15-415/615
Database Applications
Fall 2015

HW3: B+ Tree (Recitation)

Anna Etzel
Jinliang Wei
Carnegie Mellon University
Overview

• You are given a basic B+ tree implementation
• **Task**: extend the B+ tree implementation for new operations
• **Goal**: get familiar with B+ tree and recursive code that manipulates the tree & pages
Roadmap

• B+ tree overview
• This B+ tree package
• To be implemented
• Makefile (for your convenience)
• Example insertion algorithm
Why B+ tree?

- Efficient indexing for block I/O devices
- Block I/O devices (e.g. HDD)
  - Access latency is large (in milliseconds)
- B+ tree
  - Large fan out and balanced
  - Shallow – find a key with only a few disk reads
  - B+ tree vs. B tree: Store all values on leaves to for more compact index
- For more details, see the lecture slides
Basic B+ Tree Implementation

• Creates an “inverted index” in the form of a B+ tree
  – key: word, value: document name
• Supports: insert, scan, search, print
• No duplicate keys are allowed
• No support for deletion
• The tree is stored on disk
Roadmap

- B+ tree overview
- This B+ tree package
- To be implemented
- Makefile (for your convenience)
- Example insertion algorithm
B+ Tree Package

• Folders
  – **DOC**: documentation
  – **SRC**: source code
  – **Datafiles**: sample document data
  – **Tests**: test files

• **B-TREE_FILE, POSTINGSFILE, TEXTFILE**, `parms` are created by the b+ tree.
  – Want a new tree? Delete them
B+ Tree Structure
Structure of a Page (def.h)

**PageHdr**
- ‘N’ or ‘L’
- Page Number
- Next Leaf Page No
- NumBytes
- NumKeys
- KeyListPtr
- Ptr to the rightmost child

**KeyRecord**
- Page Number
- KeyLen
- Key Ptr
- Posting Ptr
- Next

- page containing keys < “aaa”
- Posting file

- “aaa”

**KeyRecord**
- Page Number
- KeyLen
- Key Ptr
- Posting Ptr
- Next

- “aab”

- ...
Existing Functions

- **C**: print all the keys
- **i <document_name>**: insert the document
  - key: word, value: document_name
- **p <page_no>**: print the info on the page
- **s <key>**: search the key
- **S <key>**: search the key, and print the documents
- **T**: print the tree
Example code, for searching

- search.c
  - search function entrance, used in main.c
  - calls treesearch to locate the page to which the key belongs

- treesearch.c (study this carefully!)
  - recursive call to locate the page for the key
  - calls FindPageNumOfChild to find the correct children (looks down)

- FindPageNumOfChild.c
  - traverse a non-leaf page
• B+ tree overview
• This B+ tree package
• To be implemented
• Makefile (for your convenience)
• Example insertion algorithm
To be implemented: count (#)

- Should display the number of pages fetched by the command prior to this
- Print and reset
- Hint: Try to leverage existing functionalities
To be implemented: Left & Right Bracket Ops

- Right bracket – “]”
  Print next key without given prefix
- Left bracket – “[”
  Print previous key without given prefix
- Return *NONE* if no key satisfies the requirement
  - Note the *’s
Left & Right Bracket Examples

- Dictionary: {“abraham”, “clara”, “peter”, “peterson”, “tom”}
- [ “pet” -> “clara”
- ] “pet” -> “tom”
- ] “zebra” -> *NONE*
- [ “ab” -> *NONE*
Keep in Mind

● The ‘count’ is not a hard limit.
  ○ May not exactly match the reference count
  ○ Should be reasonable
● A fully sequential scan is a strict no-no
  ○ Will show up as a very large count
● Will be graded to check for efficiency later.
  ○ Again ‘sequential’ scans will be penalized
● Understand the provided infrastructure before starting!
Roadmap

• B+ tree overview
• This B+ tree package
• To be implemented
• Makefile (for your convenience)
• Example insertion algorithm
Build Infra (makefile)

- make load
  - Initialize the tree
  - Insert all datafiles
- make test_sanity
  - Runs load
  - Tests the very minimal functionality. No diffs = test pass!
  - make sure the output is formatted correctly. This is absolutely necessary for autograding.
- make test_leftbracket/make test_rightbracket
  - The questions asked in the handout
Testing Mechanism

• Correctness
  – output the correct list of words (don’t forget to check all the corner cases!)

• Format
  – Make sure the output follows the same format as the sanity test solutions.
Hand-in

• Create a **tar file** of your source code, as well as the makefile. (make handin)
• **Hard-copy** of a document with the functions that you modified/added.
• Please make sure that the “make” command compiles all the source code without any errors
• Submit **your code** on blackboard.
Roadmap

• B+ tree overview
• This B+ tree package
• To be implemented
• Makefile (for your convenience)
• Example insertion algorithm
Insertion Visualized

- insert [3, 7, 13, 9, 1, 6] into an empty tree
- B+ tree is of order 3
- note the number of children allowed

Singleton: [1, b-1]
Root: [2, b]
Internal: [ceil(b/2), b]
Leaf: [ceil(b/2), b-1]
Insertion: 3 -> []
Insertion: 7 -> [3]
Insertion: 13 -> [3, 7]
Insertion: 13 \rightarrow [3, 7]
Insertion: 9 -> [3, 7, 13]
Insertion: 9 -> [3, 7, 13]
Insertion: 1 -> [3, 7, 9, 13]
Insertion: 6 -> [1, 3, 7, 9, 13]
Insertion: 6 -> [1, 3, 7, 9, 13]
Insertion: [1, 3, 6, 7, 9, 13]

Leaf: $[\text{ceil}(b/2), b-1]$
Root: $[2, b]$
Internal: $[\text{ceil}(b/2), b]$
Questions?

• Come to office hours (5 TAs + instructor)
• Read the handout before starting
• Post your questions on blackboard
• Start early