


CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture#9: Indexing (R&G ch. 10)




CMU SCS

Outline

- Motivation
- ISAM
- B-trees (not in book)
- B+ trees
- duplicates
- B+ trees in practice

Faloutsos CMU SCS 15-415 2



CMU SCS

Introduction

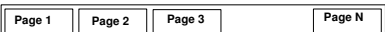
- How to support range searches?
- equality searches?

Faloutsos CMU SCS 15-415 3

CMU SCS

Range Searches

- ``Find all students with $\text{gpa} > 3.0$ ``
- may be slow, even on sorted file
- What to do?



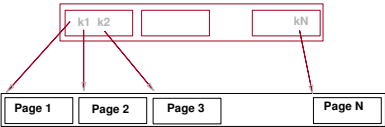
Data File

Faloutsos CMU SCS 15-415 4

CMU SCS

Range Searches

- ``Find all students with $\text{gpa} > 3.0$ ``
- may be slow, even on sorted file
- Solution: Create an `index` file.



Index File

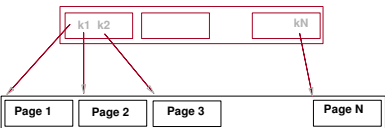
Data File

Faloutsos CMU SCS 15-415 5

CMU SCS

Range Searches

- More details:
- if index file is small, do binary search there
- Otherwise??



Index File

Data File

Faloutsos CMU SCS 15-415 6

CMU SCS

ISAM

- Repeat recursively!

Faloutsos CMU SCS 15-415 7

CMU SCS

ISAM

- OK - what if there are insertions and overflows?

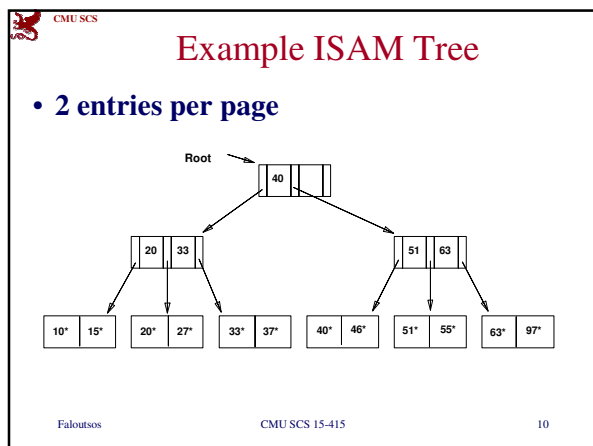
Faloutsos CMU SCS 15-415 8

CMU SCS

ISAM

- Overflow pages, linked to the primary page

Faloutsos CMU SCS 15-415 9



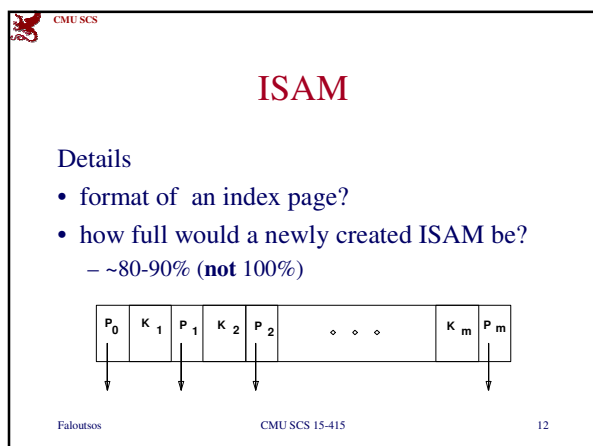
CMU SCS


ISAM

Details

- format of an index page?
- how full would a newly created ISAM be?

Faloutsos CMU SCS 15-415 11






CMU SCS

ISAM is a STATIC Structure

- that is, index pages don't change
- *File creation*: Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then overflow pgs.

Faloutsos CMU SCS 15-415 13




CMU SCS

ISAM is a STATIC Structure

- *Search*: Start at root; use key comparisons to go to leaf.
- $\text{Cost} = \log_F N$;
- $F = \# \text{ entries/pg (i.e., fanout)}$,
- $N = \# \text{ leaf pgs}$

Faloutsos CMU SCS 15-415 14



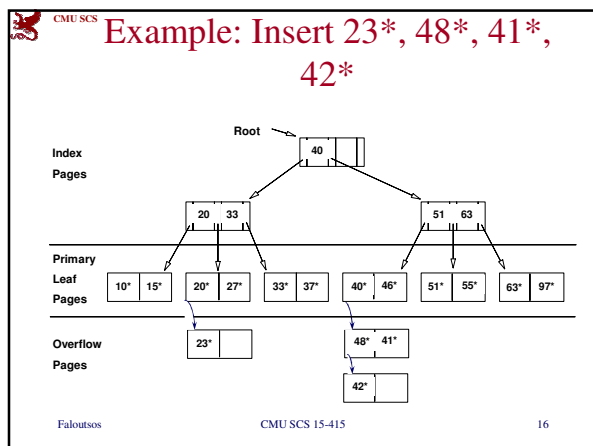
CMU SCS

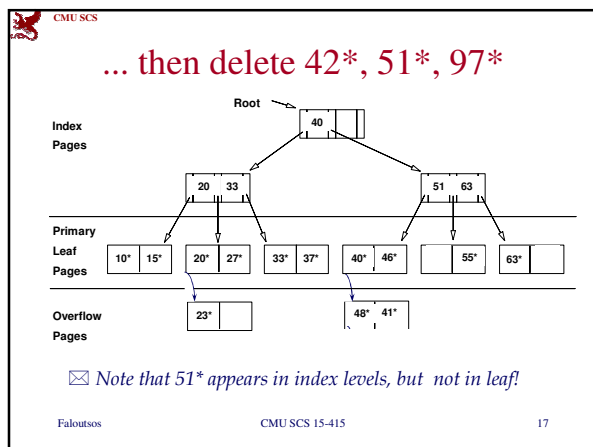
ISAM is a STATIC Structure

Insert: Find leaf that data entry belongs to, and put it there. Overflow page if necessary.

Delete: Find and remove from leaf; if empty page, de-allocate.

Faloutsos CMU SCS 15-415 15





ISAM ---- Issues?

- Pros
 - ????
- Cons
 - ????

Faloutsos CMU SCS 15-415 18

CMU SCS

Outline

- Motivation
- ISAM
- **B-trees (not in book)**
- B+ trees
- duplicates
- B+ trees in practice

Faloutsos CMU SCS 15-415 19

CMU SCS

B-trees


- the **most successful** family of index schemes (B-trees, B⁺-trees, B^{*}-trees)
- Can be used for primary/secondary, clustering/non-clustering index.
- balanced “n-way” search trees

Faloutsos CMU SCS 15-415 20

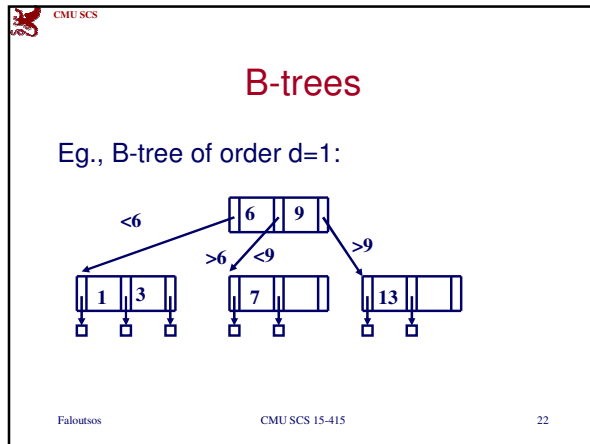
CMU SCS

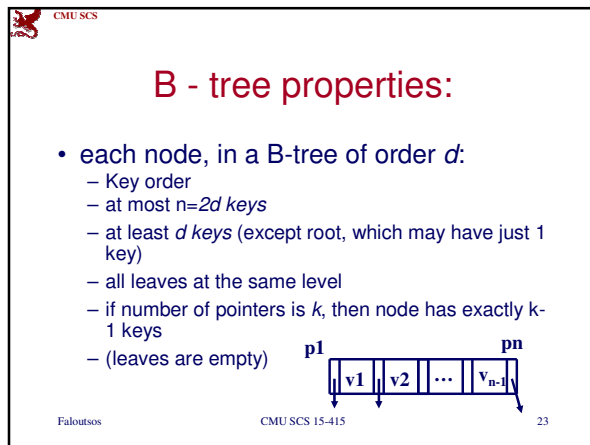
B-trees

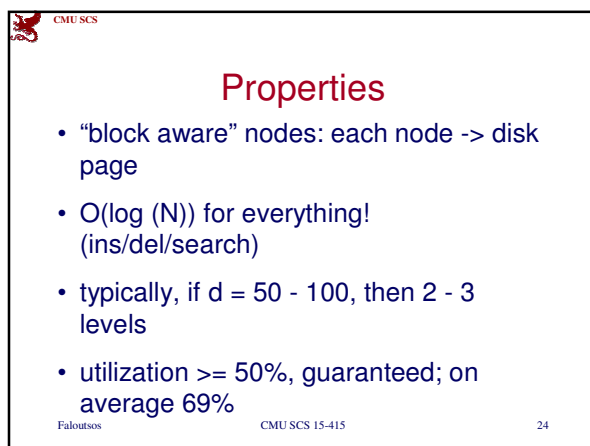
[Rudolf Bayer and McCreight, E. M.
Organization and Maintenance of Large
Ordered Indexes. Acta Informatica 1, 173-
189, 1972.]



Faloutsos CMU SCS 15-415 21







CMU SCS

Queries

- Algo for exact match query? (eg., ssn=8?)

Faloutsos CMU SCS 15-415 25

CMU SCS

JAVA animation!

<http://slady.net/java/bt/>

strongly recommended!

Faloutsos CMU SCS 15-415 26

CMU SCS

Queries

- Algo for exact match query? (eg., ssn=8?)

Faloutsos CMU SCS 15-415 27

CMU SCS

Queries

- Algo for exact match query? (eg., $ssn=8$?)

Faloutsos CMU SCS 15-415 28

CMU SCS

Queries

- Algo for exact match query? (eg., $ssn=8$?)

Faloutsos CMU SCS 15-415 29

CMU SCS

Queries

- Algo for exact match query? (eg., $ssn=8$?)

Faloutsos CMU SCS 15-415 30

CMU SCS

Queries

- what about range queries? (eg., $5 < \text{salary} < 8$)
- Proximity/ nearest neighbor searches? (eg., $\text{salary} \sim 8$)

Faloutsos CMU SCS 15-415 31

CMU SCS

Queries

- what about range queries? (eg., $5 < \text{salary} < 8$)
- Proximity/ nearest neighbor searches? (eg., $\text{salary} \sim 8$)

Faloutsos CMU SCS 15-415 32

CMU SCS

Queries

- what about range queries? (eg., $5 < \text{salary} < 8$)
- Proximity/ nearest neighbor searches? (eg., $\text{salary} \sim 8$)

Faloutsos CMU SCS 15-415 33

Queries

- what about range queries? (eg., $5 < \text{salary} < 8$)
- Proximity/ nearest neighbor searches? (eg., **salary** ~ 8)

Faloutsos CMU SCS 15-415 34

Queries

- what about range queries? (eg., $5 < \text{salary} < 8$)
- Proximity/ nearest neighbor searches? (eg., **salary** ~ 8)

Faloutsos CMU SCS 15-415 35

B-trees: Insertion

- Insert in leaf; on overflow, push middle up (recursively)
- split: preserves B - tree properties

Faloutsos CMU SCS 15-415 36

CMU SCS

B-trees

Easy case: Tree T0; insert '8'

Faloutsos CMU SCS 15-415 37

CMU SCS

B-trees

Tree T0; insert '8'

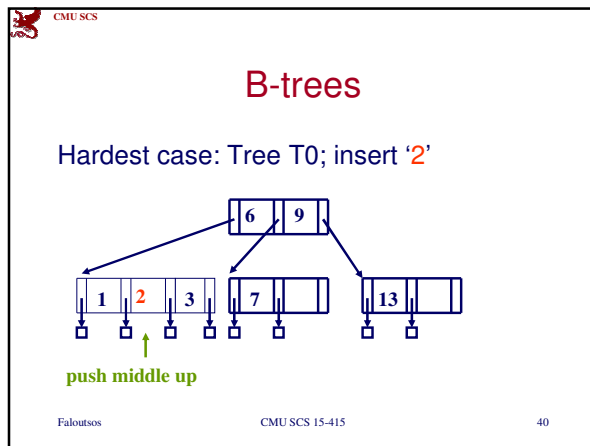
Faloutsos CMU SCS 15-415 38

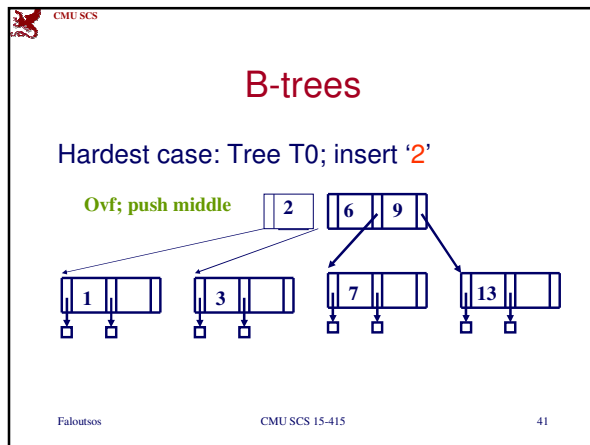
CMU SCS

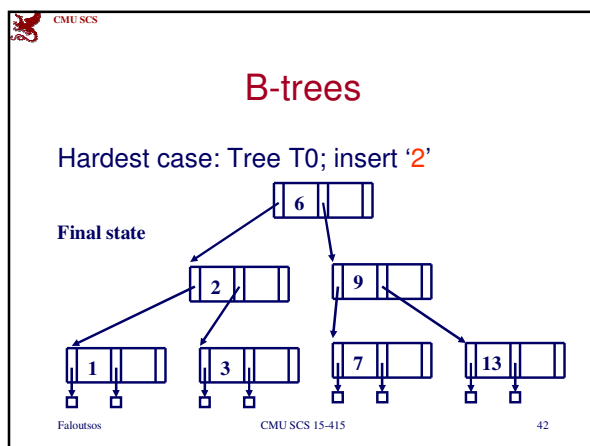
B-trees


Hardest case: Tree T0; insert '2'

Faloutsos CMU SCS 15-415 39










CMU SCS

B-trees: Insertion

- Insert in leaf; on overflow, push middle up (recursively – 'propagate split')
- split: preserves all B - tree properties (!!)
- notice how it grows: height increases when root overflows & splits
- Automatic, incremental re-organization (contrast with ISAM!)

Faloutsos CMU SCS 15-415 43



CMU SCS


Pseudo-code

```

INSERTION OF KEY 'K'
  find the correct leaf node 'L';
  if ( 'L' overflows ){
    split 'L', and push middle key to parent node 'P';
    if ('P' overflows){
      repeat the split recursively; }
    else{
      add the key 'K' in node 'L';
      /* maintaining the key order in 'L' */ }
  }

```

Faloutsos CMU SCS 15-415 44



CMU SCS

Overview

- ...
- B – trees
 - Dfn, Search, insertion, **deletion**
- ...

Faloutsos CMU SCS 15-415 45

CMU SCS

Deletion

Rough outline of algo:

- Delete key;
- on underflow, may need to merge

In practice, some implementors just allow underflows to happen...

Faloutsos CMU SCS 15-415 46

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

Faloutsos CMU SCS 15-415 47

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

Faloutsos CMU SCS 15-415 48

CMU SCS

B-trees – Deletion

- Case1: delete a key at a leaf – no underflow
- Case2: delete non-leaf key – no underflow
- Case3: delete leaf-key; underflow, and 'rich sibling'
- Case4: delete leaf-key; underflow, and 'poor sibling'

Faloutsos CMU SCS 15-415 49

CMU SCS

B-trees – Deletion

- Case1: delete a key at a leaf – no underflow (delete 3 from T0)

Faloutsos CMU SCS 15-415 50

CMU SCS

B-trees – Deletion

- Case2: delete a key at a non-leaf – no underflow (eg., delete 6 from T0)

Faloutsos CMU SCS 15-415 51

B-trees – Deletion

- Case2: delete a key at a non-leaf – no underflow (eg., delete 6 from T0)

Delete & promote, ie:

Faloutsos CMU SCS 15-415 52

B-trees – Deletion

- Case2: delete a key at a non-leaf – no underflow (eg., delete 6 from T0)

Delete & promote, ie:

Faloutsos CMU SCS 15-415 53

B-trees – Deletion

- Case2: delete a key at a non-leaf – no underflow (eg., delete 6 from T0)

FINAL TREE

Faloutsos CMU SCS 15-415 54

B-trees – Deletion

- Case2: delete a key at a non-leaf – no underflow (eg., delete 6 from T0)
- Q: How to promote?
- A: pick the largest key from the left sub-tree (or the smallest from the right sub-tree)
- Observation: every deletion eventually becomes a deletion of a leaf key

Faloutsos CMU SCS 15-415 55

B-trees – Deletion

- Case1: delete a key at a leaf – no underflow
- Case2: delete non-leaf key – no underflow
- ⇒ • Case3: delete leaf-key; underflow, and 'rich sibling'
- Case4: delete leaf-key; underflow, and 'poor sibling'

Faloutsos CMU SCS 15-415 56

B-trees – Deletion

- Case3: underflow & 'rich sibling' (eg., delete 7 from T0)

Delete & borrow, ie:

Faloutsos CMU SCS 15-415 57

B-trees – Deletion

- Case3: underflow & 'rich sibling' (eg., delete 7 from T0)

Rich sibling

Delete & borrow, ie:

Faloutsos CMU SCS 15-415 58

B-trees – Deletion

- Case3: underflow & 'rich sibling'
- 'rich' = can give a key, without underflowing
- 'borrowing' a key: THROUGH the PARENT!

Faloutsos CMU SCS 15-415 59

B-trees – Deletion

- Case3: underflow & 'rich sibling' (eg., delete 7 from T0)

Rich sibling

Delete & borrow, ie:

NO!!

Faloutsos CMU SCS 15-415 60

B-trees – Deletion

- Case3: underflow & 'rich sibling' (eg., delete 7 from T0)

Delete & borrow, ie:

Faloutsos CMU SCS 15-415 61

B-trees – Deletion

- Case3: underflow & 'rich sibling' (eg., delete 7 from T0)

Delete & borrow, ie:

Faloutsos CMU SCS 15-415 62

B-trees – Deletion

- Case3: underflow & 'rich sibling' (eg., delete 7 from T0)

FINAL TREE

Delete & borrow, through the parent

Faloutsos CMU SCS 15-415 63

CMU SCS

B-trees – Deletion

- Case1: delete a key at a leaf – no underflow
- Case2: delete non-leaf key – no underflow
- Case3: delete leaf-key; underflow, and 'rich sibling'
- ⇒ • Case4: delete leaf-key; underflow, and 'poor sibling'

Faloutsos CMU SCS 15-415 64

CMU SCS

B-trees – Deletion

- Case4: underflow & 'poor sibling' (eg., delete 13 from T0)

Faloutsos CMU SCS 15-415 65

CMU SCS

B-trees – Deletion

- Case4: underflow & 'poor sibling' (eg., delete 13 from T0)

Faloutsos CMU SCS 15-415 66

B-trees – Deletion

- Case4: underflow & 'poor sibling' (eg., delete 13 from T0)

Faloutsos CMU SCS 15-415 67

B-trees – Deletion

- Case4: underflow & 'poor sibling' (eg., delete 13 from T0)
- Merge, by pulling a key from the **parent**
- exact reversal from insertion: 'split and push up', vs. 'merge and pull down'
- ie.:

Faloutsos CMU SCS 15-415 68

B-trees – Deletion

- Case4: underflow & 'poor sibling' (eg., delete 13 from T0)

Faloutsos CMU SCS 15-415 69

CMU SCS

B-trees – Deletion

- Case4: underflow & 'poor sibling' (eg., delete 13 from T0)

FINAL TREE

Faloutsos CMU SCS 15-415 70

CMU SCS

B-trees – Deletion

- Case4: underflow & 'poor sibling'
- > 'pull key from parent, and merge'
- Q: What if the parent underflows?
- A: repeat recursively

Faloutsos CMU SCS 15-415 71

CMU SCS

B-tree deletion - pseudocode

```

DELETION OF KEY 'K'
  locate key 'K', in node 'N'
  if( 'N' is a non-leaf node) {
    delete 'K' from 'N';
    find the immediately largest key 'K1';
    /* which is guaranteed to be on a leaf node 'L' */
    copy 'K1' in the old position of 'K';
    invoke this DELETION routine on 'K1' from the leaf node 'L';
  }
  else {
    /* 'N' is a leaf node */
    ... (next slide..)
  }

```

Faloutsos CMU SCS 15-415 72



B-tree deletion - pseudocode

```

/* 'N' is a leaf node */
if( 'N' underflows ){
    let 'N1' be the sibling of 'N';
    if( 'N1' is "rich" ){ /* ie., N1 can lend us a key */
        borrow a key from 'N1' THROUGH the parent node;
    } else { /* N1 is 1 key away from underflowing */
        MERGE: pull the key from the parent 'P',
        and merge it with the keys of 'N' and 'N1' into a new
        node;
        if( 'P' underflows ){ repeat recursively }
    }
}

```

Faloutsos

CMU SCS 15-415

73



Outline

- Motivation
- ISAM
- B-trees (not in book)
 - algorithms
 - extensions
- B+ trees
- duplicates
- B+ trees in practice

Faloutsos

CMU SCS 15-415

74



Variations

- How could we do even better than the B-trees above?

Faloutsos

CMU SCS 15-415

75

B*-tree

- In B-trees, worst case util. = 50%, if we have just split all the pages
- how to increase the utilization of B - trees?
- ..with B* - trees!

Faloutsos CMU SCS 15-415 76

B-trees and B*-trees

Eg., Tree T0; insert '2'

Faloutsos CMU SCS 15-415 77

B*-trees: deferred split!

- Instead of splitting, LEND keys to sibling! (through PARENT, of course!)

Faloutsos CMU SCS 15-415 78

B*-trees: deferred split!

- Instead of splitting, LEND keys to sibling!
(through PARENT, of course!)

FINAL TREE

Faloutsos CMU SCS 15-415 79

B*-trees: deferred split!

- Notice: shorter, more packed, faster tree
- It's a rare case, where space utilization and speed improve together
- BUT: What if the sibling has no room for our 'lending'?

Faloutsos CMU SCS 15-415 80

B*-trees: deferred split!

- A: 2-to-3 split: get the keys from the sibling, pool them with ours (and a key from the parent), and split in 3.
- Could we extend the idea to 3-to-4 split, 4-to-5 etc?

Faloutsos CMU SCS 15-415 81



B*-trees: deferred split!

- A: 2-to-3 split: get the keys from the sibling, pool them with ours (and a key from the parent), and split in 3.
- Could we extend the idea to 3-to-4 split, 4-to-5 etc?
- Yes, but: diminishing returns

Faloutsos

CMU SCS 15-415

82



Outline

- Motivation
- ISAM
- B-trees (not in book)
- B+ trees
- duplicates
- B+ trees in practice

Faloutsos

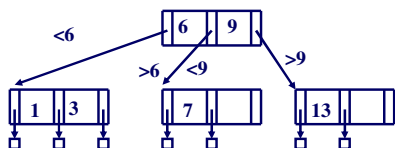
CMU SCS 15-415

83



B+ trees - Motivation

B-tree – print keys in sorted order:



Faloutsos

CMU SCS 15-415

84

B+ trees - Motivation

B-tree needs back-tracking – how to avoid it?

Faloutsos CMU SCS 15-415 85

B+ trees - Motivation

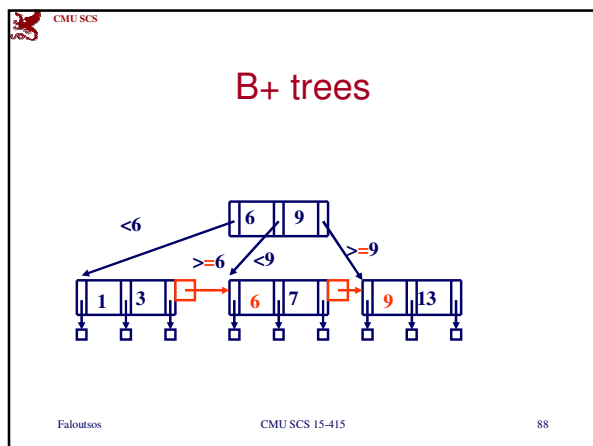
Stronger reason: for clustering index, data records are scattered:

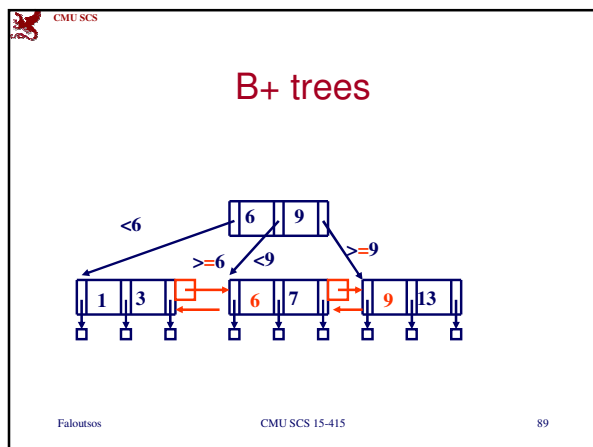
Faloutsos CMU SCS 15-415 86

Solution: B⁺ - trees

- facilitate sequential ops
- They string all leaf nodes together
- AND
- replicate keys from non-leaf nodes, to make sure every key appears at the leaf level
- (vital, for clustering index!)

Faloutsos CMU SCS 15-415 87





CMU SCS

B+ trees

- More details: next (and textbook)
- In short: on split
 - at leaf level: **COPY** middle key upstairs
 - at non-leaf level: push middle key upstairs (as in plain B-tree)

Faloutsos CMU SCS 15-415 90

Example B+ Tree

- Search begins at root, and key comparisons direct it to a leaf (as in ISAM).
- Search for 5*, 15*, all data entries $\geq 24^*$
- ...

Based on the search for 15, we know it is not in the tree!*

Faloutsos CMU SCS 15-415 91

B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
 - average fanout = $2 * 100 * 0.67 = 134$
- Typical capacities:
 - Height 4: $133^4 = 312,900,721$ entries
 - Height 3: $133^3 = 2,406,104$ entries

Faloutsos CMU SCS 15-415 92

B+ Trees in Practice

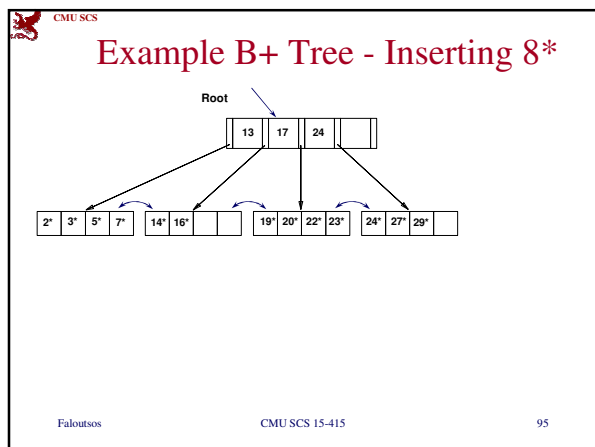
- Can often keep top levels in buffer pool:
 - Level 1 = 1 page = 8 KB
 - Level 2 = 134 pages = 1 MB
 - Level 3 = 17,956 pages = 140 MB

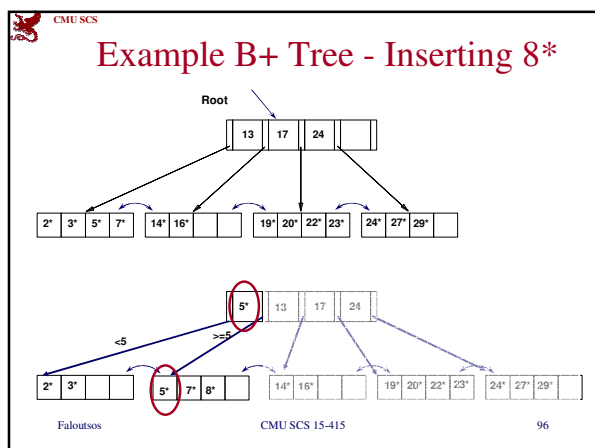
Faloutsos CMU SCS 15-415 93

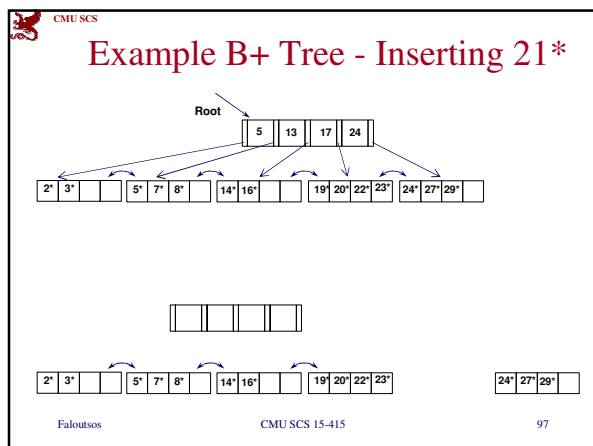
Inserting a Data Entry into a B+ Tree

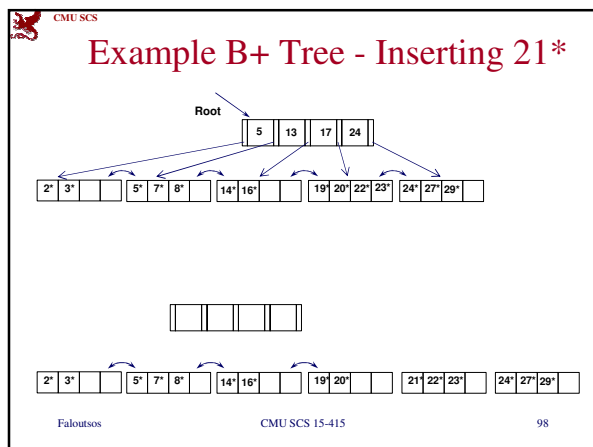
- Find correct leaf L .
- Put data entry onto L .
 - If L has enough space, *done!*
 - Else, must *split* L (into L and a new node $L2$)
 - Redistribute entries evenly, *copy up* middle key.
- parent node may overflow
 - but then: *push up* middle key. Splits “grow” tree; root split increases height.

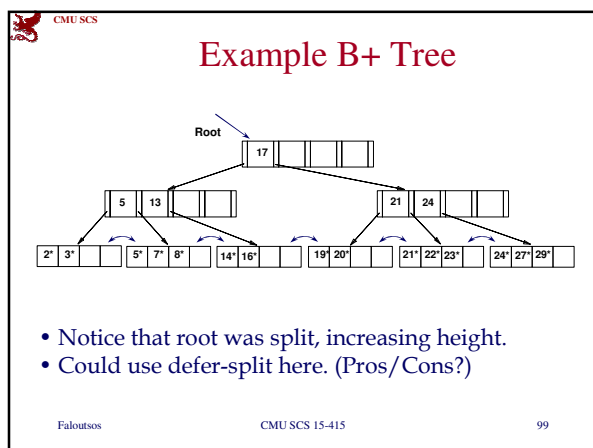
Faloutsos CMU SCS 15-415 94














CMU SCS

Example: Data vs. Index Page Split

- leaf: 'copy'
- non-leaf: 'push'
- why not 'copy' @ non-leaves?

2*

3*

5*

7*

8*

5

5

13

17

21

24

5

13

17


21

24

Faloutsos

CMU SCS 15-415

100



CMU SCS

Now you try...

Root

30

5

13

20

2*

3*

5*

7*

8*

11*

14*

16*

21*

22*


23*

Insert the following data entries (in order): 28*, 6*, 25*

Faloutsos

CMU SCS 15-415

101



CMU SCS

Answer...

After inserting 28*, 6*

30

5

7

13

20

2*

3*

5*

6*

7*

8*

11*

14*

16*

21*

22*

23*

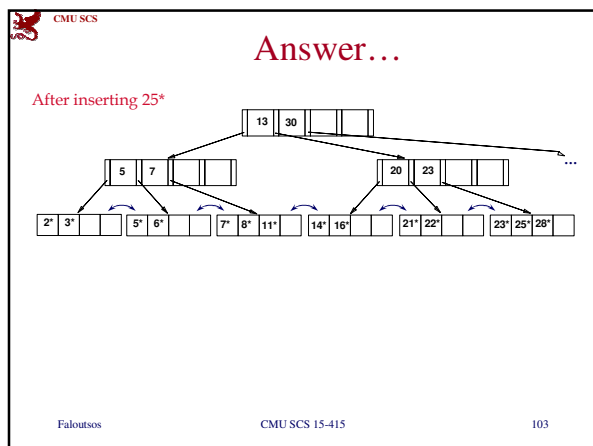
28*

After inserting 25*

Faloutsos

CMU SCS 15-415

102

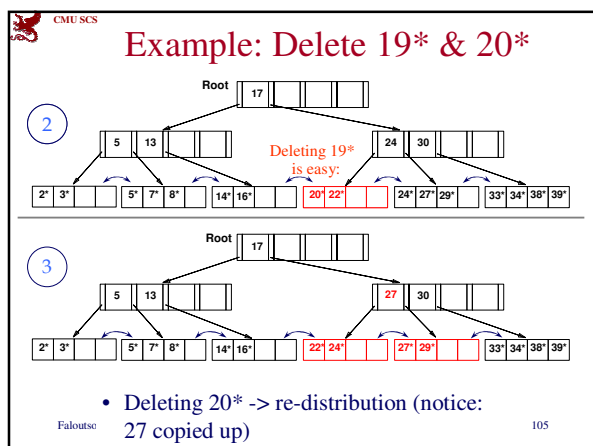


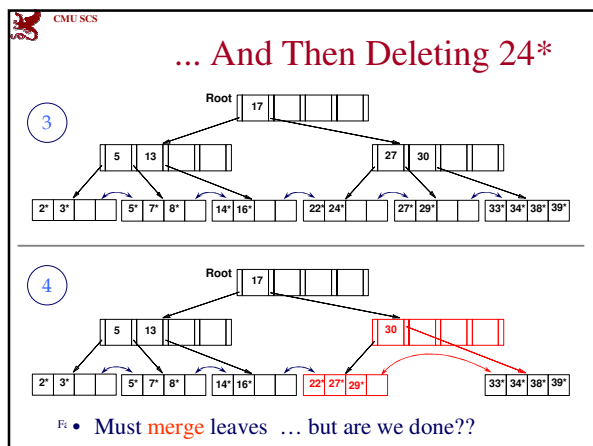
CMU SCS

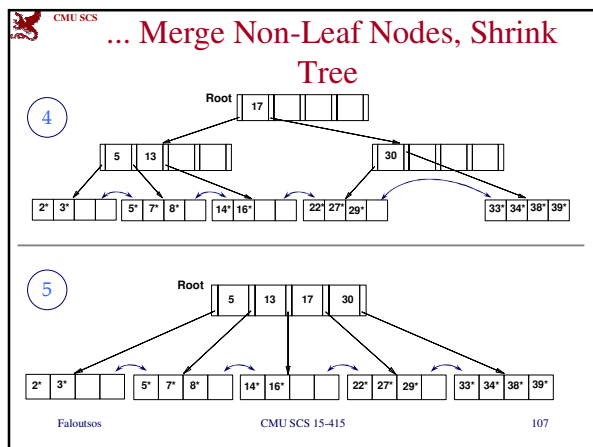
Deleting a Data Entry from a B+ Tree

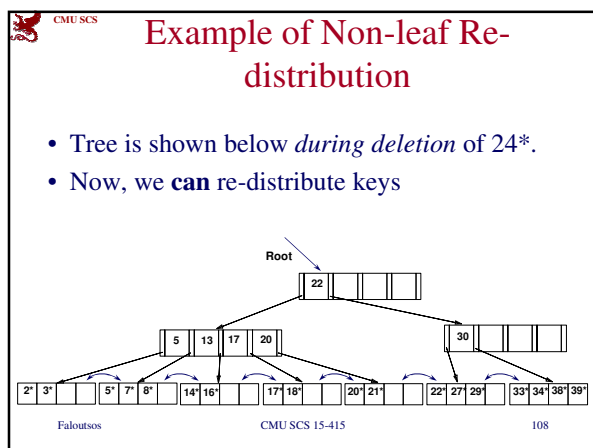
- Start at root, find leaf L where entry belongs.
- Remove the entry.
 - If L is at least half-full, *done!*
 - If L underflows
 - Try to **re-distribute**, borrowing from *sibling* (adjacent node with same parent as L).
 - If re-distribution fails, **merge** L and sibling.
 - update parent
 - and possibly merge, recursively

Faloutsos CMU SCS 15-415 104









CMU SCS

After Re-distribution

- need only re-distribute '20'; did '17', too
- why would we want to re-distributed more keys?

Faloutsos CMU SCS 15-415 109

CMU SCS

Main observations for deletion

- If a key value appears twice (leaf + nonleaf), the above algorithms delete it from the leaf, only
- why not non-leaf, too?


Faloutsos CMU SCS 15-415 110

CMU SCS

Main observations for deletion

- If a key value appears twice (leaf + nonleaf), the above algorithms delete it from the leaf, only
- why not non-leaf, too?
- 'lazy deletions' - in fact, some vendors just mark entries as deleted (~ underflow),
– and reorganize/compact later

Faloutsos CMU SCS 15-415 111




CMU SCS

Recap: main ideas

- on overflow, split (and ‘push’, or ‘copy’)
 - or consider deferred split
- on underflow, borrow keys; or merge
 - or let it underflow...

Faloutsos CMU SCS 15-415 112




CMU SCS

Outline

- Motivation
- ISAM
- B-trees (not in book)
- B+ trees
- duplicates
- B+ trees in practice
 - prefix compression; bulk-loading; ‘order’

Faloutsos CMU SCS 15-415 113



CMU SCS

B+ trees with duplicates

- Everything so far: assumed unique key values
- How to extend B+-trees for duplicates?
 - Alt. 2: <key, rid>
 - Alt. 3: <key, {rid list}>
- 2 approaches, roughly equivalent

Faloutsos CMU SCS 15-415 114

CMU SCS

B+ trees with duplicates

- approach#1: repeat the key values, and extend B+ tree algo's appropriately - eg. many '14's

Faloutsos CMU SCS 15-415 115

CMU SCS

B+ trees with duplicates

- approach#1: subtle problem with deletion:
- treat *rid* as part of the key, thus making it unique

Faloutsos CMU SCS 15-415 116

CMU SCS

B+ trees with duplicates

- approach#2: store each key value: once
- but store the {rid list} as variable-length field (and use overflow pages, if needed)

Faloutsos CMU SCS 15-415 117

CMU SCS

Outline


- Motivation
- ISAM
- B-trees (not in book)
- B+ trees
- duplicates
- B+ trees in practice
 - prefix compression: bulk-loading; ‘order’

Faloutsos CMU SCS 15-415 118

CMU SCS

Prefix Key Compression

- Important to increase fan-out. (Why?)
- Key values in index entries only ‘direct traffic’; can often compress them.




Faloutsos CMU SCS 15-415 119

CMU SCS

Prefix Key Compression

- Important to increase fan-out. (Why?)
- Key values in index entries only ‘direct traffic’; can often compress them.



Faloutsos CMU SCS 15-415 120

CMU SCS

Bulk Loading of a B+ Tree

- In an empty tree, insert many keys
- Why not one-at-a-time?

Faloutsos CMU SCS 15-415 121

CMU SCS

Bulk Loading of a B+ Tree

- *Initialization:* Sort all data entries
- scan list; whenever enough for a page, pack
- <repeat for upper level - even faster than book's algo>


Faloutsos CMU SCS 15-415 122

CMU SCS

Bulk Loading (Contd.)

- Book's algo
- (any problems?)

Faloutsos




CMU SCS

Outline

- Motivation
- ISAM
- B-trees (not in book)
- B+ trees
- duplicates
- B+ trees in practice
 - prefix compression; bulk-loading; 'order'

Faloutsos CMU SCS 15-415 124




CMU SCS

A Note on 'Order'

- *Order (d)* concept replaced by physical space criterion in practice (*at least half-full*).
- Why do we need it?
 - Index pages can typically hold many more entries than leaf pages.
 - Variable sized records and search keys mean different nodes will contain different numbers of entries.
 - Even with fixed length fields, multiple records with the same search key value (*duplicates*) can lead to variable-sized data entries (if we use Alternative (3)).

Faloutsos CMU SCS 15-415 125




CMU SCS

A Note on 'Order'

- Many real systems are even sloppier than this: they allow underflow, and only reclaim space when a page is *completely* empty.
- (what are the benefits of such 'sloppiness'?)

Faloutsos CMU SCS 15-415 126




CMU SCS

Conclusions

- B+tree is the prevailing indexing method
- **Excellent**, $O(\log N)$ worst-case performance for ins/del/search; (~3-4 disk accesses in practice)
- guaranteed 50% space utilization; avg 69%

Faloutsos CMU SCS 15-415 127



CMU SCS

Conclusions

- Can be used for any type of index: primary/secondary, sparse (clustering), or dense (non-clustering)
- Several fine-extensions on the basic algorithm
 - deferred split; prefix compression; (underflows)
 - bulk-loading
 - duplicate handling

Faloutsos CMU SCS 15-415 128
