WORKSHEET, WEEK 06: SEARCH TREES

6.1 Trees and Binary Trees

Introduction: How to Encode General Tree to a Binary Tree

Sometimes non-binary (ordered) trees should be represented in binary-tree data structures. See https://bit.ly/3khnC0p for details. The two rules to encode are these:

- Every node v in the general tree with its *first child* w maps to the same node v in the binary tree, where the corresponding w is its *left child*.
- Every node v in the general tree having w as its *sibling to the right* has the same w in the binary tree as its *right child*.

One can also decode: given a binary tree (if its root only has the left child), it is possible to restore the original general tree.

Consider an example general tree on Figure *Multiway Tree*.

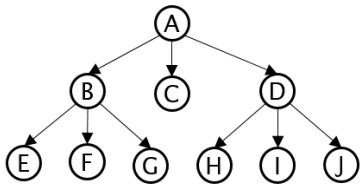


Fig. 1: Multiway Tree

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*.

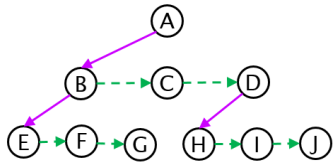


Fig. 2: Tree with Horizontal Edges

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*.

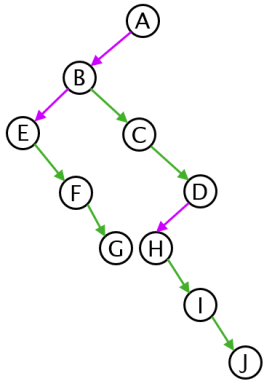


Fig. 3: Encoded Binary Tree

Question 6.1.1 (Decode Binary to a Multiway Tree):

(A) List all the nodes in Figure *Binary tree for Question 6.1.1* using the in-order tree traversal.

(B) Binary tree B shown in *Binary tree for Question 6.1.1* has been obtained by encoding some general (multiway) tree T (The tree T is rooted and ordered, but it is not necessarily binary.) Restore the general tree T by decoding the given binary tree.

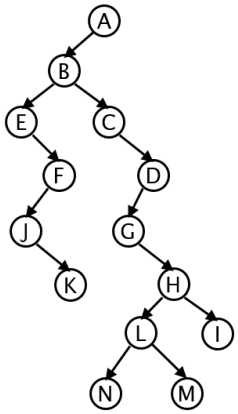


Fig. 4: Binary tree for Question 6.1.1

Question 6.1.2 (Binary Trees):

Define a new integer number $N \in \{0, 1, 2, \ldots, 9\}$ from the digits of your Student ID:

$$N \ = \ (a + b) \bmod 10.$$

(A) Redraw the binary tree in Figure; replace letters a, b with your values. We denote this tree by B.

(B) List all the nodes of B in their in-order DFS traversal order.

(C) Draw a general tree (denoted by G) that is obtained by decoding the tree B. See Encoding general trees as binary trees or https://bit.ly/3kdyg8n.

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

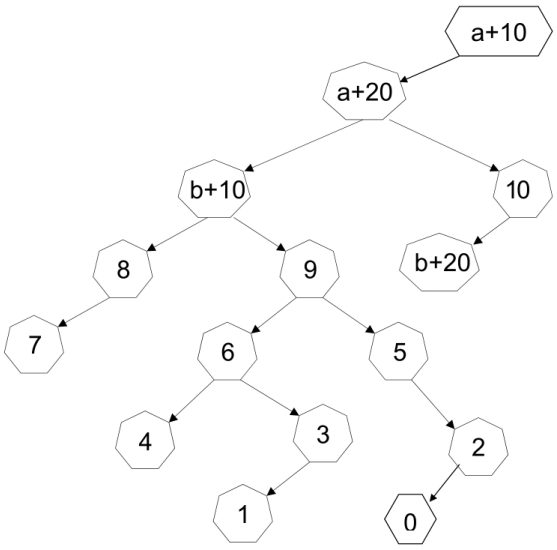


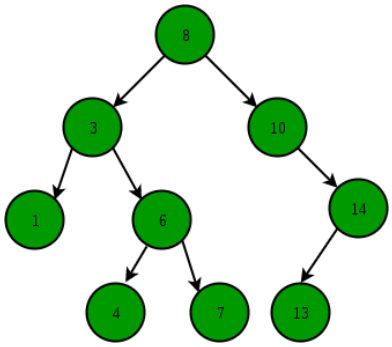
Fig. 5: Binary tree B for inorder traversal and converting to a general tree G

6.2 BST 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 $z.key \geq z.key$.

Question 6.2.1 (Recurrences to count BSTs): 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 $B_2 = 2$ (in the case of two

Question 6.2.2 (Search Random Keys in BST): Consider the binary tree shown below.



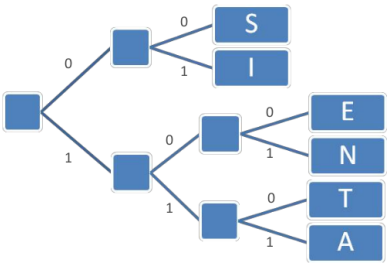
Every key in this tree is being searched with the same probability. Find the expected number of pointers that are followed as we search for a random key in this tree. (For example, searching the key at the root means following 1 pointer, searching the key that is a child of the root means following 2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

6.3 Prefix Codes and 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*)

Question 6.3.1 (Decrypt/Encrypt Prefix Code): 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 $\mathcal{A}$).

Question 6.3.2 (Huffman Code): Let the alphabet be $\mathcal{A} = \{A, B, C, D, E, F\}$ and their probabilities are shown in the table.

A	B	C	D	E	F
27%	9%	11%	15%	30%	8%

Question 6.3.3 (Entropy and average code length):

(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.)

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 $H(\mathcal{A}) + 1$.

6.4 Inserting and Deleting from BSTs

Question 6.4.1 (Insert new nodes): Generate a random sequence of 10 different integer numbers and build a BST tree out of these numbers. What is the average depth of a node in this tree?

Question 6.4.2 (Delete nodes from BST): From the tree build in the previous question delete the following:

- Any leaf
- Any inner node with just one child (at best, pick an innder node that has another inner node as a child)
- Any inner node with two children.

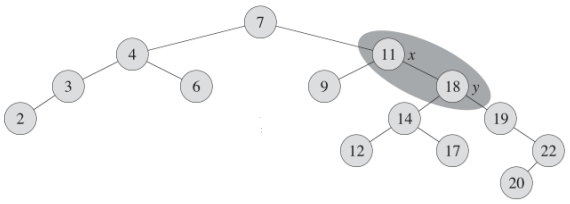
6.5 Inserting and Deleting from an AVL Tree

Question 6.5.1 (AVL tree with min nodes): 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 $k < n$).

Question 6.5.1 (AVL and Rotations): 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?

6.6 (2,4) and Red-Black Trees

Definition: A tree is named a *Red-Black Tree*, if it is a Binary Search Tree, every node is either red or black (extra boolean flag stores this color) and it satisfies these *red-black invariants*:

Root property The root is black.

External property Every leaf (a node with NULL key) is also black.

Internal property If a node is red, then both its children are black.

Depth property For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

Note: See Figure *Sample Red-Black Tree*; leaves with NIL keys have black-height equal to 0. As we move to the root, we increment the black-height h_{black} whenever the path crosses some black node. The Depth property guarantees that each internal node gets the same black-height, no matter which path from a leaf to a root we choose.

Question 6.6.1 (Insert Nodes in a Red-Black Tree):

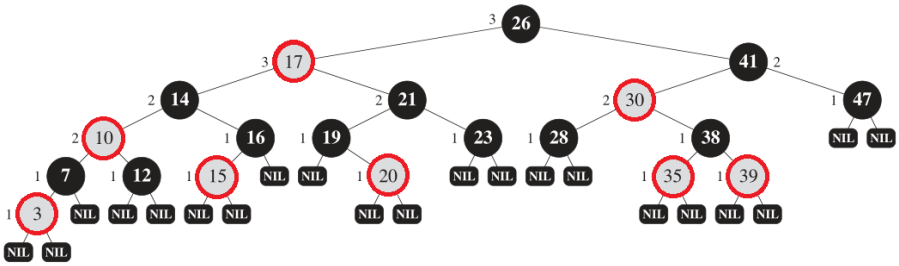


Fig. 6: Sample Red-Black Tree

(A) Compute the following three key values (u, v, and w):

$$\begin{cases} u = 3(a+b)+2 \\ v = 3(b+c)+1 \\ w = 3(c+a) \end{cases}$$

Here a, b, c are the last 3 digits of your Student ID.

Verify the "black height" of every node in the graph – all NULL leaves have black height equal to zero. Any other node has black height equal to the number of black nodes that are on some descendant path. (According to the depth property – the black height of any node should not depend on the path to the leaf we chose.)

(B) Show how the tree looks after the nodes u, v and w (in this order) are inserted in the Red-Black Tree shown in Figure *Sample Red-Black Tree*.

If any of the values u, v, w coincide with existing nodes, they should not be inserted. (Red-Black trees and BSTs in general can handle duplicates; but here we assume that it stores a map/set with unique keys.)

Show the intermediate steps – the tree after each successive inserted node. Clearly show, which are the red/black vertices in the submitted answers.

Note: Check that your inserts preserve the BST order invariant (along with all the Red-Black tree invariants). Secondly, try to follow the standard algorithm when inserting new nodes (still, preserving the invariants is more important).