**Exercises: Trees Representation and Traversal (BFS and DFS)**

This document defines the lab for the ["Data Structures – Fundamentals (C#)" course @ Software University](https://softuni.bg/trainings/3672/data-structures-fundamentals-with-csharp-february-2022).

Please submit your solutions (source code) of all below-described problems in [Judge](https://judge.softuni.org/Contests/2468/04-Trees-Representation-and-Traversal-BFS-and-DFS-Exercise).

Write C# code for solving the tasks on the following pages.

## Tree Problems Overview

You are given a **tree of N + 1 nodes** represented as a set of **N** pairs of nodes (parent node, child node). For each problem, you have a specified definition of the implementation required to complete the task. You can use different approaches which you find suitable for the task you are solving.

In general, all the problems require basic tree knowledge and understanding of **DFS** and **BFS traversal algorithms**.

Because of the way you should read the tree from the console, it **cannot contain duplicate elements** in any circumstances. The tests do not include such cases as well.

For some of the exercises where the tree order does not matter, the tests’ input may be shuffled.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Comments** | **Tree** | **Definitions** |
| 7 19  7 21  7 14  19 1  19 12  19 31  14 23  14 6  27  43 | N = 8  Nodes:  7 🡪 19,  7 🡪 21,  7 🡪 14,  19 🡪 1,  19 🡪 12,  19 🡪 31,  14 🡪 23,  14 🡪 6  P = 27  S = 43 |  | Root key: 7  Leaf keys: 1, 6, 12, 21, 23, 31  Internal Keys: 14, 19  Leftmost deepest key: 1  Leftmost longest path: 7 -> 19 -> 1 (length = 3)  Paths of sum 27: 7 -> 19 -> 1 7 -> 14 -> 6  Subtrees of sum 43: 14 + 23 + 6 |

## Implement Factory Class

Implement the factory class provided in the skeleton. It should return the root of the tree after it builds it.

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
| 7 19  7 21  7 14  19 1  19 12  19 31  14 23  14 6 | Root node: 7 |  |

Hints

Use the recursive Tree<T> definition. Keep the **value**, **parent,** and **children** for each tree node. Use appropriate data structure to find and map nodes by their value. Write a logic to **find the tree node by its value or create a new node** if it does not exist. Create a mechanism to add an edge to the tree. Now you are ready to **create the tree**. You are given the **tree edges** (parent + child). Use the map to look up the parent and child nodes by their values. Finally, you can find the root (the node that has no parent).

## Tree As String

Implement the AsString method of the Tree class print it in the following format (each level indented +2 spaces):

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
|  | 7  19  1  12  31  21  14  23  6 |  |

Hints

Find the root and recursively print the tree.

## Get Leaf Keys

Implement the provided method. It should find all **leaf** nodes and return their keys:

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
|  | 1 12 31 21 23 6 |  |

## Internal Nodes

Implement the provided method. It should find all **internal** nodes **without** **the** **root** and return their keys.

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
|  | 14 19 |  |

## Deepest Node

Implement the provided method. It should find the deepest node (leftmost) in the tree and return it’s key.

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
|  | 1 |  |

## Longest Path to Root

Implement the provided method. Find the **longest path** to the root of the tree (the leftmost if several paths have the same longest length).

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
|  | 7 19 1 |  |

## All Paths With a Given Sum

Implement the method provided in the IntegerTree class. Find all paths (from **leaf to root or vice-versa**) in the tree with a **given sum** of their nodes (from the leftmost to the rightmost).

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
| 27 | 7 19 1  7 14 6 |  |

## All Subtrees With a Given Sum

Implement the method provided in the IntegerTree class. Find all **subtrees with the given sum** of their nodes (from the leftmost to the rightmost).

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Tree** |
| 43 | 14 23 6 |  |