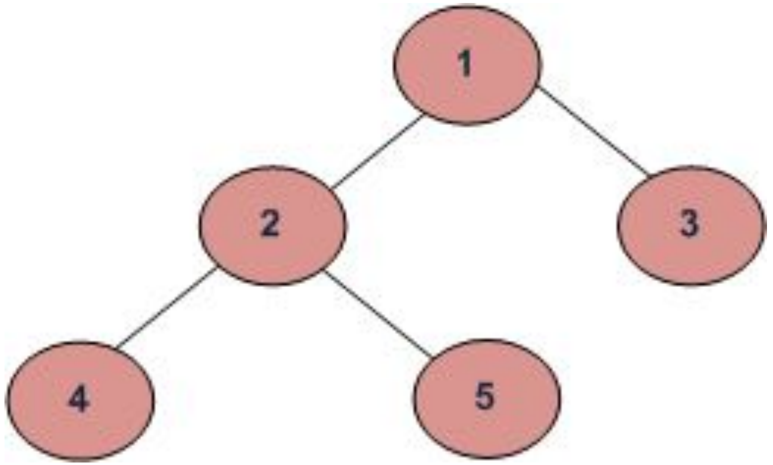
Data Structure 2019
Lab 08

MohammadSadegh Najafi - Jonghyun Lee

May 3, 2019

Binary Tree

A tree data structure in which each node has at most 2 children, typically called left and right child.



Binary Tree Traversal

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways.

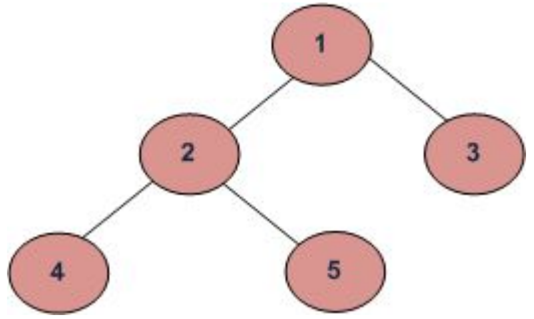
The following are the generally used ways for traversing trees.

(a) Inorder : 4 2 5 1 3

(b) Preorder : 1 2 4 5 3

(c) Postorder : 4 5 2 3 1

(d) Level Order : 1 2 3 4 5



Inorder Traversal

Inorder(node)

1. Traverse the left subtree, i.e., call Inorder(node)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(node)

Preorder Traversal

Preorder(node)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(node)
3. Traverse the right subtree, i.e., call Preorder(node)

Postorder Traversal

Postorder(node)

1. Traverse the left subtree, i.e., call Postorder(node)

2. Traverse the right subtree, i.e., call Postorder(node)

3. Visit the root.

Today’s Task:

Given `BinaryTree.java` interface, implement it in `myTree.java` using a linked representation of binary trees

1) Implementing linked representation of binary trees
2) Implementing binary tree traversals and other useful methods

Today's Task 1 : Linked Binary Tree representation

1) Create the necessary data members and constructor(s) for `myTree` class. You should include as data members an '`Object root`', a left child, and a right child.

2) Create new `BinaryTreeNode` class in myTree.java file, and add to it the data members, constructor(s) and methods as needed

3) Implement the methods on the right from `BinaryTree` interface.

public boolean isEmpty()
- @return true iff tree is empty */

public Object root()
- @return root element if tree is not empty
- @return null if tree is empty

public BinaryTree removeLeftSubtree()
- remove the left subtree
- @throws IllegalArgumentException when tree is empty
- @return removed subtree

public BinaryTree removeRightSubtree()
- remove the right subtree
- @throws IllegalArgumentException when tree is empty
- @return removed subtree

Today’s Task 2: Implementing other Binary Tree methods

1) Implement the rest of the methods in the `BinaryTree` interface, shown on the right.

   - You may create helper methods as needed.

   - For some of them, a recursive solution will facilitate the implementation.

public int size()
- count and @return number of nodes in tree

public int height()
- @return tree height

public String preOrder()
- @return a String of the preorder traversal
- one space after each element (see next slide)

public String inOrder()
- @return a String of the inorder traversal
- one space after each element (see next slide)

public String postOrder()
- @return a String of the postorder traversal
- one space after each element (see next slide)

Output (running myTree.java)

```
[ds19@ds] ~$ javac myTree.java
[ds19@ds] ~$ java myTree
Preorder sequence is
1 2 4 8 9 5 10 12 13 11 3 6 7

Inorder sequence is
8 4 9 2 12 10 13 5 11 1 6 3 7

Postorder sequence is
8 9 4 12 13 10 11 5 2 6 7 3 1

Number of nodes = 13

Height = 5
Preorder sequence is
1 2 4 8 9 5 11 3 6 7

Number of nodes = 10

Height = 4

Postorder sequence is
6 7 3 1

Number of nodes = 4

Height = 3

[ds19@ds] ~$
```