CS 3310 Data and File Structures

Instructor Dr. Ajay K. Gupta

Western Michigan University

Grad TA: Rajani Pingili

Anfaal Faisal

SOFTWARE LIFE CYCLE REPORT – FOR HW ASSIGNMENT 6

PHASE 1: SPECIFICATION

1. The main goal of this assignment was to
    - to understand trees data structures
    - To practice programming using binary search trees
    - Practice developing high-performance solutions.
    - Analyze what are the advantages and the disadvantages of various techniques to manipulate trees.
2. Implement a binary tree and some operations with the tree
3. Practice good coding conventions

The application had to do the following:

- Creating binary tree
- Perform requested operation as:
    a. insert
    b. delete
    c. search
    d. post-order printing
    e. preorder printing
    f. in-order printing

PHASE 2: DESIGN

1. Java Project
    a. App
        i. **void** main(S**tring**[] args) Perform all main actions
    b. MyData
        i. Mydata(**String** stuName, **int** courseNumber, **char** grade) constructor

ii. **String** getStuName() Public getter
iii. setStuName(**String** stuName) Public setter
iv. **int** getCourseNumber() Public getter
v. void setCourseNumber(**int** courseNumber) Public setter
vi. **char** getGrade() Public getter
vii. void setGrade(**char** grade) Public setter
viii. **String** toString() Create and return String representation

c. BinaryStreeImplicit
   i. BinaryStreeImplicit() Simple constructor
   ii. **int** root() Return root number
   iii. **int** height(**int** i) Return heigth of i-th node
   iv. **int** leftchild(**int** i) Return index of left child node
   v. **int** rightchild(**int** i) Return index of right child node
   vi. **int** parent(**int** i) Return index of parent node
   vii. void inorderTraversal() Public inorder print function
   viii. void printInorder(**int** i) Prints one node
   ix. void printNode(**int** i) Private inorder print function
   x. void preorderTraversal() Public preorder print function
   xi. void printPreorder(**int** i) Private preorder print function
   xii. void postorderTraversal() Public postorder print function
   xiii. void printPostorder(**int** i) Private postorder print function
   xiv. **int** insert(**Mydata** x) Insert data to tree
   xv. **int** insert(**int** i, **Mydata** x) private inserting with pre-selected root
   xvi. void extend() Extend array
   xvii. **int** delete(**Mydata** x) Delete selected information
   xviii. **int** search(**Mydata** x) Search data in binary tree
   xix. **int** search(**int** i, **Mydata** x) Private search from selected root
   xx. **Mydata** getElement(**int** i) Return data at selected index

In fact, BinaryStreeImplicit implement simple binary tree, with rearranging and initial zero state. This tree is not generic but used for saving Mydata information only. Nevertheless, it can be easily modified.

Generally,a binary search tree is a binary tree with additional properties: the value of the left child is less than the value of the parent, and the value of the right child is greater than the value of the parent for each node of the tree. That is, the data in the binary search tree is stored in sorted form. Each time you insert a new or delete an existing node, the sorted tree order is saved. When searching for an element, the sought value is compared with the root. If the search is larger than the root, then the search continues in the right descendant of the root, if less, then in the left, if equal, then the value is found and the search stops.

Space complexity analysis

Analysis of space complexity for the code is summarized in the table below:

Method	Input space	Private fields space
Main	1	About 9 total object 4 of which are compound data structures (BufferedWriter, File etc)
Mydata	3	3
toString	1	0
root	0	0
height	1	3
leftchild	1	3
rightchild	1	3
parent	1	5
inorderTraversal	0	0
printInorder	1	0
printNode	1	0
preorderTraversal	0	0
printPreorder	1	0
postorderTraversal	0	0
printPostorder	1	0
insert	1	0
insert	2	1
extend	0	1
delete	1	3
search	1	1
search	2	1
getElement	1	0

Time complexity

Asymptotic analysis of time complexity for the code is summarized in the table below:

Method	**Time Complexity**
Main	O(n)
Mydata	O(1)
toString	O(1)
root	O(1)
height	O(log n)
leftchild	O(log n)
rightchild	O(log n)
parent	O(log n)
inorderTraversal	O(n)
printInorder	O(log n)
printNode	O(1)
preorderTraversal	O(n)
printPreorder	O(log n)
postorderTraversal	O(n)
printPostorder	O(log n)
insert	O(log n)
insert	O(log n)
extend	O(n)
delete	O(log n)
search	O(log n)
search	O(log n)
getElement	O(1)

Answer on questions

1. Derive the relationships among height h, total number of nodes n, number of leaves L, and the number of edges m in a fully-complete quad tree T (a fully-complete tree is a complete tree in which all possible nodes at every level in the tree are present including the last level).
In particular, given h, what is n? Given n, what is h? Given m, what is h? Given m, what is n? Given L, what is h? Given h, what is L? Derive exact expressions first and then express them using big-theta/big-oh notation. Show all your work otherwise no credit.
Answer: Quad tree - a tree in which each internal node has exactly 4 children
So, in h=1 n=5 =4+1, l=4,m=4. In h=2 n=21=16+4+1, l=16,m=20=16+4. In h=3 n=85=64+16+4+1, l=64,m=84=64+16+4.

Where h=x, $n=\sum_{i=0}^{x}4^i$,l=4^x, $m=\sum_{i=1}^{x}4^i=n-1$

2. Repeat question 1 if T is a complete quad tree.
Answer: complete tree -is tree, in which all layers are filled, except last layer. So, in h=2 n=6…20=(1..15)+4+1, l=1…15,m=5…19=1…15+4. In h=3 n=22…84=1…63+16+4+1, l=1…63,m=21…83=1…63+16+4.

Where h=x, l=1…(4^x-1) $n=\left(\sum_{i=0}^{x-1}4^i\right)+l$, $m=\left(\sum_{i=1}^{x-1}4^i\right)+l=n-1$

3. Describe an implicit representation of a quad-tree T using arrays. Give the declaration of the array that stores the data of the nodes of T. Assume that the data consists of three fields: ssid, name (in the format firstname:lastname), phoneNumber. Given an arbitrary node v of T at index i, what are the indices of the four children of v and the parent of v?
In saving tree in arrays/ each data field I'll be get number as (i*3)+ni, where I – number of nodes, ni – number of fields.
It depends not only f index I, but also from height h

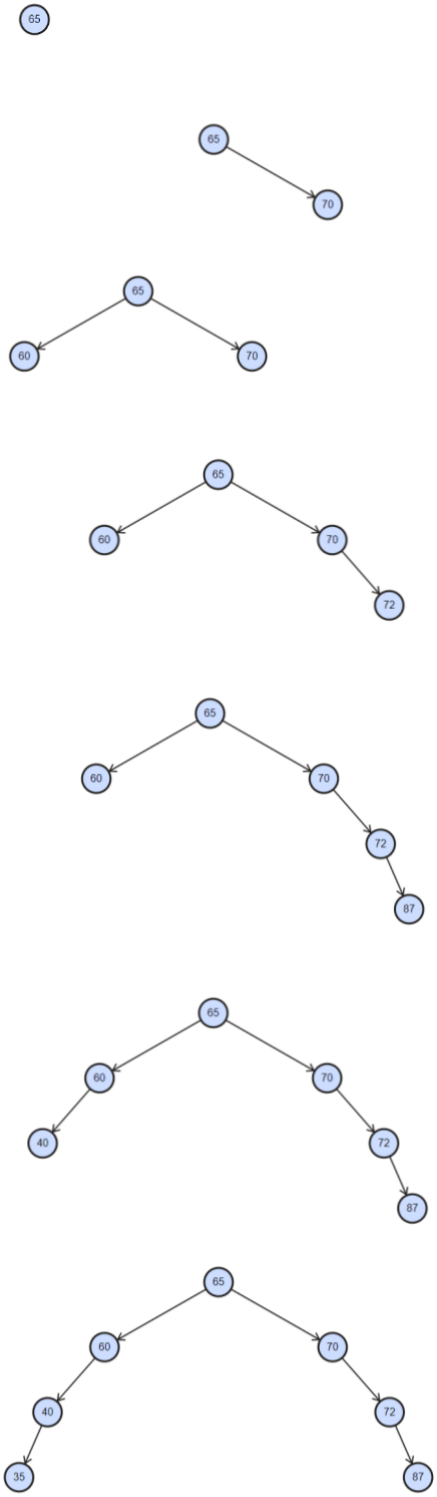
If node in I index, t height h, then index in row $j=i-\left(\sum_{i=0}^{h-1}4^i\right)$, and number of childrens will be

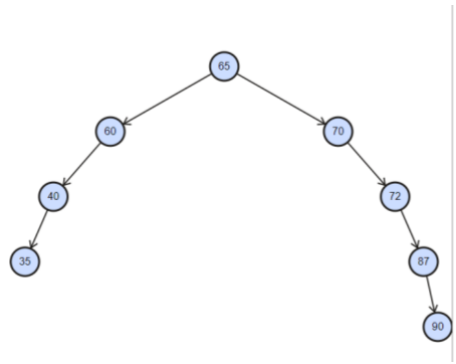
z,z+1,z+2 and z+3 where $z=\left(\sum_{i=0}^{h}4^i\right)+(4\times j)=4i+4^h-3\left(\sum_{i=0}^{h-1}4^i\right)$

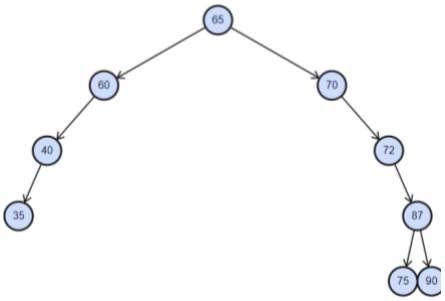
In the same time, number of parent of that node will be $k=\left(\sum_{i=0}^{h-2}4^i\right)+j=i-4^{h-1}$

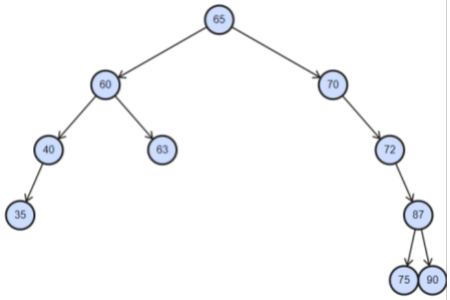
4. Let T be a (unbalanced) binary search tree:

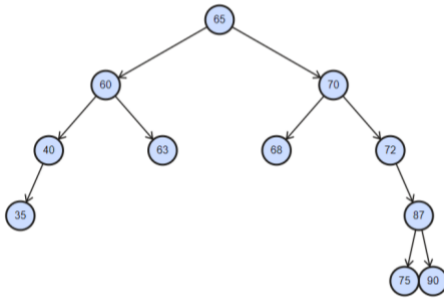
a. Insert the sequence 65, 70, 60, 72, 87, 40, 35, 90, 75, 63, 68, 69, 61, 20, 25, 28, 37 (of integer keys) into T. Show the tree after each insert operation.

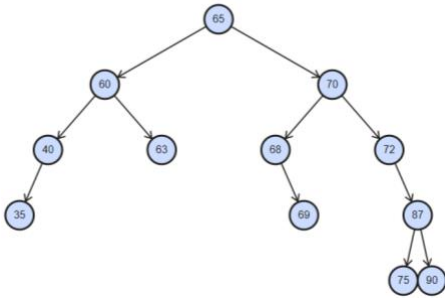


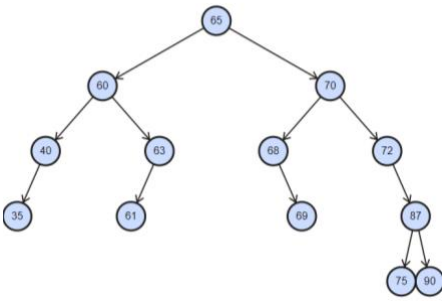

65
60
70
40
72
35
87
90

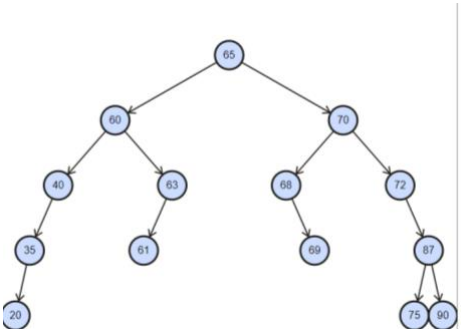

65
60
70
40
72
35
87
75
90

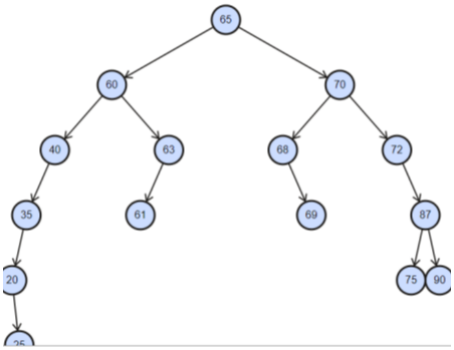

65
60
70
40
63
72
35
87
75
90

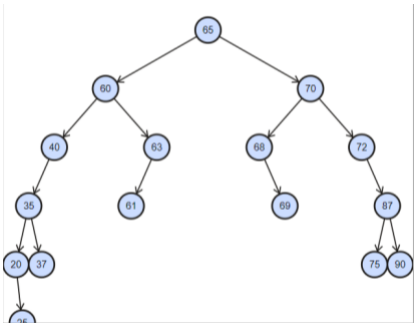

65
60
70
40
63
68
72
35
87
75
90


65
60
70
40
63
68
72
35
69
87
75
90









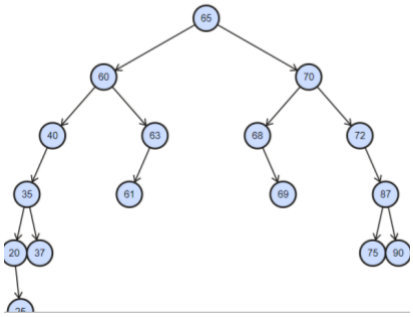
b. Show the output of Preorder, Inorder and Postorder traversals of T after last insertion (i.e., after key 37 is inserted).

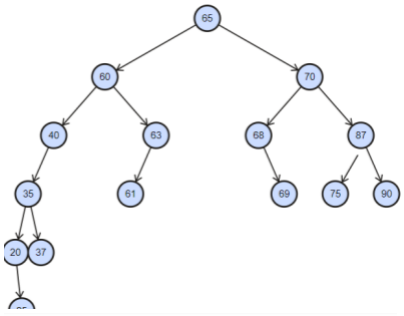
Preorder 65,60,40,35,20,25,28,37,63,61,70,68,69,72,87,75,90

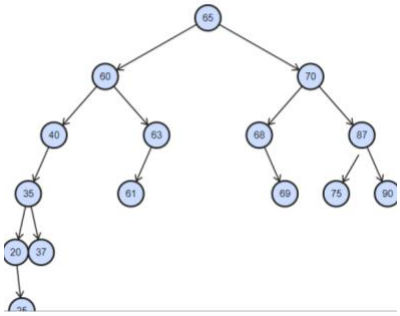
Inorder 20,25,28,35,37,40,60,61,63,65,68,69,70,72,75,87,90

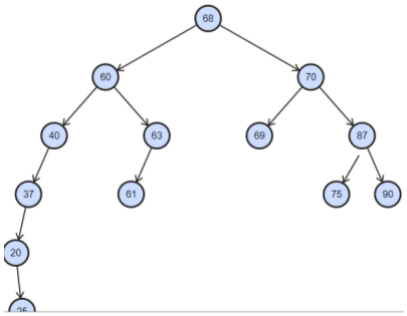
Postorder 28,25,20,37,35,40,61,63,60,69,68,75,90,87,72,70,65

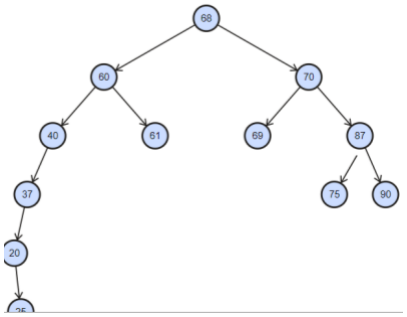
c. Delete the sequence 28, 72, 65, 35, 63, 87, 70 from T. Show T after each deletion. Note that to be consistent in deletion, follow the same algorithm when you have to "borrow" a key to fill a hole.

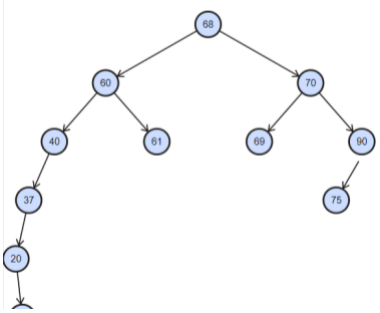

65
60
70
40
63
68
72
35
61
69
87
20
37
75
90

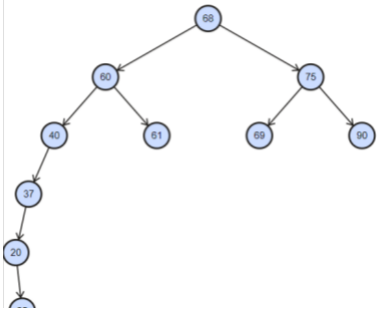

65
60
70
40
63
68
87
35
61
69
75
90
20
37


65
60
70
40
63
68
87
35
61
69
75
90
20
37


68
60
70
40
63
69
87
37
61
75
90
20


68
60
70
40
61
69
87
37
75
90
20





PHASE 3: RISK ANALYSIS

There are no risks associated with this application. There is a possible risk of a file not found exception.

PHASE 4: VERIFICATION

The algorithms were tested multiple times, and the list/array was printed out multiple times to verify that each step of the process was computed correctly. The printing was then removed once the process was verified.

PHASE 5: CODING

The code of this program is include in the zip file. The code is explained by in line comments or in Javadoc.

PHASE 6: TESTING

This program was tested in manual mode.

PHASE 7: REFINING THE PROGRAM

No refinements are needed for this program.

PHASE 8: PRODUCTION

A zip file including the source files from eclipse and the output of the program have been included.

PHASE 9: MAINTENANCE

This application can be further improved once feedback from the grader is obtained.