Tutorial 9

Introduction

This tutorial focus on Tree implementation using array.
Example 01 demonstrates how to implement a generic tree using arrays (label array and parent array).
Example 02 shows the binary tree implementation using one array.
In the exercise section, students are asked to implement some additional operator on array-based Tree and array-based Binary Tree.

Examples

1. Example 01 – Array-based Tree

The Tree's ADT is introduced in the lecture 09, as well as the basic concept of Tree implementation using array. In this example, we consider a generic Tree with strings is the Tree's nodes labels. The following operators are implemented:

- isFull() : boolean – returns true if the current Tree is full, otherwise returns false.
- isEmpty() : boolean – returns true if the current Tree is empty, otherwise returns false.
- addNode(String *label*, int *parent*) : void – Adds a new node into the current Tree as the child of the node *parent*. The argument *label* contains the new node's label.
- getParent(int *node*) : int – returns the parent of *node* .
- getNodeLabel(int *node*) : String – returns the label of *node*.
- setNodeLabel(String *label*, int *node*) : void – set *node*'s label to the new *label* argument.
- leftMostChild(int *node*) : int – returns the left most child of the *node*. Returns -1 if the *node* has no children.
- rightSibling(int *node*) : int – returns the right sibling node of the *node*. Returns -1 if the *node* has no right sibling.

Please refer to class ***ArrayTree*** and ***ArrayTreeApp*** in the Tutorial Example Code.

2. Example 02 – Array-based Binary Tree

This example demonstrates how to implement a binary tree using array. Suppose that every node in the
Binary Tree has a string label. The following operators are implemented:

- addRoot(String *label*) : void – add the *root* (with *label*) to an empty binary tree.
- getLeftChild(int *node*) : int – returns the left child of *node*.
- getRightChild(int *node*) : int – returns the right child of *node*.

- getParent(int *node*) : int – returns the parent of *node*.
- getNodeLabel(int *node*) : String – returns label of current *node*.
- setNodeLabel(int *node*) : void – set label value for current *node*.
- addLeftChild(String *label*, int *node*) : void – add a node(with *label*) as the left child of *node*.

addRightChild(String *label*, int *node*) : void – add a node (with *label*) as the right chld of *node*.

- preOrderTravel(int *node*) : void – Tree traversal in pre-order starting from *node*.

Please refer to class ***ArrayBinaryTree*** and ***ArrayBinaryTreeApp*** in the Tutorial Example Code.

Exercises

1. Exercise 1

Please extend the ***ArrayTree*** class above to support the following operations on the Tree:

- getDegree(int *node*) : int - returns the degree of a node in the tree. The degree of a node is the numbers of its children.
- isLeaf(int *node*) : boolean - returns true if current *node* is a leaf of the tree, otherwise the method returns false. A leaf node is the node which has no children.

  countLeaves() : int – returns the total number of leaves in the tree.
- getLevel(int *node*) : int – returns the level of current *node*. The level of a node is the distance from this node to the root of the tree.

getDepth(): int – returns the depth of the tree which is the maximum level of all leaves of the tree.

- search(String *label*) : int – returns the position of the first node which has *label* as the label value. The method will return -1 if the *label* not found.

2. Exercise 2

Please extend the ***ArrayBinaryTree*** class above to support the following operations on the Binary Tree:

- isLeaf(int *node*) : boolean - returns true if current *node* is a leaf of the tree, otherwise the method returns false. A leaf node is the node which has no children.
- countLeaves() : int – returns the total number of leaves in the tree.
- getLevel(int *node*) : int – returns the level of current *node*. The level of a node is the distance from this node to the root of the tree.

- getDepth(): int – returns the depth of the tree which is the maximum level of all leaves of the tree.
- inOrderTravel(int *node*) : void – Tree traversal in in-order starting from *node*.
- postOrderTravel(int *node*) : void – Tree traversal in post-order starting from *node*.
- search(String *label*) : int – returns the position of the first node which has *label* as the label value. The method will return -1 if the *label* not found.