
 CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 – DB 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 15-415/615 2

 CMU SCS

Introduction

- How to support range searches?
- equality searches?

Faloutsos CMU SCS 15-415/615 3

CMU SCS

Range Searches

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

Data File

Faloutsos CMU SCS 15-415/615 4

CMU SCS

Range Searches

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

Index File

Data File

Faloutsos CMU SCS 15-415/615 5

CMU SCS

Range Searches

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

Index File

Data File

Faloutsos CMU SCS 15-415/615 6

CMU SCS

ISAM

- Repeat recursively!

Non-leaf Pages

Leaf Pages

Faloutsos CMU SCS 15-415/615 7

CMU SCS

ISAM

- OK - what if there are insertions and overflows?

Non-leaf Pages

Leaf Pages

Faloutsos CMU SCS 15-415/615 8

CMU SCS

ISAM

- Overflow pages, linked to the primary page

Non-leaf Pages

Leaf Pages

Overflow page

Primary pages

Faloutsos CMU SCS 15-415/615 9

CMU SCS

Example ISAM Tree

- 2 entries per page

Faloutsos CMU SCS 15-415/615 10

CMU SCS

ISAM

Details

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

Faloutsos CMU SCS 15-415/615 11

CMU SCS

ISAM

Details

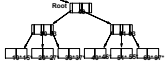
- format of an index page?
- how full would a newly created ISAM be?
– ~80-90% (**not** 100%)

Faloutsos CMU SCS 15-415/615 12

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.

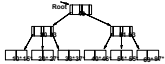


Faloutsos CMU SCS 15-415/615 13

CMU SCS

ISAM is a STATIC Structure

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



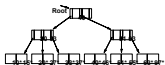
Faloutsos CMU SCS 15-415/615 14

CMU SCS

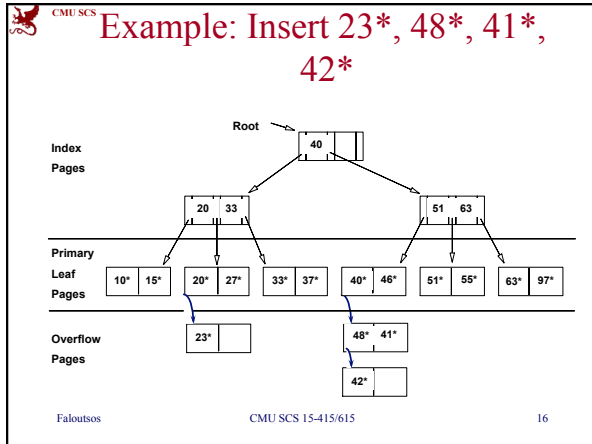
ISAM is a STATIC Structure

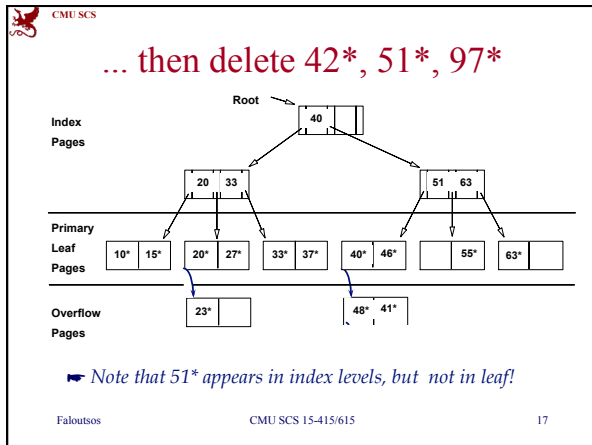
Insert: Find leaf that data entry belongs to, and put it there. Overflow page if necessary.

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



Faloutsos CMU SCS 15-415/615 15





ISAM ---- Issues?

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

CMU SCS

Outline

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

Faloutsos CMU SCS 15-415/615 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 15-415/615 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 15-415/615 21

CMU SCS

B-trees

Eg., B-tree of order $d=1$:

Faloutsos CMU SCS 15-415/615 22

CMU SCS

B - tree properties:

- each node, in a B-tree of order d :
 - Key order
 - at most $n=2d$ keys
 - at least d keys (except root, which may have just 1 key)
 - all leaves at the same level
 - if number of pointers is k , then node has exactly $k-1$ keys
 - (leaves are empty)

Faloutsos CMU SCS 15-415/615 23

CMU SCS

Properties

- “block aware” nodes: each node \rightarrow disk page
- $O(\log(N))$ for everything! (ins/del/search)
- typically, if $d = 50 - 100$, then 2 - 3 levels
- utilization $\geq 50\%$, guaranteed; on average 69%

Faloutsos CMU SCS 15-415/615 24

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 25

CMU SCS

JAVA animation!

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

strongly recommended! (with all usual precautions – VM etc)

Faloutsos CMU SCS 15-415/615 26

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 27

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 28

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 29

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 30

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 31

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 32

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 33

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 34

CMU SCS

Queries

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

Faloutsos CMU SCS 15-415/615 35

CMU SCS

B-trees: Insertion

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

Faloutsos CMU SCS 15-415/615 36

CMU SCS

B-trees

Easy case: Tree T₀; insert '8'

Faloutsos CMU SCS 15-415/615 37

CMU SCS

B-trees

Tree T₀; insert '8'

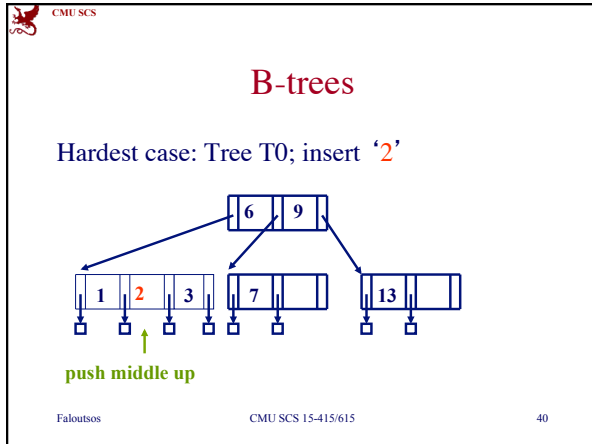
Faloutsos CMU SCS 15-415/615 38

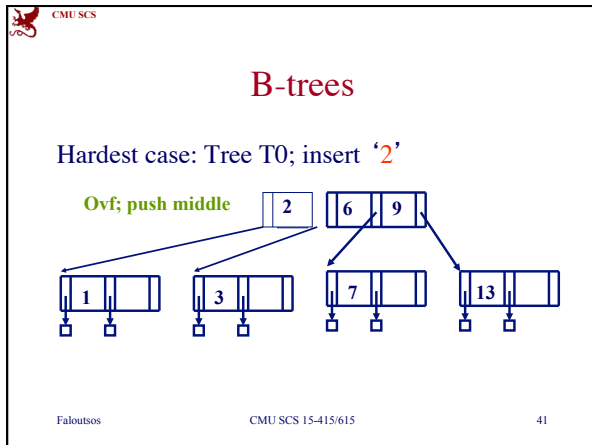
CMU SCS

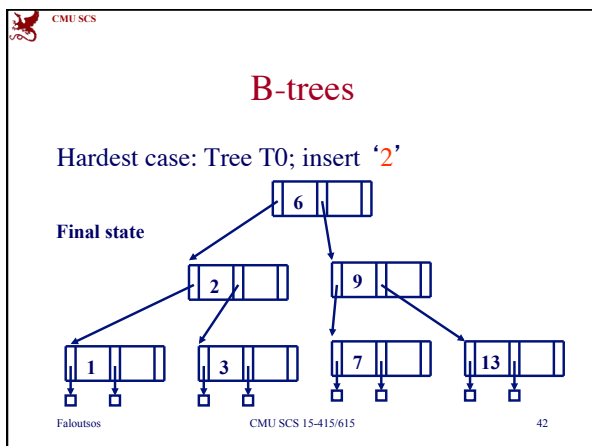
B-trees

Hardest case: Tree T₀; insert '2'

Faloutsos CMU SCS 15-415/615 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 15-415/615 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 15-415/615 44

CMU SCS

Overview

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

Faloutsos CMU SCS 15-415/615 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 15-415/615 46

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

Faloutsos CMU SCS 15-415/615 47

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

Faloutsos CMU SCS 15-415/615 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 15-415/615 49

CMU SCS

B-trees – Deletion

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

Faloutsos CMU SCS 15-415/615 50

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

Faloutsos CMU SCS 15-415/615 51

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

Faloutsos CMU SCS 15-415/615 52

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

Faloutsos CMU SCS 15-415/615 53

CMU SCS

B-trees – Deletion

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

FINAL TREE

Faloutsos CMU SCS 15-415/615 54

CMU SCS

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 15-415/615 55

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 15-415/615 56

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos CMU SCS 15-415/615 57

B-trees – Deletion

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

Rich sibling

Delete & borrow, ie:

Faloutsos CMU SCS 15-415/615 58

B-trees – Deletion

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

Faloutsos CMU SCS 15-415/615 59

B-trees – Deletion

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

Rich sibling

Delete & borrow, ie:

NO!!

Faloutsos CMU SCS 15-415/615 60

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos CMU SCS 15-415/615 61

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos CMU SCS 15-415/615 62

CMU SCS

B-trees – Deletion

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

FINAL TREE

Delete & borrow, through the parent

Faloutsos CMU SCS 15-415/615 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 15-415/615 64

CMU SCS

B-trees – Deletion

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

Faloutsos CMU SCS 15-415/615 65

CMU SCS

B-trees – Deletion

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

Faloutsos CMU SCS 15-415/615 66

CMU SCS

B-trees – Deletion

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

Faloutsos CMU SCS 15-415/615 67

CMU SCS

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 15-415/615 68

CMU SCS

B-trees – Deletion

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

Faloutsos CMU SCS 15-415/615 69

CMU SCS

B-trees – Deletion

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

FINAL TREE

Faloutsos CMU SCS 15-415/615 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 15-415/615 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 15-415/615 72

CMU SCS

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 15-415/615 73

CMU SCS

Outline

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

Faloutsos CMU SCS 15-415/615 74

CMU SCS

Variations

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

Faloutsos CMU SCS 15-415/615 75

CMU SCS

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 15-415/615 76

CMU SCS

B-trees and B*-trees

Eg., Tree T0; insert '2'

Faloutsos CMU SCS 15-415/615 77

CMU SCS

B*-trees: deferred split!

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

Faloutsos CMU SCS 15-415/615 78

CMU SCS

B*-trees: deferred split!

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

FINAL TREE

Faloutsos CMU SCS 15-415/615 79

CMU SCS

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 15-415/615 80

CMU SCS

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 15-415/615 81

CMU SCS

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 15-415/615 82

CMU SCS

Outline

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

Faloutsos CMU SCS 15-415/615 83

CMU SCS

B+ trees - Motivation

For clustering index, data records are scattered:

Faloutsos CMU SCS 15-415/615 84

CMU SCS

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 15-415/615 85

CMU SCS

B+ trees

Faloutsos CMU SCS 15-415/615 86

CMU SCS

B+ trees

Faloutsos CMU SCS 15-415/615 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 15-415/615 88

CMU SCS

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 15-415/615 89

CMU SCS

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 15-415/615 90

CMU SCS

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 15-415/615 91

CMU SCS

Inserting a Data Entry into a B+ Tree

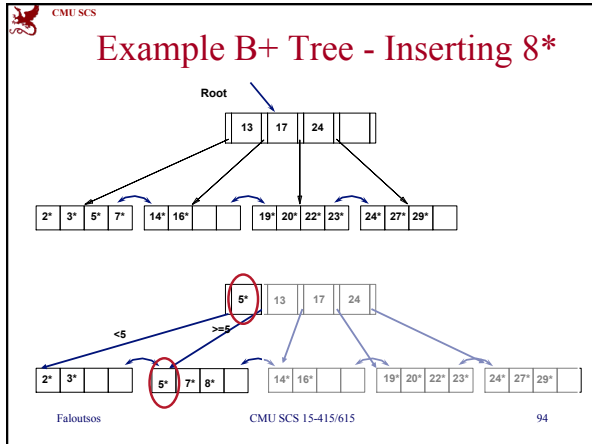
- Find correct leaf *L*.
- Put data entry onto *L*.
 - If *L* has enough space, *done!*
 - Else, must *split L* (into *L* and a new node *L2*)
 - Redistribute entries evenly, **copy up** middle key.
- parent node may overflow
 - but then: **push up** middle key. Splits “grow” tree; root split increases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

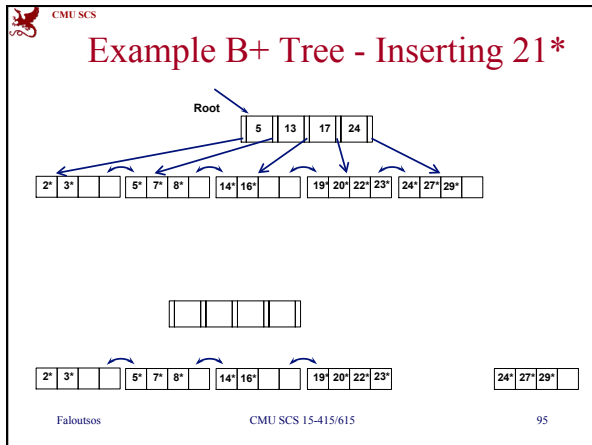
Faloutsos CMU SCS 15-415/615 92

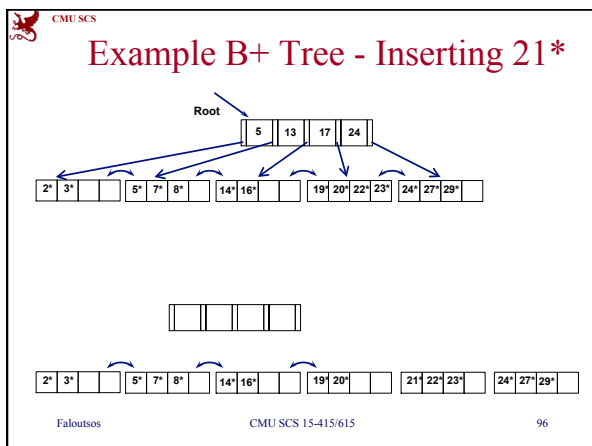
CMU SCS

Example B+ Tree - Inserting 8*

Faloutsos CMU SCS 15-415/615 93







CMU SCS

Example B+ Tree

- Notice that root was split, increasing height.
- Could use defer-split here. (Pros/Cons?)

Faloutsos CMU SCS 15-415/615 97

CMU SCS

Example: Data vs. Index Page Split

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

Faloutsos CMU SCS 15-415/615 98

CMU SCS

Now you try...

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

Faloutsos CMU SCS 15-415/615 99

CMU SCS

Answer...

After inserting 28*, 6*

After inserting 25*

Faloutsos CMU SCS 15-415/615 100

CMU SCS

Answer...

After inserting 25*

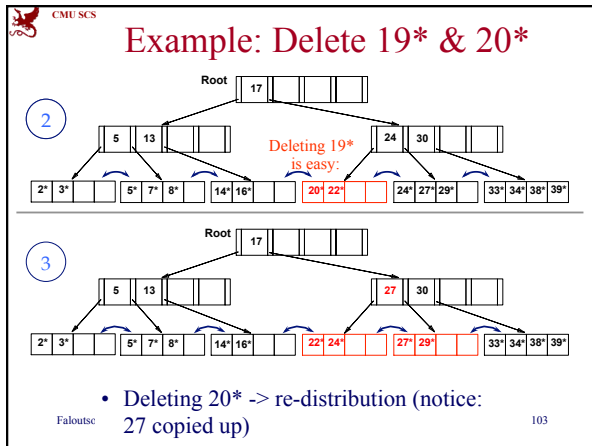
Faloutsos CMU SCS 15-415/615 101

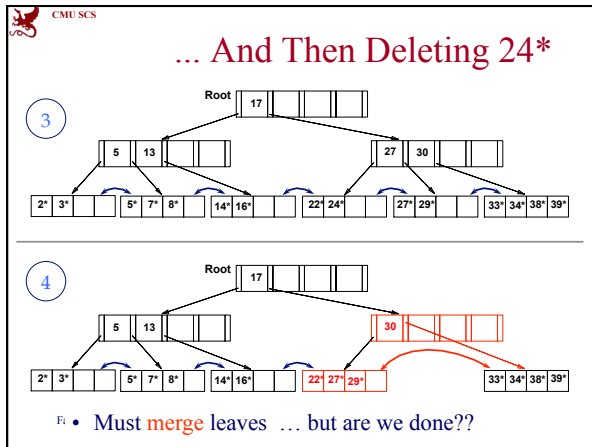
CMU SCS

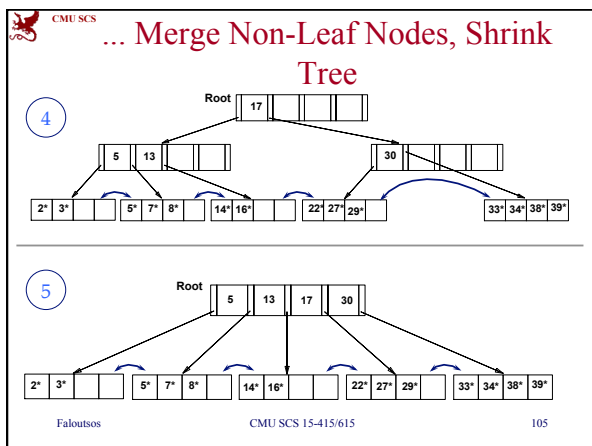
Deleting a Data Entry from a B+ Tree

- 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 15-415/615 102







Example of Non-leaf Re-distribution

- Tree is shown below *during deletion* of 24*.
- Now, we **can** re-distribute keys

Faloutsos CMU SCS 15-415/615 106

After Re-distribution

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

Faloutsos CMU SCS 15-415/615 107

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 15-415/615 108

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 15-415/615 109

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 15-415/615 110

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 15-415/615 111

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 15-415/615 112

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 15-415/615 113

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 15-415/615 114

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 15-415/615 115

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 15-415/615 116

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 15-415/615 117

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 15-415/615 118

CMU SCS

Bulk Loading of a B+ Tree

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

Faloutsos CMU SCS 15-415/615 119

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 15-415/615 120

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 15-415/615 122

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 15-415/615 123

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 'slopiness'?)

Faloutsos CMU SCS 15-415/615 124

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 15-415/615 125

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 15-415/615 126
