


CMU SCS

15-826: Multimedia Databases and Data Mining

Lecture#2: Primary key indexing – B-trees
Christos Faloutsos - CMU
www.cs.cmu.edu/~christos




CMU SCS

Reading Material

[Ramakrishnan & Gehrke, 3rd ed, ch. 10]

15-826 Copyright: C. Faloutsos (2016) 2



CMU SCS

Problem

Given a large collection of (multimedia) records, find similar/interesting things, ie:

- Allow fast, approximate queries, and
- Find rules/patterns

15-826 Copyright: C. Faloutsos (2016) 3




CMU SCS

Outline

Goal: 'Find similar / interesting things'

- Intro to DB
- ➔ • Indexing - similarity search
- Data Mining

15-826 Copyright: C. Faloutsos (2016) 4




CMU SCS

Indexing - Detailed outline

- ➔ primary key indexing
 - B-trees and variants
 - (static) hashing
 - extendible hashing
- secondary key indexing
- spatial access methods
- text
- ...

15-826 Copyright: C. Faloutsos (2016) 5




CMU SCS

In even more detail:

- ➔ B – trees
 - B+ - trees, B*-trees
 - hashing

15-826 Copyright: C. Faloutsos (2016) 6




CMU SCS

Primary key indexing

- find employee with ssn=123

15-826 Copyright: C. Faloutsos (2016) 7



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

15-826 Copyright: C. Faloutsos (2016) 8

CMU SCS

Citation



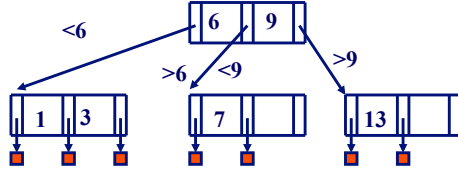
- Rudolf Bayer and Edward M. McCreight, *Organization and Maintenance of Large Ordered Indices*, Acta Informatica, 1:173-189, 1972.
- Received the 2001 SIGMOD innovations award
- among the most cited db publications
 - www.informatik.uni-trier.de/~ley/db/about/top.html

15-826 Copyright: C. Faloutsos (2016) 9

CMU SCS

B-trees

Eg., B-tree of order 3:

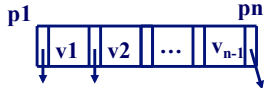


15-826 Copyright: C. Faloutsos (2016) 10

CMU SCS

B - tree properties:

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



15-826 Copyright: C. Faloutsos (2016) 11

CMU SCS

Properties

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

15-826 Copyright: C. Faloutsos (2016) 12

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2016) 13

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2016) 14

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2016) 15

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2016) 16

CMU SCS

Queries

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

H steps (= disk accesses)

15-826 Copyright: C. Faloutsos (2016) 17

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2016) 18

CMU SCS

Queries

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

H steps (= disk accesses)

15-826 Copyright: C. Faloutsos (2016) 19

CMU SCS

Queries

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

H steps (= disk accesses)

15-826 Copyright: C. Faloutsos (2016) 20

CMU SCS

B-trees: Insertion

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

15-826 Copyright: C. Faloutsos (2016) 21

CMU SCS

B-trees

Easy case: Tree T0; insert '8'

A B-tree with root node [6 | 9] and three leaf nodes [1 | 3], [7], and [13]. The root has pointers labeled <6, >6, <9, and >9. The leaf nodes have three pointers each, pointing to red squares representing leaf slots.

15-826 Copyright: C. Faloutsos (2016) 22

CMU SCS

B-trees

Tree T0; insert '8'

The same B-tree as in slide 22, but with the value '8' inserted into the second leaf node [7], making it [7 | 8].

15-826 Copyright: C. Faloutsos (2016) 23

CMU SCS

B-trees

Hardest case: Tree T0; insert '2'

The same B-tree as in slide 22, but with the value '2' inserted into the first leaf slot, making the first leaf node [1 | 3 | 2].

15-826 Copyright: C. Faloutsos (2016) 24

B-trees

Hardest case: Tree T0; insert '2'

push middle up

15-826 Copyright: C. Faloutsos (2016) 25

B-trees

Hardest case: Tree T0; insert '2'

Ovf; push middle

15-826 Copyright: C. Faloutsos (2016) 26

B-trees

Hardest case: Tree T0; insert '2'

Final state

15-826 Copyright: C. Faloutsos (2016) 27

B-trees: Insertion

- Q: What if there are two middles? (eg, order 4)
- A: either one is fine

15-826 Copyright: C. Faloutsos (2016) 28

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

15-826 Copyright: C. Faloutsos (2016) 29

CMU SCS

Overview

- B – trees
- ➔ – Dfn, Search, insertion, **deletion**
- B+ - trees
- hashing

15-826 Copyright: C. Faloutsos (2016) 30

CMU SCS

Deletion

Rough outline of algo:

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

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

15-826 Copyright: C. Faloutsos (2016) 31

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2016) 32

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2016) 33

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2016) 34

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'

15-826 Copyright: C. Faloutsos (2016) 35

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

15-826 Copyright: C. Faloutsos (2016) 36

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

15-826 Copyright: C. Faloutsos (2016) 37

CMU SCS

B-trees – Deletion

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

Delete & promote, ie:

15-826 Copyright: C. Faloutsos (2016) 38

CMU SCS

B-trees – Deletion

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

FINAL TREE

15-826 Copyright: C. Faloutsos (2016) 39

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

15-826 Copyright: C. Faloutsos (2016) 40

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’

15-826 Copyright: C. Faloutsos (2016) 41

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 42

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 43

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 44

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

Rich sibling

NO!!

15-826 Copyright: C. Faloutsos (2016) 45

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2016) 46

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2016) 47

CMU SCS

B-trees – Deletion

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

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2016) 48

CMU SCS

B-trees – Deletion

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

FINAL TREE

Delete & borrow, through the parent

15-826 Copyright: C. Faloutsos (2016) 49

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’

15-826 Copyright: C. Faloutsos (2016) 50

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 51

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 52

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 53

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.:

15-826 Copyright: C. Faloutsos (2016) 54

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 55

CMU SCS

B-trees – Deletion

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

FINAL TREE

15-826 Copyright: C. Faloutsos (2016) 56

CMU SCS

B-trees – Deletion

- Case4: underflow & ‘poor sibling’
- -> ‘pull key from parent, and merge’
- Q: What if the parent underflows?

15-826 Copyright: C. Faloutsos (2016) 57

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2016) 58

CMU SCS

Overview

- B – trees
- ➔ • **B+ - trees, B*-trees**
- hashing

15-826 Copyright: C. Faloutsos (2016) 59

CMU SCS

B+ trees - Motivation

if we want to store the whole record with the key -> problems (what?)

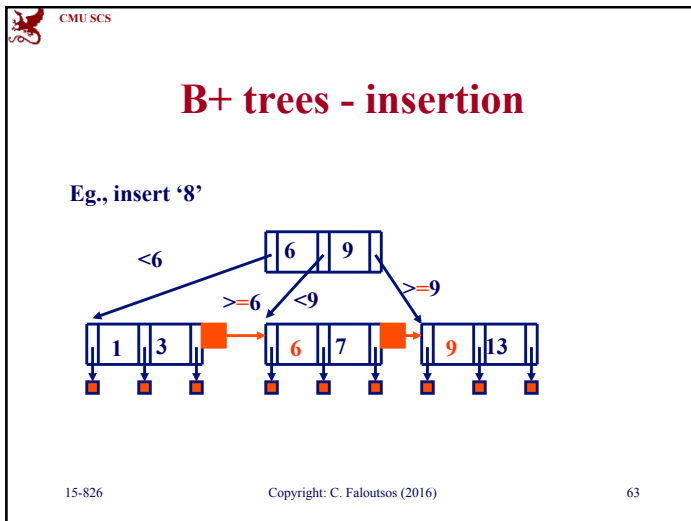
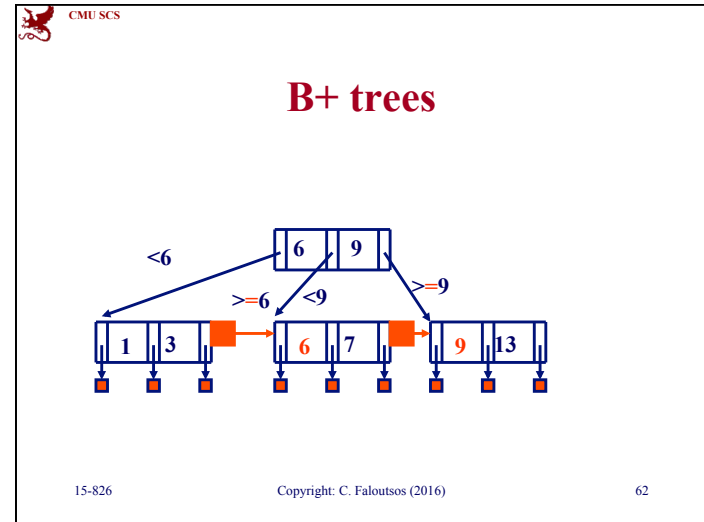
15-826 Copyright: C. Faloutsos (2016) 60

CMU SCS

Solution: B⁺ - trees

- They string all leaf nodes together
- AND
- replicate keys from non-leaf nodes, to make sure every key appears at the leaf level

15-826 Copyright: C. Faloutsos (2016) 61



CMU SCS

Overview

- B - trees
- ➔ • B⁺ - trees, B^{*}-trees
- hashing

15-826 Copyright: C. Faloutsos (2016) 64

CMU SCS

B*-trees

- splits drop util. to 50%, and maybe increase height
- How to avoid them?

15-826 Copyright: C. Faloutsos (2016) 65

CMU SCS

B*-trees: deferred split!

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

15-826 Copyright: C. Faloutsos (2016) 66

CMU SCS

B*-trees: deferred split!

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

FINAL TREE


15-826 Copyright: C. Faloutsos (2016) 67

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'?

15-826 Copyright: C. Faloutsos (2016) 68




CMU SCS

B*-trees: deferred split!

- BUT: What if the sibling has no room for our 'lending'?
- 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.
- Details: too messy (and even worse for deletion)

15-826 Copyright: C. Faloutsos (2016) 69



CMU SCS

Conclusions

- Main ideas: recursive; block-aware; on overflow -> split; **defer** splits
- All B-tree variants have excellent, **$O(\log N)$ worst-case performance for ins/del/search**
- B+ tree is the prevailing indexing method
- More details: [Knuth vol 3.] or [Ramakrishnan & Gehrke, 3rd ed, ch. 10]

15-826 Copyright: C. Faloutsos (2016) 70