15-415 Database Application  
Spring 2012, Faloutsos  
HW3: Indexing  

TA: U Kang  
Carnegie Mellon University
Overview

• You are given a basic B+ Tree implementation
• Task: extend the B+ tree implementation for new operations
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
B+ Tree Package

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

• B-TREE_FILE, POSTINGSFILE, TEXTFILE, params are created by the b+ tree.
  – Want a new tree? Delete them
B+ Tree Structure

B-TREE_FILE

POSTINGSFILE

TEXTFILE

- american
- usual

- american beauty
- usual suspect
- american history
- X
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

KeyRecord

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

For leaf pages only
For nonleaf pages only

≤ 3
> 3 ≤ 7
> 7

Page containing keys < "aaa"
Posting file
"aaa"

Page containing keys < "aab"
Posting file
"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: search

```c
16 search(key, flag)
17 char *key;
18 int flag;
19 {
20
21 PPOSTINGSPTR treesearch();
22 PPOSTINGSPTR pptr;
23
24 /* Print an error message if strlen(key) > MAXORDERSIZE */
25 if (strlen(key) > MAXORDERSIZE) {
26     printf("ERROR in \"search\": Length of key Exceeds Maximum Allowed\n");
27     printf(" and key May Be Truncated\n");
28 }
29 if (iscommon(key) | {
30     printf("\"%s\" is a common word - no searching is done\n", key);
31     return;
32 }
33 if (check_word(key) == FALSE) {
34     return;
35 }
36 /* turn to lower case, for uniformity */
37 strtolower(key);
38
39 pptr = treesearch(ROOT,key);
40 if (pptr == NONEXISTENT) {
41     printf("key \"%s\": not found\n", key);
42     sqCount++;
43 } else {
44     if(flag) {
45         getpostings(pptr);
46         sqCount++;
47     } else {
48         printf("Found the key!\n");
49     }
50 }
51
52 }
```
Example code: search

- treesearch.c

```c
POSTINGSPTR treeseach(PageNo, key)
PAGENO PageNo;
char *key;
{
    POSTINGSPTR result;
    PAGENO ChildPage;
    struct KeyRecord *KeyListTraverser; /* Pointer to list of keys */
    struct PageHdr *PagePtr;
    PAGENO FindPageNumOfChild();
    struct PageHdr *FetchPage();

    PagePtr = FetchPage(PageNo);

    if (IsLeaf(PagePtr)) {
        result = searchLeaf(PagePtr, key);
    }

    /* The root page contains zero keys */
    else if (((IsNonLeaf(PagePtr)) && (PagePtr->NumKeys == 0)) { /* keys, if any, will be stored in Page# 2
        THESE PIECE OF CODE SHOULD GO soon! */
        result = treeseach(FIRSTLEAFPG, key);
    } else if (((IsNonLeaf(PagePtr)) && (PagePtr->NumKeys > 0)) { /* -christos-: free the space of PagePtr -- DONE! */
        KeyListTraverser = PagePtr->KeyListPtr;
        ChildPage = FindPageNumOfChild(PagePtr, KeyListTraverser,
        key, PagePtr->NumKeys);
        result = treeseach(ChildPage, key);
    }
```

```c
return (result);
```
Example code: search

- **FindPageNumOfChild.c**

```c
13  PAGENO FindPageNumOfChild(PagePtr, KeyListTraverser, Key, NumKeys)
14  struct PageHdr *PagePtr;
15  struct KeyRecord *KeyListTraverser; /* A pointer to the list of keys */
16  NUMKEYS    NumKeys;
17  char       *Key;        /* Possible new key */
18  {
19      /* Auxiliary Definitions */
20      int    Result;
21      char  *Word;        /* Key stored in B-Tree */
22      int    CompareKeys();
23
24      /* Compare the possible new key with key stored in B-Tree */
25      Word = KeyListTraverser->storedKey;
26      if ((Word + KeyListTraverser->Keylen)) = \"0\"
27      Result = CompareKeys(Key, Word);
28
29      NumKeys = NumKeys - 1;
30      if (NumKeys > 0) {
31          if (Result == 2) { /* New key > stored key; keep searching */
32              KeyListTraverser = KeyListTraverser->next;
33              return(FindPageNumOfChild(PagePtr, KeyListTraverser, Key, NumKeys));
34          } else { /* New key < stored key */
35              return(KeyListTraverser->PageNum); /* return left child */
36          } else { /* This is the last key in this page */
37              return(PagePtr->PageToFinalPfgP); /* return rightmost child */
38          }
39      }
40  }
```
To be implemented

- m : print the minimum of all the keys
- M : print the maximum of all the keys
- n : print the number of all the keys
- R : print the list of the keys in the reverse order
- k <ranking> : print the kth largest key
Hint

• Example
  – make demo
    • Initialize the tree
    • Insert sample document.
    • Perform the ‘S american’ command.
Hint

• Sample Tests
  – make test_number
    • Initialize the tree
    • Insert sample document.
    • Perform the ‘n’ command.
    • Store the output to a file.
    • Compare the output to the correct solution (‘diff’ command)
Hint

• Sample Tests
  – make test_minimum
    • Initialize the tree
    • Insert sample document.
    • Perform the ‘m’ command.
    • Store the output to a file.
    • Compare the output to the correct solution (‘diff’ command)
Testing Mechanism

• Correctness
• Efficiency
  – n, R, k : linear algorithm
  – m, M : logarithmic algorithm
• Format
  – Make sure the output follows the same format as the sample test solutions
Hand-in

• Create a tar file of your source code, as well as the makefile
• Please make sure `make’ command compiles all the source code
• Mail to ukang@cs.cmu.edu with the subject “submission homework 3”. 