


CMU SCS

## 15-826: Multimedia Databases and Data Mining

Primary key indexing – B-trees  
Christos Faloutsos - CMU  
[www.cs.cmu.edu/~christos](http://www.cs.cmu.edu/~christos)




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 (2006) 2




CMU SCS

## Outline

Goal: 'Find similar / interesting things'

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

15-826 Copyright: C. Faloutsos (2006) 3




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 (2006) 4




CMU SCS

## Primary key indexing

- find employee with ssn=123

15-826 Copyright: C. Faloutsos (2006) 5



CMU SCS


## B-trees

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

15-826 Copyright: C. Faloutsos (2006) 6

CMU SCS

## Citation



- Rudolf Bayer and Edward M. McCreight, *Organization and Maintenance of Large Ordered Indices*, Acta Informatica, 1:173-189, 1972.

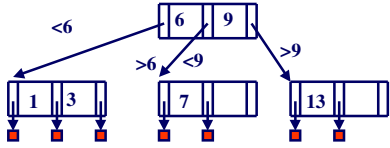
- Received the 2001 SIGMOD innovations award
- among the most cited db publications
  - [www.informatik.uni-trier.de/~ley/db/about/top.html](http://www.informatik.uni-trier.de/~ley/db/about/top.html)

15-826 Copyright: C. Faloutsos (2006) 7

CMU SCS

## B-trees

Eg., B-tree of order 3:

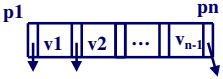


15-826 Copyright: C. Faloutsos (2006) 8

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 (2006) 9

CMU SCS

## Properties

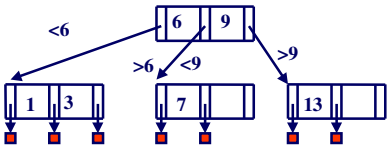
- “block aware” nodes: each node -> disk page
- $O(\log(N))$  for everything! (ins/del/search)
- typically, if  $m = 50 - 100$ , then 2 - 3 levels
- utilization  $\geq 50\%$ , guaranteed; on average 69%

15-826 Copyright: C. Faloutsos (2006) 10

CMU SCS

## Queries

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

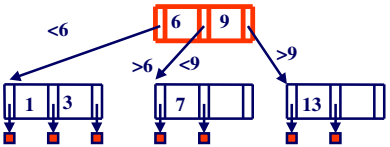


15-826 Copyright: C. Faloutsos (2006) 11

CMU SCS

## Queries

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



15-826 Copyright: C. Faloutsos (2006) 12

CMU SCS

### Queries

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

15-826 Copyright: C. Faloutsos (2006) 13

CMU SCS

### Queries

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

15-826 Copyright: C. Faloutsos (2006) 14

CMU SCS

### Queries

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

15-826 Copyright: C. Faloutsos (2006) 15

CMU SCS

### Queries

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

15-826 Copyright: C. Faloutsos (2006) 16

CMU SCS

### Queries

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

15-826 Copyright: C. Faloutsos (2006) 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 (2006) 18

**B-trees: Insertion**

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

15-826 Copyright: C. Faloutsos (2006) 19

**B-trees**

Easy case: Tree T0; insert '8'

15-826 Copyright: C. Faloutsos (2006) 20

**B-trees**

Tree T0; insert '8'

15-826 Copyright: C. Faloutsos (2006) 21

**B-trees**

Hardest case: Tree T0; insert '2'

15-826 Copyright: C. Faloutsos (2006) 22

**B-trees**

Hardest case: Tree T0; insert '2'

15-826 Copyright: C. Faloutsos (2006) 23

**B-trees**

Hardest case: Tree T0; insert '2'

Ovf; push middle

15-826 Copyright: C. Faloutsos (2006) 24

CMU SCS

## B-trees

Hardest case: Tree T0; insert '2'

Final state

15-826 Copyright: C. Faloutsos (2006) 25

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 (2006) 26

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 (2006) 27

CMU SCS

## Overview

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

15-826 Copyright: C. Faloutsos (2006) 28

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 (2006) 29

CMU SCS

## B-trees – Deletion

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2006) 30

**B-trees – Deletion**

Easiest case: Tree T0; delete '3'

15-826 Copyright: C. Faloutsos (2006) 31

**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 (2006) 32

**B-trees – Deletion**

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

15-826 Copyright: C. Faloutsos (2006) 33

**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 (2006) 34

**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 (2006) 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 (2006) 36

**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 (2006) 37

**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 (2006) 38

**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 (2006) 39

**B-trees – Deletion**

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

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2006) 40

**B-trees – Deletion**

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

Rich sibling

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2006) 41

**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 (2006) 42

**B-trees – Deletion**

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

Rich sibling

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2006) 43

**B-trees – Deletion**

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

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2006) 44

**B-trees – Deletion**

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

Delete & borrow, ie:

15-826 Copyright: C. Faloutsos (2006) 45

**B-trees – Deletion**

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

FINAL TREE

Delete & borrow, through the parent

15-826 Copyright: C. Faloutsos (2006) 46

**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 (2006) 47

**B-trees – Deletion**

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

15-826 Copyright: C. Faloutsos (2006) 48

**B-trees – Deletion**

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

15-826 Copyright: C. Faloutsos (2006) 49

**B-trees – Deletion**

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

15-826 Copyright: C. Faloutsos (2006) 50

**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 (2006) 51

**B-trees – Deletion**

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

15-826 Copyright: C. Faloutsos (2006) 52

**B-trees – Deletion**

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

**FINAL TREE**

15-826 Copyright: C. Faloutsos (2006) 53

**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 (2006) 54

CMU SCS

## Overview

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

15-826 Copyright: C. Faloutsos (2006) 55

CMU SCS

## B+ trees - Motivation

B-tree – print keys in sorted order:

15-826 Copyright: C. Faloutsos (2006) 56

CMU SCS

## B+ trees - Motivation

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

15-826 Copyright: C. Faloutsos (2006) 57

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

15-826 Copyright: C. Faloutsos (2006) 58

CMU SCS

## B+ trees

15-826 Copyright: C. Faloutsos (2006) 59

CMU SCS

## B+ trees - insertion

Eg., insert '8'

15-826 Copyright: C. Faloutsos (2006) 60

CMU SCS

## Overview

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

15-826 Copyright: C. Faloutsos (2006) 61

CMU SCS

## B\*-trees

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

15-826 Copyright: C. Faloutsos (2006) 62

CMU SCS

### B\*-trees: deferred split!

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

15-826 Copyright: C. Faloutsos (2006) 63

CMU SCS

### B\*-trees: deferred split!

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

FINAL TREE

15-826 Copyright: C. Faloutsos (2006) 64

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 (2006) 65

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 (2006) 66



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.]

15-826 Copyright: C. Faloutsos (2006) 67