CS 3310 Data and File Structure

Instructor Dr.Ajay K. Gupta

Name: Mariam Ghali

**Btrees**

**Phase 1: objective**

1. The main goal of the assignment is to practice building and searching B-Trees
2. Practice developing high-performance solutions
3. See the difference of the implementation of the 2-3 trees and 2-3-4 trees
4. Practice the level by level searching for the BTrees

**Phase 2: Description**

The program consists of 6 classes

1. The main class will read the acsii file and create 4 different 2-3 trees for it and 2-3-4 trees too based on each column: decimal, char, hex, and oct. After it creates the 4 different trees it ask the user for an input to search for in the four trees. The result of the searching is the values you searched for and the tree we searched in with the other attached row from the ascii table. So for example if we searched for 1 it will search in decimal tree, hex tree, oct tree and char tree and return the corresponding row for each of them. It will also return the time that each search took for searching the value.
2. In both TTNode and TTTNode classes, it will create the keys, values and childs for each node. In the TTNode which is for the 2-3 tree , it will create the right and the left keys with the values and having method to set those values. Then it will have the left , right and the center child. In the TTTnode which is for 2-3-4 tree, will have one more key with one more value and for the child it will have one more child. So total it will have 4 child and 3 keys.
3. It will also have the insertion method. So if the root is null, then there is no tree created and we create the first node. If it is a leaf then we check if the left key is empty then we insert the value in the left key. If the left key is not empty check the right if it isn’t empty insert. If it is then create a child if it didn’t exist check if the left or the right key is empty. Same with the right and center child. Once we reached a full keys of the child then we need to spilt.
4. In the TowTreeTree and the TwoThreeFourTree we call the insert function and we create the search method in the tree. So we first check if the tree exist. Then check the left key if it exists. If it is, then check its value with the value you are looking for. If they are the same then return it. If not then check the right key. If not check the left key of the left child if it exists. Then the right key. Then check the center child then the right child. If you didn’t find it return not found.
5. ReleveantValues is a class and we create and object out of it to hold the values of each keys depends of the tree you are creating.
6. **Pseudocode**
7. Main method

Enter a value n

Try open acsii file

While(Line is not empty)

Read each line

Split by the tap to decimal , hex, oct and char

Insert in the decimal tree

Insert in the hex tree

Insert in the char Tree

Insert in the oct tree

Catch

File not found

Call the Travers method for the decimal tree

Call the Travers method for the hex tree

Call the Travers method for the oct tree

Call the Travers method for the char tree

If n is number

Call the find method of the decimal tree

Else

No decimal number of the input

If N is oct number

Call the find method of the oct tree

Else

No oct number of the input

If N is hex number

Call the find method of the hex tree

Else

No hex number of the input

If Call the find method of the oct tree is not null

Call the find method of the char tree

Else

No char number of the input

Boolean isNumber

Return the value of it matches with the integers from 0 to 9

Boolean isOct

Return the value of it matches with the integers from 0 to 7

Boolen isHex

Return the value of it matches with the integers from 0 to 9 or from a to z or from A to Z

1. TTNode class

TTNode

Set first = last = center to null

TTNode

Initialize for the left, right keys and the children

isLeaf

return left = null

TTNode<key K , E> lchild

Set the left child to left

TTNode<key K , E> lchild

Set the center child to center

TTNode<key K , E> rchild

Set the right child to right

TTNode insert

If root is null

Create a node and set the key and the value

If isLeaf

Insert to the key

If the left key dosen’t exists

Create one and add the value ot the left key

If the right key is null

Create a key and add the value to the right

TTNode add

If left key is smaller than the one we insert

Then insert to the right if it is null

If it is not null

Split

If the key we insert is between the left and the right

Insert to the center if it is not null

If the key is bigger

Insert to the right

1. TwoThreeTree

Insert

If node is null

Call the inserthelp from TTNoode

Create a node and add the keys to it

Else

Find where we add the new node

FindHelp

If the value we are looking for is similar to the value in the root

Return the value

If the value we are looking for equal to the value of the left key

Return the value

If the value we are looking for equal to the value of the right key

Return the value

If the value is smaller than the left key

Search in the left child

If the value is between the right and the left key

Search in the center

If the value is bigger than the right

Search in the right child

Travers

Create a Queue

If not left or right key

Tree doesn’t exit

Add the root to the queue

While (queue is not empty )

Remove the root

If root is the leaf

Print the left and the right key

Else

Print the parent left and right key, left ,right, and center children

Enqueue the left child

Enqueue the right child

Enqueue the center child

1. RelevantValue class

RelevantValue

First value for the key

Second value for the key

Third value of the key



**Thermotical analysis**

As we did before for the BST Trees. BST trees is a version of the BTrees but it is unbalanced. For this assignment. 2-3 Tress is a balanced tree that will have 2 keys at most and 3 child at most. The insertion for the 2-3 tree is log(n) and the search is log2(n). For the 2-3-4 trees we have 3 keys at most and 4 children. The time complexity for 2-3-4 tress insertion is also log(n) and the search is log3(n). So every time we add one more key, log(n) where n is the number of the keys.

**The empirical analyses**

It agrees with the thermotical analysis by showing that the graph of the difference between the 4 trees: decimal, hex, oct and the char trees. As the number of how many times the user search for the value. The times increase no matter if the value is search for hex,oct,decimal or char. And if the value is integer, it is more likely that the time increase because the number can be either hex, decimal, char and oct.