Due: Friday, 11/22/2019 before 5:00PM

Total points = 50

1 Overview

In this assignment, we will review binary tree structure, traversals of binary trees, and enforcing a binary heap property. We will also observe the relation between the binary heap as a linked tree structures and the flattened version inside an array.

2 Objectives

In this assignment you will:

- Traverse trees in order to ensure a heap property.
- Build and maintain an immutable data storage.
- Write tests for your code.

3 Instructions

Problem 1. (5+5 points) **Expect this problem to take 1-2 hours.**
Complete the method `buildHeapTreeFromHeapArray` in `TreeUtilities.scala`. Given an input array containing a binary heap, this method should build the tree representation of the given heap. Notes:

- The array contains a valid heap. An empty array is a valid heap.
- All leaves in the tree should be an `Empty` object. This means a tree should never be `null`.

You should submit comprehensive tests for this function in `TreeUtilitiesTest.scala`. Each test should be prefixed with `buildHeapTreeFromHeapArray:` (or immediately follow behavior of `"..."`) or they will not be graded for this part.

Problem 2. (5+5 points) **Expect this problem to take <1 hour.**
Complete the method `flattenHeapTreeToHeapArray` in `TreeUtilities.scala`. Given an input tree containing a binary heap, this method should construct the array representation of the given heap. Notes:

- The tree contains a valid heap. An `Empty` tree is a valid heap.
- All leaves in the tree should be an `Empty` object and do not appear within the actual heap.

You should submit comprehensive tests for this function in `TreeUtilitiesTest.scala`. Each test should be prefixed with `flattenHeapTreeToHeapArray:` (or immediately follow behavior of `"..."`) or they will not be graded for this part.

haohuafe@buffalo.edu - Nov 13, 2019, 10:59:57 AM EST

Problem 3. (10 points) **Expect this problem to take 1-2 hours.**
Complete the method `isValidBinaryHeap` in `TreeUtilities.scala`. Given an input tree containing a binary tree and a partial ordering predicate, this method should return true if the input binary tree is a valid maximum binary heap and return false otherwise. Hint: carefully consider the fact that `comp` is only a partial ordering. Notes:

- The value of `root` will always be a valid `Tree`.
- The function `comp` will always satisfy the properties of a partial ordering.

Problem 4. (10 points) **Expect this problem to take 1-2 hours.**
Complete the method `applyTree` in `TreeUtilities.scala`. Given a complete binary tree as input, this method should return the value stored within the node at index `index` within the tree. The index of a vertex corresponds to the index of the vertex within the flattened array form of the input tree. Notes:

- You may not flatten the tree and then return the value at the given index.
- The input tree will be a complete binary tree.
- If the index is not valid, you should return `None`.

Problem 5. (10 points) **Expect this problem to take 3-5 hours.**
Complete the method `updateHeap` in `TreeUtilities.scala`. Given a binary heap as input, this method should return the root of a new binary heap with the value at `index` updated to the given value. The index of a vertex corresponds to the index of the vertex within the flattened array form of the input tree. Notes:

- `Node` is immutable and cannot be changed. You must create a new `Node` with a different value to make a change.
- You should not produce an entirely new tree. You should only modify at most the branch containing the node to update.
- This also means you should not flatten the tree, update the value, and then create a new heap.
- The input tree will be a valid binary tree.
- You may assume that `comp` is a total ordering.
- You should follow the algorithms from lecture to make updates correctly.
- If the tree is unmodified, return the root of the original tree.

Allowed Library Usage

You may use any containers from `collection.mutable` or `collection`. You may not use `breakable`. It isn't needed.

Recommended Approach

You are strongly encouraged to complete and test the problems in the order they are given. If you do not, you will likely not complete the assignment.

haohuafe@buffalo.edu - Nov 13, 2019, 10:59:57 AM EST

4 Submission

For each section (PA4 Programming and PA4 Testing), you will be allowed 5 submissions to each, without penalty. Starting from the 6th submission, you will receive

- a 5 points per submission deduction from your score on the respective assignment.

Note: your score is what you receive on your latest submission. If you receive a score and then resubmit, even if you receive a lower score/0 points, that will be your score for the assignment.

Also note: if you submit early/late, the bonus/penalty would be awarded against the entirety of the assignment, not just the single part you submitted early/late. The timestamp for your submission is that of the latest part submitted.

Creating Your Submission

Your submission should be a single file `TreeUtilities.scala` for PA4 Programming and the single file `TreeUtilitiesTest.scala` for PA4 Testing. No other files should be provided for grading.

haohuafe@buffalo.edu - Nov 13, 2019, 10:59:57 AM EST