GROUP MEMEBERS

1. **Abraham Koome- COM/B/01-00096/2022**
2. **Warui Mocha- SIT/B/01-03284/2023**
3. **JohnPaul Juma- SIT/B/01-03306/2023**
4. **Livingstone Mapesa EDS/B/01-05048/2024**

SOLUTIONS:

QUESTION 1:

Step 1: Understand the structure of the tree.

A tree consists of nodes connected by edges. The height \(h \) of a tree is the length path from the root to a leaf. The height (\(h \)) of a tree in the height \(f \) is achieved when every leaf of the tree is fully populated.

Step 2 (Comparison of the Number of nodes at each level)

Level (Rot) - Level 0 - Rot node is at level 0 and contributes 4 node.

Level 1 - After label on line a maximum of 2 nodes in

level 2 - The level on line a maximum of 4 nodes.

The therefore conclude level 1 has 2 nodes.

Step 3: Maximum number of nodes (Total number of nodes)

Given a height \(h \), the total number of nodes in a tree of height \(h \) is:

\[N = 2 + 4 + \ldots + 2h.\]

This form a geometric series with sum:

\[N = \sum_{i=0}^{n} 2^i = \frac{2^{n+1}}{2-4} = 2^{n+1}-1.\]

<u>QUESTION 2:</u>

1). Depth first search - Is a tree or graph traversal algorithm

that explores or for as possible along each branch before

backtracking?

Example:

DEE Traversal (Reorder, Rat->Left-> Right):

Chasing from A:

1 Visit A:

2 Go to B - Visit B:

3 Go to E -> Visit E(Backtrack to B):

4 Go to F -> Visit F(Backtrack to B -> A:

5 Go to C -> Visit C(Backtrack to A):

6 Go to D -> Visit D:

7 Go to G -> Visit G(Backtrack to D):

8 Go to H -> Visit H.

Order:

A -> B -> E -> F -> C -> D

➔ G -> H

2) Breadth Search

It is a search algorithm that stops at the search of the root and explores all the nodes of the present depth before moving on the nodes of the next depth level.

Example:
Starting from A:

1. Visit A
2. Visit B, C, D
3. Visit E, F, G, H

BFS Order: A → B → C → D → E → F → G → H

Use case: Shortest path in an unweighted graph.

3) AVL Search

It is performed on AVL Tree, which is a self-balancing binary search tree where the height difference between left and right subtrees at most 1.

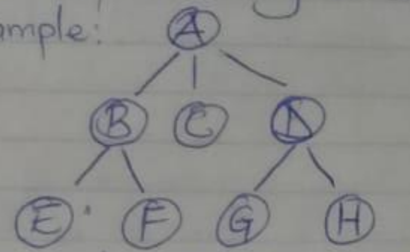
Balance factors for different nodes are:

- 4-2, 1-2, 1-5, 1-1, 1-0, 1-4, 0, and 4-0; hence it is an AVL Tree since all differences are less than or equal to

Usecase: Pathfinding in maze.

(ii) Breadth - Search ⇒ It is a search algorithm that starts at the search at the root and explores all nodes at the present depth before moving on the nodes at the next depth level.

Example:



Starting from A:
1. Visit A
2. Visit B, C, D.
3. Visit E, F, G, H

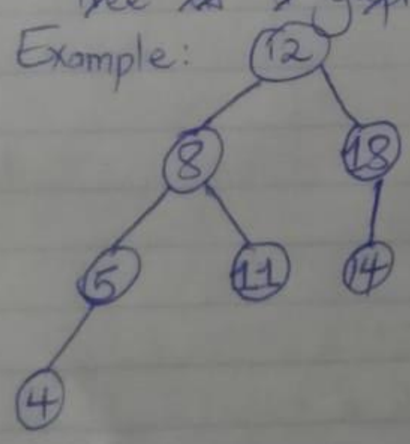
BFS Order: A → B → C → D → E → F → G → H.

Use case: Shortest path in an unweighted graph.

(iii) AVL Search - It is performed on AVL Tree, which is a self-balancing binary search tree where the height differe[nce] (balance factor) between left and right subtrees is at most 1.

~~(3) Tree as a graph data structure.~~

Example:



Balance factors for different nodes are: 12:1, 18:1, 5:1, 11:0, 14:0, and 4:0; hence it is an AVL Tree since all differences are less than or equal to 1.

(3) Tree as a graph data structure.

A tree is a specific type of graph characterized by being

(1) Step 1: Understand the Structure of the Tree

- A tree consists of nodes connected by edges. The height h of a tree is the length path from the root to a leaf. The ~~height (h) of a tree is the length~~ maximum number of nodes is achieved when every level of the tree is fully populated.

Step 2: Computation of the Number of nodes at each Level

Level (Root) - Level 0 - Root node is at level 0 and contributes 1 node.

Level 1 - ~~Has~~ This ~~max~~ level can have a maximum of 2 nodes. ~~wh~~

Level 2 - This level can have a maximum of 4 nodes.

∴ We therefore conclude level i has 2^i nodes.

Step 3: Maximum number of nodes (Total number of nodes)

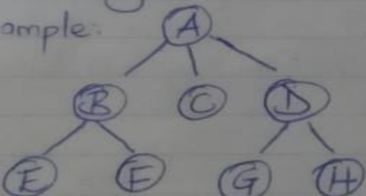
- Given a height h, the total number of nodes in a tree of height h is: $N = 1 + 2 + 4 + \dots + 2^h$

This form a geometric series with sum:

$$N = \sum_{i=0}^{h} 2^i = \frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1.$$

(2) • Depth first search - Is a tree or graph traversal algorithm that explores as far as possible along each branch before backtracking.

Example:



DFS Traversal (Preorder: Root → Left → Right).

Starting from A:

1. Visit A.
2. Go to B - Visit B
3. Go to E → Visit E (Backtrack to B)
4. Go to F → Visit F (Backtrack to B → A
5. Go to C → Visit C (Backtrack to A)
6. Go to D → Visit D
7. Go to G → Visit G (Backtrack to D)
8. Go to H → Visit H.

Order: A → B → E → F → C → D → G → H.

QUESTION 4

```
class Node { int key; Node left, right;

public Node(int item) {
   key = item;
   left = right = null;
}

}

class BinaryTree { Node root;

BinaryTree() {
   root = null;
}

void insert(int key) {
   root = insertRec(root, key);
}

Node insertRec(Node root, int key) {
   if (root == null) {
      root = new Node(key);
      return root;
   }
   if (key < root.key)
```

```
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);
    return root;
}

void inorder() {
    inorderRec(root);
}

void inorderRec(Node root) {
    if (root != null) {
        inorderRec(root.left);
        System.out.print(root.key + " ");
        inorderRec(root.right);
    }
}

Node search(Node root, int key) {
    if (root == null || root.key == key)
        return root;
    if (root.key > key)
        return search(root.left, key);
    return search(root.right, key);
}
```

```
void delete(int key) {
   root = deleteRec(root, key);
}

Node deleteRec(Node root, int key) {
   if (root == null) return root;
   if (key < root.key)
      root.left = deleteRec(root.left, key);
   else if (key > root.key)
      root.right = deleteRec(root.right, key);
   else {
      if (root.left == null) return root.right;
      else if (root.right == null) return root.left;
      root.key = minValue(root.right);
      root.right = deleteRec(root.right, root.key);
   }
   return root;
}

int minValue(Node root) {
   int minv = root.key;
   while (root.left != null) {
      minv = root.left.key;
      root = root.left;
   }
   return minv;
```

```
}

public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
    tree.insert(50);
    tree.insert(30);
    tree.insert(70);
    tree.insert(20);
    tree.insert(40);
    tree.insert(60);
    tree.insert(80);

    System.out.println("Inorder traversal:");
    tree.inorder();

    System.out.println("\nDeleting 40");
    tree.delete(40);
    System.out.println("Inorder traversal after deletion:");
    tree.inorder();
}

}
```