Overview

• You are given a basic B+ Tree implementation
  • Task: extend it, for new operations

• Today:
  — Brief intro to B+ trees
  — Overview of the code
  — Homework description
Brief intro to B+ Trees

- N-ary search trees
- Highly used in Databases
- Similar to B-trees
  - Each non-leaf node contains only keys
  - Leaf nodes contain key-value pairs
  - Leaf nodes form a linked list

Basic search in B+ Trees

Search for key ‘14’
Basic search in B+ Trees

Start from root

14<17

Root

Find the correct child node

14>5

Basic search in B+ Trees
Basic search in B+ Trees

1. Start at root.
2. Select right child of '13' (14 > 13).
3. Found it!

Example of Non-leaf Redistribution:
- Tree is shown below during deletion of 24*.
- Now, we can (and must) redistribute keys.

After Redistribution:
- Need only redistribute 20*; did 17* too.
- Why would we want to redistribute more keys?

Found it!
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

• File: parms
  128  // page size in bytes – leave it as is
  1.618  // ignore it: (expansion rate for postings list)
• Folders
  – DOC: documentation
  – SRC: source code
  – Datafiles: sample documents data
  – Tests: test files
B+ Tree Package

- B-TREE_FILE, POSTINGSFILE, TEXTFILE, 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

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

**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

**search.c**

```c
search(key, flag)
{
    char *pkey;
    int flag;
    // ...
}
```

**treesearch.c**

```c
int treesearch()
{
    // ...
}
```
Example code: search

- **FindPageNumOfChild.c**

```c
/*...*/

```/n

In this homework

- implement 2 different types of string search
  - Prefix search:  
    P organ // matches ‘organism’, ‘organs’, ...
  - Substring search (‘M’iddle of word)  
    M organ // matches ‘morgana’, ‘organs’,
- Also report  
  - count #pages read
Testing Mechanism

Correctness
• Run your code against the given test files:
  – `diff` should be empty
• Make sure you also test on your own datasets:
  – Empty B-tree
  – Only one key value in B-tree
  – Root just split
  – ...

Our output
Your output

Hand-in

• **Hard copy** (in class):
  A. Print answers to the questions
  B. ONLY new/changed code (save the trees 😊)

• **Electronic copy** (Blackboard):
  A. `btree_andrewId.tar` file with only the necessary files & the makefile.

  % tar xvf; make # should run your code on
  # sample scripts
Hand-in

FYI, we will use scripts to grade, thus please follow carefully the instructions for the output format!

Questions?

• Come to office hours (4 TAs + 2 instructors)
• Post your questions on blackboard.