**CSED 2014**

12

**B+ Indexing**

**Ashraf Saleh Mohamed (20)**

**Hazem Ahmed Mousa (32)**

Problem Statement

1. implementation a B+ tree in which leaf level pages contain entries of the form *<key, rid of a data record>*
2. implementation the full search and insert algorithms as discussed in class.
3. implement two page-level classes, **BTIndexPage** and **BTLeafPage.**

Implementation Issues

One of the issues we met is how to organize the header page to carry some information about the B+ tree. The Header page is a HFPage organized as follow:

* nextPage = Root page
* first record = Key type
* Second record = key size
* Third record = delete fashion

Pseudo code   
Scan class

# Algorithm : BTFileScan (header, lower, upper)

{

Page 🡨 root //get it from header page

If (lower = null) // it means to start with the first entry

Traverse down the tree to get the leftmost leaf node

Set the rid with the first record in this page

Else if (lower > upper)

Swap lower with the upper

If (page is index)

While (page!= leaf)

Traverse this index page to get a link to next level

// now page is leaf

Traverse the page until holding the right record

}

# Algorithm : get\_next()

{

If (rid == null) // no more records in the current page   
 leaf 🡨 next leaf page

Rid 🡨 page.first Rec

If (upperkey = null && rid = null)  
 return null //all range is covered

Else (current keyData entry > upper)

Return null // all range is covered

Return current keyDataEntry (dEntry)

}

# Algorithm : KeySize()

{

Return the key size from the header page ;

Second record in the header page represents the key size ;

}

# Algorithm : delete\_current()

{

Deletes the current page by calling the .deleteEntry(rid) of the current leaf page using the current rid

}

BTreeFile class

Algorithm insert: insert (nodepointer, entry, newchildentry)  
  
proc / / Inserts entry into subtree of \*nodepointer; degree is d;  
/ /'newchildentry' null initially, and null on return unless child is split  
if \*nodepointer is a non-leaf node, say N,  
then  
find'i such that k(i) S entry's key value < J(i+1); / / choose subtree  
insert(.R;, entry, newchildentry); / / recursively, insert entry  
if newchildentry is null,  
then  
return; / / usual case; didn't split child  
else, / / we split child, must insert \*newchildentry in N  
if N has space, / / usual case  
then  
put \*newchildentry on it, set newchildentry to null, return;  
else,   
split N: / / 2d + 1 key values and 2d + 2 nodepointers  
first d key values and d + 1 nodepointers stay,  
last d keys and d + 1 pointers move to new node, N2;  
/ / \*newchildentry set to guide searches between Nand N2  
newchildentry = & ((smallest key value on N2,  
pointer to N2));  
end if;  
end if;  
if N is the root, / / root node was just split  
then  
create new node with (pointer to N, \*newchildentry);  
make the tree's root-node pointer point to the new node;  
return;  
end if;  
else //\*nodepointer is a leaf node, say L,  
if L has space, / / usual case  
then  
put entry on it, set newchildentry to null, and return;  
else, / / once in a while, the leaf is full  
split L: first d entries stay, rest move to brand new node L2;  
newchildentry = & ((smallest key value on L2, pointer to L2));  
set sibling pointers in Land L2;  
return;  
end if;  
end if;  
endproc

Algorithm :Delete(key,rid)

traverse the tree recursively to get the suitable leaf page  
remove the record from this leaf page