WORKSHEET, WEEK 06: SEARCH TREES

6.1 Concepts and Facts

Definition: A tree is named *Binary Search Tree* (BST) if the nodes satisfy the *order invariant*: Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $y.key \leq x.key$. If z is a node in the right subtree of x, then $z.key \geq x.key$.

Definition: In a binary tree, the *inorder predecessor* of a node v is a node u iff v directly follows u in the inorder traversal of the nodes. Similarly, the *inorder successor* of a node v is w iff w directly follows v in the inorder traversal.

To delete an internal node from a BST (having both left and right children), you can replace it either by the inorder predecessor or the inorder successor.

Definition: The *height* of a node in a tree is defined by induction:

- Null trees (empty trees) have height -1
- Leaves (single node trees) have height 0
- Any node v has height $h(v) = \max(h(v_{\text{left}}), h(v_{\text{right}})) + 1$

Definition: A binary search tree is called an *AVL tree* iff each node v is balanced. Namely, the heights of both its subtrees do not differ by more than 1.

$$|h(v_{\text{left}}) - h(v_{\text{right}})| \leq 1$$

To see, if a tree is an AVL tree (the representation invariant must be preserved even after we insert or delete something!) we need to store the balance inside the tree node. Such additional information is called graph/tree node augmentation. (Augmentations are used by many algorithms and data structures. AVL trees needing the height is just one example.)

6.2 Problems

Problem 1: Even binary trees that are not complete can be represented as arrays – their nodes written out layer by layer just like a complete tree in a heap. If any node in the tree is missing, it is replaced by Λ. The last non-empty node in the tree is the last element of the array. Draw the trees corresponding to the following arrays (here a, b, c are variable letters stored in the nodes). Distinguish the left and right children clearly.

(A) `int a[]` $= \{1, 2, 3, 7, \Lambda, 5\}$;

(B) `int a[]` $= \{1, 2, 4, a, \Lambda, \Lambda, 6, b, \Lambda, \Lambda, \Lambda, \Lambda, \Lambda, \Lambda, c\}$;

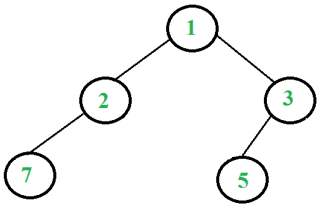
(C) Write a pseudocode to check, if the input array represents a binary tree (return True), or it is inconsistent (for example, there are non-empty children under some Λ).

(D) Write a pseudocode to count the internal nodes and the leaves, if you receive an array as an input.

(E) Write a pseudocode to list the vertices of this tree in the post-order traversal order.

Answer:

(A)



Problem 2: Non-binary ordered trees can be encoded as binary trees (using a bijective encoding function). See https://bit.ly/3khnC0p for details. (If in a general tree the node w is the first child of v, then in the corresponding binary tree w becomes the *left child* of v. If w is the sibling to the right of v, then in the corresponding binary tree w is the *right child* of v.) In this problem we use *bracket representations* for all trees (see https://bit.ly/425tzVa).

(A) Consider the following general tree: `A(B(E)(F)(G))(C)(D(H)(I)(J))`.

Draw this multiway/general tree. Encode it as the binary tree and draw it.

(B) `A(B(E()(F(J()(K))()))(C()(D(G()(H(L(N)(M))(I))()))))()`.

Given the binary tree, restore the original general tree.

(C) Consider the binary tree from (B); list its nodes in their in-order DFS traversal order.

(D) What is the depth of the node N (defined above) in the new (general) tree?

Answer:

(A) The general tree is given on Figure *Multiway Tree*.

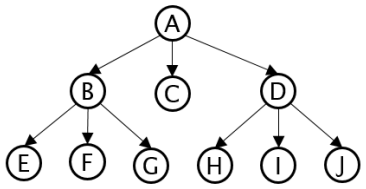


Fig. 1: Multiway Tree

Encoding Step 1 Redraw edges (only connect each node with its first child and also to the sibling to the right). To see clearly which edges will be left-going, and which are right-going, can color them differently. See Figure *Tree with Horizontal Edges*.

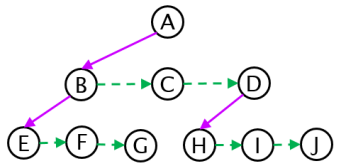


Fig. 2: Tree with Horizontal Edges

Encoding Step 2 Adjust the levels in the new binary tree so that it takes a more conventional look (left children to the left, right children to the right). See Figure *Encoded Binary Tree*.

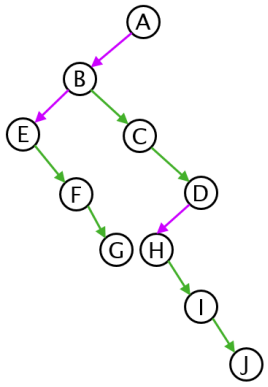


Fig. 3: Encoded Binary Tree

(B) The binary tree looks like this:

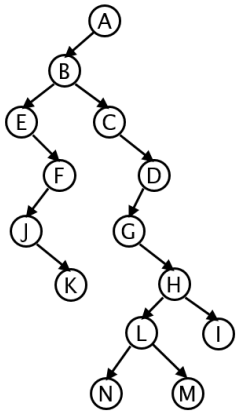


Fig. 4: Binary tree for Question 6.1.1

Restored tree can be obtained, if one colors the edges and turns the left children into first children, and the right children to siblings.

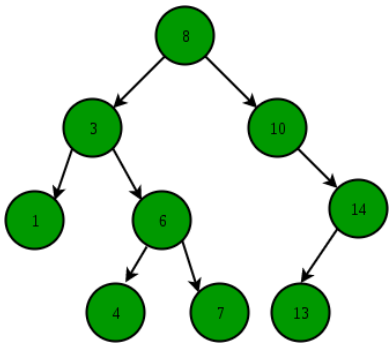
Note: See Encoding general trees as binary trees or https://bit.ly/3kdyg8n.

Problem 3: Let B_n denote how many different BSTs for n different keys there exist (all the trees should have correct order invariant). We have $B_1 = 1$ (one node only makes one tree). And $B_2 = 2$.

Draw all the binary search trees to store numbers $\{1, 2, 3\}$ and also the numbers $\{1, 2, 3, 4\}$.

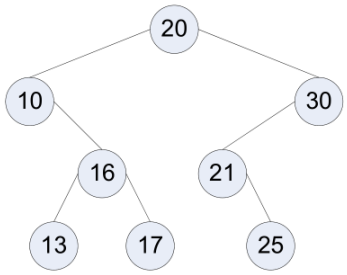
Find the values B_3 and B_4 (the number of binary search trees).

Problem 7: Consider the binary tree shown below.



Every key in this tree is being searched with the same probability. Find the expected number of pointers that are followed as we search for a random key in this tree. (For example, searching the key at the root means following 1 pointer, searching the key that is a child of the root means following 2 pointers and so on.)

Problem 4: Consider the following Binary Search Tree (BST).



Let a, b be the first two digits of your Student ID. Compute the following numbers:

$$
\begin{aligned}
X &= 2a, \\
Y &= 20 + b, \\
Z &= 3b, \\
S &= b, \\
T &= 2(a + b) \bmod 40 \\
U &= (a + b) \bmod 10
\end{aligned}
$$

Run the following commands on this BST (and draw the intermediate trees whenever there is the show command):

$BST.\textsc{insert}(X)$
$BST.\textsc{insert}(Y)$
$BST.\textsc{delete}(20)$
$BST.\textsc{show}()$
$BST.\textsc{insert}(Z)$
$BST.\textsc{insert}(S)$
$BST.\textsc{delete}(13)$
$BST.\textsc{show}()$
$BST.\textsc{insert}(T)$
$BST.\textsc{insert}(U)$
$BST.\textsc{delete}(X)$
$BST.\textsc{show}()$

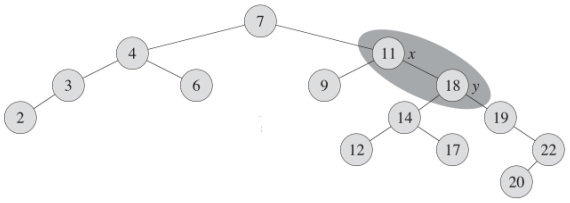
Ignore a command, if it asks to insert a key that already exists or deletes a key that does not exist.

Question 5: Let T_n be an AVL tree of height n with the smallest possible number of nodes. For example $|T_0| = 1$ (just one node is an AVL tree of height 0); $|T_1| = 2$ (a root with one child only is an AVL tree of height 1) and so on.

(A) Draw AVL trees T_2, T_3, T_4 and T_5.

(B) Write a recurrence to find the number of nodes $|T_n|$ (recurrent formula expresses the number $|T_n|$ using the previous numbers $|T_k|$ with $k < n$).

Problem 6: Let T be some (unknown) BST tree that also satisfied the AVL balancing requirement. After k nodes were inserted (without any re-balancing actions) the tree T' now looks as in the image below.



(A) Find the smallest value of k – the nodes that were inserted into the original T to get T'.

(B) Show the tree after LEFTROTATE(T', x) – the left rotation around the node x. Is the resulting tree an AVL tree now?

Problem 7: Assume that a Binary Search Tree T is created by inserting the following keys into an empty tree: $[39, 20, 65, 11, 29, 50, 26]$ (in the given order).

(A) Do the following actions on this tree one after another: T.INSERT(22), T.INSERT(60), T.DELETE(11).

(B) Suggest a sequence of inserts/deletes for the original tree :math;`T` (with 7 nodes) so that the last delete operation in that sequence causes two rotations.