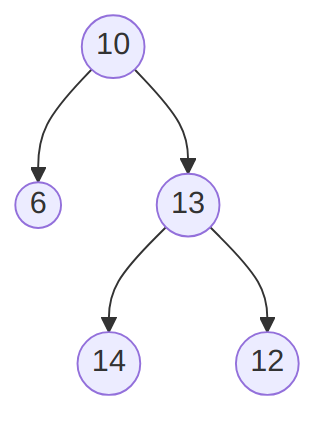
2025-09-30

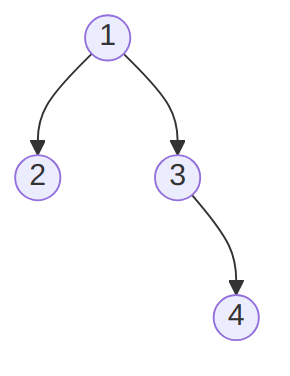
# Trees (Solutions)

## Exercises

1. Consider the following tree:

* 
* A binary tree that is not a binary search tree. ([text version](https:/princomp.github.io/diag/gra/bstree_example_4.txt), [image version](https:/princomp.github.io/diag/gra/bstree_example_4.png), [svg version](https:/princomp.github.io/diag/gra/bstree_example_4.svg))
  1. Explain why it is **not** a binary search tree.
  + Solution
  + The left child of the node with value 13 has value 14, which is greater than 13, hence violating the binary search tree principle that values in the left sub-tree should be strictly less than the value in the root of the subtree. The same goes for 12.
  1. Pick one among *inorder*, *preorder* and *postorder* traversal, and give
     1. A brief description of how it proceeds,
     + Solution
     + One among the following:
       - Inorder traversal processes (recursively) first the left subtree, then the data at the root, then the right subtree.
       - Preorder traversal processes (recursively) first the data at the root, then the left subtree, then the right subtree.
       - Postorder traversal processes (recursively) first the left subtree, then the right subtree, then the data at the root.
     1. What it would produce for the given tree.
     + Solution
     + One among the following:
       - Inorder gives 6, 10, 14, 13, 12
       - Preorder gives 10, 6, 13, 14, 12
       - Postorder gives 6, 14, 12, 13, 10

1. Consider the following implementation of “random” binary trees:

* public class RBTree<T>  
  {  
    
  private class Node  
   {  
   public T Data { get; set; }  
   public Node left;  
   public Node right;  
   public Node(  
   T dataP = default(T),  
   Node leftP = null,  
   Node rightP = null  
   )  
   {  
   Data = dataP;  
   left = leftP;  
   right = rightP;  
   }  
   }  
    
  private Node root;  
    
  public RBTree()  
   {  
   root = null;  
   }  
    
  public void Insert(T dataP)  
   {  
   root = Insert(dataP, root);  
   }  
    
  private Node Insert(T dataP, Node nodeP)  
   {  
   if (nodeP == null)  
   {  
   return new Node(dataP, null, null);  
   }  
   else  
   {  
   Random gen = new Random();  
   if(gen.NextDouble() > 0.5)  
   {  
   nodeP.left = Insert(dataP, nodeP.left);  
   }  
   else  
   {  
   nodeP.right = Insert(dataP, nodeP.right);  
   }  
   }  
   return nodeP;  
   }  
  }
* Note that the Insert(T dataP, Node nodeP) method uses the gen.NextDouble() > 0.5 test that will be randomly true half of the time, and false the other half.
  1. Explain the T dataP = default(T) part of the Node constructor.
  + Solution
  + This makes the first argument of the constructor optional: if no value is provided, then the default value for T is used. For example, for int, then 0 would be used.
  1. Write a ToString method for the Node class, remembering that only a node Data needs to be part of the string returned.
  + Solution
  + public override string ToString()  
     {  
     return Data.ToString();  
     }
  1. Write a series of statements that would
     1. create a RBTree object,
     2. insert the values 1, 2, 3, and 4 in it (in this order).
     + Solution
     + RBTree<int> btree = new RBTree<int>();  
        btree.Insert(1);  
        btree.Insert(2);  
        btree.Insert(3);  
        btree.Insert(4);
  2. Make a drawing of a possible RBTree obtained by executing your code.
  + Solution
  + Any binary tree containing 1, 2, 3 and 4, with 1 at the root, 2 a child of 1, 3 a child of 1 or 2, and 4 a child of 1, 2 or 3, is correct. One such example is:
  + 
  + The “random” binary tree obtained by inserting 1, 2, 3 and 4 (in that order). ([text version](https:/princomp.github.io/diag/gra/bstree_example_5.txt), [image version](https:/princomp.github.io/diag/gra/bstree_example_5.png), [svg version](https:/princomp.github.io/diag/gra/bstree_example_5.svg))
  1. Write a Find method that takes one argument dataP of type T and returns true if dataP is in the RBtree calling object, false otherwise.
  + Solution
  + public bool Find(T dataP)  
     {  
     bool found = false;  
     if (root != null)  
     {  
     found = Find(root, dataP);  
     }  
     return found;  
     }  
      
     private bool Find(Node nodeP, T dataP)  
     {  
     bool found = false;  
     if (nodeP != null)  
     {  
     if (nodeP.Data.Equals(dataP))  
     {  
     found = true;  
     }  
     else  
     {  
     found =  
     Find(nodeP.left, dataP)  
     || Find(nodeP.right, dataP);  
     }  
     }  
     return found;  
     }