Solutions to ADC practice questions: Binary trees

Question 1.

Prove the following proposition:

> Depth d of a binary tree has a maximum of 2^d nodes.

Solution

The proof is by induction on d. n_d is the number of nodes at depth d of a given binary tree. It is then necessary to prove that $n_d \leq 2^d$ always holds.

Base case. The base case is $d = 0$. A binary tree of depth 0 has by definition precisely 1 node, so $n_d = 1$. Moreover, $2^0 = 1$holds. So the inequality holds for the base case.

Induction step. Let d be any number larger than 0, and assume that $n_k \leq 2^k$ for all $0 \leq k < d$.

From this, the following can be derived:

$$n_d \leq 2 \cdot n_{d-1} \leq 2 \cdot 2^{d-1} = 2^d$$

The first inequality follows from the fact that every node in a binary tree has a maximum of two children, and the second inequality follows from the induction hypothesis.

Question 2.

Prove the following proposition:

> A binary tree with height h has a maximum of $2^{h+1} - 1$ nodes.

Solution

The proof is by induction on h. n_h is the maximum number of nodes of a binary tree of height h. It is then necessary to prove that $n_h \leq 2^{h+1} - 1$ always holds.

Base case. The base case is $h = 0$. A binary tree of height h as by definition precisely 1 node, so $n_h = 1$. Moreover, $2^1 - 1 = 1$. So the inequality holds for the base case.

Induction step. Let h be any number larger than 0, and assume that $n_k \leq 2^{k+1} - 1$ for all $0 \leq k < h$.

From this, the following can be derived:

$$n_h \leq 1 + 2 \cdot n_{h-1} \leq 1 + 2 \cdot (2^{(h-1)+1} - 1) = 1 + 2 \cdot 2^h - 2 = 2^{h+1} - 1$$

The first inequality follows from the fact that a tree of depth h can be regarded as a root with thereunder two trees of a height of a maximum $h - 1$, and the second inequality follows from the induction hypothesis.

Question 3.

Prove the following proposition:

A binary tree with n nodes has a height of at least $\lceil \log(n+1) \rceil - 1$.

Solution

Suppose that $n = 2^k - 1$, for any k. Then $\log(n+1)$ is an integer. It follows from the solution to the last question that a given binary tree of height $\log(n+1) - 1$ can have a maximum of $2^{\log(n+1)-1+1} - 1 = n$ nodes. The tree is then completely filled. It can readily be appreciated that the tree will not be completely filled when n is just a little smaller than $2^k - 1$. $\log(n+1)$ will need to be rounded up to find the correct integer length, so the height is at least $\lceil \log(n+1) \rceil - 1$.

Question 4.

Given a binary tree with positive integer keys. Give an algorithm that determines whether the tree is sorted in order.

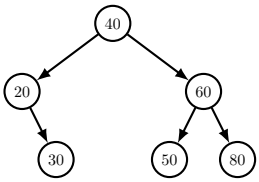
Solution

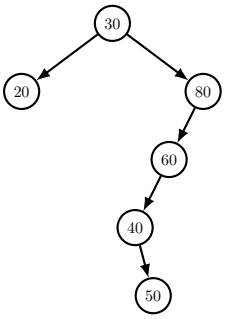
The tree can readily be verified as being sorted in order by working through the tree and checking that the key of each node is at least as large as its parent. The algorithm is: A value `val` and a `tree` are assigned to each call. The result consists of an integer and a boolean, whereby the boolean indicates whether the tree `tree` is sorted in order (and all values are at least as large as `val`), and the integer indicates the largest key in the tree. As all keys are positive, the method can initially be called with `val` $= 0$.

```
def sortedInOrder(val,tree)
    if tree == null:             # Obviously, an empty tree is sorted. The
        return val,True          # largest value contained in the tree does not change.

    # Go to the left, and swap with the largest key found until now.
    # Use this method to check that the left sub-tree is sorted in order and
    # that all keys are smaller than the key of the root.
    # The integer max is assigned to the largest key of the left sub-tree.

    max,ok = sortedInOrder(val,tree.left)

    if not ok or tree.key<max:    # If the left sub-tree is not sorted or the key
        return max,False          # of the root is smaller than the largest key of the
                                  # left sub-tree then the tree is not sorted.

    # Check that the right tree is also sorted,
    # whereby the new maximum is the key of the root:

    return sortedInOrder(tree.key,tree.right)
```

Question 5.

Take the following BSTs:

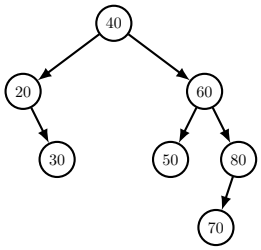


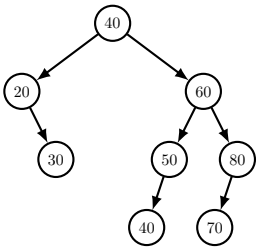


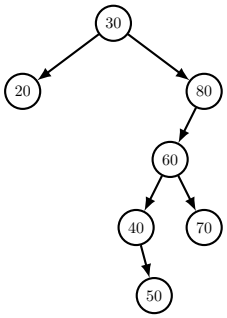
1. In both trees, insert first 70 and then 40.
2. From the above results, delete first 60 and then 70.

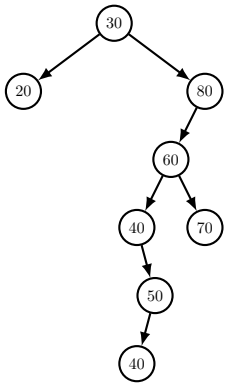
Solution

1. Inserting first 70 and then 40 in both trees yields the following results:

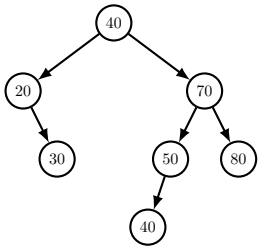


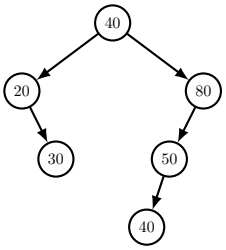


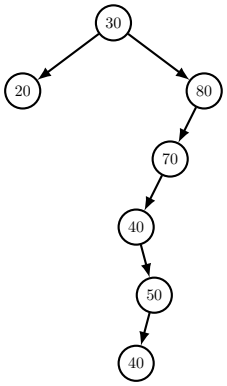


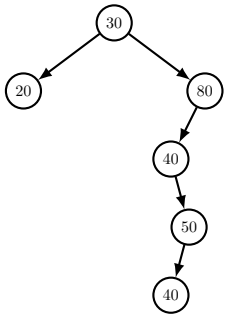


2. Deleting first 60 and then 70 from both trees yields the following trees:









Question 6.

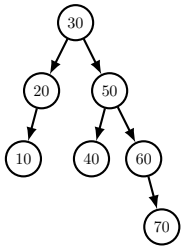
Give the definition of a *balanced* binary tree. Then give a criterion for the balancedness of a binary tree of n nodes n terms of the depth of the tree.

Solution

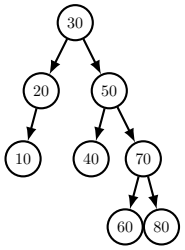
There are at least three potential definitions of balancedness:

1. A binary tree is balanced when every level, except possibly the last, is completely filled.
2. A binary tree is balanced when it has the minimum possible depth.
3. A binary tree is balanced when the difference between the depth of the left and the right sub-tree from every node is at most 1.

The fact that these definitions are different is apparent from an inspection of the following two binary trees:



(a)



(b)

Tree (a) is not balanced according to the first criterion, as the second level is not completely filled. Nor is the second criterion met, as a configuration of 7 nodes with a smaller depth is also feasible (by moving the node currently at the bottom to the empty position to the bottom right of 20 (and then, obviously, sorting the keys once again to restore the sequence)). However, the tree does meet the third criterion. The third criterion would appear to be less stringent than the first two.

Tree (b) is not balanced according to the first criterion as, once again, the second level is not completely filled. However, the tree is balanced according to the second criterion, as moving one of the two bottom nodes to the empty position under 20 would not change the depth of the tree. The tree is also balanced according to the third criterion.The second criterion would appear to be less stringent than the first.

The third definition is the most customary definition, as this definition is the easiest to maintain. Nevertheless, this definition does result in trees that are reasonably balanced (with properties including the property that the difference in the depth of two leaves is always at most 1).

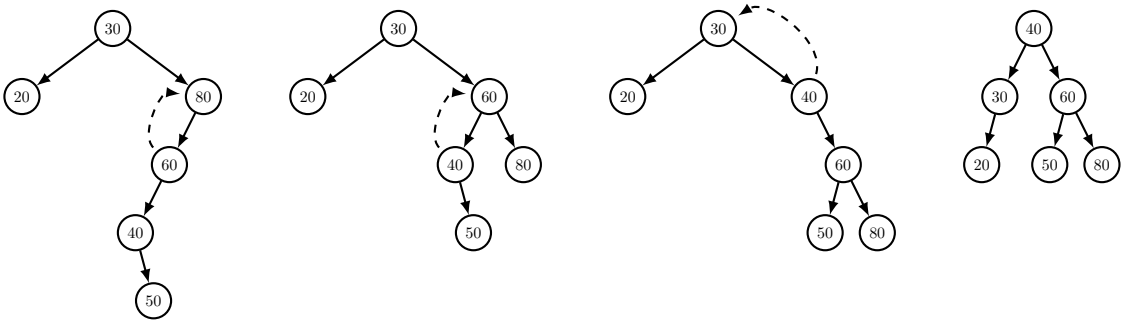
It is known that the minimum depth of a binary tree with n nodes is $\lfloor \log n \rfloor$. Consequently, the tree can be regarded as balanced when its depth is actually $\lfloor \log n \rfloor$ (note that this is in agreement with the second criterion referred to above).

Question 7.

Take the second BST in question 5, and transform it into a balanced tree using rotations.

Solution

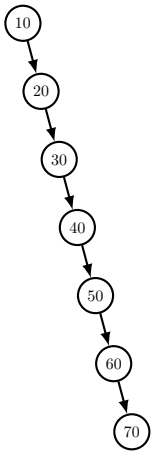
Question 8.



1. Insert, in sequence, the elements 10, 20, ..., 70 in an initially empty BST. How balanced is the result?
2. Now repeat the above, but in this case rotate the tree to restore its balance when the insertion of an element unbalances the tree.

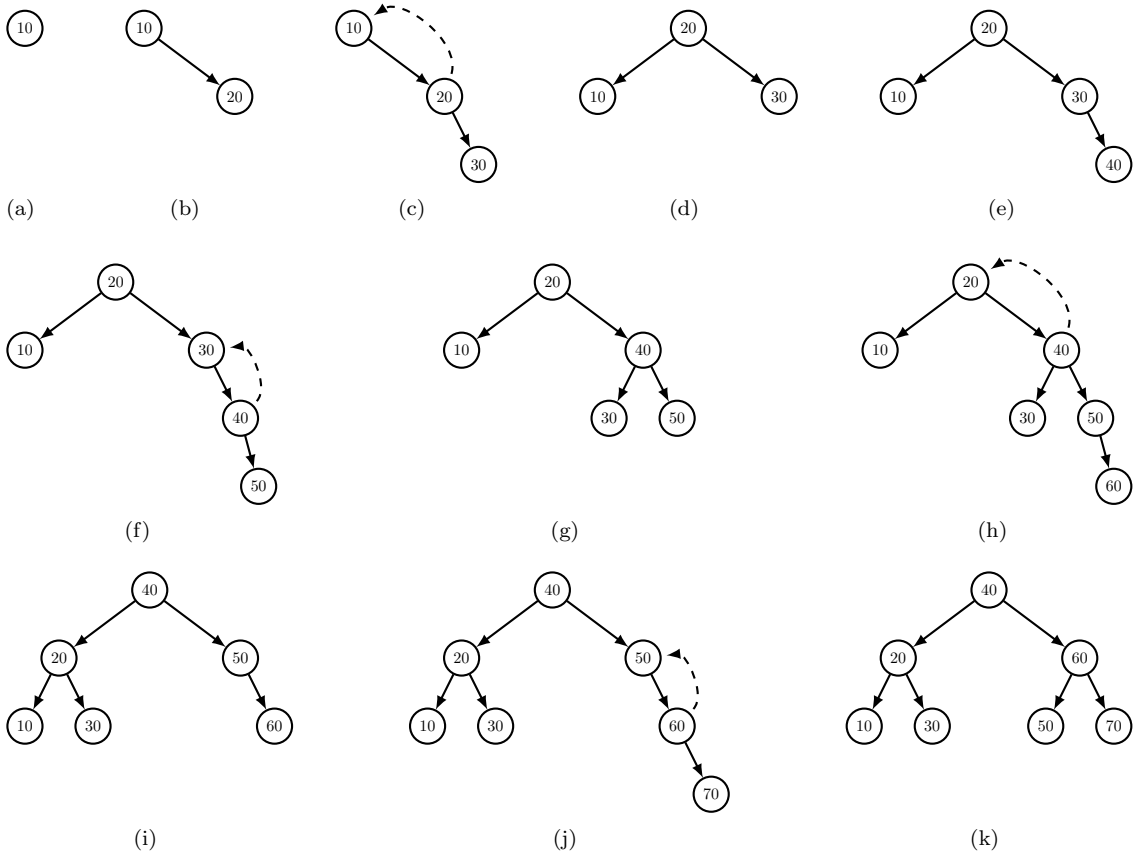
Solution

1. The result is shown in Figure 1. This tree is obviously not very balanced.



Figuur 1: An unbalanced tree.

2. The tree is rebalanced as necessary as shown in Figure 2.



Figuur 2: A BST that remains balanced.