Homework: Trees and Tree-Like Data Structures

This document defines the **homework assignments** for the "Data Structures" course @ Software University. Please submit a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems.

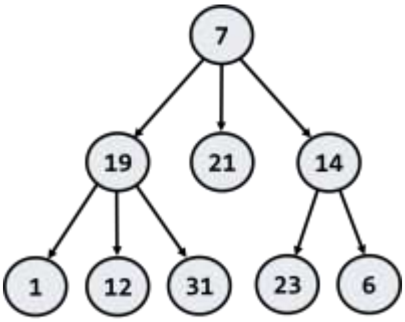
Problem 1. Play with Trees

You are given a **tree of N nodes** represented as a set of N-1 pairs of nodes (parent node, child node).

Write a program to read the tree from the console and find:

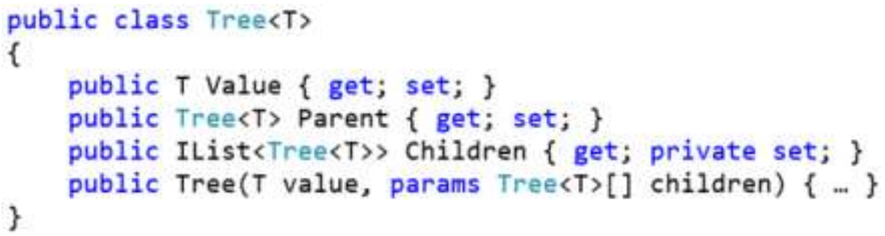
- The **root** node
- All **leaf** nodes (in increasing order)
- All **middle** nodes (in increasing order)
- * The **longest path** in the tree (the leftmost if several paths have the same longest length)
- * All paths in the tree with **given sum P** of their nodes (from the leftmost to the rightmost)
- ** All **subtrees with given sum S** of their nodes (from the leftmost to the rightmost)

Input	Comments	Tree	Output

9
7 19
7 21
7 14
19 1
19 12
19 31
14 23
14 6
27
43
	N = 9
	Nodes: 7→19, 7→21, 7→14, 19→1, 19→12, 19→31, 14→23, 14→6
	P = 27
	S = 43
			Root node: 7
			Leaf nodes: 1, 6, 12, 21, 23, 31
			Middle nodes: 14, 19
			Longest path:
			7 -> 19 -> 1 (length = 3)
			Paths of sum 27:
			7 -> 19 -> 1
			7 -> 14 -> 6
			Subtrees of sum 43:
			14 + 23 + 6

Hints:

- Use the recursive **Tree<T>** definition. Keep the **value**, **parent** and **children** for each tree node:

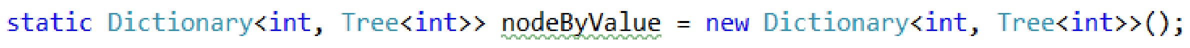


```
public class Tree<T>
{
    public T Value { get; set; }
    public Tree<T> Parent { get; set; }
    public IList<Tree<T>> Children { get; private set; }
    public Tree(T value, params Tree<T>[] children) { … }
}
```

- Modify the **Tree<T> constructor** to **assign a parent** for each child node:

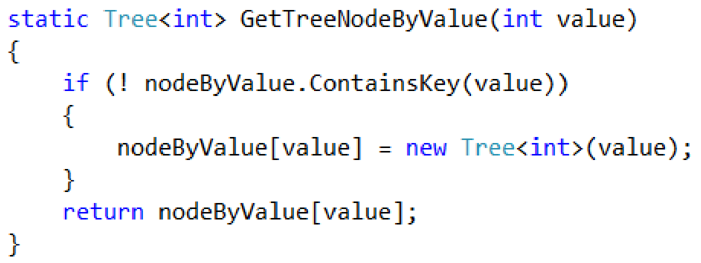
```
public Tree(T value, params Tree<T>[] children)
{
    this.Value = value;
    this.Children = new List<Tree<T>>();
    foreach (var child in children)
    {
        this.Children.Add(child);
        child.Parent = this;
    }
}
```

- Use a **dictionary** to map nodes by their value. This will allow you to find the tree nodes during the tree construction (when you read the input data, you get the node values):



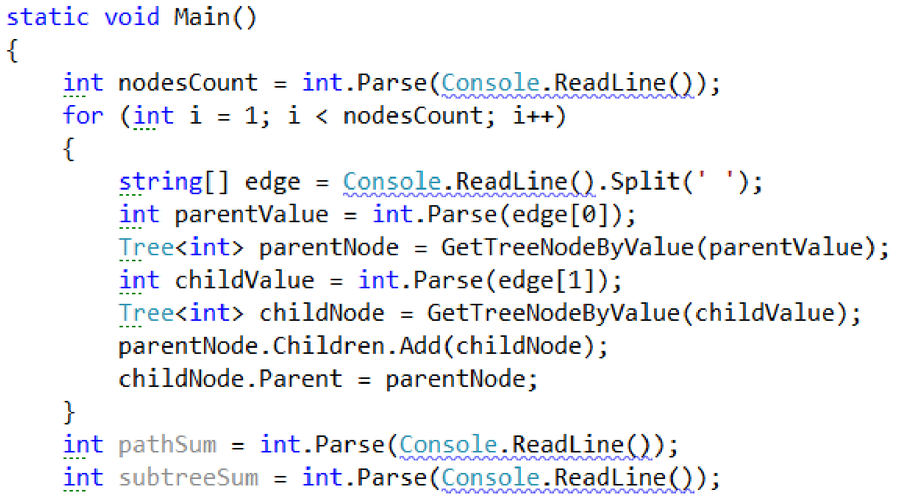
```
static Dictionary<int, Tree<int>> nodeByValue = new Dictionary<int, Tree<int>>();
```

- Write a method to **find the tree node by its value or create a new node** if it does not exist:



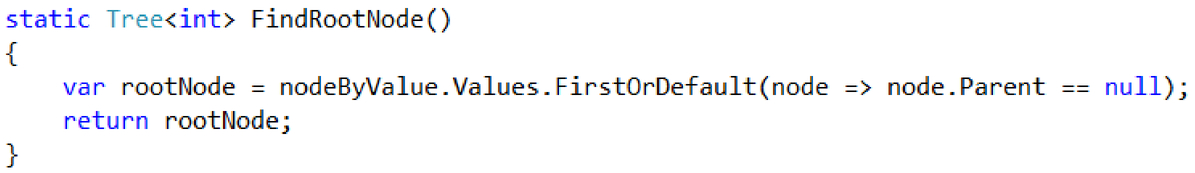
```
static Tree<int> GetTreeNodeByValue(int value)
{
    if (! nodeByValue.ContainsKey(value))
    {
        nodeByValue[value] = new Tree<int>(value);
    }
    return nodeByValue[value];
}
```

- Now you are ready to **read the input data**. You are given the **tree edges** (parent + child). Use the dictionary to lookup the parent and child nodes by their values:



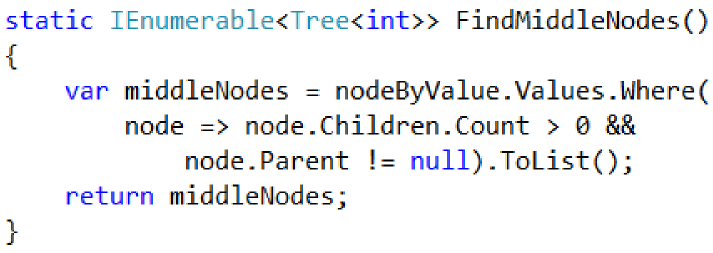
```
static void Main()
{
    int nodesCount = int.Parse(Console.ReadLine());
    for (int i = 1; i < nodesCount; i++)
    {
        string[] edge = Console.ReadLine().Split(' ');
        int parentValue = int.Parse(edge[0]);
        Tree<int> parentNode = GetTreeNodeByValue(parentValue);
        int childValue = int.Parse(edge[1]);
        Tree<int> childNode = GetTreeNodeByValue(childValue);
        parentNode.Children.Add(childNode);
        childNode.Parent = parentNode;
    }
    int pathSum = int.Parse(Console.ReadLine());
    int subtreeSum = int.Parse(Console.ReadLine());
```

- Find the **root** node:



```
static Tree<int> FindRootNode()
{
    var rootNode = nodeByValue.Values.FirstOrDefault(node => node.Parent == null);
    return rootNode;
}
```

- Find all **middle** nodes:



```
static IEnumerable<Tree<int>> FindMiddleNodes()
{
    var middleNodes = nodeByValue.Values.Where(
        node => node.Children.Count > 0 &&
            node.Parent != null).ToList();
    return middleNodes;
}
```

Problem 2. Traverse and Save Directory Contents in a Tree

Define two classes to keep files and folders:

- **`File { string name, int size }`**
- **`Folder { string name, File[] files, Folder[] childFolders }`**

Write a program to **build a tree keeping all files and folders** from the hard drive starting from **C:\WINDOWS**. You may use the .NET directory listing APIs: `DirectoryInfo.GetFiles()` and `DirectoryInfo.GetDirectories()`.

Implement a method that calculates the **sum of the file sizes in given subtree** of the tree and test it accordingly. **Use recursive tree traversal**.

Problem 3. *** Calculate Arithmetic Expression

Write a program to **calculate the value of given arithmetic expression**. Take into account that arithmetic operations have different priorities. Consider also processing brackets correctly. Handle the unary minus as well. Examples:

Input	Output
`5 + 6`	`11`
`(2 + 3) * 4.5`	`22.5`
`2 + 3 * 1.5 - 1`	`5.5`
`-2 - -1`	`-1`
`3 ++ 4`	`error`
`1.5 – 2.5 * 2 * (-3)`	`16.5`
`1/2`	`0.5`

Hint: consider implementing the "Shunting Yard" algorithm.