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

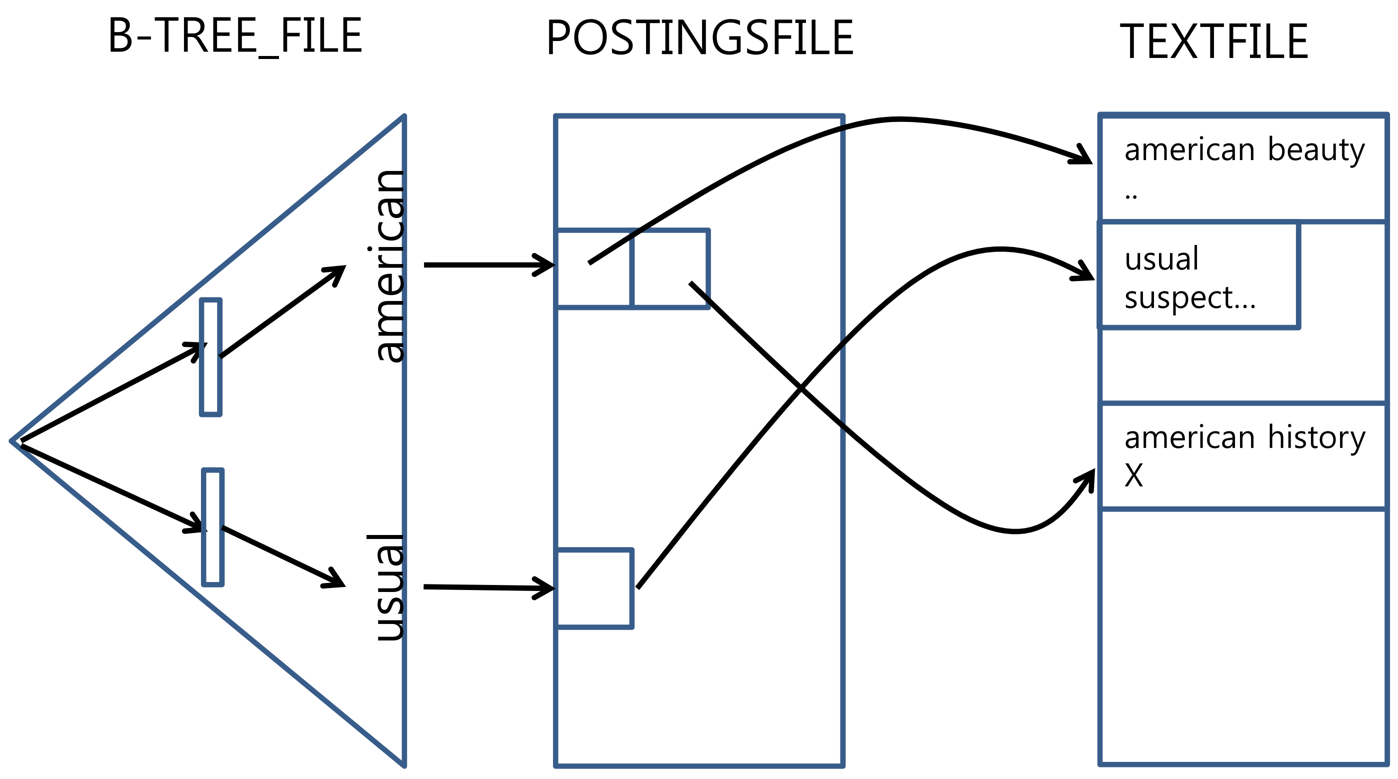
Basic B+ Tree Implementation

- 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

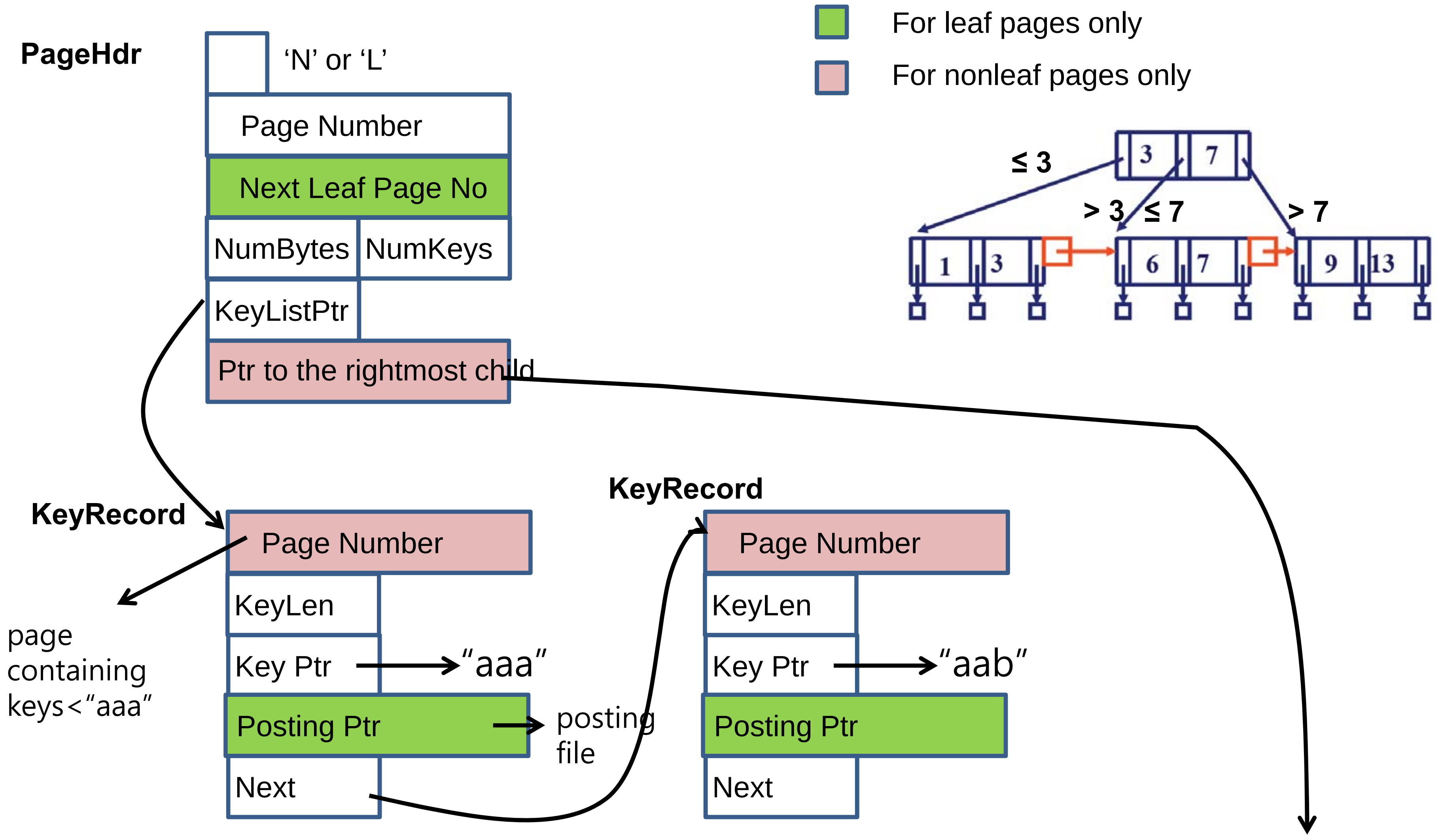
B+ Tree Package

- Folders
  - DOC: documentation
  - SRC: source code
  - Datafiles : sample documents data
  - Tests: test files
- B-TREE_FILE, POSTINGSFILE, TEXTFILE, parms are created by the b+ tree.
  - Want a new tree? Delete them

B+ Tree Structure



Structure of a Page (def.h)



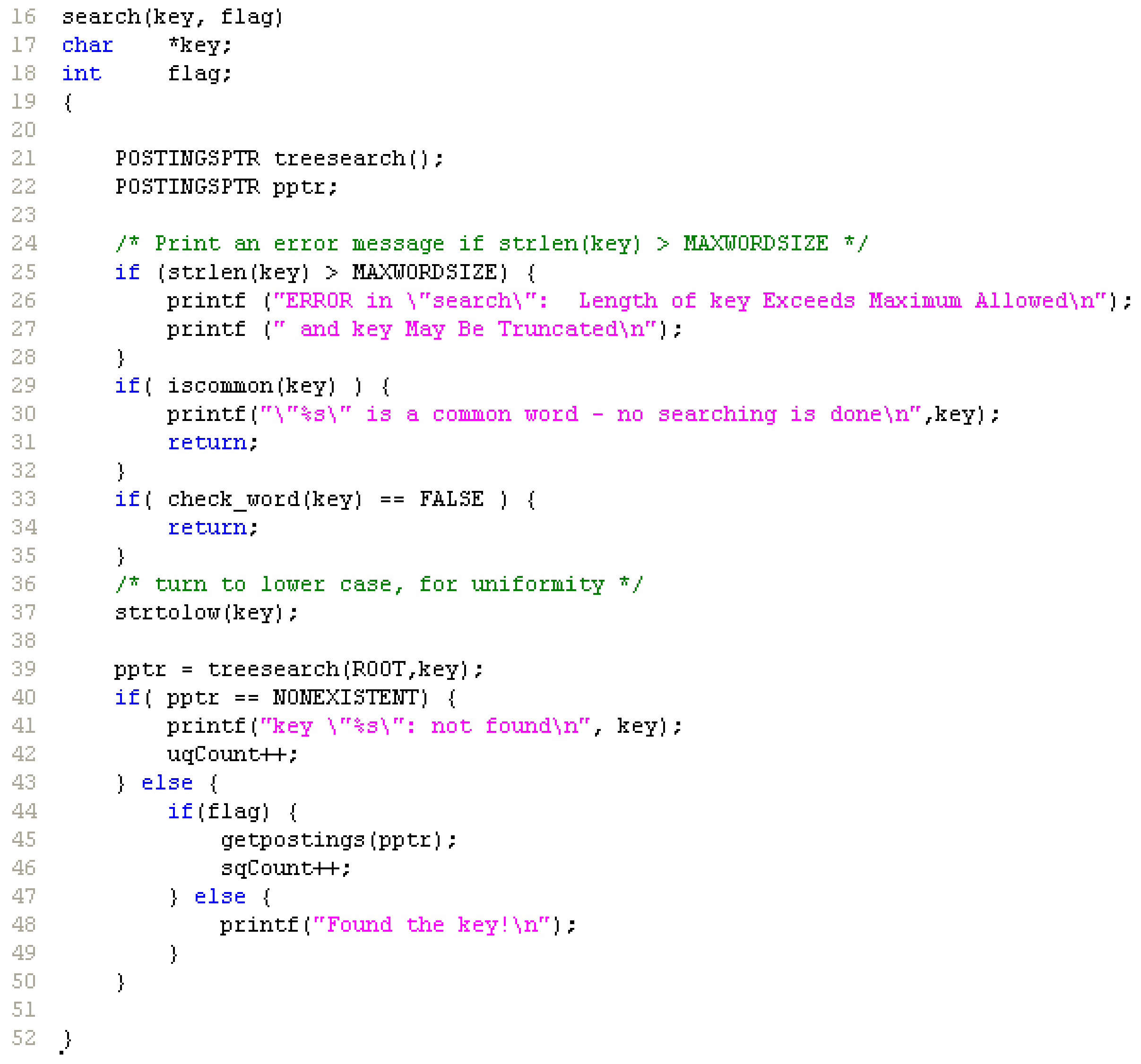
Existing Functions

- 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

Demo

Example code : search

- search.c



```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 */
    strtolow(key);

    pptr = treesearch(ROOT,key);
    if( pptr == NONEXISTENT) {
        printf("key \"%s\": not found\n", key);
        uqCount++;
    } else {
        if(flag) {
            getpostings(pptr);
            sqCount++;
        } else {
            printf("Found the key!\n");
        }
    }

}
```

Example code : search

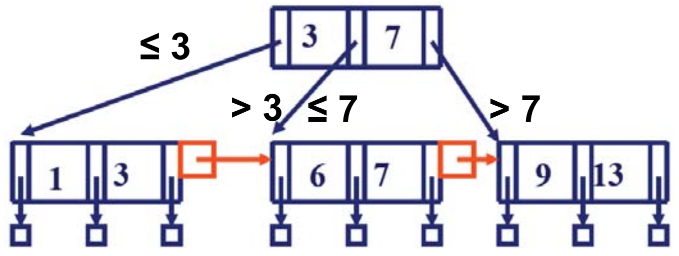
- treesearch.c

```c
POSTINGSPTR treesearch( 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 = treesearch(FIRSTLEAFPG,key);
    } else if ((IsNonLeaf(PagePtr)) && (PagePtr->NumKeys > 0))  {
        KeyListTraverser = PagePtr->KeyListPtr;
        ChildPage = FindPageNumOfChild(PagePtr,KeyListTraverser,
                                      key,PagePtr->NumKeys);
        result = treesearch(ChildPage,key);
    }
    /* -christos-: free the space of PagePtr - DONE! */
    FreePage(PagePtr);
    return ( result);
}
```



Example code : search

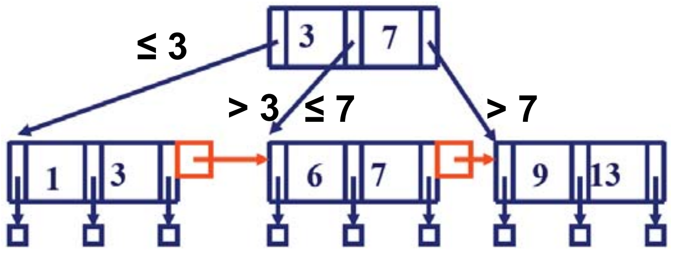
- FindPageNumOfChild.c

```
PAGENO FindPageNumOfChild(PagePtr,KeyListTraverser,Key,NumKeys)
struct PageHdr   *PagePtr;
struct KeyRecord *KeyListTraverser; /* A pointer to the list of keys */
NUMKEYS           NumKeys;
char             *Key;              /* Possible new key */
{
    /* Auxiliary Definitions */
    int   Result;
    char *Word;                    /* Key stored in B-Tree */
    int CompareKeys();


    /* Compare the possible new key with key stored in B-Tree */
    Word = KeyListTraverser->StoredKey;
    (*(Word + KeyListTraverser->KeyLen)) = '\0';
    Result = CompareKeys(Key, Word);

    NumKeys = NumKeys - 1;

    if (NumKeys > 0) {
        if (Result == 2) {     /* New key > stored key:  keep searching */
            KeyListTraverser = KeyListTraverser->Next;
            return(FindPageNumOfChild(PagePtr,KeyListTraverser,Key,NumKeys));
        } else                 /* New key <= stored key */
            return(KeyListTraverser->PgNum); /* return left child */
    } else                     /* This is the last key in this page */
    {
        if ((Result == 1) || (Result == 0)) /* New key <= stored key */
            return(KeyListTraverser->PgNum);  /* return left child */
        else                   /* New key > stored key */
            return(PagePtr->PtrToFinalRtgPg);  /* return rightmost child */
    }
}
```



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.

Demo

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)

Demo

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)

Demo

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”.