**HOMEWORK 6: BINARY SEARCH TREES**

**Name: Trần Gia Bảo**

**ID: 22127034**

**A. Theory part:**

**A1.**

**A diagram of a tree

Description automatically generated**

a. Here are the answers to the questions about the given binary search tree:

- Root: Node with key 60

- Pairs of parents - children: (60, 20), (60, 70), (20, 10), (20, 40), (40, 30), (40, 50)

- Siblings: (20, 70), (10, 40), (30, 50)

- Ancestors of 50: 60, 20, 40

- Descendants of 20: 10, 40, 30, 50

- Leaves: Nodes 10, 30, 50, 70

b. The height of the tree is 3.

c. To create the binary search tree shown, you would insert the items in the following order: 60, 20, 70, 10, 40, 30, 50.

d.

- Search algorithm trace for searching 30:

+ Start at the root (60).

+ Move left to node 20.

+ Move right to node 40.

+ Move left to node 30.

- Search algorithm trace for searching 15:

+ Start at the root (60).

+ Move left to node 20.

+ Move left to node 10.

+ 15 is not found in the tree.

e. After inserting the entries 80, 65, 75, 45, 35, and 25, the resulting tree will be:

A diagram of a diagram

Description automatically generated

f. After removing the entries 50 and 20, the resulting tree will be:

A diagram of a diagram

Description automatically generated

**A2. Question error**

**A3.**

**A diagram of a network

Description automatically generated**

a. This is not a binary search tree. Because H is larger than G, H must lie on the right branch of G, similarly V and U are both larger than T so they lie on the right branch of T.

b.

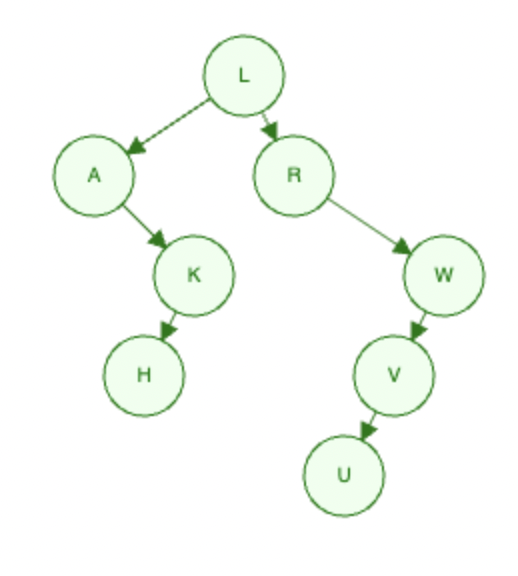
- In-order: A D H G K L M R U V T W  
- Pre-order: M G D A H K L T R V U W

- Post-order: A H D L K G M U V R W T

c. After editing the above tree into a binary tree:

A diagram of a diagram

Description automatically generated

After removing M, D, G, and T:  
  
 

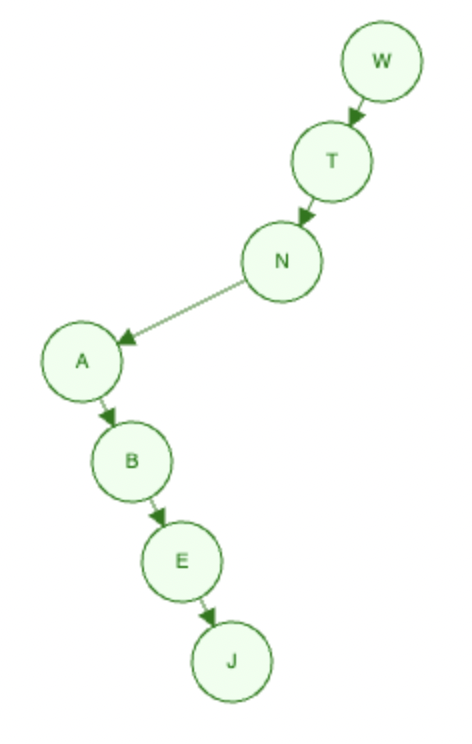
**A4.** Binary search tree is formed when I insert the following values in the order given:

a. W, T, N, J, E, B, A

A diagram of a diagram

Description automatically generated

b. W, T, N, A, B, E, J

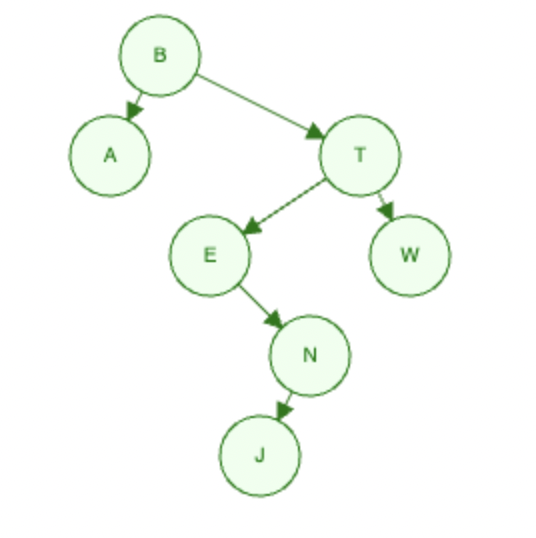


c. A, B, W, J, N, T, E

A diagram of a diagram

Description automatically generated

d. B, T, E, A, N, W, J



**A5.**

A diagram of a triangle

Description automatically generated

- Binary trees: b, c, d, e, f.

- Binary search trees: e, f

- Full binary trees: c, f

- Complete binary trees: b, d, e.

**A6.** The binary search tree (BST) property states that for any node in the tree, all the nodes in its left subtree have keys less than its key, and all the nodes in its right subtree have keys greater than its key. Given this property and the fact that the sequence of keys checked during the search of a specific key follows the path from the root of the tree down to the node containing the key, let's analyze each of the provided sequences:

- 5, 2, 1, 10, 39, 34, 77, 63: This sequence is a possible path for the search of key 45. It starts at the root (50) and moves left to 25, then left to 10, then right to 39, and finally left to 34. At this point, the search would go right to 40, and then right to 77. Since 45 is smaller than 77, the search would move left to 63, and 45 is not found, but this sequence is a valid path.

- 50, 25, 26, 27, 40, 44, 42: This sequence is not a possible path for the search of key 45. It starts at the root (50) and moves left to 25, then right to 26, which is incorrect because 45 is greater than 26 and should be in the right subtree of 50.

- 1, 2, 3, 4, 5, 6, 7, 8: This sequence is not a possible path for the search of key 45. It starts at the root (1) and keeps moving right in a linear fashion, which means it can't reach any key greater than 1, and 45 is not on this path.

- 50, 25, 26, 27, 40, 44: This sequence is not a possible path for the search of key 45. It starts at the root (50) and moves left to 25, then right to 26, which is again incorrect because 45 is greater than 26. The search would continue in the right subtree, but the sequence ends prematurely.

- So, the correct answer is: 5, 2, 1, 10, 39, 34, 77, 63

**A7.** To determine which sequence of keys is NOT a valid sequence checked during the search of key 363 in a binary search tree containing integers from 1 to 1000, let's analyze each option:

- 2, 252, 401, 398, 330, 344, 397, 363: This sequence is valid. The search starts at the root (which is 2) and goes down the tree based on comparisons until it reaches the key 363.

- 924, 220, 911, 244, 898, 258, 362, 363: This sequence is valid. It might look strange at first, but it's possible to have such a sequence in a binary search tree. The tree's structure might have led to this sequence during the search process.

- 2, 399, 387, 219, 266, 382, 381, 278, 363: This sequence is valid. Like the second option, even though the sequence might seem unusual, it's still possible in a binary search tree.

- 925, 202, 911, 240, 912, 245, 363: This sequence is valid. The search could have taken this path in the tree, moving left or right based on the comparisons.

-935, 278, 347, 621, 299, 392, 358, 363: This sequence is NOT valid. In a binary search tree, when you encounter a node with a value greater than the target key (363 in this case), you should continue searching in the left subtree. However, in this sequence, after key 621, the value 299 is encountered, which is greater than 363. This violates the properties of a binary search tree, where all nodes in the left subtree should have values less than the parent node.

- Therefore, the sequence "935, 278, 347, 621, 299, 392, 358, 363" is the one that is NOT a valid sequence of keys checked during the search for key 363 in a binary search tree.

**A8.**

- Full Binary Tree: A full binary tree is a type of binary tree in which every node has either 0 or 2 children. In other words, every node in a full binary tree has either no children (a leaf node) or exactly two children. There are no nodes with only one child.

- Complete Binary Tree: A complete binary tree is a binary tree in which all levels are filled except possibly the last level, and the nodes on the last level are as left as possible. In simpler terms, in a complete binary tree, all levels are filled from left to right, except possibly the last level which is filled from left to right up to a point. This structure is commonly used in heaps and priority queues.

- Perfect Binary Tree: A perfect binary tree is a type of binary tree in which all internal nodes have exactly two children, and all leaf nodes are at the same level. This means that all levels of a perfect binary tree are fully filled with nodes, resulting in a tree where the number of nodes at each level follows the pattern 2^level. The number of nodes in a perfect binary tree can be calculated using the formula (2^(h+1)) - 1, where h is the height of the tree.

- To summarize the differences:

+ Full Binary Tree: Every node has either 0 or 2 children.

+ Complete Binary Tree: All levels are filled except possibly the last level, and nodes on the last level are left-aligned.

+ Perfect Binary Tree: All internal nodes have exactly two children, and all leaf nodes are at the same level.

**A9.**

- The number of differently shaped n-node binary trees and n-node binary search trees can be calculated using recursive definitions.

- Differently Shaped n-Node Binary Trees:

+ Let's denote the number of differently shaped n-node binary trees as T(n). For any value of n, there are n possible choices for the root node. Once the root is chosen, the remaining nodes can be split into a left subtree and a right subtree. The number of nodes in the left subtree and the right subtree can vary, but their sum will be n - 1 (since one node is used as the root). We can now recursively calculate the number of different shapes for each of these subtrees. So, the recursive definition is:

T(n) = Σ(T(i) \* T(n - i - 1)) for i = 0 to n - 1

+ The base case is T(0) = 1 (an empty tree) and T(1) = 1 (a tree with one node).

- Differently Shaped n-Node Binary Search Trees:

+ The number of differently shaped n-node binary search trees is often referred to as the nth Catalan number and is denoted as C(n). The recursive definition for C(n) is:

C(n) = Σ(C(i) \* C(n - i - 1)) for i = 0 to n - 1

+ The base case is C(0) = 1.

- It's important to note that n-node binary search trees are a subset of n-node binary trees, and the number of binary search trees will generally be smaller than the total number of binary trees with the same number of nodes.

**A10.**

a. A full binary tree of height h ≥ 0 has – 1 nodes:

- Base Case: For h = 0, a full binary tree has only one node (the root), and – 1 = 0, which is true.

- Inductive Hypothesis: Assume that for some positive integer k, a full binary tree of height k has - 1 nodes.

- Inductive Step: We want to show that a full binary tree of height k + 1 has - 1 nodes.

- Consider a full binary tree of height k + 1. It consists of a root node and two subtrees, both of which are full binary trees of height k. By the inductive hypothesis, each of these subtrees has - 1 nodes. Therefore, the total number of nodes in the tree of height k + 1 is:

Number of nodes = 1 (root) + ( - 1) (left subtree) + ( - 1) (right subtree)

= + - 1

= - 1

- This completes the inductive step. Thus, by mathematical induction, we have proven that a full binary tree of height h has - 1 nodes for h ≥ 0.

b. The maximum number of nodes in a binary tree of height h is – 1:

- This statement is essentially the same as part a, so we can use the same induction proof. The base case and the inductive step remain the same, and we've already shown that a full binary tree of height h has – 1 nodes. This also proves that the maximum number of nodes in any binary tree of height h is – 1.

c. A binary tree with n nodes has exactly n + 1 empty subtrees:

- Base Case: For a tree with a single node (just the root), there are 0 empty subtrees, and n + 1 = 1.

- Inductive Hypothesis: Assume that for some positive integer k, a binary tree with k nodes has exactly k + 1 empty subtrees.

- Inductive Step: We want to show that a binary tree with k + 1 nodes has exactly k + 2 empty subtrees.

- Consider a binary tree with k + 1 nodes. Remove the root node, and you will have two subtrees. The original tree with k + 1 nodes can be constructed by adding the root node to these two subtrees. Therefore, each of these subtrees has k nodes. By the inductive hypothesis, each subtree has k + 1 empty subtrees.

- Adding the root node creates one additional empty subtree (the whole tree without any nodes). Thus, the total number of empty subtrees for the original tree with k + 1 nodes is (k + 1) + (k + 1) + 1 = k + 2.

- This completes the inductive step. Thus, by mathematical induction, we have proven that a binary tree with n nodes has exactly n + 1 empty subtrees.

- These proofs demonstrate the validity of the given statements using mathematical induction.