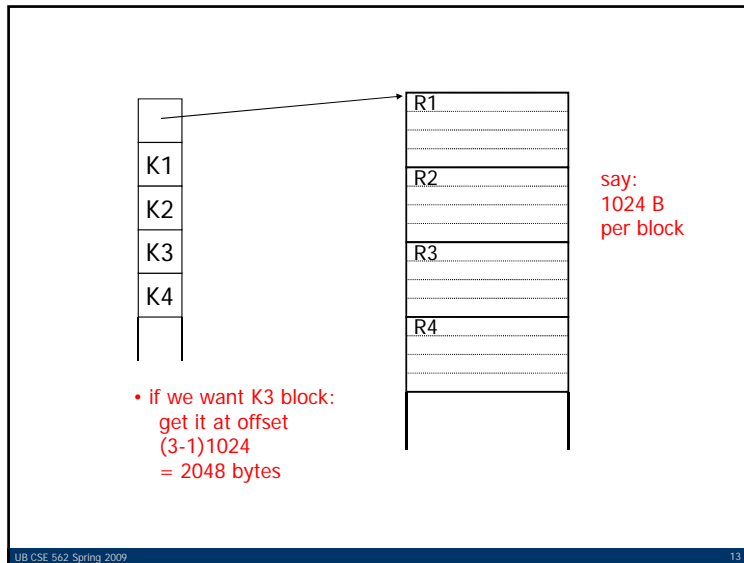
Notes on Pointers

- *Record pointers* consist of *block pointer* and position of record in the block
- Using the block pointer only saves space at no extra disk accesses cost
- Block pointer (sparse index) can be smaller than record pointer



Notes on Pointers

- If file is contiguous, then we can omit pointers (i.e., compute them)



Terms

- Index sequential file
- Search key (≠ primary key)
- Primary index (on Sequencing field)
 - The index on the attribute (a.k.a. search key) that determines the sequencing of the table
- Secondary index
 - Index on any other attribute
- Dense index (all Search Key values in)
- Sparse index
- Multi-level index

UB CSE 562 Spring 2009 14

Sparse vs. Dense Tradeoff

- Sparse: Less index space per record can keep more of index in memory
- Dense: Can tell if any record exists without accessing file

(Later:

- sparse better for insertions
- dense needed for secondary indexes)

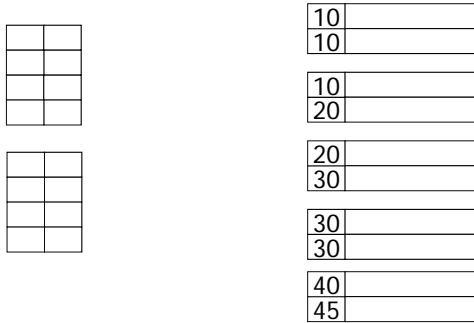
UB CSE 562 Spring 2009 15

Next

- Duplicate keys
- Deletion/Insertion
- Secondary indexes

UB CSE 562 Spring 2009 16

Duplicate Keys

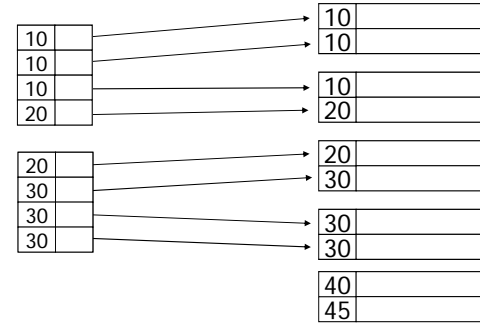


UB CSE 562 Spring 2009

17

Duplicate Keys

Dense index, one way to implement?

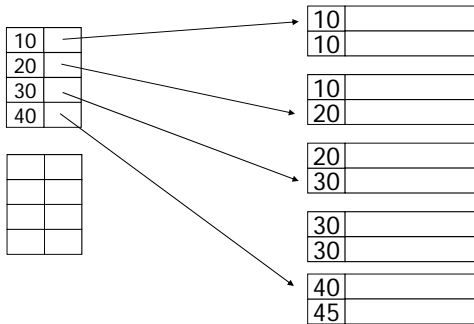


UB CSE 562 Spring 2009

18

Duplicate Keys

Dense index, better way?

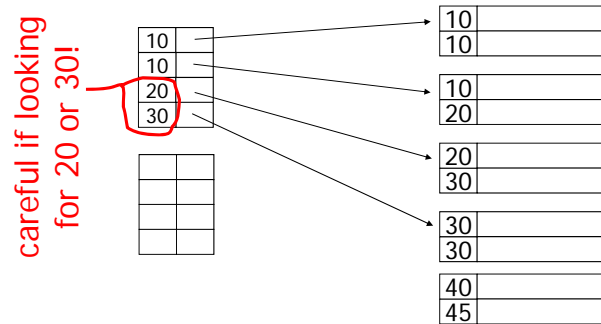


UB CSE 562 Spring 2009

19

Duplicate Keys

Sparse index, one way?



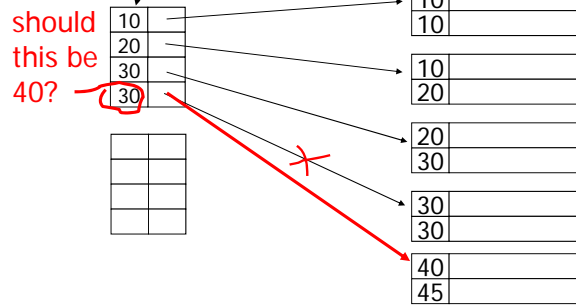
UB CSE 562 Spring 2009

20

Duplicate Keys

Sparse index, another way?

– place first new key from block



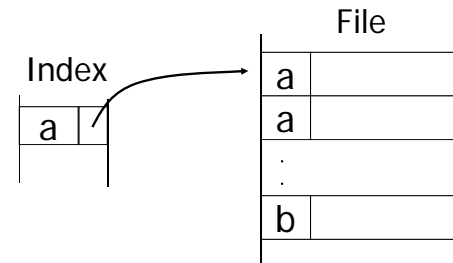
UB CSE 562 Spring 2009

21

Summary

Duplicate values,
primary index

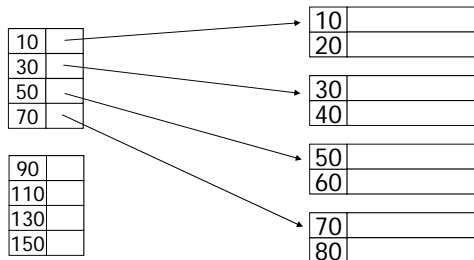
- Index may point to first instance of each value only



UB CSE 562 Spring 2009

22

Deletion from sparse index

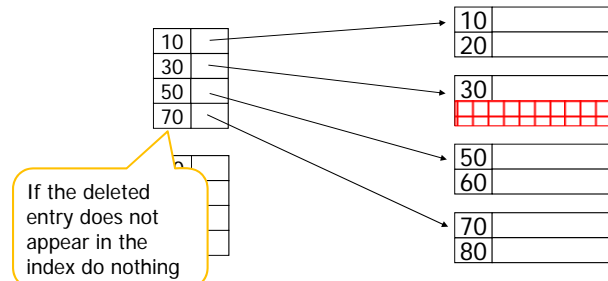


UB CSE 562 Spring 2009

23

Deletion from sparse index

– delete record 40

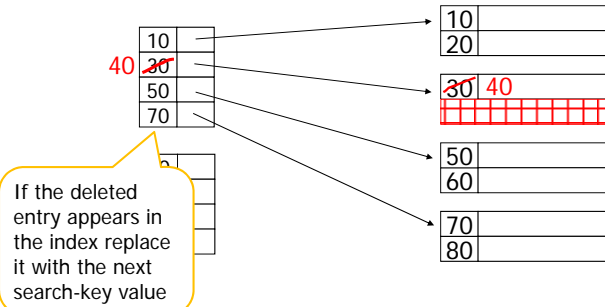


UB CSE 562 Spring 2009

24

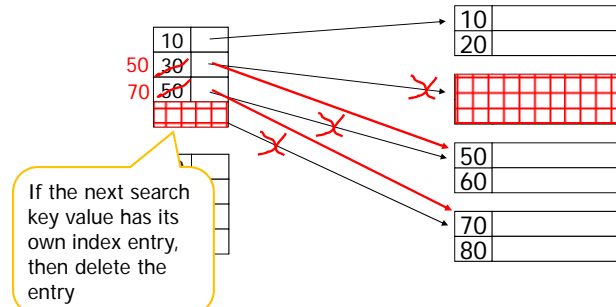
Deletion from sparse index

- delete record 30

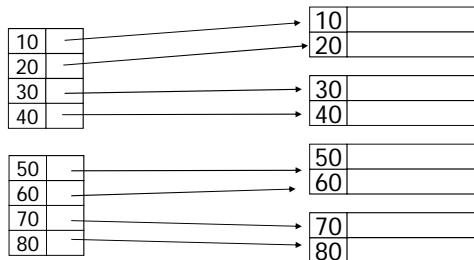


Deletion from sparse index

- delete records 30 & 40

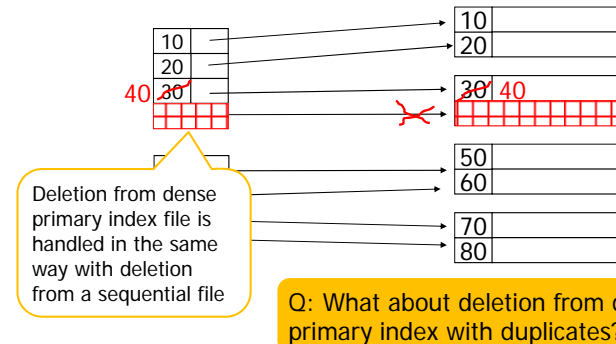


Deletion from dense index



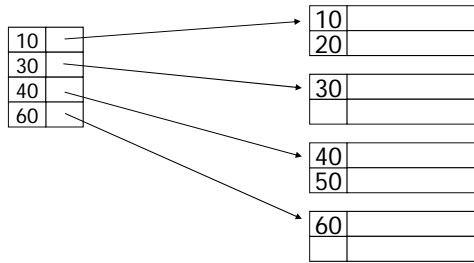
Deletion from dense index

- delete record 30



Q: What about deletion from dense primary index with duplicates?

Insertion, sparse index case

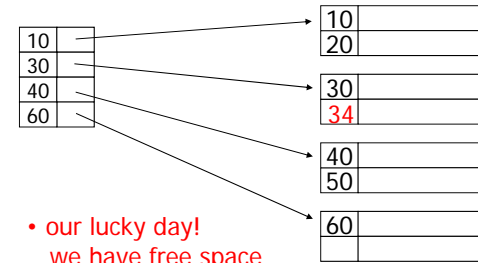


UB CSE 562 Spring 2009

29

Insertion, sparse index case

– insert record 34



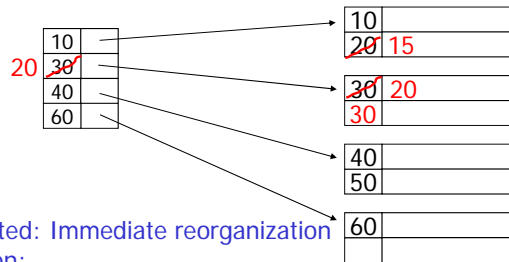
- our lucky day!
we have free space
where we need it!

UB CSE 562 Spring 2009

30

Insertion, sparse index case

– insert record 15



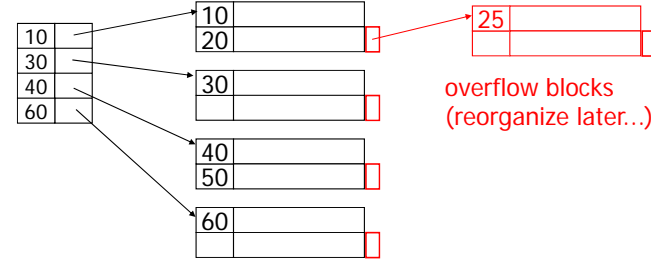
- Illustrated: Immediate reorganization
- Variation:
 - insert new block (chained file)
 - update index

UB CSE 562 Spring 2009

31

Insertion, sparse index case

– insert record 25



overflow blocks
(reorganize later...)

- How often do we reorganize and how expensive is it?
B-Trees offer convincing answers

UB CSE 562 Spring 2009

32

Insertion, dense index case

- Similar
- Often more expensive . . .

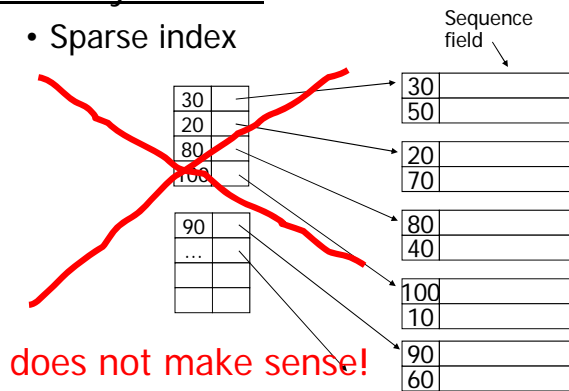
Secondary indexes

File not sorted on secondary search key

	Sequence field
30	
50	
20	
70	
80	
40	
100	
10	
90	
60	

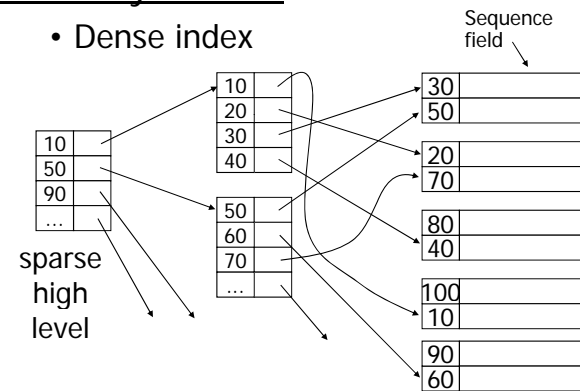
Secondary indexes

- Sparse index



Secondary indexes

- Dense index

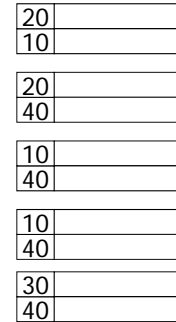


With secondary indexes:

- Lowest level is dense
- Other levels are sparse

Also: Pointers are record pointers
(not block pointers; not computed)

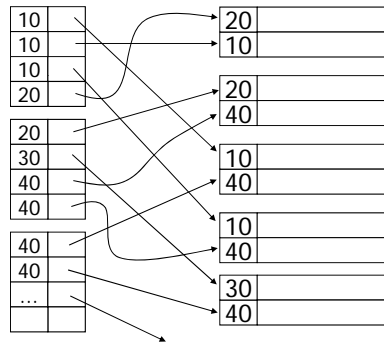
Duplicate values & secondary indexes



Duplicate values & secondary indexes

one option...

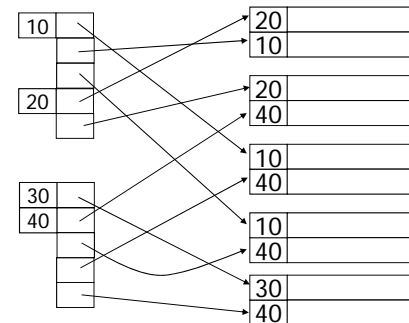
Problem:
excess overhead!
• disk space
• search time



Duplicate values & secondary indexes

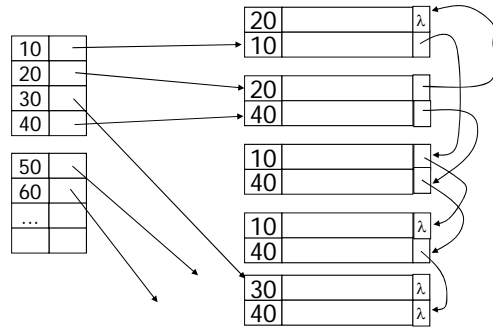
another option...

Problem:
variable size
records in
index!



Duplicate values & secondary indexes

Another idea:
Chain records
with same
key?



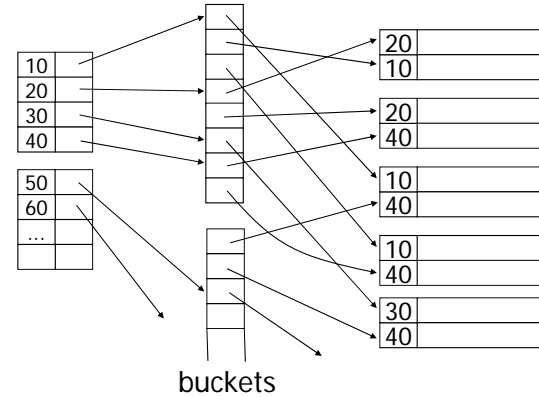
Problems:

- Need to add fields to records, messes up maintenance
- Need to follow chain to know records

UB CSE 562 Spring 2009

41

Duplicate values & secondary indexes



UB CSE 562 Spring 2009

42

Why "bucket" idea is useful

Indexes

Name: primary
Dept: secondary
Floor: secondary

Records

EMP (name,dept,floor,...)

- Enables the processing of queries working with pointers only
- Very common technique in Information Retrieval

UB CSE 562 Spring 2009

43

Advantage of Buckets: Process Queries Using Pointers Only

Find employees in Toys dept on the 4th floor:

```
SELECT Name FROM Employee
WHERE Dept="Toys" AND Floor=4
```

Dept Index

Toys	→
PCs	→
Pens	→
Suits	→

Aaron	Suits	4
Helen	Pens	3
Jack	PCs	4
Jim	Toys	4
Joe	Toys	3
Nick	PCs	2
Walt	Toys	5
Yannis	Pens	1

Floor Index

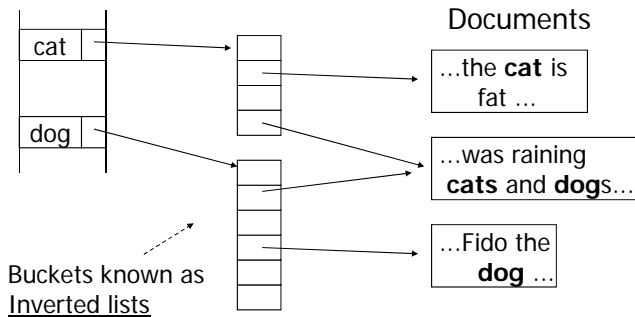
...	1
...	2
...	3
...	4

Intersect Toys bucket and
4th floor bucket to get
set of matching EMP's

UB CSE 562 Spring 2009

44

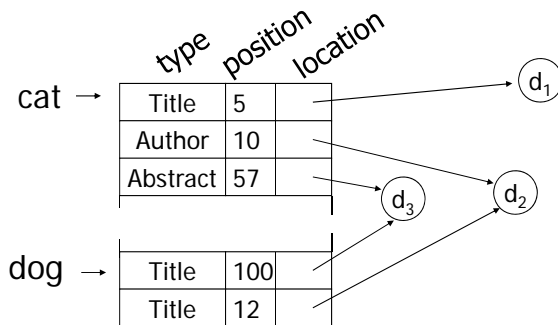
This idea used in
text information retrieval



IR QUERIES

- Find articles with "cat" and "dog"
 - Intersect inverted lists
- Find articles with "cat" or "dog"
 - Union inverted lists
- Find articles with "cat" and not "dog"
 - Subtract list of dog pointers from list of cat pointers
- Find articles with "cat" in title
- Find articles with "cat" and "dog" within 5 words

Common technique:
more info in inverted list



Posting: an entry in inverted list.
Represents occurrence of term in article

Size of a list: 1 Rare words or miss-spellings
(in postings)

↓

10⁶ Common words

Size of a posting: 10-15 bits (compressed)

IR DISCUSSION

- Stop words
- Truncation
- Thesaurus
- Full text vs. Abstracts
- Vector model

Vector space model

	w1	w2	w3	w4	w5	w6	w7	...
DOC =	<1	0	0	1	1	0	0	...>
Query=	<0	0	1	1	0	0	0	...>
PRODUCT =				↓				
				1	+	=	score

- Tricks to weigh scores + normalize

e.g.: Match on common word not as useful as match on rare words...

Summary of Indexing So Far

- Basic topics in conventional indexes
 - multiple levels
 - sparse/dense
 - duplicate keys and buckets
 - deletion/insertion similar to sequential files
 - secondary indexes

Conventional Indexes

Advantage:

- Simple algorithms
- Index is sequential file
good for scans

Disadvantage:

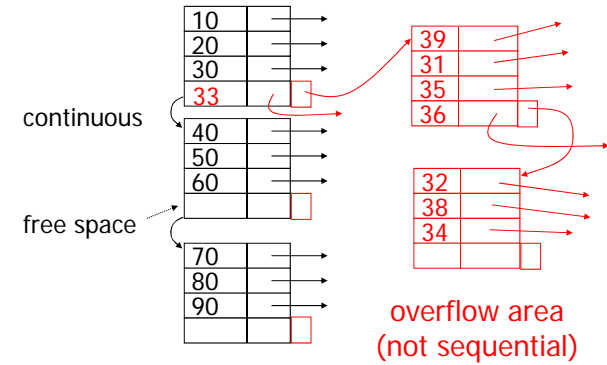
- Inserts expensive, and/or
- Eventually sequentiality is lost
because of overflows,
reorganizations are needed

UB CSE 562 Spring 2009

53

Example

Index (sequential)



UB CSE 562 Spring 2009

54

Topics

- Conventional indexes
- B-Trees \Rightarrow NEXT
- Hashing schemes

UB CSE 562 Spring 2009

55

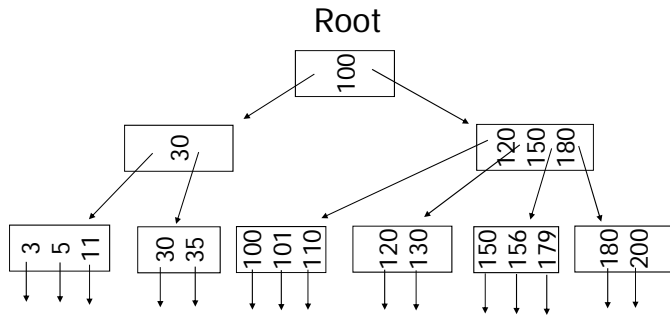
- NEXT: Another type of index
 - Give up on sequentiality of index
 - Try to get "balance"

UB CSE 562 Spring 2009

56

B+ Tree Example

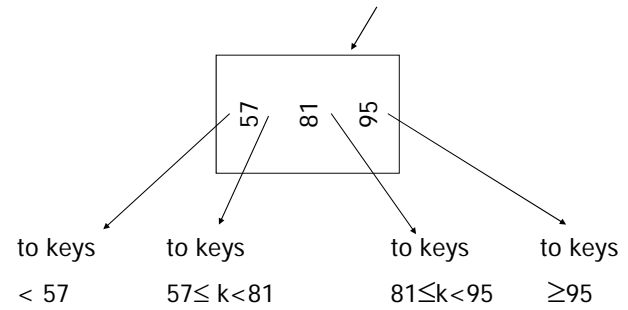
n=3



UB CSE 562 Spring 2009

57

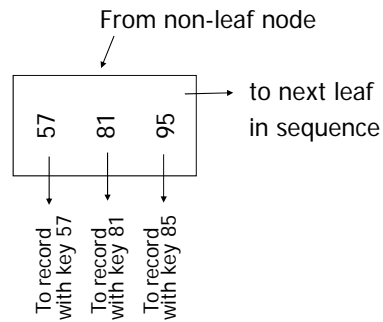
Sample non-leaf



UB CSE 562 Spring 2009

58

Sample leaf node:



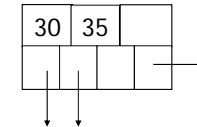
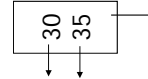
UB CSE 562 Spring 2009

59

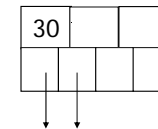
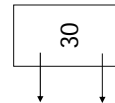
In textbook's notation

n=3

Leaf:



Non-leaf:



UB CSE 562 Spring 2009

60

Size of nodes: $\left\{ \begin{array}{l} n+1 \text{ pointers} \\ n \text{ keys} \end{array} \right.$ (fixed)

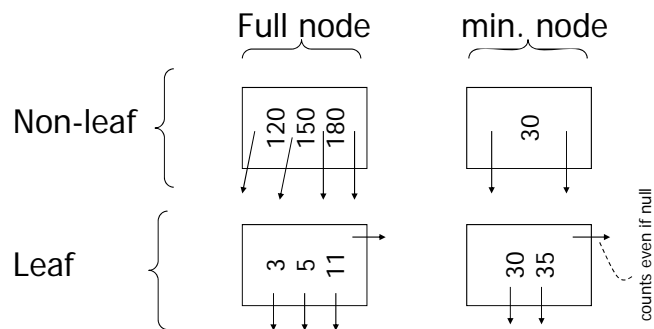
Don't want nodes to be too empty

- Non-root nodes have to be at least half-full
- Use at least

Non-leaf: $\lceil (n+1)/2 \rceil$ pointers

Leaf: $\lfloor (n+1)/2 \rfloor$ pointers to data

$n=3$



B+ Tree rules tree of order n

- (1) All leaves at same lowest level (balanced tree)
- (2) Pointers in leaves point to records except for "sequence pointer"

(3) Number of pointers/keys for B+Tree

	Max ptrs	Max keys	Min ptrs→data	Min keys
Non-leaf (non-root)	$n+1$	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
Leaf (non-root)	$n+1$	n	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root	$n+1$	n	1	1

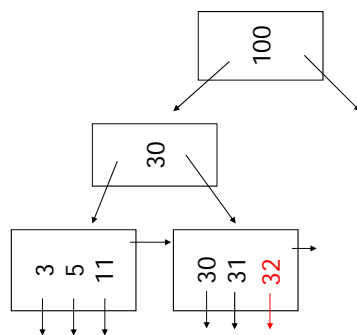
Counting sequence pointer also

Insert into B+Tree

- (a) simple case
 - space available in leaf
- (b) leaf overflow
- (c) non-leaf overflow
- (d) new root

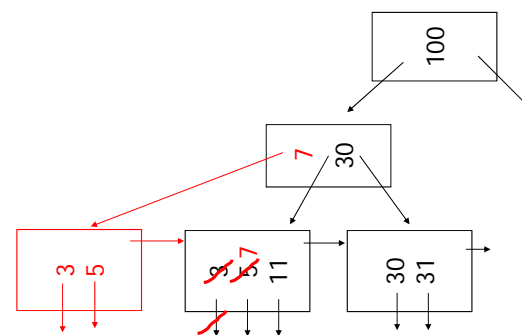
(a) Insert key = 32

$n=3$



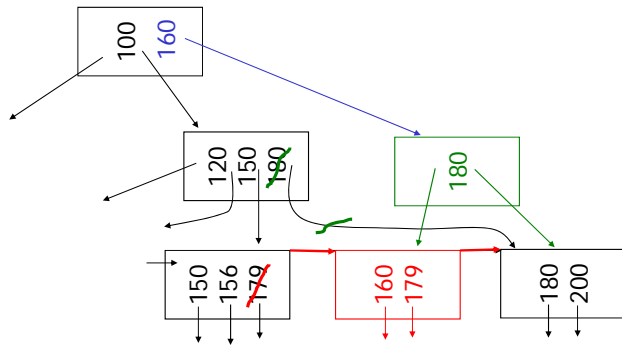
(a) Insert key = 7

$n=3$



(c) Insert key = 160

n=3

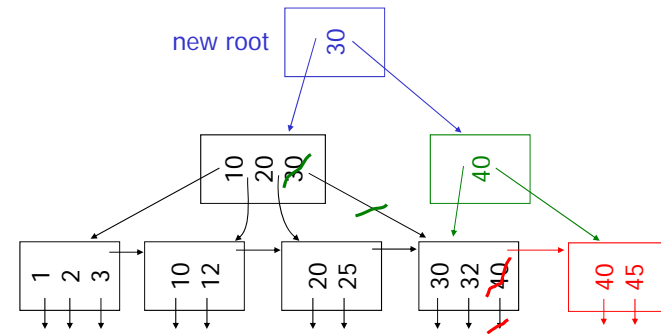


UB CSE 562 Spring 2009

69

(d) New root, insert 45

n=3



UB CSE 562 Spring 2009

70

Deletion from B+ Tree

- (a) Simple case - no example
- (b) Coalesce with neighbor (sibling)
- (c) Re-distribute keys
- (d) Cases (b) or (c) at non-leaf

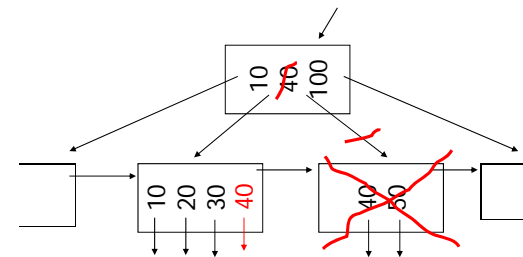
UB CSE 562 Spring 2009

71

(b) Coalesce with sibling

- Delete 50

n=4



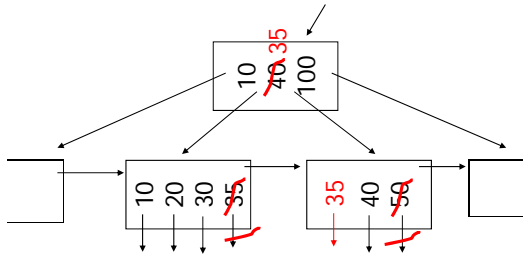
UB CSE 562 Spring 2009

72

(c) Redistribute keys

- Delete 50

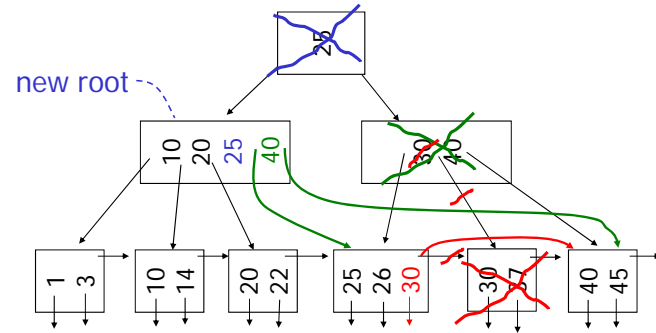
n=4



(d) Non-leaf coalesce

- Delete 37

n=4



B+ Tree deletions in practice

- Often, coalescing is not implemented
 - Too hard and not worth it!

Comparison: B-Trees vs. static indexed sequential file

Ref #1: Held & Stonebraker
"B-Trees Re-examined"
CACM, Feb. 1978

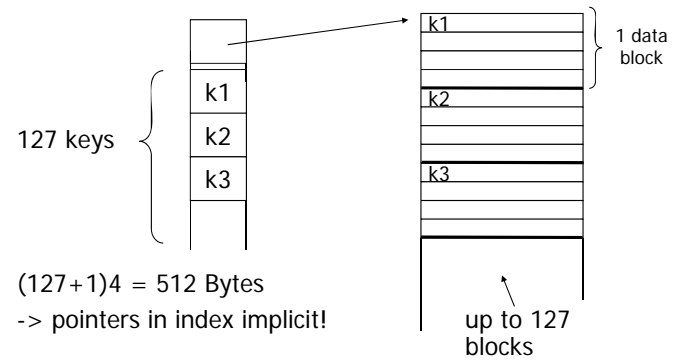
Ref # 1 claims:

- Concurrency control harder in B-Trees
- B-Tree consumes more space

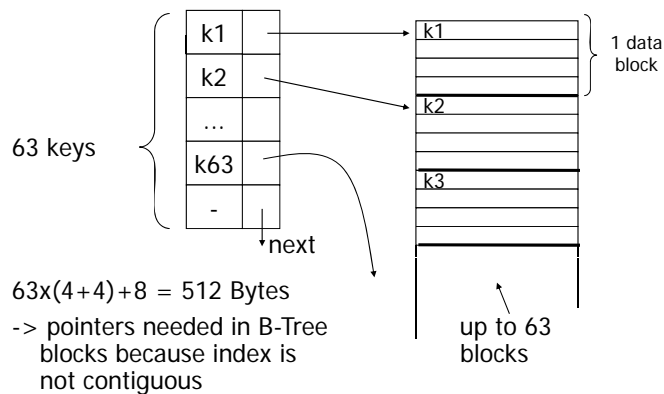
For their comparison:

block = 512 bytes
 key = pointer = 4 bytes
 4 data records per block

Example: 1 block static index



Example: 1 block B-Tree



Size comparison

Ref. #1

<u>Static Index</u>		<u>B-Tree</u>	
# data blocks	height	# data blocks	height
2 -> 127	2	2 -> 63	2
128 -> 16,129	3	64 -> 3,968	3
16,130 -> 2,048,383	4	3,969 -> 250,047	4
		250,048 -> 15,752,961	5

Ref. #1 analysis claims

- For an 8,000 block file,
 { after 32,000 inserts
 { after 16,000 lookups
⇒ Static index saves enough accesses
to allow for reorganization

Ref. #1 conclusion → Static index better!!

Ref #2: M. Stonebraker,
"Retrospective on a database
system," TODS, June 1980

Ref. #2 conclusion → B-Trees better!!

Ref. #2 conclusion → B-Trees better!!

- DBA does not know when to reorganize
- DBA does not know how full to load
pages of new index

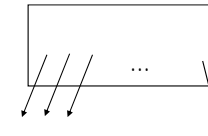
Ref. #2 conclusion → B-Trees better!!

- Buffering
 - B-Tree: has fixed buffer requirements
 - Static index: must read several overflow
blocks to be efficient
(large & variable size
buffers needed for this)

- Speaking of buffering...
 - Is LRU a good policy for B+Tree buffers?
- Of course not!
- Should try to keep root in memory at all times
 - (and perhaps some nodes from second level)

Interesting problem:

For B+Tree, how large should n be?



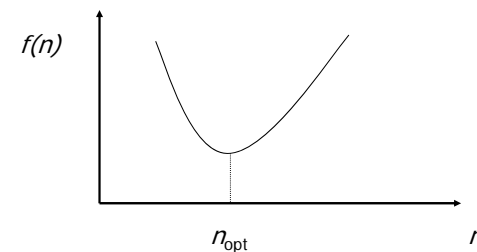
n is number of keys / node

Sample assumptions:

- (1) Time to read node from disk is $(S+Tn)$ msec.
- (2) Once block in memory, use binary search to locate key:
 - $(a + b \text{LOG}_2 n)$ msec.
 - For some constants a, b ; Assume $a \ll S$
- (3) Assume B+Tree is full, i.e.,
 - # nodes to examine is $\text{LOG}_n N$
 - where $N = \#$ records

Can get:

$f(n) = \text{time to find a record}$



✎ FIND n_{opt} by $f'(n) = 0$

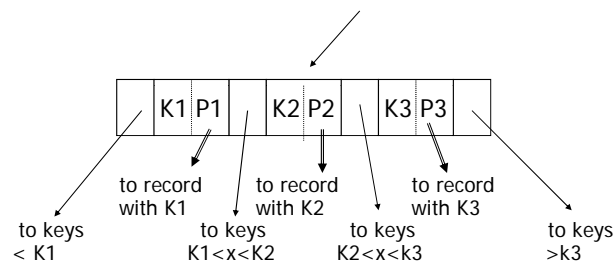
Answer should be $n_{\text{opt}} = \text{"few hundred"}$

✎ What happens to n_{opt} as

- Disk gets faster?
- CPU get faster?

Variation on B+ Tree: B-Tree (no +)

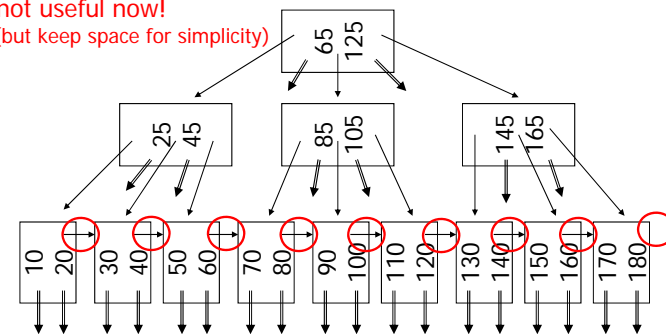
- Idea:
 - Avoid duplicate keys
 - Have record pointers in non-leaf nodes



B-Tree example

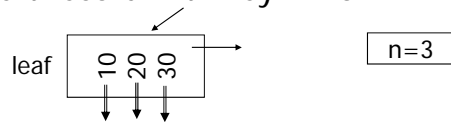
$n=2$

- sequence pointers
not useful now!
(but keep space for simplicity)

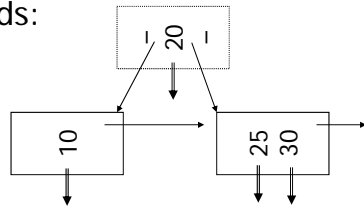


Note on inserts

- Say we insert record with key = 25



- Afterwards:



UB CSE 562 Spring 2009

93

So, for B-Trees:

	MAX			MIN		
	Tree Ptrs	Rec Ptrs	Keys	Tree Ptrs	Rec Ptrs	Keys
Non-leaf non-root	$n+1$	n	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$	$\lceil (n+1)/2 \rceil - 1$
Leaf non-root	1	n	n	1	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root non-leaf	$n+1$	n	n	2	1	1
Root Leaf	1	n	n	1	1	1

UB CSE 562 Spring 2009

94

Tradeoffs:

- ☺ B-Trees have faster lookup than B+ Trees
 - ☹ in B-Tree, non-leaf & leaf different sizes
 - ☹ in B-Tree, smaller fan-out
 - ☹ in B-Tree, deletion more complicated
- ➔ B+ Trees preferred!

UB CSE 562 Spring 2009

95

But note:

- If blocks are fixed size
(due to disk and buffering restrictions)
- Then lookup for B+ Tree is actually better!!

UB CSE 562 Spring 2009

96

Example:

- Pointers 4 bytes
- Keys 4 bytes
- Blocks 100 bytes (just example)
- Look at full 2 level tree

B-Tree:

Root has 8 keys + 8 record pointers
+ 9 son pointers
= $8 \times 4 + 8 \times 4 + 9 \times 4 = 100$ bytes

Each of 9 sons: 12 rec. pointers (+12 keys)
= $12 \times (4+4) + 4 = 100$ bytes

2-level B-Tree, Max # records =
 $12 \times 9 + 8 = 116$

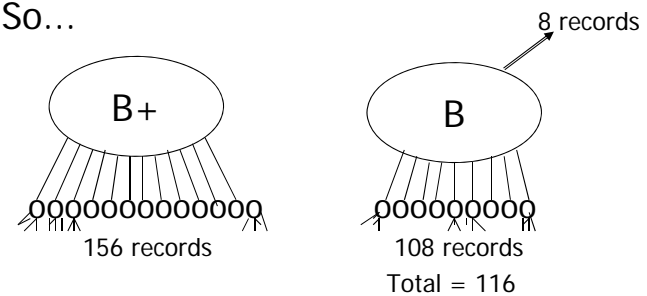
B+ Tree:

Root has 12 keys + 13 son pointers
= $12 \times 4 + 13 \times 4 = 100$ bytes

Each of 13 sons: 12 rec. ptrs (+12 keys)
= $12 \times (4 + 4) + 4 = 100$ bytes

2-level B+ Tree, Max # records
= $13 \times 12 = 156$

So...



- Conclusion:
 - For fixed block size,
 - B+ Tree is better because it is bushier

Topics

- Conventional Indexes
 - Chapter 14: 14.1
 - Sparse vs. dense
 - Primary vs. secondary
- B-Trees
 - Chapter 14: 14.2
 - B+Trees vs. B-Trees vs. indexed sequential
- Hashing schemes --> Next