Lab Assignment #3 Deadline Code: 6th, July, Wed 11:59pm

Submitted by: Oscar I. Ricaud Report & Demo: 12th, July, Tues 12:30pm

ID: 80466791

Email Address: [oiricaud@miners.utep.edu](mailto:xyz@miners.utep.edu)

Class Time: Mon- Fri 10:55 am – 12:00 pm



**Introduction:**

In this lab the main task is to implement a *B-Tree*. A B-Tree has a time complexity O (log n) with the following properties: Let some *node* x, be the root of an empty tree and and let *T* be an integerto obtain the bounds of keys each node can have. For example, if T = 3, the maximum number of keys for node x is 2T-1 = 5 keys. The minimum number of keys is T-1 = 2 keys. Therefore, each node can contain a size of array between 2 < Array Size < 5 and where some node x key[i] is where the integer is stored. With that being said the following was implemented:

**Proposed Solution:**

**Define classes**

1. *Main method Class: will contain a node x with a height, and a counter and call the BTreeNode class.* 
   1. Initialize BTree Class
   2. Ask for int T = 3
   3. Ask for List = {6, 3, 16, 11, 7, 17, 14, 8, 5, 19, 15, 1, 2, 4, 18, 13, 9, 20, 10, 12}
   4. for i to list size

BTree insert (list get at i)

* 1. end for
  2. end Main
  3. x.TreeInfo()

1. *BTree Class: will contain a node x with a height, counter.* 
   1. BTreeNode root
   2. Int Height
   3. Int counter = 0
   4. int T
   5. **BTree( int t )**

Root = new node(t)

T = t

Height = 0

**end BTree**

* 1. **insert (int new key )**

if (root is full)

* + - 1. print root is full
      2. split the current root // call a method call split
      3. height++

root.insertNonFull(new key) // insertion will happen in the BTree Node class

* 1. **end insert method.**
  2. **split()**
     1. Left Child = new BTreeNode(T)
     2. Right Child = new BTreeNode(T)
     3. Left Child.n && Right Child.n = T.n – 1
     4. Median = T – 1
     5. //Traverse left child first up to the median and obtain the children from the root if any and
     6. // Obtain the keys up to the mid element
     7. for i = 0 *TO* T-1
     8. left child.c[i] = root c [i]
     9. left child key[i] = root key[i]
     10. end for
     11. leftChild.c[median] = root.c[median]
     12. Similarly do the same for the right child except start from the median + 1
     13. for j = median + 1 to the number of elements in the root
     14. right child.c[j-median+1] = root.c[j]
     15. rightt key.c[j-median+1] = root.c[j]
     16. end for
     17. rightChild.c[median] = root.c[root.n]
     18. // Assign the median to be the new root
     19. root.key[0] = median
     20. // Shorten the number of keys in the root
     21. root.n = 1
     22. // Assign the children of the new root
     23. root.c[0] = left child
     24. root.c[1] = right child
     25. // Now since the root has children now it cannot be a leaf
     26. rootisLeaf = false
  3. **end split**
  4. **printHeight ()**

return height

* 1. **end printHeight()**
  2. **printAscendingOrder()**
     1. Root.PrintAscendingOrder() // call method from the BTreeNode class
  3. **end printAscendingOrder**
  4. **printNodes()**
     1. root.printNodes()
  5. **end printNodes**
  6. **printDescendingOrder()**
     1. root printDescendingOrder()
  7. **end printDescendingOrder**
  8. **findMin()**
     1. root.findMin()
  9. **end findMin**
  10. **findMax()**
      1. root.findMax()
  11. **end findMax**
  12. **findNumberofNodesDepthD(BTreeNode, depth)**
      1. node.findNumberofNodesinDepthD(depth)
  13. **end findNumberofNodesDepthD**
  14. **findNumberofNodes(BTreeNode node)**
      1. if( root is a leaf)
      2. counter++
      3. else
      4. counter++
      5. for(i = 0 to the number of keys in the root + 1)
      6. if( child node is not a leaf)
      7. findNumberofNodes(node.c[i])
      8. end if
      9. else
      10. counter++
      11. end for
      12. return counter
  15. **end findNumberofNodes**
  16. **findNumberofKeys(BTreeNode node)**
      1. if( root is a leaf)
      2. for( i = 0 to the number of keys in the current node)
      3. counter++
      4. end for
      5. else // Count the number of keys starting at the root
      6. for( i = 0 to the number of keys in the current node/root)
      7. count++
      8. findNumberofKeys(node.c[i])
      9. end for
      10. findNumberofKeys(node.c[node.n]
      11. end else
      12. return counter
  17. **end findNumberofNodes**
  18. **findAllSumKeys(BTreeNode node)**
      1. if( root == null)
      2. return counter
      3. else
      4. if(node is a leaf)
      5. counter = counter + node.findAllSumKeys() // call node class
      6. else
      7. for( i = 0 to the number of keys in the current node)
      8. counter = counter + node.key[i]
      9. end for
      10. for( i = 0 to the number of keys in the current node + 1)
      11. findAllSumKeys(node.c[i])
      12. end for
      13. return counter
  19. **end findAllSumKeys**
  20. **exactlyNKeys(BTreeNode node, int keys)**
      1. if( node == null)
      2. return counter
      3. else
      4. if(number of elements in the node == keys)
      5. counter ++
      6. end if
      7. for( i = 0 to the number of keys in the current node + 1 )
      8. exactlyNkeys(node.c[i], keys)
      9. end for
      10. return counter
  21. **end exactlyNKeys**
  23. **numberofMinKeys(BTreeNode node)**
      1. if( node == null)
      2. return counter
      3. else
      4. if(number of elements in the node == T-1)
      5. counter ++
      6. end if
      7. for( i = 0 to the number of keys in the current node + 1 )
      8. numberofMinKeys(node.c[i])
      9. end for
      10. return counter
  24. **end numberofMinKeys**
  26. **numberofNodesFull (BTreeNode node)**
      1. if( node == null)
      2. return counter
      3. if(node.isFull)
      4. counter ++
      5. else
      6. for( i = 0 to the number of keys in the current node + 1 )
      7. numberofNodesFull (node.c[i])
      8. end for
      9. return counter
  27. **end numberofNodesFull**
  28. **numberofLeaves (BTreeNode node)**
      1. if(node.isLeaf)
      2. counter ++
      3. else
      4. for( i = 0 to the number of keys in the current node + 1 )
      5. numberofLeaves (node.c[i])
      6. end for
      7. return counter
  29. **end numberofLeaves**
  30. **printChildren(BTreeNode node)**
      1. if(node.isLeaf)
      2. Print [
      3. for(i = 0 to number of keys in the current node)
      4. print key [i]
      5. end for
      6. print ]
      7. else
      8. print [ + node.key[0] + ]
      9. for i = 0 to <= number of keys in the current node
      10. printChildren(node.c[i])
      11. end for
  31. **end printChildren**
  32. **printRoot (BTreeNode node)**
      1. for i = 0 to < number of keys in the current root
      2. print root.key[i]
      3. end for
  33. **end printRoot**

1. *BTreeNode Class: The same node x will contain y generation children starting at child 0 up to child number of elements node x contains – 1.* 
   1. *Int [] key // stores the keys in the current node*
   2. BTreeNode children
   3. *Boolean is leaf*
   4. *int n // the number of keys in the current node*
   5. *int T // each node has at least T-1 and at most 2T-1 keys*
   6. *int counter = 0*

*BTreeNode Parameters:* *int t*

1: T = t

2: isLeaf = true

3: key = new int [2\*T-1]

4: c = new BTreeNode[2\*T]

5: n =0

*BTreeInsertNonFull (int new key)*

1. int i = -1 // the current keys that are stored
2. if(isLeaf)
3. while(i > = 0 && newKey < key[i] )
4. // In other words the while loop is checking for elements that are greater than the new key.
5. // If you are inserting the first element to the tree then this while loop will get skip but if you are
6. // adding the second, third elements we must check that the root will always be in ascending order.
7. Key[i+1] = key[i]
8. i--
9. End while
10. n++
11. key[i+1] = newKey
12. else // that means that the current node is not a leaf so we must check to which child we must add the new element
13. // So essentially what this while loop does is compare the elements in the current node and if the current key is greater
14. // than the new key we get in the while loop and then we add the elements in the most left child
15. while(i > = && new key< key[i]
16. i--
17. end while
18. int insertChild = i + 1
19. if(c[insertChild]. Is full ) // we must first check if the current child is full
20. n++
21. c[n] = [n-1] // this copies lets say new child[3] = child[2]
22. for( j = number of elements in the current node – 1 TO insertChild, j--)
23. c[j] = c[j-1] // then child[1] = child[0] and key[2] = key[1]
24. key[j] = key[j-1]
25. end for
26. key[insertChild] = c[insertChild].key[T-1] // Cut and paste that’s pretty much what its doing
27. // Pretty much what this means is we’re cutting the full child in half and copying the new father into the existing fathers
28. c[insertChild].n = T-1
29. // The following will be our new right child
30. // c[insertChild].c[k+T] = c[insertChild.c[T]
31. BTreeNode newNode = newBTreeNode (T)
32. for ( k = 0 to T-1)
33. new node .c[k] = c[insertChild].c[k+T]
34. new node.key[k] = c[insertChild].key[k+T]
35. end for
36. newNode.c[T-1] = c[insertChild].c[2\*T-1]
37. newNode.n = T-1
38. newNode.isLeaf = c[InsertChild].isLeaf
39. // if the new key is less than key[0]
40. if(newKey < key[insertdChild])
41. c[insertChild].BTreeInsertNonFull(newKey)
42. if(newKey > key[insertChild])
43. c[insertChild+1].BTreeInsertNonFull(
44. end if
45. else // meaning the child we are trying to insert is non full
46. c[insertChild].BTreeInsertNonFull(newKey)

*printAscendingOrder():*

1. if(isLeaf)
2. for(int i = 0 TO number of elements in the node )
3. print key[i]
4. end for
5. else
6. for(i = 0 TO number of elements in node)
7. c[i].PrintAscendingOrder()
8. print key[i]
9. end for
10. c[n].printAscendingOrder();

*printNodes():* Prints in preorder

1. for(int i = 0 to n )
2. if(i == 0)
3. print [
4. print key[i]
5. if(is not a leaf)
6. for ( j = 0 to n)
7. c[j].printNodes()
8. end for
9. if(i == n-1)
10. print ]
11. end for
12. end method

*printDescendingOrder():*

1. if(is leaf )
2. for( i = number of elements – 1 to 0 **)**
3. print key[i]
4. end for
5. else
6. for(int i number of elements – 1
7. c[i+1 ] . printDescendingOrder()
8. print key[i]
9. end for
10. c[0].printDescendingOrder()

*printFindMin ():*

1. if(is a leaf)
2. simply just return key[0]
3. else
4. return c[0].findMin()

*printFindMax ():*

1. if(isLeaf)
2. return key[n-1] **// return the last key of the list**
3. **else**
4. **return c[n].findMax();**

*findAllSumKeys ():*

1. **for(int i = 0 to the number of keys )**
2. **counter = counter + key[i]**
3. **end for**
4. **return counter**

*numOfNodesFull ():*

1. for(i = 0 to the number of keys in the node)
2. **if(isLeaf && c[i] is full )**
3. **counter++**
4. **end for**
5. **return counter**

*printChildren ():*

1. if(isLeaf)
2. for(i = 0 to the number of keys stored in nodes
3. print key[i]
4. end for
5. end method

*numberofNodesDepthD( int depth )*

1. count = 0
2. if depth == 0
3. **count = 1**
4. **else**
5. **if(is not a leaf)**
6. **for(int i = 0 i to the number of keys in the node)**
7. **count = count + c[i].numberofNodesDepthD(depth -1)**
8. **end for**
9. **return count**

*search (int SearchKey):*

1. if(is leaf )
2. **for(int i = 0 to n )**
3. **if(key[i] == searchKey**
4. **return found key**
5. **end for**
6. **else**
7. **for(int i = 0 to n)**
8. **if(key[i] == searchKey)**
9. **print found**
10. **else**
11. **c[i].search(serachkey)**
12. **c[n].search(searchKey)**
13. **end for**
14. **end method**

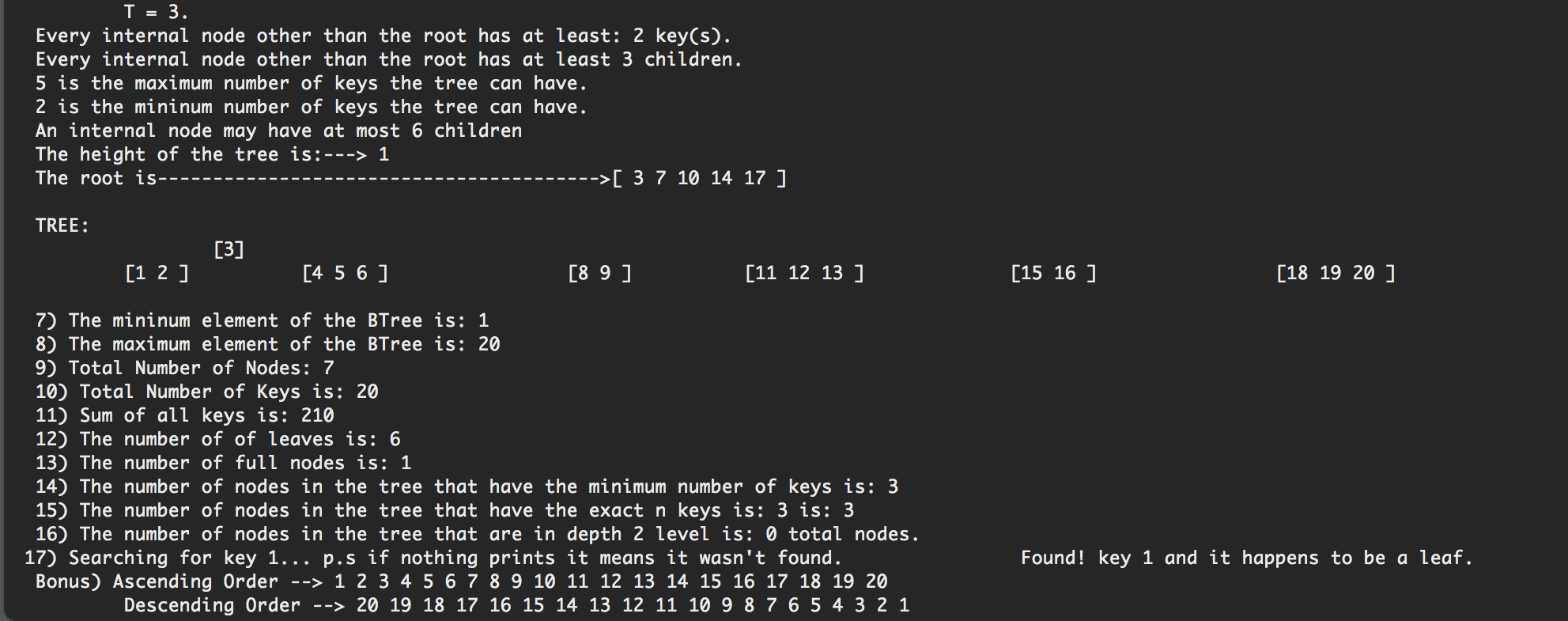
*deletions ():*

1. finish this

Input:

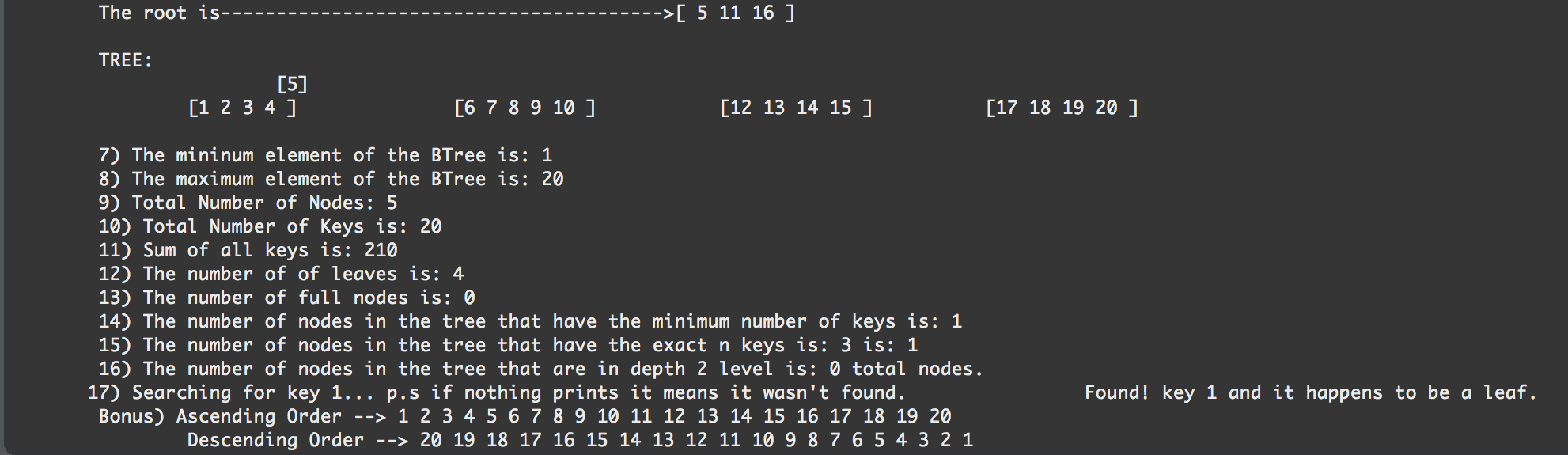
T = 3

List = {6, 3, 16, 11, 7, 17, 14, 8, 5, 19, 15, 1, 2, 4, 18, 13, 9, 20, 10, 12}



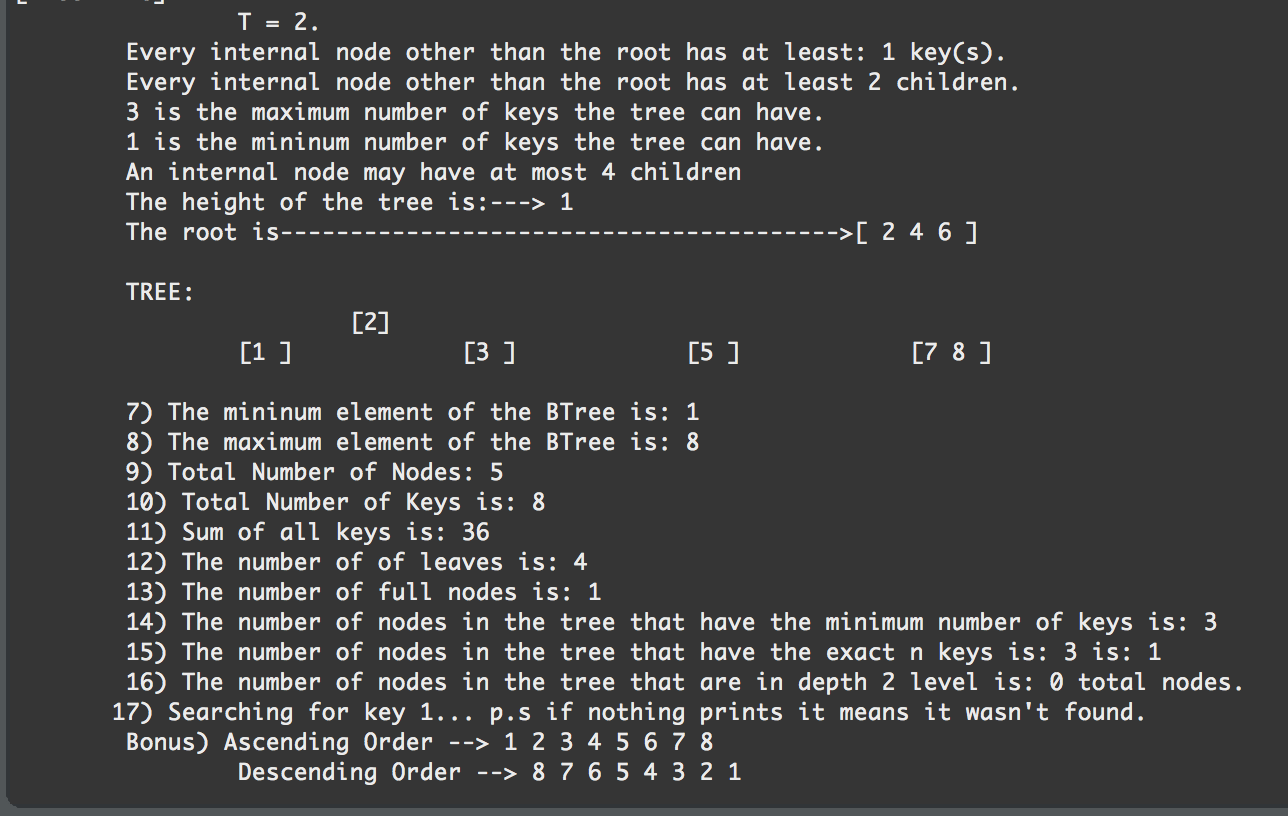
T = 4

List = {6, 3, 16, 11, 7, 17, 14, 8, 5, 19, 15, 1, 2, 4, 18, 13, 9, 20, 10, 12}



T = 2

List = {1, 2, 3, 4, 5, 6, 7, 8}



Appendix: ­

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //

// Author: Oscar Ivan Ricaud.

/\*Assignment: Lab 3

Instructor: Professor Julio Urenda

TA: Saiful Abu

Course 2302

Data of Last modification July 8th, 2016

\*/

// Program purpose:

/\*Is to be able to understand how B-Tree's work. I implemented many methods including search, insertion, count the number of leaf's and

\* much more. Note~ Deletion method needs to be completed

// How to operate BTree.java:

/\* Simply click the play button

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //

package lab3beta2;

public class TestHere {

public static void main(String[] args) {

// Create BTree x

BTree x = new BTree(3);

System.*out*.println("");

int [] list = {6, 3, 16, 11, 7, 17, 14, 8, 5, 19, 15, 1, 2, 4, 18, 13, 9, 20, 10, 12};

// Insert Elements to the tree

for(int i = 0; i < list.length; i++){

x.insert(list[i]);

}

System.*out*.println("Below is just a walkthrough of how the B-TREE is being implemented. ");

System.*out*.println("Everything works except Searching and Deletion I have not given up! I will keep working on it.");

//Tree Information is at the bottom of the console!

x.treeInfo();

x.printAscendingOrder();

System.*out*.println("");

x.printDescendingOrder();

}

**package lab3beta2;**

**public class BTree {**

**private BTreeNode root;**

**private int T;**

**public int height;**

**public int counter = 0;**

**public BTree(int t){**

**root = new BTreeNode(t);**

**T = t;**

**height = 0;**

**}**

**public void printHeight(){**

**System.*out*.println("Tree height is "+height);**

**}**

**public void insert(int newKey){**

**System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\* begin");**

**System.*out*.println("Inserting new key " + newKey);**

**if(root.isFull()){**

**System.*out*.println("Root is full:");**

**//System.out.print("root --> ");**

**//root.print();**

**//System.out.println("");**

**//int median = root.key.length/(2);**

**// int promote = root.key[median];**

**// System.out.println("Root is full we must promote the mid node: " + promote);**

**split();**

**height++;**

**}**

**root.BTreeInsertNonFull(newKey);**

**System.*out*.print("print current node-->");**

**root.printNodes();**

**System.*out*.println("");**

**// System.out.print("Current keys -->") r**

**//root.printNodes();**

**// System.out.println("");**

**System.*out*.println("\*\*\*\* end");**

**System.*out*.println("");**

**}**

**public void printAscendingOrder(){**

**System.*out*.print(" Bonus) Ascending Order --> ");**

**// Wrapper for node print method**

**root.printAscendingOrder();**

**}**

**public void printNodes(){**

**// Wrapper for node print method**

**root.printNodes();**

**}**

**public void printDescendingOrder(){**

**System.*out*.print(" Descending Order --> ");**

**root.printDescendingOrder();**

**}**

**// Split method creates 2 children, left child and right child.**

**// It then assigns the number of keys each child which is T-1;**

**// and then I assign the median, which is the key that will be promoted**

**// I then traverse up to the mid element and assign all of the root children**

**// to be now the left child. Then I grab all of the keys the root has to be**

**// now the left child's key, except the mid element. Then I traverse from**

**// the mid element up until I reach the last element of the root and do the exact**

**// same thing as I did for the left child**

**private void split() {**

**System.*out*.println("SPLIT METHOD ");**

**System.*out*.println("");**

**BTreeNode leftChild = new BTreeNode(T);**

**BTreeNode rightChild = new BTreeNode(T);**

**leftChild.isLeaf = root.isLeaf;**

**rightChild.isLeaf = root.isLeaf;**

**leftChild.n = T-1;**

**rightChild.n = T-1;**

**int median = T-1;**

**// Add elements up to the middle element and store them in the left child**

**for(int i = 0 ; i < T-1 ; i ++){**

**leftChild.c[i] = root.c[i];**

**leftChild.key[i] = root.key[i];**

**}**

**for(int j = median+1; j < root.n; j++){**

**rightChild.c[j-median-1] = root.c[j];**

**rightChild.key[j-median-1] = root.key[j];**

**}**

**rightChild.c[median]=root.c[root.n];**

**System.*out*.print("Print left child--> ");**

**leftChild.printNodes();**

**System.*out*.println("");**

**System.*out*.print("Print right child-->");**

**rightChild.printNodes();**

**System.*out*.println("");**

**root.key[0]= root.key[median];**

**root.n = 1;**

**root.c[0] = leftChild;**

**root.c[1] = rightChild;**

**System.*out*.print("Print father of the left and right child--> ");**

**root.printNodes();**

**System.*out*.println("");**

**root.isLeaf = false;**

**}**

**// This is where most of the data gets printed //**

**public void treeInfo() {**

**System.*out*.println("[Tree info] ");**

**/\*\* Change the following to find the number of nodes at a specific depth-level**

**\* To also count the number of nodes that contain exact n-keys**

**\* and to also search for a specific key.**

**\*/**

**int depth = 2;**

**int keys = 3;**

**int searchKey = 1;**

**int deleteKey = 8;**

**System.*out*.println(" T = " + T + ".");**

**System.*out*.println(" Every internal node other than the root has at least: " + ( T - 1 )+ " key(s).");**

**System.*out*.println(" Every internal node other than the root has at least " + T + " children.");**

**System.*out*.println( " " + (2\*T-1) + " is the maximum number of keys the tree can have.");**

**System.*out*.println( " " + (T-1) + " is the mininum number of keys the tree can have.");**

**System.*out*.println(" An internal node may have at most " + (2\*T) + " children");**

**System.*out*.println(" The height of the tree is:---> " + height);**

**System.*out*.print(" The root is---------------------------------------->[ " );**

**printTheRoot();**

**System.*out*.print("]");**

**System.*out*.println("");**

**System.*out*.println("");**

**System.*out*.println(" TREE: " );**

**printTheChildren(root);**

**System.*out*.println("");**

**findMin(root);**

**counter =0;**

**findMax(root);**

**counter=0;**

**System.*out*.println(" 9) Total Number of Nodes: " + findNumberofNodes(root));**

**counter=0;**

**System.*out*.println(" 10) Total Number of Keys is: " + findNumberofKeys(root));**

**counter=0;**

**System.*out*.println(" 11) Sum of all keys is: " + findAllSumKeys(root));**

**counter=0;**

**System.*out*.println(" 12) The number of of leaves is: " + NumberofLeaves(root));**

**counter=0;**

**System.*out*.println(" 13) The number of full nodes is: " + numOfNodesFull(root));**

**counter=0;**

**System.*out*.println(" 14) The number of nodes in the tree that have the minimum number of keys is: " + NumberOfMinKeys(root));**

**counter=0;**

**System.*out*.println(" 15) The number of nodes in the tree that have the exact n keys is: " + keys + " is: " + ExactlyNKeys(root, keys));**

**counter=0;**

**numberOfNodesDepthD(root, depth);**

**System.*out*.print(" 17) Searching for key " + searchKey + "... p.s if nothing prints it means it wasn't found. ");**

**root.search(searchKey);**

**/\*\*\*\*\*\* The DELETE METHOD needs to be fixed!! \*\*\*\*\*\*/**

**//System.out.println(" 17) Delete " + deleteKey + root.deletions(deleteKey));**

**//System.out.println(" 17) return true if found else false--> " + deleteKey + root.deletions(deleteKey));**

**}**

**private void findMin(BTreeNode root) {**

**// TODO Auto-generated method stub**

**System.*out*.println(" 7) The mininum element of the BTree is: " + root.findMin());**

**}**

**private void findMax(BTreeNode root) {**

**// TODO Auto-generated method stub**

**System.*out*.println(" 8) The maximum element of the BTree is: " + root.findMax());**

**}**

**private int findNumberofNodes(BTreeNode node) {**

**if(node.isLeaf){**

**counter++;**

**}**

**else{**

**counter++;**

**for(int i = 0 ; i < node.n+1; i++){**

**//findNumberofNodes(root.c[i]);**

**//= counter4 + root.findNumberofNodes();**

**if(!(node.c[i].isLeaf)){**

**//System.out.println("counter4b: " + counter4);**

**findNumberofNodes(node.c[i]);**

**}**

**else{**

**counter++;**

**//System.out.println("counter4c: " + counter4);**

**}**

**}**

**}**

**//System.out.println("counter4d: " + counter4);**

**return counter;**

**}**

**// The following method finds the number of keys in the tree.**

**private int findNumberofKeys(BTreeNode node) {**

**// count the number of keys in the leafs**

**if(node.isLeaf){**

**for(int i = 0; i < node.n; i++){**

**counter++;**

**}**

**}**

**else{**

**// count the number of keys in the non-leaf nodes**

**for(int i = 0 ; i < node.n; i++){**

**counter++;**

**findNumberofKeys(node.c[i]);**

**}**

**findNumberofKeys(node.c[node.n]);**

**}**

**return counter;**

**}**

**// The following method finds the sum of all the keys**

**private int findAllSumKeys(BTreeNode node) {**

**if(node == null){**

**return counter;**

**}**

**else{**

**if(node.isLeaf){**

**// System.out.println("root is a leaf");**

**counter = counter + node.findAllSumKeys();**

**//System.out.println("counter1: " + counter);**

**}**

**else{**

**for(int i = 0; i < node.n; i++){**

**//System.out.println(" key3: " + root.key[i]);**

**counter = counter + node.key[i];**

**}**

**for(int i = 0 ; i < node.n+1; i++){**

**//System.out.println(" counter3: " + counter);**

**findAllSumKeys(node.c[i]);**

**}**

**}**

**}**

**return counter;**

**}**

**// Finds the number of nodes at that specific level**

**public void numberOfNodesDepthD(BTreeNode node7, int depth) {**

**System.*out*.println(" 16) The number of nodes in the tree that are in depth " + depth + " level is: " + node7.numberOfNodesDepthD(depth) + " total nodes." );**

**}**

**// Exactly that contains n- keys**

**public int ExactlyNKeys(BTreeNode node, int keys) {**

**// TODO Auto-generated method stub**

**if(node == null){**

**return counter;**

**}**

**else{**

**if(node.n == keys){**

**counter++;**

**}**

**for(int i = 0 ; i < node.n+1; i++){**

**ExactlyNKeys(node.c[i], keys);**

**}**

**}**

**return counter;**

**}**

**// Number of Min keys**

**private int NumberOfMinKeys(BTreeNode node5) {**

**if(node5 == null){**

**return counter;**

**}**

**else{**

**if(node5.n == T-1){**

**counter++;**

**}**

**for(int i = 0 ; i < node5.n+1; i++){**

**NumberOfMinKeys(node5.c[i]);**

**}**

**}**

**return counter;**

**}**

**private int numOfNodesFull(BTreeNode node4) {**

**if(node4 == null){**

**return counter;**

**}**

**if(node4.isFull()){**

**//System.out.println("Node is FULL ");**

**counter++;**

**}**

**else{**

**for(int i = 0 ; i < node4.n+1; i++){**

**numOfNodesFull(node4.c[i]);**

**}**

**}**

**return counter;**

**}**

**// Counts the number of leaves**

**private int NumberofLeaves(BTreeNode node) {**

**if(node.isLeaf){**

**counter++;**

**}**

**else{**

**for(int i = 0 ; i < node.n+1; i++){**

**NumberofLeaves(node.c[i]);**

**}**

**}**

**return counter;**

**}**

**private void printTheChildren(BTreeNode node) {**

**if(node.isLeaf){**

**System.*out*.print(" [");**

**for(int i = 0 ; i < node.n; i++){**

**if(i == 0){**

**}**

**System.*out*.print(node.key[i] + " ");**

**}**

**System.*out*.print("]");**

**}**

**else{**

**System.*out*.print(" [" + node.key[0] + "]");**

**System.*out*.println("");**

**for(int i = 0 ; i <= node.n; i++){**

**printTheChildren(node.c[i]);**

**}**

**System.*out*.println(" ");**

**}**

**}**

**private void printTheRoot() {**

**for(int i = 0; i < root.n; i++){**

**System.*out*.print(root.key[i] + " ");**

**}**

**}**

**} package lab3beta2;**

**public class BTreeNode {**

**public int[] key; // stores the key in the current node**

**public BTreeNode[] c;**

**boolean isLeaf;**

**public int n; /\*\* The number of keys stored in the node. \*/**

**private int T; //Each node has at least T-1 and at most 2T-1 keys**

**int counter = 0 ;**

**int counter3 = 0;**

**// Create BTreeNode**

**public BTreeNode(int t){**

**T = t;**

**isLeaf = true;**

**key = new int[2\*T-1]; // Max number of keys**

**c = new BTreeNode[2\*T];**

**n=0;**

**}**

**// Checks if current node is full**

**public boolean isFull(){**

**return n==(2\*T-1);**

**}**

**// First checks if the current key is a leaf, if so meaning the tree is empty it keeps inserting recursievly until**

**// the max keys is reached. I used an integer i to be able to distinguish if the key is < or > than the current root**

**// its just to place the newKey in the proper position.**

**// Now if the root is full it becomes a non-leaf root and then it splits and promotes the mid element to be the new root**

**// and it then recurisvely cls the method to check if the children are full.**

**public void BTreeInsertNonFull( int newKey) {**

**int i= n-1; // n = the number of current keys stored in the node**

**if(isLeaf){**

**System.*out*.println("Current node is a leaf! ");**

**// In other words the while loop is checking for elements that are greater than the newKey**

**// once the newKey found the element greater than the current element**

**// it assigns the next position of the current element to be the current element**

**// Once out of the while loop we replace the current element next position to the new key. .**

**while ((i>=0)&& (newKey<key[i])) {**

**System.*out*.println(key[i] + " > " + newKey + " ?");**

**key[i+1] = key[i];**

**i--;**

**}**

**n++;**

**key[i+1]=newKey;**

**}**

**else{**

**System.*out*.println(" Current node is NOT a leaf. ");**

**System.*out*.println(" key[" + i + "] = " + key[i]);**

**// So essentially what this while loop does is compare the elements in the current node**

**// and if the current key is greater than the new key we get in the while loop**

**// and then we add the elements in the most left child.**

**while ((i>=0)&& (newKey<key[i])) {**

**i--;**

**}**

**int insertChild = i + 1; // where i is 1 less than the number of keys stored in the node**

**// If the children is full we must split.**

**if(c[insertChild].isFull()){**

**System.*out*.println("child[" + insertChild + "] is FULL THUS WE MUST SPLIT the child and create a new father");**

**n++;**

**// System.out.println("----> n value " + n);**

**// System.out.println("----> n value " + c[n]);**

**// This next line creates a new child and copies all variables from the full child.**

**// new child[3] = child[2]**

**c[n] = c[n-1];**

**// The for loop moves all the children to the left copies child[2] = child[1] && key[2] = key[1]**

**// then child[1] = child[0] && key[2] = key[1]**

**for(int j = n-1;j>insertChild;j--){**

**c[j] =c[j-1];**

**key[j] = key[j-1];**

**}**

**// System.out.println("What is the value of T? " + T);**

**// key[0] = c[0].key[T-1] where T = 3**

**// key[0] = c[0].key[2]**

**// We are pretty much creating the root in the next line**

**key[insertChild]= c[insertChild].key[T-1];**

**// c[0].n = 2 pretty much what this means is we're cutting the full child in half and copying the new father into the existing fathers**

**// System.out.println("c[insertChild].n " + c[insertChild].n);**

**c[insertChild].n = T-1;**

**BTreeNode newNode = new BTreeNode(T);**

**// If necessary we must also copy the Father's children children (or grandchildren)**

**//Traverse up to the middle element and start creating children**

**// newNode.c[0] = c[0].c[0+3] == c[0].c[3]**

**// We must also create a new key and set it equal to the c[0] keys**

**// This wil be our new right child**

**// System.out.println("c[insertChild].c[k+T]= " + c[insertChild].c[T]);**

**//System.out.println("c[insertChild].key[k+T= " + c[insertChild].key[3]);**

**for(int k=0;k<T-1;k++){**

**newNode.c[k] = c[insertChild].c[k+T];**

**newNode.key[k] = c[insertChild].key[k+T];**

**}**

**// the next lines essentially assigns**

**// newNode.c[2] = c[0].c[5] == null**

**// System.out.println( "c[insertChild].c[2\*T-1] = " + c[insertChild].c[2\*T-1]);**

**newNode.c[T-1] = c[insertChild].c[2\*T-1];**

**newNode.n=T-1;**

**newNode.isLeaf = c[insertChild].isLeaf;**

**c[insertChild+1]=newNode;**

**// if the new key is less than key[0]**

**if (newKey <key[insertChild]){**

**c[insertChild].BTreeInsertNonFull(newKey);**

**}**

**if(newKey > key[insertChild]){**

**c[insertChild+1].BTreeInsertNonFull(newKey);**

**}**

**}**

**else**

**c[insertChild].BTreeInsertNonFull(newKey);**

**}**

**}**

**//Prints all keys in the tree in ascending order**

**public void printAscendingOrder(){**

**if (isLeaf){**

**for(int i =0; i<n;i++){**

**System.*out*.print(key[i]+" ");**

**}**

**}**

**else{**

**for(int i =0; i<n;i++){**

**c[i].printAscendingOrder();**

**System.*out*.print(key[i]+" ");**

**}**

**c[n].printAscendingOrder();**

**}**

**}**

**public void printNodes(){**

**//Prints all keys in the tree, node by node, using preorder**

**//It also prints the indicator of whether a node is a leaf**

**//Used mostly for debugging purposes**

**for(int i =0; i<n;i++){**

**if(i == 0){**

**System.*out*.print("[ ");**

**}**

**System.*out*.print(key[i]+" ");**

**if (!isLeaf){**

**for(int j =0; j<=n;j++){**

**c[j].printNodes();**

**}**

**}**

**if(i == n-1){**

**System.*out*.print("]");**

**}**

**}**

**}**

**// Prints from Largest - to smallest**

**public void printDescendingOrder() {**

**if(isLeaf){**

**for(int i = n-1; i >= 0 ; i--){**

**System.*out*.print(key[i] + " ");**

**}**

**}**

**else{**

**// root.n - 1**

**for(int i = n-1; i >= 0; i--){**

**c[i+1].printDescendingOrder();**

**System.*out*.print( key[i] + " ");**

**}**

**c[0].printDescendingOrder();**

**}**

**}**

**// Finds the smallest element of the tree**

**public Integer findMin() {**

**if(isLeaf){**

**return key[0];**

**}**

**return c[0].findMin();**

**}**

**// Finds the largest element of the tree**

**public Integer findMax() {**

**if(isLeaf){**

**return key[n-1];**

**}**

**return c[n].findMax();**

**}**

**// Finds the sum of all the keys**

**public int findAllSumKeys() {**

**//System.out.println("Node class");**

**for(int i = 0 ; i < n ; i ++){**

**// System.out.println("key[" + i + "]= " + key[i]);**

**counter = counter + key[i];**

**}**

**//System.out.println("The total sum of keys in the children: " + counter);**

**return counter;**

**}**

**// Counts the number of full nodes**

**public int numOfNodesFull() {**

**System.*out*.println("numOfNodesFull method ");**

**for(int i = 0 ; i < n; i++) {**

**if(isLeaf && c[i].isFull()){**

**counter3++;**

**}**

**}**

**return counter3;**

**}**

**// Print children**

**public void printChildren() {**

**if(isLeaf){**

**for(int i = 0 ; i < n; i++){**

**System.*out*.print(key[i] + " ");**

**}**

**}**

**}**

**// Prints the number of nodes in that specific depth**

**public int numberOfNodesDepthD(int depth) {**

**int count = 0;**

**if(depth==0){**

**count = 1;**

**}**

**else{**

**if(!isLeaf){**

**for(int i = 0 ; i <= n; i++){**

**count = count + c[i].numberOfNodesDepthD(depth-1);**

**}**

**}**

**}**

**return count;**

**}**

**public void search(int searchKey) {**

**if(isLeaf){**

**for(int i =0; i<n;i++){**

**if(key[i] == (searchKey)){**

**System.*out*.println(" Found! key " + searchKey + " and it happens to be a leaf. ");**

**return;**

**}**

**}**

**}**

**else{**

**for(int i = 0 ; i < n ; i ++){**

**if(key[i]==(searchKey)){**

**System.*out*.println(" Found! key " + searchKey + " and it happens to be a non-leaf node. ");**

**return;**

**}**

**else{**

**c[i].search(searchKey);**

**}**

**c[n].search(searchKey);**

**}**

**}**

**}**

**public String deletions(int deleteKey) {**

**if(isLeaf){**

**for(int i =0; i<n;i++){**

**if(key[i] == (deleteKey)){**

**System.*out*.println("We found key " + deleteKey + " its a leaf! " + key[i]+" ");**

**}**

**}**

**}**

**else{**

**for(int i = 0 ; i < n ; i ++){**

**if(key[i]==(deleteKey)){**

**System.*out*.println("We found the key we want to delete =" + key[i]);**

**}**

**c[i].deletions(deleteKey);**

**System.*out*.println(key[i] +"<- ");**

**}**

**c[n].deletions(deleteKey);**

**}**

**return "blah";**

**}**

**}**

**}**

Conclusion:

Attempting to build a B-Tree has given me a more intuitive understanding on how hard drives work. I always wondered why in my old laptop I would hear the disk drive spin (I am using the new flash drive now) when I would look through elements in my OS. Although due to time constraints I was not able to complete the deletion method I still did research and read where to delete keys. ­I also built an intuitive understanding why B-Trees are O(logn)

Academic Honesty Certification

“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_