WORKSHEET, WEEK 06: SEARCH TREES

6.1 Binary Trees as Arrays

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, Λ, 5};`

(B) `int a[]` $=$ `{1, 2, 4, a, Λ, Λ, 6, b, Λ, Λ, Λ, Λ, Λ, Λ, 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.

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))())))()`.

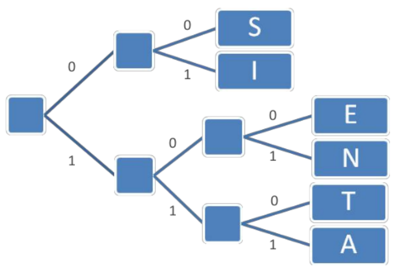
Given the binary tree, restore the original general tree.

(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?

6.2 Prefix Codes and Huffman Algorithm

Problem 3: Consider the following Prefix Tree to encode letters in alphabet $\mathcal{A} = \{S, I, E, N, T, A\}$.



Every letter is encoded as a sequence of 0s and 1s (the path from the root to the respective letter).

(A) Decode the following sequences:

- `11100110100`
- `0001100101111`

(B) Explain, if there are sequences of bits that are *ambiguous* (can be decoded in more than one way).

(C) Explain, if there are sequences of bits that are *impossible* (do not represent any word in the alphabet $\mathcal{A}$).

Huffman Algorithm: Let C be the collection of letters to be encoded; each letter has its frequency $c.freq$ (frequencies are numbers describing the probability of each letter).

```
HUFFMAN($C$):
  $n = |C|$
  $Q$ = PRIORITYQUEUE($C$)      (Minimum heap by "c.freq")
  for $i = 1$ to $n - 1$      (Repeat n-1 times)
    $z$ = NODE()
    $z.left = x$ = EXTRACTMIN(Q)
    $z.right = y$ = EXTRACTMIN(Q)
    $z.freq = x.freq + y.freq$
    INSERT($Q, z$)
  return EXTRACTMIN(Q)      (Return the root of the tree)
```

Problem 4: Let the alphabet have 6 characters $\mathcal{A} = \{A, B, C, D, E, F\}$ and their probabilities are shown in the table:

c	A	B	C	D	E	F
$P(c)$	27%	9%	11%	15%	30%	8%

Use the Huffman algorithm to create a Prefix Tree to encode these characters.

Problem 5:

(A) For the alphabet (and letter frequencies) taken from the previous question compute the Shannon entropy:

$$H(\mathcal{A}) = \sum_{c \in \mathcal{A}} (-\log_2 P(c)) \cdot P(c),$$

where $P(c)$ denotes the probability of the character c in the alphabet.

(B) Also compute the expected number of bits needed to encode one random letter by the Huffman code you created in the previous question. (Assume that letters arrive with the probabilities shown in the table.)

Note: Theory (not in the scope of our course) tells that nobody can encode the alphabet $\mathcal{A}$ better than the Shannon's entropy. On the other hand, Huffman code is an optimal prefix code; the expected number of bits spent per one letter does not exceed $H(\mathcal{A}) + 1$.

6.3 Binary Search Trees

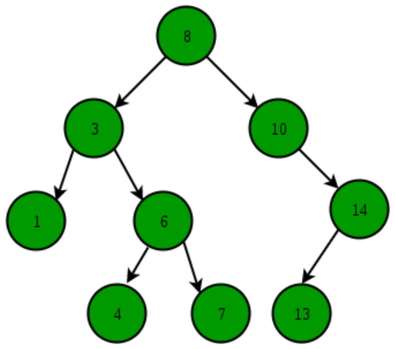
Definition: A tree is named *Binary Search Tree* (BST) if the nodes satisfy the *order invariant*: Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $y.key \leq x.key$. If z is a node in the right subtree of x, then $z.key \geq x.key$.

Problem 6: 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.

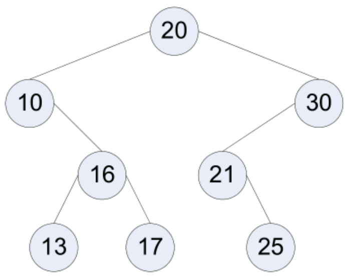


Every key in this tree is being searched with the same probability. Find the expected number of pointers that are followed as we search for a random key in this tree. (For example, searching the key at the root means following 1 pointer, searching the key that is a child of the root means following 2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

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.

Problem 8: 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):

$BST.\text{INSERT}(X)$
$BST.\text{INSERT}(Y)$
$BST.\text{DELETE}(20)$
$BST.\text{SHOW}()$
$BST.\text{INSERT}(Z)$
$BST.\text{INSERT}(S)$
$BST.\text{DELETE}(13)$
$BST.\text{SHOW}()$
$BST.\text{INSERT}(T)$
$BST.\text{INSERT}(U)$
$BST.\text{DELETE}(X)$
$BST.\text{SHOW}()$

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

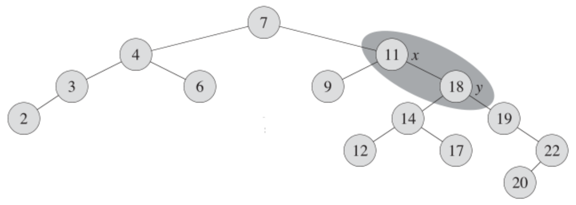
6.4 AVL Trees

Question 9: 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 10: 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 $\text{LEFTROTATE}(T', x)$ – the left rotation around the node x. Is the resulting tree an AVL tree now?