


CMU SCS

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

C. Faloutsos & A. Pavlo
Lecture#9 (R&G ch. 10)
Indexing




CMU SCS

Outline

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

Faloutsos - Pavlo
CMU SCS 15-415/615
2




CMU SCS

Introduction

- How to support range searches?
- equality searches?

Faloutsos - Pavlo
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?

Page 1

Page 2

Page 3

Page N

Data File

Faloutsos - Pavlo
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.

The diagram illustrates a linear search process. An 'Index File' contains a sequence of keys: k_1 , k_2 , ..., k_N . Each key is enclosed in a red box. Red arrows point from these keys to a 'Data File' consisting of pages labeled 'Page 1', 'Page 2', 'Page 3', ..., 'Page N'. This represents a direct mapping from index to data.

Faloutsos - Pavlo CMU SCS 15-415/615 5

CMU SCS

Range Searches

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

This diagram is identical to the one on slide 5, showing the mapping from an index file to a data file.

Faloutsos - Pavlo CMU SCS 15-415/615 6

CMU SCS

ISAM

- Repeat recursively!

The diagram shows a hierarchical tree structure. The root node is a 'Non-leaf Page'. It branches into two child nodes, which are also 'Non-leaf Pages'. This structure continues down to 'Leaf Pages' at the bottom. Red arrows indicate the flow from parent nodes to child nodes. Ellipses (...) are used to indicate that there are more nodes in the tree than shown.

Faloutsos - Pavlo CMU SCS 15-415/615 7

CMU SCS

ISAM

- OK - what if there are insertions and overflows?

This diagram is identical to the one on slide 7, illustrating the ISAM tree structure.

Faloutsos - Pavlo CMU SCS 15-415/615 8

CMU SCS

ISAM

- Overflow pages, linked to the primary page

The diagram illustrates the ISAM structure. At the top, a tree of non-leaf pages (red boxes) branches downwards. Below this, a horizontal line separates the non-leaf pages from the leaf pages. Leaf pages (white boxes) are shown below the line. Some leaf pages are labeled as 'Primary pages' and others as 'Overflow pages'. Arrows indicate the flow of data from non-leaf pages to leaf pages, and from primary pages to overflow pages.

Faloutsos - Pavlo CMU SCS 15-415/615 9

CMU SCS

Example ISAM Tree

- 2 entries per page

The diagram shows an example ISAM tree. The root node is labeled 'Root' and contains the value 40. It has two children: a left internal node with values 20 and 33, and a right internal node with values 51 and 63. Each internal node has two children. The left internal node has leaf nodes with values 10*, 15*, 20*, and 27*. The right internal node has leaf nodes with values 33*, 37*, 40*, 46*, 51*, 55*, and 63*, 97*.

Faloutsos - Pavlo CMU SCS 15-415/615 10

CMU SCS

ISAM

Details

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

The diagram shows an ISAM tree with a root node containing 48. It has two internal nodes: the left one contains 20 and 36, and the right one contains 50 and 60. The left internal node has four leaf nodes with values 10*, 15*, 20*, and 27*. The right internal node has four leaf nodes with values 33*, 37*, 40*, and 46*. The leaf nodes are arranged in a sequence: 10*, 15*, 20*, 27*, 33*, 37*, 40*, 46*, 51*, 55*, 63*, 97*.

Faloutsos - Pavlo CMU SCS 15-415/615 11

CMU SCS

ISAM

Details

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

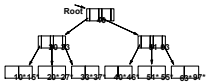
The diagram shows an ISAM tree with a root node containing 48. It has two internal nodes: the left one contains 20 and 36, and the right one contains 50 and 60. The left internal node has four leaf nodes with values 10*, 15*, 20*, and 27*. The right internal node has four leaf nodes with values 33*, 37*, 40*, and 46*. The leaf nodes are arranged in a sequence: 10*, 15*, 20*, 27*, 33*, 37*, 40*, 46*, 51*, 55*, 63*, 97*.

Faloutsos - Pavlo 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.

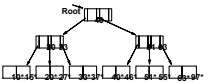


Faloutsos - Pavlo CMU SCS 15-415/615 13

CMU SCS

ISAM is a STATIC Structure

- *Search*: Start at root; use key comparisons to go to leaf.
- Cost = ??

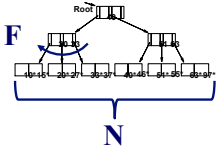


Faloutsos - Pavlo CMU SCS 15-415/615 14

CMU SCS

ISAM is a STATIC Structure

- *Search*: Start at root; use key comparisons to go to leaf.
- Cost = ??

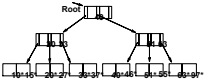


Faloutsos - Pavlo CMU SCS 15-415/615 15

CMU SCS

ISAM is a STATIC Structure

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



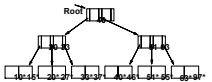
Faloutsos - Pavlo CMU SCS 15-415/615 16

CMU SCS

ISAM is a STATIC Structure

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

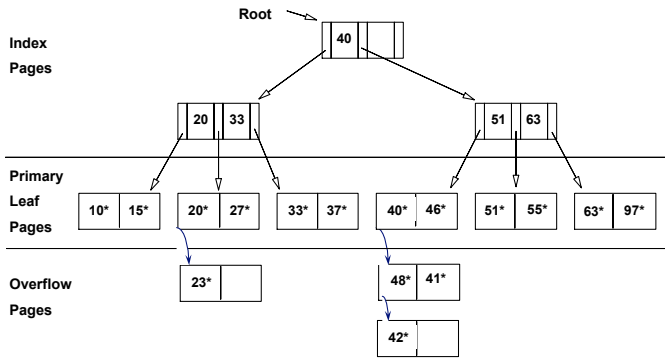
Delete: Find and remove from leaf; if empty page, de-allocate.



Faloutsos - Pavlo CMU SCS 15-415/615 17

CMU SCS

Example: Insert 23*, 48*, 41*, 42*



Faloutsos - Pavlo CMU SCS 15-415/615 18

CMU SCS

21* means

- <21> + rest of record
- (it's a bit more complicated – but we stay with that, for the moment).
- ‘21’ plain means just 4 bytes, to store integer 21

21* →

21	(name, age, etc)
----	------------------

 ~record

21 →

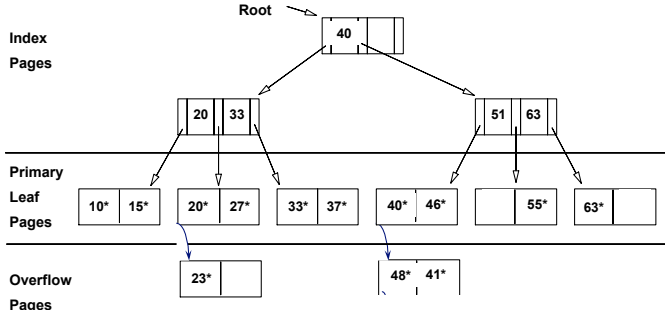
21

 divider

Faloutsos - Pavlo CMU SCS 15-415/615 19

CMU SCS

... then delete 42*, 51*, 97*



➡ Note that 51 appears in index levels, but not in leaf!

Faloutsos - Pavlo CMU SCS 15-415/615 20

CMU SCS

ISAM ---- Issues?

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

Faloutsos - Pavlo CMU SCS 15-415/615 21

CMU SCS

Outline

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

Faloutsos - Pavlo CMU SCS 15-415/615 22

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 - Pavlo CMU SCS 15-415/615 23

CMU SCS

B-trees

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



Faloutsos - Pavlo CMU SCS 15-415/615 24

CMU SCS

B-trees

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

Faloutsos - Pavlo CMU SCS 15-415/615 25

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 - Pavlo CMU SCS 15-415/615 26

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 - Pavlo CMU SCS 15-415/615 27

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 28

CMU SCS

JAVA animation!

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

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

Faloutsos - Pavlo CMU SCS 15-415/615 29

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 30

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 31

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 32

CMU SCS

Queries

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

H steps (= disk accesses)

Faloutsos - Pavlo 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 - Pavlo 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 - Pavlo CMU SCS 15-415/615 35

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 36

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 37

CMU SCS

Queries

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

Faloutsos - Pavlo CMU SCS 15-415/615 38

CMU SCS

B-trees: Insertion

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

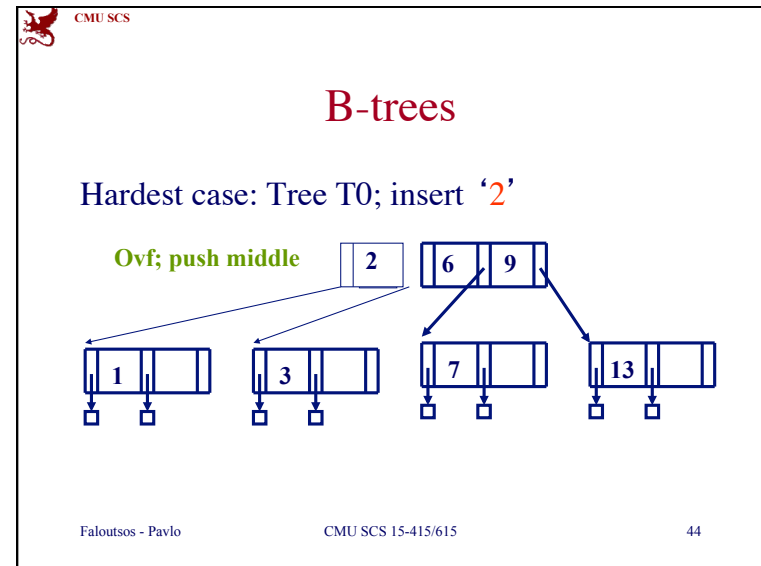
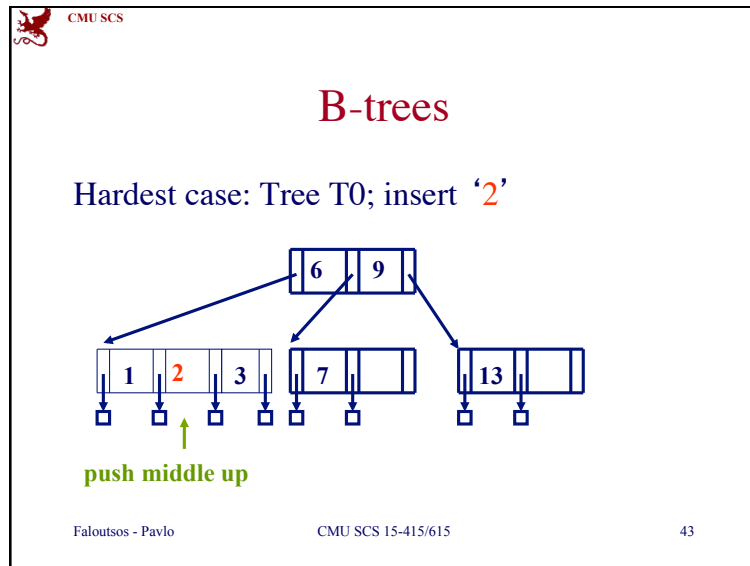
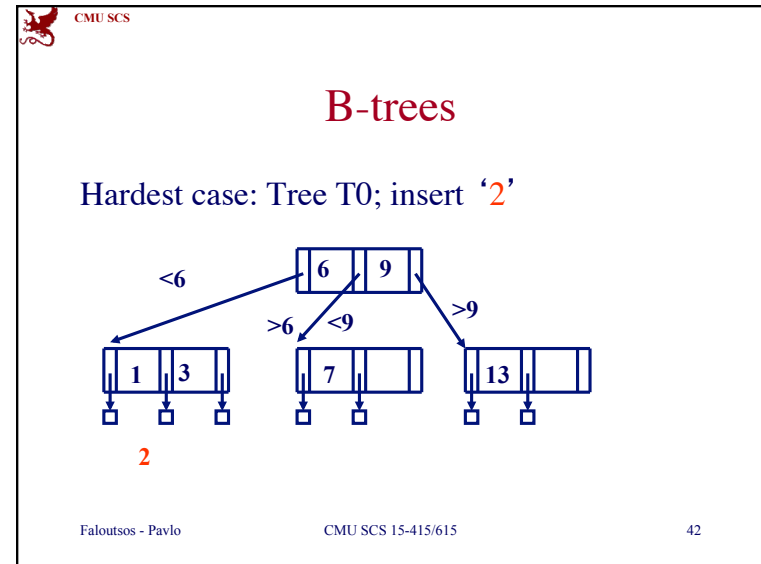
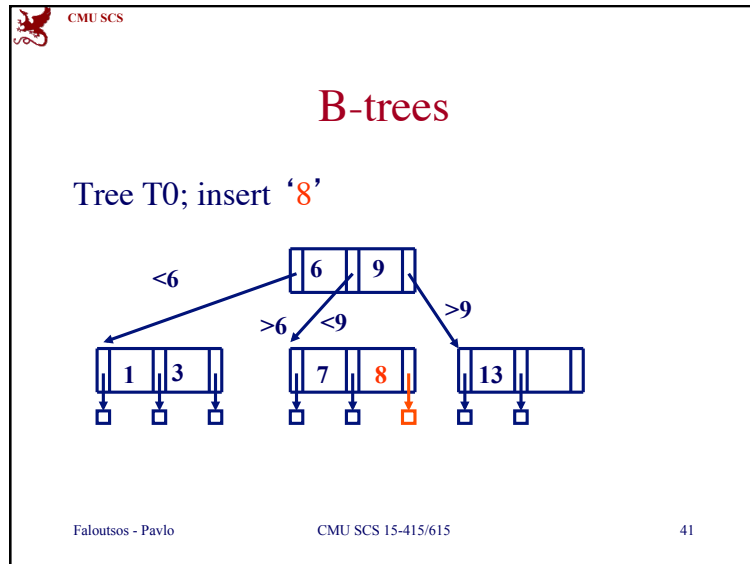
Faloutsos - Pavlo CMU SCS 15-415/615 39

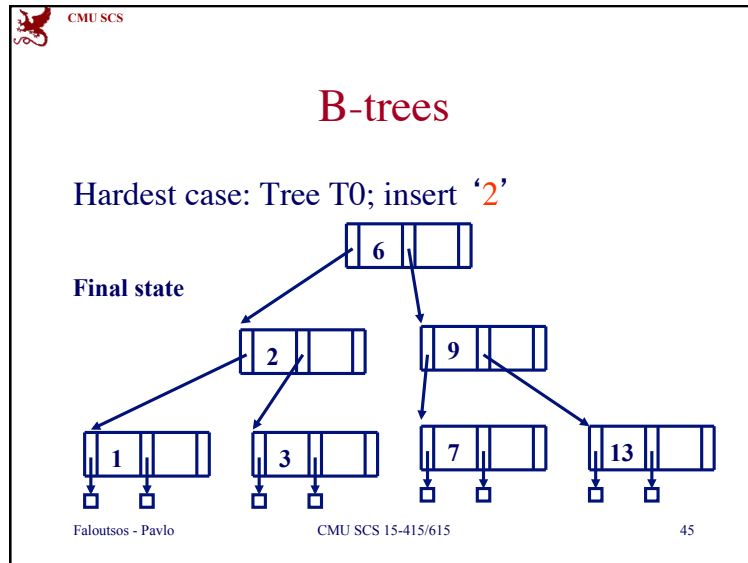
CMU SCS

B-trees

Easy case: Tree T0; insert '8'

Faloutsos - Pavlo CMU SCS 15-415/615 40





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 - Pavlo CMU SCS 15-415/615 46

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 - Pavlo CMU SCS 15-415/615 47

CMU SCS

Overview

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

Faloutsos - Pavlo CMU SCS 15-415/615 48

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 - Pavlo CMU SCS 15-415/615 49

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

Faloutsos - Pavlo CMU SCS 15-415/615 50

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

Faloutsos - Pavlo CMU SCS 15-415/615 51

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 - Pavlo CMU SCS 15-415/615 52

CMU SCS

B-trees – Deletion

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

Delete 3 from T0

Faloutsos - Pavlo 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)

Delete & promote, ie:

Faloutsos - Pavlo 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)

Delete & promote, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 55

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 56

CMU SCS

B-trees – Deletion

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

FINAL TREE

Faloutsos - Pavlo CMU SCS 15-415/615 57

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 - Pavlo CMU SCS 15-415/615 58

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 - Pavlo CMU SCS 15-415/615 59

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 60

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 61

CMU SCS

B-trees – Deletion

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

Faloutsos - Pavlo CMU SCS 15-415/615 62

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 63

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 64

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Faloutsos - Pavlo CMU SCS 15-415/615 65

CMU SCS

B-trees – Deletion

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

Delete & borrow, through the parent

FINAL TREE

Faloutsos - Pavlo CMU SCS 15-415/615 66

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 - Pavlo CMU SCS 15-415/615 67

CMU SCS

B-trees – Deletion

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

Faloutsos - Pavlo CMU SCS 15-415/615 68

CMU SCS

B-trees – Deletion

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

Faloutsos - Pavlo CMU SCS 15-415/615 69

CMU SCS

B-trees – Deletion

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

Faloutsos - Pavlo CMU SCS 15-415/615 70

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 - Pavlo CMU SCS 15-415/615 71

CMU SCS

B-trees – Deletion

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

Faloutsos - Pavlo CMU SCS 15-415/615 72

CMU SCS

B-trees – Deletion

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

FINAL TREE

Faloutsos - Pavlo CMU SCS 15-415/615 73

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 - Pavlo CMU SCS 15-415/615 74

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 - Pavlo CMU SCS 15-415/615 75

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 - Pavlo CMU SCS 15-415/615 76

CMU SCS

Outline

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

Faloutsos - Pavlo CMU SCS 15-415/615 77

CMU SCS

Variations

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

Faloutsos - Pavlo CMU SCS 15-415/615 78

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 - Pavlo CMU SCS 15-415/615 79

CMU SCS

B-trees and B*-trees

Eg., Tree T0; insert '2'

Faloutsos - Pavlo CMU SCS 15-415/615 80

CMU SCS

B*-trees: deferred split!

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

Faloutsos - Pavlo CMU SCS 15-415/615 81

CMU SCS

B*-trees: deferred split!

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

FINAL TREE

Faloutsos - Pavlo CMU SCS 15-415/615 82

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 - Pavlo CMU SCS 15-415/615 83

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 - Pavlo CMU SCS 15-415/615 84

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 - Pavlo CMU SCS 15-415/615 85

CMU SCS

Outline

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

Faloutsos - Pavlo CMU SCS 15-415/615 86

CMU SCS

B+ trees - Motivation

For clustering index, data records are scattered:

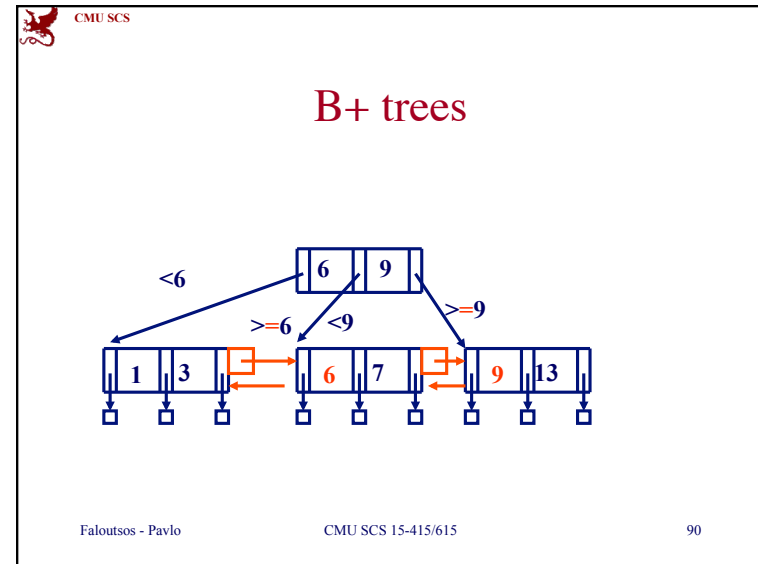
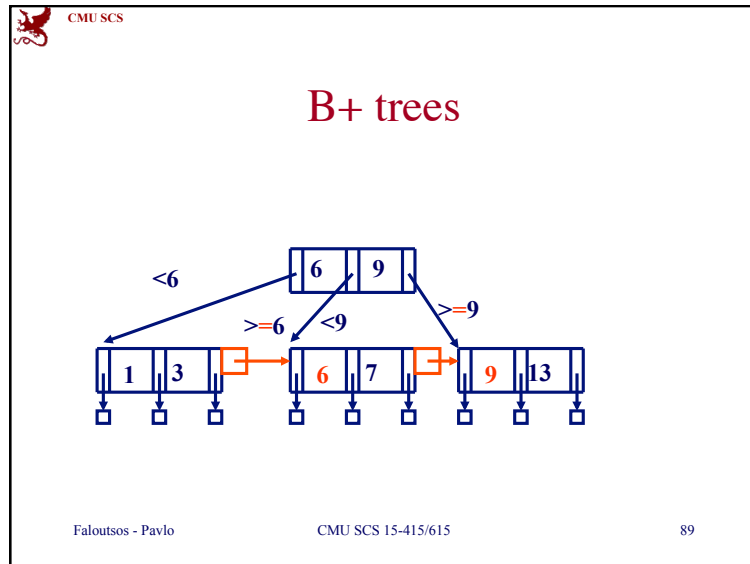
Faloutsos - Pavlo CMU SCS 15-415/615 87

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 - Pavlo CMU SCS 15-415/615 88



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 - Pavlo CMU SCS 15-415/615 91

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 - Pavlo CMU SCS 15-415/615 92

CMU SCS

B+ Trees in Practice

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

Faloutsos - Pavlo CMU SCS 15-415/615 93

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 - Pavlo CMU SCS 15-415/615 94

CMU SCS

Inserting a Data Entry into a B+ Tree

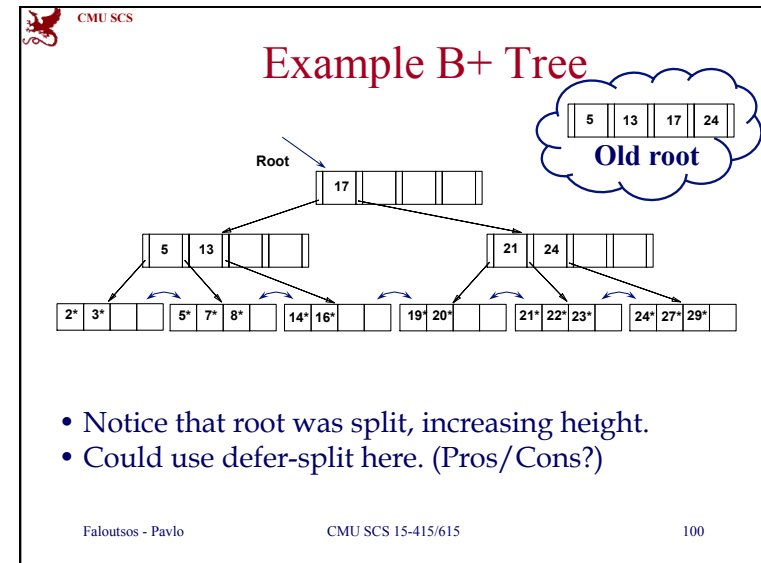
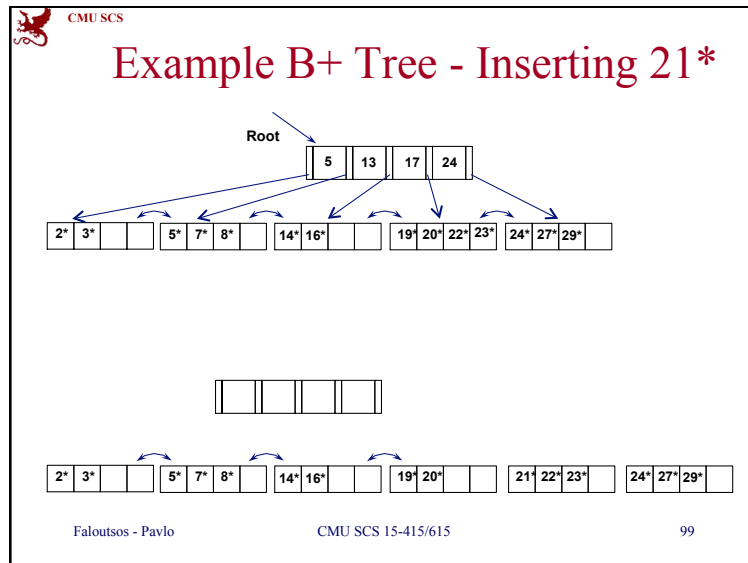
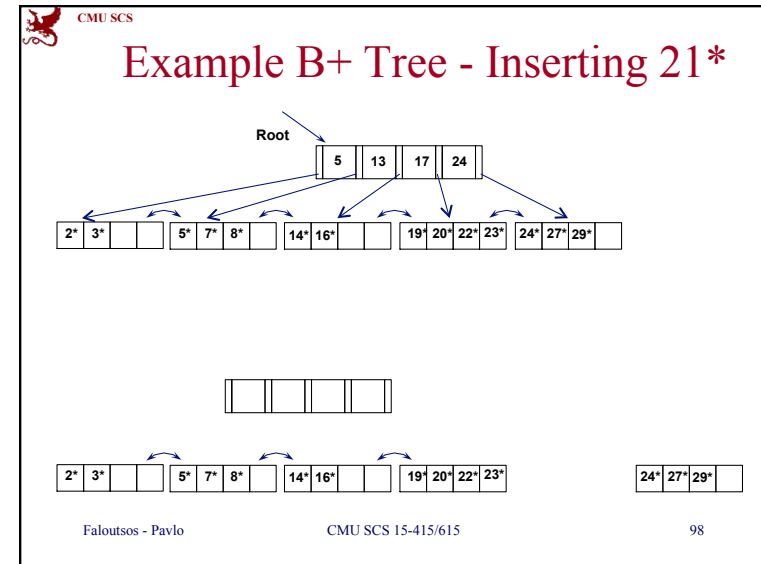
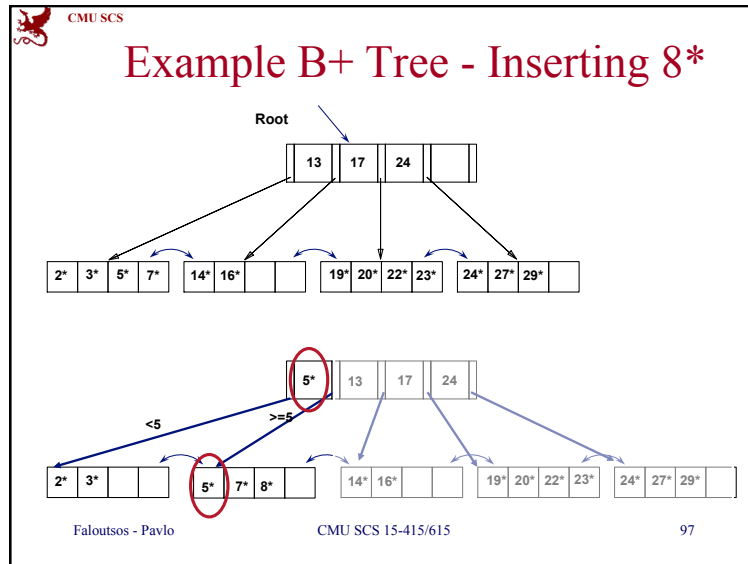
- Find correct leaf L .
- Put data entry onto L .
 - If L has enough space, *done!*
 - Else, must *split* L (into L and a new node $L2$)
 - Redistribute entries evenly, **copy up** middle key.
- parent node may overflow
 - but then: **push up** middle key. Splits “grow” tree; root split increases height.

Faloutsos - Pavlo CMU SCS 15-415/615 95

CMU SCS

Example B+ Tree - Inserting 8*

Faloutsos - Pavlo CMU SCS 15-415/615 96



Example: Data vs. Index Page Split

Split

- leaf: 'copy'
- non-leaf: 'push'

Data Page Split

Index Page Split

- why not 'copy' @ non-leaves?

Faloutsos - Pavlo CMU SCS 15-415/615 101

Now you try...

... (not shown)

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

Faloutsos - Pavlo CMU SCS 15-415/615 102

Now you try...

After inserting 28*

... (not shown)

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

Faloutsos - Pavlo CMU SCS 15-415/615 103

Answer...

After inserting 28*, 6*

Faloutsos - Pavlo CMU SCS 15-415/615 104

CMU SCS

Answer...

After inserting 28*, 6*

insert 25*:
 Q1: which pages will be affected:
 Q2: how will the root look like after that?

Faloutsos - Pavlo CMU SCS 15-415/615 105

CMU SCS

Answer...

After inserting 28*, 6*

insert 25*:
 Q1: which pages will be affected: A1: red arrows
 Q2: how will the root look like after that? A2: (13; 30; _; _)

Faloutsos - Pavlo CMU SCS 15-415/615 106

CMU SCS

Answer...

After inserting 25*

25* causes propagated split!

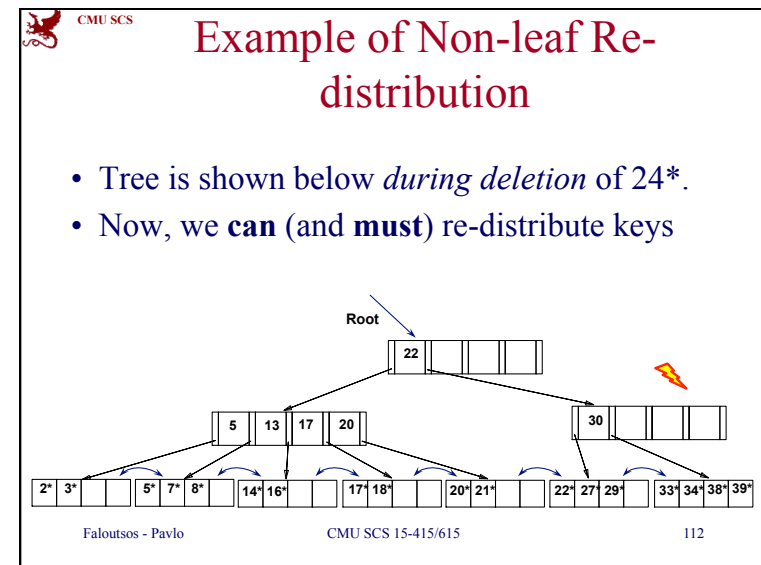
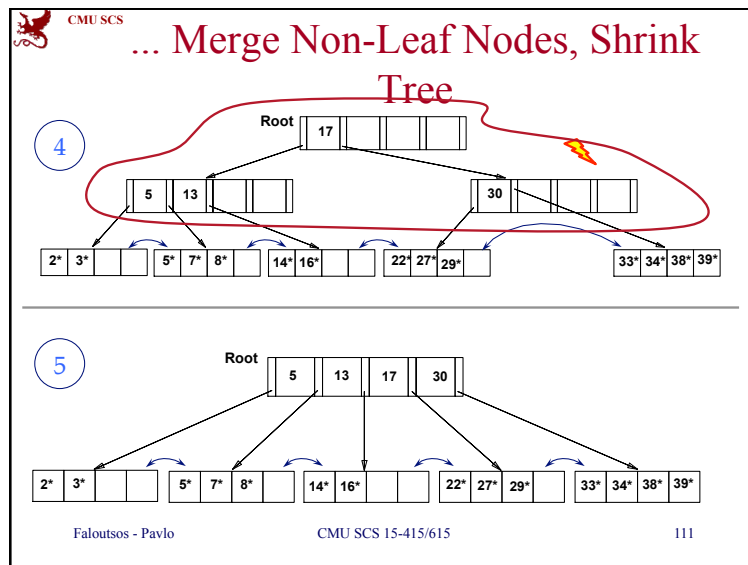
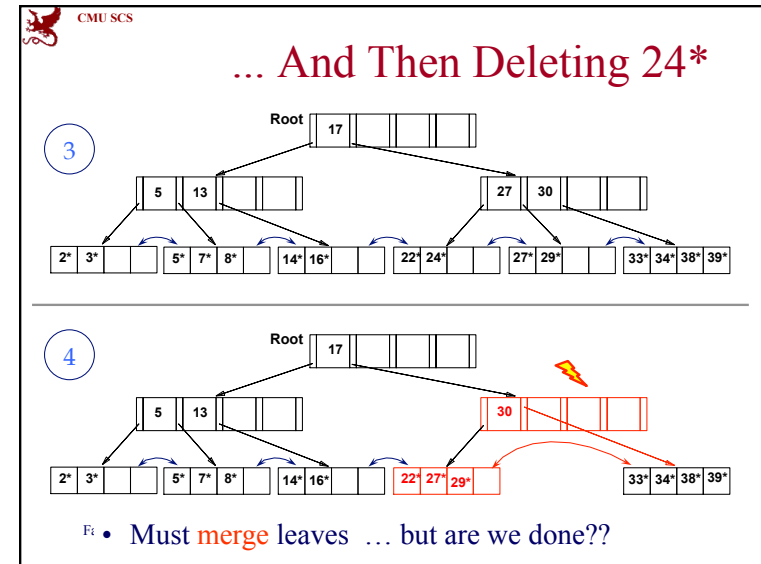
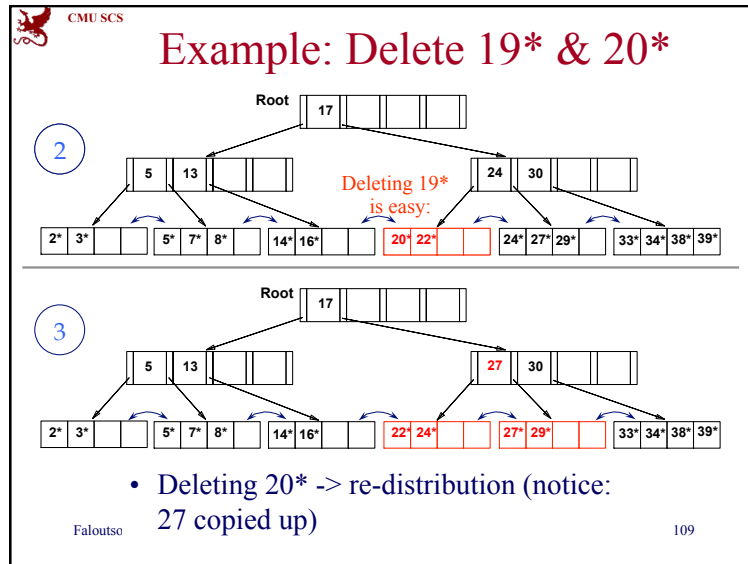
Faloutsos - Pavlo CMU SCS 15-415/615 107

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 - Pavlo CMU SCS 15-415/615 108



CMU SCS

After Re-distribution

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

Faloutsos - Pavlo CMU SCS 15-415/615 113

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 - Pavlo CMU SCS 15-415/615 114

CMU SCS

Main observations for deletion

- 'lazy deletions' - in fact, some vendors just mark entries as deleted (~ underflow),
– and reorganize/compact later

Faloutsos - Pavlo CMU SCS 15-415/615 115

CMU SCS

Main observations for deletion

- 'lazy deletions' - in fact, some vendors just mark entries as deleted (~ underflow),
– and reorganize/compact later

Q: Now, what?

Faloutsos - Pavlo CMU SCS 15-415/615 116

CMU SCS

Main observations for deletion

- ‘lazy deletions’ - in fact, some vendors just mark entries as deleted (~ underflow),
– and reorganize/compact later

Q: Now, what? A: ‘Merge’

Faloutsos - Pavlo CMU SCS 15-415/615 117

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 - Pavlo CMU SCS 15-415/615 118

CMU SCS

Outline

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

Faloutsos - Pavlo CMU SCS 15-415/615 119

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 - Pavlo CMU SCS 15-415/615 120

CMU SCS

B+ trees with duplicates

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

Faloutsos - Pavlo CMU SCS 15-415/615 121

CMU SCS

B+ trees with duplicates

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

Faloutsos - Pavlo CMU SCS 15-415/615 122

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 - Pavlo CMU SCS 15-415/615 123

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)

Used in HW3

Faloutsos - Pavlo CMU SCS 15-415/615 124

CMU SCS

Outline

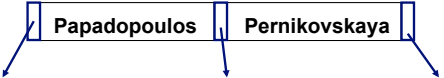
- Motivation
- ISAM
- B-trees (not in book)
- B+ trees
- duplicates
- B+ trees in practice
 - **prefix compression**; bulk-loading; ‘order’

Faloutsos - Pavlo CMU SCS 15-415/615 125

CMU SCS

Prefix Key Compression

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




Faloutsos - Pavlo CMU SCS 15-415/615 126

CMU SCS

Prefix Key Compression

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



Faloutsos - Pavlo CMU SCS 15-415/615 127

CMU SCS

Bulk Loading of a B+ Tree

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

Faloutsos - Pavlo CMU SCS 15-415/615 128

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>

Sorted pages of data entries; not yet in B+ tree

Faloutsos - Pavlo CMU SCS 15-415/615 129

CMU SCS

Bulk Loading (Contd.)

- Book's algo
- (any problems?)

Data entry pages not yet in B+ tree

Faloutsos - Pavlo CMU SCS 15-415/615 130

CMU SCS

Outline

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


Faloutsos - Pavlo CMU SCS 15-415/615 131

CMU SCS

A Note on 'Order'

- *Order (d)* concept replaced by physical space criterion in practice (*at least half-full*).
- Why do we need the distinction?

Faloutsos - Pavlo CMU SCS 15-415/615 132




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 - Pavlo CMU SCS 15-415/615 133




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 - Pavlo CMU SCS 15-415/615 134




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 - Pavlo CMU SCS 15-415/615 135



CMU SCS

Conclusions

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

Faloutsos - Pavlo CMU SCS 15-415/615 136