Com S 311 Spring 2021

Bonus Assignment: Complete Binary Trees

Due: May 4, 11:59 pm
No late submission is accepted

Haadi Majeed
In collaboration with Nathan Tucker and Matthew Hoskins

Guidelines

- Your score (out of a total of 65 points) in this bonus assignment will be added to your HW2 score.
- For each problem, if you write the statement "I do not know how to solve this problem" (and nothing else), you will receive 20% credit for that problem. If you do write a solution, then your grade could be anywhere between 0% to 100%. To receive this 20% credit, you must explicitly state that you do not know how to solve the problem.
- You are allowed to discuss with classmates, but you must work on the homework problems on your own. You should write the final solutions alone, without consulting anyone. Your writing should demonstrate that you understand the proofs completely.
- When proofs are required, you should make them both clear and rigorous. Do not hand-waive.
- Please submit your assignment via Canvas.
  - You **must** type your solutions. Please submit a PDF version.
  - Please make sure that the file you submit is not corrupted and that its size is reasonable (e.g., roughly at most 10-11 MB).

    If we cannot open your file, your homework will not be graded.
- Any concerns about grading should be expressed within one day of returning the homework.

Problem

Consider a nearly complete binary tree that is represented by an array A of elements, where each node of the tree corresponds to an element of the array, and the root of the tree corresponds to $A[1]$. Let n be the number of nodes in the tree, which is also the length of the array A. For index $i = 1, 2, 3, ..., n$, the value of node i is $A[i]$, and the indices of its parent, left child and right child (if they exist) are $\lfloor i/2 \rfloor$, $2i$ and $2i+1$. The height of a node in the tree is the number of edges on the longest simple downward path from the node to a leaf, and the height of the tree is the height of the root.

Given an array A of n numbers which represents a nearly complete binary tree of height h, we want to produce an array B of length h such that for $k = 1, 2, 3, ..., h$, $B[k]$ is the value at the last node of height k (with the largest index). Note that h is a function of n. For this assignment, the values in the nodes may not satisfy a heap property.

Example (30 points)

Given the following array A of numbers, write down all the values in the array B in increasing order of array indices.

```
Array A: 10 30 20 50 70 55 65 25 45 85 15 90 75 60 35 80 95 40

Index  :  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18

Array B: 45 50 30 10

Index  :  1  2  3  4
```

Algorithm (35 points)

Design an $O(\log n)$ algorithm for producing the output array B on an input array A of length n. Analyze your algorithm to obtain its running time.

```
1  ArrB
   |  Input: Array of a BST a
2  |  if a.size==1 then
3  |  |  return
4  |  height = 0
5  |  node = a[a.size]
6  |  while node ≥ 1 do
7  |  |  height++
8  |  |  node = ⌊node/2⌋
9  |  b = arr with size height
10 |  node = a[a.size]
11 |  for i= 0 to height do
12 |  |  b.add(a[⌊node/2⌋])
13 |  |  node = ⌊node/2⌋
14 |  return b
```

With the algorithm from above, we can use the BST to determine the height and last index with relative ease. We can establish the height of the tree by starting at the end of the array and working backwards from it; we know that the last element in the array is a leaf node that is the deepest within the tree. From there we can find the parent of the node, and then the parent of that one until we encounter the root node of the tree, the number of iterations that occur in that is how deep the tree is. Due to the method of traversal, we go by increments of $\lfloor \frac{i}{2} \rfloor$ for every iteration which results in a run time of $log(n)$.
With the height being found, we can then follow the path we used to determine the hieght back to the root node to assign to the new array B. Because it utilised the same path as prior, this also will have a runtime of $log(n)$
Summed together, it has a runtime of $2 * log(n) \rightarrow O(log(n))$