


**15-826: Multimedia Databases
and Data Mining**


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



Reading Material

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

15-826 Copyright: C. Faloutsos (2012) 2




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 (2012) 3




CMU SCS

Outline

Goal: 'Find **similar / interesting** things'

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

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 5



CMU SCS

In even more detail:

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

15-826 Copyright: C. Faloutsos (2012) 6

CMU SCS

Primary key indexing

- find employee with ssn=123

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 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 (2012) 9

CMU SCS

B-trees

Eg., B-tree of order 3:

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 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 (2012) 12

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 13

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 14

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 15

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 16

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 18

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 19

CMU SCS

Queries

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

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 21

CMU SCS

B-trees

Easy case: Tree T0; insert '8'

15-826 Copyright: C. Faloutsos (2012) 22

CMU SCS

B-trees

Tree T0; insert '8'

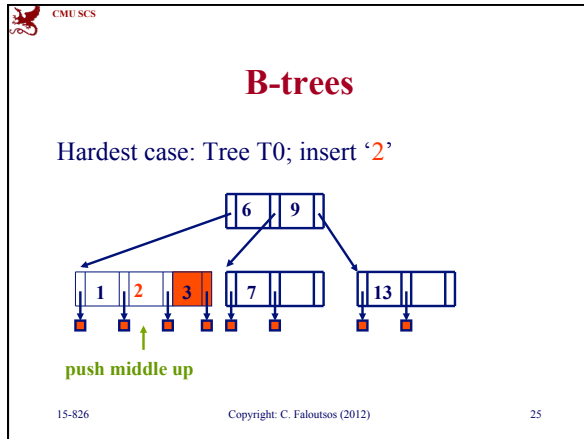
15-826 Copyright: C. Faloutsos (2012) 23

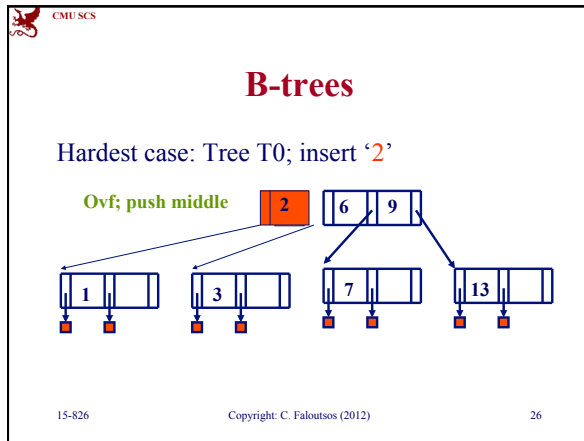
CMU SCS

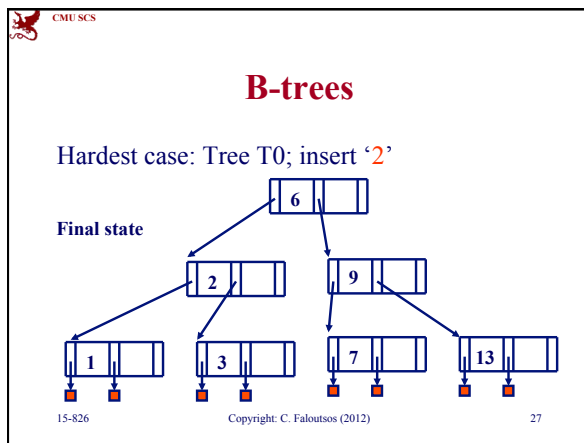
B-trees


Hardest case: Tree T0; insert '2'

15-826 Copyright: C. Faloutsos (2012) 24










CMU SCS

B-trees: Insertion

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

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 29



CMU SCS

Overview

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

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 31

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2012) 32

CMU SCS

B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2012) 33

B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2012) 34

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 (2012) 35

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 (2012) 36

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 (2012) 37

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 (2012) 38

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 (2012) 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 (2012) 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 (2012) 41

CMU SCS

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 42

B-trees – Deletion

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

Rich sibling

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2012) 43

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 (2012) 44

B-trees – Deletion

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

Rich sibling

Delete & borrow, ie:

NO!!

15-826 Copyright: C. Faloutsos (2012) 45

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 46

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 47

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 48

B-trees – Deletion

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

FINAL TREE

15-826 Copyright: C. Faloutsos (2012) 49

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 (2012) 50

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 51

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 52

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 53

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’
- I.e.:

15-826 Copyright: C. Faloutsos (2012) 54

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 55

B-trees – Deletion

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

FINAL TREE

15-826 Copyright: C. Faloutsos (2012) 56

B-trees – Deletion

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

15-826 Copyright: C. Faloutsos (2012) 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 (2012) 58

CMU SCS

Overview

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

15-826 Copyright: C. Faloutsos (2012) 59

CMU SCS

B+ trees - Motivation

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

```

graph TD
    Root["6 | 9"]
    Leaf1["1 | 3"]
    Leaf2["7"]
    Leaf3["13"]
    
    Root -- "<6" --> Leaf1
    Root -- ">6 <9" --> Leaf2
    Root -- ">9" --> Leaf3
    
    Leaf1 --> R1[ ]
    Leaf1 --> R2[ ]
    Leaf2 --> R3[ ]
    Leaf2 --> R4[ ]
    Leaf3 --> R5[ ]
    Leaf3 --> R6[ ]
  
```

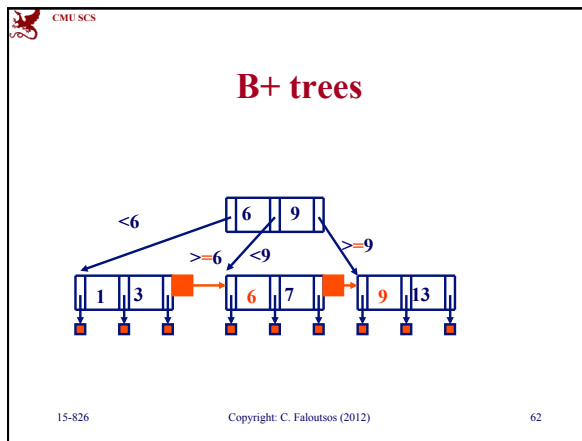
15-826 Copyright: C. Faloutsos (2012) 60

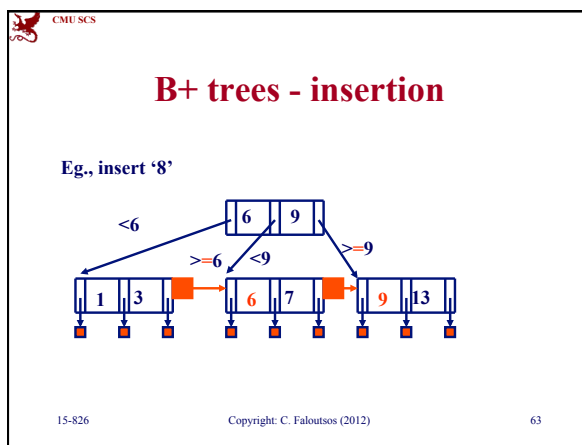
CMU SCS

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 (2012) 61





Overview

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

15-826

Copyright: C. Faloutsos (2012)

64

B*-trees

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

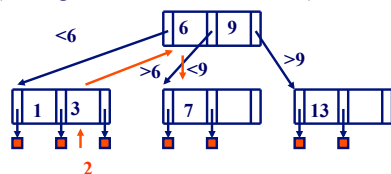
15-826

Copyright: C. Faloutsos (2012)

65

B*-trees: deferred split!

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



15-826

Copyright: C. Faloutsos (2012)

66

B*-trees: deferred split!

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

FINAL TREE

15-826 Copyright: C. Faloutsos (2012) 67

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 (2012) 68

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 (2012) 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 (2012)

70
