DOĞRUSAL OLMAYAN VERİ YAPILARI

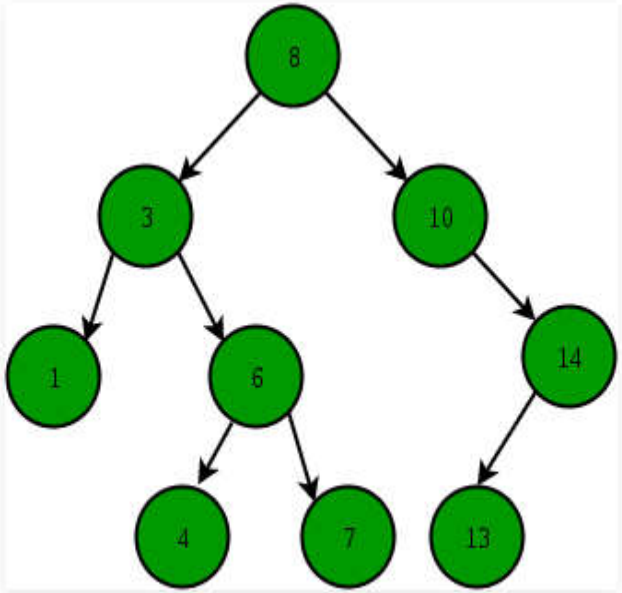
AĞAÇ YAPISI - BINARY SEARCH TREE

Binary Search Tree

Recent Articles on Binary Search Tree !

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.



DOĞRUSAL OLMAYAN VERİ YAPILARI

AĞAÇ YAPISI - BINARY SEARCH TREE

Tree (*Ağaç*) yapılarında, bir öğenin birden fazla alt öğesi olabilir

Hatta **Root** (*Kök*) öğe bile, gerekirse birden fazla olabilir.

Tree (*Ağaç*) veri yapılarına ilişkin tanımlamalar;

•**Root** (*Kök*)

Ağacın başlangıç node'udur

•**Node** (*Düğüm*)

Ağacın her bir node'una verilen isimdir

•**Child** (*Çocuk*)

Bir node'a bağlı olan node'lara verilen isimdir

•**Parent** (*Aile*)

Node'ların bağlı olduğu node'a verilen isimdir

•**Sibling** (*Kardeş*)

Aynı node'a bağlı olan node'lara verilen isimdir

•**Degree** (*Derece*)

Bir node'dan başlayarak en alt node'a gidene kadar geçilen katman sayıdır

•**Path** (*Yol*)

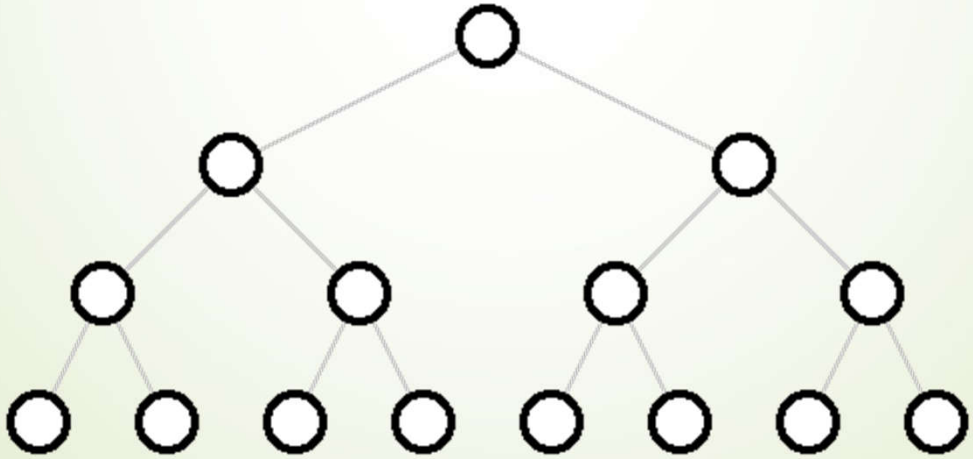
Bir node'a varabilmek için izlenmesi gereken yola verilen isimdir

DOĞRUSAL OLMAYAN VERİ YAPILARI

AĞAÇ YAPISI - BINARY SEARCH TREE

Binary Search Tree yapıları, **Binary Tree** yapılarının özelleşmiş bir halidir.

Root (*Kök*) olarak gene tek bir *node*'a sahiptir, her *node*, kendisine bağlı en fazla 2 *node*'a sahip olabilir. Bu noktaya kadar **Binary Tree** yapıları ile aynı özelliklere sahip, farklı olarak, *node*'lara bağlanan *child node*'ların içerdiği değerler, *parent* node'un kendi değerinden küçükse soldaki, büyük veya eşitse sağdaki node'a kaydedilir. Böylece, **Tree** (*Ağaç*) yapısı üzerinde yapılacak aramalar çok hızlı sonuçlanacaktır. **Tree** (*Ağaç*) yapısının **Degree** (*Derece*) değeri kaç ise, o kadar adımda aranan değere ulaşılabilir.



BST C# kodu

Insert (Ekle)


```
class Dugum
    {
        public int deger;
        public Dugum sol;
        public Dugum sag;
        public void goster()
        {
            Console.Write("[");
            Console.Write(deger);
            Console.Write("]");
        }
    }
    class Agac
    {
        public Dugum kok;
        public Agac()
        {
            kok = null;
        }
        public Dugum Returnkok()
        {
            return kok;
        }
```

BST C# kodu


```
public void Ekle(int id)
        {
            Dugum yeniDugum = new Dugum();
            yeniDugum.deger = id;
            if (kok == null)
                kok = yeniDugum;
            else
            {
                Dugum mevcut = kok;
                Dugum ebeveyn;
                while (true)
                {
                    ebeveyn = mevcut; mevcut.goster();
                    if (id < mevcut.deger)
                    {
                        mevcut = mevcut.sol;
                        if (mevcut == null)
                        {
                            ebeveyn.sol = yeniDugum;
                            return;
                        }
                    }
                    else
                    {
                        mevcut = mevcut.sag;
                        if (mevcut == null)
                        {
                            ebeveyn.sag = yeniDugum;
                            return;
                        }
                    }
            } } }
```

BST C# kodu


```
public void Preorder(Dugum kok)
{
    if (kok != null)
    {
        Console.Write(kok.deger + " ");
        Preorder(kok.sol);
        Preorder(kok.sag);
    }
}
public void Inorder(Dugum kok)
{
    if (kok != null)
    {
        Inorder(kok.sol);
        Console.Write(kok.deger + " ");
        Inorder(kok.sag);
    }
}
public void Postorder(Dugum kok)
{
    if (kok != null)
    {
        Postorder(kok.sol);
        Postorder(kok.sag);
        Console.Write(kok.deger + " ");
    }
}  }
```

BST C# kodu


```
class Program
    {
        static void Main(string[] args)
        {
            Agac BST = new Agac();
            BST.Ekle(30);
            BST.Ekle(35);
            BST.Ekle(57);
            BST.Ekle(15);
            BST.Ekle(63);
            BST.Ekle(49);
            BST.Ekle(89);
            BST.Ekle(77);
            BST.Ekle(67);
            BST.Ekle(98);
            BST.Ekle(91);
            Console.WriteLine("Inorder gezinti : ");
            BST.Inorder(BST.Returnkok());
            Console.WriteLine(" ");
            Console.WriteLine();
            Console.WriteLine("Preorder gezinti : ");
            BST.Preorder(BST.Returnkok());
            Console.WriteLine(" ");
            Console.WriteLine();
            Console.WriteLine("Postorder gezinti : ");
            BST.Postorder(BST.Returnkok());
            Console.WriteLine(" ");
            Console.ReadLine();
        }
    }
}
```

BST C# kodu

Delete (silme)

```
private Node DeleteN(Node root, Node deleteNode)
{
    if (root == null)
    {
        return root;
    }
    if (deleteNode.data < root.data)
    {
        root.left = DeleteN(root.left,
deleteNode);
    }
    if (deleteNode.data > root.data)
    {
        root.right = DeleteN(root.right,
deleteNode);
    }
```

```
    if (deleteNode.data == root.data)
    {
        //No child nodes
        if (root.left == null && root.right == null)
        {
            root = null;
            return root;
        }
        //No left child - DONT WORK
        else if (root.left == null)
        {
            Node temp = root;
            root = root.right;
            temp = null;
        }
        //sağ çocuk yok
        else if (root.right == null)
        {
            Node temp = root;
            root = root.left;
            temp = null;
        }
     }
    return root;
}
```

BST C# kodu

yselim

```
public object DeleteNode (object data)

{

    TNode tempDelete = this.GetNode(data);

    if (tempDelete != null)

    {

       if ((tempDelete.Left == null ) &&(tempDelete.Right == null)) //Its a
Leaf node

       {

          tempParent = tempDelete.Parent;

          if(tempDelete == tempParent.Left) //Justremove by making it null

              tempParent.Left = null;

          else

              tempParent.Right = null;

       }
```

BST C# kodu


```
else if ((tempDelete.Left == null ) ||(tempDelete.Right == null))
//It has either Left orRight child

        {

            tempChild = tempDelete.Left == null? tempDelete.Right :
tempDelete.Left; //Get the child

            tempParent = tempDelete.Parent; //Getthe parent

            if(tempDelete == tempParent.Left) //Makeparent points to it's
child so it will automatically deleted like Linked list

                tempParent.Left = tempChild;

            else

                tempParent.Right = tempChild;

           }
```

BST C# kodu


```
else if ((tempDelete.Left != null) ||(tempDelete.Right != null)) //It has
both Left andRight child
        {
           TNodepredNode = this.GetNode(this.TreePredecessor_Ite(data));
//Findit's predecessor

           if(predNode.Left != null) // Predecessor node canhave no or left
child. Do below two steps only if it has left child
           {
                tempChild = predNode.Left;
                predNode.Parent.Right = tempChild; //Assignleft child of
predecessor to it's Parent's right.
            }
            tempDelete.Data = predNode.Data; //Replace the value of
predecessor nodeto the value of to be deleted node
                  //predNode = null; //Remove predecessornode as it's no
longer required.
        }
         return data + " Deleted";
    }
    else
         return "Please enter the valid tree element!";

}
```