BTree: An Advanced Haskell Library For Binary Tree

Xinyao Peng (xp2196), Ari An (ja3569)

Abstract

We aim to create a package for binary trees and relevant operations. The Haskell official packages ("Data.BinaryTree" and "Data.Tree") merely consist of basic operations and lack useful functions. Our goal is to optimise the package and add in a few basic and advanced operations. It might help to skip some tedious work and dive into some complex algorithm. Note: We define a binary tree as the node containing an element with left and right subtrees. Operations we might include are as follows.

Basic Operation

1. treeToList: converts a binary tree into a list of integers.
2. listToTree: converts a list of integers into a binary tree
3. printTree: display a "diagram" of binary tree
4. showPath: display all root-to-leaf paths of a given binary tree

Advanced Operations

*Note in all the examples below: tree is defined as [root, child1,child2, child1 of child1, child2 of child1,etc]

1. ifSubtree

Given two binary trees, return true if one of the trees is subtree of another and false otherwise. Note that subtree is a tree that consists of a node in the tree and all of its descendants. For example, if the input trees are [1,2,3,4,5] and [2,4,5], where tree [2,4,5] is subtree of tree [1,2,3,4,5], then the output is true.

2. pruneNode

Given a binary tree and a node, cut off the node and its children, and return the new tree after pruning. For example, if the input tree is [1,2,3,4,5] and the node value is 2, then the list version of output is [1,3].

3. getDepthNode

Given a binary tree and the number of depth, return all nodes in the current depth. For example, if the input tree is [1,2,3,4,5], and the number of depth is 2, then the output is [2,3]. We will apply depth-first search in this function.

4. searchSubtree

Given a binary tree and a value of node, return the subtree rooted with that node if the node is found in the binary tree and return an empty tree if not found. For example, if the input tree is [1,2,3,4,5] and the value of node is 2, then the output is [2,4,5]. We will apply greedy algorithm in this function.

5. sortTree

Given a binary tree, sort nodes in each depth and return a binary search tree. For example, if the original input tree is [1,3,2,4,6,5], then the output tree is [1,2,3,4,5,6], which is defined as a binary search tree.

6. findMaximumPath

Given a binary tree, find a path of nodes with maximum sum. Note that each node can appear in the path at most once and the path must start from the root. For example, if the original input tree is [1,2,3,4,5,6], then the maximum-sum path is [1,3,6].

7. findPath

Given a binary tree, a start node and an end node, find the shortest path from the start node to the end node. For example, if the original input tree is [1,2,3,4,5,6], the start node is 4 and the end node is 6, then the path is [4,2,1,3,6].