Implementation of Binary Search Tree in Javascript - GeeksforGeeks

> A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

In this article, we would be implementing the Binary Search Tree data structure in Javascript. A tree is a collection of nodes connected by some edges. A tree is a non linear data structure. A Binary Search tree is a binary tree in which nodes that have lesser value are stored on the left while the nodes with a higher value are stored at the right.
Now let's see an example of a Binary Search Tree node:

```
class  Node

{

constructor(data)

{

this``.data = data;

this``.left =  null``;

this``.right =  null``;

}

}
```

As in the above code snippet we define a node class having three property *data*, *left* and *right*, Left and right are pointers to the left and right node in a Binary Search Tree. *Data* is initialized with *data* which is passed when object for this node is created and left and right is set to null.

Now let's see an example of a Binary Search Tree class.

```
class  BinarySearchTree

{
```

```
constructor()

{

this``.root =  null``;

}

}
```

The above example shows a framework of a Binary Search tree class, which contains a private variable *root* which holds the root of a tree, it is initialized to null.
Now lets implement each of this function:

1. **insert(data)** – It inserts a new node in a tree with a value *data*

```
insert(data)
{
var newNode =  new  Node(data);
if``(``this``.root ===  null``)
this``.root = newNode;
else
this``.insertNode(``this``.root, newNode);
}
insertNode(node, newNode)
{
if``(newNode.data < node.data)
{
if``(node.left ===  null``)
node.left = newNode;
else
this``.insertNode(node.left, newNode);
}
else
{
if``(node.right ===  null``)
node.right = newNode;
else
this``.insertNode(node.right,newNode);
}
}
```

In the above code we have two methods ***insert(data)*** and ***insertNode(node, newNode)***. Let's understand them one by one:-

- **insert(data)** – It creates a new node with a value data, if the tree is empty it add this node to a tree and make it a root, otherwise it calls *insert(node, data)*.
- **insert(node, data)** – It compares the given *data* with the data of the current node and moves left or right accordingly and recur until it finds a correct node with a null value where new node can be added.

2. **remove(data)** – This function removes a node with a given data.

```
remove(data)
{
this``.root =  this``.removeNode(``this``.root, data);
}
removeNode(node, key)
{
if``(node ===  null``)
return  null``;
else  if``(key < node.data)
{
node.left =  this``.removeNode(node.left, key);
return  node;
}
else  if``(key > node.data)
{
node.right =  this``.removeNode(node.right, key);
return  node;
}
else
{
if``(node.left ===  null  && node.right ===  null``)
{
node =  null``;
return  node;
}
if``(node.left ===  null``)
{
node = node.right;
return  node;
}
else  if``(node.right ===  null``)
```

```
{
node = node.left;
return  node;
}
var aux =  this``.findMinNode(node.right);
node.data = aux.data;
node.right =  this``.removeNode(node.right, aux.data);
return  node;
}
}
```

In the above code we have two methods ***remove(data)*** and ***removeNode(node, data)***, let understand them one by one:

- **remove(data)** – It is helper methods which call removeNode by passing root node and given data and updates the root of the tree with the value returned by the function
- **removeNode(node, data)** – It searches for a node with a given data and then perform certain steps to delete it.

While deleting a node from the tree there are three different scenarios as follows:-

- **Deleting the leaf node** – As leaf node does not have any children, hence they can be easily removed and null is returned to the parent node
- **Deleting a node with one child** – If a node has a left child, then we update the pointer of the parent node to the left child of the node to be deleted and similarly if a node have a right child then we update the pointer of the parent node to the right child of the node to be deleted
- **Deleting a node with two children** – In order to delete a node with two children we find the node with minimum value in its right subtree and replace this node with the minimum valued node and remove the minimum valued node from the tree

In the above code we have used ***findMinNode(node)***, it is defined in a helper method section.
Recommendation: Binary Search Tree | Set 2 (Delete) It contains a detail explanation and a video tutorial for deleting a node from a binary search tree.

Tree Traversal

Now Lets understand different ways of traversing a Binary Search Tree.

1. **inorder(node)** – It performs inorder traversal of a tree starting from a given *node*
   Algorithm for inorder:
   1. Traverse the left subtree i.e perform inorder on left subtree
   2. Visit the root
   3. Traverse the right subtree i.e perform inorder on right subtree

```
inorder(node)
{
```

```
if``(node !==  null``)
{
this``.inorder(node.left);
console.log(node.data);
this``.inorder(node.right);
}
}
```

2. **preorder(node)** – It performs preorder traversal of a tree starting from a given *node*.
   Algorithm for preorder:
   1. Visit the root
   2. Traverse the left subtree i.e perform preorder on left subtree
   3. Traverse the right subtree i.e perform preorder on right subtree

```
preorder(node)
{
if``(node !==  null``)
{
console.log(node.data);
this``.preorder(node.left);
this``.preorder(node.right);
}
}
```

3. **postorder(node)** – It performs postorder traversal of a tree starting from a given *node*.
   Algorithm for postorder:
   1. Traverse the left subtree i.e perform postorder on left subtree
   2. Traverse the right subtree i.e perform postorder on right subtree
   3. Visit the root

```
postorder(node)
{
if``(node !==  null``)
{
this``.postorder(node.left);
this``.postorder(node.right);
console.log(node.data);
}
}
```

Helper Methods

Let’s declare some helper method which is useful while working with Binary Search Tree.

1. **findMinNode(node)** – It searches for a node with a minimum value starting from *node*.

```
findMinNode(node)
{
if``(node.left ===  null``)
return  node;
else
return  this``.findMinNode(node.left);
}
```

   As seen in the above method we start from a *node* and keep moving to the left subtree until we find a node whose left child is null, once we find such node we return it.

2. **getRootNode()** – It returns the root node of a tree.

```
getRootNode()
{
return  this``.root;
}
```

3. **search(data)** – It searches the node with a value *data* in the entire tree.

```
search(node, data)
{
if``(node ===  null``)
return  null``;
else  if``(data < node.data)
return  this``.search(node.left, data);
else  if``(data > node.data)
return  this``.search(node.right, data);
else
return  node;
}
```

Note : Different helper method can be declared in the *BinarySearchTree* class as per the requirement.

Implementation

Now lets use the *BinarySearchTree* class and its different methods described above.

```
var BST =  new  BinarySearchTree();

BST.insert(``15``);

BST.insert(``25``);

BST.insert(``10``);

BST.insert(``7``);
```

```
BST.insert(``22``);

BST.insert(``17``);

BST.insert(``13``);

BST.insert(``5``);

BST.insert(``9``);

BST.insert(``27``);

var root = BST.getRootNode();

BST.inorder(root);

BST.remove(``5``);

var root = BST.getRootNode();

BST.inorder(root);

BST.remove(``7``);

var root = BST.getRootNode();

BST.inorder(root);

BST.remove(``15``);

var root = BST.getRootNode();

console.log(``"inorder traversal"``);

BST.inorder(root);

console.log(``"postorder traversal"``);

BST.postorder(root);

console.log(``"preorder traversal"``);

BST.preorder(root);
```

For more on binary trees, please refer to the following article: Binary tree Data Structure

This article is contributed by **Sumit Ghosh**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Source