CSP2348/CSP5243 Data Structures

Solutions to Tutorial 06: Binary Tree Data Structures

Tasks:

Complete the following.

Task 1: Explain the relationship between the depth and node of a binary tree using the examples given below:

a. How many nodes does a fully-balanced binary tree of depth 4 have?

   $\mathbf{n = 2^{d+1} - 1 = 2^{4+1} - 1 = 2^5 - 1 = 31}$

b. What is the maximum depth of an balanced binary tree of 30 nodes;

   $\mathbf{d = \mathit{floor}(\log_2 30) = 4}$

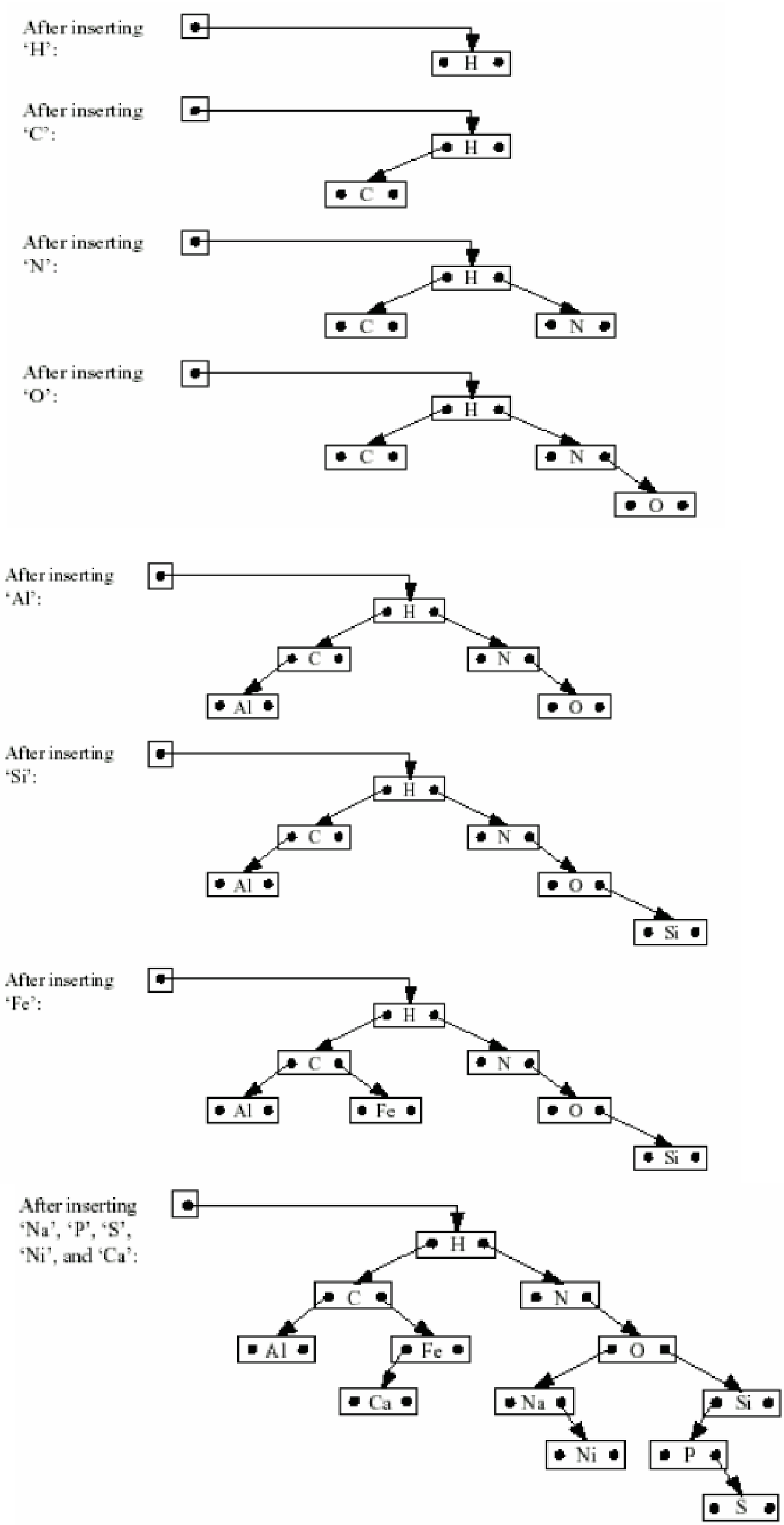
c. Verify your answers above by drawing illustrative binary trees.

   (Try doing it by yourself.)

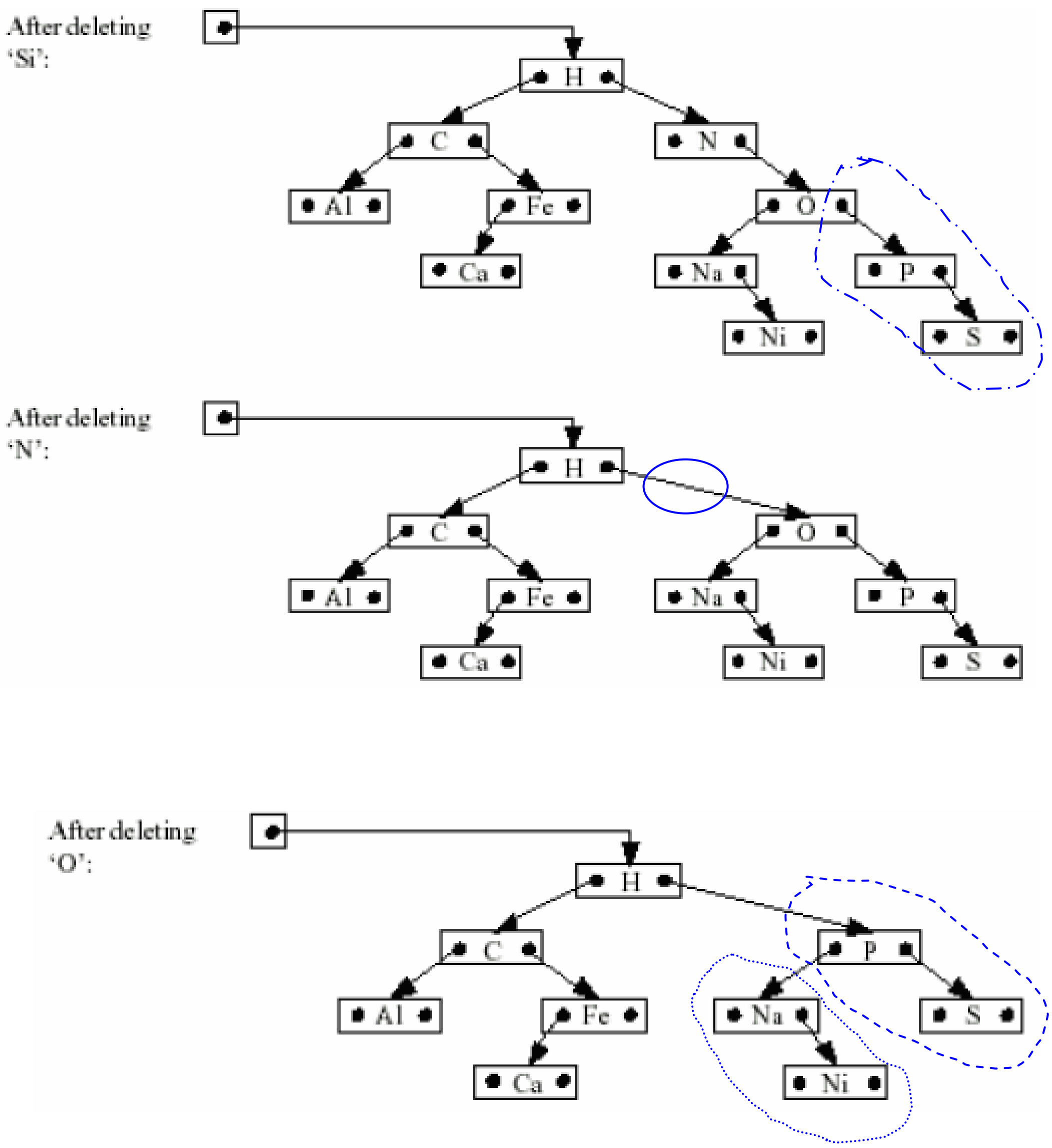
Task 2: Consider a binary search tree (BST) whose elements are abbreviated names of chemical elements.

a. Starting with an empty BST, show the effect of successively inserting the following elements: H, C, N, O, Al, Si, Fe, Na, P, S, Ni, Ca.
b. Show the effect of successively deleting Si, N, O from the resulting BST.

(a)


After inserting 'H':
After inserting 'C':
After inserting 'N':
After inserting 'O':
After inserting 'Al':
After inserting 'Si':
After inserting 'Fe':
After inserting 'Na', 'P', 'S', 'Ni', and 'Ca':

(b)



Task 3: Test the Java implementation of a `Binary Search Tree` given in TreeTest.java (Download the Java code from Blackboard).

a. Execute this program;

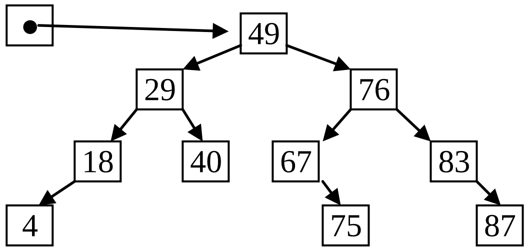
Run TreeTest directly.

Sample

```
Inserting the following values:
     49 76 67 29 75 18 4 83 87 40
Preorder traversal
     49 29 18 4 40 76 67 75 83 87
Inorder traversal
     4 18 29 40 49 67 75 76 83 87
Postorder traversal
     4 18 40 29 75 67 87 83 76 49
```

b. Use the first line of values to draw this BST;

```
Inserting the following values into a BST:
     49 76 67 29 75 18 4 83 87 40
```



c. Hand-test the visitation of this BST in terms of Pre-order, In-order, and Post-order traversals;
(do yourself)

d. Compare your results with the executed results.
Both have the same results.

Task 4 (*Optional*):

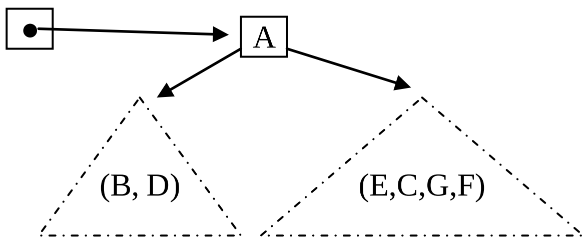
Given the following traversal orders, draw the binary tree.

Pre-order: A, B, D, C, E, F, G

In-order: B, D, A, E, C, G, F

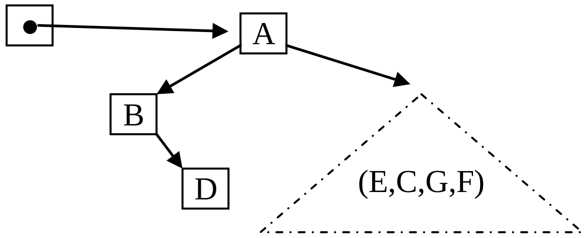
(1) Recall that the *pre-order traversal* traverses the binary tree in the order of root, (left subtree) and (right-subtree). This order can be used to determine the root node of a (sub)tree. Specifically, the first element in the pre-order must be (the element of) the root node. The *in-order traversal* traverses the binary tree in the order of (left-subtree), root, and (right-subtree). The in-order sequence can be used to determine nodes in its left and/or right subtrees.

From the given pre-order sequence, we know that A is the element of the root node of the tree (because A appears first in the pre-order). Then, from the in-order sequence, the whole sequence can be divided into three parts, as (B, D), A, (E, C, G, F). A's left subtree contains elements B, D (because B and D are the only elements appearing before A in the in-order of the binary tree). Similarly, A's right subtree contains elements E, C, G, F. We can draw initially a "conceptual" binary tree like this:



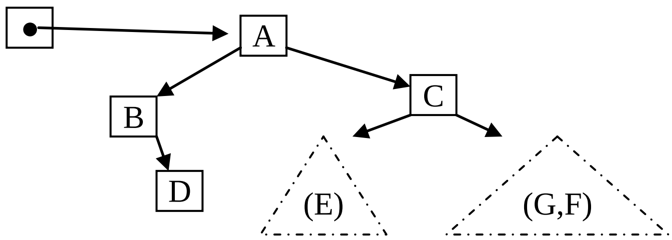
where A's left sub-tree contains elements B and D; and A's right sub-tree contains elements four elements, E, C, G, and F.

(2) From the above A's left subtree, as B appears before D in the pre-order, B must be the root of the subtree. In addition, B also appears before D in the in-order, indicating that D must be in B's right subtree, which contains D as the only node. Thus we have



(3) Similarly, in A's right subtree, C appear before E, G and F in the pre-order. This means that C is the root node of the subtree. To work out which nodes in C's left or right subtrees, we check these nodes in the in-order.

In the in-order sequence, E appears before C, indicating E is in C's left subtree. And G and F appear after C in the in-order, indicating they are in C's right subtree.



(4) Follow the same principle, we can work out the appearance of c's subtrees in the above figure (detailed description omitted here). The resulting banary tree looks like: