Binary trees → non-linear ds

Stacks
Queues
LL } linear ds

Tree → hierarchical data → folder structure
→ Directories

Binary tree: Every node can have max 2 children
(0, 1, 2 children)

NODE {
int data;
node left;
node right;
}

Root = 10 — root
Root.l = 5 — l
Root.r = 15 — r
Root.l.l = 3 — l.l
Root.l.r = 7 — l.r
Root.r.r = 20 — r.r

height → 3 — 3 levels in tree
node → 7, max → 8

$2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1$

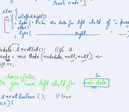
if

$\frac{2^h - 1}{2 - 1} = n \Rightarrow 2^h - 1 = n$

$\boxed{2^h = n+1}$

Class BTS
{
Class Node {
int data;
Node left;
Node right;
Node (int data, Node left, Node right)
this.data = data;
this.left = left;
this.right = right;
}
}
Node root; int size = 0
BTS {
Scanner sc = new Scanner (System.in);
this.root = this.takeInput(sc, null, false);
}

private Node takeInput(Scanner s, Node parent, boolean isLeft) {
if (parent == null) {
System.out.println("Enter the data for root node");
} else {
if (isLeft==true) {
System.out.println("Enter the data for left child of " + parent.data);
} else {
System.out.println("Enter the data for right child of " + parent.data);
}
}
int nodeData = s.nextInt();
Node node = new Node (nodeData, null, null);
int size = this.size;
size = size + 1;
boolean choice = false;
System.out.println("Do you have left child of " + node.data);
choice = s.nextBoolean();
if (choice) {
node.left = this.takeInput(s, node, true);
}
System.out.println("Do you have right child of " + node.data);
choice = s.nextBoolean();
if (choice) {
node.right = this.takeInput(s, node, false);
}
return node;
}



25 => 50 & 75
12 => 25 & 68
END => 12 & END.
14 => 68 &
END => 14 & END.

25 => 50 ⇐ 75

12 => 25 ⇐ 68

public void display(Node root) {
this.display(this.root);
}
private void display(Node node) {
if (node.left != null) {
System.out.print(node.left.data + "=>");
} else {
System.out.print("END=>");
}
System.out.print(node.data);
if (node.right != null) {
System.out.print(node.right.data);
} else {
System.out.print("(END)");
}
System.out.println();
if (node.left != null) {
this.display(node.left);
}
if (node.right != null) {
this.display(node.right);
}
}

left child recursive

1) Max

(68, 50, 75) => 75 ans

(75, 37, 69) 3) min

public int Max() {
return this.Max(this.root);
}
private int Max(Node node) {
if (node == null) {
return Integer.MIN_VALUE;
}
int lmax = this.max(node.left);
int rmax = ——— (node.right);
int myans = Math.max(lmax, Math.max(node.data, rmax));
return myans;
}

⇒ Max

⇒ Max

⇒ (-∞, 12, -∞)

2) int size

public int size() {
return this.size(this.root);
}
private int size(Node node) {
if (node == null) {
return 0;
}
int lsize = this.size(this.node.left);
int rsize = ——— (node.right);
int myans = lsize + rsize + 1;
return myans;
}

3) height

public int height() {
}
private int height(Node node) {
if (node == null) {
return -1;
}
int lheight = this.height(node.left);
int rheight = ——— (node.right);
int myans = Math.max(lheight, rheight) + 1;
return myans;
}

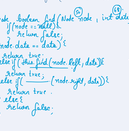
1) Max
2) Min
3) size
4) height

{ return at
8.32

find-(data)
find-(14)
→ true
→ false

public boolean find(int data) {
return this.find(this.root, data);
}
private boolean find(Node node, int data) {
if (node == null) {
return false;
}
if (node.data == data) {
return true;
} else if (this.find(node.left, data)) {
return true;
} else if (——— (node.right, data)) {
return true;
} else {
return false;
}
}

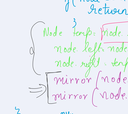
8. mirror



public void mirror() {
}
private void mirror(Node node) {
if (node == null) {
return;
}
Node temp = node.left;
node.left = node.right;
node.right = temp;
mirror(node.left);
mirror(node.right);
}

11,285 mK
temp
2mK

9. Traversals



Pre → 50, 25, 14, 37, 75, 69, 19.
In → 14, 25, 37, 50, 69, 75, 19.
Post → 14, 19, 37, 25, 69, 75, 50.

Trees
→ BST
→ Operations on BST
→ B-Tree

→ AVL
→ Operations
→ B+T

Tomorrow there is no Class.

Stacks & Queues

LL & Trees