15-415/615
Database Applications
Spring 2013, Faloutsos

HW3: Indexing

TAs: Danai Koutra
Ekaterina Taralova
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**, parms are created by the b+ tree.
  – Want a new tree? Delete them
B+ Tree Structure

B-TREE_FILE

POSTINGSFILE

TEXTFILE

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

For leaf pages only

For non-leaf pages only

Page containing keys<“aaa”

Posting file

<table>
<thead>
<tr>
<th>Key Len</th>
<th>Key</th>
<th>Posting Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>“aaa”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

 posting file

Page Number
KeyLen
Key Ptr
Posting Ptr
Next

Key Len | Key | Posting Ptr
---------|-----|-------------|
“aab”    |     |             |

posting file

Key Len | Key | Posting Ptr
---------|-----|-------------|
        |     |             |

<table>
<thead>
<tr>
<th>Page Number</th>
<th>KeyLen</th>
<th>Key Ptr</th>
<th>Posting Ptr</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>&gt; 3 ≤ 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Δ
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
search(key, flag)
char *key;
int flag;
{
    POSTINGSPTR treesearch();
    POSTINGSPTR pptr;
    
    /* Print an error message if strlen(key) > MAXWORDSIZE */
    if (strlen(key) > MAXWORDSIZE) {
        printf("ERROR in \"search\": Length of key Exceeds Maximum Allowed\n");
        printf(" and key May Be Truncated\n");
    }
    if( iscommon[key] ) {
        printf("\"s\" is a common word - no searching is done\n",key);
        return;
    }
    if( check_word(key) == FALSE ) {
        return;
    }
    /* turn to lower case, for uniformity */
    strtolower(key);
    
    pptr = treesearch(ROOT,key);
    if( pptr == NONEXISTENT ) {
        printf("key \"s\": not found\n", key);
        uCount++;
    } else {
        if(flag) {
            getpostings(pptr);
            sCount++;
        } else {
            printf("Found the key!\n");
        }
    }
}
```
Example code: search

- `treesearch.c`

```c
12  POSTINGSPTR treesearch( PageNo, key)
13  PAGENO PageNo;
14  char *key;
15  {
16      POSTINGSPTR result;
17      PAGENO ChildPage;
18      struct KeyRecord *KeyListTraverser; /* Pointer to list of keys */
19      struct PageHdr  *PagePtr;
20      PAGENO FindPageNumOfChild();
21      struct PageHdr  *FetchPage();
22      PagePtr = FetchPage(PageNo);
23      if (IsLeaf(PagePtr)) {
24          result = searchLeaf(PagePtr, key);
25      }
26      /* The root page contains zero keys */
27      else if ((IsNonLeaf(PagePtr)) && (PagePtr->NumKeys == 0)) {  
28          /* keys, if any, will be stored in Page# 2
29          THESE PIECE OF CODE SHOULD GO soon! **/
30          result = treesearch(FIRSTLEAFPG, key);
31      } else if ((IsNonLeaf(PagePtr)) && (PagePtr->NumKeys > 0)) {  
32          KeyListTraverser = PagePtr->KeyListPtr;
33          ChildPage = FindPageNumOfChild(PagePtr, KeyListTraverser,
34              key, PagePtr->NumKeys);
35          result = treesearch(ChildPage, key);
36      }
37      /* -christos-: free the space of PagePtr - DONE! */
38      FreePage(PagePtr);
39      return ( result);
40  }
```

15-415 Database Applications
Example code: search

- **FindPageNumOfChild.c**

```c
18  PAGENO FindPageNumOfChild(PagePtr, KeyListTraverser, Key, NumKeys)|
19  struct PageHdr *PagePtr;
20  struct KeyRecord *KeyListTraverser; /* A pointer to the list of keys */
21  NUMKEYS NumKeys;
22  char *Key;                /* Possible new key */
23  {
24   /* Auxiliary Definitions */
25   int Result;
26   char *Word;              /* Key stored in B-Tree */
27   int CompareKeys();
28
29   /* Compare the possible new key with key stored in B-Tree */
30   Word = KeyListTraverser->KeyListTraverser->Key;
31   (*(&Word + KeyListTraverser->KeyLen)) = '\0';
32   Result = CompareKeys(Key, Word);
33   NumKeys = NumKeys - 1;
34
35   if (NumKeys > 0) {
36     if (Result == 2) { /* New key > stored key: keep searching */
37       KeyListTraverser = KeyListTraverser->Next;
38       return(FindPageNumOfChild(PagePtr, KeyListTraverser, Key, NumKeys));
39     } else /* New key <= stored key */
40       return(KeyListTraverser->PgNum); /* return left child */
41   } else /* This is the last key in this page */
42     {
43       if ((Result == 1) || (Result == 0)) /* New key <= stored key */
44         return(KeyListTraverser->PgNum); /* return left child */
45       else /* New key > stored key */
46         return(PagePtr->PtrToFinalPg); /* return rightmost child */
47     }
48  }
```

15-415 Database Applications
To be implemented

• **f**: print all the distinct keys within the range defined by \([\text{word1}, \text{word2}]\) in alphabetical order

• **b**: print all the distinct keys within the range defined by \([\text{word1}, \text{word2}]\) in *reverse* alphabetical order
Hint

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

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

• Sample Tests
  – make test_rangeRev
    • Initialize the tree
    • Insert sample document.
    • Perform the ‘b’ command.
    • Store the output to a file.
    • Compare the output to the correct solution
      ("diff" command)
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 sample test solutions.

• **NOTE**: you are not allowed to store the words (in an array) in alphabetical order, and just print them backwards for command ‘b’. Use the structure of the B+ tree.
Hand-in

• Create a **tar file** of your source code, as well as the makefile.
• **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.
Questions?

- Come to office hours (5 TAs + instructor)
- Please do not send personal emails.
- Post your questions on blackboard.