

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

UE17CS251

Design and Analysis of Algorithms

Project Report

AVL TREES VISUALISATION

AND USING IT TO STORE STUDENT

INFORMATION

Team Details:

Abhishek Patil : PES1201701507

Manthan B Y : PES1201701498

Section : F

Semester: IV

**Abstract:**

An Adelson-Velskii Landis (AVL) tree is a self-balancing BST that maintains it's height to be O(log N) when having N vertices in the AVL tree. It consists of rotations to balance the tree. The best way to learn these using visualizing the rotations while we insert elements. The webpage helps us to visualize the AVL tree and also it shows how can we store the information using the AVL trees.

**Definition:**

**AVL tree** is a self-balancing Binary Search **Tree**(BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

**Introduction:**

An AVL tree is a subtype of binary search tree.

A BST is a data structure composed of nodes. It has the following guarantees:

1. Each tree has a root node (at the top).
2. The root node has zero or more child nodes.
3. Each child node has zero or more child nodes, and so on.
4. Each node has up to two children.
5. For each node, its left descendents are less than the current node, which is less than the right descendents.

AVL trees have an additional guarantee:

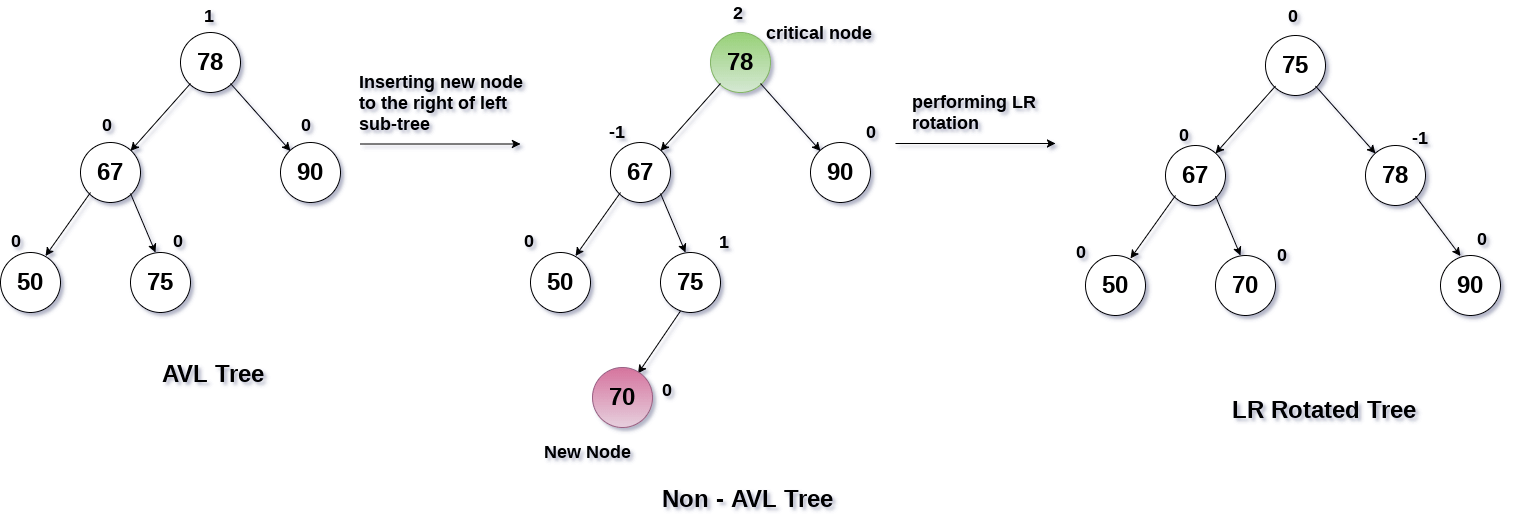
1. The difference between the depth of right and left subtrees cannot be more than one. In order to maintain this guarantee, an implementation of an AVL will include an algorithm to rebalance the tree when adding an additional element would upset this guarantee.

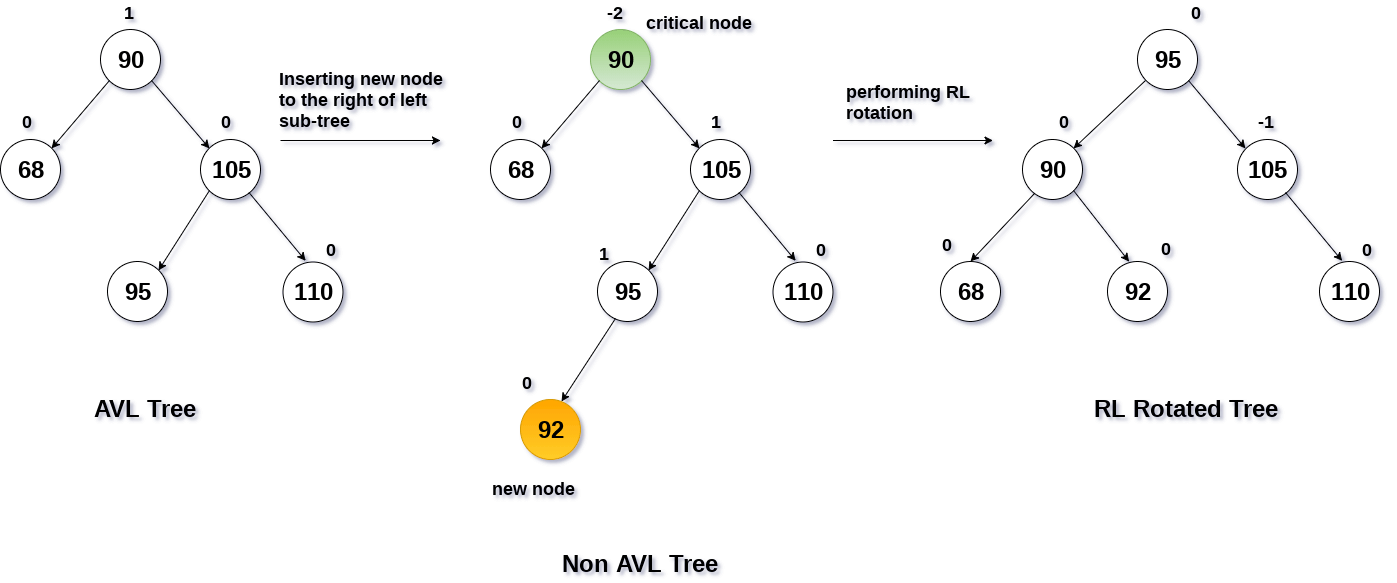
AVL trees have a worst case lookup, insert and delete time of O(log n).

Rotations:









**Implementation:**

The visualization is made possible using a webpage with textbox and buttons to work with the tree. The basic structure of the webpage is done using the HTML. The webpage is styled using Cascading Style Sheets. The main part, the heart of the project is the JavaScript code that runs in the background for any operation done for the tree. For every operation which can be done to the tree there resides a function defined using the JavaScript language which takes care of all the rotations and maintaining the balance of the tree.

The algorithm for insertion,deletion,search functions are similar to binary search tree functions.Only after the insertion and deletion operations are performed ,we use the balance function to balance the trees.

We have used a json object to store the hierarchy. So the json object of the root node is passed as an argument to an inbuilt library function d3.hierarchy() and d3.TreeMap(). The library used is d3.min.js which helps in visualization of the tree by maintaining the layout of the tree automatically.

**Code:**

**Introduction Page(HTML) :**

<!DOCTYPE html>

<html>

<head>

<title>DAA Project</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

<link href="https://fonts.googleapis.com/css?family=Montez" rel="stylesheet">

</head>

<body>

<nav class="navbar navbar-dark bg-dark">

<a class="navbar-brand" href="final.html">Click to Try Implementation</a>

</nav>

<br/>

<br/>

<header height="40px" >

<div class="container ">

<div class="jumbotron">

<div class="container">

<h1 >Design And Analysis of Algorithms Project</h1>

<hr class="my-4">

<h4>AVL trees implementation and using it to store Student information

</h4>

</div>

</div>

</div>

</header>

<div style="padding-left:25px;padding-right:25px">

<h3>What are AVL Trees?</h3>

<hr class="my-2">

<p>AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes. </p>

<br>

<h3>What is the use of AVL trees?</h3>

<hr class="my-2">

<p>Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations may become O(n) for a skewed Binary tree. If we make sure that height of the tree remains O(Logn) after every insertion and deletion, then we can guarantee an upper bound of O(Logn) for all these operations. The height of an AVL tree is always O(Logn) where n is the number of nodes in the tree.

</p>

<br>

<h3>AVL Trees Insertion</h3>

<hr class="my-2">

<p>To make sure that the given tree remains AVL after every insertion, we must augment the standard BST insert operation to perform some re-balancing. Following are two basic operations that can be performed to re-balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

1) Left Rotation

2) Right Rotation

Steps to follow for insertion

Let the newly inserted node be w

1) Perform standard BST insert for w.<br>

2) Starting from w, travel up and find the first unbalanced node.<br>

Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.<br>

3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z.<br>

There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:<br>

a) y is left child of z and x is left child of y (Left Left Case)<br>

b) y is left child of z and x is right child of y (Left Right Case)<br>

c) y is right child of z and x is right child of y (Right Right Case)<br>

d) y is right child of z and x is left child of y (Right Left Case)<br>

Following are the operations to be performed in above mentioned 4 cases.<br>

In all of the cases, we only need to re-balance the subtree rooted with z and the complete tree becomes balanced as the height of subtree (After appropriate rotations) rooted with z becomes same as it was before insertion<br>

<br>

<br>

<h2>Left Left Case</h2>

<hr class="my-2">

<img src="avl\_left\_rotation.jpg">

<br>

<br>

<br>

<h2>Right Right Case</h2>

<hr class="my-2">

<img src="avl\_right\_rotation.jpg">

<br>

<br>

<br>

<h2>Left Right Case</h2>

<hr class="my-2">

<img src="lr-rotation.png" width="600px" height="500px">

<br>

<br>

<br>

<h2>Right Left Case</h2>

<hr class="my-2" >

<img src="rl-rotation.png " width="600px" height="500px">

<br>

<br>

</p>

</div>

</body>

</html>

**Front Page :**

<!DOCTYPE html>

<head>

<meta charset="utf-8">

<style>

.node circle {

fill: #fff;

stroke: steelblue;

stroke-width: 3px;

}

.node text { font: 12px sans-serif; }

.node--internal text {

text-shadow: 0 1px 0 #fff, 0 -1px 0 #fff, 1px 0 0 #fff, -1px 0 0 #fff;

}

.link {

fill: none;

stroke: #ccc;

stroke-width: 2px;

}

.visible

{

display:block;

}

.invisible

{

display:none;

}

</style>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

<body>

<center>

<h1>Design and Analysis of Algorithms Project</h1>

<h3>Visualization of AVL Trees using Javascript</h3>

<div style="width:30%;height:2%" >

<input type="text" class="form-control " id="value" value="" autofocus placeholder="Enter a Value " size="45">

<input type="text" class="form-control " id="name" value="" autofocus placeholder="Enter Name " size="45">

<input type="text" class="form-control " id="SRN" value="" autofocus placeholder="Enter SRN " size="45">

<input type="text" class="form-control " id="Age" value="" autofocus placeholder="Enter Age " size="45">

</div>

<br>

<div >

<button type="button" class="btn btn-xs btn-dark" onclick="InsertNode();">Insert</button>

<button type="button" class ="btn btn-xs btn-danger" onclick="DeleteNode();">Delete</button>

<button type="button" class="btn btn-xs btn-success" onclick="SearchNode();">Search</button>

<button type="button" class="btn btn-xs btn-info" onclick="Print(bst.inorder);">Inorder</button>

<button type="button" class="btn btn-xs btn-primary" onclick="Print(bst.preorder);">Preorder</button>

<button type="button" class="btn btn-xs btn-warning" onclick="Print(bst.postorder);">Postorder</button>

<button type="button" class="btn btn-xs btn-dark" onclick="disp()">Display Fields</button>

<button type="button" class ="btn btn-xs btn-danger" onclick="hide();">Hide</button>

<button type="button" class="btn btn-xs btn-success" onclick="print\_bal\_factor();">Balance Factor</button>

</div>

<br>

<span id="msg"></span>

</center>

<!-- load the d3.js library -->

<script src="d3.min.js"></script>

<script src="avl\_code.js" charset="utf-8"></script>

</body>

**JavaScript code:**

function NODE(val=null,obj=null)

{

this.value=val; //val is the value of the node,i.e the key it stores

this.parent=null; //parent is uded to point to the parents node,it is null if the node is root

this.height=null; //height is the height of the tree

this.left=null; //left is used to point to the left child

this.right=null; //right is used to point to the right child

this.json={ //json is an object

name:this.value,

direction:null, //to indicate which node of the parent it is,i.e left or right

children:[]

};

this.obj=obj;

this.set\_height=function(){

var left\_height=0;

var right\_height=0;

this.json.children=[];

if(this.left)

{

left\_height=this.left.height;

this.json.children.push(this.left.json);//the left child becomes a part of the array

this.left.json.direction="left"; //to indicate that it is a left node of its parent node

}

if(this.right)

{

right\_height=this.right.height;

this.json.children.push(this.right.json);//the left child becomes a part of the array

this.right.json.direction="right"; //to indicate that it is a left node of its parent node

}

if(left\_height>right\_height)

this.height=1+left\_height;

else

this.height=1+right\_height;

}

this.bal\_fac=function(){

var left\_height=0;var right\_height=0;

if(this.left)

left\_height=this.left.height;

if(this.right)

right\_height=this.right.height;

return left\_height-right\_height;

}

} //end of function NODE

function BST\_AVL(){

this.root=null;

this.rotate\_right=function(z)

{

var left1=z.left;

var left\_r=null;

if(left1)

left\_r=left1.right;

if(left\_r)

left\_r.parent=z;

left1.right=z;

z.left=left\_r;

left1.parent=z.parent;

z.parent=left1;

z.set\_height();

left1.set\_height();

if(z==this.root)

this.root=left1;

return left1;

}

this.rotate\_left=function(z)

{

var right1=z.right;

var right\_l=null; //represents the left child of the right node of z

if(right1)

right\_l=right1.left;

if(right\_l)

right\_l.parent=z;

right1.left=z;

right1.parent=z.parent;

z.right=right\_l;

z.parent=right1;

z.set\_height();

right1.set\_height();

if(z==this.root)

this.root=right1;

return right1;

}

this.balance=function(cur)

{

if(cur.bal\_fac()==2)

{

if(cur.left.bal\_fac()==-1) //need to perform LR rotation

{

cur.left=this.rotate\_left(cur.left);

sleep(2000);

}

cur=this.rotate\_right(cur);

}

if(cur.bal\_fac()==-2)

{

if(cur.right.bal\_fac()==1)

cur.right=this.rotate\_right(cur.right);

cur=this.rotate\_left(cur);

}

return cur;

}

this.search=function(val,cur=this.root)

{

if(cur==null)

return -1;

if(cur.value==val)

return cur;

else if(cur.value<val)

return this.search(val,cur.right);

else

return this.search(val,cur.left);

}

this.insert=function(cur,val,obj){

if(cur==null)

cur=new NODE(val,obj);

else if(val<=cur.value)

{

cur.left=this.insert(cur.left,val,obj);

cur.left.parent=cur;

cur.left.json.direction="left";

cur.json.children.push(cur.left.json);

}

else

{

cur.right=this.insert(cur.right,val,obj);

cur.right.parent=cur;

cur.right.json.direction="right";

cur.json.children.push(cur.right.json);

}

cur.set\_height();

cur=this.balance(cur);

return cur;

}

this.Insert\_val=function(val,obj)

{

this.root=this.insert(this.root,val,obj);

}

this.delete\_val=function(val)

{

var node=this.search(val);

if(node==-1)

return;

this.delete(node);

if(this.root)

this.root=this.balance(this.root);

}

this.delete=function(cur){

if (cur.parent == null && cur.right == null && cur.left == null) {

this.root = null;

return;

}

if (cur.right == null) {

if (cur.parent == null) {

cur = cur.left;

cur.parent = null;

}

else if (cur.parent.left == cur)

cur.parent.left = cur.left;

else

cur.parent.right = cur.left;

if (cur.left)

cur.left.parent = cur.parent;

} else if (cur.left == null) {

if (cur.parent == null) {

cur = cur.right;

cur.parent = null;

}

else if (cur.parent.left == cur)

cur.parent.left = cur.right;

else

cur.parent.right = cur.right;

if (cur.right)

cur.right.parent = cur.parent;

} else {

var prev, temp = cur;

temp = cur.left;

prev = cur;

while (temp.right != null) {

prev = temp;

temp = temp.right;

}

cur.value = temp.value;

cur.json.name = cur.value;

if (prev == cur)

prev.left = temp.left;

else

prev.right = temp.left;

prev.set\_height();

prev = this.balance(prev);

while (prev.parent) {

prev = prev.parent;

prev.set\_height();

prev = this.balance(prev);

}

return;

}

while (cur.parent) {

cur.set\_height();

cur = this.balance(cur);

cur = cur.parent;

}

cur.set\_height();

this.root = this.balance(cur);

}

this.inorder = function(cur = this.root) {

var numbers = [];

if (cur != null) {

numbers = this.inorder(cur.left);

numbers.push(cur.value);

numbers = numbers.concat(this.inorder(cur.right));

}

return numbers;

}

this.preorder = function(cur = this.root) {

var numbers = [];

if (cur != null) {

numbers = [cur.value];

numbers = numbers.concat(this.preorder(cur.left));

numbers = numbers.concat(this.preorder(cur.right));

}

return numbers;

}

this.postorder = function(cur = this.root) {

var numbers = [];

if (cur != null) {

numbers = numbers.concat(this.postorder(cur.left));

numbers = numbers.concat(this.postorder(cur.right));

numbers.push(cur.value);

}

return numbers;

}

}

var bst=new BST\_AVL();

function sleep(delay)

{

var start=new Date().getTime();

while(new Date().getTime<start+delay);

}

var textBox = document.getElementById('value');

var msg = document.getElementById('msg');

var textName=document.getElementById('name');

var textSRN=document.getElementById('SRN');

var textAge=document.getElementById('Age');

textName.setAttribute("class","invisible");

textSRN.setAttribute("class","invisible");

textAge.setAttribute("class","invisible");

function disp()

{

textName.setAttribute("class","visible form-control");

textSRN.setAttribute("class","visible form-control");

textAge.setAttribute("class","visible form-control");

}

function hide()

{

textName.setAttribute("class","invisible");

textSRN.setAttribute("class","invisible");

textAge.setAttribute("class","invisible");

}

function InsertNode() {

if(textBox.value=='')

return;

var val = Number(textBox.value);

var obj={

name1:textName.value,

SRN:textSRN.value,

Age:textAge.value

};

bst.Insert\_val(val,obj);

textBox.value = '';

msg.innerHTML = '';

textAge.value='';

textName.value='';

textSRN.value='';

textBox.focus();

drawTree();

textName.setAttribute("class","invisible");

textSRN.setAttribute("class","invisible");

textAge.setAttribute("class","invisible");

}

function DeleteNode() {

if (textBox.value == '')

return;

var val = Number(textBox.value);

bst.delete\_val(val);

textBox.value = '';

drawTree();

msg.innerHTML = '';

textBox.focus();

}

function SearchNode() {

if (textBox.value != '') {

var val = Number(textBox.value);

var node = bst.search(val);

if (node == -1)

msg.innerHTML = 'not found';

else

msg.innerHTML = 'Name: '+ node.obj.name1+ ' SRN:'+node.obj.SRN+' Age:'+node.obj.Age;

}

textBox.value = '';

textBox.focus();

}

function print\_bal\_factor()

{

if (textBox.value != '') {

var val = Number(textBox.value);

var node = bst.search(val);

if (node == -1)

msg.innerHTML = 'not found';

else

msg.innerHTML ="The balance Factor of the node entered is "+node.bal\_fac();

}

textBox.value = '';

textBox.focus();

}

function Print(traversal) {

var numbers = traversal.call(bst);

msg.innerHTML = numbers.join(', ');

textBox.focus();

}

function drawTree() {

d3.select("svg").remove();

var treeData;

if (bst.root)

treeData = bst.root.json;

else

return;

var margin = {

top: 50,

right: 90,

bottom: 50,

left: 90

},

width = window.innerWidth - 10 - margin.left - margin.right,

height = window.innerHeight - 45 - margin.top - margin.bottom;

var treemap = d3.tree()

.size([width, height]);

var nodes = d3.hierarchy(treeData);

nodes = treemap(nodes);

var svg = d3.select("body").append("svg")

.attr("width", width + margin.left + margin.right)

.attr("height", height + margin.top + margin.bottom),

g = svg.append("g")

.attr("transform",

"translate(" + margin.left + "," + margin.top + ")");

var link = g.selectAll(".link")

.data(nodes.descendants().slice(1))

.enter().append("path")

.attr("class", "link")

.attr("d", function(d) {

if (d.parent && d.parent.children.length == 1) {

if (d.data.direction == 'right') {

if (d.parent.parent)

d.x += Math.abs(d.parent.x - d.parent.parent.x) / 2;

else

d.x += width / 4;

} else {

if (d.parent.parent)

d.x -= Math.abs(d.parent.x - d.parent.parent.x) / 2;

else

d.x -= width / 4;

}

}

return "M" + d.x + "," + d.y +

"C" + (d.x + d.parent.x) / 2 + "," + (d.y + d.parent.y) / 2 +

" " + (d.x + d.parent.x) / 2 + "," + (d.y + d.parent.y) / 2 +

" " + d.parent.x + "," + d.parent.y;

});

var node = g.selectAll(".node")

.data(nodes.descendants())

.enter().append("g")

.attr("class", function(d) {

return "node" +

(d.children ? " node--internal" : " node--leaf");

})

.attr("transform", function(d) {

return "translate(" + d.x + "," + d.y + ")";

});

node.append("circle")

.attr("r", 15);

node.append("text")

.attr("dy", ".35em")

.attr("y", function(d) {

return 0;

})

.style("text-anchor", "middle")

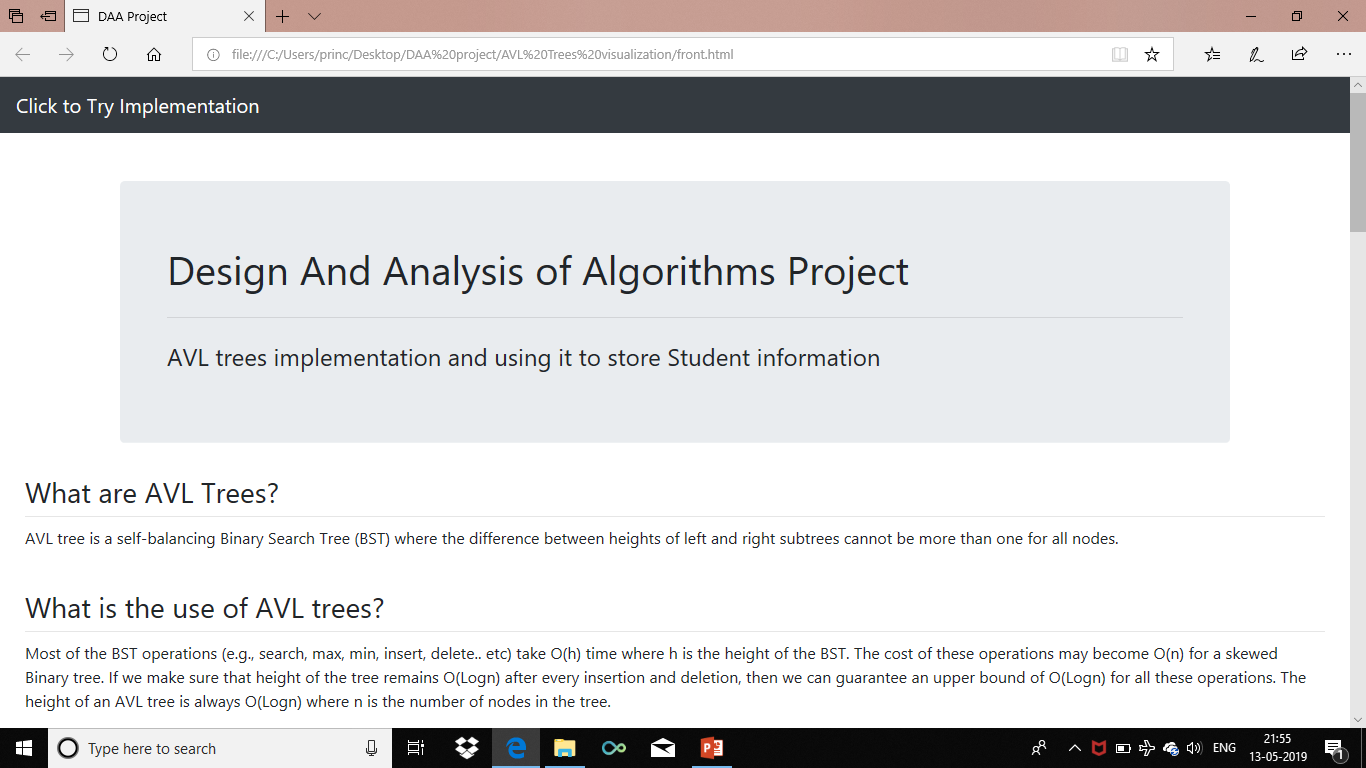
.text(function(d) {

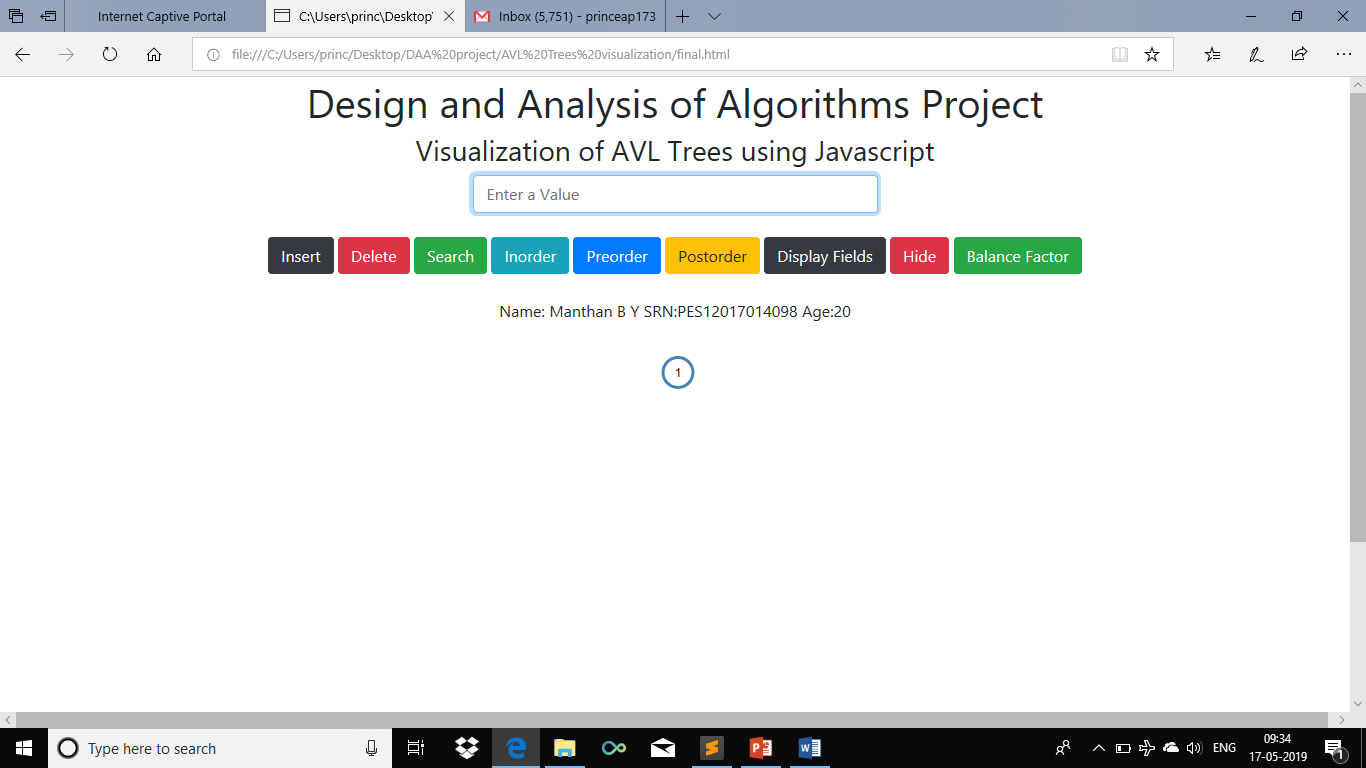
return d.data.name;

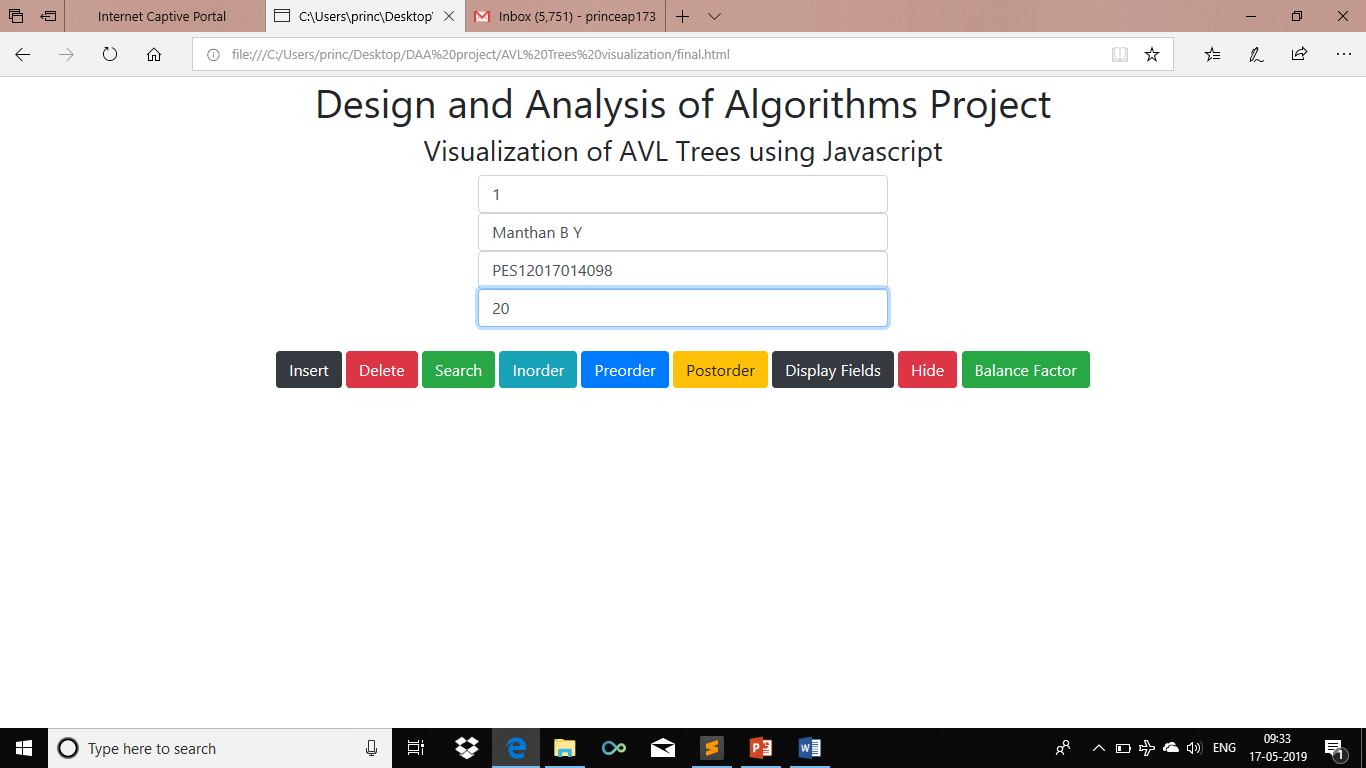
});

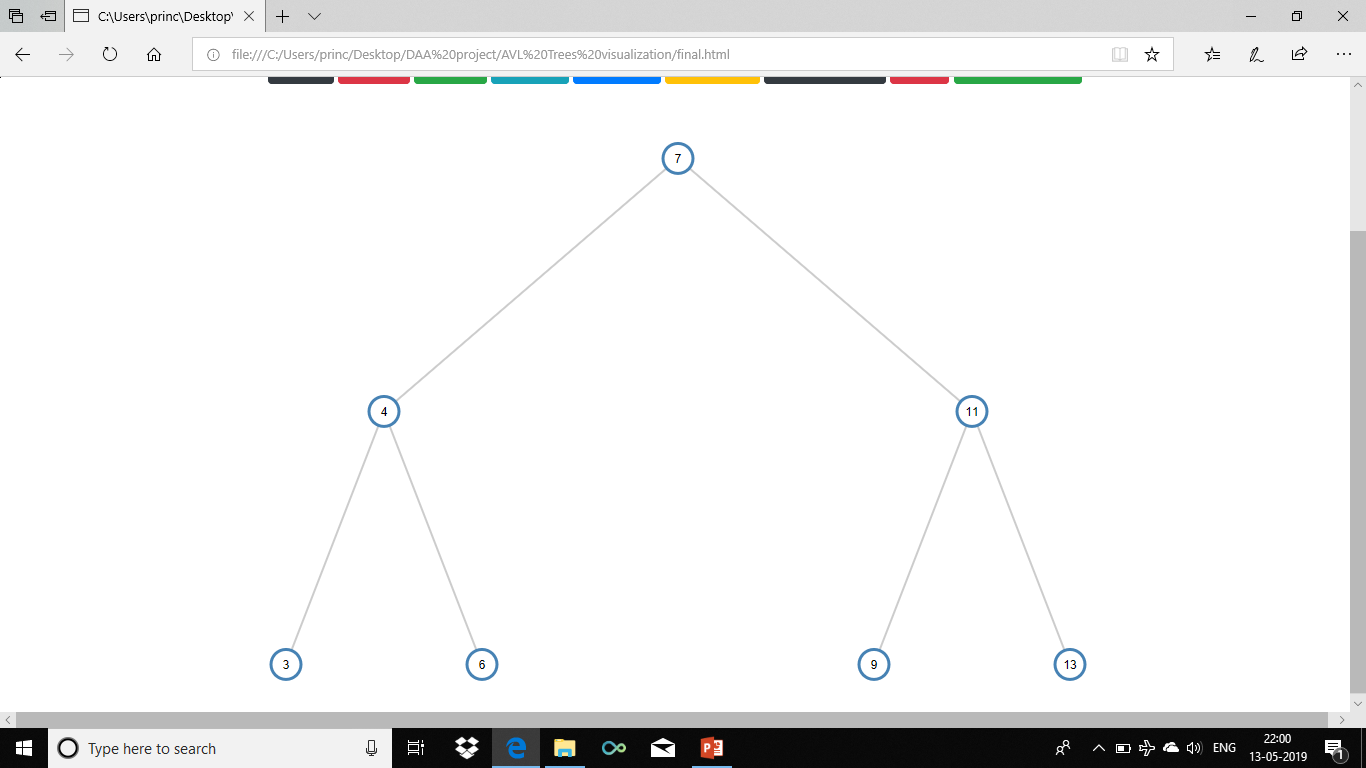
}

**Project Screenshots**

****





****

**Conclusion:**

Our project helps us to visualize AVL tree in a better way. It can be used by students to cross check the answers while learning. It reduces the time for the development of an AVL tree.

It can be used to store student data efficiently as retrieval, insertions, deletions take place in O(logN) time.

**References**

GeeksForGeeks -geeksforgeeks.com

TutorialsPoint-tutorialspoint.com

Wikipedia

D3 In Depth -www.d3indepth.com

BootStrap Documentation